Introduction

JUnit is an open-source framework for writing and running tests on your code. JUnit tests can be run from integrated development environments or the command line. JUnit tests can test that certain values are returned, exceptions are thrown when expected, and that all edge cases have been considered.

Why JUnit?

You may ask why JUnit is better than using the `main` method for testing. JUnit tests are much more useful because they are *unit tests* meaning they test each piece of your source code. If your main method fails on a line, all subsequent lines are not executed. In JUnit, if one test case fails, it only stops executing that test case, but continues on testing the other cases. In this way, we can test each method or piece of functionality individually, and know exactly where things are breaking.

Getting Started

You have two options to get started with JUnit.

- Download an IDE that supports JUnit. Some good ones are:
  - Eclipse Standard
  - IntelliJ IDEA Community Edition
- Run JUnit from the command line.

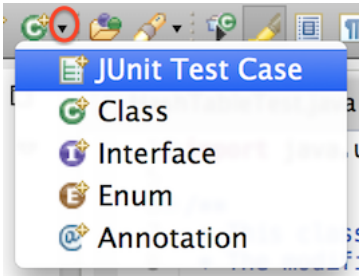
Writing Tests

Creating a New Test Class

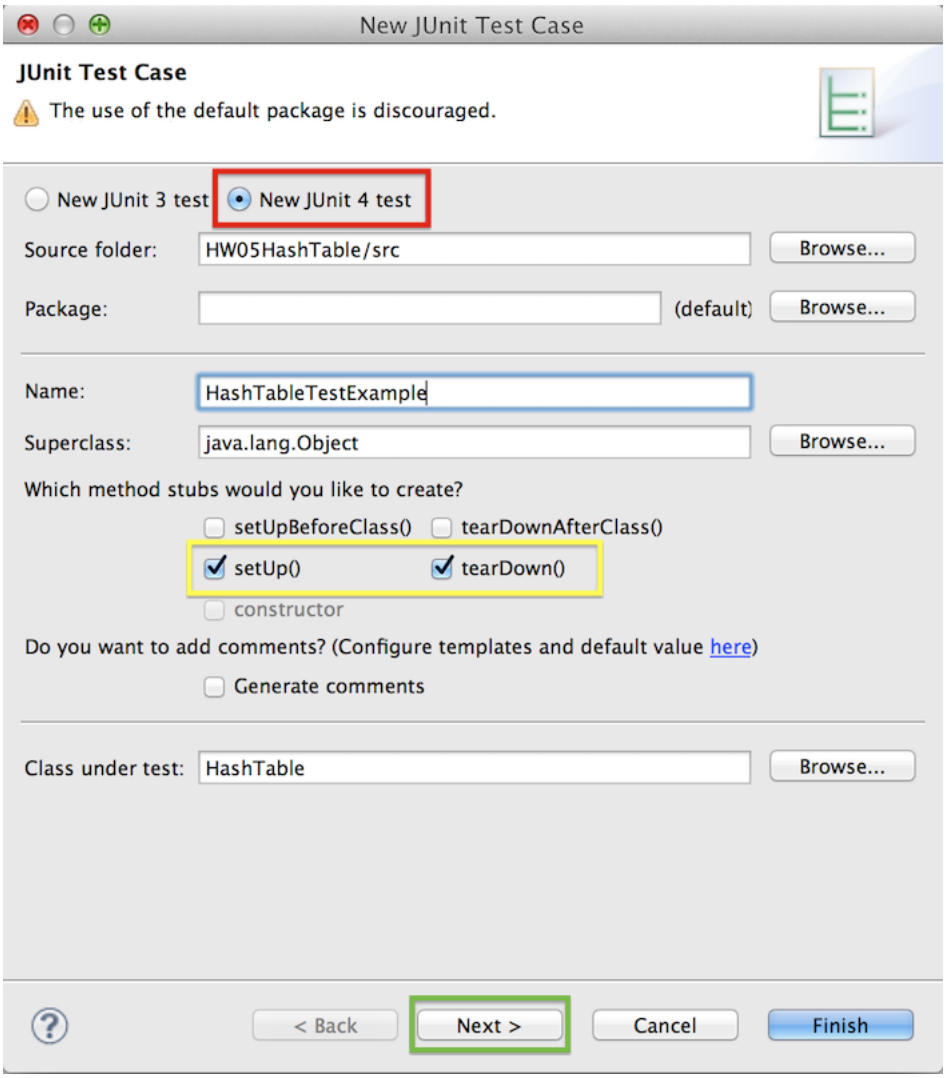
JUnit tests are classes which contain specially annotated methods that are executed by the JUnit runner. JUnit's annotated methods must be `public`, but you can call other private helper methods from within a test case.

Eclipse

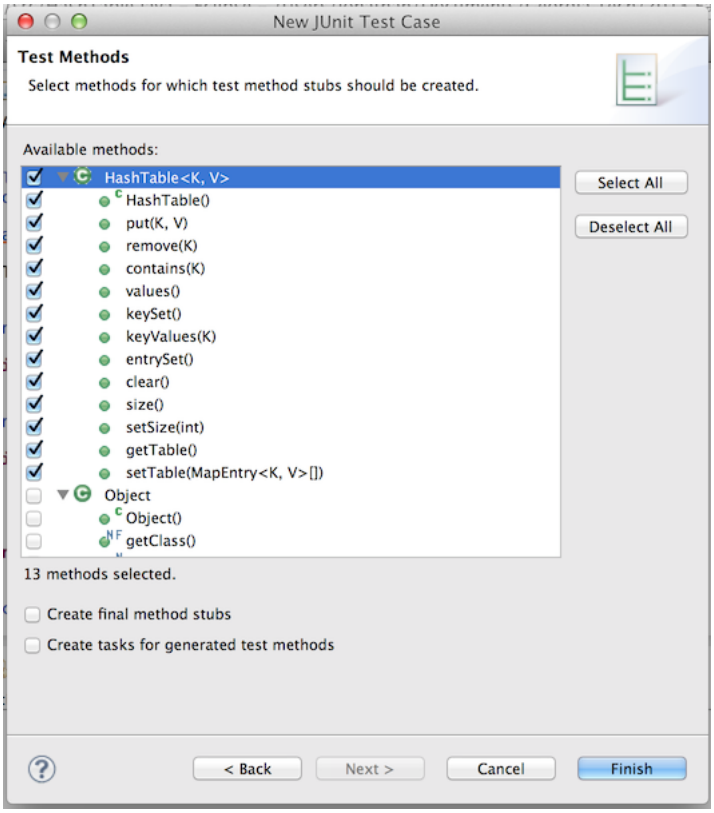
1. Open the class for which you want to create a JUnit test class.
2. Click on the little arrow next to the New Class button and click "JUnit Test Case".



3. Ensure "New JUnit 4 test" is selected, give it a name, and optionally, choose if you want the `@Before` setup method and the `@After` teardown method. Then, **click Next** (not Finish).


New JUnit Test Case
JUnit Test Case
The use of the default package is discouraged.
New JUnit 3 test
New JUnit 4 test
Source folder:
HW05HashTable/src
Browse...
Package:
(default)
Browse...
Name:
HashTableTestExample
Superclass:
java.lang.Object
Browse...
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:
HashTable
Browse...
< Back
Next >
Cancel
Finish

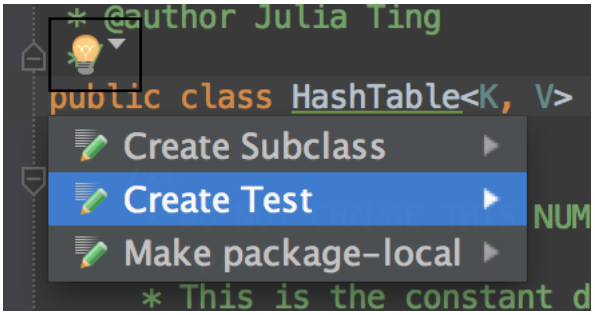
4. Check any methods you want to have test cases generated for (method stubs) that you can use as a starting point, then click Finish.



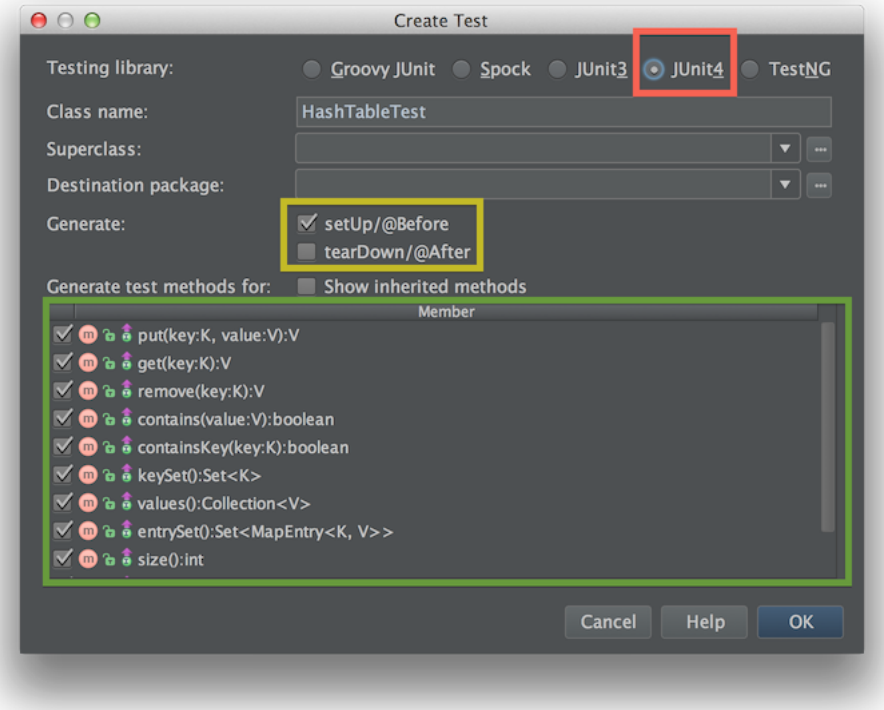
5. Eclipse will create and show the new test class.

IntelliJ IDEA

1. Click on the name of the class you want to create
2. Click on the little lightbulb that shows up at the beginning of the class declaration.
3. Click on "Create Test"



4. Make sure JUnit 4 is selected (shown in red), check the boxes for whether or not you want the `@Before` and `@After` annotations, and check the methods you'd like IntelliJ to create method stubs for. It'll create one test per method you select, and you can use it as a good starting point.



5. Click OK, and IntelliJ will create and show the new test class.

Text Editor

Create a new Java class and give it a name (preferably something ending in `Test`, such as `MyDataStructureTest`). Then, use the following import statements as needed.

```
// Annotations
import org.junit.Before;
import org.junit.Test;
import org.junit.After;

// Assertions
import static org.junit.Assert.fail;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNull;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertArrayEquals;
```

You can then create test cases and methods using the method annotations.

Method Annotations

Method annotations go right before the method header. These tell the compiler and the JVM how to compile the JUnit test cases. All annotated methods must be declared as `public`.

- `@Before` - This method runs before each test case. Initialize any instance variables here.
- `@Test` - This method is a test case. Add this annotation to each method you want to run as a test case.
- `@After` - This method runs after each test case. You can clean out any temporary data.

For example:

```
@Before
public void setUp() { }
@Test
public void testCase() { }
@After
public void tearDown() { }
```

You don't have to always have an `@After` method, but you'll almost always have a `@Before` method to initialize your variables (think of it as a constructor for each test cases).

JUnit Assertions

It's considered good style to use as few assertions as possible per test case. This makes it easier to pinpoint exactly which method calls are causing errors.

- `fail(String message)` - This method will instantly cause the test case to fail and print out the message string as the reason.
- `assertNull(Object object)` - If `object` is not `null`, then the test case will fail.
- `assertNotNull(Object object)` - If `object` is `null`, then the test case will fail.
- `assertTrue(boolean expected)` - If the boolean is not true, then the test case will fail.
- `assertFalse(boolean expected)` - If the boolean is not false, then the test case will fail.
- `assertEquals(Object expected, Object actual)` - If `actual.equals(expected)` is not true, then the test case will fail.
- `assertSame(Object expected, Object actual)` - If `actual == expected` is not true, then the test case will fail.
- `assertArrayEquals(Object[] expected, Object[] actual)` - The test case will fail unless all elements in `expected` are equal to the elements in `actual` and for every index `i`, `actual[i].equals(expected[i])`.

Limiting Run Time and Detecting Infinite Loops

To limit the amount of time a test case runs, use the `timeout` parameter for the `@Test` annotation. The timeout is specified in milliseconds.

```
@Test(timeout = TIMEOUT)
public void timedTestCase() { }
```

Expecting Exceptions

If you expect a test case to throw an exception and want to assert that the exception is thrown in your test case, you can do that using the `expected` parameter for the `@Test` annotation.

```
@Test(expected = IllegalArgumentException.class)
public void exceptionalTestCase() { }
```

To expect an exception and limit run time, use a comma to separate:

```
@Test(timeout = TIMEOUT, expected = IllegalArgumentException.class)
public void exceptionallyTimedTestCase() { }
```

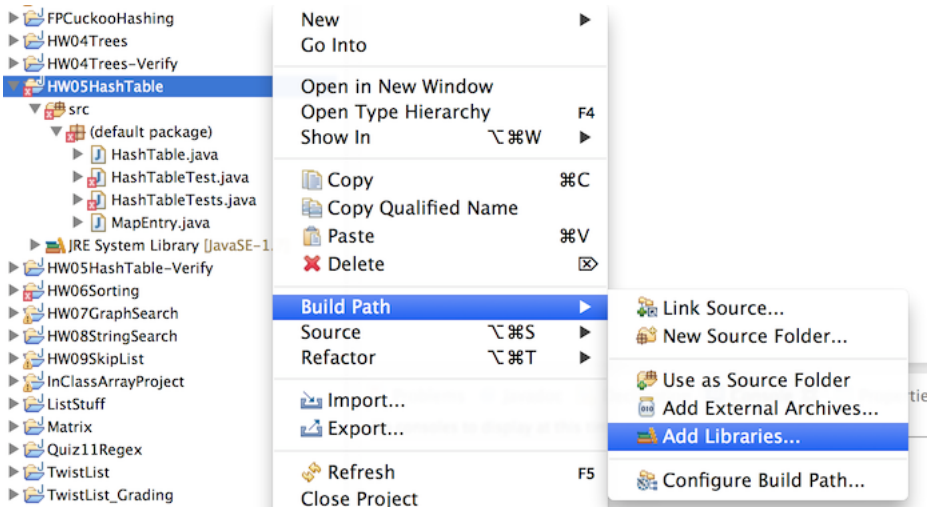
Running Tests from an IDE

Eclipse

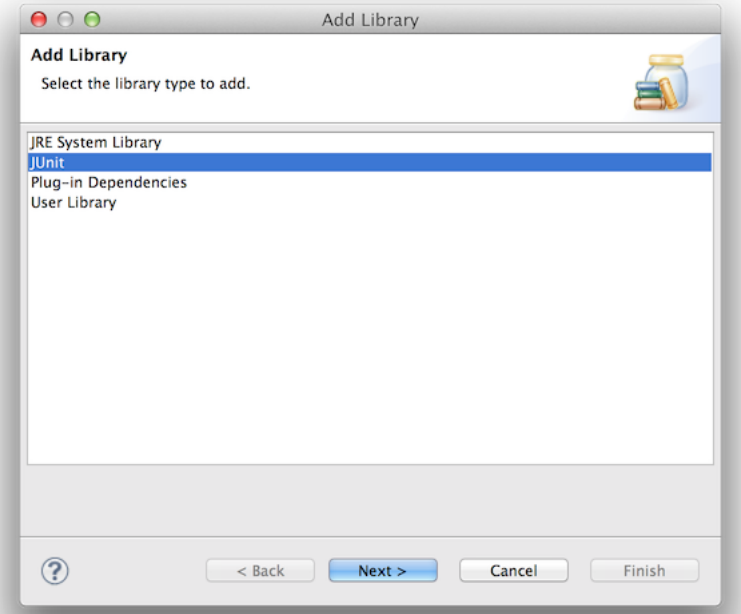
Build Path Configuration

If you've just added a test case but didn't create it using Eclipse, then you need to add JUnit 4 to your build path. If you created the test case as described above, then skip these steps.

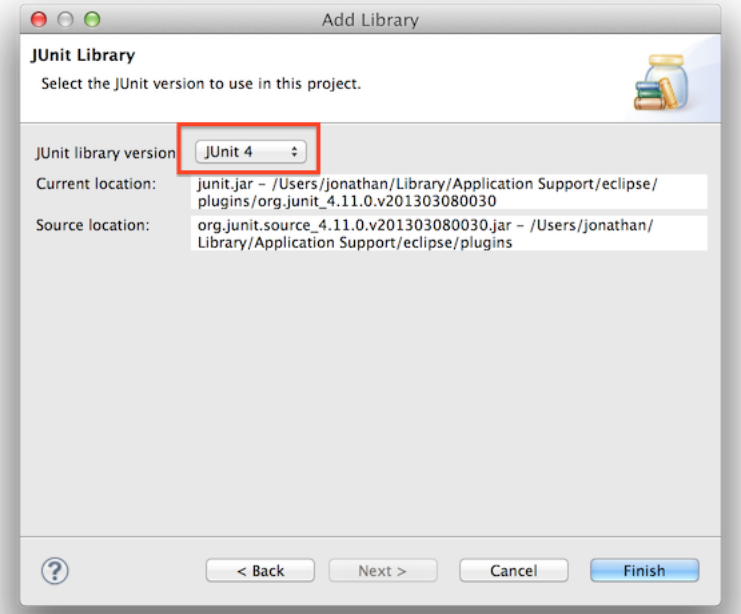
1. Right-click the current project
2. Go to Build Path → Add Libraries...



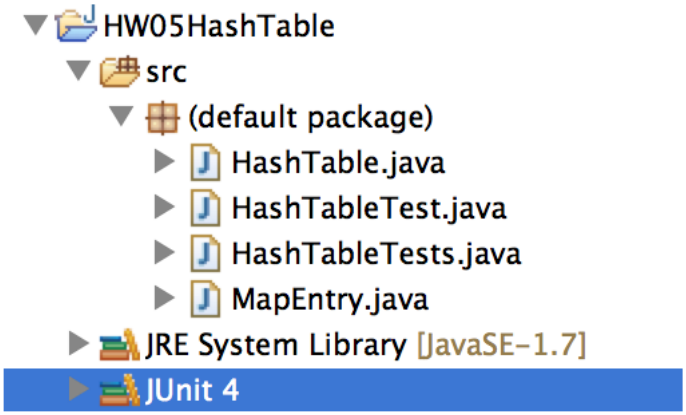
3. Select JUnit in the window and click Next.



4. Ensure JUnit 4 is selected and click Finish.



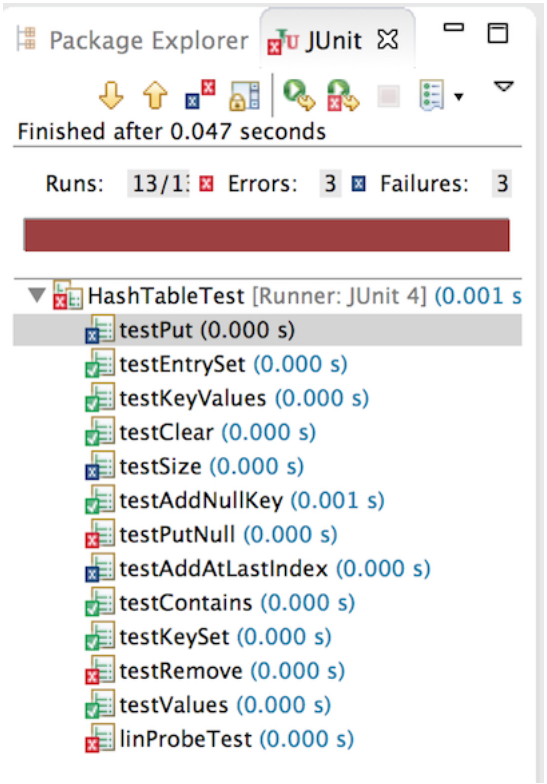
5. JUnit 4 should appear under your project's folder.



Running the Test Class

1. Open your JUnit Test Class.
2. Click the green "Run" button.

3. The JUnit test will run and show a result pane.



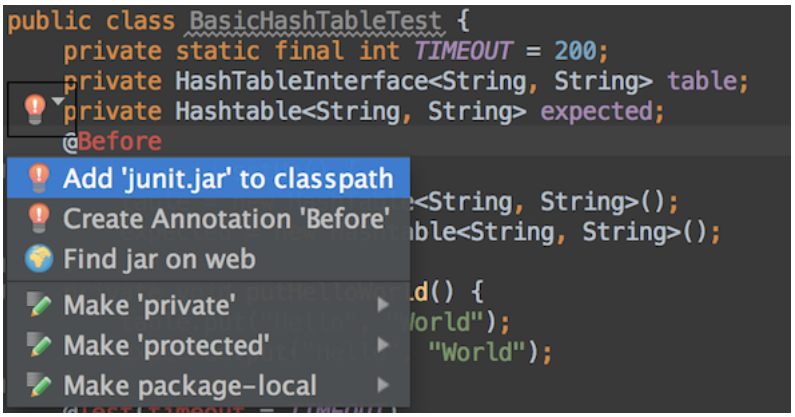
- A green check means that the test passed with no failures.
- A blue X means that the test case did not throw any exceptions, but contained assertion errors.
- A red X means that the test case failed because it threw an exception.

IntelliJ IDEA

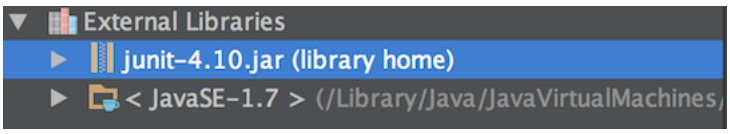
Classpath Configuration

If you've just added a test case but didn't create it using IntelliJ, then you need to add JUnit 4 to your classpath. If you created the test case as described in "Creating a New Test Class", then skip these steps.

1. Open the JUnit test class.
2. Click on any of the annotations (they should be red).
3. A little red lightbulb will pop up. Click it.
4. Click "Add 'junit.jar' to classpath"



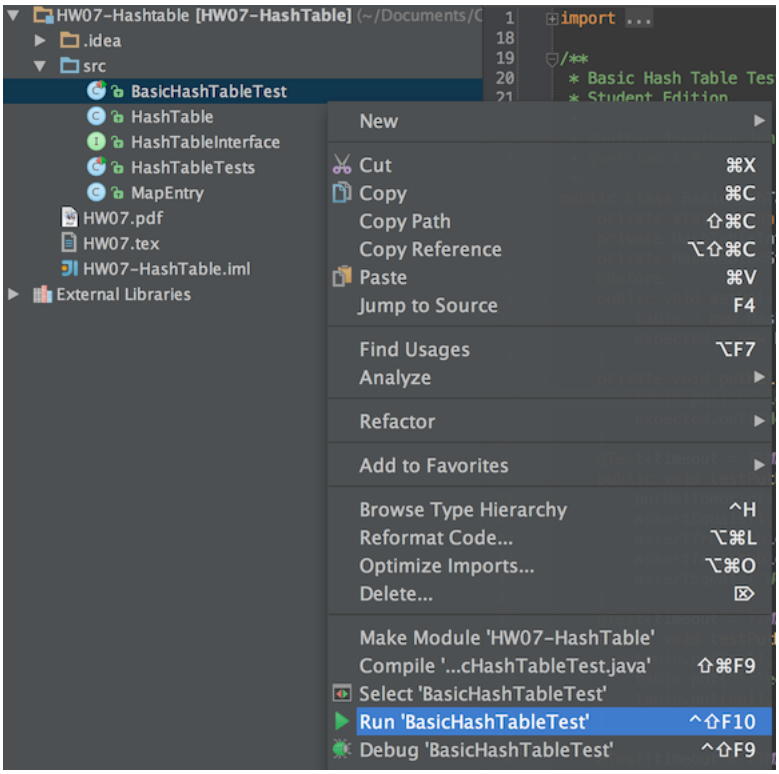
5. There should now be a JUnit JAR under the External Libraries tab in the Project window.



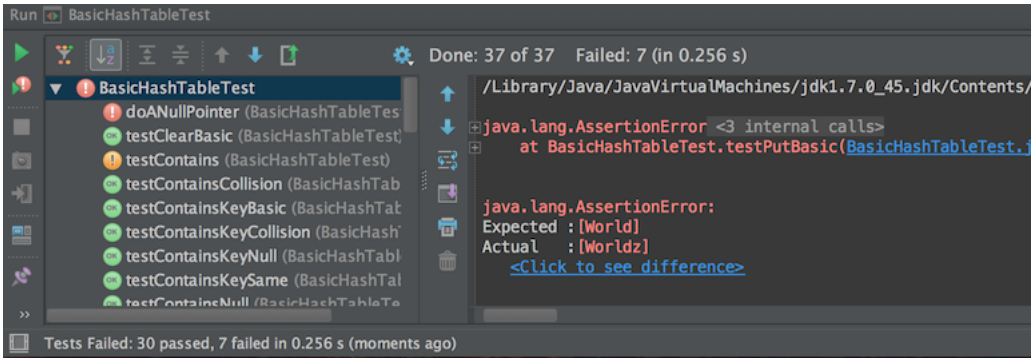
Running the Test Class

1. Right click your test class in the Project window.

2. Click "Run '<TestClassName>' "



3. The JUnit test will run and show a result pane.



- A green OK means that the test passed with no failures.
- An orange ! means that the test case did not throw any exceptions, but contained assertion errors.
- A red ! means that the test case failed because it threw an exception.

Running Tests from the Command Line

Classpath Configuration

You only need to do this once. This guide assumes that you can run `javac` from the command-line.

All Platforms

Download the JUnit and Hamcrest JAR files and put them somewhere safe on your computer. (Somewhere you won't accidentally delete them.)

Mac OS X and Linux

1. Open a Terminal window.
2. You will need to open `~/.bash_profile` in a text editor. You can use `open` to open the file in the default text editor. Open the file by typing `open ~/.bash_profile` and pressing Enter.

   Note: If you've never edited this file, you may need to create it by typing `touch ~/.bash_profile` and pressing Enter.
   Note: On Ubuntu, this file may be `~/.bashrc`
3. Add the following line to the top of the file:

   ```
   export CLASSPATH=.:/path/to/junit.jar:/path/to/hamcrest-core.jar
   ```

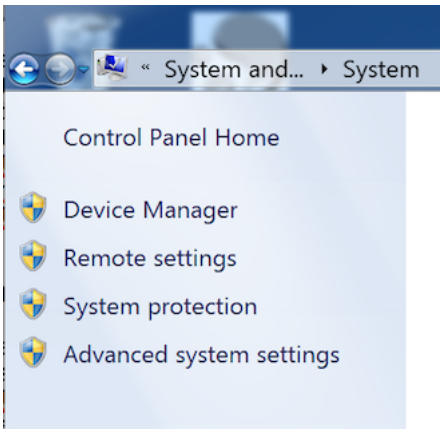
   where `/path/to/` is the directory path where the JAR files are.

   If my files were in `/home/jonathan/`, I'd add:
   `export CLASSPATH=.:/home/jonathan/junit.jar:/home/jonathan/hamcrest.jar`
4. Save the file.
5. Close and reopen the Terminal window.
6. Run `echo $CLASSPATH`. The line you added should print in the Terminal.

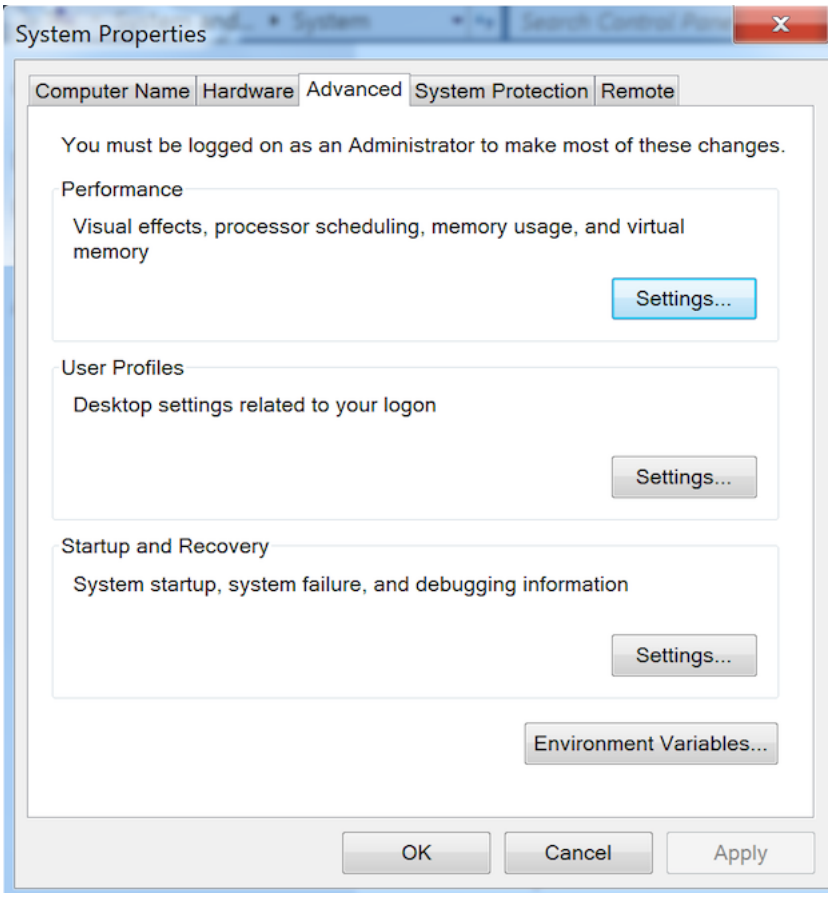
Windows

These instructions are for Windows 7 and Windows 8.

1. Open Control Panel.
2. Click "System and Security."
3. Click "System."
4. On the sidebar, click "Advanced system settings"



5. Click "Environment Variables."

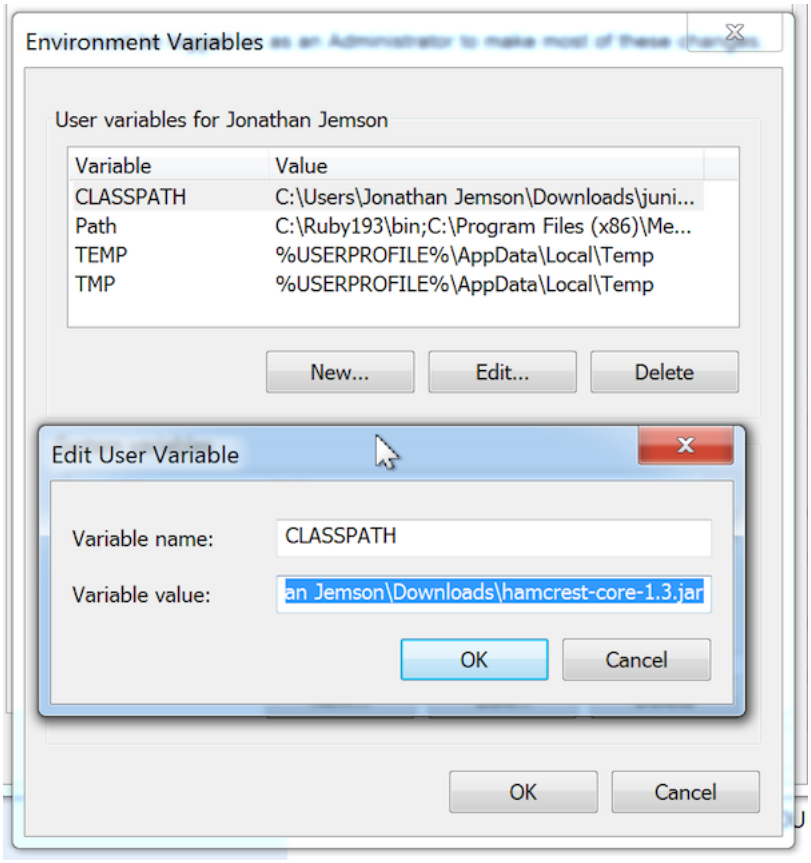


6. Under "User variables for Firstname Lastname"...
   - Click "New..." if `CLASSPATH` is not already a User Variable
   - If the `CLASSPATH` exists, click it, and click "Edit..."
7. Set the variable name to `CLASSPATH`.

8. Add the following to the existing (or new) variable value:

   `.;C:\path\to\junit.jar;C:\path\to\hamcrest.jar`

   replacing `C:\path\to` with the path to the folder with the JAR files.



9. Click OK, OK, OK.
10. Open a Command Prompt window.
11. Run `echo %CLASSPATH%` and the classpath you added should appear.
12. Run `java org.junit.runner.JUnitCore` and you should see `JUnit version 4.xx` appear.

Running Tests

Point your command line window to the directory your class files are in (using `cd`), and then run the following commands.

```
javac *.java
java org.junit.runner.JUnitCore <TestClassName>
```

The console will output the results of the test cases.