Regis University CC&IS
CS310 Data Structures
Programming Assignment 4: Stacks and Queues

Problem Scenario

The charity likes to thank their donors by having a dinner for them once a year. Unfortunately, the charity can only afford a very small venue that has five tables.

When a donor enters the venue, they are placed at the first available table. Once all of the tables have been occupied, donors are placed into a line to wait for the next table. Since the charity really wants to keep their GoldStar donors happy, a separate line is created for the GoldStar donors. When a table becomes available, the GoldStar line is checked first to see if any donor is waiting. If there are GoldStar donors waiting, that donor gets the next available table. If there are not any GoldStar donors waiting, then the other line is checked.

Program Requirements

You will still be using the CS310<lastname>.java file to read an **`assn3input.txt`** input data file (formatted the same as for previous assignments), as you will need to access the donors, and determine which of them are GoldStar donors.

The current CS310<lastname>.java file will remain as it is – you will be appending to it. And you will be building several new implementations.

A second input data file will be used to simulate donors arriving and leaving. The filename is to be **`diners.txt`** and will be located in the input folder. Each line of data will be:

```
ARRIVE donorID
```

or

```
DEPART donorID
```

The tables will be stored in a stack. When a donor arrives, the next available table is provided (by popping a table off the stack). For this assignment, you can presume the donor and entire party can be seated at one table.

You will create your own **TableStackImpl** using a 5-element array implementation. The data can be a simple integer 1 through 5 to represent the table number. The **TableStackImpl** should be able to **push** and **pop** tables from the stack. You also need to include method to determine if the stack is **empty** or **full** – and use these methods.

As donors are assigned to a table, you will need to be able to specify which donor is sitting at what table. Choose an appropriate data structure that we have studied to provide this information. This should be handled in a **SeatingImpl** class. As donors come and go, you will need to keep the seating information up to date.

You will create your own **DonorQueueImpl** using a singly linked list, with references to both the front and back nodes in the queue (you should be able to use your DonorNode class from last week to build this linked list). This implementation will be store Donor nodes, and will be used to create two separate queues (representing lines waiting for tables) – one for standard donors, and one for GoldStar donors.

If a donor arrives to the dinner, but there are not any tables available, the donor will be assigned to a queue to wait for the next available table. If the donor is a GoldStar donor, that donor is assigned to the GoldStar queue. If the donor is not, then the donor is assigned to standard donor queue.

Newly arriving donors will be added to the rear of the linked list. When a donor is seated from the queue, that donor will be taken from the front of the linked list.

The code should be able to **add** and **remove** donors from each of the queues. You will also need to build methods to check if a queue is **empty** or **full**, and use the methods. If you try to remove a donor from an empty queue, you will need to provide an error message. A queue will only be considered full if memory allocation fails when you try to instantiate a new node object.

You will also need to watch for gate crashers. If a donor attempts to get a table, but his/her donor ID is not in your Donor linked list (built using the code from Assn 3), then that donor will be ignored, and your application will move on to the next donor.

As each action occurs, you will provide an audit trail. At the end of the program, your application will provide a report providing the existing seating chart (which donor is at which table), which tables are available in the stack (the top of the stack is to be listed first), and which donors are still sitting in each of the queues in order.

All output is to be placed into a **`dinnerReport.txt`** file to be located in the **output** folder.

A sample audit trail will look like the following.

```
George Washington was seated at table 1.
Ben Franklin was seated at table 2.
Donor ID 12345 tried to crash the dinner but was turned away.
George Washington has left Table 1.
Thomas Jefferson was seated at Table 1.
Abraham Lincoln has entered the GoldStar queue.
Thomas Jefferson has left Table 1.
Abraham Lincoln was seated at Table 1.
```

The final reports should be in the following format:

```
SEATING CHART
John Lennon is at table 3.
Ringo Starr is at table 1.

STACK
Table 2 is available
Table 4 is available
Table 5 is available

GOLDSTAR QUEUE
George Harrison is waiting
Paul McCartney is waiting

STANDARD QUEUE
Pete Best is waiting
Brian Epstein is waiting
```

Deliverables

- Your input data files will still be read from the **input** folder in your project.

  Place all test data files that you create to test your program in the **input** folder of your project, and name them as follows:
  `assn4input1.txt`
  `assn4input2.txt`
  (i.e. number each data file after the filename of **`assn4input.txt`**)
  and
  `diners1a.txt`
  `diners1a.txt`
  `diners2a.txt`
  (i.e. number each data file to correspond to the **`assn4input.txt`** file used, and add letters for multiple test of the same **`assn4input.txt`** input file)

  Together, all of your test data files should demonstrate that you have tested every possible execution path within your code, including erroneous data which causes errors or exceptions.

- Your output data files will still be written to the **output** folder in your project.
- Create and/or modify **Javadoc headers**, and generate **Javadoc files**
- Add screen shots of **clean compile** of your classes to the documentation folder.

WARNING: Submittals without the clean compile screenshots will not be accepted.

Program Submission

This programming assignment is due by midnight of the date listed on the **Course Assignments by Week** page of the Content.

- Export your project from NetBeans using the same method as you did for weeks 1 – 3.
  - Name your export file in the following format:
    `CS310<lastname>Assn<x>.zip`
    For example: **`CS310SmithAssn4.zip`**

    NOTE: Save this zip file to some other directory, not your project directory.
- Submit your **.zip** file to the **Prog Assn 4** dropbox (located under the **Dropbox** tab in the online course).

  Warning: Only NetBeans export files will be accepted.
  Do not use any other kind of archive or zip utility.

Grading

Your program will be graded using the **rubric** that is linked under **Student Resources** page.

WARNING:
Programs submitted more than 5 days past the due date will ***not*** *be accepted,*
and will receive a grade of 0.