sustantiva

CON•SENTIDO

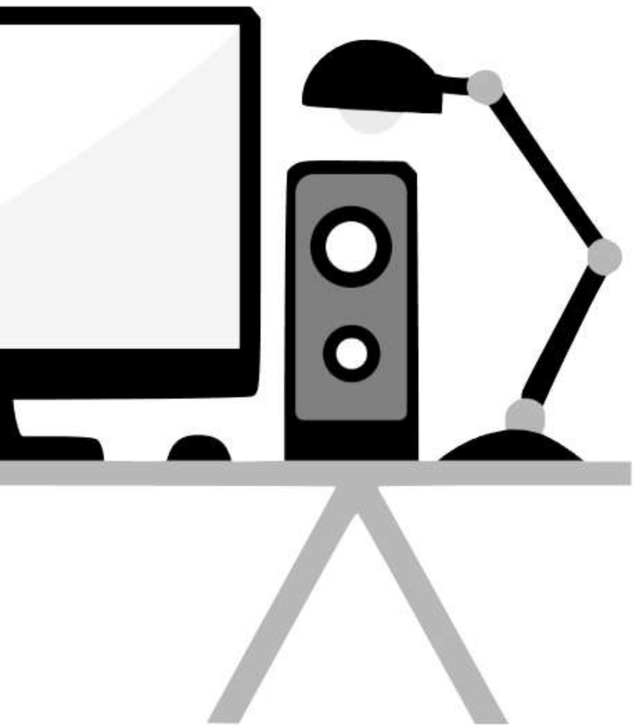
ORIENTACIÓN A OBJETOS EN JS

Lección 01

Competencia General

Utilizar los conceptos fundamentales de la programación orientada a objetos acorde al lenguaje JavaScript para resolver un problema simple

Contenidos



Conceptos POO

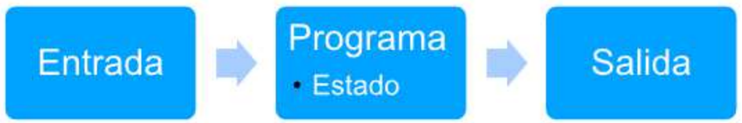
1. Paradigmas de programación
2. Abstracción y programación orientada a objetos
3. Clases, objetos e instancias
4. Elementos de una clase
5. Pilares de la POO
6. Creación de objetos literales
7. Creación de objetos mediante constructor
8. La propiedad prototype para definir métodos

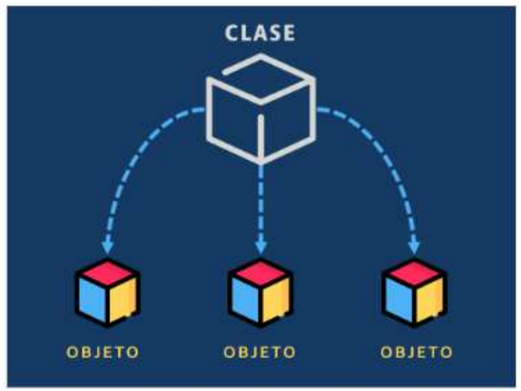
Notación de objetos en JS

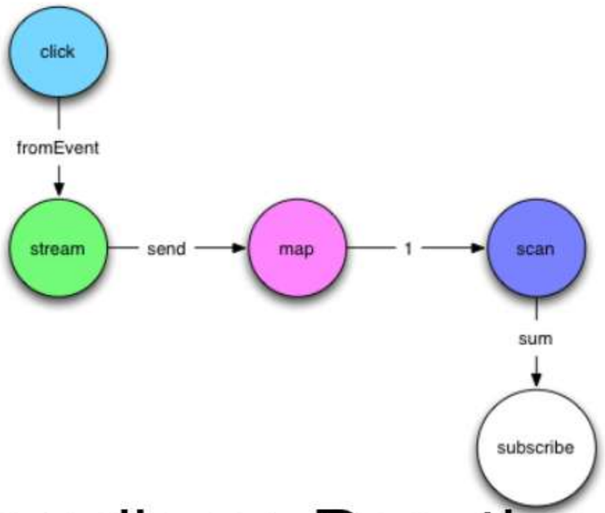
1. Qué es un JSON
2. Estructura de un JSON
3. Arreglos como JSON
4. Herramientas online para construir y validar estructuras JSON

Paradigmas de programación

Los paradigmas de programación son estilos para resolver un problema dado. Existen diferentes formas de diseñar un lenguaje de programación y varios modos de obtener una solución a un problema dado. Sin embargo, cuando aplicamos un paradigma estamos usando una serie de métodos sistemáticos para resolver dicho problema.







Paradigma Imperativo

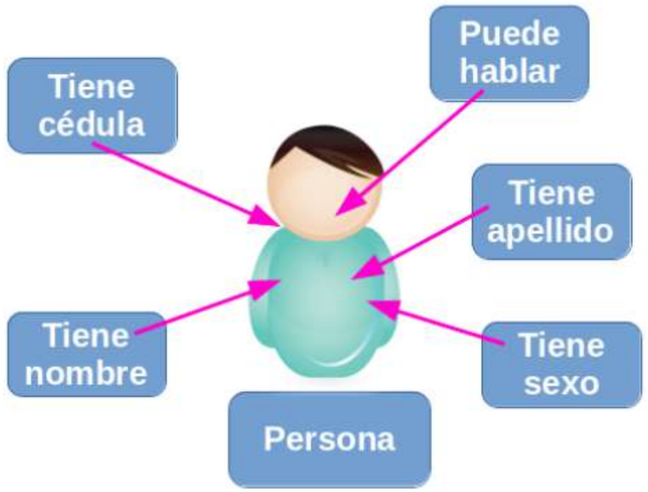
POO

Paradigma Reactivo

Abstracción y POO

Una abstracción se enfoca en la visión externa de un objeto, separa el comportamiento especifico del objeto, esta división se conoce como la barrera de abstracción, la cual se consigue aplicando el principio de mínimo compromiso

El principio de mínimo compromiso es el proceso por el cuál la interfaz de un objeto muestra su comportamiento específico y nada más.

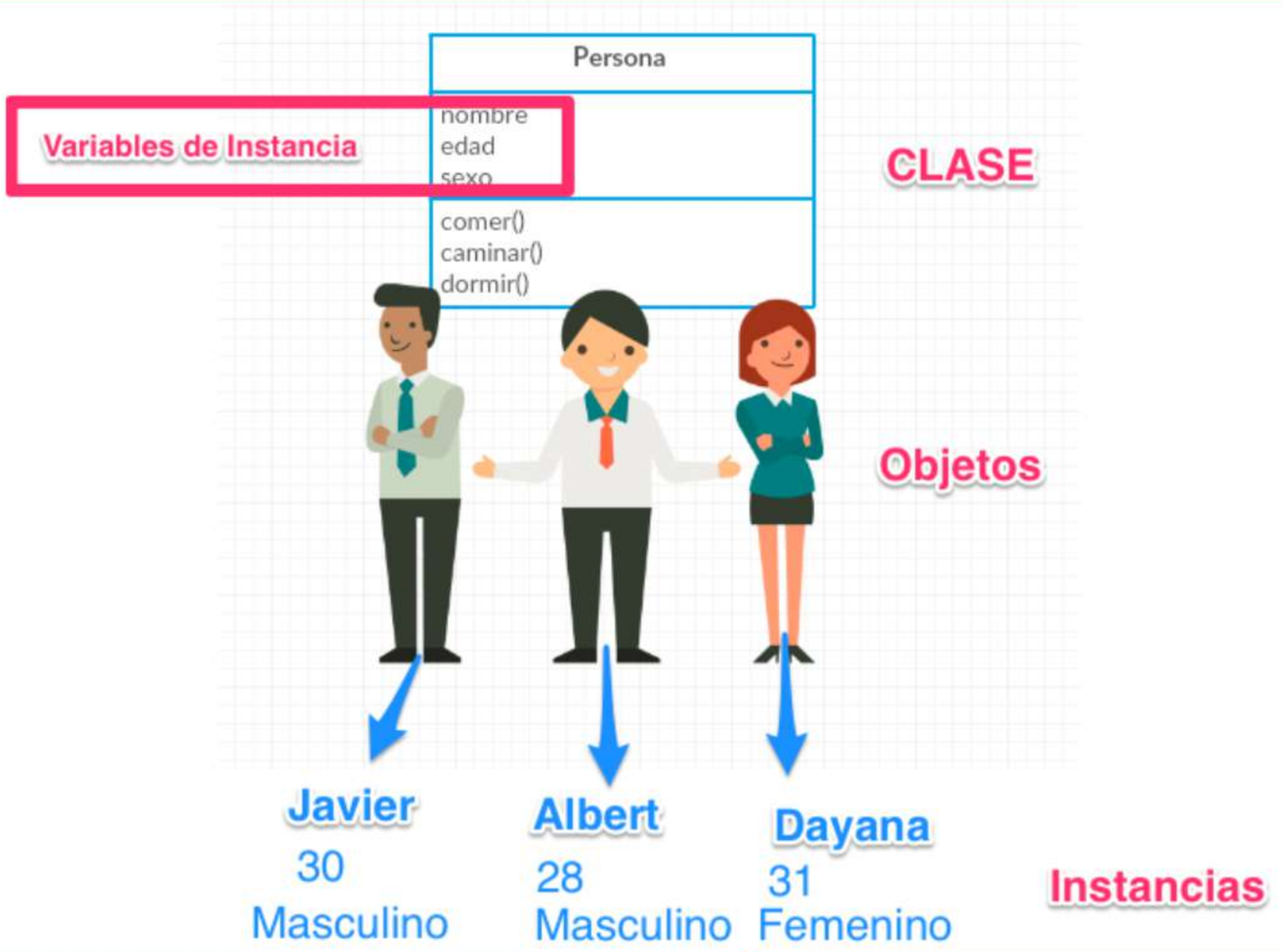


Clase, objeto e instancia

Una clase contiene las indicaciones de como generar un objeto de un tipo específico, cada objeto generado luego puede tener sus propios valores.

El objeto es lo que se construye por medio de la clase.

La instancia es una forma de llamar a dicho objeto creado.



Elementos de una clase

Para definir una clase usamos la palabra reservada ***class*** seguido de un nombre que identifique a dicha clase. También debemos definir un constructor, este es el encargado de generar el objeto con las propiedades definidas en los parámetros de entrada.

Por ejemplo la clase Rectángulo contiene un constructor que recibe por parámetro un alto y un ancho.

```
//clase rectangulo
class Rectangulo {
  constructor(alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
}
```

Creación de objetos mediante constructor

Para crear un objeto mediante constructor debemos primero generar la clase, luego, instanciar al objeto con los parámetros de entrada del constructor.

```
class Rectangulo {
  constructor (alto, ancho) {
    this.alto = alto;
    this.ancho = ancho;
  }
  get area() {
     return this.calcArea();
   }
  calcArea () {
    return this.alto * this.ancho;
  }
}

const cuadrado = new Rectangulo(10, 10);

console.log(cuadrado.area); // 100
```

Propiedades y métodos de una clase

Propiedades: datos/atributos de cada objeto, generalmente asignados en el constructor.

Métodos: funciones definidas en la clase que operan sobre esas propiedades (incluye get/set y métodos estáticos).

```
class Rectangulo {
  constructor(alto, ancho) {       // ← Propiedades
    this.alto  = alto;             //    (atributos de instancia)
    this.ancho = ancho;
  }
  area() {                         // ← Método de instancia
    return this.alto * this.ancho;
  }
  get perimetro() {                // ← Getter (método accesor)
    return 2 * (this.alto + this.ancho);
  }
  static cuadrado(lado) {          // ← Método estático
                                   (no requiere instancia)
    return new Rectangulo(lado, lado);
  }
}
```

Instanciando múltiples objetos a partir de una clase

Cuando definimos una clase como Rectangulo, podemos crear varios objetos distintos (instancias) que comparten la misma estructura y comportamiento. Esto es útil para representar múltiples entidades similares sin duplicar código.

Cada instancia (r1, r2, r3) es un objeto distinto con sus propios valores, pero todos usan el mismo "molde" definido por la clase Rectangulo. Esta capacidad es una de las principales ventajas de la programación orientada a objetos.

```
const r1 = new Rectangulo(5, 10);
const r2 = new Rectangulo(3, 7);
const r3 = new Rectangulo(8, 4);

console.log(r1.area); // 50
console.log(r2.area); // 21
console.log(r3.area); // 32
```

Propiedad prototipo

Cuando instanciamos un nuevo objeto con la palabra reservada ***new,*** se crea una nueva propiedad llamada ***protoype.*** La propiedad prototype es un objeto que contiene al constructor de la clase, los objetos creados heredan todas las propiedades de la clase mediante el objeto prototype.

La clase rectángulo creada en la diapositiva anterior es similar a crear una función y asignar un objeto prototype

```
function Rectangle(height, width) {
  this.height = height
  this.width = width
}

Rectangle.prototype.calcArea = function calcArea() {
  return this.height * this.width
}
```

Creación de objetos literales

Un objeto literal hace referencia a instanciar un objeto y definir sus propiedades y atributos en la misma instancia.

```
//Objeto literal
let perro = {
   nombre:"Scott",
   color:"Cafe",
   edad: 5,
   macho: true
};
```

Clases vs. Objetos Literales

La principal diferencia es que el **objeto literal** se crea de forma única con sus propiedades y métodos definidos directamente, mientras que una **clase actúa como un molde** para crear múltiples objetos con la misma estructura. Esto hace que las clases sean mucho más reutilizables, escalables y propias del paradigma orientado a objetos. Por ejemplo, podríamos crear muchos perros distintos (como miPerro, tuPerro, suPerro) usando la clase Perro, mientras que el objeto literal perro es uno solo y no puede reutilizarse como plantilla.

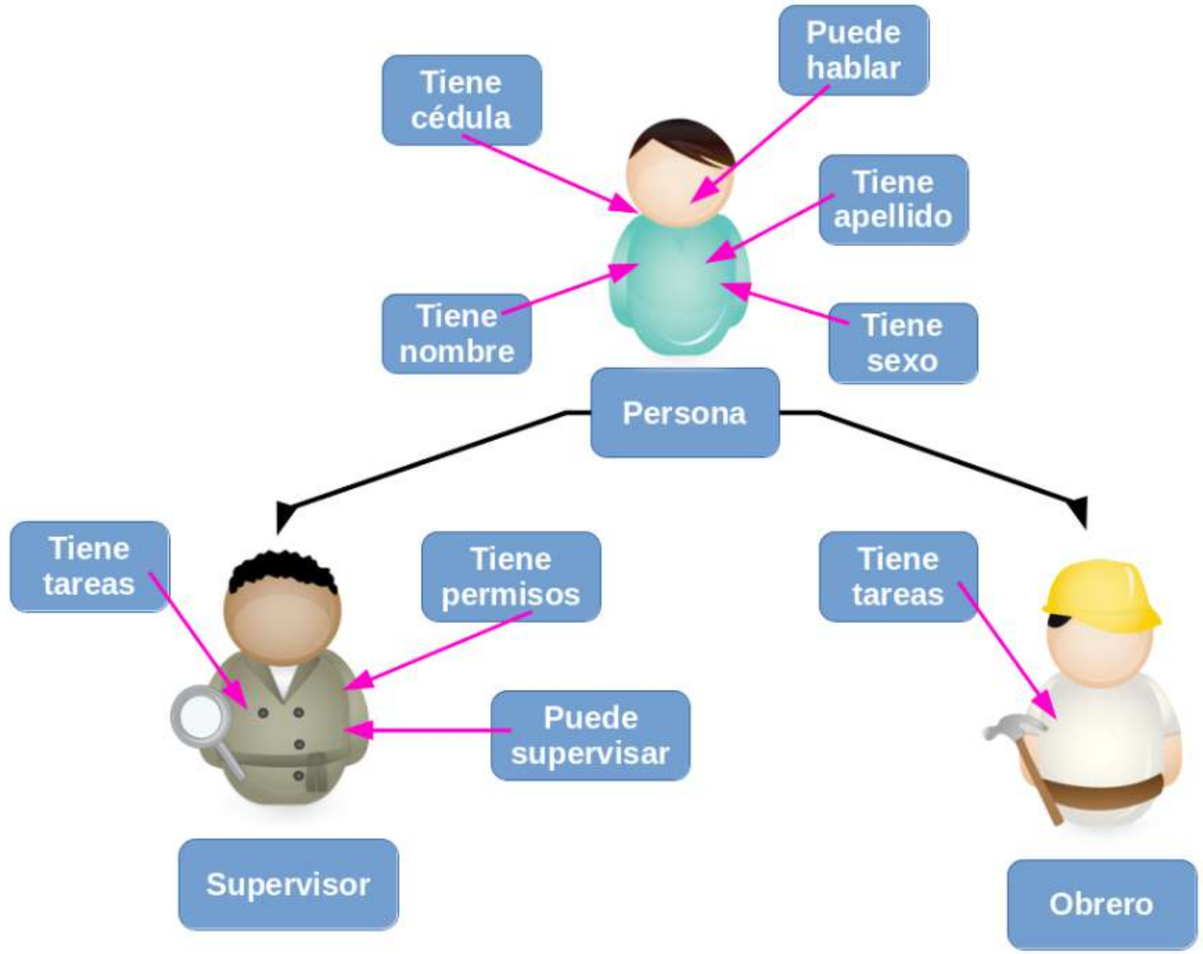
```javascript
// Objeto literal
const perro = {
  nombre: "Firulais",
  ladrar: function() {
    console.log("¡Guau!");
  }
};
```

```javascript
// Clase e instancia
class Perro {
  constructor(nombre) {
    this.nombre = nombre;
  }
  ladrar() {
    console.log("¡Guau!");
  }
}

const miPerro = new Perro("Rocky");
```

Pilares de la POO

La herencia es un mecanismo que permite derivar una clase a otra clase más específica.

Tenemos la **clase "Padre"** que deriva en las **clases "Hijas"** Supervisor y Obrero



Pilares de la POO

Por ejemplo, tenemos la clase Mamifero, que actúa como **clase padre** de Perro. Esto significa que Perro **hereda** automáticamente todas las propiedades y métodos definidos en Mamifero, como por ejemplo respirar().

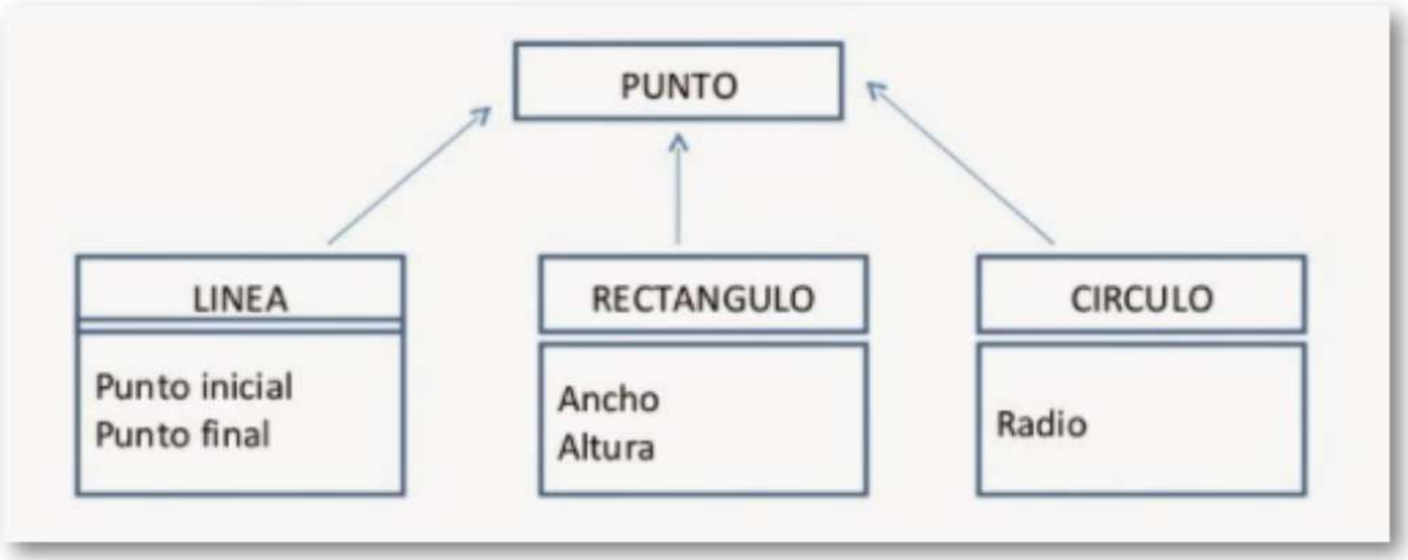
Al utilizar la palabra clave extends, indicamos que Perro es una **especialización** de Mamifero. Gracias a esto, una instancia de Perro no solo puede realizar acciones propias, como ladrar(), sino también acciones generales que comparten todos los mamíferos, como respirar().

```
// Clase padre
class Mamifero {
  constructor(nombre) {
    this.nombre = nombre;
    this.tienePelo = true;
    this.esCalido = true;
    this.alimentaConLeche = true;
  }

  respirar() {
    console.log(`${this.nombre} está respirando...`);
  }
}
```

```
// Clase hija que hereda de Mamifero
class Perro extends Mamifero {
  ladrar() {
    console.log("¡Guau!");
  }
}

// Crear una instancia
const miPerro = new Perro("Rocky");
miPerro.respirar(); // Rocky está respirando...
miPerro.ladrar();   // ¡Guau!
```

Pilares de la POO

Polimorfismo es la capacidad que tienen los objetos de responder al mismo mensaje o evento en función de los parámetros de entrada.



Pilares de la POO

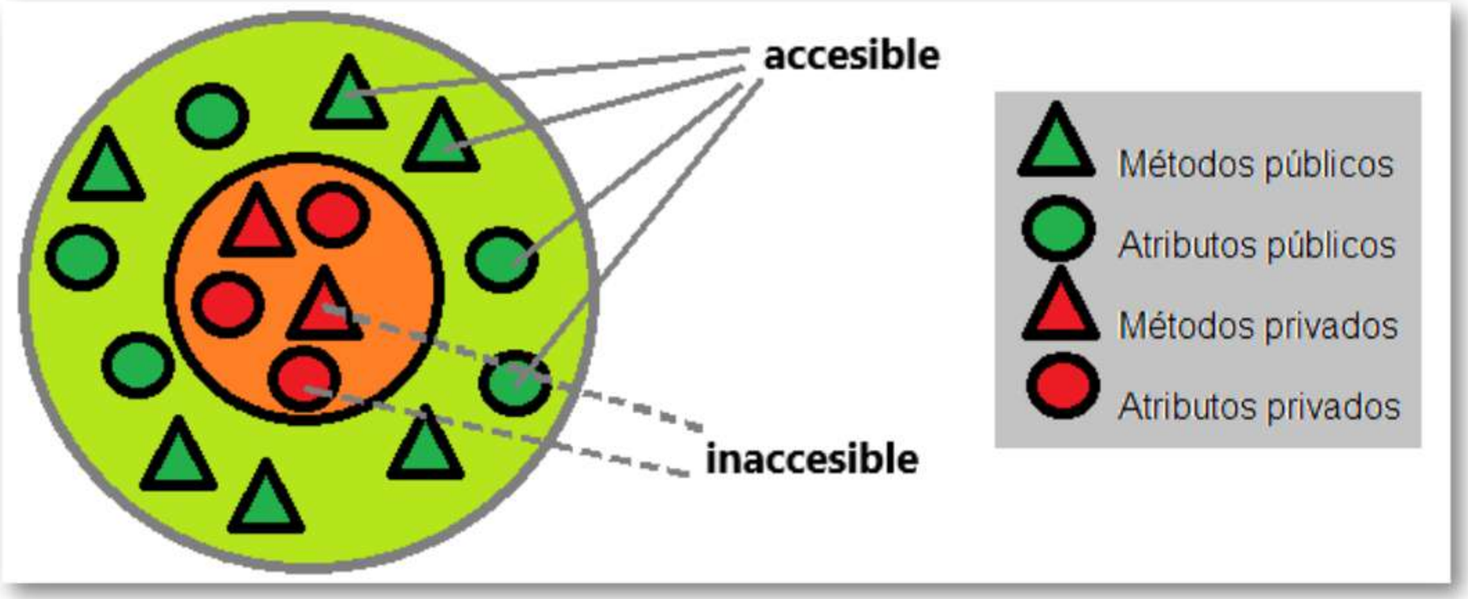
En este ejemplo, las clases Perro y Gato heredan de una clase base llamada Mamifero, la cual define un método genérico llamado hacerSonido(). Gracias al polimorfismo, cada subclase puede sobrescribir este método para implementar un comportamiento específico, redefiniendo su funcionalidad sin alterar la estructura de la clase base. Al recorrer un arreglo de instancias heterogéneas (Perro, Gato y Mamifero) e invocar hacerSonido(), JavaScript resuelve dinámicamente cuál versión del método ejecutar según el tipo de objeto en tiempo de ejecución. Esto permite tratar a todos los objetos como instancias del tipo base, mientras cada uno mantiene su comportamiento particular, promoviendo así la extensibilidad y el desacoplamiento del código.

```
Rocky dice: ¡Guau!
Mish dice: ¡Miau!
Criatura misteriosa hace un sonido genérico.
```

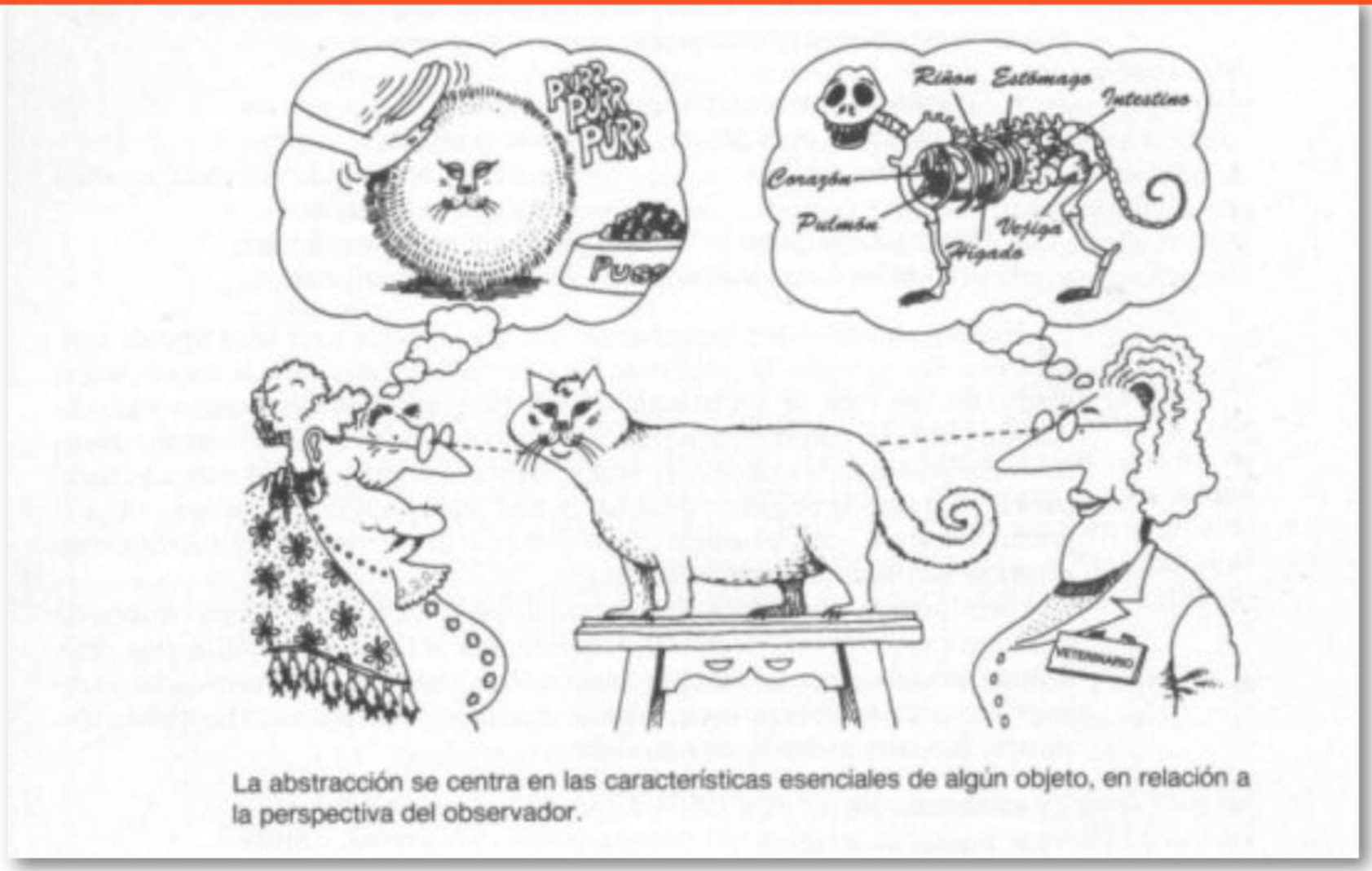
```javascript
// Clase base
class Mamifero {
  constructor(nombre) {
    this.nombre = nombre;
  }

  hacerSonido() {
    console.log(`${this.nombre} hace un sonido genérico.`);
  }
}

// Clase hija 1
class Perro extends Mamifero {
  hacerSonido() {
    console.log(`${this.nombre} dice: ¡Guau!`);
  }
}

// Clase hija 2
class Gato extends Mamifero {
  hacerSonido() {
    console.log(`${this.nombre} dice: ¡Miau!`);
  }
}

// Ejemplo de polimorfismo
const mamiferos = [
  new Perro("Rocky"),
  new Gato("Mish"),
  new Mamifero("Criatura misteriosa")
];

mamiferos.forEach(m => m.hacerSonido());
```

Pilares de la POO

Encapsulamiento es el proceso de almacenar en una misma sección los elementos de una abstracción que constituyen su estructura y su comportamiento; sirve para separar el interfaz contractual de una abstracción y su implantación.



Pilares de la POO



La abstracción se centra en las características esenciales de algún objeto, en relación a la perspectiva del observador.

Qué es un JSON

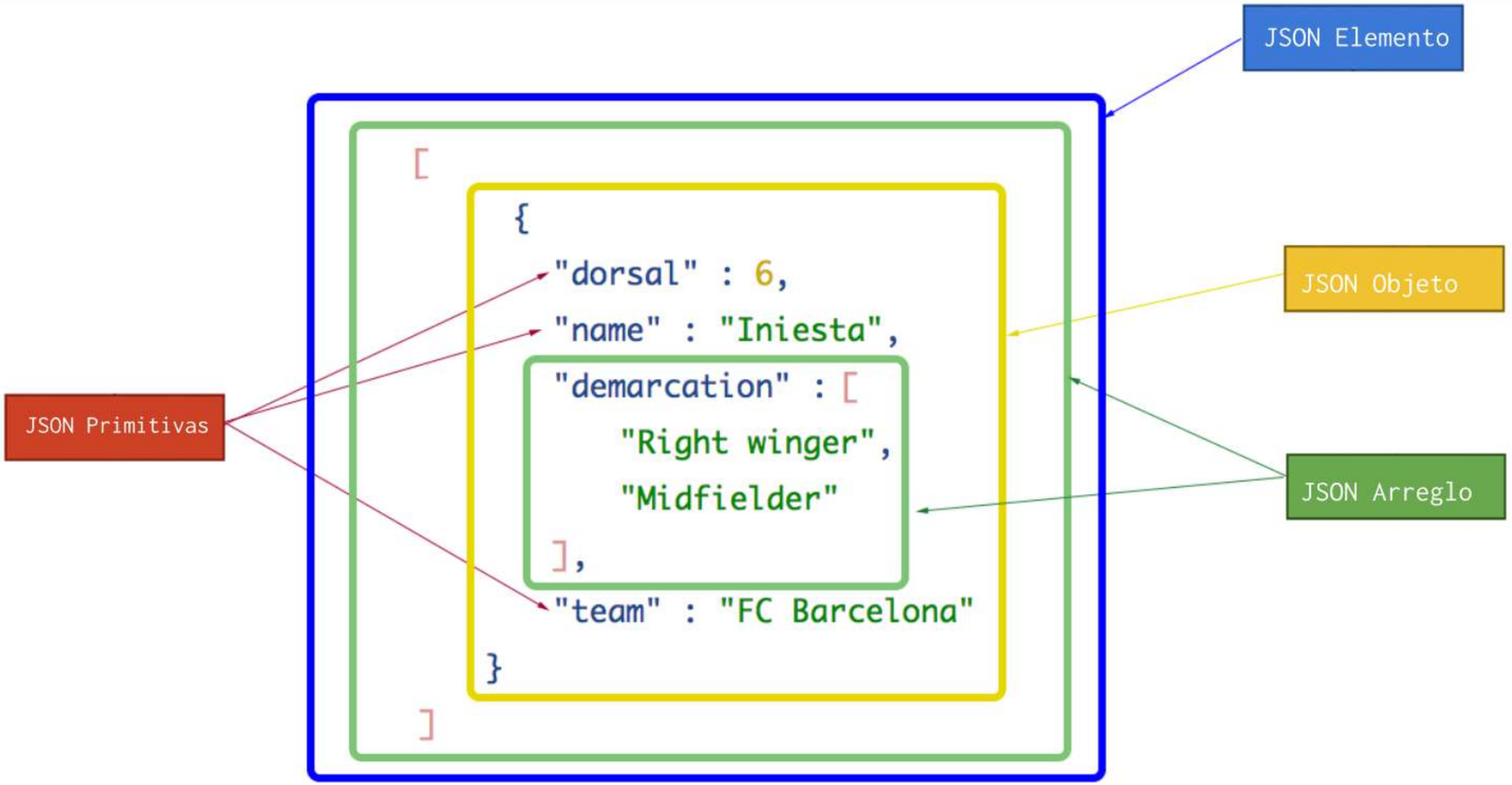
JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, que resulta fácil de leer y escribir.

Un JSON es independiente del lenguaje de programación, pero utiliza convenciones conocidas.

Un JSON define 6 tipos de valores: null, number, string, bool, matrix, object.

Una de las características principales de JSON es que al ser un formato independiente del lenguaje, permite que distintos servicios puedan comunicarse entre ellos sin necesidad de hablar el mismo “idioma”, es decir, por ejemplo el emisor puede ser JavaScript y el receptor PHP

Estructura de un JSON



Herramientas online para crear y validar JSON

JSONCompare

JSON Compare permite ingresar un JSON y validar su estructura https://jsoncompare.com/

{JSON formatter}

JSON formatter permite formatear una estructura JSON, minimizar y transformar a CVS, XML o YAML https://jsonformatter.org/

JSON GENERATOR

JSON Generator permite generar nuevos JSON con datos dummy. https://json-generator.com/

mockaroo

Mockaroo es un generador de datos de prueba. https://www.mockaroo.com/


sustantiva
CON•SENTIDO