For This Lab, go into Blackboard and place your answers in the Lab 13 "quiz" in Blackboard.

This lab uses links to web pages found at: https://www.cs.uic.edu/bin/view/CS211/LabexercisesF16

Testing is when we determine whether a program executes correctly. I.E. Does it perform as described in the program specification?

Note that debugging is trying to find the causes of failed tests and how to correct the issues. Debugging is different from testing! In this lab, we will focus on testing.

What are Test Cases and What Kinds of Testing Cases Exist?

Test cases are composed of two things:

1. A description of the actions to be given (often a set of input or parameter values) and
2. The expected results for that given input

When running a test case, the result is whether the test passed or failed. This is done by executing the set of input and seeing if the expected results were generated. If so, the test case passes. Otherwise, it fails.

Consider the following Specification for an application:

Program Specification for a Simple Tax Calculation:

Create a web form that will allow the user to calculate the amount of taxes owed to the government. The user should enter the income amount in an input field, then press/click a button and the amount of taxes owed is to be displayed. If the user enters a non-numeric income amount or a negative income amount, display an error message telling the user that a positive numeric value must be entered. The amount of taxes owed is determined by:

- *If the income amount is $5,000 or less, the tax amount is 10% of the income amount.*
- *If the income amount is more than $5,000 and is $50,000 or less, the tax amount is 15% of the income amount.*
- *If the income amount is more than $50,000, the tax amount is 20% of the income amount.*

Equivalence Class Testing: For Equivalence Class Testing, first identify a range of inputs that should result in same calculation(s) being done on any of the input value. This range will be one Equivalence class. Test cases should be developed to test inputs from all possible ranges of inputs (including error inputs). For the above specification, there are five Equivalence Classes:

EC1 - the values starting from $0 up to and including $5,000.
EC2 - the values greater than $5,000 up to and including $50,000.
EC3 - the values greater than $50,000.
EC4 - the values less than $0 (negative input).
EC5 - the non-numeric values.

Each Equivalence Class should produce (at least) one test case: a good input value should be selected near the medium (middle) of the range of possible input values. For Equivalence Classes that have an infinite range (i.e. …amount is more than $50,000…), select an input value that is toward the middle of typically "expected input" (for this example one might assume a maximum input near $500,000).

1. Which of the following is the best Equivalence Class Test Case input for EC2?

$2,500 $5,000 $10,000 $22,500 $50,000 $75,000

2. Which Equivalence Class(es) have a finite range of inputs? (Indicate all that apply.)

EC1 EC2 EC3 EC4 EC5

3. Which Equivalence Class is being testing when the input of "Apple" is given as input?

EC1 EC2 EC3 EC4 EC5

Boundary Case Testing: Test cases should also include inputs that verify the proper functioning of the program for values at the boundary between two Equivalence Classes. This are often called **Boundary Case Testing**. Each boundary should generate multiple test cases (normally three): a test for the boundary value, and a test for each value on either side of the boundary value. For the above specification,

BC1 - one Boundary Value is the value of $0. The boundary between EC1 and EC4
BC2 - another Boundary Value is the value of $5,000. The boundary between EC1 and EC2
BC3 - a third Boundary Value is the value of $50,000. The boundary between EC2 and EC3

4. Which of the following values is NOT an input value for a Boundary Case Test for this example?

-1 0 4999 22500 50000 50001

Unit Testing

Testing is often done using a technique called **Unit Testing**, where a method/function/unit in the program being tested is called with the specified input and a result is produced. This actual result is compared with the expected result. If the two are the same, the test passes. If the actual result and the expected result are not the same, the test fails.

Often this can result in a huge number of test cases. The above specification would produce at least 14 test cases (5 Equivalence Test Cases and 9 Boundary Test Cases). Most often this is done by writing code that will actually call the method/function/unit being tested with each of the test case values and compare the actual result with the expected result. Fortunately, there exist many automated ways to run test cases. In Java, one such way of doing this is using the **JUnit tool kit** which is built into many IDE's (including DrJava).

Each **JUnit** automated test is written as a method were the input, expected result and the actual result are specified as variables. This test method normally creates an instance of the class that contains the starting method of the code being testing, makes a call to the starting method with proper parameters and stores the returned value as the actual result. The JUnit code then calls the assertEquals() method to determine if the expected result matches the actual result.

In the code files of Lab13.java is a program that calculate bowling scores from the bowling frame information. The method scoreGame( ) is the method we will be testing. It takes a string with the frame-by-frame information and returns the score. Your job will be to set up a number of JUnit test cases and specify whether each test case passed or failed. The first test cases are written in a file called Lab13Test.java. Note that this "program" will not run on its own (there is no main( ) method). It relies on the built-in Testing functionality of the IDE to run the JUnit test cases.

The following specifies how to run the Test Case code using the DrJava IDE. If you do not have the DrJava IDE, you can download the Java .jar file (or other versions) at:

http://www.drjava.org/download.shtml

For the machines in the CS Lab, the DrJava IDE can be run by typing the command:
~troy/runDrJava

To run a JUnit testing file in DrJava, you do the following:

1. Load the program being tested into DrJava using the **Open** operation. For Lab 13 load the program file: Lab13.java
2. Load the JUnit testing file into DrJava using the **Open** operation. For Lab 13 load the file: Lab13Test.java
3. Edit the JUnit testing file so it contains all of the needed tests and **Save** the modified code.
4. Compile the code in both Lab13.java and Lab13Test.java using the **Compile** operation
5. Do NOT run the compiled code in DrJava, instead **Test** the code using the **Test** operation.

An example to test this method using JUnit is as follows (and see Lab13Test.java). The comments on the side are to give a line-by-line description of the JUnit code (please read them).

```
@Test                                   // Keyword for JUnit that specified a testing method
public void testAllMisses() {           // The testing method name is used in the JUnit report
 Lab13 game = new Lab13();              //  Create the class instance containing the code being tested
 String input = "--|--|--|--|--|--|--|--|--|--||";   //  Create the input ( one string in this case )
 int expected = 0;                      //  Create the expected result
 int result = game.scoreGame( input );  //  Call the method and store the actual result
 assertEquals(expected, result);        //  check if the expected result matches the actual result
}
```

When creating a new test case, you will need to modify three things:

1. The name of the testing method. (Which is testAllMisses in the above JUnit test)
2. The value given to the String input. (Which is "--|--|--|--|--|--|--|--|--|--||" above)
3. The value given to the integer expected. (Which is 0 above)

For this lab open DrJava and use the TEST operations to run the following unit testing. (You can use other IDE's for this lab, but

5. Does the following test pass or fail?

Test name:	all misses
The input is the string:	"--|--|--|--|--|--|--|--|--|--||"
The expected result:	0

Note this the first test method given in the file Lab13Test.java

6. Does the following test pass or fail?

Test name:	all strikes
The input is the string:	"X|X|X|X|X|X|X|X|X||XX"
The expected result:	300

Note this test method is given in the file Lab13Test.java, but you will need to comment out the @Ignore keyword to get it to run.

7. Does the following test pass or fail?

Test name:	five pins and miss
The input is the string:	"5-|5-|5-|5-|5-|5-|5-|5-|5-|5-||"
The expected result:	50

Note this test method NOT is given in the file Lab13Test.java, so you will need to create it.

8. Does the following test pass or fail?

Test name:	five pins and spare
The input is the string:	"5/|5/|5/|5/|5/|5/|5/|5/|5/|5/||5"
The expected result:	150

Note this test method NOT is given in the file Lab13Test.java, so you will need to create it.

9. Does the following test pass or fail?

Test name:	nine or more on average
The input is the string:	"18|X|54|9/|81|X|-8|7-|X|43||"
The expected result:	121

Note this test method NOT is given in the file Lab13Test.java, so you will need to create it.

10. Does the following test pass or fail?

Test name:	one pin
The input is the string:	"1-|--|--|--|--|--|--|--|--|--|--||"
The expected result:	1

Note this test method is given in the file Lab13Test.java.