APRENDE JAVA FÁCIL

MIENTRAS ESCAPAS DE LA CÁRCEL

Teoría y ejercicios resueltos

HUGO RIBÉ

Aprende Java mientras escapas de la Cárcel. Teoría y ejercicios resueltos

Hugo Ribé

Sobre el autor

Antes de meternos en faena, me gustaría presentarme formalmente. Me llamo Hugo y voy a ser tu profesor de Java. Hace ya algunos años empecé a aprender a programar por mi cuenta. Simplemente como un entretenimiento, pero pronto me di cuenta de que era algo que realmente me gustaba. Creo que, si algo me define bien, es la pasión por el autoaprendizaje. Siempre trato de adquirir nuevos conocimientos e intento recopilar material pedagógico de donde puedo.

Durante mi carrera laboral, he tenido la oportunidad de trabajar para dos de las multinacionales tecnológicas más importantes del planeta. Sin experiencia previa y casi sin ser consciente de ello, me vi formando parte de un equipo técnico donde el análisis de datos y la destreza con SQL eran fundamentales. Poco después, el departamento evolucionó y aprendí a entrenar y configurar modelos de inteligencia artificial. En pocos meses me ascendieron a jefe de proyecto, y comenzaron mis reuniones e interacciones con algunos de los desarrolladores de software más talentosos que he visto en mi vida.

Inmerso en ese estimulante ambiente, pronto sentí curiosidad por la programación y, aunque en mi entorno se utilizaba más Python, decidí decantarme por Java. Al principio ni siquiera pensaba que lograría llegar a tener un nivel decente. No obstante, con constancia y muchas ganas, al final fui cumpliendo todos mis objetivos y logré dominar este lenguaje, que tan buenos ratos me ha dado frente al monitor.

Por supuesto, esta nueva habilidad me ha servido para abrirme puertas dentro del ambiente corporativo en el que desempeño mi trabajo. Saber programar, a nivel laboral siempre puede resultarte útil, incluso aunque no estés pensando en un cambio o un ascenso a corto plazo. Nunca sabes lo que te deparará el futuro, y es mejor ser una persona flexible con la mayor cantidad posible de habilidades que ofrecer.

En mi trabajo, mis compañeros siempre me han comentado que tengo una gran habilidad para enseñar. Por ese motivo, solía ser el encargado de mostrar el proceso de trabajo a las nuevas incorporaciones del equipo y guiarles durante sus primeros meses. Gracias a este feedback, empezó a fraguarse la idea de tomarme más en serio la enseñanza, y posteriormente nacería la idea de crear este libro.

Aclaraciones sobre este libro y el aprendizaje de Java.

Aprender a programar no es tan difícil como piensas. En mi caso, recuerdo que, incluso antes de empezar a leer la teoría, creía que para mí sería imposible aprender programación, al menos a nivel avanzado. Más que nada, porque pensaba que para conseguirlo iba a necesitar un altísimo nivel de compromiso e invertir cientos de horas. Seré honesto: estaba convencido de que no iba a enterarme de nada y que, después de un par de semanas, iba a dejar de lado mi propósito.

No te voy a engañar, es cierto que para aprender vas a necesitar dedicar tiempo, tanto leyendo teoría como realizando ejercicios. Pero no es necesario contar con experiencia previa ni con habilidades informáticas excepcionales. Lo único que necesitas son ganas y un material pedagógico adecuado.

En internet encontré varias alternativas, pero ninguna llegó a convencerme del todo. Quizás es que no tuve suerte. Cursos demasiado caros, tutoriales incompletos, foros con respuestas a preguntas muy específicas, vídeos aleatorios y, en definitiva, un sinfín de material que, pese a que muchas veces era de alto valor, a mí no me servía para nada.

Lo que yo necesitaba era algo estructurado con lo que poder aprender paso a paso y tener así una sensación de progreso que me motivase a seguir formándome. Pensé entonces en comprar libros. En general, este tipo de material es caro y no quería gastarme mucho dinero en algo que ni siquiera sabía si me iba a gustar. Por suerte, encontré bastantes ejemplares en el mercado de segunda mano. El problema fue que, sin tener ni idea de lo que necesitaba, encontré material muy avanzado o relacionado con otras áreas de Java, como interfaces o bases de datos. Incluso conseguí algún libro para novatos en el que no entendí ni la primera frase. A día de hoy, ya sabiendo programar, los vuelvo a leer y me doy cuenta de que la estructura y el orden de los temas son pésimos.

No obstante, no tiré la toalla y fui recopilando información de donde podía. No fue un camino sencillo, ya que algunas veces estaba seguro de haber aprendido algo y, poco después, descubría que en realidad lo había entendido mal. Aunque, con perseverancia, se puede conseguir casi todo y, al final, me vi resolviendo ejercicios complejos y con una recopilación de apuntes que ahora sí explicaban todo paso a paso y de forma estructurada.

Soy consciente de que ha habido, hay y habrá mucha más gente en la misma situación. Así que la idea de este libro es ayudar a todas esas personas que, por el motivo que sea, deciden aprender a programar en Java. Mi intención desde el principio fue crear un material que los programadores avanzados lo vieran y pensaran: “Ojalá hubiera tenido este libro cuando empecé a programar”. Por mi parte, puedo decirlo sin remordimiento: ojalá lo hubiera tenido.

Antes de continuar, me gustaría dejar claras algunas cosas. No creo que memorizar conceptos sea la mejor manera de aprender nada y mucho menos a programar. Es cierto que en los siguientes temas verás muchos conceptos de teoría que son fundamentales y debes entenderlos, pero no es necesario que pierdas el tiempo memorizándolos. Poco a poco, y a base de realizar ejercicios y releer la teoría, toda esa información se irá haciendo un hueco en tu cabeza y se quedará ahí, casi sin que te esfuerces en ello.

Siguiendo con el tema, debes entender que lo que estás tratando de aprender es un lenguaje. Lo que quiero decir es que, por ejemplo, en castellano existen muchas formas para referirse al mismo concepto. En Java va a pasar lo mismo. Si necesitas crear un programa que organice tu agenda semanal, ese código se va a poder escribir de muchas maneras distintas.

En este tema vas a encontrarte con un montón de ejercicios resueltos. La idea no es que los memorices. Yo los he resuelto de una forma, pero quizás tú encuentres otra que sea incluso más eficiente u organizada. Una vez que tengas algo de experiencia programando, te darás cuenta de que lo más complicado no es la sintaxis en sí, sino pensar cómo podemos utilizarla para solucionar ese problema que debemos resolver.

Lo que me gusta de programar es que puedes ver cada ejercicio como un reto e incluso como algo divertido. Hay gente que hace sudokus, otros, sopas de letras, tú puedes resolver ejercicios de Java. Puede que ahora te haga gracia la idea, pero pronto descubrirás la satisfacción de resolver un problema que te traía de cabeza desde hacía horas, días o incluso semanas.

Como consejo personal, te sugiero que no te fuerces e intentes aprender todo de golpe. En mis primeros pasos de aprendizaje, trataba de dedicarle decenas de horas a la semana, y mi cabeza acabó saturándose. Es cierto que iba progresando bastante, pero había muchos puntos clave que no acababa de entender y era un poco frustrante. Por motivos laborales, no me quedó más remedio que mantenerme un mes alejado de Java y, cuando lo retomé, me llevé una grata sorpresa. Con la cabeza fresca, la mayoría de cosas que no comprendía ahora estaban totalmente claras, y todo el bloqueo mental que tenía había desaparecido. Así que te recomiendo que vayas paso a paso.

Este libro tiene una parte de historia, otra de teoría y otra de práctica. Creo que la historia lo hace más ameno y distinto a otras publicaciones. Además, es una motivación para seguir leyendo. Siéntete libre de saltarte esas páginas si no te interesan. Por otro lado, si ya conoces la teoría, puedes ir directamente a los ejercicios. He tratado de que el libro sea versátil y pueda satisfacer las necesidades de todos los lectores. El objetivo ha sido recopilar en un mismo tomo todo lo necesario para que cualquier persona, independientemente de su conocimiento previo, pueda acabar programando con soltura.

Obviamente, la historia es completamente ficticia y no se basa en ninguna persona o escenario real. Ahora sí, ya después de estas aclaraciones, te deseo suerte en tu aprendizaje.

Índice del libro

Capítulo 4: Pasando el rato en comisaría / Trabajando con condicionales

- Importancia de las condicionales
- Estructura básica del if
- Declaraciones else y else if
- Anidación de condicionales
- Switch case
- Operadores de comparación
- Operadores lógicos

Capítulo 5: Conociendo a Bud. Aprendiendo a utilizar los bucles o loops

- Definición e importancia de los bucles
- Bucle for
- Bucle while
- Bucle do-while
- Concepto de break
- Concepto de continue
- Manipulación de cadenas: charAt() y length()

Capítulo 6: Ayudando a Bud con sus tareas / Números aleatorios

- Importancia de los números aleatorios
- Generación de números aleatorios con Math.random
- Redondeo de números. Parte II. Usando Math.round()

Capítulo 7: La visita del primo de Chani / Arrays

- Definición e importancia de los arrays
- Declaración y creación de arrays
- Asignación de valores a arrays
- Recorrido de arrays con bucles

Capítulo 8: El trabajo de Bud / Matrices 2D

- Definición e importancia de las matrices 2D
- Declaración, creación y asignación de matrices 2D
- Acceso a elementos de una matriz 2D
- Inicialización de matrices 2D
- Recorrido de matrices 2D con bucles anidados
- Suma de elementos en una matriz
- Búsqueda de elementos específicos en una matriz

Capítulo 9: Crear un dispositivo de distracción / Con bucles, condicionales y scanner

- Importancia del bucle for-each
- Sintaxis del bucle for-each
- Ventajas y limitaciones del bucle for-each

Capítulo 10: El camino al despacho de Vicente / Trabajando con fecha y hora

- Clases LocalTime, LocalDate y LocalDateTime
- Obtener fecha, hora y fecha-hora actual
- Crear fechas y horas específicas
- Sumar y restar días, horas
- Comparar fechas
- Formatear fechas
- Calcular diferencia entre fechas

Capítulo 11: Hackear el sistema de seguridad

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 12: Planeando la vuelta a la prisión / Try & catch

- Importancia de los bloques try y catch
- Tipos de excepciones comunes
- Estructura básica de try-catch
- Múltiples bloques catch
- Bloque finally

Capítulo 13: Recopilando pruebas contra Phil

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 14: Escapar por la puerta principal / Leer y escribir en archivos de texto

- Importancia de leer y escribir archivos
- Importar clases para leer archivos
- Leer archivos de texto
- Importar clases para escribir archivos
- Escribir en archivos de texto

Capítulo 15: Burlar el sistema de contadores de tiempo

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 16: Los preparativos finales / for-each

- Importancia de los bucles for-each
- Ventajas y limitaciones de for-each
- Sintaxis del bucle for-each

Capítulo 17: Decodificar sistemas de comunicación

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 18: Interviniendo las conversaciones de los guardias

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 19: Funciones y POO

- Diferencia entre métodos y funciones
- Modificadores de acceso: public, protected, private
- Métodos que devuelven valor vs void
- Crear y llamar métodos en la clase Main
- Crear y llamar métodos en otras clases
- Diferencia entre clase e instancia
- Diferencia entre métodos estáticos y dinámicos
- Creación de objetos y uso de constructores
- Definición de métodos getter y setter

Capítulo 20: La fase final, creamos una herramienta de desbloqueo de puertas

- Utilizando nuestros conocimientos para continuar con la historia.

Capítulo 21: El desenlace

- Desenlace y despedida

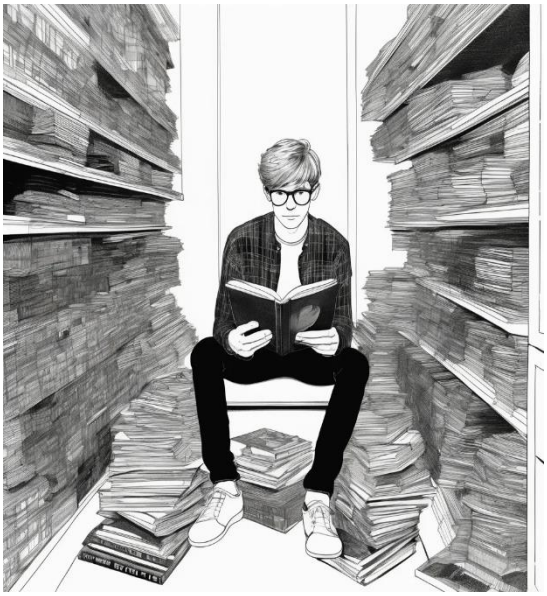
Capítulo 1 introducción.

Una vida despreocupada en Salamanca / Introducción y tipos de datos

Durante toda mi vida me he sentido intocable. No he tenido grandes preocupaciones y siempre he contado con el apoyo de amigos y familiares. No era consciente de que, en cualquier momento, mi suerte podría cambiar radicalmente. La mayoría de la gente piensa que tiene todo bajo control, pero hay cosas que no dependen de uno mismo. Son precisamente esas cosas las que pueden pillarte por sorpresa en cualquier momento. Eso es exactamente lo que me pasó a mí.

Tengo entendido que quieres aprender a programar en Java. No te preocupes, yo voy a enseñarte y, además, lo haré de forma sencilla. En realidad, la programación no es difícil, al menos no tan difícil como parece. Pero, antes de empezar, deja que me presente.

Mi nombre real es Pelayo, aunque todos me llaman Peli. Soy estudiante de primer año de universidad. Vivo en Salamanca y me gustaría dedicarme a la informática. De momento, las cosas no están saliendo como pensaba. Lo cierto es que ya hemos entrado en la segunda mitad del semestre y apenas he aparecido por clase. Muchos de los profesores ni siquiera me conocen.



Para ser honesto, he dedicado la mayor parte del tiempo a pasarlo bien y conocer gente. Sé que probablemente, debería empezar a estudiar cuanto antes, pero esa no es mi prioridad.

Este fin de semana voy a celebrar una fiesta en mi piso y quiero que sea perfecta. No me gusta alardear, pero creo que, poco a poco, las reuniones en mi casa se están haciendo muy populares. En realidad, no vivo solo; por lo tanto, no es exactamente mi piso. Comparto casa con otros dos universitarios, Rich y Chani. La verdad es que son mejores estudiantes que yo, aunque tampoco se puede decir que sean muy responsables. No nos conocíamos de antes, pero en poco tiempo nos hemos hecho buenos amigos.

Rich es un estudiante de intercambio inglés, aunque habla un perfecto castellano. Siempre va muy bien vestido y pasa muchas horas delante del espejo. Eso no significa que sea arrogante. Al contrario, es una persona muy humilde y carismática.

Cae bien a todo el mundo y siempre es el centro de atención. Eso le gusta. Durante los últimos meses hemos pasado mucho tiempo juntos y se ha convertido en mi mejor amigo.

En realidad, fue la primera persona que conocí aquí, o al menos, la primera con quien empecé a entablar amistad. En mi primera clase me tocó sentarme a su lado. No recuerdo por qué empezamos a hablar, pero desde el primer minuto parecía como que lo conocía de toda la vida.

Ese mismo día me comentó que no quería vivir en la residencia de estudiantes. Yo aún no tenía alojamiento, así que le propuse compartir casa. Aceptó de inmediato. El problema era que, entre dos, no nos alcanzaba para pagar el alquiler de un piso. Necesitábamos a una tercera persona. Pusimos un anuncio en el tablón online de la universidad y un joven asiático nos contactó a las pocas horas. Se trataba de Chani, quien se convirtió en nuestro nuevo amigo.

Chani también es un estudiante de intercambio, concretamente de Corea del Sur. Se está esforzando por aprender nuestro idioma, pero no le resulta fácil. A veces, pienso que para mí sería imposible aprender coreano.

Es bastante introvertido, pero eso no le impide asistir a todas las fiestas. Pasa casi todo el día encerrado en su habitación, durmiendo o jugando con el ordenador. Solamente sale de su cuarto para cocinar e ir al baño. Parece un vampiro. En cuanto cae la noche, decide que es momento para activarse y pasar el rato con nosotros.

Somos un grupo de amigos poco convencional, pero nos funciona bien. La convivencia es muy buena. No discutimos nunca y respetamos mucho nuestro espacio. Un par de veces a la semana pedimos pizza y cenamos todos en nuestro amplio salón. Durante la semana apenas traemos invitados, pero no nos sentimos solos ya que nos tenemos los unos a los otros. No podría haber tenido más suerte eligiendo a mis compañeros.

Nos encanta la zona común de nuestra casa. Es muy grande y espaciosa. Cuenta con unas ventanas enormes y acceso a la terraza. Hemos conseguido unos sofás muy baratos e incluso un pequeño futbolín con el que se entretienen las visitas. Nunca había vivido en una casa mejor que esta y el precio es mucho más barato de lo que parece.

Paso casi todo mi día aquí acostado, muchas veces en compañía de Rich. Él sabe que apenas estoy asistiendo a las clases de la universidad y le preocupa que no vaya a aprobar ninguno de mis exámenes. Así que se ha ofrecido a enseñarme a programar en Java. Sabe bastante del tema y explica las cosas de forma sencilla y clara.

Pero bueno, no me enrollo más. Hechas las presentaciones, creo que es un buen momento para recordar las enseñanzas de Rich. Así que, en lo que queda del capítulo, me centraré en repasar y afianzar algunos conceptos. Será algo muy básico, simplemente una pequeña introducción al mundo de Java y sus tipos de datos más utilizados.

Introducción. Definición de POO y presentación de los tipos de datos.

Lo primero que siempre se menciona cuando se enseña Java es que es un tipo de programación orientada a objetos. Da igual de dónde saques la información o quién te lo esté explicando, siempre vas a encontrarte con frases como estas:

"La programación orientada a objetos (POO) es un paradigma de programación que se basa en el concepto de "objetos". Estos objetos son entidades que combinan datos y funciones (llamadas métodos) relacionadas en una sola unidad. La idea principal detrás de la programación orientada a objetos es modelar el mundo real en el software, lo que permite representar de manera más precisa y eficiente los elementos y procesos del mundo real en nuestros programas".

Si no has entendido nada, no te preocupes. Si lo has hecho, enhorabuena, vas un paso por delante de la mayoría de novatos. Cuando programas en Java, al igual que en muchos otros lenguajes, el código no se escribe de forma lineal. Eso sería poco práctico. Hay que tener en cuenta que ciertos programas o aplicaciones complejas tienen decenas o incluso cientos de miles de líneas de código.

Es mucho más práctico dividir ese código en pequeños grupos que interactúen entre sí. Esto nos ayudará, por ejemplo, a identificar y solucionar errores en el código de forma mucho más rápida, a reutilizar código ya escrito y a evitar la duplicación del mismo, a dividir las tareas entre varias personas de forma sencilla, a realizar un mantenimiento del código eficaz y organizado, a implementar características nuevas en la aplicación y, en definitiva, a hacer nuestra vida más sencilla.

Atención ahora, porque voy a plantear una posible pregunta de examen: **dime los cuatro conceptos fundamentales en los que se basa la programación orientada a objetos.**

La POO se basa en cuatro conceptos fundamentales:

Clases: *Una clase es como un plano o plantilla que define las características y comportamientos comunes de un tipo particular de objeto. Define los datos que puede contener un objeto (llamados atributos) y las operaciones que puede realizar (llamadas métodos). Por ejemplo, si pensamos en un objeto "coche", la clase correspondiente puede definir los atributos como color, modelo y velocidad, y los métodos como acelerar y frenar.*

Objetos: *Los objetos son instancias concretas de una clase. Cada objeto tiene su propia copia de los datos definidos en la clase, pero comparte los métodos definidos en la misma. Por ejemplo, si tenemos una clase "Coche", un objeto específico podría ser un coche rojo modelo Toyota Corolla con una velocidad actual de 60 km/h.*

Encapsulación: *La encapsulación es el concepto de ocultar los detalles internos de un objeto y exponer solo la funcionalidad necesaria a través de interfaces bien definidas. Esto se logra definiendo los atributos de la clase como privados y proporcionando métodos públicos (getters y setters) para acceder y modificar estos atributos de manera controlada. La encapsulación ayuda a mantener el código modular, facilita el mantenimiento y reduce el riesgo de errores.*

Herencia: *La herencia es un mecanismo que permite que una clase (llamada subclase o clase derivada) herede los atributos y métodos de otra clase (llamada superclase o clase base). Esto permite la reutilización de código y la creación de jerarquías de clases, donde las clases más específicas pueden agregar o modificar el comportamiento heredado. Por ejemplo, una clase "Vehículo" puede ser una superclase de las clases "Coche" y "Camión", que heredan sus características básicas, pero pueden tener comportamientos específicos adicionales.*

No es necesario que aprendas estos términos de memoria. Vamos a trabajar con ellos en los siguientes temas y, poco a poco, se irán afianzando en tu cabeza. Aunque te conviene familiarizarte con ellos y conocer su existencia. De momento, nos quedamos simplemente con la idea de que Java es un lenguaje orientado a objetos.

Programar es una actividad poco intuitiva en la que se utilizan muchos términos desconocidos. Poco a poco, todo comenzará a tener sentido, pero recuerda que deberás ser constante y no tirar la toalla.

Tipos de datos y variables:

No sé si te gusta cocinar, pero seguro que alguna vez has tratado de preparar alguna receta en tu casa. Si comparamos la programación con la cocina, podríamos decir que los tipos de datos son los ingredientes, la materia prima que utilizamos para crear nuestro programa.

Las variables, en cambio, son los lugares donde guardas esos ingredientes para usarlos más adelante en tu receta, como frascos o tazones que contienen harina, azúcar, etc.

- **Tipos de datos.**

En Java, los tipos de datos se clasifican en dos grupos principales: tipos de datos primitivos y tipos de datos de referencia.

1) **Tipos de datos primitivos:** Son los tipos básicos integrados en el lenguaje. Existen ocho en total:

- byte: Un tipo entero de 8 bits que permite almacenar valores entre -128 y 127.
- short: Un entero de 16 bits que puede manejar valores entre -32,768 y 32,767.
- int: Un entero de 32 bits que abarca valores entre -2^31 y 2^31 - 1.
- long: Un entero de 64 bits que permite representar valores desde -2^63 hasta 2^63 - 1.
- float: Un número de punto flotante de 32 bits, con una precisión de aproximadamente 6-7 dígitos decimales.
- double: Un número de punto flotante de 64 bits, con una precisión de alrededor de 15 dígitos decimales.
- char: Representa un único carácter Unicode de 16 bits.
- boolean: Solo puede tener dos valores posibles: true o false.

2) **Tipos de datos de referencia:** En lugar de contener directamente los datos, estos tipos almacenan referencias a objetos en la memoria. Algunos de los más comunes incluyen:

- String: Una secuencia de caracteres utilizada para representar texto. A diferencia de muchos lenguajes, en Java no es un tipo primitivo, sino de referencia.
- Arrays: Estructuras que almacenan múltiples elementos del mismo tipo de manera contigua en la memoria.

- **Variables**

En Java, **una variable es un contenedor que almacena datos** que pueden cambiar durante la ejecución de un programa. Repito: los datos dentro de una variable pueden cambiar a medida que se va ejecutando el programa. Las variables tienen un nombre que se utiliza para referirse a ellas y un tipo de datos que especifica qué tipo de valores pueden contener.

Por ejemplo: int variable = 7;

En la variable anterior podemos ver que el tipo de dato es int y que su nombre es “variable”. Además, vemos que, en estos momentos, su valor es 7. Puede que ese valor durante la ejecución del programa varíe, pero ahora es 7.

Cuando declaramos una variable, debemos hacerlo dependiendo del tipo de dato que necesitemos. Las variables quedan condicionadas por el tipo de dato que albergan. Una variable numérica no podrá albergar datos de texto. Por lo tanto, podemos referirnos a ellas con la misma calificación que damos a los tipos de datos. Eso provoca que, en muchas ocasiones, veamos escritos indistintamente ambos términos y, al final, resulte confuso.

Es más sencillo explicarlo con un ejemplo. Imagínate que necesitamos crear una variable llamada “nombre”. Esa variable albergará un tipo de dato de texto (String), en concreto con el nombre de Pedro. Así que aquí la tenemos:

```
String nombre = "Pedro";
```

Esa variable es ahora una variable de tipo “String” ya que alberga un tipo de dato de texto.

Hay bastantes tipos de datos y es bueno que conozcas todos. Aunque, de momento y para los ejercicios que vamos a ir haciendo, simplemente te hará falta conocer unos pocos. Como he comentado en el párrafo anterior, los tipos de datos están ligados a las variables. Por eso, y para no liarnos, a partir de ahora utilizaremos solamente el término variable.

Veamos ahora una definición un poco más extensa sobre las variables que vas a necesitar conocer:

Variables enteras (int): Las variables enteras se utilizan para almacenar números enteros. Por ejemplo, podemos usar una variable entera para representar la edad de una persona o la cantidad de productos en inventario. En Java, declaramos una variable entera utilizando la palabra clave "int", seguida del nombre de la variable y opcionalmente, su valor inicial.

Por ejemplo:

int edad = 25;

Variable de texto (String): Se utiliza para almacenar secuencias de caracteres, es decir, texto. Para definir una variable de tipo String, simplemente usamos la palabra clave String, seguida del nombre de la variable y opcionalmente, asignamos un valor. Aquí tienes un ejemplo de cómo definir una variable String en Java:

String miNombre = "Juan";

Variables de punto flotante (float y double): Estas variables se utilizan para almacenar números con decimales. La diferencia entre float y double radica en la precisión del número decimal que pueden almacenar. En Java, usamos "float" para números de precisión simple y "double" para números de precisión doble. Por ejemplo:

float altura = 1.75f;
double pi = 3.14159;

Variables booleanas (boolean): Una variable booleana solo puede tomar uno de dos valores: verdadero (true) o falso (false). Estas variables son útiles para representar condiciones lógicas. Por ejemplo:

boolean esMayorDeEdad = true;

Variables de caracteres (char): Las variables de caracteres se utilizan para almacenar un solo carácter. Por ejemplo, podemos usar una variable de caracteres para representar una letra o un símbolo. En Java, declaramos una variable de carácter utilizando la palabra clave "char". Por ejemplo:

char inicial = 'A';

Las ventajas de especificar el tipo de datos que una variable puede contener son las siguientes:

1. **Hacer nuestro código más sencillo y fácil de entender:** si no especificáramos el tipo de datos que una variable puede contener, el código resultaría casi imposible de entender para ti y para otros programadores. Saber qué tipo de datos espera una variable ayuda a comprender cómo se utilizará en el programa.
2. **Detección de errores:** Si intentas asignar un valor de un tipo incorrecto a una variable, se te advertirá sobre este error. Esto ayuda a encontrar errores incluso antes de que el programa se ejecute, lo que facilita la tarea de programación y evita errores en tiempo real.
3. **Mejor rendimiento:** En algunos casos, el tipado de variables puede mejorar el rendimiento del programa, ya que el código puede ser optimizado sabiendo los tipos de datos que se están utilizando.
4. **Facilita el mantenimiento del código:** Cuando vuelvas a revisar o modificar tu código en el futuro, tener tipos de datos indicados puede ayudarte a recordar cómo se supone que se deben usar ciertas variables. Si es otra persona quien revisa el código, esto se vuelve todavía más útil. Esto facilita el mantenimiento y la actualización del código a medida que el proyecto evoluciona.

Diferencia entre la asignación y la declaración de variables.

Ahora que ya sabemos qué tipos de datos hay, necesitamos saber cómo implementarlos en nuestro código. Ya hemos visto algunos ejemplos en los párrafos anteriores:

```
String miNombre = "Juan";
int edad = 25;
double pi = 3.14159;
boolean esMayorDeEdad = true;
char inicial = 'A';
```

Como puedes apreciar siempre empezamos escribiendo el tipo de dato que necesitamos. Por ejemplo, String, int, double, boolean, char. Después, el nombre que hemos elegido para la variable. Todo ello seguido por un signo de igual. Posteriormente, el valor asignado y siempre terminando en punto y coma.

Fíjate en que, para el tipo de dato String, el texto asignado debe estar siempre entre comillas dobles: "Juan". En el caso de un char, el carácter que queramos en nuestra variable debe estar entre comillas simples: 'A'.

Ahora bien, no siempre es necesario asignar un valor a nuestra variable. Simplemente podemos declararla y hacer uso de ella posteriormente. Por ejemplo, empezamos un ejercicio que nos pide crear una variable String llamada nombre y otra variable int denominada matrícula. Como puedes ver, no es necesario asignarles ningún valor.

```
String nombre;
int matricula;
```

Puede que en estos momentos te parezca que esto no tiene sentido. ¿Por qué no asignar un valor nada más crear una variable? Es básicamente por motivos de orden y estructura. En cuanto empieces a tener algo de soltura con los ejercicios, verás que todo tiene una razón de ser.

Es posible que te estés preguntando cómo asignar un valor a una variable que ya ha sido declarada. Vamos a continuar con los dos ejemplos que hemos visto dos párrafos más arriba (nombre y matrícula). Simplemente lo haremos de la siguiente forma:

```
nombre = "Juan";
matricula = 55;
```

Lo único que debemos hacer es tener en cuenta el tipo de dato que queremos utilizar (String e int en el ejemplo). Después, simplemente seguiremos la estructura normal que ya habíamos visto: nombre de la variable, signo igual y el valor que queremos añadir.

Convención en el nombre de las variables en Java

En Java, las variables siguen ciertas reglas de convención para nombrarlas. Aquí te explico las principales:

Empiezan con letra minúscula: *El nombre de una variable debe comenzar con una letra minúscula. Por ejemplo: edad, estatura, anchura, etc.*

Usa camelCase: *Esto significa que, si el nombre de la variable está compuesto más de una palabra, la primera letra de cada palabra después de la primera se escribe en mayúscula, sin espacios ni guiones. Por ejemplo: edadActual, estaturaCompleta, anchuraTotal, etc.*

Significativo y descriptivo: *Intenta usar nombres que indiquen lo que se está representando. Por ejemplo, en lugar de x o y, podrías usar edad o saldoCuenta.*

Evita caracteres especiales: *Solo se permiten letras, números y el guion bajo _. Otros caracteres como espacios, guiones (-), puntos (.), entre otros, no deben de ser usados en el nombre de una variable.*

No uses palabras clave de Java: *Evita usar palabras reservadas por Java (como int, float, while, etc.) como nombres de variables, si lo haces, puede que después haya errores en tu código.*

No uses acentos ni caracteres especiales: *Java no permite utilizar acentos ni caracteres especiales en los nombres de variables.*

Siguiendo estas reglas, tus programas serán más fáciles de leer y entender para ti y para otras personas que lean tu código.

Resumen de lo aprendido

En este primer capítulo, has iniciado tu incursión en el mundo de la programación en Java. ¡Enhorabuena! No todo el mundo se atreve con este reto. Ahora ya se puede decir que tienes una idea general de cómo se estructura Java y lo que significa el término “programación orientada a objetos” (POO).

Además, ya sabes cuáles son los cuatro pilares fundamentales de la POO: clases, objetos, encapsulación y herencia. Si alguno de estos conceptos aún no te queda claro, no te preocupes, es normal. No creo que mucha gente lo entienda todo a la primera. Siendo realistas, no hemos profundizado mucho en el tema y aún quedan muchos interrogantes por resolver. Aunque desde este punto, aprender nuevos conceptos te resultará mucho más sencillo.

A estas alturas, seguramente te sientas más a gusto hablando de los tipos de datos y de las variables. Son conceptos más concretos y fáciles de asimilar. Cuando empecemos con la parte práctica, van a ser indispensables en todos nuestros ejercicios. Sabemos que existen tipos de datos primitivos y de referencia. Además, hemos hecho hincapié en la importancia que tiene en Java la especificación de los tipos de datos. Con ello se consigue mejorar la claridad, detectar errores y facilitar el mantenimiento del código.

También hemos tenido tiempo para aprender cuál es la diferencia entre la asignación y la declaración de variables.

Aunque de momento no sea una prioridad en tu aprendizaje, has podido echar un vistazo a las convenciones de nomenclatura en Java para nombrar variables. Pronto vas a empezar a trabajar con variables y deberás crear tu propio código, en el que tendrás total libertad para elegir los nombres de las variables. Mi consejo es que trates de seguir esta convención para que te vayas acostumbrando a seguir buenas prácticas.

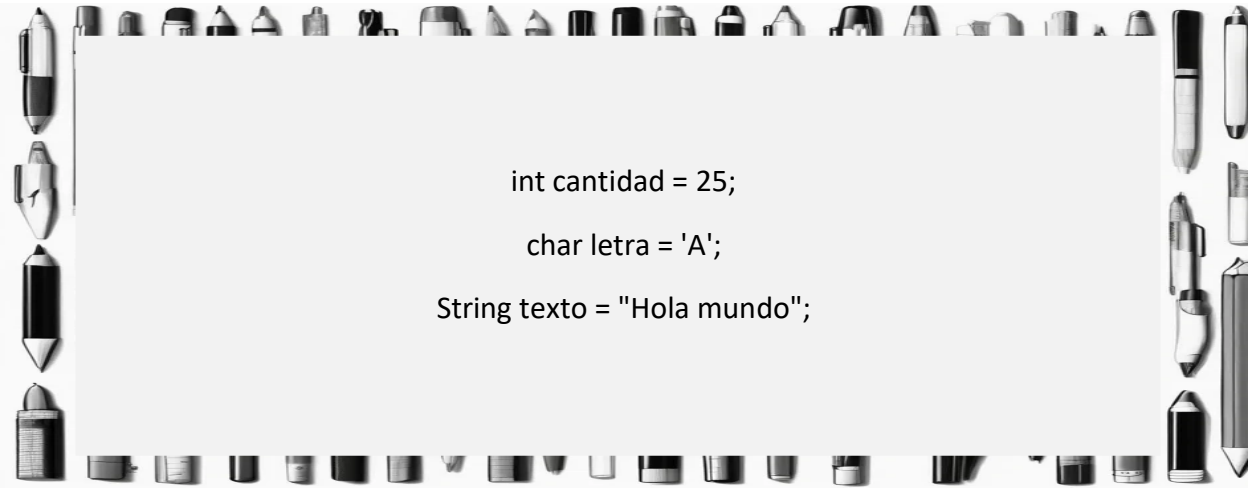
Como resumen, diremos que ya has dado tu primer paso en el mundo de la programación. Aún queda un largo camino; entiendo que la teoría a veces es un poco aburrida, pero es totalmente necesaria. Esta introducción te servirá para establecer una base sólida con la que comprender los conceptos más avanzados que se abordarán en los siguientes capítulos.

Seguro que ya tienes ganas de meterte en faena y empezar a escribir tus propios programas. Si te soy sincero, es mi parte favorita. Así que no te preocupes, porque pronto comenzaremos a solucionar ejercicios y a utilizar los conocimientos teóricos que estamos adquiriendo.

Ejercicios del tema

Ejercicio 1. Declara tres variables de diferentes tipos: una variable de tipo int, otra de tipo String y una tercera de tipo char. Asigna directamente el valor 25 a la variable de tipo int, "Hola mundo" a la variable de tipo String y el carácter 'A' a la variable de tipo char.

Solución:



```
int cantidad = 25;

char letra = 'A';

String texto = "Hola mundo";
```

Usa nombres descriptivos: Elige nombres de variables que sean claros y significativos. Un buen nombre de variable facilita la comprensión del código.

Ejercicio 2. Declara tres variables de diferentes tipos: una variable de tipo int, otra de tipo String y una tercera de tipo char. Luego, en distintas líneas de código, asígnale a cada una los siguientes valores: el valor 25 a la variable de tipo int, el valor "Hola mundo" a la variable de tipo String y el carácter 'A' a la variable de tipo char.

Solución:

```
int cantidad;
char letra;
String texto;

cantidad = 25;
letra = 'A';
texto = "Hola mundo";
```

Ejercicio 3. Declara y asigna un valor a una variable de tipo double que represente el número π (pi).

Solución:

```
double pi = 3.14159;
```

Ejercicio 4. En un misterioso mundo de aventuras, estás explorando un antiguo templo lleno de trampas. Debes crear un programa en Java que declare una variable booleana llamada activado y asignarle el valor true, lo que indica que has activado el mecanismo de seguridad del templo.

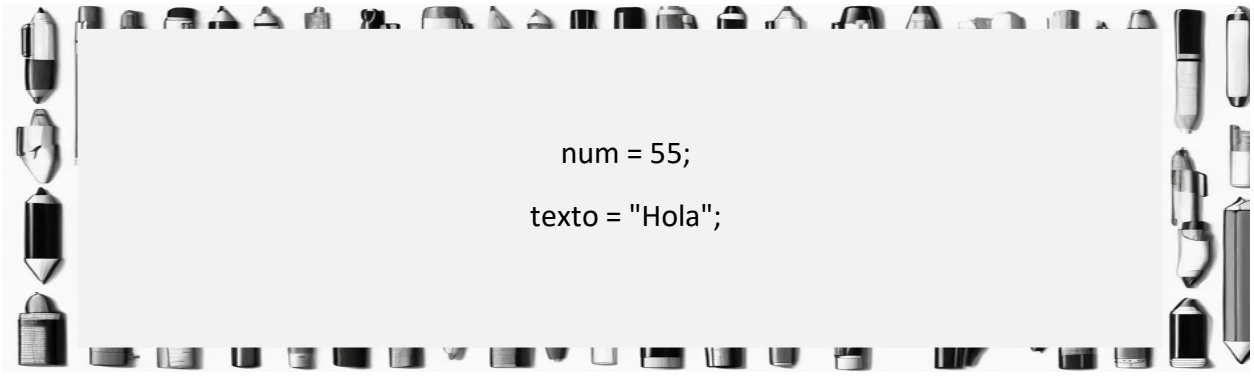
Solución:

Inicializa tus variables: En tus primeros programas, inicializa tus variables cuando las declares. Esto evita errores y comportamientos inesperados en tu programa.

Ejercicio 5. En las siguientes líneas de código hay declaradas dos variables: num y texto. Asígnales el valor 55 y "Hola", respectivamente.

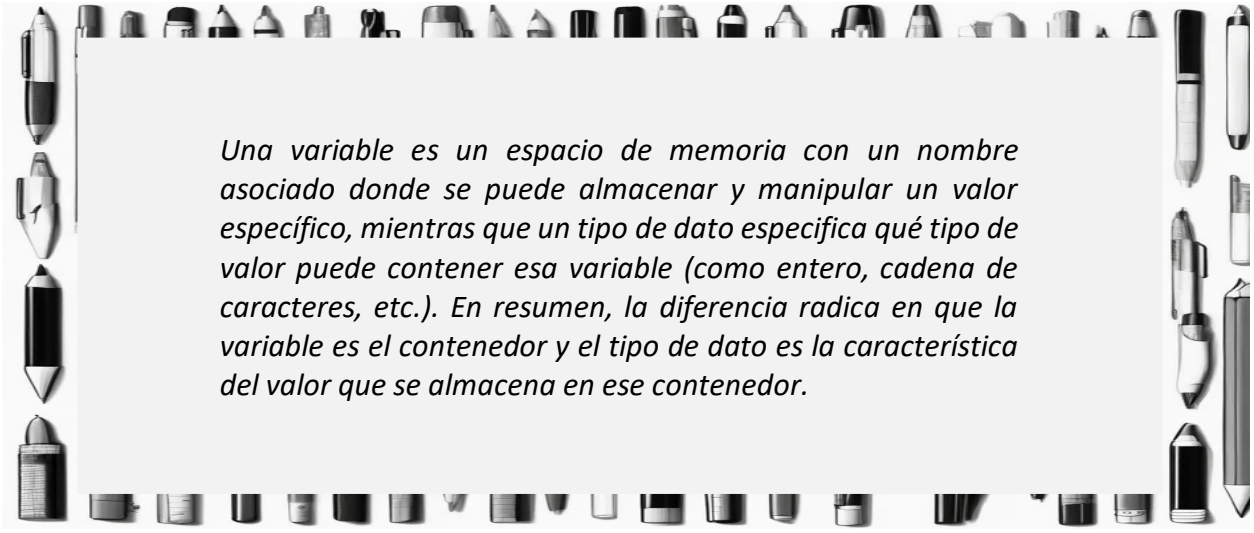
```
public class Main {

   public static void main(String[] args) {
      int num;
      String texto;
   }
}
```

Solución:



Ejercicio 6. Dime la diferencia entre una variable y un tipo de dato.

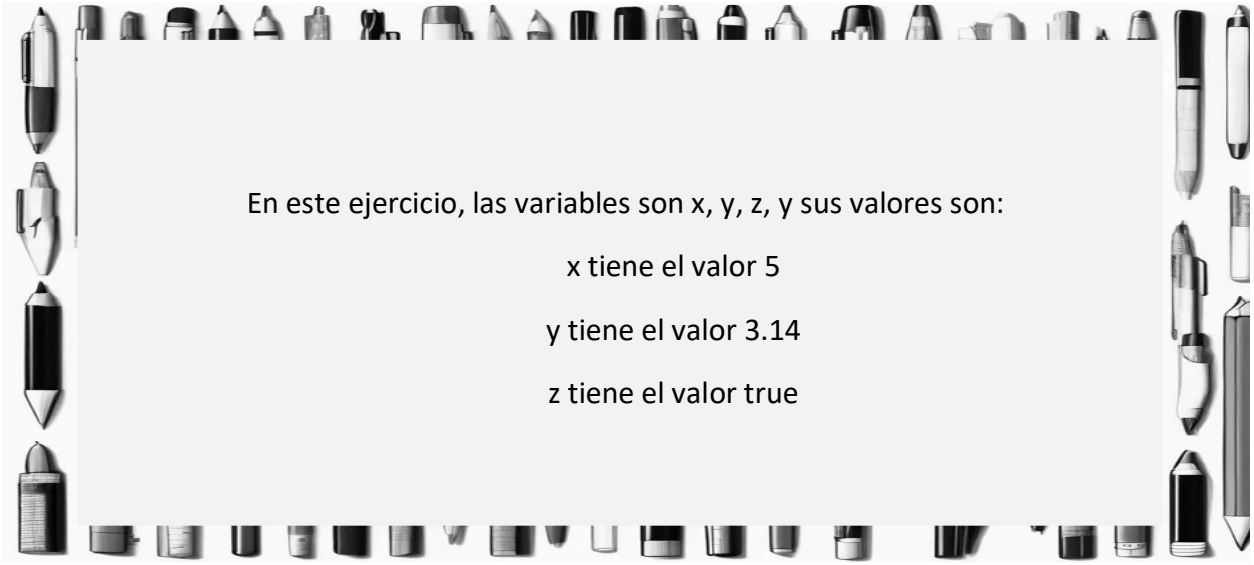
Solución:



Ejercicio 7. Mira este código e identifica las variables y sus respectivos valores.

```
public class Main {
   public static void main(String[] args) {
      int x = 5;
      double y = 3.14;
      boolean z = true;

      System.out.println("El valor de x es: " + x);
      System.out.println("El valor de y es: " + y);
      System.out.println("El valor de z es: " + z);
   }
}
```

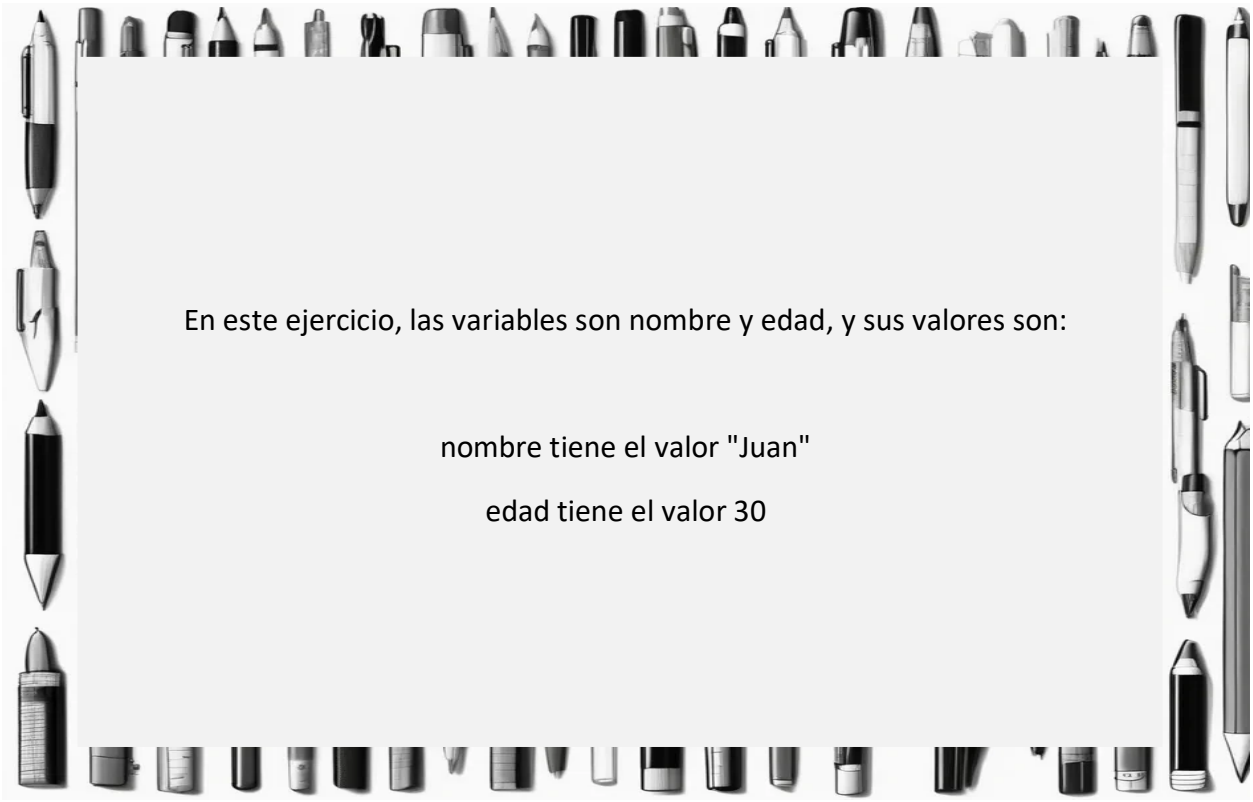
Solución:



Ejercicio 8. Mira este código e identifica las variables y sus respectivos valores.

Solución:

```
public class Main {
   public static void main(String[] args) {
      String nombre = "Juan";
      int edad = 30;

      System.out.println("El nombre es: " + nombre);
      System.out.println("La edad es: " + edad);
   }
}
```



En este ejercicio, las variables son nombre y edad, y sus valores son:

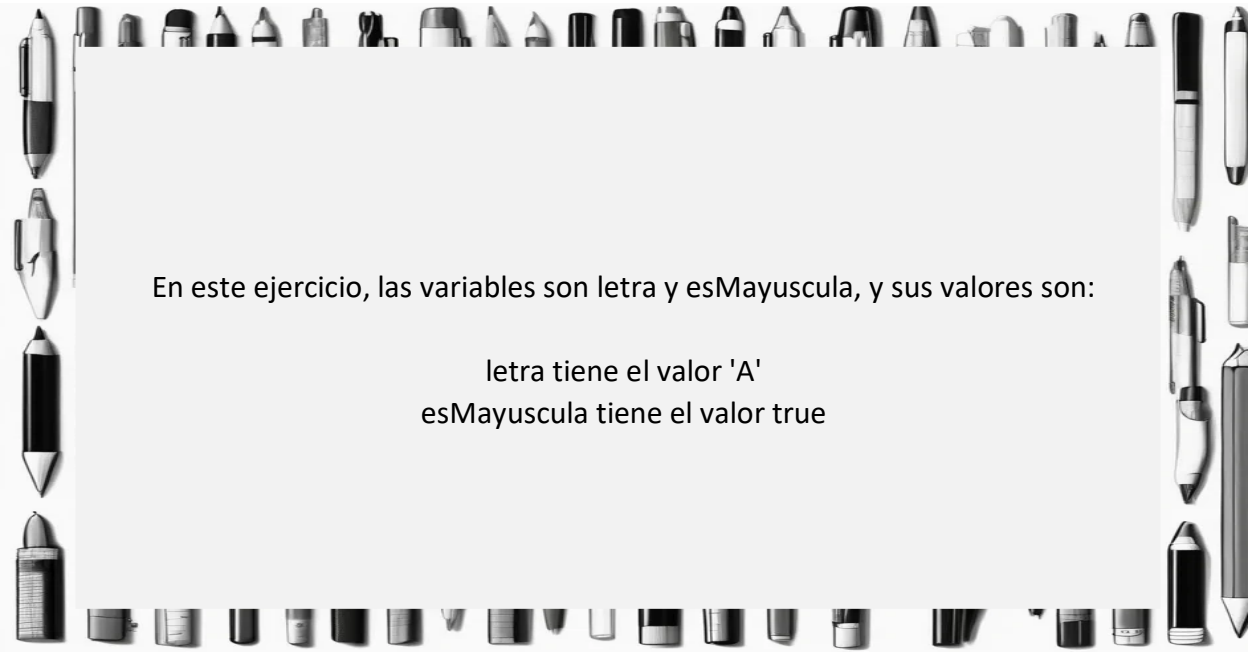
nombre tiene el valor "Juan"

edad tiene el valor 30

Ejercicio 9. Mira este código e identifica las variables y sus respectivos valores.

```
public class Main {
   public static void main(String[] args) {
      char letra = 'A';
      boolean esMayuscula = true;

      System.out.println("La letra es: " + letra);
      System.out.println("¿Es mayúscula?: " + esMayuscula);
   }
}
```

Solución:

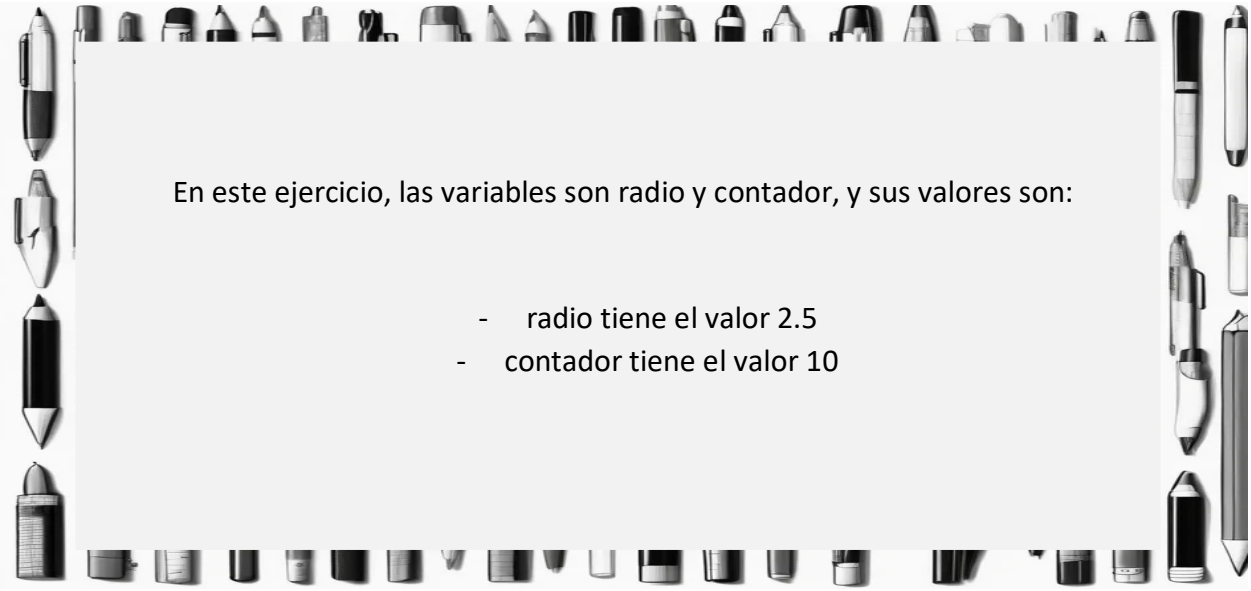


Sigue la convención de nombres: Usa la convención camelCase para nombrar variables en Java (por ejemplo, miVariable). Esto mejora la legibilidad y se ajusta a las prácticas comunes en Java.

Ejercicio 10. Mira este código e identifica las variables y sus respectivos valores.

```
public class Ejercicio10 {
   public static void main(String[] args) {
      double radio = 2.5;
      int contador = 10;

      System.out.println("El radio es: " + radio);
      System.out.println("El contador es: " + contador);
   }
}
```

Solución:



Capítulo 2.

La desaparición de Chani / Mostrar datos en consola

¿Recuerdas que dije que iba a dar una fiesta en mi casa? Pues fue un gran éxito. No sé cuántas personas podría haber allí, pero no paraba de aparecer más y más gente. No conocía ni a la mitad, pero eso no me preocupaba. Se trataba de pasarlo bien.

Antes de traer invitados a casa, nos gusta ser precavidos, así que tenemos una costumbre. Horas antes de que empiecen a llegar los asistentes, dejamos todas las cosas de valor en un armario situado dentro de la habitación de Chani. Ese armario se cierra con un candado y solo Chani tiene la llave. De esta forma, nos sentimos más seguros en caso de que se descontrole la situación. Al día siguiente, despertamos a Chani, abre el armario y recuperamos nuestras preciadas pertenencias. Luego comemos algo y nos pasamos casi toda la tarde tumbados en el sofá viendo alguna serie. Es una gran parte de nuestra rutina estudiantil.

El problema es que esta vez todo fue distinto. A la mañana siguiente después de la fiesta, todo era un caos. Estaba mucho más desordenado de lo habitual y había restos de bebida por todas partes. Por no hablar de la increíble cantidad de botellas y vasos desperdigados por todas las habitaciones.

Yo me desperté alrededor de las dos de la tarde, bastante hambriento. Me acerqué a la cocina y abrí el frigorífico. Descubrí que, desgraciadamente, no había nada de comer, y eso afectó mucho a mi estado de ánimo. Entonces empecé a plantearme mis opciones. Salir a la calle a comprar estaba totalmente descartado. Me encontraba muy cansado y no me apetecía moverme. Pedir comida parecía lo más sensato, pero no sabía exactamente qué quería.

Mientras lo decidía, me tumbé en el sofá y escribí a mis compañeros. No quería comer solo y, además, pensé que probablemente ellos también tendrían hambre al despertar. A los pocos minutos, Rich salió de la habitación. Había leído el mensaje y quería comer pizza. Chani no respondió, pero, aun así, pedimos también para él.

Las pizzas acabaron llegando sobre las tres y media de la tarde. Chani no daba señales de vida, así que no le despertamos. Comimos nuestra pizza y charlamos un rato. Aunque no teníamos nada que hacer, no nos apetecía ponernos a limpiar.

Tampoco nos apetecía dormir más. Empezábamos a estar un poco aburridos y queríamos nuestros ordenadores portátiles. La única pega era que estaban en el armario de Chani. No esperamos mucho más hasta que decidimos despertarle. Tocamos a su puerta varias veces, pero no obtuvimos ninguna respuesta. A los pocos minutos, decidimos entrar.

Al cruzar la puerta, nos miramos sorprendidos. Chani no estaba, pero lo más extraño es que su habitación estaba totalmente recogida. Era muy curioso porque desentonaba con el resto de la casa, que parecía un estercolero. Estábamos muy confusos, así que tratamos de llamarle, pero no obtuvimos respuesta.

No le dimos mucha más importancia. Suponíamos que se encontraba bien; el único fastidio era que no teníamos los ordenadores y no sabíamos qué hacer. Nos sentamos en el sofá y, al cabo de unos minutos, Rich empezó a hablar. Me preguntó por mi aprendizaje de Java. Le contesté que no había progresado nada desde la última lección, así que se ofreció a continuar con las clases en ese mismo momento.

Mostrar datos en nuestra consola.

Antes de seguir avanzando, necesitas entender las etapas involucradas en el desarrollo de un programa o aplicación en Java. Desde que se comienza a escribir el código hasta la finalización del proyecto.

1) **Escribir el código fuente:** El proceso empieza cuando el programador escribe el código en lenguaje Java. Esto se hace usando un editor de texto o un entorno de desarrollo integrado (IDE) como IntelliJ IDEA o Eclipse. El código fuente sigue una estructura y una “gramática” específica.

2) **Guardar el archivo fuente:** Cuando el código está listo, se guarda en un archivo con la extensión ".java". Este archivo incluye todas las instrucciones necesarias para que el programa funcione de manera correcta.

3) **Compilación del código:** El siguiente paso es compilar el código fuente mediante un compilador de Java. Este, toma el archivo ".java" y lo convierte en un código llamado "bytecode". Este archivo es ejecutable en cualquier sistema que tenga instalada la Máquina Virtual de Java (JVM).

4) **Generación de archivos .class:** Tras la compilación, se generan uno o varios archivos con la extensión ".class", que contienen el bytecode. Estos archivos serán los que se ejecutarán en la JVM.

5) **Ejecución del programa:** Finalmente, con los archivos de bytecode generados, se puede ejecutar el programa utilizando la JVM. La JVM interpreta el bytecode y lleva a cabo las operaciones definidas en el código que se escribió manualmente por el programador.

En estos momentos no es necesario que aprendas esos pasos de memoria. Nuestro IDE se va a encargar de realizar todos esos pasos excepto el primero. El de escribir el código. Pero es interesante conocer el proceso completo. Además, nos ha servido para introducir los siguientes dos términos:

Un IDE, o Entorno de Desarrollo Integrado: IDE es la abreviatura de Integrated Development Environment. Esta es una herramienta de software que contiene todo lo necesario para el desarrollo de aplicaciones de software. Se puede decir que es un "centro de operaciones" para los programadores, que contiene diversas herramientas y características útiles en un solo lugar.

Compilador de Java: es una herramienta que permite traducir el código fuente legible por humanos, (El código que hemos escrito) a un formato entendido por la máquina. Esto permite que el código Java sea portable y pueda ejecutarse en diferentes plataformas sin necesidad de modificarlo. Sin el compilador, no podríamos transformar nuestras ideas en aplicaciones funcionales.

Mostrando texto y variables en la consola

Importancia de la Salida de Datos

En el proceso de programación en general, la salida de datos es totalmente necesaria ya que es la principal forma en la que el programa puede comunicarnos los resultados y el estado actualizado del programa. Si no tuviésemos una salida visual, no podríamos saber lo que está pasando, si el programa está cumpliendo nuestras órdenes al pie de la letra o si algo no sale como debe. La mayoría de las veces los programadores trabajan basándose en prueba y error, es por eso que ver los resultados constantemente es vital. Para un programador novato es incluso más importante ya que ver cuál es el resultado de las líneas que escribimos va a ayudarnos a aprender y ver lo que sucede cuando escribimos nuestra sintaxis.

Usos Comunes de la Salida en la Consola

1. **Interacción con el Usuario**

La consola permite interactuar con el usuario pidiendo información y mostrando resultados. Esta interacción es fundamental en programas básicos y es una excelente manera de enseñar conceptos de entrada y salida.

2. **Mostrar Resultados de Cálculos**

Presentar los resultados de operaciones matemáticas, algoritmos o procesos lógicos es una de las funciones básicas de cualquier programa.

3. **Feedback en Procesos Largos**

Durante la ejecución de procesos que toman tiempo, proporcionar retroalimentación en la consola ayuda al usuario a entender que el programa sigue funcionando.

4. **Depuración de Código**

Imprimir valores y estados de variables ayuda a los programadores a entender cómo está funcionando el programa y a identificar posibles errores.

5. **Demostraciones y Ejemplos Educativos**

Los ejemplos que imprimen resultados en la consola son esenciales para enseñar conceptos básicos de programación de una manera visible y tangible.

Uso de System.out.println y System.out.print

Vamos a ver cómo utilizar una línea de código que nos permite mostrar datos en la consola de nuestro IDE. Para ello podremos utilizar dos comandos: println y print.

Si ya tienes algo de experiencia programando o has buscado material didáctico en cualquier otra parte, seguramente te hayas encontrado con el ejercicio más conocido de toda la programación. Consiste en que muestres la frase "hola mundo" en nuestra consola. La respuesta es la siguiente:

System.out.println("Hola mundo");

Siempre sigue la misma estructura, System.out.println();

En este caso en concreto utilizamos las comillas dobles ya que se trata de una línea de texto. También podemos mostrar variables que hayamos declarado y a las que hayamos asignado un valor anteriormente. Veamos un ejemplo. Hemos creado una variable tipo int llamada total y ya le hemos asignado un valor, en este caso 9. Podemos imprimirla en la pantalla de la siguiente forma:

```
int total = 9;
System.out.println(total);
```

Al principio hemos mencionado que podemos utilizar tanto print como println pero hay una gran diferencia entre ambas:

- **System.out.println** agrega una línea nueva después de imprimir lo que le indiques. Es como dar un salto de línea después de mostrar algo.
- **System.out.print** simplemente imprime lo que le digas sin agregar un salto de línea al final. Es como escribir en una línea continua.

Concatenación de Cadenas

Podremos mostrar operaciones de variables e incluso combinarlas con texto. Imagínate que hemos declarado las variables tipo int a y b. Respectivamente les hemos asignado los valores 5 y 7.

```
int a = 5;
int b = 7;
```

Después pedimos que se muestre el siguiente texto: "El resultado de la multiplicación entre a y b es =". Por último, el resultado de dicha multiplicación.

```
System.out.println("El resultado de la multiplicación entre a y b es = " + a * b);
```

Como puedes apreciar después de mostrar el texto hemos utilizado el símbolo + para unirlo. ("texto " + resultado).

Podríamos haberlo hecho de la siguiente manera que puede resultar algo más sencillo de comprender:

```
System.out.println("El resultado de la multiplicación entre a y b es = " + (a * b));
```

Justo después del símbolo = hemos dejado un espacio para que, al mostrarse en la consola, quede un hueco. Aunque podríamos haberlo hecho de la siguiente manera.

System.out.println("El resultado de la multiplicación entre a y b es =" + " " + (a * b));

He utilizado + " " + para crear un espacio.

Cuando combinamos texto y datos numéricos no es muy útil, pero si queremos mostrar dos variables numéricas y necesitamos que estén separadas, recurriremos a este truco.

Veamos un ejemplo más, pero en esta ocasión con sumas.

Si queremos sumar las variables a y b del ejercicio anterior, simplemente lo haremos de la siguiente forma:

System.out.println(a+b);

Pero es posible que necesitemos simplemente que ambas se muestren en la consola por separado. Una detrás de otra. En ese caso lo haremos así:

System.out.println(a+ " " +b);

Por supuesto también podemos usar los símbolos de restar y dividir (- y /) dependiendo de lo que necesitemos.

Cambiando el color de la fuente

Es posible que en algún momento necesitemos que el color que aparezca en nuestra consola sea distinto al blanco que suele venir por defecto. Esto lo vamos a conseguir de manera muy sencilla. Simplemente debemos insertar en nuestra línea un código de color de los que se muestran en la siguiente tabla:

```
BLACK  => "\033[30m",
RED    => "\033[31m",
GREEN  => "\033[32m",
YELLOW => "\033[33m",
BLUE   => "\033[34m",
PURPLE => "\033[35m",
CYAN   => "\033[36m",
WHITE  => "\033[37m",
```

Para ello lo haremos de la siguiente manera:

System.out.println("\033[31m" + "El resultado de la multiplicación entre a y b es =" + " " + (a * b));

Insertamos el código justo delante de la parte que necesitamos que cambie de color. Desde ahí y hasta el final de la línea se mostrará con el nuevo color. Si necesitamos dos colores o más, insertaremos los códigos correspondientes justo delante de las partes que se deben mostrar con esa tonalidad en concreto.

System.out.println("\033[31m" + "El resultado de la multiplicación entre a y b es =" + " " + "\033[34m" + (a * b));

Anotaciones en nuestro código que no se imprimen

Las dobles barras "//" se utilizan en Java para indicar un comentario de una sola línea. Esto significa que cualquier texto que esté después de las dobles barras en esa línea será ignorado por el compilador de Java y no afectará la ejecución del programa.

// Este es un comentario de una sola línea

int edad = 25; // Esta línea declara una variable llamada edad y le asigna el valor 25

En este caso, los comentarios sirven para hacer anotaciones en el código que ayudan a entender qué hace cada parte del programa. También son útiles para desactivar temporalmente una línea de código sin tener que borrarla por completo.

Limitando la salida de datos a dos decimales

De momento, en ningún ejercicio te va a hacer falta que limites el número de decimales. Es más, **si eres nuevo en Java deberías saltarte este apartado**. La razón por la que lo pongo aquí es porque es posible que haya gente que le resulte útil y no quieran que en las operaciones salga una larga fila de decimales.

No obstante, y aunque ahora no te sirva, acuérdate de que tienes la información aquí y puedes volver a ella cuando lo necesites. Por otro lado, si tienes la confianza suficiente, puedes tratar de limitar los datos en tus ejercicios.

Formas de limitar los decimales

- En Java, puedes usar la clase **DecimalFormat** para limitar los decimales de un número a dos de manera sencilla. Aquí tienes un ejemplo:

```
import java.text.DecimalFormat;


public class Main {
   public static void main(String[] args) {
      double numero = 3.14159;
      DecimalFormat formato = new DecimalFormat("#.00");


      System.out.println("Número con dos decimales: " + formato.format(numero));
       // Output: 3.14
   }
}
```

Otra forma de limitar los decimales a dos en Java es usando la función printf() o String.format().

- Ejemplo con printf():

```
public class Main {
   public static void main(String[] args) {
      double numero = 3.14159;
      System.out.printf("Número con dos decimales: %.2f", numero);  // Output: 3.14
   }
}
```

- Ejemplo con String.format():

```
public class Main {
   public static void main(String[] args) {
      double numero = 3.14159;
      String resultado = String.format("%.2f", numero);
      System.out.println("Número con dos decimales: " + resultado);  // Output: 3.14
   }
}
```

Cuando analicemos el tema de números aleatorios, te enseñaré otra forma distinta, que para mí es la más sencilla y la que siempre utilizo.

Resumen de lo aprendido

En Java, para mostrar información en la consola, usamos los comandos System.out.println y System.out.print. Ambos muestran datos en la pantalla, pero hay una diferencia importante: System.out.println agrega automáticamente un salto de línea después de imprimir, mientras que System.out.print no lo hace, manteniendo la siguiente salida en la misma línea.

Por ejemplo, si queremos mostrar un texto como "Hola mundo" en la consola, escribimos:

System.out.println("Hola mundo");

Si queremos mostrar una variable, como total, que tiene un valor de 9, escribimos:

int total = 9;

System.out.println(total);

También podemos realizar operaciones y combinar texto con variables. Por ejemplo, si tenemos dos variables a y b con valores 5 y 7 respectivamente, y queremos mostrar el resultado de su multiplicación junto con un texto, lo hacemos así:

int a = 5;

int b = 7;

System.out.println("El resultado de la multiplicación entre a y b es = " + (a * b));

Además, podemos cambiar el color del texto en la consola agregando códigos de color antes del texto que queremos cambiar. Por ejemplo, para mostrar el texto en rojo, escribimos**:**

System.out.println("\033[31m" + "Este texto será rojo");

Finalmente, para hacer anotaciones en nuestro código que no se imprimirán en la consola y que solo sirven para nosotros o para otros programadores que lean el código, usamos dobles barras "//" para indicar un comentario de una sola línea.

Ejercicios del tema

Ejercicio 1. Dime la diferencia entre System.out.println y System.out.print.

Solución:

Si usas System.out.println("Hola"); imprimirá "Hola" y luego dará un salto a la siguiente línea. Pero si usas System.out.print("Hola"); imprimirá "Hola" en la misma línea donde se encuentre el cursor.

Utiliza **System.out.println** para imprimir mensajes con un salto de línea al final. Es útil para mostrar información y separar diferentes partes de la salida.

Ejercicio 2. Muestra una pirámide de asteriscos en la consola de tu IDE. Algo similar a esto:

```
  *
 ***
*****
```

Solución:

```
System.out.println("   *");
System.out.println("  ***");
System.out.println(" *****");
```

Ejercicio 3. Crea un calendario de tres días con tres asignaturas y muéstralo por pantalla.

Solución:

```
System.out.println("   Lunes    |   Martes  | Miércoles   ");
System.out.println("_____________________________________");
System.out.println("  Matemáticas |   Lengua  |  Economía   ");
System.out.println("_____________________________________");
System.out.println(" Programación | Filosofía |  Geografía  ");
System.out.println("_____________________________________");
System.out.println("  E. Física   |   Lengua  |   Química  ");
System.out.println("_____________________________________");
```

Ejercicio 4.

Declara dos variables: una de tipo double llamada a y otra de tipo int llamada b. Asígnales los valores 1.4 y 7, respectivamente. Después, muestra por pantalla la multiplicación de ambas.

Solución:

```
double a;
int b;
a=1.4;
b=7;
System.out.println(a*b);
```

Ejercicio 5. Crea una variable de tipo String y muéstrala en pantalla.

Solución:

```
String nombre = "Pedro";
System.out.println(nombre);
```

Ejercicio 6. Crea una variable tipo char y muéstrala en pantalla.

Solución:

```
char simbolo = '@';
System.out.println(simbolo);
```

Ejercicio 7. Crea 4 variables de tipo int y llámales a, b, c y d. Asígnales los valores 15, 25, 5 y 8, respectivamente. Realiza una operación que sume a + b, divida ese resultado entre c y luego multiplique todo por d.

Solución:

```
int a = 15;
int b = 25;
int c = 5;
int d = 8;
System.out.println(((a + b) / c) * d);
```

Ejercicio 8. Crea 4 variables de tipo int y llámales a, b, c y d. Asígnales los valores 14, 24, 5 y 7, respectivamente. Realiza una operación que sume a + b, divida ese resultado entre c y luego multiplique todo por d.

Solución:

Si te fijas en este caso, los resultados no son fiables ya que al dividir la suma de a y b entre c dan decimales. El problema es que el tipo de dato int no puede albergar decimales. Esto hace que el decimal se pierda. Así que al final el resultado no es exacto. Antes de realizar las operaciones, debemos convertir las variables int en double. Lo haremos de la siguiente manera:

```
int a = 14;
int b= 24;
int c = 5;
int d = 7;
double aa=a;
double bb=b;
double cc=c;
double dd=d;
```

Ahora ya si que podemos realizar las operaciones:

```
System.out.println(((aa+bb)/cc)*dd);
```

Utiliza **System.out.print** cuando necesites imprimir sin añadir un salto de línea. Ideal para construir líneas de salida dinámicamente.

Ejercicio 9. Realiza un conversor de divisas básico que pase de euros a dólares. Supondremos que un euro equivale a 1.10 dólares. Para ello, deberás crear una variable de tipo double y asignarle el valor exacto de los euros que necesitas convertir. Queremos que el valor final se muestre en la consola.

Solución:

Método 1.

```
double euros= 70;
System.out.println(euros *1.1);
```

Método 2.

```
double euros= 70;
double dolares = euros * 1.1;
System.out.println(dolares);
```

Ejercicio 10. Crea un programa que calcule el IVA de una factura y se lo aplique a la cantidad neta. El programa debe mostrar desglosado tanto el valor neto como el IVA, y luego el total de la factura. El valor neto se asignará a una variable de tipo double. El IVA es del 16%.

Solución:

```
double neto = 120;
double iva = neto * 0.16;
System.out.println("El valor neto es de " + neto + " euros");
System.out.println("El valor del IVA es de " + iva + " euros");
System.out.println("El valor total es de " + (iva + neto) + " euros");
```

Concatena variables y texto con el operador +. Facilita la construcción de mensajes informativos y detallados.

Capítulo 3.

El amigo de Rich / Entrada de datos por teclado

Han pasado dos días desde la fiesta y toca limpiar. No se puede decir que sea una de mis actividades favoritas, pero esta vez no queda más remedio. No puedo dejar de pensar en la suerte que tiene Chani por librarse de esta desagradable tarea. Mucha de la gente que vino a la fiesta eran amigos suyos, y no me parece justo limpiar lo que ellos han ensuciado. Aunque, pensándolo mejor, no conocíamos la razón por la que se había ido repentinamente. Es muy posible que fuese por algo importante, y el hecho de que no conteste al teléfono me preocupa bastante.

Por otro lado, estoy enfadado. Entre otras pertenencias, mi portátil está atrapado dentro de su armario. En mi cabeza ronda una pregunta: ¿por qué, antes de irse, se tomó su tiempo para recoger su habitación? Si necesitas irte urgentemente, simplemente coges lo que necesitas y te vas.

Rich parece tranquilo, apenas dice nada. De vez en cuando sonríe y parece que tiene la mente en otro lado. A penas me ha dirigido la palabra en todo el día. Pero no me extraña en absoluto. Él a veces es así.

Se pone a pensar en sus cosas y pasa horas sin hablar con nadie. Aunque lo cierto es que, en ese estado, es bastante productivo. Tiene una capacidad de concentración asombrosa. Creo que por eso es mejor estudiante que yo.

En estos momentos, prefiero no molestarle y que se concentre en la limpieza. Cuanto antes acabemos, mejor. Le dejaré recogiendo la cocina mientras yo friego el resto de la casa. Después de esto, todo debería estar limpio.

Por fin, parece que ya ha acabado esta tortura. Después de varias horas de sufrimiento, puedo decir que hemos terminado. Mi plan es tumbarme en el sofá lo que queda del día. Rich, en cambio, parece que ha quedado con alguien. Imagino que se va a marchar de casa pronto. Está en el baño arreglándose, así que puede estar allí horas. Le encanta mirarse en el espejo y siempre sale de casa perfectamente arreglado.

Decido ponerme cómodo, pero pronto tengo que levantarme. Alguien ha llamado al timbre. Durante la semana no solemos recibir visitas, así que me parece un poco raro. Preguntan por Rich, así que abro la puerta del portal. Me quedo en la puerta, para no tener que volver a levantarme. Además, tengo mucha curiosidad por ver quién viene.

Se trata de un amigo de Rich. He hablado un par de veces con él, pero la verdad es que no recuerdo su nombre. Parece que conoce la casa, aunque no recuerdo que haya estado aquí nunca. Después de saludarme, se ha ido directo a la habitación de Rich. Estoy un poco sorprendido, pero imagino que Rich ya había quedado con él.

Parece que ya puedo relajarme, y como no tengo absolutamente nada que hacer, he cogido un libro de Java y voy a tratar de aprender un poco más.

Introducir datos por teclado

El tema es bastante sencillo, pero antes de ponernos con él, aún es necesario aprender algún concepto teórico más. Toda esta información puede resultar algo abrumadora al principio, pero poco a poco todo cobrará sentido. Como consejo personal, te diré que es importante combinar la teoría con la práctica. Los ejercicios son amenos e incluso me atrevería a decir que entretenidos. Lo importante es ir avanzando y cada día saber un poco más. Ahora toca entender de forma general lo que es una clase.

Introducción al concepto de clase

Cuando trabajamos en nuestra IDE escribimos nuestro código dentro de estas líneas:

```
public class Main {
   public static void main(String[] args)

Aquí va nuestro código.

   }
}
```

Te explico lo que significan esas dos primeras líneas que vemos ahí; es muy posible que lleves un tiempo preguntándote por qué están ahí y para qué sirven. En temas posteriores vamos a profundizar y analizar con detalle el concepto de clase. Por ahora, es importante que entiendas que esas líneas son necesarias para que el código pueda identificar que estamos trabajando con una clase y que esto será fundamental en la programación orientada a objetos.

public class Main: En Java, un programa se organiza en "clases". Una clase es como un plano o una receta que le dice a la computadora cómo hacer algo. Cuando escribes public class Main, estás diciendo que estás creando una nueva clase llamada "Main". El public significa que esta clase es accesible desde cualquier parte del programa.

Cuando comienzas un proyecto en Java, necesitas al menos una clase que actúe como punto de partida, y usualmente se llama "Main" (aunque puede tener cualquier nombre). Esta clase Main es donde comienza la ejecución del programa.

public static void main(String[] args): Es el **método principal (o "main")** que el sistema ejecuta para iniciar un programa.

- **public**: Esto significa que el método **main** es accesible desde cualquier parte del programa. Es como una puerta abierta que permite que otros métodos lo llamen.
- **static**: Esto indica que el método **main** pertenece a la clase en sí misma, en lugar de a una instancia específica de la clase. En pocas palabras, puedes llamar a **main** sin necesidad de crear un objeto de esa clase.
- **void**: Esto significa que el método **main** no devuelve ningún valor. En otras palabras, no produce ningún resultado cuando se ejecuta.
- **main**: Este es el nombre del método. Es el punto de entrada principal para el programa.
- **String[] args**: Este es un parámetro que recibe el método **main**. Es un arreglo de cadenas de texto que puede contener argumentos que se pasan al programa cuando se ejecuta. Por ejemplo, si ejecutas tu programa desde la línea de comandos y escribes algo después del nombre del programa, esos "algo" se almacenan en este arreglo. Por ejemplo, si ejecutas **java miPrograma Hola Mundo**, entonces **"Hola"** y **"Mundo"** estarían almacenados en **args**.

Puede que todo esto te resulte un poco raro de comprender, pero pronto todo te empezará a tener sentido. En los siguientes temas tocaremos estos conceptos más en profundidad. De momento, vamos a centrarnos en aprender a resolver problemas y en entender otra serie de conceptos básicos.

El orden de ejecución de un programa

Para comprender mejor este apartado, debes de entender el orden en cómo se ejecutan nuestras líneas de código en Java. El programa se va ejecutando de arriba abajo. Por ejemplo, si creamos una variable en la línea 18 y tratamos de asignarle valor en la línea 17 (una línea antes), el programa nos devolverá un error. En la línea 17, la variable aún no se ha creado, así que el programa aún no sabe que existe. El único proceso válido es primero crear una variable y después asignarle valor.

Lo mismo pasa a la hora de introducir datos por teclado. Da igual que tengamos cinco mil líneas escritas; si en la línea 22 hemos dado la instrucción de que hay que introducir un dato por teclado, el programa no leerá las siguientes líneas hasta que introduzcamos algo.

Una vez hecha esta pequeña aclaración, ya podemos meternos en profundidad en este tema. Lo primero que necesitamos es saber lo que es un Scanner y después cómo lo aplicamos a nuestros programas y aplicaciones.

Importancia de la Entrada de Datos en Aplicaciones Interactivas

La entrada de datos es fundamental en cualquier aplicación interactiva, ya que permite que los usuarios proporcionen la información necesaria para el funcionamiento del programa. Aquí te dejo algunas razones clave de por qué la entrada de datos es tan importante:

1. **Interacción Usuario-Programa**

   La entrada de datos permite la interacción directa entre el usuario y el programa. Sin esta interacción, el software no podría adaptarse a las necesidades y preferencias del usuario.

   Por ejemplo, en una aplicación bancaria, el usuario puede ingresar su número de cuenta y PIN para acceder a su información financiera.

2. **Personalización y Adaptación**

   Los programas pueden personalizarse según las entradas del usuario, ofreciendo una experiencia más relevante y personalizada.

3. **Recopilación de Información**

   Las aplicaciones recopilan datos a través de la entrada del usuario para realizar cálculos, almacenar información y tomar decisiones informadas.

4. **Control y Navegación**

   La entrada de datos permite a los usuarios navegar por las diferentes opciones y funcionalidades.

5. **Validación y Seguridad**

   A través de la entrada de datos, las aplicaciones pueden implementar mecanismos de validación y seguridad, asegurando que solo los usuarios autorizados accedan a ciertas funciones.

6. **Recogida de Feedback y Mejoras**

   Las aplicaciones pueden incluir mecanismos para que los usuarios proporcionen feedback, lo cual es crucial para la mejora continua del software.

Uso de la Clase Scanner

En Java, un "scanner" se refiere comúnmente a la clase Scanner del paquete java.util. Esta clase se utiliza para leer la entrada del usuario desde la consola o desde otros flujos de entrada, como archivos. Permite leer diferentes tipos de datos, como enteros, flotantes, cadenas, etc.

Para poder utilizar scanner en java debemos escribir la siguiente línea encima de nuestra clase principal:

```
import java.util.Scanner;
```

Nos quedaría lo de la siguiente manera:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

    //código de nuestro programa.

     }
}
```

Concepto de paquete en Java

En Java, un "paquete" es simplemente una forma de agrupar y organizar el código. Imagina que es como una carpeta donde guardas archivos relacionados. Los paquetes nos ayudan a evitar que haya confusiones con los nombres de las clases y permiten que nuestro código esté mejor organizado.

Un paquete puede contener varias cosas, como clases, interfaces y subpaquetes. Por ejemplo, si estás creando un programa para una tienda en línea, podrías tener un paquete llamado "com.tiendaonline" donde guardes todas las clases que tienen que ver con tu tienda.

Para indicar que una clase pertenece a un paquete, se usa la palabra clave package al inicio del archivo de la clase. Por ejemplo:

```
package com.tiendaonline;

public class Producto {
   // Código de la clase Producto
}
```

En este ejemplo, la clase Producto pertenece al paquete com.tiendaonline. Esto significa que otras clases en el mismo paquete pueden acceder a ella directamente sin necesidad de importarla, mientras que las clases fuera de este paquete necesitarán importarla para poder utilizarla.

Una vez que hemos importado el paquete Scanner, ya podemos continuar escribiendo el código de nuestro programa dentro de nuestra clase.

Creando y utilizando un Scanner

Para poder introducir datos por teclado, lo primero que debemos hacer es crear un Scanner y se hace siempre de la siguiente forma:

Scanner ejercicio = new Scanner(System.in);

En este caso hemos llamado a nuestro Scanner “ejercicio” pero evidentemente se puede llamar como prefieras.

Una vez creado, necesitamos **llamar a ese scanner** y deberemos hacerlo justo en el momento que necesitemos que se introduzca un dato por teclado. El programa no continuará ejecutándose hasta que no se introduzca un valor por teclado. **La línea de código que vamos a utilizar para llamar a ese Scanner va a depender del tipo de dato que vamos a introducir.**

Lectura de Diferentes Tipos de Datos

Con Scanner, puedes capturar cadenas de texto, números enteros, números de punto flotante, valores booleanos y más, todo con métodos específicos que garantizan una lectura precisa y eficiente. Sin embargo, dependiendo del tipo de dato, necesitaremos usar una línea de código distinta para "llamar" al Scanner que hemos creado previamente.

Observa estos ejemplos:

```
int a = ejercicio.nextInt();
double a = ejercicio.nextDouble();
String a = ejercicio.nextLine();
boolean a = ejercicio.nextBoolean();
char a = ejercicio.next().charAt(0);
```

Ten cuidado ya que todos tienen la misma estructura excepto en el caso de las variables tipo char.

Ejemplo práctico:

Vamos a ver un ejemplo, ya que es la manera más sencilla de entenderlo. Necesitamos crear un programa en el que se introduzcan dos números enteros. Los llamaremos a y b. El programa deberá pedirnos uno a uno que introduzcamos su valor. Después, realizará la multiplicación de ambos valores y nos mostrará el resultado en la consola.

En el código siguiente, veremos cómo importamos el paquete Scanner, luego creamos un Scanner llamado ejercicio, y por último llamamos al Scanner en el momento exacto en el que necesitamos que se introduzcan los datos.

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {

//Creamos el Scanner
Scanner ejercicio = new Scanner(System.in);
```

```
// Pedimos el valor del número a

    System.out.println("Introduce el valor del número a");
    int a = ejercicio.nextInt();

// Pedimos el valor del número b

    System.out.println("Introduce el valor del número b");
    int b = ejercicio.nextInt();

// Realizamos la operación de ambas variables y las mostramos por pantalla
System.out.println("El resultado de multiplicar es " + (a*b));
scanner.close();

  }
}
```

Cierre del Scanner

Cerrar el objeto Scanner es una práctica crucial en Java para liberar los recursos asociados con la entrada de datos. Cuando terminas de utilizar un Scanner, especialmente si está leyendo desde System.in, debes cerrarlo con el método close(). Esto no solo ayuda a liberar memoria, sino que también previene posibles problemas de seguridad y fugas de recursos. El cierre adecuado del Scanner garantiza que el programa maneje los recursos del sistema de manera eficiente y responsable. Por ejemplo, al final de tu programa, puedes utilizar scanner.close(); para cerrar correctamente el objeto Scanner.

Resumen de lo aprendido

En Java, una clase es como el diseño de un edificio: define la estructura y organiza las partes del programa. La clase Main es donde todo comienza, es decir, el punto de entrada del programa. Cuando escribimos public class Main, estamos diciendo que esta clase es accesible desde cualquier parte del código.

Dentro de la clase Main, el método public static void main(String[] args) es fundamental porque es el primer lugar donde el programa empieza a ejecutarse. Aquí te explico cada parte de este método:

public: Esto significa que el método puede ser utilizado desde cualquier lugar del programa.

static: Significa que este método pertenece directamente a la clase y no a un objeto creado a partir de la clase.

void: Indica que el método no devolverá ningún valor.

main: Es el nombre del método que siempre busca Java al iniciar un programa.

String[] args: Es un arreglo que puede contener argumentos proporcionados cuando el programa se ejecuta.

El principal concepto que se explica en este tema es la introducción de datos por teclado. Para ello utilizamos la clase Scanner. Esta clase facilita la lectura de datos como números y texto desde el teclado.

Para utilizar Scanner, primero se necesita "importar" la clase. Luego, se crea el Scanner en sí, y después lo llamamos en la parte exacta del programa donde lo necesitamos. Dependiendo del tipo de dato que estamos manejando, el código que utilizamos para llamar al Scanner será diferente. Por último, hay que recordar que es importante cerrar el Scanner al terminar, para liberar los recursos que estaba utilizando.

En resumen, ahora ya entiendes cómo funciona una clase en Java y cómo manejar la entrada de datos con Scanner. Y si aún no te ha quedado claro del todo, ahora te propongo unos ejercicios que definitivamente despejarán tus dudas.

Ejercicios del tema

Ejercicio 1. Crea un programa con el que podamos introducir un número entero y que después este se muestre por pantalla. Recuerda importar la clase Scanner. import java.util.Scanner;

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Introduce un número");
int a = ejercicio.nextInt();
System.out.println("El número introducido es: ");
System.out.println(a);
```

Ejercicio 2. Crea un programa conversor de moneda con el que introducimos una cantidad en euros y nos devuelve el resultado en dólares. Suponemos que la equivalencia de 1 euro es 1,10 dólares.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Introduce la cantidad en euros");
double euros = ejercicio.nextDouble();
System.out.println("La cantidad en euros es de:" + euros);
System.out.println("La cantidad en dólares es de: " + (a * 1.10));
```

Ejercicio 3. Crea cuatro variables de tipo double y llámalas a, b, c y d. El programa deberá pedir por teclado que introduzcamos los datos de cada una de las variables. Después, el programa multiplicará a por b, dividirá el resultado por c y, a todo ello, le sumará d. El resultado debe mostrarse en la consola.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Introduce el dato a");
double a = ejercicio.nextDouble();
System.out.println("Introduce el dato b");
double b = ejercicio.nextDouble();
System.out.println("Introduce el dato c");
double c = ejercicio.nextDouble();
System.out.println("Introduce el dato d");
double d = ejercicio.nextDouble();
System.out.println("((a * b)/c) + d = " + (((a * b) / c) + d));
```


Scanner proporciona métodos específicos para leer diferentes tipos de datos, como nextInt() para enteros, nextDouble() para números de punto flotante, y nextLine() para cadenas. Utiliza el método adecuado según el tipo de dato que esperas recibir.

Ejercicio 4. Crea un programa que calcule el área de un rectángulo. Para ello, debe pedirnos la medida de la altura y la anchura del mismo. Recuerda que el área de un rectángulo equivale a la multiplicación de sus lados.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Introduce la altura del rectángulo");
int a = ejercicio.nextInt();
System.out.println("Introduce el ancho del rectángulo");
int b = ejercicio.nextInt();
System.out.println("El área del rectángulo es = " + (a * b));
```

Ejercicio 5. Realiza el mismo programa que en el ejercicio anterior, pero en este caso, que calcule el área de un triángulo. Recuerda que el área del triángulo es la base por la altura, y todo ello dividido entre dos.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Introduce la altura del triángulo");
int a = ejercicio.nextInt();
System.out.println("Introduce la base del triángulo");
int b = ejercicio.nextInt();
System.out.println("El área del triángulo es = " + ((a * b) / 2.0));
```

Ejercicio 6. Crea un programa donde se introduzca por teclado el total de una factura y este te indique cuánto se ha pagado de IVA. El impuesto es del 16%.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("Ingresa el total de la factura");
double total = ejercicio.nextDouble();
System.out.println("El valor neto es de " + (total * 0.84) + " euros");
System.out.println("El valor del IVA es de " + (total * 0.16) + " euros");
System.out.println("El valor total es de " + total + " euros");
```

Ejercicio 7. Crea un programa que calcule el salario semanal de un trabajador dependiendo de las horas que ha trabajado. Hasta 40 horas, cobra 7 euros/hora. El programa debe mostrar el número de horas totales, además del salario total.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.print("Ingresa el número total de horas ");
int horas = ejercicio.nextInt();
System.out.println("Total de horas trabajadas: " + horas);
System.out.println("Salario en euros: " + (horas * 7));
```

Ejercicio 8. Conversor de grados Celsius a Fahrenheit. La fórmula es: (Celsius * 9 / 5) + 32.

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.println("¡Bienvenido al conversor de temperatura!");
System.out.println("Por favor, introduce la temperatura en grados Celsius:");
double celsius = ejercicio.nextDouble();
System.out.println("La temperatura en Fahrenheit es de: " + ((celsius * 9 / 5) + 32));
```

Ejercicio 9

Crea un programa en el que se nos pida ingresar un símbolo. Lo almacenaremos en una variable de tipo char. Después, el programa debe mostrar el símbolo en la consola..

Solución:

```
Scanner ejercicio = new Scanner(System.in);
System.out.print("Ingresa un símbolo ");
char a = ejercicio.next().charAt(0);
System.out.println("El símbolo introducido es: " + a);
```

Es importante cerrar el objeto Scanner con el método close() al finalizar su uso para liberar los recursos del sistema correctamente. Esto previene posibles fugas de memoria y garantiza un manejo eficiente de los recursos.

Ejercicio 10. Crea un programa que simule el control de uso de una fotocopiadora en una oficina. El programa permitirá a los empleados introducir la cantidad de copias que desean realizar y calculará el costo total. También llevará un registro del total de copias realizadas durante el día.

Solución:

```
Scanner ejercicio = new Scanner(System.in);

// Variables
int costoPorCopia = 5; // Costo por copia

// Introducción de datos por teclado
System.out.println("Bienvenido a la fotocopiadora de la oficina.");
System.out.print("Ingrese la cantidad de copias que desea realizar: ");
int cantidadCopias = ejercicio.nextInt();

// Calculando costo total
int costoTotal = cantidadCopias * costoPorCopia;

// Mostrando resultados
System.out.println("El costo total de " + cantidadCopias + " copias es de $" +
costoTotal);
System.out.println("Se han realizado un total de " + cantidadCopias + " copias
en esta operación.");

// Cerrando scanner
ejercicio.close();
```

Capítulo 4.

Pasando el rato en comisaría / Trabajando con condicionales

No sé cómo ha pasado, pero estoy en comisaría. En concreto, encerrado en una habitación junto con Rich y su amigo, que, por cierto, se llama Fernando. La historia es bastante sencilla de contar, aunque para mí no tiene ningún sentido. No sé qué está ocurriendo exactamente, ni por qué me han detenido.

Poco después de que Fernando llegase a casa, la policía irrumpió sin previo aviso. Todo ocurrió muy rápido. Varios hombres armados entraron gritando y ordenándonos que nos tumbásemos en el suelo. Nos reunieron a los tres en el salón y nos sentaron en el suelo. Mientras tanto, se pusieron a registrar toda la casa. No sé qué estarían buscando, pero estaba claro que no querían dejarse nada. El registro parecía muy minucioso.

A los pocos minutos de empezar el registro, vi salir a un policía con nuestros ordenadores. No pude evitar pensar que ellos sí que habían podido abrir el armario de Chani, mientras que nosotros llevábamos dos días sin poder acceder a nuestras cosas. No me parecía justo.

También me preguntaba quién iba a recoger todo ese desaguisado. Acabábamos de limpiar toda la casa y ahora todo estaba incluso peor que después de la fiesta. Pero lo que más veces se me pasaba por la cabeza era la mala suerte que había tenido Fernando. Nunca había venido antes y, a los 20 minutos de llegar, aparece la policía. En ningún momento he pensado que su visita tenga alguna relación con esta detención. Simplemente ha sido muy mala suerte.

Salvo el susto inicial, en ningún momento me he sentido preocupado. No tengo miedo, ya que no he hecho nada malo. Pueden revisar mi ordenador y mi móvil si quieren. No van a encontrar nada que se pueda considerar delito.

Fernando está completamente blanco y no ha dicho ni una palabra desde que nos trajeron a esta habitación. Rich, en cambio, está muy tranquilo. Siempre está tranquilo. Dice que habrá sido un error. En nuestro edificio hay muchos pisos alquilados y mucha gente que está de paso. Lo más seguro es que hayan entrado en el piso equivocado. Yo opino lo mismo.

Lo peor es no saber qué está pasando ni cuánto tiempo vamos a tener que esperar. Requisaron nuestros teléfonos, así que no tenemos mucho que hacer. La sugerencia de Rich es que dediquemos un poco de tiempo para mejorar mis conocimientos de Java. No me apetece mucho, pero eso nos mantendrá la cabeza ocupada.

Parece que el siguiente tema son las condicionales. Según Rich, esto va a ayudarnos a mejorar mucho nuestros programas. Dice que es un tema divertido y fácil de aprender, así que ha aumentado mucho mi nivel de moral. Pongámonos a ello cuanto antes.

Trabajando con condicionales

Importancia de las condicionales en Java

Las condicionales permiten que los programas tomen decisiones basadas en diferentes situaciones. Sin ellas, los programas seguirían una serie de pasos de manera estricta, sin poder adaptarse a cambios o a distintos tipos de datos. Las condicionales hacen que el código sea más flexible, lo cual es esencial para crear aplicaciones interactivas y funcionales.

Gracias a las condicionales, los programadores pueden hacer que sus programas respondan a lo que haga el usuario, a cambios en el entorno o a datos en tiempo real. Esto es muy importante para desarrollar aplicaciones útiles en el mundo real, ya que permiten que los programas manejen diferentes situaciones según las variables que introduzca el usuario.

Por ejemplo, en una aplicación bancaria, las condicionales son necesarias para verificar transacciones y validar la información que ingresa el usuario. También ayudan a mantener la seguridad del sistema. En los videojuegos, las condicionales controlan cómo el juego reacciona a las acciones del jugador, haciendo que la experiencia sea más dinámica e interactiva.

Estructura Básica de un if

Un “if” es una forma de decirle a nuestro programa que haga algo solo si una condición específica se cumple. Piensa en esto como tomar decisiones en la vida cotidiana. Por ejemplo, "Si está lloviendo, lleva un paraguas". En programación, se escribe de una manera similar:

```
if (condición) {
   // Código que se ejecuta si la condición es verdadera
}
```

1. if: Esta palabra clave indica el inicio de una condición.

2. Condición: Dentro de los paréntesis, escribimos la condición que queremos comprobar. Por ejemplo, "si un número es mayor que 10".

3. Código entre llaves { }: Dentro de las llaves, escribimos el código que queremos que se ejecute si la condición es verdadera.

Vamos a verlo con un ejemplo muy sencillo. Imagina que estás creando un programa y quieres que te diga "Es un número grande" si un número es mayor que 10. Así es cómo deberías escribir el código:

```
int numero = 15; // Hemos definido un número
if (numero > 10) {
  System.out.println("Es un número grande");
}
```

- Como puedes ver lo primero que hemos hecho ha sido definir un número y darle el valor 15.

- Después queremos que entienda que queremos que realice una acción, pero solo si el número es mayor que 10. if (numero > 10): Aquí estamos diciendo "si el número es mayor que 10".

- Por último, creamos una línea usando System.out.println que imprimirá en la pantalla si la condición es verdadera.

Por lo tanto, si número es mayor que 10 (en este caso, 15 es mayor que 10), la computadora imprimirá "Es un número grande"; si es menor, simplemente no hará absolutamente nada.

Declaraciones else y else if

Cuando usamos if, podemos decirle a nuestro programa que haga algo si una condición es verdadera. Pero, ¿qué pasa si esa condición no se cumple? Ahí es donde entran las declaraciones else y else if.

- **“Else”**

La declaración else se usa para especificar un bloque de código que se ejecutará si la condición del if no es verdadera. Es como decir: "Si la condición no se cumple, haz esto otro".

```
int numero = 15;
if (numero > 10) {
   System.out.println("Es un número grande");
} else {
   System.out.println("Es un número pequeño");
}
```

Arriba vemos el ejemplo anterior pero un poco modificado. Si el número es mayor que 10, mostrará "Es un número grande". Exactamente lo mismo que en el anterior programa, pero antes si un número era menor que 10 no hacía nada, ahora mostrará el siguiente texto: "Es un número pequeño".

- **“Else if”**

La declaración else if se usa para comprobar otra condición si la primera condición del if no se cumple. Es como decir: "Si la primera condición no se cumple, prueba esta otra condición".

```
int numero = 15;
if (numero > 10) {
   System.out.println("Es un número grande");
} else if (numero > 5) {
   System.out.println("Es un número mediano");
} else {
   System.out.println("Es un número pequeño");
}
```

Repetimos mismo ejemplo otra vez. Sólo que ahora no hay dos condicionales sino tres. El ejemplo habla por sí solo. La primera vez usamos un else if ya que aún queremos proponer una condición más. La segunda vez, un else simple, ya que es la última condición que necesitamos plantear.

Anidación de Condicionales

Explicar lo que es la anidación de condicionales es sencillo; comprenderlo quizás es algo más difícil, pero allá vamos. La anidación de condicionales significa introducir una declaración if (o else if o else) dentro de otra declaración if. Esto permite crear decisiones más complejas, verificando múltiples condiciones.

Para no hacernos un lío monumental, lo mejor será explicarlo directamente con un ejemplo. Presta atención, ya que el código se empieza a volver un poco más complejo.

Vamos a crear un programa que nos indique:

- Si el número es mayor que 10.
- Si el número es mayor que 5 pero menor o igual a 10.
- Si el número es mayor que 2 pero menor o igual a 5.
- Si el número es menor o igual a 2.

```
// Para este ejemplo, suponemos que el número a tener en cuenta es 3.
// Pero funcionaría igual con cualquier cifra.

int numero = 3;
if (numero > 10) {
   System.out.println("El número es mayor que 10");
} else {
   if (numero > 5) {
      System.out.println("El número es mayor que 5 pero menor o igual a 10");
   } else {
```

```
        if (numero > 2) {
            System.out.println("El número es mayor que 2 pero menor o igual a 5");
        } else {
            System.out.println("El número es menor o igual a 2");
        }
    }
}
```

Ese es el código, ahora vamos a analizarlo por partes.

1. Primera condición if:

- Comprobamos si número es mayor que 10.
- Si es verdadero, imprimimos "El número es mayor que 10".
- Si número no es mayor que 10, entramos en este bloque else.

2. Segunda condición if (anidada dentro del primer else):

- Comprobamos si número es mayor que 5.
- Si es verdadero, imprimimos "El número es mayor que 5 pero menor o igual a 10".
- Si número no es mayor que 5, entramos en este bloque else.

3. Tercera condición if (anidada dentro del segundo else):

- Comprobamos si número es mayor que 2.
- Si es verdadero, imprimimos "El número es mayor que 2 pero menor o igual a 5".
- Si número no es mayor que 2, imprimimos "El número es menor o igual a 2".

Haciendo un inciso y dándome cuenta de que en este caso la teoría es mucho más difícil que la práctica, me gustaría darte un consejo. Mantén tu código siempre lo mas sencillo y limpio posible. Aunque comprendo que, en algunas ocasiones, dependiendo de la complejidad del programa puede resultar imposible o al menos muy difícil.

Si tenemos que lidiar con situaciones en las que necesitamos utilizar muchas condicionales anidadas, lo mejor es que utilicemos else if. Mira en este ejemplo como el código queda mucho más claro:

```
int numero = 3;

if (numero > 10) {

   System.out.println("El número es mayor que 10");

} else if (numero > 5) {

   System.out.println("El número es mayor que 5 pero menor o igual a 10");

} else if (numero > 2) {

   System.out.println("El número es mayor que 2 pero menor o igual a 5");

} else {

   System.out.println("El número es menor o igual a 2");

}
```

Incluso explicarlo es más sencillo:

1. Primera Condición if:
    - Comprobamos si numero es mayor que 10.

2. Primera Condición else if:
    - Si numero no es mayor que 10, comprobamos si es mayor que 5.

3. Segunda Condición else if:
    - Si numero no es mayor que 5, comprobamos si es mayor que 2.

4. Condición else:

- Si ninguna de las condiciones anteriores es verdadera, ejecutamos este bloque.

Utilizando else if, podemos hacer que nuestro código sea más legible y fácil de entender. Esto es muy útil para tomar decisiones basadas en varios criterios.

Switch case

El switch-case en Java es una estructura condicional que se utiliza cuando tenemos varias opciones posibles y queremos ejecutar diferentes bloques de código dependiendo del valor de una variable. Es una versión más sencilla de escribir que un montón de if-else encadenados.

En un switch, la variable que estamos evaluando se compara con varios valores llamados case. Si encuentra una coincidencia, ejecuta el código correspondiente a ese case. Si no encuentra coincidencias, podemos usar un default, que es como un "plan B": el código en default se ejecutará si ninguna de las otras opciones se cumple.

Vamos a verlo con un ejemplo sencillo. Creamos un programa que le dice al usuario qué día de la semana es según un número del 1 al 7 (1-Lunes, 2-Martes, etc.). Usamos switch-case para decidir qué día corresponde a cada número.

```
public class Main {

  public static void main(String[] args) {

    int dia = 3;  // Supongamos que el usuario ingresa el número 3

    switch (dia) {

      case 1:

        System.out.println("Lunes");

        break;
```

```
            case 2:
               System.out.println("Martes");
               break;
            case 3:
               System.out.println("Miércoles");
               break;
            case 4:
               System.out.println("Jueves");
               break;
            case 5:
               System.out.println("Viernes");
               break;
            case 6:
               System.out.println("Sábado");
               break;
            case 7:
               System.out.println("Domingo");
               break;
            default:
               System.out.println("Número inválido");
                break;
      }
   }
}
```

Operadores de Comparación

Los operadores no solo van a sernos útiles con las condicionales; a partir de ahora, los vas a utilizar prácticamente en todos los programas y ejercicios que hagas. Por lo tanto, presta atención y no pierdas de vista esta tabla. Te aseguro que pronto la vas a necesitar, y hasta que te la aprendas de memoria, no te quedará más remedio que volver para echar un vistazo.

a == b: Comprueba si a es igual a b.

a != b: Comprueba si a es distinto de b.

a > b: Comprueba si a es mayor que b.

a >= b: Comprueba si a es mayor o igual que b.

a < b: Comprueba si a es menor que b.

a <= b: Comprueba si a es menor o igual que b.

Operadores lógicos

Lo mismo ocurre con los operadores lógicos: vas a hartarte de verlos y son fundamentales si quieres programar en Java. Ahora bien, considero que son quizás algo más difíciles de entender que los operadores de comparación. Por eso, después de enseñarte la tabla, te mostraré un pequeño ejemplo.

- && (AND lógico)

a && b: Comprueba si ambas condiciones a y b son verdaderas.

- || (OR lógico)

a || b: Comprueba si al menos una de las condiciones a o b es verdadera.

- ! (NOT lógico)

!a: Invierte el valor lógico de a.

```
public class Main {
   public static void main(String[] args) {

      int a = 6;

      // OR lógico
      // Utilizaremos || para ver si a es impar o mayor que 8.

      if (a>8 || a%2==0) {
         System.out.println("a es par o mayor que 8" );
      } else System.out.println("a no cumple ninguna de las dos condiciones");



      // AND lógico
      // Utilizaremos && para ver si a es par y además mayor que 8.

      int b = 10;
      if (b>8 && b%2==0) {

         System.out.println("b es par y mayor que 8" );
      } else System.out.println("b no cumple una de las dos condiciones ");



      // NOT lógico
      // Utilizamos este operador para invertir el valor lógico
      // En las variables a y b buscamos que fuesen pares. Para esta que sea impar.
```

```
        int c = 7;

        if (c%2!=0) {
            System.out.println("c es impar" );
        } else System.out.println("c es par" );

    }
}
```

Veamos otro ejemplo:

```
int x = 5;
int y = 10;

// AND lógico con operadores de comparación
if (x < 10 && y > 5) {
    System.out.println("x es menor que 10 y y es mayor que 5");
}


// OR lógico con operadores de comparación
if (x < 10 || y < 5) {
    System.out.println("Al menos una de las condiciones es verdadera");
}
```

```
// NOT lógico con operadores de comparación
if (!(x > 10)) {
   System.out.println("x no es mayor que 10");
}
```

- x < 10 && y > 5: Comprueba si x es menor que 10 y y es mayor que 5. Ambas condiciones son verdaderas, así que el mensaje se imprime.

- x < 10 || y < 5: Comprueba si x es menor que 10 o y es menor que 5. La primera condición es verdadera, así que el mensaje se imprime.

- !(x > 10): Comprueba si x no es mayor que 10. La condición original x > 10 es falsa, así que, !(x > 10) es verdadera y el mensaje se imprime.

Resumen de lo aprendido

Las condicionales en Java son totalmente necesarias para que los programas tomen decisiones basadas en diferentes escenarios. Sin condicionales, los programas seguirían instrucciones rígidas y no podrían adaptarse a cambios ni a datos diversos. Las condicionales añaden flexibilidad y dinamismo, permitiendo la creación de aplicaciones interactivas y funcionales que responden a las acciones del usuario, a cambios en el entorno o a datos en tiempo real.

La estructura básica de una condicional en Java comienza con la palabra clave "if", seguida de una condición que se comprueba. Si la condición es verdadera, se ejecuta el código especificado. Si la condición no se cumple, se pueden usar las declaraciones "else" y "else if" para manejar otras posibles situaciones, proporcionando bloques de código alternativos según sea necesario. La anidación de condicionales permite decisiones más complejas verificando múltiples condiciones dentro de otras.

Otra estructura un poco menos intuitiva es la de switch-case. Podríamos decir que es una versión más sencilla de escribir que un montón de if-else encadenados. La ventaja principal de esta estructura es su simplicidad cuando hay múltiples opciones posibles.

Además, hemos analizado los operadores de comparación y lógicos, ya que sin ellos no podríamos trabajar con condicionales. Los operadores de comparación incluyen ==, !=, >, >=, <, y <=.

Los operadores lógicos, como && (AND), || (OR), y ! (NOT), permiten combinar y manipular condiciones, haciendo el código más dinámico y eficiente. Utilizando estos operadores, podremos realizar comprobaciones complejas y tomar decisiones basadas en múltiples criterios.

Aunque en este punto la teoría pueda resultar un poco abrumadora, con los ejercicios que hemos resuelto no deberías tener ningún problema para manejarte con estos operadores. Una vez que completes los demás ejercicios propuestos para este tema, verás que no te quedará ninguna duda.

Para hacerte la vida más sencilla, te vuelvo a dejar la lista de todos los operadores. Tenla muy presente, ya que te va a resultar muy útil:

Operador	Descripción
+	Suma dos valores
-	Resta dos valores
*	Multiplica dos valores
/	Divide el primer valor por el segundo
%	Devuelve el resto de la división entera
++	Incrementa el valor de la variable en 1
--	Decrementa el valor de la variable en 1
+=	Agrega el valor especificado a la variable
-=	Resta el valor especificado de la variable
*=	Multiplica la variable por el valor especificado
/=	Divide la variable por el valor especificado
==	Comprueba si dos valores son iguales
!=	Comprueba si dos valores no son iguales
>	Comprueba si el primer valor es mayor que el segundo
<	Comprueba si el primer valor es menor que el segundo
>=	Comprueba si el primer valor es mayor o igual que el segundo
<=	Comprueba si el primer valor es menor o igual que el segundo
&&	Operador AND lógico
||	Operador OR lógico
!	Operador NOT lógico
?:	Operador ternario (condicional)
instanceof	Comprueba si un objeto es una instancia de una clase

Ejercicios del tema

Ejercicio 1. Escribe un día de la semana por teclado y, dependiendo del que toque, mostraremos la actividad que nos toca hacer. Las actividades son: Yoga, Bici, Gimnasio, Pintura y Lectura.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

//Creamos un scanner

      Scanner ejercicio = new Scanner(System.in);

//Ahora necesitamos pedir al usuario que introduzca un día de la semana

      System.out.println("Introduce un día de la semana");
      String dia = ejercicio.nextLine();

//Ahora escribiremos nuestra condicional

//Para el lunes. Escribimos con mayúscula y minúscula. Para ello utilizamos ||
//Con ello indicamos que nos vale cualquiera de las dos palabras

      if (dia.equals("Lunes") || dia.equals("lunes")) {
         System.out.print("Yoga");
      }
```

```
//Para el Martes.

      if (dia.equals("Martes") || dia.equals("martes")) {
         System.out.print("Bici");
      }


//Para el Miércoles. En este caso ponemos cuatro variables por si acaso alguien
olvida el acento.

 if      (dia.equals("Miércoles")      ||      dia.equals("Miercoles")      ||
dia.equals("miércoles") || dia.equals("miercoles")) {

         System.out.print("Gimnasio");

      }

//Para el Jueves.

      if (dia.equals("Jueves") || dia.equals("jueves")) {
         System.out.print("Pintura");
      }

//Para el Viernes.

      if (dia.equals("Viernes") || dia.equals("viernes")) {
         System.out.print("Lectura");
      }

   }
}
```

Ejercicio 2. Exactamente igual que el anterior, pero en caso de que introduzcamos un día incorrecto, el programa nos responderá si hemos introducido un dato erróneo.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);
      System.out.println("Introduce un día de la semana");

      String dia = ejercicio.nextLine();

//Vamos a utilizar else y system.exit de la forma que vas a ver a
continuación:

      if ( dia.equals("Lunes") || dia.equals("lunes")) {
         System.out.print("Yoga");
         System.exit(0);
      } else

//De esta forma el programa se cerrará en caso de que coincida con
el día de la semana que hemos introducido. De lo contrario utilizara
el else para irse a la siguiente condicional.

//Para el martes.

      if (dia.equals("Martes") || dia.equals("martes")) {
         System.out.print("Bici");
         System.exit(0);
      } else
```

```
//Para el Miércoles.

      if ( dia.equals("Miércoles") || dia.equals("Miercoles") ||
dia.equals("miércoles") || dia.equals("miercoles")) {

         System.out.print("Gimnasio");
         System.exit(0);

      } else

//Para el Jueves.

      if ( dia.equals("Jueves") || dia.equals("jueves")) {

         System.out.print("Pintura");
         System.exit(0);

      } else

//Para el Viernes.

      if ( dia.equals("Viernes") || dia.equals("viernes")) {

         System.out.print("Lectura");
         System.exit(0);

      } else System.out.println("Dato introducido erróneo");

//El Viernes al final, después del else introducimos la línea final que se
mostrará solo en caso de que ninguno de los días anteriores se haya
mencionado.

   }
}
```

Ejercicio 3. Introduce un número del 1 al 5 por teclado y, dependiendo del número, en la pantalla del programa saldrá un texto en un color: amarillo, azul, verde, negro o blanco.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      System.out.println("Introduce un número del 1 al 5");
      int numero = ejercicio.nextInt();

      if (numero == 1) {
         System.out.println("Amarillo");
         System.exit(0);
      }

      if (numero == 2) {
         System.out.println("Azul");
         System.exit(0);
      }

      if (numero == 3) {
         System.out.println("Verde");
         System.exit(0);
      }

      if (numero == 4) {
         System.out.println("Negro");
         System.exit(0);
      }
```

```
            if (numero == 5) {
                System.out.println("Blanco");
                System.exit(0);
            } else System.out.println("El dato indicado no es correcto");

        }
    }
```

Es importante que tus condicionales sean fáciles de leer y entender. Evita anidar demasiados if, else if y else, ya que esto puede hacer que el código se vuelva confuso. En su lugar, considera dividir la lógica en funciones separadas si se vuelve demasiado compleja.

Aprovecha los operadores lógicos (&&, ||, !) y de comparación (==, !=, >, <, >=, <=) para combinar condiciones y hacer tus if más eficientes. Asegúrate de entender cómo funcionan para evitar errores lógicos en tu código.

Ejercicio 4. Introduce un número por teclado que hará referencia a la temperatura de una piscina. Si es menor que 20, el programa dirá que es fría; para cualquier otra temperatura, dirá que es caliente. Puede tener decimales.

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      System.out.println("Introduce la temperatura de la piscina");

      double temperatura = ejercicio.nextDouble();

      if (temperatura < 20) {

         System.out.println("Fría");

      } else System.out.println("Caliente");

   }
}
```

Ejercicio 5. Vamos a introducir por teclado la nota de un examen. El programa nos dirá si el alumno ha suspendido, aprobado, sacado un notable o un sobresaliente. Si es menor de 5, es suspenso. Entre 5 y 6.99 es un suficiente. Entre 7 y 8.99 es notable, y mayor de 9 es sobresaliente.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);
      System.out.println("Introduce la nota del examen");
      double nota = ejercicio.nextDouble();

//Primero tomaremos en cuenta las notas que sean menores que 5

      if (nota < 5) {
         System.out.println("Suspenso");
         System.exit(0);
      }

//Después aquellas que sean mayores o iguales que 5
//Y a la vez menores que 7

      if (nota >= 5 && nota <7) {
         System.out.println("Suficiente");
         System.exit(0);
      }
```

```
//Ahora aquellas que sean mayores o iguales que 7
//Y a la vez menores que 9

      if (nota >= 7 && nota <9) {
         System.out.println("Notable");
         System.exit(0);
      }


 //Por último las que son mayores e iguales que 9
 //Y a la vez menores e iguales que 10
 //También incluimos una línea para indicar que cualquier valor mayor que
10 es un error.



      if (nota >= 9 && nota <=10) {
         System.out.println("Sobresaliente");
         System.exit(0);
      } else System.out.println("El dato introducido no es correcto");
   }
}
```

Revisa tu código para eliminar condiciones redundantes. Por ejemplo, si una condición ya ha sido comprobada y manejada, no es necesario volver a comprobarla. Esto simplifica el código y mejora el rendimiento.

Ejercicio 6. Vamos a crear un programa que calcule el salario de un trabajador. Las primeras 160 horas se cobran a 7,8 euros. Las horas que excedan las 160 se consideran horas extra y se pagan a 12,2 euros.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

//Creamos la variable salario. En este caso es necesario inicializarla.
//De lo contrario no podríamos utilizarla en la última línea de este programa.
//No se puede pedir al programa mostrar una variable si esta no tiene valor.
//Por lo tanto le ponemos valor cero, pero después cambiará dependiendo de los datos introducidos.

      double salario = 0;

      System.out.println("Introduce el número de horas trabajadas");

      int horas = ejercicio.nextInt();



//Primero hacemos un cálculo suponiendo que no ha habido horas extras.


      if (horas <= 160) {
         salario = (horas*7.8);
      }
```

```
// En este supuesto ha habido horas extras
// Por lo tanto sabemos que siempre hay más de 160. Así que siempre pagaremos 160 horas a 7.8 euros.
// A esas 160 horas ahora le sumamos las extras. Para saber cuántas horas extras hemos trabajado, restamos 160 al número de horas totales.
// Ejemplo. Si el trabajador hizo 180 horas. 180-160 = 20. Veinte son extras.
//Por último simplemente sumamos las horas normales más las extras.


      if (horas > 160) {

         salario = (160 * 7.8) + ((horas-160)*12.2);
      }

      System.out.println("El salario total es de " + salario + " euros");

   }
}
```


Coloca las condiciones que son más probables de cumplirse al principio de tus bloques if. Esto puede mejorar el rendimiento del programa al reducir la cantidad de comprobaciones necesarias en casos comunes.

Ejercicio 7. Introduce un número por teclado y el programa debe decirnos si es múltiplo de 5.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);
      System.out.println("Introduce el número");

      int numero = ejercicio.nextInt();

      if (numero%5==0) {
         System.out.println("El número es múltiplo de 5");
      } else System.out.println("El número no es múltiplo de 5");

   }
}
```

Ejercicio 8. Introducimos un número por teclado y pedimos que nos diga si es par o impar.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {
```

```
Scanner ejercicio = new Scanner(System.in);
      System.out.println("Introduce el número");

      int numero = ejercicio.nextInt();

      if (numero%2==0) {
         System.out.println("El número es par");
      } else System.out.println("El número es impar");

   }
}
```

Ejercicio 9. Crea un programa en el que introduzcas un número de tres cifras por teclado y el programa te dirá si es capicúa o no lo es.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

//Creamos un scanner

      Scanner ejercicio = new Scanner(System.in);

//Ahora necesitamos pedir al usuario que introduzca el número

      System.out.println("Introduce un número");
      int num = ejercicio.nextInt();
```

```
        if (num<=99) {
           System.out.println("El número introducido debe tener tres cifras");
        }
        if (num>999) {
           System.out.println("El número introducido debe tener tres cifras");
        }

//Calcular las centenas es fácil. Dividimos el número entre 100.
// Por ejemplo 356/100 = 3.56. Al ser una variable int no muestra decimales solo el 3.

        int centenas = num/100;

// Para las decenas haremos lo mismo solo que antes debemos eliminar las centenas.
//Ejemplo 678. Si pudiememos quitar el 6, nos quedaria 78.
//Si pudiesemos dividir 78 entre 10 nos quedaría 7. La centena que necesitamos.
//Entonces multiplicamos las decenas por 100. En este caso 6*100 = 600.
//Restamos a nuestro número 600; 678-600 = 78. 78/10 = 7.8 al ser un int = 7

        int decenas = ((num - (centenas * 100))/10);

// Hacemos lo mismo para conseguir las unidades.
//(Centenas * 100) + (decenas * 10) y se lo restamos a nuestro número.
//600 * 100 + 7*10 = 670. 678-670 = 8. La unidad que necesitamos.

        int unidades = (num - (centenas * 100) - decenas * 10);

        if (unidades==centenas) {
           System.out.println("El número es capicúa");
        } else System.out.println("El número no es capicúa");

    }
}
```

Ejercicio 10. Crea un programa que diga cuántas cifras tiene un número introducido por teclado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {


      Scanner ejercicio = new Scanner(System.in);


      System.out.println("Introduce un número");
      int num = ejercicio.nextInt();

      if (num>9999) {
         System.out.println("El número es demasiado grande");
      }
      if (num<10) {
         System.out.println("El número tiene una cifra");
      }
      if (num>=10&&num<100) {
         System.out.println("El número tiene dos cifras");
      }
      if (num>=100&&num<1000) {
         System.out.println("El número tiene tres cifras");
      }
      if (num>=1000&&num<10000) {
         System.out.println("El número tiene cuatro cifras");
      }
   }
}
```

Capítulo 5.

Conociendo a Bud. Aprendiendo a utilizar los bucles o loops

No paro de pensar en lo rápido que puede cambiarte la vida. Fernando y Rich han tenido bastante suerte, pero yo no he sido tan afortunado. Parece ser que la policía ha encontrado algo en mi ordenador, aunque aún no me han especificado nada. Mi abogado dice que me enfrento a varios años de cárcel, pero ni siquiera sé de qué se me acusa.

Mientras espero mi juicio, mi nuevo hogar es una pequeña celda compartida. La cama es muy incómoda, todo huele fatal y, lo peor de todo, es que no hay intimidad para ir al baño. Va a ser difícil acostumbrarme. En la vida real, tu cabeza está llena de estímulos por todas partes. Se mantiene ocupada, piensas poco y no te aburres. Aquí no hay absolutamente nada para entretenerse. Echo mucho de menos mi teléfono. Apenas puedo comunicarme con mis amigos y familiares, solo con los más cercanos. Me gustaría decirle a todo el mundo que soy inocente y que no he hecho nada malo. No sé qué pensará de mí la gente, pero seguro que nada bueno.

Cuando tienes demasiado tiempo libre, no paras de pensar en cosas. Evidentemente, pienso en lo que ha pasado. No hay que ser muy listo para darse cuenta de que el culpable es Chani. Desapareció misteriosamente pocos días antes de que apareciera la policía. Durante la fiesta, nuestros ordenadores se encontraban dentro de su armario y, por lo tanto, pudo haber accedido a cualquiera de ellos. Además, es muy raro que nadie pueda localizarlo en ningún sitio y que su teléfono esté desconectado.

Me pregunto por qué lo haría. Quiero decir, ¿por qué inculparme a mí o a Rich? Éramos amigos. Siempre lo pasábamos bien y nunca habíamos tenido ningún tipo de problema. Imagino que necesitaba a alguien a quien echarle la culpa y, al final, me ha tocado a mí. Me gustaría saber dónde está y si ha ganado algún dinero con su delito. Quiero entender por qué lo hizo y en qué clase de líos está metido. Soy consciente de que es posible que nunca lo sepa, pero aun así tengo la sensación de que algún día lo sabré.

Es obvio que estoy bastante preocupado, ya que no sé cuánto tiempo tendré que estar aquí. Cada minuto es una tortura y no me gustaría tener que pasar aquí mucho más tiempo. En el fondo, confío en que todo se aclare. No tengo nada que esconder y, además, cuento con el testimonio favorable de Rich. Él podrá ayudarme a demostrar que Chani es el verdadero culpable.

Mi familia ha conseguido un buen abogado, está preparando el caso y pronto me informará de cuál es la estrategia a seguir y a qué condena podría enfrentarme. En general, me siento bastante arropado y eso me permite seguir adelante.

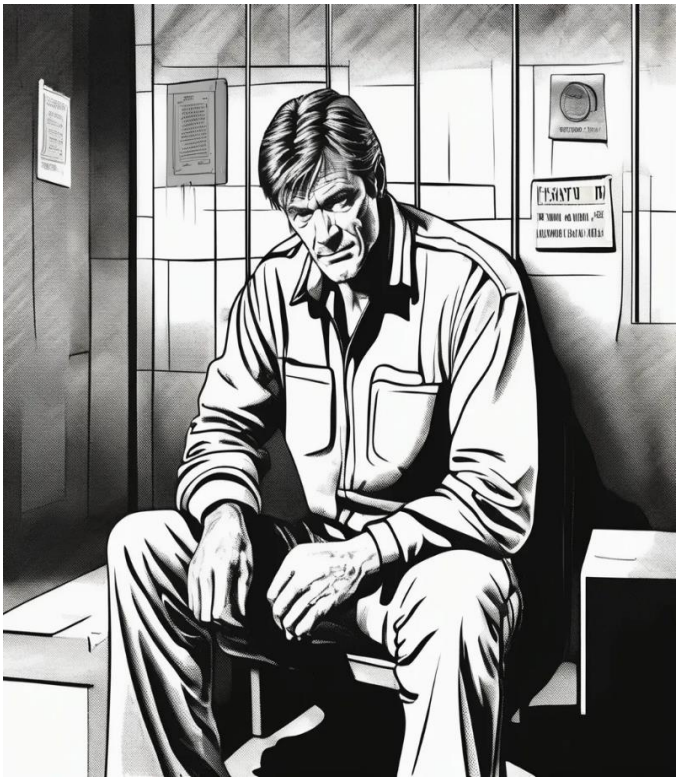
Las visitas alegran mis días y rompen la monotonía. Aunque me afecta ver sus rostros de preocupación, me reconforta saber que tengo a gente detrás luchando desde fuera. No me gustaría estar completamente solo. Aquí dentro estoy indefenso y me resulta imposible conseguir pruebas de mi inocencia. Ahora solo me queda confiar en que otras personas me saquen de esta situación.

Ahora que ya sabes que mi estado anímico es bueno, quizás te preguntes cómo es la vida en la cárcel. No me refiero solo a lo que se siente al estar dentro de la celda; eso ya lo he explicado. Estoy hablando de la relación con otros presos, la comida, el patio, la seguridad física…

No voy a mentir. El primer día estaba muy asustado. Desde el momento en que entré en prisión, noté una sensación muy rara dentro de mi cuerpo. Apenas podía respirar. Sentía una fuerte presión en el pecho que no me permitía prácticamente ni hablar. En las películas siempre dicen que hay que pretender ser duro, pero yo no podía aparentarlo. Me limité a caminar mirando al suelo hasta que llegué a mi celda.

Por suerte, esa sensación no duró mucho. Concretamente, se desvaneció cuando conocí a mi compañero de celda, un hombre de Valencia llamado “Bud”. Bueno, creo que es evidente que no es su verdadero nombre. Él se presentó así y yo nunca pregunté nada más.

Bud lleva en la misma celda los últimos 8 años y es muy respetado por todo el mundo allí. Conoce a mucha gente, tanto presos como funcionarios de prisión. Así que ser su amigo puede ofrecerme prácticamente inmunidad total. Mientras no busque pelea, la gente me dejará tranquilo.



Rápidamente, Bud se interesó por mi historia. Quería saber cómo había llegado hasta allí. Le expliqué la situación y se mostró comprensivo. Posteriormente, me informó de que él estaba allí porque trató de robar una joyería. No quiso dar más detalles y nunca me los daría.

Continuamos charlando y acabé comentándole que estaba estudiando ingeniería informática. Me preguntó si sabía programar en Java y le dije que tenía algunas nociones. Entonces, me explicó que, para reducir su condena, ayuda a los funcionarios de prisiones con el software de seguridad. Eso le permite tener un ordenador portátil de vez en cuando.

Pensé que tenía mucha suerte. Mi estancia en la celda sería mucho más amena si pudiera tener un ordenador y acceso a internet. Además de poder contactar con mis familiares y amigos, podría incluso aprender Java. Ser amigo de los carceleros puede tener grandes ventajas.

Creo que Bud me leyó la mente porque, rápidamente, me dijo que podría ayudarme a mejorar mis habilidades programando. Después de hacerme un pequeño test improvisado, pudo calcular más o menos mi nivel. Me preguntó si sabía qué eran los bucles o loops. La verdad es que no tenía ni idea, así que le dije que no.

Así fue como conseguí un profesor particular de Java en la cárcel. Me sentía muy afortunado por ello y no quería defraudar a mi nuevo mentor. Así que me lo tomé muy en serio. La lección de bucles había comenzado y a continuación te voy a contar todo lo que Bud me explicó.

Bucles o Loops

Definición, propósito e importancia de los bucles en la programación

Un bucle, o loop, en programación es una forma de hacer que tu programa pueda ejecutar repetidamente un bloque de código mientras se cumpla una determinada condición. Los bucles son una herramienta fundamental no solo en Java, sino en cualquier lenguaje de programación. Esto se debe a que nos permiten automatizar tareas repetitivas de manera eficiente.

En Java, existen principalmente tres tipos de bucles: for, while, y do-while. Cada uno de estos bucles tiene su propia sintaxis y aplicación específica, pero todos comparten el mismo propósito esencial: ejecutar repetidamente un conjunto de instrucciones.

Te explicaré los tres tipos de bucles, pero te aconsejo céntrate sólo en los dos primeros, for y while. Considero que, si estás dando tus primeros pasos en Java, al principio es fundamental dominar unas cuantas cosas básicas en vez de tratar de abarcar todo. No obstante, es importante que seas consciente de que existen tres. Si realmente te gusta la programación y se vuelve parte de tu vida, en algún momento te toparás con los bucles do-while y no quiero que te pillen por sorpresa.

Introducción a los bucles

Bucle for: Se utiliza cuando se conoce de antemano el número de iteraciones que se deben realizar.

Bucle while: Se emplea cuando no se sabe con certeza cuántas veces se debe repetir un bloque de código y la condición de terminación depende de alguna variable que se evalúa en cada iteración.

Bucle do-while: Similar al while, pero garantiza que el bloque de código se ejecute al menos una vez, ya que la condición se evalúa después de la primera ejecución.

Después de leer esta pequeña definición, lo más seguro es que aún no sepas de qué estamos hablando, aunque ya te vas haciendo una idea. Aun así, no te preocupes porque cuando veamos los ejemplos prácticos, todo te va a quedar claro. No obstante, antes de eso, es necesario enumerar las principales razones de la importancia de los bucles.

- **Automatización de Tareas Repetitivas:** Los bucles nos permiten que las tareas repetitivas se ejecuten automáticamente; de este modo, no necesitamos escribir el mismo código varias veces. Esto nos va a resultar especialmente útil para operaciones como el procesamiento de elementos de listas, la lectura de archivos, o la generación de secuencias de números.

- **Eficiencia y Rendimiento:** Si usamos bucles de manera adecuada, podemos mejorar significativamente la eficiencia y el rendimiento de nuestros programas. Evitaremos la redundancia y así, reduciremos la cantidad de código necesario para realizar tareas repetitivas. Esto hará que se optimice el uso de recursos del sistema.

- **Facilita el Mantenimiento del Código:** El código que utiliza bucles es generalmente más fácil de entender, mantener y modificar. Si se necesita cambiar la lógica de una operación repetitiva, solo se requiere modificar el bloque de código dentro del bucle en lugar de cambiar múltiples instancias del mismo código.

- **Manejo de Colecciones y Datos:** Los bucles son particularmente útiles para manejar colecciones de datos, como arrays y listas. Permiten recorrer los elementos de estas colecciones y realizar operaciones en cada uno de ellos de manera sistemática.

Bucle for

Un bucle for en Java es una forma de repetir un bloque de código **un número específico de veces.** Es útil cuando sabemos cuántas veces queremos que se ejecute el código.

Un bucle for tiene tres partes principales:

- Inicialización: Configura una variable para comenzar.
- Condición: Mientras esta condición sea verdadera, el bucle se seguirá ejecutando.
- Actualización: Cambia la variable después de cada repetición del bucle.

Aquí está la estructura básica de un bucle for:

```
for (inicialización; condición; actualización) {
   // Código a repetir
}
```

Ejemplo 1: Mostrar los números de 1 al 5.

```
for (int i = 1; i <= 5; i++) {
   System.out.println(i);
}
```

Vamos a analizar este ejercicio detenidamente. Lo primero que debemos hacer es declarar una variable. Cómo puedes ver en el ejemplo, le hemos dado el nombre de "i". Funcionaría igual independientemente del nombre, pero siempre la verás escrita así. Por lo tanto, te recomiendo que tú hagas lo mismo.

Lo segundo que vemos es que la variable i debe ser menor o igual que cinco.

Por último, i++ significa que después de cada repetición, i se incrementa en 1.

Si hubiésemos querido hacer una cuenta atrás. Que se mostrasen los números desde 5 hasta 1, lo haríamos de la siguiente forma:

```
for (int i = 5; i >= 0; i--) {
   System.out.println(i);
}
```

Otra vez comenzamos declarando la variable "i" y, además, le asignamos el valor del primer dato que necesitamos, en este caso, un 5. Después decimos que los valores que queremos mostrar deberán ser iguales o superiores a 0. Por último, utilizamos i- - para que i se vaya reduciendo de uno en uno.

Los bucles no solo sirven para mostrar listados de números. Poco a poco vas a ver que pueden utilizarse para muchas cosas. Vamos a ver otro ejemplo:

Creamos un programa que muestre "Hola" tres veces seguidas:

```
for (int i = 0; i < 3; i++) {
   System.out.println("Hola");
}
```

Necesitamos un bucle que se repita exactamente tres veces. Perfecto, como sabemos exactamente el número de veces, utilizamos un for. Seguimos la misma estructura que antes; en este caso, decimos que i empieza en 0 y necesitamos que i se repita tres veces, así que:

El primer i será = 0

El segundo i será = 1

El tercer i será = 2

Por lo tanto, necesitamos que la condición sea que i sea menor o igual que 2. Podríamos haberlo hecho de dos formas y daría exactamente igual:

- i < 3
- I <=2

En este caso, nos importa poco que nuestra variable empiece en 0, en 1 o en cualquier otro número, por ejemplo, 33. Simplemente necesitamos que el bucle se repita tres veces. Podría haber sido así:

```
for (int i = 33; i <=35; i++) {
   System.out.println("Hola");
}
```

Como ya tenemos un bucle que se repite tres veces, simplemente usamos una línea de código para que se imprima cada vez.

La cosa se va a complicar un poco más cuando veamos los ejercicios, pero espero que vayas entendiendo lo básico. El bucle for suele ser bastante sencillo de aprender, ya que simplemente hay que seguir unos pasos concretos. Creas una variable con el primer valor, defines el segundo y después acabas con i++ o i--, dependiendo de cada escenario. El bucle while, en cambio, no es tan estructurado y, aunque en principio puede parecer incluso más sencillo que el for, cuando te enfrentas a ejercicios complejos es cuando te das cuenta de que aplicar el while no es tan intuitivo como pensabas. Aun así, verás cómo enseguida lo aprendes sin problemas.

Bucle While

Un bucle while sigue esta estructura básica:

```
while (condición) {
   // Código a repetir
}
```

Parece muy sencillo, ¿verdad? Bueno, lo cierto es que lo es. Quizás te metí un poco de miedo antes diciéndote que es más difícil que el bucle for. En realidad, cuando explicamos los ejercicios básicos, todo parece muy simple, pero después, cuando los enunciados se van complicando, es cuando tendrás que darle vueltas a las cosas para hallar la respuesta que buscas. Pero no nos adelantemos, veamos un ejemplo:

Vamos a usar un bucle while para imprimir los números del 1 al 5.

```
int i = 1; // Inicialización
while (i <= 5) { // Condición
   System.out.println(i);
   i++; // Actualización
}
```

Creamos y asignamos valor a una variable tipo int fuera de nuestro bucle. De momento, y antes de iniciar el bucle, la variable sigue teniendo un valor de 1. Pero las variables pueden ir cambiando de valor a lo largo que avanza el programa. Recuerda que, al ejecutarse, el programa va leyendo las líneas de código de arriba abajo.

Cuando se ejecuta el bucle por primera vez, i vale uno, y va a mostrar la línea de código System.out.println(i); en este caso "1".

Pero, ¿qué pasa justo después? Encontramos un i++, así que ahora i vale 2. Como hemos dicho que el bucle no se cierra hasta que i sea igual o mayor que 6, mostrará de nuevo System.out.println(i); y en este caso veremos "2".

¿Qué pasa en la siguiente iteración? i vale 3… se vuelve a repetir el ciclo. Así hasta llegar a 6, y en ese momento el bucle se cierra y no lo mostrará.

¿Subimos un poco el nivel?

Utilizaré un ejercicio parecido al anterior, ya que así te resultará familiar. En este caso, crearemos un bucle del 1 al 10. Pero vamos a meter un if dentro. Mostraremos un texto que diga: "El valor es mayor que 6". Necesitamos que se muestre i y luego, al lado, el texto que he mencionado, para i = 7, 8, 9 y 10.

```
public class Main {
   public static void main(String[] args) {

      int i = 1;
      while (i <= 10) {

         if (i<=6) {
            System.out.println(i);
         } else System.out.println(i + "  El valor es mayor que 6");
         i++;

      }

   }
}
```

Es sencillo. Una vez que teníamos el bucle hecho, necesitábamos que para los 6 primeros números ocurriera una cosa y para los 4 siguientes, otra distinta. Los seis primeros muestran solo el valor de i, mientras que los 4 siguientes muestran i y además el mensaje. Puedes intentar repetir el ejercicio por tu cuenta utilizando un bucle for en vez del while que he utilizado yo. Es incluso más fácil.

Bucles do while

Un bucle do-while es una estructura que repite nuestras instrucciones mientras se cumpla una condición. Lo que lo diferencia de otros bucles es que siempre ejecuta el código al menos una vez, incluso si la condición es falsa desde el principio.

- Primero, el código dentro del bucle se ejecuta.
- Después, se evalúa la condición.
- Si la condición es verdadera, el código se ejecuta nuevamente. Si es falsa, el bucle termina.

Este tipo de bucle es útil cuando queremos asegurarnos de que el bloque de código se ejecute al menos una vez, sin importar la condición. Aquí tienes un ejemplo:

```
int numero = 1;

do {
   System.out.println(numero);          // Imprime el número actual
   numero++;                            // Incrementa el número en 1
} while (numero <= 5);                  // Repite mientras el número sea menor o igual a 5
```

- Primero, imprime el número 1.
- Luego verifica la condición numero <= 5. Como sigue siendo verdadera, el bucle continúa.
- El proceso se repite hasta que el número llega a 6, momento en el que la condición se vuelve falsa y el bucle termina.

Concepto de Break

Cuando el programa encuentra un break, sale del bucle y continúa con la siguiente línea de código ya fuera del bucle. La utilización del break es importante sobre todo por dos motivos.

- El primero es que nos permite salir de un break prematuramente. Algo que verás que va a ser necesario hacer en muchas situaciones. Sin ir más lejos, en varios de los ejercicios del final del tema.

- El segundo es que mejora la eficiencia del programa ahorrando recursos. Si sabes que ya has encontrado lo que buscabas, no hay necesidad de recorrer el bucle hasta el final.

Además, el break nos va a ayudar a simplificar la lógica del programa; hace el código más limpio, sencillo y fácil de entender. Por último, va a ser una solución elegante para el manejo de casos excepcionales. En algunos casos, dentro de un bucle puedes encontrar situaciones excepcionales como errores o condiciones inválidas que requieren cerrar el bucle de inmediato. No encontrarás nada más sencillo ni rápido que un break.

Ejemplo de uso de break

Imagina que creas un bucle infinito utilizando un bucle while. Podríamos hacerlo de la siguiente forma:

```
int a = 1;
while (a >= 0) {
   a++;
   if (a > 10) {
      System.out.println(a);
   }
}
```

Según ese bucle, mientras a sea mayor que 0, se va a seguir repitiendo. Nuestro programa estaría infinitamente mostrando número tras número de uno en uno en nuestra pantalla. Ahora bien, queremos que muestre solamente el número posterior a 10 y que después el bucle se cierre.

```
int a = 1;
 while (a >= 0) {
    a++;
    if (a > 10) {
       System.out.println("Primer número mayor que 10 es = " + a);
       break; // Sale del bucle cuando se encuentra el primer número mayor que 10
    }
 }
```

A estas alturas ya te habrás dado cuenta de que este ejercicio se podría haber hecho de otra forma. No sólo eso, sino que se puede hacer de muchas formas distintas. El código es sólo una herramienta que tiene el programador para resolver problemas. Está a nuestra disposición y se puede utilizar como mejor nos convenga en cada situación.

Concepto de continue

Un continue se utiliza para saltar a la siguiente iteración del bucle, omitiendo el código entre medio dentro del bucle para la iteración actual. Cuando nuestro programa encuentra una instrucción continue, el bucle no se detiene, sino que salta hasta la próxima iteración. Esto es útil cuando deseas omitir ciertas iteraciones bajo condiciones específicas sin detener el bucle por completo.

Ejemplo de continue

Imagina que tienes un bucle que cuenta del 1 al 10, pero quieres omitir el número 5. Puedes usar continue para lograr esto.

```
public class Main {
   public static void main(String[] args) {
      for (int i = 1; i <= 10; i++) {
         if (i == 5) {
            continue; // Omite el resto del bucle cuando i es igual a 5
         }
         System.out.println(i);
      }
   }
}
```

Ambos conceptos, break y continue, son herramientas útiles para controlar el flujo de ejecución dentro de los bucles en Java, permitiéndote manipular la iteración de acuerdo con tus necesidades específicas.

Manipulación de Cadenas

Aunque la manipulación de cadenas podría perfectamente tener un tema propio, para no alargar la cosa en exceso he decidido mostrarte cómo funciona este concepto en este tema. Además, a partir de ahora vas a ver en los ejercicios dos métodos en concreto y, para que no te pillen por sorpresa, voy a explicarte qué son y para qué se utilizan. Esto va a enriquecer notablemente tu conocimiento de la sintaxis.

Métodos charAt() y length()

- **El método charAt()**

El método charAt() en Java se utiliza para obtener el carácter en una posición específica dentro de una cadena de texto. Toma un índice como argumento y devuelve el carácter en esa posición dentro de la cadena.

Sintaxis:

char charAt(int indice)

indice: Es el índice del carácter que se desea obtener. El primer carácter de la cadena tiene un índice de 0, el segundo tiene un índice de 1, y así sucesivamente.

Ejemplo:

```
String texto = "Hola Mundo";
char caracter = texto.charAt(0);
System.out.println("El carácter en la posición 0 es: " + caracter);  // Output: 'H'
```

En este caso, charAt(0) devuelve el primer carácter de la cadena "Hola Mundo", que es 'H'. Si usas otro índice, obtendrás el carácter en esa posición, como por ejemplo charAt(5) te devolvería 'M'.

- **El método length()**

El método length() en Java se utiliza para saber cuántos caracteres tiene una cadena de texto. Devuelve un número entero que representa la longitud de la cadena.

Sintaxis:

int length()

Ejemplo:

```
String saludo = "Hola";
int longitud = saludo.length();
System.out.println("La longitud de la cadena es: " + longitud); // Output: 4
```

No tiene mayor complicación; en los siguientes ejercicios vas a encontrarte estos métodos unas cuantas veces. En cuanto empieces a trabajar con ellos, verás que son muy sencillos de utilizar.

Resumen de lo aprendido

Un bucle permite que un bloque de código se ejecute repetidamente mientras que se cumpla una condición específica. Esta técnica es esencial para automatizar tareas repetitivas de manera eficiente. En Java, los principales tipos de bucles son for, while y do-while, cada uno con una estructura y uso particular.

El bucle for se utiliza cuando se conoce de antemano el número de veces que se debe ejecutar el código. En cambio, el bucle while se emplea cuando no se sabe cuántas veces se repetirá el bloque de código, ya que la condición de terminación depende de una variable evaluada en cada iteración. Por su parte, el bucle do-while es similar al while, pero garantiza que el bloque de código se ejecute al menos una vez, evaluando la condición después de la primera ejecución.

Los bucles son necesarios por varias razones:

- En primer lugar, automatizan tareas repetitivas, permitiendo que el código se ejecute varias veces sin necesidad de escribirlo repetidamente.
- En segundo lugar, mejoran la eficiencia y el rendimiento al reducir la redundancia y optimizar el uso de recursos. Además, facilitan el mantenimiento del código, ya que es más sencillo modificar un bloque de código dentro de un bucle que múltiples instancias del mismo código.
- Por último, son especialmente útiles para manejar colecciones de datos, como arrays y listas, permitiendo realizar operaciones sistemáticas en cada elemento.

Además, en este tema hemos visto dos comandos muy útiles para cuando trabajamos con condicionales.

- El concepto de "break" se utiliza para salir de un bucle antes de completar todas las iteraciones. Esto resulta útil para mejorar la eficiencia del programa y simplificar la lógica del código, haciéndolo más limpio y fácil de entender.
- Por otro lado, "continue" se utiliza para saltar a la siguiente iteración del bucle, omitiendo el código intermedio para la iteración actual. Ambos conceptos son herramientas valiosas para controlar el flujo de ejecución dentro de los bucles en Java y manipular la iteración según las necesidades específicas del programa.

Ejercicios del tema

Ejercicio 1. Crear un programa que muestre los múltiplos de 5 entre los números 0 y 100 usando un bucle for.

Solución:

```
public class Main {
   public static void main(String[] args) {

 //Creamos un bucle for
 //i = 0 ya que es nuestro primer valor,
 //después ponemos 100 como valor máximo.
 //Por último añadimos i++ para indicar que el el valor aumente de uno en 1.

     for (int i = 0; i<=100; i++) {

  //Ahora queremos mostrar los números que sean múltiplos de 5.
  //Así que añadimos un if.

        if (i%5==0) {
           System.out.println(i);
        }

     }

   }
}
```

Ejercicio 2. Crear un programa que muestre los múltiplos de 5 entre los números 0 y 100 usando un bucle while.

Solución:

```
public class Main {
   public static void main(String[] args) {

//Es necesario crear la variable antes de escribir el bucle while.

      int i=0;

//Ahora vamos con el bucle while.
//Mientras el valor sea menor o igual que 100, nuestro programa se ejecutará

       while (i<=100) {

         //Utilizamos un if como para mostrar solamente los múltiplos de  5.
          if (i%5==0) {
             System.out.println(i);
          }

         // Utilizamos i++ para que nuestra variable aumente de uno en uno.
          i++;


       }
   }
}
```

Ejercicio 3. Mostrar números comprendidos entre 320 y 160 de veinte en veinte y hacia atrás usando un bucle for.

Solución:

```
public class Main {
   public static void main(String[] args) {

      for (int i =320; i>=160; i-- ) {
         if (i%20==0){
            System.out.println(i);
         }

      }
   }
}
```

Inicializa la variable de control: La variable de control en un bucle en Java es una variable que se usa para contar o seguir el número de veces que el bucle se ha ejecutado. Ayuda a decidir cuándo el bucle debe parar.

Ejercicio 4. Mostrar números comprendidos entre 320 y 160 de veinte en veinte y hacia atrás usando un bucle while.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int i = 320;

      while (i>=160) {

         if (i%20==0) {
            System.out.println(i);
         }
         i--;
      }

// En este caso hemos colocado i-- debajo de la línea de if
// Esto se debe a que el programa lee de arriba a abajo.
// Si hubiésemos colocado i-- antes del if, el primer valor que considerado sería 319.
// Por lo tanto hubiésemos perdido el valor 320

   }
}
```

Actualiza la variable de control: Modifica la variable de control en cada iteración para que el bucle pueda avanzar y eventualmente terminar.

Ejercicio 5. Creamos un programa que pida una contraseña para abrir una maleta. La contraseña es 7678. Si se acierta, se abre. Pero si después de 5 intentos no se consigue, se bloquea.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

// Primero creamos nuestra contraseña.

      int password = 7678;

//Creamos un scanner ya que tendremos que meter dato por teclado.
      Scanner ejercicio = new Scanner(System.in);

 //Usamos un for para tener 5 intentos.
 //Añadimos i++ para que los intentos vayan de uno en uno.

     for (int i = 0; i<5; i++ ) {

        System.out.println("Introduce una contraseña");

 //Introducimos datos por teclado


  int intento = ejercicio.nextInt();

//Ahora usamos nuestra condicional.

        if (intento == password) {
           System.out.println("Maleta abierta");
```

```
//Es necesario poner un System.exit si acertamos la contraseña
 //Así el programa se cerrará.

            System.exit(0);

         } else System.out.println("Dato erróneo");
       }

//Fuera del bucle, después de los cinco intentos, añadimos la siguiente linea:
       System.out.println("Te quedaste sin intentos");

   }
}
```

Ejercicio 6. Crea un programa que muestre la tabla de multiplicar del 8.

Solución usando for:

```
public class Main {
   public static void main(String[] args) {
      for (int i=1; i<=10; i++ ) {
         System.out.println(i +" x " + 8 + " = " + (i*8));
      }
   }
}
```

Solución usando while:

```
public class Main {
   public static void main(String[] args) {
      int i = 0;
      while (i<10) {
         i++;
         System.out.println(i +" x " + 8 + " = " + (i*8));
      }
   }
}
```

Ejercicio 7. Crea un programa que calcule la media aritmética de números introducidos por teclado hasta que se introduzca uno negativo. (El número negativo no cuenta para la media).

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

//La media aritmética es la división de la suma total entre el número de valores.
// Así que vamos a crear dos variables.

double sumaTotal =0;
double datosTotales=0;
```

```
//Necesitamos crear una variable que albergue el valor de cada vez que introduzcamos una cifra.

      int numero = 0;

//Creamos la variable media.

      double media =0;

//A todas las variables les asignamos valor cero ya que de momento es lo que valen.
//Creamos el while sólo para cuando nuestra cifra introducida sea mayor de 0.

      while (numero>=0) {

            System.out.println("Introduce una número");
            numero = ejercicio.nextInt();

 // Este while toma en consideracion todas las cifras introducidas.
 // Pero la última tiene valor negativo y no queremos usarla.
 // Así que metemos un if para que solo cuente valores positivos.

          if (numero>=0) {
            datosTotales++;
            sumaTotal = sumaTotal + numero;
          }
      }

      media = sumaTotal/datosTotales;

          System.out.println("Suma total es = " + sumaTotal);
          System.out.println("El numero de cifras introducidas es " +
          datosTotales);
          System.out.println("La media es: = " + media);

   }
}
```

Ejercicio 8. Creamos un programa que muestre el cuadrado y el cubo de los 5 números siguientes a uno introducido por teclado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      System.out.println("Introduce un número");

      int numero = ejercicio.nextInt();

      //Usamos un for. El primer valor es numero +1.
      // El ultimo valor deberá ser menor o igual que numero +5

      for (int i= numero +1; i<=numero+5;i++) {

         System.out.println("El número es: " + i + ", su cuadrado es " +
         i*i + ", el cubo es " + i*i*i);

      }
   }
}
```

Ejercicio 9. Crea un programa en el que introduzcamos 10 números por teclado y este nos diga cuántos son pares y cuántos son impares.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

// Creamos dos variables para contabilizar la cantidad de pares e impares.

      int pares = 0;
      int impares =0;

// Creamos un bucle for que nos permita introducir diez datos

      for (int i= 0; i<10;i++) {

        System.out.println("Introduce un número");
        int dato = ejercicio.nextInt();

// Ahora mediante un if, hacemos que se sume 1 a los pares o a los impares
// Se puede hacer de dos formas. Añadiendo ++ o haciendo: variable = variable +1

          if (dato%2==0) {
            pares = pares +1;
          } else impares++;

      }
```

```
//Ahora simplemente mostramos la suma total tanto de pares como impares.

        System.out.println("Total de números pares: " + pares);
        System.out.println("Total de números impares: " + impares);

    }
}
```

Ejercicio 10. Complicamos un poco el ejercicio 5. Creamos un programa que pida una contraseña para abrir una maleta. La contraseña es 7678. Si se acierta, se abre. Pero si después de 5 intentos no se consigue, se bloquea. Si introduces un número con más o menos de 4 cifras, el programa te dirá que la contraseña debe tener 4 cifras y no contará como intento.

Solución:

```
import java.util.Scanner;
public class Main {
    public static void main(String[] args) {

        int password = 7678;
        int intentosProbados = 0;


        Scanner ejercicio = new Scanner(System.in);
```

```
while (intentosProbados<5) {

        System.out.println("Introduce una contraseña");


        int intento = ejercicio.nextInt();

        if (intento > 999 && intento < 100000  ) {
           intentosProbados ++;
        }

        if (intento <= 999 || intento >= 100000 ) {
           System.out.println("La contraseña debe tener cuatro cifras");
        }else


//Ahora usamos nuestra condicional.



        if (intento == password) {
           System.out.println("Maleta abierta");

 //Es necesario poner un System.exit si acertamos la contraseña
 //Así el programa se cerrará.

           System.exit(0);

        } else System.out.println("Contraseña incorrecta");
      }

//Fuera del bucle, después de los cinco intentos, añadimos la siguiente
línea:

      System.out.println("Te quedaste sin intentos");

   }
}
```

Ejercicio 11. Crear un programa en el que se introduzcan una base y un exponente, y que muestre el resultado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce la base");
      int base = ejercicio.nextInt();

      System.out.println("Introduce el exponente");
      int exponente = ejercicio.nextInt();

//Creamos también una variable resultado que de momento tiene el mismo valor que la base.

      int resultado =base;

// Cuando elevamos algo al cuadrado multiplicamos base por base osea una multiplicación.
// Al elevarlo al cubo multiplicamos base por base, por base. Dos multiplicaciones.
// Al elevarlo a 4, son tres operaciones. Osea siempre una menos que el exponente.
// Por lo tanto este este sera nuestro for.

      for (int i = 0; i<exponente -1;i++) {
         resultado = resultado*base;
      }

      System.out.println("El resultado es " + resultado);

   }
}
```

Ejercicio 12. Creamos un programa en el que introduzcamos un número por teclado y este nos diga si es primo o no.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce un número");

      int num = ejercicio.nextInt();

//Un número primo es el que puede dividirse tan solo entre 1 y él mismo.
//Por lo tanto vamos a hacer un bucle con un int i, que vaya de uno en uno.
//Desde 1 hasta el propio número.
//Hacemos que cada vez nuestro número se divida entre i
//Creamos un int contador.
//Cada vez que nuestro número sea divisible entre i, el contador sumará 1.
//Si el contador es 2 significa que es divisible entre 1 y él mismo. Primo.
//Si el contador es mayor que dos entonces no será primo.

      int contador = 0;

      for (int i = 1; i<=num; i++) {

         if(num%i==0) {
            contador++;
         }
      }

      if (contador <=2) {
         System.out.println ("Es primo");
      } else System.out.println ("No es primo");

   }
}
```

Ejercicio 13. Crea un programa que sume los siguientes 100 números a un número insertado por teclado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce un número");

      int num = ejercicio.nextInt();

      int suma= 0;

      for (int i = num+1; i<=num+100;i++) {

         System.out.println(i);
         suma = suma+i;

      }

      System.out.println("La suma total es " + suma);

   }
}
```

Evita bucles infinitos: Verifica que la condición del bucle cambiará en algún momento para que el bucle no se ejecute indefinidamente.

Ejercicio 14. Crea un programa que pida dos números. Luego, que muestre los números entre ellos, comenzando por el primero y avanzando de 7 en 7.

Solución usando for:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce el número a");

      int numA = ejercicio.nextInt();

      System.out.println("Introduce el número b");

      int numB = ejercicio.nextInt();

      for (int i = numA; i<=numB;i+=7) {

         System.out.println(i);

      }
   }
}
```

Solución usando while

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {


      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce el número a");

      int numA = ejercicio.nextInt();

      System.out.println("Introduce el número b");

      int numB = ejercicio.nextInt();

      int datoActual=numA;


      while (datoActual<=numB) {

         System.out.println(datoActual);
         datoActual += 7;

      }
   }
}
```

Usa break y continue con moderación: Utiliza break para salir del bucle antes de tiempo y continue para saltar a la siguiente iteración, pero hazlo con cuidado para mantener tu código claro y comprensible.

Ejercicio 15. Introduce 10 números por teclado y calcula la media de los números impares y el mayor de los números pares.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);


      int dato;
      double sumaImpares = 0;
      double cantidadImpares = 0;
      int mayorPares = 0;


      for (int i=1; i <=10;i++) {

         System.out.println("Introduce el número " + i);
         dato = ejercicio.nextInt();

         if (dato%2!=0) {
            cantidadImpares ++;
            sumaImpares = sumaImpares +dato;
         } else
```

```
            if (dato>mayorPares ) {
                mayorPares = dato;

            }
        }

        System.out.println("El mayor de los pares es " + mayorPares);
        System.out.println("La media de los imapres es " +
(sumaImpares/cantidadImpares));

    }
}
```

Ejercicio 16. Programa que pida números hasta que la suma sea 1000. Después, muestra dicha suma en la pantalla.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      int suma = 0;

      while (suma <1000) {
         System.out.println("Introduce un número");
         int dato = ejercicio.nextInt();
         suma = suma + dato;
      }
```

```
        System.out.print("La cifra final es de " + suma);

    }
}
```

Ejercicio 17. Creamos un programa en el que introduzcamos un número por teclado. Después, el programa calcula todos los múltiplos de 3 hasta llegar a esa cifra, los suma y muestra el resultado en la pantalla.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Introduce un número");

      int num = ejercicio.nextInt();
      int suma = 0;

      for (int i=1;i<=num;i++){

         if (i%3==0) {
            System.out.println(i);
            suma = suma+i;
         }
      }
```

```
        System.out.println("La suma total es " + suma);

    }
}
```

Ejercicio 18. Crea un programa que muestre todos los números del 1 al 1000 que no son múltiplos del número introducido por teclado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Ingrese el multiplo");

      int multiplo = ejercicio.nextInt();

      for (int i=1;i<=1000;i++) {

         if (i%multiplo!=0) {
               System.out.println(i + " no es multiplo de " + multiplo );
         }

      }
   }
}
```

Ejercicio 19. Escribe un programa que solicite al usuario una cadena de texto y un carácter específico, luego cuente y muestre cuántas veces aparece ese carácter en la cadena.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.print("Ingrese una cadena de texto: ");
      String frase = ejercicio.nextLine();

      System.out.print("Ingrese el carácter a contar: ");
      char letra = ejercicio.next().charAt(0);

      int contador = 0;



      for (int i = 0; i < frase.length(); i++) {

         if (frase.charAt(i) == letra) {
            contador++;
         }

      }
      System.out.print("El n'umero de veces repetido es " + contador);
   }
}
```

Ejercicio 20. Crea un programa que cuente las vocales de una frase introducida por teclado.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner (System.in);

      System.out.println("Ingrese la frase");

      String cadena = ejercicio.nextLine();

      int contador = 0;

      for (int i = 0; i < cadena.length(); i++) {
         char caracter = Character.toLowerCase(cadena.charAt(i));
          if (caracter == 'a' || caracter == 'e' || caracter == 'i' || caracter ==
          'o' || caracter == 'u') {
            contador++;
         }
      }

      System.out.println(contador);

   }
}
```

Define una condición de salida clara: Establece una condición que permita al bucle terminar en un momento adecuado para evitar bucles infinitos.

Capítulo 6.

Ayudando a Bud. Números aleatorios.

Tengo noticias de mi abogado y no son nada buenas. Mi ordenador fue utilizado para acceder a una página delictiva donde se facilitaban servicios para hackear los sistemas de seguridad de empresas. Pero eso no es todo, este software está siendo utilizado en estos momentos para extorsionar a varias multinacionales.

La policía cree que ideé el plan junto con algún cómplice y que este aún sigue en libertad. Imagino que piensan que es Chani. Ojalá lo atrapen para que se pueda aclarar todo este asunto. Mientras tanto, defenderme de las acusaciones es muy difícil. Mi ordenador estuvo involucrado y poca gente cree que lo utilizaran sin mi permiso.

Las declaraciones de Rich son las únicas que pueden ayudarme. Ha venido a visitarme en un par de ocasiones y me ha comentado que ha hablado varias veces con la policía para explicarles cómo llegó mi portátil a manos de Chani. Incluso les comentó que en nuestra casa no cerrábamos las puertas con llave y que dejábamos nuestros aparatos electrónicos en cualquier lugar.

La verdad es que no soy muy optimista. Imagino que Chani habrá huido lejos. Si no le detienen, voy a pasar bastantes años aquí. De nuevo, vuelvo a pensar en cómo cambia la vida en un instante. No he hecho absolutamente nada y ahora debo pagar por un delito del que no tengo nada que ver. Lo peor es que no puedo hacer nada. Solo me queda esperar a ver qué pasa.

Mi círculo social se ha visto reducido drásticamente; antes llenaba la casa de amigos los fines de semana. Ahora, prácticamente solo hablo con Bud y una vez al día con las visitas que recibo del exterior. Mi familia está muy preocupada y sé que están tratando de ayudarme desde fuera. No me siento solo, pero a veces echo en falta tener más compañía.

Al menos tengo a Bud; pasamos casi todo el día hablando. La mayor parte del tiempo, de tonterías, nada serio. Es mejor tener la cabeza despejada e intentar desconectar un poco. A veces me pregunta por mi situación y sigue los avances, pero la verdad es que ni siquiera yo sé lo que está pasando.

Bud tiene suerte; puede pasar gran parte del día con el ordenador. Incluso puede salir de la celda en muchas ocasiones para ayudar a los empleados de la cárcel con distintas tareas informáticas. Quizás, si aprendo, yo también podré echar una mano. Puede ser una buena oportunidad para aprender cosas, entretenerme y, de paso, ver si pueden reducir mi condena.

A Bud le encanta enseñar y continuamente me hace preguntas sorpresa. Me ha dicho que, si me aplico, es posible que me dé un trabajillo o que al menos me deje ayudarle. Tenemos un trato. Va a enseñarme cómo operar con números aleatorios en Java y después me hará un pequeño examen. Si lo supero, me dejará ayudarle.

Trabajando con números aleatorios

Importancia de los números aleatorios en Java

Los números aleatorios se utilizan constantemente en todo tipo de programas. Pueden servirnos, entre otras cosas, para simular eventos aleatorios en juegos y simulaciones, generar datos para pruebas de software, y optimizar algoritmos mediante decisiones aleatorias.

Además, son vitales en la seguridad informática para crear claves y en el análisis de datos para modelar distribuciones estadísticas. En resumen, los números aleatorios en Java son herramientas versátiles que mejoran la robustez, la seguridad y el rendimiento de los sistemas informáticos en diversas aplicaciones.

Generación de Números Aleatorios con Math.random

Antes de nada, debes saber que no hay una única forma de generar números aleatorios. Voy a mostrarte la que, según mi punto de vista, es la más sencilla. En Java, la generación de números aleatorios se puede realizar utilizando la clase Math y su método estático random(). El número que obtenemos es mayor o igual a 0.0 y menor que 1.0. Esto significa que puede generar números como 0.0, pero nunca alcanza 1.0 (es decir, 0.999999999... es el valor máximo posible).

Como puedes observar tenemos alguna limitación, pero nada que no podamos solucionar con un poco de imaginación y creatividad.

Ahora, veamos cómo utilizar Math.random() en la práctica para generar números aleatorios en Java:

- **Generamos un número aleatorio entre 0.0 (inclusive) y 1.0 (exclusivo)**

```
double numero = Math.random();
System.out.println("Número aleatorio generado: " + numero);
```

- **Generamos un número aleatorio entre dos cifras. Utilizando decimales.**

  Para ello utilizamos la siguiente estructura:

  double numero = Math.random() * (max - min) + min;

  Por ejemplo, si queremos crear un número aleatorio entre 10.0 (Incluido) y 100.0 (Excluido) lo haremos de la siguiente forma:

  ```
  double numero = Math.random() * (100 - 10) + 10;
  System.out.println("Número aleatorio entre 10.0 y 100.0: " + numero);
  ```

- **Generamos un número aleatorio entre dos cifras sin decimales.**

  int numero = (int) (Math.random() * (max – min +1) + min);

  Hay dos cambios en esta línea de código. Lo primero que vemos es la utilización de (int). Gracias a eso, nuestra sintaxis pasa de referirse a una variable decimal a una entera. Fíjate que el resto de la línea también va entre paréntesis. Comienza justo antes de Math.random() y acaba al final. Esto, básicamente, lo que hace es eliminar todos los decimales. Por ejemplo, si nuestro número aleatorio es 88.87, ahora será 88.

  Si seguimos con el ejemplo y queremos números que vayan del 10 al 100, antes podíamos conseguir cifras que iban desde 10.00 hasta 99.999999… Por lo tanto, si les quitamos los decimales, lo que tenemos es un rango entre 10 y 99. El problema es que no incluye el 100. Eso nos lleva al siguiente cambio, que es el +1. Al añadir ese elemento, se soluciona nuestro problema, ya que el rango ahora se amplía hasta 100.99999…, que al convertirse en natural nos da 100. De este modo, ya tenemos el rango de números aleatorios que necesitábamos, 10-100.

Redondear el número de decimales utilizando el método Math.round()

En el segundo tema ya te expliqué dos formas de redondear la cantidad de decimales y evitar así una larga cola de números. Como ya te prometí en ese momento, aquí te traigo otra manera de conseguirlo. A mi juicio, la más sencilla, aunque consideraba importante añadir las otras dos en este libro. Como ya has visto en el título de este apartado, vamos a utilizar el método Math.round(), el mismo que llevamos todo el tema utilizando para generar números aleatorios. Te lo explico directamente con ejemplos.

Redondeando a dos decimales:

```
public class Main {
   public static void main(String[] args) {
      double numero = 3.14159;
      double resultado = Math.round(numero * 100.0) / 100.0;
      System.out.println("Número con dos decimales: " + resultado);  // Output: 3.14
   }
}
```

Redondeando a tres decimales:

```
public class Main {
   public static void main(String[] args) {
      double num = Math.random() * 10;
      double resultado = Math.round(num * 1000.0) / 1000.0;
       System.out.println("Número aleatorio redondeado a tres decimales: " + resultado);
   }
}
```

Resumen de lo aprendido

Los números aleatorios son esenciales en la programación en general. Se utilizan, entre otras cosas, para simular eventos aleatorios en juegos y simulaciones, generar datos para pruebas de software u optimizar algoritmos. Además, son cruciales en la seguridad informática, sirven para crear claves criptográficas y en el análisis de datos para modelar distribuciones estadísticas. Estos usos mejoran la seguridad y el rendimiento de los sistemas informáticos.

En Java, existen varias formas de generar números aleatorios, pero nosotros nos vamos a centrar en una en concreto. Los crearemos utilizando la clase Math y su método random(). Este método produce números entre 0.0 (inclusive) y 1.0 (exclusivo). Con ese rango numérico estamos un poco limitados, pero eso no es ningún problema ya que con ingenio podemos acceder a otros valores.

Para generar un número aleatorio entre dos números decimales, se usa la fórmula Math.random() * (max - min) + min.

Para obtener un número entero aleatorio en un rango específico, se convierte el resultado a un entero y se ajusta el rango añadiendo 1, garantizando así la inclusión del valor superior en el rango.

Ejercicios del tema

Ejercicio 1. Crea un programa que calcule la tirada de veinte dados y la suma de los resultados.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int suma =0;

//Lo primero que haremos es crear un bucle para los 20 dados.
//De esta forma no tendremos que copiar y pegar la misma línea 20 veces.

      for (int i=0; i<20;i++) {

//Utilizamos Math.random para generar números aleatorios.
//En este caso queremos que el máximo sea seis. Por eso la multiplicación.
//Al multiplicarlo por 6 nos salen números entre 0 y 5.99. Así que le sumamos 1.
//Ahora tenemos números entre 1 y 6.99 pero debemos transformarlos en números enteros.
//Esto se hace convirtiendo toda la expresión a entero con (int). Tal como se ve abajo.

         int a =(int)(Math.random()*6 + 1);
         suma = suma+a;
      }

      System.out.println("La suma de los 20 dados es " + suma);

   }
}
```

Ejercicio 2. Programa en Java que muestra aleatoriamente 100 cartas de una baraja española.

Solución:

```
public class Main {
   public static void main(String[] args) {

      String palo = "";
      String figuras = "";

     //Creamos un for para que se muestren 100 cartas

      for (int i = 0; i <= 100; i++) {

     //Creamos un número random del 1 al 4 para que se generen los palos.

         int a = (int) (Math.random() * 4 + 1);

     //Dependiendo de que valor salga le asignamos un palo

         if (a == 1) {
            palo = " oros ";
         }
         if (a == 2) {
            palo = " espadas ";
         }
         if (a == 3) {
            palo = " copas ";
         }
         if (a == 4) {
            palo = " bastos ";
         }
```

```
 //Ahora creamos números aleatorios de entre 1 y 10.
//Del 1 al 7 corresponderán a cartas normales
// 8 a sota, 9 a caballo y 10 a rey.


        int b = (int) (Math.random() * 10 + 1);

        //Damos valor a la variable figuras de la siguiente forma.

        if (b==8) {
           figuras= "Sota";
        }
        if (b==9) {
           figuras= "Caballo";
        }
        if (b==10) {
           figuras= "Rey";
        }

 //Mostramos los resultados en la pantalla ayudándonos de condicionales.

        if (b==1) {
           System.out.println("As de" + palo);
        }
        if (b>=2&&b<=7) {
           System.out.println(b + " de" + palo);
        }
        if (b>=8&&b<=10) {
           System.out.println(figuras + " de" + palo);
        }

      }
   }
}
```

Ejercicio 3. Programa en Java que muestra 20 números aleatorios entre 0 y 100 en la misma línea.

Solución:

```
public class Main {
   public static void main(String[] args) {

   for (int i=0;i<=20;i++){

      int a = (int)(Math.random()*100);

      System.out.print(" " + a );

    }
   }
}
```

Comprender el Rango: Para generar un número aleatorio entre dos valores específicos, usa la fórmula Math.random() * (max - min) + min. Asegúrate de entender que Math.random() genera un número entre 0.0 y 1.0, por lo que multiplicar por (max - min) y luego sumar min ajusta correctamente el rango.

Ejercicio 4. Programa en Java que muestra 10 números aleatorios entre 100 y 199 y luego muestre cuál fue el mayor y el menor.

Solución:

```
public class Main {
   public static void main(String[] args) {

//Creamos las variables que ves a continuación.
//El número menor que podemos encontrarnos es 0.
//El número mayor es 199.
// La idea es que cuando salga un número mayor o menor reemplace a los
que tenemos.

      int mayor = 0;
      int menor= 199;



   for (int i=0;i<=10;i++){

      int a = (int)(Math.random()*99+100);


//Si el número que ha salido es mayor que el actual mayor
//Se convierte en el nuevo mayor

      if (a>mayor) {
         mayor = a;
      }

//Si el número que ha salido es menor que el actual menor
//Se convierte en el nuevo menor
```

```
            if (a<menor) {
            menor = a;
          }
        }

          System.out.println("El numero mayor es: " + mayor);
          System.out.println("El numero menor es: " + menor);

      }
    }
```

Ejercicio 5. Crea un programa que saque números al azar hasta que salga 24. Después, hará la cuenta de todos los que han salido.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a = 0;
      int suma=0;

      while (a!=24) {
         a=(int)(Math.random()*100);
         suma++;
      }

      System.out.println("La suma total de números es: " + suma);
   }
}
```

Ejercicio 6. Crea un programa que genere aleatoriamente 20 notas y las clasifique en suficiente, aprobado, notable y sobresaliente.

Solución:

```
public class Main {
   public static void main(String[] args) {


 String nota = "";

   for (int i=0;i<=20;i++) {

      double a =(Math.random() * 10);
      String resultado = String.format("%.2f", a);

      if (a<5) {
         nota = "Suspenso";
      }
      if (a>=5&&a<7) {
         nota = "suficiente";
      }
      if (a>=7&&a<9) {
         nota = "Notable";
      }
      if (a>=9) {
         nota = "Sobresaliente";
      }

      System.out.println(resultado + " ----> " + nota);

      }
   }
}
```

Ejercicio 7. Crea un programa que muestra la tirada de dos dados hasta que salga el mismo resultado en ambos.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a =0;
      int b =1;


   while (a!=b) {

      a =(int)(Math.random() * 6+1);
      b =(int)(Math.random() * 6+1);

      System.out.println("La tirada de los dados es " + a + " " + b);

      }
   }
}
```

Uso de Enteros: Para obtener números enteros en un rango específico, conviértelo a entero con (int). Además, ajusta el rango añadiendo 1 a la fórmula: (int)(Math.random() * (max - min + 1) + min). Esto incluye tanto el valor mínimo como el máximo en el rango de posibles resultados.

Ejercicio 8. Programa en Java en el que tenemos cinco intentos para adivinar un número creado al azar.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);
      int intentos = 5;

      //Creamos una contraseña de cuatro dígitos. Entre 1000 y 9999.

      int contrasena = (int)(Math.random()*9000+1000);

      System.out.println("La contrasena es " + contrasena);

      while (intentos >0) {
      intentos--;
      System.out.println("Introduce una contraseña ");
      int prueba = ejercicio.nextInt();

      if (prueba==contrasena) {
         System.out.println("Has acertado");
         break;
        }


      if (intentos ==0) {
         System.out.println("No hay más intentos");
         }
      }
   }
}
```

Ejercicio 9. Creamos un programa que simule una quiniela de 9 partidos. Se mostrarán tres columnas donde aparecerá marcado 1, X o 2.

Solución:

```
public class Main {
   public static void main(String[] args) {

      System.out.println("El resultado de la quiniela es el siguiente:");
      System.out.println(" ");
      for (int i =1;i<=9;i++) {
         int resultado = (int)(Math.random()*3+1);
         if (resultado==1) {
            System.out.println("Partido " + i + " = " + "   1   |      |");
         }
         if (resultado==2) {
            System.out.println("Partido " + i + " = " + "      |   X   |");
         }
         if (resultado==3) {
            System.out.println("Partido " + i + " = " + "      |   X   |   2");
         }
      }

   }
}
```

Evitar Valores No Deseados: Si necesitas excluir ciertos valores del rango, usa condicionales para volver a generar el número aleatorio en caso de obtener un valor no deseado. Por ejemplo, si generas un número aleatorio y necesitas excluir el 100, puedes usar un bucle while para repetir la generación hasta obtener un valor válido.

Ejercicio 10. Ejercicio en Java de números aleatorios, pero en este caso utilizando una variable char. Crea un programa que muestre cinco veces los siguientes símbolos de forma aleatoria: &^%$.*.

Solución:

```
public class Main {
   public static void main(String[] args) {

      char a= ' ';
      for (int i=1;i<=5;i++) {

         int b = (int)(Math.random()*5+1);

         if (b==1) {
            a = '&';
         }
         if (b==2) {
            a = '*';
         }
         if (b==3) {
            a = '^';
         }
         if (b==4) {
            a = '%';
         }

         if (b==5) {
            a = '$';
         }

         System.out.println(a);
      }
   }
}
```

Capítulo 7.

La visita del primo de Rich. Arrays

Hoy he recibido una visita inesperada. Aún estoy un poco aturdido. Alrededor de las 12 de la mañana, un carcelero vino a buscarme y me llevó a la sala de visitas. Pensé que sería mi madre o quizás el abogado, pero era el primo de Chani. Yo nunca lo había visto, así que no sabía que era él hasta que se presentó.

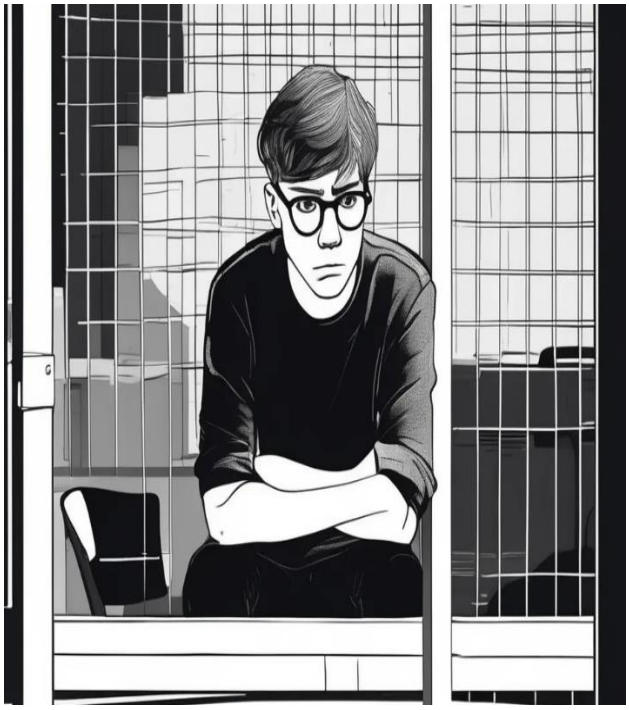
Estuve unos cuantos segundos en silencio. Sentía cómo me iba enfadando poco a poco, hasta que empecé a maldecir y a trabarme, ya que quería expresar muchas cosas al mismo tiempo. En cierto momento, uno de los guardias se acercó y me pidió que me calmara. La verdad es que fue un buen desahogo. Después de eso, me sentía bastante más tranquilo y dispuesto a ver qué me tenía que decir.

Aparte de enfadado, tenía mucha curiosidad. Todas las respuestas que llevaba anhelando durante semanas estaban justo enfrente de mí. Quería saber exactamente en qué líos estaba metido Chani, por qué lo había hecho y, sobre todo, por qué me había inculpado a mí. Quizás había sido todo un error. Puede que estuviese arrepentido y quisiese exculparme. Se me pasaban cantidad de preguntas por la cabeza y no sabía por cuál empezar.

Justo iba a decir algo cuando me interrumpió. Me dijo que sabía lo que pensaba, que sabía que yo creía que Chani era el culpable, pero que no era así. Aunque todas las pruebas apuntaban hacia él, en realidad era inocente.

Me parecía bastante difícil de creer, así que le pregunté que cómo estaba tan seguro. Entonces me dijo que Chani le había enviado para contarme la verdadera historia. Otra vez volví a enfurecer. Le dije que, si sabía dónde estaba Chani, debía decirlo o sería culpable por encubrirle. De nuevo, el guardia vino a llamarme la atención.

Una vez que me calmé, el primo procedió a contarme la historia. Según él, Chani no tenía nada que ver. No le quedó más remedio que huir cuando descubrió el plan de los verdaderos culpables: Rich y Fernando.



Parece ser que el día antes de la fiesta, Chani llegó a casa y se fue directo a su habitación. Fernando y Rich estaban en el salón y no se percataron de su presencia. Sin miedo a ser descubiertos, comenzaron a comentar su plan. En ese momento estaban consiguiendo el software ilegal que utilizarían después para extorsionar a empresas.

Chani recibió una llamada y fue descubierto. Los tres se sentaron en el salón, y fue en ese momento cuando se dio cuenta de que estaban utilizando mi ordenador. Al verse acorralados, no tuvieron más remedio que ofrecerle participar en el delito. Le explicaron el plan con todo detalle, incluida la parte en la que, si algo malo pasaba, me culparían a mí.

Pese a la insistencia, Chani se negó. Fue entonces cuando le amenazaron. Se les ocurrió una forma de incriminarle si no colaboraba. Le recomendaron que lo pensara durante algunas horas y que al día siguiente querían una respuesta definitiva.

Durante la noche obtuvieron esa respuesta. Pero no fue lo que esperaban. Así que la cosa se puso aún más tensa y le amenazaron físicamente. Le obligaron a marcharse cuanto antes y, además, le coaccionaron para que no dijera nada.

La historia tenía sentido, pero no me la creía. Rich es mi amigo. Nunca me haría una cosa así. Después de unos segundos de silencio, me miró fijamente y me preguntó si le creía. Le respondí que no, y entonces sacó su teléfono. Me enseñó una grabación de audio de la noche de la fiesta en la que se escuchaba a Rich diciéndole que se fuera esa misma noche y no volviera más.

Chani trató de buscar pruebas en su contra, pero en la fiesta había mucho ruido y Rich es muy astuto. De esa conversación solo se puede distinguir la parte en la que le dice que se marche de la ciudad esa misma noche.

Parece que Chani tiene un plan para desenmascarar a Rich, pero es posible que necesite mi ayuda. No sé cómo puedo ayudar desde aquí encerrado. Quizás en alguna de sus visitas pueda obtener una confesión o al menos algo de información. Lo malo de estar incomunicado no solo es la falta de información, sino que tampoco soy muy útil.

Después de la breve conversación, volví a mi celda. Aún estoy procesando la información. Rich ha venido a visitarme varias veces y siempre me ha parecido muy preocupado. No lo esperaba, pero tiene sentido. Ahora solo espero que venga a verme de nuevo, así podré aclarar las cosas cara a cara.

Para intentar pensar en otra cosa, le he pedido a Bud que sigamos con las clases de Java. Parece especialmente motivado. Me ha dicho que, una vez que aprenda lo que son los arrays, podrá explicarme exactamente cómo ayuda a los funcionarios con el software de la cárcel. De momento, no quiero contarle nada sobre mi situación personal. Sé que le preocuparía. Así que, de momento, pongo buena cara e intento aprender algo.

Arreglos o arrays

En Java, los arrays son una forma básica de almacenar **varios valores** del mismo tipo en **una sola variable**. A diferencia de las variables normales que solo pueden contener un valor, un array puede almacenar varios valores, llamados elementos, que se organizan en un orden específico y se acceden usando números llamados índices.

Un array se crea indicando qué tipo de datos albergará y cuántos elementos almacenará. Por ejemplo, un array int de enteros con un tamaño de 5 puede guardar cinco números enteros. Los arrays en Java empiezan a contar desde 0, por lo que el primer elemento está en la posición 0.

Importancia de los arreglos en programación

- Permiten organizar datos de manera estructurada, lo que facilita su acceso y manipulación.

- Con arrays, podemos almacenar grandes cantidades de datos relacionados de manera eficiente y acceder a ellos a través de índices, en lugar de gestionar muchas variables independientes.

- Los elementos de un array se pueden acceder de manera rápida usando su índice, lo que es crucial para aplicaciones que necesitan operaciones rápidas y frecuentes de lectura o modificación de datos.

- Además, los arrays permiten realizar operaciones comunes como iteración, búsqueda, clasificación y manipulación de datos de manera sencilla. Muchos algoritmos y estructuras de datos se basan en arrays por su simplicidad y eficiencia, como los algoritmos de clasificación y búsqueda.

- En aplicaciones reales, los arrays se utilizan ampliamente, desde el desarrollo de juegos hasta el procesamiento de datos y aplicaciones científicas, donde son esenciales para gestionar grandes conjuntos de datos y realizar cálculos numéricos complejos.

Declaración de Arrays

La declaración de un array en Java se realiza especificando el tipo de datos de los elementos que contendrá, seguido de corchetes [] y el nombre del array.

Paso 1: declarar el array

tipo[] nombreArray;

Ejemplos:

int[] numeros; // Declaración de un array de enteros

String[] nombres; // Declaración de un array de cadenas

double[] temperaturas; // Declaración de un array de dobles

Paso 2. Crear el array

nombreArray = new tipo[tamaño];

Ejemplos:

numeros = new int[5]; // Crea un array de enteros con 5 elementos

nombres = new String[3]; // Crea un array de cadenas con 3 elementos

temperaturas = new double[7]; // Crea un array de dobles con 7 elementos

Ahora bien, casi nunca vas a utilizar esta estructura. Es mucho más simple, rápido y eficaz si lo hacemos todo en la misma línea. Además de que para ti va a ser bastante más fácil de recordar.

Declaración y Creación en una Sola Línea

tipo[] nombreArray = new tipo[tamaño];

Ejemplos:

int[] numeros = new int[5]; // Declaración y creación de un array de enteros con 5 elementos.

String[] nombres = new String[3]; // Declaración y creación de un array de cadenas con 3 elementos.

double[] temperaturas = new double[7]; // Declaración y creación de un array de dobles con 7 elementos.

Asignando valor a los Arrays

int[] numeros = new int[5];

```
numeros[0] = 10;
numeros[1] = 20;
numeros[2] = 30;
numeros[3] = 40;
numeros[4] = 50;
```

Ahora cada elemento de nuestro array tiene un valor. Por ejemplo, si queremos mostrar por pantalla la posición número 5 lo haremos de la siguiente forma:

```
System.out.println(numeros[4]);
```

String[] nombres = new String[3];

```
nombres[0] = "Lucía";
nombres[1] = "Pedro";
nombres[2] = "Luís";
```

No es necesaria más explicación teórica. Este tema es bastante sencillo de comprender simplemente viendo los ejercicios. Al menos de momento. Solo he mostrado ejemplos de números naturales (Variables tipo int) y texto (Variables tipo String). Pero está claro que, por ejemplo, podemos usar números con decimales si utilizamos un double o caracteres con un char. Para este último en vez de entrecomillar utilizando " " lo haríamos con ' '.

Da igual el tipo de variable que sea, siempre se sigue la misma estructura.

Asignando valor a los arrays en una sola línea

En algunas ocasiones vas a encontrarte con corchetes en los arrays. También sirven para asignarles valor. Esta es la estructura que se utiliza con este propósito:

- tipo[] nombreArray = {valor1, valor2, valor3, ...};

Como no, vamos a verlo con un ejemplo. Empezamos con un array tipo int con 5 elementos. Les asignamos valor y mostramos el elemento 5. Recuerda que el array siempre empieza en la posición 0. Así que en la posición 4 se encuentra nuestro quinto elemento.

```
int[] numeros = {10, 20, 30, 40, 50};
System.out.println(numeros[4]);
```

Ahora seguimos con nuestro ejemplo de texto. Tenemos tres nombres y declaráramos y asignamos valor a un array en una sola línea.

- String[] nombres = {"Lucía", "Pedro", "Luís"};

Como se puede apreciar, a simple vista esta forma de crear los arrays y asignarles valor es mucho más simple. El código queda más ordenado y elegante. Aunque no todo son ventajas. Al no tener indicada la posición de cada elemento, a veces podemos perdernos y liarnos un poco. Como comentaba antes, el array siempre empieza en la posición cero y va subiendo hasta el máximo de elementos que tenemos. Tener la posición al lado de cada valor resulta muy conveniente y, con una simple ojeada, visualizamos todo el esquema de nuestro array en nuestra cabeza.

Recorrer Arrays Utilizando Bucles en Java

Recorrer un array significa acceder a cada uno de sus elementos uno por uno. Los bucles son herramientas esenciales para esta tarea, ya que permiten iterar sobre los elementos de un array de manera sencilla y eficiente. Lo más sencillo para esta tarea es utilizar un bucle for.

Ejemplo: Recorrer un Array de Enteros usando un Bucle for

```
public class Main {
   public static void main(String[] args) {
      int[] numeros = {1, 2, 3, 4, 5};
      // Sabemos que el tamaño del array es 5
      for (int i = 0; i < 5; i++) {
         System.out.println(numeros[i]);
      }
   }
}
```

Ejemplo: Recorrer un array de cadenas (Strings)

```
public class Main {
   public static void main(String[] args) {
      String[] colores = {"Rojo", "Verde", "Azul", "Amarillo"};
      // Sabemos que el tamaño del array es 4
      for (int i = 0; i < 4; i++) {
         System.out.println(colores[i]);
      }
   }
}
```

Ejemplo: Recorrer un array de caracteres (char)

```
public class Main {
   public static void main(String[] args) {
      char[] letras = {'A', 'B', 'C'};

      // Sabemos que el tamaño del array es 3
      for (int i = 0; i < 3; i++) {
         System.out.println(letras[i]);
      }
   }
}
```

Fíjate que en todos los ejemplos la primera posición del bucle es 0 ya que los arrays siempre empiezan desde 0.

Resumen de lo aprendido

En Java, un array es una estructura que permite almacenar múltiples valores del mismo tipo en una sola variable. A diferencia de una variable normal que solo puede contener un valor, un array puede almacenar varios valores, llamados elementos. Estos elementos se organizan en un orden específico y se acceden mediante índices numéricos, empezando desde 0.

Declaración y Creación de Arrays

- Declaración: Consiste en especificar el tipo de datos que contendrá el array y darle un nombre.

- Creación: Implica asignar memoria para el array, definiendo cuántos elementos podrá almacenar.

- Declaración y Creación en una Sola Línea: Para simplificar y hacer el código más eficiente, se puede declarar y crear un array en una sola línea.

Asignación de Valores a un Array

Los valores se asignan a los elementos del array utilizando su índice. Esto permite acceder y modificar cada elemento del array directamente a través de su índice.

Asignación de Valores en una sola Línea

Es posible asignar valores a un array directamente al momento de declararlo y crearlo, lo que resulta en un código más limpio y fácil de leer.

Recorrer Arrays Usando Bucles

Para acceder a cada elemento de un array, se utilizan bucles como el bucle for. Este tipo de bucle es eficiente para iterar sobre cada elemento del array y realizar diversas operaciones con ellos.

Ejercicios del tema.

Ejercicio 1. Crea un array que contenga los números del 0 al 9. Después haz que en la pantalla se muestren en orden inverso.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a [] = new int [10];

      for (int i=0;i<=9;i++) {

         a[i] = i;

      }

      for (int i =9; i>=0;i--) {
         System.out.println(a[i]);
      }
   }
}
```

Inicializa los Arrays Adecuadamente: Antes de usar un array, asegúrate de inicializarlo correctamente con el tamaño adecuado y los valores necesarios. Esto evita errores de índice fuera de rango y garantiza que el array tenga los datos que necesitas.

Ejercicio 2. Introduce 10 números por teclado y después muéstralos de forma inversa, utilizando un array.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      int a [] = new int [10];

      for (int i=0;i<=9;i++) {

         System.out.println("Introduce el número " + (i+1));
         a[i] = ejercicio.nextInt();

      }

      System.out.println("El orden inverso de los números introducidos es:");
      for (int i =9; i>=0;i--) {

         System.out.println(a[i]);
      }

   }
}
```

Ejercicio 3. Genera 50 números aleatorios entre 0 y 100 y después muestra el cuadrado y el cubo de los mismos, utilizando un array.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a [] = new int [50];

      for (int i=0;i<50;i++) {

         a[i] = (int)(Math.random()*100);
         int cuadrado = a[i]* a[i];
         int cubo = a[i] * a[i] * a[i];
         System.out.println("Número " + a[i] + ", Cuadrado " + cuadrado + ",
         Cubo " + cubo);

      }
   }
}
```

Verifica los Límites del Array: Siempre verifica que los índices utilizados estén dentro de los límites del array. Acceder a índices fuera del rango permitido puede causar errores de ejecución.

Ejercicio 4. Crea un programa que muestre 10 números aleatorios de 0 a 100. Al lado de cada uno, muestra si es par o impar.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a[] = new int [10];

      for (int i=0;i<10;i++) {

         a[i] = (int)(Math.random()*100);


         if (a[i]%2==0) {
            System.out.println(a[i] + " es par");
         }
         if (a[i]%2!=0) {
            System.out.println(a[i] + " es impar");
         }

      }
   }
}
```

Usa Bucles para Recorrer Arrays: Utiliza bucles for para recorrer los elementos del array de manera eficiente. Los bucles facilitan la manipulación y el acceso a los elementos, permitiendo realizar operaciones como sumas, búsquedas y actualizaciones de manera rápida.

Ejercicio 5. Crea un programa que muestre 10 números introducidos por teclado. Al lado de cada uno, muestra si es par o impar.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      int a[] = new int [10];

      for (int i=0;i<=9;i++) {

         System.out.println("Inserta un número");

         a[i] = ejercicio.nextInt();

      }

      for (int i=0;i<=9;i++) {

         if (a[i]%2==0) {
            System.out.println(a[i] + " es par");
         }
         if (a[i]%2!=0) {
            System.out.println(a[i] + " es impar");
         }

      }
   }
}
```

Ejercicio 6. Ejercicio en el que introducimos 10 números por teclado y pasamos el último al primer lugar.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      int a[] = new int [10];

      // Creamos un for para introducir los diez datos.

      for (int i=0;i<10;i++) {

         System.out.println("Introduce un número");
         a[i] = ejercicio.nextInt();
      }

      //Ahora mostraremos el último de los datos como el primero

      System.out.println(" ");
      System.out.println("La lista modificada es ");
      System.out.println(" ");

      System.out.println(a[9]);
      for (int i=0;i<9;i++) {
         System.out.println(a[i]);
      }
   }
}
```

Ejercicio 7. Programa que crea 50 números aleatorios entre 0 y 100. Creamos un buscador que busque un número en concreto y lo sustituya por otro número que también indicaremos.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      System.out.println("Introduce el número que quieres reemplazar");
      int original = ejercicio.nextInt();

      System.out.println("Introduce el número por el que lo quieres reemplazar");
      int reemplazo = ejercicio.nextInt();

      int a[] = new int [50];

      // Creamos un for para generar 50 datos random

      for (int i=0;i<50;i++) {
         a[i] =(int)(Math.random()*100);
         if (a[i]==original) {
            a[i]=reemplazo;
         }
      }

      System.out.println("Aquí tienes la lista de números modificada");

      for (int i=0;i<50;i++) {

         System.out.println(a[i]);

      }
   }
}
```

Ejercicio 8. Crea un programa en Java que clasifique 10 números mayores que 0, introducidos por teclado, en pares o impares, utilizando un array.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner ejercicio = new Scanner(System.in);

      //Creamos un array donde se almacenarán los 10 datos
      int a[] = new int [10];
      //Creamos un array donde se almacenarán los datos pares
      int pares[] = new int [10];
      //Creamos un array donde se almacenarán los datos impares
      int impares[] = new int [10];

       // Contadores para los números pares e impares
       int numPares = 0;
       int numImpares = 0;

      //Creamos un bucle para introducir 10 datos
      //Se almacenarán en nuestro int a
      //Dependiendo de si son pares o impares se almacenarán también
        en pares/impares

      for (int i=0;i<10;i++) {

         System.out.println("Introduce un número");
         a[i] = ejercicio.nextInt();
```

```
            if (a[i] % 2 == 0) {
                pares[numPares++] = a[i];
            } else {
                impares[numImpares++] = a[i];
            }
        }

        System.out.println(" ");
        System.out.println("Números pares");


        // Mostramos los pares
        // En las posiciones que no hay datos para el array,
        //  este valor es por defecto 0.
        // Por eso, utilizamos un if para sólo mostrar números mayores que 0

        for (int i=0;i< numPares;i++) {
            if (pares[i]>0) {
                System.out.print(pares[i] + " ");
            }
        }
        System.out.println(" ");
        System.out.println(" ");
        System.out.println("Números impares");
        for (int i=0;i< numImpares;i++) {
            if (impares[i]>0) {
                System.out.print(impares[i] + " ");
            }
        }
    }
}
```

Ejercicio 9. Programa en Java que muestra 50 números del 1 al 100. Después, muestra cuál es el máximo, utilizando un array.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int a[] = new int [50];
      int mayor = 0;

      for (int i=0;i<50;i++) {

         a[i] = (int)(Math.random()*100 +1);

         if (a[i]>mayor) {
            mayor=a[i];
         }
      }

      System.out.println("El número mayor es " + mayor);

   }
}
```

Ejercicio 10. Ejercicios de arreglos con variables de tipo char y String. Crea un array que contenga estas palabras: "Hola", "amigos", "que", "tal". Después, otro array que contenga los siguientes símbolos: ^, &, *, %. Luego, muestra todo en pantalla.

Solución:

```
public class Main {
   public static void main(String[] args) {

      String palabras []= new String [4];
      char simbolos [] = new char [4];


      palabras [0] = "hola";
      palabras [1] = "amigos";
      palabras [2] = "que";
      palabras [3] = "tal";

      simbolos [0] = '*';
      simbolos [1] = '&';
      simbolos [2] = '^';
      simbolos [3] = '%';

      for (int i=0;i<4;i++) {

         System.out.println(palabras[i]);
      }
      for (int i=0;i<4;i++) {
         System.out.println(simbolos[i]);
      }
   }
}
```

Capítulo 8.

El trabajo de bud. Matrices 2D

Los días se me hacen eternos. Estoy esperando ansioso la visita de Rich, pero está tardando más de lo habitual. Hace diez días que tuve la conversación con el primo de Chani y aún no ha venido a verme. Bud me mantiene entretenido; recientemente me ha presentado a su amigo Pedro. Él también ayuda a los funcionarios de la prisión con el software de seguridad. Es una persona muy introvertida. Apenas habla con nadie y suele pasar gran parte del día con el ordenador. Pienso que, con la personalidad que tiene, es una suerte para él que le dejen pasar tanto tiempo pegado al ordenador.

Por otro lado, Bud es más sociable. Se lo pasa bien conversando con cualquiera, pero, aun así, tener una actividad que te mantenga ocupado es lo mejor que te puede pasar mientras estás encerrado aquí.

Si quiero ser su nuevo aprendiz, debo saber exactamente lo que hacen. En estos momentos, no tengo ni idea. Bud cree que estoy preparado para realizar pequeñas tareas y que, poco a poco, me irá dando más responsabilidades.

Sin previo aviso, me informan de que tengo una visita. Espero que sea Rich, pero no lo es. De nuevo es el primo de Chani. Esta vez me comenta que tienen un plan para demostrar mi inocencia. Quieren tender una trampa a Rich. El problema es que, para llevarlo a cabo, necesito estar fuera de la cárcel. La manera más sencilla sería fingir algún tipo de dolencia, como un problema dental, y que así me trasladen a un médico externo.

El problema sería burlar a los policías que supuestamente me escoltarían hasta allí. Si lo consiguiera, supongo que el siguiente paso del plan consistiría en quedar con Rich y tratar de sacarle una confesión.

No lo tengo nada claro. Si salgo y me escapo de los funcionarios que me escoltan, me voy a meter en un lío muy gordo. Me condenarían por otro delito y, además, no me dejarían participar en ningún tipo de actividad dentro de la cárcel. Tendría que olvidarme de ayudar a Bud y Pedro con el sistema de seguridad. Pero, por otro lado, si no hago nada, lo más seguro es que tenga que pasar aquí una larga temporada.

Creo que es una locura, pero si tengo que salir de la cárcel, necesito pedir consejo a alguien que haya vivido aquí ya varios años. Así que creo que la mejor opción es comentárselo a Bud. Quizás él tenga ideas para poder salir durante algunas horas sin meterme en líos.

Pensé que me diría que estaba loco y que me olvidase de la idea. Pero nada más lejos, me dijo que podría ayudarme. Sin embargo, antes quiere que comprenda cuál es su labor en la cárcel. Para ello, dice que tengo que aprender lo que son las matrices 2D. Me va a explicar su funcionamiento y después me dará 10 ejemplos. Si consigo apañármelas con ellos, me ayudará a salir de la prisión.

Importancia de las matrices bidimensionales en Java

Las matrices bidimensionales en Java nos permiten representar datos en formato de tabla, distribuidos en filas y columnas. También conocidas como arreglos 2D, se puede decir que estas matrices son una evolución de los arrays que vimos en el tema anterior. Los arrays funcionan solo en una dimensión. Las matrices permiten almacenar información en forma de tabla. Esta disposición las hace útiles en muchas aplicaciones.

Al utilizar matrices 2D, es posible estructurar datos de manera más organizada. Además, facilitan el acceso a los elementos a través de índices, lo que agiliza tareas como la manipulación y búsqueda de datos.

Las matrices bidimensionales también son esenciales para implementar algoritmos que requieren una estructura de tabla, como algunos algoritmos de búsqueda y ordenación. Gracias a su versatilidad, se emplean en diversos campos, desde la creación de videojuegos y gráficos hasta simulaciones científicas y análisis de datos.

Declaración y Creación de Matrices 2D

Para declarar una matriz 2D, debes especificar el tipo de datos que almacenará y su nombre. Básicamente así:

```
int[][] miMatriz;
```

Después de declarar la matriz, necesitas crearla, es decir, asignar memoria para ella. Para esto, debes especificar cuántas filas y columnas tendrá. La sintaxis es:

```
nombreMatriz = new tipoDato[numFilas][numColumnas];
```

Por ejemplo, para crear una matriz de enteros con 3 filas y 4 columnas:

```
miMatriz = new int[3][4];
```

Igual que con los arrays, he empezado explicando paso a paso cómo se declara, después se crea y, por último, se le asigna valor. Creo que es la forma más gráfica de entender cómo funciona la sintaxis de Java. Aunque, en la práctica, lo más sencillo es hacer todo a la vez en una misma línea. Ahorramos código y todo queda mucho más ordenado. Vamos a ver cómo se hace.

Declaración, Creación y asignación de valor en una Sola Línea

```
int[][] miMatriz = new int[3][4];
```

Creo que el ejemplo se explica por sí solo. En la primera parte de la línea, antes del signo igual, declaramos la matriz y la nombramos de la manera que mejor nos parezca. Después del igual, creamos la matriz utilizando new más el tipo de dato y, entre corchetes, asignamos el número de filas y columnas que necesitamos.

Acceso a Elementos de una Matriz 2D

Para acceder a un elemento específico en una matriz 2D, debes utilizar dos índices: uno para la fila y otro para la columna. La sintaxis básica es:

```
nombreMatriz[fila][columna]
```

Imagínate que tienes una matriz llamada “miMatriz” de tamaño 3x3 y quieres acceder al elemento en la segunda fila y tercera columna, Probablemente lo harías así:

```
int valor = miMatriz[2][3];
```

Pero te equivocas, **recuerda que al igual que pasaba con los arrays, no empezamos desde 1. Sino desde 0**. Por lo tanto, para acceder al valor que está en la segunda fila y tercera columna lo haremos así:

```
int valor = miMatriz[1][2];
```

Lo podemos ver con otro ejemplo, tenemos dada la siguiente matriz:

```
int[][] miMatriz = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};

int valor = miMatriz[1][1]; // 5
```

Si queremos cambiar el valor 6 a 14 lo haríamos de la siguiente forma:

```
int miMatriz[1][2] = 14;
```

Inicialización de Matrices 2D

- Puedes inicializar una matriz directamente cuando la declaras, especificando todos sus elementos de antemano:

```
int[][] miMatriz = {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9}
};
```

- También puedes declarar una matriz primero y asignar valores a sus elementos después:

```
int[][] miMatriz = new int[3][3];
miMatriz[0][0] = 1;
miMatriz[0][1] = 2;
miMatriz[0][2] = 3;
miMatriz[1][0] = 4;
miMatriz[1][1] = 5;
miMatriz[1][2] = 6;
miMatriz[2][0] = 7;
miMatriz[2][1] = 8;
miMatriz[2][2] = 9;
```

Recorrer Matrices 2D Usando Bucles

Para recorrer los arrays utilizábamos un bucle. En este caso, uno no va a ser suficiente ya que tenemos elementos colocados en filas y columnas. Por lo tanto, vamos a tener que utilizar dos bucles anidados.

Esta sería la estructura que seguiríamos:

```
for (int i = 0; i < miMatriz.length; i++) { // Recorre las filas
   for (int j = 0; j < miMatriz[i].length; j++) { // Recorre las columnas
      System.out.println("Elemento en [" + i + "][" + j + "]: " + miMatriz[i][j]);
   }
}
```

Si empezamos a analizar el código, en la primera línea nos encontramos con miMatriz.length. Eso es básicamente el número de filas que tiene nuestra matriz. Podemos dejarlo escrito así o simplemente escribir el número de filas.

En la siguiente línea, nos encontramos con el segundo bucle. Por lo general, cuando creamos un bucle for, llamamos a la variable "i"; para un segundo bucle, se utiliza la letra "j". Básicamente, lo que hace es, por cada fila de nuestra matriz, recorrer cada columna.

La última línea simplemente nos sirve para imprimir los datos en la pantalla y asegurarnos de que el bucle está siendo recorrido de forma correcta y de que hemos hecho todo bien.

Suma de Elementos en una Matriz

Ahora que sabemos como se recorren las matrices, vamos con un par de operaciones muy básicas que podemos realizar gracias a los bucles. La primera de ella es la suma de los elementos. La estructura es la siguiente:

```
int suma = 0;
for (int i = 0; i < miMatriz.length; i++) {
   for (int j = 0; j < miMatriz[i].length; j++) {
      suma += miMatriz[i][j];
   }
}
System.out.println("La suma de todos los elementos es: " + suma);
```

Búsqueda de Elementos Específicos

```
int[][] miMatriz = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };
    int buscar = 5;
    boolean encontrado = false;

    for (int i = 0; i < miMatriz.length; i++) {
      for (int j = 0; j < miMatriz[i].length; j++) {
        if (miMatriz[i][j] == buscar) {
          System.out.println("Elemento " + buscar + " encontrado en [" + i + "][" + j + "]");
          break;
        }
      }
      if (encontrado) break;
    }

    if (!encontrado) {
      System.out.println("Elemento " + buscar + " no encontrado.");
    }
```

En la primera parte de este código nos encontramos es con la misma matriz que estábamos utilizando en ejercicios anteriores. Es una matriz declarada y con valores ya asignados. Después de ello, tenemos dos variables. La primera es de tipo int y es el elemento que queremos encontrar. La segunda es de tipo boolean, por defecto será false, pero cuando el programa encuentre el elemento deberemos hacer que cambie a true. Podríamos perfectamente escribir nuestro programa sin necesidad de utilizar esta última variable, pero no siempre hay una oportunidad tan buena para utilizar un boolean en nuestros ejemplos.

Hasta aquí, nada que no supiésemos. Ahora recorremos el bucle utilizando la anidación, tal y como hemos hecho en los ejemplos anteriores. Si te fijas, es exactamente el mismo código y la misma estructura. Lo único que cambia es que hemos añadido un if, y como puedes observar, su funcionamiento es muy sencillo. Si encuentra el elemento que buscamos, nos dirá exactamente en qué posición está y se cerrará el bucle gracias a que hemos añadido un break. Si no lo encuentra, seguirá buscando.

Utilizamos el boolean encontrado para hacer que el programa termine del todo si ya hemos encontrado el elemento deseado o que salga un mensaje final diciendo que el elemento no pudo ser encontrado.

Cabe destacar que **cuando se trata de variables tipo boolean y trabajamos en un if**, para escribir la condicional podemos hacerlo de la siguiente forma:

```
If (encontrado == true) {

}
```

Aunque, lo más correcto es hacerlo de la siguiente forma:

```
If (encontrado ) {

}
```

Lo mismo pasa si es false, podemos hacerlo de esta forma:

```
If (encontrado == false) {

}
```

Pero es recomendable hacerlo así:

```
if (!encontrado) {

}
```

Esto se debe a tres motivos:

Claridad y Legibilidad:

- **Es más claro y conciso escribir if (boolean) que if (boolean == true).**

La expresión if (boolean) es más fácil de leer y entender porque está directamente verificando la condición booleana.

- **Evitar Errores Comunes**:

Usar if (boolean == true) introduce la posibilidad de errores tipográficos, como accidentalmente escribir if (boolean = true).

Si escribimos el signo igual con un solo igual, va a traer errores graves en el programa y no funcionará correctamente. Muchas veces detectar este error es bastante difícil así que es mejor ahorrarse la posibilidad de que pueda suceder.

- **Convención de Programación:**

En Java y en muchos otros lenguajes de programación, es una convención y buena práctica utilizar simplemente la variable booleana en una estructura de control, en lugar de compararla explícitamente con true.

Resumen de lo aprendido

Las matrices bidimensionales o arrays 2D son estructuras capaces de organizar datos en tablas que se componen de filas y columnas. Esta forma de almacenamiento resulta especialmente útil para los programadores en numerosos escenarios. Esto se debe a que permiten gestionar la información de un modo ordenado y lógico. En una matriz bidimensional, se puede acceder a los elementos de forma sencilla empleando dos índices: uno para las filas y otro para las columnas.

Muchos algoritmos de búsqueda y clasificación utilizan los arrays 2D, ya que gracias a ellos se puede trabajar con conjuntos de datos estructurados en dos dimensiones. Por ejemplo, pueden usarse para representar tableros de juegos, gráficos e incluso bases de datos sencillas, donde la facilidad de acceso y manipulación de los datos resulta esencial.

Pero la cosa no queda ahí; las matrices bidimensionales se caracterizan por su flexibilidad y facilidad de implementación, lo cual las convierte en idóneas para una amplia gama de aplicaciones. Así que, en resumen, los arrays 2D constituyen una herramienta poderosa y versátil que, en algunos casos, simplifica la programación y proporciona soluciones para problemas complejos.

Ejercicios del tema

Ejercicio 1. Crea una matriz 2D de 5x5 que represente las celdas de una cárcel. Llena todas las posiciones con el valor 0, indicando que todas las celdas están vacías.

Solución:

```
public class Main {
   public static void main(String[] args) {
      int[][] celdas = new int[5][5];

      // Llenar la matriz con 0

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            celdas[i][j] = 0;
         }
      }

      // Imprimir la matriz

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            System.out.print(celdas[i][j] + " ");
         }

         System.out.println();

      }
   }
}
```

Ejercicio 2. Supongamos que las celdas [0][1], [2][2] y [4][3] están ocupadas por prisioneros. Actualiza la matriz del ejercicio anterior para reflejar esto usando el valor 1 para las celdas ocupadas.

Solución:

```
public class Main {
   public static void main(String[] args) {
      int[][] celdas = new int[5][5];

      // Llenar la matriz con 0

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            celdas[i][j] = 0;
         }
      }

      // Marcar las celdas ocupadas

      celdas[0][1] = 1;
      celdas[2][2] = 1;
      celdas[4][3] = 1;

      // Imprimir la matriz

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            System.out.print(celdas[i][j] + " ");
         }

         System.out.println();

      }
   }
}
```

Ejercicio 3. Cuenta cuántas celdas están ocupadas en la matriz creada en el ejercicio 2.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[][] celdas = {
          {0, 1, 0, 0, 0},
          {0, 0, 0, 0, 0},
          {0, 0, 1, 0, 0},
          {0, 0, 0, 0, 0},
          {0, 0, 0, 1, 0}
       };

       int contador = 0;

       // Contar las celdas ocupadas

       for (int i = 0; i < celdas.length; i++) {
          for (int j = 0; j < celdas[i].length; j++) {
             if (celdas[i][j] == 1) {
                contador++;
             }
          }
       }

       System.out.println("Celdas ocupadas: " + contador);
   }
}
```

Ejercicio 4. Libera al prisionero de la celda [2][2] en la matriz del ejercicio 2 y actualiza la matriz.

Solución:

```
public class Main {
   public static void main(String[] args) {

         int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      // Liberar al prisionero

      celdas[2][2] = 0;

      // Imprimir la matriz

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            System.out.print(celdas[i][j] + " ");
         }

         System.out.println();

      }
   }
}
```

Ejercicio 5. Encuentra todas las celdas vacías en la matriz del ejercicio 2 y muestra sus posiciones.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      System.out.println("Celdas vacías:");

      // Encontrar las celdas vacías

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {

            if (celdas[i][j] == 0) {
               System.out.println("[" + i + "][" + j + "]");
            }

         }

      }
   }
}
```

Ejercicio 6. Transfiere un prisionero de la celda [4][3] a la celda [1][4] en la matriz del ejercicio 2

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      // Transferir el prisionero

      celdas[4][3] = 0;
      celdas[1][4] = 1;

      // Imprimir la matriz

      for (int i = 0; i < celdas.length; i++) {

         for (int j = 0; j < celdas[i].length; j++) {

            System.out.print(celdas[i][j] + " ");

            }

         System.out.println();

      }
   }
}
```

Ejercicio 7. Cuenta cuántas celdas vecinas ocupadas tiene la celda [2][2] en la matriz del ejercicio número 2.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      int[][] direcciones = {
         {-1, -1}, {-1, 0}, {-1, 1},
         {0, -1},          {0, 1},
         {1, -1}, {1, 0}, {1, 1}
      };

      int x = 2, y = 2;
      int contador = 0;

      // Contar las celdas vecinas ocupadas

      for (int[] dir : direcciones) {
         int nuevoX = x + dir[0];
         int nuevoY = y + dir[1];
```

```
        if (nuevoX >= 0 && nuevoX < celdas.length && nuevoY >= 0 &&
nuevoY < celdas[0].length) {

            if (celdas[nuevoX][nuevoY] == 1) {
               contador++;

            }
         }
      }

      System.out.println("Celdas vecinas ocupadas de [2][2]: " +
contador);
   }
}
```

Inicialización y Declaración: Siempre inicializa tu matriz correctamente después de declararla. Puedes hacerlo en una sola línea para mantener el código limpio y fácil de leer.

Uso de Bucles Anidados: Utiliza bucles anidados para recorrer y manipular elementos en la matriz. El bucle externo recorre las filas y el bucle interno recorre las columnas. Esto es esencial para realizar operaciones en cada elemento de la matriz.

Ejercicio 8. Encuentra todos los prisioneros en la fila 0 de la matriz del ejercicio 2 y muestra sus posiciones.

Solución:

```
public class Main {
    public static void main(String[] args) {

        int[][] celdas = {
            {0, 1, 0, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 1, 0, 0},
            {0, 0, 0, 0, 0},
            {0, 0, 0, 1, 0}
        };

        System.out.println("Prisioneros en la fila 0:");

        for (int j = 0; j < celdas[0].length; j++) {

            if (celdas[0][j] == 1) {
                System.out.println("[0][" + j + "]");
            }

        }
    }
}
```

Comprueba Límites: Asegúrate de no exceder los límites de la matriz. Siempre verifica que tus índices estén dentro del rango válido.

Ejercicio 9. Rota la matriz del ejercicio 2 90 grados en el sentido de las agujas del reloj.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      int[][] rotada = new int[celdas[0].length][celdas.length];

      for (int i = 0; i < celdas.length; i++) {
         for (int j = 0; j < celdas[i].length; j++) {
            rotada[j][celdas.length - 1 - i] = celdas[i][j];
         }
      }

      // Imprimir la matriz rotada

      for (int i = 0; i < rotada.length; i++) {

         for (int j = 0; j < rotada[i].length; j++) {
            System.out.print(rotada[i][j] + " ");

         }

         System.out.println();

      }
   }
}
```

Ejercicio 10. Verifica si existe un patrón de escape en la matriz del ejercicio 2, definido como tres celdas ocupadas consecutivas en cualquier fila.

Solución:

```
public class Main{
   public static void main(String[] args) {

      int[][] celdas = {
         {0, 1, 0, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 1, 0, 0},
         {0, 0, 0, 0, 0},
         {0, 0, 0, 1, 0}
      };

      boolean escapeEncontrado = false;

      for (int i = 0; i < celdas.length && !escapeEncontrado; i++) {
         int consecutivas = 0;
         for (int j = 0; j < celdas[i].length; j++) {
            if (celdas[i][j] == 1) {
               consecutivas++;
               if (consecutivas == 3) {
                  escapeEncontrado = true;
                  break;
               }
            } else {
               consecutivas = 0;
            }
         }
      }

      if (escapeEncontrado) {
         System.out.println("¡Patrón de escape encontrado!");
      } else {
         System.out.println("No se encontró patrón de escape.");
      }
   }
}
```

Capítulo 9.

Crear un dispositivo de distracción. Con bucles, condicionales y scanner

No me había dado cuenta, pero ya llevo casi un mes y medio entre rejas. Puede que no sea mucho tiempo, pero ya he forjado una gran amistad con Bud. Le comenté mi situación y está dispuesto a ayudarme. Hoy, por fin, va a contarme el plan que me permitirá salir de aquí. No me ha dado ninguna pista, así que no tengo ni idea de cómo lo vamos a hacer.

Bud está bastante contento con mi nivel de Java. He sido un buen estudiante y estoy aprendiendo muy rápido. El problema es que, si escapo de la cárcel, ya puedo olvidarme de continuar con mi progreso. Mi futuro es muy incierto en estos momentos. Solo puedo hacer planes diarios; planificar cosas de una semana para otra es un lujo.

Mi destino es incierto. Sin ir más lejos, hoy he recibido un mensaje de uno de los guardias: me han asignado un trabajo en la lavandería. A las tres de la tarde tengo que presentarme allí, y no sé cuál será mi cometido. Sea lo que sea, seguro que es mejor que estar en la celda sin hacer nada.

He estado solo en la celda todo el día. Es agradable tener un poco de privacidad de vez en cuando. La soledad a veces te ayuda a aclarar las ideas. El tiempo se me ha pasado muy rápido y, sin darme cuenta, ya estoy de camino a la lavandería. Me pregunto cuánta gente trabajará allí. Es posible que pueda hacer nuevos amigos.

Nada más llegar, me di cuenta de que mi plan de socializar se iba totalmente al traste. Quienes me estaban esperando eran, en realidad, Bud y Pedro. Fue entonces cuando me dijeron que tenían un plan para fugarse de la cárcel. Necesitaban a una persona más y, gracias a mis conocimientos de programación, yo podría unirme.

Había un pequeño problema. Ellos querían fugarse, pero yo en realidad no. No quería pasarme el resto de la vida huyendo. Necesito salir solo durante un rato para así desenmascarar a Rich. Bud lo sabía, así que me ofreció una solución. Ellos dos se escaparían, pero yo, después de dos horas, volvería a la cárcel. Debería ser suficiente para conseguir una confesión de Rich, regresar a mi celda y que nadie se enterase de mi excursión.

Me costó decidirme algo menos de dos minutos. Por supuesto, acepté. Creo que estos dos tienen todo bien estudiado, así que parece que el riesgo es mínimo. Lo que no sabía es que el plan iba a comenzar al día siguiente. Bud no me había estado enseñando Java por casualidad. Necesitaban a una tercera persona que supiese programar para llevar el plan a cabo.

La primera parte del plan es sencilla. Bueno, al menos mi aportación. En la biblioteca de la cárcel hay varios ordenadores con los que se puede acceder a todo el sistema de seguridad. Desde ahí, Pedro va a acceder a la función de alarma de la cárcel y hacer que se dispare durante algunos minutos. Gracias a esa distracción, Bud podrá acceder al despacho del director, donde conseguirá algo que nos ayudará en el siguiente paso.

Es importante que se acceda desde uno de los ordenadores de la biblioteca, ya que, si se hace desde el ordenador de Pedro o el de Bud, los descubrirían a los pocos minutos. Ambos portátiles están muy controlados y cada acción queda registrada.

Hay un pequeño problema, y es que Pedro no tendrá mucho tiempo para hackear el sistema. Lo cierto es que hay mucho que programar y apenas unos pocos minutos para hacerlo sin ser descubierto. Ahí es donde entro yo. Debo dejar creados cinco programas en Java, ocultos en el PC de la biblioteca, que Pedro utilizará después para hacer su magia.

Dispongo exactamente de una hora de tiempo en la biblioteca. Pero normalmente los ordenadores están muy solicitados, así que es posible que solo pueda usarlos durante media hora. No importa, me han explicado exactamente lo que tengo que hacer, así que tengo tiempo de sobra.

Mañana es el gran día. Estoy algo nervioso, pero esta noche intentaré descansar. No es fácil dormir en estas circunstancias, pero trataré de pensar en que, si todo sale bien, demostraré mi inocencia y podré salir de aquí por fin.

Dentro de la biblioteca de la cárcel:

Llegué a la biblioteca de los primeros. Mi objetivo era asegurarme de no quedarme sin ordenador. He tenido suerte, ya que hoy no hay mucha gente que quiera utilizarlos. Al menos tengo media hora. No es mucho tiempo y hay bastante trabajo por hacer. Como ya te comenté, mi parte es crear y dejar ocultos varios programas de Java que posteriormente Pedro utilizará para entrar al sistema central y crear un dispositivo de distracción.

Programa 1

Lo primero será crear un programa que descubra la contraseña del sistema central. La contraseña consta de 6 dígitos, de los cuales creemos que conocemos 5. Por lo tanto, simplemente habrá que comprobar 10 combinaciones.

El problema es que, si se comprueban los datos demasiado rápido, el sistema puede detectarlo. Así que, después del noveno intento, hay que esperar al menos 30 segundos. Para ello, haremos que el programa se detenga después de probar 9 intentos, y para comprobar el último deberemos escribir “seguir” por teclado.

Aquí tienes el código que he utilizado. Revisa línea por línea y verás cómo comprendes su funcionamiento sin problemas:

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      // Pass es la contraseña que no sabemos. Está oculta en el programa.
      // De momento generamos una variable para ello y le damos valor 0.
      // Pero en las líneas de código reales de la cárcel es una contraseña real.

      int pass = 0;

      // Sabemos todos los números de la contraseña menos el último.

      int passIncompleto = 58987;

      // Declaramos la variable para los intentos de adivinar la contraseña.

      int passCompleto;

      // Creamos un Scanner ya que necesitaremos introducir texto.

      Scanner scan = new Scanner(System.in);

      // Creamos una variable para continuar
      // En caso de llegar al noveno intento Pedro deberá esperar 30 segundos
      // Después deberá pulsar "seguir"

      String seguir = "seguir";

      //Nuestro programa tendrá 10 intentos.
      //Así que creamos un bucle "for" para ello.


      for (int i = 0; i<10;i++) {

         // De esta forma generamos el último número que se añade
         // a nuestra contraseña incompleta.
```

```
passCompleto = passIncompleto*10 + i;

        i++;


    // Cuando nuestro intento coincida con la contraseña real.
    // Se mostrará un texto

          if (pass == passCompleto) {

              System.out.println("Esta es la contraseña");
              break;

          }

    // Necesitamos avisar a Pedro si el programa ha realizado 8 intentos
    // Después el decide si quiere probar el último

          if (i == 9) {


  System.out.println("Este es el noveno intento, debes esperar 30 segundos");
  System.out.println("Escribe seguir para continuar");

              String pausa = scan.nextLine();

              if (!seguir.equals(pausa)) {

                 System.out.println("Error");
                 System.exit(0);

              }
           }

        }

       System.out.println("Contraseña conseguida con éxito: " + pass);

    }
}
```

Programa 2

Con la contraseña podemos entrar al sistema central; desde allí, se puede acceder a la alarma. Esta se encuentra en un circuito donde generalmente se ahorra energía y está apagada todo el tiempo. Por lo tanto, antes de nada, hay que asegurarse de que la electricidad esté funcionando. Después de que esta condición se cumpla, Pedro deberá marcar como true la opción de alarma de incendios manual. De esta manera, posteriormente podrá hacer que salte. Vamos a dejarle el código bien preparado.

Vamos a crear un programa que busque la variable booleana alarmaActivada y la marque como true. Pero solamente si la variable booleana electricidadOn está activada. Si no, el programa tendrá que activarla.

Solución:

```
public class Main {
   public static void main(String[] args) {

 // Declaración de las variables

      boolean electricidadOn = false;
      boolean alarmaActivada = false;

 // Comprobamos el estado de electricidadOn

      if (electricidadOn) {

 // Si electricidadOn está activada, activamos la alarma

         alarmaActivada = true;
          System.out.println("La electricidad está activada. La alarma se ha
          activado.");

      } else {
```

```
// Si electricidadOn no está activada, la activamos y luego activamos la
alarma

        electricidadOn = true;
        alarmaActivada = true;

         System.out.println("La electricidad no estaba activada. Se ha
         activado la electricidad y la alarma.");

     }


// Mostrar el estado final de las variables

         System.out.println("Estado final - Electricidad: " + electricidadOn
         + ", Alarma: " + alarmaActivada);

   }
}
```

Programa 3.

La alarma no puede dispararse en cualquier momento. Es preciso que lo haga cuando todos los carceleros se encuentren en sus garitas. De este modo, hasta que lleguen a la sala central para desactivar la alarma, pasarán al menos 60 segundos. Ese es el tiempo que necesitamos para que dure la distracción. En el hipotético caso de que solo un guardia se encuentre en la sala central, podría apagar la alarma en pocos segundos y el plan se iría al traste.

El sistema de la cárcel marca cada garita con un 1 si hay alguien dentro y con un 0 si está vacía. Por lo tanto, debemos crear un programa en el que se establezca una matriz 2D que represente las 6 garitas de la cárcel. Después, creamos una variable booleana que sea true si todas las garitas están ocupadas. Si al menos una se encuentra libre, será false.

Solución:

```
public class Carcel {

   public static void main(String[] args) {

      // Crear una matriz 2D de 6 filas y 1 columna para representar las 6
garitas

      int[][] garitas = {
         {1}, // Garita 1
         {1}, // Garita 2
         {1}, // Garita 3
         {1}, // Garita 4
         {1}, // Garita 5
         {1} // Garita 6
      };

      // Variable booleana que indica si todas las garitas están ocupadas

      boolean todasOcupadas = true;

      // Comprobar el estado de las garitas

      for (int i = 0; i < garitas.length; i++) {

         if (garitas[i][0] == 0) {
            todasOcupadas = false;
            break;

         }
      }

 // Imprimir el resultado

      if (todasOcupadas) {
         System.out.println("Todas las garitas están ocupadas.");
      } else {
         System.out.println("Hay garitas libres.");
      }
   }
}
```

Programa 4.

Además de los 60 segundos de distracción sonora, necesitaremos que después se apaguen las luces del complejo. Así que el plan es que, una vez que los guardias desactiven la alarma, haya un apagón instantáneamente. Nuestro trabajo es crear un sencillo programa que cambie el estado de una variable booleana llamada apagarLuz de false a true.

Solución:

```
public class CambioEstado {
   public static void main(String[] args) {

      boolean apagarLuz = false; // Estado inicial de la variable apagarLuz
      System.out.println("Estado inicial de apagarLuz: " + apagarLuz);

      // Cambiar el estado de apagarLuz de false a true

      apagarLuz = true;
      System.out.println("Nuevo estado de apagarLuz: " + apagarLuz);
   }
}
```

Programa 5.

Esos son todos los programas que Pedro va a necesitar para ofrecer una buena distracción a Bud. Pero él también necesita cubrirse las espaldas. La alarma no puede sonar mientras él esté sentado frente a un ordenador. Sería muy obvio que él es el responsable. Debemos crear un programa muy simple en el que se introduzca por teclado el número de segundos necesarios para que empiece a sonar la alarma. Así, Pedro podrá salir de la biblioteca sin que nadie sospeche que ha sido él. Dependiendo de la situación, él mismo decidirá cuántos segundos necesita.

La única condición es que el número introducido sea mayor de 600. Eso equivale a 10 minutos, con lo cual Pedro tendrá el tiempo suficiente para llegar a su celda y que nadie sospeche.

```
import java.util.Scanner;
public class Carcel {
   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      // Pedir al usuario que introduzca un número

      System.out.print("Introduce un número mayor de 600: ");
      int numero = scanner.nextInt();


 // Verificar si el número es mayor de 600

      if (numero > 600) {
         System.out.println("El número " + numero + " es correcto.");
      } else {
         System.out.println("El número no es correcto. Debe ser mayor de 600.");

      }

      // Cerrar el scanner

      scanner.close();

   }
}
```

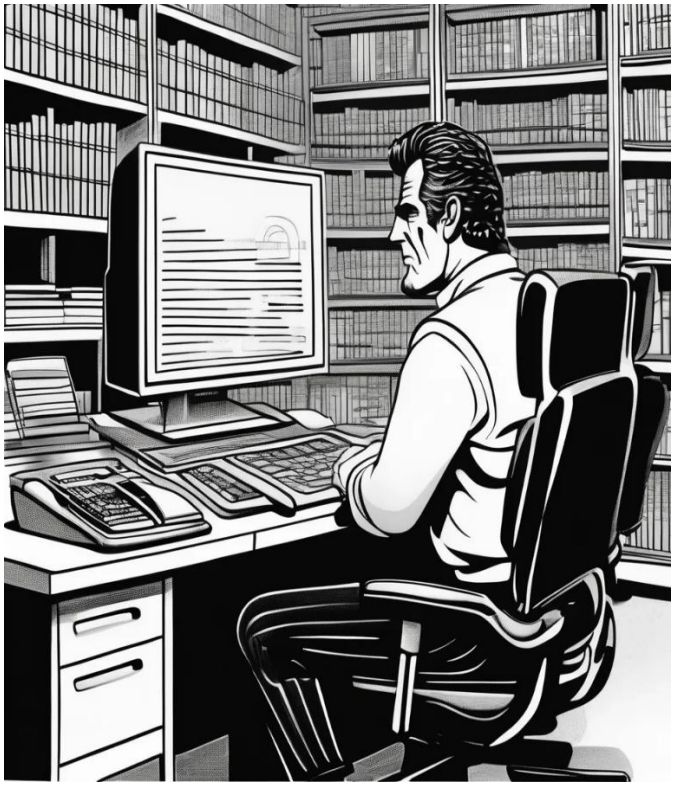
Capítulo 10.

Trabajando con fecha y hora

Mi parte ya está hecha. Después de crear el código de los programas que Pedro necesita, los he escondido dentro de una carpeta del sistema. No me costó ni media hora hacerlo. Ya no necesito más el ordenador, así que iré a leer un poco.

Estoy bastante aliviado porque, ahora que esta parte del plan salga bien, no depende de mí. A mí solo me queda esperar a ver qué pasa. Incluso el riesgo de que me pillen es mínimo. Solo podrían descubrirme si Pedro o Bud me delataran, pero confío en ellos.

Me han informado de que el primo de Chani vuelve a verme esta tarde, así que le comentaré que voy a salir pronto. Mi excursión será corta, de unas horas, pero aún no sé exactamente qué día. Por lo tanto, deberemos seguir en contacto.



Mañana por la tarde, Pedro vendrá a la biblioteca y, en menos de diez minutos, tendrá que utilizar mis programas para crear el dispositivo de distracción. Antes de que empiece a sonar la alarma, tendrá unos minutos para salir de la biblioteca y volver a la celda. Así, nadie sospechará de él.

Una vez que suene la alarma, los funcionarios de prisiones dedicarán todo su esfuerzo a apagarla y a comprobar que el sistema de seguridad sigue activado. Bud utilizará ese tiempo para conseguir abrir la celda y atravesar el primer pasillo, en el que no debería haber ningún guardia. Aunque tendrá que tener cuidado con las cámaras.

En el segundo pasillo hay un vigilante que nunca abandona su puesto, pero hace patrullas en círculos. Así que, sabiendo su patrón de vigilancia, Bud deberá sortearlo para evitar que le vea. Recuerda que, cuando consigan desactivar la alarma, se apagarán las luces. Eso dará aún más tiempo a Bud. En teoría, será tiempo suficiente para que entre al despacho sin ser visto.

La parte más arriesgada de todo el plan será la salida del despacho. Bud tendrá que controlar las pautas de vigilancia de los guardias para salir exactamente en el momento preciso. Deberá tener en cuenta también los patrones de las cámaras, exactamente lo mismo que hizo para llegar hasta el despacho.

Para llevar a cabo este complicado plan, Bud va a tener que utilizar algunos conceptos de Java que yo aún no he aprendido, como trabajar con fecha y hora. Yo no voy a necesitar saber hacer eso, pero me gustaría aprenderlo de todos modos.

Le he pedido a Bud que me enseñe y él está encantado. Puede que al final no esté tan tranquilo como parece y a lo mejor le apetezca mantener la mente ocupada. Sea como fuere, yo se lo agradezco. Me gusta aprender cosas nuevas y, la verdad, también me viene bien tener la cabeza ocupada.

Teoría para Trabajar con Fechas y Horas en Java

La explicación teórica en este tema va a ser un poco diferente a la de los anteriores. Bastante más resumida. La primera razón es que no creo que sea necesario mencionar la importancia que tiene la fecha y hora en cualquier aplicación o programa. Todos tenemos un teléfono móvil o un ordenador y sabemos que muchas funciones de nuestros dispositivos ni siquiera funcionan sin los datos horarios correctos.

La segunda razón es que, a estas alturas, ya eres capaz de comprender y escribir una gran parte del código que se utiliza en Java. Entiendes, entre otras cosas, lo que son las variables, las estructuras principales y el orden de funcionamiento de los programas. Por lo tanto, esta parte te va a resultar pan comido.

La tercera razón es que la sintaxis de este tema no es nada compleja. Ni siquiera se puede decir que sea una estructura. Simplemente utilizaremos una línea u otra de código dependiendo exactamente del tipo de fecha u hora que necesitemos obtener.

Las clases LocalTime, LocalDate y LocalDateTime

- LocalTime representa únicamente la hora del día, sin incluir ninguna información sobre la fecha o la zona horaria.
- LocalDate representa solo una fecha, sin incluir ninguna información sobre la hora o la zona horaria.
- LocalDateTime combina la información de LocalDate (fecha) y LocalTime (hora), representando tanto la fecha como la hora, pero sin incluir información sobre la zona horaria.

Las importamos de la siguiente forma:

- import java.time.LocalTime;
- import java.time.LocalDate;
- import java.time.LocalDateTime;

Ahora, voy a mostrarte, punto por punto, el código que necesitarás utilizar dependiendo de cada situación concreta.

- **Obtener la fecha actual**

Para obtener la fecha actual en Java, se utiliza la clase LocalDate. Esta clase representa una fecha sin hora, en formato ISO (aaaa-mm-dd).

LocalDate fechaActual = LocalDate.now();

- **Obtener la hora actual**

Para obtener la hora actual, se usa la clase LocalTime, que representa una hora sin fecha en formato hh:mm

LocalTime horaActual = LocalTime.now();

- **Obtener la fecha y hora actual**

La clase LocalDateTime combina LocalDate y LocalTime para representar una fecha y una hora en el mismo objeto.

LocalDateTime fechaHoraActual = LocalDateTime.now();

- **Crear una fecha específica**

Puedes crear una fecha específica utilizando el método of de la clase LocalDate, que requiere año, mes y día como parámetros.

LocalDate fecha = LocalDate.of(2024, 5, 29);

- **Crear una hora específica**

Similarmente, LocalTime también tiene un método of para crear una hora específica, utilizando horas, minutos y segundos como parámetros.

LocalTime hora = LocalTime.of(14, 30, 45);

- **Sumar días a una fecha**

Puedes sumar días a una fecha utilizando el método plusDays de la clase LocalDate.

LocalDate nuevaFecha = fechaActual.plusDays(10); // Diez son los días sumados en este ejemplo

- **Restar días a una fecha**

LocalDate nuevaFecha = fechaActual.minusDays(10); // Diez son los días restados

- **Sumar horas a una hora**

LocalTime nuevaHora = horaActual.plusHours(3); // 3 son las horas sumadas en este ejemplo

- **Restar horas a una hora**
-

Para restar horas a una hora, se utiliza el método minusHours de la clase LocalTime.

LocalTime nuevaHora = horaActual.minusHours(3); // 3 son las horas restadas en este ejemplo

- **Comparar dos fechas**

Para comparar dos fechas, LocalDate proporciona métodos como isAfter, isBefore e isEqual.

boolean esDespues = fecha1.isAfter(fecha2);

boolean esAntes = fecha1.isBefore(fecha2);

boolean esIgual = fecha1.isEqual(fecha2);

- **Formatear una fecha**

Para formatear una fecha en un formato específico, se usa la clase DateTimeFormatter.

```
DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
String fechaFormateada = fecha.format(formatter);
```

- **Obtener la diferencia entre dos fechas**

Para calcular la diferencia entre dos fechas, se usa la clase Period

```
Period diferencia = Period.between(fechaInicio, fechaFin);
```

Resumen de lo aprendido

En el lenguaje de programación Java, si necesitas obtener la fecha actual, puedes usar la clase LocalDate, mientras que, para obtener la hora actual, se utiliza la clase LocalTime. Cuando se requiere manejar tanto la fecha como la hora conjuntamente, la clase LocalDateTime es la opción adecuada. Para establecer una fecha específica, puedes emplear el método of de LocalDate. De manera similar, para definir una hora específica, se utiliza el método of de LocalTime.

Los métodos plusDays y minusDays de LocalDate te permiten sumar o restar días a una fecha, respectivamente. Asimismo, con los métodos plusHours y minusHours de LocalTime puedes sumar o restar horas a una hora determinada.

La clase LocalDate proporciona métodos como isAfter, isBefore e isEqual para comparar dos fechas. Para dar formato a una fecha en un estilo específico, se usa la clase DateTimeFormatter. Finalmente, para determinar la diferencia entre dos fechas, se emplea la clase Period.

Ejercicios del tema

Ejercicio 1. Obtener la fecha actual

Solución:

```
import java.time.LocalDate;
public class Main {
   public static void main(String[] args) {

      LocalDate fechaActual = LocalDate.now();
      System.out.println("La fecha actual es: " + fechaActual);

   }
}
```

Ejercicio 2. Obtener la hora actual

Solución:

```
import java.time.LocalTime;
public class Main {

   public static void main(String[] args) {
      LocalTime horaActual = LocalTime.now();
      System.out.println("La hora actual es: " + horaActual);

   }
}
```

Ejercicio 3. Obtener la fecha y hora actual

Solución:

```
import java.time.LocalDateTime;
public class Main {


    public static void main(String[] args) {
        LocalDateTime fechaHoraActual = LocalDateTime.now();
        System.out.println("La fecha y hora actual es: " + fechaHoraActual);

    }
}
```

Ejercicio 4. Crear una fecha específica

Solución:

```
import java.time.LocalDate;
public class Main {

    public static void main(String[] args) {
        LocalDate fecha = LocalDate.of(2024, 5, 29);
        System.out.println("Fecha específica: " + fecha);

    }
}
```

Ejercicio 5. Crear una hora específica

Solución:

```
import java.time.LocalTime;
public class Main {

   public static void main(String[] args) {
      LocalTime hora = LocalTime.of(14, 30, 45);
      System.out.println("Hora específica: " + hora);

   }
}
```

Ejercicio 6. Sumar días a una fecha

Solución:

```
import java.time.LocalDate;
public class Main {

   public static void main(String[] args) {
      LocalDate fechaActual = LocalDate.now();
      LocalDate nuevaFecha = fechaActual.plusDays(10);
      System.out.println("Fecha actual: " + fechaActual);
      System.out.println("Fecha después de 10 días: " + nuevaFecha);

   }
}
```

Ejercicio 7. Restar horas a una hora

Solución:

```
import java.time.LocalTime;
public class Main {

   public static void main(String[] args) {
      LocalTime horaActual = LocalTime.now();
      LocalTime nuevaHora = horaActual.minusHours(3);
      System.out.println("Hora actual: " + horaActual);
      System.out.println("Hora después de restar 3 horas: " + nuevaHora);

   }
}
```

Ejercicio 8. Comparar dos fechas

Solución:

```
import java.time.LocalDate;
public class Main {

   public static void main(String[] args) {

      LocalDate fecha1 = LocalDate.of(2024, 5, 29);
      LocalDate fecha2 = LocalDate.of(2023, 12, 25);
```

```
        if (fecha1.isAfter(fecha2)) {
            System.out.println(fecha1 + " es después de " + fecha2);
        } else if (fecha1.isBefore(fecha2)) {
            System.out.println(fecha1 + " es antes de " + fecha2);
        } else {
            System.out.println(fecha1 + " es igual a " + fecha2);
        }
    }
}
```

Ejercicio 9. Formatear una fecha

Solución:

```
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;

public class Main {
    public static void main(String[] args) {

        LocalDate fecha = LocalDate.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy");
        String fechaFormateada = fecha.format(formatter);
        System.out.println("Fecha formateada: " + fechaFormateada);
    }
}
```

Ejercicio 10. Obtener la diferencia entre dos fechas

Solución:

```
import java.time.LocalDate;
import java.time.Period;

public class Main {

    public static void main(String[] args) {
        LocalDate fechaInicio = LocalDate.of(2023, 5, 1);
        LocalDate fechaFin = LocalDate.of(2024, 5, 29);

        Period diferencia = Period.between(fechaInicio, fechaFin);
        System.out.println("Diferencia entre fechas: " +
            diferencia.getYears() + " años, " +
            diferencia.getMonths() + " meses, " +
            diferencia.getDays() + " días.");

    }
}
```

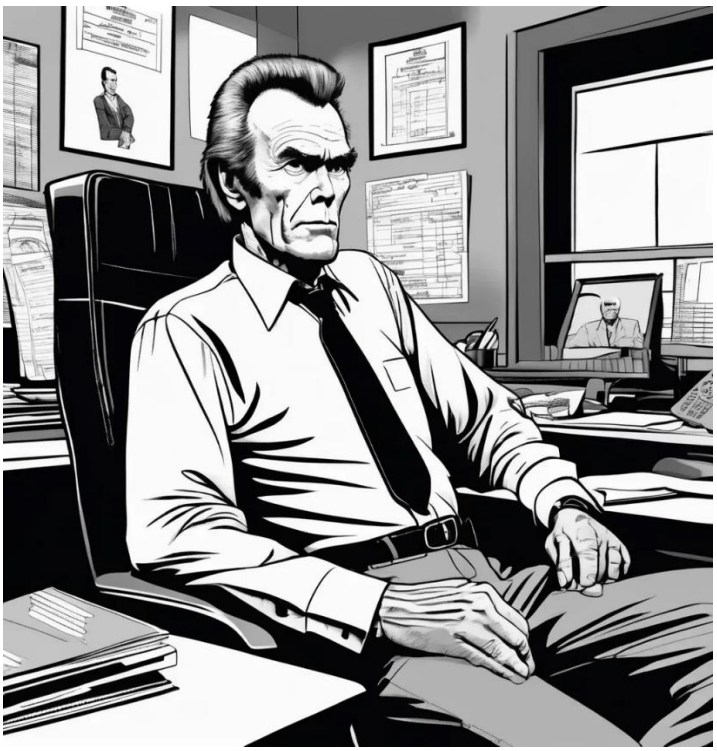
Capítulo 11.

Volviendo a hackear el sistema de seguridad

El día ha llegado, la fuga ha comenzado oficialmente y, a partir de hoy, no hay marcha atrás. Pedro debe estar ya en la biblioteca, así que, si todo ocurre como esperamos, en breves momentos sonará la alarma. Bud ya está preparado con su ordenador; sin él no podría llegar hasta el despacho del director. No decimos nada; Bud mira la pantalla fijamente, muy concentrado.

No todos los días le dejan tener el portátil en la celda, pero hoy se aseguró de que así fuera. Se las ingenió para que el sistema de ventilación de la cocina dejase de funcionar y, en teoría, está tratando de repararlo.

Colarse en el despacho del director tiene un único objetivo: acceder a la libreta del director, donde apunta todo manualmente. Es un señor de edad avanzada, al que no le gusta mucho la informática. Se llama Vicente. La información que tratamos de conseguir es muy valiosa, ya que nos servirá para las siguientes fases del plan.



Por fin, el momento llegó: la alarma se disparó. Era mucho más ruidosa de lo que había imaginado. Vi que Bud empezaba a teclear a toda velocidad. El asalto al despacho de Vicente había comenzado.

Para hacer más amena la espera, vamos a meternos en la piel de Bud y a programar todo el código necesario para llevar a cabo esta parte del plan. Si queremos llegar hasta el final, necesitamos completar cinco pasos y, además, tener algo de fortuna.

Ejercicio 1. Bud necesita desbloquear la puerta principal de su celda. Para ello, debe escribir un programa que verifique un código de seguridad antes de insertarlo. Si se ingresa el código incorrecto, la puerta podría bloquearse. El código correcto está guardado en un array y debe ser comparado con una entrada del usuario.

Instrucciones:

- Crea un array que contenga el código de seguridad, por ejemplo: {1, 4, 5, 7, 9}.
- Pide al usuario que ingrese un código de cinco dígitos.
- Compara el código ingresado con el código almacenado en el array.
- Si el código es correcto, muestra un mensaje de éxito. Si no, muestra un mensaje de error.

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {

       int[] codigoSeguridad = {1, 4, 5, 7, 9};

       Scanner scanner = new Scanner(System.in);



 System.out.println("Ingrese el código de seguridad de 5 dígitos:");

       int[] codigoIngresado = new int[5];

       for (int i = 0; i < 5; i++) {
          codigoIngresado[i] = scanner.nextInt();
       }
```

```
        boolean esCorrecto = true;

        for (int i = 0; i < 5; i++) {
            if (codigoIngresado[i] != codigoSeguridad[i]) {
                esCorrecto = false;
                break;
            }
        }

        if (esCorrecto) {

            System.out.println("Código correcto. La puerta está desbloqueada.");

        } else {

            System.out.println("Código incorrecto. Inténtelo de nuevo.");

        }
    }
}
```

Ejercicio 2. Evitar la cámara de Seguridad. Bud necesita moverse por el primer pasillo sin ser detectado por una cámara de seguridad que se enciende y apaga a intervalos regulares. Debe escribir un programa que determine si puede moverse en un momento dado.

En realidad, el programa de Bud es un poco más elaborado que el que vamos a hacer nosotros, ya que él no sabe en qué segundos se encienden las cámaras. Esa información la obtiene directamente gracias a su conexión con el sistema central, y su programa funciona con esos datos.

Instrucciones:

- Usa un bucle for para simular los segundos de un minuto (0 a 59).
- La cámara está encendida durante los primeros 10 segundos de cada minuto y apagada durante los siguientes 50 segundos.
- Pide al usuario que ingrese un segundo en el que planea moverse.
- Determina si la cámara estará encendida o apagada en ese segundo y muestra un mensaje correspondiente.

Solución:

```
Import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Ingrese el segundo en el que planea moverse (0-59):");
      int segundo = scanner.nextInt();

      if (segundo >= 0 && segundo < 10) {

         System.out.println("La cámara está encendida. Espere un momento más.");

      } else if (segundo >= 10 && segundo < 60) {

         System.out.println("La cámara está apagada. ¡Puede moverse!");

      } else {

         System.out.println("Entrada inválida. Intente de nuevo.");

      }
   }
}
```

Ejercicio 3. Hay otro pequeño problema: el despacho se encuentra en una zona separada del pabellón principal. Eso significa que aquí no sonará la alarma de incendios. Por lo tanto, el vigilante permanecerá ahí, justo delante de la puerta a la que necesitamos acceder. Bud tendrá que evitarlo. La idea es sencilla y a la vez muy efectiva: el plan consiste en aumentar la temperatura del pasillo hasta que el vigilante no pueda soportarla más y se vea obligado a ir a la sala central para bajar los grados. Esto le dará a Bud el tiempo suficiente para entrar y salir del despacho sin ser visto. Las funciones del sistema que no requieren un nivel alto de seguridad, como la regulación del termostato, no están controladas, así que Bud podrá modificar la temperatura sin que nadie lo detecte.

Crea un programa en el que se cumplan los siguientes requisitos:

- Ingresa la temperatura inicial del pasillo.
- Ingresa la cantidad de grados que deseas aumentar.
- Suma el aumento deseado a la temperatura inicial.
- Muestra la nueva temperatura del pasillo después del ajuste.

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {

      // Crear un escáner para leer la entrada del usuario

Scanner scanner = new Scanner(System.in);

      // Solicitar al usuario la temperatura inicial

System.out.print("Ingresa la temperatura inicial del pasillo (en grados
Celsius): ");
double temperaturaInicial = scanner.nextDouble();

      // Solicitar al usuario la cantidad de aumento deseado
```

```
System.out.print("Ingresa la cantidad de grados para aumentar: ");
double aumento = scanner.nextDouble();


// Calcular la nueva temperatura

double nuevaTemperatura = temperaturaInicial + aumento;

      // Mostrar el resultado

System.out.println("La nueva temperatura del pasillo es: " +
nuevaTemperatura + "°C");

      // Cerrar el escáner

      scanner.close();

   }
 }
```

Ejercicio 4. Contraseña para entrar al despacho. Bud necesita generar un código de seguridad basado en la hora actual (minutos y segundos). Este código cambia cada minuto y debe ser ingresado correctamente para desbloquear una puerta.

Instrucciones:

1. Usa la clase **LocalTime** para obtener la hora actual.
2. Genera un código de seguridad sumando los minutos y los segundos actuales.
3. Muestra el código generado al usuario.
4. Pide al usuario que ingrese el código mostrado.
5. Verifica si el código ingresado es correcto y muestra un mensaje adecuado.

Solución:

```
import java.time.LocalTime;
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {

// Obtener la hora actual
LocalTime ahora = LocalTime.now();
int minutos = ahora.getMinute();
int segundos = ahora.getSecond();

// Generar el código de seguridad sumando minutos y segundos
int codigoGenerado = minutos + segundos;

// Mostrar el código generado
System.out.println("El código de seguridad generado es: " +
codigoGenerado);

// Pedir al usuario que ingrese el código mostrado
Scanner scanner = new Scanner(System.in);
System.out.println("Ingrese el código de seguridad para validar:");
int codigoIngresado = scanner.nextInt();

// Verificar si el código ingresado es correcto

      if (codigoIngresado == codigoGenerado) {

         System.out.println("Código correcto. La puerta está abierta.");

      } else {

         System.out.println("Código incorrecto. Inténtelo de nuevo.");

      }
   }
}
```

Ejercicio 5. Una vez dentro, Bud deberá coger la libreta y salir tan rápido como pueda. Aunque probablemente, para cuando la encuentre, el guardia ya habrá regresado a su puesto. Para salir sin ser visto, hay que hacer que el guardia se mueva de nuevo. En la misma sala donde el guardia bajó la temperatura del pasillo, hay una máquina de café. Esta se encuentra estropeada y, cada vez que inicia el proceso de autolimpiado, hace un ruido terrible. Es necesario apagarla manualmente. Así que la idea es activar ese autolimpiado y que el vigilante no tenga más remedio que ir a apagarla.

Por suerte para Bud, este programa ya está creado, dado que la cafetera viene programada de fábrica; aun así, nosotros vamos a crear uno. Como apenas hemos utilizado las condicionales switch-case, creo que es un buen momento para que las repases.

Instrucciones:

- Implementa un menú simple que permita al usuario seleccionar entre preparar café o limpiar la cafetera.
- Cuando el usuario elija la opción de limpiar la cafetera, muestra mensajes que indiquen que la limpieza ha comenzado y que ha terminado.
- Permite que el usuario repita el proceso tantas veces como desee hasta que seleccione la opción de salir del programa.

Solución:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        // Crear un objeto Scanner para leer la entrada del usuario
        Scanner scanner = new Scanner(System.in);

        // Variable para controlar si el programa sigue ejecutándose
        boolean continuar = true;
```

```
// Bucle principal del programa
      while (continuar) {

// Mostrar el menú
          System.out.println("\n--- Menú de la Cafetera ---");
          System.out.println("1. Preparar café");
          System.out.println("2. Activar función de limpieza manual");
          System.out.println("3. Salir");
          System.out.print("Elige una opción: ");

// Leer la opción seleccionada por el usuario
          int opcion = scanner.nextInt();

// Procesar la opción seleccionada
switch (opcion) {
    case 1:
          System.out.println("Preparando café... ¡Disfruta tu bebida!");
          break;
     case 2:
          System.out.println("Iniciando la función de limpieza manual...");
          // Simulación del proceso de limpieza
          System.out.println("La cafetera se está limpiando...");
          System.out.println("¡La limpieza ha finalizado! La cafetera está lista para
usarse.");
          break;
      case 3:
          System.out.println("Saliendo del programa. ¡Hasta luego!");
          continuar = false; // Salir del bucle
          break;
          default:
          System.out.println("Opción no válida. Por favor, elige una opción del
menú.");
                break;
          }
       }

       // Cerrar el scanner
       scanner.close();
   }
}
```

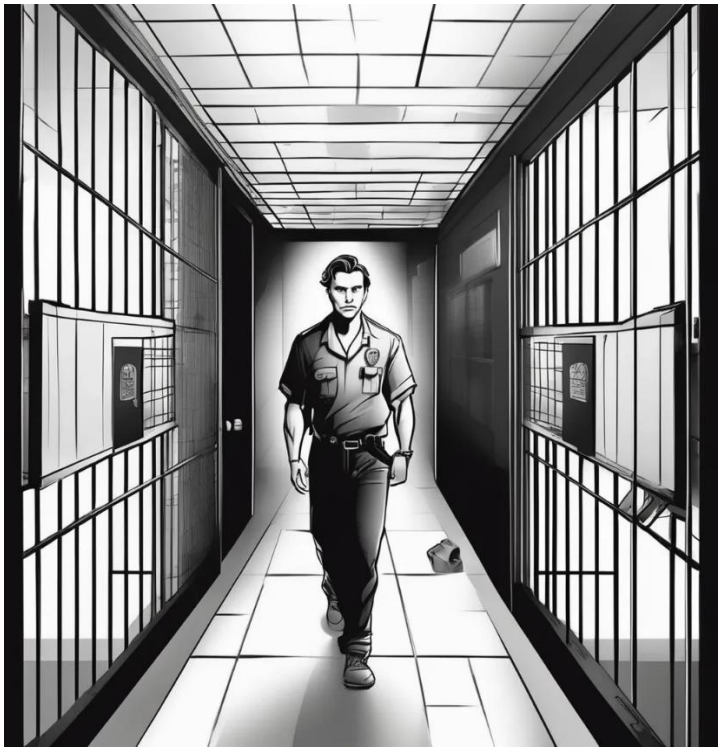
Capítulo 12.

Try & catch

Bud ha conseguido las claves que necesitaba del despacho de Vicente. Van a resultar esenciales para ejecutar el plan. Salir de la prisión no va a ser sencillo, pero regresar a ella sin que nadie se dé cuenta va a ser aún más complicado.

Parece ser que uno de los funcionarios que trabaja en el hospital de la cárcel ha estado robando algunos medicamentos para venderlos entre los presos. La idea es extorsionarlo para que nos ayude a introducirme de nuevo. No debería ser muy difícil; seguro que coopera sin problemas.

Una vez fuera, cuando llegue el momento de regresar, me esconderé en el maletero de su coche y simplemente entraremos hasta el aparcamiento de empleados. Desde allí, acceder a la biblioteca sin ser visto es muy sencillo. Deberé esconderme hasta que llegue la hora diaria de visita a la biblioteca y luego mezclarme con el grupo.



El guardia en cuestión se llama Phil y no nos ayudará sin motivo. De hecho, si se entera de nuestros planes de fuga, nos delataría inmediatamente.

Necesitamos pruebas de su fechoría; una vez que las tengamos, no tendrá más remedio que colaborar. Bud se encargará de ello. Tiene muchos recursos y parece que ha estado siguiendo de cerca las actividades de Phil. Por el momento, yo no tengo que hacer nada, aunque ya me han avisado que mi aportación será crucial al final de la huida

Ya tenemos una fecha aproximada para la fase final de nuestro plan. En dos semanas habrá unos días festivos, y las medidas de seguridad suelen reducirse. No mucho, pero al menos eso nos da algo de ventaja. Con esta información, ahora puedo coordinarme con el primo de Chani y decirle más o menos qué día voy a salir.

Una vez especificado este rango de días, espero que Chani se encargue de contactar con Rich y organizar un encuentro. No sé cómo vamos a conseguir la confesión; supongo que la idea es grabar un audio con el móvil, pero como estoy prácticamente incomunicado, aún no tengo más detalles.

Ayer tuve una larga conversación con Bud, y me comentó lo que piensa hacer cuando nuestros caminos se separen. Pedro y él llevan mucho tiempo planeando la fuga. Es una oportunidad que no pueden dejar escapar. No sé cuántos años tienen exactamente, pero calculo que rondan los 50. Aún les queda una cantidad considerable de condena y no quieren pasar lo que les queda de vida encerrados.

Tienen contactos fuera y van a abandonar el país el mismo día en que salgan. El plan es estar muy lejos una vez que los guardias se den cuenta de que han escapado. Espero que todo salga bien.

Hay que ponerse manos a la obra y conseguir esas pruebas que incriminen a Phil. Para ello, debemos llegar hasta el hospital de la cárcel y, para eso, voy a necesitar utilizar algunos conceptos de Java que aún no he aprendido. Así que, sin que yo le dijera nada, Bud se ha ofrecido a darme una nueva lección. Es un gran profesor, así que no puedo negarme.

Introducción e importancia de los bloques try y catch en Java

En Java, se utilizan bloques de código try y catch para manejar excepciones. Las excepciones son situaciones que pueden ocurrir mientras se ejecuta un programa y que podrían interrumpir el flujo normal de ejecución del programa. Por lo tanto, para evitar esto, debemos tener en cuenta esos posibles eventos y controlarlos de antemano.

Estos bloques son totalmente necesarios ya que son una forma organizada y controlada de tratar errores y excepciones durante la ejecución del programa. Gracias a try y catch vamos a evitar posibles errores en nuestro programa con los que ni siquiera contamos.

Las excepciones más típicas

División por Cero: Envolviendo la división dentro de un bloque try, puedes capturar la ArithmeticException y mostrar un mensaje de error adecuado.

Conversión de Cadenas a Números: Utiliza try-catch para capturar NumberFormatException y manejar entradas no válidas**.**

Acceso a Índices de Arrays: Protege el acceso a los elementos del array con try-catch para capturar ArrayIndexOutOfBoundsException.

Operaciones Aritméticas Seguras: Usa Math.subtractExact dentro de un bloque try para capturar ArithmeticException en caso de desbordamiento.

Concatenación de Cadenas: Envuelve el bucle de concatenación en un bloque try para capturar OutOfMemoryError y manejar el error.

Estructura de los bloques try-catch

- **Bloque try**

```
try {
    // Código que puede generar una excepción
}
```

- **Bloque catch**

Se coloca directamente después del bloque try y tiene esta estructura:

```
catch (TipoDeExcepcion e) {
      // Código para manejar la excepción
 }
```

- **Ejemplo básico**

```
public class Main {
public static void main(String[] args) {
     try {
          int resultado = 10 / 0; // Esto lanza ArithmeticException
          System.out.println("El resultado es: " + resultado);
      } catch (ArithmeticException e) {
           System.out.println("Error: No se puede dividir por cero.");
     }
  }
}
```

Múltiples Bloques catch

Puedes tener múltiples bloques catch para manejar diferentes tipos de excepciones. Cada catch maneja una excepción específica.

```
try {
    // Código que puede lanzar diferentes excepciones
} catch (ArithmeticException e) {
    // Maneja ArithmeticException
} catch (NullPointerException e) {
    // Maneja NullPointerException
}
```

El Bloque finally (Opcional): Puedes añadir un bloque finally después de los bloques catch. El bloque finally se ejecuta siempre, independientemente de si se lanzó o no una excepción, y es útil para liberar recursos como cerrar archivos o conexiones de base de datos.

```
try {
    // Código que puede lanzar excepciones
} catch (Exception e) {
    // Maneja cualquier excepción
} finally {
    // Código que siempre se ejecuta, con o sin excepción
}
```

Resumen de lo aprendido

Como te comentaba al principio del tema, en Java, los bloques try y catch son fundamentales para gestionar situaciones imprevistas durante la ejecución de un programa, evitando interrupciones en su desarrollo normal.

Puede que en ejercicios sencillos no sea necesario utilizar esta estructura, pero cuando los programas se hacen más complejos es necesario tener en cuenta todos los posibles sucesos que pueden acontecer durante la ejecución de nuestro código. Estos bloques permiten manejar, entre otras cosas, errores como división por cero, conversiones de cadenas incorrectas, accesos fuera de rango en arrays y operaciones aritméticas que podrían exceder límites.

Llegados a este punto, la estructura try-catch no debe resultarte muy complicada. Dentro del bloque try encontramos el código que podría generar una excepción. Si esta se produce, es cuando el bloque catch entra en acción y se encarga de manejar el error. Puedes tener varios bloques catch para diferentes tipos de excepciones y opcionalmente usar el bloque finally para ejecutar código que siempre debe ejecutarse, como la liberación de recursos.

Ejercicios del tema

Ejercicio 1. División por cero. Escribe un programa que solicite al usuario dos números e intente dividir el primero por el segundo. Maneja la excepción ArithmeticException en caso de división por cero.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      System.out.println("Introduce el primer número:");
      int num1 = scanner.nextInt();

      System.out.println("Introduce el segundo número:");
      int num2 = scanner.nextInt();

      try {

         int result = num1 / num2;
         System.out.println("El resultado es: " + result);

      } catch (ArithmeticException e) {

         System.out.println("Error: No se puede dividir por cero.");

      }

      scanner.close();

   }
}
```

Ejercicio 2. Manejo de NumberFormatException: Escribe un programa que solicite al usuario un número entero e intente convertirlo. Maneja la excepción NumberFormatException en caso de que el formato no sea válido. Una vez completado el ejercicio, trata de insertar un número decimal a ver qué pasa.

Solución:

```
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("Introduce un número entero:");
        String input = scanner.nextLine();

        try {

            int number = Integer.parseInt(input);
            System.out.println("El número introducido es: " + number);

        } catch (NumberFormatException e) {

            System.out.println("Error: Formato de número no válido.");


        }

        scanner.close();

    }
}
```

Ejercicio 3. Índice fuera de rango en un array. Escribe un programa que cree un array de 5 elementos y trate de acceder a una posición fuera de rango. Maneja la excepción ArrayIndexOutOfBoundsException

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] array = {1, 2, 3, 4, 5};

      try {

         int value = array[10];
         System.out.println("El valor es: " + value);

      } catch (ArrayIndexOutOfBoundsException e) {

         System.out.println("Error: Índice fuera de rango.");

      }
   }
```

Ejercicio 4. Operación aritmética no válida. Escribe un programa que solicite al usuario dos números y los reste. Maneja la excepción ArithmeticException si la operación no es válida por alguna razón (e.g., resta de valores extremadamente grandes).

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.println("Introduce el primer número:");
      long num1 = scanner.nextLong();

      System.out.println("Introduce el segundo número:");
      long num2 = scanner.nextLong();

      try {

         long result = Math.subtractExact(num1, num2);
         System.out.println("El resultado es: " + result);

      } catch (ArithmeticException e) {

         System.out.println("Error: Operación aritmética no válida.");

      }

      scanner.close();

   }
}
```

Ejercicio 5. Concatenación de cadenas. Escribe un programa que intente concatenar un número muy grande de cadenas, causando un OutOfMemoryError. Maneja la excepción.

Solución:

```
public class Main {
   public static void main(String[] args) {

     try {

         String str = "";
         for (int i = 0; i < Integer.MAX_VALUE; i++) {
            str += "a";

         }

       } catch (OutOfMemoryError e) {

         System.out.println("Error: Memoria insuficiente para la operación.");

       }
   }
}
```

Capítulo 13.

Recopilando pruebas contra Phil

Tras el incidente con la alarma y el posterior apagón, los funcionarios de la prisión están alerta. Empiezan a sospechar algo y se están tomando medidas de precaución extraordinarias. La mayoría de ellas no nos afectan, pero para poder acceder al hospital de la cárcel, lo primero que Bud deberá hacer es desactivar la localización GPS de su portátil. Después de lo que ocurrió con la alarma de incendios, se están tomando muchas medidas de seguridad y todo el mundo está controlado. Es muy probable que los funcionarios de la prisión ya sospechen de Bud y Pedro. Desde ahora, hay que ser totalmente invisibles.

Ejercicio 1. Cambiar la señal de GPS: Bud y Pedro deben cambiar la señal de GPS de sus ordenadores para que los guardias no puedan rastrear su ubicación real. Escribiremos un programa que nos ayude con esta tarea.

Instrucciones:

1. Crea un array que contenga las coordenadas reales de cada ordenador.
2. Permite al usuario ingresar coordenadas falsas.
3. Verifica si las coordenadas ingresadas están dentro de un rango aceptable (por ejemplo, dentro de un radio de 1 km de las coordenadas reales).
4. Si las coordenadas son aceptables, muestra un mensaje de éxito. Si no, muestra un mensaje de error.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      double[] coordenadasReales = {40.7128, -74.0060}; // (ejemplo)

      Scanner scanner = new Scanner(System.in);

      System.out.println("Ingrese las coordenadas falsas de una en una (latitud y longitud):");

      double latitudFalsa = scanner.nextDouble();

      double longitudFalsa = scanner.nextDouble();

      double distancia = Math.sqrt(Math.pow(latitudFalsa - coordenadasReales[0], 2) + Math.pow(longitudFalsa - coordenadasReales[1], 2));



      if (distancia <= 0.01) { // Aproximadamente 1 km de radio
         System.out.println("Coordenadas falsas aceptadas. Los guardias no podrán rastrearte.");

      } else {

         System.out.println("Coordenadas falsas fuera de rango. Inténtelo de nuevo.");

      }
   }
}
```

Nada más entrar al hospital, hay una sala con un nivel de seguridad medio que cuenta con cinco cámaras. Además, los funcionarios son registrados siempre al entrar para evitar que introduzcan objetos del exterior. El problema es que no se les registra al salir, lo que permite a Phil robar el material. En esa sala, Bud deberá evitar las cámaras.

Lo primero que hay que hacer es comprobar que todas las cámaras están encendidas, ya que, de ser así, Bud podrá interceptarlas y apagarlas. Si alguna está apagada de antemano, no se podrá interceptar. Que nuestro programa indique que una cámara está apagada no significa necesariamente que realmente lo esté; puede que utilice otro tipo de fuente de alimentación para funcionar. Sería un riesgo muy grande. Por tanto, el objetivo es verificar que todas estén activas y, luego, apagarlas.

Ejercicio 2. Interceptar Señales de Video. Bud necesita interceptar las señales de video de las cámaras de seguridad. Escribe un programa que simule la interceptación de las señales de video y que muestre un mensaje cada vez que se intercepte una señal.

Instrucciones:

1. Declara 5 variables boolean. (En el programa real de la cárcel no sabremos si están declaradas como true o false. Simplemente para ver si el programa funciona las declaramos con valores aleatorios)
2. Usa un bucle for para simular la interceptación de señales de video de varias cámaras (por ejemplo, 5 cámaras). Haz que compruebe el estado de las 5 cámaras.
3. Muestra un mensaje indicando que la señal de cada cámara ha sido interceptada. True significa que está encendida y por tanto el programa la intercepta. False será que está apagada y el programa no podrá interceptarla.

Solución:

```
public class Main {
   public static void main(String[] args) {

// Declaración de las variables boolean
// Como comentamos en el enunciado, en el sistema de la cárcel no sabremos el estado de las cámaras. Por eso creamos este programa.
//Simplemente agregamos valores para ver si el programa funciona.

      boolean camara1 = true;
      boolean camara2 = false;
      boolean camara3 = true;
      boolean camara4 = false;
      boolean camara5 = true;


// Bucle for para comprobar el estado de cada cámara

      for (int i = 1; i <= 5; i++) {

//Creamos un boolean estado. Hay que asignarle valor para que no haya un error.
//Realmente no importa si es true o falsa. Cambiará dependiendo de cada cámara.

         boolean estado = false;

         if (i == 1) {

            estado = camara1;

         } else if (i == 2) {

            estado = camara2;

         } else if (i == 3) {

            estado = camara3;
```

```
        } else if (i == 4) {

            estado = camara4;

        } else if (i == 5) {

            estado = camara5;
        }


        if (estado) {

            System.out.println("Cámara " + i + " está en estado: Interceptada");
        } else {

            System.out.println("Cámara " + i + " está en estado: No interceptada");

        }
    }
  }
}
```

En la siguiente sala, el nivel de seguridad es alto. Ya no se utilizan cámaras, sino sensores de movimiento, en concreto, cinco. Para llegar a la sala de los ordenadores, es necesario atravesar esta habitación.

Ejercicio 3: Desactivar Sensores de Movimiento. Ahora Bud deberá desactivar los sensores de movimiento para moverse sin ser detectado. Escribe un programa que permita desactivar los sensores uno por uno.

Instrucciones:

1. Usa un array booleano para representar el estado de los sensores (activado o desactivado).
2. Usa un bucle para permitir al usuario desactivar cada sensor.
3. Muestra el estado de los sensores después de cada cambio.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      boolean[] sensores = {true, true, true, true, true}; // Todos los sensores
están activados inicialmente

      Scanner scanner = new Scanner(System.in);

      for (int i = 0; i < sensores.length; i++) {


         System.out.println("¿Desea desactivar el sensor " + (i + 1) + "?
(true/false)");
         sensores[i] = !scanner.nextBoolean();
      }

      System.out.println("Estado de los sensores:");
      for (int i = 0; i < sensores.length; i++) {

         System.out.println("Sensor " + (i + 1) + ": " + (sensores[i] ? "Activado"
: "Desactivado"));

      }
   }
}
```

Protegiendo la sala de los ordenadores, hay una puerta que requiere una clave de acceso numérica. Si se introduce una contraseña incorrecta o no numérica, el programa lo detectará y se activará una alerta. Necesitamos crear un programa utilizando try y catch que avise en caso de introducir un código no numérico o incorrecto. Esta es una precaución por si el código robado a Vicente no funciona. Si eso ocurriese, el plan se vería comprometido y no podríamos escapar, pero al menos no nos descubrirían tratando de huir.

Ejercicio 4. Manejo de Excepciones al Hackear el Sistema de Seguridad. Bud intenta acceder al sistema de seguridad, pero puede cometer errores durante el proceso. Escribe un programa que simule el hackeo y que maneje posibles excepciones (como ingresar un valor no numérico).

Instrucciones:

1. Pide al usuario que ingrese un código numérico.
2. Usa try y catch para manejar posibles excepciones si el usuario ingresa un valor no numérico.
3. Si el valor es correcto, muestra un mensaje de éxito. Si no, muestra un mensaje de error.

Solución:

```
import java.util.Scanner;
public class Main {
   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);
```

```
        try {
            System.out.println("Ingrese el código de hackeo (número
entero):");
            int codigo = Integer.parseInt(scanner.nextLine());
            System.out.println("Código correcto. Hackeo exitoso.");
        } catch (NumberFormatException e) {

            System.out.println("Error: Ingresó un valor no numérico.
Inténtelo de nuevo.");

        }
    }
}
```

Una vez dentro de la última habitación necesitamos acceder al ordenador. Esta última contraseña funciona de forma peculiar. Cada vez utiliza un nuevo código para poder entrar. Esto hace que sea prácticamente imposible de hackear. Para acceder al sistema vamos a necesitar una clave obtenida del despacho de Vicente que después se combina de forma aleatoria con dos otros números. Antes de ingresarla hay que comprobar exactamente cuáles son esos dos números.

Ejercicio 5. Crea un **programa que encuentre la clave completa de 6 dígitos a partir de una clave base de 4 dígitos. La clave base se combina** con un número variable de dos dígitos para formar una clave completa.

- Crea tres variables int, una para la contraseña incompleta que tenemos, otra para la contraseña final y otra temporal que utilizaremos dentro de nuestro for para comprobar todas las combinaciones de 00 a 99.
- Usa un bucle for para iterar desde 0 hasta 99, representando todas las posibles combinaciones de los últimos dos dígitos.
- En cada iteración del bucle, combina la clave base con el número actual del bucle multiplicado por 100, sumando el valor del bucle. Esto genera una posible clave completa (claveComprobada).

- Compara la clave generada en cada iteración (claveComprobada) con la clave completa conocida. Si se encuentra una coincidencia, imprime la clave completa encontrada y finaliza el bucle.

En el programa que Bud utilice para encontrar la contraseña, evidentemente no tendría la contraseña final. Pero nosotros la hemos incluido en este ejercicio para comprobar que funciona bien.

Solución:

```
public class Main {

    public static void main(String[] args) {
        // Clave base conocida
        int claveBase = 8787;

        // Clave completa que se quiere encontrar
        int claveCompleta = 878793;

        // Esta clave aumentara en nuestro bucle de uno en uno
        int claveComprobada;

        // Comprobar combinaciones de los últimos dos dígitos
        boolean encontrado = false;

        for (int i = 0; i < 100; i++) {

            claveComprobada = (claveBase * 100) +i;
```

```
            System.out.println(claveComprobada);
            if (claveComprobada == claveCompleta) {
               System.out.println("Esta es la clave completa:" +
claveComprobada);
               break;
            }

      }

   }
}
```

Capítulo 14.

Leer y escribir en archivos de texto

Como ya tenemos el registro de las medicinas robadas por Phil, es hora de chantajearlo. Pedro se reunirá con él hoy mismo. No tenemos ninguna duda de que va a colaborar. Si presentáramos las pruebas, no solo lo despedirían, sino que se vería inmerso en un proceso legal en el cual posiblemente lo condenarían a algún año de prisión.

El plan está saliendo a la perfección. Nadie sospecha nada, y ya hemos superado más de la mitad del camino. Antes del último paso, y de salir por fin, necesitamos aún dos cosas: tener un registro de las medidas de seguridad que nos esperan en el último tramo antes de la salida y acceder a la comunicación de los guardias.

No te lo he comentado aún porque es una locura, pero la idea es salir por la puerta principal. Parece un suicidio, pero nadie va a sospechar que alguien tratase de escapar por ahí. Como podrás imaginar, es la parte de la prisión más vigilada y debemos saber exactamente a qué nos enfrentamos.

Para analizar la actividad y las tareas de los guardias, contamos con algunos registros de actividades de los guardias. Estos se encuentran en formato de texto y debemos crear unos programas en Java para ayudarnos a entender la información que contienen e incluso modificarla si es necesario.

Como de costumbre, no tengo ni idea de lo que pasa en el exterior. Espero que Chani se esté encargando de todo y que tenga comunicación con Rich. Espero que cuando salga no me entere de que Rich se ha fugado y que todo este esfuerzo ha sido en vano.

Por otro lado, sigo en contacto con mi familia, pero he perdido algo de contacto con mis amigos cercanos. La verdad es que en estos momentos no quiero tener visitas. Estoy centrado en el plan de escape. No quiero pensar en lo que la gente estará diciendo de mí. Eso podría afectarme. Voy a demostrar mi inocencia y demostrar a todos los que dudan de mí que se equivocan.

Parece que mi profesor particular ya está listo para enseñarme a leer y editar archivos usando Java. La verdad es que no es un tema que en concreto me apasione demasiado, pero es necesario para continuar con este plan. Bud ya me ha avisado de que es posiblemente la parte más compleja de todo el temario, así que necesito mentalizarme. Aunque con un profesor tan bueno, seguro que no me cuesta nada entenderlo.

Importancia de leer y escribir archivos en Java

Como estás dando tus primeros pasos en Java, lo normal es que leer y escribir archivos en Java no sea una prioridad para ti. La mayoría de los programas no necesitan esta funcionalidad. Ahora bien, no sería inteligente obviar que es una herramienta fundamental para muchas aplicaciones. Al leer un archivo, una aplicación puede procesar datos externos y trabajar con ellos. De la misma forma que hasta ahora hemos estado dando instrucciones manualmente a nuestros programas, ahora podrán seguir esas mismas directrices desde una hoja de texto.

De igual forma, escribir en archivos permite a los programas guardar resultados, crear logs o almacenar datos importantes que luego pueden ser utilizados o analizados.

Otra ventaja importante es que Java ofrece una gran variedad de clases y métodos dentro de su biblioteca para manejar archivos de manera segura y sencilla. Esto facilita la manipulación de grandes volúmenes de datos, ya sea en formato de texto o binario. Además, el manejo de archivos en Java es esencial para desarrollar aplicaciones que requieren persistencia de datos, como sistemas de gestión de información o bases de datos.

Leyendo Archivos de Texto

En mi opinión, cuando aprendes Java, lo importante no es memorizar conceptos como un loro, sino entender cómo funciona la programación y comprender la lógica que hay detrás. Si lo haces de esta forma, te va a resultar mucho más sencillo; la sensación de progreso va a ser mucho más satisfactoria y, además, sabrás programar para el resto de tu vida. Sí que es cierto que si pasas un tiempo largo sin tocar código es posible que no recuerdes casi nada, pero dedicándole unas pocas horas, esos conocimientos vuelven rápidamente a tu cabeza.

Después de tantas horas programando, ya tendrás un conocimiento bastante avanzado sobre la sintaxis de Java. Lo que al principio parecía imposible de aprender ahora tiene sentido y resulta hasta sencillo. No todo el mundo llega hasta aquí, así que te doy la enhorabuena.

Para este tema en concreto, y muy a mi pesar, vamos a tener que aprender alguna cosa de memoria. Para que nuestros programas puedan leer y escribir en texto, tenemos que importar una serie de clases.

Importar las clases necesarias para leer archivos

```
import java.io.File;
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;
```

Crear un objeto file que represente el archivo de texto

Evidentemente, si tienes un archivo (file) con el que quieres trabajar, no necesitas crearlo, pero lo que sí debes hacer es conocer la ruta en el que el archivo está ubicado en tu ordenador.

```
File archivo = new File("ruta/al/archivo.txt");
```

Este es un ejemplo de una ruta real de un archivo que uso para los ejercicios de ejemplo:

```
File archivo = new File("C:\\Users\\ribeh\\IdeaProjects\\Proyecto 1\\actividad.txt");
```

Leer el archivo utilizando FileReader y BufferedReader

Me gustaría decirte que el programa ya puede leer los datos de nuestro archivo e incluso imprimirlos en la pantalla, pero no es así. Aún queda un paso más.

Necesitamos usar try y catch, pero como ya sabemos cómo funcionan, no habrá ningún problema.

```
try (FileReader fr = new FileReader("actividad.");
   BufferedReader br = new BufferedReader(fr)) {


      String linea;
      while ((linea = br.readLine()) != null) {
         System.out.println(linea); // Imprime cada línea del archivo
      }


   } catch (IOException e) {
      e.printStackTrace();
   }
```

Al FileReader lo llamamos “fr” y al BufferedReader “br”, pero puedes utilizar el nombre que más te convenga. Después creamos un bucle while que recorre las líneas de nuestro archivo y va imprimiendo siempre y cuando haya datos para hacerlo. Lo último es el bloque de catch por si hay alguna excepción.

Escribiendo Archivos de Texto

De nuevo nos va a tocar importar las clases necesarias. En este caso son estas:

```
import java.io.File;
import java.io.FileWriter;
import java.io.BufferedWriter;
import java.io.IOException;
```

Por supuesto, vamos a necesitar disponer de la ruta de nuestro archivo. Ya he explicado como debe ser la ruta:

File archivo = new File("ruta/al/archivo.txt");

Este es un ejemplo de una ruta real de un archivo que uso para los ejercicios de ejemplo:

File archivo = new File("C:\\Users\\ribeh\\IdeaProjects\\Proyecto 1**actividad**.txt");

Escribir en el archivo utilizando FileWriter y BufferedWriter

```
String actividad = "actividad.txt"; // Define el nombre del archivo
try (FileWriter fw = new FileWriter(actividad);
     BufferedWriter bw = new BufferedWriter(fw)) {

    bw.write("Hola, este es un nuevo archivo de texto.");
    bw.newLine(); // Añade una nueva línea
    bw.write("Aquí podemos agregar más contenido.");

} catch (IOException e) {
    e.printStackTrace();
}
```

Hemos utilizado “Fw” para denominar a nuestro FileWriter y “Bw” para el BufferedWriter, pero, como en el caso anterior, podríamos ponerles otro nombre si nos conviene más. En este caso, no nos es necesario crear un bucle dentro del bloque try porque no necesitamos ir recorriendo todas las líneas. Solo necesitamos añadir una o varias líneas. Por último, como es lógico, cerramos con el bloque catch por si sucede una posible excepción.

Resumen de lo aprendido

Para leer archivos de texto en Java, lo primero que debemos hacer es importar las clases File, FileReader, BufferedReader e IOException. Para leer un archivo, debes elegir dicho documento y copiar su ruta. Luego, utilizas FileReader y BufferedReader para leer el archivo línea por línea y mostrar su contenido. Es esencial manejar posibles excepciones con bloques try y catch para que el programa pueda gestionar cualquier error que surja. Por suerte, aprendimos a hacerlo en el tema anterior.

Vamos con la otra cara de la moneda: escribir en esos archivos de texto. Para empezar, también necesitas importar algunas clases, que en este caso son: File, FileWriter, BufferedWriter e IOException. Igual que antes, vamos a necesitar la ruta de nuestro archivo. Después, usaremos FileWriter y BufferedWriter para escribir contenido en el archivo, añadiendo nuevas líneas cuando sea necesario. Es crucial manejar excepciones en este proceso para asegurarte de que cualquier problema durante la escritura sea tratado de manera adecuada.

Estos pasos te permiten leer y escribir archivos de texto en Java de forma efectiva, gestionando errores y excepciones de manera controlada.

Ejercicios del tema

Ejercicio 1. Escribe un programa que guarde la frase 'Hola Mundo' en un archivo llamado salida.txt.

Solución:

```
import java.io.FileWriter;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {

      try (FileWriter writer = new FileWriter("salida.txt")) {
         writer.write("Hola Mundo");

         System.out.println("Texto escrito en salida.txt");

      } catch (IOException e) {

         e.printStackTrace();

      }
   }
}
```

Ejercicio 2. Escribe un programa que lea el contenido de el archivo llamado salida.txt y lo muestre en la consola.

Solución:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {

try (BufferedReader reader = new BufferedReader(new
FileReader("salida.txt"))) {
          String line;
          while ((line = reader.readLine()) != null) {
             System.out.println(line);
          }

       } catch (IOException e) {
          e.printStackTrace();

       }
   }
}
```

Ejercicio 3. Escribe un programa que copie el contenido de salida.txt a destino.txt

Solución:

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {

      try (FileReader reader = new FileReader("salida.txt");
          FileWriter writer = new FileWriter("destino.txt")) {
         int c;
         while ((c = reader.read()) != -1) {
            writer.write(c);
         }

          System.out.println("Contenido copiado de salida.txt a
          destino.txt");

      } catch (IOException e) {
         e.printStackTrace();

      }
   }
}
```

Ejercicio 4. Escribe un programa que cuente el número de palabras del archivo llamado destino.txt

Solución:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {
      int wordCount = 0;
          try (BufferedReader reader = new BufferedReader(new
          FileReader("destino.txt"))) {

          String line;

          while ((line = reader.readLine()) != null) {
             String[] words = line.split("\\s+");
             wordCount += words.length;
          }

           System.out.println("Número de palabras en destino.txt: " +
           wordCount);

          } catch (IOException e) {
          e.printStackTrace();

      }
   }
}
```

Ejercicio 5. Escribe un programa que guarde los números del 1 al 10 en un archivo llamado numeros.txt, con cada número en una línea.

Solución:

```
import java.io.FileWriter;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {

      try (FileWriter writer = new FileWriter("numeros.txt")) {

         for (int i = 1; i <= 10; i++) {
            writer.write(i + "\n");
         }

         System.out.println("Números escritos en numeros.txt");
      } catch (IOException e) {

         e.printStackTrace();

      }
   }
}
```

Ejercicio 6. Escribe un programa que lea números enteros de un archivo llamado numeros.txt y calcule la suma de estos números.

Solución:

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {
   public static void main(String[] args) {
      int sum = 0;
         try (BufferedReader reader = new BufferedReader(new
         FileReader("numeros.txt"))) {
         String line;
         while ((line = reader.readLine()) != null) {
            sum += Integer.parseInt(line);
         }
         System.out.println("Suma de los números en numeros.txt: " + sum);
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Ejercicio 7. Escribe un programa que añada la frase 'Fin del archivo' al final de un archivo llamado log.txt.

Solución:

```
import java.io.FileWriter;
import java.io.IOException;

public class Main {
   public static void main(String[] args) {
      try (FileWriter writer = new FileWriter("log.txt", true)) {
         writer.write("Fin del archivo\n");
         System.out.println("Texto añadido a log.txt");
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Ejercicio 8. Escribe un programa que lea el contenido del archivo números.txt y lo muestre en la consola cada valor multiplicado por 10.

Solución:

```
import java.io.FileReader;
import java.io.BufferedReader;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {
      String archivo = "numeros.txt"; // Archivo que contiene los números

      try (BufferedReader br = new BufferedReader(new
FileReader(archivo))) {
         String linea;
         while ((linea = br.readLine()) != null) {
            int numero = Integer.parseInt(linea); // Convierte la línea en un
número
            int resultado = numero * 10; // Multiplica el número por 10
            System.out.println(resultado); // Muestra el resultado
         }
      } catch (IOException e) {
         e.printStackTrace();
      } catch (NumberFormatException e) {
         System.out.println("El archivo contiene datos no numéricos.");
      }
   }
}
```

Ejercicio 9. Escribe un programa que escriba los bytes del 0 al 255 en un archivo llamado output.bin.

Solución:

```
import java.io.FileOutputStream;
import java.io.IOException;

public class Main {
    public static void main(String[] args) {
        try (FileOutputStream fos = new FileOutputStream("output.bin")) {
            for (int i = 0; i <= 255; i++) {
                fos.write(i);
            }
            System.out.println("Bytes escritos en output.bin");
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Ejercicio 10. Escribe un programa que escriba en un fichero los datos de un trabajador, Nombre, Edad Y Cargo. Después el programa leerá esos datos y los mostrará por pantalla.

Solución:

```
import java.io.FileWriter;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class Main {

   public static void main(String[] args) {
      String archivo = "trabajador.txt"; // Nombre del archivo

      // Escribir los datos del trabajador en el archivo
      try (FileWriter fw = new FileWriter(archivo)) {
         fw.write("Nombre: Juan Perez\n");
         fw.write("Edad: 35\n");
         fw.write("Cargo: Ingeniero de Software\n");
      } catch (IOException e) {
         e.printStackTrace();
      }



      // Leer los datos del archivo y mostrarlos en pantalla
```

```
 try (BufferedReader br = new BufferedReader(new
FileReader(archivo))) {
        String linea;
        while ((linea = br.readLine()) != null) {
           System.out.println(linea); // Mostrar cada línea en pantalla
        }
     } catch (IOException e) {
        e.printStackTrace();
     }
   }
}
```

Capítulo 15.

Burlar el sistema de contadores de tiempo

El pasillo que da a la salida de la prisión es el más vigilado. Podrías pensar que es por miedo a que los presos escapen por ahí, pero no es así. Lo que ocurre es que todo el mundo entra y sale por el mismo lugar. Tanto visitas como funcionarios y presos siguen siempre el mismo camino.

Cada vez que alguien accede a la cárcel, se le hace un cacheo y las imágenes quedan grabadas. Además, hay que pasar por tres puntos de registro, divididos en varias estancias. En cada parada hay un guardia que no deja su puesto hasta que llega un relevo.

Al final del pasillo hay dos puertas. Una lleva a las celdas y la otra a una sala de espera en la que, a su vez, encontramos otras puertas. Desde ahí se puede acceder a otras partes de la prisión, pero no nos interesan ya que no necesitamos poner un pie en ellas.

Sabiendo que hay al menos cinco guardias custodiando la salida en todo momento, parece poco sensato intentar utilizar esa vía. Sin embargo, no todo es como parece, ya que durante la noche no hay visitas y, después del cambio de turno a las 22:00, ya no entra nadie. Eso quiere decir que no necesitan tener a cinco personas custodiando la entrada.

Bud cree que, como mucho, habrá dos personas. Incluso es posible que en algún momento necesiten realizar alguna tarea durante su turno y haya un corto período de tiempo en el que podamos estar solos.

El plan es identificar, medir y analizar las actividades de los carceleros para saber exactamente a qué hora debemos intentar la huida. Para ello, lo primero será analizar cuándo se abren y cierran las puertas. De esa forma, podremos identificar cuándo hay movimiento. Crearemos un programa que calcule el tiempo de inactividad de la puerta que conduce a la sala de espera. La puerta que da a las celdas podemos controlarla visualmente, y por la puerta de la calle no hay que preocuparse, ya que nadie la usa durante la noche.

Tenemos una pequeña ventaja: con una de las claves conseguidas en el despacho de Vicente, Pedro ha logrado acceder a los siete registros de actividades de la puerta de la semana pasada, uno para cada día. Con esto, ya podemos hacernos una idea de qué día es el más tranquilo y las horas en las que los vigilantes entran y salen. Lo primero será determinar qué día escogemos. Para ello, necesitamos crear un programa que lea los períodos de actividad de las puertas y los sume. Elegiremos el día para escapar que tenga el menor tiempo total de actividad.

El archivo divide el día en minutos, y en cada línea nos informa de cuántos segundos estuvo activa la puerta en ese minuto. Para que te hagas una idea, tiene este formato:

```
0
0
0
0
60
60
12
0
0
```

Ejercicio 1. Registrar el Tiempo de actividad. Necesitamos registrar el tiempo de actividad de las puertas para elegir un día para nuestro escape. Para ello necesitamos escribir un programa que lea el tiempo de inactividad de un archivo y registre el total de tiempo de inactividad en otro archivo.

Instrucciones:

1. Utilizamos el archivo **actividad.txt** conseguido por Pedro, con tiempos de actividad en segundos (uno por línea).
2. Lee los tiempos de actividad del archivo.
3. Calcula el tiempo total de inactividad.
4. Escribe el tiempo total de inactividad en un archivo **total_actividad.txt**

Solución:

```
import java.io.*;
import java.util.*;
public class Main {
   public static void main(String[] args) {

      String archivoEntrada = "actividad.txt";

      String archivoSalida = "total_actividad.txt";

      try (BufferedReader br = new BufferedReader(new
FileReader(archivoEntrada));
          BufferedWriter bw = new BufferedWriter(new
FileWriter(archivoSalida))) {

         String linea;
         int totalActividad = 0;

         while ((linea = br.readLine()) != null) {
            totalActividad += Integer.parseInt(linea);
         }

         bw.write("Total tiempo de actividad: " + totalActividad + " segundos");



      } catch (IOException e) {

         e.printStackTrace();

      }
   }
}
```

El lunes es el día con menor actividad, así que ya sabemos cuándo debemos emprender la huida. Según el registro, desde el minuto 1322 hasta el 1438, es decir, desde las 22:02 hasta las 23:58, la puerta está inactiva. Luego, aproximadamente vuelve a activarse a las 2:00 y, posteriormente, cada hora hasta las 6:00, cuando se abre la puerta principal y se llena de policías. En cada ocasión, las puertas están activas y abiertas por menos de dos minutos.

Una vez que sabemos a qué horas se abren las puertas, necesitamos determinar si se abrieron para dejar entrar a alguien o quizás para que alguien saliera. Los funcionarios deben realizar una serie de tareas durante su guardia. Pedro ha conseguido acceso al registro de estas tareas nocturnas y las veces que los guardias las anotan como realizadas. Necesitamos crear un pequeño programa que registre esas actividades y las devuelva en un formato de texto. Sabemos que sus actividades principales son: patrullaje regular, inspección y mantenimiento, y limpieza y desinfección.

El archivo de registro luce así:

"iRKJi2KpMW", "ixpFHX0v8s", "rMxNXbeUxC", "bjO9IUsFGc", "LpTB2ffO8w", "S4WwZffjqB", "HuuSb3RH6l", "O8OmH1ApJx", "LpFKbFT7hU", "TZ9BmcroA7", "dyLWJCvbU6", "qNlWgK2wzD", "s6vEOgmfsY", "ccDtnysyec", "waITKPmWbY", "HH9N0Ufih4", "cGIet9lE4y", "MCwCJ6LNQP", "OfecGXWhqY", "SOd3Tco6mI", "r8QwQDbHyp", "4V6bJhJ20x", "nzkiWhmxVj", "0tT2nDToVa", "67NEMfhs2x", "W3HXrDI7SJ", "KgGoiFJfMN", "2WK7UZVmgX", "3KNx0fDu54", "Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6", "oNSsEfNwaP", "V4zx2r6XR8", "2dPXcKbZNQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1", "pXpGUMRsZq", "1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ", "EsWhUsnhLa", "hevhsUt0ux", "bYwU4Qy6Gl", "sCW32VXSfD", "mwCdwQ809e", "zyRwiYkfDk", "ko4o26MErP", "Lk1rspAJQK", "d5Y9IPuMk5", "yKj992kdnn", "Q1XsuVqUGr", "fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG", "30DYrnKWoA", "r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW", "yT2klJvDqN", "AbIqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7", "kUaN7Cnv96", "tAROxtCUiP", "aZEZynwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj", "ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo", "wA7iC3Fq1k", "NqTC2OpWGr", "IvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi", "zTS8EJE0hh", "7gYXpaaPIs", "UiZ8tMlvfl", "uuSBzI6IfZ", "BdcQ3ci06N", "VNrQV6uZvU", "gZILASO2gT", "iOIRRKFxhc", "jkSGZFtIpw", "2gkQKh6J8L", "jnTyMxcG8l", "X07Mj6M0Rj", "SqjfH3cyLO", "GfKiA3DZ4w", "RBIquXXt0k", "udPC3Q0PXo", "CM44vkuwyT", "20n5Uwv6s8", "5ST85ixStb", "uliTvpFukE", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP", "ThjBwzns36", "8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj", "rotkzVlY3k", "Tah5BZU49N", "YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv", "OvKuo3mvSX", "htDbhBg7vs", "O8ZBRQB5cO", "TlUDfrzdUi", "7yIlltvZSl", "HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc", "a7LmXNTfap", "MlHEWHZhrz", "ADKHiZeBAS", "0KQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd", "BKs9UTKTQJ", "DBNrFCM5d7", "hse8XLelZK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW",

"iXvhHmq2pn", "gqnfV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs", "LSWyFeFyYv", "Patrullaje regular", "4Bpd1idsum", "vCgKs8GFTI", "uhUIpi1VLd", "AQn63M3ANF", "Y39NIKxIgJ", "YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij", "6Tb4opfNeC", "aTmzLU9opz", "QSHg6ZRmyH", "lVUz2kHAOn", "Inspección y mantenimiento", "wUxaxSm9lC", "DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w", "gQ6g0rJ7ax", "bejbKLd6ZR", "SMj8MjDEWY", "fanVKMvsmZ", "Patrullaje regular", "gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li", "SjalZaHsOA", "yxDLOhr4Od", "qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG", "QpYvFfSY6S", "I7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvgHjprUx", "SVlNByrKBN", "DNEmWPvl51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWTYz", "bpuMR5EVFB", "sMMP5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzI3gD", "L7pspbXgSw", "RvAtxqkYF2", "9gOPHDbw31", "uUDPlKRjU3", "NkPKW7zkZK", "dBOiqar26k", "GXPsNzuULq", "KwwpvvPqQ2", "Tif0qfs2eq", "3xTc242fPW", "c5jJz3DVrV", "zrK0VW0AQH", "zPJt2lGxKU", "C3ssWKqU5s", "XsfI0o5asg", "QiR4pDd4N0", "q4iY9xGn3V", "msQISGxfMJ", "yRyD1ec8fb", "dBfcrywOUD", "O9cMzNXF5t", "INuV5TSkbO", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"

Hay que tener en cuenta que, si no se introduce ninguna actividad durante cierto tiempo, se genera un código aleatorio.

Ejercicio 2. Generar un Log de Actividades. Bud necesita generar un log de actividades para analizar el comportamiento de los carceleros. Escribe un programa que registre las diferentes actividades del sistema en un archivo.

Instrucciones:

1. Crea un array con diferentes actividades ("Patrullaje regular", "Inspección y mantenimiento", " Limpieza y desinfección").

Escribe estas actividades en un archivo **log_actividades.txt**, una por línea.

Solución:

```
import java.io.*;
public class Main {
   public static void main(String[] args) {

      String[] actividades = {"iRKJi2KpMW", "ixpFHX0v8s", "rMxNXbeUxC", "bjO9IUsFGc",
"LpTB2ffO8w", "S4WwZffjqB", "HuuSb3RH6l", "O8OmH1ApJx", "LpFKbFT7hU",
"TZ9BmcroA7", "dyLWJCvbU6", "qNlWgK2wzD", "s6vEOgmfsY", "ccDtnysyec",
"waITKPmWbY", "HH9N0Ufih4", "cGIet9lE4y", "MCwCJ6LNQP", "OfecGXWhqY",
"SOd3Tco6mI", "r8QwQDbHyp", "4V6bJhJ20x", "nzkiWhmxVj", "0tT2nDToVa",
"67NEMfhs2x", "W3HXrDI7SJ", "KgGoiFJfMN", "2WK7UZVmgX", "3KNx0fDu54",
"Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6", "oNSsEfNwaP", "V4zx2r6XR8",
"2dPXcKbZNQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1", "pXpGUMRsZq",
"1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ",
"EsWhUsnhLa", "hevhsUt0ux", "bYwU4Qy6Gl", "sCW32VXSfD", "mwCdwQ809e",
"zyRwiYkfDk", "ko4o26MErP", "Lk1rspAJQK", "d5Y9IPuMk5", "yKj992kdnn",
"Q1XsuVqUGr", "fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG",
"30DYrnKWoA", "r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW",
"yT2kIJvDqN", "AbIqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7",
"kUaN7Cnv96", "tAROxtCUiP", "aZEZynwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj",
"ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo", "wA7iC3Fq1k", "NqTC2OpWGr",
"IvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi", "zTS8EJE0hh", "7gYXpaaPIs",
"UiZ8tMlvfl", "uuSBzI6IfZ", "BdcQ3ci06N", "VNrQV6uZvU", "gZILASO2gT", "iOIRRKFxhc",
"jkSGZFtIpw", "2gkQKh6J8L", "jnTyMxcG8I", "X07Mj6M0Rj", "SqjfH3cyLO",
"GfKiA3DZ4w", "RBIquXXt0k", "udPC3Q0PXo", "CM44vkuwyT", "20n5Uwv6s8",
"5ST85ixStb", "uliTvpFukE", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP",
"ThjBwzns36", "8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj",
"rotkzVlY3k", "Tah5BZU49N", "YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv",
"OvKuo3mvSX", "htDbhBg7vs", "O8ZBRQB5cO", "TlUDfrzdUi", "7yIIltvZSI",
"HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc", "a7LmXNTfap", "MlHEWHZhrz",
"ADKHiZeBAS", "0KQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd", "BKs9UTKTQJ",
"DBNrFCM5d7", "hse8XLelZK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW",
"iXvhHmq2pn", "gqnfV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs",
"LSWyFeFyYv", "Patrullaje regular", "4Bpd1idsum", "vCgKs8GFTI", "uhUIpi1VLd",
"AQn63M3ANF", "Y39NIKxIgJ", "YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij",
"6Tb4opfNeC", "aTmzLU9opz", "QSHg6ZRmyH", "lVUz2kHAOn", "Inspección y
mantenimiento", "wUxaxSm9lC", "DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w",
"gQ6g0rJ7ax", "bejbKLd6ZR", "SMj8MjDEWY", "fanVKMvsmZ", "Patrullaje regular",
"gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li", "SjalZaHsOA", "yxDLOhr4Od",
"qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG",
```

```
"QpYvFfSY6S", "I7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvgHjprUx",
"SVlNByrKBN", "DNEmWPvl51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWTYz",
"bpuMR5EVFB", "sMMP5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzI3gD",
"L7pspbXgSw", "RvAtxqkYF2", "9gOPHDbw31", "uUDPlKRjU3", "NkPKW7zkZK",
"dBOiqar26k", "GXPsNzuULq", "KwwpvvPqQ2", "Tif0qfs2eq", "3xTc242fPW",
"c5jJz3DVrV", "zrK0VW0AQH", "zPJt2lGxKU", "C3ssWKqU5s", "XsfI0o5asg",
"QiR4pDd4N0", "q4iY9xGn3V", "msQlSGxfMJ", "yRyD1ec8fb", "dBfcrywOUD",
"O9cMzNXF5t", "lNuV5TSkbO", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"};


    String archivoSalida = "log_actividades.txt";

    try (BufferedWriter bw = new BufferedWriter(new FileWriter(archivoSalida))) {

      for (String actividad : actividades) {
        bw.write(actividad);
        bw.newLine();
      }

    } catch (IOException e) {

      e.printStackTrace();

    }
  }
}
```

No todas las tareas se realizan todos los días. Nuestro plan es escapar un lunes, así que crearemos un programa que nos muestre cuántas veces se realiza cada tarea al día. Tenemos que encontrar alguna que no se realice en absoluto.

Cada tarea se lleva a cabo siempre a la misma hora, independientemente del día. Por ejemplo, la inspección y el mantenimiento se realizan siempre de 2:00 a 3:00. Por tanto, si el lunes esta actividad no se repitiera en absoluto, significaría que, durante ese periodo, los guardias no tendrían nada que hacer y posiblemente estén descansando. Sabemos a qué hora, en teoría, se realiza cada una de las tareas. Ahora solo nos queda verificar si el lunes hay alguna que no se realice.

Ejercicio 3. Necesitamos analizar el log de actividades para identificar patrones repetitivos durante la noche. Si alguno de ellos no se repite durante la noche que pretendemos escapar, significa que puede que el pasillo esté vacío.

Instrucciones:

1. Lee el archivo **log_actividades.txt**.
2. Cuenta la cantidad de veces que cada actividad ocurre.
3. Escribe los resultados en un archivo **resumen_actividades.txt**.

Solución:

```
import java.io.*;
import java.util.*;

public class Main {
   public static void main(String[] args) {

      String archivoEntrada = "log_actividades.txt";
      String archivoSalida = "resumen_actividades.txt";

      Map<String, Integer> contadorActividades = new HashMap<>();

      try (BufferedReader br = new BufferedReader(new
FileReader(archivoEntrada));
           BufferedWriter bw = new BufferedWriter(new
FileWriter(archivoSalida))) {
```

```
            String linea;
            while ((linea = br.readLine()) != null) {
                contadorActividades.put(linea,
contadorActividades.getOrDefault(linea, 0) + 1);
            }

            for (Map.Entry<String, Integer> entry :
contadorActividades.entrySet()) {
                bw.write(entry.getKey() + ": " + entry.getValue());
                bw.newLine();
            }

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Hemos revisado el registro y los resultados son mejores de lo que esperábamos. Según el análisis, la actividad de 'Limpieza y desinfección' no se realiza los lunes. Esta tarea se lleva a cabo entre las 3 y las 4 de la mañana. Ahora sabemos que durante esa hora el pasillo permanece inactivo, sin ningún tipo de actividad.

Con tanto tiempo disponible, no tendremos que preocuparnos por nada. Será pan comido. Sin embargo, debemos estar completamente seguros de que no habrá sorpresas ni imprevistos.

Bud y Pedro deberían estar satisfechos, pero están demasiado serios. Ante estas buenas noticias, yo me siento optimista. Estamos a solo un paso de escapar. No obstante, creo que Bud desconfía. Algo se nos escapa, y necesitaremos ayuda externa para descubrirlo.

Por suerte, conocemos a un guardia que no puede negarse a colaborar. Phil nos ha informado de que recientemente se han implementado dos actividades más: 'Monitoreo de cámaras' y 'Comunicación con el puesto de control'. Debemos actualizar el registro para verificar si, durante el periodo de tiempo que creíamos libre, los guardias en realidad están en el pasillo realizando alguna de estas dos tareas.

Ejercicio 4. Actualizar el registro de Actividades. Necesitamos actualizar el log de actividades en tiempo real para reflejar cambios en el sistema. Escribe un programa que agregue nuevas actividades al archivo de log de actividades.

El nuevo log que hemos encontrado es el siguiente:

"iRKJi2KpMW", "ixpFHX0v8s", "rMxNXbeUxC", "bjO9IUsFGc", "LpTB2ffO8w", "S4WwZffjqB", "HuuSb3RH6l", "O8OmH1ApJx", "LpFKbFT7hU", "TZ9BmcroA7", "dyLWJCvbU6", "qNlWgK2wzD", "s6vEOgmfsY", "ccDtnysyec", "waITKPmWbY", "HH9N0Ufih4", "cGIet9lE4y", "MCwCJ6LNQP", "OfecGXWhqY", "SOd3Tco6mI", "r8QwQDbHyp", "4V6bJhJ20x", "nzkiWhmxVj", "0tT2nDToVa", "67NEMfhs2x", "W3HXrDI7SJ", "KgGoiFJfMN", "2WK7UZVmgX", "3KNx0fDu54", "Rn20v6baaz", "t8jChEgFSz", "489dXWEFi6", "oNSsEfNwaP", "V4zx2r6XR8", "2dPXcKbZNQ", "WpsxQvEbVw", "7f6floF3VU", "W1joPNEiK1", "pXpGUMRsZq", "1M1jDBpiGu", "xjZBVieX1Q", "y3dVBuhTrJ", "s0YB6CVSFV", "vZQgDge2HJ", "EsWhUsnhLa", "hevhsUt0ux", "bYwU4Qy6Gl", "sCW32VXSfD", "mwCdwQ809e", "zyRwiYkfDk", "ko4o26MErP", "Lk1rspAJQK", "d5Y9IPuMk5", "yKj992kdnn", "Q1XsuVqUGr", "fHMDmXiAcp", "dql0nmQJcm", "qRqSHARwSA", "CgV8ne6fRG", "30DYrnKWoA", "r1QrdR0es2", "HXTV8XvYoP", "GQAWwyhMSL", "4yMmh7e5NW", "yT2klJvDqN", "AbIqpbKj5h", "haAriF42gR", "O6MciwEwQ4", "hW8Pf9IkX7", "kUaN7Cnv96", "tAROxtCUiP", "aZEZynwwpj", "cqzR5Y3Ipi", "M69Qr3BWJj", "ySTegpfDxw", "f47AKb3Bsn", "XOsmCGvcSo", "wA7iC3Fq1k", "NqTC2OpWGr", "IvvkjHR0Q2", "JyT5lu4QPo", "2whhr0ixDg", "xcYX1ELvDi", "zTS8EJE0hh", "7gYXpaaPIs", "UiZ8tMlvfl", "uuSBzI6IfZ", "BdcQ3ci06N", "VNrQV6uZvU", "gZILASO2gT", "iOIRRKFxhc", "jkSGZFtIpw", "2gkQKh6J8L", "jnTyMxcG8I", "X07Mj6M0Rj", "SqjfH3cyLO", "GfKiA3DZ4w", "RBIquXXt0k", "udPC3Q0PXo", "CM44vkuwyT", "20n5Uwv6s8", "5ST85ixStb", "uliTvpFukE", "ErQPLyDuRw", "XX5eaAXr1a", "nLenQQgqoP", "ThjBwzns36", "8RghH8Vneq", "FWFaHu0c2a", "HqGai14Ao6", "nFEi87Bafj", "rotkzVlY3k", "Tah5BZU49N", "YILUBnOTTh", "kyhPp7TWZp", "q1kiVAMsHv", "OvKuo3mvSX", "htDbhBg7vs", "O8ZBRQB5cO", "TlUDfrzdUi", "7yIlltvZSl", "HkAkB7qO3c", "j4DvSZL3Ry", "ONmRHuWdGc", "a7LmXNTfap", "MlHEWHZhrz", "ADKHiZeBAS", "0KQ6aXlsZY", "OKMfZsps6e", "vJpOxdfzEd", "BKs9UTKTQJ", "DBNrFCM5d7", "hse8XLelZK", "3jz3bLreNc", "ZZSHQ35FpT", "jOmaC6iMOW", "iXvhHmq2pn", "gqnfV11and", "DDfLkv3uBg", "UQjsb1ixF0", "08zSNrhvxs", "LSWyFeFyYv", "Patrullaje regular", "4Bpd1idsum", "vCgKs8GFTI", "uhUIpi1VLd", "AQn63M3ANF", "Y39NIKxIgJ", "YcpNNmT3GX", "C8Di70GqS7", "Ybg4P3BOij", "6Tb4opfNeC", "aTmzLU9opz",

"QSHg6ZRmyH", "lVUz2kHAOn", "Inspección y mantenimiento", "wUxaxSm9lC", "DVTb4JEZQ9", "sjahqvnVsk", "TfvPongU7w", "gQ6g0rJ7ax", "bejbKLd6ZR", "SMj8MjDEWY", "fanVKMvsmZ", "Patrullaje regular", "gmqX2oBxvm", "Mb9B4SHLvM", "64rh8hh6Li", "SjalZaHsOA", "yxDLOhr4Od", "qfN1GbZNFA", "C4q5ridJ7I", "aQLm7SJeEo", "LNDWA1UHzG", "QpYvFfSY6S", "l7ySXRZkQH", "WwtkFmM7pP", "gl794m6lyl", "6fvgHjprUx", "SVlNByrKBN", "DNEmWPvl51", "Mm6NxxNkdF", "gyLw8TP8kH", "vcqjXfWTYz", "bpuMR5EVFB", "sMMP5HpxJg", "9hU32sjepp", "RsYnv80Hx2", "SilPvzI3gD", "L7pspbXgSw", "RvAtxqkYF2", "9gOPHDbw31", "uUDPlKRjU3", "NkPKW7zkZK", "dBOiqar26k", "GXPsNzuULq", "KwwpvvPqQ2", "Tif0qfs2eq", "3xTc242fPW", "c5jJz3DVrV", "zrK0VW0AQH", "zPJt2lGxKU", "C3ssWKqU5s", "XsfI0o5asg", "QiR4pDd4N0", "q4iY9xGn3V", "msQISGxfMJ", "yRyD1ec8fb", "dBfcrywOUD", "O9cMzNXF5t", "INuV5TSkbO", "YEgbviK7AV", "HKrrLQBtt2", "ZU0hqtXuNa"

Instrucciones:

Usa un array para simular nuevas actividades que se agregan al sistema: "Monitoreo de cámaras", "Comunicación con el puesto de control"

1. Usa un array para simular nuevas actividades que se agregan al sistema, " Monitoreo de cámaras", " Comunicación con el puesto de control".
2. Agrega estas actividades al final del archivo **log_actividades.txt**.

Solución:

```
import java.io.*;

public class Main {
   public static void main(String[] args) {

      String[] nuevasActividades = {"6UX5w1vjQJ", "KAB1GksspV", "ziuAVNowpG", "uG0BEclHuq", "0xgydgyaK7", "IkSjUcqTbg", "7ZzjJYTCID", "7oGiZxDCIP", "9BRePDOokd", "SLR5QqI1gm", "dzHGx84mQH", "tTP235qyws", "Fk7qbJXHUN", "b82VQ4j1R5", "B2I4iZ096c", "yI6ntrP2cz", "y4rC1OQMJg", "WIjc2dUrde", "OYukZtwS1L", "qkwGHmjPao", "HycTVJeLB4", "ZaUckOVqbT", "qODG1FAhVE", "erLTxdRe61", "qZ1NgPNFt9", "bUifZB8w4s", "Monitoreo de cámaras", "zCrzN1ZDUg",
```

```
"gzCzIGm23u", "cDFm0hkrFw", "IDI0VBjizw", "klhN9ZnUyu", "GuscO6H64v",
"DJrhWpPTqr", "zM3MWg5rSj", "DMYzD7vkHF", "BNghDoM55o", "yr89K38y7l",
"H2qEcM6elW", "iw6OVfFJDh", "yj16xbky6M", "lFcJ6bQs54", "CNSJhoARCw",
"jOxs8ry2fx", "INZj1DoLqA", "GSEGuaChrR", "jrE68aipWX", "h8doJ65vcr",
"2EoBD02CKL", "C1av3xn5eG", "R8w7lp6uKQ", "sr1E5Nf6cz", "6C5Eff5czg",
"WHVPxvyciZ", "KlO6H5Vduk", "x6NIpTkJBK", "3YvHL9eQQ0", "qYYtIPWjpv",
"UZKlgaZ4TD", "msiexEsxj9", "RzoCEriGyZ", "vLuY3gwLJO", "d45CH4k7cs",
"rGdyVlst2Q", "3XRVIfo9mc", "eWUlCZ8UOo", "Monitoreo de cámaras",
"ic7SdX8di8", "C4qAoBxA3D", "Comunicación con el puesto de control",
"VBxKrQngfS", "fICz9PFnDX", "dk2ncCGc8x", "QG1MBYgT9i", "HAJkKvCVwT",
"gYoMT0TLOu", "vaNsJoXLnd", "TpBnt4Ieav", "TduiTd43zP", "Qm2zhvHZXB",
"n7BOCacWYA", "VY4N9FrFPB", "TACDi2dNQY", "VW4PHE8ZDb",
"SDYMTOCNJc", "Jnt9v9dgKy", "pY0yYILavE", "G9ERpaXMNl", "qDkqFSu4Sr",
"O3VyiIOh6E", "5qB9sTCmKH", "g5MyR0lXeu", "egzksA1lg2", "h6uLmm8tbp",
"Fx7ay5mNkD", "neDWml2spW", "IwkPB704pi", "7WqwiNEBrv",
"9ed11mNwh7", "kf3qNbMBkS", "MYFOZ3hHC8", "t4I0RK0M8O", "tkvxwGzdjg",
"8YsZ8xVw0m", "EdFd9zOWF5", "Te6y3Oa5XP", "9sg2XIQxbQ", "kJhouzOXkS",
"128laUqDWt", "nJDSEaneBg", "2rGSBCuzTd", "a5GbJeDTDh", "WVvQGSEkjL",
"Q26giaMbeM", "ArdXbDImAh", "Ll3nyJ5PRS", "WjZL4th5V5", "fTtNnPheOS",
"73rwDzESwL", "8eXQ7OW9gv", "SMgcsFvILx", "wRKnDVc55v", "UWC1QxcVZc"};
    String archivoSalida = "log_actividades.txt";

    try (BufferedWriter bw = new BufferedWriter(new
FileWriter(archivoSalida, true))) {

      for (String actividad : nuevasActividades) {
        bw.write(actividad);
        bw.newLine();
      }

    } catch (IOException e) {

      e.printStackTrace();

    }
  }
}
```

Ahora utilizaremos de nuevo el programa creado en el ejercicio tres para comprobar si alguna de las nuevas actividades se realiza los lunes.

Las noticias no son buenas. Parece que, en el momento en que pensábamos que el pasillo estaba vacío, en realidad hay dos vigilantes realizando tareas de monitoreo y comunicación con el puesto de control.

La esperanza de escapar estaba puesta en el lunes. No hemos comprobado el resto de los días, ya que, según la actividad de las puertas, hay aún más movimiento de guardias. Solo nos queda una solución: debemos ingeniárnoslas para que "Monitoreo de cámaras" y "Comunicación con el puesto de control" no se realicen el próximo lunes.

La idea de Pedro es cambiar la lista de tareas del sistema y eliminar esas dos nuevas actividades. El problema es que ya están programadas y solo pueden modificarse si el director o el subdirector deciden rotar el orden de las mismas. No es algo que hagan todas las semanas, ni mucho menos; de hecho, a veces tardan años en hacerlo.

Sin embargo, si el sistema entra en modo de mantenimiento, se pueden modificar los ficheros, y podremos eliminar las tareas que no queremos que se realicen el lunes. Una buena razón para que el sistema entre en mantenimiento es desajustar el registro de actividad de las puertas; además, es algo muy fácil de llevar a cabo.

Recordemos que ese fichero registra minuto a minuto cuántos segundos estuvo activa la puerta. Lo que haremos será asignar un segundo más de actividad a cada minuto. En los minutos en que la puerta esté activa durante 60 segundos, pasará a mostrar 61 segundos. Esto generará un error, ya que el sistema detectará que un minuto solo puede tener como máximo 60 segundos de actividad.

Ejercicio 5. Manipular Tiempos de actividad de las puertas. Escribe un programa que lea los tiempos de actividad y les asigne de un archivo y los modifique.

Instrucciones:

1. Crea un archivo tiempos_registro.txt con tiempos de registro en milisegundos (uno por línea).
2. Lee los tiempos de registro del archivo.
3. Incrementa cada tiempo en una cantidad fija (1 segundo).
4. Escribe los tiempos modificados en un archivo tiempos_modificados.txt.

Solución:

```
import java.io.*;
public class Main {

   public static void main(String[] args) {

      String archivoEntrada = "actividad.txt ";
      String archivoSalida = "tiempos_modificados.txt";
      int incremento = 1;

      try (BufferedReader br = new BufferedReader(new FileReader(archivoEntrada));
          BufferedWriter bw = new BufferedWriter(new FileWriter(archivoSalida))) {

         String linea;
         while ((linea = br.readLine()) != null) {

            long tiempoOriginal = Long.parseLong(linea);
            long tiempoModificado = tiempoOriginal + incremento;
            bw.write(Long.toString(tiempoModificado));
            bw.newLine();

         }

      } catch (IOException e) {

         e.printStackTrace();

      }
   }
}
```

Como era de esperar, el programa detectó rápidamente el error y procedió a identificarlo y repararlo. Simplemente reinició el sistema, y los registros volvieron a la normalidad.

Logramos nuestro objetivo: durante ese periodo de mantenimiento y búsqueda de errores, se abrió una ventana temporal que permitía modificar todos los archivos. Pedro aprovechó ese momento para cambiar la lista de tareas programadas para el lunes por la lista que nosotros necesitábamos, la que deja el pasillo libre durante una hora.

Capítulo 16.

Los preparativos finales / Bucles for -each

Ya no queda nada, nos escapamos el próximo lunes. Hoy es martes, así que tenemos tiempo suficiente para preparar todo. Mañana vendrá el primo de Chani y le daré toda la información. Por otro lado, debo explicarle exactamente dónde nos encontraremos. También necesito decírselo a Phil, ya que será él quien me recoja para traerme de vuelta a la prisión sin que nadie se dé cuenta.

He tenido mucho tiempo para elegir la localización y, sin duda, creo que es el lugar perfecto. Una vez que escapemos, iremos a una casa propiedad de una amiga de Pedro. Allí será donde nos separaremos, ya que cada uno tiene su propio plan. Cerca de esa casa hay una pequeña biblioteca pública que casi nadie utiliza. Es posible reservar salas, así que Chani incluso podrá esconder de antemano algunos micrófonos para grabar la conversación.

Hay un pequeño aparcamiento cercano en el que siempre hay plazas libres. Allí es donde me reuniré con Phil. Está situado en un barrio algo marginal, así que es muy poco probable que haya cámaras. Aun así, nadie me estará buscando, ya que todos asumirán que sigo en prisión. No obstante, me gusta ser precavido, algo que he aprendido de Bud y Pedro.

Es importante que en la grabación en la que se incrimine a Rich no aparezca mi voz. De lo contrario, las autoridades se enterarían de que salí de la cárcel y me vería en apuros. Chani podría eliminar las partes en las que salgo, pero un audio editado perdería veracidad. Por lo tanto, una vez que entre en la sala de la biblioteca, deberé permanecer callado. Para que no resulte extraño, es vital que salude a Rich en el exterior de la biblioteca. Podré mantener una conversación totalmente natural y, después, ya dentro de la sala, dejar que ellos dos conversen.

Tengo todo bien atado, pero espero que no haya muchos imprevistos. La verdad es que me preocupa no poder controlar lo que pasa fuera de la prisión. Aquí es más sencillo y, además, tengo ayuda. Fuera estaré solo; contaré con el apoyo de Chani y de Phil, pero no es lo mismo. No me fío de ellos tanto como lo hago de mis compañeros presos.

Esta noche continuaré con mis clases de Java. Estoy seguro de que sacaré una buena nota en mi próximo examen de la universidad. Parece que vamos a tratar un tipo de bucle o loop que no vimos cuando abordamos el tema anteriormente. La verdad es que me gustaron bastante los ejercicios de bucles, así que imagino que este tema también será entretenido y sencillo de entender.

Ventajas de los bucles for each

Hace ya algunos temas, tratamos el concepto de bucle y trabajamos en profundidad los bucles for y while. Ahora nos vamos a centrar en otro tipo, que, en su momento, dejé de lado adrede. Incluso centrándonos simplemente en dos, es un concepto algo denso y un tema complicado. Pero tu nivel de programación ya es mucho más alto así que puedes con esto y más.

Los bucles for-each son una herramienta fundamental debido a su simplicidad y eficiencia a la hora de iterar sobre colecciones y arrays. Las principales ventajas de usar este tipo de bucles son las siguientes:

- **Simplicidad y mejor legibilidad:** La sintaxis es más clara y concisa que la de los bucles tradicionales (for o while). Esto hace que el código sea más sencillo de entender.

- **Reducción de errores:** Al eliminar la necesidad de manejar índices manualmente, se reduce el riesgo de errores como el desbordamiento de índices (acceder a una posición fuera del rango del array).

- **Conveniencia:** Es especialmente útil para iterar sobre colecciones de objetos, como listas y conjuntos, donde los índices no son tan relevantes.

Pero no todo son ventajas, sino utilizaríamos solamente este tipo de bucles, así que vamos a ver los inconvenientes y aquellas situaciones en las que será mejor no utilizarlos.

- **Limitación al trabajar con índices**. No se puede acceder al índice del elemento actual directamente. Si necesitas el índice, debes usar un bucle for tradicional.

- **No se puede modificar la colección:** Si necesitas agregar o eliminar elementos de la colección mientras iteras, un bucle for-each no es adecuado.

- **Solo iteración hacia adelante:** El bucle for-each solo permite la iteración hacia adelante, por lo que no puedes recorrer la colección en sentido inverso.

- **Limitaciones en Arrays multidimensionales:** Se debe básicamente a que en estos casos resulta algo más complicado y menos intuitivo usar for-each en comparación con los bucles tradicionales.

Sintaxis del bucle for-each

La sintaxis básica del bucle for-each en Java es la siguiente:

```
for (TipoElemento variableTemporal : array) {
   // Código que usa variableTemporal
}
```

La sintaxis es sencilla: primero vemos el tipo de dato y después la variable temporal. Básicamente, esta tomará un valor diferente cada vez que el bucle haga una iteración. Por último, ponemos el array que será recorrido. Veámoslo con un ejemplo con datos reales:

```
public class Main {
   public static void main(String[] args) {
      int[] numeros = {1, 2, 3, 4, 5};

      for (int numero : numeros) {
         System.out.println(numero);
      }
   }
}
```

En este ejemplo:

- int es el tipo de los elementos en el array “numeros”.
- “numero” es la variable temporal que toma el valor de cada elemento en “numeros”.
- “numeros” es el array que se está iterando.

Ahora vamos a ver un ejemplo con un array de texto:

```
public class Main {
   public static void main(String[] args) {
      String[] frutas = {"Manzana", "Naranja", "Plátano"};

      for (String fruta : frutas) {
         System.out.println(fruta);
      }
   }
}
```

En este ejemplo:

- String es el tipo de los elementos en el array frutas.
- fruta es la variable temporal que toma el valor de cada elemento en frutas.
- frutas es el array que se está iterando.

Como este tema ha sido muy corto y una continuación del tema de bucles, no vamos a hacer un resumen del mismo. Hacerlo sería simplemente repetir lo mismo otra vez. Cómo has podido observar, la sintaxis es muy sencilla. Aunque para que se te grabe en la cabeza, y ganes confianza, te recomiendo que hagas todos los ejercicios que te propongo a continuación.

Ejercicios del tema

Ejercicio 1. Recorre un array de enteros e imprime cada número en una nueva línea.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5};

      for (int numero : numeros) {
         System.out.println(numero);
      }

   }
}
```

Ejercicio 2. Recorre un array tipo String e imprime cada cadena en una nueva línea.

Solución:

```
public class Main {
   public static void main(String[] args) {

      String[] palabras = {"Hola", "Mundo", "Java", "Programación"};


      for (String palabra : palabras) {
         System.out.println(palabra);
      }

   }
}
```

Ejercicio 3. Suma todos los elementos de un array de enteros e imprime el resultado.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5};
      int suma = 0;

      for (int numero : numeros) {

         suma += numero;

      }

      System.out.println("Suma: " + suma);

   }
}
```

Ejercicio 4. Encuentra el valor máximo en un array de enteros e imprime el resultado.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5};

      int max = numeros[0];
      for (int numero : numeros) {

         if (numero > max) {
            max = numero;
         }

      }

      System.out.println("Máximo: " + max);

   }
}
```

Ejercicio 5. Cuenta cuántas veces aparece un determinado número en un array de enteros.

Solución:

```
public class Main {

   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 2, 4, 2, 5};
      int numeroABuscar = 2;
      int contador = 0;
      for (int numero : numeros) {

         if (numero == numeroABuscar) {
            contador++;
         }

      }

System.out.println("El número " + numeroABuscar + " aparece " +
contador + " veces.");

   }
}
```

Ejercicio 6. Imprime las posiciones de un determinado número en un array de enteros.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 2, 4, 2, 5};
      int numeroABuscar = 2;


      for (int i = 0; i < numeros.length; i++) {

         if (numeros[i] == numeroABuscar) {

          System.out.println("El número " + numeroABuscar + " está en la
          posición " + i);

         }

      }
   }
}
```

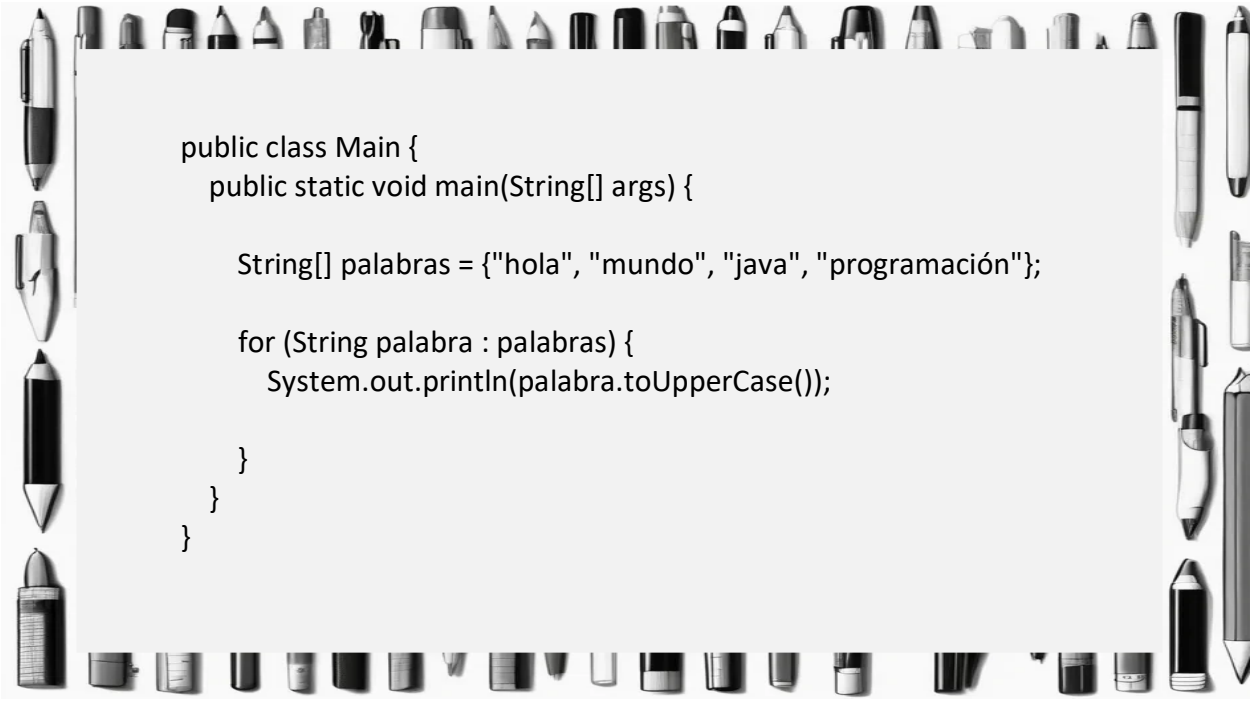
Ejercicio 7: Crea un array de enteros y calcula el promedio de sus elementos.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5};
      int suma = 0;

      for (int numero : numeros) {

         suma += numero;

      }

      double promedio = (double) suma / numeros.length;
      System.out.println("Promedio: " + promedio);

   }

}
```

Ejercicio 8. Recorre un array tipo string y convierte cada cadena a mayúsculas, imprimiéndolas después.

Solución:



```
public class Main {
   public static void main(String[] args) {

      String[] palabras = {"hola", "mundo", "java", "programación"};

      for (String palabra : palabras) {
         System.out.println(palabra.toUpperCase());

      }
   }
}
```

Como hemos visto, Para convertir texto en mayúsculas en Java, utilizamos el método toUpperCase() de la clase String. Este método toma una cadena de texto y devuelve una nueva cadena con todas las letras convertidas a mayúsculas. Aquí te explico cómo hacerlo de manera sencilla. Veamos otro ejemplo sencillo:

```
String texto = "hola mundo";

String textoEnMayusculas = texto.toUpperCase();

System.out.println(textoEnMayusculas);
```

Ejercicio 9. Recorre un array de enteros y multiplica cada elemento por 2, imprimiendo los resultados.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5};

      for (int numero : numeros) {
         System.out.println(numero * 2);
      }

   }
}
```

Ejercicio 10. Recorre un array tipo String e imprime solo los pares.

Solución:

```
public class Main {
   public static void main(String[] args) {

      int[] numeros = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

      int contadorPares = 0;

      for (int numero : numeros) {

         if (numero % 2 == 0) {
            contadorPares++;

         }
      }

      System.out.println("Número de elementos pares: " + contadorPares);

   }
}
```

Capítulo 17.

Decodificar sistemas de comunicación

Mis compañeros de huida me han puesto deberes. Creo que lo que pretenden es ayudarme a mantener la mente ocupada y que así esté un poco más tranquilo. Bud va a acceder al sistema de comunicaciones de los guardias y me ha pedido que le ayude.

Si alguien se entera de que hemos escapado antes de que yo pueda volver a la prisión, el plan habrá sido un completo fracaso, al menos para mí. Todo está bien calculado para que eso no ocurra, pero si nos topamos con algún carcelero, debemos evitar que dé la voz de alarma. Para ello, crearemos un cortafuegos.

Dentro de la cárcel, la situación será un poco distinta. Si nos descubre un guardia, no nos quedará más remedio que reducirlo y atarlo. Mientras no pueda pedir refuerzos, el plan seguirá en marcha.

Bud y Pedro van a llevar el ordenador con ellos y, entre otras cosas, lo usarán para monitorear y bloquear las comunicaciones entre los guardias de la prisión. Por supuesto, no todas las comunicaciones, solo las que nos interesan.

Así que, aprovechando que acabo de aprender el bucle for-each, voy a hacer los cinco programas que me han pedido utilizando ese concepto. Por supuesto, antes de comenzar el plan, Bud y Pedro los van a revisar para asegurarse de que todo está bien.

Para el primer ejercicio, tendré que interceptar algunos de los mensajes, en concreto aquellos que mencionen uno de los tres sectores por los que vamos a pasar: el sector 6, el 12 y el 14.

Ejercicio 1. Interceptar Mensajes.

- Inicializa un array con tres elementos:

"Mensaje 12: sector 12"

"Mensaje 6: sector 6"

"Mensaje 14: sector 14"

- Utiliza un bucle for-each para imprimir el mensaje actual en la consola precedido por la cadena "Mensaje interceptado: ".

Solución:

```
public class Main {
    public static void main(String[] args) {
        String[] mensajes = {

            "Mensaje 12: sector 12",
            "Mensaje 6: sector 6",
            "Mensaje 14: sector 14"

        };
```

```
        for (String mensaje : mensajes) {
            System.out.println("Mensaje interceptado: " + mensaje);
        }
    }
}
```

En caso de que se cuele algún mensaje de otro sector, deberemos hacer algo para que este los filtre y evite falsos positivos. Recordemos que para nosotros solo son importantes los mensajes que se refieran a los sectores 6, 12 y 14.

Ejercicio 2. Filtrar mensajes.

- Inicializa un array con catorce elementos:

  "Mensaje 1: Alerta en el sector 1",
  "Mensaje 2: Alerta en el sector 2",
  "Mensaje 3: Alerta en el sector 3",
  "Mensaje 4: Alerta en el sector 4",
  "Mensaje 5: Alerta en el sector 5",
  "Mensaje 6: Alerta en el sector 6",
  "Mensaje 7: Alerta en el sector 7",
  "Mensaje 8: Alerta en el sector 8",
  "Mensaje 9: Alerta en el sector 9",
  "Mensaje 10: Alerta en el sector 10",
  "Mensaje 11: Alerta en el sector 11",
  "Mensaje 12: Alerta en el sector 12",
  "Mensaje 13: Alerta en el sector 13",
  "Mensaje 14: Alerta en el sector 14"

- Utiliza un bucle for-each para filtrar y mostrar mensajes importantes:

Dentro del bucle for-each, verifica si el mensaje actual contiene el número "6", "12" o "14".

Si la condición se cumple, imprime el mensaje actual en la consola precedido por la cadena "Mensaje importante: ".

```
public class Main {
   public static void main(String[] args) {
      String[] mensajes = {

            "Mensaje 1: Alerta en el sector 1",
            "Mensaje 2: Alerta en el sector 2",
            "Mensaje 3: Alerta en el sector 3",
            "Mensaje 4: Alerta en el sector 4",
            "Mensaje 5: Alerta en el sector 5",
            "Mensaje 6: Alerta en el sector 6",
            "Mensaje 7: Alerta en el sector 7",
            "Mensaje 8: Alerta en el sector 8",
            "Mensaje 9: Alerta en el sector 9",
            "Mensaje 10: Alerta en el sector 10",
            "Mensaje 11: Alerta en el sector 11",
            "Mensaje 12: Alerta en el sector 12",
            "Mensaje 13: Alerta en el sector 13",
            "Mensaje 14: Alerta en el sector 14"

      };
```

```
for (String mensaje : mensajes) {

if (mensaje.contains("6") || mensaje.contains("12") || mensaje.contains("14") ) {

            System.out.println("Mensaje importante: " + mensaje);

        }
    }
  }
}
```

Una vez identificados y filtrados los mensajes que nos incomodan, es hora de bloquearlos. De esta manera evitaremos que se extiendan y alerten a todos los guardias de la prisión.

Ejercicio 3. Bloquear Comunicaciones

- Inicializa un array con tres elementos:

"Mensaje 12: sector 12"

"Mensaje 6: sector 6"

"Mensaje 14: sector 14"

- Utiliza un bucle for-each para mostrar mensajes bloqueados. Dentro del bucle for-each, imprime el mensaje actual en la consola precedido por la cadena "Bloqueando mensaje:".

Solución:

```
public class Main {
   public static void main(String[] args) {
      String[] mensajes = {
         "Mensaje 12: Alerta en el sector 12",
         "Mensaje 6: Alerta en el sector 6",
         "Mensaje 14: Alerta en el sector 14"
      };

      for (String mensaje : mensajes) {
         System.out.println("Bloqueando mensaje: " + mensaje);
      }
   }
}
```

Además de bloquearlos, necesitamos redirigirlos hacia nuestro programa principal. Bud y Pedro utilizan dos ordenadores distintos, y ambos necesitarán estar atentos para saber si algún guardia ha dado la voz de alarma.

Ejercicio 4. Redirigir comunicaciones.

- Inicializa el arreglo con tres elementos:

- "Mensaje 12: sector 12"
- "Mensaje 6: sector 6"
- "Mensaje 14: sector 14"

- Utiliza un bucle for-each para iterar sobre mostrar mensajes redirigidos:

Dentro del bucle for-each, imprime el mensaje actual en la consola precedido por la cadena "Redirigiendo mensaje: " y seguido por la cadena " a sistema externo".

Solución:

```
public class Main {
   public static void main(String[] args) {
      String[] mensajes = {

          "Mensaje 12: sector 12",
          "Mensaje 6: sector 6",
          "Mensaje 14: sector 14"
      };

for (String mensaje : mensajes) {
System.out.println("Redirigiendo mensaje: " + mensaje + " a sistema
externo");

      }
   }
}
```

Además de las comunicaciones entre los vigilantes, existe otra medida de seguridad que nos preocupa: el monitoreo de cámaras. En la prisión se utiliza un software de inteligencia artificial que analiza las imágenes y genera un mensaje por cada minuto. Tres de esos mensajes son inocuos, pero debemos crear un software que detecte el que nos incrimina. Después, Bud completará el software para que esas comunicaciones se bloqueen, tal como hicimos con los ejercicios anteriores. A continuación, te muestro los cuatro mensajes que genera el software de IA. Como imaginarás, el mensaje que dice 'Alerta' es el que debemos identificar y neutralizar.

Ejercicio 5. Monitoreo de Cámaras

- Inicializa el arreglo con cuatro elementos:

"Imagen 1: Pasillo vacío"

"Imagen 2: Alarma"

"Imagen 3: Puerta abierta"

"Imagen 4: Todo en orden"

- Utiliza un bucle for-each para iterar sobre cada elemento del arreglo imágenes. Detectar Actividad Sospechosa.

Dentro del bucle for-each, verifica si la imagen actual contiene las palabras "Puerta abierta" o "Guardia".

Si la condición se cumple, imprime el mensaje en la consola precedido por la cadena "Actividad sospechosa detectada: ".

Si no se cumple la condición, imprime el mensaje en la consola precedido por la cadena "Monitoreo normal: ".

Solución:

```
public class Main {
   public static void main(String[] args) {
      String[] imagenes = {

   "Imagen 1: Pasillo vacío",
"Imagen 2: Alarma",
"Imagen 3: Puerta abierta",
"Imagen 4: Todo en orden"

};
```

```
        for (String imagen : imagenes) {
            if (imagen.contains("Puerta abierta") || imagen.contains("Guardia")) {
                System.out.println("Actividad sospechosa detectada: " + imagen);
            } else {
                System.out.println("Monitoreo normal: " + imagen);
            }
        }
    }
}
```

Ya he cumplido con mi parte. Con estos programas, Bud y Pedro tienen todo lo que necesitan. A partir de ahora, me toca descansar y prepararme mentalmente para la última fase de la escapada. De momento, todo va perfecto y no ha habido ningún contratiempo. Espero que la buena racha continúe.

Capítulo 18.

Interviniendo las conversaciones de los guardias

Durante la última fase de nuestro plan, Rich y Pedro se mantendrán un poco más rezagados, y yo me adelantaré para encargarme de abrir la puerta. Tengo instrucciones muy claras, así que no creo que haya ningún problema.

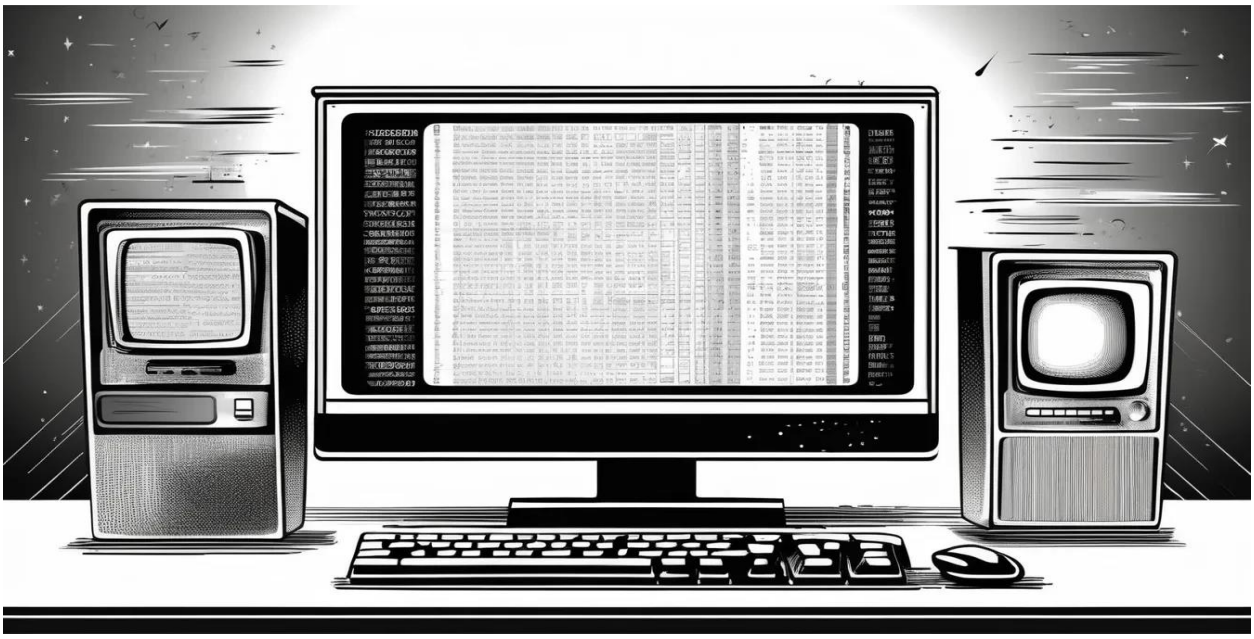
Evidentemente, Bud y Pedro tienen mucha más habilidad programando. En teoría, ellos deberían encargarse de abrir la puerta, pero se ocuparán de otra función igual de importante: controlar las comunicaciones de los guardias y vigilar las cámaras.

Durante la escapada, Pedro deberá acceder a uno de los puestos de control en el pasillo principal. Recuerda que estos puestos solo están ocupados durante el día. Por la noche no es necesario que haya nadie vigilando, así que la idea es utilizar el ordenador que hay ahí para monitorear todas las cámaras de la cárcel. De esta forma, sabremos si alguien se aproxima.

Para estar aún más seguros de que nadie nos descubre, Bud estará junto a mí y se encargará de dos cosas. Controlar la comunicación interna de los guardias y, al mismo tiempo, comunicarse con Pedro. De esta forma, si Pedro ve algo sospechoso, podrá avisar a Bud, y este a mí, para que podamos volver a nuestras celdas o escondernos.

Las cosas no siempre son sencillas, y en este caso todo se complica. El problema es que todas las comunicaciones de entrada y salida del ordenador de Bud están monitoreadas. No es un ordenador normal; recordemos que Bud es un preso; tiene muchas opciones restringidas y no se le permite comunicarse con personas del exterior.

Así que, para comunicarnos entre nosotros, no nos ha quedado más remedio que crear un sistema de mensajes codificados. La idea es sustituir cada letra por símbolos. De esta manera, los mensajes serán totalmente imposibles de leer si no se sabe descifrarlos. Para evitar aún más sospechas, solo enviaremos mensajes si es absolutamente necesario. Incluso si los guardias llegan a sospechar algo, para cuando se den cuenta ya habremos terminado.



Una vez que salgamos, tendremos aproximadamente tres horas hasta la revisión de habitaciones. Por lo tanto, debo darme prisa. Tardaré unos 30 minutos en llegar a la casa de la amiga de Pedro. Desde allí, necesitaré otros 15 minutos para llegar a la biblioteca. Después de la reunión con Chani y Rich, necesitaré otros 15 minutos para llegar al aparcamiento donde me encontraré con Phil. Luego, me tomará entre 15 y 30 minutos regresar a la cárcel y, posiblemente, otros 15 a 30 minutos adicionales para llegar a mi celda, ya que podría tener que esconderme y esperar a que no haya vigilantes.

Esto suma un total de entre 1 hora y 45 minutos y 2 horas. Considerando posibles contratiempos, calculo un margen de 15 minutos adicionales, lo que hace un total de 2 horas y 15 minutos. Por lo tanto, la reunión deberá durar aproximadamente media hora. El primo de Chani ya conoce mi itinerario y el tiempo que tengo, así que supongo que tendrá todo planeado en función de eso. Parece que todo está saliendo como habíamos planeado, así que estoy bastante tranquilo. Ahora vamos a revisar juntos que los programas de cifrar y descifrar mensajes funcionan.

Ejercicio 1. Crea un programa para codificar el texto introducido por teclado, reemplazando todas las letras minúsculas por símbolos específicos definidos. Funcionará en un bucle que procesará múltiples mensajes.

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      while (true) {
         System.out.println("Introduce el texto a codificar (o escribe 'salir' para terminar):");
         String texto = scanner.nextLine();
         if (texto.equalsIgnoreCase("salir")) {
            break;
         }

         String textoCodificado = texto.replace('a', '!')

                                 .replace('b', '@')
                                 .replace('c', '#')
                                 .replace('d', '$')
                                 .replace('e', '%')
                                 .replace('f', '^')
                                 .replace('g', '&')
                                 .replace('h', '*')
```

```
                .replace('i', '(')
                .replace('j', ')')
                .replace('k', '-')
                .replace('l', '_')
                .replace('m', '=')
                .replace('n', '+')
                .replace('o', '[')
                .replace('p', ']')
                .replace('q', '{')
                .replace('r', '}')
                .replace('s', ';')
                .replace('t', ':')
                .replace('u', '<')
                .replace('v', '>')
                .replace('w', ',')
                .replace('x', '.')
                .replace('y', '?')
                .replace('z', '/');

            System.out.println("Texto codificado: " + textoCodificado);
        }
    }
}
```

Ejercicio 2. Crea un programa para descodificar el texto previamente codificado, reemplazando los símbolos por las letras minúsculas originales. Funcionará en un bucle para procesar múltiples mensajes.

Solución:

```
import java.util.Scanner;

public class Main {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      while (true) {
         System.out.println("Introduce el texto a decodificar (o escribe 'salir' para terminar):");
         String textoCodificado = scanner.nextLine();
         if (textoCodificado.equalsIgnoreCase("salir")) {
            break;
         }

         String textoDecodificado = textoCodificado.replace('!', 'a')
                                    .replace('@', 'b')
                                    .replace('#', 'c')
                                    .replace('$', 'd')
                                    .replace('%', 'e')
                                    .replace('^', 'f')
                                    .replace('&', 'g')
                                    .replace('*', 'h')
                                    .replace('(', 'i')
                                    .replace(')', 'j')
                                    .replace('-', 'k')
                                    .replace('_', 'l')
                                    .replace('=', 'm')
                                    .replace('+', 'n')
                                    .replace('[', 'o')
                                    .replace(']', 'p')
                                    .replace('{', 'q')
                                    .replace('}', 'r')
                                    .replace(';', 's')
                                    .replace(':', 't')
                                    .replace('<', 'u')
                                    .replace('>', 'v')
                                    .replace(',', 'w')
                                    .replace('.', 'x')
                                    .replace('?', 'y')
                                    .replace('/', 'z');

         System.out.println("Texto decodificado: " + textoDecodificado);
      }
   }
}
```

Capítulo 19.

Funciones y POO

Antes de pasar a la fase final de nuestro plan, necesitas entender un par de cosas más. Me refiero a las funciones y los ejercicios de programación orientada a objetos. A estas alturas ya dominas la sintaxis, pero programas de una forma lineal. Solo hemos hecho programas dentro de la clase Main. En los programas reales es imposible encontrarnos con esa linealidad. Los programas reutilizan código y este se almacena en distintas clases. Después, desde la clase Main se van llamando y ejecutando dependiendo del orden en el que los necesitemos.

Vamos a aprender a reutilizar código y a trabajar con distintas clases; para ello, empezaremos con las funciones y los métodos. En Java, la terminología para "métodos" y "funciones" puede ser un poco confusa, especialmente si vienes de otros lenguajes de programación. En términos de Java:

Método: Es una función que está definida dentro de una clase. En Java, todas las funciones deben estar dentro de una clase, por lo que **todas las funciones en Java son métodos.** El término "método" es más común en Java.

Función: Es un término más general y se usa en muchos lenguajes de programación para referirse a un bloque de código que realiza una tarea específica. En Java, se refiere a lo mismo que un método, pero se utiliza con menos frecuencia.

Diferencia entre public, protected o private

public: Un método o variable que es declarado como public se puede acceder desde cualquier otra clase. No importa el paquete en el que se encuentre. Lo utilizamos cuando queremos que el método o la variable estén disponibles globalmente.

Cuando un método o variable está declarada como **private,** entonces solamente será accesible dentro de la misma clase en la que se define. Se utiliza para proteger los datos de un acceso externo.

Por último, un método o variable **protected** es accesible dentro del mismo paquete y por subclases, incluso si están en paquetes diferentes.

Diferencia entre métodos que devuelve un valor y métodos “void”

- **Métodos que devuelven un valor:** Estos métodos especifican un tipo de retorno (por ejemplo, int, String, boolean, etc.) en su declaración. Deben utilizar la palabra clave **return** para devolver un valor del tipo especificado. Son útiles cuando se necesita procesar datos y devolver un resultado.

- **Métodos void:** Estos métodos usan la palabra clave void para indicar que no devolverán ningún valor.

Reutilizando Métodos estáticos dentro de la clase Main

- Empezaremos con un ejemplo sencillo. La idea es crear unas líneas de código que sumen dos a un número. Escribiré el código y después lo analizaré poco a poco.

```
public class Main {

  public static void sumaDos(int numero) {
    int resultado = numero + 2;
    System.out.println("El resultado de sumar 2 a " + numero + " es: " + resultado);
  }
  public static void main(String[] args) {

  sumaDos(5);

  }
}
```

Vamos con la explicación. En negro está resaltado nuestro método. Pero, antes de nada, vemos la línea con la que se hace referencia a la clase en la que estamos trabajando. Como comenté en los primeros párrafos del tema, todos los métodos deben ir dentro de una clase; si no, el programa nos va a devolver un mensaje de error. En este caso, la clase Main.

Pero, por otro lado, va **encima** de el método principal de la clase:

public static void main(String[] args) {

}

Por lo tanto, esta clase tendrá dos métodos, pero solo se ejecuta el código que está dentro del principal. Por eso, si queremos utilizar un método secundario, debemos llamarlo desde el principal. En este ejemplo, hemos creado un método secundario, pero podría haber tantos como sea necesario.

Dentro del método principal, *public static void main(String[] args)*, vemos como se está llamando a al método sumaDos:

sumaDos(5);

Colocamos el número al que queremos que se le sume dos entre paréntesis. Tal como hemos creado el método, es obligatorio que introduzcamos un valor en el paréntesis, ya que lo hemos especificado en el código al escribir (int numero). Pero no siempre vamos a necesitar indicar un valor; eso nos lleva al siguiente ejemplo.

- Ahora vamos a crear el mismo ejemplo, pero queremos que, cuando llamemos al método, nos pida por teclado el número que queremos introducir y que después nos muestre cuál es la suma de esa cifra más dos.

```
import java.util.Scanner;
public class Main {

   public static void sumaDos() {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Introduce un número: ");
      int numero = scanner.nextInt();
      int resultado = numero + 2;
      System.out.println("El resultado de sumar 2 a " + numero + " es: " + resultado);
   }

   public static void main(String[] args) {

   sumaDos();

   }
}
```

Igual que antes, escribimos nuestro método dentro de la clase Main y antes del método principal. En este caso, hemos creado un código que nos pide que introduzcamos un número por teclado y el programa sumará 2 a este valor y, además, mostrará por pantalla el resultado.

- Imaginemos ahora que, cada cierto tiempo, tenemos que introducir un texto bastante largo en nuestro código. Podríamos escribirlo todas las veces, pero lo más elegante y ordenado es crear un método para reutilizar ese texto. Lo podríamos hacer tal y como te muestro a continuación.

```
public class Main {

   public static void texto() {

      System.out.println("A veces un texto puede ser demasiado largo cómo para que estemos utilizándolo una y otra vez");
      System.out.println("Es mejor crear un método y despues llamarlo cada vez que necesitemos que se muestre dicho texto.");
      System.out.println("Tu programa quedará mucho más limpio y ordenado");

   }

   public static void main(String[] args) {

   texto();

   }
}
```

En el código anterior, hemos visto un método que básicamente consiste en tres líneas de texto. En el momento y en cualquier lugar donde necesitemos que esas líneas aparezcan, simplemente tenemos que llamar a nuestro método y así evitaremos tener grandes textos mezclados con la sintaxis.

- Como último ejemplo (De momento) vamos a ver un método en el que pedimos dos números y nos devuelve la multiplicación de los mismos.

```
import java.util.Scanner;
public class Main {

   public static int producto(int a, int b) {

      int c = a*b;

      System.out.println("El producto de a * b = " + c);

   return c;

   }

   public static void main(String[] args) {

   producto(4,8);

   }
}
```

Con estos 4 ejemplos hemos visto 3 funciones void y la última que devuelve un valor. Por eso cuando la creamos en vez de static void utilizamos static int. Además si te has fijado en los tres ejemplos void, en el primero entre paréntesis metimos una variable: **public static void sumaDos(int numero).** En los otros dos no. Esto se debe a que en el primero queríamos escribir nosotros mismos la cifra manualmente. Sin embargo, en el segundo, el programa no necesitaba de ningún dato ya que iba a empezar a ejecutarse y a pedirnos el mismo que introdujéramos un dato por teclado. En cuanto al ejercicio del texto tampoco teníamos que especificar ningún dato, el método siempre muestra el mismo texto.

Reutilizando métodos creados en otras clases distintas de la clase Main

Lo primero que vamos a hacer es crear una **clase que llamaremos ejercicios.** Inicialmente tendrá este aspecto antes de que empecemos a escribir nuestro texto:

```
public class Ejercicios {

    // Aquí escribiremos nuestros métodos.

}
```

Nuestra clase Main será así:

```
public class Main {

   public static void main(String[] args) {

     // Aquí los vamos a llamar.

   }
}
```

Ahora que la tenemos creada vamos a ir ejemplo a ejemplo escribiendo los métodos dentro de ella y posteriormente llamándolos desde dentro del método Main de la clase Main.

- En el primer ejemplo vamos a crear un método en la clase Ejercicios que nos diga si un número es par o no lo es. El número lo introduciremos manualmente cuando llamemos al método desde la clase Main.

  Este es el código de nuestro método:

```
public class Ejercicios {

   public static boolean esPar(int numero) {

      if (numero % 2 == 0) {
         System.out.println("El número es par");
      } else System.out.println("El número es impar");

      return numero % 2 == 0;  // Retorna true si el número es par, false si es impar
   }

}
```

Hasta aquí no hay mucha pérdida, el ejercicio es sencillo. En lo que debes fijarte es en qué clase está y en el que dentro de ella hemos creado un método public static boolean. Ahora debemos llamarlo desde la clase Main.

Para ello seguimos siempre una estructura muy sencilla. Dentro de nuestra clase Main y dentro del método main escribimos el nombre de nuestra clase, después un punto y todo ello seguido del nombre del método y unos paréntesis. Si durante la declaración del método no hemos utilizado ninguna variable en los paréntesis, cuando llamemos al método desde el método main, los paréntesis estarán vacíos. Si durante la declaración utilizamos alguna variable, en los paréntesis pondremos el dato, o datos, que necesitemos. En este caso quedaría de la siguiente manera:

```
Ejercicios.esPar(6);
```

El código entero del método Main es el siguiente:

```
public class Main {

   public static void main(String[] args) {

    Ejercicios.esPar(6);

   }

}
```

- Ahora vamos a hacer el mismo ejercicio, pero en este caso tenemos que introducir por teclado el número, después el programa nos indicará si es par o no lo es.

En la clase Ejercicios es donde hacemos todo el trabajo. Así es cómo quedaría:

```
import java.util.Scanner;
public class Ejercicios {

   public static void esPar() {

      Scanner ejercicio = new Scanner(System.in);

      System.out.println("Introduce un número");
```

```
    int numero = ejercicio.nextInt();

     if (numero % 2 == 0) {
        System.out.println("El número es par");
     } else System.out.println("El número es impar");

  }

}
```

Vemos que el Scanner lo importamos en esta clase y no en la clase Main. Por lo demás es un código muy sencillito.

Para llamar al método desde la clase Main lo haremos de la siguiente forma:

```
public class Main {

  public static void main(String[] args) {

   Ejercicios.esPar();

  }
}
```

- Para acabar utilizamos otro método void de tipo String. El mismo que utilizamos al principio del tema.

La clase ejercicios será la siguiente:

```
public class Ejercicios {
   public static void texto() {

      System.out.println("A veces un texto puede ser demasiado largo como para que estemos utilizándolo una y otra vez");
      System.out.println("Es mejor crear un método y después llamarlo cada vez que necesitemos que se muestre dicho texto.");
      System.out.println("Tu programa quedará mucho más limpio y ordenado");

   }
}
```

Y esta será la clase Main:

```
public class Main {
   public static void main(String[] args) {

    Ejercicios.texto();

   }
}
```

Diferencia entre clase e instancia

Quiero explicarte qué es eso de static que aparece todo el tiempo en nuestros métodos, pero para ello antes debes entender la diferencia entre clase e instancia. Una clase, como ya habíamos comentado, es un molde o plantilla que define las propiedades (atributos) y comportamientos (métodos) comunes a todos los objetos que se crearán a partir de ella.

Una instancia es un objeto creado a partir de una clase que tiene valores específicos para los atributos definidos en la clase.

Diferencia entre métodos estáticos y dinámicos

Los métodos estáticos (Todos los que hemos estado utilizando hasta ahora) están asociados directamente a la clase, no a objetos específicos de esa clase. Esto significa que se pueden invocar sin necesidad de crear una instancia (objeto) de la clase.

Por ejemplo:

```
public class Calculadora {
   public static int sumar(int a, int b) {
      return a + b;
   }
}
```

// Para usar el método sumar, no necesitas crear un objeto. Podríamos llamarlo desde la clase Main así:

```
int resultado = Calculadora.sumar(5, 10);
```

Los métodos dinámicos pertenecen a instancias específicas de una clase. Cada objeto creado a partir de una clase tiene su propia copia de los métodos dinámicos.

```
public class Coche {
   public void acelerar() {
      System.out.println("El coche está acelerando.");
   }
}

// Para usar el método acelerar, necesitas crear una instancia (objeto)
Coche miCoche = new Coche();
miCoche.acelerar();
```

El concepto de objeto aún es algo desconocido para ti, pero vamos a poner remedio a ello. Sé que después de todo este tiempo programando de forma lineal, la utilización de métodos ha cambiado un poco tus esquemas y ahora, encima, se nos juntan las instancias. Pero la verdad es que es mucho más fácil de lo que parece. En lo que queda del tema, vamos a meternos de lleno en ello y pronto no te quedará ninguna duda al respecto.

Trabajando con instancias u objetos

¿Qué es un Objeto?

Un objeto es una entidad concreta que podemos usar en nuestros programas. Imagina que tienes un plano para construir una casa. Este plano sería la clase, que describe cómo será la casa, pero la casa real que construyes a partir del plano es el objeto. En programación, un objeto es una instancia de una clase y contiene atributos (características) y métodos (acciones).

Clases vs Objetos

- Clase: Es como un plano o receta. Describe qué características y comportamientos tendrán los objetos que se creen a partir de ella.

- Objeto: Es una instancia concreta creada a partir de la clase. Es la "cosa" real que podemos usar y manipular en nuestro programa.

¿Qué es un Constructor en Java?

Un constructor es un tipo especial de método que se usa para inicializar objetos. Es el primer método que se llama cuando creamos un objeto de una clase. Su propósito es dar valores iniciales a los atributos del objeto. Se crea dentro de la clase a la que pertenece ese objeto. Lo vamos a ver de forma mucho más clara en el ejemplo que te muestro en el siguiente punto.

Ejemplo Básico de una Clase y Objeto

- Creamos la clase:

```
public class Persona {
  String nombre;
  int edad;

  // Constructor
  public Persona(String nombre, int edad) {
    this.nombre = nombre;
    this.edad = edad;
  }
```

```
    // Método para mostrar la información de la persona

    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}
```

- Ahora vamos con el objeto

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de la clase Persona
        Persona persona1 = new Persona("Juan", 25);

        // Usar el objeto para llamar al método mostrarInformacion
        persona1.mostrarInformacion();
    }
}
```

Cómo puedes ver, en la clase persona tenemos dos métodos. El primero es el constructor. Gracias a él, después en la clase Main, vamos a poder crear objetos y asignarles valores. El siguiente método simplemente va a mostrar los valores por pantalla. Utiliza la información que nos proporciona el método constructor.

En la clase Main, creamos un objeto, que en este caso llamamos Persona1, y le asignamos los valores nombre y edad. Para este caso concreto, el nombre es Juan y la edad es 25 años. Una vez que nuestro objeto ha sido creado y tiene valores asignados, utilizamos el segundo método para mostrar la información por pantalla.

Definición de Métodos Getter

Propósito: Un método getter se usa para obtener el valor de un atributo privado de una clase. Es una forma segura de acceder a los datos sin permitir que se modifiquen directamente.

```
public class Persona {
   private String nombre;

   // Método getter para el atributo nombre
   public String getNombre() {
      return nombre;

   }
}
```

Definición de Métodos Setter

Propósito: Un método setter se usa para establecer o modificar el valor de un atributo privado de una clase. Permite controlar cómo se cambia el valor y puede incluir validaciones.

Sintaxis:

```
public class Persona {
   private String nombre;

   // Método setter para el atributo nombre
```

```
    public void setNombre(String nombre) {
        this.nombre = nombre;

    }
}
```

Ejemplo de Getters y Setters

Veamos un ejemplo completo con una clase Persona que tiene un atributo edad y usa métodos getter y setter para manejarlo.

```
public class Persona {
    private int edad;

    // Método getter para el atributo edad
    public int getEdad() {
        return edad;
    }

    // Método setter para el atributo edad

    public void setEdad(int edad) {

        if (edad > 0) {
            this.edad = edad;
        }
    }
}
```

```
public class Main {
   public static void main(String[] args) {


      Persona persona1 = new Persona();
      persona1.setEdad(25);  // Establecer la edad usando el setter
      System.out.println("Edad: " + persona1.getEdad());  // Obtener la edad usando el getter


   }
}
```

Esta vez, en la clase persona no hemos creado ningún constructor. Por lo tanto, si creáramos un objeto en la clase Main, nos sería imposible asignarles valor a sus atributos.

Pero no hay problema; en vez de un método constructor, utilizaremos un método Getter y uno Setter. Lo vamos a crear en la clase Persona y después lo llamaremos desde Main. En este caso, solo tenemos una variable, así que se crea un Getter y un Setter por cada variable. Ambos siguen siempre la misma estructura que puedes apreciar en el ejemplo.

En la clase Main, lo primero que hacemos es crear un objeto, de nuevo Persona1. Para llamar al Setter, ponemos el nombre de nuestro objeto, un punto, el nombre de nuestro Setter y entre paréntesis el valor que queremos asignar. Como puedes observar, para llamar al Getter, seguimos exactamente la misma estructura.

Ejercicios típicos de objetos utilizando constructores

Ejercicio 1

Crear y Mostrar Información de un Libro. Crea una clase Libro con atributos título y autor. Usa un constructor para inicializar estos atributos y un método para mostrar la información del libro.

Solución:

```
public class Libro {

    private String titulo;
    private String autor;

    // Constructor

    public Libro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }


    // Método para mostrar información del libro

    public void mostrarInformacion() {
        System.out.println("Título: " + titulo + ", Autor: " + autor);
    }

}
```

```
public class Main {

   public static void main(String[] args) {
      Libro libro1 = new Libro("Cien Años de Soledad", "Gabriel G. Márquez");
      libro1.mostrarInformacion();

   }
}
```

Ejercicio 2

Crea una clase Estudiante con atributos nombre y edad. Usa un constructor para inicializar estos atributos y un método para mostrar la información del estudiante.

Solución:

```
public class Estudiante {

   private String nombre;
   private int edad;

   // Constructor

   public Estudiante(String nombre, int edad) {
      this.nombre = nombre;
      this.edad = edad;
   }
```

```
    // Método para mostrar información del estudiante


    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }

}

public class Main {

    public static void main(String[] args) {
        Estudiante estudiante1 = new Estudiante("Ana", 20);
        estudiante1.mostrarInformacion();
    }

}
```

Ejercicio 3. Crea una clase Producto con atributos nombre y precio. Usa un constructor para inicializar estos atributos y un método para mostrar la información del producto.

Solución:

```
public class Producto {

    private String nombre;
    private double precio;


    // Constructor

    public Producto(String nombre, double precio) {
        this.nombre = nombre;
        this.precio = precio;
    }

    // Método para mostrar información del producto

    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Precio: $" + precio);
    }
}

public class Main {

    public static void main(String[] args) {
        Producto producto1 = new Producto("Laptop", 799.99);
        producto1.mostrarInformacion();
    }

}
```

Ejercicio 4. Crea una clase Coche con atributos marca y modelo. Usa un constructor para inicializar estos atributos y un método para mostrar la información del coche.

Solución:

```
public class Coche {

   private String marca;
   private String modelo;



   // Constructor

   public Coche(String marca, String modelo) {
      this.marca = marca;
      this.modelo = modelo;
   }

   // Método para mostrar información del coche

   public void mostrarInformacion() {
      System.out.println("Marca: " + marca + ", Modelo: " + modelo);
   }
}

public class Main {

   public static void main(String[] args) {
      Coche coche1 = new Coche("Toyota", "Corolla");
      coche1.mostrarInformacion();
   }

}
```

Ejercicio 5. Crea una clase Película con atributos título y director. Usa un constructor para inicializar estos atributos y un método para mostrar la información de la película.

Solución:

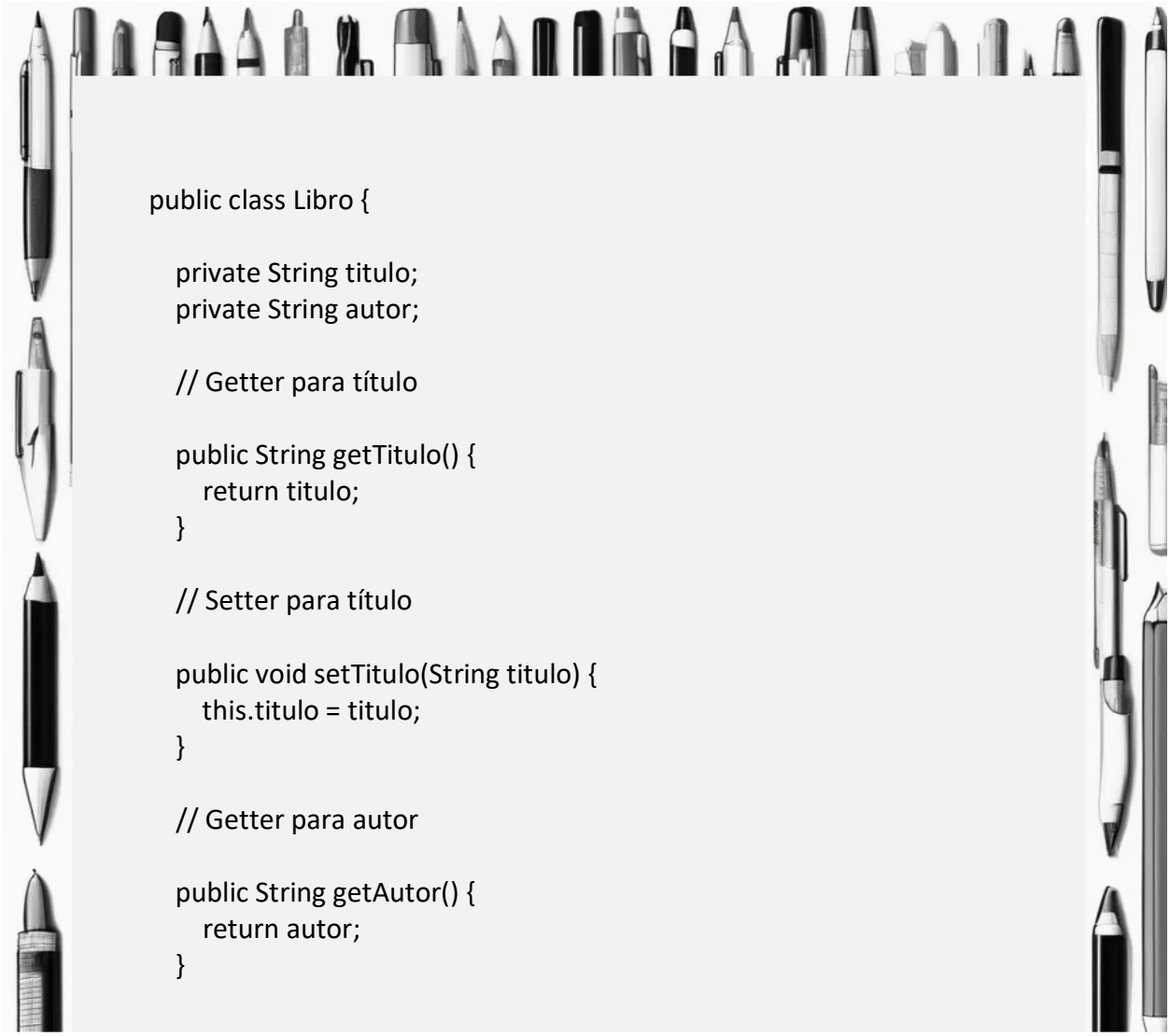
```
public class Pelicula {

    private String titulo;
    private String director;

    // Constructor

    public Pelicula(String titulo, String director) {
        this.titulo = titulo;
        this.director = director;
    }

    // Método para mostrar información de la película

    public void mostrarInformacion() {
        System.out.println("Título: " + titulo + ", Director: " + director);

    }
}

public class Main {

    public static void main(String[] args) {
        Pelicula pelicula1 = new Pelicula("Inception", "Christopher Nolan");
        pelicula1.mostrarInformacion();

    }
}
```

Ejercicios típicos de objetos utilizando Getter y Setter

Ejercicio 1

Crea una clase Libro con atributos título y autor. Usa métodos getter y setter para acceder y modificar estos atributos, y un método para mostrar la información del libro.

Solución:



```
public class Libro {

   private String titulo;
   private String autor;

   // Getter para título

   public String getTitulo() {
      return titulo;
   }

   // Setter para título

   public void setTitulo(String titulo) {
      this.titulo = titulo;
   }

   // Getter para autor

   public String getAutor() {
      return autor;
   }
```

```
// Setter para autor

   public void setAutor(String autor) {
      this.autor = autor;
   }

   // Método para mostrar información del libro

   public void mostrarInformacion() {
      System.out.println("Título: " + titulo + ", Autor: " + autor);
   }

}



public class Main {

   public static void main(String[] args) {
      Libro libro1 = new Libro();
      libro1.setTitulo("Cien Años de Soledad");
      libro1.setAutor("Gabriel García Márquez");
      libro1.mostrarInformacion();

   }
}
```

Ejercicio 2

Crea una clase Estudiante con atributos nombre y edad. Usa métodos getter y setter para acceder y modificar estos atributos, y un método para mostrar la información del estudiante.

Solución:

```
public class Estudiante {

    private String nombre;
    private int edad;

    // Getter para nombre

    public String getNombre() {
        return nombre;
    }


    // Setter para nombre

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Getter para edad

    public int getEdad() {
        return edad;
    }

    // Setter para edad

    public void setEdad(int edad) {
        this.edad = edad;
    }
```

```
    // Método para mostrar información del estudiante

    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}

public class Main {

    public static void main(String[] args) {
        Estudiante estudiante1 = new Estudiante();
        estudiante1.setNombre("Ana");
        estudiante1.setEdad(20);
        estudiante1.mostrarInformacion();
    }
}
```

Ejercicio 3

Crea una clase Producto con atributos nombre y precio. Usa métodos getter y setter para acceder y modificar estos atributos, y un método para mostrar la información del producto.

Solución:

```
public class Producto {

    private String nombre;
    private double precio;

    // Getter para nombre

    public String getNombre() {
        return nombre;
    }

    // Setter para nombre

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Getter para precio

    public double getPrecio() {
        return precio;
    }

    // Setter para precio

    public void setPrecio(double precio) {
        this.precio = precio;
    }
```

```
    // Método para mostrar información del producto


    public void mostrarInformacion() {
        System.out.println("Nombre: " + nombre + ", Precio: $" + precio);
    }
}



public class Main {

    public static void main(String[] args) {
        Producto producto1 = new Producto();
        producto1.setNombre("Laptop");
        producto1.setPrecio(799.99);
        producto1.mostrarInformacion();

    }
}
```

Ejercicio 4

Crea una clase Coche con atributos marca y modelo. Usa métodos getter y setter para acceder y modificar estos atributos, y un método para mostrar la información del coche.

Solución:

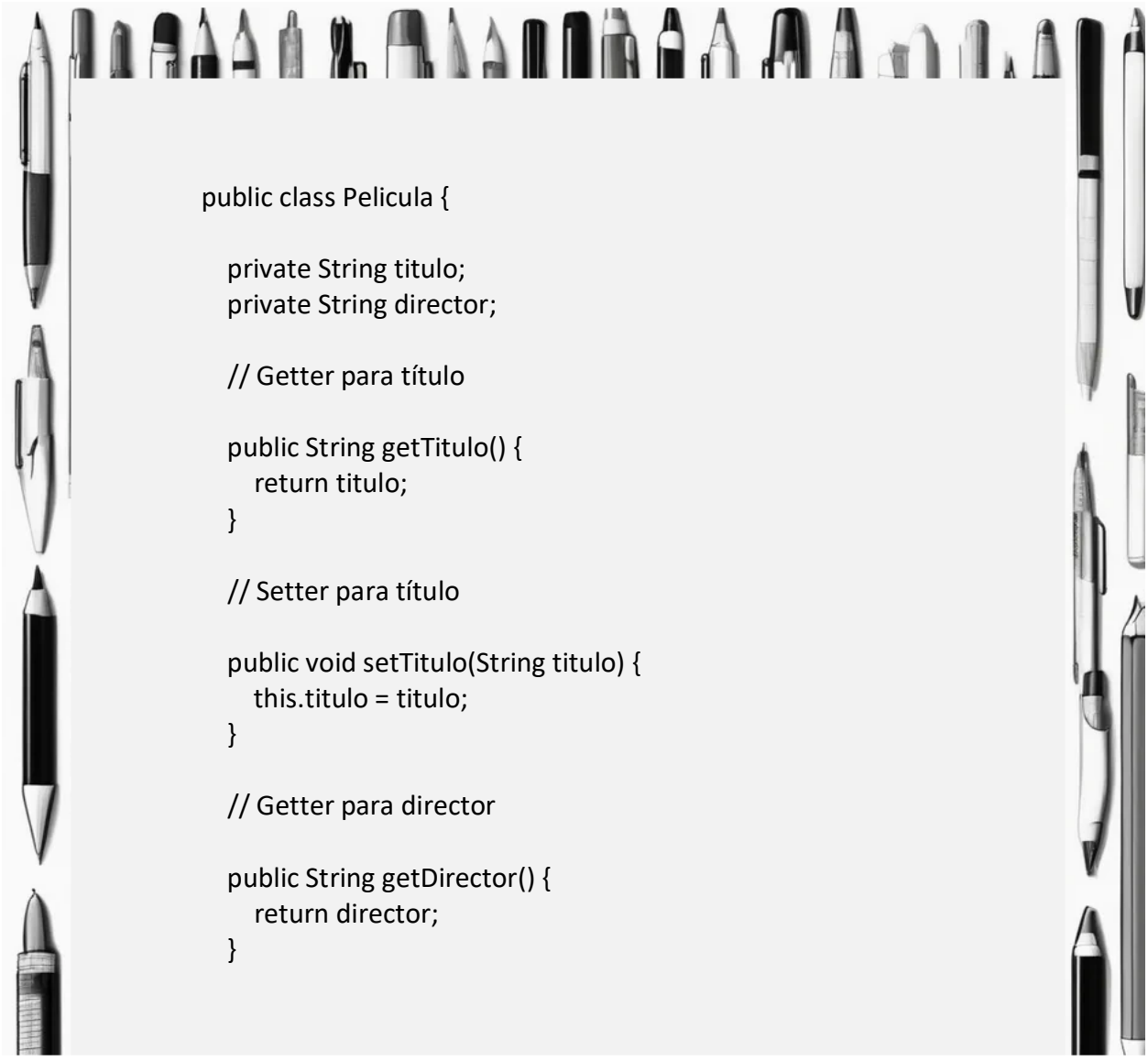
```
public class Coche {

    private String marca;
    private String modelo;

    // Getter para marca

    public String getMarca() {
        return marca;
    }


    // Setter para marca


    public void setMarca(String marca) {
        this.marca = marca;
    }

    // Getter para modelo

    public String getModelo() {
        return modelo;
    }
```

```
  // Setter para modelo

   public void setModelo(String modelo) {
      this.modelo = modelo;
   }


   // Método para mostrar información del coche

   public void mostrarInformacion() {
      System.out.println("Marca: " + marca + ", Modelo: " + modelo);
   }
}

public class Main {

   public static void main(String[] args) {
      Coche coche1 = new Coche();
      coche1.setMarca("Toyota");
      coche1.setModelo("Corolla");
      coche1.mostrarInformacion();
   }

}
```

Ejercicio 5

Crea una clase Pelicula con atributos título y director. Usa métodos getter y setter para acceder y modificar estos atributos, y un método para mostrar la información de la película.

Solución:



```
public class Pelicula {

    private String titulo;
    private String director;

    // Getter para título

    public String getTitulo() {
        return titulo;
    }

    // Setter para título

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    // Getter para director

    public String getDirector() {
        return director;
    }
```

```
// Setter para director

    public void setDirector(String director) {
        this.director = director;
    }


    // Método para mostrar información de la película

    public void mostrarInformacion() {
        System.out.println("Título: " + titulo + ", Director: " + director);
    }

}

public class Main {

    public static void main(String[] args) {
        Pelicula pelicula1 = new Pelicula();
        pelicula1.setTitulo("Inception");
        pelicula1.setDirector("Christopher Nolan");
        pelicula1.mostrarInformacion();
    }

}
```

Capítulo 20.

La fase final, creamos una herramienta de desbloqueo de puertas

Tenemos todo listo para la fuga. Como ya te he comentado, yo seré el encargado de abrir la puerta final. Justo al lado de la salida hay una mesa con un ordenador desde el cual se puede acceder a todo el sistema y, de paso, abrir la puerta principal. Normalmente, se utilizan cinco claves de seguridad y, sin ellas, es imposible abrirla. No hace falta decir que con mis conocimientos informáticos actuales me resultaría imposible hackearla.

No obstante, Bud y Pedro conocen el sistema a la perfección. Han estado años trabajando con él e incluso han programado gran parte del código que controla las funciones principales de seguridad. Han querido facilitarme las cosas para asegurarse de que podemos salir sin complicaciones y rápidamente. Para ello, han creado cinco programas que se encuentran alojados en cinco clases diferentes. Estas clases ya están ocultas dentro del sistema, así que mi único cometido es crear una clase Main y llamar a los métodos que se encuentran en ellas.

Es una tarea humilde, pero de vital importancia. No voy a ir a ciegas; Bud se ha encargado de enseñarme los cinco programas y explicarme para qué sirven. Además, ya hemos practicado cómo debo llamar a los métodos, así que se puede decir que sé de memoria lo que tengo que hacer. No me gustaría ponerme nervioso y quedarme en blanco justo en el momento más crítico.

Para que puedas hacerte una idea, a continuación, te voy a mostrar las clases y te daré una explicación de para qué sirve cada una. Evidentemente, Bud ha dedicado mucho tiempo a trabajar en ellas y ha comprobado muchas veces que no hay ningún error que pueda comprometer nuestra fuga.

1. **Clase Clave**

Te había dicho que hacen falta cinco claves para abrir la puerta de forma normal. Nosotros solo tenemos una; la conseguimos cuando accedimos al despacho de Vicente. Tenemos suerte, ya que es la más importante; gracias a ella se pueden seguir los demás pasos para, al final, conseguir desarmarla completamente. Esta clase permite al sistema comprobar que la clave que tenemos todavía está activa y vigente. Por lo tanto, esta clase representa una clave y verifica si la llave es válida.

```
public class Clave {
   private String Clave;


   public Clave(String key) {
      this.Clave = Clave;
   }

   public boolean Valida() {
      // Verificación simple de la llave
      return "claveCorrecta".equals(this.Clave);
   }
}
```

Una vez introducida la clave de acceso, el sistema quiere saber quién es la persona que trata de acceder. Fingiremos que somos el director. Para ello, contamos con su código de identificación personal. Es muy sencillo conocer el número de identificación de cualquier trabajador de la prisión; no es ningún secreto, ya que aparece en cualquier documento o registro de empleados.

2. **Clase AccessCode**

Esta clase representa un código de acceso y verifica si el código es correcto.

```
public class AccessCode {
    private int code;

    public AccessCode(int code) {
        this.code = code;
    }

    public boolean isValid() {
        // Verificación simple del código de acceso (puede ser cualquier lógica)
        return this.code == 1234;
    }
}
```

3. **Clase Light**

Para que la puerta se abra, es totalmente obligatorio que el foco de luz que ilumina la puerta desde el exterior esté encendido. Esto ayuda a aumentar la seguridad y la visibilidad de la zona. Sin embargo, nosotros preferimos la oscuridad, para que las cámaras no nos detecten. Por lo tanto, debemos hacer creer al sistema que la luz está encendida. Lo haremos con el siguiente método.

```
public class Light {
   public void turnOn() {

      // Simula encender la luz

      System.out.println("La luz está encendida.");

   }
}
```

4. **Clase Alarm**

Cuando la puerta se abre sin autorización, suena una alarma, y no queremos que eso ocurra. Debemos crear un método que indique que la alarma está activada, aunque ya nos hemos encargado de desactivarla previamente.

```
public class Alarm {
   public void enable() {

      // Simula que  la alarma esta activa.

      System.out.println("La alarma está activada.");

   }
}
```

5. **Clase Puerta**

Por último, queremos engañar al sistema de seguridad e indicar que la puerta está abierta. Simplemente tendremos que crear una clase que indique que, efectivamente, la puerta está abierta. De esa forma, realmente lo estará y, por fin, podremos salir.

```
public class Puerta {
   public void abrir() {

      // Simula abrir la puerta

      System.out.println("La puerta está abierta.");

   }
}
```

Clase Main

En esta clase, creamos los objetos necesarios y llamamos a sus métodos para abrir la puerta.

```
public class Main {
   public static void main(String[] args) {

      // Crear objetos

Clave clave = new Clave("67898");
AccessCode accessCode = new AccessCode(787656);
```

```
        Light light = new Light();
        Alarm alarm = new Alarm();
        Door door = new Door();

        // Proceso para abrir la puerta

        if (Clave.isValid() && accessCode.isValid()) {
           light.turnOn();
           alarm.disable();
           door.open();
        } else {

           System.out.println("La llave o el código de acceso son incorrectos. No
se puede abrir la puerta.");

        }
     }
}
```

Básicamente, esto es todo. Esperemos que esta noche todo salga bien. Nos ha costado mucho esfuerzo llegar hasta aquí, y sería un gran problema no conseguir nuestra fuga. Aunque sería mucho peor que nos atraparan. Si eso pasara, me esperaría una larga temporada encerrado, Rich quedaría impune y nunca se haría justicia.

Capítulo 21.

El desenlace

Todo ha salido como esperábamos. Hemos conseguido salir de la prisión sin ninguna complicación. Seguimos con el plan y ya estamos en casa de la amiga de Pedro. No tengo demasiado tiempo, así que la despedida será corta. No tengo ni idea de cuál será el siguiente destino de mis compañeros de huida. Nunca me lo dijeron. Lo que tengo claro es que no voy a volver a verlos más. Es una sensación muy desoladora, ya que, aunque no los conozco desde hace mucho, he creado una gran amistad con ellos y han sido un gran apoyo.

Desde esta casa me dirijo a pie a la biblioteca. Mientras camino, pienso que estar en la calle es una sensación muy extraña. No sé cómo describirlo, pero el mero hecho de ir andando por cualquier calle me hace feliz. Espero que a mis nuevos amigos les vaya bien y espero que lo mismo me suceda a mí. Estoy bastante nervioso porque no sé exactamente qué va a pasar, y es una sensación muy angustiosa.

Aunque Chani me está ayudando a demostrar mi inocencia, mi relación con él ya no será la misma. Tengo pensado marcharme de esta ciudad y olvidar todo lo que me ha pasado aquí. Eso también incluye a Chani. De Rich, mejor ni hablar. Era uno de mis mejores amigos y me traicionó sin pensarlo por algo de dinero.

Sigo caminando y mi cabeza no para de dar vueltas. Supongo que es normal, pero tengo un muy mal presentimiento. Quizás sean los nervios. Disfruto de lo que podrían ser mis últimos minutos de libertad, aunque no puedo aprovecharlos como me gustaría. Debo andar muy deprisa, ya que no tengo mucho tiempo.

Así, sin aviso previo, alguien me agarra del brazo y casi me da un infarto. Nadie me conoce por esta zona, así que no esperaba que intentaran detenerme por ningún motivo. Al voltearme, me llevé una pequeña sorpresa: era Rich. Pensé que sería una coincidencia, ya que habíamos quedado en el mismo sitio y seguíamos el mismo camino. Pero en realidad, no era así. Me estaba esperando allí para advertirme.

Nada más verlo, me entró una inmensa sensación de enfado. Pensé en golpearlo, pero antes de que pudiera decir nada, comenzó a hablar de forma apresurada. Apenas podía entenderlo.

Me esforcé mucho por saber lo que estaba diciendo y llegué a la conclusión de que me estaba llevando a una trampa. No tengo mucho tiempo, así que le pedí que se explicara mejor y de forma más pausada.

Entonces comenzó a explicarme una historia. Según él, Chani había planeado todo junto con su novia coreana. Al parecer, todo había sido idea de esta última. Lo había planeado desde la lejanía. Además, contaban con el primo de Chani como cómplice. La idea principal era inculparme a mí y a Rich, pero parece que no tuvieron el tiempo suficiente para acceder a su ordenador y desistieron.

Siguiendo con la historia, en teoría, Rich no sabía nada hasta que Chani y su novia le contactaron hace unas pocas semanas. Le ofrecieron una buena suma de dinero si testificaba contra mí y corroboraba su historia. Según Rich, se negó a hacerlo y entonces vinieron las amenazas. Primero sugirieron que podían hacer que le encerraran y, después, viendo que su víctima aún dudaba, empezaron las amenazas físicas hacia él y su familia. Le aconsejaron mantenerse al margen, y es por eso que no había recibido noticias suyas en un largo período de tiempo.

Mientras tanto, Chani y sus compinches habían estado tejiendo un plan para asegurarse de mantenerme encerrado por mucho tiempo. La idea era convencerme de escapar de la cárcel y que, cuando llegara al lugar acordado, me estuviese esperando allí la policía. Al parecer, eso es lo que me aguarda allí: un montón de agentes de policía camuflados, desde los alrededores hasta dentro de la biblioteca.

La historia parece creíble; incluso la forma de expresarse de Rich es muy convincente, pero esta trama es bastante difícil de creer. Llevo meses planeando la huida de la prisión basándome en una información, y no me entra en la cabeza que las cosas puedan ser diferentes. Por otro lado, es normal que Rich quiera engañarme; en teoría, es a él a quien le interesa que yo esté en prisión.

Después de pensarlo menos de un minuto, le digo que no me lo creo y que pienso llegar hasta la biblioteca. Aunque en realidad ya me han entrado dudas y estoy extremadamente nervioso. Pero, llegados hasta este punto, no sé qué más hacer. La única opción que tengo es seguir con el plan y confiar en que todo salga bien. La historia que me acaban de contar tiene mucho sentido, pero si yo estuviese en su lugar, también trataría de engañarme.

Por suerte para mí, Rich tenía pruebas para mostrarme. La primera vez que contactó con Chani y su novia, consiguió una grabación. En cierto momento de la reunión, se ausentó para ir al baño y activó la grabación de audio de su móvil. Al principio de la conversación se escuchan dos personas hablando en coreano, aunque en cierto momento empiezan a hablar en inglés con Rich. Es sin duda la voz de Chani; empieza diciendo que soy un criminal y que debo pagar por mis delitos. Luego le aconseja que no me ayude y termina con una amenaza, dejando claro que, si le ayuda de algún modo, acabará dando explicaciones a la justicia.

Chani es listo, así que en ningún momento deja entrever su implicación y es muy comedido en sus amenazas. Esta grabación me ha abierto los ojos, aunque, aun así, no estoy convencido al 100%. Puede que fuese parte del plan para conseguir que Rich confiase en ellos y así obtener una grabación incriminatoria. Sea como fuere, me decanto más por creer la versión de Rich. Pero aún estoy dudando; sopeso mis alternativas y repaso mentalmente todas las conversaciones que he mantenido con los implicados desde que entré en prisión.

Como Rich ve que aún sigo dudando, decide mostrarme una prueba más. Algo irrefutable. Me pide que le siga, aunque me sugiere que me mantenga a distancia. Otea el horizonte con mucho cuidado y va pegado a la pared. Parece que está buscando algo. Llega a la altura de un contenedor que hace esquina; desde ahí se puede observar la biblioteca y toda la calle principal frente a ella. No sé qué estamos buscando, pero casi sin decirme nada me señala una furgoneta. Hay gente dentro, parecen estar vigilando. Después me muestra otra furgoneta que está aparcada al otro lado. Está claro que son policías y están esperando mi llegada.

En ese momento, empiezo a sentir una cantidad de ansiedad que apenas puedo controlar. Pienso que, si los policías están esperando, es porque ya saben que me he escapado de la prisión y todo el plan se ha ido al garete. Lo más sencillo sería tratar de huir. Ojalá supiese a dónde ha ido Bud; quizás podría llevarme con él o al menos ayudarme a salir del país.

Aún no sabía que Bud seguía monitorizando la comunicación que llegaba a la prisión. Dentro de ella, los policías todavía no sabían que me había fugado, así que no me estaban buscando. Si volvía antes de que hiciesen el registro de mi celda, nadie tendría pruebas de mi escapada y nadie tendría motivos para dudar de mí.

De pronto recordé la grabación que el primo de Chani me mostró para inculpar a Rich, donde pedía a Chani que se fuese del país y no volviese más. Se lo comenté a Rich, y me dijo que era una frase sacada de contexto. La conversación no era con Chani, sino con un amigo inglés de su misma ciudad. Es una excusa tan absurda que debe de ser cierta.

No sé qué hacer; me he quedado bloqueado. Por suerte, Rich comienza a hablar en voz baja y me comenta que tiene un plan. Parece que lo tenía planeado desde que Chani le pidió participar en esta reunión trampa. En realidad, apenas tengo que hacer nada; todo va a depender de él. Bueno, para ser sinceros, si me pide algo, es que le golpee en la cara. Me sorprende, pero solo tuvo que pedirlo dos veces. Está sangrando de la nariz, se ha limpiado la camiseta e incluso se ha manchado de barro parte de la ropa. Por último, me dice que escape de allí cuanto antes. Mi intención es volver a prisión con Phil; aunque en la cárcel ya sepan que no estoy, puedo inventarme alguna historia, como que me quedé dormido en la lavandería. Le indico a Rich mi punto de reunión con Phil, y simplemente se despide, diciéndome que me da dos minutos de ventaja.

Pasados esos dos minutos, Rich salió corriendo a la calle principal pidiendo ayuda desesperadamente. Fue entonces cuando todos los policías se acercaron a ver lo que estaba sucediendo. Les contó que alguien le había agredido y que se había ido corriendo. Por supuesto, les indicó la dirección contraria a la que utilicé para escapar. En esos momentos, yo ya estaba llegando al parking, donde esperaría a Phil agazapado.

Gracias a esta estrategia, la biblioteca se quedó sin protección. Los policías indicaron a Rich que debía quedarse allí esperando a la ambulancia, pero eso no era lo que él tenía en mente. Dentro de la biblioteca no se habían enterado de lo que había pasado, así que Chani y su novia aún pensaban que la reunión seguía en pie y que las autoridades estaban ahí fuera esperando para arrestarme

Les explicó que se encontró conmigo en la entrada, que le había atacado y que luego había huido al ver la presencia policial. Esto les hizo pensar que en esos momentos yo estaba a punto de ser capturado y que su plan había sido todo un éxito. Por otro lado, después del ataque, pensaban que a partir de ahora Rich estaba totalmente de su lado.

Comenzaron a hablar distendidamente, y Rich les explicó en detalle cómo había sido agredido por sorpresa. Sabía que tenía que ganarse su confianza antes de sacar algún tema incriminatorio. Si sospechasen algo, todo estaría perdido. Solo hay una oportunidad para conseguirles una confesión.

La novia de Chani empezó a decir que yo era un peligro y que, gracias a esto, iban a condenarme por una larga temporada. Chani agradeció a Rich su ayuda y se dieron la mano. Entonces el tono de la conversación cambió y se hizo más distendido. Durante algunos minutos, ambas partes estuvieron comentando sus planes de futuro inmediato. Entonces, Rich aprovechó la situación.

Comenzó a quejarse de que, pese a ser cómplice del delito de la pareja coreana, él no había recibido nada de dinero, mientras que ellos, en cambio, habían conseguido una buena suma, lo suficiente como para vivir bien durante una larga temporada. A estos comentarios les siguió un largo silencio. Eran bastante desconfiados y no querían corroborar las palabras de Rich.

Una vez pasada la tensión, prosiguieron diciendo que no habían recibido dinero alguno y que simplemente se alegraban de haber ayudado a las autoridades a detener a un delincuente. Las cosas iban por mal camino y era hora de realizar el último intento. Viendo que la pareja era muy hermética, decidió probar una nueva táctica: enfadarles. Estaba seguro de que en algún momento explotarían. Ya se habían mostrado muy agresivos con él en el pasado, y sabía que podría volver a pasar.

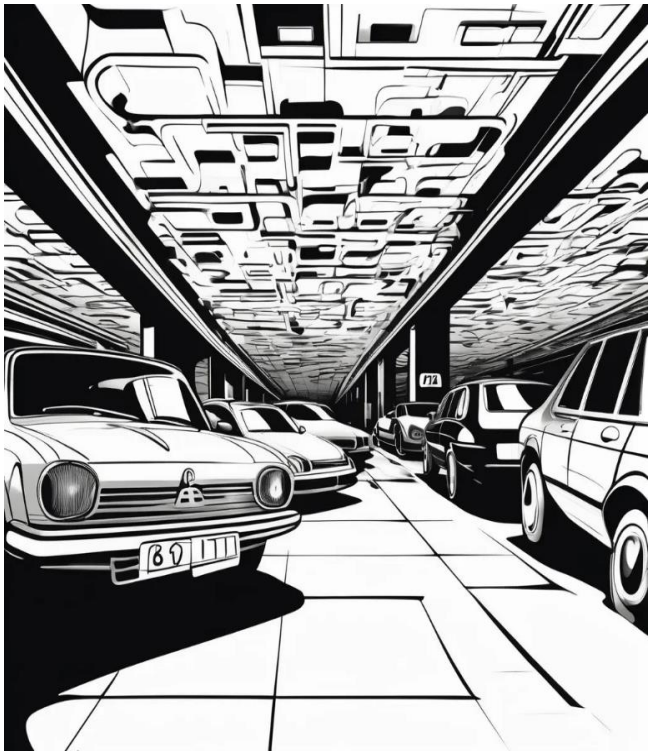
Comenzó a jugar su última baza, amenazándoles con hablar si no recibía dinero. Lo importante era que pensasen que yo no le importaba en absoluto y que lo único que quería era dinero. De esta forma, no sospecharían nada. Llegados a este punto, nadie podría culparle si reclamaba una parte del botín. Además, era lo justo, ya que se jugaba mucho siendo cómplice e incluso les había ayudado a tenderme esta trampa.

En cuanto a mi injusta inculpación, Chani, extremadamente cauto, ni confirmó ni desmintió las palabras de Rich. Se limitó a decir que lo importante era que todos se encontraban bien físicamente, lo cual sonaba a una sutil amenaza. Recalcó que lo único que deseaba para todos era que pudiesen seguir siendo amigos y continuar cada uno con su vida, tratando de olvidar lo que había pasado.

Pese a que Chani se mantenía cauteloso, Rich lo intentó una última vez. Esta vez, de manera directa, le dijo que, si no recibía una compensación justa, confesaría ante los policías esa misma noche e incluso testificaría contra ellos en un juicio. Eso me exculparía y los metería a ellos dos en prisión. Fue entonces cuando la novia de Chani perdió el control y se enfureció de forma desmedida. Comenzó a gritar asegurando que él nunca recibiría nada del dinero que habían robado y que, si no quería sufrir un "accidente", así que más le valía mantener la boca cerrada.

Habían mordido el anzuelo, y Rich ya tenía lo que quería. Ahora, para evitar problemas, debía reducir la tensión y salir de allí con las pruebas grabadas. Se limitó a decir que no era tan estúpido como para confesar, ya que eso también podría traerle problemas a él. Por último, añadió que estarían en contacto porque no renunciaba a una parte del dinero y que tendrían tiempo para discutirlo con más calma. Chani, más calmado, aceptó. Le dio la mano y se despidieron. Rich salió de allí con el objetivo cumplido.

Mientras tanto, yo estaba agazapado en el aparcamiento, esperando a Phil. No tenía ningún modo de comunicarme con él, solo la esperanza de que aparecería a la hora acordada. Con el chantaje que le habíamos hecho, no dudaba de que cumpliría. Cinco minutos antes de lo pactado, llegó y apagó las luces del coche. Durante al menos dos minutos, observé la zona en busca de algún movimiento sospechoso. Al ver que todo estaba tranquilo, me acerqué al coche con cautela. Phil salió y me metió en el maletero. No sería un viaje cómodo y, para ser sincero, era bastante claustrofóbico, pero era la única manera de regresar a la prisión sin ser detectado.



El aparcamiento para miembros de la prisión es subterráneo y desde allí se puede acceder a varias salas. Antes de escapar, Bud me mostró una ruta de vuelta a mi celda. La puerta debería seguir abierta y, una vez allí, podré cerrarla desde dentro. Era el plan perfecto, aunque me preocupa que los policías hayan alertado a los guardias de la prisión sobre mi posible fuga. A estas alturas, probablemente ya hayan revisado mi celda y descubierto que no estaba. Aun así, confío en el plan. Nadie me ha visto fuera de estos muros y no pueden demostrar que me he fugado. Si logro regresar sin ser detectado, podré seguir adelante con nuestra estrategia.

Regresar a mis humildes aposentos fue bastante sencillo. Cerré la celda desde dentro y ahora solo espero la inspección matutina. No veo mucho revuelo, lo que significa que no están buscando a nadie; una muy buena señal. Cuando llegó el vigilante, su expresión cambió por completo. Por un momento, me asusté, pero entonces recordé que Bud se había escapado y todavía no lo sabían.

Rápidamente, el guardia dio la voz de alarma y comenzaron las labores de búsqueda. Fue entonces cuando me di cuenta de que Bud, desde fuera, me había ayudado bloqueando las comunicaciones entre la policía y la prisión. El plan había salido a la perfección.

Por supuesto, fui interrogado por la desaparición de mis compañeros de fuga, especialmente porque compartía celda con uno de ellos. Me limité a decir que estaba dormido y que, al despertar, me encontraba solo en la celda. Era una explicación poco creíble, pero imposible de refutar.

Rich, por su parte, entregó el audio a mi abogado y juntos empezaron a trabajar en mi defensa. Gracias a las nuevas pruebas y al testimonio de Rich, no tardé mucho en recuperar mi libertad. Finalmente, pude demostrar mi inocencia. Ahora tengo toda una nueva vida por delante y, además, he aprendido a programar en Java.

Chani y su novia no tuvieron la misma suerte. Fueron encontrados culpables de extorsión y condenados a varios años de cárcel. Las empresas afectadas recuperaron el dinero, y todo se aclaró por fin.

Gracias por haberme acompañado en este arduo camino. Juntos hemos aprendido a programar en Java, y espero que te sea tan útil como lo ha sido para mí.

No importa si eres principiante o ya tienes nociones de programación, este libro está diseñado para guiar paso a paso en el dominio completo de la sintaxis de Java. A lo largo de sus páginas, aprenderás los fundamentos del lenguaje de una manera clara y accesible, con ejercicios resueltos que refuerzan cada concepto.

La obra se divide en dos partes: una teórica, repleta de ejemplos prácticos y una segunda, donde pondrás a prueba tus habilidades en un escenario ficticio. Acompaña a Peli, el protagonista de esta historia, en su intrépida aventura pora escapar de prisión, donde cada decisión que toma está condicionada por sus conocimientos en programación.

Cada capítulo está diseñado para aumentar progresivamente en dificultad, proporcionando una sensación constante de avance y manteniendo el interés y la motivación hasta el final.

¡Empieza a aprender Java ahora mismo y vive una experiencia única que te atrapará desde la primera página!