Material descargable

egg

Teoría **JAVA IX**

Programación Orientada a Objetos

La programación orientada a objetos es el paradigma de programación que utiliza "objetos" y sus interacciones para diseñar aplicaciones y programas de software. Los objetos son instancias de "clases", que pueden contener variables de instancia, métodos (funciones), y otros datos.

Clases e instancias

Una **clase** en la Programación Orientada a Objetos (OOP) es un modelo o plantilla que define las características (atributos o campos) y comportamientos (métodos) que un cierto tipo de objeto debe tener. Una clase es un conjunto de instrucciones que detallan cómo deben ser construidas las instancias de esa clase.

Una **instancia** de una clase es un objeto específico que ha sido creado a partir de esa clase utilizando el operador *new*. Cada objeto creado a partir de una clase (instanciado) tiene su propio conjunto de valores para los atributos que se definen en la clase. Cada instancia puede acceder a los métodos que se definen en la clase.

Modificador static

Una **variable estática** es una variable que pertenece a la clase, no a las instancias de la clase. Hay solo una copia de una variable estática, independientemente del número de objetos creados a partir de la clase. Esto significa que todos los objetos de la misma clase comparten la misma variable estática. Esto es útil cuando queremos tener una variable que conserve su valor y sea común a todas las instancias de la clase.

Un **método estático** es un método que pertenece a la clase, no a una instancia de la clase. Como resultado, los métodos estáticos no pueden acceder a las variables de instancia o los métodos de instancia porque estos requieren una instancia de la clase para existir. Los métodos estáticos suelen usarse para realizar operaciones que no requieren ningún dato específico del objeto.

```
public class Clase {
    Integer variableNoEstatica;
    static Integer variableEstatica;

    public static void metodoEstatico() {

    }

    public void metodoNoEstatico() {

    }
}
```

Se pueden **declarar** variables estáticas dentro de una clase, pero fuera de cualquier método, constructor o bloque. No pueden declararse dentro de métodos, constructores o bloques de código (a menos que estén en un bloque estático, pero ahí están limitadas a ese bloque).

Para **invocar** métodos y variables estáticas no puedes usar una variable que referencie a una instancia de la clase, tienes que invocarlos directamente con la clase.

Por ejemplo: `Clase.methodoEstatico()` o `Clase.variableEstatica`. Para invocar métodos y variables no estáticas, necesitas una variable que referencie a una instancia de la clase: `instancia.methodoNoEstatico()` o `instancia.variableNoEstatica`.

Palabra clave *this* y método Constructor

En Java, "this" es una palabra clave que se utiliza como referencia al objeto actual dentro de una clase.

Un **constructor** es un método especial que se utiliza para inicializar los objetos de una clase. Tiene el mismo nombre que la clase y no tiene ningún tipo de retorno, ni siquiera *void*.

El propósito principal de un constructor es establecer los valores iniciales de los campos de un objeto cuando se crea ese objeto.

En Java, si no declaramos explícitamente un constructor en una clase, el compilador proporciona automáticamente un constructor sin parámetros, conocido como constructor predeterminado o **constructor por defecto**. Este constructor no hace nada más que crear una instancia de la clase. Los campos de la instancia se inicializan con valores predeterminados: 0 para tipos numéricos, false para booleanos, y null para referencias de objetos.

```
public class Persona {
    String nombre;// propiedad de los objetos de la clase
    public Persona() {}// constructor por defecto, no hace falta declararlo
}
```

Si decides declarar un constructor (o varios constructores) con parámetros en tu clase, el compilador ya no proporcionará un constructor por defecto. Esto significa que debes proporcionar un constructor sin parámetros explícitamente si aún deseas poder crear objetos sin pasar ningún argumento.

A este comportamiento se le llama "**sobrecarga del constructor**", que es un tipo de "sobrecarga de métodos".

💡 *Recuerda que la sobrecarga de métodos se produce cuando se declaran varios métodos con el mismo nombre pero con diferentes listas de parámetros. La JVM determina qué método debe invocarse en tiempo de ejecución basándose en la cantidad y tipo de argumentos pasados.*

si tienes una clase con una variable de instancia llamada "nombre" y un método con un parámetro también llamado "nombre", puedes utilizar "this" para referirte a la variable de instancia:

Los constructores también pueden utilizar la palabra clave **this** para hacer referencia al objeto que se está construyendo. Además, puedes utilizar **this** seguido de paréntesis para invocar a otro constructor en la misma clase

(conocido como "llamada al constructor" o "delegación al constructor"). Por ejemplo:

```
public class Persona {
    private String nombre;
    private Integer edad;

    // constructor por defecto
    public Persona() {
        this("Desconocido", 0);
    }

    // constructor con parámetros
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

En el ejemplo, **this()** llama al constructor sin parámetros y **this("Desconocido", 0)** llama al constructor que toma *nombre* y *edad* como parámetros. Estas llamadas al constructor deben ser la primera línea en el constructor. Si creas una nueva instancia de Persona sin parámetros *new Persona()*, se llamará al constructor por defecto, que a su vez llama al constructor con parámetros, pasándole "Desconocido" como nombre y 0 como edad (de esta manera evitas que las propiedades se inicialicen con *null*). Si creas una nueva instancia con parámetros n*ew Persona("Juan", 30)*, se llamará directamente al constructor con parámetros.

Encapsulación y Ocultación

Encapsulación y ocultación de información son dos conceptos fundamentales de la programación orientada a objetos que ayudan a mantener la integridad de los datos y a hacer el código más manejable.

Cuando **encapsulamos**, estamos agrupando los datos (atributos) y las operaciones (métodos) que actúan sobre esos datos en una sola unidad, la clase. La **ocultación** de información ocurre cuando limitamos el acceso a los detalles internos de esa clase, ocultando sus atributos y permitiendo la interacción con ellos únicamente a través de los métodos que hemos definido tal caso. Esto se realiza a menudo utilizando **modificadores de acceso**, como

private, y proporcionando métodos públicos, como los **métodos getters y setters**, para acceder y modificar los datos.

Aplicar estos conceptos en la práctica ofrece varios beneficios importantes:

- **Control de Acceso:** Al utilizar métodos *getter* y *setter* en lugar de permitir el acceso directo a los atributos, tienes un control completo sobre cómo y cuándo se accede o modifica la información de un objeto. Por ejemplo, puedes realizar una validación en un método setter para asegurarte de que nadie pueda establecer un valor inválido para un atributo (luego explicaremos de qué manera se debe implementar la validación).
- **Flexibilidad y mantenimiento:** Al ocultar los detalles internos de cómo se implementa una clase, puedes cambiar la implementación en cualquier momento sin afectar a las otras partes del código que utilizan esa clase. Si otras partes del código usan directamente los atributos de la clase en lugar de usar los métodos getter y setter, cualquier cambio en esos atributos requeriría cambiar también todas esas partes del código.
- **Seguridad de los datos:** Los atributos de un objeto pueden ser sensibles o críticos para el estado coherente del objeto. Permitir el acceso directo a estos atributos puede poner en riesgo la seguridad e integridad de los datos, ya que pueden ser alterados de maneras no deseadas o inesperadas.

Modificadores de acceso

Los modificadores de acceso en Java determinan la visibilidad de las clases, los métodos y los atributos. Hay cuatro niveles de acceso:

- **public**: La clase, el método o el atributo es accesible desde cualquier lugar.
- **protected**: La clase, el método o el atributo es accesible dentro del mismo paquete y también por subclases de cualquier paquete (explicaremos qué son las subclases más adelante).
- **default**: La clase, el método o el atributo es solo accesible dentro del mismo paquete. No es necesario declararlo, si no se especifica un modificador de acceso es el comportamiento por defecto.
- **private**: La clase, el método o el atributo es accesible solamente dentro de la clase.

Métodos getters y settters

Los métodos getters y setters son una parte integral de la encapsulación y la ocultación. Son métodos que se utilizan para recuperar (getters) y actualizar (setters) el valor de un atributo privado de una clase.

```
public class Persona {
    // Los atributos son privados, lo que significa que solo pueden ser
accedidos directamente dentro de esta clase.
    private String nombre;
    private int edad;

    // Este es el constructor de la clase.
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Un método getter para el nombre.
    public String getNombre() {
        return nombre;
    }

    // Un método setter para el nombre.
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Un método getter para la edad.
    public int getEdad() {
        return edad;
    }

    // Un método setter para la edad.
    public void setEdad(int edad) {
        this.edad = edad;
    }
}
```