Java Programming

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

Today's Lecture

- JUnit and Automated Testing

Testing

- 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.

Automated and Manual Tests

- 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!!!

Benefits of Automated 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.

- 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

Bad 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
}
```

- 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

Brute Force Unit Test (not great)

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

```
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!");
}
```

Test SetId for both valid and invalid data

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)

- Now we will move on to JUnit…

JUnit

JUnit

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

Test Packages

- In NetBeans:
  - Source Packages folder contains all your source code.
  - Test Packages folder contains all your testing code.

- The Test Packages folder does not initially appear under the project.
- It will get created when you create a new test class (see next slide).

- **IMPORTANT! You must add a special Maven dependency to use JUnit 5. NetBeans will add some dependencies but not all. Details for how to do this are on an upcoming slide.**

Test Packages

Setup Test Packages in Project

Add Test Packages

- Right-click the project.
- 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.
- Test Packages will now appear under the project (inside the Projects window).
- Click cancel on the dialog (we just wanted to create Test Packages).

Add Package to Test Packages

- Right-click Test Packages.
- Choose New|Other from the context menu. A dialog will appear.
- Choose Java on the left (under Categories) and Java Package on the right (under File Types toward the bottom of the list). Click Next.
- Package Name. The package name should be the same as the package name that the original class resides in. Click Finish.

Source Packages/**mycompany.mystuff**/Person

Test Packages/**mycompany.mystuff**/PersonTest

Assume you are testing Person. You must create a package with the same name as the one that Person is under. Create the PersonTest class there.

Setup Test Packages

Test Class Naming and Setup

- A JUnit convention is to have a matching test class for each class that you want to test (1 to 1 correspondence between classes and test classes).
- You are not required to do it this way, but it is recommended.
- Each test class should be located under Test Packages in the same package as the class being tested.

Source Packages

com.mycompany.hr

Employee.java

Manager.java

com.mycompany.sales

Purchase.java

Test Packages

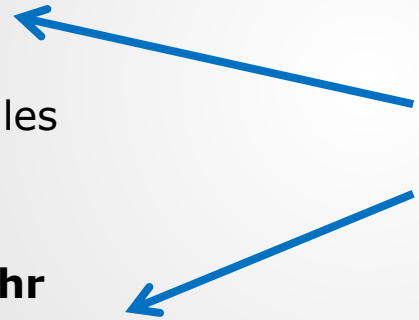
com.mycompany.hr

EmployeeTest.java

ManagerTest.java

com.mycompany.sales

PurchaseTest.java



Employee and Manager are under the package com.mycompany.hr so their matching test classes should be under that package in Test Packages

Test Class Naming and Setup

Adding Test Classes

- Right-click the **package** under **Test Packages** that you want to add the test class to (the test class must be under Test Packages in a package with the same name as where the original class is located).
- 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 Person the test class should be named PersonTest.
- Click Finish.
- A new test class should now appear under Test Packages. For example: Test Packages/mycompany.mystuff/PersonTest.java.

Adding Test Classes

Notes Regarding the New Test Class

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>
      <version>3.0.0-M5</version>
    </plugin>
  </plugins>
</build>
```

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

- We will now add test methods to a test class…

Test Method

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 (instructions on running a test are on an upcoming slide).

Test Method

Assertions

- Use assertions to check results of running methods in a JUnit test class.
- Do NOT use if statements!

- **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

assertEquals and Objects

- There is an overload of assertEquals that compares Objects.
- This overload will call the equals method to check for equality.
- This means that it will do a value compare as opposed to a reference compare (assuming the class being compared has an override of equals that does a value compare).
- It is important to override the equals method on classes you create if you want to do a value compare.

assertEquals and Objects

Running Tests in NetBeans

- Testing code does NOT run when you normally execute your program.

- 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).

- Running test code does NOT run the main method (the above only runs the JUnit tests).

- **The results of the tests will be displayed in the Test Results window. If the Test Results window is not showing, go to Window|IDE Tools|Test Results to display it.**

Running Test in NetBeans

Running Tests in NetBeans

- Any methods that have assert statements that fail will cause messages to appear in the Test Results window.
- If there are failure messages you can go to the original source code and look to see what went wrong.

Running Test in NetBeans

Sample Test Class and Test Method

- 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 methods go 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

End of Slides