OBJECTS, CLASSES AND INHERITANCE

OBJECTS

CLASSES

HERENCIA

OBJECTS

QUÉ SON

- Todo lo que no son primitivos
- Es una variable compuesta de varias propiedades
- Cada propiedad es otro primitive u otro objeto

OBJECTS

CREACIÓN

- LITERALMENTE

```
let empty = {};
let point = { x: 0, y: 0 };
let p2 = { x: point.x, y: point.y+1 };
let book = {
    "main title": "JavaScript",
    "sub-title": "The Definitive Guide",
    for: "all audiences",
    author: {
        firstname: "David",
        surname: "Flanagan"
    }
};
```

- new()

```
let o = new Object();  // Create an empty object: same as {}.
let a = new Array();   // Create an empty array: same as [].
let d = new Date();    // Create a Date object representing the curr
let r = new Map();     // Create a Map object for key/value mapping
```

OBJECTS

ACCESO Y ASIGNACIÓN

- Diferentes maneras de acceder

```
let author = book.author;
let name = author.surname;
let title = book["main title"];
```

- Diferentes maneras de asignar

```
book.edition = 7;
book["main title"] = "ECMAScript";
```

OBJECTS

GESTIÓN DE ERRORES

- Cuando se accede a una propiedad que no existe -> undefined
- Cuando se accede a una propiedad de undefined -> error

```
> let a = {}
undefined
> a.suerte
undefined
> a.suerte.yepe
Uncaught TypeError: Cannot read properties of undefined (reading 'yepe')
```

OBJECTS

BORRADO DE PROPIEDADES

- DELETE

```
> a.beba = "yepe"
'yepe'
> a.beba
'yepe'
> delete a.beba
true
> a.beba
undefined
```

OBJECTS

EVALUACIÓN DE PROPIEDADES

- Evaluación de keys

```
> a = {"suerte": 3}
{ suerte: 3 }
> a.suerte
3
> "suerte" in a
true
> "yepe" in a
false
```

- **hasOwnProperty():** true solo si es una propiedad suya pero false si no existe o es heredada

OBJECTS

RECORRIDO DE PROPIEDADES

- Bucle for para acceder a las keys

```
> for (let p in a){console.log(p);}
suerte
doblesuerte
```

- **.keys():** para acceder a las keys en forma de lista

OBJECTS

COPIA DE PROPIEDADES

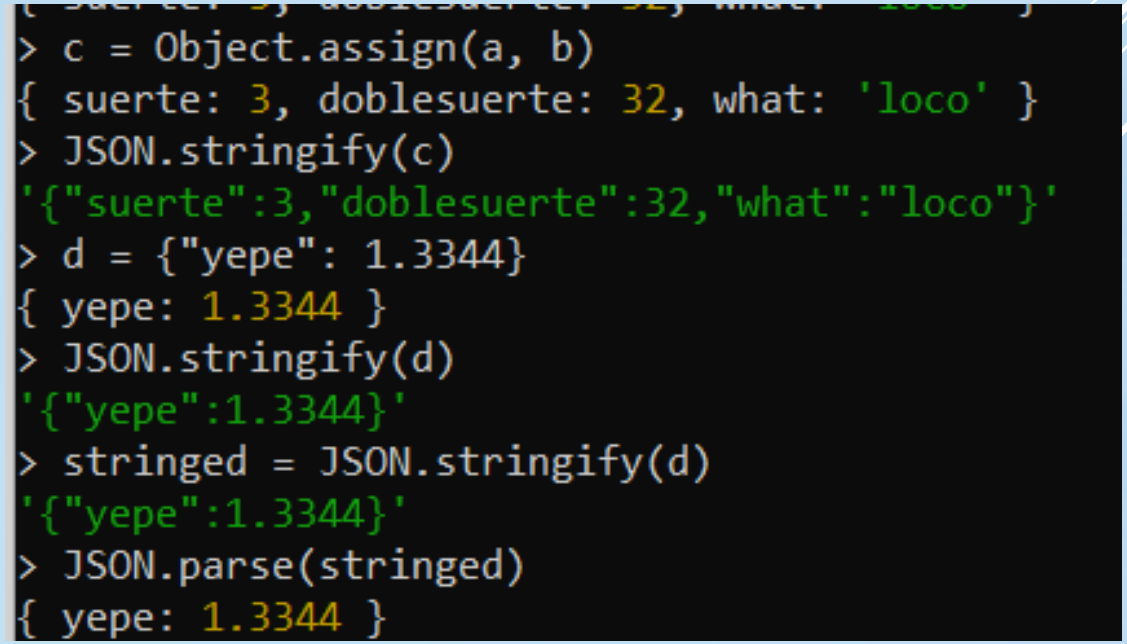
- **assign:** extiende un objeto sobre otro o lo copia

```
> Object.assign(a, b)
{ suerte: 3, doblesuerte: 32, what: 'loco' }
```

OBJECTS

SERIALIZADO DE PROPIEDADES

- **Streaming:** se convierte a string para poder enviarlo



CLASSES

DEFINICIÓN

- Las clases son objetos que comparten propiedades entre ellas, de tal manera que puede ser útil juntarlas.
- Se componen de miembros o instancias
- Variables que comparten variables y métodos.
- Basadas en prototipos

CLASSES

EJEMPLO ANTIGUO

```
// This is a constructor function that initializes new Range objects.
// Note that it does not create or return the object. It just initializes this.
function Range(from, to) {
  // Store the start and end points (state) of this new range object.
  // These are noninherited properties that are unique to this object.
  this.from = from;
  this.to = to;
}

// All Range objects inherit from this object.
// Note that the property name must be "prototype" for this to work.
Range.prototype = {
  // Return true if x is in the range, false otherwise
  // This method works for textual and Date ranges as well as numeric.
  includes: function(x) { return this.from <= x && x <= this.to; },

  // A generator function that makes instances of the class iterable.
  // Note that it only works for numeric ranges.
  [Symbol.iterator]: function*() {
    for(let x = Math.ceil(this.from); x <= this.to; x++) yield x;
  },

  // Return a string representation of the range
  toString: function() { return "(" + this.from + "..." + this.to + ")"; }
};

// Here are example uses of this new Range class
let r = new Range(1,3);   // Create a Range object; note the use of new
r.includes(2)          // => true: 2 is in the range
r.toString()          // => "(1...3)"
```

CLASSES

EJEMPLO class

```
class Range {
    constructor(from, to) {
        // Store the start and end points (state) of this new range c
        // These are noninherited properties that are unique to this
        this.from = from;
        this.to = to;
    }

    // Return true if x is in the range, false otherwise
    // This method works for textual and Date ranges as well as numer
    includes(x) { return this.from <= x && x <= this.to; }

    // A generator function that makes instances of the class iterabl
    // Note that it only works for numeric ranges.
    *[Symbol.iterator]() {
        for(let x = Math.ceil(this.from); x <= this.to; x++) yield x;
    }

    // Return a string representation of the range
    toString() { return `(${this.from}...${this.to})`; }
}
```

CLASSES

OTROS

- **instanceof:** comprueba si un objeto es de una clase

```
> {a: "suerte"} instanceof Object
true
```

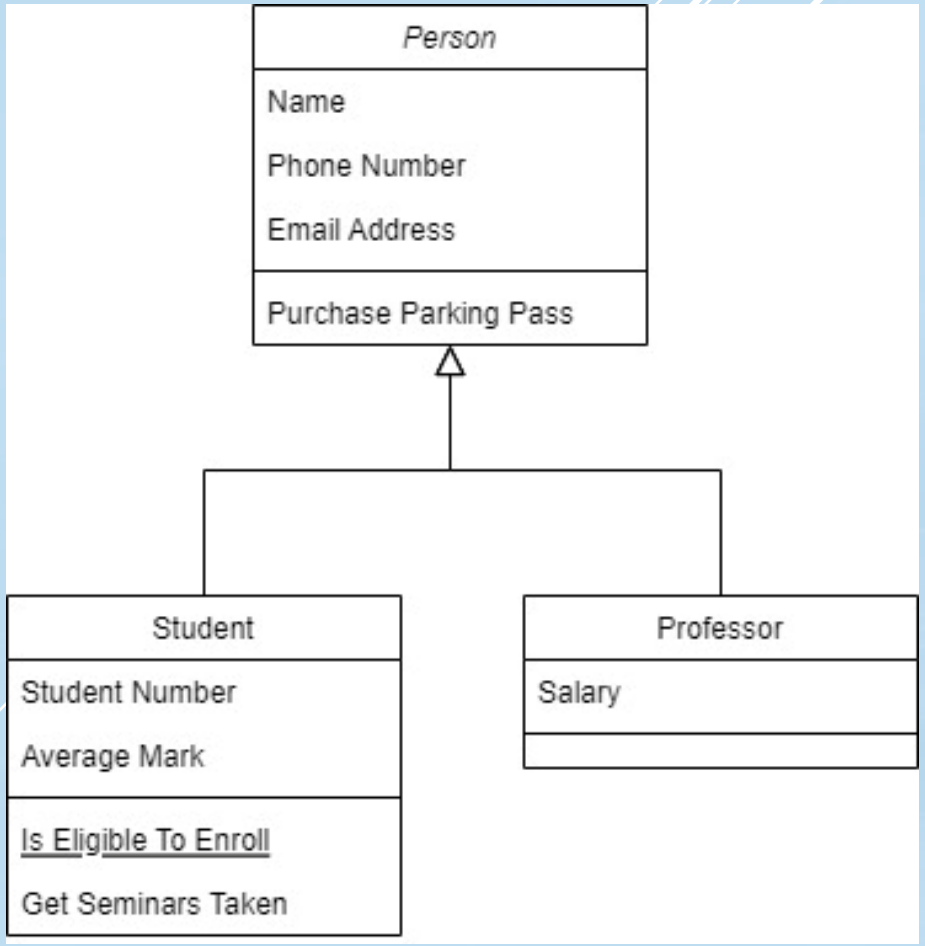
- Añadir una función o propiedad sobre una clase existente

```
Number.prototype.times = function(f, context) {
    let n = this.valueOf();
    for(let i = 0; i < n; i++) f.call(context, i);
};
```

HERENCIA

QUÉ ES

- Una nueva clase que se basa en otra hereda sus mismas funciones y
- Propiedades
- Define y/o sustituye (override) las que ya existen



HERENCIA

EJEMPLO

```
// A trivial Array subclass that adds getters for the first and last
class EZArray extends Array {
    get first() { return this[0]; }
    get last() { return this[this.length-1]; }
}

let a = new EZArray();
a instanceof EZArray  // => true: a is subclass instance
a instanceof Array    // => true: a is also a superclass instance.
a.push(1,2,3,4);      // a.length == 4; we can use inherited methods
a.pop()               // => 4: another inherited method
a.first               // => 1: first getter defined by subclass
a.last                // => 3: last getter defined by subclass
a[1]                  // => 2: regular array access syntax still work
Array.isArray(a)      // => true: subclass instance really is an arra
EZArray.isArray(a)    // => true: subclass inherits static methods, t
```

HERENCIA

COMPOSICIÓN FRENTE A HERENCIA

- Creación de miles de clases
- No siempre hay que generar clases porque sí
- Existe la composición: clases que usan clases, sin necesidad de heredar

HERENCIA

EJEMPLO

```
class Histogram {
    // To initialize, we just create a Map object to delegate to
    constructor() { this.map = new Map(); }

    // For any given key, the count is the value in the Map, or zero
    // if the key does not appear in the Map.
    count(key) { return this.map.get(key) || 0; }

    // The Set-like method has() returns true if the count is non-ze
    has(key) { return this.count(key) > 0; }

    // The size of the histogram is just the number of entries in th
    get size() { return this.map.size; }
```

HERENCIA

JERARQUÍA DE VARIOS NIVELES

- Clases abuelas (abstractas), padres, hijas…

BIBLIOGRAFÍA

1. JavaScript: The Definitive Guide, 7th Edition
2. Eloquent JavaScript, 3th edition, Marijn Haverbeke
3. Clean Code: A Handbook of Agile Software Craftsmanship (Robert C. Martin Series)
4. Modern c++ Programming with Test-Driven Development, Jeff Langr
5. Refactoring: Improve the design of existing Code, Martin Fowler
6. Game programming patterns, Robert Nystrom