

OOP y Prototypes

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Temas



- Introducción a POO.
- Creación de objetos en JS.
- Objetos y clases en JS (prototipos).
- Sugar sintaxis para clases y objetos.
- Contexto de this.
- Call, apply y bind.

Introducción a la POO

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Propósito de los Objetos y Array

- Un **objeto** es una **entidad del mundo real** que tiene **información** que lo distinguen (**atributos**) y pueden **hacer cosas** (**métodos**).
- Un **arreglo** es un **conjunto** o una **lista** de valores simples o complejos.



Usamos objetos cuando necesitamos representar un elemento de la vida real.



Usamos arreglos cuando necesitamos representar conjuntos o grupos.



Introducción a la POO

DEV.FX
DESARROLLAMOS(PERSONAS);

Paradigma de programación

Es una **forma o estilo de programación**. Se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para definir el cómo implementar el código.

Los lenguajes de programación **adoptan uno o varios paradigmas** en función del objetivo para el que fueron creados.

Por ejemplo, Python o JavaScript, que son **multiparadigmas**.



- Secuencial.
- Orientado a eventos.
- Orientado a objetos.
- Funcional.
- Declarativo.

Algunos paradigmas

- **Programación estructurada:** El código siempre se escribe de arriba hacia abajo.
- **Programación funcional:** El core del lenguaje se basa en la creación y separación en funciones.
- **Programación imperativa:** Cada línea de código define explícitamente qué se debe hacer. *“Ve por pan, toma dinero, agarra las llaves, sal a la calle, ve a la panaderia...”*
- **Programación declarativa:** Mediante abstracción existe código no tan explícito, simplemente se puede interpretar sin fijar las condiciones o ciclos. *“Trae pan”*
- **Programación Orientada a Eventos:** El sistema reacciona a partir de acciones del usuario.
- **Programación Orientada a Objetos:** El diseño se centra en crear plantillas para instanciar objetos y poder hacerlos definidos pero escalables. Así mismo se mantienen comunicados.

amazon.com.mx

Crear cuenta

Tu nombre

Nombre y apellido

Número de celular o correo electrónico

Contraseña

Debe tener al menos 6 caracteres

La contraseña debe contener al menos seis caracteres.

Vuelve a escribir la contraseña

Continuar

Al crear una cuenta, aceptas las [Condiciones de Uso](#) y el [Aviso de Privacidad](#) de Amazon.

¿Ya tienes una cuenta? [Iniciar sesión](#)

Sign Up

It's quick and easy.

First name

Last name

Mobile number or email

New password

Birthday

Sep

8

2023

Gender

Female

Male

Custom

People who use our service may have uploaded your contact information to Facebook. [Learn more.](#)

By clicking Sign Up, you agree to our [Terms](#), [Privacy Policy](#) and [Cookies Policy](#). You may receive SMS Notifications from us and can opt out any time.

Sign Up

Google

Create a Google Account

Enter your name

First name

Last name (optional)

Next

¿Alguna vez te has registrado en una página?

- Te imaginas que cada que un usuario nuevo se registra el desarrollador de ese sistema tuviera que armar un nuevo objeto?



OOP

Object Oriented Programming

DEV.FX
DESARROLLAMOS(PERSONAS);

dev

Oriented Object Programming

Es un paradigma de programación que centra su diseño en imaginar los sistemas como un conjunto de objetos relacionados entre sí.

- Busca **dejar de centrarnos en la lógica** pura de los programas, para empezar a pensar en objetos del **mundo real**.
- Define una **plantilla base para crear objetos** del mismo tipo.
- **Instanciar objetos** a partir de la plantilla.
- Los objetos están **relacionados**.
- En vez de pensar en funciones, pensamos en las **relaciones o interacciones de los diferentes componentes del sistema**.
- Permite que el código sea **reutilizable, organizado y fácil de mantener**.

Principios de POO

- **Abstracción:** Todo en el mundo real es un objeto con atributos y métodos.
- **Encapsulamiento:** Agrupar elementos y datos dentro de los elementos a los que le corresponden por escalabilidad e integridad de información.
- **Herencia:** Las características y funciones de un objeto padre son heredadas por los hijos.
- **Polimorfismo:** Los objetos pueden tener distintos comportamientos, es decir un método puede hacer distintas cosas de acuerdo a su implementación (recibir distintos parámetros).



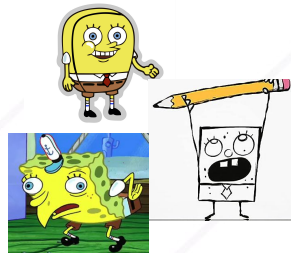
Abstracción



Encapsulamiento



Herencia



Polimorfismo

Conceptos

- **Clase: Modelo o plantilla** a partir del cual se crean objetos.
- **Objeto:** Es un **elemento concreto** del mundo real.
- **Instanciar:** Crear un objeto concreto a partir de una plantilla.
- **Atributo:** También se le conoce como **característica** o **propiedad**.
- **Método:** También se les conoce como **función** y son las acciones que el objeto de la clase puede realizar.



Carro
marca: String
modelo: int
cantidadGasolina: double
Carro(String, String)
getModelo():String
tanquear(double):void

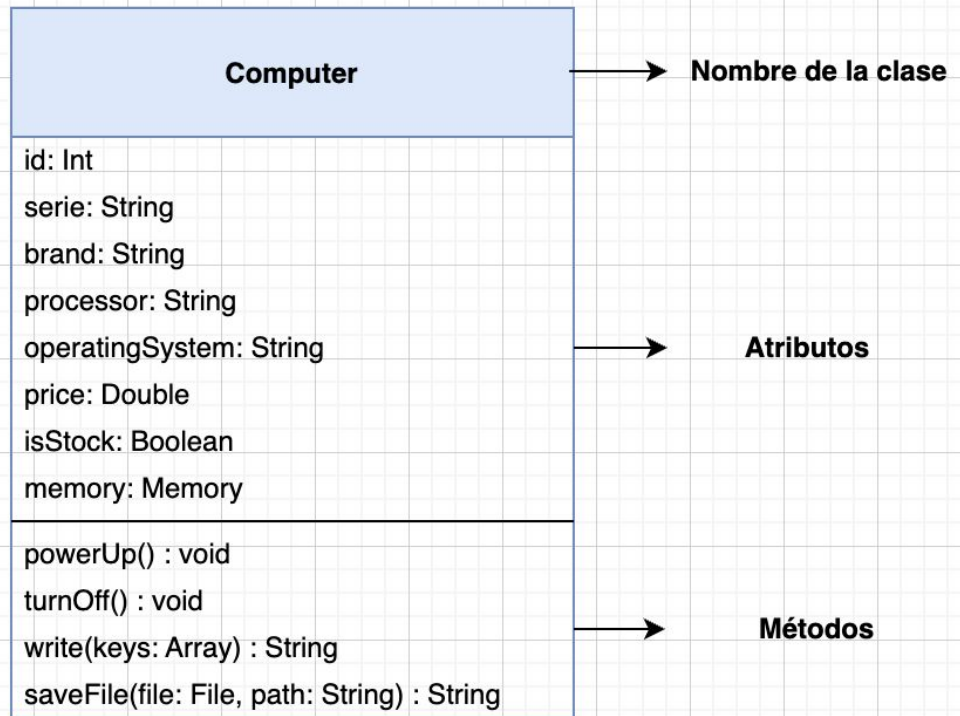
UML

Unified Model Language

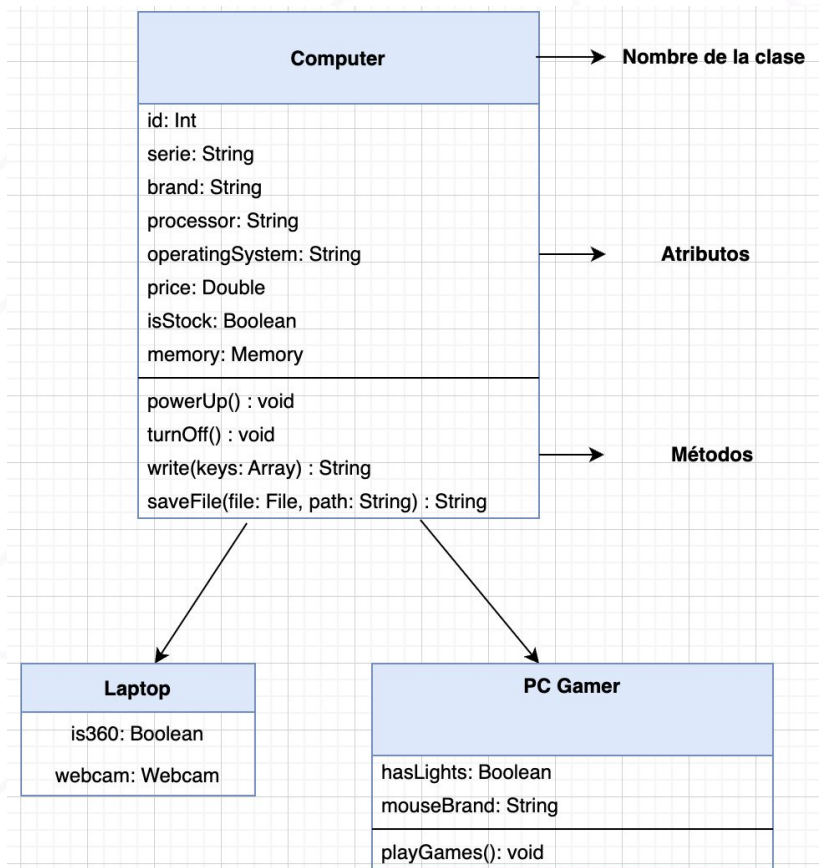
DEV.FX
DESARROLLAMOS(PERSONAS);

dev

Diagrama UML de una clase



Ejemplo de Herencia



Práctica

- Realizar el diagrama UML de una jerarquía de clases de autos con draw.io.



Creación de objetos en JS

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Creación de objetos en JS

Javascript es un lenguaje que por naturaleza no es orientado a objetos sino orientado a prototipos. Es decir, inicialmente no contaba con clases para definir objetos sino que creaba objetos y extendía sus características a través de una propiedad que podían compartir y que compartían todo de Object.

- **Objetos literales:** Creación de objetos de forma manual c/vez que los necesite, no se tiene una plantilla base.
- **Función creadora (factory o constructora):** Crear una función que genere objetos.
- **Función constructora con new:** Crear una función constructor con new y this.
- **Prototype:** Generar clases mediante funciones o prototipos.
- **Sugar syntax:** Usar class, extends, new, super, this y constructor para crear clases de una manera más legible.

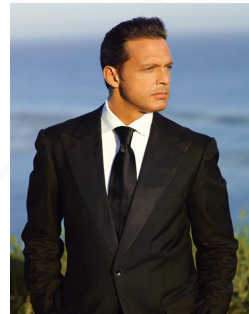
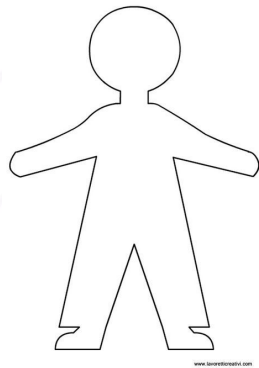
Cadena de prototipos

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Lenguaje basado en prototipos

Los objetos pueden tener un objeto base que utilizan como plantilla (**prototipo**) para crear nuevos objetos y heredar métodos y/o propiedades.



Cadena de prototipos

Un objeto prototipo del objeto puede tener a su vez otro objeto prototipo, el cual hereda métodos y propiedades, y **así sucesivamente**. Esto es conocido con frecuencia como la cadena de prototipos.



Principios de la cadena de prototipos

- **Object:** Es el objeto de mayor jerarquía en JS y todos los objetos se creen lo tienen como prototipo.
- **Prototipo:** Es similar a una clase, es un molde creado en JS que se usa como base para crear objetos.
- **Cadena de prototipos:** Simula el concepto de herencia mediante funciones constructoras y la propiedad **prototype**. Cada elemento tiene un prototipo del que hereda atributos y métodos.

NOTA: Inicialmente JavaScript contaba con herencia prototipal pero a raíz de las características de otros lenguajes se implementó Class, Constructor, extends, get y set como medios para hacer más natural la implementación de JavaScript.

Teoría de los Prototipos

- En JavaScript, **todos los objetos tienen una propiedad oculta `[[Prototype]]`**.
- Los métodos y propiedades **se agregan a la propiedad `prototype`**, que reside en la función constructora del objeto.
- Para acceder al prototipo desde un objeto se utiliza la propiedad `__proto__`.
Ejemplo: `obj.__proto__`
- Para acceder al constructor directo usar **`obj.constructor`**.
- Si queremos leer una propiedad de objeto o llamar a un método, y no existe, entonces JavaScript intenta encontrarla en sus prototipo y si hay no lo encuentra ira al prototipo del prototipo.
- Un patrón bastante común para la mayoría de definiciones de objetos es declarar las propiedades dentro del constructor, y los métodos en el prototipo.

Teoría de los Prototipos

- Si llamamos a `obj.method()`, y `method` se toma del prototipo, this todavía hace referencia a `obj`.
- Por lo tanto, los métodos siempre funcionan con el objeto actual, incluso si se heredan.

Métodos interesantes de prototipos y Object

- `obj.__proto__`
- `obj.constructor`
- `obj.constructor.name`
- `fnConstructora.constructor`
- `obj.hasOwnProperty(propName);`
- `obj.isPrototypeOf(obj);`
- `Object.is`
- `Object.getOwnPropertyNames`
- `Object.hasOwn`

Práctica

Generar una jerarquía de clases con prototype.



Sugar Syntax

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Keywords para clases en JS.

- Class.
- Extends.
- Constructor.
- This.
- Super.
- Get y Set.

```
1 function createSuperHumano(name, altura, colorPiel, superpoder, tipo, alias) {
2   this.__proto__ = new createPersona(name, altura, colorPiel);
3   this.superpoder = superpoder;
4   this.tipo = tipo;
5   this.alias = alias;
6   Object.seal(this);
7 };
8
9 createSuperHumano.prototype.volar = () => {
10   console.log('Soy superhumano y puedo volar');
11 };
```

```
1 class SuperHumano extends Persona {
2   constructor(name, altura, colorPiel, superpoder, tipo, alias) {
3     super(name, altura, colorPiel);
4     this.superpoder = superpoder;
5     this.tipo = tipo;
6     this.alias = alias;
7   }
8   volar() {
9     console.log('Soy superhumano y puedo volar');
10  }
11 }
```

Práctica

Generar una jerarquía de clases con sugar syntax.



Contexto de this

DEV.F
DESARROLLAMOS(PERSONAS);

dev

This

- La palabra clave `this` de una función se comporta un poco diferente en Javascript en comparación con otros lenguajes. Además tiene algunas diferencias entre el modo estricto y el modo no estricto.
- Hace referencia al objeto en cuestión. Es decir, si estamos creando cualquier función, la palabra clave `this` será el objeto que mandó ejecutar esa función.
- Dependiendo del contexto de ejecución `this` puede tener un valor distinto.

Práctica

Realizar practicar sobre el contexto de this en:

- Global.
- En una función.
- En una función con strict mode.
- En objetos.
- Desde fuera de objeto.
- A partir de un objeto nuevo creado por una función Constructora.



Call, Apply y bind

DEV.F
DESARROLLAMOS(PERSONAS);

dev

Call, bind y apply

- **Call:** Permite establecer el contexto `this` con el que se llamará a una función y con argumentos provistos individualmente..
- **Apply:** Permite establecer el contexto `this` con el que se llamará a una función y con argumentos se informan como un arreglo.
- **Bind:** Crea una nueva función con un nuevo contexto `this`.

Práctica

- Ejecutar un cambio de contexto del this por medio de call, bind y apply.



Docs

DEV.F
DESARROLLAMOS(PERSONAS);

dev

- [¿Qué es la programación orientada a objetos?](#)
- [Object-oriented Programming in JavaScript: Made Super Simple.](#)
- [OOP in JS \(De prototypes a clases\).](#)
- [Creación de Objetos JavaScript.](#)
- [Classes.](#)
- [Herencia y cadena de prototipos.](#)