ES6 Classes

Here we introduce the basic ideas around ES6 classes, the new way of creating classes in Javascript.

Relevant links

- Exploring ES6, chapter 15

Notes

In past versions of Javascript the creation of "classes" was a bit of a hack, as by default there was no idea of a class: All you could do is create a "constructor" function, which created objects based on a "prototype" object.

While we may still discuss some of these aspects, the current way of implementing classes is via the new ES6 features that we will discuss in this section.

Classes are constructed using the `class` keyword. The `constructor` keyword is used to define the class constructor (only one constructor allowed in Javascript). Here is a simple example of defining a class for managing tasks:

```
class Task {
  constructor(title, dueDate = null, completed = false) {
    this.title = title;
    this.dueDate = dueDate;
    this.completed = completed;
  }
  isPastDue() {
    return this.dueDate !== null && this.dueDate < Date.now();
  }
  complete() {
    this.completed = true;
  }
}
```

We are also seeing here for the first time default values in a function. Note how the `dueDate` parameter defaults to `null`, while the `completed` parameter defaults to `false`. Note also the use of `this.title` etc to set the object's "properties". Let's look at a quick use of this:

```
let aTask = new Task("My awesome task!");
aTask.isPastDue();      // false
aTask.completed;        // false
aTask.complete();       // sets completed to true
aTask.completed;        // true now
```

As you can see, an object's properties are visible to the outside world. That's an important difference! We will see later how we can try to hide these properties.

We can also create subclasses. A `DatedTask` is a `Task` that has the current date as the default date. We use the keyword `super` to call the parent's methods:

```
class DatedTask extends Task {
    constructor(title, dueDate = null, completed = false) {
        if (dueDate === null) { dueDate = Date.now(); }
        super(title, dueDate, completed);
    }
}

let task2 = new DatedTask("my dated task!");
task2.dueDate;          // Now has a value!
task2.complete();       // Just works!
```

Question: Would the following work also?

```
class DatedTask extends Task {
    constructor(title, dueDate = Date.now(), completed = false) {
        super(title, dueDate, completed);
    }
}
```

We can also have static methods, if we prepend the method definition with the keyword static.

Practice

1. Create a class Countdown that represents "countdown" objects. Each countdown object contains two properties: A number count to count down from, and a function action to call when that count reaches 0. The counter provides a method dec that decrements the count by 1, and if that count has reached 0 then calls the function. The count should only be decremented if it is positive; the function should do nothing otherwise.
2. Amend your Countdown class to contain a static property that stores the number of countdown objects that have been created, along with a static method that returns that number. Whenever a new Countdown object is created, that number needs to increase by 1.