**Tipos de datos y Clases Wrapper en Java**

**Paso 1: Introducción a los Tipos de Datos en Java**

En Java, los **tipos de datos** se dividen en dos categorías:

* **Tipos primitivos**: Son los tipos básicos de datos que Java proporciona. Incluyen int, float, double, char, boolean, etc.
* **Tipos de referencia**: Estos son objetos que incluyen clases, arrays y más.

**Tarea 1: Investiga y responde**

¿Cuáles son los tamaños y valores predeterminados de los siguientes tipos de datos primitivos en Java?

* int
* double
* char
* boolean

**Pista**: Consulta la documentación oficial de Java o realiza una búsqueda en tu IDE preferido para encontrar estos detalles.

**Paso 2: ¿Qué son las Clases Wrapper?**

Los tipos primitivos en Java no tienen métodos asociados, por lo que Java proporciona **Clases Wrapper**. Estas clases permiten tratar a los tipos primitivos como objetos, lo que facilita su manipulación en colecciones como ArrayList y otras estructuras que solo admiten objetos.

Ejemplos de Clases Wrapper:

* Integer para int
* Double para double
* Character para char
* Boolean para boolean

**Pregunta 1:**

¿Qué ventaja tiene usar una clase Wrapper en lugar de un tipo primitivo?

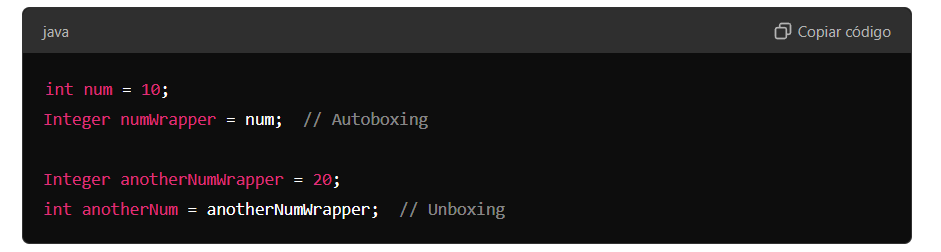
La ventaja principal de usar una clase Wrapper es que puedes tratar a los tipos primitivos como objetos. Esto es útil cuando necesitas trabajar con colecciones como ArrayList, que solo pueden almacenar objetos, no tipos primitivos. Además, las clases Wrapper proporcionan métodos adicionales para trabajar con los valores, como la conversión entre tipos.

**Ejemplo**: Piensa en cómo almacenar un número entero en un ArrayList. ¿Podrías hacerlo con un int o necesitarías usar Integer?

**Paso 3: AutoBoxing y UnBoxing**

* **AutoBoxing**: Es el proceso automático de convertir un tipo primitivo en su clase wrapper correspondiente.
* **UnBoxing**: Es la conversión automática de un objeto wrapper a su tipo primitivo correspondiente.

Ejemplo:



### Ejercicio 1: Practiquemos Autoboxing y Unboxing

Escribe un programa en Java que:

1. Cree una lista (ArrayList) de enteros.
2. Añada varios números enteros a la lista (utiliza autoboxing).
3. Recorra la lista y obtenga la suma de todos los elementos (utiliza unboxing).

**Pista**: Usa la clase Integer en tu código y experimenta cómo Java realiza las conversiones de manera automática.

### Paso 4: Métodos Útiles en las Clases Wrapper

Las clases Wrapper ofrecen varios métodos útiles. Por ejemplo:

* Integer.parseInt(String s) convierte una cadena en un entero.
* Double.valueOf(String s) convierte una cadena en un número decimal.

#### Pregunta 2:

¿Cuál es la diferencia entre parseInt y valueOf en la clase Integer?

 Integer.parseInt(String s): Convierte una cadena en un tipo primitivo int.

 Integer.valueOf(String s): Convierte una cadena en un objeto Integer (la clase wrapper).

#### Ejercicio 2: Aplicando lo Aprendido

Escribe un programa que:

1. Lea un número entero como una cadena desde la consola.
2. Utilice Integer.parseInt para convertir la cadena en un entero.
3. Muestre en pantalla si el número es par o impar.

**Pista**: Usa Scanner para leer la entrada del usuario.

### Paso 5: Comprendiendo el Valor Nulo

A diferencia de los tipos primitivos, los objetos (incluidas las clases Wrapper) pueden ser null. Esto puede ser útil en algunos casos, pero también puede causar errores si no se maneja adecuadamente.

#### Pregunta 3:

¿Qué sucede si intentas hacer UnBoxing en un objeto Wrapper que es null? Intenta hacer el siguiente experimento y observa el resultado:



Si intentas hacer unboxing en un objeto Wrapper que es null, obtendrás una excepción de tipo NullPointerException. Esto ocurre porque no puedes convertir un valor nulo en un tipo primitivo.

### Paso 6: Práctica Final

Desarrolla un pequeño sistema de gestión de notas donde:

1. Los estudiantes pueden ingresar sus calificaciones como cadenas.
2. Convierte esas cadenas a números utilizando clases Wrapper.
3. Calcula el promedio de las calificaciones y muestra si el estudiante aprobó (si el promedio es mayor o igual a 60).

(subido al repo)

**Estructuras de control en Java**

**Paso 1: Introducción a las Estructuras de Control**

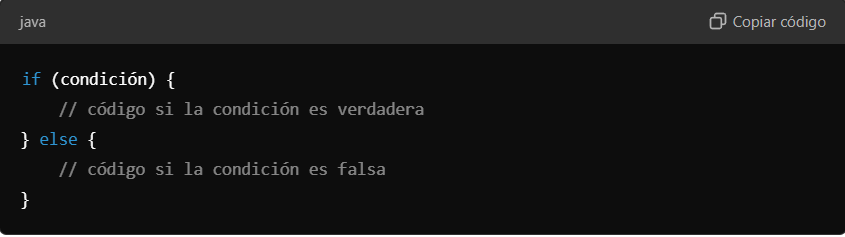
En Java, las estructuras de control te permiten tomar decisiones y repetir bloques de código, lo que hace que tus programas sean más dinámicos y útiles. Las principales estructuras de control son:

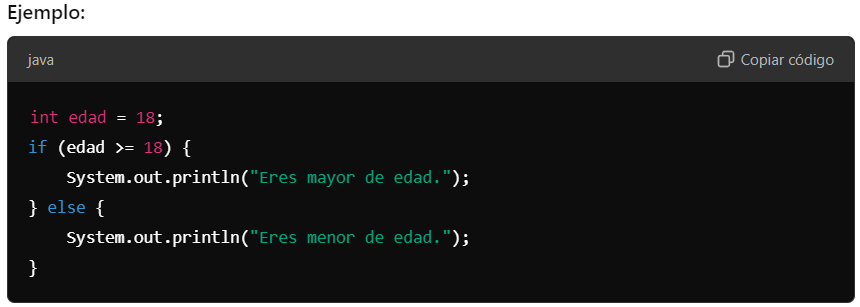
* **Condicionales**: if, else if, else, switch
* **Repetitivas**: for, while, do-while

**Paso 2: Estructura Condicional if-else**

El **if-else** permite ejecutar un bloque de código solo si se cumple una condición. Si la condición no se cumple, puedes usar else para ejecutar una alternativa.

**Sintaxis básica:**

****

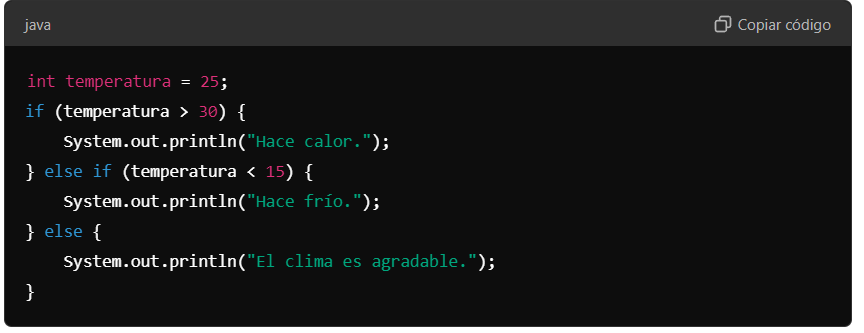
****

### Pregunta 1:

¿Qué sucede si tienes varias condiciones? ¿Cómo puedes usar else if?

Se puede usar else if para evaluar varias condiciones de manera secuencial. Si una condición es verdadera, se ejecuta su bloque de código y se ignoran las demás.

#### Ejemplo con else if:

****

### Ejercicio 1: Verifica si un número es positivo, negativo o cero

Escribe un programa que lea un número del usuario e imprima si el número es positivo, negativo o cero usando estructuras condicionales if-else.

### Paso 3: Estructura de Control switch

El **switch** es otra estructura condicional que evalúa el valor de una expresión y ejecuta un bloque de código basado en el caso que coincida.

#### Sintaxis básica:

#### 

#### 

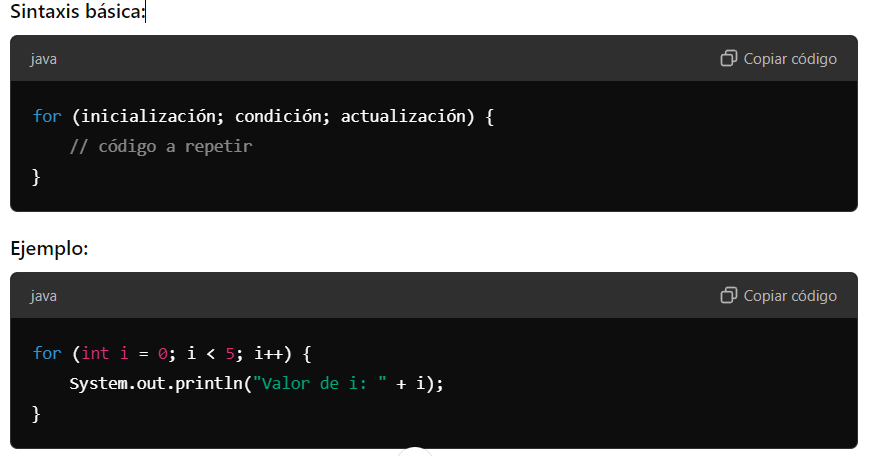
### Ejercicio 2: Calculadora con switch

Escribe un programa que lea dos números y un operador (+, -, \*, /) y realice la operación matemática usando un switch.}

### Paso 4: Estructuras Repetitivas

#### ****Bucle**** for

El bucle for se usa cuando sabes cuántas veces necesitas repetir un bloque de código.



#### ****Bucle**** while

El bucle while repite el bloque de código mientras se cumpla una condición.



### Pregunta 2:

¿Cuál es la diferencia entre for y while? ¿Cuándo elegirías uno sobre el otro?

La diferencia principal es que en un bucle for sabemos de antemano cuántas veces queremos repetir el código, mientras que en un while el número de repeticiones depende de una condición que puede cambiar durante la ejecución del programa. Usarías for cuando conoces el número exacto de iteraciones y while cuando no lo sabes de antemano.

### Ejercicio 3: Imprimir los primeros 10 números pares usando for

Escribe un programa que imprima los primeros 10 números pares usando un bucle for.

#### ****Bucle**** do-while

El bucle do-while es similar al while, pero garantiza que el bloque de código se ejecutará al menos una vez.



### Ejercicio 4: Solicitar números hasta que se introduzca un número negativo

Escribe un programa que pida al usuario que ingrese números hasta que introduzca un número negativo, utilizando un bucle do-while.

**Clases, Métodos, Atributos e Instancias en Java**

### Paso 1: ¿Qué es una Clase en Java?

Una **clase** es una plantilla o molde a partir del cual se crean los objetos. En términos simples, define las características (atributos) y las acciones (métodos) de un objeto.

****

### Pregunta 1:

¿Qué diferencia hay entre una clase y un objeto?

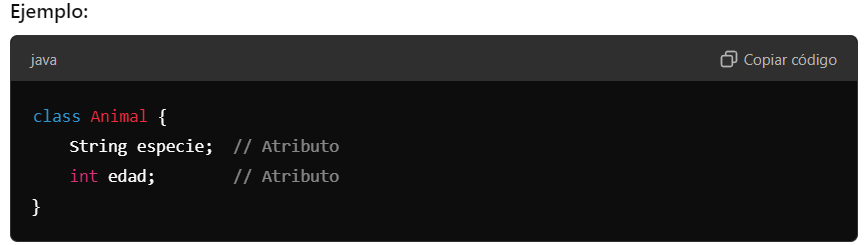
Una **clase** es solo una plantilla, mientras que un **objeto** es una instancia de esa clase. El objeto es la representación concreta de la clase en la memoria, con valores específicos en sus atributos.

### Ejercicio 1: Creando una clase

Escribe una clase Coche con dos atributos: marca y modelo, y un método que imprima esta información.

### Paso 2: ¿Qué son los Atributos?

Los **atributos** son las variables definidas dentro de una clase que representan el estado o las propiedades del objeto.



### Pregunta 2:

¿Cuál es el propósito de los atributos en una clase?

Los atributos permiten almacenar datos que describen las propiedades o características del objeto, y estos datos pueden variar entre diferentes instancias de la misma clase.

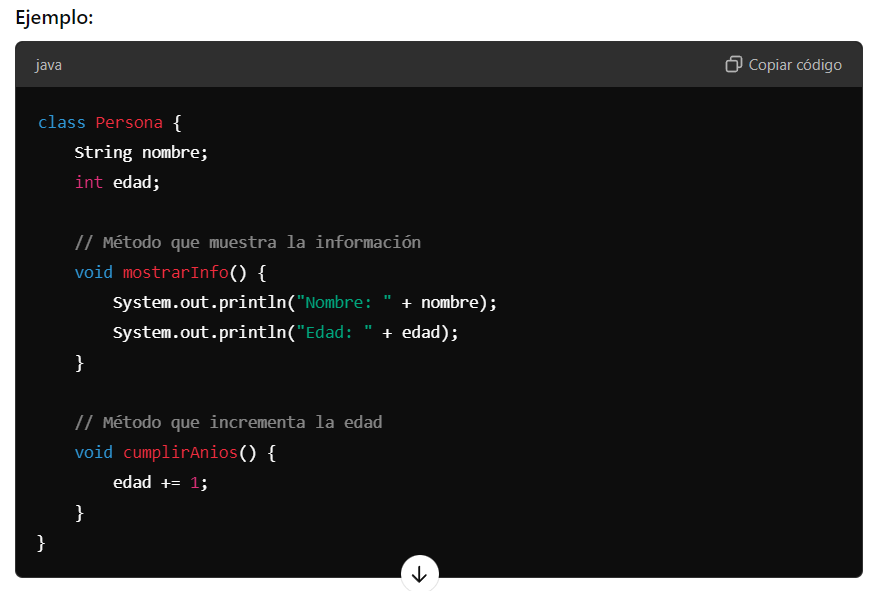
### Ejercicio 2: Definir y utilizar atributos

Crea una clase Libro con los atributos titulo, autor y numeroDePaginas. Luego, crea un método que imprima estos valores.

### Paso 3: ¿Qué son los Métodos?

Los **métodos** son funciones definidas dentro de una clase que especifican el comportamiento de los objetos. Un método puede acceder a los atributos del objeto y realizar acciones.



****

### Pregunta 3:

¿Qué es un método constructor y por qué es útil?

Un **constructor** es un método especial que se llama automáticamente cuando se crea una instancia de la clase. Su función es inicializar los atributos del objeto. Es útil para asegurarse de que los objetos siempre se creen en un estado válido.

### Ejercicio 3: Creando métodos

Añade un método constructor a la clase Coche que inicialice marca y modelo al crear un objeto.

### Paso 4: ¿Qué es una Instancia?

Una **instancia** es un objeto creado a partir de una clase. Para crear una instancia, se usa el operador new.



### Pregunta 4:

¿Cómo se diferencian los objetos entre sí si provienen de la misma clase?

Aunque los objetos provienen de la misma clase, cada uno tiene su propio conjunto de valores en los atributos, lo que los hace diferentes. Los atributos pueden tener diferentes valores en cada instancia.

### Ejercicio 4: Creando instancias

Crea tres instancias de la clase Libro con diferentes valores y muestra su información.

### Paso 5: Métodos con parámetros y retorno

Los métodos pueden aceptar parámetros y devolver un valor. Esto permite mayor flexibilidad y reutilización del código.



### Pregunta 5:

¿Por qué es útil devolver valores desde los métodos?

Devolver valores permite que el método proporcione resultados que pueden ser utilizados en otras partes del programa. Esto hace que el código sea más modular y reutilizable.

### Ejercicio 5: Calculadora simple

Crea una clase Calculadora con métodos para sumar, restar, multiplicar y dividir. Los métodos deben recibir dos números como parámetros y devolver el resultado.

**Modificadores de acceso**

**Paso 1: ¿Qué son los Modificadores de Acceso?**

Los **modificadores de acceso** determinan el nivel de visibilidad de los atributos y métodos de una clase. En Java, los modificadores más comunes son:

* **public**: El miembro es accesible desde cualquier lugar.
* **protected**: El miembro es accesible dentro del mismo paquete o subclases.
* **private**: El miembro solo es accesible dentro de la misma clase.

### Pregunta 1:

¿Por qué son importantes los modificadores de acceso?

Los modificadores de acceso son importantes porque proporcionan **encapsulamiento**, lo que significa que controlan qué partes del código pueden acceder o modificar los datos de una clase. Esto ayuda a proteger la integridad de los datos y evita el acceso no deseado.

### Paso 2: Modificador public

Cuando un atributo o método se declara como public, puede ser accedido desde cualquier clase, independientemente del paquete en el que esté definida.

****

En este ejemplo, el atributo marca y el método mostrarMarca() son accesibles desde cualquier clase.

### Ejercicio 1: Creando un atributo y un método públicos

Crea una clase Persona con un atributo nombre y un método saludar, ambos public. Luego, accede a estos desde otra clase.

### Paso 3: Modificador private

El modificador private solo permite el acceso a los miembros desde dentro de la misma clase. Ni otras clases ni subclases pueden acceder a esos miembros.

****

En este ejemplo, saldo es un atributo privado y no puede ser accedido directamente desde fuera de la clase.

### Pregunta 2:

¿Qué sucede si intentas acceder a un atributo private desde otra clase?

Si intentas acceder a un atributo private desde otra clase, obtendrás un error de compilación porque el atributo no es visible fuera de la clase en la que fue declarado.

### Ejercicio 2: Atributo privado

Crea una clase Empleado con un atributo salario que sea private. Añade un método public que permita acceder al valor de salario.

### Paso 4: Modificador protected

El modificador protected permite que un atributo o método sea accesible dentro del mismo paquete y desde subclases, incluso si las subclases están en diferentes paquetes.



En este ejemplo, especie es un atributo protected, lo que significa que la clase Perro, que es una subclase de Animal, puede acceder a él.

### Pregunta 3:

¿Qué diferencia hay entre protected y private?

La diferencia es que protected permite que el atributo o método sea accesible desde subclases o dentro del mismo paquete, mientras que private restringe el acceso solo a la clase en la que se define.

### Ejercicio 3: Atributo protegido

Crea una clase Vehiculo con un atributo protected llamado tipo. Luego, crea una subclase Coche que acceda a tipo y lo imprima.

### Paso 5: Combinando Modificadores de Acceso

Puedes combinar los modificadores de acceso para controlar qué partes de una clase son visibles. Por ejemplo, puedes tener atributos private y métodos public para acceder a ellos, lo que es una buena práctica conocida como **encapsulamiento**.



En este ejemplo, el atributo saldo es privado, pero se puede modificar y consultar a través de los métodos públicos depositar() y getSaldo().

### Pregunta 4:

¿Por qué es recomendable usar atributos private en lugar de public?

Es recomendable usar atributos private para proteger el estado interno de la clase y controlar cómo se accede o modifica ese estado, permitiendo que solo métodos específicos (públicos o protegidos) realicen esas operaciones.

### Ejercicio 4: Encapsulamiento

Crea una clase CuentaBancaria con un atributo privado saldo y métodos públicos depositar y retirar. Asegúrate de que el saldo no pueda quedar negativo.

**Final en clases, métodos y atributos.**

**Paso 1: ¿Qué es final en Java?**

La palabra clave final se puede usar en tres contextos principales:

1. **Clases**: Una clase final no puede ser heredada.
2. **Métodos**: Un método final no puede ser sobrescrito (override) por las subclases.
3. **Atributos**: Un atributo final se convierte en una **constante**, es decir, no puede ser modificado después de haber sido inicializado.

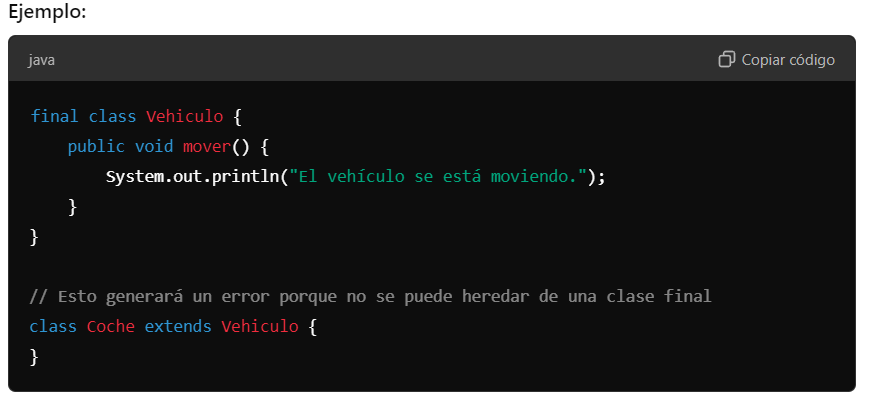
**Pregunta 1:**

¿Por qué es útil la palabra clave final en Java?

final es útil para garantizar la **inmutabilidad** en los atributos y prevenir la **herencia** o modificación de clases y métodos que no deberían ser alterados, lo que aporta mayor seguridad y claridad al diseño del programa.

**Paso 2: final en Clases**

Cuando declaramos una clase como final, estamos indicando que esa clase no puede ser extendida. Esto significa que no se puede crear una subclase a partir de una clase final.



### Ejercicio 1: Creando una clase final

Crea una clase final llamada Persona con un método saludar. Luego, intenta crear una subclase que extienda Persona. ¿Qué sucede?

**Resultado**: El compilador te dará un error porque no se puede heredar de una clase final.

**Paso 3: final en Métodos**

Un método declarado como final no puede ser sobrescrito por una subclase. Esto es útil cuando deseas asegurarte de que un comportamiento específico de un método no sea alterado por clases que extiendan la clase original.



**Pregunta 2:**

¿Cuándo usarías un método final?

Usaría un método final cuando quiero asegurarme de que el comportamiento de un método en una clase base no sea modificado por ninguna subclase.

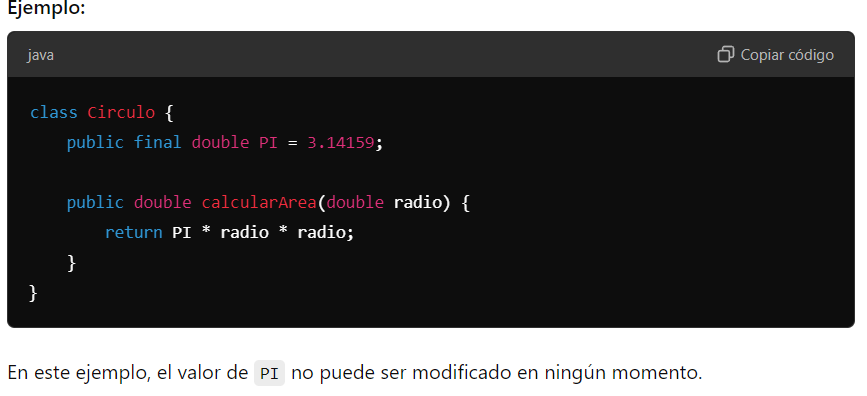
**Ejercicio 2: Creando un método final**

Crea una clase Empleado con un método final llamado trabajar. Luego, intenta sobrescribir ese método en una subclase llamada Gerente.

**Resultado**: El compilador te dará un error porque no puedes sobrescribir un método final.

**Paso 4: final en Atributos**

Cuando declaras un atributo como final, estás creando una constante. Esto significa que el valor del atributo no puede cambiar una vez que ha sido inicializado. Es importante inicializar los atributos final al momento de su declaración o en el constructor de la clase.



**Pregunta 3:**

¿Qué sucede si intentas modificar un atributo final después de haber sido asignado?

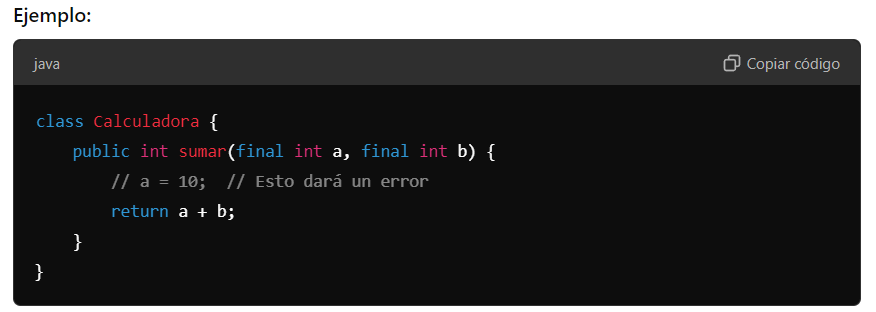
El compilador generará un error porque un atributo final no puede cambiar su valor después de haber sido inicializado. Es inmutable.

**Ejercicio 3: Creando un atributo final**

Crea una clase Rectangulo con un atributo final llamado LADOS, que siempre debe tener el valor 4. Luego, intenta cambiar el valor de LADOS en el código.

### Paso 5: final en Parámetros de Métodos

También puedes usar final en los parámetros de un método para asegurarte de que esos parámetros no cambien dentro del cuerpo del método.



**Pregunta 4:**

¿Por qué es útil declarar parámetros de método como final?

Declarar los parámetros como final es útil para asegurarse de que su valor no se modifique accidentalmente dentro del método, lo que hace el código más seguro y fácil de entender.

**Ejercicio 4: Parámetros final**

Crea un método multiplicar en la clase Calculadora que tome dos parámetros final y asegúrate de que no se puedan modificar dentro del método.

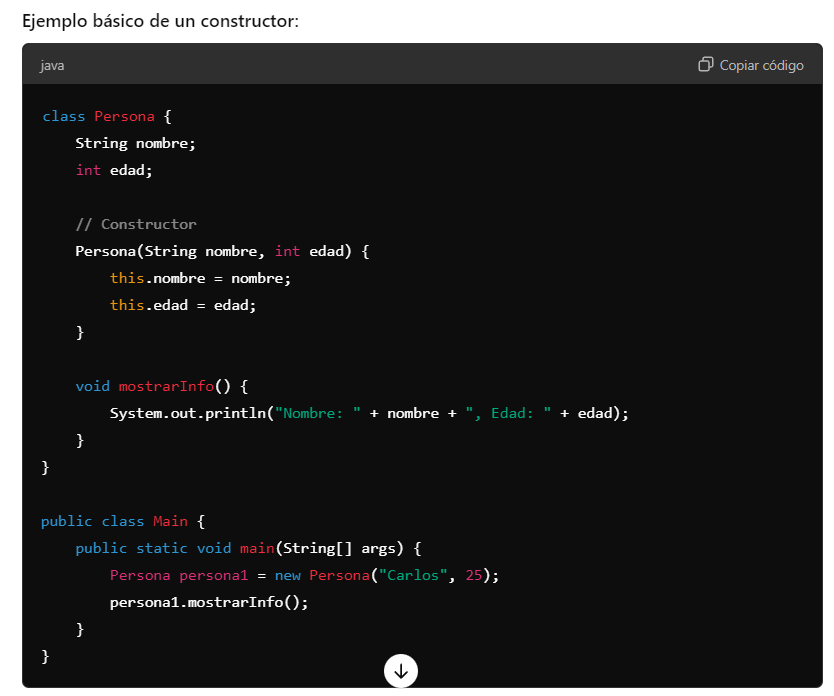
**Constructores en Java y Sobrecarga**

### Paso 1: ¿Qué es un Constructor?

Un constructor en Java es un bloque de código especial que se ejecuta cuando creamos un objeto de una clase. Su propósito principal es inicializar el objeto, asignando valores a sus atributos.

#### Características clave de los constructores:

* Tiene el mismo nombre que la clase.
* No tiene un valor de retorno, ni siquiera void.
* Se ejecuta automáticamente al crear un objeto.

****

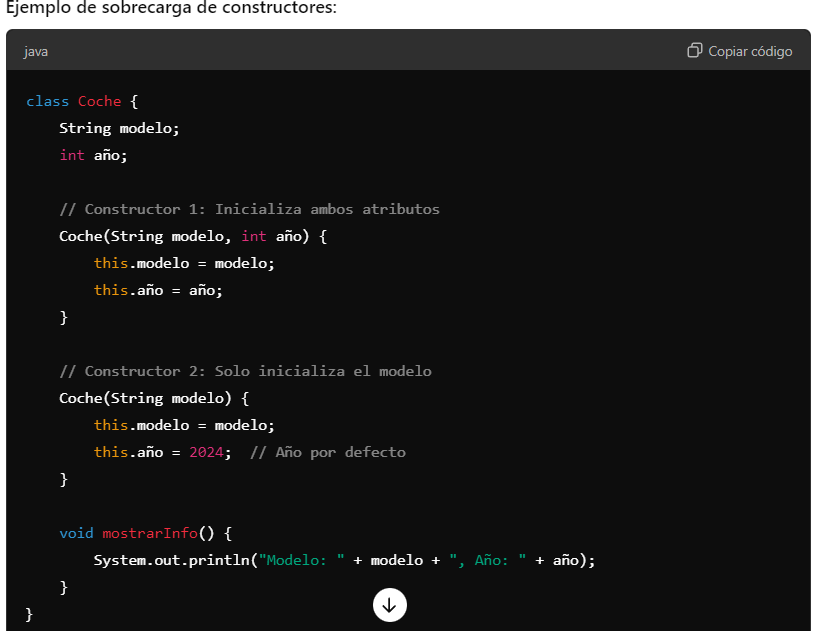
**Pregunta 1:**

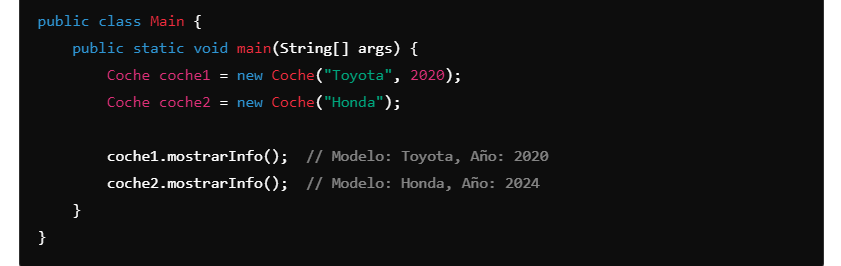
¿Qué hace el constructor en el ejemplo anterior?

El constructor inicializa los atributos nombre y edad del objeto Persona con los valores proporcionados cuando se crea el objeto (en este caso, "Carlos" y 25).

**Paso 2: Sobrecarga de Constructores**

La **sobrecarga de constructores** te permite definir más de un constructor en una clase, siempre y cuando tengan diferentes parámetros. Esto proporciona flexibilidad, ya que puedes crear objetos de una clase con distintos datos de inicialización.





**Pregunta 2:**

¿Por qué es útil la sobrecarga de constructores?

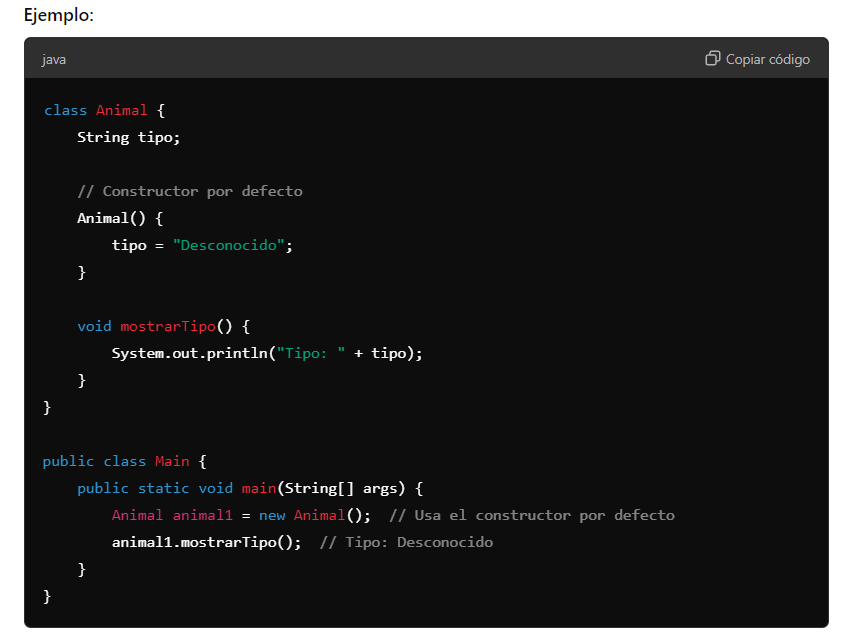
La sobrecarga de constructores es útil porque te permite crear objetos de la misma clase con diferentes conjuntos de datos iniciales, ofreciendo flexibilidad en la creación de instancias.

**Ejercicio 1: Creando constructores**

Crea una clase llamada Libro con los atributos titulo y autor. Define dos constructores: uno que inicialice ambos atributos y otro que solo inicialice el titulo, asignando un valor predeterminado al autor.

### Paso 3: Constructores por defecto

Si no defines un constructor en tu clase, Java proporciona automáticamente un **constructor por defecto** que no toma parámetros y no realiza ninguna inicialización explícita. Sin embargo, si defines un constructor con parámetros, el constructor por defecto deja de existir a menos que lo definas tú mismo.



**Pregunta 3:**

¿Qué sucede si defines un constructor con parámetros pero no un constructor por defecto? ¿Podrías crear un objeto sin pasar argumentos?

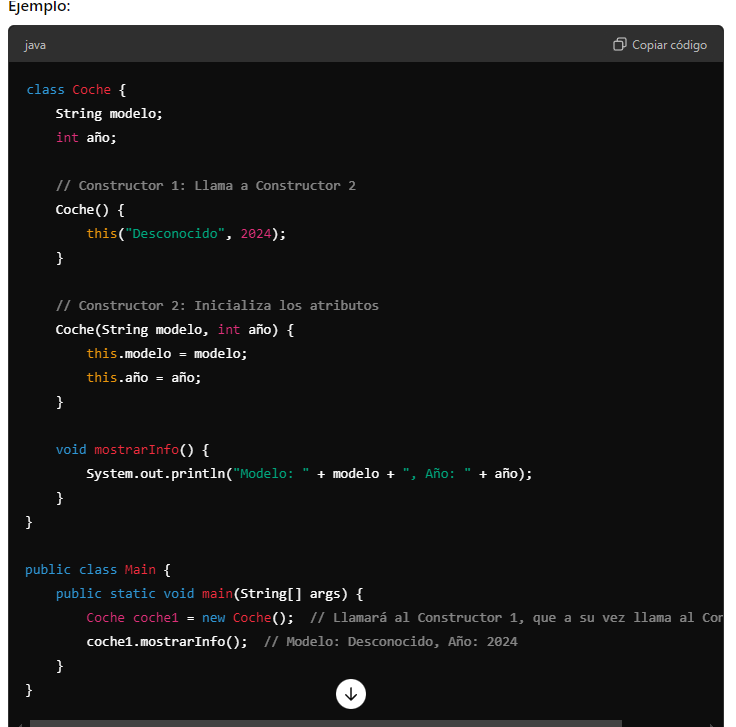
Si defines un constructor con parámetros y no defines uno por defecto, **no** podrás crear un objeto sin pasar argumentos, ya que Java no creará el constructor por defecto automáticamente. Tendrás que definirlo explícitamente si lo necesitas.

**Ejercicio 2: Añadir un constructor por defecto**

A la clase Coche que creaste anteriormente, añade un **constructor por defecto** que inicialice el modelo a "Desconocido" y el año a 2024.

### Paso 4: this() en Constructores

Puedes llamar a otro constructor de la misma clase desde un constructor usando this(). Esto se llama **invocación de constructor** y te permite evitar la repetición de código.



**Pregunta 4:**

¿Por qué usarías this() en un constructor?

Usaría this() para **reutilizar código** y evitar la duplicación en los constructores, llamando a otro constructor de la misma clase con un conjunto predeterminado de parámetros.

**Ejercicio 3: Usando this() en constructores**

Modifica la clase Libro para que el constructor que solo inicializa el titulo llame al constructor principal usando this().

**Herencia en Java y Polimorfismo**

### Paso 1: ¿Qué es la Herencia?

La **herencia** es un mecanismo que permite a una clase adquirir las propiedades (atributos y métodos) de otra clase. En Java, esto se hace con la palabra clave extends. La clase que hereda es llamada **subclase** (o clase derivada), y la clase de la que hereda es llamada **superclase** (o clase base).

### Pregunta 1:

¿Qué significa que la clase Perro hereda de la clase Animal?

Significa que la clase Perro puede usar los atributos y métodos definidos en la clase Animal, como el método comer y el atributo nombre.

**Ejercicio 1: Implementar herencia**

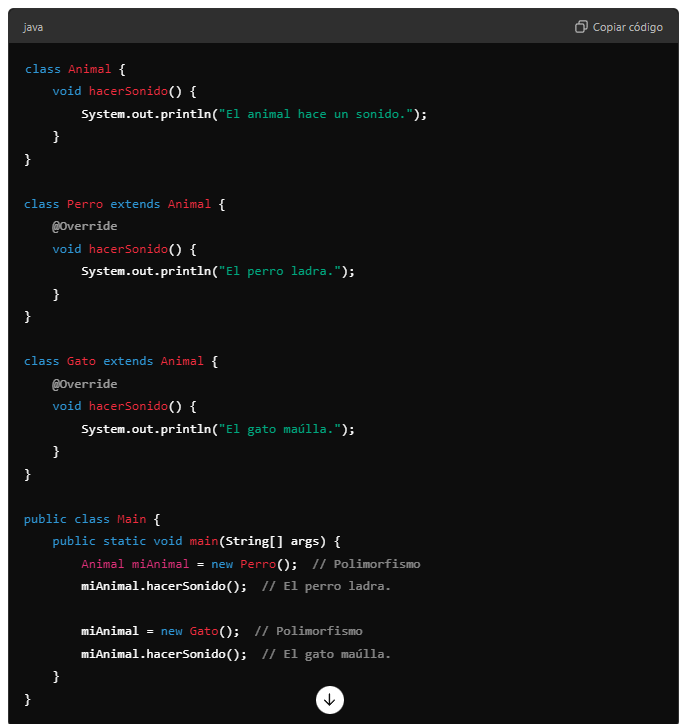
Crea una clase Vehiculo con los atributos marca y modelo, y un método mover(). Luego, crea una subclase Coche que herede de Vehiculo y tenga un método adicional tocarBocina().

### Paso 2: Polimorfismo

El **polimorfismo** permite a los objetos de diferentes clases responder al mismo método de maneras distintas. En Java, esto se logra utilizando **herencia** y **sobreescritura de métodos**.

#### Sobreescritura de métodos:

Las subclases pueden redefinir los métodos de la superclase para adaptarlos a sus necesidades específicas.



**Pregunta 2:**

¿Qué es la sobreescritura de métodos y cómo está relacionada con el polimorfismo?

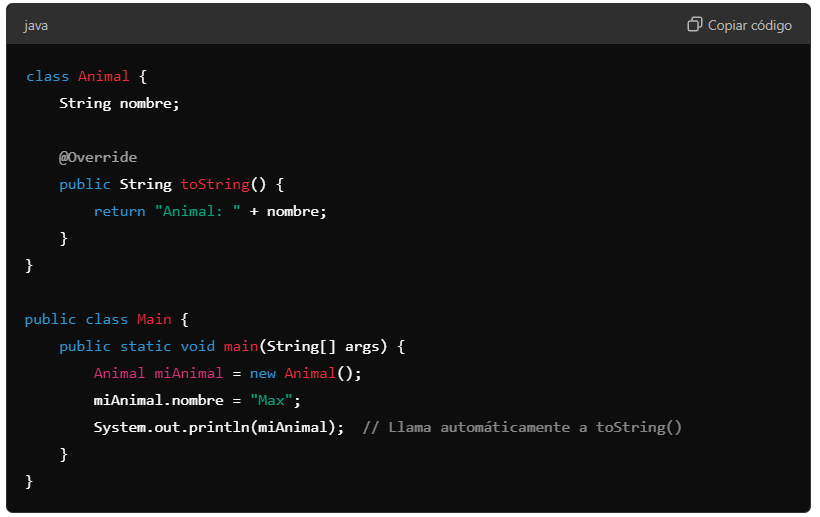
La sobreescritura de métodos permite a las subclases redefinir los métodos heredados de la superclase. Esto está relacionado con el polimorfismo porque, mediante la sobreescritura, diferentes subclases pueden proporcionar implementaciones distintas para el mismo método, y el comportamiento específico se decide en tiempo de ejecución.

**Ejercicio 2: Aplicar polimorfismo**

Usa las clases Vehiculo y Coche creadas anteriormente. Crea una nueva clase Moto que también herede de Vehiculo. Sobrescribe el método mover() en ambas subclases (Coche y Moto) para que impriman mensajes específicos de cada tipo de vehículo. Luego, usa polimorfismo para invocar el método mover() desde objetos de tipo Vehiculo.

### Paso 3: Clase Object y Polimorfismo

En Java, **todas las clases** heredan implícitamente de la clase Object. Esto significa que cualquier clase puede ser tratada como un objeto de tipo Object, y puedes usar los métodos definidos en Object, como toString() o equals().



**Pregunta 3:**

¿Por qué es importante la clase Object en Java?

**Respuesta:**

La clase Object es importante porque es la superclase de todas las clases en Java. Proporciona métodos comunes, como toString(), equals(), y hashCode(), que todas las clases pueden utilizar o sobrescribir.

**Ejercicio 3: Sobrescribir toString()**

Modifica las clases Coche y Moto para sobrescribir el método toString(), de modo que devuelvan una representación adecuada del objeto.

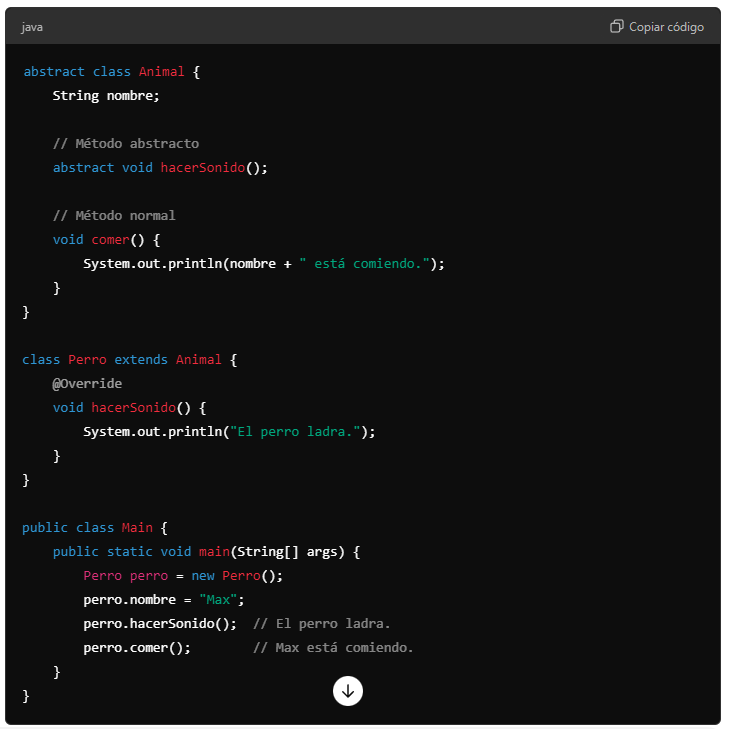
**Clases abstractas e interfaces.**

### Paso 1: Clases Abstractas

Una **clase abstracta** en Java es una clase que no se puede instanciar por sí misma. Sirve como una base para otras clases y puede contener métodos abstractos, que son métodos sin implementación que deben ser implementados por las subclases.

#### Conceptos Clave:

* **Clase Abstracta**: No puede ser instanciada directamente.
* **Método Abstracto**: Método sin cuerpo, que debe ser implementado en las subclases.
* **Método Normal**: Puede tener implementación.



**Pregunta 1:**

¿Qué diferencias hay entre una clase abstracta y una clase concreta?

Una clase abstracta no puede ser instanciada directamente y puede contener métodos abstractos, mientras que una clase concreta puede ser instanciada y debe proporcionar implementaciones para todos los métodos abstractos heredados.

**Ejercicio 1: Crear una Clase Abstracta**

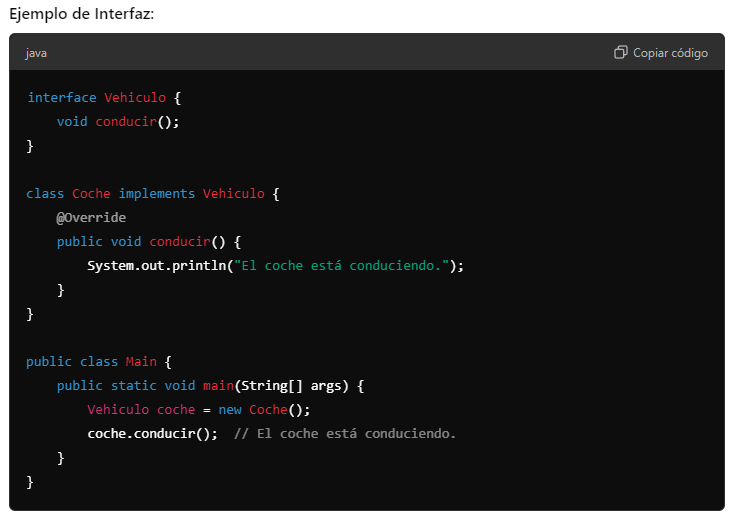
Crea una clase abstracta llamada Forma con un método abstracto calcularArea(). Luego, crea dos subclases concretas Cuadrado y Círculo que implementen el método calcularArea().

### Paso 2: Interfaces

Una **interfaz** en Java es un contrato que define un conjunto de métodos que las clases deben implementar. A diferencia de las clases abstractas, las interfaces no pueden contener métodos con implementación (hasta Java 8, cuando se introdujeron los métodos predeterminados).

#### Conceptos Clave:

* **Interfaz**: Define métodos que las clases implementarán.
* **Método de Interfaz**: Método sin cuerpo (hasta Java 8).
* **Método Predeterminado**: Método con cuerpo, disponible desde Java 8.



**Pregunta 2:**

¿Cuál es la diferencia principal entre una clase abstracta y una interfaz?

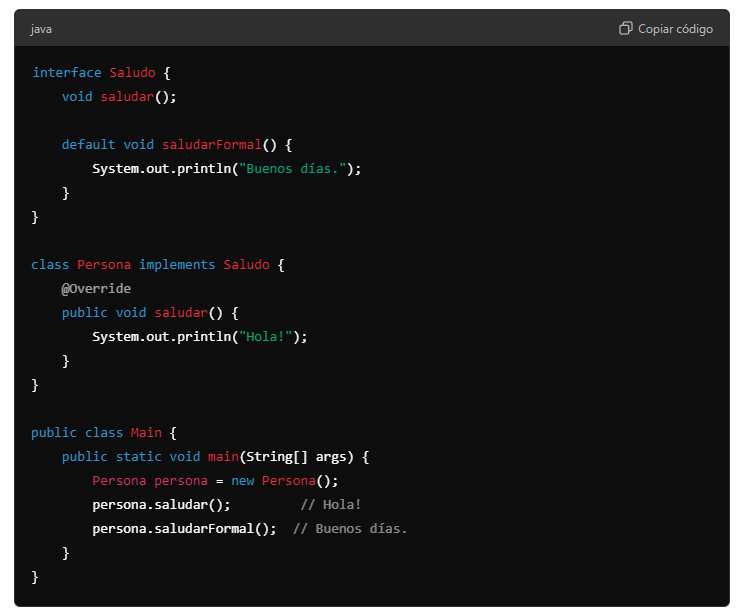
Una clase abstracta puede tener atributos y métodos con implementación, y se usa cuando existe una relación "es-un". Una interfaz define un contrato que las clases deben seguir y se usa para especificar capacidades que las clases deben implementar, sin definir una relación de herencia.

**Ejercicio 2: Implementar una Interfaz**

Crea una interfaz Imprimible con un método imprimir(). Luego, crea dos clases, Documento y Foto, que implementen la interfaz y proporcionen su propia implementación del método imprimir().

### Paso 3: Métodos Predeterminados en Interfaces

Desde Java 8, las interfaces pueden tener métodos predeterminados, que tienen una implementación por defecto.



**Pregunta 3:**

¿Qué ventajas ofrecen los métodos predeterminados en interfaces?

Los métodos predeterminados permiten añadir nuevas funcionalidades a las interfaces sin afectar a las clases que ya las implementan, lo que facilita la evolución de las interfaces en el tiempo.

**Ejercicio 3: Usar Métodos Predeterminados**

Modifica la interfaz Imprimible para incluir un método predeterminado prepararImpresion() que imprima un mensaje indicando que el objeto está listo para imprimir. Luego, usa esta interfaz en tus clases Documento y Foto.

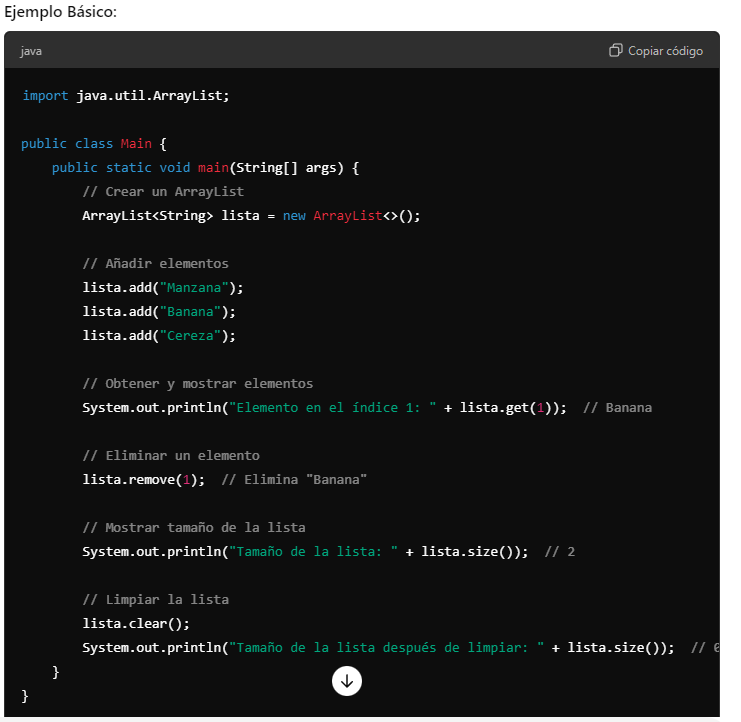
**Array List en java**

### Paso 1: Introducción a ArrayList

Un **ArrayList** es una implementación de la interfaz List en Java que permite almacenar elementos dinámicamente. A diferencia de los arreglos tradicionales, el tamaño de un ArrayList puede cambiar a medida que se añaden o eliminan elementos.

#### Conceptos Clave:

* **ArrayList**: Una colección que permite almacenar elementos y realizar operaciones de acceso y manipulación.
* **Métodos Comunes**:
  + add(E e): Añade un elemento al final de la lista.
  + get(int index): Obtiene el elemento en la posición especificada.
  + remove(int index): Elimina el elemento en la posición especificada.
  + size(): Devuelve el número de elementos en la lista.
  + clear(): Elimina todos los elementos de la lista.

****

**Pregunta 1:**

¿Cuál es la principal diferencia entre un arreglo y un ArrayList en Java?

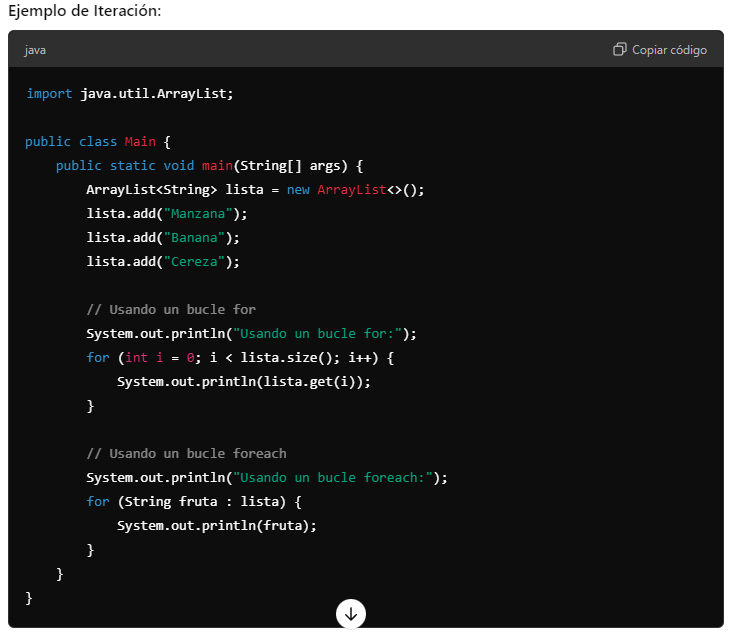
La principal diferencia es que un arreglo tiene un tamaño fijo, mientras que un ArrayList puede cambiar de tamaño dinámicamente a medida que se añaden o eliminan elementos.

**Ejercicio 1: Crear y Manipular un ArrayList**

Crea un ArrayList de enteros. Añade cinco números enteros a la lista, muestra el tamaño de la lista, elimina el segundo elemento, y luego muestra el contenido de la lista.

### Paso 2: Iterar sobre un ArrayList

Puedes iterar sobre un ArrayList utilizando bucles for y foreach. Esto es útil para acceder a cada elemento en la lista.



**Pregunta 2:**

¿Qué diferencias hay entre usar un bucle for y un bucle foreach para iterar sobre un ArrayList?

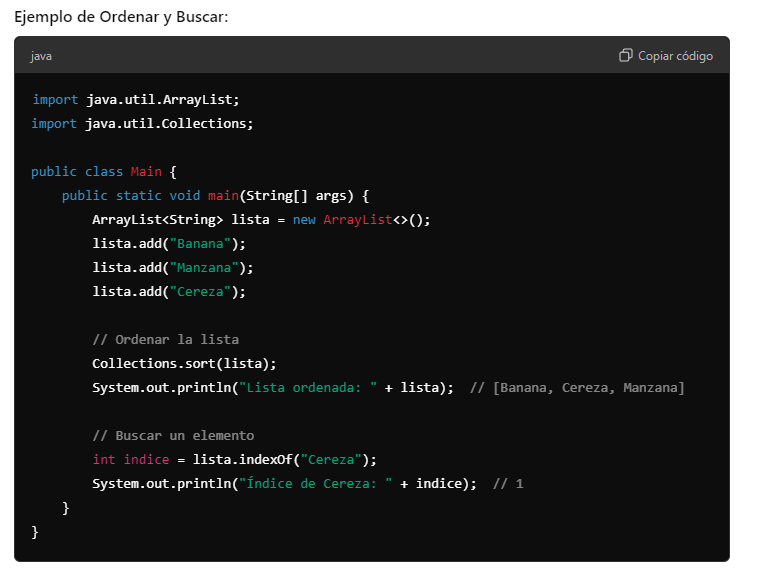
Un bucle for permite el acceso al índice de los elementos, lo que puede ser útil si necesitas la posición del elemento. Un bucle foreach es más sencillo y legible cuando solo necesitas acceder a los elementos, sin preocuparte por los índices.

**Ejercicio 2: Iterar sobre un ArrayList**

Crea un ArrayList de nombres de estudiantes. Utiliza un bucle foreach para imprimir cada nombre en la consola. Luego, usa un bucle for para imprimir los nombres en orden inverso.

### Paso 3: Otras Operaciones con ArrayList

* **Ordenar la lista**: Puedes usar Collections.sort() para ordenar un ArrayList de elementos comparables.
* **Buscar un elemento**: Puedes usar el método indexOf() para encontrar la posición de un elemento.



**Pregunta 3:**

¿Cómo puedes ordenar un ArrayList de objetos personalizados en Java?

Para ordenar un ArrayList de objetos personalizados, la clase del objeto debe implementar la interfaz Comparable y proporcionar una implementación del método compareTo(). Alternativamente, puedes usar un Comparator para definir el criterio de ordenación.

**Ejercicio 3: Ordenar un ArrayList de Objetos Personalizados**

Define una clase Persona con atributos nombre y edad. Crea una lista de Personas, ordénala por edad y muestra el resultado.

**Agregación de clases en java**

### Paso 1: Introducción a la Agregación

La **agregación** es una forma de asociación en la que una clase contiene referencias a objetos de otra clase. A diferencia de la composición, en la agregación, el ciclo de vida de los objetos asociados no está estrictamente ligado al ciclo de vida del objeto que los contiene. En otras palabras, un objeto de una clase puede existir sin el objeto de la clase agregada.

#### Conceptos Clave:

* **Agregación**: Relación en la que una clase (containing class) contiene referencias a objetos de otra clase (contained class).
* **Ejemplo Real**: Un Departamento que tiene Empleados. El Departamento puede existir sin Empleados, y un Empleado puede existir fuera del Departamento.

****

**Pregunta 1:**

¿Qué diferencia hay entre agregación y composición en Java?

La diferencia principal es que en la **agregación**, el ciclo de vida de los objetos contenidos no está estrictamente ligado al ciclo de vida del objeto que los contiene. En la **composición**, el ciclo de vida de los objetos contenidos está ligado al ciclo de vida del objeto contenedor, es decir, si el objeto contenedor se destruye, los objetos contenidos también se destruyen.

**Ejercicio 1: Crear Clases con Agregación**

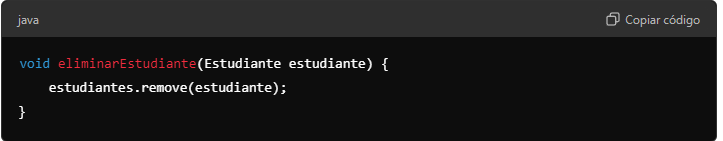
Crea una clase Curso que tenga una lista de Estudiantes. Luego, añade métodos para añadir y listar estudiantes en el curso.

### Paso 2: Relaciones entre Clases

La agregación permite que una clase (como Curso) mantenga una lista de objetos de otra clase (como Estudiantes). Esta relación permite que las clases sean más modulares y reutilizables.

#### Ejemplo de Modificación:

Puedes modificar la clase Curso para permitir la eliminación de estudiantes.



**Pregunta 2:**

¿En qué situaciones es más adecuado usar agregación en lugar de composición?

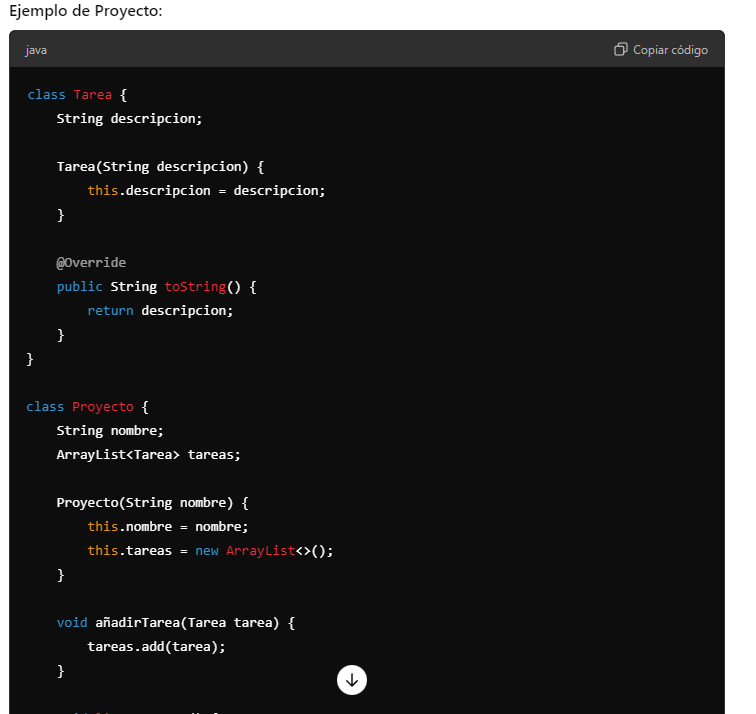
La agregación es adecuada cuando la relación entre las clases no implica un ciclo de vida dependiente. Por ejemplo, un Curso y Estudiantes tienen una relación de agregación porque un Estudiante puede existir sin estar en un Curso, y un Curso puede existir sin estudiantes.

**Ejercicio 2: Ampliar Funcionalidad**

Modifica la clase Curso para permitir la eliminación de un estudiante y para comprobar si un estudiante está en el curso antes de añadirlo.

### Paso 3: Uso de Agregación en Proyectos Reales

En proyectos reales, la agregación te permite modelar relaciones entre clases de manera flexible. Por ejemplo, en un sistema de gestión de proyectos, podrías tener una clase Proyecto que contiene una lista de Tareas, y cada Tarea podría existir sin un Proyecto.





**Funciones de Cadena en Java**

### Paso 1: Introducción a las Cadenas en Java

Una cadena en Java es un objeto de la clase String. Las cadenas son inmutables, lo que significa que una vez que se crea una cadena, no se puede modificar. Sin embargo, se pueden crear nuevas cadenas basadas en la original.

#### Conceptos Clave:

* **Inmutabilidad**: Las cadenas no pueden ser modificadas una vez creadas.
* **Métodos de String**: Funciones que permiten manipular cadenas de texto.

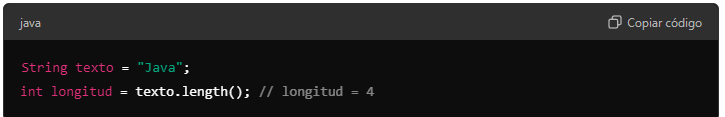
****

### Paso 2: Funciones Básicas de Cadena

A continuación, exploraremos algunas de las funciones más comunes de la clase String.

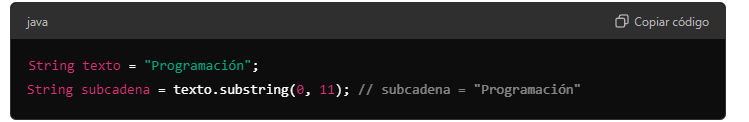
#### ****1. Longitud de la Cadena****

El método length() devuelve la longitud de una cadena.



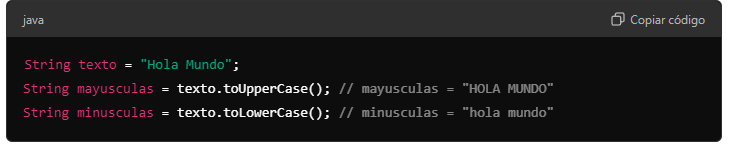
#### ****2. Subcadena****

El método substring(int beginIndex, int endIndex) devuelve una nueva cadena que es una subcadena de la original.



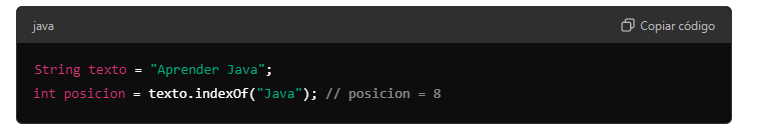
#### ****3. Convertir a Mayúsculas y Minúsculas****

Los métodos toUpperCase() y toLowerCase() convierten la cadena a mayúsculas o minúsculas, respectivamente.



#### ****4. Buscar Subcadena****

El método indexOf(String str) devuelve la posición de la primera aparición de la subcadena.



#### ****5. Reemplazar Texto****

El método replace(CharSequence target, CharSequence replacement) reemplaza todas las apariciones de una subcadena por otra.



### Pregunta 1:

¿Qué método usarías para obtener una subcadena a partir de un índice de inicio y un índice de final?

Utilizarías el método substring(int beginIndex, int endIndex).

### Ejercicio 1: Manipulación Básica de Cadenas

Escribe un programa que:

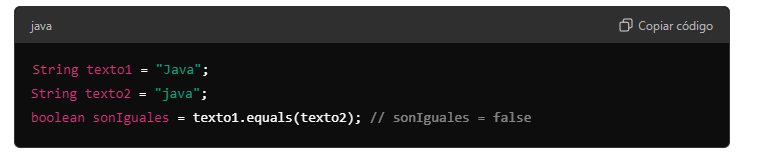
1. Cree una cadena con el texto "Aprender Java es divertido".
2. Obtenga la longitud de la cadena.
3. Extraiga la palabra "divertido" usando substring.
4. Reemplace "Java" por "Python".
5. Convierta la cadena a mayúsculas y muéstrala en la consola.

### Paso 3: Métodos Avanzados de Cadenas

Vamos a explorar algunos métodos adicionales útiles para manipular cadenas.

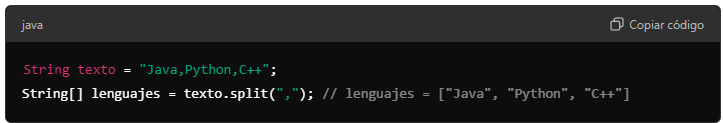
#### ****1. Comparar Cadenas****

El método equals(String anotherString) compara dos cadenas para ver si son iguales.



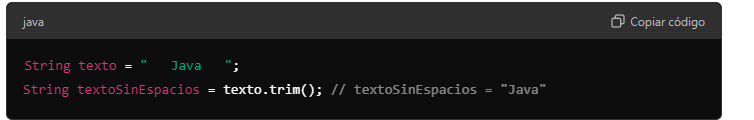
#### ****2. Dividir Cadenas****

El método split(String regex) divide la cadena en un array de subcadenas usando una expresión regular.



#### ****3. Eliminación de Espacios****

El método trim() elimina los espacios en blanco al principio y al final de la cadena.



### Pregunta 2:

¿Cómo puedes dividir una cadena en un array de subcadenas?

Utilizas el método split(String regex) de la clase String.

### Ejercicio 2: Manipulación Avanzada de Cadenas

Escribe un programa que:

1. Cree una cadena con el texto " Java, Python, C++, Ruby ".
2. Divida la cadena en un array usando la coma como delimitador.
3. Elimine los espacios en blanco de cada subcadena.
4. Imprima cada lenguaje en una nueva línea.

### Paso 4: Aplicaciones Prácticas

La manipulación de cadenas es útil en muchas aplicaciones, como:

* **Procesamiento de texto**: Limpiar y formatear datos.
* **Validación de entrada**: Comprobar formatos o valores.
* **Generación de informes**: Crear y formatear textos para salidas.

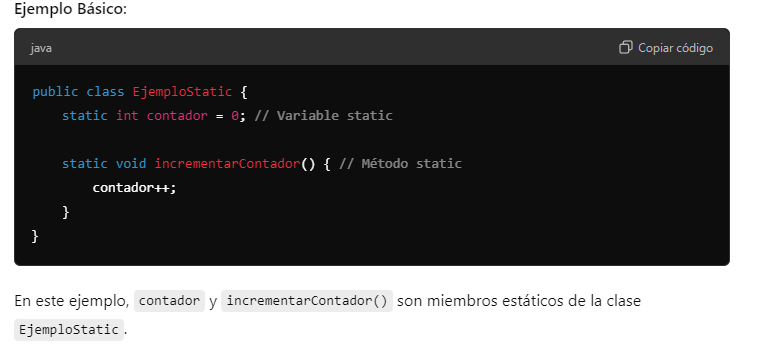
**Métodos y variables static**

### Paso 1: Introducción a static

En Java, los miembros static pertenecen a la clase en lugar de a una instancia específica de la clase. Esto significa que se pueden acceder sin necesidad de crear una instancia de la clase.

#### Conceptos Clave:

* **Variable static**: Una variable que es compartida por todas las instancias de una clase. Solo hay una copia de la variable, y se comparte entre todas las instancias.
* **Método static**: Un método que pertenece a la clase en lugar de a un objeto. Puede ser llamado sin crear una instancia de la clase.

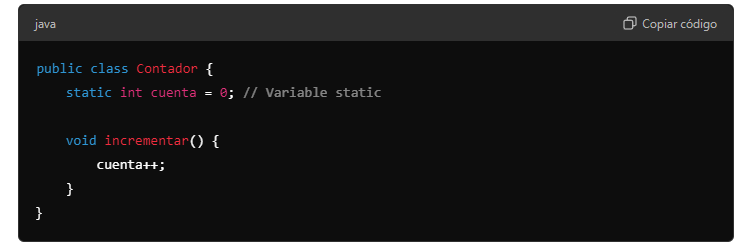


### Paso 2: Uso de Variables static

Las variables static se utilizan cuando se necesita una variable que sea común a todas las instancias de una clase.

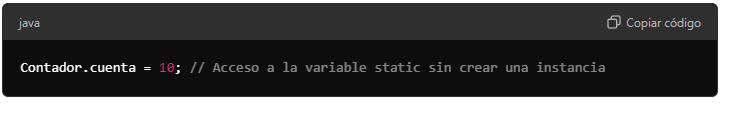
#### ****1. Definición y Acceso****

Una variable static se define dentro de la clase pero fuera de cualquier método.



#### ****2. Acceso a Variables**** static

Puedes acceder a una variable static utilizando el nombre de la clase.



### Pregunta 1:

¿Cuál es la diferencia entre una variable static y una variable de instancia?

Una variable static es compartida entre todas las instancias de la clase, mientras que una variable de instancia es única para cada objeto creado de la clase.

### Ejercicio 1: Variables static

Escribe un programa que:

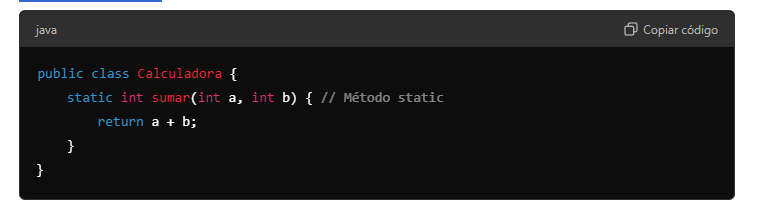
1. Defina una clase Banco con una variable static llamada saldoTotal y un método static llamado mostrarSaldoTotal().
2. Crea dos objetos de la clase Banco y modifica la variable saldoTotal usando uno de los objetos.
3. Llama al método mostrarSaldoTotal() para mostrar el saldo total.

### Paso 3: Uso de Métodos static

Los métodos static se utilizan para operaciones que no dependen del estado de una instancia específica de la clase.

#### ****1. Definición y Acceso****

Un método static se define dentro de la clase y puede ser llamado sin necesidad de crear una instancia de la clase.



#### ****2. Llamada a Métodos**** static

Puedes llamar a un método static utilizando el nombre de la clase.



### Pregunta 2:

¿Por qué utilizarías un método static en lugar de un método de instancia?

Utilizarías un método static cuando el método no necesita acceder a los datos de una instancia específica de la clase. Los métodos static son útiles para funciones utilitarias que deben ser accesibles sin crear una instancia de la clase.

### Ejercicio 2: Métodos static

Escribe un programa que:

1. Defina una clase Matematica con un método static llamado esPar(int numero) que devuelva true si el número es par y false si es impar.
2. Llama al método esPar desde el método main para verificar si varios números son pares o impares.

### Paso 4: Aplicaciones Prácticas de static

Los miembros static son útiles en situaciones como:

* **Contadores globales**: Contar el número de instancias creadas.
* **Métodos utilitarios**: Funciones matemáticas o de utilidad que no dependen del estado de un objeto.
* **Configuraciones compartidas**: Valores que deben ser comunes a todas las instancias de una clase.