Simon Fraser University
School of Computing Science

Cmpt 275 Project

Date: 2015-07-21

Team name: Eleven ®

Project Deliverable #: 7

Project Deliverable Name: Test Cases Document

Phase(s) covered by this deliverable: White Box Testing, Black Box Testing

Phase leader(s): Janice Sargent

Team members and Student #:

Yinglun Qiao	301191776
Janice Sargent	301160485
Zhi Cheng	301184486
Susan Hamilton	301209641
Seong Jun Kim	301154246
Nari Shin	301178863
Fan Liu	301168674
Te Lun Chen	301200832
Roy Chan	301202770

Grade:

Revision History

Revision	Status	Publication/ Revision Date	By:
1.0	created test case for black box testing	*July 21, 2015*	Janice Sargent
1.1	added purpose, inputs, and subset to blackbox testing	*July 22, 2015*	Nari Shin
1.2	formatted and edited black box testing formatted white box testing table of contents added	*July 23, 2015*	Susan Hamilton
1.3	added sections 1-4 to white box testing	*July 23, 2015*	Susan Hamilton Nari Shin
1.4	Added sections 5 and 6 to white box testing Format addition to black box testing - added 2	*July 23, 2015*	Susan Hamilton
1.5	Added equivalent classes for black box testing	*July 23, 2015*	Janice Sargent
1.6	Added demonstrate test case	*July 24, 2015*	Janice Sargent
1.7	Edited table of contents, entire document for clarification	*July 25, 2015*	Nari Shin

Table of Contents

White Box Testing

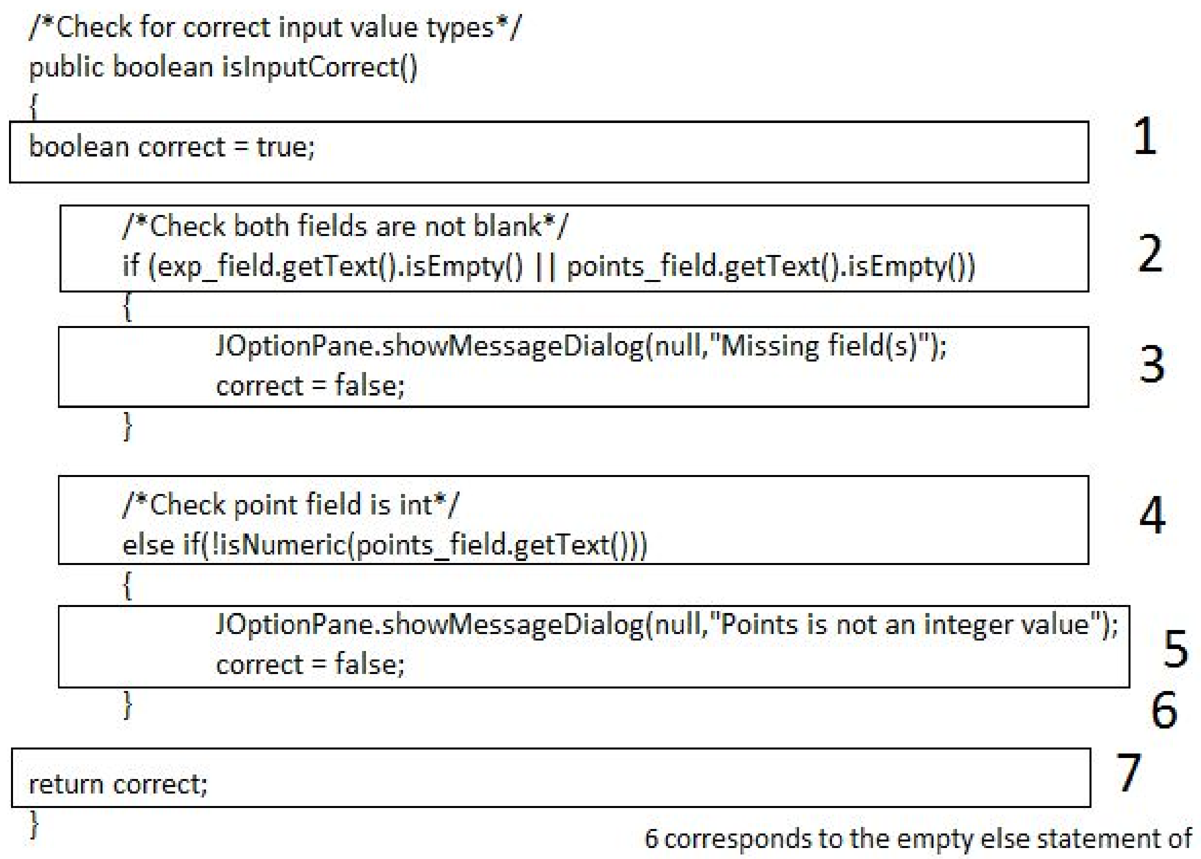
1 – Purpose

Describe the purpose of the method

The point of this method is to ensure that when creating a rubric that both expectation and points fields are not left blank and that the points field is an integer.

2 – Numbered Blocks

Analyze the code for the method and break it into appropriate numbered blocks

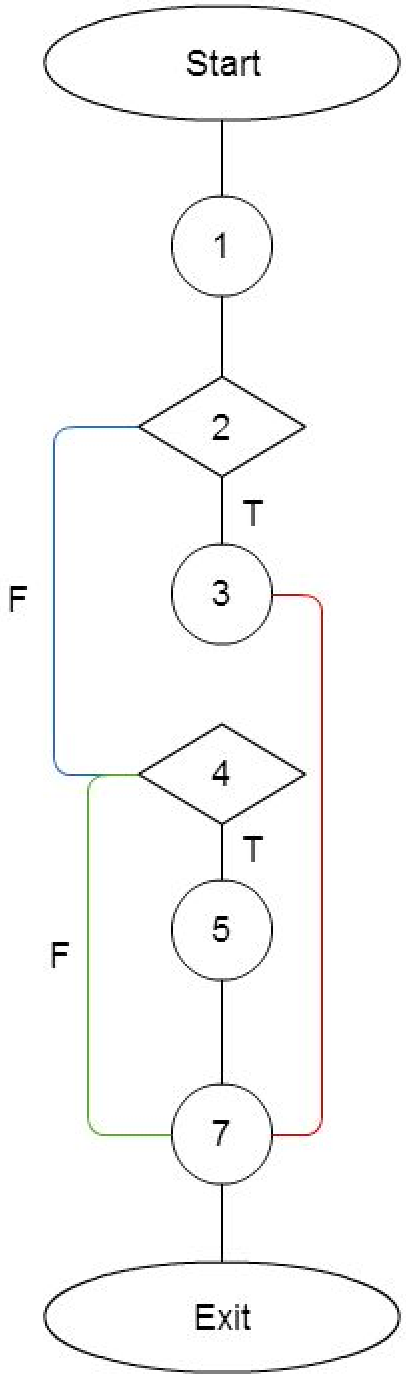


```
/*Check for correct input value types*/
public boolean isInputCorrect()
{
boolean correct = true;

        /*Check both fields are not blank*/
        if (exp_field.getText().isEmpty() || points_field.getText().isEmpty())
        {
                JOptionPane.showMessageDialog(null,"Missing field(s)");
                correct = false;
        }

        /*Check point field is int*/
        else if(!isNumeric(points_field.getText()))
        {
                JOptionPane.showMessageDialog(null,"Points is not an integer value");
                correct = false;
        }

return correct;
}
```

6 corresponds to the empty else statement of the if in statement 2.

3 – Flow Chart

Develop a flow chart for your method based on the blocks of code the method was broken into in the previous step



4 – Types of Coverage
Comment on the types of coverage needed to test the method

Since our methods contains selection statements and no loops, we will use branch coverage to test all the branches of our method to test each possible outcome.

Branch coverage: since the isInputCorrect() method contains selection statements, we will select input values that will cause each branch in the isInputCorrect() method to be executed once.
The method has two selection statements: one to check if the fields are blank, and the other to check if the points field is an integer.

In order to execute the branches of the missing fields statement, input should be blank in both or either fields then both filled. In order to execute the branches of the “not an integer value” statement, input should be a string or a character (eg. 12abc or abcde) and then an integer value (eg. 12345). This will achieve 100% branch coverage.

5 – List of Test Cases
Develop a list of test cases that will adequately test the method. Do not fully develop each test case. ONLY list the coverage of each test and explain why the test is needed/useful

Test case id – 1B coverage: Start-1-2-3-7-Exit
Inputs: Expectation field is empty

Test case id – 2B coverage: Start-1-2-3-7-Exit
Inputs: Points field is empty

Test case id – 3B coverage: Start-1-2-3-7-Exit
Inputs: Both fields are empty

Test case id – 4B coverage: Start-1-2-4-5-7-Exit

Inputs: Both fields are completed, points field is a string (contains characters eg. abcde)

Test case id – 5B coverage: Start-1-2-4-7-Exit
Inputs: Both fields are completed, points field is all integers. (eg. 10)

Test cases 1B,2B, and 3B ensure that no combination of fields are empty.
Test case 4B ensures that after both fields are complete, the Points field is not entered with a string.
Test case 5B ensures that after both fields are complete, the Points field contains only integers and the method is successful.

6 – One Test Case
Choose one of the test cases and describe it in detail in terms of the test case format used in class

Test id - 3B
Test Purpose: Unit test isInputCorrect() using white box testing strategy (branch coverage: Start - 1 -2 - 3 - 7- Exit)
Requirement: #1.4.2
Inputs: both exp_field and points_field are empty
Testing Procedures:

1) create two JTextField object - exp_field and points_field in the test driver
2) set the both text fields to empty by invoking exp_field.setText(“”) and points_field.setText(“”)
3) call isInputCorrect()

Evaluation: no step required
Expected behaviours and results: since both exp_field and points_field were set to empty, the method returned false
Actual behaviours and results: exp_field and point_field were empty, the method returned false as a result.

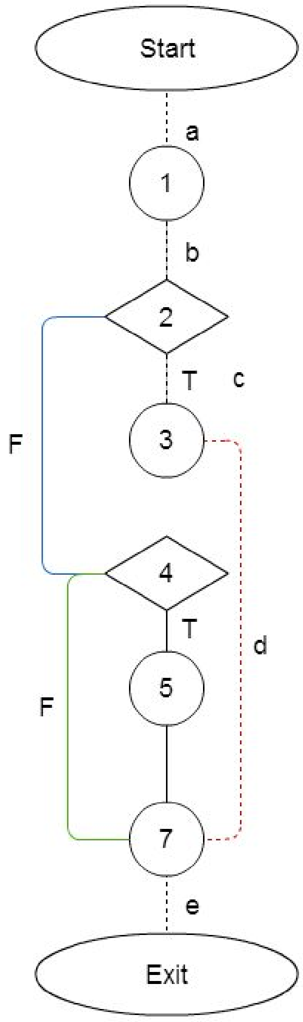
7 – Demonstrate Test Case

Demonstrate how your selected test case (input values) will help test the intended coverage.

The selected input values, exp_field = “” and points_field = “” are used to test the branch coverage. These inputs will help us test the intended coverage by testing our first if statement in block 2. Since both fields are blank the if condition should be true therefore testing one branch of our method.

The method isInputCorrect() is invoked, return value of false is expected.

Steps are as shown in the control flow chart below:



a) go to 1 and initialize local variables

b) go to 2 and execute 2 to check if any field is empty

c) since our selected input values for exp_field and points_field are set to empty, the if-statement returns true. The control flow then goes to 3 and executes the statements in 3 (the variable ‘correct’ is set to false)

d) since the if-statement in 2 is true, block 4, 5, and 6 are skipped, the flow then goes to 7 and executes it

e) the method returns false and exits

In this test case, the actual return value matches our expected return value.

<u>Black Box Testing</u>

1 – Purpose, Parameters, and Preconditions

Describe the purpose of the method, its parameters (arguments), and any preconditions necessary for successful execution of the method

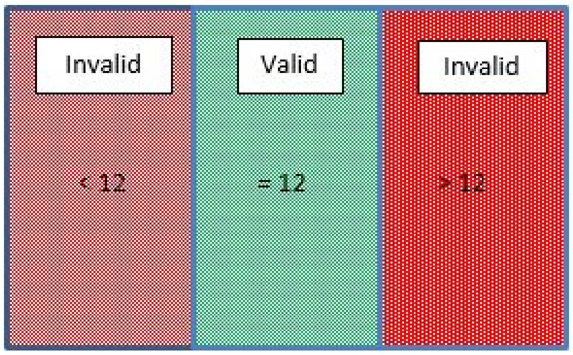
Method: testID(int id) - the method verifying if employee ID is a 12 digit number. It takes one parameter, an id, specified as integer.

Purpose: To determine whether the inserted employee ID is a valid 12 digit number. The method will use several representative input values to ensure that the employee ID fits all the requirements of containing only integers and having exactly a length of 12 digits, no more or less. If the employee ID fits the requirement of being a 12-digit number, the test case passes.
Parameters: “int id” will contain a 12-digit number.
Preconditions: The input of the parameter must be an integer in order to execute the method successfully.

2 – Equivalence Classes

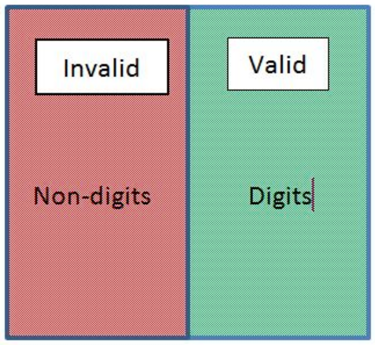
Describe the equivalence classes for each of the inputs and preconditions of this method

<u>Equivalence classes for employee ID</u>
Preconditions: input is an integer
There are 3 equivalence classes for range of values n:
- one valid equivalence class (program accepts n = 12)
- two invalid equivalence classes (n > 12 or n < 12)



Precondition: Input is an integer
There are two equivalence classes for the value of n:

- one valid equivalence class (program accepts integers - all digits)
- one invalid equivalence class (n is a string - non digits)



3 – Representative Input Values
Describe how you would choose representative input values to test this method

In order to test this method we would select input values that test for every possible case that a user may select.
We can have an n-digit number where n can be any length.
We know that in order to have a successful execution we need to enter a 12 digit number, so n=12 for our only valid input.
Other possible inputs would be a n-digit number where n<12 or n>12 but n != 12.
Therefore choose one value from each equivalence class for n.
Also choose the boundary values for each equivalence class.

Invalid	**Valid**	**Invalid**
<12	12	>12

e.g. choosing representative values n = 6, 12, 14
choosing boundary values n = 11, 12, 13
test values n = 6,11,12,13,14

We may also have an input that does not contain only digits. We may have an input that consists of characters (abcd), or a combination of digits and characters (a2b3c4).

Invalid	**Valid**
Non-digits	Digits

e.g. choosing representative values n = abcde, 12
test values n = abc123, abcd, 12

All Possible Inputs:
<12 digit number
12 digit number
>12 digit number
a string of characters of length n
a string of characters and numbers of length n

4 - Test Cases

Briefly describe how you would use the representative values to construct a set of test cases to test the method. DO NOT develop these test cases beyond describing them in terms of the values of the parameters and the preconditions.

First, test with all valid test cases and preconditions. After, test varying invalid test cases and invalid preconditions.
Precondition = Input is an integer.
N represents length of input.

Proposed Subset:
1. Variable within valid range, precondition true
(n = 12, all integers)
id = 123456789012, (int)id = true

2. Variable within valid range, precondition false
(n = 12, all characters)
id = abcdefghijkl, (int)id = false

3. Variable within valid range, precondition false
(n = 12, integers+characters)
id = 12345678901a, (int)id = false

4. Variable invalid range, precondition true
(n = 6, all integers)
id = 123456, (int)id = true

5. Variable invalid range, precondition false
(n = 6, all characters)
id = abcdef, (int)id = false

6. Variable invalid range, precondition false
(n = 6, integers +characters)
id = abc123, (int)id = false

7. Variable invalid range, precondition true
(n = 11, all integers)
id = 12345678901, (int)id = true

8. Variable invalid range, precondition false
(n = 11, all characters)
id = abcdefghijk, (int)id = false

9. Variable invalid range, precondition false
(n = 11, integers +characters)
id = abcdef12345, (int)id = false

10. Variable invalid range, precondition true
(n = 13, all integers)
id = 1234567890123, (int)id = true

11. Variable invalid range, precondition false
(n = 13, all characters)
id = abcdefghijklm, (int)id = false

12. Variable invalid range, precondition false
(n = 13, integers +characters)
id = abcdef1234567, (int)id = false

13. Variable invalid range, precondition true
(n = 14, all integers)
id = 12345678901234 , (int)id = true

14. Variable invalid range, precondition false
(n= 14, all characters)
id = abcdefabcefab, (int)id = false

15. Variable invalid range, precondition false
(n = 14, integers+characters)
id = abcdef12345678, (int)id = false

5 – Single Test Case
Select a single test case from those discussed in the previous item.
Test case chosen from above is subset #1 where i = 123456789012.

6 – Test Case Format
Describe your test case using the test case format seen in class.

Test case id – 1

Test purpose: Unit test UserAccount class method testID(int id) using black box testing strategy.

Requirement: #2.2.1.3

Inputs: i = 123456789012

Testing procedure:
- Create an object of UserAccount class type
- Prompt user to enter employee id
- setEmployeedId(id) invokes testId(i = 123456789012)

Evaluation: Show database

Expected behaviours and results: Id is a 12-digit number which fulfills both requirements of being the correct length and consisting of only integers. Method should validate the employeeID.

Actual behaviors and results: Id passes both test cases employeeId is valid and therefore validated for setEmployeedId(id) method when it invokes testid(id