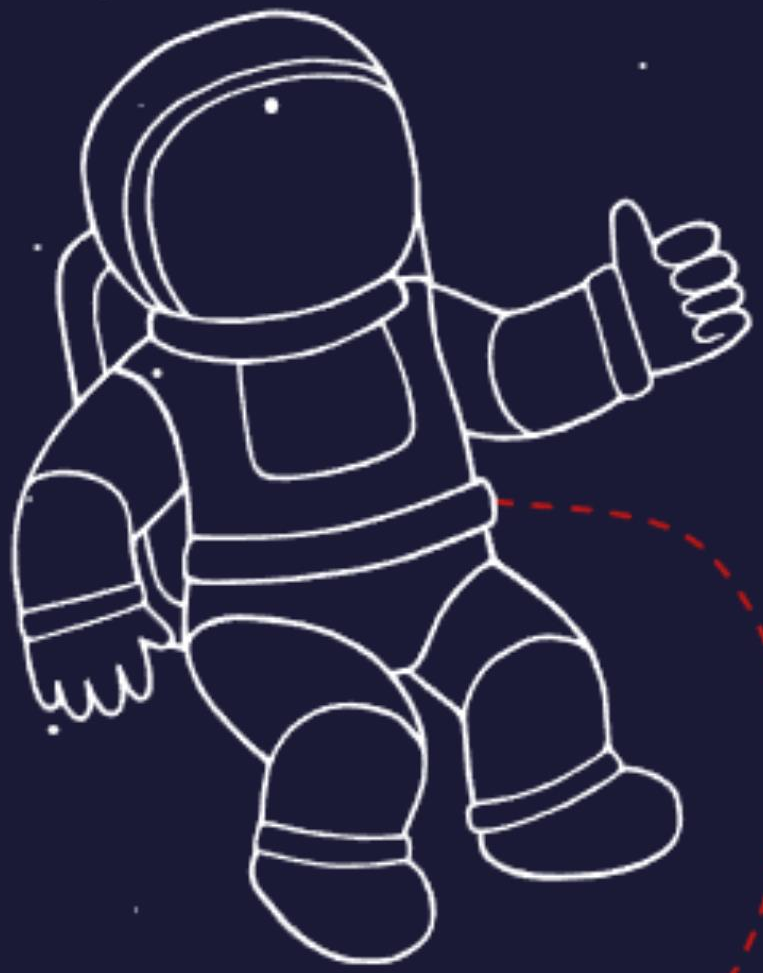
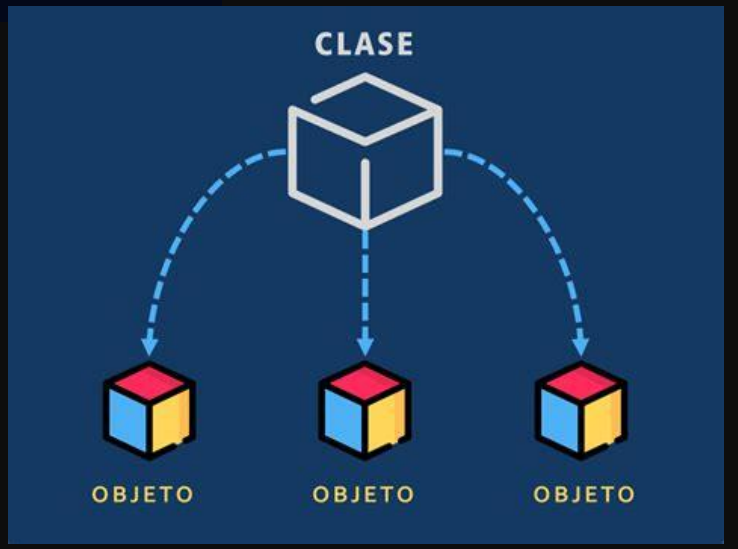
Programa académico CAMPUS



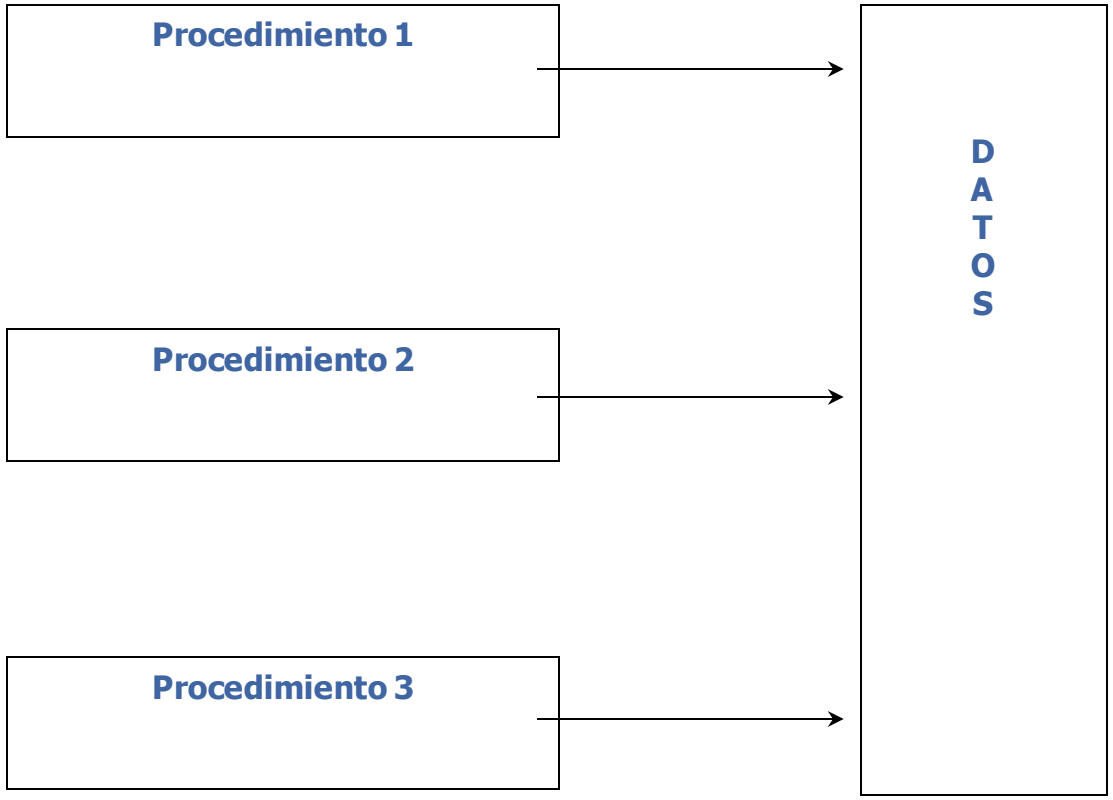
PROGRAMACIÓN ORIENTADA A OBJETOS (POO)



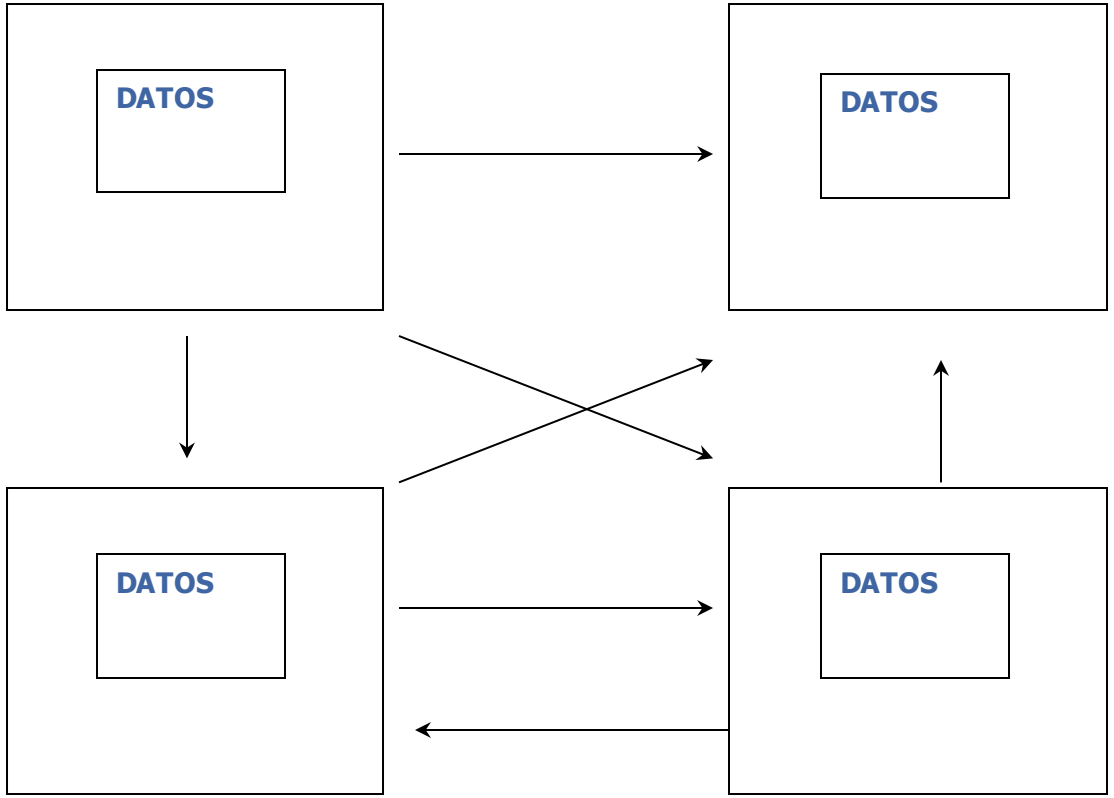
PILARES DE LA POO

Programación procedimental VS POO

Procedimental



POO



Programación procedimental VS POO

¿Qué es?	• Es un paradigma de programación que usa objetos y las interacciones entre los mismos
¿Por qué?	• Necesidad de organizar el código fuente • Evitar líneas de código innecesarias(Repetidas)
¿Para qué?	• Diseñar programas informáticos y aplicaciones • Proteger los datos de modificaciones incontroladas

Programación procedimental VS POO

- La **POO** está basada en el comportamiento o modo de actuar del hombre y no en el de la máquina.
- La **POO** se fundamenta o sustenta en los siguientes conceptos:
  - Clase
  - Objetos
  - Instancia
  - Atributos
  - Métodos

CLASES

General - Genérico

CLASE: Asociación de

Ejemplo: Silla

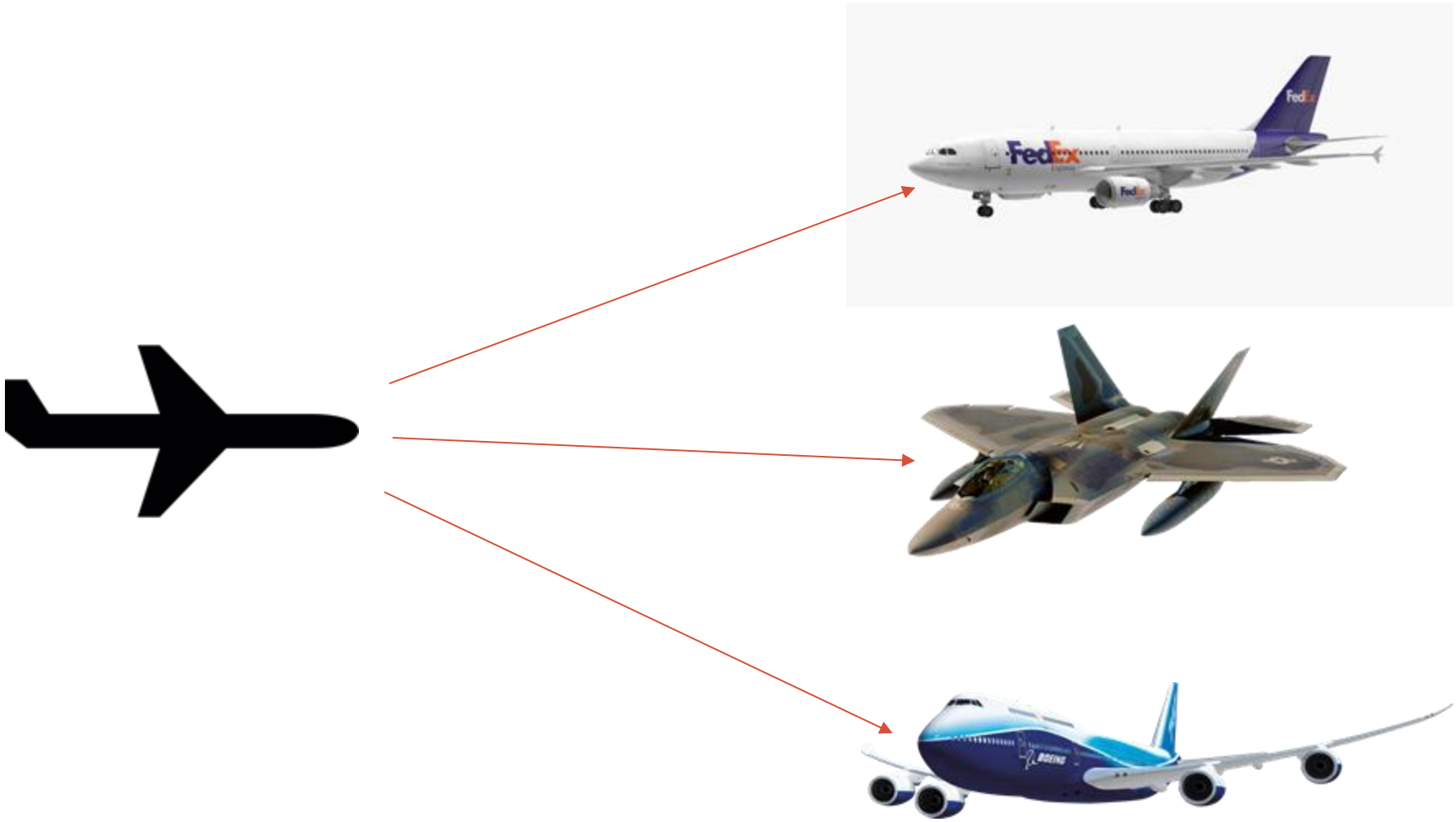
Atributos (Características o propiedades) => Información que identifica la clase

Silla=> Color, material, número de patas

Métodos (Operaciones – Acciones) => Usos de la clase

Silla=> sentar(), golpear(), alcanzar()

CLASES



OBJETOS

Particular - Específico, Instancia de una Clase

OBJETO: Asociación de

Ejemplo: La silla donde estamos sentados, la silla del comedor, la silla de espera en un Banco

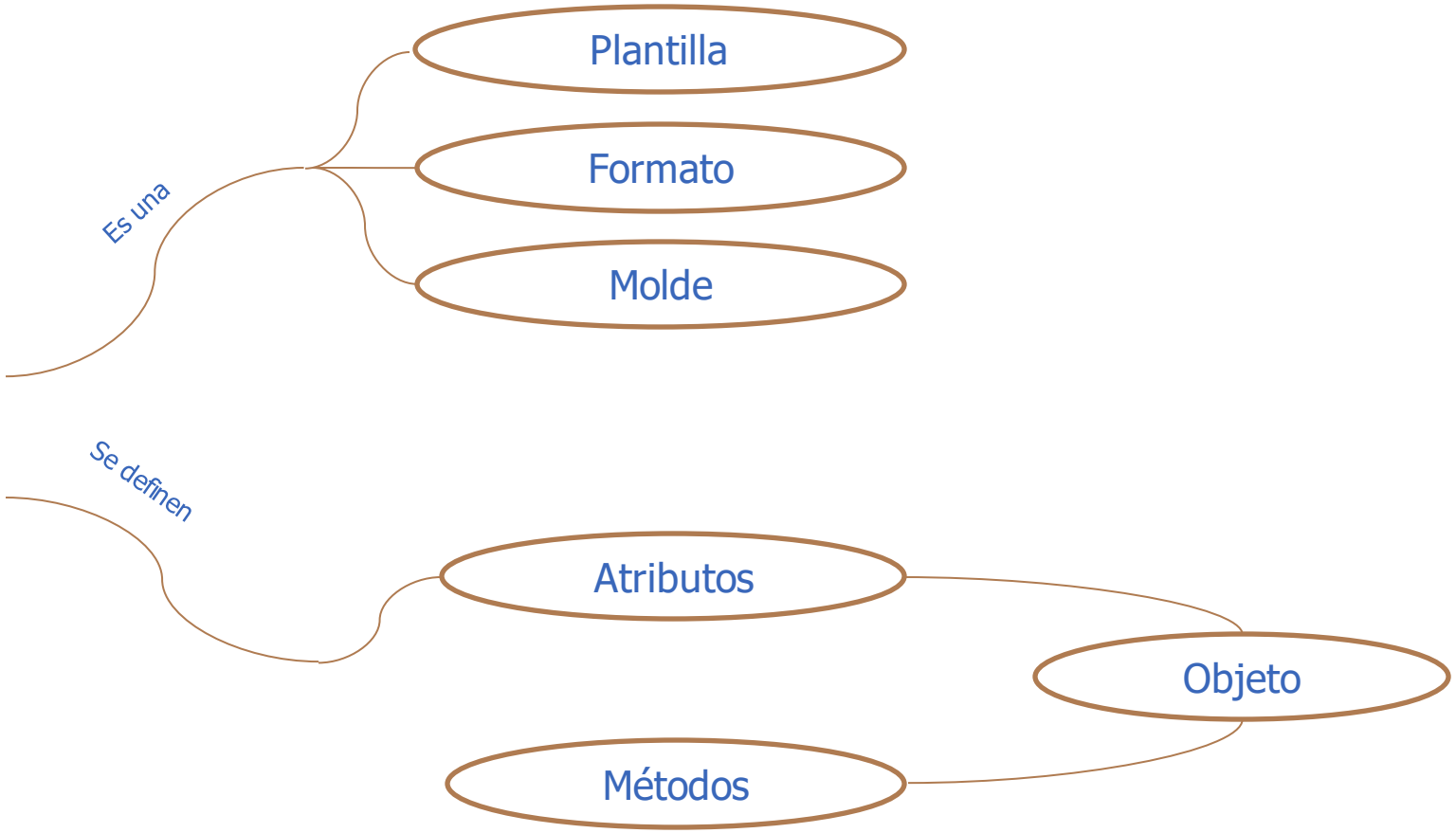
Atributos (Características o propiedades)

Silla=> Color= Negro, material=madera, número de patas=2

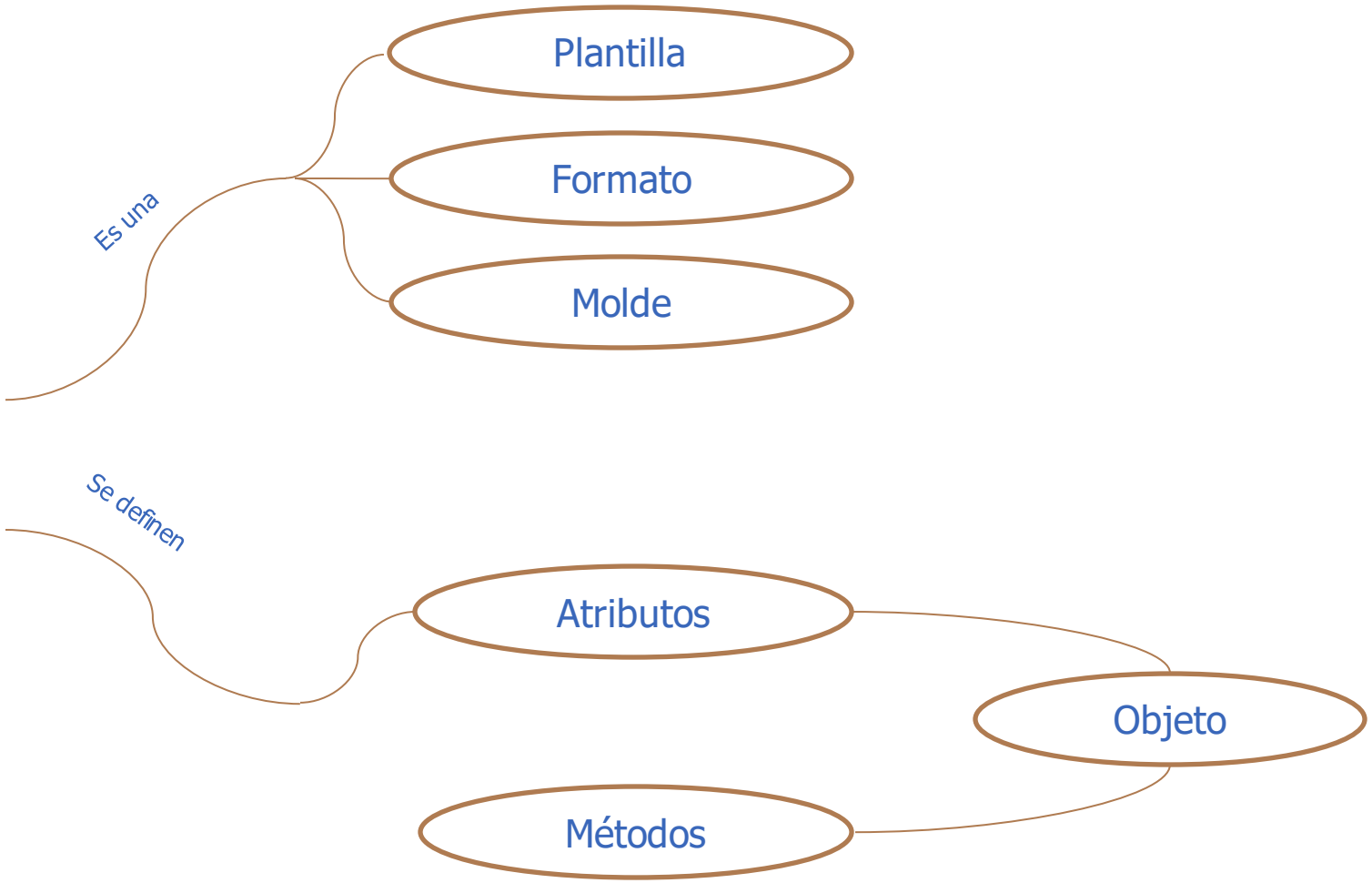
Métodos (Operaciones – Acciones)

Silla=> sentar(), golpear(), alcanzar()

CLASES


Clase
Es una
Plantilla
Formato
Molde
Se definen
Atributos
Objeto
Métodos

OBJETOS


Clase
Es una
Plantilla
Formato
Molde
Se definen
Atributos
Objeto
Métodos

INSTANCIAS

Se le llama instancia a una ocurrencia de la clase.

La instanciación se produce al momento de crear un objeto.

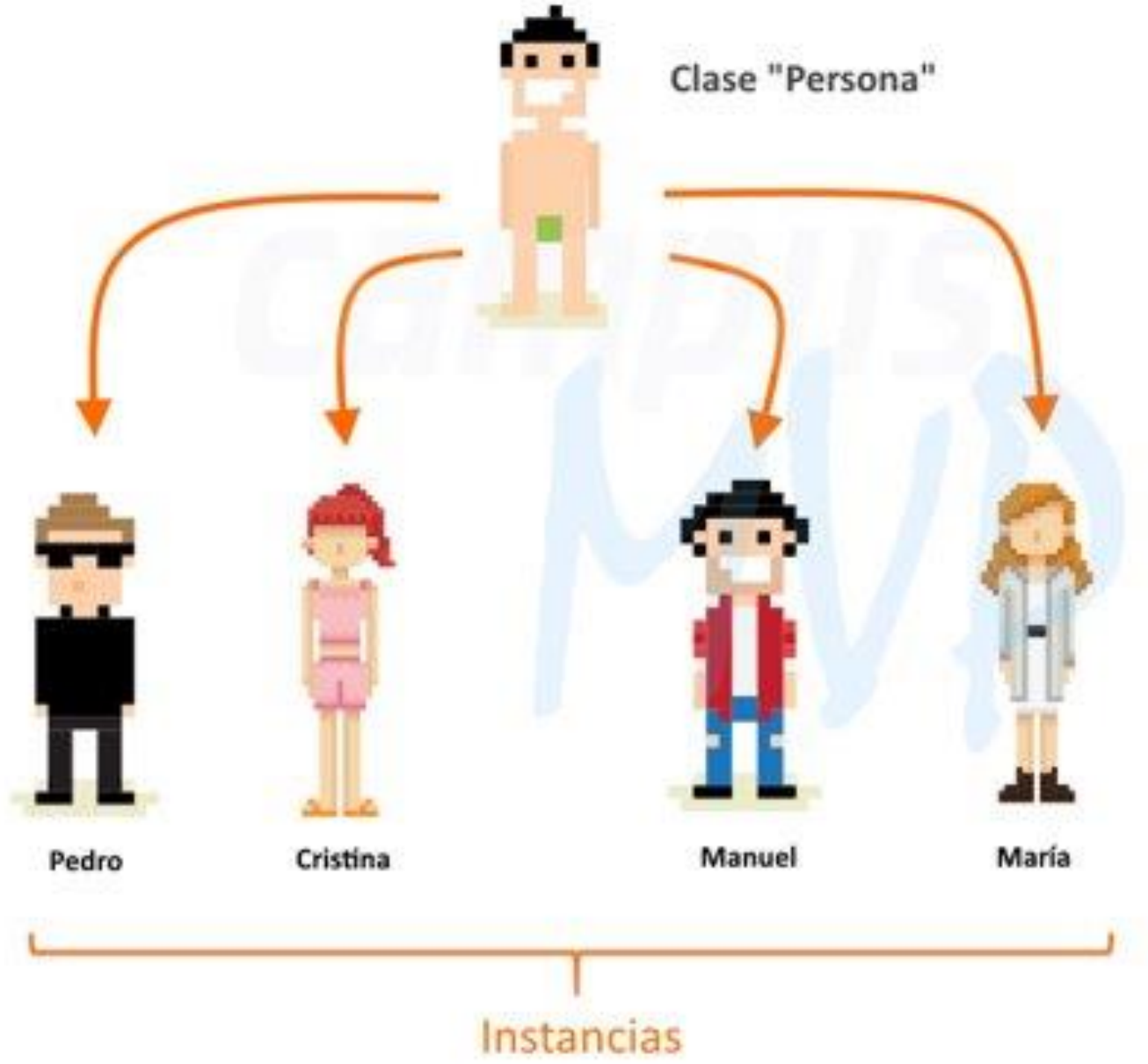
Un objeto es una instancia de una clase específica.

EJEMPLO:

Tenemos la clase Vehículo

- El auto con placa OWF-463 es una instancia de la clase Vehículo, o sea, un objeto de esa clase.
- La camioneta patente HZT-283 es otra instancia de la clase Vehículo

INSTANCIAS



ATRIBUTOS

Son las características de los objetos de una clase y determinan el estado de un objeto.

- Marca
- Año
- Color
- Placa
- Etc.

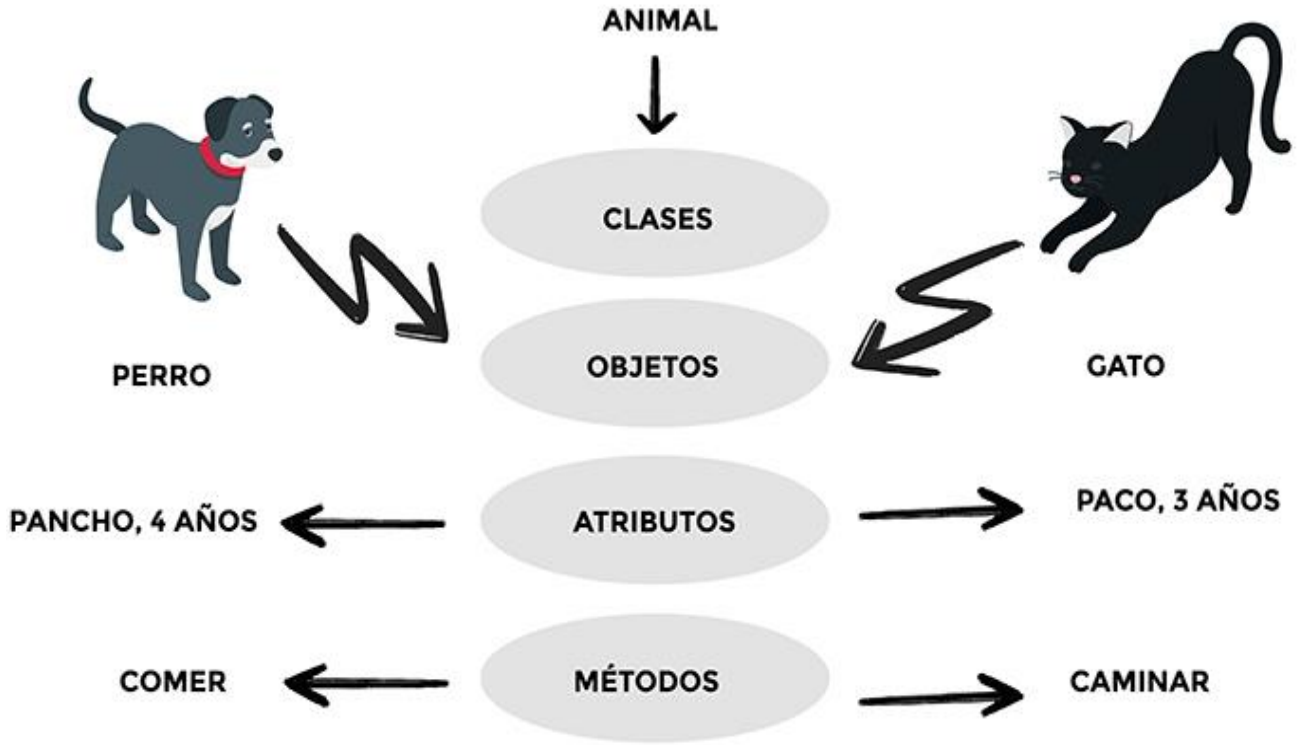
MÉTODOS

Se definen como todas aquella acciones que se pueden realizar sobre un objeto de cierta clase.

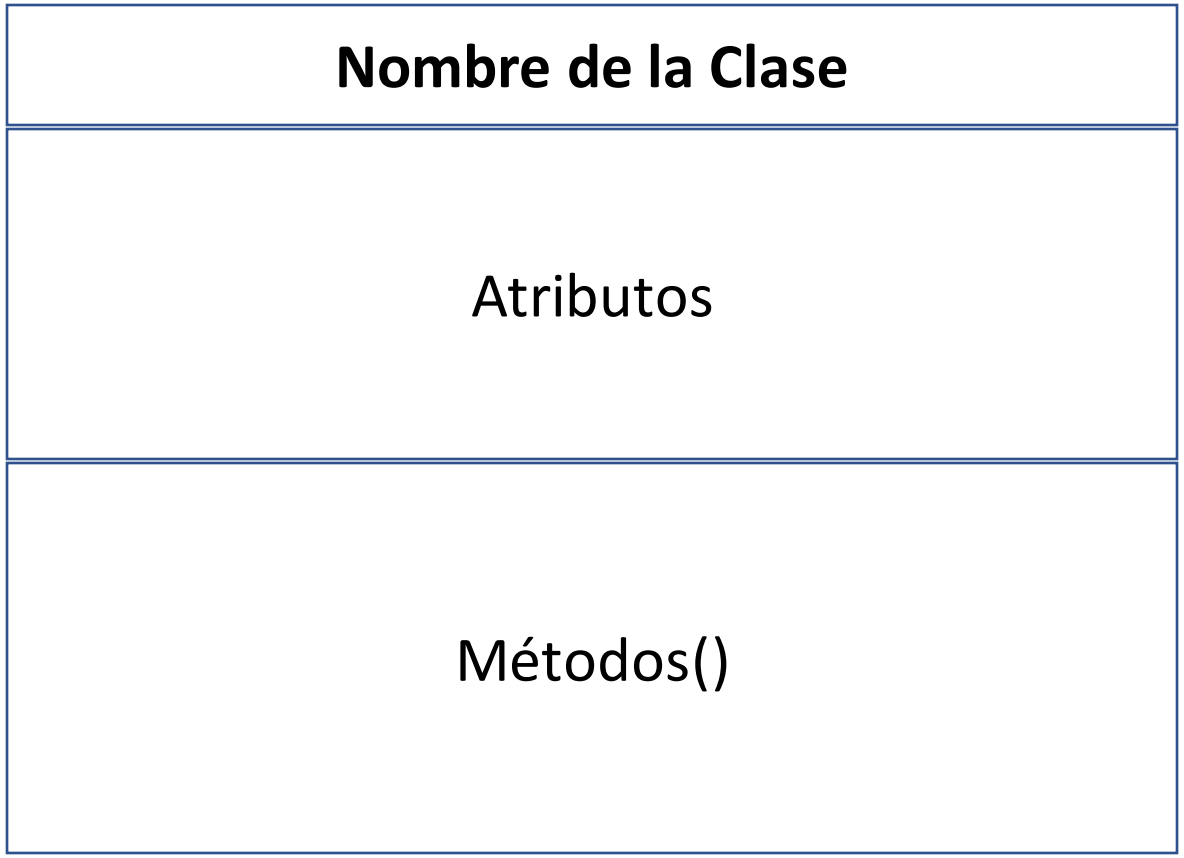
En las implementación, estos métodos son segmentos de código en la forma de funciones.

La clase Vehículo puede incluir los métodos:

- Encender
- Acelerar
- Virar
- Frenar



REPRESENTACIÓN



REPRESENTACIÓN

Por ejemplo:

la clase Silla

REPRESENTACIÓN

Por ejemplo:

la clase Silla

EJEMPLO

- Clase: **Cuenta corriente**
- **Atributos:**
  - Número
  - Nombre
  - Saldo
- **Métodos:**
  - Depositar
  - Girar
  - Consultar saldo

EJEMPLO

Clase: **Cuenta corriente**

Clase: **Cuenta corriente**
- **Atributos:**
  - Número
  - Nombre
  - Saldo
- **Métodos:**
  - Depositar
  - Girar
  - Consultar saldo

- Instanciación: **Cuenta corriente A, B**

Objeto **A**

Numero 1234

Nombre: Juan

Saldo: 350.000

Métodos

Objeto **B**

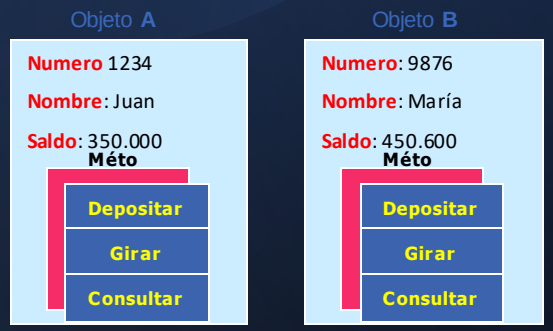
Numero: 9876

Nombre: María

Saldo: 450.600

Métodos

EJEMPLO EN CÓDIGO DE OBJETOS



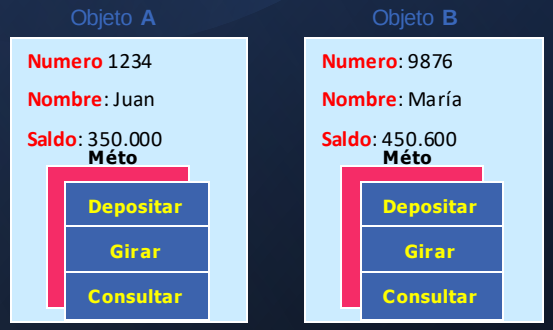
Clase: **Cuenta corriente**

Clase: **Cuenta corriente**

- **Atributos:**
  - Número
  - Nombre
  - Saldo
- **Métodos:**
  - Depositar
  - Girar
  - Consultar saldo

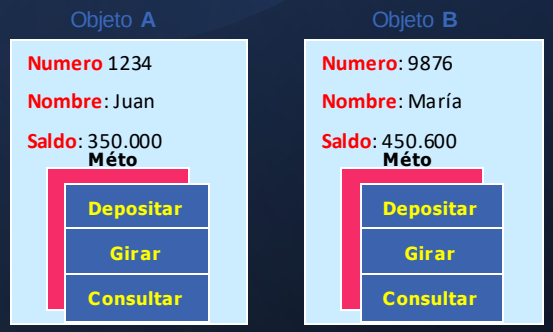
```
const CuentaCorriente = {
    numero: '123',
    nombre: 'David Rodriguez',
    saldo: 1500000,
    mostrarSaldo() {
        return `La cuenta ${this.numero} de
${this.nombre} tiene un saldo de ${this.saldo}`;
    }
}

console.log(CuentaCorriente.mostrarSaldo());
```

EJEMPLO EN CÓDIGO DE VARIOS OBJETOS



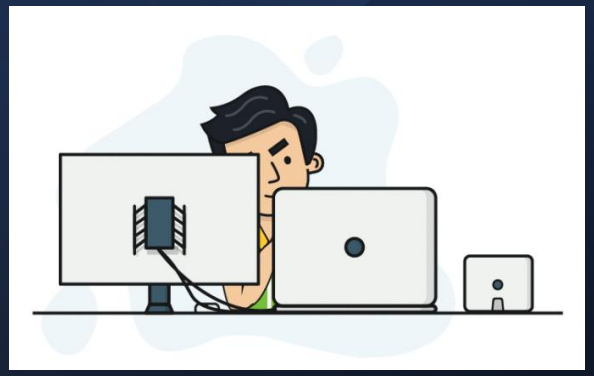

```
const CuentaCorriente1 = {
    numero: '123',
    nombre: 'David Rodriguez',
    saldo: 1500000,
    mostrarSaldo() {
        return `La cuenta ${this.numero} de ${this.nombre}
tiene un saldo de ${this.saldo}`;
    }
}
console.log(CuentaCorriente1.mostrarSaldo());

const CuentaCorriente2 = {
    numero: '579',
    nombre: 'Dario Manzanares',
    saldo: 2300000,
    mostrarSaldo() {
        return `La cuenta ${this.numero} de ${this.nombre}
tiene un saldo de ${this.saldo}`;
    }
}
console.log(CuentaCorriente2.mostrarSaldo());
```

EJEMPLO EN CÓDIGO DE VARIOS OBJETOS




```
class CuentaCorriente {
  constructor(numero, nombre, saldo) {
    this.numero = numero;
    this.nombre = nombre;
    this.saldo = saldo;
  }
  mostrarSaldo() {
    return `La cuenta ${this.numero} de
${this.nombre} tiene un saldo de ${this.saldo}`;
  }
}

const cuenta1 = new CuentaCorriente(579, "David
Rodriguez", 1_500_000);
const cuenta2 = new CuentaCorriente(123, "Dario
Manzanares", 2_300_000);

console.log(cuenta1.mostrarSaldo());
console.log(cuenta2.mostrarSaldo());
```

EJERCICIO #1

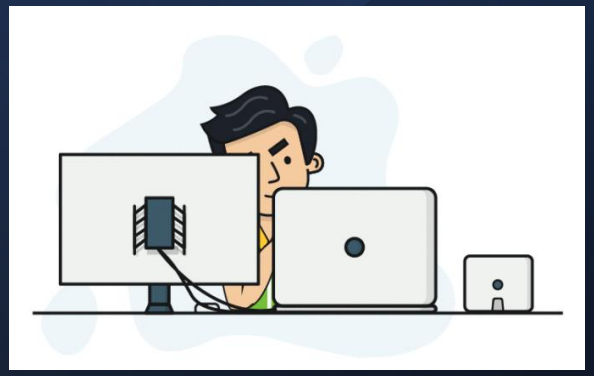


Dada un número entero, Se pide:

- Conocer si es par o impar
- Conocer si es positivo, negativo o cero
- Conocer si es primo

Realizar el programa en JavaScript que resuelva la situación problema presentada, utilizando el concepto de Clases y Objetos (POO).

EJERCICIO #2



Dada la siguiente información sobre los suscriptores del servicio de agua, de los cuales se conoce:

- Código
- Nombre
- Estrato(1,2,3,4,5)
- Consumo

Se pide calcular el valor a pagar por concepto de servicio de agua de cada suscriptor y el TOTAL (Todos), de acuerdo con la siguiente indicación:

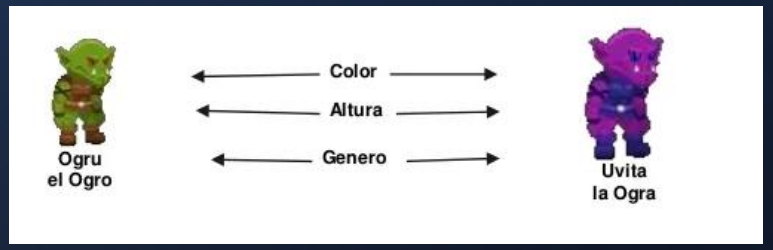
El valor del servicio de agua es la suma del valor de la tarifa básica y el valor del consumo. La tarifa básica depende del estrato, así:

Estrato	Tarifa Básica
1	10.000
2	15.000
3	20.000
4	25.000
5	30.000

El valor del consumo es igual al consumo por $100.

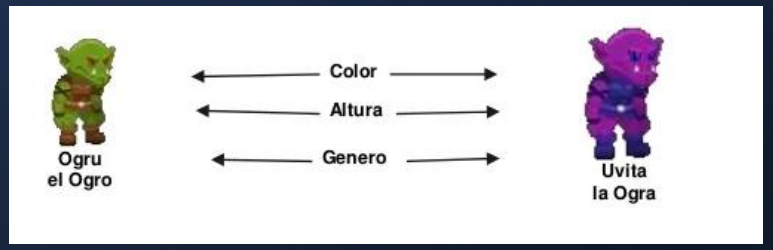
Realizar el programa en JavaScript que resuelva la situación problema presentada, utilizando programación WEB (formularios) el concepto de Clases y Objetos (POO) y Maps.

POO
ATRIBUTOS



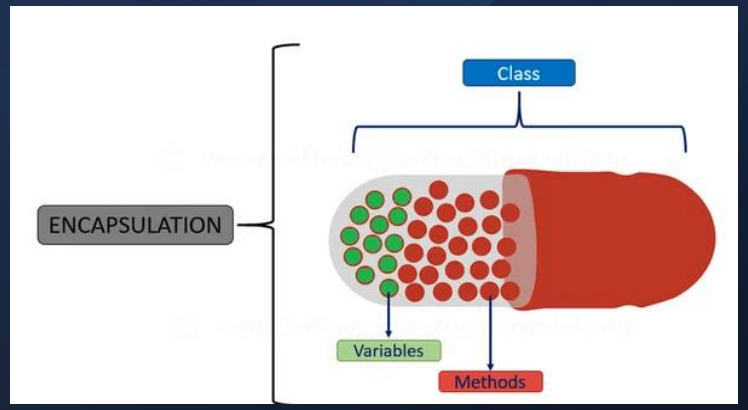
- Los objetos tienen atributos que llevan consigo cuando se usan en un programa.
- Los atributos son especificados como parte de la clase de la cual se toma el objeto.
- Los atributos existen antes que el método sea llamado en un objeto, mientras el método se esté ejecutando y después que el método completa la ejecución.
- Las clases constan de uno o más métodos que manipulan los atributos que pertenecen a un objeto particular de la clase.
- Son los datos que caracterizan a los objetos de una clase y determinan el estado de un objeto.

POO
ATRIBUTOS




```
class CuentaCorriente {
  constructor(numero, nombre, saldo=0) {
    this.numero = numero;
    this.nombre = nombre;
    this.saldo = saldo;
  }
  depositar(monto) {
    this.saldo += monto;

    return true;
  }
  girar(monto) {
    if (this.saldo >= monto) this.saldo -= monto;
    else return false;

    return true;
  }
  consultarSaldo() {
    return this.saldo;
  }
}
```

POO
ENCAPSULAMIENTO



¿Qué es?

- Proceso para ocultar los atributos y métodos de una clase

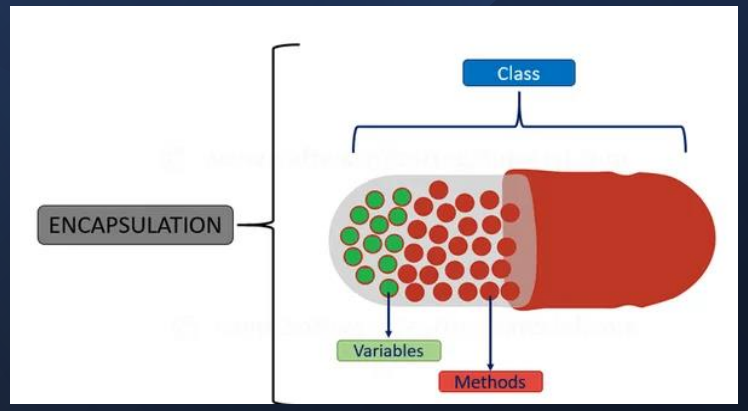
¿Por qué?

- El comportamiento y los atributos de un objeto no debe ser alterado

¿Para qué?

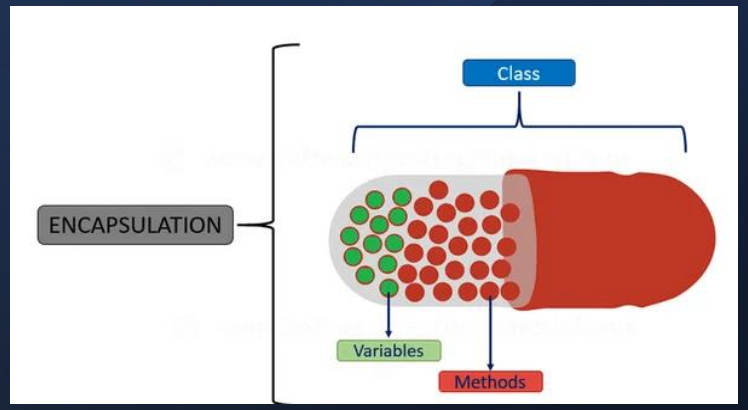
- Proteger los datos de modificaciones incontroladas

POO
ENCAPSULAMIENTO

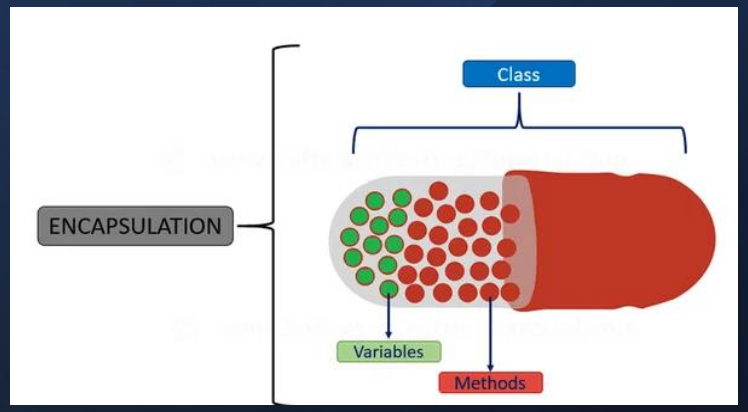


- En Java se definen métodos y atributos públicos para poder revisar su contenido e incluso ser modificados.
- El encapsulamiento consiste en ocultar los atributos y métodos de una clase, para evitar el acceso y la modificación desde otra clase.
- Para implementar el encapsulamiento se definen estos atributos y métodos con acceso privado.
- El encapsulamiento es útil al compartir las clases con otros programadores.
- Los métodos encapsulan un conjunto de instrucciones que se puede ejecutar tantas veces como sea necesario. El código del método queda oculto y sólo es necesario conocer su interfaz (parámetros y valor de retorno).

POO
ENCAPSULAMIENTO



POO
MÉTODOS SET / GET



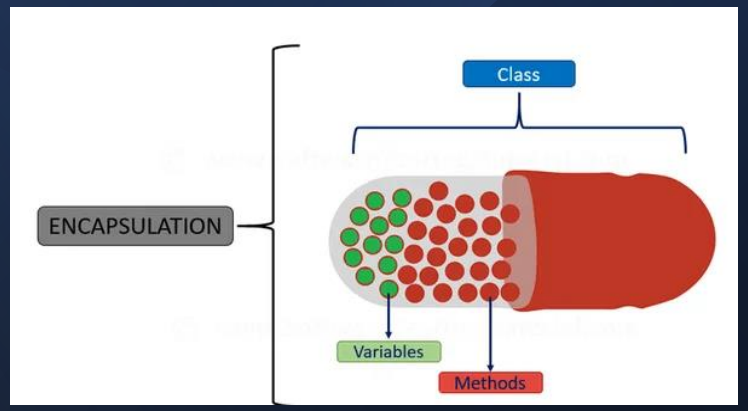
- En ocasiones es necesario dar acceso a algunos atributos de nuestra clase, sin eliminar el encapsulamiento.

- Estos métodos permiten el acceso a los atributos de una clase, que están encapsulados como privados.

- Con el **método set** se establecen o asignan valores a los atributos encapsulados en una clase. Este recibe un parámetro de entrada con el cual se establece el valor al atributo encapsulado y no retorna nada.

- Con el **método get** se obtiene o retorna el valor de un atributo encapsulado. El nombre del método se define con el tipo de valor que se desea retornar. No recibe parámetros.

POO
MÉTODOS SET / GET



CuentaCorriente

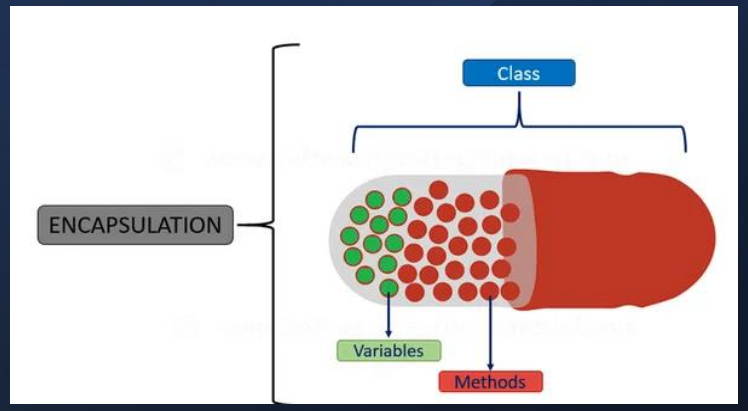
- numero:int
- nombre:string
- saldo:float

+ depositar(monto:int):boolean
+ girar(monto:int):boolean
+ consultarSaldo():int

POO
MÉTODOS SET / GET

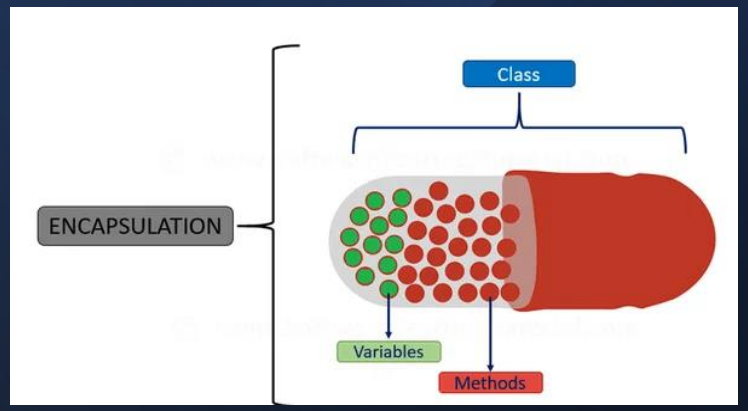
SINTAXIS

```
get propiedadAtributo() {
    return this.propiedadAtributo;
}

set propiedadAtributo(param) {
    this.propiedadAtributo = param;
}
```



POO
MÉTODOS SET / GET

EJEMPLO



CuentaCorriente

- numero:int
- nombre:string
- saldo:float

+ depositar(monto:int):boolean
+ girar(monto:int):boolean
+ consultarSaldo():int

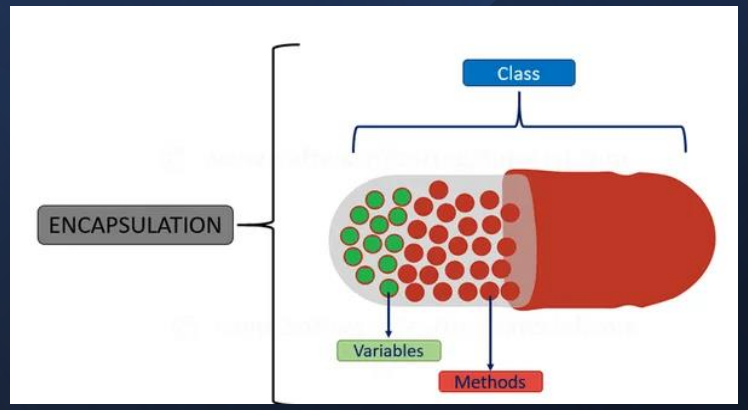
+ set numero(nuevoNumero:int):void
+ get numero():int

+ set nombre(nuevoNombre:string):void
+ get nombre(): string

+ set saldo(nuevoSaldo:float):void
+ get saldo():float

POO
MÉTODOS SET / GET

EJEMPLO




```
class CuentaCorriente {
  constructor(numero, nombre, saldo = 0) {
    this.numero = numero;
    this.nombre = nombre;
    this.saldo = saldo;
  }
  set numero(nuevoNmero) {
    this.numero = nuevoNumero;
  }
  get numero() {
    return this.numero;
  }
  set nombre(nuevoNombre) {
    this.nombre = nuevoNombre;
  }
  get nombre() {
    return this.nombre;
  }
  set saldo(nuevoSaldo) {
    this.numero = nuevoSaldo;
  }
  get saldo() {
    return this.saldo;
  }
}
```

POO
MÉTODOS / ATRIBUTOS ESTÁTICOS

En JavaScript, los métodos y atributos **estáticos** son **miembros de una clase** que **son compartidos** por todas las instancias de esa clase y **no dependen de ninguna** instancia en particular. Esto significa que **se pueden acceder directamente** desde la clase en sí, **sin necesidad de crear una instancia** de la misma.

POO
MÉTODOS / ATRIBUTOS
ESTÁTICOS

Los métodos estáticos se definen usando la palabra clave **static** antes del nombre del método.

POO
MÉTODOS / ATRIBUTOS
ESTÁTICOS

Ya hemos usado métodos y atributos estáticos.

Por ejemplo:

```
console.log(Math.random() * 10 + 1);
console.log(Math.PI);
console.log(Math.floor(Math.random() * 10
+ 1));

console.log(Number.MAX_SAFE_INTEGER);
console.log(Number.EPSILON);
```

POO
MÉTODOS / ATRIBUTOS
ESTÁTICOS

Por ejemplo, si tenemos una clase MiMath en JavaScript, podemos definir un método estático cuadrado que devuelve el cuadrado de un número:

```
class MiMath {
    static cuadrado(number) {
      return number * number;
    }
}

MiMath.cuadrado(5);
```

POO
MÉTODOS / ATRIBUTOS
ESTÁTICOS

Por ejemplo, si quisiéramos saber cuántas cuentas corrientes se han creado.

En UML se representa como un atributo subrayado

POO
MÉTODOS / ATRIBUTOS
ESTÁTICOS

CuentaCorriente

- numero:int
- nombre:string
- saldo:float

+ cantidad:int

+ depositar(monto:int):boolean
+ girar(monto:int):boolean
+ consultarSaldo():int
+ set numero(nuevoNumero:int):void
+ get numero():int
+ set nombre(nuevoNombre:string):void
+ get nombre(): string
+ set saldo(nuevoSaldo:float):void
+ get saldo():float

POO
MÉTODOS / ATRIBUTOS ESTÁTICOS

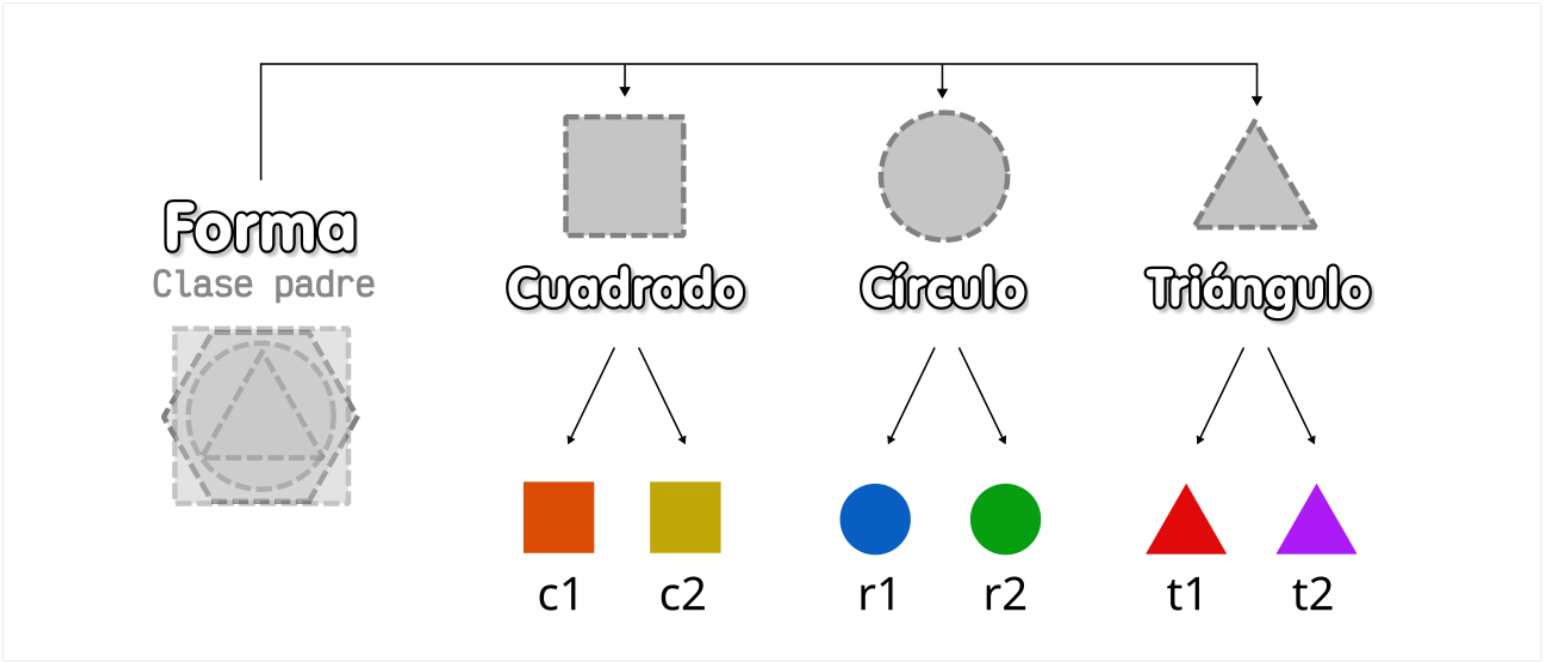
Por ejemplo, si quisiéramos saber cuántas cuentas corrientes se han creado.

```
class CuentaCorriente {
  static cantidad = 0;

  constructor(numero, nombre, saldo = 0) {
    this._numero = numero;
    this._nombre = nombre;
    this._saldo = saldo;

    ++CuentaCorriente.cantidad;
  }
  // ....
}

const ctaCorrA = new CuentaCorriente(1234, "Juan");
const ctaCorrB = new CuentaCorriente(9876, "Maria", 450600);

console.log(CuentaCorriente.cantidad); // 2
```

POO

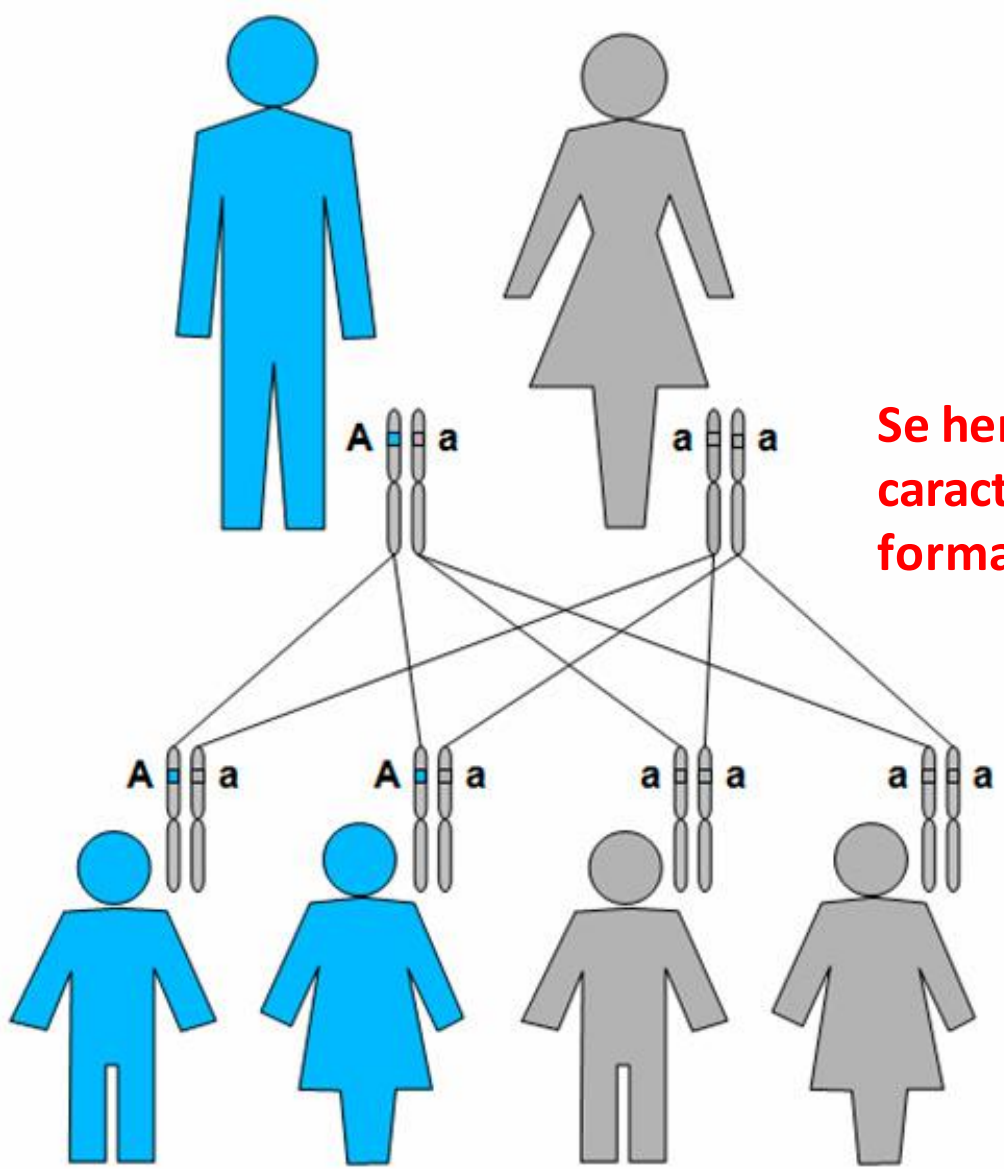
HERENCIA

La **herencia** es un pilar importante de POO (Programación Orientada a Objetos). Es el mecanismo en JavaScript por el cual una clase permite heredar las características (atributos y métodos) a otras clases.



POO

HERENCIA



Se heredan características físicas, formas de ser o actuar

POO

HERENCIA

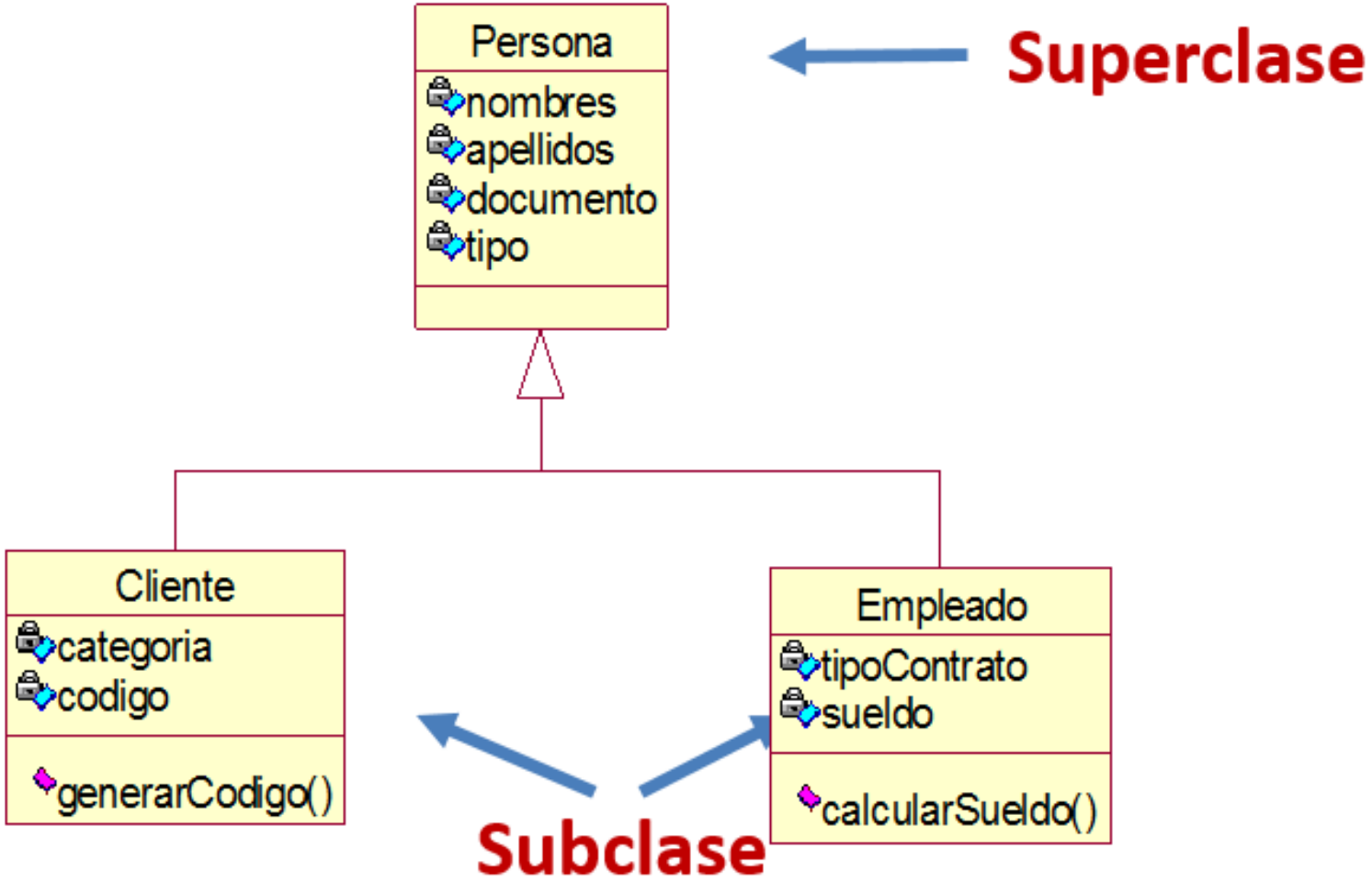
Terminología importante:

- **Superclase**: la clase cuyas atributos y métodos se heredan se conoce como superclase (o una clase base o una clase principal).

- **Subclase**: la clase que hereda la otra clase se conoce como subclase (o una clase derivada, clase extendida o clase hija). La subclase puede agregar sus propios atributos y métodos además de los atributos y métodos de la superclase.

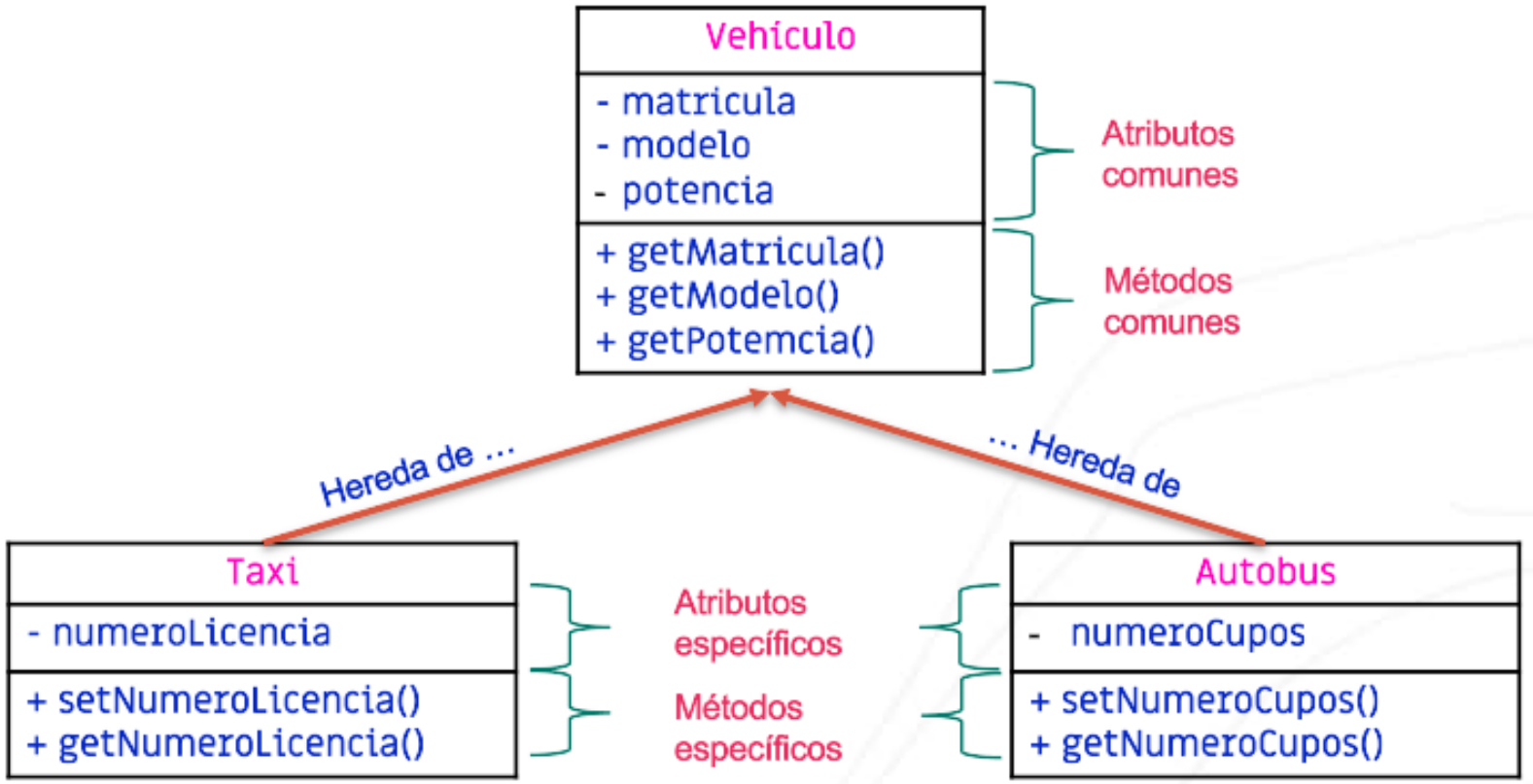
POO

HERENCIA



POO

HERENCIA



POO

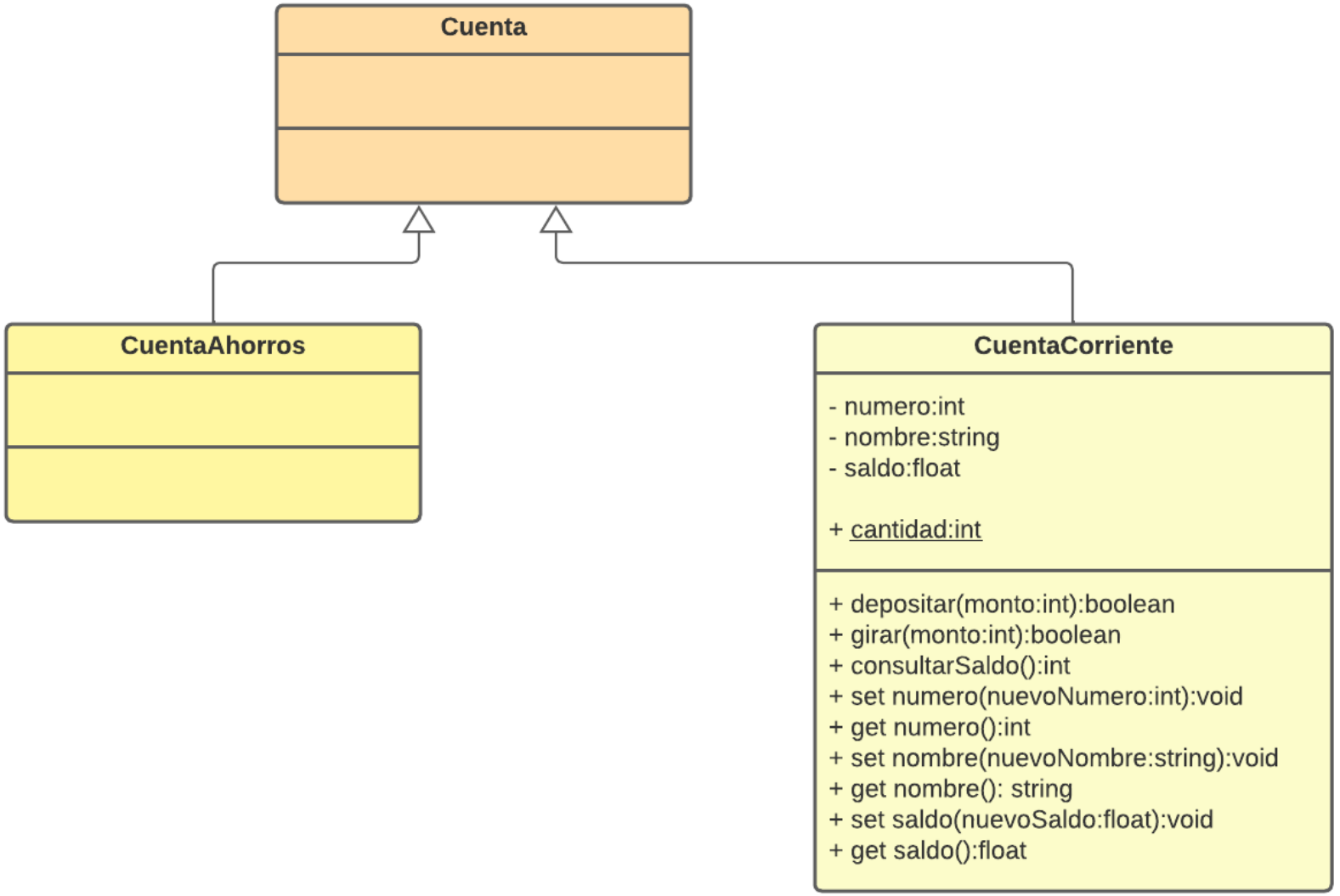
HERENCIA

Ejemplo

Rediseñe la clase cuenta corriente para que herede de una clase cuenta y tenga una clase hermana cuenta de ahorros.

POO

HERENCIA



POO

EJERCICIOS

Ejercicios:

- **Ejercicio 2**: descargar aquí.
- **Ejercicio 3**: descargar aquí.

POO

CLASES ABSTRACTAS

DEFINICIÓN

Una **clase abstracta** no es más que una **clase** común la cual posee atributos, métodos, constructores y por lo menos un método abstracto. Una **clase abstracta** no puede ser instanciada, solo heredada.

POO

CLASES ABSTRACTAS

PROPÓSITO

Una clase abstracta sirve c**omo una plantilla** base para otras clases relacionadas. Proporciona una implementación parcial o completa de una clase, pero no se puede instanciar directamente.

POO

CLASES ABSTRACTAS

CARACTERÍSTICAS

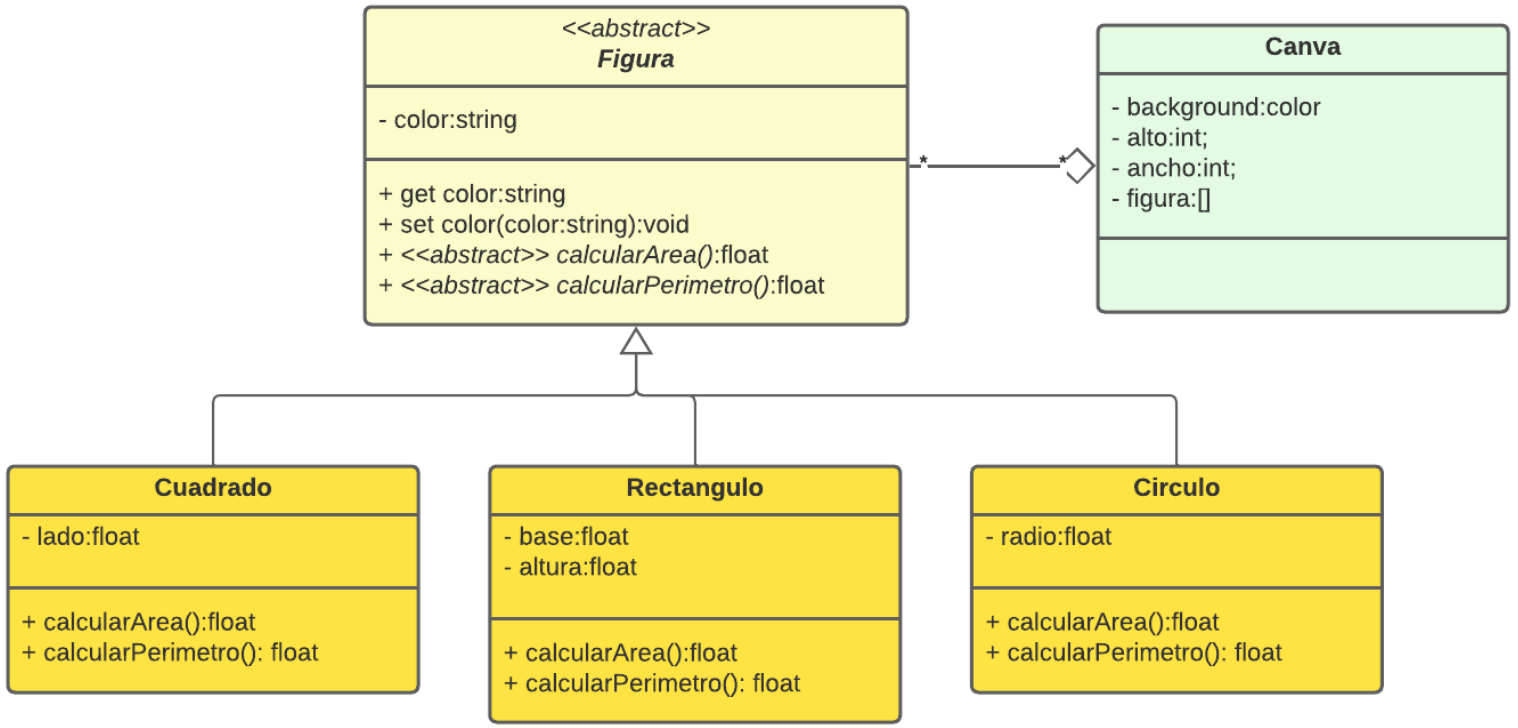
- ✓ Puede contener métodos abstractos (sin implementación).
- ✓ Puede tener propiedades, campos y constructores.
- ✓ Las clases derivadas extienden la clase abstracta y deben proporcionar implementaciones para los métodos abstractos.
- ✓ Una clase abstracta puede heredar de otra clase abstracta o de una clase concreta.

POO

CLASES ABSTRACTAS

REPRESENTACIÓN UML

Las clases y métodos se denotan en cursiva o la palabra ***abstract.***



POO

CLASES ABSTRACTAS

IMPLEMENTACIÓN JAVASCRIPT

En JavaScript ES6, no hay una sintaxis específica para declarar clases abstractas o interfaces como en otros lenguajes de programación orientados a objetos. Sin embargo, puedes utilizar ciertos patrones y convenciones para lograr comportamientos similares.

POO

CLASES ABSTRACTAS

IMPLEMENTACIÓN JAVASCRIPT

```
class AbstractClass {
  constructor() {
    if (new.target === AbstractClass) {
      throw new Error("No se puede instanciar.");
    }
  }

  abstractMethod() {
    throw new Error("Método no implementado");
  }

  normalMethod() {
    console.log("Método normal");
  }
}
```

POO

CLASES ABSTRACTAS

IMPLEMENTACIÓN JAVASCRIPT

```
class ConcreteClass extends AbstractClass {
  abstractMethod() {
    console.log("Implementación método
abstracto");
  }
}
```

```
const instancia= new ConcreteClass();
instancia.abstractMethod();
instancia.normalMethod();

// Error por que al intentar instanciar una clase abstracta
const abstractInstancia = new AbstractClass();
```

POO

INTERFACES

POO

INTERFACES

DEFINICIÓN

Es una colección de ***métodos abstractos*** y propiedades constantes. En las **interfaces** se especifica qué se debe hacer pero no su implementación. Serán las clases que implementen estas **interfaces** las que describen la lógica del comportamiento de los métodos.

POO

INTERFACES

PROPÓSITO

Una interfaz define **un contrato o conjunto de métodos** que una clase concreta **debe implementar**. Sirve para establecer una especificación común y asegurar que las clases que la implementen cumplan con dicha especificación.

POO

INTERFACES

CARACTERÍSTICAS

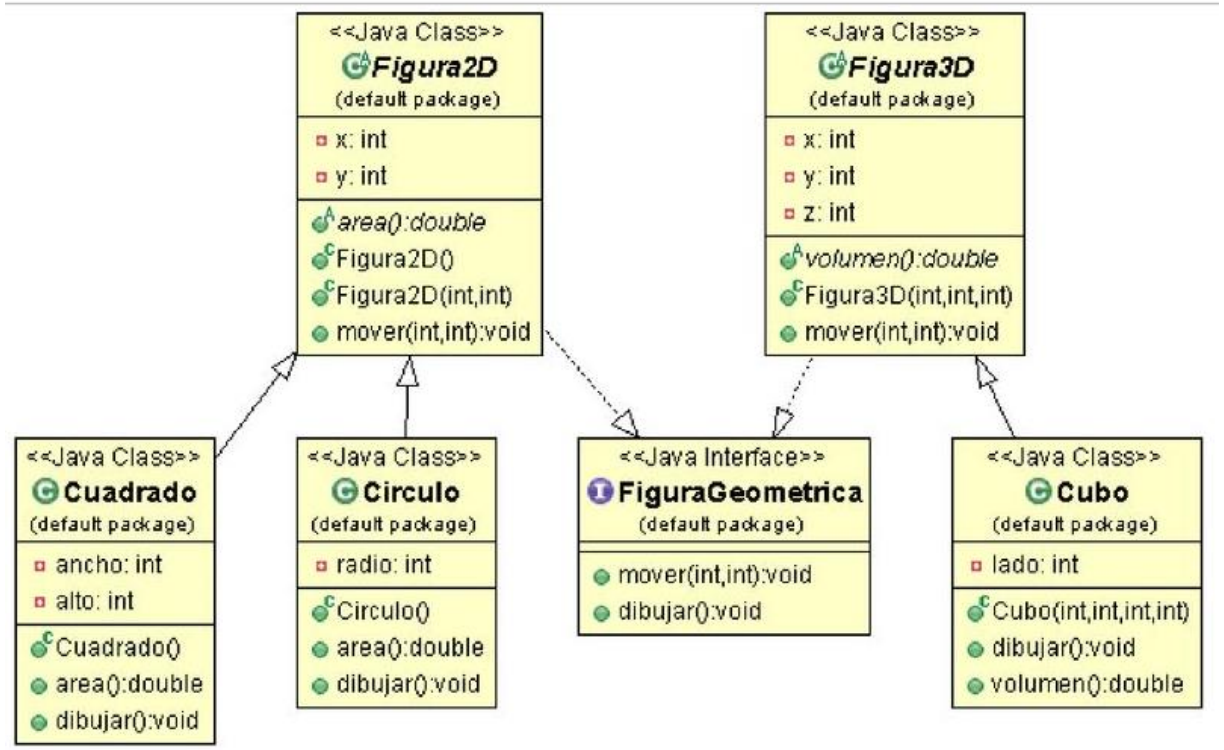
- ✓ Solo contiene métodos abstractos y no tiene implementación de ningún método.
- ✓ No puede tener campos, propiedades o constructores.
- ✓ No tiene estado ni comportamiento propio.
- ✓ Sirve como una abstracción pura de un conjunto de comportamientos requeridos.

POO

INTERFACES

REPRESENTACIÓN UML

Las clases y métodos se denotan en cursiva o la palabra ***<<interface>>.***



POO

INTERFACES

IMPLEMENTACIÓN JAVASCRIPT

En JavaScript ES6, no hay una sintaxis específica para declarar clases abstractas o interfaces como en otros lenguajes de programación orientados a objetos. Sin embargo, puedes utilizar ciertos patrones y convenciones para lograr comportamientos similares.

POO

INTERFACES

IMPLEMENTACIÓN JAVASCRIPT

```
class MyInterface {
  metodo1() {
    throw new Error("metodo1 debe ser
implementado");
  }


  metodo2() {
    throw new Error("metodo2 debe ser
implementado");
  }
}
```

POO

INTERFACES

IMPLEMENTACIÓN JAVASCRIPT

```
class MyClassA extends MyInterface {
  metodo1() {
    console.log("Implementación del metodo1 en
Clase A");
  }


  metodo2() {
    console.log("Implementación del metodo2 en
Clase A");
  }
}
```

POO

INTERFACES

IMPLEMENTACIÓN JAVASCRIPT

```
class MyClassB extends MyInterface {
  metodo1() {
    console.log("Implementación del metodo1
en Clase B");
  }
}
```

POO

INTERFACES

IMPLEMENTACIÓN JAVASCRIPT

```
const instanceA = new MyClassA();
// Salida: "Implementación del metodo1 en Clase A"
instanceA.metodo1();
// Salida: "Implementación del metodo2 en Clase A"
instanceA.metodo2();


const instanceB = new MyClassB();
// Salida: "Implementación del method1 en Clase B
instanceB.metodo1();
// Lanza un error, el metodo 2 no está implementado
en claseB
instanceB.metodo2();
```

EXPRESIONES REGULARES

IMPLEMENTACIÓN JAVASCRIPT

Las expresiones regulares, también conocidas como regex, son **patrones** de búsqueda utilizados para encontrar y manipular texto de manera eficiente. Son secuencias de caracteres que definen un conjunto de reglas de búsqueda.

EXPRESIONES REGULARES

IMPLEMENTACIÓN JAVASCRIPT

En JavaScript ES6, puedes trabajar con expresiones regulares utilizando el objeto **RegExp** y sus métodos.

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

TUTORIALES EN LINEA

- ✓ RegexOne - Learn Regular Expressions - Lesson 1: An Introduction, and the ABCs
- ✓ regex101: build, test, and debug regex
- ✓ Regex Learn - Paso a paso, de cero a avanzado.
- ✓ RegExr: Learn, Build, & Test RegEx

EXPRESIONES REGULARES

SINTAXIS LITERAL

IMPLEMENTACIÓN JAVASCRIPT

```
const regex = /patrón/;
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

```
const regex = new RegExp("patrón");
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

test(): Comprueba si una cadena coincide con la expresión regular y devuelve true o false.

```
const regex = /hello/;
const text = "hello world";
console.log(regex.test(text));
// Output: true
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

exec(): Busca una coincidencia en una cadena y devuelve un objeto de coincidencia que contiene información sobre la coincidencia. Si no encuentra coincidencias, devuelve null.

```
const regex = /hello/;
const text = "hello world";
const match = regex.exec(text);
console.log(match[0]);
// Output: "hello"
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

match(): Busca todas las coincidencias en una cadena y devuelve un array con todas las coincidencias encontradas.

```
const regex = /lo/g;
const text = "hello world";
const matches = text.match(regex);
console.log(matches);
// Output: ["lo"]
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

replace(): Reemplaza una coincidencia en una cadena con un nuevo texto.

```
const regex = /world/;
const text = "hello world";
const newText = text.replace(regex,
"planet");
console.log(newText);
// Output: "hello planet"
```

EXPRESIONES REGULARES

UTILIZANDO EL CONSTRUCTOR REGEXP

IMPLEMENTACIÓN JAVASCRIPT

split(): Divide una cadena en un array de subcadenas utilizando una expresión regular como separador.

```
const regex = /[, ]/;
const text = "apple, banana,
cherry";
const parts = text.split(regex);
console.log(parts);
// Output: ["apple", "banana",
"cherry"]
```