Unit Test Document for PIMS

1. Introduction

In this document, we give the test plan and test results for unit testing of some of the key modules of the Personal Investment Management System (PIMS).

2.Unit Testing Methodology

2.1 Selection of units*:* We selected only the most important functional and critical classes for formal unit testing. In the test environment we used, a unit for unit testing is a class. Here we illustrate the testing methodology by discussing only two classes.

2.2 Test Scripts: As we used Junit for unit testing. The test scripts were java programs and each test case corresponded to a method in these java programs.

2.3 Fixing of Defects: The programmer fixed the defects found. Unit testing was successfully complete only if the script executed without any defects.

2.4 Test Script Enhancement: As testing proceeded, some new test cases were added. This was done by adding new test methods to the testing program.

3. The Testing Tool: Junit

We used Junit as the tool for unit testing. It is open source software which can be used to test Java modules. It can be freely downloaded from the website *www.junit.org*
The general way to test the module (usually a 'class') by Junit: We create a class extending TestCase(a predefined class of Junit), and write the following methods:

(a) setUp(): In this we instantiate various objects needed to perform the testing.
(b) tearDown(): In this we deallocate all or some of the memory which was used up by objects created in setUp() method. This is called at the end of all tests.
(c) suite(): This method is used to create a *test suite*, which specifies as to which tests will be performed.
(d) Various methods of the name testXXX(): These methods contain the actual code for testing. In any such method, we do whatever operations we want to do, and then call the method assertTrue()/assertFalse(), with a boolean as the argument, which specifies as to what condition we wish to hold true/false, for being convinced that the tested method performs correctly.

4. Tests Performed

We unit tested the methods of following two classes: (a) Alerts, and (b) Investment

(a) Testing methods of class Alerts:

Operation performed	**Condition tested**	**Actual result**
Nil	Number of alerts should be	Test passed.

	zero	
Create a new alert, with a particular date and particular description.	Retrieve the first alert. Its date and description should match with the date and description with which we created the alert.	Test passed.
Delete the first alert.	Number of alerts should be zero.	Test passed.
Create two new alerts, one with a past date and other with a future date.	Number of alerts should be two.	Test passed.
Retrieve for pending alerts.	Number of alerts returned should be one.	Test passed.
Delete alert number 0 twice.	Number of alerts should be zero.	Test passed.

(b) Testing the methods of class Investment for correct manipulation of portfolios and securities

Operation performed	**Condition tested**	**Actual result**
Create a portfolio with name PF1	Check that there exists a portfolio with name PF1	Test Passed.
Create a bank type security bankTest in PF1	Check that there exists a security bankTest in portfolio PF1.	Test Passed
Add three transactions, with any attribtues.	Retrieve all the transactions, and check that a particular transaction had the same details as you entered.	Test Passed
Delete security bankTest.	Check that there is no security with name bankTest in portfolio PF1	Test Passed
Add a security shareTest in PF1	Check that shareTest exists in PF1	Test Passed
Rename shareTest to shareTestNew.	Check that shareTest does not exist in PF1 and shareTestNew exists in PF1.	Test Passed
Delete shareTestNew	Check that shareTest New does not exist in PF1.	Test Passed
Rename PF1 to PF2.	Check that portfolio PF1 does not exist and PF2 exists.	Test Passed

Delete PF2	Check that PF2 does not exist.	Test Passed

Operation performed	**Condition tested**	**Actual result**
Create portfolio PF1	Check that PF1 exists	Test Passed
Create security bankTest in PF1 with rate of interest = 1%.	Check that bankTest exists in PF1	Test Passed
-	Check that roi of bankTest is 1%	Test Passed
Add three transactions.	Find out the networth of this bank security and compare with correct value.	Test Passed
Delete bankTest	Check that bankTest does not exist.	Test Passed
Add security shareTest	Check that shareTest does not exist.	Test Passed
Add five transactions.	Find networth and compare with correct value.	Test Passed
-	Find roi and compare with correct value.	Test Failed. A bug in roi computation found and fixed. Then test passed.
Delete shareTest.	Check that shareTest does not exist in PF1.	Test Passed
Delete PF1	Check that PF1 does not exist.	Test Passed

5. Results:

When the script written (which is in fact a Java file) and compiled and run, it gives us the number of tests actually executed, and also in how many of them expected results were obtained, in how many expected results were not obtained, and how many tests could not go to completion.

For first test suite (testing for Alert.java):

6 tests: success: 6, failure: 0, error: 0

For second test suite (first test suite for Investment.java)

9 tests: success: 9, failure: 0, error: 0

For third suite (second test suite for Investment.java)

10 tests: success:9, failure: 1, error:0

The failure was due to a bug in the ROI calculation; the bug was fixed.