Back to Week 1

JS | OOP - objects, methods and the 'this' keyword

LESSON

Learning Goals

After this lesson you will be able to:

- Explain what Object Oriented Programming is
- Comprehend the importance of thinking about objects
- Create objects with the literal pattern
- Access properties of an object
- Use the `this` keyword

Introduction

> Official definition of **object-oriented programming (OOP)** is that it is a programming paradigm based on the concept of “objects”, which can contain data, in the form of fields (often known as *attributes*), and code, in the form of procedures (often known as *methods*).

Okay, so the main take away from this is - **objects** are the main key/tool/means of OOP. We will try to explain this through example.

Ways of constructing the objects

Some of these topics we already covered but now we will take them one step ahead.

- Object Literals

In this lesson, we will create a very simple Monopoly game with 16 squares and 3 players. At every turn, a player launches one dice, moves forward and updates their cash based on the square’s value.

You can see an example below illustrating the game we will create:

→

100	-10	0	0	-40
-10				-10
-50		turn 0		-10
0				5
0	-10	-50	-10	0

Player	Position	Cash
Joaquim	0	1,000€
Maxence	0	1,000€
Mostafa	0	1,000€

→

100	-10	0	0	-40
-10				-10
-50		turn 1		-10
0				5
0	-10	-50	-10	0

Player	Position	Cash
Joaquim	6	990€
Maxence	4	960€
Mostafa	1	990€

If we use *object literal* approach, which is the simplest approach we’ve seen so far, we could simulate the game the following way:

```
// Example of a VERY simple Monopoly game simulation

// Each square represents the cash earned when a player is on it. For example:
// - If a user is at position 0, the cash will increase by 100€
// - If a user is at position 4, the cash will decrease by 40€
const squares = [100, -10, 0, 0, -40, -10, -10, 5, 0, -10, -50, -10, 0, 0, -50, -10];

// --- Initialization ---

let player1 = {
  name: 'Joaquim',
  color: 'red',
  position: 0,
  cash: 1000,
};

let player2 = {
  name: 'Maxence',
  color: 'blue',
  position: 0,
  cash: 1000,
};

let player3 = {
  name: 'Mostafa',
  color: 'black',
  position: 0,
  cash: 1000,
};

// --- Turn of Player 1 ---
// The dice is a random integer between 1 and 6
let dice = 1 + Math.floor(6 * Math.random());
// The position is always between 0 and 15 (the numbers of squares - 1)
player1.position = (player1.position + dice) % squares.length;
// The cash is updated based on the values of squares and player1.position
player1.cash += squares[player1.position];
// If the player doesn't have anymore cash, player is out of game
if (player1.cash < 0) {
  console.log(`Game over for ${player1.name}.`);
}

// --- Turn of Player 2 ---
let dice = 1 + Math.floor(6 * Math.random());
player2.position = (player2.position + dice) % squares.length;
player2.cash += squares[player2.position];
if (player2.cash < 0) {
  console.log(`Game over for ${player2.name}.`);
}

// --- Turn of Player 3 ---
let dice = 1 + Math.floor(6 * Math.random());
player3.position = (player3.position + dice) % squares.length;
player3.cash += squares[player3.position];
if (player3.cash < 0) {
  console.log(`Game over for ${player3.name}.`);
}

// --- Display info  ---
console.log(player1);
console.log(player2);
console.log(player3);
```

Explain this code

Using this simple approach has some **pros**:

- it's super convenient, straight-forward
- very flexible in a declaration
- very little code when declaring them
- you can create them at any point in your code and use them without a lot of previous set up

However, this approach has some **cons** as well:

- **We don't have a fast way to create the object**. Instead, we always have to specify all the properties, which means a lot of copy pasting and making minor changes from object to object. (In our previous example, we are initializing the `position` to `0` and the `cash` to `1000`.)
- **We don't have any methods for our objects**. It would be nice to have a method `player.move()` instead of writing the same code again and again.

Object methods and `this` keyword

Let's change our code to add 2 methods: `move()` and `displayInfo()`.

```
// Example of a VERY simple Monopoly game simulation

let squares = [100, -10, 0, 0, -40, -10, -10, 5, 0, -10, -50, -10, 0, 0, -50, -10];

// --- Initialization with methods ---

let player1 = {
  name: 'Joaquim',
  color: 'red',
  position: 0,
  cash: 1000,
  move() {
    let dice = 1 + Math.floor(6 * Math.random());
    this.position = (this.position + dice) % squares.length;
    this.cash += squares[this.position];
    if (this.cash < 0) {
      console.log(`Game over for ${this.name}.`);
    }
  },
  displayInfo {
    console.log(`${this.name} is at position ${this.position} and has ${this.cash}€`);
  },
};

let player2 = {
  name: 'Maxence',
  color: 'blue',
  position: 0,
  cash: 1000,
  move() {
    let dice = 1 + Math.floor(6 * Math.random());
    this.position = (this.position + dice) % squares.length;
    this.cash += squares[this.position];
    if (this.cash < 0) {
      console.log(`Game over for ${this.name}.`);
    }
  },
  displayInfo () {
```

```
    console.log(`${this.name} is at position ${this.position} and has ${this.cash}€`);
  },
};

let player3 = {
  name: 'Mostafa',
  color: 'black',
  position: 0,
  cash: 1000,
  move() {
    let dice = 1 + Math.floor(6 * Math.random());
    this.position = (this.position + dice) % squares.length;
    this.cash += squares[this.position];
    if (this.cash < 0) {
      console.log(`Game over for ${this.name}.`);
    }
  },
  displayInfo () {
    console.log(`${this.name} is at position ${this.position} and has ${this.cash}€`);
  },
};

// --- Turn 1  ---
player1.move();
player2.move();
player3.move();

// --- Turn 2  ---
player1.move();
player2.move();
player3.move();

// --- Display info  ---
player1.displayInfo();
player2.displayInfo();
player3.displayInfo();
```

Explain this code

Let’s analyze the new code and compare it to the previous one.

As we can see, the **code is more readable**, especially the part for “Turn 1” and “Turn 2”.

Our objects `player1`, `player2` and `player3` now have 2 extra properties: `move` and `displayInfo`. Both of them are functions, called **methods**, and they have a new keyword: `this`.
In this context, `this` refers to the current object. For `player1`, inside the `displayInfo`, `this.name` === `player1.name` === `Joaquim`.

> When invoking a method on an object, `this` **becomes the object itself**.

The last problem with have is that we don’t have a fast way to create the objects `player1`, `player2` and `player3`. We will solve this problem in the next lecture!

Exercise with `this`

```js
// TODO: write the methods getAge, addJoke and getRandomJoke

const chuck = {
  firstName: 'Chuck',
  lastName: 'Norris',
  birthDate: new Date('1940-03-10'),
  jokes: ['Chuck Norris counted to infinity... Twice.', 'Chuck Norris is the only man to ever defeat a brick wall in a game of tennis'],
  displayInfo() {
    console.log(`My name is ${this.firstName} ${this.lastName} and I have ${this.jokes.length} jokes.`);
  },
  getAge() {
    // TODO
    // Hint: to get the current time, you can do: new Date()
    // Hint: to get the birthDate, you can do: this.birthDate
    // Hint: you can subtract 2 dates and you get the number of milliseconds
  },
  addJoke(joke) {
    // TODO (don't use return statement)
  },
  getRandomJoke() {
    // TODO
  }
};

chuck.displayInfo();

console.log('getAge', chuck.getAge()); // Should return 80 if you are in 2020

chuck.addJoke('Chuck Norris can divide by zero.');
console.log('getRandomJoke', chuck.getRandomJoke());
chuck.addJoke('Chuck Norris kills flies with his gun.');
console.log('getRandomJoke', chuck.getRandomJoke());
chuck.addJoke('Chuck Norris was once in a knife fight, and the knife lost.');
console.log('getRandomJoke', chuck.getRandomJoke());

chuck.displayInfo();
```

Explain this code

You can edit this Repl.it note.

Summary

We've seen how to create methods (a function linked to an object) and we've seen the keyword **`this`** that refers to the current object.

Extra resources

- Learn OOP (video)
- Object - fundamentals (video)

Mark as completed

PREVIOUS

← JS | Value vs Reference and Mutable Data Types

NEXT

JS | OOP - class and inheritance →