JavaScript Classes

Classes, Constructors, Properties, Unit Tests

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Content

SoftUni Foundation

1. **Defining Classes**
2. **Class Body** and **Method Definitions**
   - Prototype Methods
   - Fields
3. **Class Inheritance**
4. **Unit Tests** on Classes

Have a Question?

SoftUni Foundation

sli.do

#JS-CORE

Classes in JS

Definition, Declaration, Expression, Hoisting

Class Definition

Structure for objects

- Classes define:
  - **Data** (properties, attributes)
  - **Actions** (behavior)
- One class may have **many instances** (objects)
- The class syntax has two components:
  - **Class Expressions** and **Class Declarations**

Class Declaration

- Use the **class keyword** with the name of the class
- The **constructor** defines class data

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

Class Expression

SoftUni Foundation

Another way to **define a class**

- Class expressions can be **named** or **unnamed**

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

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

Hoisting

- **Function declarations are hoisted and class declarations are not**
- You first need to declare your class and then access it, otherwise a **ReferenceError** will be thrown

```
const p = new Rectangle(); // ReferenceError
class Rectangle {}
```

- **Class expressions** are subject to the same **hoisting restrictions**

Problem: Rectangle

SoftUni Foundation

Write a JS **class** for a rectangle object

- It needs to have **width**, **height**, **color** and **calcArea()** method

```
let rect = new Rectangle(4, 5, 'red');
console.log(rect.width);       // 4
console.log(rect.height);      // 5
console.log(rect.color);       // Red
console.log(rect.calcArea()); // 20
```

Solution: Rectangle

```
class Rectangle {
    constructor(width, height, color) {
        this.width = width;
        this.height = height;
        this.color = color;
    }
    calcArea(){
        return this.width * this.height;
    }
}
```

Class Body and Methods

Definition, Constructor, Prototype, Fields

Class Body

- The **constructor** is a special method for **creating** and **initializing** an object created with a class
- A **SyntaxError** will be thrown if a class contains **more than one** occurrence of a **constructor method**

```
class Rectangle {
    // Class Body
}
```

```
constructor() {
    // Class Body
}
```

```
class Rectangle() {
    // Syntax Error
}
```

Prototype

- JS objects **inherit properties** and **methods** from a **prototype**
- The **Prototype Property** allows you to add **new properties** to object **constructors**

```
function Person(first, last, age) {
    this.firstName = first;
    this.lastName = last;
    this.age = age;
}
Person.prototype.nationality = "Bulgarian";
```

Prototype Methods

SoftUni Foundation

Before ES2015 (ES6), **classes** were composed **manually**

```
function Rectangle(width, height) {
  this.width = width;
  this.height = height;
}
Rectangle.prototype.area = function () {
  return this.width * this.height;
}
let rect = new Rectangle(3, 5);
```

Comparison with the New Syntax

SoftUni Foundation

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

  area() {
    return this.width * this.height;
  }
}
```

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

```
Rectangle.prototype.area = function() {
  return this.width * this.height;
},
```

Static Methods

SoftUni Foundation

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

```
static staticMethod() {
    return 'Static method has been called';
}
```

- Called **without instantiating** their class and **cannot be called** through a class instance
- To call a static method of the same class, you can use the **this** keyword

```
static anotherStaticMethod() {
    return this.staticMethod() + ' from another method';
}
```

Accessor Properties

SoftUni Foundation

```
class Circle {
  constructor(radius) { this.radius = radius; }
  get diameter() { return 2 * this.radius; }
  set diameter(diameter) {
    this.radius = diameter / 2;
  }
  get area() {
    return Math.PI * this.radius * this.radius;
  }
}
```

Property getter

Property setter

Read-only property "area"

Accessor Properties in Action

```
let c = new Circle(2);
console.log(`Radius: ${c.radius}`); // 2
console.log(`Diameter: ${c.diameter}`); // 4
console.log(`Area: ${c.area}`); // 12.566370614359172
```

```
c.diameter = 1.6;
console.log(`Radius: ${c.radius}`); // 0.8
console.log(`Diameter: ${c.diameter}`); // 1.6
console.log(`Area: ${c.area}`); // 2.0106192982974678
```

Private Properties

- Prefix each private property name with an **underscore**

```
function Point(x, y) {
    this._x = x;
    this._y = y;
}
```

- To make a private property **readable**/**writable** from any function, it's common to define **getters**/**setters**

Accessing Private Properties

SoftUni Foundation

```
Point.prototype.getX = function () {
    return this._x;
};
```

```
Point.prototype.setX = function (x) {
    this._x = x;
};
```

```
Point.prototype.getY = function () {
    return this._y;
};
```

```
Point.prototype.setY = function (y) {
    this._y = y;
};
```

Problem: Person

Write a JS class that represent a personal record

- It needs to have the following properties:
  - **firstName**, **lastName**, **age** and **email**
- And a **toString()** method

```
let person = new Person('Maria', 'Petrova', 22, 'mp@yahoo.com');
console.log(person.toString());
// Maria Petrova (age: 22, email: mp@yahoo.com)
```

Solution: Person

```
class Person {
    constructor(fName, lName, age, email) {
        this.firstName = fName;
        this.lastName = lName;
        this.age = age;
        this.email = email;
    }
    toString() {
        return `${this.firstName} ${this.lastName}
                (age: ${this.age}, email: ${this.email})`
    }
}
```

Problem: Get Persons

SoftUni Foundation

Write a JS **function** that returns an array of **Person objects**

- Use the class from the previous task
- There will be **no input**, the data is **static** and matches on this data

First Name	Last Name	Age	Email
Maria	Petrova	22	mp@yahoo.com
SoftUni			
Stephan	Nikolov	25	
Peter	Kolev	24	ptr.@gmail.com

Solution: Get Persons

```
class Person {

    constructor(firstName, lastName, age, email){
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
        this.email = email;
    }

    toString (){
        return `${this.firstName} ${this.lastName}
                (age: ${this.age}, email: ${this.email})`

        }

    }

    return [new Person('Maria', 'Petrova', 22, 'mp@yahoo.com'),
            ... //TODO for the rest of the persons
```

Class Inheritance

Inheriting Data and Methods

Class Inheritance

Classes can **inherit** (extend) other classes

- Child class inherits data + methods from its parent
- The **extends** keyword is used to create a class which is a **child of another class**
- **Child class** can:
  - Add **properties** (data)
  - Add / replace **methods**
  - Add / replace **accessor** properties

Class Inheritance - Example

```
class Person {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }
}
```

```
class Teacher extends Person {
  constructor(name, email, subject) {
    super(name, email);
    this.subject = subject;
  }
}
```

Class Inheritance - Example

```
let p = new Person("Maria", "maria@gmail.com");
console.log(`Person: ${p.name} (${p.email})`);
// Person: Maria (maria@gmail.com)
```

```
let t = new Teacher("Ivan", "iv@yahoo.com", "PHP");
console.log(
    `Teacher: ${t.name} (${t.email}), teaches ${t.subject}`
);
// Teacher: Ivan (iv@yahoo.com), teaches PHP
```

Unit Tests on Classes

Unit Tests on Classes - Example

SoftUni Foundation

```
class SortedList {
    constructor() { this.list = []; }
    add(element) { this.list.push(element); }
    remove(index) { this.list.splice(index, 1); }
    get size() { return this.list.length; } }
```

```
describe("Sorted List", function () {
    let sortedList;
    beforeEach(function () {
        sortedList = new SortedList();
    });
```

Unit Tests on Classes - Example

SoftUni Foundation

```
it('must initialize data to an empty array', function () {
   expect(sortedList.list instanceof Array).to.equal(true, 'Data must be
of type array');
   expect(sortedList.list.size).to.equal(0, 'Data array must be
initialized empty');
});
```

```
it('should remove correctly', function () {
   sortedList.add(123);
   sortedList.add(1234);
   sortedList.remove(1);
   expect(sortedList.size).to.equal(1);
});
```

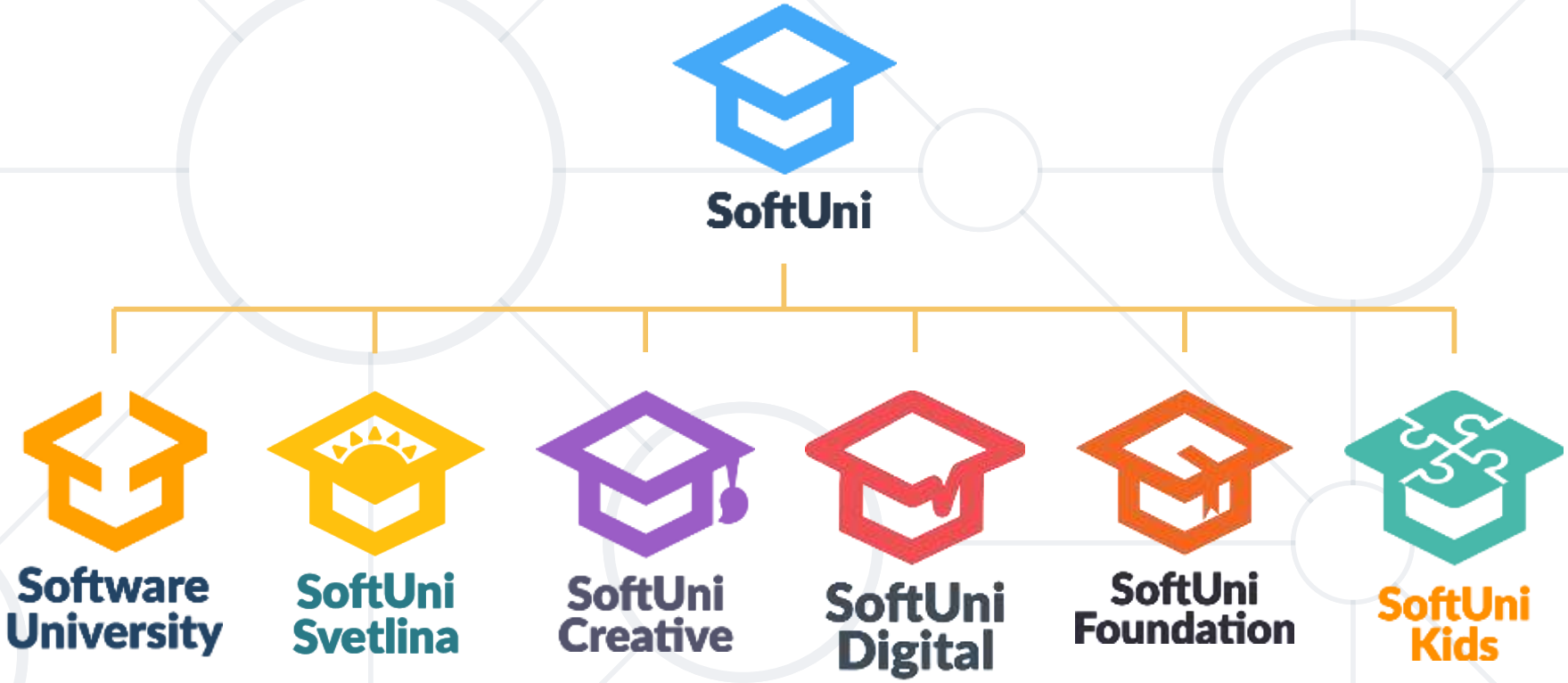
Live Exercises

Summary

Classes:

- Provide **structure** for objects.
- May define **methods**.
- May define **accessor properties**.
- Can **inherit** other classes.

Questions?



https://softuni.bg/courses/javascript-advanced

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

Lukanet.com

WORLD OF MYTHS

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license