**setUp() and tearDown()**

Like its cousin JUnit, JsUnit supports `setUp()` and `tearDown()`. The JsUnit flavor is similar to JUnit in that `setUp()` and `tearDown()` are optional, with `setUp()` being called before every test and `tearDown()` being called after every test. Before you run off and start using `setUp()` and `tearDown()` extensively, you need to know about the two key differences between the JUnit and JsUnit implementations of the `setUp()` and `tearDown()` methods. In JUnit, each test run results in the creation of a new instance of the Test class, meaning any instance variables that are declared are “reset” between test runs. JsUnit, however, does not reload the test page for each test run, so the variable state will be preserved across tests. The other key difference concerns test order. With JUnit, the order your tests are executed in is not guaranteed. In JsUnit, tests will be executed in the order they are declared in your test page, starting at the top.³

Listing 6-5 shows a rather contrived example of using the `setUp()` and `tearDown()` methods. This builds on the simple `add` method you created before, but this time you'll add a form to the equation. You’ll use the `setUp()` and `tearDown()` methods to populate the form and then clean up after yourself.

Listing 6-5. Using `setUp()` and `tearDown()`

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>
  <head>
    <title>Using setUp and tearDown</title>
    <script language="JavaScript" src="jsunit/app/jsUnitCore.js"></script>
    <script language="JavaScript" src="simpleJS.js"></script>
    <script language="JavaScript">
      function setUp() {
        document.getElementById("value1").value = "2";
        document.getElementById("value2").value = "2";
      }
      function testValidArgs() {
        assertEquals("2 + 2 should equal 4", 4, addNumbers());
      }
      function addNumbers() {
        var val1 = document.getElementById("value1").value;
        var val2 = document.getElementById("value2").value;
        return addTwoNumbers(val1, val2);
      }
    </script>
  </head>
  <body>
    <form>
      <input type="text" id="value1"><br>
      <input type="text" id="value2"><br>
      <input type="button" value="Add numbers">
    </form>
    <p>Valid arguments: 2 + 2 should equal 4</p>
  </body>
</html>
```

³. The normal caveats about tests depending on each other still apply, though. Just because your tests will be executed in a defined order doesn't imply you should write tests that depend on execution order!