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.

Person Class

- Assume the following class definition:

```
public class Person {
	private String name;
	private int id;

	public String getName() { return name; }
	public int getId() { return id; }

	public void setName(String name) {
		this.name = name;
	}

	public void setId(int id) {
		this.id = id;
	}
}
```

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

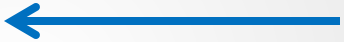
```
Person p = new Person();
p.setName("Derek");
```

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 = this.name;    // Incorrect assign
        //this.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)

```
void setId(int id) {
   if (id >= 0) {
      this.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

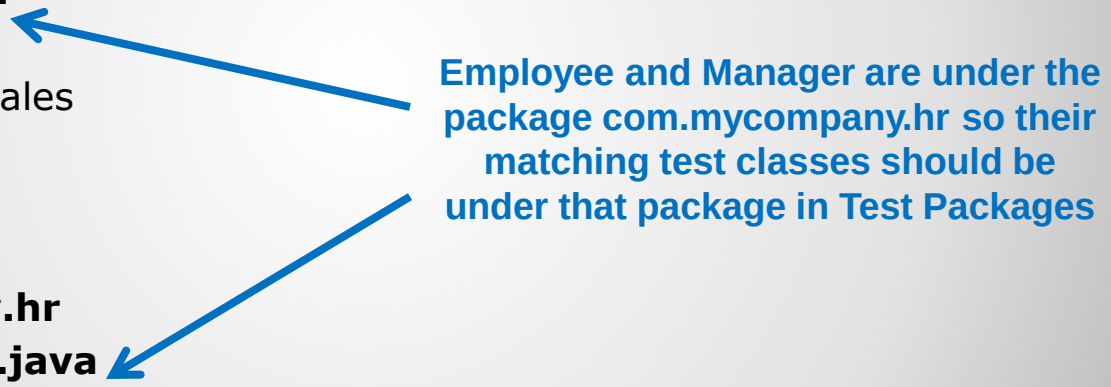
- JUnit – Used for unit testing in Java applications.
- We will be discussing JUnit 5.

- For each class you want to test you need to create a matching test class for it.

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 under test in the same package as the class being tested.

main
  java
    com.mycompany.hr
      Employee.java
      Manager.java
    com.mycompany.sales
      Purchase.java
test
  java
    com.mycompany.hr
      EmployeeTest.java
      ManagerTest.java
    com.mycompany.sales
      PurchaseTest.java

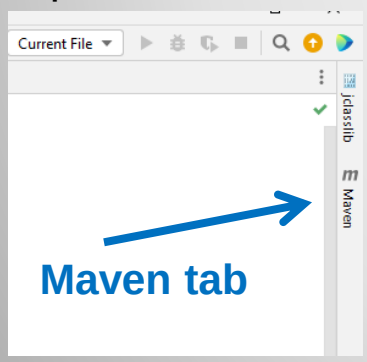


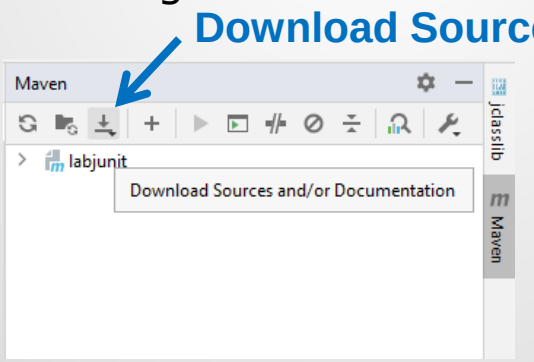
Test Class Naming and Setup

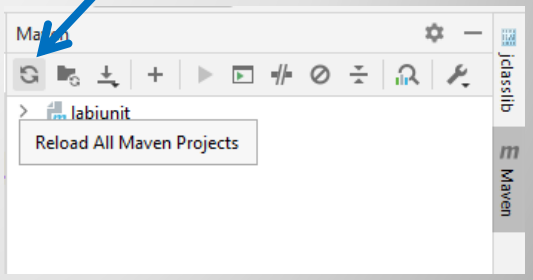
- Create a console application that uses Maven.
- Add the following dependency to the pom.xml file (it should be a child of <dependencies>):

```
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.10.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

- Open Maven tab on the right. Choose download sources and then Reload.

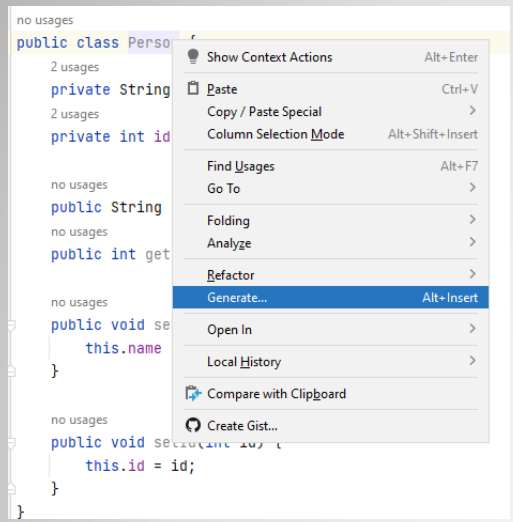




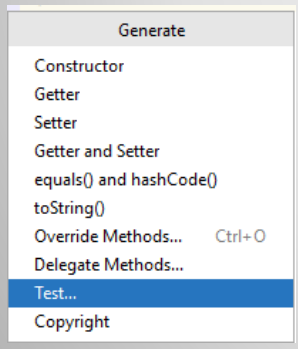


JUnit 5 Dependency

- Right click a class and choose Generate from the context menu.

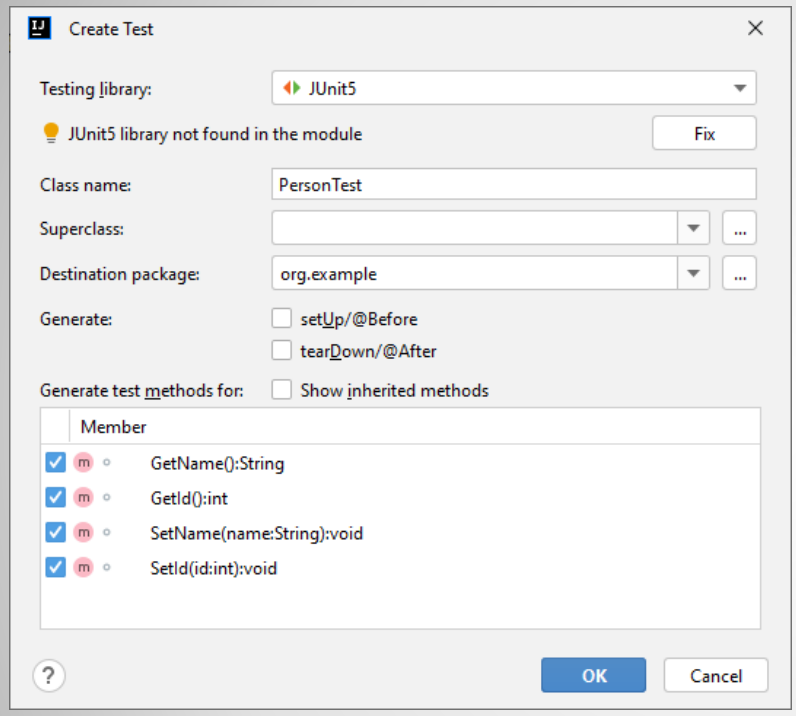


- Choose Test from the Generate Menu.



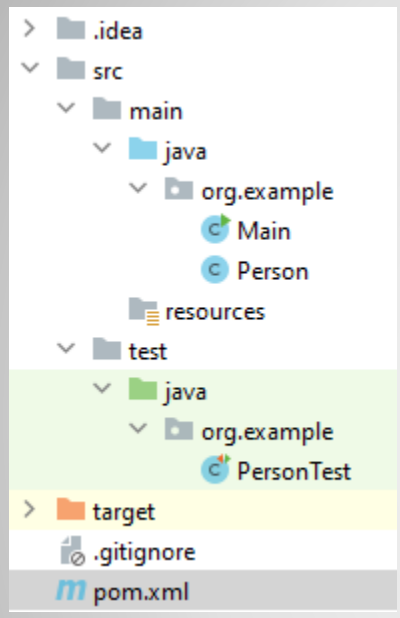
Create JUnit Test Class

- Set the Create Test dialog, choose the methods you want to test and press OK.



Create JUnit Test Class

- A new class will be created under test. The name of the class will be the same as the original class except Test is appended.



Create JUnit Test Class

- Here is the PersonTest class that was created:

Person.java | PersonTest.java

```
2
3   import ...
6
7   class PersonTest {
8
9       @Test
10      void getName() {
11      }
12
13      @Test
14      void getId() {
15      }
16
17      @Test
18      void setName() {
19      }
20
21      @Test
22      void setId() {
23      }
24  }
```

Create JUnit Test Class

- We will now add testing code 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.

Test Method

Sample Test Class and Test Method

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

```
class PersonTest {

    @Test
    void setName() {
        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 IntelliJ will show that in the Test Results window

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

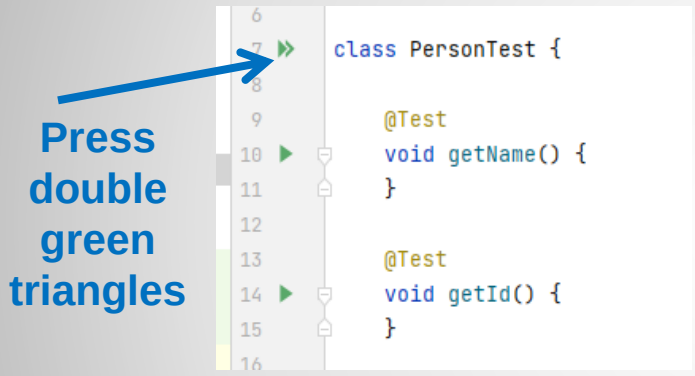
assertEquals and Objects

- 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.

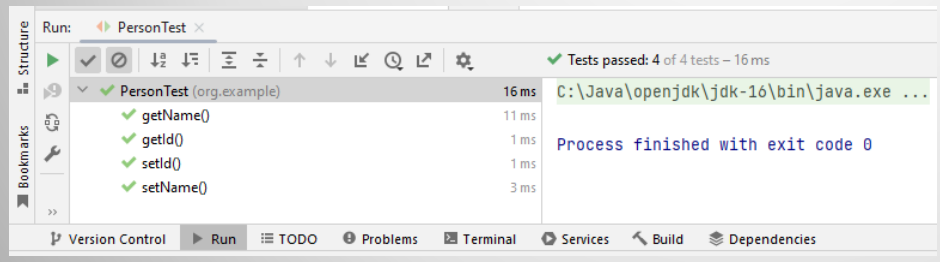
assertEquals and Objects

Running Tests in IntelliJ

- Run unit tests. Click the double green triangles in the left margin on the class header line. Choose the Run option from the menu.
- This will execute all the methods decorated with the @Test annotation.
- Running test code does NOT run the main method (the above only runs the JUnit tests).



- Test results appear in bottom window.



Running Tests in IntelliJ

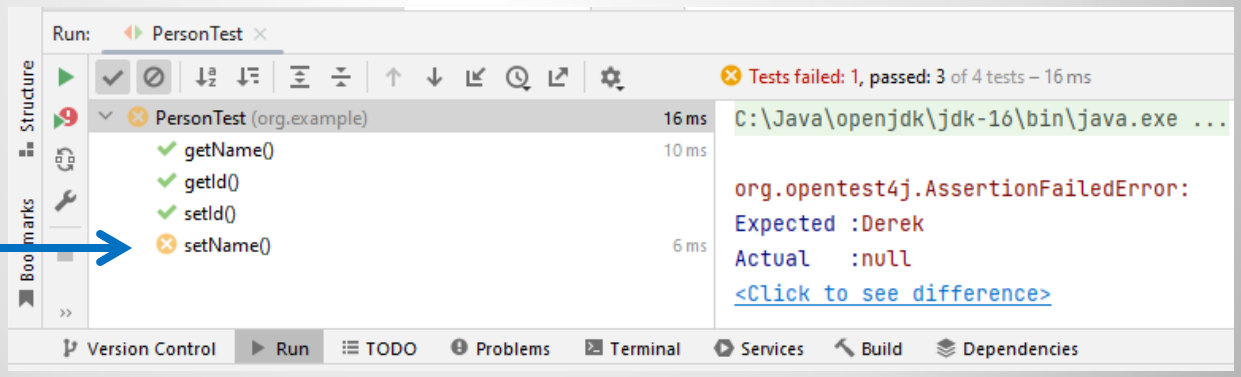
Sample Test Failure

- Any methods that have assert statements that fail will cause messages to appear in the test output window.
- For example, change Person.setName to the following:

```
public void setName(String name) {
    //this.name = name;
    name = this.name; // This now causes an error
}
```

- Run the tests again and you should see the following output:

setName now fails!!!



Sample Test Failure

End of Slides