CST8116 Lab Exercise 02 (22S)

# Instructions

* The five parts of the Software Development Process as presented by Cay Horstmann [1] will be used as the basis for this lab exercise.

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

# Part 1 Understand the Problem

## What is the problem to solve?

“A programming company you are working for is developing software for a grocery store and has designed simple UML class diagrams for a variety of store items. You have been tasked with creating a detailed UML class diagram for class Cabbage, as well as creating pseudocode and flowcharts for sample class Cabbage methods, then creating a working-class Cabbage as a Java program using a provided testing harness pseudocode.”

Note: Your chief learning resource for this lab exercise are the week 3 lecture materials.

Please see the Lab Exercise 02 (22W) Companion File for the Simple UML Class Diagram of class Cabbage, as well as the provided pseudocode, and flowchart for method main of class Exercise02.

There is one worker method that will need to perform a mathematical calculation. Show the mathematics needed to calculate the price of a single head of cabbage given the weight in kilograms and the costPerKilogram for cabbage.

Clarification: Your Cabbage class will have two separate worker methods ( calculatePrice(), and printReport() )

* Sample values are: weight of 2.95 kilograms and $2.25 per 1 kilogram with price calculated as $6.6375.

# Part 2 Develop and Describe an Algorithm

## UML Class Diagram (Detailed)

* Provide a detailed UML class diagram for class Cabbage, this includes a no-argument constructor, overloaded Constructor for weight and costPerKilogram as well as get and set methods for each of the class level variables, the worker methods along with access-modifiers (+/-), and data-types and method return data types.

## Pseudocode

* Provide pseucode for: methods getWeight(), setWeight(num weight), calculatePrice(), and printReport()

## Flowcharts

* Provide flowcharts for: methods getWeight(), setWeight(num weight), calculatePrice(), and printReport()

Notes:

For both the flowcharts and pseudocode provide a statement that tells the reviewer that the methods are part of class Cabbage and that they should refer to the UML Class Diagram to review the class-level-variables.

You are not obligated to pass in pseucode or flowcharts for all of the get/set methods, however you should consider creating the pseudocode for them as part of testing the design of your algorithms, especially for your two testing tables.

# Part 3 Test Algorithm with Simple Inputs

* Use a table to test your program design a sample test is provided here, however you should use weight of 2.95 kilograms and $2.25 per 1 kilogram. i.e. **Use different input values for your version of the test plan.**

|  |  |  |  |
| --- | --- | --- | --- |
| input | Expected output | Actual output | Description |
| 2.33  2.5 | cabbage.getWeight() is: 2.33  cabbage.getCostPerKilogram() is: 2.5  cabbage.calculatePrice() is: 5.825  cabbage.printReport() is:  Cabbage: weight 2.33, cost per kilogram 2.5  Testing overloaded constructor with anotherCabbage  Weight 3.5, cost per kilogram 2.50  annotherCabbage.printReport() is:  Cabbage: weight 3.5, cost per kilogram 2.5 | cabbage.getWeight() is: 2.33  cabbage.getCostPerKilogram() is: 2.5  cabbage.calculatePrice() is: 5.825  cabbage.printReport() is:  Cabbage: weight 2.33, cost per kilogram 2.5  Testing overloaded constructor with anotherCabbage  Weight 3.5, cost per kilogram 2.50  annotherCabbage.printReport() is:  Cabbage: weight 3.5, cost per kilogram 2.5 | Hand trace of method calls, matches expected output |

# Part 4 Translate the Algorithm into Java

* Create an Exercise02 project in Eclipse and create two classes, Exercise02 and Cabbage
* Follow your UML class diagram, and pseucode and flowchart(s) and write the code for class Cabbage
* Follow the provided pseucode and flowchart to write the body of method main, which will test your class Cabbage.
* Follow Java coding conventions as demonstrated in the lecture materials.
* Ensure that your program outputs your full name as see in ACSIS as part of the program output.
* Comment your source code: comment at the top of each file with your full name, class-header comment for each class you create, and constructor and method header comments for every constructor and method you create.

# Part 5 Compile and Test Your Program

* Compile and run your program.
* Be prepared to demonstrate your program in the lab period, in the week before the formal lab submission is due.
* Re-use the test plan from your algorithm testing but create a second table in your MS Word document first verify that your program works as expected using the same test values. Then also test with incorrect data with your program and document what happens. For example, entering text instead of a number or entering negative numbers for either weight or cost per kilogram.
* **If the program test plan is an unchanged copy of the algorithm test plan, you may not receive marks for the program test plan.**

# Lab Demonstration Notes

* Your lab professor will ask you to demonstrate your program in lab, typically the week before the formal submission
* Your lab professor may also ask you a brief question(s) on your code.

# Submission Requirements

* Submit your MS Word document and .java file(s) by the due date for this lab exercise.
* Follow your lab professor’s submission guidelines.

# Sample of Running Program

Enter weight: **2.33**

Enter cost per kilogram: **2.5**

cabbage.getWeight() is: 2.33

cabbage.getCostPerKilogram() is: 2.5

cabbage.calculatePrice() is: 5.825

cabbagge.printReport() is:

Cabbage: weight 2.33, cost per kilogram 2.5 *🡨 method call output from printReport()*

Testing overloaded constructor with anotherCabbage

weight 3.5, cost per kilogram 2.50 *🡨 plain-text-output eg. output "weight 3.5, cost per kilogram 2.50"*

anotherCabbage.printReport() is:

Cabbage: weight 3.5, cost per kilogram 2.5 *🡨 method call output from printReport()*

Program implemented by Stanley Pieda

In the program example above, the user inputs were formatted in this handout with black-font, bold-lettering, and a yellow-highlight. The default settings in the Eclipse IDE will colour user inputs a light-green color.

Additional indicators using 🡨 in yellow show what output came from which method, however the program will not output messages starting with 🡨.

# Additional Notes

The intent for method main is to demonstrate testing class Cabbage by (what your main method should demonstrate):

* Using the no-argument (no-parameter) constructor
* Setting data using set methods
* Getting data using get methods
* Outputting the results to verify that set and get methods work
* Then calling worker methods to ensure they work.
* Then using the overloaded constructor, and either get methods, or printReport() to verify that the overloaded constructor is working correctly.
* Any time you create a class, it is a good practice to test each get/set worker method and constructor to ensure they work correctly.
* Attempting to troubleshoot later on, after a class is incorporated into a larger project, becomes more complex.

# Microsoft Word Document Format

See the template example (from exercise 01), suggested headings below:

* Understand the Problem
* Detailed UML Class Diagram
* Pseudocode
* Flow Charts
* Algorithm Test Plan
* Program Test Plan

# Grading (8 Points)

|  |  |  |  |
| --- | --- | --- | --- |
| Criteria | Missing / Incorrect (0) | Below Expectations (0.5) | Meets Expectations (1) |
| Understand the Problem | Missing or incorrect. | Partly correct. | Briefly outlines the necessary steps, in order, as an overview. Has example of math needed to process input into output, if applicable to the problem statement. |
| Pseudocode(s) and Detailed UML Class Diagram | Missing or incorrect. | Partly correct. | Correct format, steps are in correct sequence and lead to correct outputs.  Detailed UML Class diagram is correct. |
| Flowchart(s) | Missing or incorrect. | Partly correct. | Correct format, correct shapes used, steps are in correct sequence, matching pseudocode and lead to correct outputs. |
| Test Plan for Pseudocode and Flowchart | Missing or incorrect. | Partly correct. | Has correct format as shown in the lab handout, has test values and expected and actual outputs. |
| Demo in lab period | Missing or student could not answer any questions correctly. | Student program may not compile or run correctly, student partly answers question(s) related to their program code, or answer(s) are partly correct. | Student program compiles, and runs correctly, student correctly answers question(s) related to their program code. |
| Source Code: \*.java file(s) Comments and Conventions | Missing or poorly done. | Missing a comment-header from one or more of class declaration and / or method main declaration. Code loosely follows Java coding conventions for identifiers, indentation. | File comment header with student full name is present. Class and method declarations have comment headers. Code closely follows Java coding conventions for identifiers, indentation. |
| Source Code:  \*.java file(s) program structure and logic. | Missing or poorly done or program does not follow from the pseudocode, and flowchart(s). | Program may have small syntax mistakes and will not compile, and / or produces incorrect output when run. Program loosely follows the student’s pseudocode and flowchart(s). | Program has correct syntax and program logic that produces correct output. Program closely follows the student’s pseudocode and flowchart(s). |
| Test Plan for Program | Missing or poorly done or is only an unchanged copy of the provided algorithm test plan. | May not have correct format, does not verify that the program outputs match expectations. | Has correct format as shown in the lab handout, verifies that the program outputs match, and documents variations in output including samples of invalid inputs. |

# Reminder / Primer: How to Console Input with Class Scanner

* See your lecture notes for more details and code samples.

Step 1: import Scanner from the java API at the top of your source code file

import java.util.Scanner;

Step 2: inside of method main create a Scanner object:

Scanner keyboard = new Scanner(System.in);

// keyboard is only a variable name, you could use input etc.

Step3: use method nextDouble() of class Scanner to read in user input with decimal places.

double weight;

weight = keyboard.nextDouble();

* To read in numbers without decimal places, use nextInt() of class Scanner
* To read in text data, use method nextLine() of class Scanner

Important Tip: if your program reads in numbers, and text, ensure you call method nextLine() immediately after calling nextDouble() or nextInt(). Your lecture professor will likely go into more detail on this.

e.g.

double weight;

weight = keyboard.nextDouble(); // will leave the line terminator character in input stream

keyboard.nextLine(); // clean up the input stream i.e. System.in, removing left over line terminator characters

# Recommended Resource

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html>