códigofacilito

Java Profesional

Bloque 5 : POO, Programación Orientada a Objetos

Tatiana Borda
https://tatianaborda.com/
https://www.linkedin.com/in/tatiana-borda/
https://www.youtube.com/@AlienExplorer

códigofacilito

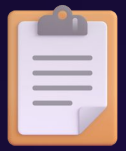

Tabla de contenido

códigofacilito

/01 Qué es la POO?

La Programación Orientada a Objetos es un paradigma que organiza el código en clases que modelan entidades del mundo real. Ejemplo: Un programa de gestión de vehículos puede tener clases como Coche, Moto, cada una con sus atributos y métodos.
Un paradigma en programación es un modelo o enfoque fundamental para diseñar y construir software. Define la forma en que nosotros los programadores estructuramos el código para resolver un problema. No es un lenguaje de programación en sí mismo, sino una forma de abordar la programación.

códigofacilito

Programación Orientada a Objetos

Los lenguajes de programación que soportan la programación orientada a objetos (POO) son muchos y varían en su popularidad y uso. Algunos de los más comunes incluyen Java, C++, Python, PHP, Ruby, C#, JavaScript, TypeScript y Objective-C. Siendo JAVA el más utilizado.
La Programación Orientada a Objetos (POO) difiere de otros paradigmas de programación, como la Programación Estructurada y la Programación Funcional, principalmente por su enfoque en los objetos como unidades fundamentales de la estructura del código. La POO se centra en la encapsulación, la herencia y el polimorfismo para organizar y reutilizar el código, mientras que otros paradigmas pueden enfocarse en procesos o funciones. Perooooo veremos eso en profundidad más adelante!

códigofacilito

Ejemplos de uso:

Crear un sistema de gestión de usuarios, donde cada usuario es un objeto con atributos como nombre, email y contraseñas.
Diseñar un juego, donde los personajes, objetos y elementos del juego son objetos con comportamiento específico.
Desarrollar una aplicación web, donde cada página, componente o interacción es un objeto.

Es decir cuando:
*Tenemos sistemas complejos con muchas entidades interactuando (ej: videojuegos, aplicaciones empresariales).
* Trabajamos en proyectos largos donde la organización es crítica.
*En equipos grandes donde simultáneamente se desarrollan funcionalidades distintas por lo que se necesita modularidad.

códigofacilito

Críticas a la POO:

*Complejidad:
La POO puede agregar complejidad innecesaria a proyectos simples, donde la programación estructurada podría ser suficiente.
*Sobrecarga de memoria:
La creación y gestión de objetos puede consumir más memoria que la programación estructurada, lo que puede afectar el rendimiento, especialmente en entornos con recursos limitados.
*Mantenimiento:
La POO puede dificultar el mantenimiento del código, especialmente en proyectos grandes con una jerarquía compleja de clases.
*No es una solución universal:
La POO no es la solución ideal para todos los problemas. En algunos casos, la programación funcional o otros paradigmas pueden ser más adecuados.

códigofacilito

Beneficios de la POO:

*Flexibilidad:
El software puede ser adaptado fácilmente a nuevas necesidades y funcionalidades
*Modularidad:
El código se organiza en unidades independientes, lo que facilita la comprensión y el mantenimiento del software.
*Reutilización:
Las clases y objetos pueden ser reutilizados en diferentes partes del programa o en otros proyectos, ahorrando tiempo y esfuerzo.
*Escalabilidad:
El código POO es más fácil de expandir y mantener a medida que el proyecto crece.

códigofacilito

/02 Definiendo una clase y creando objetos

Una clase es un plantilla que define atributos (datos) y métodos (comportamiento).
Y un objeto es una instancia concreta de una clase, con valores específicos.
Entonces:
La clase es como un molde, mientras que una instancia se refiere a un objeto creado a partir de esa clase, con valores específicos para sus atributos y comportamiento. En otras palabras, una instancia es un objeto con sus propios datos y funcionalidades, que se deriva de la clase.
Veamoslo un ejemplo!

Ejemplo:

```java
// Clase
public class Coche {
    // Atributos
    String marca;
    int anio;

    // Metodo
    void arrancar() {
        System.out.println("El coche arranca.");
    }
}

// Objeto
Coche miCoche = new Coche();
miCoche.marca = "Toyota";
miCoche.anio = 2023;
miCoche.arrancar();  // "El coche arranca."
```

códigofacilito

/03

Atributos y Métodos

En Programación Orientada a Objetos (POO), los atributos y métodos son componentes clave para definir el comportamiento y las características de los objetos. Los atributos son las propiedades o datos que describen un objeto, mientras que los métodos son las acciones o funciones que el objeto puede realizar.

códigofacilito

Atributos:

Los atributos son variables o datos que almacenan información sobre un objeto en particular. Representan las características o propiedades de un objeto.

En nuestro ejemplo:

En la clase "Coche", atributos son "marca" y "anio", le podríamos agregar "color" y "modelo".

Los atributos se pueden acceder y modificar a través de los métodos de la clase.

códigofacilito

Métodos:

Los métodos ya los vimos en el bloque 4, dijimos que son funciones que definen el comportamiento de un objeto. Especifican qué acciones puede realizar un objeto.

En nuestro ejemplo:

En la clase "Coche", tenemos el método "arrancar" pero podríamos agregar "parar", "acelerar" y "frenar".

Los métodos se llaman a través del objeto de la clase y se utilizan para interactuar con los atributos del objeto.

códigofacilito

/04 Encapsulamiento y constructores:

El encapsulamiento y los constructores son conceptos clave para la gestión de objetos y sus datos. El encapsulamiento, en esencia, oculta el estado interno de un objeto y expone una interfaz pública para interactuar con él, controlando el acceso a sus datos. Los constructores, por otro lado, son métodos especiales que se utilizan para crear instancias de una clase y, opcionalmente, inicializar sus atributos.

códigofacilito

Encapsulamiento:

*Ocultación de datos:
El encapsulamiento permite limitar el acceso a los datos e información de los objetos, protegiendolos y ocultando detalles de implementación.

*Control de acceso:
Permite definir qué elementos de una clase son accesibles desde el exterior y cuáles no.

*Modularidad y reutilización:
Fomenta la modularidad del código, facilitando la reutilización de componentes y la creación de software más robusto y seguro.

*Ejemplo:
Se puede declarar un atributo como private para que solo sea accesible dentro de la clase, o como public para que sea accesible desde cualquier parte del programa. Ocultamos los atributos con private y accedemos a ellos con getters y setters

códigofacilito

Constructores:

*Inicialización de objetos:
Los constructores se utilizan para inicializar los atributos de un objeto al momento de su creación.
El constructor te permite crear objetos más rápido y con valores iniciales, pero los getters/setters siguen siendo útiles para cambiar los datos después o leerlos sin exponer los atributos directamente. Se ejecuta al crear un objeto.
No necesitás constructor si vas a crear el objeto vacío y luego rellenar los datos uno por uno

*Nombre y tipo:
Los constructores tienen el mismo nombre que la clase y no tienen tipo de retorno (ni void).

```
// Constructor
public Coche(String marca, int anio) {
    this.marca = marca;
    this.anio = anio;
}
```

códigofacilito

Getters y setters:

Los métodos getters y setters se utilizan para proteger sus datos, especialmente al crear clases.

Para cada instancia de variable, un método getter devuelve su valor, mientras que un método setter lo establece o actualiza. Por ello, los métodos getter y setter también se conocen como métodos de acceso y de modificación , respectivamente .

Por convención, los getters empiezan con la palabra "get" y los setters con la palabra "set", seguida del nombre de la variable. En ambos casos, la primera letra del nombre de la variable se escribe en mayúscula:

```java
// Getters
public String getMarca() {
    return marca;
}
```

```java
// Setters
public void setMarca(String marca) {
    this.marca = marca;
}
```

códigofacilito

Getters y setters:

En nuestro ejemplo del coche:

```java
// Getters
public String getMarca() {
    return marca;
}

public int getAnio() {
    return anio;
}

// Setters
public void setMarca(String marca) {
    this.marca = marca;
}

public void setAnio(int anio) {
    this.anio = anio;
}
```

códigofacilito

FAQ:

Si uso getter y setter necesito igual un constructor?

La respuesta corta es: no es obligatorio usar un constructor si ya tenés getters y setters, pero sí es muy útil y muchas veces recomendable. Si queremos crear objetos ya con datos desde el principio, pero también queremos permitir que se modifiquen después

Cuándo no es necesario usar un constructor?

Cuando vamos a crear el objeto vacío y asignar los valores después usando los setters y el orden y completitud de los datos no es crítica en el momento de instanciar.
Ya que Java, si no definimos un constructor, automáticamente nos da uno por defecto (sin parámetros)Y luego usamos setters para asignar los valores.

códigofacilito

En nuestro ejemplo del coche: 📌

CREAR PRIMERO, ASIGNAR DESPUÉS

```
public class Coche {

  private String marca;
  private int anio;

}

public static void main(
  String[] args) {

  Coche miCoche = new Coche();
  miCoche.setMarca("Toyota");
  miCoche.setAnio(2022);

}
```

CREAR CON TODO LISTO

```
public class Coche {

  public Coche(String marca,
  int anio) {

    this.marca = marca;
    this.anio = anio;
  }
}

public static void main(
  String[] args) {

  Coche miCoche = new Coche
  ciToyota, 2022);

}
```

códigofacilito

/05 Herencia

La herencia en Java es uno de los pilares de la POO que permite que una subclase (clase hija) herede los atributos y métodos de otra superclase (clase padre)

Beneficios de la herencia:
*Las clases hijas pueden acceder y utilizar los métodos y atributos de la clase padre, evitando la duplicación de código.
*Al crear una relación jerárquica entre las clases, facilita la organización y la comprensión del código.
*Las clases hijas pueden extender o modificar el comportamiento de la clase padre, añadiendo nuevas funcionalidades

códigofacilito

Herencia:

La herencia permite que una clase hija (subclase) "**sea un**" tipo de clase padre (superclase).
Por ejemplo, si tenemos una clase Vehiculo y una clase Coche que hereda de Vehiculo, entonces Coche "**es un**" Vehiculo.
En Java, se utiliza la palabra clave `extends` para indicar la herencia.

Veamoslo con el ejemplo completo!

códigofacilito

Superclase

Vehiculo es una clase general. Todo vehículo tiene marca y año, y puede mostrar su información:

```java
public class Vehiculo {
    private String marca;
    private int anio;

    public Vehiculo(String marca, int anio) {
        this.marca = marca;
        this.anio = anio;
    }

    public void mostrarInfo() {
        System.out.println("Marca: " + marca + ", Año: " + anio);
    }
}
```

códigofacilito

Subclase

Coche es una subclase que extiende de vehiculo y le agrega un nuevo atributo y metodo

```java
public class Coche extends Vehiculo {
    private int puertas;

    public Coche(String marca, int anio, int puertas) {
        super(marca, anio);
        this.puertas = puertas;
    }

    public void mostrarCantidadDePuertas() {
        System.out.println("Puertas: " + puertas);
    }
}
```

códigofacilito

Main:

Uso en main()

```java
public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche("Toyota", 2020, 4);
        miCoche.mostrarInfo();              // método heredado
        miCoche.mostrarCantidadDePuertas(); // método propio
    }
}
```

Salida:

```
Marca: Toyota, Año: 2020
Puertas: 4
```

códigofacilito

/06 Polimorfismo y sobrescritura de métodos

Es otro de los conceptos fundamentales de POO y en esencia, significa "múltiples formas". En el contexto de Java, significa que un objeto puede tener diferentes comportamientos dependiendo su tipo real (clase concreta) en tiempo de ejecución

Ventajas en el desarrollo de software:

*Permite que el código sea más flexible y adaptable a diferentes situaciones

*Facilita la reutilización y extensión del código, ya que las clases pueden heredar métodos y atributos de otras clases.

*Simplifica el mantenimiento del código, ya que los cambios en una clase no afectan necesariamente a otras clases que implementen una interfaz común.

códigofacilito

Formas de implementar Polimorfismo:

*Polimorfismo por Herencia (Sobreescritura de métodos)

Una clase hija redefine(sobreescribe) con `@Override` un método de la clase padre.

*Polimorfismo por Interfaces

Varias clases implementan la misma interfaz, cada una con su propia lógica para los métodos definidos. Lo veremos mejor en el siguiente video.

*Polimorfismo Paramétrico (Genéricos <>)

Clases o métodos que trabajan con tipos genéricos (T, E, etc.) para operar con múltiples tipos de datos. Podés verlo en el curso de Java Avanzado!

códigofacilito

Sobrescritura de métodos

Esto significa que una clase puede implementar una versión específica de un método que ya está definido en su superclase, adaptándolo a sus necesidades particulares:

```java
public class Vehiculo {
    public void mover() {
        System.out.println("El vehículo se mueve.");
    }
}
```

Subclases que sobreescriben el método de la superclase:

```java
public class Coche extends Vehiculo {
    @Override
    public void mover() {
        System.out.println("El coche avanza por la carretera.");
    }
}
```

```java
public class Motocicleta extends Vehiculo {
    @Override
    public void mover() {
        System.out.println("La motocicleta acelera entre el tráfico.");
    }
}
```

Uso en main() con polimorfismo:

```
public class Main {
    public static void main(String[] args) {
        Vehiculo v1 = new Coche();
        Vehiculo v2 = new Motocicleta();

        v1.mover(); // "El coche avanza por la carretera."
        v2.mover(); // "La motocicleta acelera entre el tráfico."
    }
}
```

Aunque v1 y v2 parecen ser "vehículos", al ejecutarse, Java sabe cuál es su forma real (coche o moto) y elige el comportamiento adecuado. Eso es polimorfismo.

códigofacilito

/07 Abstracción e interfaces

En POO la abstracción es un concepto fundamental que permite simplificar la complejidad de un programa al enfocarse en los aspectos esenciales de los objetos y ocultar los detalles innecesarios de implementación. Esto se logra mediante la creación de clases abstractas o interfaces, que definen las características generales y comportamientos de un grupo de objetos, sin especificar cómo se implementan esos comportamientos.

códigofacilito

Qué es la abstracción?

Es el proceso de ocultar los detalles complejos y mostrar solo lo esencial. En Java, esto se hace con una clase abstract, que no se puede instanciar directamente.

```java
public abstract class Vehiculo {
    public abstract void encender(); // método abstracto

    public void mostrarTipo() {
        System.out.println("Soy un vehículo.");
    }
}
```

abstract class → No se puede crear un Vehiculo directamente.
abstract void encender() → No tiene cuerpo, solo define que todos los vehículos deberán implementar su propia forma de encenderse.
mostrarTipo() → Es un método normal, y puede tener implementación.

códigofacilito

Creamos las subclases

```java
public class Coche extends Vehiculo {
    @Override
    public void encender() {
        System.out.println("El coche se enciende con la llave.");
    }
}
```

```java
public class Motocicleta extends Vehiculo {
    @Override
    public void encender() {
        System.out.println("La motocicleta se enciende con el botón.");
    }
}
```

Coche es un vehículo concreto, y por eso está obligado a implementar encender()
Motocicleta también implementa su propia versión de encender().
Ya no son abstractas, así que sí se pueden instanciar.

códigofacilito

En el main()

```java
public class Main {
    public static void main(String[] args) {
        Vehiculo v1 = new Coche();
        Vehiculo v2 = new Motocicleta();

        v1.encender(); // "El coche se enciende con la llave."
        v2.encender(); // "La motocicleta se enciende con el botón."
    }
}
```

Esto es abstracción + polimorfismo:
Cada vehículo tiene su forma de encender, aunque los llamemos a través del mismo tipo Vehiculo.

códigofacilito

Qué es una interfaz en Java?

Una interfaz es un contrato: obliga a las clases que la implementan a definir ciertos métodos.
A diferencia de una clase abstracta, una interfaz:
No tiene atributos con estado
No puede implementar lógica (salvo con default, que no usamos en cursos básicos)
Se **implementa**, no se extiende

Crear una interfaz Electrico, es un contrato: "si sos eléctrico, tenés que saber cargar batería."

```
public interface Electrico {
    void cargarBateria();
}
```

códigofacilito

Creamos una clase para implementar la interfaz

CocheElectrico hereda de la clase Coche (usando extends Coche).
También implementa la interfaz Electrico (usando implements Electrico).
Proporciona la implementación del método cargarBateria() requerido por la interfaz.
El método imprime un mensaje indicando que se está cargando la batería.
La anotación @Override indica que estamos sobrescribiendo un método de la interfaz
CocheElectrico es una subclase de Coche, pero además implementa la capacidad de ser eléctrico:

```
public class CocheElectrico extends Coche implements Electrico {
    @Override
    public void cargarBateria() {
        System.out.println("Cargando batería del coche eléctrico...");
    }
}
```

códigofacilito

En main()

Creamos una instancia de CocheElectrico llamada tesla
Llamamos al método encender() que hereda de la clase Coche (y Coche lo hereda de Vehículo)
Llama al método cargarBateria() que proviene de la implementación de la interfaz Electrico:

```
public class Main {
    public static void main(String[] args) {
        CocheElectrico tesla = new CocheElectrico();
        tesla.encender();          // hereda de Coche → Vehiculo
        tesla.cargarBateria();     // método de la interfaz
    }
}
```

códigofacilito

/08 HANDS ON

A codear, vamos al editor de código!

REPOSITORIO CON EL CÓDIGO:
https://github.com/tatianaborda/java-course

códigofacilito

Objetivo:
Mostrar cómo transformar un programa estructurado simple en un programa orientado a objetos completo, sin aumentar la complejidad del contenido, y aplicando todos los pilares de la POO en un contexto realista y conocido.

Concepto	Aplicación
Encapsulamiento	Atributos private, uso de getters/setters
Herencia	Tarea hereda de Actividad
Polimorfismo	ejecutar() se llama sobre una referencia del tipoActividad
Constructor personalizado	Inicializa nombre y prioridad al crear la tarea
Código familiar	Mantiene la lógica y estructura del bloque 4 (menú, acumulador, Scanner)