Mixins

This lesson teaches the concept of mixins in JavaScript, their syntax, and their implementation using an example.

We'll cover the following

What are Mixins?

So far, you've learned that for a class to call *methods* from another class it first needs to inherit those methods. The child class extends the parent's class, inherits its methods, and then invokes them. However, there is a limiting factor: a class can only inherit from another class.

That brings us to the question:

Is there a class whose methods can be inherited by other classes without it having to be their parent class? This where **mixins** are implemented.

A **mixin** is a class that contains various methods implementing different functionalities. Other classes can then inherit those methods without having the *mixin* class be their parent class.

> **Note:** A mixin class is not used alone. It only provides other classes with extra *methods*.

Syntax

Let's take a look at the syntax to implement a mixin:

```
var mixin = {
  //methods defined
}
```

As seen from above, a *mixin* can be made easily by making it an object containing various *methods*. In order for other classes to use these methods, the *mixin* can be set as their `prototype`.

Example

Let's take a look at an example implementing a mixin:

```
//creating a mixin
var mixin = {
  getName() {
    console.log(`Name is ${this.name}`);
  },
  getSides() {
    console.log(`Sides are ${this.sides}`);
  }
}

//creating a class Shape
class Shape {
  constructor(shapeName,shapeSides) {
    this.name = shapeName
    this.sides = shapeSides
  }
}

//setting mixin to be the prototype of Shape
Shape.prototype =  mixin;
//setting constructor of prototype equal to Shape
Shape.prototype.constructor = Shape

//creating a new Shape
var rectangle = new Shape('Rectangle',4)
rectangle.getName()
rectangle.getSides()
```

Explanation

- A `mixin` containing the functions `getName` and `getSides` is defined.
- Next, a class `Shape` containing the properties `name` and `sides` is defined.
- The prototype of the class `Shape` is then set to `mixin`, i.e., `mixin` becomes

the prototype of the class `Shape`.

- Next, we set the `Shape.prototype.constructor` to point to `Shape` since it was pointing to `mixin` object after we set it as the prototype.

Now when the class object `rectangle` calls the `getName` and `getSides` functions, they get retrieved from `mixin` since it is set as the prototype of the `Shape` class.

Now that you have learned in detail about prototypes, prototypal inheritance, class-based inheritance, and mixins, let’s put all that knowledge to test in the next lesson.