CSCI 2110 Data Structures and Algorithms
Assignment 2
Release Date: Sept 29th
Due Date: Oct 11th
23h55 (five minutes to midnight)

Generics and Unordered Lists

Introduction

This assignment will help you develop an understanding of multiple generics and the application of unordered lists. There are two exercises to complete.

Marking Scheme

Exercise 1 /25

- Working code, Outputs included, Efficient, Good basic comments included: 25/25
- No comments or poor commenting: subtract up to 3 points
- Unnecessarily inefficient: subtract up to 4 points
- No outputs and/or not all test cases covered: subtract up to 2 points
- Code not working: subtract up to 14 points depending upon how many classes and methods are incorrect.
- Assignment submission and test case text files are not well labeled and organized: **subtract up to 2 points.**

Exercise 2 /22

- Working code, Outputs included, Efficient, Good basic comments included: 22/22
- No comments or poor commenting: subtract up to 3 points
- Unnecessarily inefficient: subtract up to 4 points
- No outputs and/or not all test cases covered: subtract up to 2 points
- Code not working: subtract up to 11 points depending upon how many classes and methods are incorrect.
- Assignment submission and test case text files are not well labeled and organized: **subtract up to 2 points.**

Error checking: Unless otherwise specified, you may assume that the user enters the correct data types and the correct number of input entries, that is, you need not check for errors on input.

Submission: All submissions are through Brightspace. Log on dal.ca/brightspace using your Dal NetId.

What to submit:
Submit one ZIP file containing all source code (files with .java suffixes) and a text documents containing sample outputs. For each exercise you will minimally have to submit a demo Class with a main method to be tested and your test code, sample outputs.

Your final submission should include the following files: *StorageBin.java, StorageBinTester.java, Tool.java, Hammer.java, Saw.java,Leve.java, Exercise1out.txt, Node.java, LinkedList.java, GradeAnalyzer.java, GradeAnalyzerTester.java, Record.java, Grades.txt, Exercise2out.txt*

You MUST SUBMIT .java files that are readable by your TAs. If you submit files that are unreadable such as .class, you will lose points. Please additionally comment out package specifiers. Your code must compile. Failure to compile may result in a mark of 0.

Exercise 1

In this exercise you will use multiple generics to develop a StorageBin class. The StorageBin class will store a generic item and a generic unique identifier for that item. For example, the StorageBin may store hardware tools and a unique ID for each tool.

You are provided with the following starter code files: *StorageBinTester.java, StorageBin.java, Tool.java*

Create or complete the following classes:

The *StorageBin.java* class should have the following methods and functionality:

- Two generic ArrayLists to store generic items - Why not a HashMap?
- A constructor to set up the ArrayLists - Constructor to set up hashmap
- sizeofBin():int method which returns the number of items stored
- addItem(T item, U id):void method which adds an item and its unique ID to the arrays(Hash map).
    - This method prints an error if an item is added with an already existing ID
- removeItem(U id): T method remove the item with the given ID from the bin and returns it to the caller.
- displayItems(): void method prints the contents of the storage bin in the format Name and ID (see test cases below)

Create a *Tool.java* class with the following properties:

- String to store the name of the tool
- Tool(String name) constructor to set up the name of the tool
- getName():String method to return the name
- toString(): String method which returns the name of the tool.

Create a *Hammer.java, Saw.java, and Level.java* class by **extending** the Tool class. Each class should have the following properties:

- A String to store the function of the tool
- Hammer(String name): constructor to set the name by calling super()
- setFunction(String function):void method to set the function of the tool
    - Hammer function: “I hammer stuff”
    - Saw function: “I saw stuff”
    - Level function: “I make sure everything is even”
- toString():String override the parent’s toString() method to return the tool’s name + function

Create a *StorageBinTester.java* class with a main method which tests the functions of the StorageBin by adding tools to a bin, removing them from the bin, and adding them to a workbag implemented as an ArrayList:

- Create a StorageBin<Tool,Integer> object which holds tools and a unique ID assigned as an integer

- Create an ArrayList of Tools called workbelt, which will store tools which are removed from the storage bin.
- The main method should ask the user to enter an undetermined amount of tools to the storage bin in the format: ToolType ToolID
    - Saw, Hammer, or Level
    - Entering "quit" ends the list
- After entering the tools, print the items in storage in the following format:
    - Name Function ID
- Prompt the user to retrieve tools from the bin and add them to the workbelt
    - The user will enter the unique tool ID to search the storage bin
    - If the tool is in the bin, add it to the workbelt, otherwise print an error
    - Entering "no" stops adding tools to the workbelt.
- Print the contents of the storage bin and the workbelt
    - See sample output below

Input/output

Your program should accept input in the following format:

- Adding tools to the storage bin: ToolName ToolID
- Enter "quit" to stop adding tools to the storage bin
- Adding tools to the workbelt: ToolID
- Enter "no" to stop adding tools

Your program should provide output in the following format:

- See sample output below

Sample inputs and outputs:

```
Enter tools to be stored in the format ItemName ID
Enter quit to end the list of tools
Hammer 1
Saw 2
Level 3
Level 3
Error, ID already used
Level 4
Saw 5
quit
Items in storage:
Item:Hammer I hammer stuff ID: 1

Item:Saw I saw stuff ID: 2

Item:Level I make sure everything is even ID: 3

Item:Level I make sure everything is even ID: 4

Item:Saw I saw stuff ID: 5

Would you like to retrieve a tool and add it to the workbelt? yes/no
yes
Enter item ID
5
Adding tool to workbelt
Would you like to retrieve a tool and add it to the workbelt? yes/no
```

```
yes
Enter item ID
1
Adding tool to workbelt
Would you like to retrieve a tool and add it to the workbelt? yes/no
no
Okay, have a nice day!
Tools left in bin:Item:Saw I saw stuff ID: 2

Item:Level I make sure everything is even ID: 3

Item:Level I make sure everything is even ID: 4

Tools in workbelt:
Saw I saw stuff
Hammer I hammer stuff
```

<u>Student Generated Test Cases:</u>

Demonstrate the following functionality of your program:

- Create 6 tools of the type Hammer, Level, or Saw
- Show that your storage bin will only accept unique IDs
- Demonstrate removing tools from the storage bin and adding them to the workbelt
- Print the final contents of the storage bin and the workbelt

Save your output in a text filed named *Exercise1out.txt*

<u>What to submit:</u>
Submit the following java files in a **zipped folder (.zip only)*:*** *StorageBin.java, StorageBinTester.java, Tool.java, Hammer.java, Saw.java, Level.java, Exercise1out.txt*

<u>Exercise 2</u>

In this exercise, you will use unordered lists to store student records and generate a histogram from an input text file.

You are provided with the following starter code files: *List.java, Node.java,LinkedList.java,GradeAnalyzer.java,GradeAnalyzerTester.java.*

<u>Do not</u> alter the code of *List.java, Node.java, LinkedList.java*

Create or complete the following classes:

Create *Record.java* class to store basic student info:

- String to store the name of the student
- int to store the grade of the student
- Basic getters and setters

Complete the *GradeAnalyzer.java* class

- addRecord(Record r): void method to add a record to the list
- removeRecord(Record r): void to remove a record from the list
- clearAllRecords(): void clears the record list

- printAllRecords(): void prints the records (see format below)
- printHistogram():void prints a histogram of the grade data with interval size 9
    - ex) [0-10), [10-20),…
    - See example data below

Create a *GradeAnalyzerTester.java* class
- Sets up a new GradeAnalyzer
- Reads from the input file *Grades.txt*
    - Each letter in the file corresponds to a name, each int a grade.
- Prints the grade list
- Prints the histogram
    - See sample outputs below

Input/output

Your program should accept input in the following format:
- Prompt the user to enter the filename to read from
- Read the input text file *Grades.txt*

Your program should provide output in the following format:
- Print the contents of *Grades.txt*
- Print a histogram
- See sample output below

Sample inputs and outputs:

```
Enter the filename to read from: [your file path here]
Z1,49
Y1,93
X1,80
W1,81
V1,83
U1,81
T1,55
S1,62
R1,81
Q1,78
P1,73
O1,72
N1,73
M1,75
L1,67
K1,100
J1,44
I1,94
H1,72
G1,67
F1,67
E1,66
D1,65
C1,77
B1,67
A1,55
Z,100
Y,99
X,80
W,88
```

```
V,85
U,85
T,82
S,81
R,81
Q,78
P,92
O,91
N,90
M,89
L,88
K,66
J,67
I,88
H,90
G,56
F,55
E,15
D,11
C,10
B,5
A,1
_______________________________________________
0-10|**
10-20|***
20-30|
30-40|
40-50|**
50-60|****
60-70|*********
70-80|********
80-90|***************
90-100|*********
```

<u>Student Generated Test Cases:</u>

Demonstrate the following functionality of your program:

- Read in the given text file
- Print the grade list
- Print the histogram
- Save the output in a file *Exercise2out.txt*

<u>What to submit:</u>

Submit the following java files in a **zipped folder (.zip only)*:*** *Node.java, LinkedList.java, GradeAnalyzer.java, GradeAnalyzerTester.java, Record.java, Grades.txt, Exercise2out.txt*