

ECSE 321 - Tutorial 4

JUnit & Travis



Dominic Charley-Roy

<https://github.com/dominiccharleyroy>

dominic.charley-roy @ mail.mcgill

Basic Introduction

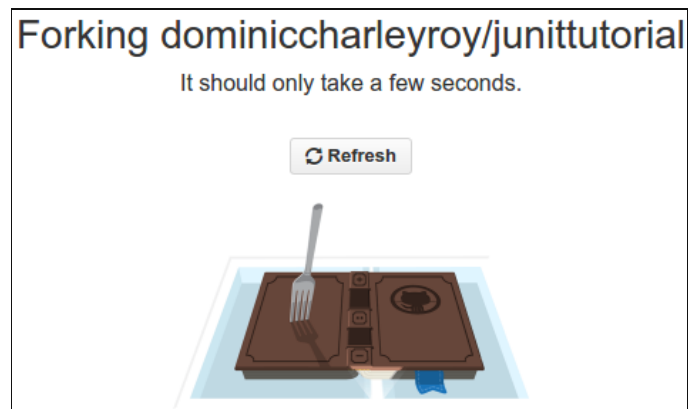
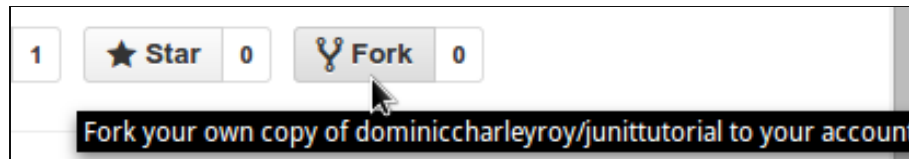
JUnit is a unit testing framework for Java.

What does that even mean?!

Unit tests are small tests which are responsible for testing a single rule of your program. JUnit provides **classes and methods** for writing these tests and runs them **automatically**.

Motivating Example

- A Grading System class which lets you add grade, get a list of student's grades, and compute the average.
- You will want to fork <https://github.com/dominiccharleyroy/junittutorial> in order to have your own copy of this tutorial.



Importing into Eclipse

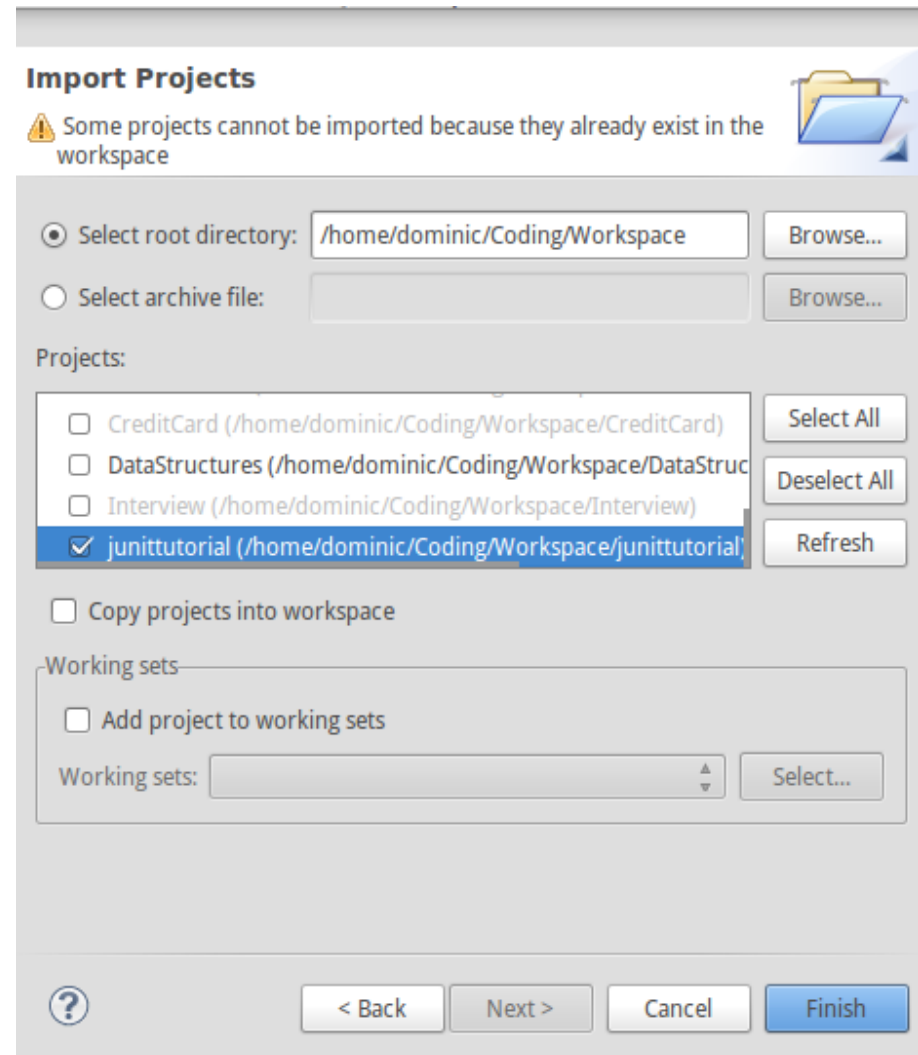
1. Clone your new repository:
git clone url

2. Open Eclipse

3. File > Import > Existing
Projects into Workspace

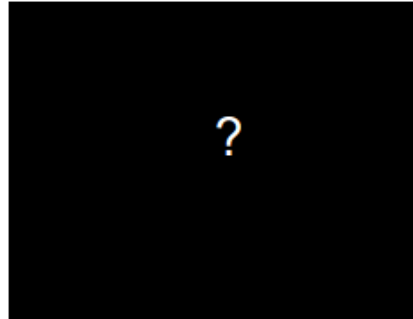
4. Set directory to the
directory containing your
cloned repository.

5. Select the project an import!



Example Overview

```
public interface GradingSystem {  
  
    /**  
     * This method registers a grade for a student.  
     * @param studentId The ID of the student.  
     * @param grade The grade obtained by the student. Should be  
     *             between 0 and 100.  
     * @throws IllegalArgumentException if the grade is not  
     *             in the valid range.  
     */  
    public void registerGrade(String studentId, int grade);  
  
    /**  
     * This method fetches the list of grades for a given student.  
     * The grades are presented in the order they were registered.  
     * @param studentId The ID of the student.  
     * @return A list of grades for the student, or null if the  
     *         student does not exist.  
     */  
    public List<Integer> getGrades(String studentId);  
  
    /**  
     * This method fetches the average grade for a student.  
     * @param studentId The ID of the student.  
     * @return The average grade of the student, or -1 if  
     *         the student does not exist.  
     */  
    public int getAverage(String studentId);  
}
```

```
public class BrokenGradingSystem implements GradingSystem {  
  
      
  
}
```

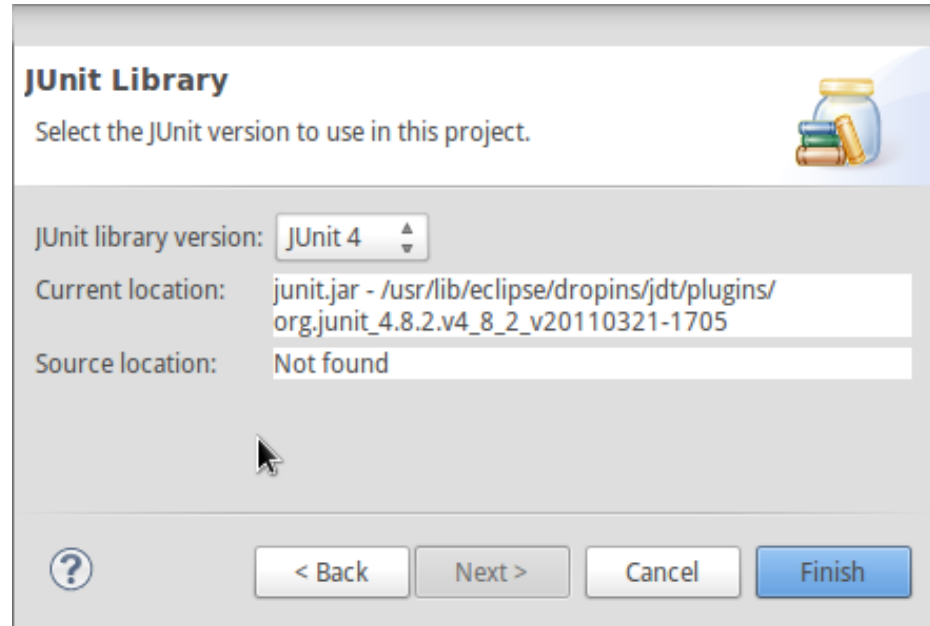
Adding JUnit

In order to test our implementation, we need to add the JUnit library to our project.

1. Right-click project > Build Path > Add Libraries...

2. Select JUnit and Next

3. Set JUnit version to 4



Non-Eclipse Users

If you aren't using Eclipse, you have to take some extra steps to install JUnit in your project.

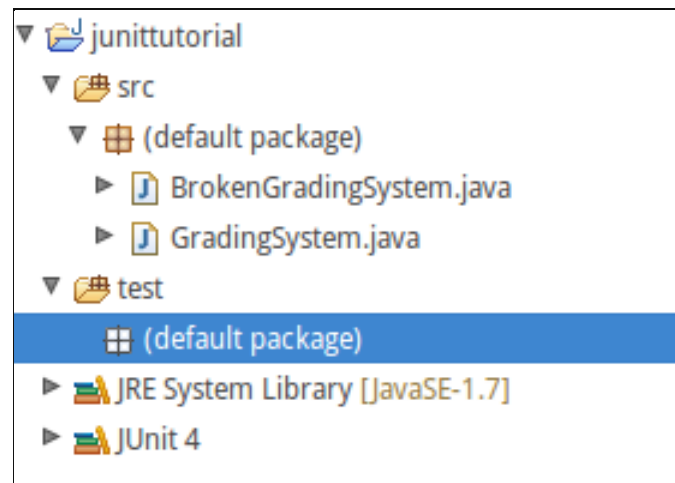
1. Go to <https://github.com/junit-team/junit/wiki/Download-and-Install>
2. Download junit.jar
3. Add junit.jar to your project's class path

Anatomy of a JUnit Test

- A **Java class** usually has a corresponding **test case** class. If the class is called **A**, then the test case class is **ATest**.
- A **test case** has **tests** which are methods. Each test should correspond to one single functionality / logic rule. Usually prefixed with *test* and describe what is being tested, eg. *testGetName* or *testSetAgeWithNegativeValue*.
- Many **test cases** can be bundled together in a **test suite**.
- JUnit lets you run either a single test, all tests in a single test case, or all tests in a test suite.

Where are tests stored?

- Tests are usually kept separate from our src/ folder to keep things clean and allow tests to be easily included/excluded from a jar file.
- We store them in a test/ source folder. In Eclipse, we create this like:
 1. Right-click project and New > Source Folder
 2. Set Folder Name to test and click Finish.

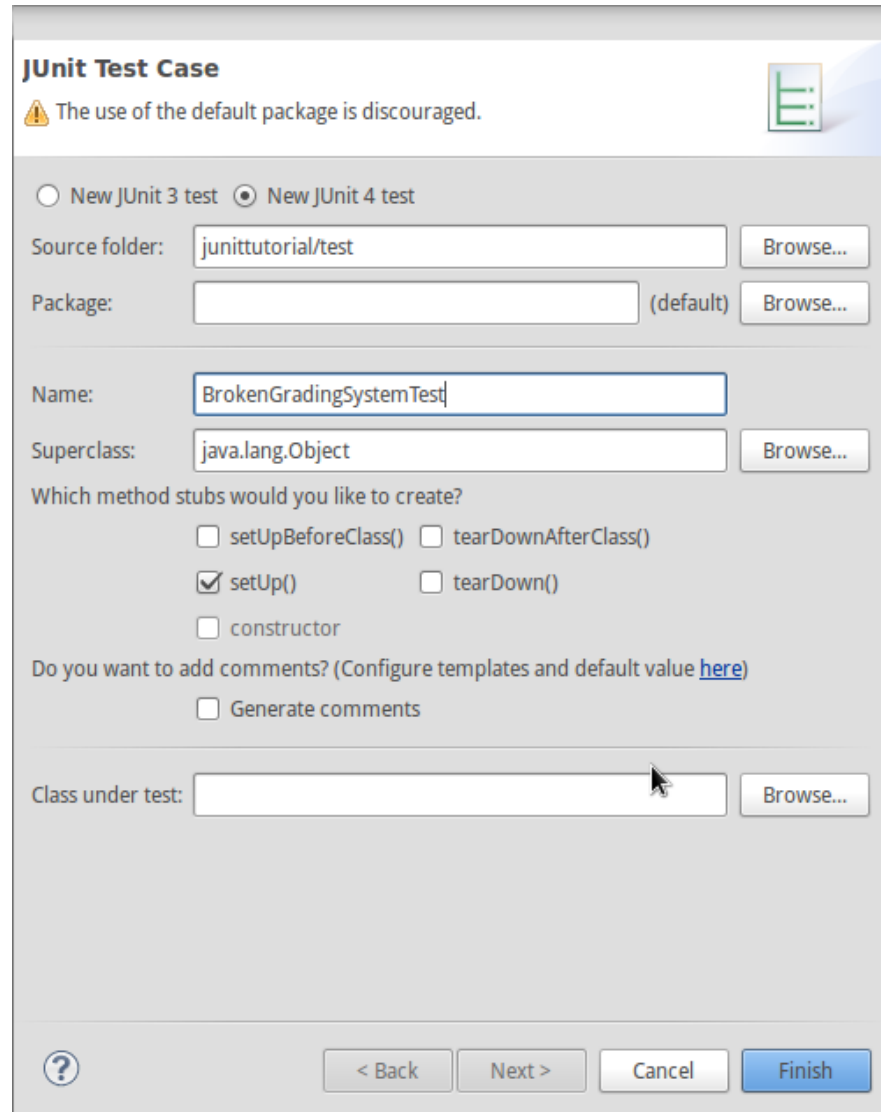


Creating a Test Case

Let's create a test case for BrokenGradingSystem!

1. Right-click test/, New > JUnit Test Case

2. Set the name to BrokenGradingSystemTest.



The image shows the 'JUnit Test Case' dialog box in an IDE. At the top, there is a warning icon and the text 'The use of the default package is discouraged.' Below this, there are two radio buttons: 'New JUnit 3 test' (unselected) and 'New JUnit 4 test' (selected). The 'Source folder:' field contains 'junittutorial/test' with a 'Browse...' button. The 'Package:' field is empty with '(default)' in parentheses and a 'Browse...' button. The 'Name:' field contains 'BrokenGradingSystemTest'. The 'Superclass:' field contains 'java.lang.Object' with a 'Browse...' button. Under the heading 'Which method stubs would you like to create?', there are four checkboxes: 'setUpBeforeClass()' (unchecked), 'tearDownAfterClass()' (unchecked), 'setUp()' (checked), and 'tearDown()' (unchecked). There is also an unchecked checkbox for 'constructor'. Below this, the text 'Do you want to add comments? (Configure templates and default value [here](#))' is followed by an unchecked checkbox for 'Generate comments'. At the bottom, the 'Class under test:' field is empty with a 'Browse...' button. The bottom of the dialog has a help icon, and four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

JUnit Test Case

⚠ The use of the default package is discouraged.

☐ New JUnit 3 test ☒ New JUnit 4 test

Source folder: Browse...

Package: (default) Browse...

Name:

Superclass: 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](#))

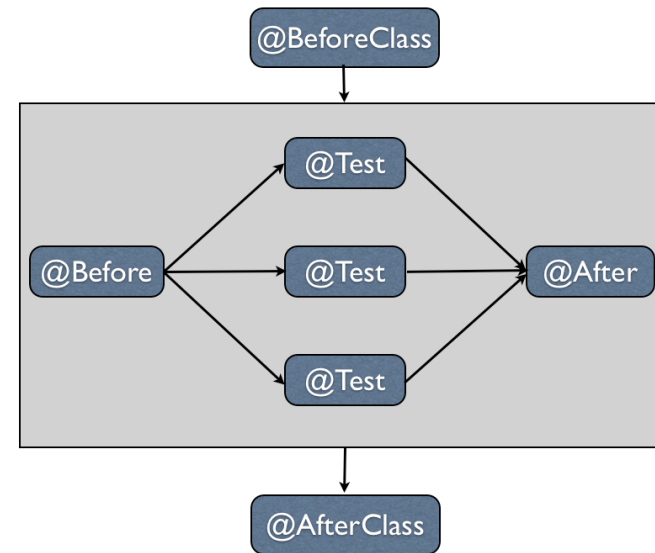
☐ Generate comments

Class under test: Browse...

? < Back Next > Cancel Finish

Running a Test Case

Once we've added tests, we'll want to run an entire test case. JUnit follows a special process for this based on **method annotations**. These must be imported (eg. `import org.junit.Test`).



BeforeClass/AfterClass methods should be *static* and are only run *once*. Before/After methods should be *non-static* and are run *before and after every test* (for setting up / cleaning up).

What's in a test?

- A test is any method annotated with **@Test**.
- A test runs code and makes **assertions** about it.
- If any assertions are false in the test, the test **fails**.
- JUnit provides many different assert statements, including: assertEquals, assertNull, assertTrue, assertFalse.
- JUnit also provides a method for automatically failing, **fail**.
- Generally put import static **org.junit.Assert.***; to have access to all assert methods without qualifying.
- <http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

Our First Test!!

The first test is that **getGrades** returns null for a student that doesn't exist.

We use @Before to create a new Grading System for each test.

```
import static org.junit.Assert.*;

import org.junit.Before;
import org.junit.Test;

public class BrokenGradingSystemTest {

    private BrokenGradingSystem system;

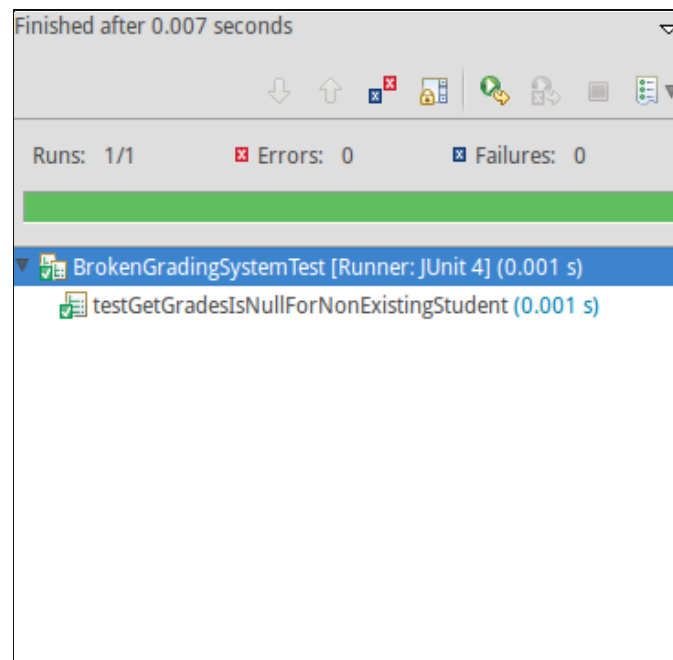
    @Before
    public void setUp() throws Exception {
        system = new BrokenGradingSystem();
    }

    @Test
    public void testGetGradesIsNullForNonExistingStudent() {
        assertNull(system.getGrades("I DON'T EXIST"));
    }

}
```

Running our Test Case

We're now ready to run our test! We do this by either hitting the run button when our test file is open, or by right-clicking on the *Test file and Run As > JUnit Test



Testing getGrades Works!

```
@Test
public void testGetGradesReturnsListOfGrades() {
    system.registerGrade("dominic", 80);
    system.registerGrade("dominic", 100);
    system.registerGrade("david", 90);

    List<Integer> grades = system.getGrades("dominic");

    assertEquals("There should be 2 registered grades for dominic.",
        2, grades.size());
    assertEquals(80, (int) grades.get(0));
    assertEquals(100, (int) grades.get(1));
}
```

Note that assertEquals, as do most other assertions, have two signatures.

1. assertEquals(expected value, obtained value)
2. assertEquals(message, expected value, obtained value)

Testing Exceptions

Some tests should only pass if an exception is thrown! There are two ways of testing this.

1. (Preferred way)

@Test(expected=ExceptionName.class)

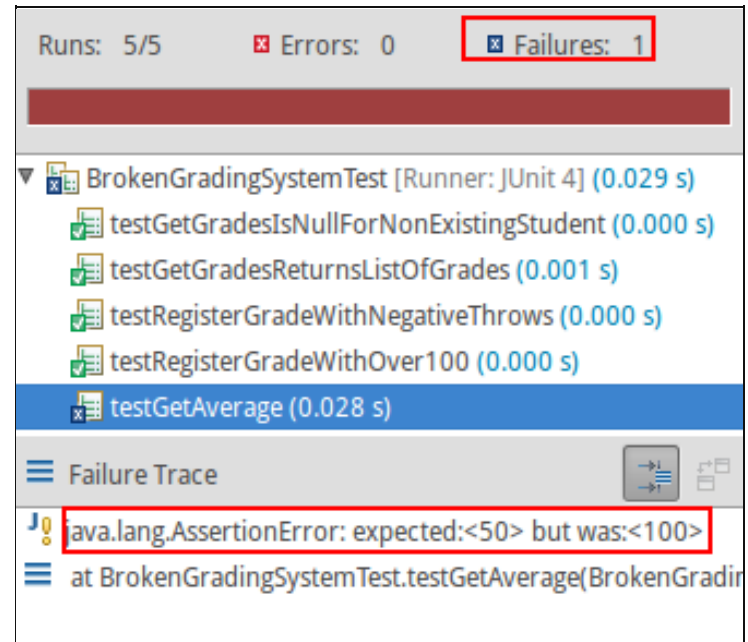
2. Wrap the code in a try catch and put a fail() after the call which should throw an exception.

```
@Test(expected=IllegalArgumentException.class)
public void testRegisterGradeWithNegativeThrows() {
    system.registerGrade("dominic", -1);
}

@Test
public void testRegisterGradeWithOver100() {
    try {
        system.registerGrade("dominic", 101);
        fail();
    } catch (Exception e) {}
}
```


Finding a Bug!

```
@Test
public void testGetAverage() {
    system.registerGrade("dominic", 25);
    system.registerGrade("dominic", 75);
    assertEquals(50, system.getAverage("dominic"));
}
```



The screenshot shows an IDE's test runner interface. At the top, it displays 'Runs: 5/5', 'Errors: 0', and 'Failures: 1'. Below this is a list of test cases for 'BrokenGradingSystemTest'. The test 'testGetAverage' is highlighted in blue and marked with a failure icon (a red 'x'). The failure trace below shows the error: 'java.lang.AssertionError: expected:<50> but was:<100>'.

Runs: 5/5 Errors: 0 Failures: 1

BrokenGradingSystemTest [Runner: JUnit 4] (0.029 s)

- testGetGradesIsNullForNonExistingStudent (0.000 s)
- testGetGradesReturnsListOfGrades (0.001 s)
- testRegisterGradeWithNegativeThrows (0.000 s)
- testRegisterGradeWithOver100 (0.000 s)
- testGetAverage (0.028 s)**

Failure Trace

java.lang.AssertionError: expected:<50> but was:<100>

at BrokenGradingSystemTest.testGetAverage(BrokenGradingSystemTest.java:10)



Who is this **Travis** fella?

- Travis is a **continuous integration** service.
- Brief summary: Travis looks after the **health** of your project, running tests every time you push a commit to GitHub.

Getting started with Travis

First step is to go to <https://travis-ci.org/>

2.

Sign in with GitHub

3.

Authorize application

4. Enabling Travis for your repo

dominiccharleyroy/junittutorial A repo containing the code for the ECSE321 JUnit tutorial.



Note about Travis

Travis uses a **.travis.yml** and **build.xml** file to describe what Travis should do. I've included these files as well as a lib folder in the original repository for Travis.

If you want to use Travis on your own project, the simplest way is to keep the same src/ and test/ structure and copy over travis.yml, build.xml, and the lib folder.

Our first Travis build

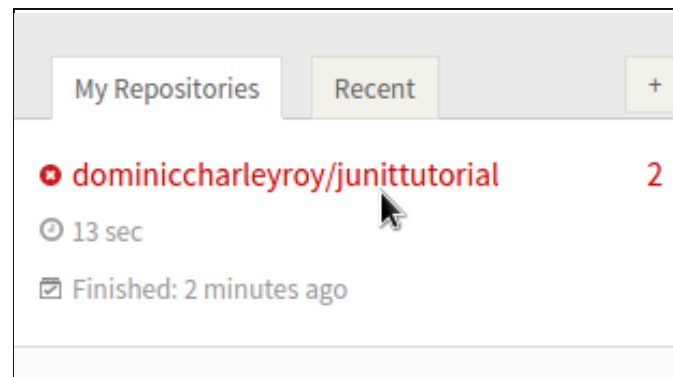
- Travis will build your project and run tests on every commit!
- To get Travis started, let's push your test:

```
git add .  
git commit -m "Initial tests."  
git push -u origin master
```

Travis will take a while to get set up on your first push, so just be patient.

Viewing your Repo Status

Once we've kicked off our first build, after waiting a minute or so we should be able to see the status by going to the Travis page and clicking on our repository.



dominiccharleyroy/junittutorial

build failing



A repo containing the code for the ECSE321 JUnit tutorial.

Current

Build History

Pull Requests

Branch Summary



master - Initial tests.

#3 failed

ran for 22 sec
less than a minute ago



Dominic Charley-Roy authored and committed

[Commit 38d1a73](#)



[Compare 1b665ab..38d1a73](#)



Build Matrix

Job	Duration	Finished	JDK
✖ 3.1	14 sec	less than a minute ago	oraclejdk7
✖ 3.2	5 sec	about a minute ago	openjdk7
✖ 3.3	3 sec	2 minutes ago	openjdk6

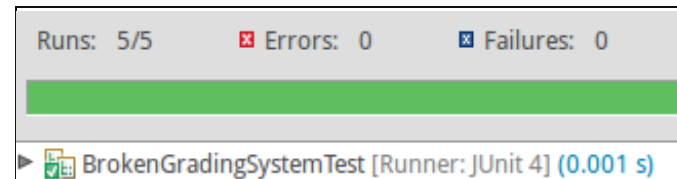
Time to fix our code!

Our test helped us pinpoint a bug in `getAverage`, so let's take a look at our function.

```
public int getAverage(String studentId) {  
    // Check for non-existing  
    if (!grades.containsKey(studentId)) {  
        return -1;  
    }  
  
    // Sum up the total number of grades  
    int total = 0;  
    for (int grade : grades.get(studentId)) {  
        total += grade;  
    }  
  
    // Divide the total by the number of grades  
    int average = total / grades.get(studentId).size();  
  
    return total;  
};
```

Once it's fixed...

After fixing our bug, we run the JUnit tests and then commit.



dominiccharleyroy/junittutorial build **passing**

A repo containing the code for the ECSE321 JUnit tutorial.

Current Build History Pull Requests Branch Summary

master - Fixed bug. **#7 passed**

ran for 21 sec
less than a minute ago

Dominic Charley-Roy authored and committed [Commit f9ff448](#) [Compare 55b244d..f9ff448](#)

Build Matrix

Job	Duration	Finished	JDK
L1	5 sec	2 minutes ago	oraclejdk7
L2	16 sec	less than a minute ago	openjdk7

TL;DR

- **JUnit** lets you write tests which can be run automatically to find bugs in your application.
- **Travis** is a tool that runs these tests every time you push to GitHub.
- These let you find new bugs as well as make sure new features aren't breaking old code (**regression bugs**).
- **Note:** Test-driven development works by writing tests *first* and then coding to make the tests pass.