

Java

Clases

```
[modificadoresDeAcceso] class NombreDeClase
    [extends NombreSuperClase]
    [implements NombreInterfaces separadas por comas] {

    // Atributos
    [modificadoresDeAcceso] [tipoDato] atributo1;
    [modificadoresDeAcceso] [tipoDato] atributo2;
    // ...

    // Constructor(es)
    public NombreDeClase(/* parámetros */) {
        // Inicialización de miembros de datos y otras operaciones
    }

    // Método(s)
    [modificadoresDeAcceso] [tipoRetorno] nombreMetodo(/* parámetros */) {
        // Implementación del método
        // Puede incluir operaciones, lógica y más
    }

    // Otros métodos y elementos de la clase...
}
```

```
public class Coche {
    // Atributos
    String modelo;
    String color;
    int anioFabricacion;

    // Métodos
    void acelerar() {
        System.out.println("El coche está acelerando.");
    }

    void frenar() {
        System.out.println("El coche está frenando.");
    }
}
```

Clases abstractas

Una clase abstracta es una clase que no puede ser instanciada directamente. Esto significa que no puedes crear objetos directamente a partir de una clase abstracta mediante el operador new.

```
abstract class Figura {  
    // Definiciones de la clase abstracta Figura  
}
```

El propósito principal de una clase abstracta es proporcionar una abstracción común para un conjunto de clases relacionadas. Estas clases relacionadas pueden compartir comportamientos comunes a través de métodos abstractos. Una característica de las clases abstractas es la presencia de métodos abstractos. Un método abstracto es un método declarado sin una implementación en la clase abstracta.

```
abstract class Figura {  
    abstract void dibujar(); // Método abstracto  
}
```

Objetos

```
public class Coche {  
    // Atributos  
    String modelo;  
    String color;  
    int anioFabricacion;  
  
    // Métodos  
    void acelerar() {  
        System.out.println("El coche está acelerando.");  
    }  
  
    void frenar() {  
        System.out.println("El coche está frenando.");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        // Creación de objetos  
        Coche miCoche = new Coche();  
        Coche cocheVecino = new Coche();  
  
        // Acceso a atributos y métodos  
        miCoche.modelo = "Sedán";  
        miCoche.color = "Azul";  
        miCoche.anioFabricacion = 2022;  
  
        cocheVecino.modelo = "SUV";  
        cocheVecino.color = "Rojo";  
        cocheVecino.anioFabricacion = 2021;  
  
        miCoche.acelerar();  
        cocheVecino.frenar();  
    }  
}
```

Modificadores de acceso

Público

Los atributos o métodos definidos como públicos son accesibles desde cualquier parte del programa, ya sea dentro de la misma clase, en clases del mismo paquete o desde clases externas.

```
public class Estudiante {  
    // Atributo Público  
    public String nombre;  
  
    // Método Público que muestra un saludo con el nombre del estudiante  
    public void saludar() {  
        System.out.println("¡Hola! Soy el estudiante " + nombre);  
    }  
}
```

```
public class ProgramaPrincipal {  
  
    public static void main(String[] args) {  
        // Crear una instancia de la clase Estudiante  
        Estudiante estudiante1 = new Estudiante();  
  
        // Establecer el nombre del estudiante  
        estudiante1.nombre = "Juan";  
  
        // Llamar al método público para saludar  
        estudiante1.saludar();  
    }  
}
```

Modificadores de acceso

Privado

Los atributos o métodos declarados como privados solo son accesibles dentro de la propia clase donde se definen.

```
public class Estudiante {  
    // Miembro de Datos Privado  
    private int edad;  
}
```

Protegido

Los atributos o métodos protegidos son accesibles desde la misma clase, clases del mismo paquete y clases hijas (subclases), pero no desde clases externas que no sean subclases.

```
public class Estudiante {  
    // Miembro de Datos Protegido  
    protected boolean esEstudianteActivo;  
}
```

Modificadores de acceso

Sin Modificador

Si no se especifica un modificador de acceso, el atributo o método es accesible solo dentro del mismo paquete.

```
class Estudiante {  
    // Miembro de Datos con Acceso por Paquete  
    String numeroDeEstudiante;  
}
```

Atributos

```
public class Ejemplo {  
    // Declaración de Atributos  
    TipoDato nombreMiembro1;  
    TipoDato nombreMiembro2;  
    // ...  
}
```

```
public class Estudiante {  
    // Atributos con Inicialización al Declarar  
    private String nombre = "Sin Nombre";  
    // ...  
}
```

```
public class Estudiante {  
    // Atributos  
    private String nombre;  
  
    // Constructor que inicializa el miembro de datos  
    public Estudiante(String nombreInicial) {  
        this.nombre = nombreInicial;  
    }  
}
```


Métodos

```
public class Circulo {  
    // Atributo  
    private double radio;  
  
    // Método para obtener el radio  
    public double getRadio() {  
        return radio;  
    }  
  
    // Método para establecer el radio  
    public void setRadio(double radio) {  
        this.radio = radio;  
    }  
  
    // Método para calcular el área del círculo  
    public double calcularArea() {  
        return Math.PI * Math.pow(radio, 2);  
    }  
}
```

Sobrecarga

```
public class Circulo {  
    // Atributo  
    double radio;  
  
    // Método para calcular el área del círculo  
    double calcularArea() {  
        return Math.PI * Math.pow(radio, 2);  
    }  
  
    // Sobrecarga del método para calcular el perímetro del círculo  
    double calcularPerimetro() {  
        return 2 * Math.PI * radio;  
    }  
  
    // Sobrecarga del método para establecer el radio del círculo  
    void setRadio(double nuevoRadio) {  
        radio = nuevoRadio;  
    }  
  
    // Sobrecarga del método para establecer el radio con un valor por defecto  
    void setRadio() {  
        radio = 1.0; // Valor por defecto si no se proporciona un nuevo radio  
    }  
}
```

Constructores

Un constructor es un método especial que se llama automáticamente cuando se crea un objeto a partir de una clase. Su función principal es asignar valores iniciales a los atributos de la clase y realizar cualquier inicialización necesaria.

```
public class Persona {  
    // Atributos  
    String nombre;  
    int edad;  
  
    // Constructor sin parámetros  
    public Persona() {  
        nombre = "Sin Nombre";  
        edad = 0;  
    }  
  
    // Constructor con parámetros  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

Constructores

La palabra clave `this` se utiliza dentro de un constructor para referirse a la instancia actual de la clase. En el segundo constructor del ejemplo, `this.nombre` y `this.edad` se utilizan para distinguir entre los parámetros del constructor y los atributos de la clase.

```
public class Persona {  
    // Atributos  
    String nombre;  
    int edad;  
  
    // Constructor sin parámetros  
    public Persona() {  
        nombre = "Sin Nombre";  
        edad = 0;  
    }  
  
    // Constructor con parámetros  
    public Persona(String nombre, int edad) {  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
}
```

Constructores

Java permite la sobrecarga de constructores, lo que significa que una clase puede tener varios constructores con diferentes listas de parámetros.

- **Inicialización coherente:** Los constructores garantizan que los objetos se creen en un estado inicial coherente.
- **Flexibilidad en la Creación de Objetos:** La sobrecarga de constructores permite crear objetos con diferentes configuraciones según los requisitos del programa.
- **Evita Estados Inválidos:** Los constructores ayudan a evitar que los objetos se encuentren en estados inválidos al proporcionar valores iniciales adecuados.
- **Mejora la Claridad del Código:** El uso de constructores claros y bien definidos mejora la legibilidad y comprensión del código.

Ejemplo 1

Nos piden hacer un programa que gestione empleados. Los empleados se definen por tener:

- Nombre
- Edad
- Salario

También tendremos una constante llamada PLUS, que tendrá un valor de \$300. Tenemos dos tipos de empleados: repartidor y comercial. El comercial, aparte de los atributos anteriores, tiene uno más llamado comisión (double). El repartidor, aparte de los atributos de empleado, tiene otro llamado zona (String).

Ejemplo 1

Crea sus constructores, getters and setters y toString.

No se podrán crear objetos del tipo Empleado (la clase padre) pero si de sus hijas.

Las clases tendrán un método llamado plus, que según en cada clase tendrá una implementación distinta. Este plus básicamente aumenta el salario del empleado.

- En comercial, si tiene más de 30 años y cobra una comisión de más de \$200, se le aplicara el plus.
- En repartidor, si tiene menos de 25 y reparte en la “zona 3”, este recibirá el plus.

Puedes hacer que devuelva un booleano o que no devuelva nada, opcional.

Crea una clase ejecutable donde crees distintos empleados y le apliques el plus para comprobar que funciona.

Ejercicio 1

Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular y cantidad (puede tener decimales).

El titular será obligatorio y la cantidad es opcional. Crea dos constructores que cumpla lo anterior.

Crea sus métodos get, set y toString.

Tendrá dos métodos especiales:

- ingresar(double cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- retirar(double cantidad): se retira una cantidad a la cuenta, si restando la cantidad actual a la que nos pasan es negativa, la cantidad de la cuenta pasa a ser 0.

Ejercicio 2

Haz una clase llamada Persona que siga las siguientes condiciones:

- Sus atributos son: nombre, edad, DNI, sexo (H hombre, M mujer), peso y altura. No queremos que se accedan directamente a ellos. Piensa que modificador de acceso es el más adecuado, también su tipo. Si quieres añadir algún atributo puedes hacerlo.
- Por defecto, todos los atributos menos el DNI serán valores por defecto según su tipo (0 números, cadena vacía para String, etc.). Sexo será hombre por defecto, usa una constante para ello.
- Se implantaran varios constructores:
 - Un constructor por defecto.
 - Un constructor con el nombre, edad y sexo, el resto por defecto.
 - Un constructor con todos los atributos como parámetro.

Ejercicio 2

- Los métodos que se implementaran son:
 - `calcularIMC()`: calcula si la persona esta en su peso ideal (peso en $\text{kg}/(\text{altura}^2 \text{ en m})$), si esta fórmula devuelve un valor menor que 20, la función devuelve un -1, si devuelve un número entre 20 y 25 (incluidos), significa que esta por debajo de su peso ideal la función devuelve un 0 y si devuelve un valor mayor que 25 significa que tiene sobrepeso, la función devuelve un 1. Te recomiendo que uses constantes para devolver estos valores.
 - `esMayorDeEdad()`: indica si es mayor de edad, devuelve un booleano.
 - `comprobarSexo(char sexo)`: comprueba que el sexo introducido es correcto. Si no es correcto, será H. No será visible al exterior.
 - `toString()`: devuelve toda la información del objeto.
 - `generaDNI()`: genera un número aleatorio de 8 cifras, genera a partir de este su número su letra correspondiente. Este método será invocado cuando se construya el objeto. Puedes dividir el método para que te sea más fácil. No será visible al exterior.
 - Métodos set de cada parámetro, excepto de DNI.

Ejercicio 2

Ahora, crea una clase ejecutable que haga lo siguiente:

- Pide por teclado el nombre, la edad, sexo, peso y altura.
- Crea 3 objetos de la clase anterior, el primer objeto obtendrá las anteriores variables pedidas por teclado, el segundo objeto obtendrá todos los anteriores menos el peso y la altura y el último por defecto, para este último utiliza los métodos set para darle a los atributos un valor.
- Para cada objeto, deberá comprobar si esta en su peso ideal, tiene sobrepeso o por debajo de su peso ideal con un mensaje.
- Indicar para cada objeto si es mayor de edad.
- Por último, mostrar la información de cada objeto.

Puedes usar métodos en la clase ejecutable, para que sea mas fácil.

Referencias

- de Roer, D. D. (2013, December 20). Ejercicios propuestos y resueltos programación orientado a objetos Java. *Disco Duro de Roer* -. <https://www.discoduroderoer.es/ejercicios-propuestos-y-resueltos-programacion-orientado-a-objetos-java/>
- *Programación orientada a objetos*. (n.d.). Exercises Java. Retrieved August 13, 2024, from <https://www.exercisesjava.com/es/programacion-orientada-a-objetos/>