

Classes

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

Example :-

car blueprint



properties	functionality
- its colour	- drive
- its model	- reverse
- engine size	- brake

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

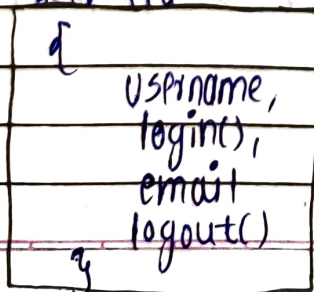


new white car
new yellow car
new green car

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



new User('xyu', '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';

}

}

→ ▶ User {username:

"mario"}

const userOne = new User();

console.log(userOne);

- 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"}

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]

Page No.

Date

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

Class Methods and Method Chaining

- class Users

```
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('man', '@12');
```

```
const userTwo = new User('hello', '@40');
```

```
userOne.login();
```

```
userTwo.login();
```

```
userOne.logout();
```

```
userTwo.logout();
```

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

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

man logged in
hello logged in
man logged out
hello logged out

function
we use regular function because arrow function don't bind
value to "this" keyword and if we use arrow function then "this" will refer
directly to window object

here we are using the regular
using shorthand notation and

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;  
}
```

this is one method →

```
logout() {  
  console.log(`${this.username} logged out`);  
  return this;  
}
```

this is another method →

```
incScore() {  
  this.score += 1;  
  console.log(`${this.username} has scored of ${this.score}`);  
}
```

```
incScore() {
```

```
  this.score += 1;
```

```
  console.log(`${this.username} has scored of ${this.score}`);  
}
```

```
}
```

```
const UserOne = new User('mas', 'hell@gmail.com');
```

```
const UserTwo = new User('hello', 'nomo@gmail.com');
```

```
UserOne.login().incScore().incScore().logout();
```

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

mas logged in
mas has scored of 1
mas has scored of 2
mas logged out

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 left side