Java Programming

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

- JUnit and Automated Testing

Today's Lecture

- It is important to test code so that you eliminate any errors it may contain.
- All companies do some degree of testing on their software before they release it to customers.

Testing

- Automated Test – Run a program that tests if the application is working properly. No human interaction.
- Manual Test – A human sits at the screen and interacts with the application.
- AUTOMATED TESTS ARE BETTER!!!

Automated and Manual Tests

- Automated tests are faster than manual tests.
- Automated tests are easily repeatable. You are guaranteed to do the exact same test each time you run it.
- Automated tests allow you to easily test the program on extreme loads (lots of users or data).
- For example, simulating thousands of users logging on to a website or loading millions of pieces of data into a program.

Benefits of Automated Tests

- Assume the following class definition:

```
class Person {
        private String m_Name;
        private int m_Id;

        String GetName() { return m_Name; }
        int GetId() { return m_Id; }

        void SetName(String name) {
                m_Name = name;
        }

        void SetId(int id) {
                m_Id = id;
        }
}
```

Person Class

Does the following code test if the SetName method works correctly?

```
Person p = new Person();
p.SetName("Derek");
```

Testing Code

Does the following code test if the SetName method works correctly?

NO!

```
Person p = new Person();
p.SetName("Derek");
```

Incorrect assignment in SetName will NOT be caught by this testing code.

```
Public void SetName(String name) {
        name = m_Name;      // Incorrect assign
        //m_Name  = name;   // Correct assign
}
```

Bad Testing Code

- Actually testing that the value returned is what we expect would be better.
- The example on the next slide shows a brute force unit test (does not use JUnit).
- Examples later in the slides will use JUnit instead.
- JUnit has extra features as opposed to the brute force method that make unit testing easier.

Brute Force Unit Testing Code

The following testing code will catch the error in SetName from the previous slide…

```
Person p = new Person();
String testName = "Derek";
p.SetName(testName);

if (testName.equals(p.GetName())) {
        System.out.println("Person Get/Set Name: Pass");
}
else
{
        System.out.println("Person Get/Set Name: FAIL!");
}
```

Checks if the value sent in is set correctly

Brute Force Unit Test (not great)

Test SetId for both valid and invalid data

```
void SetId(int id) {
   if (id >= 0) {
      m_Id = id;
   }
}
Person p = new Person();
int validId = 10;
p.SetId(validId);
if (validId == p.GetId()) {
   System.out.println("Person Get/Set Id, Valid Value: Pass");
} else {
   System.out.println("Person Get/Set Id, Valid Value: FAIL!");
}
int invalidId = -77;
p.SetId(invalidId);
if (validId == p.GetId()) {
   System.out.println("Person Get/Set Id, Invalid Value: Pass");
} else {
   System.out.println("Person Get/Set Id, Invalid Value: FAIL!");
}
```

GetId should return validId the get/set worked properly

GetId should return the original id (10 from previous SetId call) since the invalid value should not be allowed to go in

Brute Force Test Valid and Invalid Data (not great)

- JUnit – Used for unit testing in Java applications.
- We will be discussing JUnit 5.

JUnit

Test Packages

- In NetBeans:
  - Source Packages contains all your source code.
  - Test Packages contains all your testing code.
- Test Packages does not initially appear under the project. NetBeans will create it automatically when you create a test class (next slide).

- Note: You should add a package under Test Packages for each source package that will be tested (after Test Package itself is created).
- For example, if there is a package named mycompany.mystuff under Source Packages then you should add a package named mycompany.mystuff under Test Packages.
- Next, add test classes as necessary under Test Packages/mycompany.mystuff (next slide).

Test Packages

Test Classes

- Add a test class to your testing package.

- First, right-click the testing package. For example, under Testing Packages you should right-click mycompany.mystuff to add a new test class.
- Choose New|Other from the context menu. A dialog will appear.
- Choose Unit Tests on the left (under Categories) and JUnit Test on the right (under File Types). Click Next.
- Give the new test class a name. The name should be the name of the class you are testing with Test appended to the end. For example, if you are testing a class named Employee the test class should be named EmployeeTest. Click Finish.
- A new test class should now appear under Testing Packages/mycompany.mystuff.

Test Classes

JUnit Maven Dependencies and NetBeans

- Once you add a test class to a project NetBeans will automatically add SOME of the necessary JUnit Maven dependencies.
- After adding a test class, look in the project's pom.xml file and you will see the JUnit dependencies.
- **IMPORTANT!** JUnit 5 requires 2.22.0 or higher of the Maven Surefire Plugin. The plugin is in bold below. Add the plugin to your pom.xml file.

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <!-- JUnit 5 requires Surefire version 2.22.0 or higher -->
      <version>2.22.0</version>
    </plugin>
  </plugins>
</build>
```

If <build> and <plugins> do not exist in your pom.xml file then add those too (under <project>)

Notes Regarding the New Test Class

Test Method

- Use the @Test annotation to create a test method in a test class.
- For example:

```
@Test
void myTestMethod() {
    // Testing code goes here…
}
```

- All methods in the test class that are decorated with @Test are testing methods.
- When you run the test NetBeans will automatically run all test methods.

Test Method

Assertions

- Use assertions to check results of running methods.

- **assertEquals** – Succeeds if its arguments are EQUAL.

```
assertEquals(10, 10); // Succeeds
assertEquals(10, 20); // Fails
```

- **assertNotEquals** – Succeeds if its arguments are NOT EQUAL.

```
assertNotEquals(10, 10); // Fails
assertNotEquals(10, 20); // Succeeds
```

- NetBeans will indicate that a test method fails if any of the assertions in the method fail.

Assertions

Running Tests in NetBeans

- Right-click the project to bring up a context menu.
- Choose Test. This will execute all the methods decorated with the @Test annotation.

OR

- Right-click a test class.
- Choose Test file (only runs that particular test class).

- These do NOT run the main method (these only run tests).
- The results of the tests will be displayed in the Test Results window. If Test Result window is not showing go to Window|IDE Tools|Test Results to display it.

Running Test in NetBeans

- Here is a test class for the Person class defined earlier in the slides:

```
public class PersonTest {

    @Test
    public void testGetSetName() {
        Person p = new Person();
        String testName = "Derek";

        p.SetName(testName);

        assertEquals(testName, p.GetName());
    }

   // Other testing code here…
}
```

Make testGetSetName a test method by decorating with @Test

Set the name

Make sure the name we get back is the name we put in using SetName

If the assertEquals fails then NetBeans will show that in the Test Results window

Sample Test Class and Test Method

End of Slides

Written by Arthur Hoskey, Ph.D.