UNIVERSITY OF ONTARIO INSTITUTE OF TECHNOLOGY

SOFTWARE QUALITY ASSURANCE
(CSCI3060/SOFE3980)

Watermelon Banking System

Phase 5: Back End Unit Testing

March 30th, 2016

Luisa Rojas (100518772)
Truyen Truong (100516976)
Denesh Parthipan (100536986)

The testing methods to be used in this phase are statement coverage, decision coverage and loop coverage; moreover, the unit testing Framework utilized to apply them was Junit (version 4.10).

The first one mentioned above, statement coverage, will traverse through every line of code in the Watermelon System, which is comprised in four classes: `Main.java`, `User.java`, `UpdateMaster.java` and `Utilities.java`. These will be the ones the tests will be developed for and applied to.

Loop coverage will aim to exercise the body of a certain loop in the program zero times, once, twice and “many times”. For the purpose of this assignment, the loop chosen was the if statement in the `getAllMasterAccounts()` method under the `Utilities` class and it was executed 3 times as the “many times” statement.

Finally, decision coverage consists of designing the test cases in a way that will allow the program to exercise every each decision every way possible. This is to be applied on if statements, while loops, switch statements, etc.

Note that an overlap between them when developing the tests, as stated during lecture, is inevitable.

> TEST RUN RESULTS

CLASS NAME: `MainTest.java`

TEST NAME	DESCRIPTION
`mainTest1`	If the executable file for this system is called with a number of arguments different than two, then the output should be an error to the terminal.
`mainTest2`	If the executable file for this system is called exactly two arguments, then the transaction file name variable should be assigned to the second argument provided.

CLASS NAME: `UserTest.java`

TEST NAME	DESCRIPTION
`userConstructorTest1`	Ensure that every time this constructor is called, the plan member gets assigned correctly.
`userConstructorTest2`	Ensure that every time this constructor is called, all class members are assigned correctly.

CLASS NAME: `UtilitiesTest.java`

TEST NAME	DESCRIPTION
`storeFileTest1`	Will ensure that if the system is unable to read in the corresponding file, it will output an error message.
`storeFileTest2`	If the inputs are all correct, the file will be read in and stored into a data structure successfully.
`tokenizeTransactionTest1`	Check that the method splits a typical transaction file-style line into its different components correctly.
`tokenizeTransactionTest2 *`	Tests that the system prints out a fatal error and then terminates the program if it notices bad input in the transaction file inputted.
`tokenizeMasterTest1`	Check that the method splits a typical master file-style line into its different components correctly.
`tokenizeMasterTest2 *`	Tests that the system prints out a fatal error and then terminates the program if it notices bad input in the master file inputted.
`isAdminTest1`	Make sure the output is false for an input of `N` (Non-student)
`isAdmintTest2`	Make sure the output is false for an input of `S` (Student)
`isAdminTest3`	Make sure the output is false for an input of that is not `N` nor `S`. Assumed the printing of the error gets read in, since the `return;` line is *after* it.
`getAllMasterAccountsLoopTestZero`	**(LOOP COVERAGE)** This test will test that the for loop in the `getAllMasterAccounts` method is able to <u>not</u> get executed a single time.
`getAllMasterAccountsLoopTestOne`	**(LOOP COVERAGE)** This test will test that the for loop in the `getAllMasterAccounts` method is able to get executed <u>once</u>.
`getAllMasterAccountsLoopTestTwo`	**(LOOP COVERAGE)** This test will test that the for loop in the `getAllMasterAccounts` method is able to get executed a <u>two</u> times.
`getAllMasterAccountsLoopTestMany`	**(LOOP COVERAGE)** This test will test that the for loop in the `getAllMasterAccounts` method is able to not get executed a “many” times; for this test, it will be executed <u>three</u> times.
`isNameUniqueTest1`	Ensure that, if the name passed as a parameter already exists

	in the system, the function output is `false`.
`isNameUniqueTest2`	Check that, if the name passed as a parameter does not yet exist in the system, the function output is `true`.
`isNumberUniqueTest1`	Test that, if the number passed as a parameter has already been assigned to a user in the system, the function output is `false`.
`isNumberUniqueTest2`	Check that, if the number passed as a parameter is available to be assigned to a user in the system, the function output is `true`.
`getAccIndexTest1`	Ensure that when the function gets passed the account number of an existing user, it outputs its index in the data structure it is stored in.
`getAccIndexTest2`	Make sure that when the function gets passed the account number of a user that does not exist, it returns `-1`.
`rewriteCurrentFileTest1`	Using the vector provided in the test, output a file based on the information included in said vector and then make sure the contents of the file are the same as the contents of the vector.
`rewriteCurrentFileTest2`	If the system is unable to write to or create a file, then an error is printed to the screen.
`rewriteMasterFileTest1`	Using the vector provided in the test, output a file based on the information included in said vector and then make sure the contents of the file are the same as the contents of the vector.
`rewriteMasterFileTest2`	If the system is unable to write to or create a file, then an error is printed to the screen.

***** *The test was coded and we attempted to execute it, but due to library issues that are out of our hands it was not possible. Note that it was still included in the submission; commented out, however.*

CLASS NAME: `UpdateMasterTest.java`

TEST NAME	**DESCRIPTION**
`updateMasterTest1`	Ensure that, in the switch statement, the cases `10` (`withdrawal`), `01` (`login`) and `00` (`logout`) get executed as well as their respective methods.
`updateMasterTest2`	Ensure that, in the switch statement, the `case 02` (`transfer`) gets executed as well as its respective method.
`updateMasterTest3`	Ensure that, in the switch statement, the `case 03` (`paybill`) gets executed as well as its respective method.
`updateMasterTest4`	Ensure that, in the switch statement, the `case 04` (`deposit`) gets executed as well as its respective method.

`updateMasterTest5`	Ensure that, in the switch statement, the `case 05` (`create`) gets executed as well as its respective method.
`updateMasterTest6`	Ensure that, in the switch statement, the `case 06` (`delete`) gets executed as well as its respective method.
`updateMasterTest7`	Ensure that, in the switch statement, the `case 07` (`disable`) gets executed as well as its respective method.
`updateMasterTest8`	Ensure that, in the switch statement, the `case 08` (`changeplan`) gets executed as well as its respective method.
`updateMasterTest9`	Ensure that, in the switch statement, the `case 09` (`enable`) gets executed as well as its respective method.
`withdrawalTest1`	Check that, given an inexistent account number, the method is unable to complete the transaction and returns `false`.
`withdrawalTest2`	Check that, if an administrator calls this transaction and the funds from the user are enough, the transaction goes through and the return value is `true`.
`withdrawalTest3`	Check that, if a standard user with a student plan calls this transaction and their funds are enough, the transaction goes through and the return value is `true`.
`withdrawalTest4`	Check that, if a standard user with a non-student plan calls this transaction and their funds are enough, the transaction goes through and the return value is `true`.
`withdrawalTest5`	Check that, if a standard user whose plan is not student nor non-student calls this transaction, the process does not go through and the return value is `false`.
`transferTest1`	Test that when a non-student standard user refers to the transfer transaction, the process gets carried out successfully - the new account balances are correct as well as count for number of transactions count.
`transferTest2`	Test that when a student standard user refers to the transfer transaction, the process gets carried out successfully - the new account balances are correct as well as the count for number of transactions count.
`transferTest3`	Test that when a user refers to the transfer transaction but the `withdrawal` transaction fails due to insufficient funds, then it prints an error to the screen and no transfer is processed.
`paybillTest1`	Ensure that when a user attempting to refer to paybill has insufficient funds, then the method outputs an error to the terminal.
`depositTest1`	Check that if a non-existent account number is passed to the

	deposit method as a parameter, then the transaction will not be completed and the return value will be false.
depositTest2	Check that if an administrator calls the deposit method and all constraints are valid, then it returns true.
depositTest3	Check that if a standard user, whose plan is student, calls the deposit method and all constraints are valid, then it returns true.
depositTest4	Check that if a standard user, whose plan is non-student, calls the deposit method and all constraints are valid, then it returns true.
depositTest5	Check that if a user, who is not an administrator nor a standard user, calls the deposit method, then the transaction doesn't go through and the method returns false.
createTest1	Make sure that if an account is created and it meets all the requirements, then it is successfully added to the corresponding data structure.
createTest2	Make sure that if an account is about to be created but one or more requirements are not met, then an error is printed to the screen and the size of the corresponding data structure remains the same.
deleteTest1	If an account is to be deleted, but it can not be found in the system, then an error is outputted to the terminal and the transaction is not completed.
deleteTest2	If an existing account is to be deleted, then check that the size of the data structure in which accounts are stored in is reduced in size by 1.
disableTest1	If an account is to be disabled, but it can not be found in the system, then an error is outputted to the terminal and the transaction is not completed.
disableTest2	If an existing account is to be disabled, then check that after the corresponding method has been called, the status flag has changed from 'A' to 'D'.
disableTest3	If an account is to be disabled, but the account status is already disabled ("D"), then an error is outputted to the terminal and the transaction is not completed.
changeplanTest1	Test that if this transaction is called, but the account to apply the change of plans to does not exist, then print an error to the screen.
changeplanTest2	Test that if this transaction is called, and the account subject

	to the change is currently a student (“S”), then its plan gets switched to non-student (“N”).
changeplanTest3	Test that if this transaction is called, and the account subject to the change is currently a non-student (“N”), then its plan gets switched to student (“S”).
enableTest1	If the enable method is referred to, but the account to be enabled does not exist in the system, then an error gets printed to the terminal and the transaction does not get completed.
enableTest2	If an existing account is to be enabled, then check that after the corresponding method has been called, the status flag has changed from ‘D’ to ‘A’.
enableTest3	If an existing account is to be enabled, but the account status is already enabled/active (“A”), then an error is outputted to the terminal and the transaction is not completed.
loginTest1	Check that if a standard user attempts to log in using an account number that does not exist, then an error is printed to the screen.
loginTest2	Check that if a standard user is able to successfully login into their account, given that all the requirements are met.
loginTest3	Check that if an administrator is able to successfully login into their account.
loginTest4	Test that if a user attempts to login as a mode different from standard (“S”) or administrator (“A”), then an error is displayed on the terminal and the login transaction is unsuccessful.
logoutTest	Ensure that when a user logs out of their account, the User currUser object is reset.

> IDENTIFICATION OF FAILURES

CLASS NAME	TEST CASE NAME	FAILURE DESCRIPTION
UpdateMaster.java	updateMasterTest9	The changeplan function checked for different constraints, but it did not actually change or assign the plan values to ‘S’ or ‘N’.
UpdateMaster.java	updateMasterTest9	The changeplan assigned empty chars to the different account tested. This was because, when implementing the

		functionality above, the lines that would no longer work with the new code were not removed.
`UpdateMaster.java`	`withdrawalTest1`	The corresponding amount would not get deducted from the account balance; it would remain the exact same for students (only).
`UpdateMaster.java`	`withdrawalTest2`	The `login` function in the class takes in an account number as its argument; however, administrators do not have account numbers.