1. Tipos de datos básicos

Ejercicio 1:
Crea un programa que solicite al usuario introducir su edad y su altura. La edad debe almacenarse en una variable de tipo `int`, y la altura en una variable de tipo `double`. Posteriormente, el programa debe imprimir ambos valores con un mensaje apropiado.

Ejercicio 2:
Escribe un programa que convierta grados Celsius a Fahrenheit. El usuario debe ingresar la temperatura en Celsius como un `float`, y el programa debe mostrar el resultado en Fahrenheit utilizando el tipo `double`. La fórmula para la conversión es `F = C * 9/5 + 32`.

2. Estructuras de control

Ejercicio 1:
Crea un programa que pida al usuario un número y determine si es par o impar utilizando una estructura `if-else`. Además, el programa debe permitir al usuario introducir números hasta que ingrese un 0, momento en el cual el programa terminará.

Ejercicio 2:
Escribe un programa que imprima los primeros 10 números de la serie Fibonacci. La serie Fibonacci es una secuencia donde cada número es la suma de los dos anteriores, empezando por 0 y 1.

3. Clases

Ejercicio 1:
Define una clase `Coche` que tenga como atributos el `modelo` y la `velocidadMaxima`. Añade un método `mostrarCaracteristicas()` que imprima los datos del coche. Crea una instancia de esta clase en el `main` y llama al método `mostrarCaracteristicas()`.

Ejercicio 2:
Crea una clase `Calculadora` que contenga métodos para sumar, restar, multiplicar y dividir dos números pasados como parámetros. Luego, crea una instancia de esta clase en el `main` y utiliza cada uno de los métodos.

4. Encapsulación y visibilidad

Ejercicio 1:
Diseña una clase `CuentaBancaria` que encapsule el comportamiento de una cuenta. Debe tener los atributos privados `titular` y `saldo`, y los métodos públicos para `depositar()` y `retirar()` dinero, así como `getSaldo()` para consultar el saldo. Implementa también un método `mostrarInformacion()` para imprimir el titular y el saldo de la cuenta.

Ejercicio 2:
Crea una clase `Password` que represente una contraseña. La clase debe tener un atributo privado `password` de tipo `String`. Implementa un método `cambiarPassword(String nuevoPassword)` que permita cambiar la contraseña solo si la nueva cumple con ciertos criterios (por ejemplo, mínimo 8 caracteres y al menos un número). Incluye también un método `esFuerte()` que determine si la contraseña tiene al menos 2 dígitos y 1 carácter especial.

5. Herencia

Ejercicio 1:
Crea una clase `Vehiculo` con atributos `marca`, `modelo` y `kilometraje`. Luego, crea una clase `Coche` que herede de `Vehiculo` y añade un atributo `numeroDePuertas`. Implementa los métodos para obtener y establecer el valor de los atributos. Crea instancias de `Coche` y utiliza los métodos heredados y propios.

Ejercicio 2:

Vamos a implementar una jerarquía de clases para representar diferentes tipos de vehículos. Deberás crear una clase base `Vehiculo` y luego otras clases que hereden de ella, como `Coche`, `Motocicleta` y `Camion`.

1. **Clase Vehiculo:**
   - Atributos: marca, modelo y año.
   - Métodos: `mostrarDatos()` que imprima la información del vehículo.
2. **Clase Coche:**
   - Atributos específicos: número de puertas.
   - Métodos: Debe sobrescribir `mostrarDatos()` para incluir el número de puertas.
3. **Clase Motocicleta:**
   - Atributos específicos: si tiene sidecar o no.
   - Métodos: Debe sobrescribir `mostrarDatos()` para mostrar si tiene sidecar.
4. **Clase Camion:**
   - Atributos específicos: capacidad de carga en toneladas.
   - Métodos: Debe sobrescribir `mostrarDatos()` para incluir la capacidad de carga.

6. Interfaces y Clases Abstractas

Ejercicio 1:
Crea una interfaz `OperacionesMatematicas` con métodos para `sumar`, `restar`, `multiplicar` y `dividir` dos números. Luego crea una clase `Calculadora` que implemente esta interfaz y proporcione la implementación de estos métodos.

Ejercicio 2:
Define una clase abstracta `FiguraGeometrica` con un método abstracto `calcularArea()`. Crea clases `Circulo`, `Rectangulo` y `Triangulo` que extiendan `FiguraGeometrica` y proporcionen implementaciones concretas para `calcularArea()`. Crea una instancia de cada clase y muestra cómo se calculan las diferentes áreas.

7. Polimorfismo

Ejercicio 1:
Implementa una clase base `Animal` con un método `hacerSonido()`. Luego, crea las clases `Perro` y `Gato` que extiendan de `Animal` y sobrescriban el método `hacerSonido()` para que el `Perro` "ladre" y el `Gato` "maúlle". Demuestra polimorfismo creando un método que acepte un objeto `Animal` y llame a `hacerSonido()`.

Ejercicio 2:
Extiende el ejercicio anterior añadiendo una nueva clase `Vaca` que herede de `Animal` y sobrescriba el método `hacerSonido()` para que diga "Muuu". Añade una instancia de `Vaca` al método `main` y utiliza el método `hacerSonar()` para demostrar el polimorfismo.

8. Arrays

Ejercicio 1:
Crea un array de enteros de tamaño 10. Llena el array con números aleatorios del 1 al 100 y luego imprime todos los elementos del array en orden. A continuación, imprime los elementos del array en orden inverso.

Ejercicio 2:
Crea una función que reciba un array de enteros y devuelva la suma de todos sus elementos. Utiliza esta función en el método `main` para sumar los elementos de un array que hayas llenado previamente con valores aleatorios.

Ejercicio 3:
Crea una matriz de 3x3 de tipo `int` y llénala con números aleatorios del 1 al 9. Después, imprime la matriz en formato de cuadrícula, mostrando los números en su disposición de filas y columnas.

Ejercicio 4:
Escribe un programa que defina una matriz de 4x4 y la llene con números secuenciales del 1 al 16. Luego, calcula e imprime la suma de los elementos de la diagonal principal (de arriba a la izquierda hacia abajo a la derecha).

9. Strings

Ejercicio 1:

Crea un programa en Java que realice las siguientes acciones:

1. Solicita al usuario que introduzca dos cadenas de texto (String) por consola.
2. Concatena ambas cadenas en una nueva variable.
3. Muestra la cadena resultante.
4. Pide al usuario que introduzca un carácter y busca ese carácter en la cadena concatenada, mostrando la posición en la que se encuentra. Si no se encuentra, muestra un mensaje indicando que el carácter no está presente.

Ejercicio 2:

Crea un programa en Java que realice las siguientes acciones:

1. Define dos variables `String` con valores "CadenaDePrueba" y "CADENAdePRUEBA" respectivamente.
2. Convierte ambas cadenas a minúsculas y compara si son iguales, mostrando un mensaje con el resultado de la comparación.
3. Crea un substring de la primera cadena que contenga los primeros 5 caracteres y muestra este nuevo substring por consola.
4. Reemplaza todas las letras 'a' de la segunda cadena por el carácter '@' y muestra el resultado por consola.

10. Colecciones

List

Ejercicio 1:

Crea un programa en Java que:

1. Declare una `ArrayList` de números enteros.
2. Pida al usuario que introduzca 5 números enteros y los añada a la lista.
3. Muestre la lista completa.
4. Elimine el tercer elemento de la lista y muestre la lista actualizada.
5. Ordene la lista en orden ascendente y muestre el resultado.

Ejercicio 2:

Crea un programa en Java que:

1. Declare una `LinkedList` de cadenas de texto (Strings).
2. Añada a la lista las siguientes cadenas: "Java", "Python", "C++", "JavaScript".
3. Muestre la lista.
4. Solicite al usuario una cadena de texto para buscar en la lista y muestre si está o no presente.
5. Invierta el orden de los elementos en la lista y muestre el resultado.

Set

Ejercicio 1:

Crea un programa en Java que:

1. Declare un `HashSet` y añada nombres de personas, asegurándote de que algunos nombres se repitan.
2. Muestre el conjunto para que se vea que los duplicados se eliminan automáticamente.
3. Solicite al usuario un nombre y compruebe si está en el conjunto, mostrando un mensaje adecuado.

Ejercicio 2:

Crea un programa en Java que:

1. Declare un `TreeSet` y añada números enteros.
2. Muestre el conjunto para ver que los elementos se ordenan automáticamente.
3. Utiliza el método `first()` para mostrar el primer elemento y `last()` para mostrar el último.
4. Pida al usuario un número y utilice `higher()` y `lower()` para encontrar el siguiente número más alto y el anterior número más bajo en el conjunto, respectivamente.

Queue

Ejercicio 1:

Crea un programa en Java que:

1. Declare una `LinkedList` como una `Queue` y añada nombres de tareas en una lista de tareas pendientes.
2. Muestre la cola completa.
3. Procese la cola mostrando y eliminando cada tarea por el orden en que fueron añadidas, usando el método `poll()`.

Ejercicio 2:

Crea un programa en Java que:

1. Declare una `PriorityQueue` de números enteros.
2. Añada números aleatorios a la cola.
3. Muestre cómo, al recuperar y eliminar los elementos con `poll()`, se obtienen en un orden basado en su prioridad natural (menor a mayor).

Map

Ejercicio 1:

Crea un programa en Java que:

1. Declare un `HashMap` y añada pares de palabras (palabra y sinónimo).
2. Muestre el contenido del `Map`.
3. Solicite al usuario una palabra y muestre su sinónimo si está en el `Map`, o un mensaje indicando que no se encontró.

Ejercicio 2:

Crea un programa en Java que:

1. Declare un `TreeMap` para contar la frecuencia de cada palabra en un texto.
2. Añada varias palabras al `Map`, incrementando el contador cada vez que se añade una palabra repetida.
3. Muestre el `Map` para ver las palabras ordenadas y su frecuencia.

11. Excepciones

Ejercicio 1:

Crea un programa en Java que:

1. Solicite al usuario que introduzca dos números enteros por consola.
2. Realice la división del primer número entre el segundo y muestre el resultado.
3. Implemente manejo de excepciones para capturar y manejar adecuadamente `ArithmeticException` en caso de que el usuario intente dividir por cero.
4. Asegúrese de que, después de una excepción, el programa siga funcionando y permita al usuario intentar de nuevo con nuevos números.

Ejercicio 2:

Crea un programa en Java que:

1. Defina una clase de excepción personalizada `InvalidInputException` que se lance cuando el usuario ingrese un dato no válido.
2. Solicite al usuario que ingrese su edad y valide que sea un número entero positivo.
3. Si la entrada no es válida (por ejemplo, una cadena de texto o un número negativo), lance `InvalidInputException`.
4. Implemente un bloque `try-catch` para capturar la excepción personalizada y proporcione al usuario un mensaje de error claro, invitándolo a ingresar un valor válido.
5. El programa debe continuar solicitando la edad hasta que se ingrese un valor válido.

12. Streams

Flujos de caracteres

Ejercicio 1:
Crea un programa en Java que utilice `FileWriter` para escribir tres líneas de texto en un archivo llamado "texto.txt". Luego, usa `FileReader` para leer el contenido del archivo y mostrarlo por consola.

Ejercicio 2:
Modifica el ejercicio anterior para utilizar `BufferedReader` y `BufferedWriter` para escribir y leer un archivo de texto llamado "texto_buffered.txt".

Flujos de bytes

Ejercicio 1:
Crea un programa que copie un archivo de imagen (por ejemplo, "foto.jpg") a otro archivo (por ejemplo, "foto_copia.jpg") utilizando `FileInputStream` y `FileOutputStream`.

Ejercicio 2:
Escribe un programa que use `DataOutputStream` para escribir una serie de valores primitivos en un archivo "datos.bin" y luego use `DataInputStream` para leerlos y mostrarlos por consola.

13. Ficheros

Manejo de archivos con `File`

Ejercicio 1:
Escribe un programa que liste todos los archivos y directorios de un directorio dado por el usuario.

Ejercicio 2:
Desarrolla un programa que cree un archivo llamado "miArchivo.txt" y luego lo elimine. El programa debe informar en cada paso si la operación fue exitosa o no.

Lectura y escritura de archivos con `FileReader` y `FileWriter`

Ejercicio 1:
Crea un programa que permita al usuario escribir entradas en un diario. Cada entrada debe agregarse a un archivo "diario.txt", sin sobrescribir el contenido anterior.

Ejercicio 2:
Escribe un programa que lea el contenido de un archivo "libro.txt" y muestre el texto en consola, página por página. Considere que una página está compuesta por un número fijo de líneas.

Lectura y escritura de archivos con `BufferedReader` y `BufferedWriter`

Ejercicio 1:
Desarrolla un programa que copie el contenido de un archivo de texto "original.txt" a un nuevo archivo "copia.txt". Asegúrate de que cada línea se copie correctamente.

Ejercicio 2:
Escribe un programa que lea un archivo de texto "texto.txt" y cuente el número de palabras que contiene. Asume que las palabras están separadas por espacios.

Clases de alto nivel para lectura y escritura (`Scanner`, `PrintWriter`, `FileOutputStream`, etc.)

Ejercicio 1:
Crea un programa que permita al usuario introducir datos de nuevos usuarios y los almacene en un archivo "usuarios.txt". La información de cada usuario debe contener un nombre y un correo electrónico.

Ejercicio 2:
Escribe un programa que almacene una serie de números enteros en un archivo binario "numeros.bin". Luego, lee el archivo y muestra los números por consola.

Serialización de objetos

Ejercicio 1:
Crea un programa que permita al usuario guardar información de un objeto de una clase `Persona` (con nombre y edad) en un archivo usando serialización, y luego recuperar esa información y mostrarla en consola.

Ejercicio 2:
Modifica el ejercicio anterior para trabajar con una lista de objetos `Persona`. El programa debe permitir guardar y cargar la lista completa de objetos en un archivo.