Instructor Notes:

Add instructor notes here.

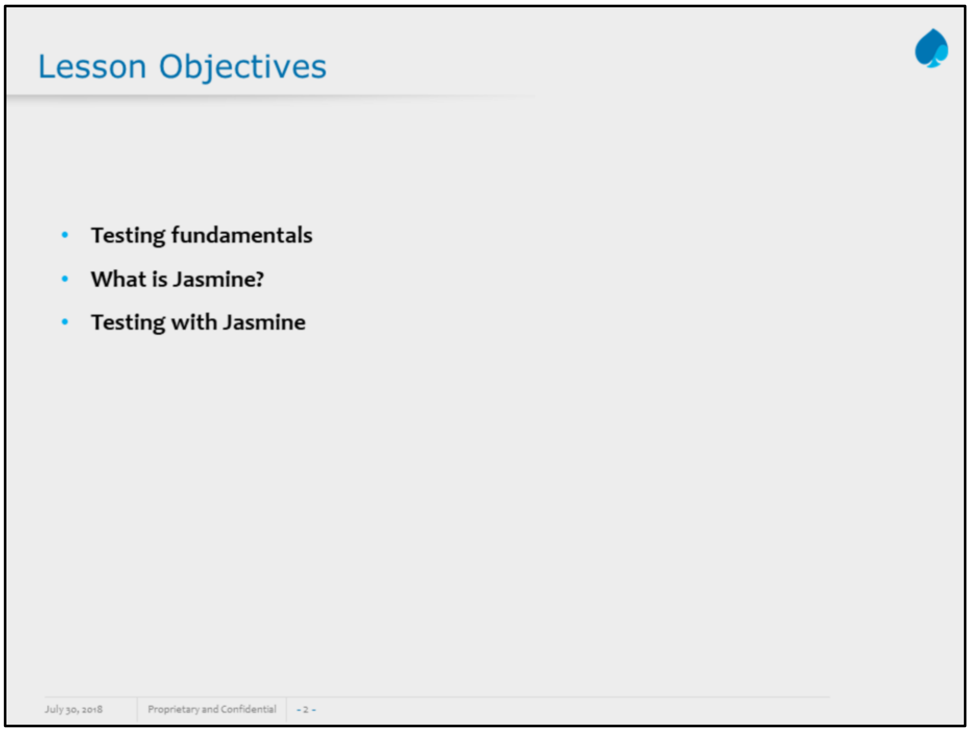
JavaScript testing with Jasmine

Lesson-1

Capgemini

Instructor Notes:

Add instructor notes here.



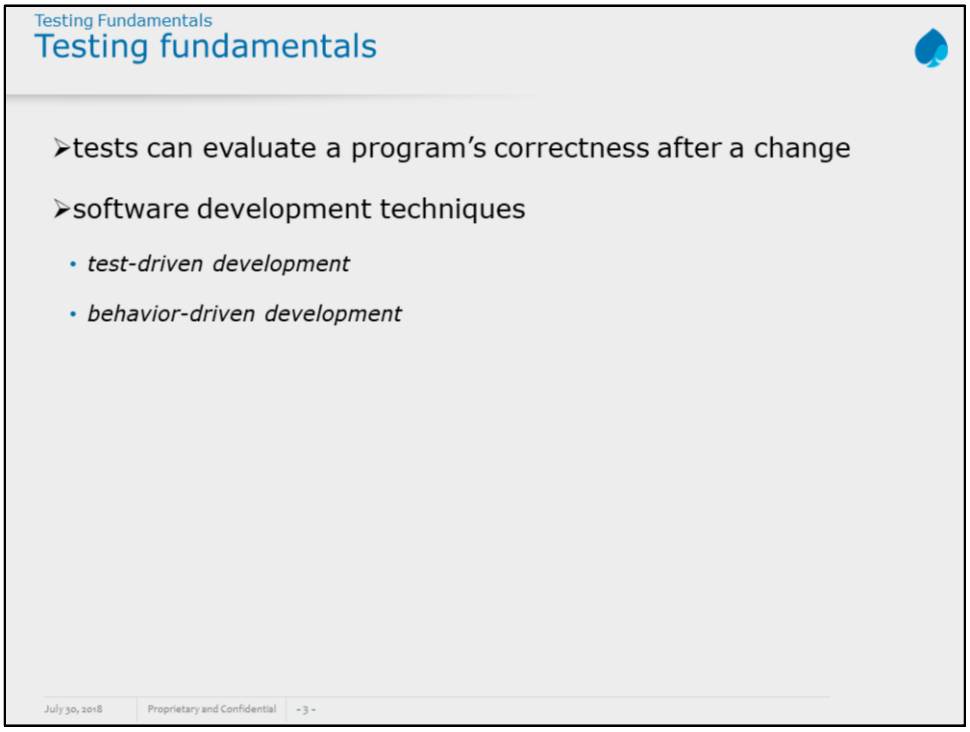
Lesson Objectives

- Testing fundamentals
- What is Jasmine?
- Testing with Jasmine

July 30, 2018 | Proprietary and Confidential | - 2 -

Instructor Notes:

Add instructor notes here.



Testing Fundamentals

Testing fundamentals

➢tests can evaluate a program's correctness after a change

➢software development techniques

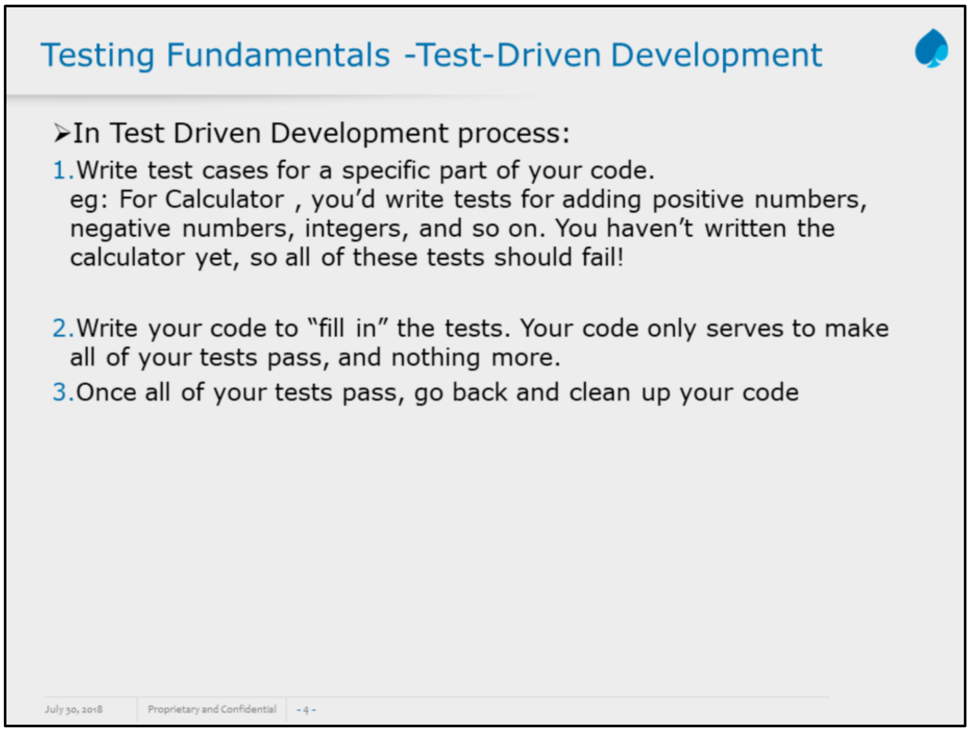
- *test-driven development*
- *behavior-driven development*

July 30, 2018 Proprietary and Confidential - 3 -

Testing in a nutshell: basically, your program will have a bunch of functions and classes. You want to make sure that, no matter what you throw at them, they'll perform how you want them to. For example, this function should always return a string that says "hello" in it. Testing ensures that everything goes down exactly how you planned.

Instructor Notes:

Add instructor notes here.



Testing Fundamentals -Test-Driven Development

➢In Test Driven Development process:

1. Write test cases for a specific part of your code.
   eg: For Calculator , you'd write tests for adding positive numbers, negative numbers, integers, and so on. You haven't written the calculator yet, so all of these tests should fail!
2. Write your code to "fill in" the tests. Your code only serves to make all of your tests pass, and nothing more.
3. Once all of your tests pass, go back and clean up your code

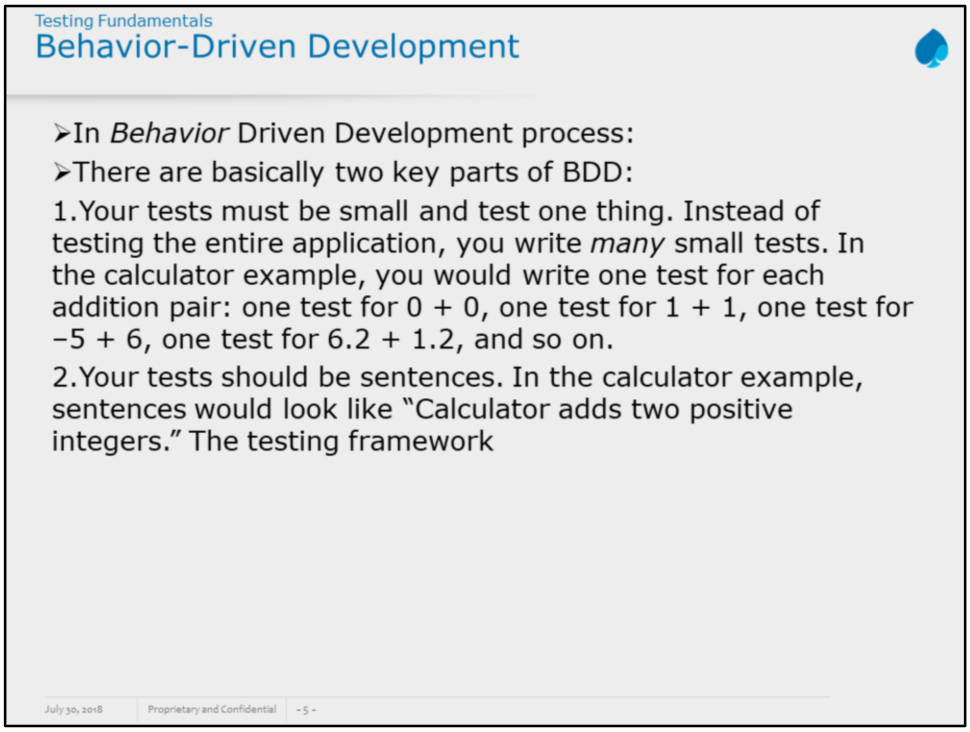
July 30, 2018 | Proprietary and Confidential | - 4 -

TDD in its simplest form is just this:

1. Write your tests
2. Watch them fail
3. Make them pass
4. Refactor
5. Repeat

Instructor Notes:

Add instructor notes here.



Testing Fundamentals

Behavior-Driven Development

- In *Behavior* Driven Development process:
- There are basically two key parts of BDD:

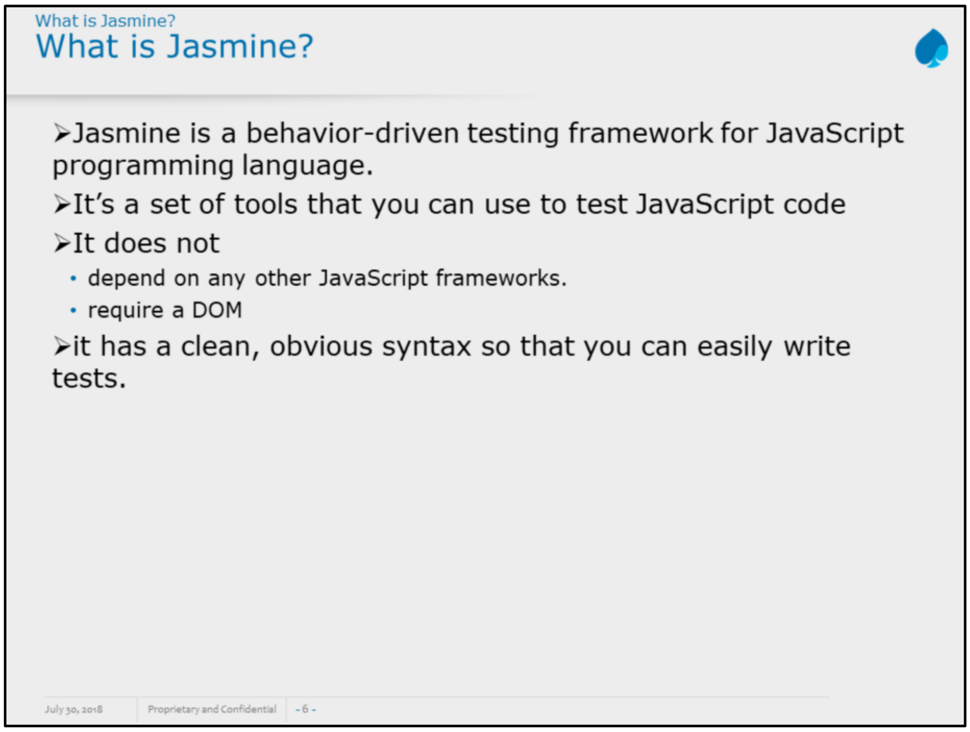
1. Your tests must be small and test one thing. Instead of testing the entire application, you write *many* small tests. In the calculator example, you would write one test for each addition pair: one test for 0 + 0, one test for 1 + 1, one test for −5 + 6, one test for 6.2 + 1.2, and so on.
2. Your tests should be sentences. In the calculator example, sentences would look like "Calculator adds two positive integers." The testing framework

July 30, 2018 Proprietary and Confidential - 5 -

- Establishing the goals of different stakeholders required for a vision to be implemented
- Involving stakeholders in the implementation process through outside-in software development
- Using examples to describe the behavior of the application, or of units of code
- Automating those examples to provide quick feedback and regression testing

Instructor Notes:

Add instructor notes here.



What is Jasmine?

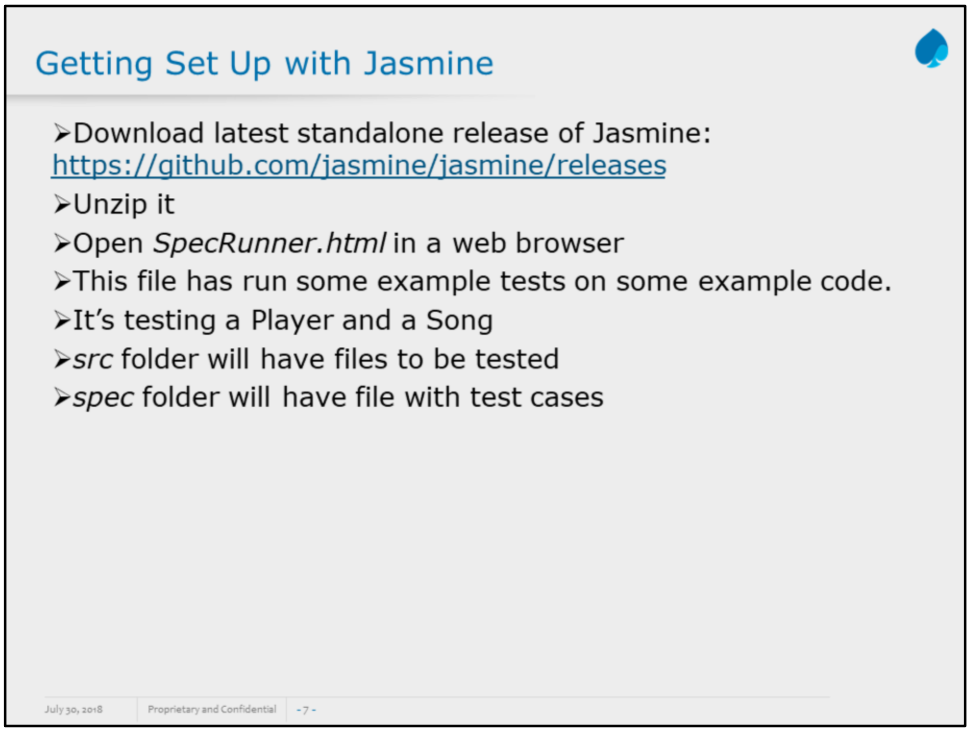
What is Jasmine?

- ➢Jasmine is a behavior-driven testing framework for JavaScript programming language.
- ➢It's a set of tools that you can use to test JavaScript code
- ➢It does not
  - depend on any other JavaScript frameworks.
  - require a DOM
- ➢it has a clean, obvious syntax so that you can easily write tests.

July 30, 2018 | Proprietary and Confidential | - 6 -

Instructor Notes:

Add instructor notes here.



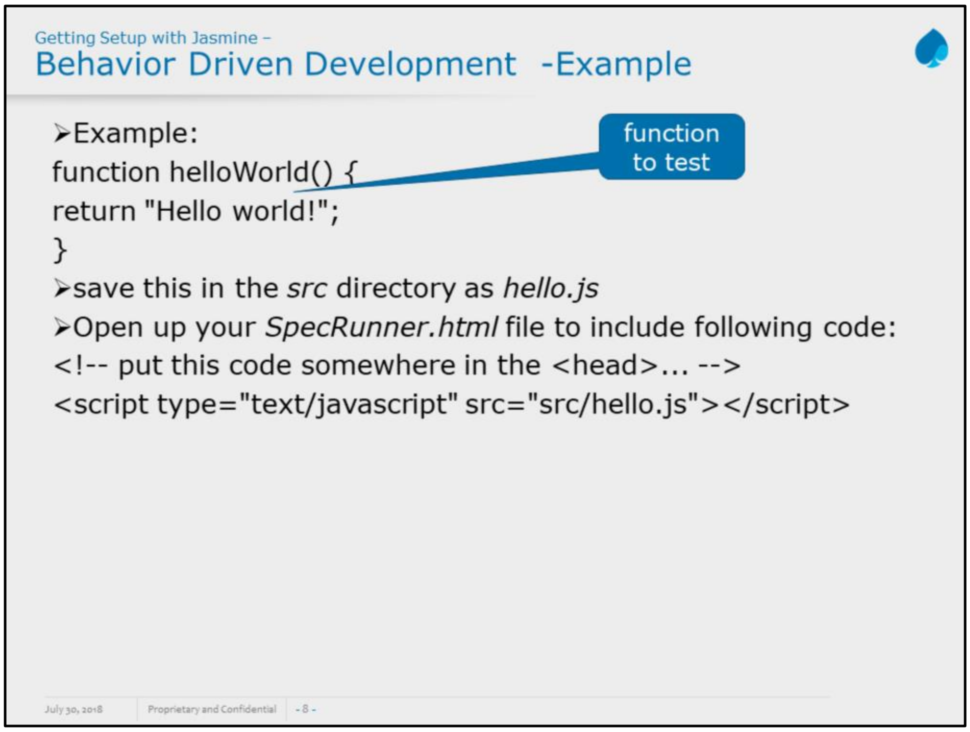
Getting Set Up with Jasmine

- ➢Download latest standalone release of Jasmine: https://github.com/jasmine/jasmine/releases
- ➢Unzip it
- ➢Open *SpecRunner.html* in a web browser
- ➢This file has run some example tests on some example code.
- ➢It's testing a Player and a Song
- ➢*src* folder will have files to be tested
- ➢*spec* folder will have file with test cases

July 30, 2018 | Proprietary and Confidential | - 7 -

Instructor Notes:

Add instructor notes here.



Getting Setup with Jasmine –

Behavior Driven Development -Example

➢Example:

```
function helloWorld() {
return "Hello world!";
}
```

function to test

➢save this in the *src* directory as *hello.js*

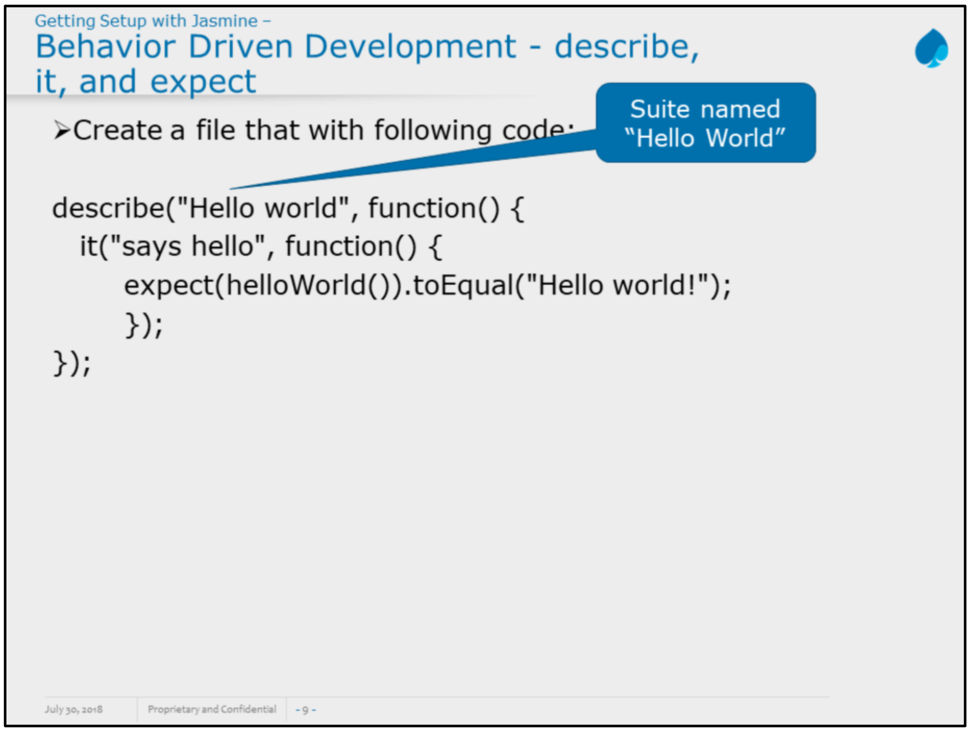
➢Open up your *SpecRunner.html* file to include following code:

```
<!-- put this code somewhere in the <head>... -->
<script type="text/javascript" src="src/hello.js"></script>
```

July 30, 2018 Proprietary and Confidential - 8 -

Instructor Notes:

Add instructor notes here.



Getting Setup with Jasmine –

Behavior Driven Development - describe, it, and expect

➢Create a file that with following code:

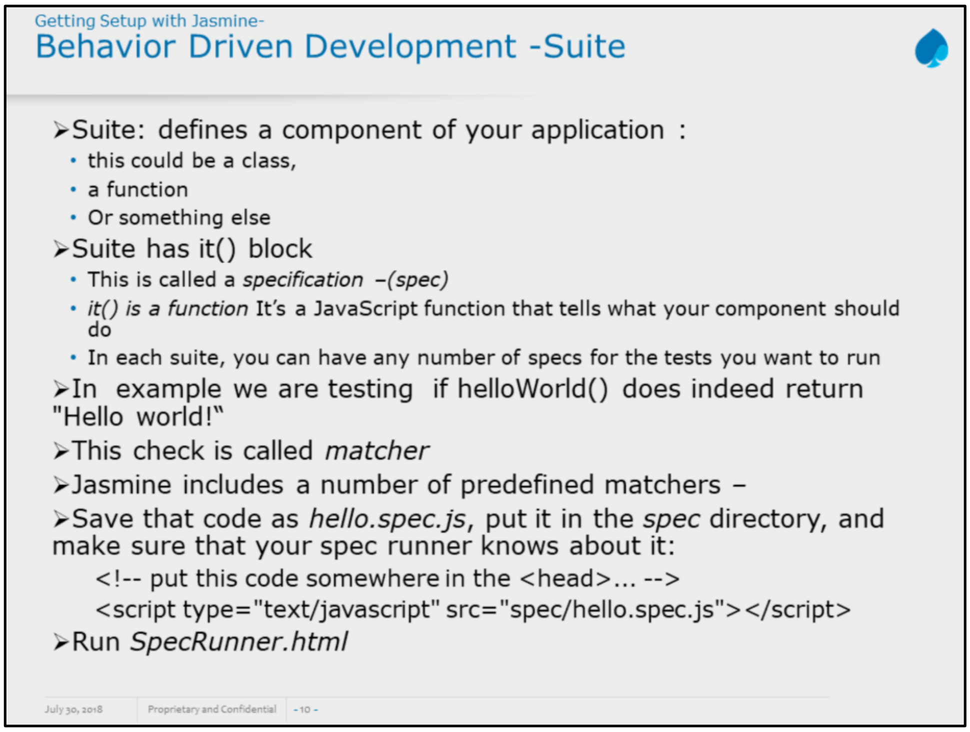
Suite named "Hello World"

```
describe("Hello world", function() {
  it("says hello", function() {
      expect(helloWorld()).toEqual("Hello world!");
      });
});
```

July 30, 2018 | Proprietary and Confidential | - 9 -

Instructor Notes:

Add instructor notes here.



Getting Setup with Jasmine-

Behavior Driven Development -Suite

- Suite: defines a component of your application :
  - this could be a class,
  - a function
  - Or something else
- Suite has it() block
  - This is called a *specification –(spec)*
  - *it() is a function* It's a JavaScript function that tells what your component should do
  - In each suite, you can have any number of specs for the tests you want to run
- In example we are testing if helloWorld() does indeed return "Hello world!"
- This check is called *matcher*
- Jasmine includes a number of predefined matchers –
- Save that code as *hello.spec.js*, put it in the *spec* directory, and make sure that your spec runner knows about it:

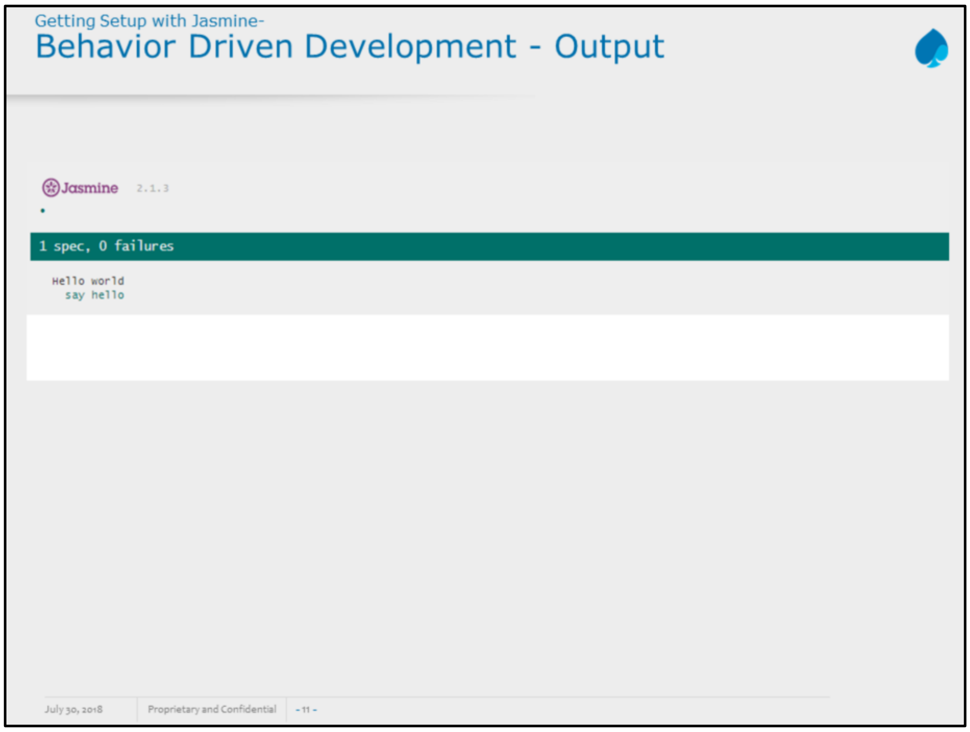
```
<!-- put this code somewhere in the <head>... -->
<script type="text/javascript" src="spec/hello.spec.js"></script>
```

- Run *SpecRunner.html*

July 30, 2018 Proprietary and Confidential - 10 -

Instructor Notes:

Add instructor notes here.



Getting Setup with Jasmine-

Behavior Driven Development - Output

Jasmine 2.1.3

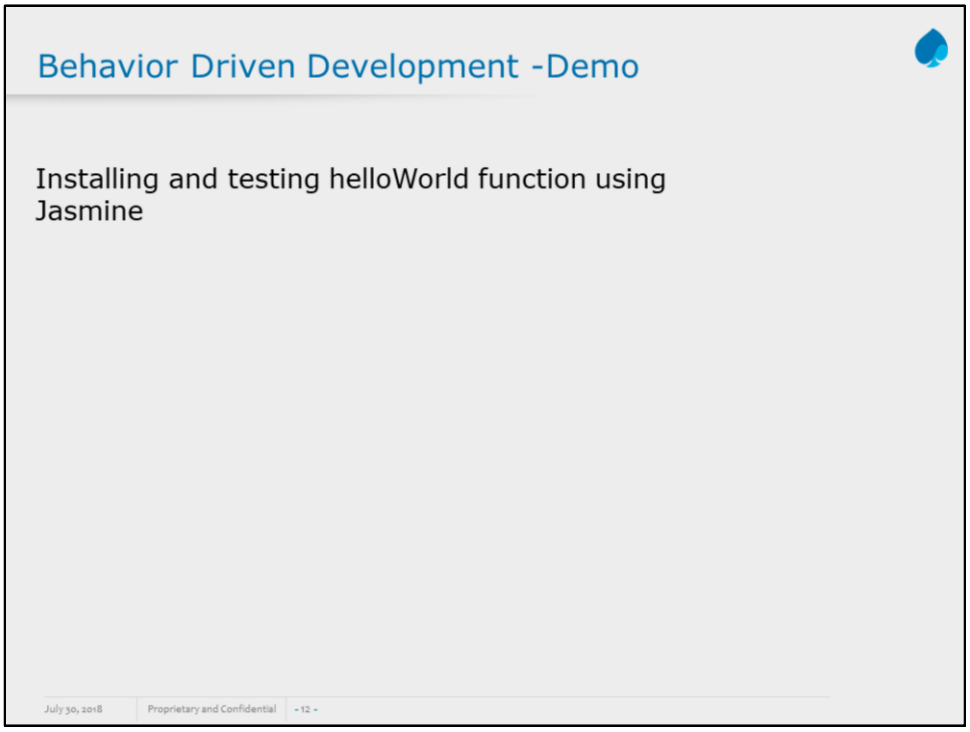
```
•
1 spec, 0 failures

  Hello world
    say hello
```

July 30, 2018 | Proprietary and Confidential | - 11 -

Instructor Notes:

Add instructor notes here.



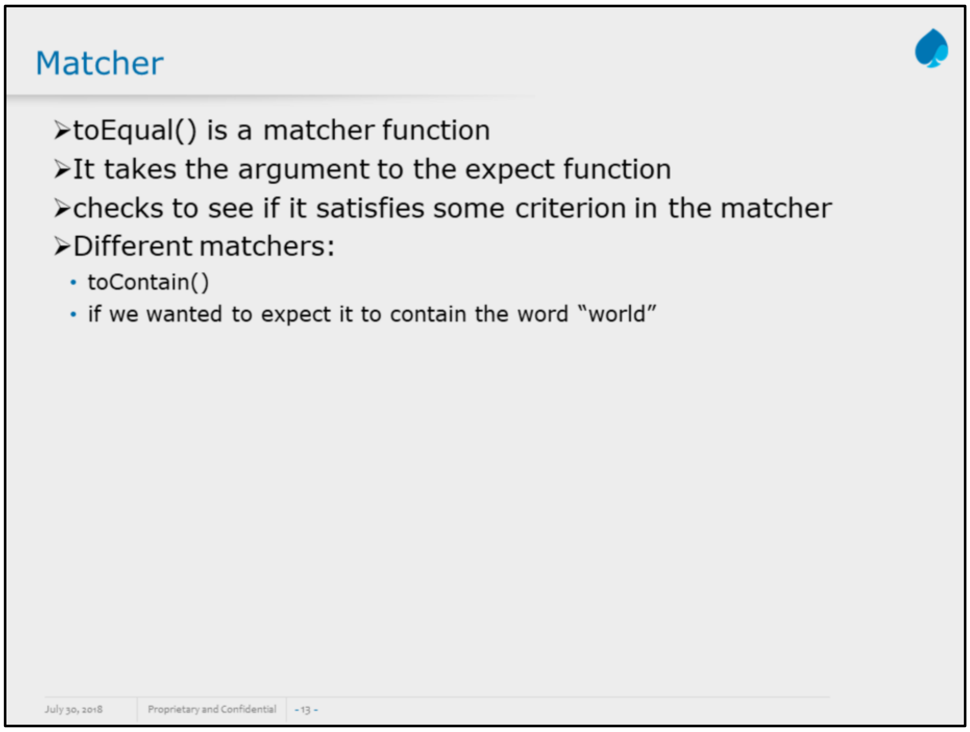
Behavior Driven Development -Demo

Installing and testing helloWorld function using Jasmine

July 30, 2018 | Proprietary and Confidential | - 12 -

Instructor Notes:

Add instructor notes here.



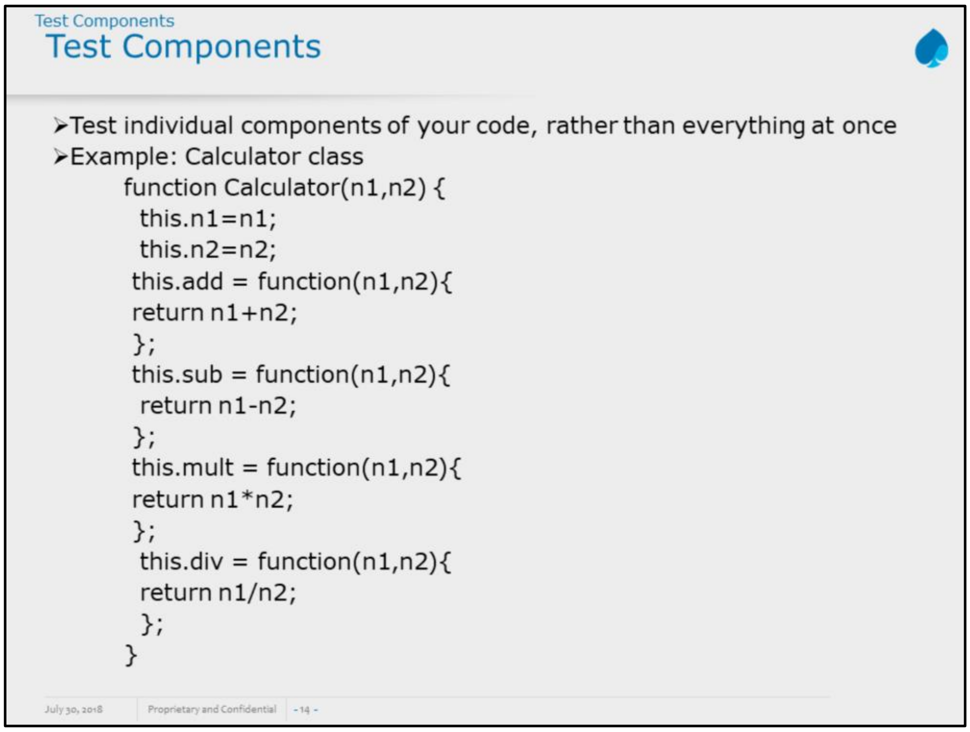
Matcher

- ➢toEqual() is a matcher function
- ➢It takes the argument to the expect function
- ➢checks to see if it satisfies some criterion in the matcher
- ➢Different matchers:
  - toContain()
  - if we wanted to expect it to contain the word "world"

July 30, 2018 | Proprietary and Confidential | - 13 -

Instructor Notes:

Add instructor notes here.


Test Components

Test Components

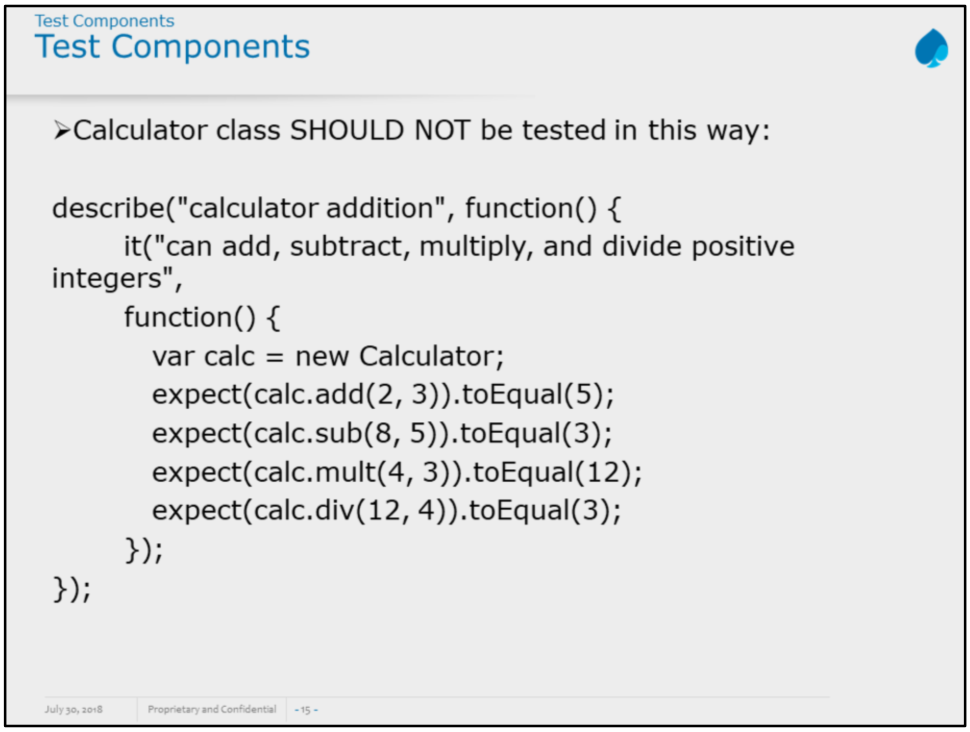
- ➢Test individual components of your code, rather than everything at once
- ➢Example: Calculator class

```
function Calculator(n1,n2) {
  this.n1=n1;
  this.n2=n2;
 this.add = function(n1,n2){
 return n1+n2;
 };
 this.sub = function(n1,n2){
  return n1-n2;
 };
 this.mult = function(n1,n2){
 return n1*n2;
 };
  this.div = function(n1,n2){
  return n1/n2;
  };
}
```

July 30, 2018 | Proprietary and Confidential | - 14 -

Instructor Notes:

Add instructor notes here.



Test Components

Test Components

➢Calculator class SHOULD NOT be tested in this way:

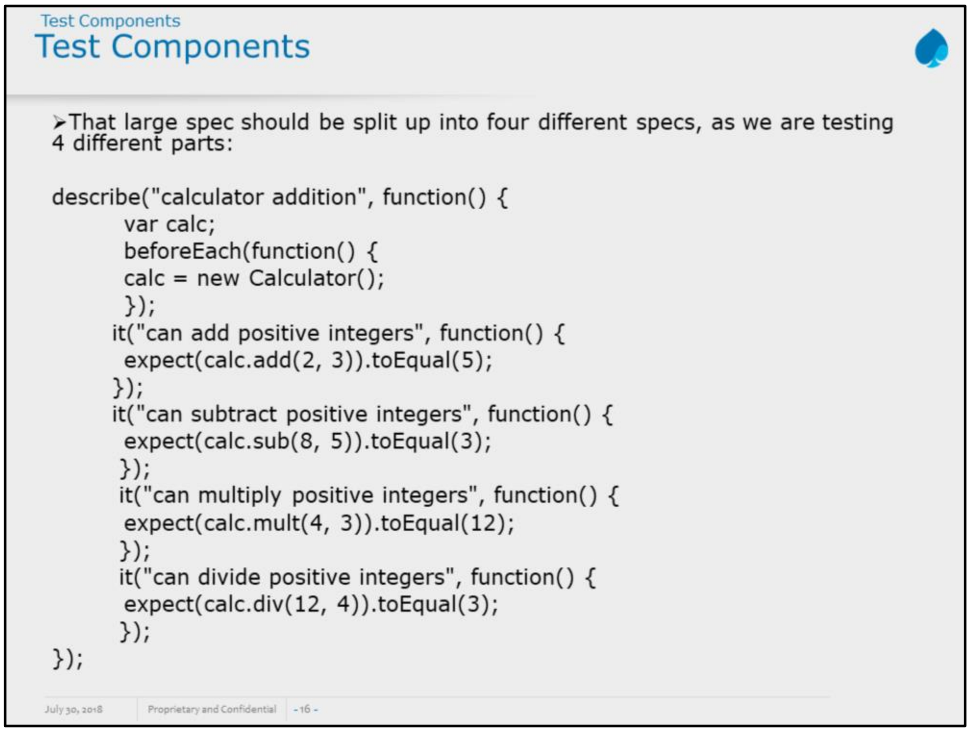
```
describe("calculator addition", function() {
      it("can add, subtract, multiply, and divide positive
integers",
      function() {
         var calc = new Calculator;
         expect(calc.add(2, 3)).toEqual(5);
         expect(calc.sub(8, 5)).toEqual(3);
         expect(calc.mult(4, 3)).toEqual(12);
         expect(calc.div(12, 4)).toEqual(3);
      });
});
```

July 30, 2018 | Proprietary and Confidential | - 15 -

- The describe(string, function) function defines what we call a *Test Suite*, a collection of individual *Test Specs*.
- The it(string, function) function defines an individual *Test Spec*, this contains one or more *Test Expectations*.
- The expect(actual) expression is what we call an *Expectation*.
  In conjunction with a *Matcher* it describes an *expected* piece of behaviour in the application.
- The matcher(expected) expression is what we call a *Matcher*. It does a boolean comparison with the expected value passed in vs. the actual value passed to the expect function, if they are false the *spec* fails.

Instructor Notes:

Add instructor notes here.



Test Components

Test Components

➢That large spec should be split up into four different specs, as we are testing 4 different parts:

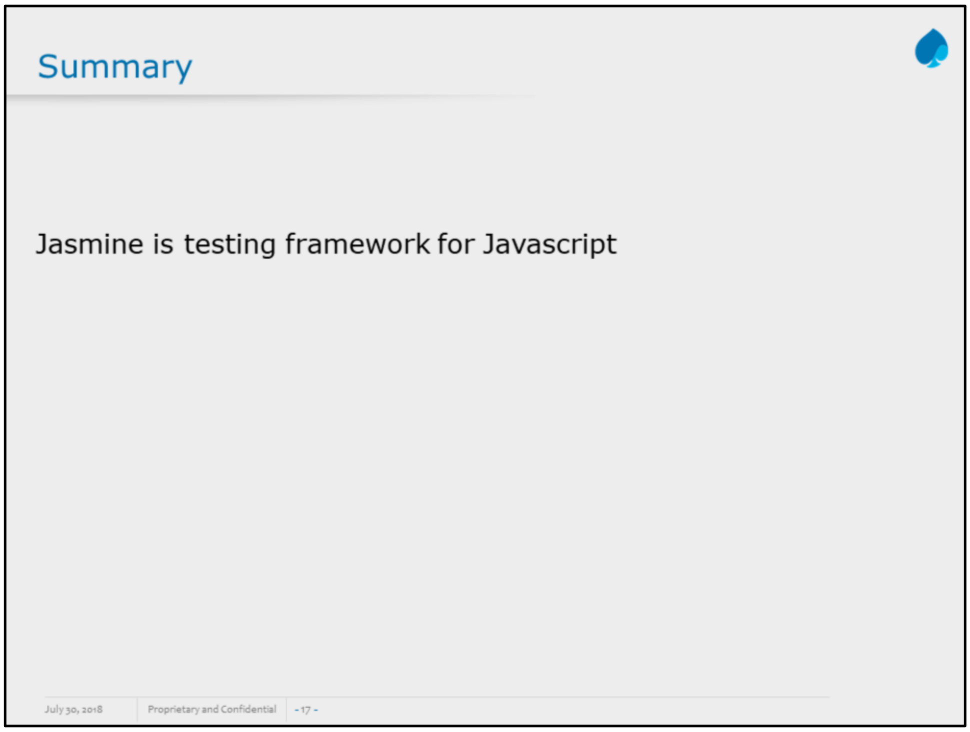
```
describe("calculator addition", function() {
        var calc;
        beforeEach(function() {
        calc = new Calculator();
        });
       it("can add positive integers", function() {
        expect(calc.add(2, 3)).toEqual(5);
       });
       it("can subtract positive integers", function() {
        expect(calc.sub(8, 5)).toEqual(3);
        });
        it("can multiply positive integers", function() {
        expect(calc.mult(4, 3)).toEqual(12);
        });
        it("can divide positive integers", function() {
        expect(calc.div(12, 4)).toEqual(3);
        });
});
```

July 30, 2018 Proprietary and Confidential - 16 -

Each spec should test only one case or scenario at a time. In the previous example, if you had an error in your mult function, the spec would fail even if the other components worked perfectly. In this example, only one test will fail, and you'll be able to more quickly pinpoint that your multiplication is broken.

Instructor Notes:

Add instructor notes here.



Summary

Jasmine is testing framework for Javascript

July 30, 2018 | Proprietary and Confidential | - 17 -

Add the notes here.