Computer Science

DEV.F:
DESARROLLAMOS(PERSONAS);

Temas

- Introducción a POO.
- 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);

¿Qué es la programación orientada a objetos?

Es un paradigma de programación, es decir, un estilo de programación que nos da unas guías sobre cómo programar los sistemas.

- Se basa en el concepto de clases y objetos.
- Busca dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos del mundo real.
- 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.
- **Encapsulamiento:** Agrupar elementos dentro de un mismo nivel de abstracción.
- **Polimorfismo:** Los objetos pueden tener distintos comportamientos.
- **Herencia:** Las características y funciones de un objeto padre son heredadas por los hijos.
- **Modularidad:** Se separa en archivos llamados clases, las funcionalidades.
- **Principio de ocultamiento:** Los objetos sólo deben acceder a la información por medio de getters y setters.

Conceptos

- **Clase: Modelo** o **plantilla** a partir del cual se crean objetos.
- **Objeto:** Es un **elemento concreto** del mundo real.
- **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.

Diagrama UML de una clase

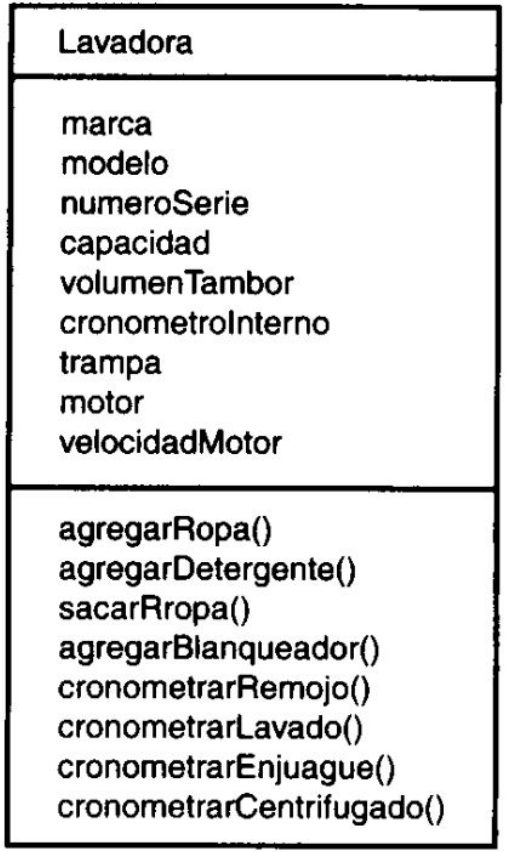
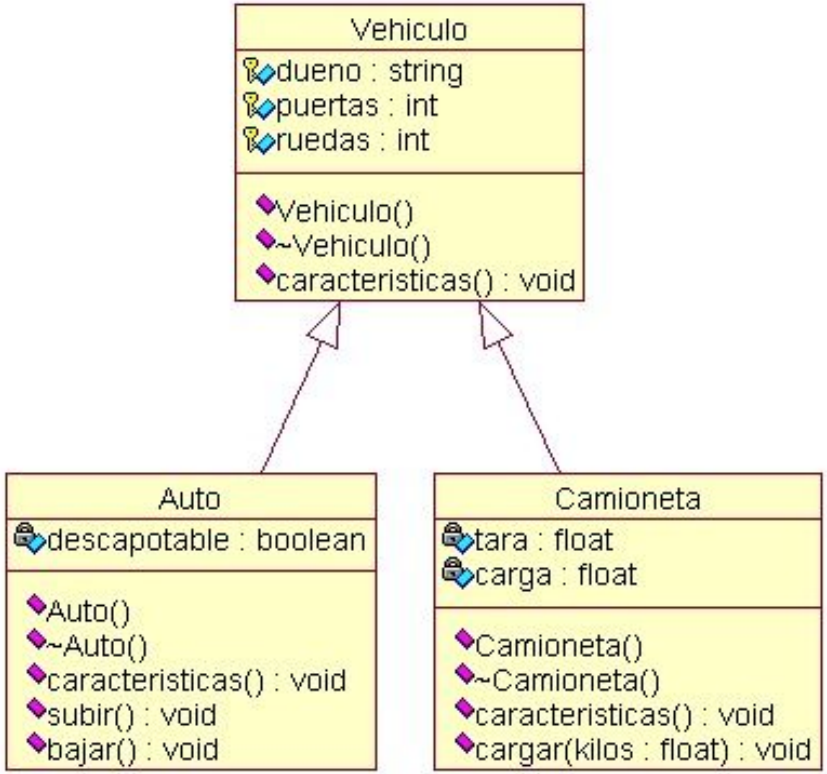


Diagrama UML jerarquía de clases



Práctica 1

- Realizar el diagrama UML de una jerarquía de clases de autos.

Creación de objetos en JS

DEV.F:
DESARROLLAMOS(PERSONAS);

Creación de objetos en JS

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

JavaScript y la cadena de prototipos

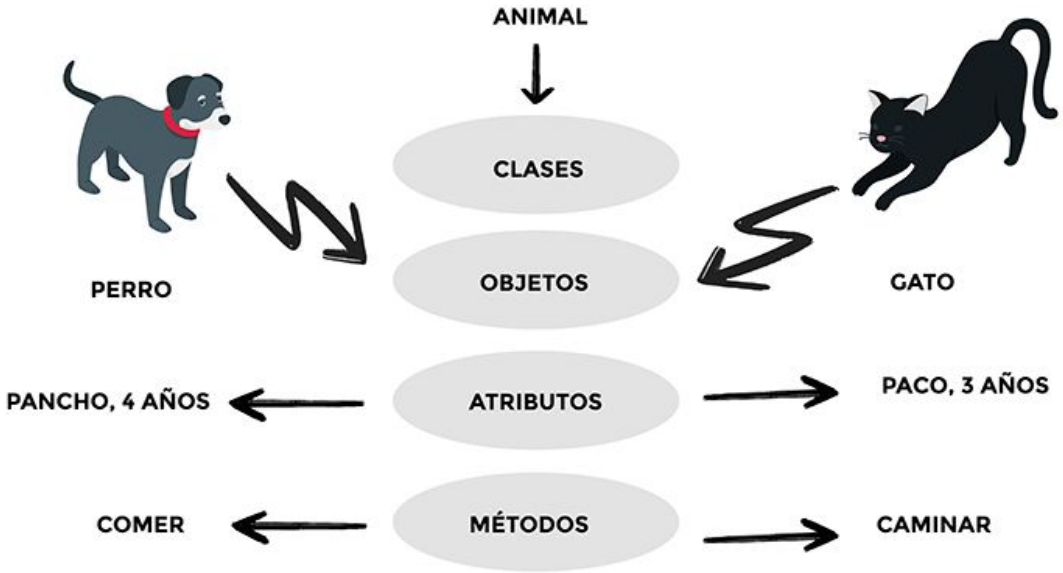
DEV.F:
DESARROLLAMOS(PERSONAS);

Principios de la cadena de prototipos

- **Prototipo:** Es similar a una clase.
- **Object:** El objeto de mayor jerarquía en JS.
- **Cadena de prototipos:** Simula el concepto de herencia mediante el objeto prototype.

Lenguaje basado en prototipos

JavaScript es a menudo descrito como un 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.



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).

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.

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.

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 Syntaxis para clases y objetos

DEV.F:: DESARROLLAMOS(PERSONAS);

Keywords para clases en JS.

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

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.