Classes and Modules in ECMA6

Objectives

- ECMAScript 6
- Browser Support
- Writing classes
- Instantiating classes
- Inheritance
- Defining Modules

What is ECMAScript

- The ***JavaScript programming language*** is standardized by ***ECMA*** (a standards body like W3C) under the name ***ECMAScript***.

- Among other things, ECMAScript defines:

  - Language syntax – parsing rules, keywords, statements, declarations, operators, etc.
  - Types – boolean, number, string, object, etc.
  - Prototypes and inheritance
  - The standard library of built-in objects and functions – JSON, Math, Array methods, Object introspection methods, etc.

ECMAScript 6

The previous editions of the ECMAScript standard were numbered 1, 2, 3, and 5

ES6 is different.ES6 will change the way we write JS code

- Let,const …
- Classes
- Modules
- Lambdas

& many more

Browser Support

- The following Web page shows current browser support
  - https://kangax.github.io/compat-table/es6/
- In summary, these features **will work** in current versions of
  - Chrome / Firefox / Safari / Edge
- These features will **not work** in
  - Internet Explorer 11

Classes

- Classes are "special functions“

- function declarations are hoisted and class declarations are not. We first need to declare a class and then access it

- Class declaration in next slide.

Defining Classes

- Classes defined in a similar way to Java/C#

```
class Car {

        constructor(make, model) {
            this.make = make;
            this.model = model;
            this.speed = 0;
        }

        accelerate(){
            // must use the this keyword to access the property
            this.speed++;
        }

}
```

Constructors:

- The constructor method is a special method for creating and initializing an object created with a class.

- There can only be one special method with the name "constructor" in a class.

- A Syntax Error will be thrown if the class contains more than one occurrence of a constructor method.

Creating Instances

- Instances are created using the **new** keyword
- Properties can be accessed in the same way as ECMA 5

```
class Car {
    …
}

var car = new Car("BMW", "5 series");
car.accelerate();
console.log(car.speed);
```

Getters and Setters

- JavaScript also has the ability to define properties a bit like C#
  - The constructor sets a field with an **_** in front of the name
  - The **get** and **set** keywords are used by methods with the chosen property name
  - The property is then accessed using the name of the get/set functions

```
class Dog {
        constructor(name) {
            this._name = name;
        }

        get name() {
            return this._name;
        }

        set name(newName){
            if (newName) {
                this._name = newName;
            }
        }
    }

    var doggie = new Dog("Fido");
    console.log(doggie.name);
    doggie.name = “Barnie”;
```

Static methods

- The static keyword defines a static method for a class.

- Static methods are called without instantiating their class and cannot be called through a class instance.

- Static methods are often used to create utility functions for an application.

Static Methods

- Classes can have static methods
    - Static properties are not possible

```
class Car {

        static bogStandardCar() {
            return new Car("Ford", "Fiesta");
        }

        constructor(make, model) {
            this.make = make;
            this.model = model;
            this.speed = 0;
}
var ordinaryCar = Car.bogStandardCar();
```

Inheritance

- Inheritance is very similar to Java/C#

```
class SportsCar extends Car {
            constructor(make, model, turboBoost) {
                // call superclass constructor
                super(make,model);
                this.turboBoost = turboBoost;
            }
            // method overriding
            accelerate() {
                super.accelerate(); // call superclass method
                this.speed = this.speed * this.turboBoost;
            }
}

sportsCar = new SportsCar("Maserati", "4200", 4);
sportsCar.accelerate();
console.log(sportsCar.speed);
```

Defining Modules

- There are a number of ways of defining a module
  - Exporting specific functions and variables in a JavaScript file
  - Exporting all functions and variables in a JavaScript file
  - Export a mixure of Functions and variables from multiple files from one file - mixins

Exporting Examples

```
// exporting specific functions
export function makePayment(amount) {
    return "payment made for " + amount;
}

export function issueRefund(amount) {
    console.log("refund issued for " +
amount);
}
```

```
// exporting one thing
class Account {
}

// more things in the file not exported

export default Account
```

Summary

- Writing classes
- Instantiating classes
- Inheritance
- Defining Modules