Instructor Notes:

Add instructor notes here.

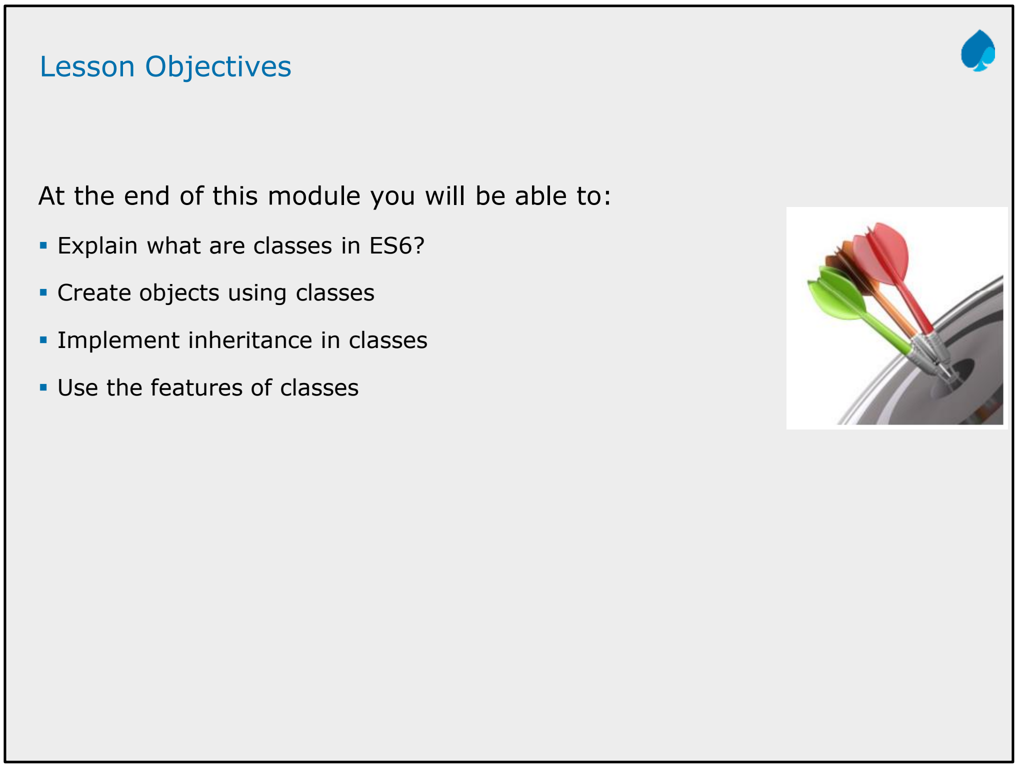
JavaScript ES6

Lesson: 15
Working with Classes

Capgemini

Instructor Notes:

Add instructor notes here.



Lesson Objectives

At the end of this module you will be able to:

- Explain what are classes in ES6?
- Create objects using classes
- Implement inheritance in classes
- Use the features of classes

Instructor Notes:

Add instructor notes here.

ES6 – Class Introduction

Although it's an object-oriented programming language; JavaScript never had the concept of classes.

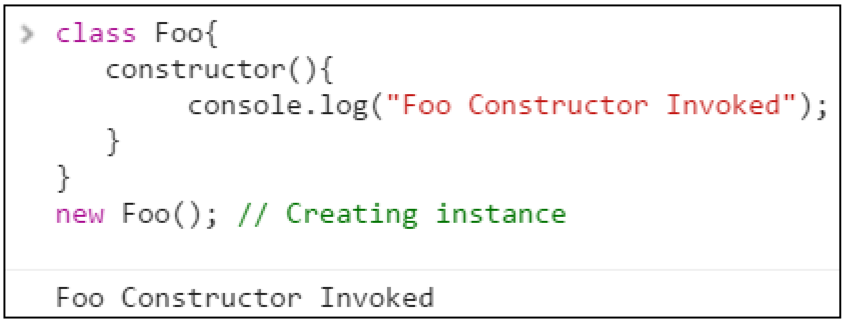
Programmers from the other programming language background often found it difficult to understand JavaScript's object-oriented model and inheritance due to lack of classes.

ES6 introduced classes that provide a much simpler and clearer syntax to creating constructors and dealing with inheritance.

There can only be one constructor method in a class. Defining more than one constructor will throw the SyntaxError exception.

The constructor method, by default, returns the new instance if there is no return statement in it. If there is a return statement, then whatever is the value in the return statement is returned.

Function calls can be made above the function definition. But in case of class, you cannot use a class before its defined. Trying to do so in classes will throw the ReferenceError exception. i.e. Classes are not hoisted.



```
> class Foo{
     constructor(){
          console.log("Foo Constructor Invoked");
     }
  }
  new Foo(); // Creating instance

  Foo Constructor Invoked
```

Instructor Notes:

Add instructor notes here.

Using Classes

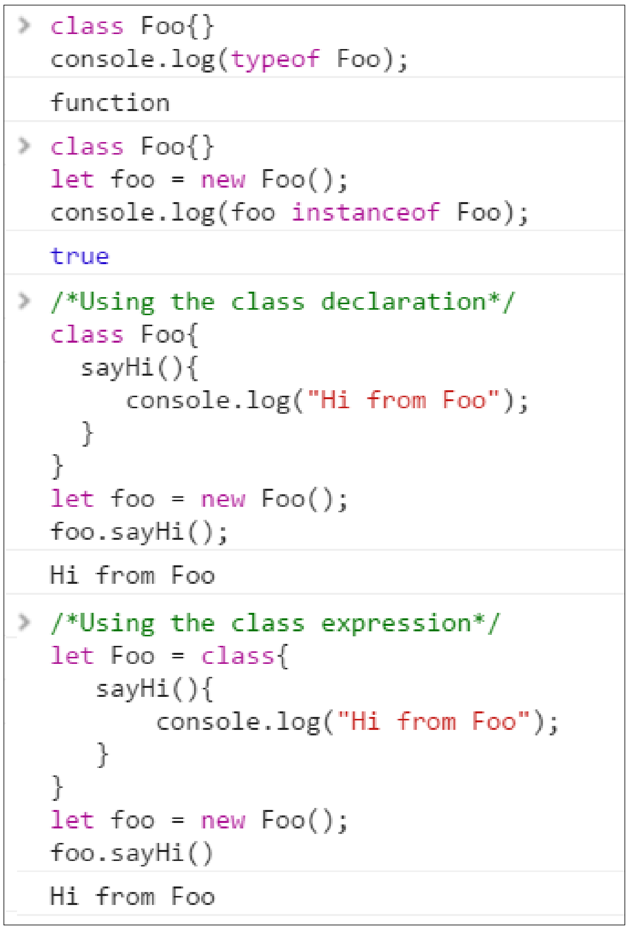
ES6 classes aim to provide a much simpler and clearer syntax for dealing with the constructors and inheritance.

Classes are just a new syntax for creating functions that are used as constructors. In fact, classes are functions.

There are two ways to define a class

- **Using the class declaration** : For defining a class using the class declaration required the class keyword, and a name for the class
- **Using the class expression** : A class expression has a similar syntax to a class declaration. However, with class expressions, class name can be omitted.

ES6 class doesn't pollute the global namespace. i.e. it wont get attached to window object and classes are not hoisted.



```
> class Foo{}
  console.log(typeof Foo);
  function
> class Foo{}
  let foo = new Foo();
  console.log(foo instanceof Foo);
  true
> /*Using the class declaration*/
  class Foo{
    sayHi(){
       console.log("Hi from Foo");
    }
  }
  let foo = new Foo();
  foo.sayHi();
  Hi from Foo
> /*Using the class expression*/
  let Foo = class{
     sayHi(){
         console.log("Hi from Foo");
     }
  }
  let foo = new Foo();
  foo.sayHi()
  Hi from Foo
```

Instructor Notes:

Add instructor notes here.

Demo

Using-Classes

Instructor Notes:

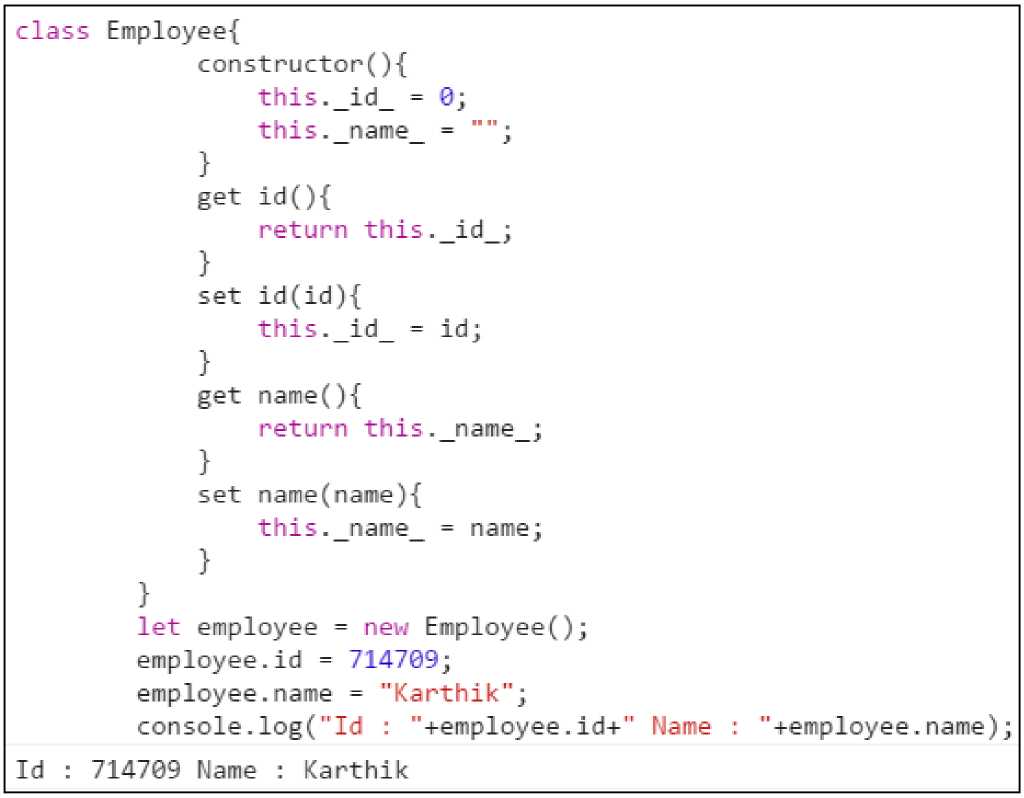
Add instructor notes here.

The get and set methods

Class instance members / fields can be initialized inside the constructor using ***this*** keyword.

ES6 introduced the get and set prefixes for methods, which is used to encapsulate the instance members.

When **get** and **set** methods are used in a class body, they are added to the **prototype** property of the class.



```
class Employee{
            constructor(){
                this._id_ = 0;
                this._name_ = "";
            }
            get id(){
                return this._id_;
            }
            set id(id){
                this._id_ = id;
            }
            get name(){
                return this._name_;
            }
            set name(name){
                this._name_ = name;
            }
        }
        let employee = new Employee();
        employee.id = 714709;
        employee.name = "Karthik";
        console.log("Id : "+employee.id+" Name : "+employee.name);
```

```
Id : 714709 Name : Karthik
```

Instructor Notes:

Add instructor notes here.

Demo

get-set-methods

Instructor Notes:

Add instructor notes here.

The static method

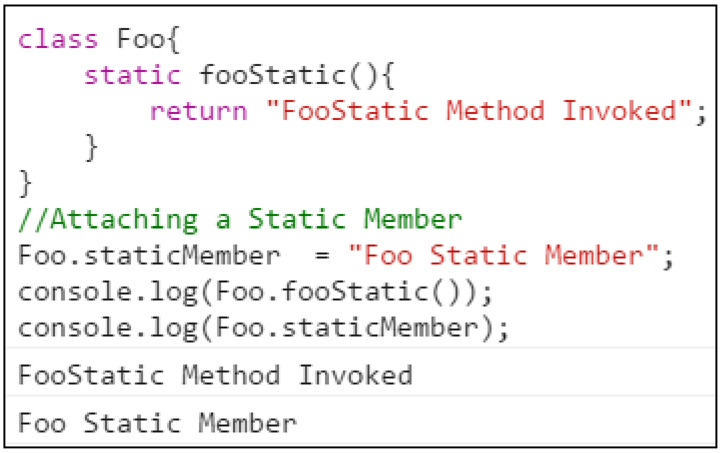
The methods that are added to the body of the class with the static prefix are called as static methods.

Static methods are the own methods of the class, i.e. they are not added to the prototype property of the class, rather they are added to the class itself.

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

Static methods should be access through its class name.

At present we can create only static methods in order to create static members of the class it need to attached to it's class name outside the class.



```
class Foo{
    static fooStatic(){
        return "FooStatic Method Invoked";
    }
}
//Attaching a Static Member
Foo.staticMember  = "Foo Static Member";
console.log(Foo.fooStatic());
console.log(Foo.staticMember);
```

```
FooStatic Method Invoked
Foo Static Member
```

Instructor Notes:

Add instructor notes here.

Demo

static-method

Instructor Notes:

Add instructor notes here.

Implementing inheritance in classes

ES6 implement the inheritance hierarchy in the classes with the help of extends clause, and the super keyword.

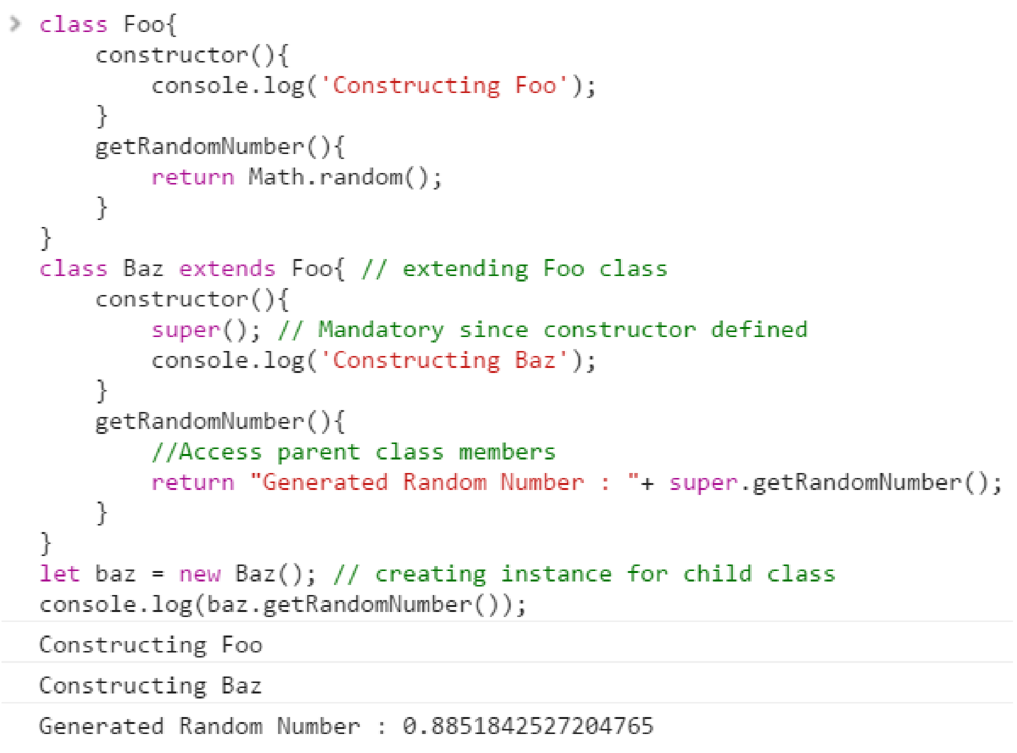
By using the extends clause, a class can inherit static and non-static properties of another constructor(which may or may not be defined using a class).

The super keyword is used in two ways:

- It's used in a class constructor method to call the parent constructor;
- When used inside methods of a class, it references the static and non-static methods of the parent constructor
- super keyword can be used with object literal as well.

If constructor is defined in child class, then super() need to be compulsorily provided even though constructor is not defined in parent class.

If a child class doesn't have a constructor method, then the default behavior will invoke the constructor method of the parent class.



```
> class Foo{
      constructor(){
          console.log('Constructing Foo');
      }
      getRandomNumber(){
          return Math.random();
      }
  }
  class Baz extends Foo{ // extending Foo class
      constructor(){
          super(); // Mandatory since constructor defined
          console.log('Constructing Baz');
      }
      getRandomNumber(){
          //Access parent class members
          return "Generated Random Number : "+ super.getRandomNumber();
      }
  }
  let baz = new Baz(); // creating instance for child class
  console.log(baz.getRandomNumber());
  Constructing Foo
  Constructing Baz
  Generated Random Number : 0.8851842527204765
```

Instructor Notes:

Add instructor notes here.

Demo

inheritance

Instructor Notes:

Add instructor notes here.

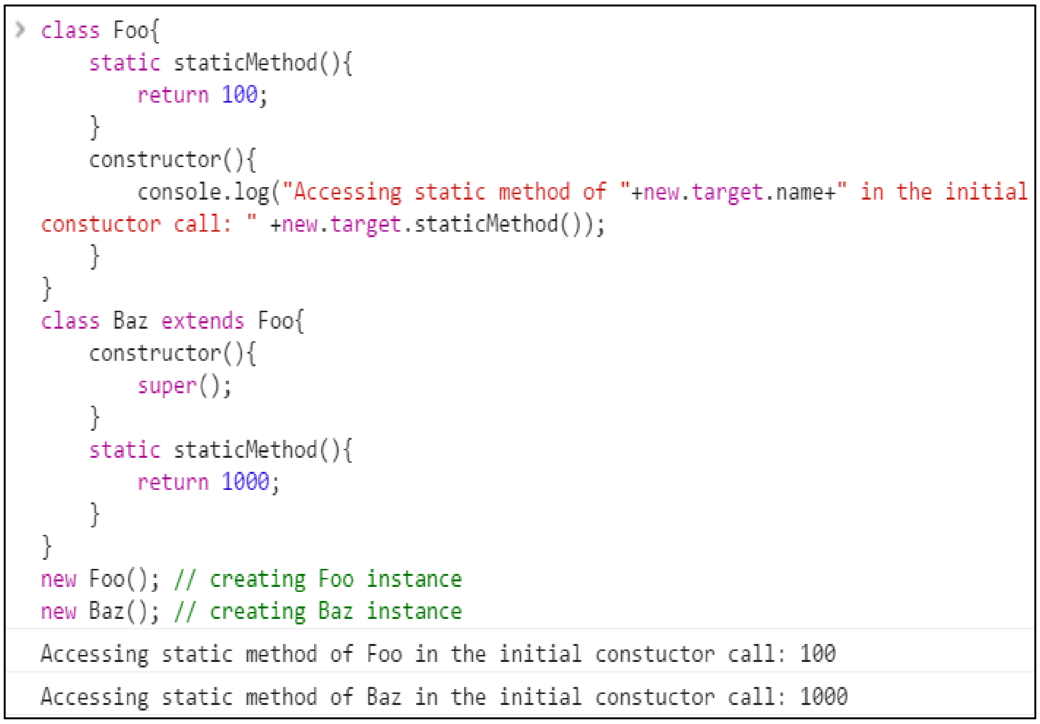
The "new.target" implicit parameter

ES6 adds a parameter named new.target to all the functions.

The default value of new.target is undefined, but when a function is invoked as a constructor, the value of the new.target parameter depends on the following conditions

- If a constructor is invoked using a new operator, then new.target points to this constructor
- If a constructor is invoked via super keyword, then the value of new.target in it is the same as the value of new.target of the constructor that is called super.

Using new.target static methods can be accessed in the initial constructor call even though it is a part of the prototype chain.



```
class Foo{
    static staticMethod(){
        return 100;
    }
    constructor(){
        console.log("Accessing static method of "+new.target.name+" in the initial
constuctor call: " +new.target.staticMethod());
    }
}
class Baz extends Foo{
    constructor(){
        super();
    }
    static staticMethod(){
        return 1000;
    }
}
new Foo(); // creating Foo instance
new Baz(); // creating Baz instance
Accessing static method of Foo in the initial constuctor call: 100
Accessing static method of Baz in the initial constuctor call: 1000
```

Instructor Notes:

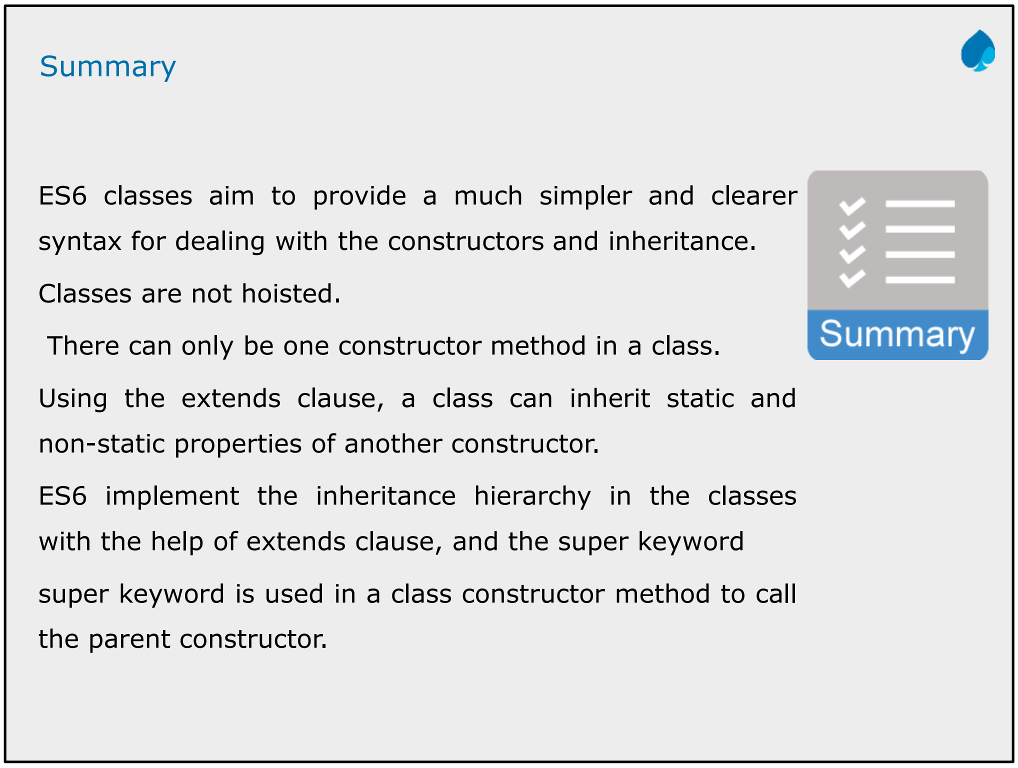
Add instructor notes here.

Demo

new.target

Instructor Notes:

Add instructor notes here.



Summary

ES6 classes aim to provide a much simpler and clearer syntax for dealing with the constructors and inheritance.

Classes are not hoisted.

There can only be one constructor method in a class.

Using the extends clause, a class can inherit static and non-static properties of another constructor.

ES6 implement the inheritance hierarchy in the classes with the help of extends clause, and the super keyword

super keyword is used in a class constructor method to call the parent constructor.