ES6 classes

JS - Prototype Constructor

- Objects are dynamic bags, containing properties.

- Objects can inherit properties from other objects.

- All objects inherit properties and methods from a prototype.

- When searching for a property on an object, JS looks within the object first,
 and if not found it looks on its prototype or prototype chain, until found.

- Each object has a private property **__proto__** which is a link to another object (it's prototype).

- Prototype may have it's own prototype as well, and so on until we reach **Object** constructor.

- **Object** constructor has **null** as its prototype. **null** acts as a final link because it has no prototype. This is where the *prototype chain* ends/starts.

.__proto__ and .prototype

```
const obj = {}

obj.__proto__;    // Link to the prototype
obj.__proto__.constructor;  // points to the prototype object




obj.__proto__ ===  Object.prototype;    // true
obj.__proto__.constructor === Object;         // true
```

ES5 - Functional Prototype Constructor

<u>DON'T HAVE</u> PROTOTYPE INHERITANCE

```
function Car (brand) {
        var result  = {};          // create a instance object
        result.brand = brand;       // create property on the instance and assign it a passed value

        Object.assign( result, carMethods) ;      // assign the prototype methods
        return result;
}


// Object containing methods for the Car prototype/blueprint
var  carMethods = {}
carMethods.start = function () {
        console.log( “Engine start” );
};


var  toyota = Car (“Toyota”);        // {brand: "Toyota", start: ƒ}
```

ES5 - Pseudo Classical Prototype Constructor

HAVE PROTOTYPE INHERITANCE

```
function Car (brand) {
      // var this = Object.create (Car.prototype)
                    // this = {};
                    // this.__proto__ = Car.prototype;
      this.brand = brand;      // create property on the new instance
      // return  this;
}

Car.prototype.start = function () {
      console.log("Engine start");
};
var bmw = new Car ("BMW");     // Car {brand: "BMW", start: ƒ }
```

```
function HybridCar (brand, color) {
      // var this = Object.create (HybridCar.prototype)
      Car.call = (this, brand);      // create brand property as defined in Car function by binding context
      this.brand = brand;      // create property on the new instance
      // return  this;
}

HybridCar.prototype = Object.create(Car.prototype); // Extend the HybridCar and connect it to theCar.prototype
HybridCar.prototype.constructor = Car;//  point the prototype.constructor to theCar.prototype
HybridCar.prototype.description = function () {
            console.log(`${brand} - ${color} color`);
      }

var bmw = new HybridCar ("BMW", "black");  // HybridCar {brand: "BMW", color: "black", start: ƒ, description: ƒ}
```

ES6 - Classes

HAVE PROTOTYPE INHERITANCE

```
class Car {
      constructor (brand) {
            this.brand = brand;  // create property on the new instance
      }
      start () {  // create method on the Prototype
            console.log("Engine start");
      }
}

var bmw = new Car ("BMW");     // Car {brand: "BMW", start: ƒ }
```

```
class HybridCar extends Car {
      constructor (brand, color) {
            super(brand);     // create property `brand` on the new instance as defined in `Car`
            this.color = color;  // create property on the new instance
      }
      description () {
            console.log(`${brand} - ${color} color`);
      }
      static className () { return 'HybridCar'}
}

var bmwHybrid = new HybridCar ("BMW", "black");  // HybridCar {brand: "BMW", start: ƒ, description: ƒ }
```

ES6 - Classes

- Are a syntactic sugar, that gives us a cleaner syntax for creating objects with prototypes.

- ES6 classes are ‘**Special functions**’ used to create object instances.

- ES6 classes are not hoisted. Ensure to defined the constructor before calling it.

- ES6 classes allow us to declare static methods, available only within the constructor.