Lab 6A, Testing

# Lab Intro & Prep

In this week’s lab, you are given some code with a simple test harness, and four empty functions which fail most of the tests. Your task is to implement the functions according to the descriptions in the comments so that they pass all the tests.

* Create a new Java Project in Eclipse named **Week 6**

## Learning Objectives

* Writing a simple test harness for black-box testing
* Revision of a range of Java features

# Exercise 1 – Function Writing

1. Download **TestHarness.java** and **Week6Functions.java** from Moodle (located in **Week 6 Lab Source Code** folder) and import into a new Java project in Eclipse.
2. Run the code (**TestHarness.java)** – you will see a lot of text scrolling past with the results of tests – some passed, some failed.
3. Look at the **main** method of the **TestHarness** class – you will see that it calls a number of test procedures, each testing one of the functions in the **Week6Functions** class.

Your task is to implement the functions so that all of the tests pass. You can comment out all of the calls to the test procedures in the main class apart from the one you are working on at any one time.

## The Functions (located in the Week6Functions class)

**int** average( **int** [] inputArray )

Returns the average value of an array of integers, rounded down to the nearest integer. An empty array should give an average of 0 (revision: arrays).

**float** range( **float** [] inputArray )

Returns the maximum value minus the minimum value for an array of floats which all lie between -1000 and 1000. Empty array returns 0.0f. (revision: arrays).

**int** collatz( **int** n )

Returns the Collatz function, which is equal to 3\*n + 1 if n is odd, or n/2 if n is even (revision: modulus operator %).

**int** compoundInterest( **int** capital, **int** ratePC, **int** years )

Returns the value of **capital** rounded to the nearest integer after years (whole number of **years**) at a percentage interest rate **ratePC**. The formula for this is



You can use the java function **Math.pow( x, p )** which returns 𝑥p.

A number can be rounded using **Math.round( double )**

# Extension Exercise

For this exercise, you are required to produce a class with a single function which calculates the interest rate problem from this week’s lecture, and a testing program to test the function. The test plan developed in the lecture has 10 test cases (provided in Table 1). The function should indicate invalid inputs by returning -1.

**Suggested Approach:**

1. Make a class called **TestInterest** with a main method, and a class **BankAccount** with a single public method **int balance( int balance, int years)**. This can return 0 as a placeholder.
2. Using this week’s **TestHarness** class for inspiration (in particular the **testCompoundInterest** function), write the testing code. You will need to add a testing function, and an instance of the **BankAccount** class as a member of the test class.
3. Check that the tests run (even if they fail), then work on the **balance** function. The solution to the **compoundInterest** function in this week’s lab should be a big help here!

Table - Test Plan

|  |  |  |  |
| --- | --- | --- | --- |
| **Test#** | **Deposit** | **Term** | **Expected Result** |
| 1 | 1 | 1 | 1 |
| 2 | 1000 | 10 | 1051 |
| 3 | 1001 | 1 | 1016 |
| 4 | 10000 | 10 | 11605 |
| 5 | 10001 | 5 | 11315 |
| 6 | 50000 | 10 | 64004 |
| 7 | 0 | 1 | Error |
| 8 | 50001 | 10 | Error |
| 9 | 1000 | 0 | Error |
| 10 | 1000 | 11 | Error |