# ES6: clases

Francisco Arce
www.pacoarce.com

# Class

- Ahora podemos hacer clases por medio de la sentencia “class”.
- Podemos utilizar el método “ constructor()” para crear la función constructora.
- Los lenguajes tradicionales basados en clases ofrecen la palabra reservada ***this*** para referencia la instancia actual de la clase.
- En Javascript ***this*** se refiere ***al contexto de la llamada*** y como tal puede ser cambiado a algo más que un objeto.

Francisco Arce
www.pacoarce.com

# ES6: instancias

Francisco Arce
www.pacoarce.com

# Objeto

- Un objeto es una instancia de la clase, la cual es creada usando el operador ***new***.
- Cuando se usa un punto para acceder al método del objeto, ***this*** se va a referir al objeto a la izquierda al punto.

Francisco Arce
www.pacoarce.com

# Objeto

let burger = new Hamburger();

burger.listToppings();

- En este código vemos que cuando ***this*** es usada desde adentro de la clase Hamburger, Se va a referir al objeto ***burger***.

# ES6: herencia

Francisco Arce
www.pacoarce.com

# Herencia

- Al igual que en otros lenguajes de programación, una clase puede extender otra clase heredando métodos o propiedades de la clase padre.

# Herencia

- La función ***super()*** ejecuta el método con el mismo nombre desde el que se está llamando a ***super()***, de esta forma al definir el nuevo constructor llamamos a ***super()*** y le pasamos los mismos parámetros que recibe el constructor, entonces se ejecuta ese constructor y luego código del nuevo.

# ES6: Getters y Setters

Francisco Arce
www.pacoarce.com

# Getters y Setters

- En algunos lenguajes de programación (como Java) existen los ***getters*** y ***setters***.
- Estos métodos que se usan para controlar variables internas de un objeto (propiedades).
- Para usarlos simplemente se agrega ***get*** o ***set*** delante del nombre del método de la siguiente forma:

Francisco Arce
www.pacoarce.com

# Getters y Setters

- Definir un método ***get*** con el nombre que quieras (no puede ser el nombre de la propiedad) y este debería devolver el valor deseado (técnicamente puede hacer cualquier cosa el método), o defines un método ***set*** con otro nombre (tampoco el mismo de la propiedad) y que recibe el nuevo valor y lo asigna a ***this*** .

# Getters y Setters

- Aunque esto hace bastante más legible y limpio el código, al tener métodos específicos para obtener o modificar propiedades del objeto, la verdad es que no son necesarios ya que simplemente usando la sintaxis de objetos de toda la vida puedes obtener el valor de una propiedad y modificarlo.

# ES6: Métodos estáticos

Francisco Arce
www.pacoarce.com

# Métodos estáticos

- Al igual que en otros lenguajes también va a ser posible crear métodos estáticos usando la palabra clave ***static*** antes del nombre del método.

```
class miClase {
  static miMetodo() {
    return 'hola mundo'
  }
}
```

# Métodos estáticos

- Luego para poder usarlo simplemente llamas al método desde la clase sin instanciar:

```
let mensaje = miClase.miMetodo(); // 'hola mundo';
```

# ES6: Características

Francisco Arce
www.pacoarce.com

# Características

- Los nombres de las clases no pueden ser ***eval*** ó ***arguments***;
- No están permitidos nombres de clase repetidos.
- El nombre constructor solo puede ser usado para métodos, no para ***getters***, ***setter*** o un generador de métodos
- Las clases no se pueden llamar antes de definirse.
- Todavía se puede instanciar la clase desde cualquier parte, solo es necesario esperar a que esté definida.

# ES6

## Class Inheritance, From Expressions

Francisco Arce
www.pacoarce.com

# Class Inheritance, From Expressions

- Class Inheritance, From Expressions
- Support for mixin-style inheritance by extending from expressions yielding function objects. [Notice: the generic aggregation function is usually provided by a library like this one, of course]
-

# Class Inheritance, From Expressions

```
var aggregation = (baseClass, ...mixins) => {
    let base = class _Combined extends baseClass {
        constructor (...args) {
            super(...args)
            mixins.forEach((mixin) => {
                mixin.prototype.initializer.call(this)
            })
        }
    }
    let copyProps = (target, source) => {
        Object.getOwnPropertyNames(source)
```

Francisco Arce
www.pacoarce.com

# Class Inheritance, From Expressions

Base Class Access

Intuitive access to base class constructor and methods.

```
class Shape {
    ...
    toString () {
        return `Shape(${this.id})`
    }
}
class Rectangle extends Shape {
    constructor (id, x, y, width, height) {
```

Francisco Arce
www.pacoarce.com

# Class Inheritance, From Expressions

Static Members
Simple support for static class members.

```
class Rectangle extends Shape {
    …
    static defaultRectangle () {
        return new Rectangle("default", 0, 0, 100, 100)
    }
}
class Circle extends Shape {
    …
```

Francisco Arce
www.pacoarce.com

# Class Inheritance, From Expressions

Getter/Setter also directly within classes (and not just within object initializers, as it is possible since ECMAScript 5.1).

```
class Rectangle {
    constructor (width, height) {
        this._width  = width
        this._height = height
    }
    set width  (width)  { this._width = width           }
    get width  ()       { return this._width            }
    set height (height) { this._height = height         }
```

Francisco Arce
www.pacoarce.com

# Referencias

https://medium.com/@lehiarteaga/ecmascript-6-es6-y-sus-caracter%C3%ADsticas-55a1fc9275b1
http://www.enrique7mc.com/2015/12/novedades-de-es6/
https://platzi.com/blog/ecmascript-nueva-sintaxis/
http://es6-features.org/#ClassInheritanceFromExpressions