Page No.
Date

Classes

ES6 - A class is like a blueprint for an object (it describes how one should be made)

example :-

car blueprint

properties
- its colour
- its model
- engine size

functionality
- drive
- reverse
- brake

All cars are not completely identical, some of the properties are unique like colour, some may be red, some may be blue, other may be white

new white car
new yellow car
new green car

these all cars have same functionality

Now this same principle applies to object in javascript, only in javascript we used 'class' to describe the type of object

User class

```
{
    username,
    login(),
    email
    logout()
}
```

new User('ryu', 'abc@.com')
new User('ken', 'efg@.com')
new User('hell', 'ajt@.com')

Class Constructors

- Constructor function is the thing that actually constructs our object and sets the properties on it.

- the 'new' keyword
  1. it creates a new empty object {}
  2. it binds the value of 'this' to the new empty object
  3. it calls the constructor function to 'build' the object

```
- class User {
      constructor() {
        //set up properties
        this.username = 'mario';
      }
}
const userOne = new User();
console.log(userOne);
```

→ ▶ User {username: "mario"}

```
- class User {
      constructor() {
        //set up properties
        this.username = 'mario';
      }
}
const userOne = new User();
const userTwo = new User();
console.log(userOne, userTwo);
```

↓

▶ User {username: "mario"} ▶ User {username: "mario"}

Page No.
Date

Now here we see two objects which are identical to each other
to make the objects more unique consider below example

```
- class User {
    constructor(username) {
      this.username = username;
    }
}
const userOne = new User('mario');
const userTwo = new User('luigi');
console.log(userOne, userTwo);
```

↓

▶ User {username: "mario"} ▶ User {username: "luigi"}

So this makes the object more unique

```
- class User {
    constructor(username, email) {
      this.username = username;
      this.email = email;
    }
}
const userOne = new User('mar', 'hello@gmail.com');
const userTwo = new User('lui', 'lui@gmail.com');
console.log(userOne, userTwo);
```

↓

▶ User {username: "mar", email: "hello@gmail.com"}
▶ User {username: "lui", email: "lui@gmail.com"}

[whenever we create an object then we are creating an instance of that class.]

- instance refers to the individual object that we actually create using the class.

Class methods and method chaining

```
class User {
    constructor(username, email){
        //set up properties
        this.username = username;
        this.email = email;
    }
    login(){
        console.log(`${this.username} logged in`);
    }
    logout(){
        console.log(`${this.username} logged out`);
    }
}
const userOne = new User('max', '@12');
const userTwo = new User('hello', '@40');
userOne.login();
userTwo.login();
userOne.logout();
userTwo.logout();
```

Here we are using shorthand notation and function keyword

Don't give here comma (inside a class we don't comma separate different methods)

And when we use regular function and inside that we use "this" keyword it will be the object instance and that's the new object we create

We use regular function because arrow function don't bind "this" keyword directly to window object

```
max logged in
hello logged in
max logged out
hello logged out
```

To return object instance we have written that line "return this" because "this" refers to the object instance

- method chaining

```
class Users{
    constructor(username, email){
        this.username = username;
        this.email = email;
        this.score = 0;
    }
    login(){
        console.log(`${this.username} logged in`);
        return this;
    }
    logout(){
        console.log(`${this.username} logged out`);
        return this;
    }
    incScore(){
        this.score += 1;
        console.log(`${this.username} has scored of ${this.score}`);
    }
}
const UserOne = new Users('max', 'hel1@gmail.com');
const UserTwo = new Users('hello', 'nono@gmail.com');
UserOne.login().incScore().incScore().logout();
```

this is one method

this is another method

```
max logged in
max has scored of 1
max has scored of 2
max logged out
```

You don't have to return this. instead you can return a different value and if you want to chain them together then you should return this

this method chaining works because we explicitly return the instance at end of each method. Note you don't have to do always and if you don't want your methods chainable then go to leftside