**Points possible:** 70

| Category | Criteria | % of Grade |
|---|---|---|
| **Functionality** | Does the code work? | 25 |
| **Organization** | Is the code clean and organized? Proper use of white space, syntax, and consistency are utilized. Names and comments are concise and clear. | 25 |
| **Creativity** | Student solved the problems presented in the assignment using creativity and out of the box thinking. | 25 |
| **Completeness** | All requirements of the assignment are complete. | 25 |

**Instructions:** In Eclipse, or an IDE of your choice, write the code that accomplishes the objectives listed below. Ensure that the code compiles and runs as directed. Take screenshots of the code and of the running program (make sure to get screenshots of all required functionality) and paste them in this document where instructed below. Create a new repository on GitHub for this week's assignments and push this document to the repository. Additionally, push your Java project code to the same repository. Add the URL for this week's repository to this document where instructed and submit this document to your instructor when complete.

**Coding Steps:**

1.  Create a new Maven project. In Eclipse…
    a.  Right-click in Project Explorer, select "New / Project". Select "Maven Project". Click "Next".
    b.  Check "Create a simple project (skip archetype selection)". Click "Next".
    c.  Enter Group Id: "my.unit.test". Enter Artifact Id: "unit-test-assignment". Click "Finish".
    d.  The project "unit-test-assignment" should appear in the Package Explorer. Click the down arrow next to "unit-test-assignment" to expand it. Double-click on "pom.xml" to open it in the editor.
    e.  Put a few blank lines between `<version>0.0.1-SNAPSHOT</version>` & `</project>`.
    f.  Copy and paste the following code into the blank area you just created.
    g.  Save the file.
    h.  For Eclipse only: right-click on "unit-test-assignment" in the Project Explorer. Click on "Properties". Click "Java Compiler". Make sure "Enable project specific settings" is checked. Uncheck "Use compliance from execution environment 'J2SE-1.5' on the 'Java Build Path'. Set "Compiler compliance level" to 11. Click "Apply and Close".
    i.  If asked to rebuild the project, click "Yes".

2.  Create a class named "TestDemo" under src/main/java in the default package.
    a.  Create an instance method (not static) named addPositive. It should take two int parameters and return an int.
    b.  If both parameters are positive (greater than zero) return the sum of the parameters. If either parameter is zero or negative, throw an IllegalArgumentException with the message "Both parameters must be positive!". IllegalArgumentException is in the java.lang package so you won't need an import statement.
    c.  Save the file.

3. In Package Explorer, find "src/test/java" and right-click on it. Select "New / JUnit Test Case". In the "Name" field, enter "TestDemoTest". Make sure that "New JUnit Jupiter test" is selected. Make sure that "@BeforeEach setUp()" is checked. Click "Finish".

4. In TestDemoTest.java, add a private instance variable of type TestDemo named testDemo.
   a. In the setUp method, create the TestDemo object. This will ensure that a new TestDemo object is created before each test.
   b. Change "`@Test`" to "`@ParameterizedTest`". Add the import statement for `org.junit.jupiter.params.ParameterizedTest`.
   c. Change the name of method "test" to "`assertThatTwoPositiveNumbersAreAdded-Correctly`".
   d. Add four parameters to `assertThatTwoPositiveNumbersAreAddedCorrectly` as shown:

   | Type | Name |
   |---|---|
   | int | a |
   | int | b |
   | int | expected |
   | Boolean | expectException |

   e. Write the test. Remove the "`fail`" line. Test the value of `expectException`. If it is `false`, assert that when TestDemo.addPositive is called with values `a` and `b`, that the result is the same as the parameter `expected`. The assertion should look like this:

   ```
   if(!expectException) {
     assertThat(testDemo.addPositive(a, b)).isEqualTo(expected);
   }
   ```
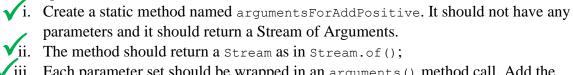
   f. Add the test for the thrown exception in an else clause. Use `assertThatThrownBy` for this. Add the `static import org.assertj.core.api.Assertions.assertThatThrownBy;`
   g. As a parameter to `assertThatThrownBy`, add a Lambda expression with no parameters. The Lambda body should be the method call to `testDemo.addPositive`.
   h. Use the assertion `isInstanceOf(IllegalArgumentException.class)` to ensure that the correct exception is thrown.
   i. If this is too confusing, you can "cheat" and copy this:

   ```
   assertThatThrownBy(() ->
       testDemo.addPositive(a, b))
           .isInstanceOf(IllegalArgumentException.class);
   ```

j. Add the parameter source method.
   i. Create a static method named `argumentsForAddPositive`. It should not have any parameters and it should return a Stream of Arguments.
   ii. The method should return a `Stream` as in `Stream.of()`;
   iii. Each parameter set should be wrapped in an `arguments()` method call. Add the static import for arguments: `org.junit.jupiter.params.provider.-Arguments.arguments`.
   iv. So, if you are adding 2 and 4 to get the value of 6 and are not expecting an exception, you need to do:

   ```
   arguments(2, 4, 6, false)
   ```

   v. Add as many arguments lines as needed to test the `addPositive` method thoroughly. Make sure to add some zero or negative arguments.
k. Just below the `@ParameterizedTest` annotation, add the annotation `@MethodSource`. Pass a single parameter to `@MethodSource`. It must be the fully-qualified (includes package) class name of the test followed by a # sign followed by the name of the method that supplies the parameters. Since the test is in the default package, there is no package in the fully-qualified class name. So,

   ```
   @MethodSource("TestDemoTest#argumentsForAddPositive")
   ```

5. In TestDemo.java, add another method named `randomNumberSquared`. This method obtains a random int between 1 and 10 and then returns the square of the number.
   a. `randomNumberSquared` should return an int and not take any parameters.
   b. It should call another method in the same class named `getRandomInt`. This method takes no parameters and must be package visibility so that the test can see it. `getRandomInt` should look like this:

   ```
   int getRandomInt() {
     Random random = new Random();
     return random.nextInt(10) + 1;
   }
   ```

   c. `randomNumberSquared` should return the value obtained from `getRandomInt` multiplied by itself.
6. Write a test for `randomNumberSquared` in `TestDemoTest.java`. Since you don't know what `getRandomInt` will return (that's the point of random, after all), you will need to mock it out and supply a known value.
   a. Create a method annotated with `@Test` named `assertThatNumberSquaredIsCorrect`. The method must have package visibility (not public!) or JUnit won't find it. The annotation `@Test` is in the `org.junit.jupiter.api` package.
   b. To mock the `TestDemo` class, use `Mockito.spy`. The spy method can be imported with a static import of org.mockito.Mockito.spy.

   ```
   TestDemo mockDemo = spy(testDemo);
   ```

✓ c.  Program the mocked `TestDemo` object to return 5 when the `getRandomInt` method is called. Remember to use the form: `doReturn(aValue).when(mockedObject).-methodCall()`. You can use a static import for `doReturn`: `import static org.mockito.Mockito.doReturn;`

   `doReturn(5).when(mockDemo).getRandomInt();` ✓

✓ d.  Call the method `randomNumberSquared` on the mocked TestDemo object. This will call the stubbed out (mocked) method `getRandomInt`, which now should return the value 5.

   `int fiveSquared = mockDemo.randomNumberSquared();` ✓

✓ e.  Use `assertThat` to test that the value returned from `randomNumberSquared` is equal to 5 squared.

   `assertThat(fiveSquared).isEqualTo(25);` ✓

✓ f.  You don't need to verify the mocked method call – you know it was called since the return value is correct.

# Screenshots of Code:

**TestDemo.java**

```
J TestDemo.java ×   J TestDemoTest.java
 1  import java.util.Random;
 2
 3  public class TestDemo {
 4
 5
 6      public int addPositive(int a, int b) {
 7          if(a > 0 && b > 0) {
 8              int sumOfPositives = a + b;
 9              return sumOfPositives;
10          }//end IF
11          else {
12              throw new IllegalArgumentException("Both parameters must be positive!");
13          }//end ELSE
14      }//end METHOD addPositive
15
16
17      int getRandomInt() {
18          Random random = new Random();
19          return random.nextInt(10) + 1;
20      }//end METHOD getRandomInt
21
22      public int randomNumberSquared() {
23          int random = getRandomInt();
24          int randomSquared = random * random;
25
26          return randomSquared;
27      }//end METHOD randomNumberSquared
28
29  }//end CLASS
30  |
```

## TestDemoTest.java

```java
  TestDemo.java        TestDemoTest.java  ×
  1⊖ import static org.assertj.core.api.Assertions.assertThat;
  2  import static org.assertj.core.api.Assertions.assertThatThrownBy;
  3  import static org.junit.jupiter.params.provider.Arguments.arguments;
  4  import static org.mockito.Mockito.doReturn;
  5  import static org.mockito.Mockito.spy;
  6
  7  import java.util.stream.Stream;
  8
  9  import org.junit.jupiter.api.BeforeEach;
 10  import org.junit.jupiter.api.Test;
 11  import org.junit.jupiter.params.ParameterizedTest;
 12  import org.junit.jupiter.params.provider.Arguments;
 13  import org.junit.jupiter.params.provider.MethodSource;
 14
 15  class TestDemoTest {
 16      private TestDemo testDemo;
 17
 18⊖     @BeforeEach
 19      void setUp() {
 20          testDemo = new TestDemo();
 21      }// end METHOD setUp
 22
 23⊖ @ParameterizedTest
 24  @MethodSource("TestDemoTest#argumentsForAddPositive")
 25      void assertThatTwoPositiveNumbersAreAddedCorrectly(int a, int b, int expected, Boolean expectException) {
 26
 27 //GIVEN: two integers
 28
 29          if (!expectException) {
 30              assertThat(testDemo.addPositive(a, b)).isEqualTo(expected);
 31          } // end IF
 32          else {
 33              assertThatThrownBy(() -> testDemo.addPositive(a, b)).isInstanceOf(IllegalArgumentException.class);
 34          } // end ELSE
 35
 36  //WHEN: method adds them together
 37
 38  //THEN: the sum of the integers is added
 39
 40  //AND: throw and exception if an int is <=0
 41
 42      }// end METHOD assertThatTwoPositiveNumbersAreAddedCorrectly
 43
 44⊖     static Stream<Arguments> argumentsForAddPositive() {
 45          // @formatter:off
 46          return Stream.of(
 47              arguments(2, 4, 6, false),
 48              arguments(1, 2, 3, false),
 49              arguments(-1, 2, 1, true),
 50              arguments(1, -2, 1, true),
 51              arguments(0, 1, 1, true),
 52              arguments(1, 0, 1, true),
 53              arguments(0, 0, 0, true)
 54          );
 55          // @formatter:on
 56      }// end STREAM
 57
 58⊖ @Test
 59      void assertThatNumberSquaredIsCorrect() {
 60
 61          TestDemo mockDemo = spy(testDemo);
 62          doReturn(5).when(mockDemo).getRandomInt();
 63
 64          int fiveSquared = mockDemo.randomNumberSquared();
 65
 66          assertThat(fiveSquared).isEqualTo(25);
 67
 68      }// end TEST assertThatNumberSquaredIsCorrect
 69
 70
 71  }// end CLASS
 72
```

# Screenshots of Running Application:

## TestDemo.java - Methods



## TestDemoTest.java - Tests

## URL to GitHub Repository:

https://github.com/JaxYoungblood/Week12-UnitTestAssignment.git