Lesson 7.2 - Class Definition and Inheritance

Let's discuss how to define classes, and how ES2015 introduces class inheritance shall we?

Class Definition

Defining a class is a very simple thing to do. It can be simple as:

```
class Vehicle {}
```

Of course, this class doesn't do much, and we can't initialize it with anything useful. Let's add a constructor:

```
class Vehicle {
  constructor(make, model, year) {
    Object.assign(this, { make, model, year });
  }
}
```

Ok cool, so what we've done here is simply set the variable names that are passed in when creating an instance of the class on `this`. You can create an instance of this class by using the new keyword:

```
var car = new Vehicle('Toyota', 'Corolla', 2016)
```

That's great, but it's still not very useful. One thing we can do is add a method to the class that make it a bit more useful:

```
class Vehicle {
  constructor(make, model, year) {
    Object.assign(this, { make, model, year });
  }

  print() {
    return `${this.year} ${this.make} ${this.model}`
  }
}
```

We made this class a bit more useful by adding a function that can be accessed from the instance of the class:

```
> var car = new Vehicle('Toyota', 'Corolla', 2016)
> car.print()
-> 2016 Toyota Corolla
```

Inheritance

ES2015 introduced class inheritance along with the `class` keyword. With inheritance, you can make a class `extend` another class, and thus inheriting all of it's properties. Let's take a look:

```
class Truck extends Vehicle {
  constructor(make, model, year, drive) {
    super(make, model, year)
    this.drive = drive
  }
}
```

What you see here is that we've created a new class called `Truck` that inherits from `Vehicle`. The `Vehicle` class here is known as the base class. It is called a base class because it does not inherit from any other class. Conversely, when referring to the `Vehicle` class from the perspective of the `Truck` class, it is know as the superclass. It is the prototype of the `Truck` class. Let's look at this in the REPL:

```
> Object.getPrototypeOf(Truck)
-> Vehicle
```

You can see here that the prototype of the `Truck` class is the `Vehicle` class. `Vehicle` is the superclass of `Truck`.

The `Truck` class will act in the same way as a `Vehicle` class, but requires one extra parameter during initialization.

```
> var truck = new Truck('Dodge', 'Ram', 2016, '4WD')
> truck.print()
-> 2016 Dodge Ram
```

You'll notice that in the constructor of the `Truck` class I used the `super` keyword. What this does is invoke the parent class' constructor method with the variables you pass into it.

Now that we know how to create a class, let's convert all the code we have been writing for our tax calculator and receipt printer over to a class based module. Let's open up the `calculator.js` file and replace the whole thing with our brand new class:

```
import { strip } from './strings'

export default class Calculator {
  constructor(items = [], taxRate = 8) {
    Object.assign(this, { items, taxRate });
  }

  calculateTax({ isTaxable, price }) {
    if (!isTaxable) return 0
    return price * this.taxRate / 100
  }

  calculate() {
    return this.items.reduce((prev, curr) => {
      prev.total += curr.price;
      prev.totalTax += this.calculateTax(curr);
      return prev;
    }, {
      total: 0,
      totalTax: 0
    });
  }

  printReceipt() {
    let {total, totalTax} = this.calculate()

    return strip`
      ${this.items.map(item => item.price).join('\n')}
      ${'-'.repeat(30)}
      Sub-Total: ${total}
      ${'='.repeat(30)}
      Tax: ${totalTax}
      ${'='.repeat(30)}
      Total: ${total + totalTax}
    `
  }
}
```

Cool! Now we can do this:

```
> var { items } = require('./calculator')
> var Calculator = require('./calculator').default
> var calculator = new Calculator(items)
> calculator.calculate()
-> { total: 45.45, totalTax: 1.876 }
> calculator.printReceipt()
->
10
22
13.45
------------------------------
```

```
Sub-Total: 45.45
==============================
Tax: 1.876
==============================
Total: 47.326
```

Cool! We’ve wrapped up all of that functionality into a class.