JHONNY DAVID JARAMA TRAPIELLO
Y
LEYRIN BRIDNEYS AGUILAR

IEEE AESS UNI

PROGRAMACIÓN ORIENTADA A OBJETOS

SECCIÓN 2

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

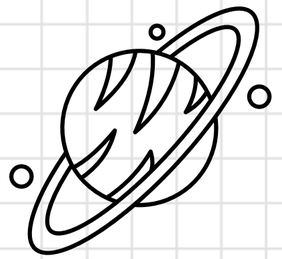
Universidad
Nacional de Ingeniería
IEEE Student Branch

IMPLEMENTACIÓN DE CLASES Y OBJETOS EN JAVASCRIPT

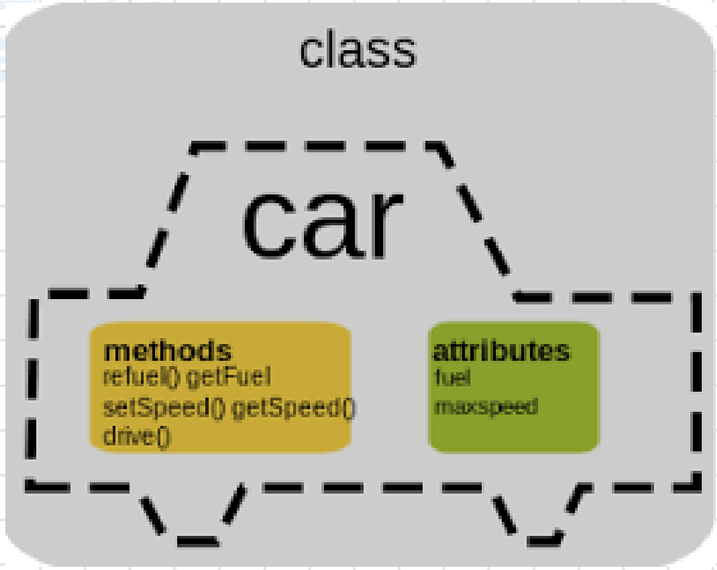
CLASES EN JAVASCRIPT:

A partir de ECMAScript 2015 (ES6), JavaScript introdujo una nueva sintaxis para definir clases que se asemeja más a la programación orientada a objetos.

```
CLASS MICLASE {
  CONSTRUCTOR(PARAMETRO1, PARAMETRO2) {
    THIS.ATRIBUTO1 = PARAMETRO1;
    THIS.ATRIBUTO2 = PARAMETRO2;
  }

  METODO1() {
    // CÓDIGO DEL MÉTODO
  }
}
```

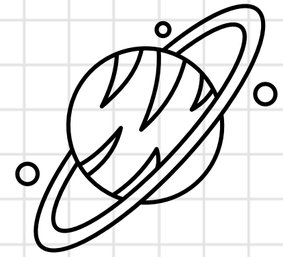


- La palabra clave class se utiliza para definir una nueva clase.
- El método constructor se llama cuando se crea una nueva instancia de la clase y se utiliza para inicializar los atributos del objeto.
- Los métodos dentro de la clase son funciones que definen el comportamiento del objeto.



OBJETOS EN JAVASCRIPT:

Una vez que tienes una clase definida, puedes crear objetos (instancias) de esa clase utilizando el operador new.

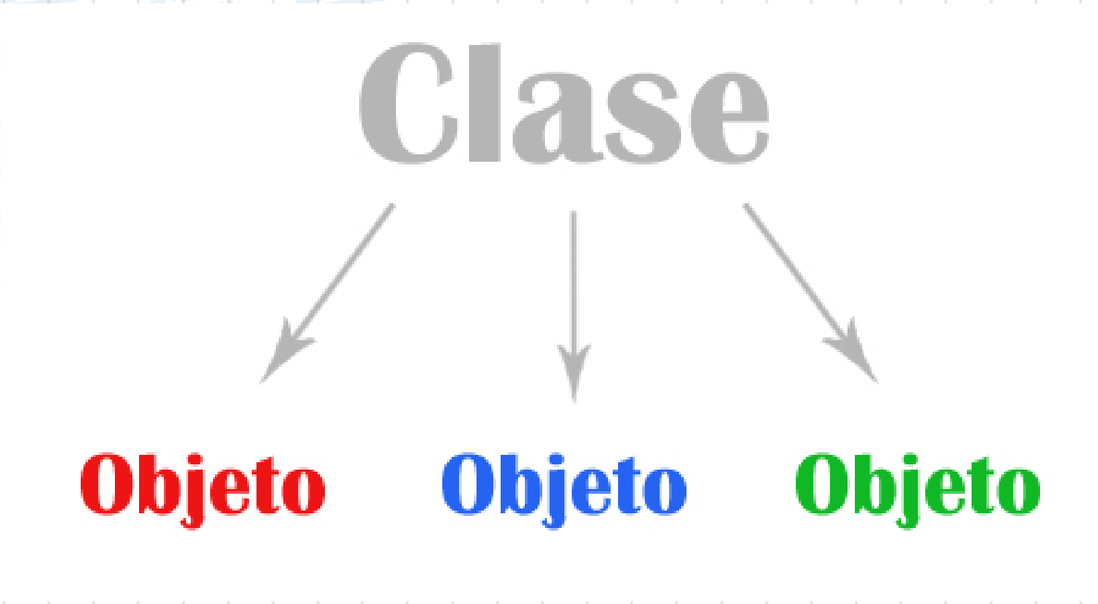


```
CONST OBJETO1 = NEW MICLASE(VALOR1, VALOR2);
// CREACIÓN DE UN OBJETO

CONSOLE.LOG(OBJETO1.ATRIBUTO1); // ACCESO A ATRIBUTOS
OBJETO1.METODO1(); // LLAMADA A MÉTODOS
```

- La palabra clave new se usa para crear una nueva instancia de una clase.

- Los atributos y métodos de la instancia se acceden utilizando la notación de punto (objeto.atributo o objeto.metodo()).

Las clases en JavaScript proporcionan una manera estructurada de definir la estructura y el comportamiento de los objetos. Una vez definida una clase, puedes crear múltiples objetos (instancias) basados en esa clase y trabajar con ellos utilizando sus atributos y métodos.



EJEMPLO CLARO

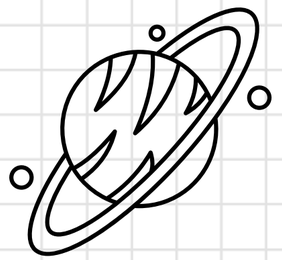
```
// DEFINICIÓN DE UNA CLASE LLAMADA "PERSONA"
CLASS PERSONA {
    CONSTRUCTOR(NOMBRE, EDAD) {
        THIS.NOMBRE = NOMBRE;
        THIS.EDAD = EDAD;
    }

    SALUDAR() {
        CONSOLE.LOG(`HOLA, SOY ${THIS.NOMBRE} Y TENGO ${THIS.EDAD} AÑOS.`);
    }
}

// INSTANCIACIÓN DE OBJETOS BASADOS EN LA CLASE "PERSONA"
CONST PERSONA1 = NEW PERSONA("JUAN", 30);
CONST PERSONA2 = NEW PERSONA("MARÍA", 25);

// USO DE MÉTODOS EN LOS OBJETOS
PERSONA1.SALUDAR();  // SALIDA: "HOLA, SOY JUAN Y TENGO 30 AÑOS."
PERSONA2.SALUDAR();  // SALIDA: "HOLA, SOY MARÍA Y TENGO 25 AÑOS."
```

MÉTODOS Y ATRIBUTOS ESTÁTICOS Y NO ESTATICOS

Los métodos y atributos pueden ser estáticos o no estáticos (también conocidos como métodos y atributos de instancia). La diferencia clave radica en cómo se accede y se utiliza cada uno de ellos.

ATRIBUTOS Y MÉTODOS NO ESTÁTICOS (DE INSTANCIA):

- Atributos de Instancia: Son propiedades únicas para cada instancia de la clase. Cada objeto tiene su propio conjunto de valores para estos atributos.



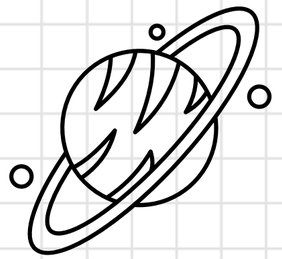
- Métodos de Instancia: Son funciones asociadas a objetos específicos creados a partir de la clase. Tienen acceso a las propiedades y métodos de instancia a través de la palabra clave this.

AESS IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY
Universidad Nacional de Ingeniería
IEEE Student Branch

ATRIBUTOS Y MÉTODOS ESTÁTICOS:

- Atributos Estáticos: Son propiedades compartidas por todas las instancias de una clase. Se definen en la clase misma y no están asociados con objetos individuales.

- Métodos Estáticos: Son funciones que pertenecen a la clase en sí, no a instancias individuales. No pueden acceder a propiedades de instancia directamente, ya que no tienen acceso a this.

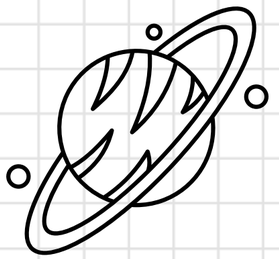


EJEMPLO CLARO

```
CLASS PERSONA {
    CONSTRUCTOR(NOMBRE) {
        THIS.NOMBRE = NOMBRE; // ATRIBUTO DE INSTANCIA
    }
    SALUDAR() {
        CONSOLE.LOG(`HOLA, SOY ${THIS.NOMBRE}`);
    }
    STATIC CONTARPERSONAS() {
        CONSOLE.LOG("HAY UN TOTAL DE PERSONAS: " + PERSONA.CONTADOR); // ATRIBUTO ESTÁTICO
    }
}
// ATRIBUTO ESTÁTICO PARA CONTAR PERSONAS
PERSONA.CONTADOR = 0;

CONST PERSONA1 = NEW PERSONA("JUAN");
CONST PERSONA2 = NEW PERSONA("MARÍA");

PERSONA1.SALUDAR();  // SALIDA: "HOLA, SOY JUAN"
PERSONA2.SALUDAR();  // SALIDA: "HOLA, SOY MARÍA"

PERSONA.CONTADOR++;  // INCREMENTAR EL CONTADOR ESTÁTICO

PERSONA.CONTARPERSONAS();  // SALIDA: "HAY UN TOTAL DE PERSONAS: 2"
```

En este ejemplo, nombre es un atributo de instancia, ya que cada objeto Persona tiene su propio nombre. saludar() es un método de instancia, ya que está disponible para objetos individuales y puede acceder a las propiedades de instancia a través de this.

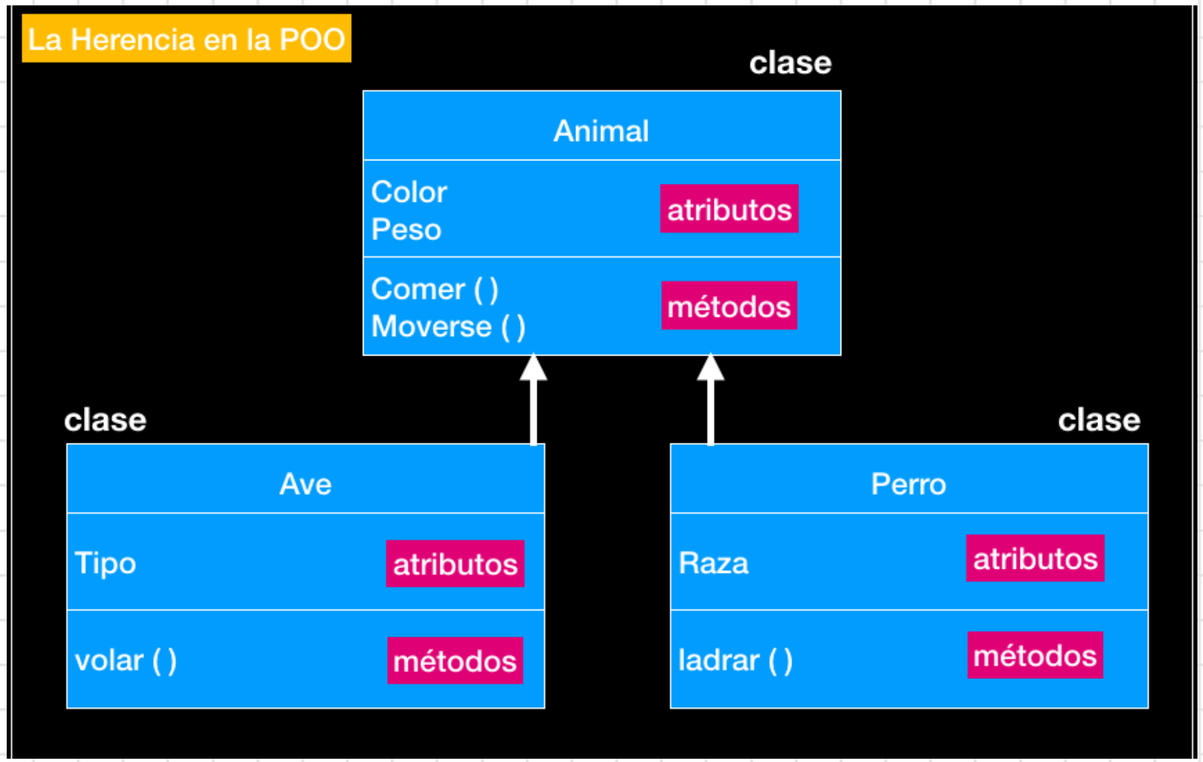
contarPersonas() y **contador** son ejemplos de atributo y método estático. **contador** es un atributo estático que se comparte entre todas las instancias de **Persona**, y **contarPersonas()** es un método estático que puede acceder a atributos estáticos, pero no a propiedades de instancia.



HERENCIA Y POLIFORMISMO EN POO

HERENCIA:

La herencia en JavaScript se implementa utilizando la palabra clave extends para crear subclases que heredan de una clase base. Las subclases pueden ampliar o sobrescribir métodos y propiedades de la clase base.

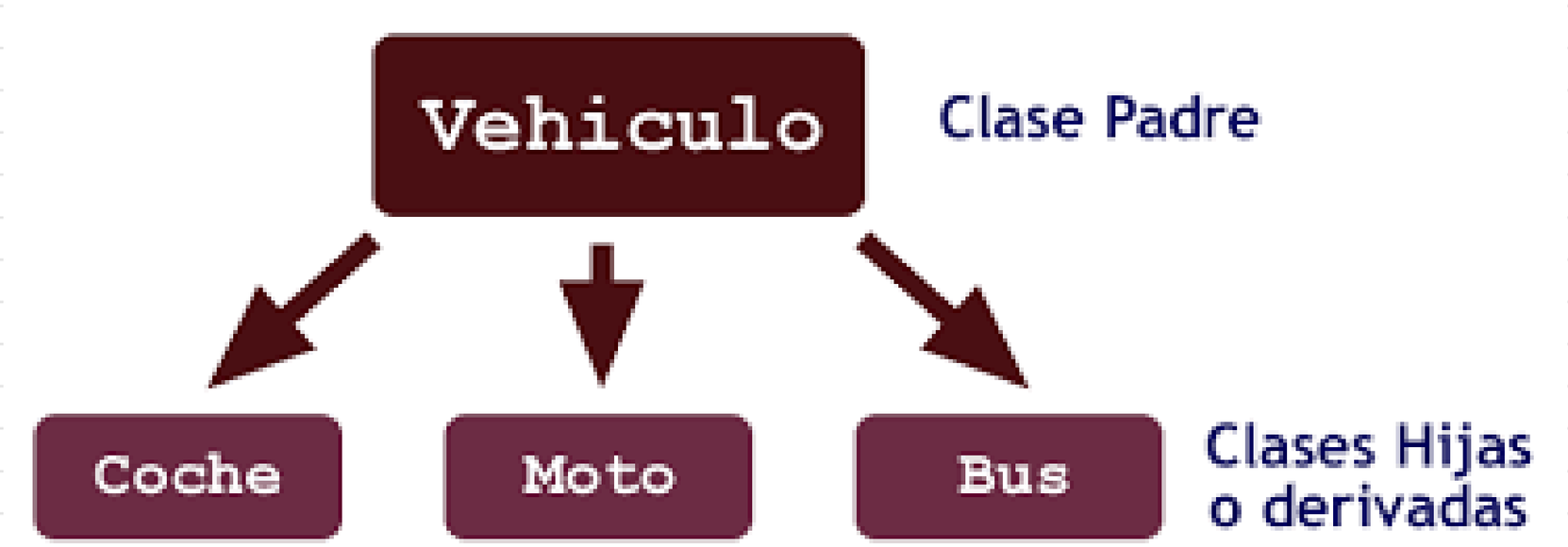


EJEMPLO CLARO

```
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!";
  }
}
CONST PERRO = NEW PERRO("BUDDY");
CONST GATO = NEW GATO("WHISKERS");

CONSOLE.LOG(PERRO.HACERSONIDO());  // SALIDA: "WOOF!"
CONSOLE.LOG(GATO.HACERSONIDO());   // SALIDA: "MEOW!"
```

POLIMORFISMO:

El polimorfismo en JavaScript se refleja en cómo diferentes objetos pueden responder al mismo método, pero con implementaciones diferentes. Aquí un ejemplo:



EJEMPLO CLARO

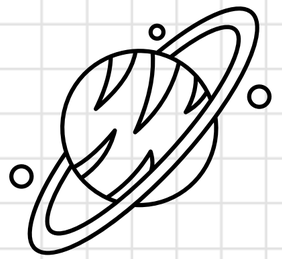
```
CLASS FORMA {
  AREA() {
    RETURN 0;
  }
}
CLASS CIRCULO EXTENDS FORMA {
  CONSTRUCTOR(RADIO) {
    SUPER();
    THIS.RADIO = RADIO;
  }
  AREA() {
    RETURN MATH.PI * THIS.RADIO * THIS.RADIO;
  }
}
CLASS RECTANGULO EXTENDS FORMA {
  CONSTRUCTOR(ANCHO, ALTO) {
    SUPER();
    THIS.ANCHO = ANCHO;
    THIS.ALTO = ALTO;
  }
  AREA() {
    RETURN THIS.ANCHO * THIS.ALTO;
  }
}
CONST CIRCULO = NEW CIRCULO(5);
CONST RECTANGULO = NEW RECTANGULO(4, 6);

CONSOLE.LOG(CIRCULO.AREA());     // SALIDA: ÁREA DEL CÍRCULO
CONSOLE.LOG(RECTANGULO.AREA());  // SALIDA: ÁREA DEL RECTÁNGULO
```

Ambas subclases (Circulo y Rectangulo) heredan de la clase base Forma y tienen su propio método area(). Cuando se llama al método area() en cada objeto, JavaScript utiliza la implementación correspondiente de acuerdo a su tipo, demostrando así el polimorfismo.

INTERFACES Y CLASES ABSTRACTAS EN POO

A diferencia de algunos otros lenguajes de programación orientada a objetos, como Java o C#, JavaScript no tiene soporte nativo para interfaces o clases abstractas. Sin embargo, puedes lograr funcionalidades similares utilizando ciertos patrones y características de JavaScript.

INTERFACES:



Una interfaz en POO define un contrato que las clases deben seguir, especificando métodos que deben ser implementados. En JavaScript, no hay una sintaxis nativa para definir interfaces, pero puedes lograr algo similar documentando tus clases para indicar que deben cumplir con ciertos métodos.

AESS IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

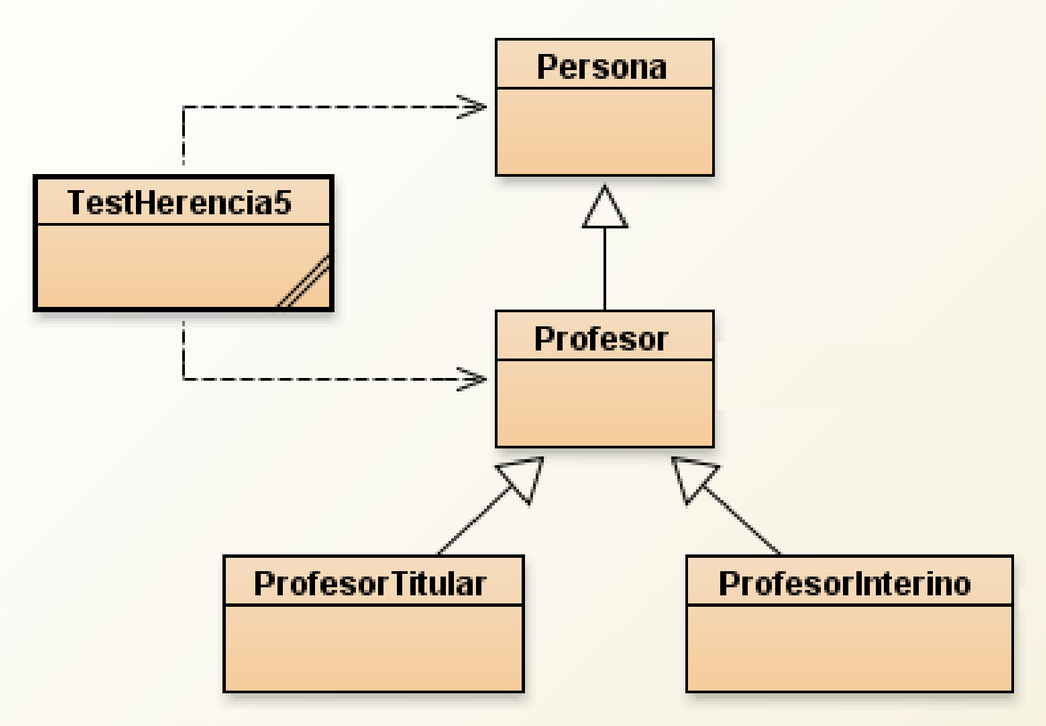
Universidad Nacional de Ingeniería
IEEE Student Branch

EJEMPLO CLARO

```
// EJEMPLO DE "INTERFAZ" USANDO COMENTARIOS DE DOCUMENTACIÓN
/**
 * @INTERFACE
 */
CLASS IANIMAL {
    HACERSONIDO() {}
}

CLASS PERRO EXTENDS IANIMAL {
    HACERSONIDO() {
        RETURN "WOOF!";
    }
}

CLASS GATO EXTENDS IANIMAL {
    HACERSONIDO() {
        RETURN "MEOW!";
    }
}
```

CLASES ABSTRACTAS:

Una clase abstracta es una clase que no puede ser instanciada directamente y generalmente proporciona una base para que otras clases hereden. En JavaScript, puedes simular clases abstractas usando métodos lanzadores de excepciones que deben ser sobrescritos por las subclases.

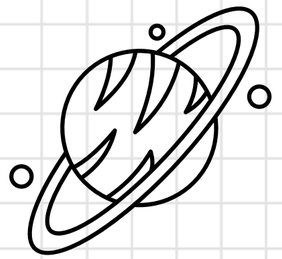


EJEMPLO CLARO

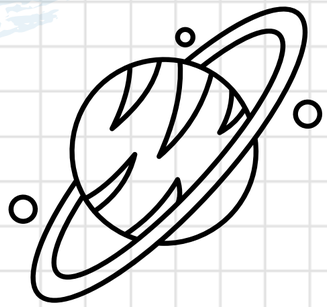
```
CLASS FORMA {
  CONSTRUCTOR() {
    IF (NEW.TARGET === FORMA) {
      THROW NEW ERROR("FORMA NO PUEDE SER INSTANCIADA DIRECTAMENTE.");
    }
  }
  AREA() {
    THROW NEW ERROR("EL MÉTODO 'AREA' DEBE SER IMPLEMENTADO POR SUBCLASES.");
  }
}
CLASS CIRCULO EXTENDS FORMA {
  CONSTRUCTOR(RADIO) {
    SUPER();
    THIS.RADIO = RADIO;
  }
  AREA() {
    RETURN MATH.PI * THIS.RADIO * THIS.RADIO;
  }
}
```

En este ejemplo, la clase Forma es una clase abstracta y su método area() lanza una excepción indicando que debe ser implementado por las subclases. Las clases concretas como Circulo deben proporcionar una implementación para el método area().

Recuerda que estos enfoques no son tan estrictos como en lenguajes con soporte nativo para interfaces y clases abstractas, pero pueden ayudarte a lograr estructuras similares en JavaScript.



MUCHAS GRACIAS



AESS
IEEE AEROSPACE AND ELECTRONIC SYSTEMS SOCIETY

Universidad Nacional de Ingeniería
IEEE Student Branch