Lesson Monday

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Static Methods and Properties

Text

All of the custom methods we've written up to this point have been **instance methods**. In this lesson, we'll learn how to write a **static method** which can be called directly on an object type. We'll also learn how to create **static properties**.

Before proceeding, keep in mind that "object type" and "class" are interchangeable terms in JavaScript. Also, in many languages "static methods" are also called "class methods". All that said, always keep in mind that JavaScript only uses classes as syntactic sugar, which sits on top of the original object-creation syntax, using object literals, constructor functions, and prototype methods.

We have already been introduced to the concepts of instance and static, both in lessons and via the instance and static methods we have implemented in our code, however we'll still review exactly what "instance" and "static" means, and work through some examples.

Writing a Static Method

We'll use a `Car` class to demonstrate how to write a static method. Here is what is in our class:

```
class Car {
  constructor(make, model) {
    this.make = make,
    this.model = model
  }

  drive() {
    return "Vvrrrooooom!"
  }
}
```

First, let's take a look at an example of an instance method:

```
const car = new Car("pontiac", "aztec");
car.drive();
"Vvrrrooooom!"
```

We first have to create an instance of a `Car` before we can drive it. Because the method is called on a single instance of a car (and not all cars), it is an **instance** method.

A static method, on the other hand, is called on the class itself. What if we wanted to sort a factory full of cars by color? Well, we can't call that on one car. We need to call it on *all* of the cars, which means we'd need a static method. We can define a static method in a class like this:

```
class Car {
  ...

  static sort(color) {
    // Code for method here.
  }
}
```

As we can see, all we need to do is add the `static` keyword.

Then, when we want to call the method, we can call it directly on the class:

```
Car.sort("red");
```

C#/.NET and Ruby/Rails students will learn a lot more about instance versus static/class methods in their backend courses.

But what do static methods have to do with the code we are writing to make API calls? Well, it would be nice to encapsulate our API logic in its own ES6 class. **Encapsulation** is the process of storing information inside a class to keep it separate from other logic in our application. It helps us keep our code more modular and organized. That means we can make our code better by creating a class to hold our API logic.

However, it doesn't make much sense to have an instance of the class before we make our API call. It's not just extra code — there's just no need to create an instance instead of just calling our method on the class itself. This will become apparent when we refactor our code in the next lesson.

By the way, you've already worked with static methods, even if you haven't realized it. For instance, whenever you call a `Math` class method, you are using a static method. In fact, the code we used to generate a random number a few lessons ago uses two static methods: `Math.random()` and `Math.floor()`.

```
Math.floor(Math.random() * Math.floor(2));
```

Also, the code we use to parse an API's response is a static method of the `JSON` object type.

```
JSON.parse(apiResponse);
```

We also briefly discussed the `Promise.all()` method a few lessons ago — this is also a static method. A key thing to note about documentation regarding static methods is that they don't use the `prototype` keyword so it would be incorrect to say `Promise.prototype.all()`. Static methods aren't automatically inherited, unlike prototypal methods. (However, we can use the `extends` keyword (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes/extends) to ensure that static methods are inherited but that's beyond the scope of this lesson.)

Static Properties

We can also have static properties as well. That just means that we attach properties to the class itself, not an instance of the class. For instance, we could do something like this:

```
Car.colors = ["red", "green", "blue"];
```

The class itself now has a `colors` property with a list of all the colors a car might be.

If we wanted to declare the `colors` property in our class, we'd also use the `static` keyword:

```
class Car {
  constructor(make, model) {
    this.make = make,
    this.model = model
  }

  static colors = ["red", "green", "blue"];

  drive() {
    return "Vvrrrooooom!"
  }
}
```

You probably won't need to use **static properties** in any of your projects (and they aren't required for any independent projects), but it's still good to know they exist. You will, however, be required to use a **static method** for this section's independent project.

Previous (/intermediate-javascript/asynchrony-and-apis/promises-with-api-calls)
Next (/intermediate-javascript/asynchrony-and-apis/separating-promise-logic)

Lesson 19 of 33

Last updated more than 3 months ago.

disable dark mode

Epicodus (http://www.epicodus.com)