CLASE 5

POO

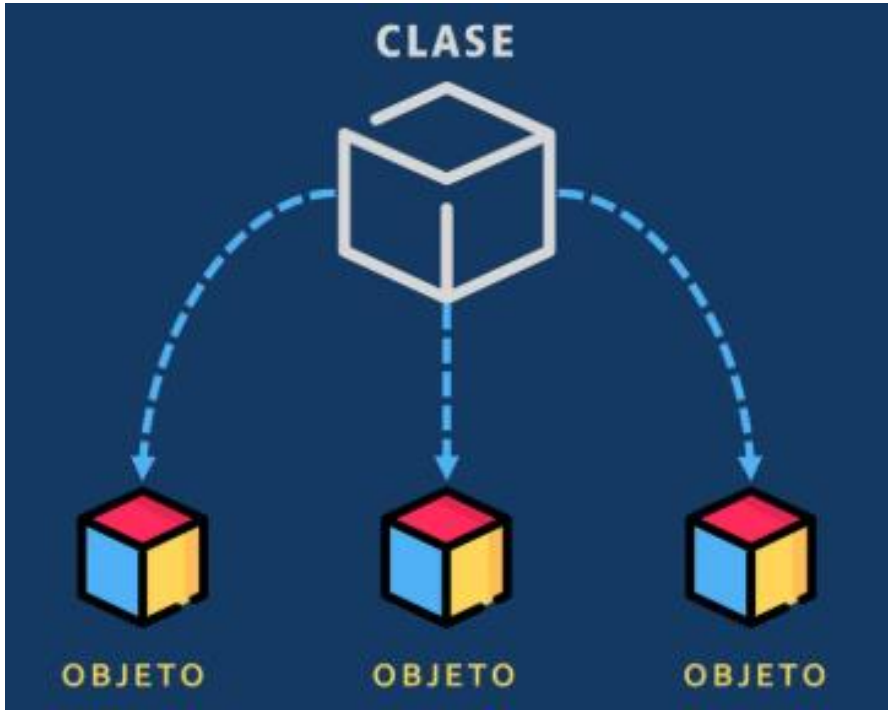
Programación Orientada a Objetos

AGENDA

- Programación Orientada a Objetos (POO)
- Conceptos de Programación Orientada a Objetos en JAVA
- Pilares de la Programación Orientada a Objetos

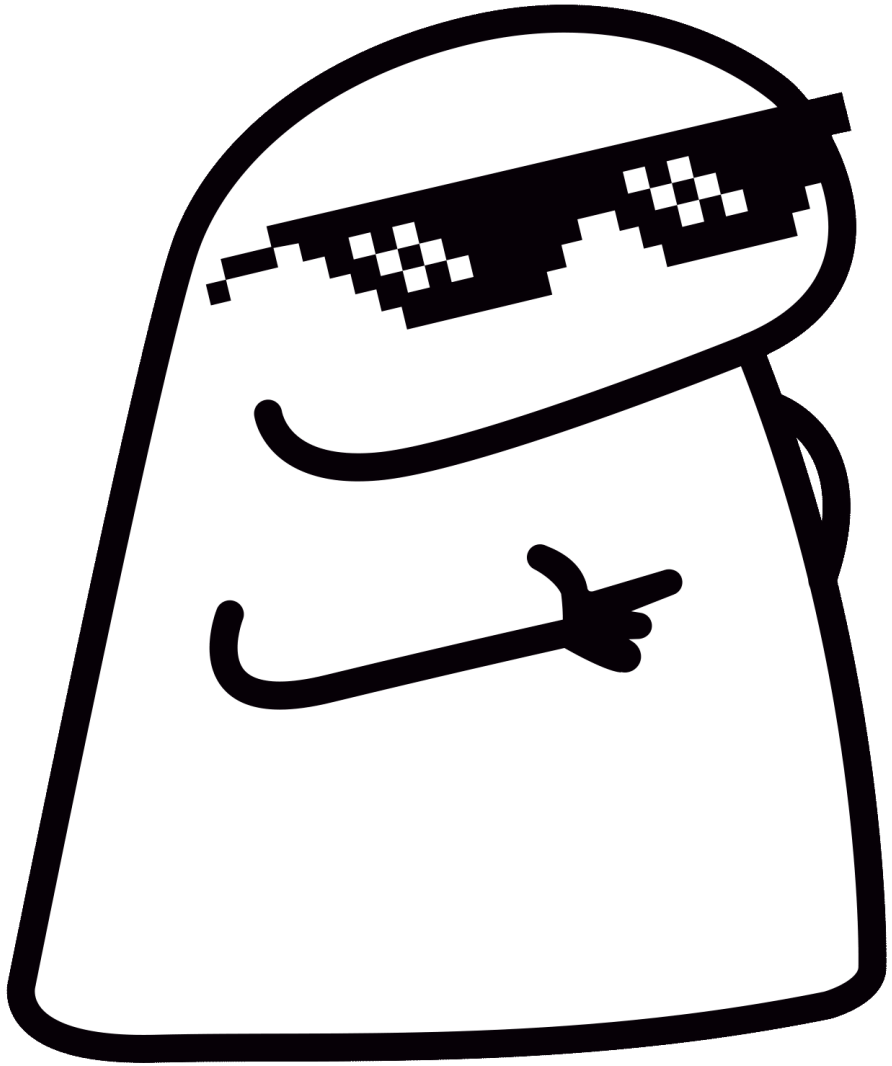
PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

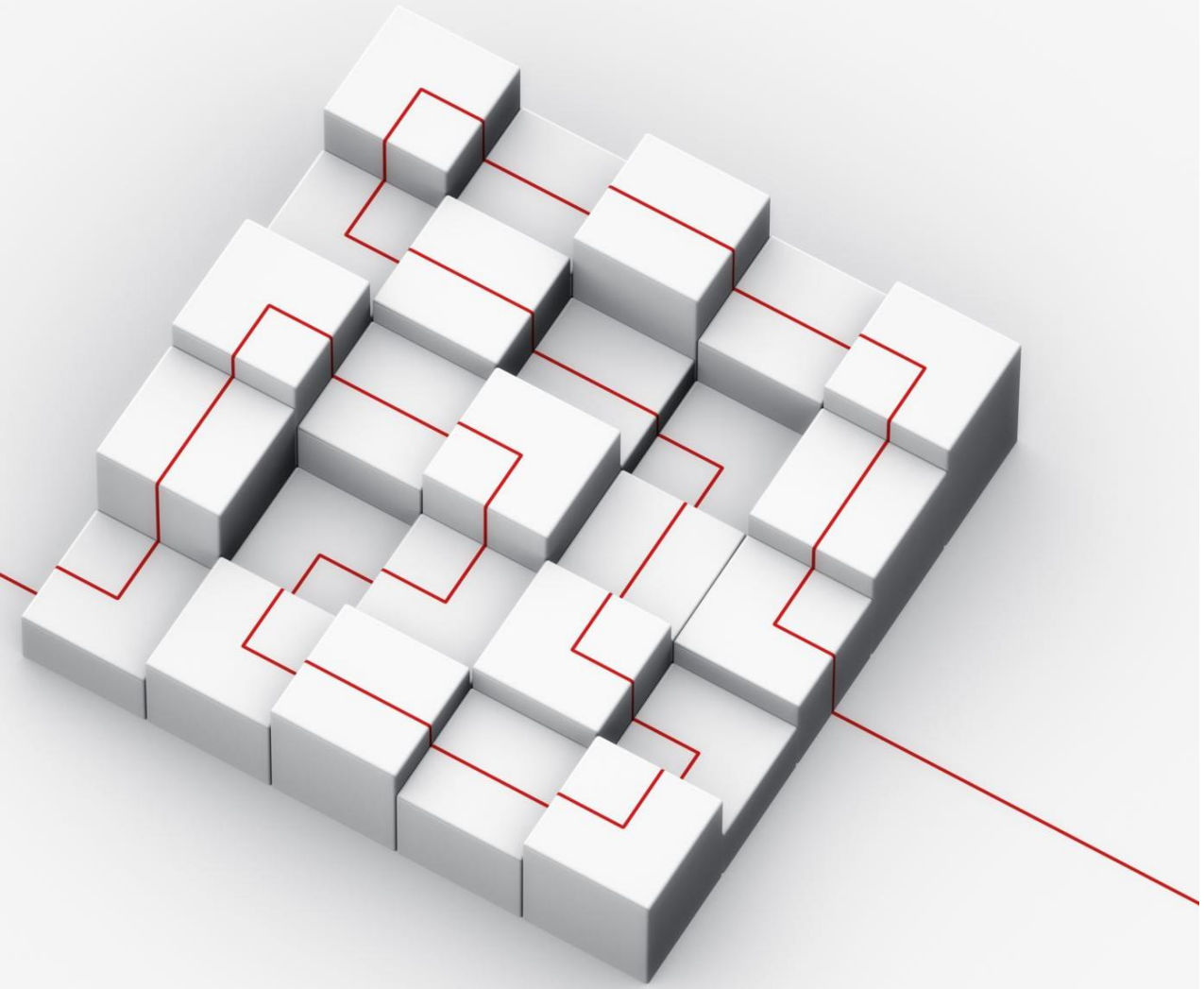
Este tipo de programación se emplea para estructurar un programa de software en piezas simples y reutilizables de planos de código (clases) para crear instancias individuales de objetos.



CONCEPTOS DE PROGRAMACIÓN ORIENTADA A OBJETOS EN JAVA

¿Porqué Java es orientado a objetos? ¿Qué diferencia a Java de otros lenguajes orientados a objetos? Relación entre el paradigma de objetos y Java.





JAVA Y SU PARADIGMA

Java es orientado a objetos y clases. Utiliza un modelo de estructuras para almacenar datos y funciones que interactúan con otras estructuras para modificar sus datos, y así construir aplicaciones en un siguiente nivel de abstracción. Toda ejecución de un programa de Java ocurre a partir de una secuencia de instrucciones en un método procedural, que al mismo tiempo es interpretado como un método estático por la JVM (main). Este método es único para un programa en Java, e indica el flujo inicial de las tareas que ejecutará cada instancia de forma individual.

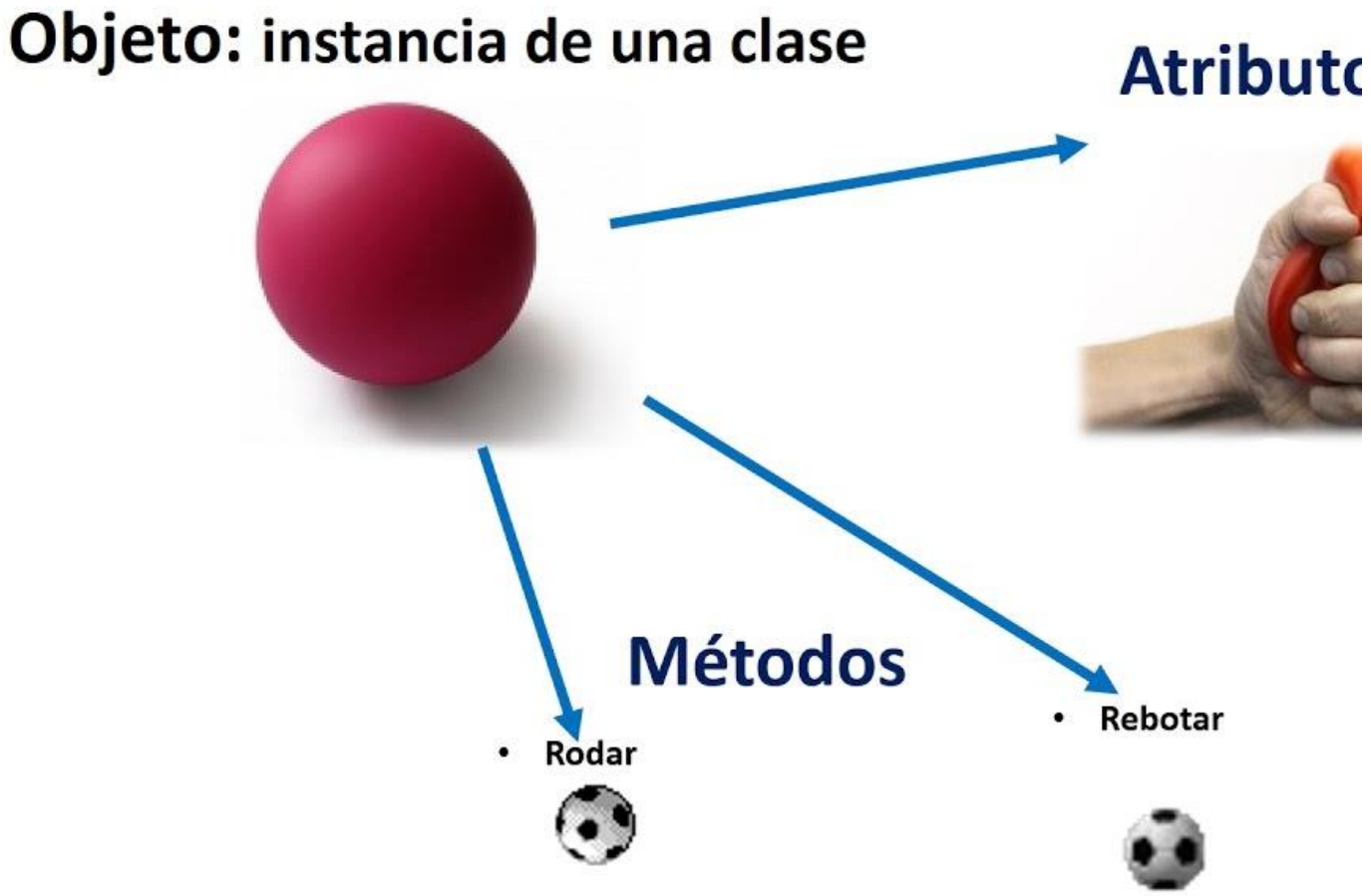
CONSTRUCTORES

Un constructor en Java es un método especial que se utiliza para inicializar objetos. Un constructor es llamado cuando un objeto de una clase es instanciado. Se utiliza para asignar valores iniciales a atributos. Los métodos y funciones de un objeto son almacenados en el mismo entorno que la clase, pero son definidos en tiempo de compilación.

```java
package com.java.white.box;

/**
 * @author nconde
 */
public class Persona {

    private String nombre;
    private String apellido;

    /**
     * Constructor vacio
     */
    public Persona(){

    }

    /**
     * Constructor sobre cargado
     * @param nombre es el nombre de la persona
     */
    public Persona(String nombre){
        this.nombre = nombre;
    }

}

```

OBJETO

Representa alguna entidad de la vida real, es decir, alguno de los objetos únicos que pertenecen al problema con el que nos estamos enfrentando, y con el que podemos interactuar.

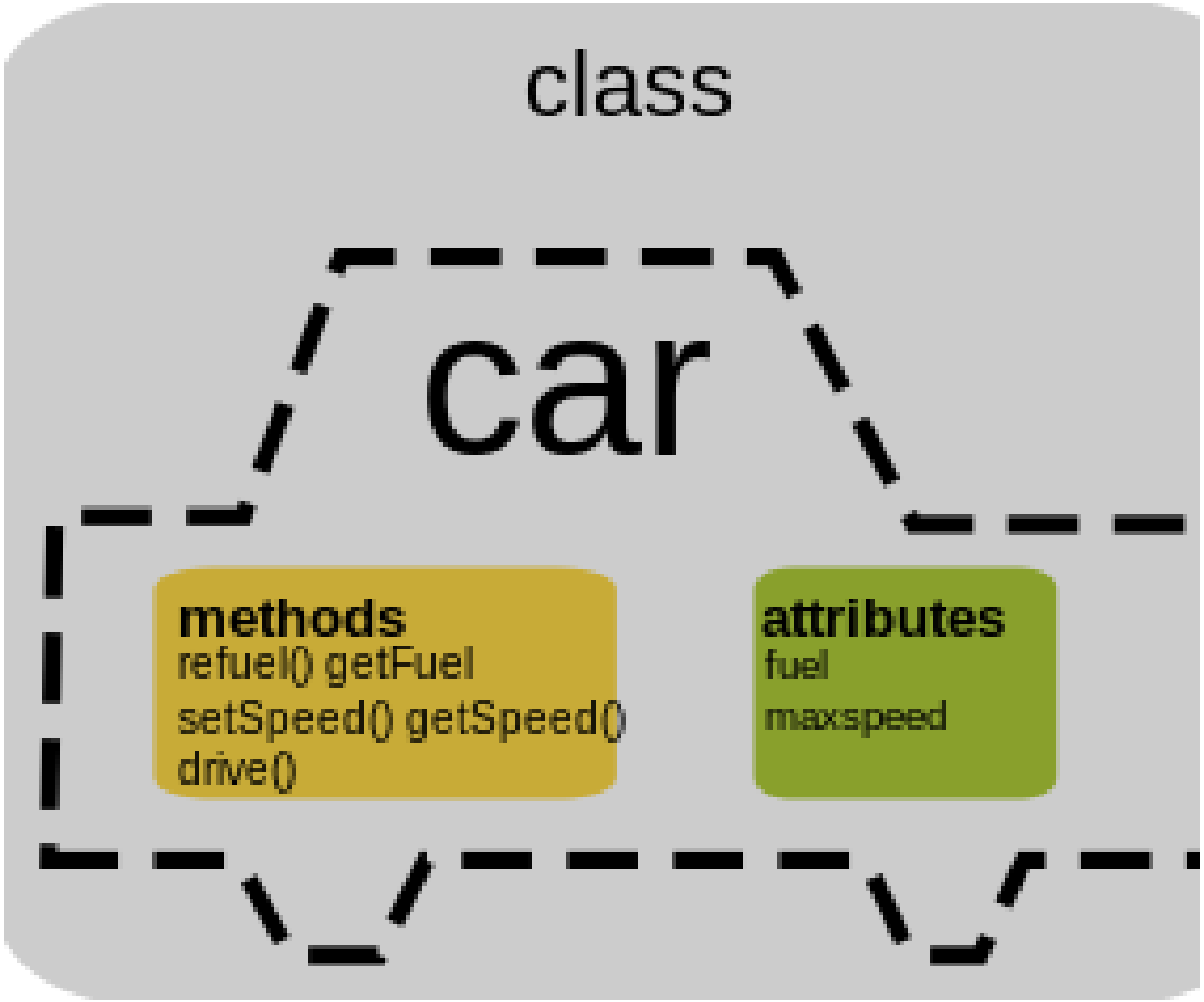


OBJETO
JUGADOR

- Atributos: Son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades.
- Métodos: Son funciones o procedimientos que se definen dentro de una clase y se utiliza para representar el comportamiento de un objeto
- Constructor: Es una subrutina cuya misión es inicializar un objeto de una clase.

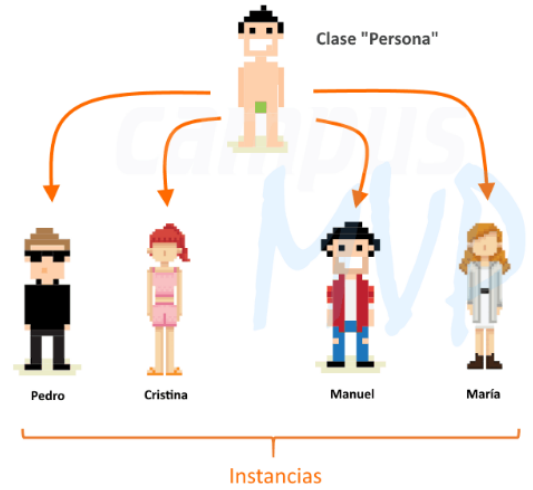
CLASE

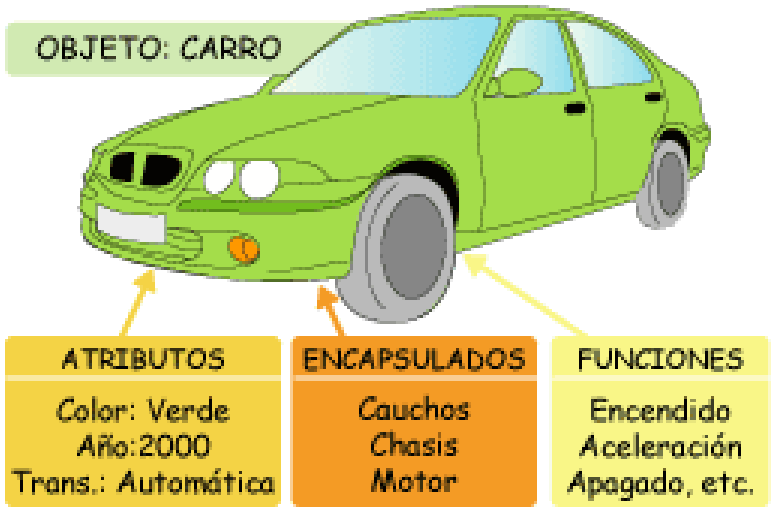
Una clase es un elemento de la programación orientada a objetos que actúa como una plantilla y va a definir las características y comportamientos de una entidad.



INSTANCIA

Permite la creación de objetos concretos a partir de una clase y su posterior manipulación a través de métodos.

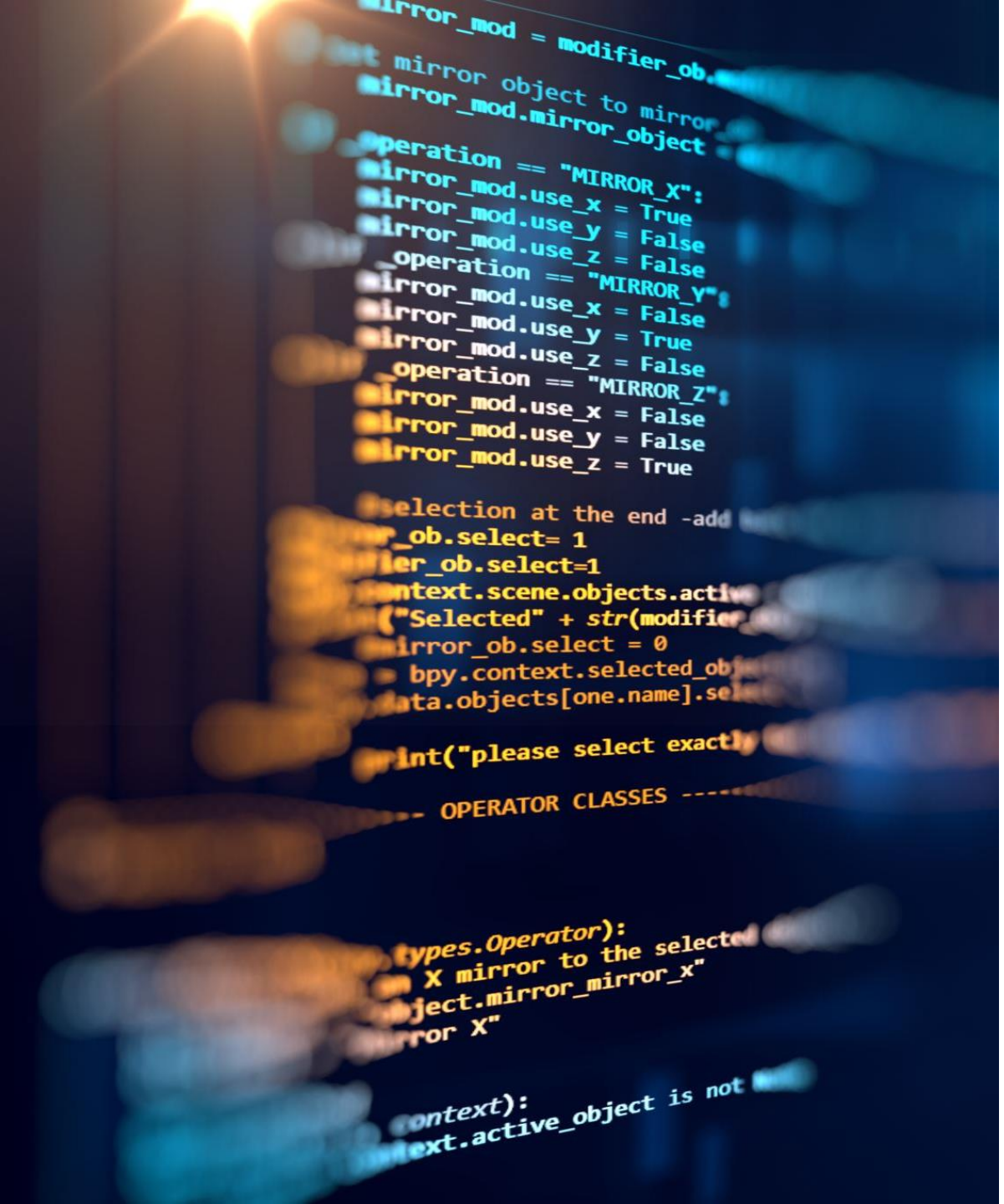




PILARES DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

POO

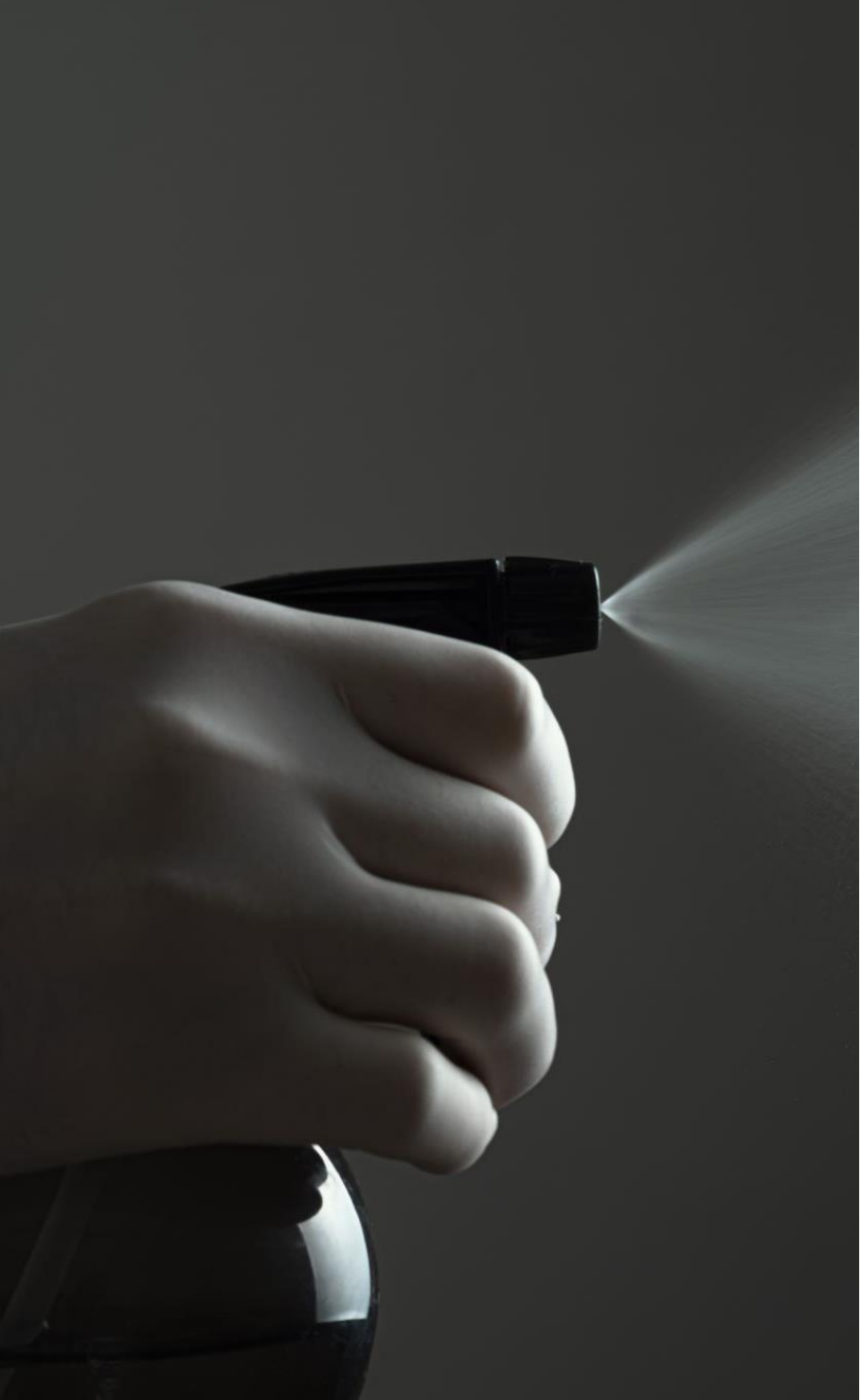
- Abstracción
- Encapsulamiento
- Polimorfismo
- Herencia



ABSTRACCIÓN

Por definición, la abstracción es usar cosas simples para representar cosas complejas. En Java, esto se traduce como la creación de objetos, variables y clases para representar código y datos enmascarados. Evita repeticiones y trabajo, así como facilita el entendimiento y la integración.

En resumen, la abstracción es el proceso de definir los atributos de y métodos de una clase.



ENCAPSULAMIENTO

Es la práctica de mantener los campos de una clase ocultos, proveyendo acceso a estos por medio de métodos y funciones públicas. Esto crea barreras protectoras entre los datos y códigos, para la misma clase. Así, los objetos son reutilizables en componentes de código o variables que no permitan acceder a la data de un sistema completo.

GET (Obtener)

- Es una función la cuál hace que se pueda acceder al atributo de un objeto.

SET (Modificar)

- Es un método que permite cambiar el valor del atributo de un objeto.

EJEMPLO ENCAPSULAMIENTO

```java
class Persona {
    // Atributos privados (encapsulamiento)
    private String nombre;
    private int edad;

    // Constructor
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    // Get
    public String getNombre() {
        return nombre;
    }

    // Set
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    // Get
    public int getEdad() {
        return edad;
    }

    // Set
    public void setEdad(int edad) {
        if (edad > 0) {
            this.edad = edad;
        } else {
            System.out.println("La edad no puede ser igual o menor a 0");
        }
    }

    // Mostrar información de la persona
    public void mostrarInfo() {
        System.out.println("-----------------------------------");
        System.out.println("Persona");
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
        System.out.println("-----------------------------------");
    }
}
```

POLIMORFISMO

El polimorfismo es un principio de POO, el cuál permite que una misma interfaz o método pueda comportarse de diferentes maneras según el objeto que lo implemente.

Existen dos tipos principales:

- Polimorfismo en tiempo de compilación (estático): Se logra mediante sobrecarga de métodos o operadores.
  - Ejemplo: Un método con el mismo nombre pero diferente cantidad o tipo de parámetros.
- Polimorfismo en tiempo de ejecución (dinámico): Se logra mediante herencia y sobreescritura de métodos, donde una clase hija redefine el comportamiento de un método de su clase base.

El polimorfismo permite escribir código más flexible y reutilizable, dando la opción de tratar objetos de diferentes tipos de manera uniforme.

```java
class Calculadora {
    // Método para sumar valores numéricos
    int sumar(int valor1, int valor2) {
        return valor1 + valor2;
    }

    // Sobrecarga: Método para sumar 3 valores
    int sumar(int valor1, int valor2, int valor3) {
        return valor1 + valor2 + valor3;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        System.out.println("Suma de dos valores: " + calc.sumar(1, 2));
        System.out.println("Suma de tres valores: " + calc.sumar(9, 6, 4));
    }
}
```

EJEMPLO POLIMORFISMO EN TIEMPO DE COMPILACIÓN

```java
class Animal {
    // Método base
    void hacerSonidoAnimal() {
        System.out.println("Soy un Animal!");
    }
}

class Perro extends Animal {
    // Sobrescritura del método "hacerSonidoAnimal"
    @Override
    void hacerSonidoAnimal() {
        System.out.println("Soy un perro: Guau guau");
    }
}

class Gato extends Animal {
    // Sobrescritura del método "hacerSonidoAnimal"
    @Override
    void hacerSonidoAnimal() {
        System.out.println("Soy un gato: Miau");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal miAnimal;  // Referencia base

        miAnimal = new Perro();
        miAnimal.hacerSonidoAnimal(); // Llama al método de Perro

        miAnimal = new Gato();
        miAnimal.hacerSonidoAnimal(); // Llama al método de Gato
    }
}
```

EJEMPLO POLIMORFISMO EN TIEMPO DE EJECUCIÓN

HERENCIA

Permite a una clase (llamada clase hija o subclase) heredar propiedades y métodos de otra clase (llamada clase padre o superclase).

Características principales:

- Reutilización de código: La subclase reutiliza atributos y métodos de la superclase.
- Extensibilidad: La subclase puede añadir nuevos atributos y métodos o sobrescribir los existentes de la superclase.
- Relación jerárquica: Representa una relación del tipo "es un" (Gato "es un" Animal).

EJEMPLO HERENCIA

```java
// Superclase
class Animal {
    String nombre;

    // Constructor
    public Animal(String nombre) {
        this.nombre = nombre;
    }

    // Método común
    void comer() {
        System.out.println(nombre + " [ocupado (está comiendo)]");
    }
}

// Subclase
class Perro extends Animal {
    String raza;

    // Constructor
    public Perro(String nombre, String raza) {
        super(nombre); // Llama al constructor de la superclase
        this.raza = raza;
    }

    // Método exclusivo de clase Perro
    void ladrar() {
        System.out.println(nombre + " está ladrando: ¡Raaaa!");
    }

    // Sobrescritura de método
    @Override
    void comer() {
        System.out.println(nombre + " (un " + raza + ") está comiendo");
    }
}
```

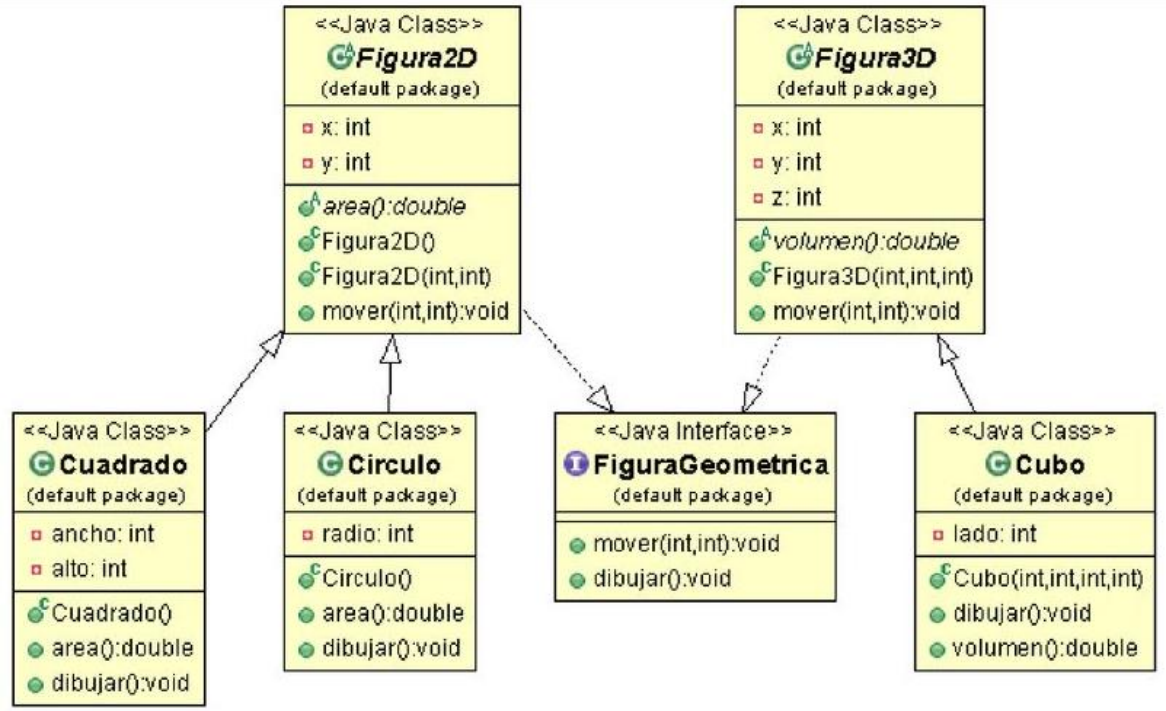
MODIFICADORES DE VISIBILIDAD

Privado: No se puede accesar fuera de una clase.

Publico: Se puede acceder desde cualquier clase desde cualquier paquete.

Protegido: Se puede acceder desde cualquier clase del mismo paquete.

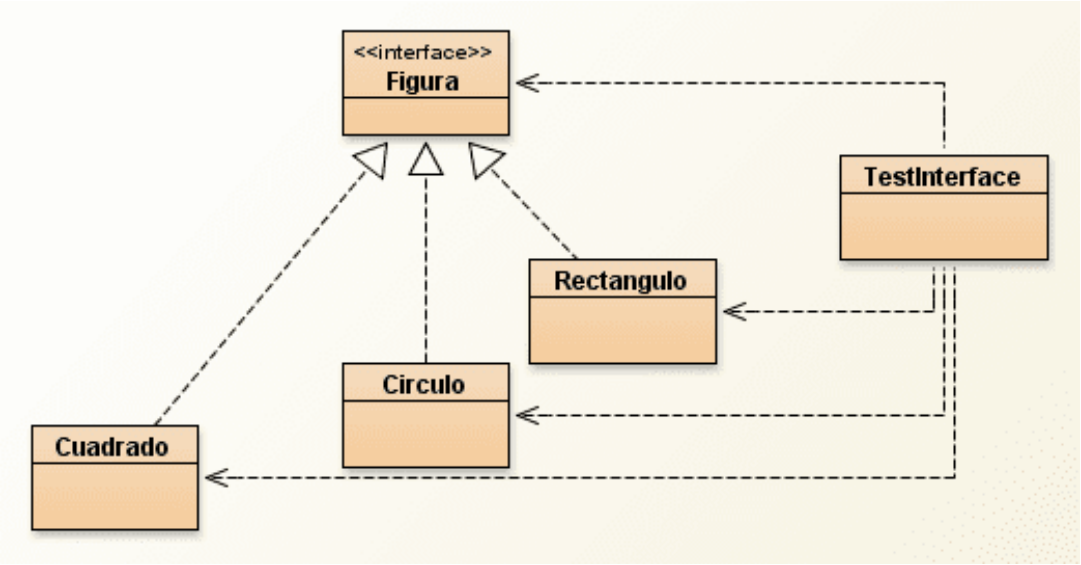
CLASE ABSTRACTA



Las clases abstractas son aquellas que dejan sin implementar algunos miembros o todos ellos para que las clases derivadas puedan proporcionar las implementaciones

INTERFAZ

Una interfaz es un medio común para que los objetos no relacionados se comuniquen entre sí. Estas son definiciones de métodos y valores sobre los cuales los objetos están de acuerdo para cooperar.



VIDEO POO

https://www.youtube.com/watch?v=I848HdWjLMo&feature=youtu.be

VIDEO POLIMORFISMO

https://www.youtube.com/watch?v=bblFTvuk4pY&feature=youtu.be

VIDEO HERENCIA

https://www.youtube.com/watch?v=yh8bTKqCOtU&feature=youtu.be

VIDEO ENCAPSULAMIENTO

https://www.youtube.com/watch?v=8aQSD36paWU&feature=youtu.be

DUDAS O COMENTARIOS