# JavaScript Inheritance Using extends & super

**Summary**: in this tutorial, you will learn how to implement JavaScript inheritance by using `extends` and `super` in ES6.

## Implementing JavaScript inheritance using extends and super

Prior to ES6, implementing a proper inheritance required multiple steps. One of the most commonly used strategies is prototypal inheritance.

The following illustrates how the `Bird` inherits properties from the `Animal` using the prototypal inheritance technique:

```
function Animal(legs) {
    this.legs = legs;
}

Animal.prototype.walk = function() {
    console.log('walking on ' + this.legs + ' legs');
}

function Bird(legs) {
    Animal.call(this, legs);
}

Bird.prototype = Object.create(Animal.prototype);
Bird.prototype.constructor = Animal;


Bird.prototype.fly = function() {
    console.log('flying');
}

var pigeon = new Bird(2);
```

```
pigeon.walk(); // walking on 2 legs
pigeon.fly();  // flying
```

ES6 simplified these steps by using the `extends` and `super` keywords.

The following example defines the `Animal` and `Bird` classes and establishes the inheritance through the `extends` and `super` keywords.

```
class Animal {
    constructor(legs) {
        this.legs = legs;
    }
    walk() {
        console.log('walking on ' + this.legs + ' legs');
    }
}

class Bird extends Animal {
    constructor(legs) {
        super(legs);
    }
    fly() {
        console.log('flying');
    }
}


let bird = new Bird(2);

bird.walk();
bird.fly();
```

How it works.

First, use the `extends` keyword to make the `Bird` class inheriting from the `Animal` class:

```
class Bird extends Animal {
   // ...
```

```
}
```

The `Animal` class is called a **base class** or **parent class** while the `Bird` class is known as a **derived class** or **child class**. By doing this, the `Bird` class inherits all methods and properties of the `Animal` class.

Second, in the `Bird` 's constructor, call `super()` to invoke the `Animal` 's constructor with the `legs` argument.

JavaScript requires the child class to call `super()` if it has a constructor. As you can see in the `Bird` class, the `super(legs)` is equivalent to the following statement in ES5:

```
Animal.call(this, legs);
```

If the `Bird` class doesn't have a constructor, you don't need to do anything else:

```
class Bird extends Animal {
    fly() {
        console.log('flying');
    }
}
```

It is equivalent to the following class:

```
class Bird extends Animal {
    constructor(...args) {
        super(...args);
    }
    fly() {
        console.log('flying');
    }
}
```

However, the child class has a constructor, it needs to call `super()` . For example, the following code results in an error:

```
class Bird extends Animal {
    constructor(legs) {
    }
    fly() {
        console.log('flying');
    }
}
```

Error:

```
ReferenceError: Must call super constructor in derived class before accessing 'this' or retur
```

Because the `super()` initializes the `this` object, you need to call the `super()` before accessing the `this` object. Trying to access `this` before calling `super()` also results in an error.

For example, if you want to initialize the `color` property of the `Bird` class, you can do it as follows:

```
class Bird extends Animal {
        constructor(legs, color) {
                super(legs);
                this.color = color;
        }
        fly() {
                console.log("flying");
        }
        getColor() {
                return this.color;
        }
}

let pegion = new Bird(2, "white");
console.log(pegion.getColor());
```

## Shadowing methods

ES6 allows the child class and parent class to have methods with the same name. In this case, when you call the method of an object of the child class, the method in the child class will shadow the method in the parent class.

The following `Dog` class extends the `Animal` class and redefines the `walk()` method:

```
class Dog extends Animal {
    constructor() {
        super(4);
    }
    walk() {
        console.log(`go walking`);
    }
}

let bingo = new Dog();
bingo.walk(); // go walking
```

To call the method of the parent class in the child class, you use `super.method(arguments)` like this:

```
class Dog extends Animal {
    constructor() {
        super(4);
    }
    walk() {
        super.walk();
        console.log(`go walking`);
    }
}

let bingo = new Dog();
bingo.walk();
// walking on 4 legs
// go walking
```

## Inheriting static members

Besides the properties and methods, the child class also inherits all static properties and methods of the parent class. For example:

```
class Animal {
    constructor(legs) {
        this.legs = legs;
    }
    walk() {
        console.log('walking on ' + this.legs + ' legs');
    }
    static helloWorld() {
        console.log('Hello World');
    }
}

class Bird extends Animal {
    fly() {
        console.log('flying');
    }
}
```

In this example, the `Animal` class has the `helloWorld()` static method and this method is available as `Bird.helloWorld()` and behaves the same as the `Animal.helloWorld()` method:

```
Bird.helloWorld(); // Hello World
```

## Inheriting from built-in types

JavaScript allows you to extend a built-in type such as Array, String, Map, and Set through inheritance.

The following `Queue` class extends the `Array` reference type. The syntax is much cleaner than the `Queue` implemented using the constructor/prototype pattern.

```
class Queue extends Array {
    enqueue(e) {
```

```
        super.push(e);
    }
    dequeue() {
        return super.shift();
    }
    peek() {
        return !this.empty() ? this[0] : undefined;
    }
    empty() {
        return this.length === 0;
    }
}

var customers = new Queue();
customers.enqueue('A');
customers.enqueue('B');
customers.enqueue('C');

while (!customers.empty()) {
    console.log(customers.dequeue());
}
```

## Summary

- Use the `extends` keyword to implement the inheritance in ES6. The class to be extended is called a base class or parent class. The class that extends the base class or parent class is called the derived class or child class.
- Call the `super(arguments)` in the child class's constructor to invoke the parent class's constructor.
- Use `super` keyword to call methods of the parent class in the methods of the child class.