Module-4 Discussion: Test Driven Development, Unit Testing and Eclipse

In this discussion my hope to add another tool to your development arsenal: the unit test and test-driven development. These are software engineering principles that help us write correct code. Along with the trusty debugger, unit tests provide a framework for development that focuses on writing simple, correct code.

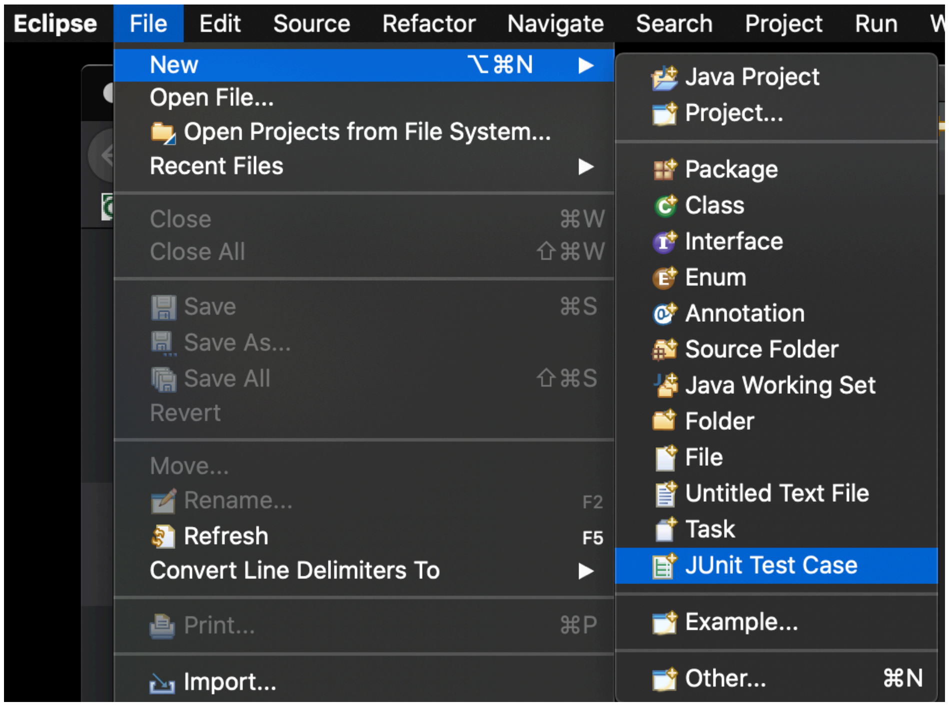
Before beginning these activities read the documents in the ***reading*** folder inside ***module-4***

Also . . . consume the following videos, articles and tutorials on Unit Testing and JUnit with Eclipse.

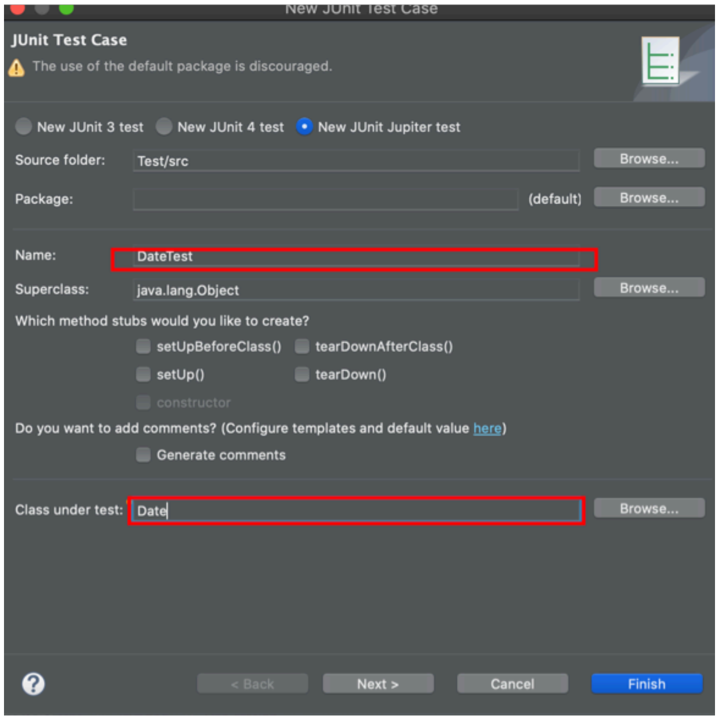
- **Test Driven Development:** http://www.agiledata.org/essays/tdd.html
- **Unit Testing Video:** https://www.youtube.com/watch?v=jFQfuIEd8sU&feature=emb_rel_end
- **Unit Testing:** https://www.guru99.com/unit-testing-guide.html
- **Unit Testing with Eclipse:** https://www.codejava.net/testing/junit-tutorial-for-beginner-with-eclipse

Let's write some unit tests for the Date class.

1. Start Eclipse and create a new Java project
2. Import ***Date.java*** from the ***code*** directory inside ***module-4***
3. Add JUnit to the current Eclipse project by selecting **File >> New >> JUnit Test Case**



4. Name the JUnit file ***DateTest*** and tell JUnit that the ***Date*** class is what we are testing



New JUnit Test Case

JUnit Test Case

⚠ The use of the default package is discouraged.

New JUnit 3 test New JUnit 4 test [✓ New JUnit Jupiter test]

Source folder: Test/src

Package: (default)

Name: DateTest

Superclass: java.lang.Object

Which method stubs would you like to create?

☐ setUpBeforeClass() ☐ tearDownAfterClass()

☐ setUp() ☐ tearDown()

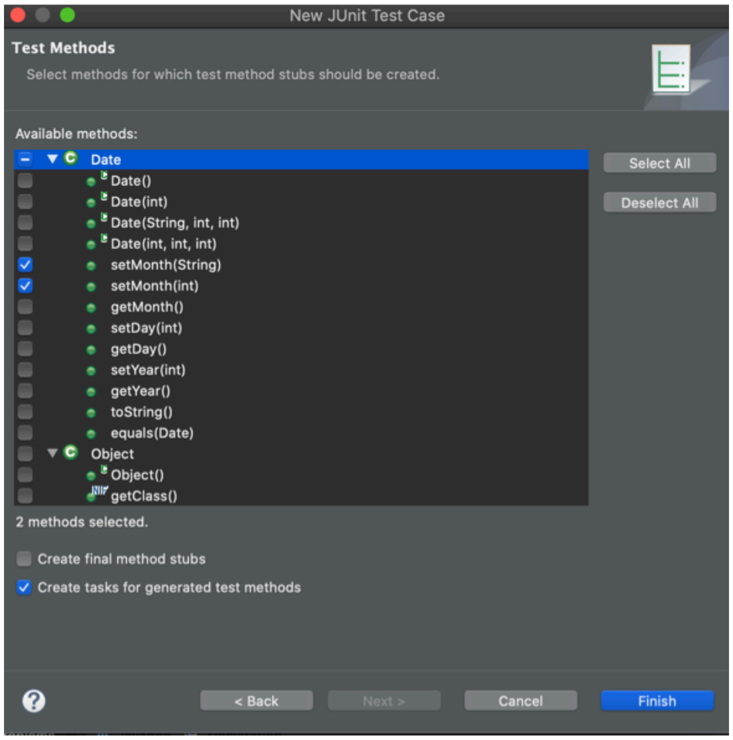
☐ constructor

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Class under test: Date

5. Deciding what to test is up to the context of the application. When testing an object's interface, we really only want to test those methods which perform some sort of domain validation. If a method only sets a variable with no logic, there is no point testing it. When you click ***Next >*** Eclipse will allow you to choose the methods you'd like to test. I will demonstrate a single method and leave the rest as an exercise. I'm going to choose the overloaded ***setMonth*** methods.



New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

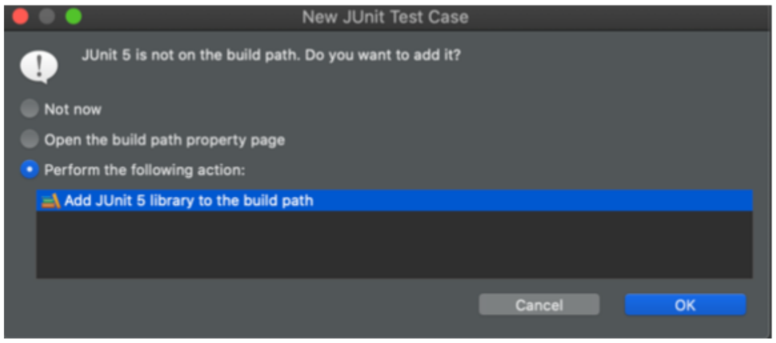
- Date
  - ☐ Date()
  - ☐ Date(int)
  - ☐ Date(String, int, int)
  - ☐ Date(int, int, int)
  - ✓ setMonth(String)
  - ✓ setMonth(int)
  - ☐ getMonth()
  - ☐ setDay(int)
  - ☐ getDay()
  - ☐ setYear(int)
  - ☐ getYear()
  - ☐ toString()
  - ☐ equals(Date)
- Object
  - ☐ Object()
  - ☐ getClass()

2 methods selected.

☐ Create final method stubs

✓ Create tasks for generated test methods

6. Click ***Finish.*** Eclipse will now ask if you would like to add JUnit to your project's build path. You definitely want this to happen, although you could in fact configure this after the fact. Click OK

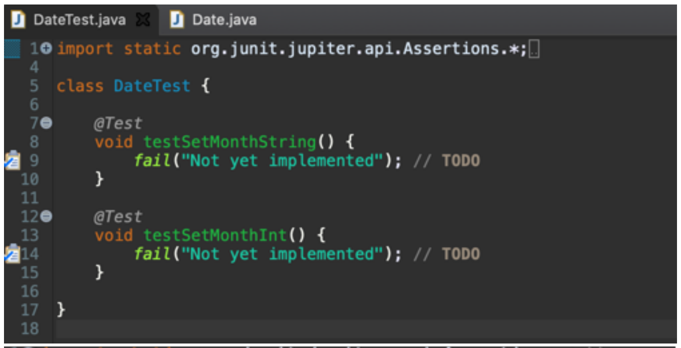


New JUnit Test Case

JUnit 5 is not on the build path. Do you want to add it?

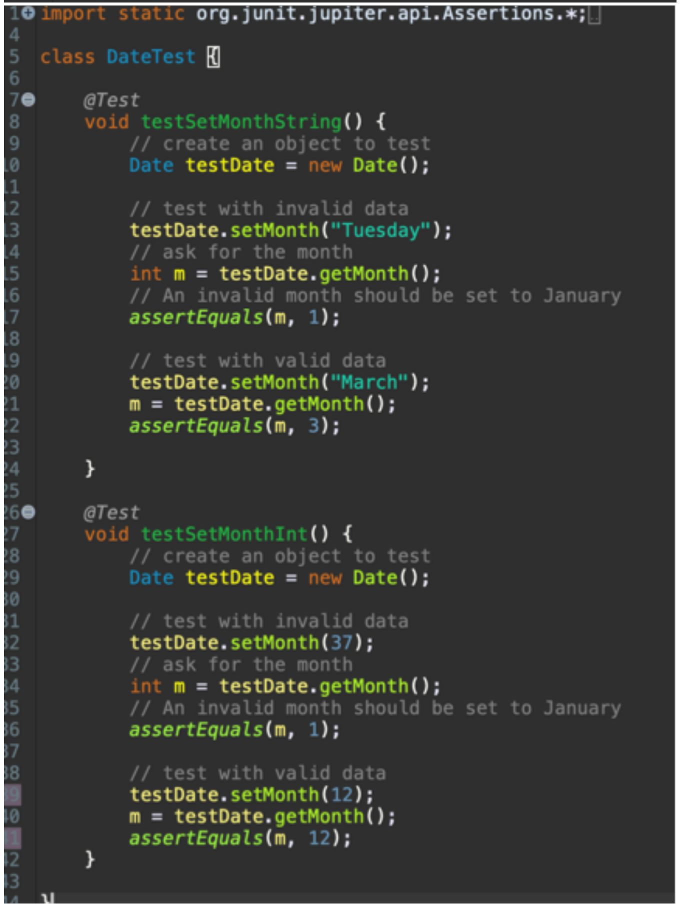
Not now
Open the build path property page
Perform the following action:
Add JUnit 5 library to the build path

7. You will now see a new file appear in your Package Explorer. This is the JUnit test file. There are two method stubs that Eclipse wrote for you. Obviously, these tests are configured to fail out of the box, because there is no code for them to actually test. Let's write these tests.



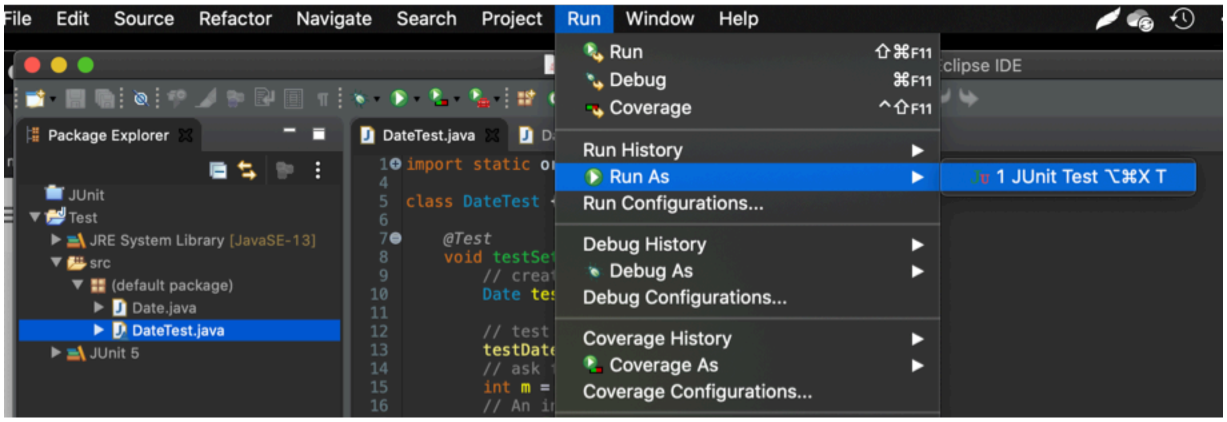
```java
import static org.junit.jupiter.api.Assertions.*;

class DateTest {

    @Test
    void testSetMonthString() {
        fail("Not yet implemented"); // TODO
    }

    @Test
    void testSetMonthInt() {
        fail("Not yet implemented"); // TODO
    }

}
```

8. Both of these methods perform domain validation on the value they are setting. We want to ensure that these methods can accurately determine that an argument violates our domain policies, and that it responds accordingly. For the ***setMonth*** methods we can pass in some invalid data and test our object's response. Any invalid month should be set to January (1). This is our policy. We can see how our Object transformed the input by calling ***getMonth*** to see what is returned. Notice the JUnit test method ***assertEquals***. This method will assert that the two values provided as arguments are equal. If the values are equal this test method will give you a pass. If the values are not equal the assertion will report a failure.

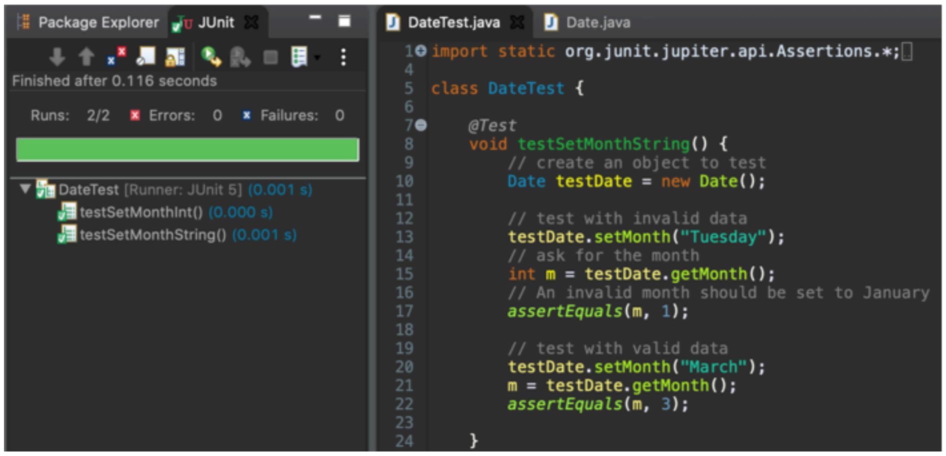


```java
import static org.junit.jupiter.api.Assertions.*;

class DateTest {

    @Test
    void testSetMonthString() {
        // create an object to test
        Date testDate = new Date();

        // test with invalid data
        testDate.setMonth("Tuesday");
        // ask for the month
        int m = testDate.getMonth();
        // An invalid month should be set to January
        assertEquals(m, 1);

        // test with valid data
        testDate.setMonth("March");
        m = testDate.getMonth();
        assertEquals(m, 3);

    }

    @Test
    void testSetMonthInt() {
        // create an object to test
        Date testDate = new Date();

        // test with invalid data
        testDate.setMonth(37);
        // ask for the month
        int m = testDate.getMonth();
        // An invalid month should be set to January
        assertEquals(m, 1);

        // test with valid data
        testDate.setMonth(12);
        m = testDate.getMonth();
        assertEquals(m, 12);
    }

}
```

9. There are a couple of ways to execute your JUnit tests. With your unit test file selected, navigate to the Run and select ***Run As -> JUnit Test***



You can also right click (or CMD+Click for Mac users) the JUnit file and select ***Run As JUnit Test***. This will open the JUnit tab next to the Package Explorer. This pane will show the results of your testing. For this example, we got all passes. You can see the summary information listed. There were two tests executed, 0 errors and 0 Failures. The green bar indicates success (This may be an issue for color blind humans, so focus on the quantity of failures for accurate reporting)

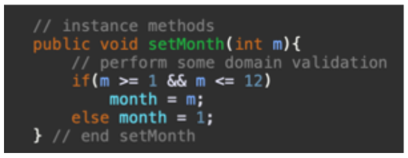


```
Finished after 0.116 seconds

Runs:  2/2    Errors:  0    Failures:  0

DateTest [Runner: JUnit 5] (0.001 s)
    testSetMonthInt() (0.000 s)
    testSetMonthString() (0.001 s)
```

```java
import static org.junit.jupiter.api.Assertions.*;

class DateTest {

    @Test
    void testSetMonthString() {
        // create an object to test
        Date testDate = new Date();

        // test with invalid data
        testDate.setMonth("Tuesday");
        // ask for the month
        int m = testDate.getMonth();
        // An invalid month should be set to January
        assertEquals(m, 1);

        // test with valid data
        testDate.setMonth("March");
        m = testDate.getMonth();
        assertEquals(m, 3);

    }
```

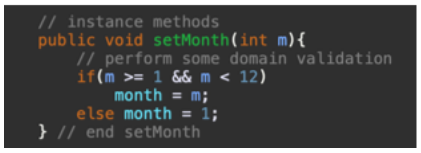
10. So . . . what do failures look like? To demonstrate this, I will purposefully introduce an ***off by one error*** in my month validation. Notice the subtle difference. The incorrect method will not identify month 12 as being correct.

Correct



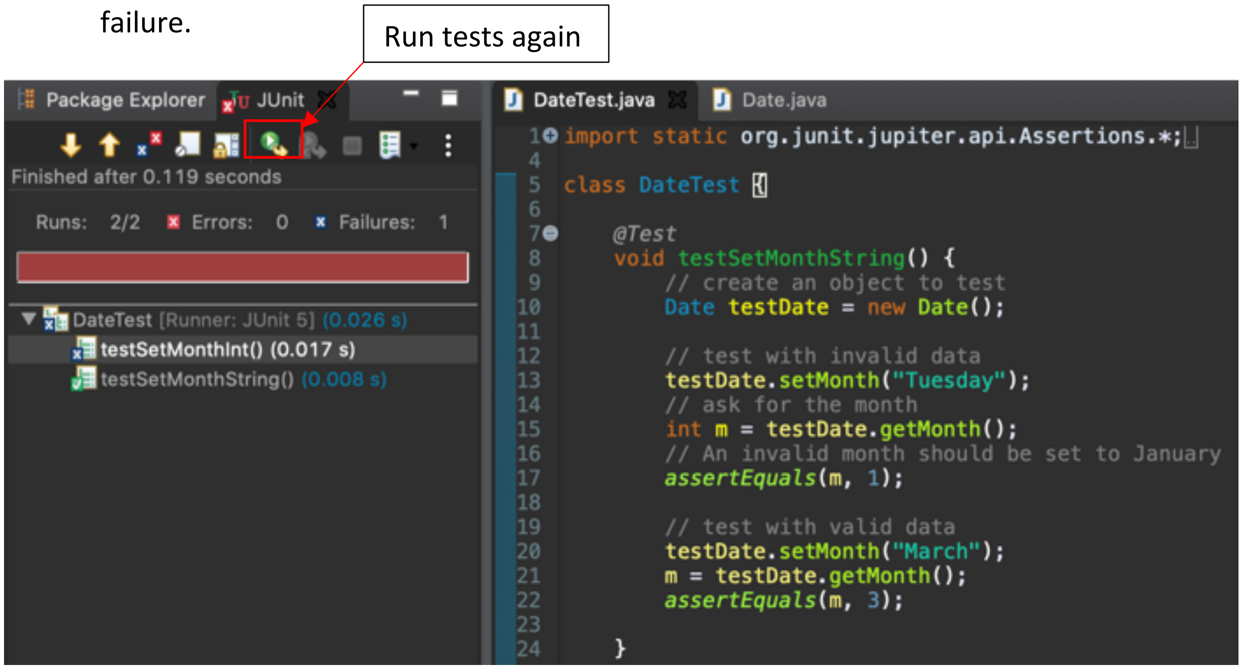
```java
// instance methods
public void setMonth(int m){
    // perform some domain validation
    if(m >= 1 && m <= 12)
        month = m;
    else month = 1;
} // end setMonth
```

Incorrect



```java
// instance methods
public void setMonth(int m){
    // perform some domain validation
    if(m >= 1 && m < 12)
        month = m;
    else month = 1;
} // end setMonth
```

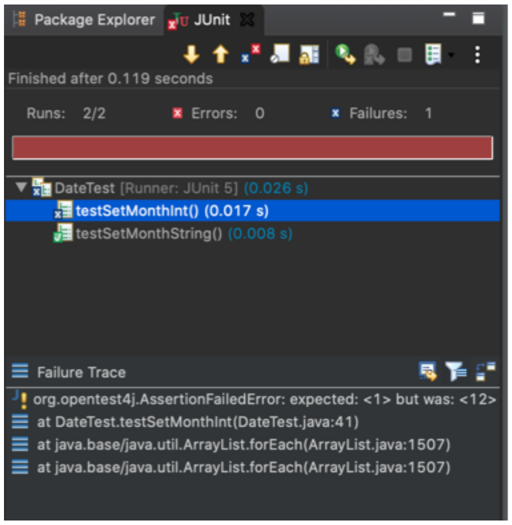
11. With the error introduced in the code I can now re-run the tests and see that I have a failure.



```
Finished after 0.119 seconds

Runs:  2/2     Errors:  0     Failures:  1

DateTest [Runner: JUnit 5] (0.026 s)
    testSetMonthInt() (0.017 s)
    testSetMonthString() (0.008 s)
```

```java
import static org.junit.jupiter.api.Assertions.*;

class DateTest {

    @Test
    void testSetMonthString() {
        // create an object to test
        Date testDate = new Date();

        // test with invalid data
        testDate.setMonth("Tuesday");
        // ask for the month
        int m = testDate.getMonth();
        // An invalid month should be set to January
        assertEquals(m, 1);

        // test with valid data
        testDate.setMonth("March");
        m = testDate.getMonth();
        assertEquals(m, 3);

    }
```

12. When you see a failure the first thing you should do is read the ***failure trace.*** You can find this on the JUnit pane. This area will provide you with summary information about the failure. The file name, the line number and the expected and gotten results of the test.



```
Finished after 0.119 seconds

Runs:  2/2     Errors:  0     Failures:  1

DateTest [Runner: JUnit 5] (0.026 s)
    testSetMonthInt() (0.017 s)
    testSetMonthString() (0.008 s)

Failure Trace
org.opentest4j.AssertionFailedError: expected: <1> but was: <12>
at DateTest.testSetMonthInt(DateTest.java:41)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1507)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1507)
```

Discussion Tasks: Finish writing unit tests for the Date class. Also demonstrate that you can unit test the constructors. Post screen shots of your successes and failures. Help your fellow classmates out. Communicate on Discord . . . whatever you need to do to help this stuff sink in.