OOP y Prototypes

DEV.F:
DESARROLLAMOS(PERSONAS);

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.

DEV.F::

Introducción a la POO

DEV.F:
DESARROLLAMOS(PERSONAS);

Propósito de los Objetos y Array

- Un **objeto** es una **entidad del mundo real** que tiene **información** que lo distinguen (**atributos**) y **hacen 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.

DEV.F

Introducción a la POO

DEV.F::
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.

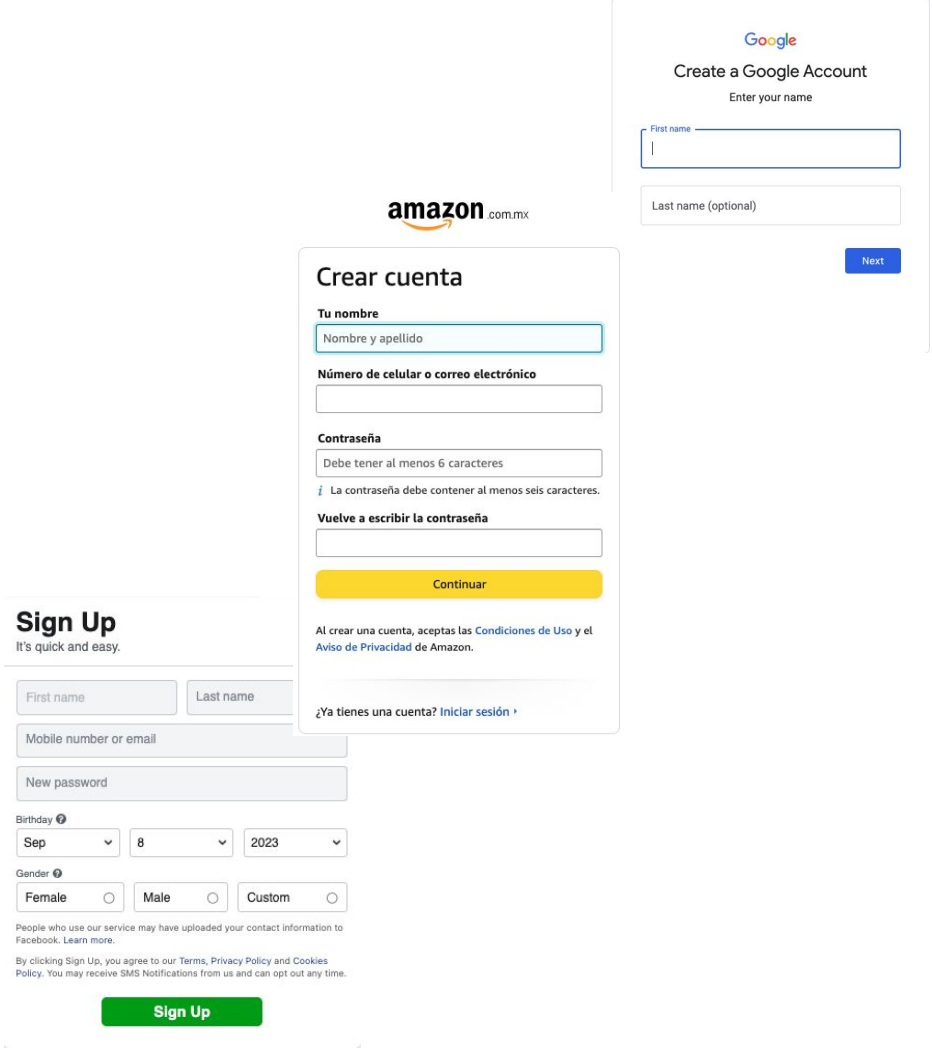
Los lenguajes de programación **adoptan uno o varios paradigmas** en función del objetivo para el que fueron creados.

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

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

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.
- **Programación declarativa:** Mediante abstracción existe código no tan explícito, simplemente se puede interpretar sin fijar las condiciones o ciclos.
- **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.



¿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?

DEV.F::

OOP
Object Oriented Programming

DEV.F:
DESARROLLAMOS(PERSONAS);

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**.

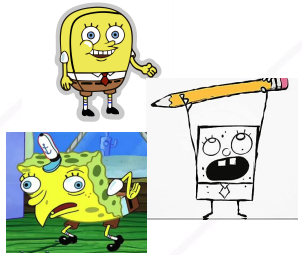
Principios de POO

- **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

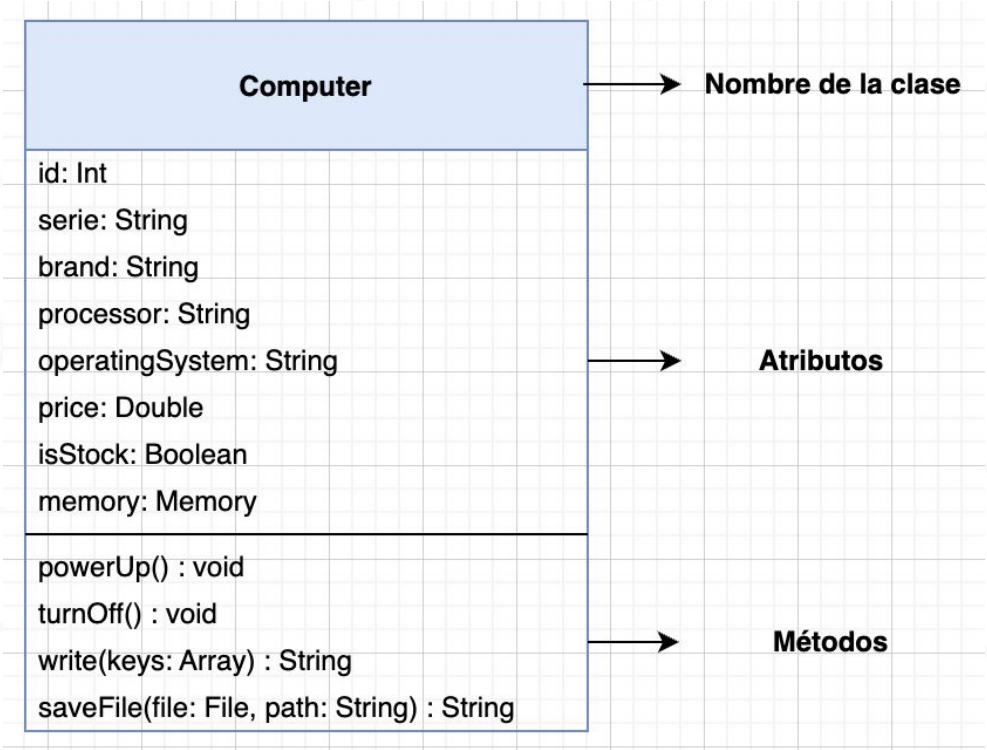
- **Clase: Modelo** o **plantilla** a partir del cual se crean objetos.
- **Objeto:** Es un **elemento concreto** del mundo real.
- **Instanciar:** Crear un objeto concreto a partir de una plantilla.
- **Atributo:** También se le conoce como **característica** o **propiedad**.
- **Método:** También se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

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

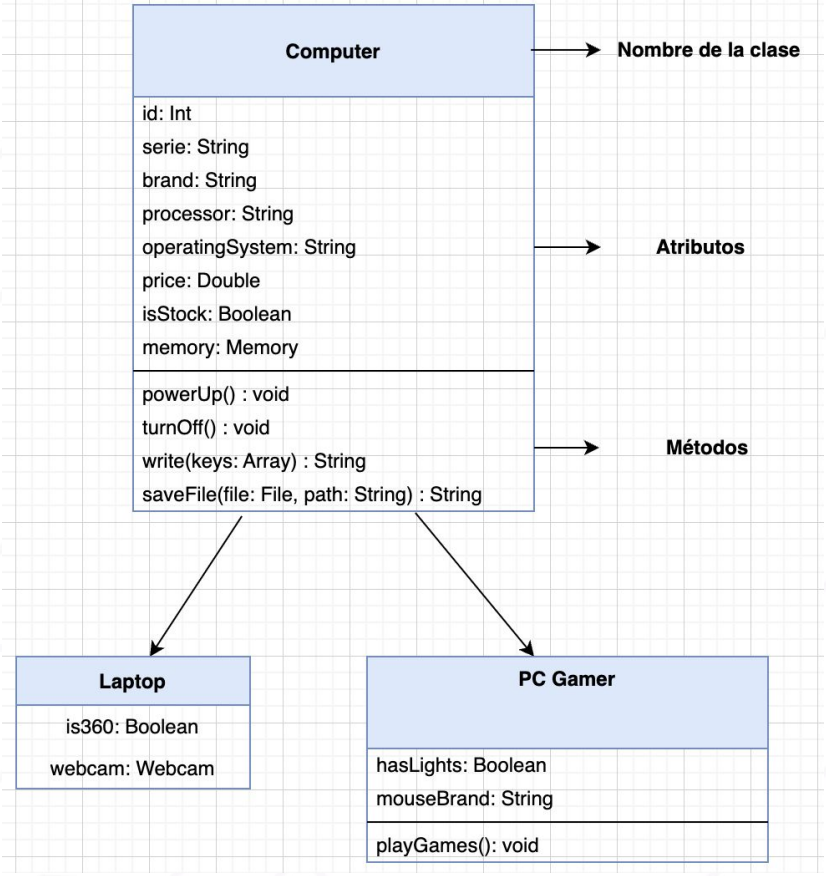
UML
Unified Model Language

DEV.F::
DESARROLLAMOS(PERSONAS);

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);

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.

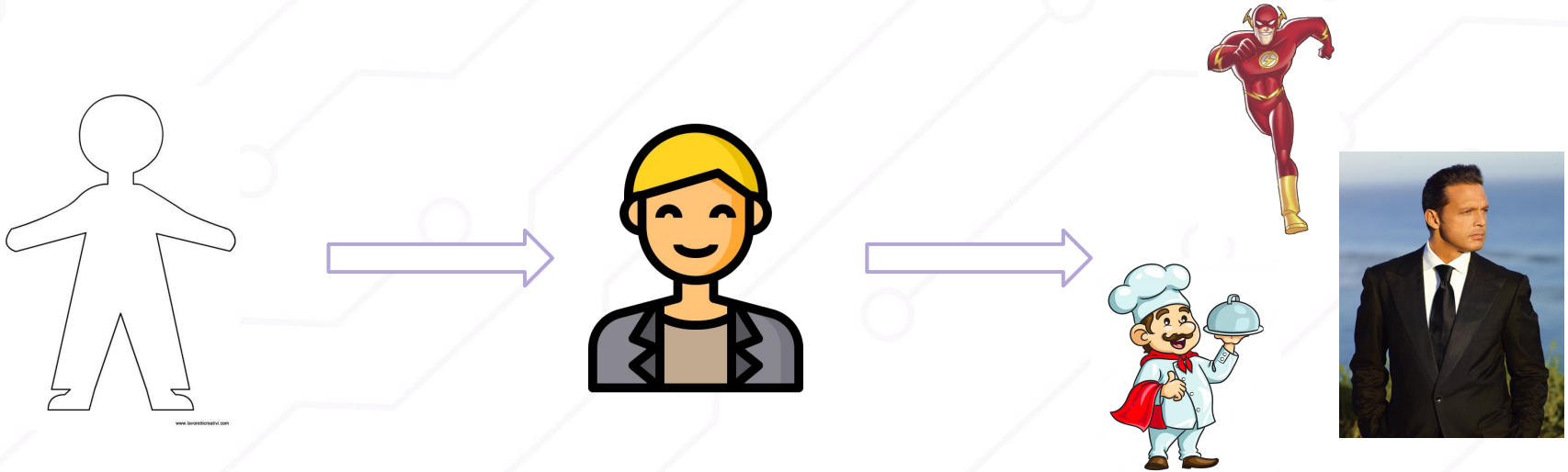
- Objetos literales.
- Object create.
- Función constructora.
- Función constructora con new.
- Prototype.
- Sugar syntaxis.

Cadena de prototipos

DEV.F::
DESARROLLAMOS(PERSONAS);

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

- **Prototipo:** Es similar a una clase, es un molde base para crear objetos.
- **Object:** Es el objeto de mayor jerarquía en JS y todos los objetos se creen lo tienen como prototipo.
- **Cadena de prototipos:** Simula el concepto de herencia mediante el objeto prototype.

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]]** que es otro objeto o nulo.
- 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__**.
- Podemos usar **obj.__proto__** para acceder a él (un getter/setter histórico, hay otras formas, que se cubrirán pronto).
- También se puede acceder directamente al prototype mediante .prototype.

Teoría de los Prototipos

- También se puede acceder desde la función constructora apuntando a su **constructor**.
- Si queremos leer una propiedad de obj o llamar a un método, y no existe, entonces JavaScript intenta encontrarla en el 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

- Las operaciones de escritura/eliminación actúan directamente sobre el objeto, no usan el prototipo (suponiendo que sea una propiedad de datos, no un setter).
- 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.
- El bucle for..in itera sobre las propiedades propias y heredadas. Todos los demás métodos de obtención de valor/clave solo operan en el objeto mismo.

Métodos interesantes de prototipos y Object

- Object.getPrototypeOf(flash(
- flash.__proto__
- createSuperHumano.constructor
- flash.constructor.name;
- flash.hasOwnProperty('alias'); // true
- flash.hasOwnProperty('colorTraje'); // false
- createSuperHumano.isPrototypeOf(flash); // true

- Object.is(flash, mujerMaravilla); // false
- Object.getOwnPropertyNames(flash);
- Object.hasOwn(flash, 'volar');

Práctica 2

Generar una jerarquía de clases con cada uno de los métodos mencionados.

Sugar Sintaxis

DEV.F::
DESARROLLAMOS(PERSONAS);

Keywords para clases en JS.

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

```
function createSuperHumano(name, altura, colorPiel, superpoder, tipo, alias) {
    this.__proto__ = new createPersona(name, altura, colorPiel);
    this.superpoder = superpoder;
    this.tipo = tipo;
    this.alias = alias;
    Object.seal(this);
};

createSuperHumano.prototype.volar = () => {
    console.log('Soy superhumano y puedo volar');
};
```

```
class SuperHumano extends Persona {
    constructor(name, altura, colorPiel, superpoder, tipo, alias) {
        super(name, altura, colorPiel);
        this.superpoder = superpoder;
        this.tipo = tipo;
        this.alias = alias;
    }
    volar() {
        console.log('Soy superhumano y puedo volar');
    }
}
```

Práctica 3

Generar una jerarquía de clases con sugar syntaxis.

Contexto de this

DEV.F:
DESARROLLAMOS(PERSONAS);

Práctica 4

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);

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 5

- Ejecutar un cambio de contexto de this con call, bind y apply.

Docs

DEV.F::
DESARROLLAMOS(PERSONAS);

Docs

- ¿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 prototypos.