**НАСЛЕДОВАНИЕ И ES6**

Классы в ES6

Классы, которые появились в ES6, являются только оберткой над механизмом прототипов. То есть, на самом деле, JavaScript все так же использует прототипы, просто делает это неявно для нас, предоставляя нам более удобный и понятный синтаксис.

class Car {

constructor(brand) {

this.brand = brand;

this.wheels = 4;

}

drive() {

console.log('Car is on road');

}

}

const bmw = new Car('M5');

bmw.drive();

Используя старый синтаксис, этот класс выглядел бы следующим образом:

function Car(brand) {

this.brand = brand;

this.wheels = 4;

};

Car.prototype.drive = function() {

console.log('Car is on road');

};

const bmw = new Car('M5');

bmw.drive();

Разберемся что происходит при использовании нового синтаксиса.  
Метод constructor – определяет функцию, которая представляет собой класс. Это специальный метод, который служит для создания и инициализации объектов, созданных с использованием ключевого слова class. Запуск этого метода происходит во время инициализации нового экземпляра класса с помощью new. Фактически это аналог знакомой нам функции-конструктора.  
Все остальные методы, которые находятся внутри тела класса, помещаются в прототип объекта.  
В теле класса могут определяться только методы, все свойства должны определяются в методе constructor.  
Рассмотрим остальные «виды» методов, которые могут использоваться в классах.  
Статические методы являются методами самого класса, которые вызываются непосредственно на самом классе и принадлежат ему. Определяются такие методы с помощью ключевого слова static.

class Car {

constructor(brand) {

this.brand = brand;

this.wheels = 4;

}

drive() {

console.log('Car is on road');

}

static info() {

console.log('Class for creating cars.');

}

}

const bmw = new Car('M5');

Car.info(); // Class for creating cars.

bmw.info(); // Uncaught TypeError: bmw.info is not a function

Эти методы не используются для наследования и создаются исключительно для собственных потребностей класса.  
Геттеры и сеттеры – это так называемые свойства-аксессоры (свойства доступа). Они, как правило, используются для получения/определения значений свойств объекта, чтобы исключить прямой доступ к этим свойствам.

class Car {

constructor(brand) {

this.brand = brand;

this.\_wheels = 4;

}

drive() {

console.log('Car is on road');

}

get wheels() {

console.log(this.\_wheels);

}

set wheels(value) {

this.\_wheels = value;

}

}

const bmw = new Car('M5');

bmw.wheels; // 4

bmw.wheels = 10;

bmw.wheels; // 10;

Как видно, геттер/сеттер вызываются не как обычные методы, а как свойства. Соответственно, для получения значения необходимо просто обратиться как к свойству, а для определения значения, задать необходимое значение с помощью оператора присвоения (=).

Наследование с помощью классов

Для наследования в ES6 существует специальное ключевое слово extends.

class Car {

constructor(name = 'default') {

this.wheels = 4;

this.name = name;

}

drive() {

console.log(`Car ${this.name} is on road`);

}

stop() {

console.log(`Car ${this.name} is stopped`);

}

}

class ElectroCar extends Car {

constructor(name) {

super(name);

}

}

const tesla = new ElectroCar('Model S');

tesla.drive();

tesla.stop();

Обратите внимание на использование super. С его помощью возможен вызов конструктора-родителя. Ключевое слово super вызывает конструктор родительского класса и позволяет «записать» все свойства из него в текущий класс (то есть в this текущего объекта).  
С помощью старого синтаксиса это реализовывалось через ParentFunction.apply(this, arguments).  
Кроме того, использование super помогает получить доступ к методам родительского класса, без необходимости напрямую обращаться к прототипу конструктора родителя, как это реализовывалось в старом синтаксисе ParentFunction.prototype.methodName.apply(this, arguments).

class Car {

constructor(name = 'default') {

this.wheels = 4;

this.name = name;

}

drive() {

console.log(`Car ${this.name} is on road`);

}

stop() {

console.log(`Car ${this.name} is stoped`);

}

}

class ElectroCar extends Car {

constructor(name) {

super(name);

}

drive() {

super.drive();

console.log(`Tesla ${this.name} is using battery to drive`);

}

}

var tesla = new ElectroCar('Model S');

tesla.drive();

// Car Model S is on road

// Tesla Model S is using battery to drive

Конструкция super.drive() дала возможность использовать результат вызова родительского метода и расширить его новым функционалом, который необходим для дочернего класса.  
Если наследование реализуется с использованием синтаксиса ES6 и существует необходимость работать с дочерним конструктором, super() следует вызывать перед добавлением любого нового свойства в дочерний конструктор. В ином случае, новые свойства будут затёрты.  
Если же в дочернем классе не будет указан собственный метод constructor, то ему (дочернему классу), автоматически будет присвоен конструктор родительского класса.  
Полная реализация наследования выглядит так:

class Car {

constructor(name = 'default') {

this.wheels = 4;

this.name = name;

}

drive() {

console.log(`Car ${this.name} is on road`);

}

stop() {

console.log(`Car ${this.name} is stoped`);

}

}

class ElectroCar extends Car {

constructor(name, year) {

super(name);

this.year = year;

}

drive() {

super.drive();

console.log(`Tesla ${this.name} is using battery to drive`);

console.log(`Year ${this.year}`);

}

}

var tesla = new ElectroCar('Model S', 2017);

tesla.drive();

// Car Model S is on road

// Tesla Model S is using battery to drive

// Year 2017

[ПРЕДЫДУЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-advanced/?course_type=content&course_page=3&lecture=1)[СЛЕДУЮЩАЯ ЧАСТЬ](https://www.jsexpert.net/course/clear-javascript-advanced/?course_type=section-quiz&course_page=3&lecture=2&section-quiz=1)