Object-oriented JS

Estimated time for completion: 30 minutes

Overview:

In this lab you will edit a simple HTML file that contains an HTML5 `<canvas>` element and some buttons that can add shapes to it. You'll define the "classes" that implement the drawing functionality for those shapes.

Goals:

- Be comfortable simulating classes with constructor functions and prototypes
- Implement the "classes" to actually draw on the canvas

A Hierarchy of Shapes

Implement constructor functions for the shapes corresponding to the buttons in the HTML.

Criteria:

- Clicking on the buttons should result in a shape being added to the canvas.

Steps:

1. Open functional.html in the `~/before` folder in your favorite browser. You should be able to "see" the empty canvas and the buttons to its left.
    a. Clicking on the buttons doesn't do anything. This is what you'll fix.
2. Open functional.html in the `~/before` in your favorite text editor. There's both HTML and JavaScript in this file, but no TODO comments.
    a. Scroll to the bottom and notice how the HTML includes both helpers.js and objects.js.
    b. You can examine helpers.js to see the code that you can use to actually perform the drawing on the canvas. This file just contains a single object with some helper methods. You won't be changing anything in this file, but it'll be important to see how each of the methods gets invoked.
    c. You'll be doing your work in objects.js. Open this file up in your favorite text editor and notice the complete lack of any code and just some terse instructions.

3. Create a “base class” for your shapes. You can call it something generic like `DisplayObject`.
    a. Remember, JavaScript doesn’t have classes (yet) so you’ll be defining a constructor function instead of something that looks like a class.
    b. In this constructor function, assign a generic name to `this.name` and also provide a `clicked` method on the constructor function’s prototype. You can just alert `this.name` inside that `clicked` method.
4. Start implementing the `Rect` derived “classes”.
    a. The `Rect` constructor should take in an options object. This is passed in from the code in objects.html which it gets from the “data-” attributes on the button elements in the actual HTML. These options will all be strings (that’s the nature of HTML attributes) so you may need to convert some to numbers.
    b. Rect instances will need `draw` and `contains` methods.
    c. The `draw` method will be passed a canvas 2D context passed in to it. You’ll be passing this object on to the methods in helpers.js. The other arguments required by the methods in helpers.js will come from the options object passed in to your constructor.
    d. The `contains` method be passed `x` and `y` numbers. They represent a point on the canvas and this `contains` method needs to return `true` or `false` depending on whether that point is actually contained by the rectangle that gets drawn onto the canvas.
    e. Make sure your `Rect` “class” “derives” from the `DisplayObject` “class”. You’ll need to override the `name` property you initialized on this in the `DisplayObject` constructor function to something more appropriate for rectangles (something like “rectangle”).
5. Do the same for a `Circle` and a `Logo` class.
6. The code in the `~/after` directory uses different styles of “classes” for each display object so you can look in there to see the different options. You can use prototypes or not for each class, but you should be comfortable with the different options.
7. Test your objects out by clicking the buttons in the page and seeing if any shapes appear. Try clicking on the shapes to ensure your `contains` methods are working correctly and that your objects did “inherit” the `clicked` method from their base “class”.

Solutions:

The final solution for this lab is available in the `~/after` directory.