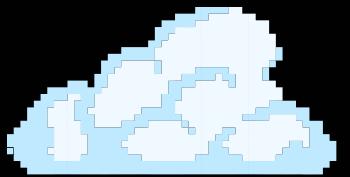


# PRACTICA DE POO

START



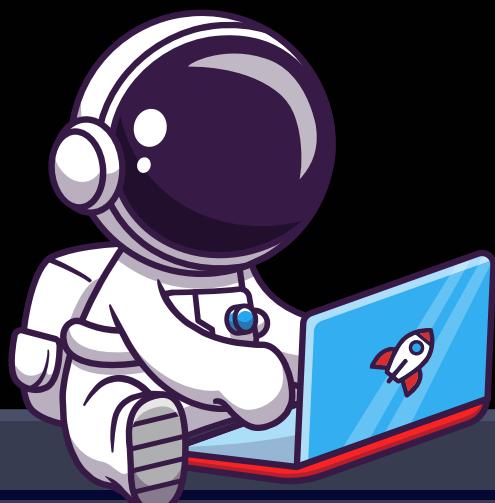
Universidad  
Nacional de Ingeniería  
IEEE Student Branch



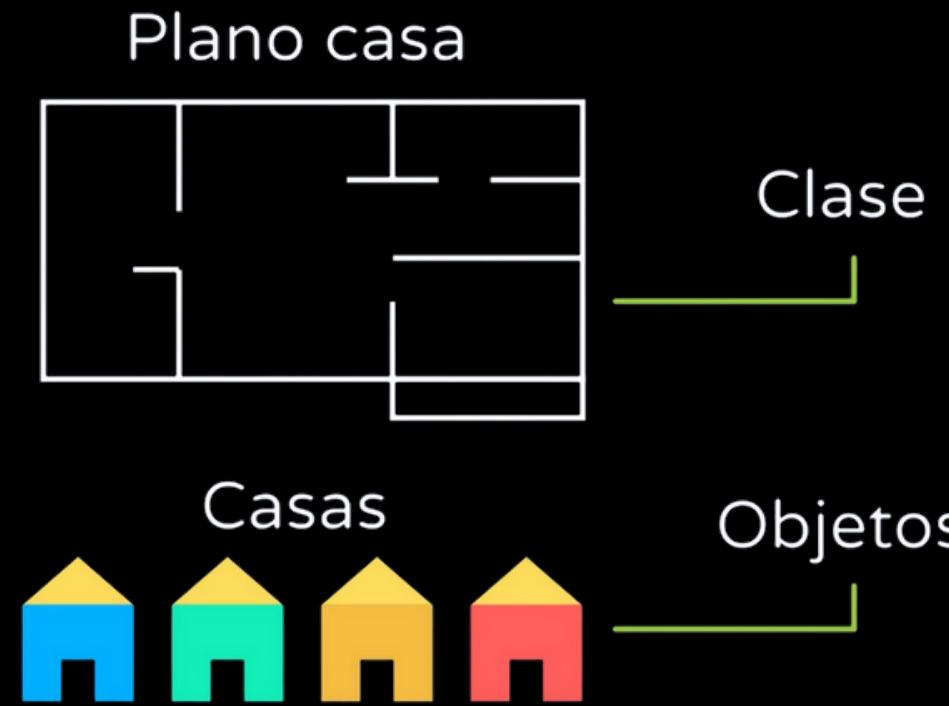
# 01

## EJERCICIO

### CLASES Y OBJETOS



Universidad  
Nacional de Ingeniería  
IEEE Student Branch



Crea una clase **Producto** que tenga atributos como nombre, precio y cantidad. Luego, crea un objeto de esta clase e imprime sus atributos.

```
Producto: Producto { nombre: 'Camiseta', precio: 20, cantidad: 3 }
```





```
// Definición de la clase "Producto"
class Producto {
    // El constructor es un método especial que se ejecuta al crear una nueva instancia de la clase.
    // Toma tres parámetros: nombre, precio y cantidad.
    constructor(nombre, precio, cantidad) {
        // "this" se refiere a la instancia actual del objeto que se está creando.
        // Establecemos las propiedades "nombre", "precio" y "cantidad" en la instancia con los valores
        // proporcionados.
        this.nombre = nombre;
        this.precio = precio;
        this.cantidad = cantidad;
    }
}

// Creación de una instancia de la clase "Producto" llamada "producto1"
const producto1 = new Producto("Camiseta", 20, 3);

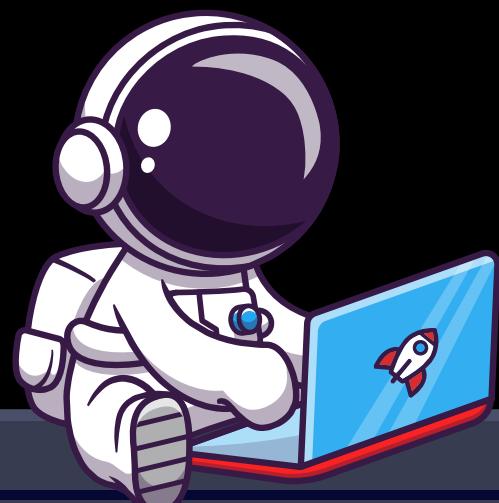
// Imprimir en la consola información sobre el producto1
console.log("Producto:", producto1);
```



02

## EJERCICIO

MÉTODOS Y ATRIBUTOS ESTÁTICOS



Universidad  
Nacional de Ingeniería  
IEEE Student Branch



Implementa una clase Calculadora con un método estático sumar y un método de instancia restar. Prueba ambos métodos.

```
10 libras en kilogramos: 4.53592
```

```
5 metros en pulgadas: 196.8505
```

```
25 grados Celsius en Fahrenheit: 77
```



```
// Definición de una clase llamada Convertidor.
class Convertidor {
    // Método estático para convertir libras a kilogramos.
    static librasAKilos(libras) {
        // Factor de conversión de libras a kilogramos.
        const kilosPorLibra = 0.453592;
        // Multiplica las libras por el factor para obtener kilogramos y lo devuelve.
        return libras * kilosPorLibra;
    }

    // Método estático para convertir metros a pulgadas.
    static metrosAPulgadas(metros) {
        // Factor de conversión de metros a pulgadas.
        const pulgadasPorMetro = 39.3701;
        // Multiplica los metros por el factor para obtener pulgadas y lo devuelve.
        return metros * pulgadasPorMetro;
    }

    // Método estático para convertir grados Celsius a grados Fahrenheit.
    static celsiusAFahrenheit(celsius) {
        // Fórmula de conversión de Celsius a Fahrenheit.
        const fahrenheit = (celsius * 9) / 5 + 32;
        // Retorna el valor convertido a Fahrenheit.
        return fahrenheit;
    }
}

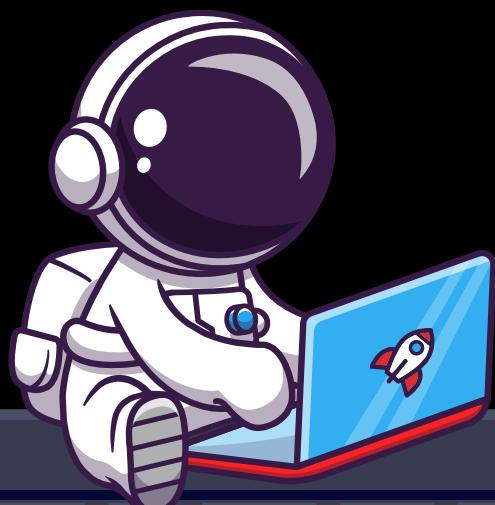
// Imprime en la consola los resultados de las conversiones.
console.log("10 libras en kilogramos:", Convertidor.librasAKilos(10));
console.log("5 metros en pulgadas:", Convertidor.metrosAPulgadas(5));
console.log(
    "25 grados Celsius en Fahrenheit:",
    Convertidor.celsiusAFahrenheit(25)
);
```



# 03

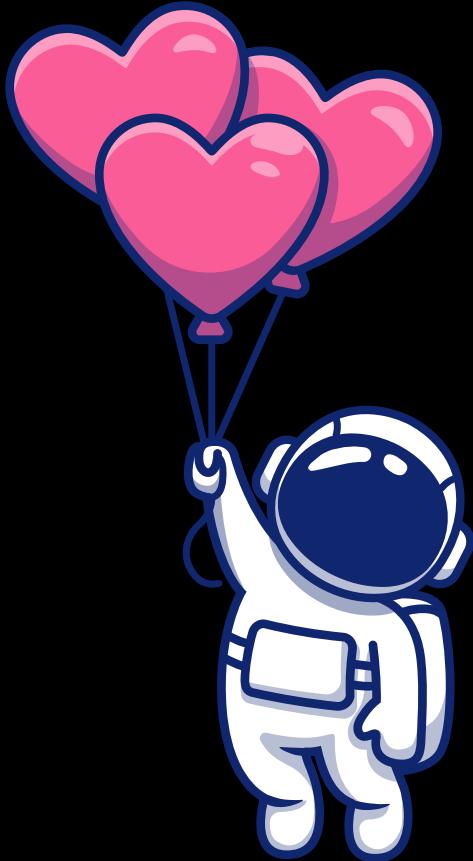
## EJERCICIO

### HERENCIA Y POLIMORFISMO



Universidad  
Nacional de Ingeniería  
IEEE Student Branch

Crea una clase Vehiculo con atributos como marca y modelo, y un método detalles que imprima información sobre el vehículo. Luego, crea clases Auto y Moto que hereden de Vehiculo y sobrescribe el método detalles en cada una para mostrar detalles específicos.



Auto:

Vehículo: Toyota Corolla

Moto:

Vehículo: Honda CBR



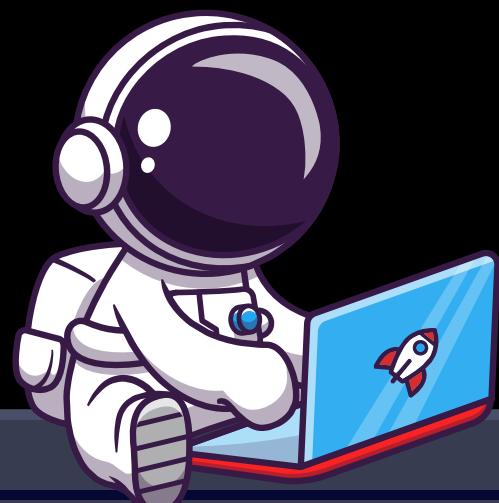
```
// Definición de la clase base llamada Vehiculo.  
class Vehiculo {  
    // Constructor de la clase Vehiculo que recibe la marca y el modelo.  
    constructor(marca, modelo) {  
        // Inicializa las propiedades de marca y modelo del vehículo.  
        this.marca = marca;  
        this.modelo = modelo;  
    }  
  
    // Método para mostrar los detalles del vehículo.  
    detalles() {  
        // Imprime en la consola el nombre del vehículo utilizando su marca y modelo.  
        console.log(`Vehículo: ${this.marca} ${this.modelo}`);  
    }  
}  
  
// Definición de la clase Auto que hereda de Vehiculo.  
class Auto extends Vehiculo {  
    // Método detalles() específico para la clase Auto.  
    detalles() {  
        // Imprime en la consola "Auto:" antes de llamar al método detalles() de la clase base.  
        console.log("Auto:");  
        // Llama al método detalles() de la clase base usando la palabra clave 'super'.  
        super.detalles();  
    }  
}  
  
// Definición de la clase Moto que hereda de Vehiculo.  
class Moto extends Vehiculo {  
    // Método detalles() específico para la clase Moto.  
    detalles() {  
        // Imprime en la consola "Moto:" antes de llamar al método detalles() de la clase base.  
        console.log("Moto:");  
        // Llama al método detalles() de la clase base usando la palabra clave 'super'.  
        super.detalles();  
    }  
}  
  
// Creación de instancias de las clases Auto y Moto.  
const auto = new Auto("Toyota", "Corolla");  
const moto = new Moto("Honda", "CBR");  
  
// Llama al método detalles() de las instancias auto y moto.  
auto.detalles();  
moto.detalles();
```



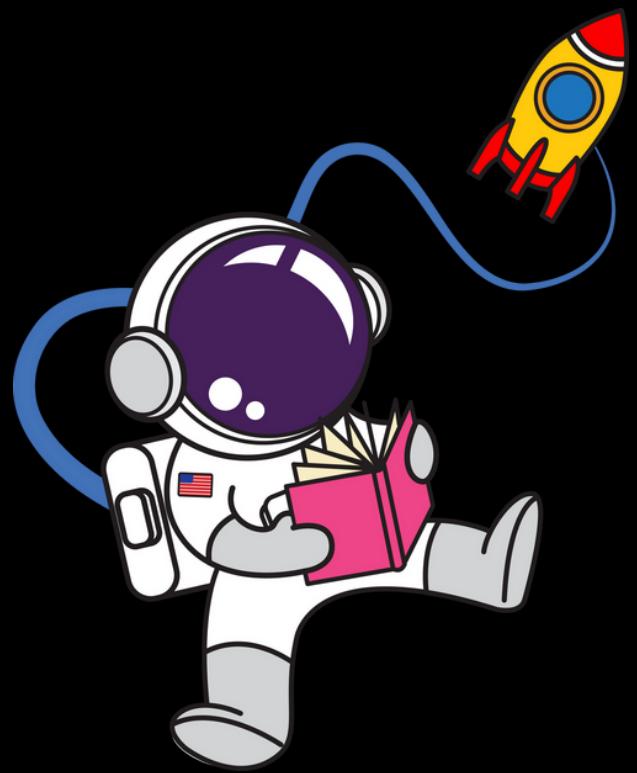
# 04

## EJERCICIO

INTERFACES Y CLASES ABSTRACTAS



Universidad  
Nacional de Ingeniería  
IEEE Student Branch



Crea una "interfaz" Forma que exija la implementación del método calcularArea(). Luego, crea clases Circulo y Rectangulo que implementen esta "interfaz".

Área del círculo: 78.53981633974483

Área del rectángulo: 24



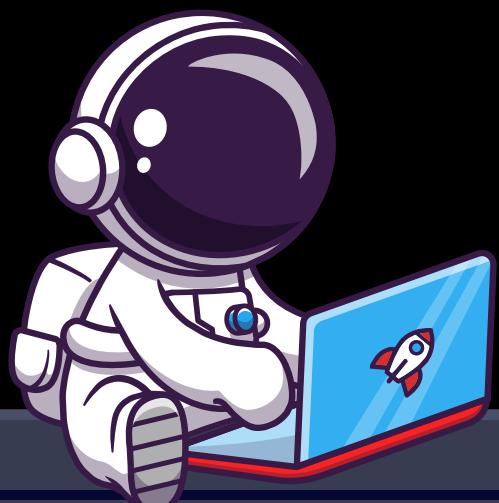
```
// Definición de la clase base llamada Forma.  
class Forma {  
    // Método que debe ser implementado en las clases derivadas para calcular el área.  
    calcularArea() {  
        // Lanza un error si este método no es implementado en una clase derivada.  
        throw new Error("El método calcularArea debe ser implementado.");  
    }  
}  
  
// Definición de la clase Circulo que hereda de Forma.  
class Circulo extends Forma {  
    constructor(radio) {  
        super(); // Llama al constructor de la clase base (Forma).  
        this.radio = radio;  
    }  
  
    // Implementación del método calcularArea específico para la clase Circulo.  
    calcularArea() {  
        // Calcula y devuelve el área del círculo usando la fórmula π * radio^2.  
        return Math.PI * this.radio ** 2;  
    }  
}  
  
// Definición de la clase Rectangulo que hereda de Forma.  
class Rectangulo extends Forma {  
    constructor(base, altura) {  
        super(); // Llama al constructor de la clase base (Forma).  
        this.base = base;  
        this.altura = altura;  
    }  
  
    // Implementación del método calcularArea específico para la clase Rectangulo.  
    calcularArea() {  
        // Calcula y devuelve el área del rectángulo usando la fórmula base * altura.  
        return this.base * this.altura;  
    }  
}  
  
// Creación de instancias de las clases Circulo y Rectangulo.  
const circulo = new Circulo(5);  
const rectangulo = new Rectangulo(4, 6);  
  
// Llama al método calcularArea() en las instancias circulo y rectangulo para calcular y mostrar el área.  
console.log("Área del círculo:", circulo.calcularArea());  
console.log("Área del rectángulo:", rectangulo.calcularArea());
```



# 05

## EJERCICIO

### ENCAPSULAMIENTO

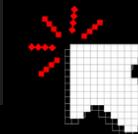


Universidad  
Nacional de Ingeniería  
IEEE Student Branch

Crea una clase CuentaBancaria que tenga atributos privados como saldo y titular. Implementa métodos para depositar, retirar y consultar el saldo de la cuenta, pero asegúrate de que solo se pueda acceder a estos atributos y métodos a través de métodos públicos



```
1000 depositado. Saldo actual: 1000
500 retirado. Saldo actual: 500
300 depositado. Saldo actual: 800
Historial de transacciones:
Depósito: +1000
Retiro: -500
Depósito: +300
```



```
// Definición de la clase CuentaBancaria.
class CuentaBancaria {
    // Constructor que inicializa el titular, el saldo inicial y un historial de transacciones.
    constructor(titular, saldoInicial = 0) {
        // Asigna el titular proporcionado a la cuenta.
        this._titular = titular;
        // Asigna el saldo inicial proporcionado a la cuenta (por defecto, es 0 si no se proporciona).
        this._saldo = saldoInicial;
        // Crea un arreglo vacío para almacenar el historial de transacciones.
        this._historialTransacciones = [];
    }

    // Método para depositar una cantidad en la cuenta.
    depositar(cantidad) {
        if (cantidad > 0) {
            // Verifica si la cantidad a depositar es positiva.
            this._saldo += cantidad; // Incrementa el saldo por la cantidad depositada.
            // Agrega una entrada al historial de transacciones indicando el depósito realizado.
            this._historialTransacciones.push(`Depósito: +${cantidad}`);
            // Muestra un mensaje en la consola con el detalle del depósito y el saldo actual.
            console.log(`${cantidad} depositado. Saldo actual: ${this._saldo}`);
        }
    }

    // Método para retirar una cantidad de la cuenta.
    retirar(cantidad) {
        if (cantidad > 0 && cantidad <= this._saldo) {
            // Verifica si la cantidad a retirar es válida.
            this._saldo -= cantidad; // Decrementa el saldo por la cantidad retirada.
            // Agrega una entrada al historial de transacciones indicando el retiro realizado.
            this._historialTransacciones.push(`Retiro: -${cantidad}`);
            // Muestra un mensaje en la consola con el detalle del retiro y el saldo actual.
            console.log(`${cantidad} retirado. Saldo actual: ${this._saldo}`);
        } else {
            console.log("Fondos insuficientes o cantidad inválida.");
        }
    }

    // Método para consultar el saldo actual de la cuenta.
    consultarSaldo() {
        // Muestra en la consola el saldo actual.
        console.log(`Saldo actual: ${this._saldo}`);
    }

    // Método para mostrar el historial de transacciones.
    mostrarHistorial() {
        console.log("Historial de transacciones:");
        // Itera a través de cada entrada en el historial de transacciones y la muestra en la consola.
        for (let i = 0; i < this._historialTransacciones.length; i++) {
            // En cada iteración, obtiene el elemento actual del historial usando
            this._historialTransacciones[i];
            const transaccion = this._historialTransacciones[i];
            // Muestra en la consola la transacción actual.
            console.log(transaccion);
        }
    }
}

// Creación de una instancia de la clase CuentaBancaria con el titular "Juan".
const cuenta = new CuentaBancaria("Juan");

// Llamadas a los métodos para realizar operaciones en la cuenta.
cuenta.depositar(1000); // Deposita 1000 en la cuenta.
cuenta.retirar(500); // Retira 500 de la cuenta.
cuenta.depositar(300); // Deposita 300 en la cuenta.
cuenta.mostrarHistorial(); // Muestra el historial de transacciones.
```

\*\*\*\*\*  
GAME OVER

GRACIAS

