Classes in JavaScript

&

Object.create()

Outline

- Classes in JavaScript
- Defining Classes
  - Class Declaration
  - Class Expression
- Constructor and Methods
- Class vs. Constructor Function
- Getters and Setters in Class
- Static Methods
- Object.create()

Classes in JavaScript

- Introduced in ECMAScript 2015
- It’s a syntactical sugar over JavaScript's existing prototype-based inheritance
- *It does not* introduce a new object-oriented inheritance model to JavaScript

Defining Classes

1) Class declarations

2) Class expressions

Class Declaration

```
class MyClass {
  // class methods
  constructor() { ... }
  method1() { ... }
  method2() { ... }
  method3() { ... }
  ...
}
```

Class Declaration

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}

const p = new Rectangle(20, 30);
```

Hoisting

- Class declarations are not hoisted

```
const p = new Rectangle(20,30);
// ReferenceError

class Rectangle {
}
```

Class Expressions (unnamed)

```
let Rectangle = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};

console.log(Rectangle.name); //"Rectangle"
```

Class Expressions (named)

```
let Rectangle = class Rectangle2 {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};

console.log(Rectangle.name); //"Rectangle2"
```

Constructor and Methods

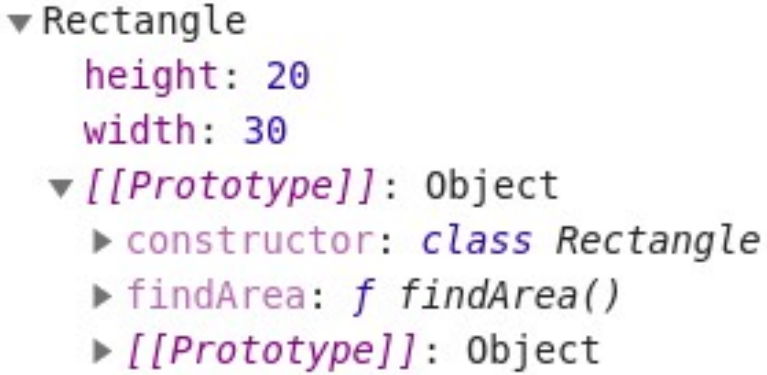
- The constructor method is a special method for creating and initializing an object created with a class
- There can only be one special method with the name "constructor" in a class
- A SyntaxError will be thrown if the class contains more than one occurrence of a constructor method
- A constructor can use the super keyword to call the constructor of the super class
- Methods defined in class are added in the prototype and not considered as property

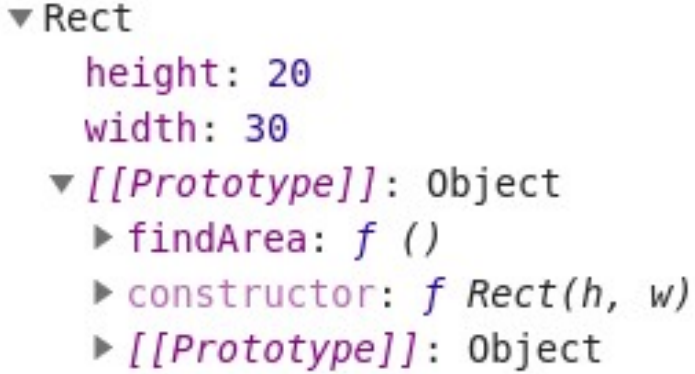
Class vs. Function

```
class Rectangle {
    constructor(height, width) {
        this.height = height;
        this.width = width;
    }

    findArea() {
        return this.height * this.width;
    }
}
const rect1 = new Rectangle(20, 30);
console.dir(rect1);
```

```
function Rect(h, w) {
    this.height = h;
    this.width = w;
}
Rect.prototype.findArea = function() {
    return this.height * this.width;
};
const rect2 = new Rect(20, 30);
console.dir(rect2);
```





Class vs. Function

- Unlike a regular function, a class constructor can’t be called without new
- Class methods are non-enumerable
- Classes always use strict mode. All code inside the class construct is automatically in strict mode.

Getters/Setters Methods – (1)

```
class Person {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    get fullName() {
        return this.firstName + " " + this.lastName;
    }

    set fullName(fname) {
        if (fname.includes(" ")) {
            [this.firstName, this.lastName] = fname.split(" ");
        } else console.log("Given name is not a full name");
    }
}
let p = new Person("Sachin", "Tendulkar");
console.log(p.fullName);
p.fullName = "Rohit";
```

```
Sachin Tendulkar
Given name is not a full name
```

Getters/Setters Methods – (2)

```
class User {
    constructor(name) {
        this.name = name;
    }
    get name() {
        return this._name;
    }
    set name(value) {
        if (value.length < 3) {
            console.log("Name is too short.");
            return;
        }
        this._name = value;
    }
}
let user = new User("Virat");
console.log(user.name);
user = new User("");
```

```
Virat
Name is too short.
```

Static Methods

- The static keyword defines a static method for a class
- Static methods are called only on class and not on the instance.
- Static methods are often used to create utility functions for an application

Static Methods

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }

    static distance(a, b) {
        const dx = a.x - b.x;
        const dy = a.y - b.y;

        return Math.hypot(dx, dy);
    }

    display() {
        return `(${this.x},${this.y})`;
    }
}
const p1 = new Point(8, 9);
const p2 = new Point(5, 5);
console.log(
    `Distance between ${p1.display()} and ${p2.display()} : ${Point.distance(
    p1,
    p2
  )}`
);
```

```
Distance between (8,9) and (5,5) : 5
```

Object.create()

The Object.create() method creates a new object, using an existing object as the prototype of the newly created object.

- Syntax:

  Object.create(prototypeObject, propertiesObject)

  - prototypeObject: Newly created object's prototype object. It has to be an object or null.

  - propertiesObject: Properties of the new object. This argument is optional

Create an object with no prototype

```
var person = Object.create(null);

console.log(typeof person);
console.log(person);
```

```
object
▼ {}
    No properties
```

```
var person = Object.create(null);

console.log(typeof person);
console.log(person);

// Set property to person object
person.name = "Virat";

console.log(person);
```

```
object
▶ {}
▼ {name: "Virat"}
    name: "Virat"
```

Create an object with prototype

```
prototypeObject = {
    fullName: function() {
        return this.firstName + " " + this.lastName;
    },
};
var person = Object.create(prototypeObject);

console.log(person);
```

```
▼ {}
  ▼ [[Prototype]]: Object
    ▶ fullName: ƒ ()
    ▶ [[Prototype]]: Object
```

```
prototypeObject = {
    fullName: function() {
        return this.firstName + " " + this.lastName;
    },
};
var person = Object.create(prototypeObject);

console.log(person);

// Adding properties to the person object
person.firstName = "Virat";
person.lastName = "Kohli";

console.log(person.fullName());
```

```
▼ {}
    firstName: "Virat"
    lastName: "Kohli"
  ▼ [[Prototype]]: Object
    ▶ fullName: ƒ ()
    ▶ [[Prototype]]: Object

Virat Kohli
```

Object.create() with propertiesObject

propertiesObject is used to create properties on a new object. It acts as a descriptor for the new properties to be defined.

```
prototypeObject = {
  fullName: function() {
      return this.firstName + " " + this.lastName;
  },
};
var person = Object.create(prototypeObject, {
    firstName: {
        value: "Virat",
        writable: true,
        enumerable: true,
    },
    lastName: {
        value: "Kohli",
    },
});
console.log(person);
console.log(Object.keys(person));
for (var prop in person) {
    console.log(prop);
}
```

```
▼ {firstName: "Virat", lastName: "Kohli"}
    firstName: "Virat"
    lastName: "Kohli"
  ▼ [[Prototype]]: Object
    ▶ fullName: ƒ ()
    ▶ [[Prototype]]: Object
▶ ["firstName"]
firstName
fullName
```

References

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes
- https://mediumhttps://medium.com/@happymishra66/object-create-in-javascript-fa8674df6ed2