P08 Badger Coaster

Overview

The executives of Theme Parks Inc. have decided to build a new theme park in the Madison area featuring some Bucky Badger themed rides . Hearing of your awesome programming skills, they have decided to come to you for assistance. You are tasked with helping the creation of an application for the park that uses boarding passes and a queue to manage ride lines as well as provide some other basic functionalities.

Learning Objectives

The goals of this assignment includes implementing a variation on a traditional queue as well as re-enforcing previous course topics.

Grading Rubric

5 points	**Pre-Assignment Quiz:** The P8 pre-assignment quiz is accessible through Canvas before having access to this specification by 9:59PM on Sunday 4/5/2020. Access to the pre-assignment quiz will be unavailable passing its deadline.
20 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is otherwise correct. To become more confident of this, you should run additional tests of your own.
20 points	**Manual Grading and Supplemental Automated Tests:** When your final grading feedback appears on Gradescope, it will include the feedback from these additional automated grading tests, as well as feedback from human graders who review the code in your submission by hand.

1 Getting Started

Start by creating a new Java Project in Eclipse, giving it a relevant name. Like previous assignments, ensure that the project uses Java 11.

2 BoardingGroup Class

Create a new java class called `BoardingGroup`. This is an instantiable class that will be used to represent groups entering the ride line. Each boarding group object will have a name and a number of people in the group. You will need to write a basic constructor for the object as well as accessors for all of the class' fields. The names of these methods should follow standard naming conventions and follow in accordance with the CS300 Course Style Guide. Once you have finished implementing this class, you may move onto the next step.

3 Ride Queue: Set-Up

Now, it is time to start creating the queue for the boarding groups. You will be using a linked list implementation to create the queue. Add the following class to your Java project that will be used as the linked list nodes of the queue: BGNode.java. Create a new java class called `RideQueue` for the queue. This will be a queue of boarding groups and must implement the following interface: QueueADT.java.

3.1 Fields & Constructor

Your `RideQueue` class should have five private fields with names and data types as follows:

- ***BGNode*:** front, back
- ***int*:** capacity, numOfPeople, numOfGroups

Note that here, the capacity field is the max number of people that can fit into the queue. After you have added the fields to your class, implement the constructor as described in this javadoc.

3.2 Other Preliminary Methods

There are two other simple methods that should be implemented at this time as they may be helpful in the implementation of other methods. Implement the `size()` and `isEmpty()` methods as described in this javadoc.

Then you can add the following `toString()` to your `RideQueue` class. You may find that it proves helpful when debugging your code. Note that there is one section of code you must fill in for yourself.

```java
// Returns a string representation of this RideQueue.
 public String toString() {
    String s = "Number of People in Queue: " + numOfPeople + "\n";
    s += "Number of Groups in Queue: " + numOfGroups + "\n";
    s += "Group Names in Queue: ";

    BGNode current = front;
    while (current != null) {

      String groupName = /*CALL YOUR ACCESSOR FOR GROUP FOLLOWED BY
      YOUR ACCESSOR FOR NAME HERE ON current*/
      s += groupName + " ";
      current = current.getNext();
    }

    return s;
  }
```

4 Entering the Queue: Enqueue

Now it is time to be able to add groups to the ride's queue. Since the queue has a max capacity of people it can hold, sometimes certain groups may not be able to enter the queue. To do this, implement the `enqueue()` method as described in this javadoc.

Once you are confident you have it implemented correctly, add the following driver class and text file to your project: ThemeParkApp.java, sample.txt. This driver reads various commands from the text file. Important details about these commands and the driver can be found in the **Driver Details & Commands** section towards the end of this write-up. Right now jump to that section, reading and understanding it thoroughly before coming back here. Once you've done that, only the "Enter" and "Status" commands should work correctly as the others are yet to be implemented.

5 Ride Breakdowns: Clear

Like every piece of machinery, sometimes the ride will break down. When this happens, the theme park's policy is to force everyone out of the ride queue. To do this, implement the `clear()` method as described in this javadoc. At this time the "Breakdown" command should work correctly in addition to those previously supported.

6 Boarding the Ride: Peek & Dequeue

Sometimes your fellow app developers may want to see who is at the front of the queue. To do this, implement the `peek()` method as described in this javadoc.

In addition, groups will exit the queue when they board the ride. To do this, implement the `dequeue()` method as described in this javadoc. At this time all of the driver's commands should work correctly.

7 Support VIP Groups

The owners of the theme park have decided to enact a policy that allows people to pay additional money to become a VIP group. VIP groups are different from normal groups in that they automatically are put at the front of the queue. They are put at the front regardless of whether or not the group at the front is a VIP group. (Yes, this is a unfair park policy.) Just like normal groups, they should not be allowed to enter the queue if it would exceed capacity.

7.1 Modify the BoardingGroup Class

Add a field to the `BoardingGroup` class that marks whether or not the group is a VIP group. When a new BoardingGroup object made, the group will be a normal one by default. You should also add an accessor and a mutator method for the field to the class. The mutator should make a BoardingGroup object a VIP group. The names of these methods should follow standard naming conventions and follow in accordance with the CS300 Course Style Guide.

7.2 Modify RideQueue.enqueue()

Now return to your `RideQueue` class and modify it according to the new rules described at the begining of this section.

TIP: You may want to make/save a copy of your working `enqueue()` from section 4. That way you still have it in case you break it while attempting this part.

8 Assignment Submission

Once you have completed the project (which includes testing your code thoroughly) submit only the `BoardingGroup.java`, `RideQueue.java`, and `ThemeParkApp.java` source files to Gradescope on or before the deadline. Remember to give your files and classes headers, methods proper Javadoc headers and comments and any other specifics that are given by the CS300 Course Style Guide. Your score for this assignment will be based on your **"active"** submission made prior to the hard deadline of **11:59PM on April 8**th.

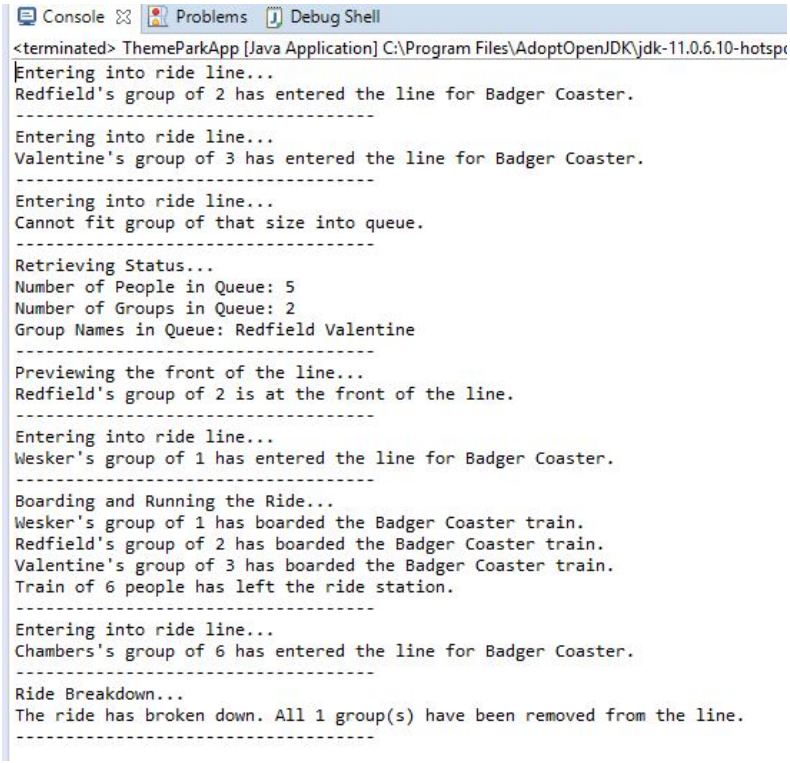
Driver Details & Commands

The driver parses the commands from a text file by the title of "sample.txt". The file must be in the directory of your project folder. By default the queue capacity is 50 and the ride capacity is 24. You are welcome to change these values if desired. Note: There are some comments (seven in total) which denote places where you need to call different methods in your `BoardingGroup` class. Fill these in so the drive compiles and run.

Here are the commands that the `ThemeParkApp` class supports:

- Enter → E <name> <numOfPeople>
- Status → S
- Breakdown → B
- Preview → P
- Ride → R

Commands are case insensitive and have only one space between them. The "Enter" command can have an additional argument of "V" that will denote a VIP group. You can use the driver as a tool to test your queue method implementations by editing the commands in "sample.txt". The provided sample.txt yields the output as shown below.



Additional Notes

- **java.util.NoSuchElementException** is the only import statement needed in `RideQueue.java`. There should be no other import statements in your submitted files.
- You **are not** allowed to add any additional fields to the classes you write besides those in this write-up.
- You **are** allowed to use local variables and private helper methods.
- You are not required to write a tester class for this assignment. However, testing your code is still highly encouraged. A alternative method for testing has been provided via the driver.
- You can ignore any instructions concerning VIP groups until you have reached Section 7.
- The "Ride" command will keep removing groups from the queue until either
  1. The queue is empty.
  2. The next group in the queue cannot fit onto the ride. **Ex.** The ride holds 12 people. Currently there are 11 on it but the next group has 2 people.