**Clases**:

Hablar de package

Constructores

Imports

Clases y objetos

Como declarar

Ejercicios

Hablar null

**Que es una clase:**

Toda clase es un objeto en Java, pero no todo objeto es una clase

* **Clase**: En Java, una clase es como un plano o una plantilla que define las propiedades y comportamientos de un objeto. Por ejemplo, si estás modelando un coche, la clase "Coche" podría definir sus atributos (como marca, modelo, color) y métodos (como acelerar, frenar).
* Una clase en Java es un modelo que define el comportamiento y las propiedades de un objeto. Se compone de atributos y métodos que describen el objeto. Las clases permiten la reutilización de código a través de la herencia.
* Las clases pueden tener relaciones como asociación, composición y herencia. Estas relaciones modelan la interacción entre objetos. La composición implica que un objeto está compuesto por otros objetos

[visibilidad] = public, protected o private

[modificadores] = final, static y abstract

public class Coche {

// Atributos

String marca;

String modelo;

String color;

// Método para acelerar

public void acelerar() {

// Código para aumentar la velocidad

}

// Método para frenar

public void frenar() {

// Código para reducir la velocidad

}

}

* **Objeto**: Un objeto es una instancia específica de una clase. Usando la clase "Coche" como ejemplo, un objeto podría ser un coche en particular con una marca, modelo y color específicos.

Coche miCoche = new Coche(); // Creación de un objeto de la clase Coche

miCoche.marca = "Toyota";

miCoche.modelo = "Corolla";

miCoche.color = "Rojo";

miCoche.acelerar(); // Llamada a un método del objeto

**Tipos de clase:**

**Clases regulares**: Son las clases más comunes que se utilizan para definir objetos y sus comportamientos. Estas clases pueden tener atributos y métodos.

**Clases finales**: Las clases finales son aquellas que no se pueden heredar, es decir, no se pueden extender. Esto significa que no se pueden crear subclases a partir de una clase final.

**Clases internas**: Java permite definir clases dentro de otras clases. Estas clases se conocen como clases internas o clases anidadas. Pueden ser estáticas o no estáticas y se utilizan para mejorar la encapsulación y la estructura del código.

**Clases Wrapper**: Estas clases se utilizan para envolver tipos de datos primitivos en objetos. Por ejemplo, Integer, Double, Boolean, etc., que proporcionan funcionalidades adicionales a los tipos primitivos.

Ejemplos:

1. **Clases regulares:**

public class Persona {

// Atributos

String nombre;

int edad;

// Método para mostrar información de la persona

public void mostrarInfo() {

System.out.println("Nombre: " + nombre + ", Edad: " + edad);

}

}

1. **Clases finales:**

public final class ClaseFinal {

// Contenido de la clase final

}

1. **Clases internas:**

public class OuterClass {

private int x = 10;

// Clase interna estática

static class InnerStaticClass {

void mostrarX() {

// Puede acceder a miembros estáticos de la clase externa

System.out.println("El valor de x es: " + new OuterClass().x);

}

}

// Clase interna no estática

class InnerClass { void mostrarX() {

// Puede acceder a miembros no estáticos de la clase externa

System.out.println("El valor de x es: " + x);

}

}

}

1. **Clases Wrapper:**

public class EjemploWrapper {

public static void main(String[] args) {

int num = 10; Integer numWrapper = Integer.valueOf(num);

// Envoltorio de tipo primitivo int

System.out.println("Número: " + numWrapper); // Salida: Número: 10 } }

**Conclusión**

Las clases y objetos son fundamentales en Java para la construcción de sistemas complejos. La programación orientada a objetos ofrece un enfoque modular y reutilizable para el desarrollo de software.

|  |  |  |
| --- | --- | --- |
| **MODIFICADOR DE ACCESO** | **EFECTO** | **APLICABLE A** |
| **private** | Restringe la visibilidad al **interior de la clase**. Un atributo o método definido como private solo puede ser usado en el interior de su propia clase. | Atributos  Métodos |
| **<Sin modificador>** | Cuando no se especifica un modificador, el elemento adquiere el *acceso por defecto o friendly*. También se le conoce como acceso de package (paquete). Solo puede ser usado por las **clases dentro de su mismo paquete**. | Clases  Atributos  Métodos |
| **protected** | Se emplea en la herencia. El elemento puede ser utilizado por **cualquier clase dentro de su paquete** **y por cualquier subclase** independientemente del paquete donde se encuentre. | Atributos  Métodos |
| **public** | Es el nivel máximo de visibilidad. El elemento es visible desde cualquier clase. | Clases  Atributos  Métodos |

|  |  |  |
| --- | --- | --- |
| **MODIFICADOR** | **EFECTO** | **APLICABLE A** |
| **abstract** | Aplicado a una clase, la declara como clase abstracta. No se pueden crear objetos de una clase abstracta. Solo pueden usarse como superclases.  Aplicado a un método, la definición del método se hace en las subclases. | Clases  Métodos |
| **final** | Aplicado a una clase significa que no se puede extender (heredar), es decir que no puede tener subclases.  Aplicado a un método significa que no puede ser sobrescrito en las subclases.  Aplicado a un atributo significa que contiene un valor constante que no se puede modificar | Clases  Atributos  Métodos |
| **static** | Aplicado a un atributo indica que es una variable de clase. Esta variable es única y compartida por todos los objetos de la clase.  Aplicado a un método indica que se puede invocar sin crear ningún objeto de su clase. | Atributos  Métodos |
| **volatile** | Un atributo volatile puede ser modificado por métodos no sincronizados en un entorno multihilo. | Atributos |
| **transient** | Un atributo transient no es parte del estado persistente de las instancias. | Atributos |
| **Sincronizad** | Métodos para entornos multihilo. | Métodos |