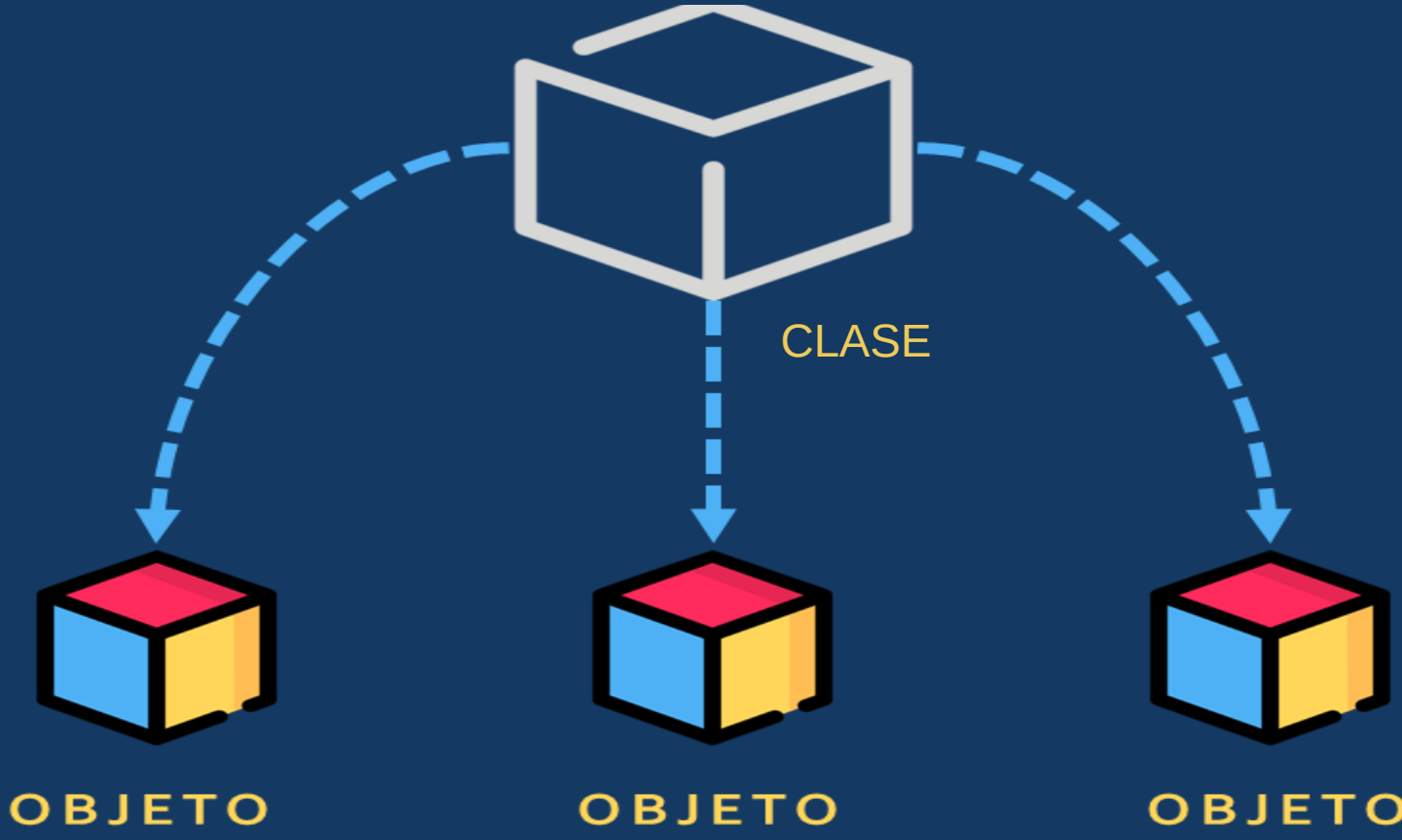
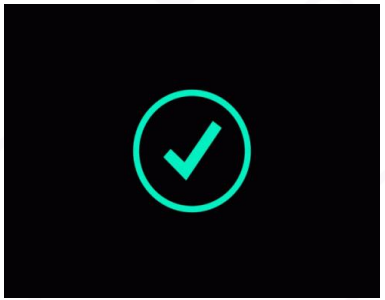
PROGRAMACIÓN ORIENTADA A OBJETOS



OBJETIVO.

Describir el Paradigma de Orientación a Objetos incluyendo los conceptos relacionados al análisis, diseño y programación



DEFINICIÓN:

La Programación Orientada a Objetos, se define como un paradigma de la programación, una manera de programar específica, donde se organiza el código en unidades denominadas clases, de las cuales se crean objetos que se relacionan entre sí para conseguir los objetivos de las aplicaciones.

VENTAJAS.

- Proximidad de los conceptos modelados respecto a objetos del mundo real.
- Facilita la reutilización de código.
  - Y por tanto el mantenimiento del mismo.
- Se pueden usar conceptos comunes durante las fases de análisis, diseño e implementación.
- Disipa las barreras entre el *qué* y el *cómo.*

QUE ES UN OBJETO.

Es una abstracción de la realidad que tiene un significado concreto y claro para el problema que se está modelando. Un ejemplo de una entidad física representada como un objeto conceptual puede ser “Un Auto”.

Todos los objetos tienen 3 características principales:

-Estado
-Comportamiento
-Identidad

QUE ES UN OBJETO.

```
const objeto = new Object(); // Esto es un objeto «genérico» vacío
```

```
const objeto = {}; // Esto es un objeto vacío
```

```
// Declaración del objeto
const player = {
  name: "Manz",
  life: 99,
  strength: 10,
};
```

ESTADO.

Lo que el objeto sabe…

- El estado de un objeto es una de las posibles condiciones en que el objeto puede existir.
- El estado normalmente cambia en el transcurso del tiempo
- El estado de un objeto es implementado por un conjunto de propiedades (atributos), además de las conexiones que puede tener con otros objetos..

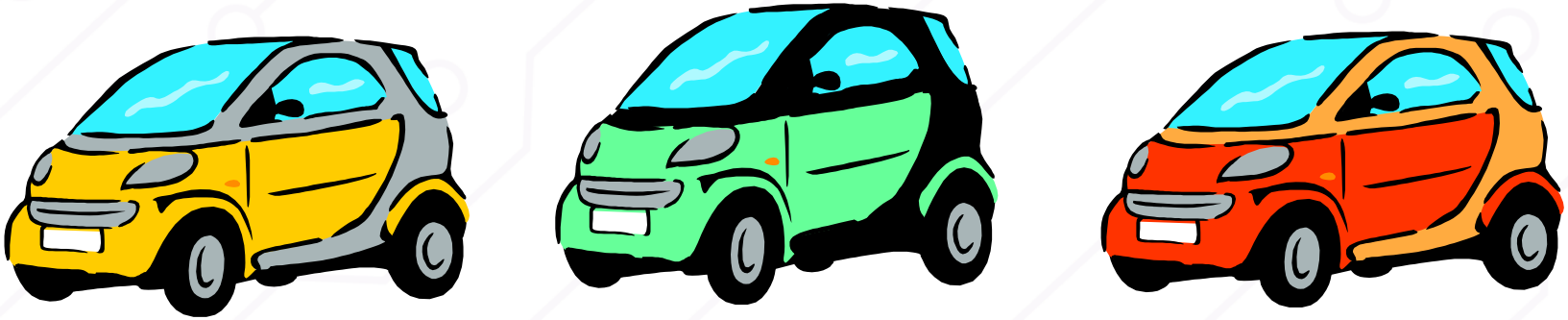
COMPORTAMIENTO.

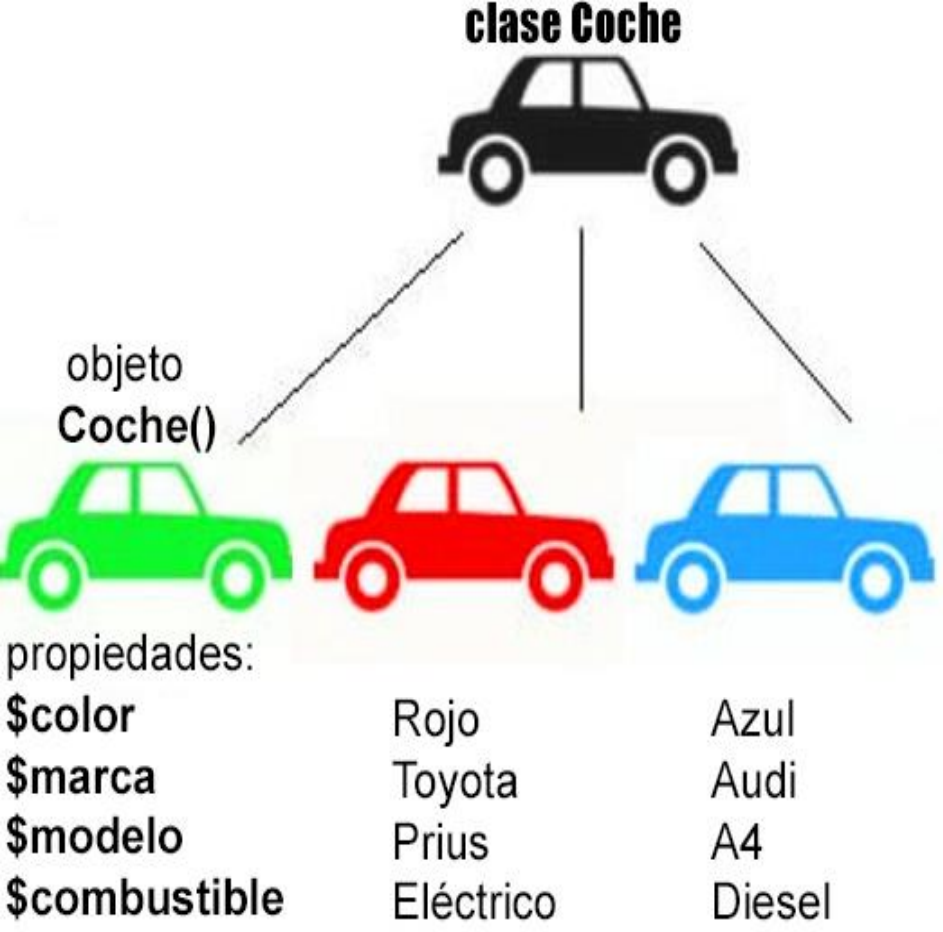
Lo que el objeto puede hacer.

- El comportamiento de un objeto determina cómo éste actúa y reacciona frente a las peticiones de otros objetos.
- Es modelado por un conjunto de mensajes a los que el objeto puede responder (operaciones que puede realizar).
- Se implementa mediante métodos.

IDENTIDAD.

- Cada objeto tiene una identidad única, incluso si su estado es idéntico al de otro objeto.





CLASE.

- Una clase es una descripción de un grupo de objetos con:
  - Propiedades en común (atributos).
  - Comportamiento similar (operaciones).
  - La misma forma de relacionarse con otros objetos (relaciones).
  - Una semántica en común (significan lo mismo).
- Es una abstracción que hacemos de nuestra experiencia sensible. El ser humano tiende a agrupar seres o cosas (objetos) con características similares en grupos (clases).

Como vemos, el ser humano tiende, de un modo natural a clasificar los objetos del mundo que le rodean en clases.

Una clase puede tener distintas características.

A estas características llamaremos ***atributos.***

Vaso
+nombre +forma +color +material +dibujo

Terminología de la POO

- Clase
- Objeto
- Atributos
- Métodos
- Instancia
- Abstracción
- Encapsulamiento
- Modularidad
- Jerarquía
- Generalización
- Herencia
- Asociación
- Agregación
- Polimorfismo
- Constructor
- Destructor
- Miembro Público
- Miembro Privado
- Miembro Protegido

ABSTRACCIÓN.

Es la capacidad que permite representar las características esenciales de un objeto sin preocuparse de las restantes características (no esenciales).

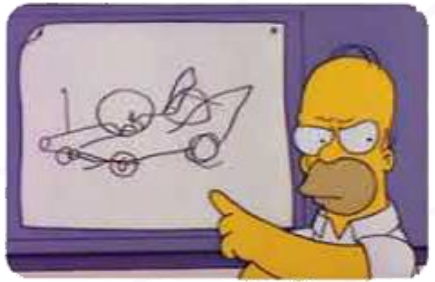
Nos permite trabajar con la complejidad del mundo real.

-Resaltando los aspectos relevantes de los objetos de una clase.

-Ocultando los detalles particulares de cada objeto.

Separaremos el comportamiento de la implementación.

Es más importante saber qué se hace en lugar de cómo se hace.



Énfasis en el ¿Qué hace? Más que en el ¿Cómo lo hace?

ENCAPSULAMIENTO.

Es la propiedad que permite asegurar que los aspectos externos de un objeto se diferencie de sus detalles internos.

Ninguna parte de un sistema complejo debe depender de los detalles internos de otra.

Complementa a la abstracción.

Se consigue:

- Separando la interfaz de su implementación.
- Ocultando la información interna de un objeto.
- Escondiendo la estructura e implementación de los métodos (algoritmos).
- Exponiendo solo la forma de interactuar con el objeto.

MODULARIDAD.

Es la propiedad que permite dividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en si y de las restantes partes.

- Consiste en separar el sistema en bloques poco ligados entre sí: módulos.
  - Organización del código.
- Es una especie de encapsulamiento de más alto nivel.
- Difícil pero muy importante en sistemas grandes.

JERARQUÍA.

Es una clasificación u ordenación de las abstracciones.

Hay dos jerarquías fundamentales:

- Estructura de clases:
  - Jerarquía “*es un/a*”
  - Relaciones de herencia
- Estructura de objetos:
  - Jerarquía “*parte de*”
  - Relaciones de agregación.

HERENCIA.

Del mismo modo, cuando definimos una clase a partir de una clase padre estamos creando una subclase. La definición de una subclase se le denomina herencia.

- ¡Relación característica de la OOP!
- Puede expresar tanto especialización como generalización.
- Evita definir repetidas veces las características comunes a varias clases.
- Una de las clases comparte la estructura y/o el comportamiento de otra(s) clase(s).
- También se denomina relación “*es un/a*” (*is a*).

- **ASOCIACIÓN:**

Una asociación es una relación semántica entre objetos. Cuando un objeto accede a los atributos y métodos de otro objeto estamos definiendo una asociación entre ellos.

- **GENERALIZACIÓN:**

Una clase que comparte atributos y métodos similares con otras clases se le llama superclase o clase padre. Cuando definimos una clase padre estamos generalizando.

AGREGACIÓN.

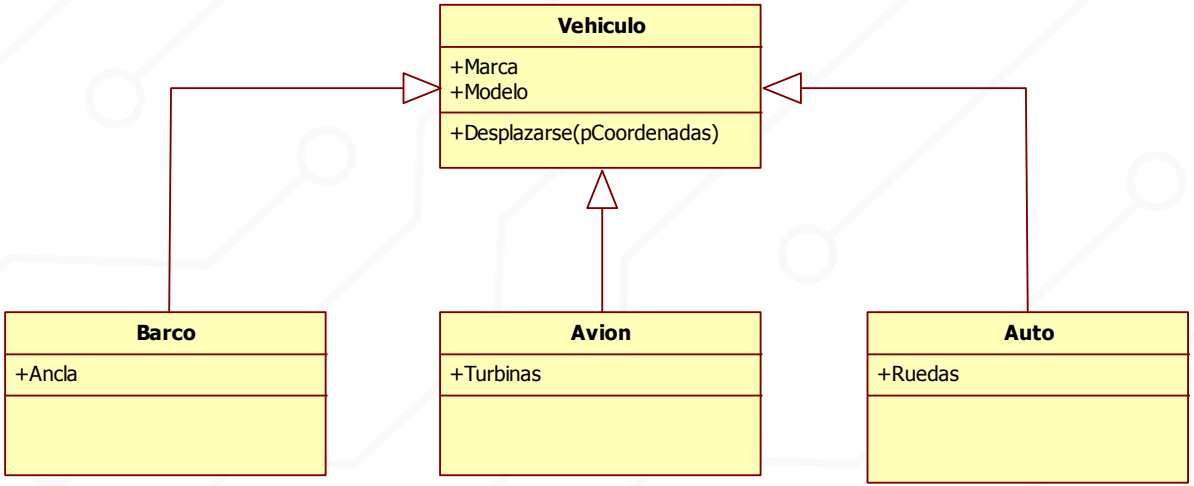
La agregación es una relación que define que un objeto es parte de otro objeto. Cuando definimos que un objeto tiene como atributo otro objeto decimos que es una agregación. A través de la agregación se definen objetos compuestos.

Una clase contiene a otra clase
Ésta “es parte de” aquélla.

También se denomina relación “es *parte de*” (has a)

POLIMORFISMO

Es el mecanismo de definir un mismo método en varios objetos de diferentes clases pero con distintas formas de implementación.



- **Miembro Público:**

Atributo o método de una clase que puede ser accesado desde cualquier parte del programa.

- **Miembro Privado:**

Atributo o método de una clase que puede ser accesado solo dentro de esa clase.

- **Miembro Protegido:**

Atributo o método de una clase que puede ser accesado desde esa clase y sus clases heredadas.

PROTOTYPE EN JAVASCRIPT

DEV.F:
DESARROLLAMOS(PERSONAS);

OBJETIVO.

Comprender los prototipos de objetos de Javascript, cómo funciona la cadena de prototipo, y cómo añadir nuevos métodos a la propiedad prototipo.

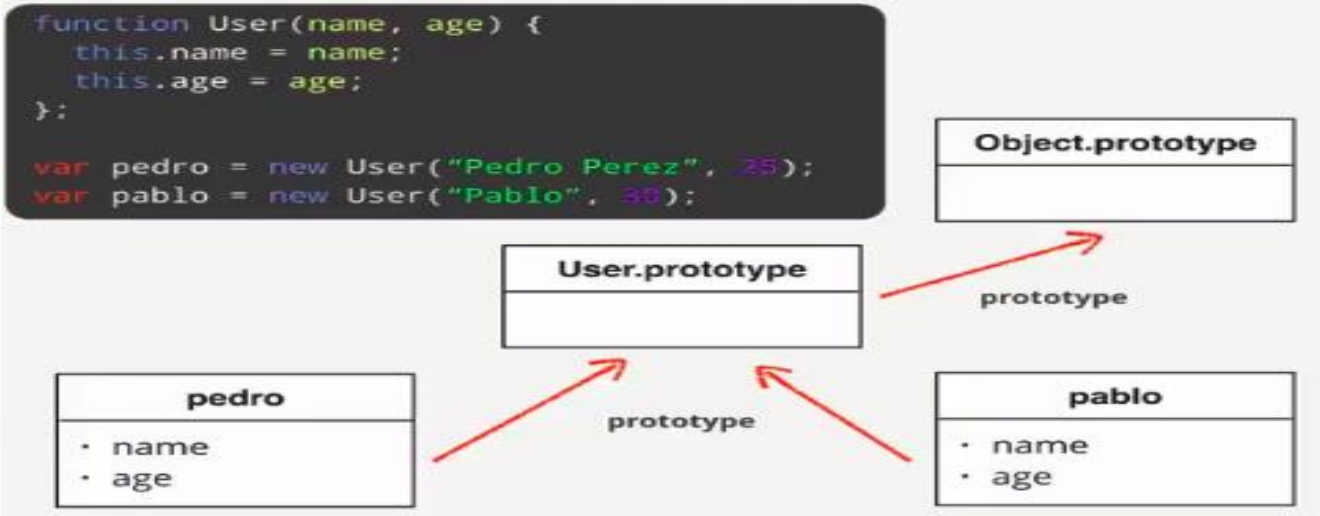
PROTOTIPOS.

Los prototipos son un mecanismo mediante el cual los objetos en JavaScript heredan características entre sí.

JavaScript es a menudo descrito como un lenguaje basado en prototipos para proporcionar mecanismos de herencia, los objetos pueden tener un objeto prototipo, el cual actúa como una plantilla objeto que hereda métodos y propiedades.

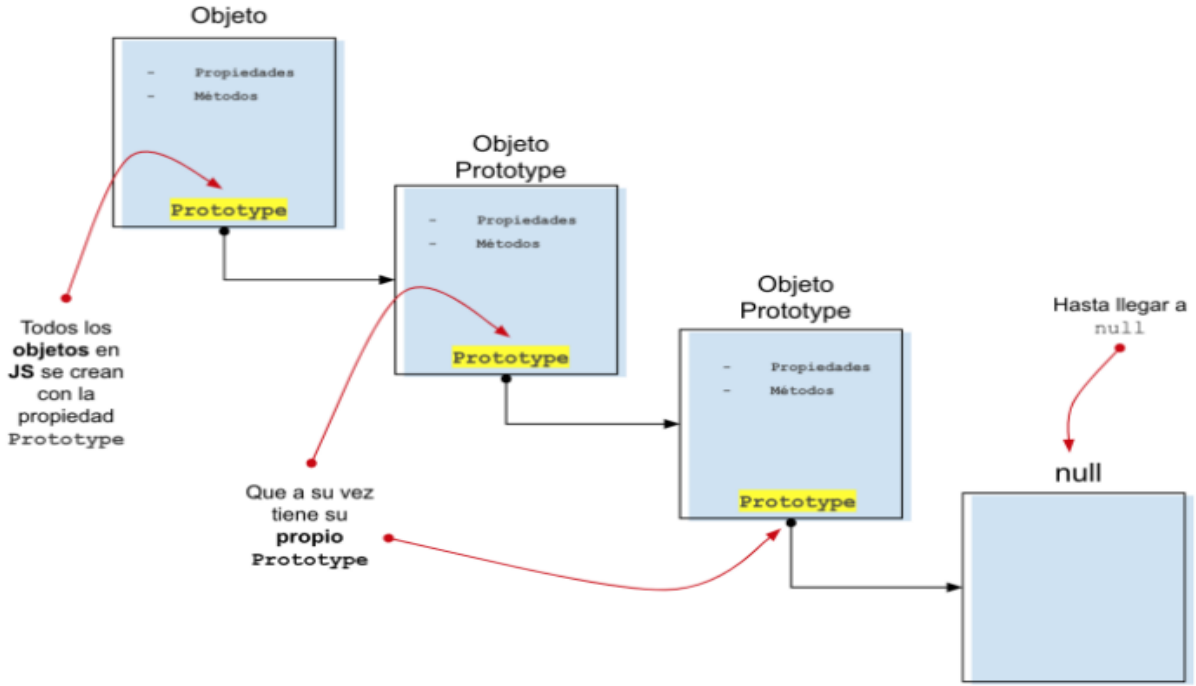
```
function User(name, age) {
  this.name = name;
  this.age = age;
};

var pedro = new User("Pedro Perez", 25);
var pablo = new User("Pablo", 30);
```

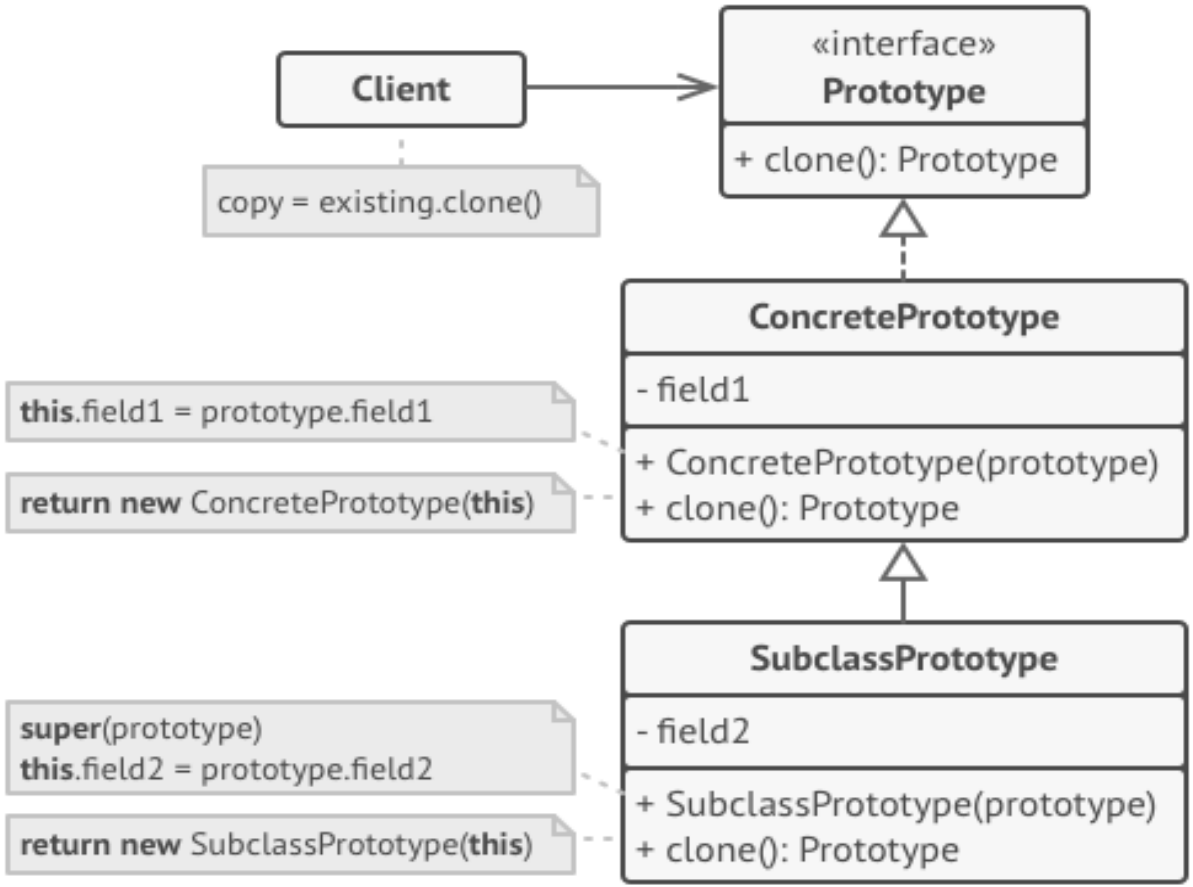


FUNCIONAMIENTO.

Si una propiedad no se encuentra en el objeto mismo, hay un intento de buscarla en el prototipo, si no se encontró entonces se busca en el prototipo del prototipo, etc.



Estructura.



```
1  function double(num){
2      return num * 2;
3  }
4
5  double.toString() // where is this method coming from?
6
7  Function.prototype // {toString: [f], call: [f], bind: [f]}
8
9  double.hasOwnProperty('name') // where is this method coming from?
10
11 Function.prototype.__proto__ // -> Object.prototype {hasOwnProperty: [f]}
```

Global Memory

Las relaciones de prototipo en JavaScript tienen forma de árbol, y la raíz de esta estructura es Object.prototype. Este provee unos pocos métodos que se mostrarán en casi todos los objetos, como toString, que convierte un objeto en una representación, en una cadena de texto.

Las funciones derivan de Function.prototype, y los arrays derivan de Array.prototype.

```
console.log(Object.getPrototypeOf(isNaN) ==
            Function.prototype);
// → true
console.log(Object.getPrototypeOf([]) ==
            Array.prototype);
// → true
```

Como un objeto prototipo tiene su propio prototipo normalmente Object.prototype, entonces este indirectamente provee de métodos como toString.

La función Object.getPrototypeOf obviamente devuelve el prototipo de un objeto. Puedes usar Object.create para crear un objeto con un prototipo específico.

```
var protoConejo = {
  hablar: function(linea) {
    console.log("El conejo " + this.type + " dice '" +
                linea + "'");
  }
};
var conejoAsesino = Object.create(protoConejo);
conejoAsesino.type = "asesino";
conejoAsesino.hablar("SKREEEE!");
// → El conejo asesino dice 'SKREEEE!'
```

CONSTRUCTORES

En JavaScript, llamar a una función con la palabra clave new delante de ella, hace que sea tratada como un constructor. El constructor tendrá su variable this en los límites del objeto creado, y si no se específica otro valor de objeto este será el nuevo objeto que retorne la llamada.

Un objeto creado con `new` se dice que es una instancia de su constructor.

```
function Conejo(tipo) {
  this.tipo = tipo;
}

var conejoAsesino = new Conejo("asesino");
var conejoNegro = new Conejo("negro");
console.log(conejoNegro.tipo);
// → negro
```

```
function Conejo(tipo) {
  this.tipo = tipo;
}

var conejoAsesino = new Conejo("asesino");
var conejoNegro = new Conejo("negro");
console.log(conejoNegro.tipo);
// → negro
```

```
negro
```

```
Conejo.prototype.hablar = function(linea) {
  console.log("El conejo " + this.tipo + " dice '" +
              linea + "'");
};
conejoNegro.hablar("Maldición...");
// → El conejo negro dice 'Maldición...'
```

```
El conejo negro dice 'Maldición...'
```

SUGAR SYNTAX EN JAVASCRIPT

DEV.F::
DESARROLLAMOS(PERSONAS);

¿Qué significa ‘sugar' en términos de JavaScript?

Sugar syntax es un término general acuñado por un informático llamado Peter Landin.

Lo contrario de " syntactic sugar" es "syntactic salt ", que es una característica diseñada para dificultar la escritura de código incorrecto.

Todos los lenguajes de programación son “Sugar Syntax" para otra cosa.
Están diseñados para hacer la vida más fácil a los programadores.
Desde bits hasta código de máquina, C, C++, Java, JavaScript, Python...
cada abstracción es azúcar sintáctica para una abstracción anterior, con la que era más difícil trabajar.

Ejemplo.

```
1  for (var i = 0; i < 10; i++) {
2          println(i);
3  }
```

```
1  int i = 0;
2  while(i < 10) {
3          println(i);
4          i++;
5  }
```

Ejemplo.

```
// Crear la matriz aircrafts
let aircrafts = ['the bus', 'zephyr one', 'quinjet']

// Recorrer toda la matriz aircrafts
for (let i = 0; i < aircrafts.length; i++) {

// Imprimirá el elemento en el índice i de la matriz de airecrafts
  console.log(aircrafts[i])
}
```

Ejemplo.

```
for (variable of iterable) {
 // Instrucción que se ejecuta para cada variable
}
```

```
// Crear la matriz aircrafts
    let aircrafts = ['the bus', 'zephyr one', 'quinjet']

// Recorrer toda la matriz aircrafts
    for (let i of aircrafts) {

// Imprimirá el elemento en el índice i de la matriz de airecrafts
     console.log(i)

    }
```

CONCLUSIÓN.

```
function Dog(name, breed, color) {
  this.name = name
  this.breed = breed
  this.color = color
}

Dog.prototype.bark = function() {
  return 'Woof!'
}
```

```
class Dog {
  constructor(name, breed, color) {
    this.name = name
    this.breed = breed
    this.color = color
  }

  bark() {
    return 'Woof!'
  }
}
```