

Instituto de Enseñanza Secundaria
Santiago Hernández

DESARROLLO WEB EN ENTORNO CLIENTE

(Curso 2021 – 2022)

**UD3. JAVASCRIPT AVANZADO.
CLASES Y OBJETOS.**

OBJETOS

Objetos

- Elementos que pueden contener cero o más pares clave/valor asociados.
- Ejemplo:
 - `var persona = {`
 `nombre: "Luis",`
 `apellidos: "Martinez Campo",`
 `hijos: "2"`
■ `}`
- Esta forma de definir sin clases es anterior a ES6.
- Válido para objetos simples

OBJETOS

Funciones constructoras de objetos

- Permiten crear objetos diferentes con la misma estructura.
- Cualquier función puede convertirse en un constructor usando `this`.
- Ejemplo:
 - `function Persona(nombre,apellidos){`
 - `this.nombre = nombre;`
 - `this.apellidos = apellidos;`
 - `}`
- Esta función permite “crear Personas” diferentes:
 - `var persona1 = new Persona('Ana', 'Pérez');`
 - `var persona2 = new Persona('Lola', 'Gómez');`
- Buenas prácticas: nombrar la función constructora comenzando con mayúscula.

OBJETOS

Seguridad en funciones constructoras

- Verificación de valores pasados para la “construcción”.
 - `var Persona = function(nombre,apellidos,edad){ //equivalente a function Persona`
`this.nombre = new String(nombre);`
`this.apellidos = new String(apellidos);`
`if (isNaN(edad)){`
`alert("El valor dado para edad no es un número");`
`this.edad=0; //opcionalmente podemos dar un valor por defecto.`
`}else{`
`this.edad = new Number(edad);`
`}`
■ `}`
 - Ejemplo en <https://cursosgs.es/dwec/2017/11/02/2-10-clases-y-objetos-old-style/>

Es lo mismo que:
`function Persona (nombre, apellidos, edad){}`

OBJETOS

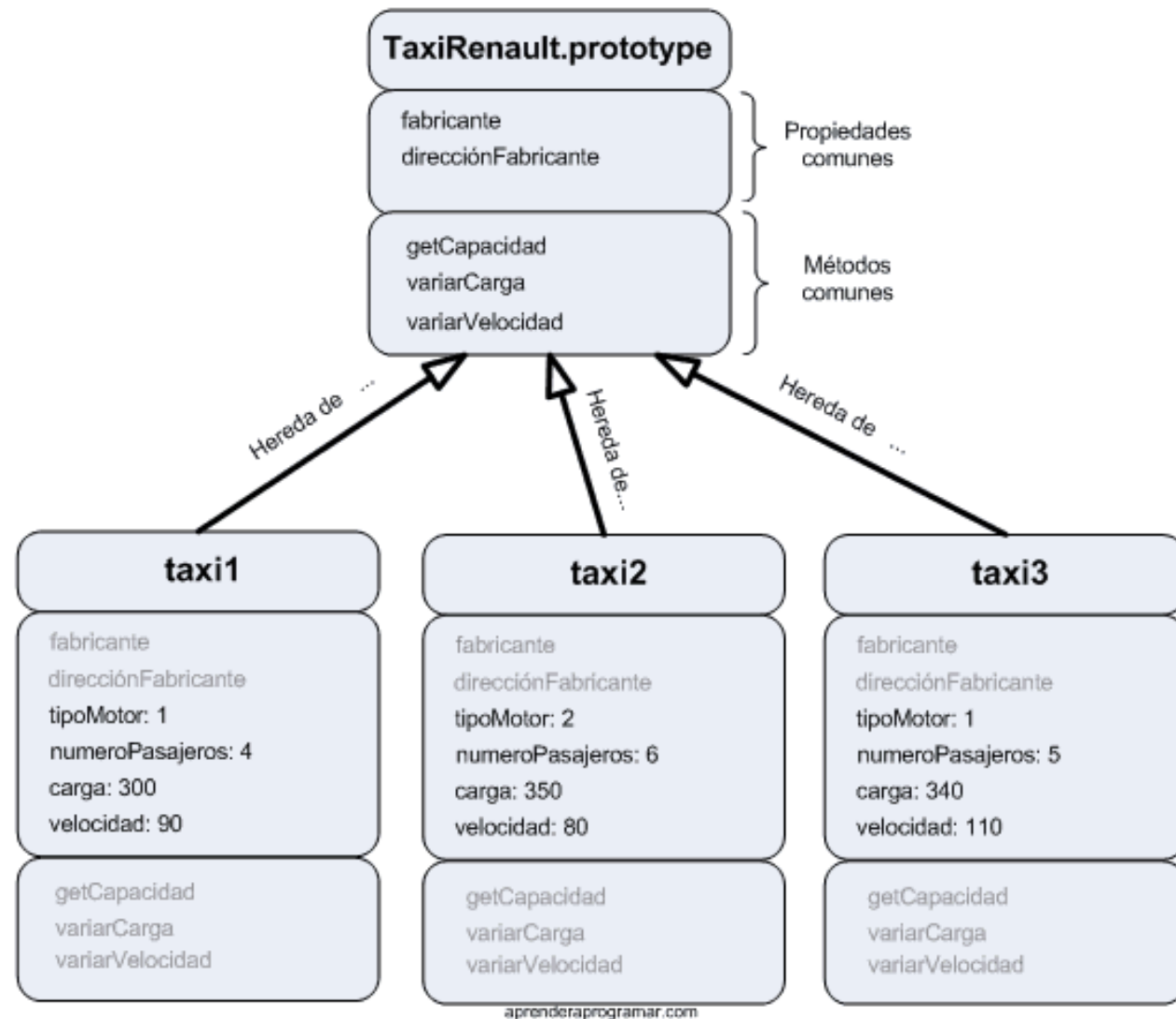
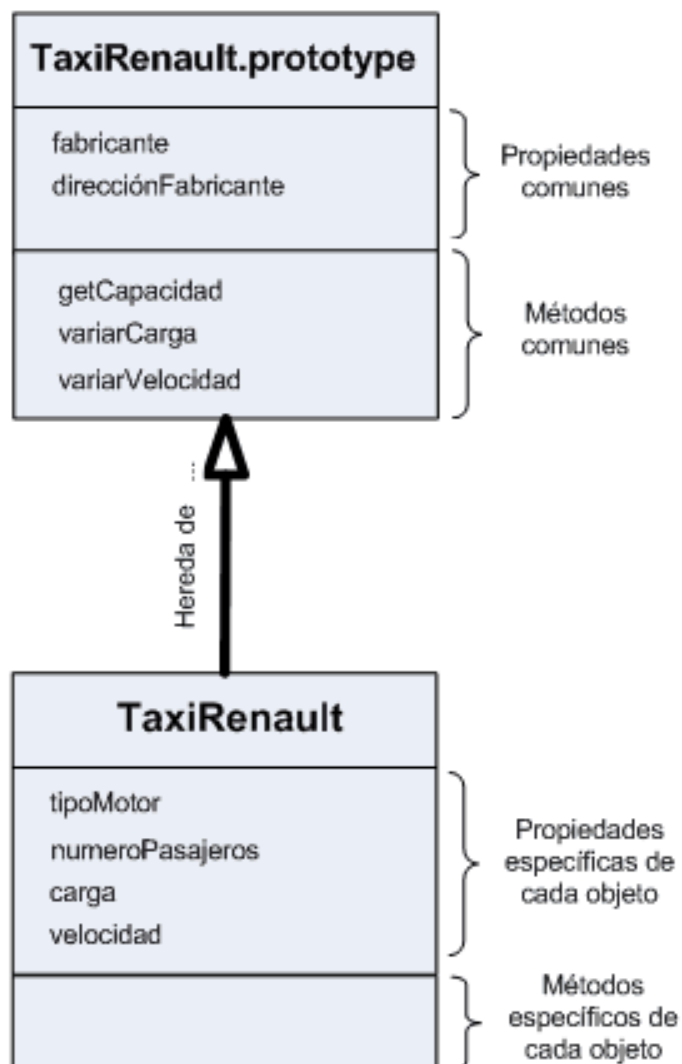
Métodos en funciones constructoras

- Uso de funciones dentro del objeto.
- Definir esas funciones fuera para que no se creen cada vez que se instancie un objeto.
 - `function Persona(nombre,apellidos){`
 - `this.nombre = nombre;`
 - `this.apellidos = apellidos;`
 - `this.nombreCompleto = completar;`
 - `}`
 - `function completar(){`
 - `• alert(this.nombre + " " + this.apellidos);`
 - `}`
- Ejemplo en <https://cursosgs.es/dwec/2017/11/02/2-10-clases-y-objetos-old-style/>

OBJETOS

Prototype

- Las funciones constructoras disponen del método `prototype`.
- `nombreObjeto.prototype` es a su vez un objeto
- Este método aloja toda la información común que comparten los objetos creados con ella (“objetos hijos”) → ahorro de memoria y tiempo de ejecución.
- La propiedad `Object.prototype` representa al objeto prototipo de `Object`.
- Los objetos creados en JS son instancias de `Object` → adquieren automáticamente sus propiedades y métodos
- Esto permite añadir propiedades a nuestros objetos no definidos en la función constructora.
- Posiblemente será relegada al olvido por la llegada de las clases con ES6.



OBJETOS

Atributos y métodos privados

- Sólo pueden ser accedidos desde la función constructora
- Se definen omitiendo `this`.
- Si intentamos usarlos se nos devolverá `undefined`.
- Tampoco prototype nos permitirá acceder a ellos.
- Ejemplo:

```
■ function Animal(n){  
    this.nombre = n; //atributo público.  
    var patas = 4; //atributos privados.  
    var tipo = "cuadrupedo";  
    }  
■ }
```

- <https://cursosgs.es/dwec/2017/11/02/2-11-clases-y-herencia-javascript-prees6/>

OBJETOS

Atributos y métodos estáticos

fuera es estatico. Son atributos que estan afuera de la declaracion del constructor y que no varian (Son fijos)

- Se añaden a la función constructora después de definirla (fuera) ~~(no static)~~
- No son accesibles desde la función constructora (al instanciar)
- No son accesibles a través de instancias creadas con la función constructora
- Ejemplo:

- `var Constructora = function(){`
 `//atributos y métodos NO ESTÁTICOS`
- `}`
- `Constructora.atributoEstatico = 22; //Asignamos atributos y métodos estáticos.`
- `Constructora.metodoEstatico = function(){`
 `console.log("Hola, soy un método estático.");`
- `};`
- `console.log(Constructora.atributoEstatico); // 22`
- `Constructora.metodoEstatico(); // "Hola, soy un método estático."`

OBJETOS

Herencia prototípica

- Definidas dos funciones constructoras, ambas tendrán su objeto prototype accesible mediante Constructora.prototype
- La función constructora tendrá todos los métodos y atributos definidos para ella más todos los métodos y atributos definidos en su prototype.
- Por defecto prototype tiene todos los métodos y atributos de Object.
- Podemos añadirle más asignándole un objeto instanciado con una función constructora:
 - [Ejemplo](#) y ejercicio.
- El método `Object.create()`: permite crear un objeto usando el prototype de otro para que el suyo sea igual.



Es object.create

OBJETOS

Herencia múltiple

- En JS una función constructora SOLO hereda de un padre.
- No obstante es posible conseguir una herencia múltiple:
 - El método `Object.assign()`: se utiliza para copiar los valores de todas las propiedades enumerables de uno o más objetos fuente a un objeto destino
 - El método `Object.call(this)`: permite ejecutar otra función constructora pasándole como parámetro todos los atributos y métodos de la función constructora desde la que se invoca. Los métodos y atributos asignados fuera de la función constructora no estarán disponibles.
- Ejemplo canino: <https://cursosgs.es/dwec/2017/11/02/2-11-clases-y-herencia-javascript-prees6/>

CLASES

- Una clase define cómo es un objeto (una casa).
- Un objeto es la plasmación efectiva de esa clase (mi casa).
- **ES6** y su sintaxis en forma de clases al estilo JAVA han reducido considerablemente el uso de la herencia prototípica. Pero no debemos olvidar que lo que subyace es herencia prototípica.
- **class:** aparece con ES6. Es un tipo de función que:
 - Está reservada para su uso como constructor, no puede llamarse como cualquier función.
 - No admite redeclaraciones posteriores a su definición (similar a una constante).
- Class puede no ser compatible con algunos navegadores → herramientas tipo [babel](#)
- Deben ser siempre declaradas antes de poder instanciar objetos.

CLASES

Sintaxis:

- No precisa de argumentos en su definición (sin paréntesis).
- No necesita un ; final.
- Recomendable que su nombre empiece con mayúscula.
- Sintaxis básica con constructor:
 - `class TarjetaFelicitation {`
 - `constructor(mensaje) {` //se inicializa con un mensaje asociado
 - `this.mensajeTarjeta = mensaje;`
 - `}`
 - `}`

CLASES

Sintaxis

➤ Añadiendo métodos

■ `class TarjetaFelicitation {`

`constructor(mensaje) { //se inicializa con un mensaje asociado`

`this.mensajeTarjeta = mensaje;`

`}`

`personaliza(nombre) { // Recibe el destinatario y devuelve un mensaje personalizado`

`return (nombre + ", tengo este mensaje para ti: " + this.mensajeTarjeta);`

`}`

■ `}`

CLASES

Sintaxis

- Instanciando la clase:
 - `miTarjeta = new TarjetaFelicitacion("Feliz cumple!");` // Instanciamos la tarjeta con un mensaje
- Usando el objeto instanciado:
 - `document.getElementById("miDiv").innerHTML = miTarjeta.personaliza("Carlos");`
// Obtenemos el mensaje personalizado y lo metemos en el HTML de miDiv
 - `document.getElementById("miDiv").innerHTML = miTarjeta.mensajeTarjeta;`
//Obtenemos SOLO el mensaje y lo metemos en el HTML de miDiv

CLASES

Getters y Setters

- En JS son atajos sintácticos sin funcionalidad adicional.

```
■ class TarjetaFelicitation { //codepen
    constructor(mensaje) {      //se inicializa con un mensaje asociado
        this.mensajeTarjeta = mensaje;
    }
    set mensaje(m) { // Cambia el mensaje principal. No puede llamarse igual que el atributo.
        this.mensajeTarjeta = m;
    }
    get mensaje() { // Devuelve el mensaje principal
        Return (this.mensajeTarjeta);
    }
    personaliza(nombre) { // Recibe el destinatario y devuelve un mensaje personalizado
        return (nombre + ", tengo este mensaje para ti: " + this.mensajeTarjeta) ;
    }
}
```