JHONNY DAVID JARAMA TRAPIELLO
Y
LEYRIN BRIDNEYS AGUILAR

IEEE AESS UNI

PROGRAMACIÓN ORIENTADA A OBJETOS

IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

Universidad Nacional de Ingeniería
IEEE Student Branch

PRESENTACIÓN

Web Master de la IEEE AESS UNI

Ing. de software con I.A. (Senati)

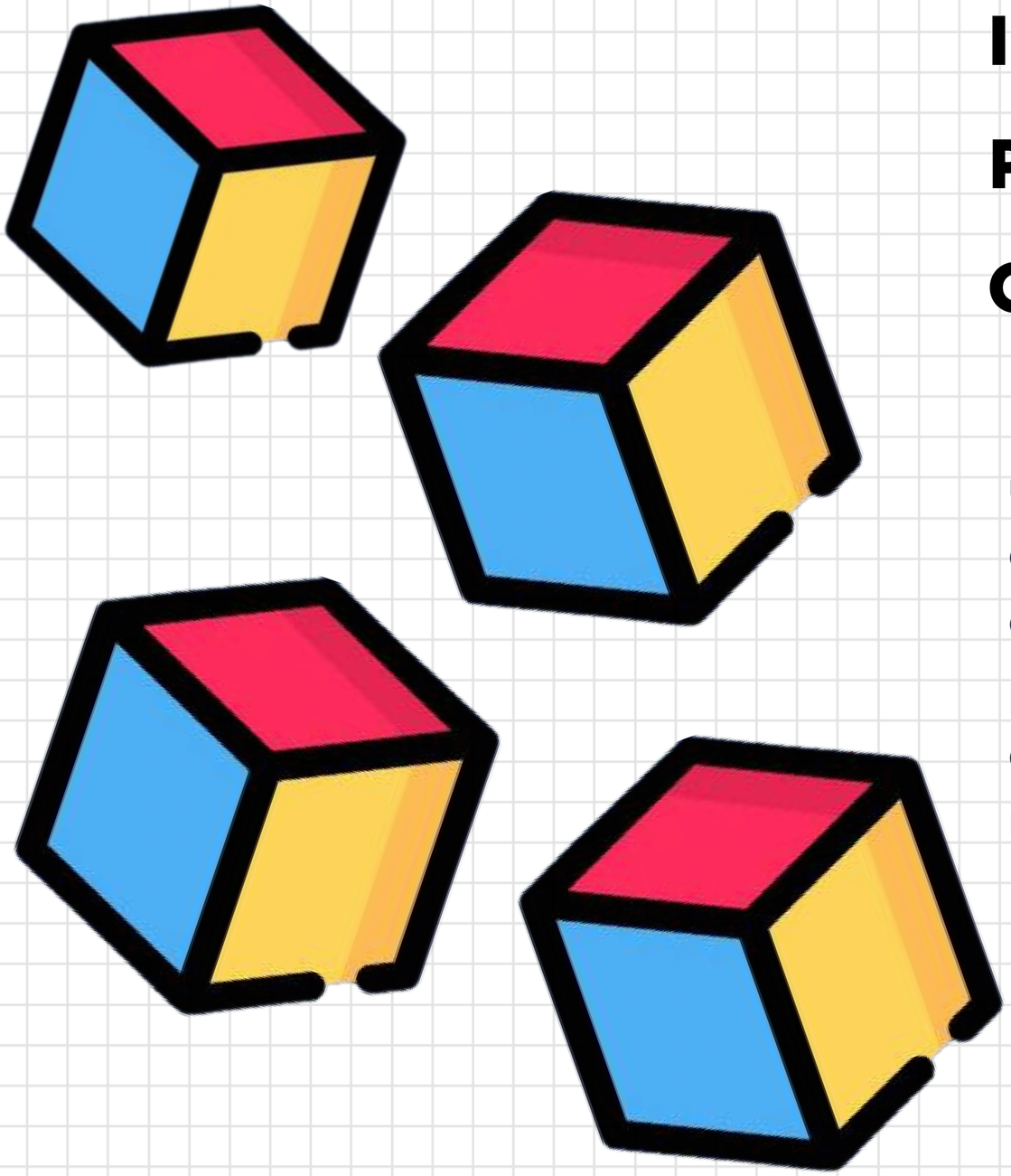
Ciencias Físicas (UNMSM)

Auxiliar en programación Contable

Jhonny David
Jarama Trapiello

AESS
IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

Universidad
Nacional de Ingeniería
IEEE Student Branch



INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTA A OBJETOS

La Programación Orientada a Objetos (POO) es un enfoque para escribir programas informáticos que se basa en la idea de modelar el mundo real en forma de "objetos". Estos objetos tienen propiedades y comportamientos asociados, lo que permite representar sistemas complejos de manera más organizada y comprensible.

AESS
IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

Universidad Nacional de Ingeniería
IEEE Student Branch

En POO, se busca modelar problemas complejos dividiéndolos en unidades más pequeñas (objetos) y definiendo cómo interactúan entre sí. Esto facilita la resolución de problemas y el mantenimiento del código a medida que los sistemas se vuelven más grandes y complejos.

En lenguajes de programación que admiten POO (como Java, Javascript, C++, Python, C#, etc.), puedes crear clases, definir objetos a partir de esas clases y trabajar con atributos y métodos para construir programas orientados a objetos.

Recuerda que la POO es un paradigma poderoso, pero también puede tener cierta curva de aprendizaje. A medida que te familiarices con los conceptos y practiques, te sentirás más cómodo/a creando sistemas y aplicaciones utilizando este enfoque.

EJEMPLOS DE LA VIDA REAL QUE PUEDEN AYUDARTE A ENTENDER MEJOR LA PROGRAMACIÓN ORIENTADA A OBJETOS (POO):

Coche:

- Clase: "Coche"
- Atributos: Marca, Modelo, Año, Color
- Métodos: Arrancar, Detener, Acelerar, Frenar

Banco:

- Clase: "Cuenta Bancaria"
- Atributos: Titular, Saldo, Número de cuenta
- Métodos: Depositar, Retirar, Consultar saldo

Red Social:

- Clase: "Usuario"
- Atributos: Nombre de usuario, Nombre completo, Lista de amigos
- Métodos: Publicar mensaje, Agregar amigo, Comentar

Tienda en línea:

- Clase: "Producto"
- Atributos: Nombre, Precio, Descripción
- Métodos: Agregar al carrito, Realizar pedido, Calcular total

Película:

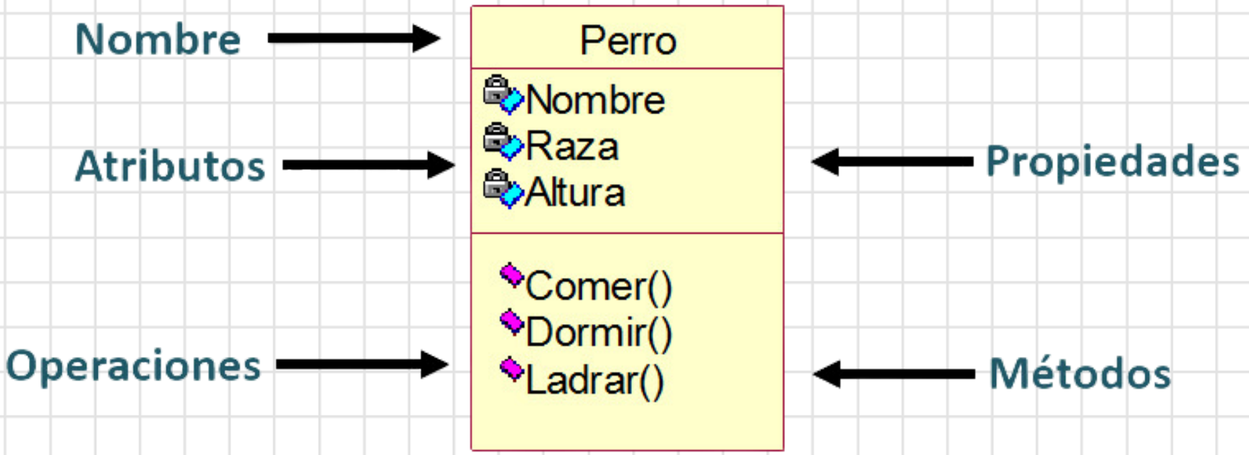
- Clase: "Película"
- Atributos: Título, Director, Año de lanzamiento, Género
- Métodos: Reproducir, Pausar, Detener

CONCEPTOS FUNDAMENTALES DE POO CON CONCEPTOS Y EJEMPLOS CLAROS

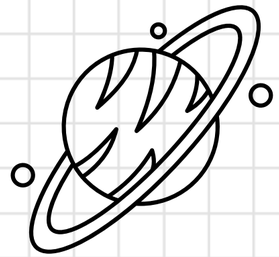
Clases: Una clase es una plantilla que define la estructura y el comportamiento de los objetos. Es como un plano para crear objetos. Contiene atributos y métodos que describen las características y acciones que tendrán los objetos de esa clase.

Imagina que estás diseñando un juego de rol. Una "Clase" podría ser "Mago". Define las características y habilidades generales que tendrán todos los personajes de esa clase, como hechizos y habilidades mágicas.

```
class Mago {
    constructor(nombre, edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```



Objetos: Los objetos son instancias concretas de una clase. Representan elementos específicos con atributos y comportamientos definidos por la clase.
!!No olvidar el concepto de instanciar!!
En el juego de rol, un "Objeto" sería un personaje específico. Por ejemplo, un personaje llamado "Gandalf" sería un objeto de la clase "Mago".

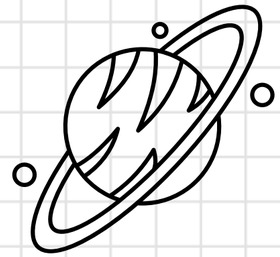


const mago1 = new Mago("Gandalf", 30);

Atributos: Los atributos son propiedades que describen el estado de un objeto. Son variables que pertenecen a la instancia de la clase.
Los "Atributos" son las características de un objeto. En el caso de "Gandalf", los atributos podrían ser "nombre", "edad"
const nombrePersona = persona1.nombre; // Acceso al atributo "nombre"

Métodos: Los métodos son funciones que definen las acciones que un objeto puede realizar. Representan el comportamiento del objeto y pueden interactuar con los atributos.
Los "Métodos" son las acciones que un objeto puede realizar.
En el ejemplo del mago, un método podría ser "saludar()", que hace que el personaje salude en el juego.

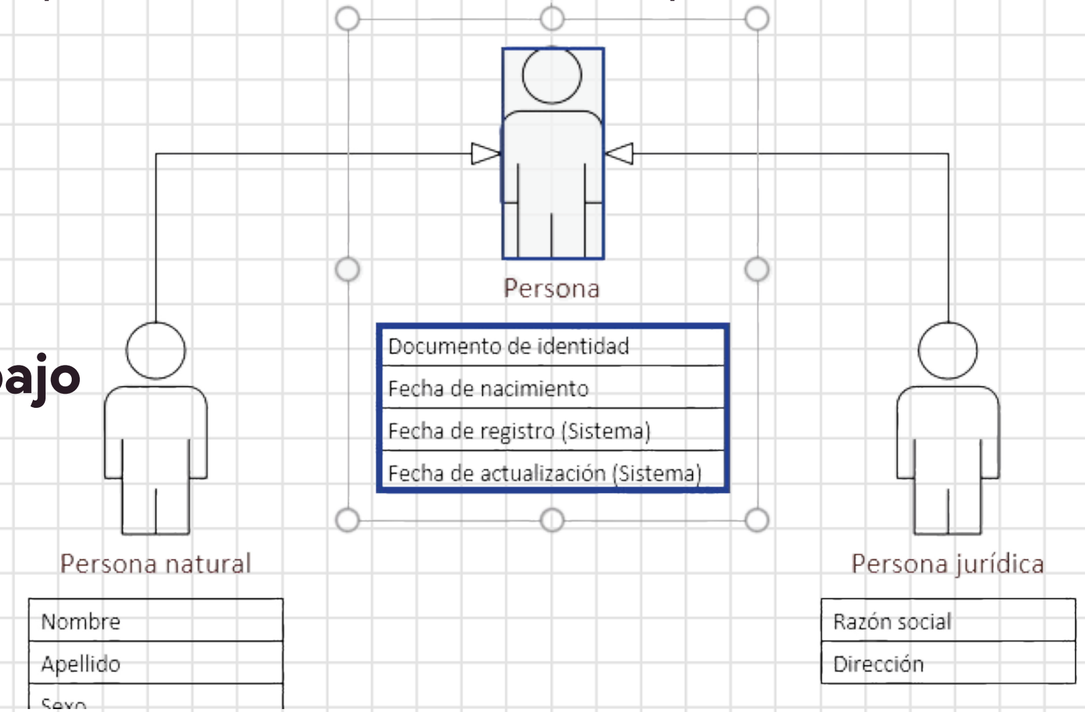
```
class Mago{
   // ... constructor y atributos aquí ...
   saludar() {
      console.log(`Hola, soy ${this.nombre} y tengo ${this.edad} años.`);
   }
}
mago1.saludar();  // Llama al método "saludar"
```



Encapsulamiento: El encapsulamiento es uno de los principios fundamentales de la POO. Consiste en ocultar los detalles internos de una clase u objeto y exponer solo una interfaz controlada para interactuar con él. Esto se logra definiendo atributos como privados (inaccesibles desde fuera de la clase) y proporcionando métodos públicos para acceder y modificar esos atributos. El encapsulamiento ayuda a mantener el estado interno de un objeto protegido y permite controlar cómo se accede y se modifica la información.

En JavaScript, el encapsulamiento se logra mediante la convención de nombres y el uso de métodos para acceder a atributos, ya que no existen atributos privados directos como en otros lenguajes.

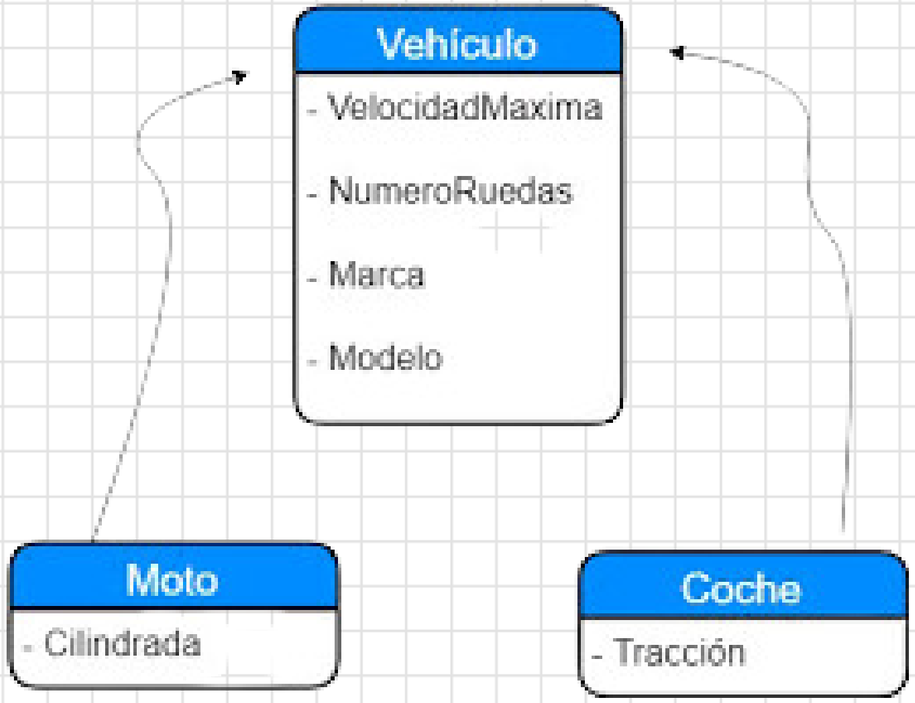
```
class CuentaBancaria {
   constructor(saldo) {
      this._saldo = saldo;
 // Convención de nombre con guión bajo
   }
   obtenerSaldo() {
      return this._saldo;
   } }
```



Herencia: La herencia es un mecanismo mediante el cual una clase puede heredar propiedades y métodos de otra clase. La clase original se llama "superclase" o "clase base", y la clase que hereda de ella se llama "subclase" o "clase derivada". La herencia permite reutilizar código, establecer jerarquías de clases y crear nuevas clases con funcionalidades adicionales basadas en clases existentes.

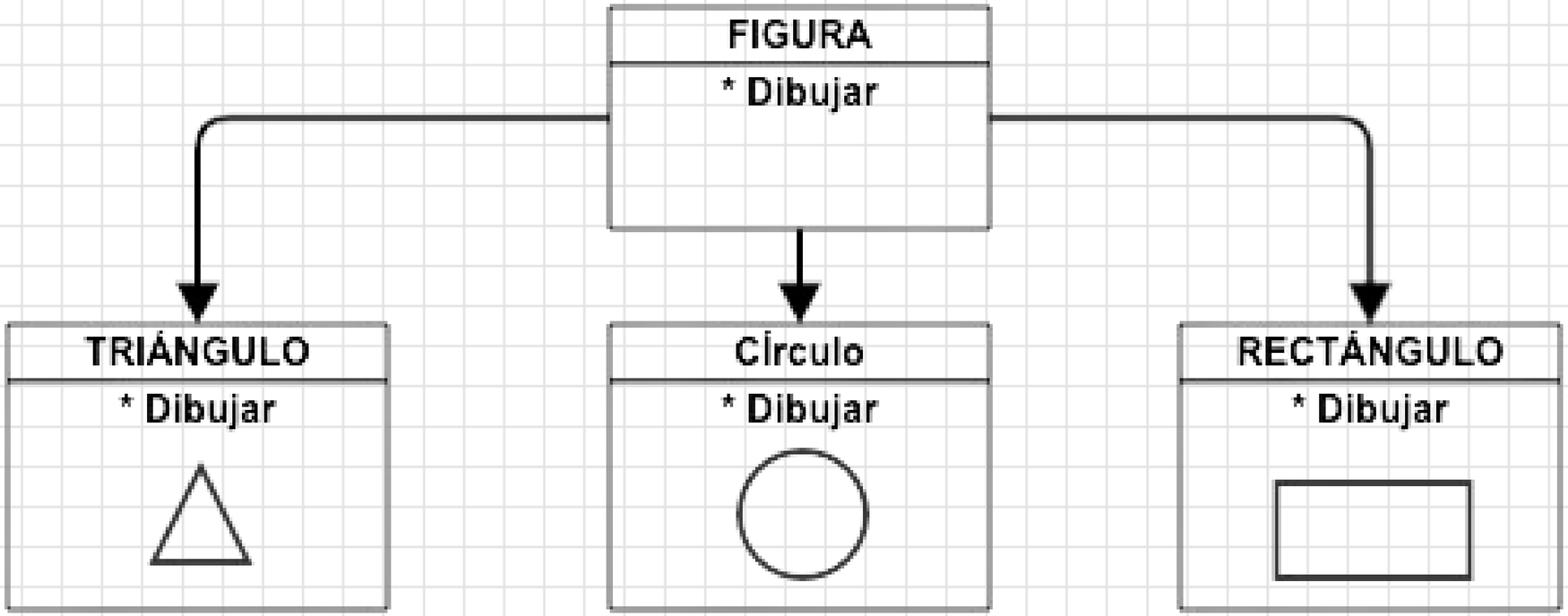
La herencia en JavaScript se logra utilizando el prototipo de objetos. Se crea una nueva clase basada en otra clase existente, heredando propiedades y métodos.

```
class Estudiante extends Persona {
    constructor(nombre, edad, curso) {
        super(nombre, edad);
 // Llama al constructor de la superclase
        this.curso = curso;
    }
}
```

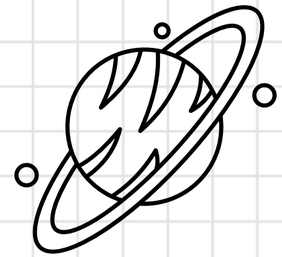


Polimorfismo: El polimorfismo se refiere a la capacidad de diferentes clases (o subclases) de responder al mismo mensaje (llamada a método) de manera única. Esto permite tratar objetos de diferentes clases de manera uniforme a través de interfaces comunes. El polimorfismo puede manifestarse a través de la herencia, donde diferentes subclases implementan un mismo método de formas diferentes.

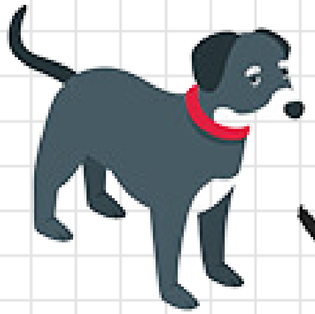
El polimorfismo en JavaScript se logra mediante la capacidad de objetos diferentes para responder al mismo método.



```
class Animal {
    constructor(nombre) {
        this.nombre = nombre;
    }

    hacerSonido() {
        return "Sonido genérico";
    }
}

class Perro extends Animal {
    hacerSonido() {
        return "Woof!";
    }
}

class Gato extends Animal {
    hacerSonido() {
        return "Meow!";
    }
}

function hacerRuido(animal) {
    console.log(animal.nombre + " hace " +
animal.hacerSonido());
}

const perro = new Perro("Buddy");
const gato = new Gato("Whiskers");

hacerRuido(perro);  // Salida: "Buddy hace Woof!"
hacerRuido(gato);   // Salida: "Whiskers hace Meow!"
```

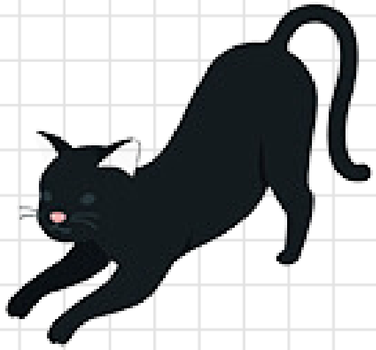


ANIMAL



CLASES

OBJETOS



PERRO

GATO

PANCHO, 4 AÑOS

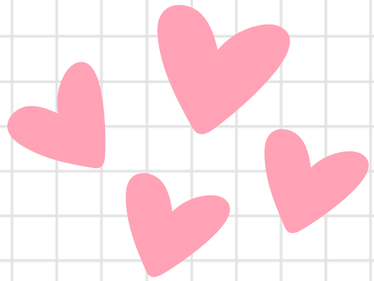
ATRIBUTOS

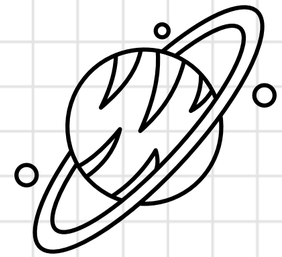
PACO, 3 AÑOS

COMER

MÉTODOS

CAMINAR





PILARES DE LA PROGRAMACIÓN ORIENTA A OBJETOS

ABSTRACCIÓN

Es el proceso de **definir los atributos y los métodos** de una clase.

ENCAPSULAMIENTO

Protege la información de manipulaciones no autorizadas.

POLIMORFISMO

Da la misma orden a varios objetos para que respondan de diferentes maneras.

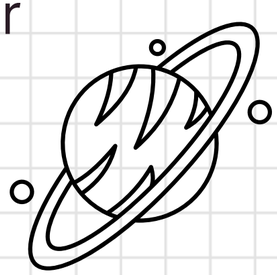
HERENCIA

Las **clases hijo heredan atributos y métodos** de las clases padre.

Según el paradigma, la programación orientada objetos, se basa en estos 4 pilares. **Estos definen la simplicidad y la funcionalidad del código.**

DISEÑO Y MODELADO DE SISTEMAS ORIENTADOS A OBJETOS

Diseñar y modelar sistemas orientados a objetos es un proceso esencial para crear aplicaciones de software eficientes y mantenibles.



Entender el problema:

- Antes de empezar, debes comprender a fondo el problema que estás tratando de resolver. Esto implica identificar los requisitos funcionales y no funcionales del sistema, así como comprender el contexto en el que operará.

Identificar objetos y clases:

- Identifica los objetos principales que estarán presentes en el sistema. Cada objeto debería ser una entidad del mundo real o una abstracción que sea relevante para el problema. Agrupa objetos similares en clases.

Definir atributos y métodos:

- Para cada clase, define los atributos que representarán las propiedades del objeto y los métodos que definirán su comportamiento. Piensa en qué datos necesita almacenar y qué acciones debe realizar cada objeto.

Relaciones entre clases:

- Determina las relaciones entre las clases. Pueden ser asociaciones (relaciones entre objetos), agregaciones (una clase contiene otras clases) o herencias (clases que heredan propiedades de otras).

Diagramas UML:

- Utiliza diagramas UML (Lenguaje de Modelado Unificado) para visualizar y comunicar tus diseños. Los diagramas de clases, diagramas de secuencia y diagramas de casos de uso son especialmente útiles en el modelado de sistemas orientados a objetos.

Encapsulamiento y Abstracción:

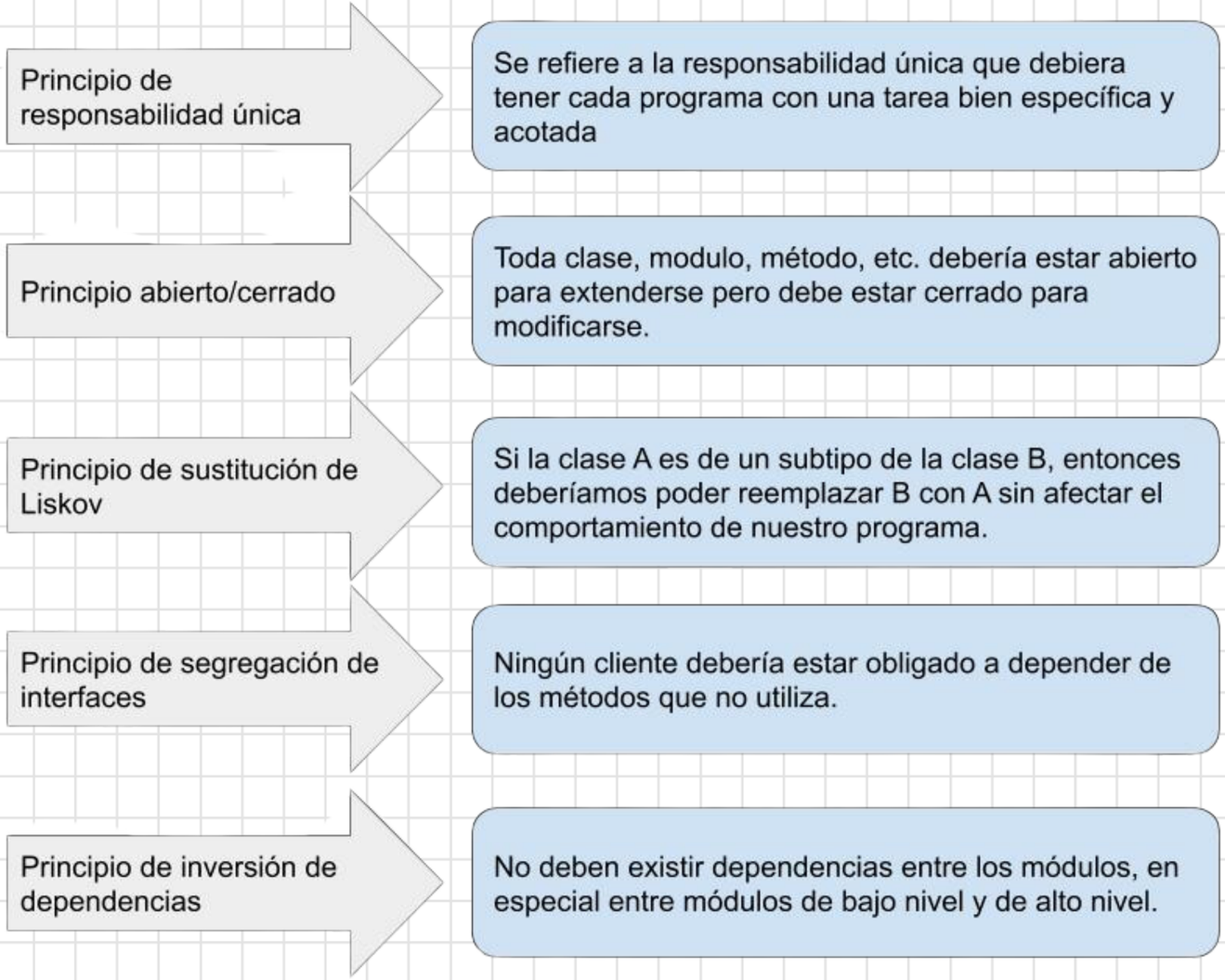
- Diseña las clases de manera que reflejen adecuadamente el mundo real y encapsulen la funcionalidad relevante. Utiliza la abstracción para representar solo los detalles necesarios y ocultar la complejidad innecesaria.

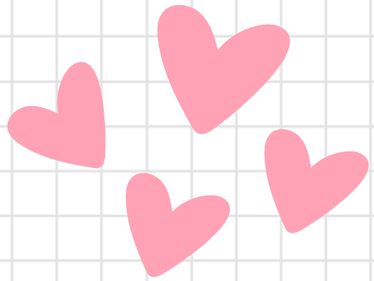
Herencia y Polimorfismo:

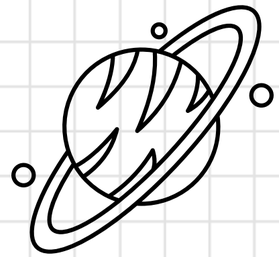
- Si es apropiado, utiliza la herencia para crear jerarquías de clases que comparten propiedades y métodos comunes. Aprovecha el polimorfismo para permitir que diferentes clases respondan al mismo mensaje de manera única.

Principios SOLID:

- Familiarízate con los principios SOLID (Single Responsibility, Open/Closed, Liskov Substitution, Interface Segregation, Dependency Inversion). Estos principios guían el diseño de clases y ayudan a crear sistemas más flexibles y fáciles de mantener.







Prototipos y Pruebas:

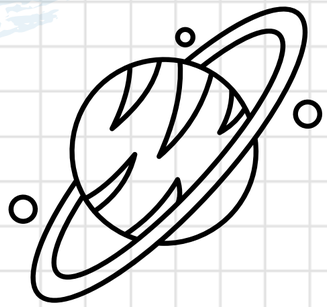
- Crea prototipos o modelos iniciales para validar tus diseños antes de implementar el sistema completo. Las pruebas y la iteración son esenciales para asegurarte de que tus modelos se ajusten adecuadamente a los requisitos y necesidades.

Refactorización:

- A medida que avances en el desarrollo, es posible que necesites ajustar y mejorar tu diseño a medida que surgen nuevos requisitos o cambian las circunstancias. La refactorización es el proceso de reorganizar el código para mantener la calidad y la estructura del diseño.

Recuerda que el diseño y modelado de sistemas orientados a objetos es un proceso iterativo. Es importante ser flexible y estar dispuesto a revisar y ajustar tus diseños a medida que evoluciona el proyecto.

MUCHAS GRACIAS



AESS
IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

Universidad Nacional de Ingeniería
IEEE Student Branch