COMP1035 – Lab 06: JUnit Testing

You should do this worksheet individually, but "sit" in your groups (and talking)

- **No one share submission as a team**.
- Create a Lab 06 subfolder in the /src/ part of your team repository, with 5 or 6 subfolders (1 per team member) in it.
- You should git commit at the end and add links to your individual work in the README.md.

First Java:

- I have been informed that everyone has his/her own preferred Java IDE, so please continue to use the IDE you wish.
- The instruction below is based on Eclipse IDE but should work similarly with other IDEs.
- Create a workspace whenever is default.

Useful Links:

1. https://www.tutorialspoint.com/junit/junit_test_framework.htm
2. https://junit.org/junit4/javadoc/latest/org/junit/Assert.html

---

A. Initial Exercises

1. Start a New >> Java Project.
   a. Call it anything you like, perhaps "COMP1035-Lab06".
   b. Untick "Use default location".
   c. Browse to your chosen folder (personal repository or personal subfolder in team repository).
   d. Leave all defaults, press finish, and DON'T create a module-info.java.

2. Expand your project in the package explorer and right click on the "src" folder in it.

3. Add a Java Class File – call it "`MathModule`".
   a. Tick "public static void main".

4. Add a function to the code, so it looks as below,

```
public class MathModule {
  public static void main(String[] args) {
       // TODO Auto-generated method stub }

  public static int myMultiply(int firstNum, int secondNum) {
       return firstNum * secondNum; }
}
```

5. Right click on the project (e.g. COMP1035-Lab06).

6. Add a JUnit Test.
    a. Call it something like "`TestMyMathModule`".
    b. Set the class that it is testing to be: "`MathModule`", or whatever you called it.
    c. Press "Finish".

7. It will ask you if you want to add JUnit to your project.
    a. Press "OK"!

8. Run it with the `fail()` command – and see it fail.

9. Change `@Test` to `@Ignore` for the existing test.
    a. A red line will appear under your `@Ignore`.
    b. Clicking on the red alert icon to the left of the line will allow you to add an import.

10. Add the following test,

```
@Test
void test1() {
     int myAnswer = MathModule.myMultiply(4, 2);
     assertEquals(8, myAnswer);
}
```

11. Creating a `@BeforeAll` command,
    a. First, define two private static ints – input1 and input2 at the top of the JUnit class.
        i. Immediately after the line "`class TestMyMathModule`".
    b. Create a `@BeforeAll` function after that (but before your tests).

```
@BeforeAll
static void setup() {
     input1 = 3;
     input2 = 6;
}
```

    c. Use the red alert icon to import `BeforeAll` (as with the `Ignore`).
    d. Edit your `test1` to use the input1 and input2.
        i. If you don't change the `assertEquals` it will fail.
        ii. And then you can double click on the failure trace to see why.
        iii. Then change the `assertEquals` to be 18.
        iv. And running it again will pass.

B. Harder Exercises (you may wish to use online resources to help)

12. Add the following lines of code to the public static void main of your original Java class,

```
System.out.println(myMultiply(2000000000, 6));
```

    a. What did the system print out? Compared to what you expected. (you may re-read the slides).

13. Let's build a test for this in the JUnit file.
    a. Create a third variable `input3`.
    b. Set it to = `2000000000` in your `@BeforeAll` function.
    c. Change `myAnswer` to be the multiple of `input2` and `input3`.
    d. Add a second `assertEquals` in the `test1()` to match what it should be.
    e. What red-underline error is showing?
        i. We could convert the function to work with `Longs` instead of `Ints`, but instead, let's stay within `Ints`, and test for errors!
    f. Change your multiply function,

```
long newAnswer = (long)firstNum * secondNum;
if (newAnswer > Integer.MAX_VALUE) {
     throw new Exception("Number too big");
}
return firstNum * secondNum;
```

    This will offer you the option to add a "`throw`" to the function.

    g. For practice, change your `public static void main` to try/catch the error,

```
try {
     int test = myMultiply(2000000000, 6);
     System.out.println(test);
} catch(Exception e) {
     System.out.println(e.toString());
}
```

    h. Now, let's fix the JUnit test – convert your JUnit test to become,

```
int myAnswer = 0;
try {
     myAnswer = MathModule.myMultiply(input1, input2);
} catch(Exception e) { }
assertEquals(18, myAnswer);
```

i. Now, let's write a test for the error being created.

```
void testError() {
     int myAnswer = 0;
     try {
          myAnswer = MathModule.myMultiply(input2,
input3);
     } catch(Exception e) {
          if(e.getClass() == Exception.class) {
               return; // it passed
          }
     }
     fail("it failed");
}
```

You can now do this `assertThrows()` in JUnit5 – you can have a go! https://howtodoinjava.com/junit5/expected-exception-example/

14. How would you have to change your multiply function to handle `input4 = -2000000000`?
    a. You may need to use `Integer.MIN_VALUE`.

C. Advanced Exercises (you may wish to use online resources to help)

15. You may want to comment out that test code in the `public static void main`.

16. Plan a series of tests for a new dividing function, including a test for "divide by zero".
    a. Create a stub method in the `MathModule` class for it.

17. Write JUnit test code for the planned tests for the dividing function.
    a. Do an `assertEquals` test for 6 divided by 3 (to test it works).
    b. E.g. `assertEquals` – how should you handle values between integers? E.g. 7 divided by 3.
    c. E.g. write a test that checks that an error is thrown correctly (passes if error is thrown) for dividing by 0.
    d. Read https://junit.org/javadoc/latest/org/junit/Assert.html for more.

18. Now, to pass your tests, write the Java code in the new dividing function.
