CST8116 Assignment 2 (21S)

Software Development Process, Object Oriented program with decisions and loops.

# Instructions

The Software Development Process as presented by Cay Horstmann [1] will be used as the basis for this lab assignment.

1) Understand the problem

2) Develop and Describe an Algorithm

3) Test Algorithm with Simple Inputs

4) Translate the Algorithm into Java

5) Compile and Test Your Program

# Overview

* A potato chip company makes plain potato chips in two different bag sizes, Regular at 9.25 oz and Large at 15.75 oz
* For quality control, the company samples bags for weight as they are produced at the factory using a sensitive scale that can measure weights in ounces to 4 decimal places.
* The chip company wants software that will allow a worker to input the size of the bag and the actual weight measured. Once the data is entered the program will report the number of good and bad bags and also report the total number of bags entered so far. The program should repeat this process until the user no longer needs to enter measurements. If a bag weight is 0.01 oz different or more from the expected weight (either above weight, or below weight) then this is considered a bad bag.
* For this prototype program you are not required to validate invalid input e.g. text when a number is expected, or negative values, however users can make mistakes by entering an incorrect chip bag size, in that event report an error and ask them to enter a correct size (do not increase any counts if there is an incorrect chip bag size entered).
* A junior programmer started a program but could not complete it, a senior programmer made some corrections and then added //ToDo comments and has now passed the project to you to complete it.

# Task 1: Understand the problem

* Examine the starter code, read the ToDo comments, briefly outline in general how you will code these methods:

public boolean isBagCorrectWeight()

public static void main(String[] args)

# Task 2: Develop and Describe an Algorithm

* Document the classes you have been provided using UML Class diagrams.
  + Remember, static class members should be underlined within a UML Class Diagram
  + You are not permitted to change the structure of the provided classes.
* Create pseudocode and flow-charts for the two methods listed above.
  + Just the two methods, you are not required to pseucode or flowchart all of the classes and other methods.
  + You are not permitted to alter the other methods provided, aside from adding programmer comments.
  + Your main method must use an instance of class PotatoChipBag and class User.
  + You may use any decision structure (if, or switch) and either a while or do-while loop in solving this assignment (A for loop is not recommended for this project)
  + The Conditional Operator is not accepted in place of a decision structure for marks in this assignment.
* Place the UML Class Diagrams, psveudocode, and flow charts in your MS Word document.
* See lecture materials weeks 9 and 10 for guidance
* Also see the Exercise 06 (21S) companion files for examples as well.

# Task 3: Test Algorithm with Simple Inputs

* Use three test tables:

## Table 1: Test Plan for method main for the repetition structure logic (looping tests)

* Feel free to use as a starting point but add more tests.
* Each row should represent a separate run of the program
* Suggestions: is the check to keep looping case-insensitive? Does the program end with “no”, or other e.g. “carrot”?

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Description |
| 1  9.25  yes | Good Bag: 1  Bad Bag: 0  Total Bags: 1  Continue? (yes/no)  Enter Bag Size:  1 for regular size  2 for large size | Good Bag: 1  Bad Bag: 0  Total Bags: 1  Continue? (yes/no)  Enter Bag Size:  1 for regular size  2 for large size | Test passes, program continues |
|  |  |  |  |

## Table 2: Test Plan for method main for good chip bag, bad chip bag, total bags

* Check counts of good and bad chip bags, to check logic
* Record the reporting information
* You may use the table below as a start, but check more cases
* Suggested tests
  + 1 good bag
  + 1 bad bag
  + 1 good, and 1 bad bag
  + Invalid size entry
  + A mix of good and bad bag entries, with an invalid entry, then at least one more good or bad bag

|  |  |  |  |
| --- | --- | --- | --- |
| Input | Expected Output | Actual Output | Description |
| 1  9.25  No | Good Bag: 1  Bad Bag: 0  Total Bags: 1 | Good Bag: 1  Bad Bag: 0  Total Bags: 1 | Matches, one good bag, correct report, program exits |
| 1  9.25  Yes  2  15.75  No | Good Bag: 1  Bad Bag: 0  Total Bag: 1  Good Bag: 2  Bad Bag: 0  Total Bags: 2 | Good Bag: 1  Bad Bag: 0  Total Bag: 1  Good Bag: 2  Bad Bag: 0  Total Bags: 2 | Matches, two good bags, correct report, program exits |
| 0  No | Invalid bag size entered | Invalid bag size entered | Matches, invalid bag size entered, no reports, program exits |

## Table 3: Test Plan for method isBagCorrectWeight()

* Feel free to use as a starting point but add tests.
* Suggested tests:
  + In bounds for size Regular (3 separate tests): 9.24, 9.25, 9.26
  + In bounds for size Large (3 separate tests): 15.74, 15.75, 15.76
  + Out of bounds size Regular (2 separate tests): 9.2399, 9.2601
  + Out of bounds size Large (2 separate tests): 15.7399, 15.7601
* Optional for practice: When testing your Java program in Task 5 (below), reference Hybrid 02 from Week 02, and use the debugging tools in Eclipse to step through your method to verify that one or two of the tests in the table below are passing as expected.

|  |  |  |  |
| --- | --- | --- | --- |
| Field values | Expected return value | Actual return value | Description |
| size = 1  weight = 9.24 | true | true | Matches |
|  |  |  |  |

# Task 4: Translate the Algorithm into Java

* Follow the ToDo sections in the starter code
* Use your pseudocode and flowcharts to help work things out for the methods you have been asked to complete
* Type brief comments in the source code files for
  + Top of the file, and each class header, and each constructor and method header

# Task 5: Compile and Test Your Program

* Compile and run your program, take a screen shot to demonstrate some features:
  + In one program run: enter a regular bag, continue, enter a large bag, continue, enter an invalid bag size e.g. 0 or 42 (etc.), enter one more good bag, exit program.
* Use your Algorithm test plans and document testing your Java program, e.g. update the description column to document the program tests.
  + You are **not** required to take screen shots of all of the tests.

# Starter Code

* The .java files are provided.
* Files:

Assignment02.java

PotatoChipBag.java

User.java

# Grading (22 points)

* **If you do not demonstrate this Assignment in the lab period to your lab professor your score for the entire assignment may become zero.**

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Missing / Poor (0) | Below Expectations (1) | Meets Expectations (2) |
| Understand the problem | Missing or poorly done or missing demo. | Problem solution statement is partly correct, may not have examples of the calculations or logic required to solve the problem. | Problem solution statement is correct. Sample calculations or logic needed to solve the problem are provided. |
| Algorithm: UML class diagrams | Missing or poorly done or missing demo. | Class diagram(s) are not in correct format, properties and methods may not be assigned correctly to the classes and / or the diagrams do not follow the starter code provided. | Class diagram(s) are correct format, properties and methods are assigned to appropriate classes, based on the word problem and provided starter code. |
| Algorithm: pseudocode | Missing or poorly done or missing demo. | Not in correct format and / or steps are not in an order that produces correct results. Use of repetition structure(s) and / or selection structure(s) partly correct. | Correct format, steps are in order that produces correct results. Use of repetition structure(s) and / or selection structure(s) is correct. |
| Algorithm: flowchart | Missing or poorly done or missing demo. | Not in correct format and / or steps are not in an order that produces correct results. Use of repetition structure(s) and / or selection structure(s) partly correct. Flowchart logic may differ from pseudocode. | Correct format, steps are in order that produces correct results. Use of repetition structure(s) and / or selection structure(s) is correct. Flowchart logic closely matches pseudocode. |
| Test Plan: Algorithm | Missing or poorly done or missing demo. | Does not have correct table format as seen in lecture notes and lab exercises, and / or does not test program using suggested tests. | Has correct table format as seen in lecture and lab exercises, tests program using suggested tests. |
| Source Code: \*.java file(s) Comments and Conventions | Missing or poorly done or missing demo or is starter code with no modifications. | File comment header with student name is present. Class and / or class-member (constructors, methods) are missing comment headers. Loosely follows Java coding conventions for identifiers, indentation. | File comment header with student full name is present. Class and / or class-member (constructors, methods) have comment headers. Closely follows Java coding conventions for identifiers, indentation. |
| Source Code:  \*.java file(s) program structure and logic. | Missing or poorly done or missing demo or is starter code with no modifications. | Program may have small syntax mistakes or produces incorrect output. Use of if or switch, and / or while or do-while not fully correct (has syntax, runtime, and or logic errors) | Program has correct syntax and program logic that produces correct output. Use of if or switch, and / or while or do-while is correct (no syntax, runtime, or logic errors) |
| Source Code: Demo | Missing or poorly done. Student cannot correctly answer basic questions on their program code. | Student answers to basic questions on their program code are partly correct. | Student can correctly answer basic questions on their program code. |
| Running Program: Demo | Missing or poorly done. Student cannot demonstrate compilation and execution of their program in Eclipse. | Student can demonstrate compilation and execution of their program in Eclipse. Program may not work correctly with input values specified by lab professor. | Student can demonstrate compilation and execution of their program in Eclipse. Program does work correctly with input values specified by lab professor. |
| Test Plan: Program | Missing or poorly done or unchanged copy of algorithm test table, or missing demo. | Follows from algorithm test table but does not have updated descriptions for all of the tests. | Follows from algorithm test table, does have updated descriptions for all of the tests. |
| Submission | Missing or missing demo. | Student does not provide both MS Word and .java file(s) with their submission, and/or does not follow lab professor’s submission requirements. | Student does provide both the MS Word document and .java file(s) with their submission, and does follow lab professor’s submission requirements. |

# Submission Requirements

* Upload your MS Word document as well as your Java file(s) to the Brightspace submission area by the due date. (See Brightspace for due date).
* Follow your lab professor’s instructions regarding lab submissions for their lab section.

# Sample Run of Completed Program

* Note: The user input was formatted here with bold text, and yellow highlighting. This style of formatting is not required for the Java program. In an Eclipse console user inputs would be in a light-green color by default.

Enter bag size:

1 for regular size

2 for large size

**1**

Enter weight: **9.25**

Good bags: 1

Bad bags: 0

Total bags: 1

Program by Stanley Pieda

Continue Program? (yes/no)

**yes**

Enter bag size:

1 for regular size

2 for large size

**2**

Enter weight: **10.00**

Good bags: 1

Bad bags: 1

Total bags: 2

Program by Stanley Pieda

Continue Program? (yes/no)

**yes**

Enter bag size:

1 for regular size

2 for large size

**0**

invalid bag size entered

Program by Stanley Pieda

Continue Program? (yes/no)

**yes**

Enter bag size:

1 for regular size

2 for large size

**2**

Enter weight: **10.75**

Good bags: 1

Bad bags: 2

Total bags: 3

Program by Stanley Pieda

**no**

Program has shut down

# References

[1] Cay Horstmann. (2019). Big Java Early Objects. 7th Ed. Wiley.