JUnit Testing

Testing

- Code breaks
  - It is inevitable
- To mitigate risk, it is advisable to test one's code
  - There are many ways to do this:
    - Smoke testing
    - Functional testing
    - Unit testing
    - ...
- It is impossible to prove code is bug free
  - It is possible to show that code is free of bugs that have been tested for
  - Extensive testing is always recommended

Unit Tests

- A logical way to break tests up is to break small logical *units* (such as a single method or small group of related methods)
- Methods and classes are tested in isolation to avoid having bugs in one part of the code mask other problems
  - Nothing is worse than two bugs that mask each other for the cases you test for...

Writing Tests

- When determining what to test (hint: everything), you need to identify the possible cases
  - It is usually impossible to test all inputs to a problem
  - Identify classes of inputs
  - "There are three numbers CS people care about: 0, 1, and everything else"
  - Identify edge cases, and examples of general input, and determine what the expected output is
- You should then write code to test your code

Repeatable Testing

- Running tests by hand and visually verifying output is fine for small projects
  - By small, I mean smaller than some of the assignments you've worked on in here
- It is preferable to write a set of test cases that can be run automatically that checks expected output against what your code outputs
- This allows you to perform "regression testing"
  - Avoid the "But it worked yesterday!" shouts
- For this, we like to use Unit Test Frameworks

JUnit Testing

- JUnit is a framework for writing repeatable tests
- Designed for Java, but ported to PHP, C#, ...
- JUnit 3 and 4 has full support in NetBeans

Building Tests

- This works in NetBeans 8.1, but can change between versions
- After writing your class you want to test, right-click the name of the class in the `Projects` window and choose `Tools, Create/Update Tests`
- Choose the method access levels you would like to generate test code for
  - NetBeans will create stubs of test methods for you to fill in

Annotations in JUnit

- JUnit uses annotations to mark methods that will run at different times
  - `@BeforeClass`, `@AfterClass`
    - use methods marked with these annotations for code you want to run once, either before or after the test cases are run
  - `@Before`, `@After`
    - use methods marked with these annotations for code you want to run before and after each method test
  - `@Test`
    - These methods are your test cases. NetBeans will autogenerate stubs for you
    - The generated methods are designed to automatically fail until you write them and remove the `fail` statement.

Test Method Stub

```
@Test
public void testFoo() {
    System.out.println("foo");
    int origVal = 0;
    FooBar instance = new FooBar();
    int expResult = 0;
    int result = instance.foo(origVal);
    assertEquals(expResult, result);
    // TODO review the generated test code and
    // remove the default call to fail.
    fail("The test case is a prototype.");
}
```

Modifying the Test

- You should modify the methods to reflect the original input and the expected output
- You should also remove the `fail` statement so your test passes

Modified Unit Test

```
@Test
public void testFoo() {
    System.out.println("foo");
    int origVal = 5;
    FooBar instance = new FooBar();
    int expResult = 15;
    int result = instance.foo(origVal);
    assertEquals(expResult, result);
}
```

- You can run your tests by right clicking on the test java file and selecting `Test File`, or hitting `Ctrl + F6` on your keyboard. A progress bar will show you what did and did not succeed (if all tests pass, the feedback window may not show)

A Failed Test

More on Annotations

- You can use the `expected` parameter wth the `@Test` annotation if you want to check for an exception
- You can skip unit tests by using the `@Ignore` annotation