Tip 37: Build Readable Classes

In this tip, you’ll learn how to create extendable classes in JavaScript.

We'll cover the following

- Classes in JavaScript
- Example: Making a class
  - Defnining a constructor
  - Setting & accessing properties
  - Adding methods

Classes in JavaScript

One of the longest-running criticisms of JavaScript was that it lacked a class syntax. Well, it’s here! But it didn’t arrive without controversy. Proponents of classes argue it’s a common development paradigm that’s very familiar to developers in other languages. Skeptics think it obscures the underlying nature of the language and encourages bad habits.

Like many controversies, the rhetoric is excessive. Classes are now part of JavaScript, and if you use any popular framework such as Angular or React, you’ll introduce them in your code. And that’s great.

As you’ll see in Tip 39, Extend Existing Prototypes with Class, the base language hasn’t changed. JavaScript is still a *prototype-based* language. Now you have familiar syntax masking slightly complicated concepts. As a result, there are some surprises.

In this tip, you’ll get a quick look at how to write classes in JavaScript. If you’ve written classes in any other language, the interface should seem pretty familiar.

Example: Making a class

To start off, make a class called `Coupon`. You declare a class with the `class` keyword. You can then create new instances using the `new` keyword.

```
class Coupon {
}
const coupon = new Coupon();
console.log(coupon);
```

When you create an instance of a class, the first thing you're doing is running a *constructor function*, which can define a number of properties. You *aren't* forced to declare a constructor function, but it's where you will declare your *properties*, so you'll write one in most cases.

Defnining a constructor

The next step is to create a constructor method. You'll need to name it `constructor()`. Add it to the class using what looks like function syntax, but without the `function` keyword. Because the constructor is a function, you can pass as many arguments as you want.

Part of the job of the `constructor` is creating a `this` context. Inside your constructor, you add properties to a class by assigning them to `this` with a *key* like you would if you were adding *key-values* to an object. And because you're able to pass arguments to the `constructor`, you can dynamically set properties when you create a `new` instance. Currently, you are required to set all properties inside the `constructor`. That will likely change in the future.

Setting & accessing properties

For now, set two properties on your `Coupon`: `price` and `expiration`. After setting the properties, you can call them using the familiar dot syntax or even array syntax. Remember that this is still JavaScript, and you're still working with objects.

```
class Coupon {
    constructor(price, expiration) {
        this.price = price;
        this.expiration = expiration || 'two weeks';
    }
}
const coupon = new Coupon(5);
console.log(coupon.price);
console.log(coupon['expiration']);
```

> Note that you aren't declaring properties as *public* or *private*. Currently, everything is *public*. Private fields—*properties or methods*—are working their way through the TC39 committee.

Adding methods

The class and object instance are getting a little more interesting, but they still can't do much. The next step is to add *two* simple methods: `getPriceText()` to return a formatted price and `getExpirationMessage()` to get a formatted message.

You can add methods using the same syntax as a constructor. The methods will be normal functions, not arrow functions. This may not seem like a big deal, but arrow functions behave differently in classes than normal functions, just as you saw in Tip 36, Prevent Context Confusion with Arrow Functions. You'll see how to use arrow functions in classes in Tip 42, Resolve Context Problems with Bind().

Speaking of context. You have full access to the `this` context in the methods if you call them directly on an instance of a class. This will work as predicted most of the time. You'll see the exceptions in upcoming tips.

This means that you can create methods that refer to properties or other methods.

```
class Coupon {
    constructor(price, expiration) {
        this.price = price;
        this.expiration = expiration || 'two weeks';
    }
    getPriceText() {
        return `${this.price}`;
    }
    getExpirationMessage() {
        return `This offer expires in ${this.expiration}.`;
    }
}
const coupon = new Coupon(5);
console.log(coupon.getPriceText());
console.log(coupon.getExpirationMessage());
```

Now that you have a very basic but useful class, you can create a new object using a constructor function that sets up a `this` binding. You can call methods and access properties. And everything uses an intuitive interface. The basics are

familiar, but it’s important you note the quirks, particularly in regard to setting properties. You’re building objects, so you’ll still encounter some context and scope issues.

In the next tip, you’ll see how to share code between classes using inheritance.