CS: 114 OPPDA Spring, 2025

Project3 (100 points) Due 2/25/2025

Submit your solutions to canvas. Do not send the entire project. All that is needed are the files ending in .java. Each class will be defined in its own separate .java file. All driver code should be put in one .java file. For this project that file should be named `Project3_Driver.java`. Please make sure your name is included at the top of each .java file. Make one zip file that includes all the .java files and submit that one zip file to canvas.

You must use the P1-P3 format. Also, make sure your code conforms to the Javadoc tags. It should be easy to generate a Javadoc html folder from your code with good documentation for all your methods. This gives the interface to our black box abstraction concept.

The grading breakdown for this project is as follows:

15% Readability – Is the program easy to read and understand (indentation, documentation, good use of white space, good output format, user prompts)?

10% Java – Does the program make good use of the Java constructs (functions, control flow, etc.)?

25% Robustness - Does the program compile and run, and not crash or throw exceptions with expected input data?

50% Correctness – Does the program solve the intended problem and work on a variety of reasonable inputs?

Learning Objective: To see how a Java Collections `Set` can be applied to computer science such as membership, and duplicate removal.

.

For this assignment, please submit in a zip file the following:

1) `PersonP3.java`

2) `Project3_Driver.java` This file will contain three static void methods, `P1`, `P2`, and `P3`.

3) A UML class diagram for PersonP3 class

You DO NOT need to submit a javadoc file, but it should be easy to generate one from your `PersonP3.java` file.

<u>Problem 1</u>

Use the `JFileChooser` to open a file. Read in a sequence of integers numbers from a file into a Java List of your choice. Make use of the `HashSet` to create a new `List` such that all the duplicate numbers have been removed, and the remaining numbers are sorted in order. Once you read the numbers in from the file you should not need to do any looping to remove the duplicate numbers. A useful fact is you can construct a Collections framework data structure by passing in another Collections framework data structure. For example, you can construct a `Set` with a `List`, or you can construct a `List` with a `Set`.

```
List<Integer> sequence1 = new ArrayList<>();
.
.
Set<Integer> set = new HashSet<>(sequence1); // Create a
Set from a list
```

If you read in the file sequence1.txt and process it a `println` of your `Set` should look like this:

```
The selected file is sequence1.txt
[100, 200, 201, 301, 450, 500, 600, 700]
```

<u>Problem 2</u>

Use the `JFileChooser` file selector GUI to read two files. Each file contains a sequence of integer numbers. Using a `HashSet` to compute the union of the two sequences of numbers. Store the union in a `List` and display the `List` in sorted order in an output file called `unionOut.txt`. You should sort the `List` with your own sort algorithm/method**. DO NOT use** `Collections.sort`. Also, compute the intersection of the two number sequences and store the result in a List. Output the intersection list in sorted order to an output file called `xOut.txt`. Include both output files in the final zip file.

If you read in the files `sequence1.txt` and `sequence2.txt` and compute the union and intersection a `prinln` of your sorted `List`s should look like this:

```
The selected file is sequence1.txt
The selected file is sequence2.txt

set1 [100, 200, 201, 301, 450, 500, 600, 700]
set2 [100, 111, 112, 119, 120, 121, 200, 201, 301, 450, 500, 600, 900]

Union [100, 111, 112, 119, 120, 121, 200, 201, 301, 450, 500, 600, 700, 900]
```

```
Intersection [100, 200, 201, 301, 450, 500, 600]
```

Problem3

Use the JFileChooser file selector GUI to open a bank transaction file. Each record in the file will contain three fields. The fields are id, name, and transaction type. The id can be represented as an integer number. The name and transaction type can be represented using the Java data type String. For each record in the file, you are to create a PersonP3 object. You are to define the PersonP3 class such that it has all the necessary accessor and mutator methods. It should have one parametric constructor.

After you read a record and create a PersonP3 object you are to store the PersonP3 object into a Set data structure such as a HashSet. As you read a record first check if the newly created object already exists in your data structure. One of two outcomes can occur from this check for membership. They are:

1. The newly read PersonP3 object exists in your data structure. In this case find the already existing object (reference variable) and update that object's transaction list.

2. The newly read PersonP3 object does not exist in your Set data structure. In this case add the first transaction to the Person3 transaction list. Then store the newly created object (reference variable) in your Set data structure. The transaction list is a data member in the PersonP3 class that holds a list of all the transactions as List of String objects. Possible transactions are the Strings: DEP, WIT, and BAL.

After your program processes all the records in the transaction file display each bank customer along with the number of transactions for that customer, and the list of transactions.

If you read and process the file Bank1.txt your programs output should look this:

```
The selected file is Bank1.txt

Reading the bank transaction file
100 Jim DEP
200 Ron WIT
300 Harry BAL
100 Jim WIT
100 Jim VIS
200 Ron DEP
300 Harry DEP
```

```
400 Kim DEP
400 Kim DEP
400 Kim VIS
100 JIM WIT
200 RON WIT

Display of customer transaction log
id = 400 name = Kim
There were 3 transactions.
DEP
DEP
VIS
id = 100 name = Jim
There were 4 transactions.
DEP
WIT
VIS
WIT
id = 200 name = Ron
There were 3 transactions.
WIT
DEP
WIT
id = 300 name = Harry
There were 2 transactions.
BAL
DEP
```

The iteration order of the transaction log above may vary with each run if you are using a `HashSet`. It should iterate through in the order that the customer's were entered int the `Set` if the `Set` you test with is a `LinkedHashSet`. The natural ordering for a `List` of customers is alphabetically by `name`. Two customers objects are considered equal if their id's are equal. If the `Set` is a `HashTree` the iteration order should be alphabetically by `name`. Be sure you test your code with all three types of `Set` types.