INFO 1531 – Java II Programming – Assignment 6 – Collections

Assignment

Overview

The Happy Accidents paint company's invoice system is very old and archaic. They have suffered from lost jobs because their organization and getting paint jobs done in a timely manner. We are tasked with writing and invoice system that reads in their invoices from a text file, allows them to add new invoices in and assign them to one of their three painting teams to complete.

Some of the coding has been done by myself but I need your help to put the last finishing touches on it. Mainly the code to keep track of the invoices in a custom queue class, and finishing some methods to manage the invoices in the different queues created.

Directions

To Get Started:

1. Create a new java project in IntelliJ(or your preferred IDE) called YOUR_MCC_Name-Assignment6.

2. Download and extract the files neede for the assignment: **1531Assignment6Starters.zip**

-
    - Place the java files in your 'src' folder
    - Place the PaintJobInvoices.txt file in your project folder

3. Quick breakdown of the Java files

- Invoice is the object for each invoice
- PaintFinish is an enum type defining the finish of a paint - flat, eggshell, satin, semi-gloss, and gloss
- PaintJobDB is the database for reading in the files and holding a queue of the invoice. You will be adding code in this file.
- PaintJobSystem is the runner file that provides a menu to view, add and complete jobs. You will be doing a lot of coding in this file as well.

- PaintJobInvoices.txt contains the text data separated by tabs (\t) per each property. This is already setup.

BIG NOTE: When you add these files to your project you may notice some errors pop up. Specifically with a missing Queue and no implementation of the Comparable interface with the Invoice class. That is because part of the assignment is that you will be creating your own and setting up the compareTo(). Also the IDE may try to fix these problems by importing the Java Queue, but again you will have errors with that. The Java Queue in the Collections frame work is an interface and doesn't allow you to create an actual queue. If you see these errors you will need to delete the import java.util.Queue; *at the top of your file. This will then allow you to use your own custom queue.*

Part 1: Implement a custom Queue object

The Queue in the Java Collections frame work is limited AND it has a lot of functionality that we don't need. So instead we are going to create our own Queue class that we can implement basic functionality that we need. Create a new Java class called Queue.java.

- The Queue should be defined as a generic in the public class header.
- Define a private instance variable for a LinkedList object that can store the elements of the queue.
- Implement the following methods:
  - enqueue(E element) - adds an element to the linked list
  - dequeue() - returns the element at the top of the queue
  - peek() - returns the element at the top of the queue but does not remove it.
  - size() - returns the number of elements in the queue
  - isEmpty() - returns true if no items are in the list, false otherwise
  - addAll(List<E> list) - takes a generic list and adds all the elements into the queue

Once this is complete the compiler errors for the queue should be fixed. BUT double check your System and DB file to make sure that the import java.util.Queue; is not at the top.

Reference p.521 in your text book on custom collections.

Part 2: Sorting the Invoices via dates

Another error that will still be showing is that there is not Comparable/compareTo() method setup on the Invoice class file. That is something you need to fix and code. Implement the use of the Comparable interface and add the compareTo() method to the Invoice class.

The compareTo() method will take a little work here. We are going to compare via the date of the invoice. The dates of the Invoice class are by default saved in a MM-DD-YYYY format. So they have a dash '-' between each part of the date. So you will need to split the date of the current invoice AND split the date of the object sent to the method.

We have three things to compare against here. First we need to check the year. If the years are the same then you should go another step forward and compare the months. If the months are the same then you will lastly have to compare the day.

Use whatever method you want. Remember that these are in String format, so you could try the compareTo() on String but what happens if the date is 09 for month, does it compare that correctly. Otherwise think about parsing the data to ints.

Part 3: Finish the PaintJobSystem.java file

Complete the viewNextJob() method.

This takes on the database object and should use the queue's peek() method to see what the top job is and print it out. BUT it shouldn't print 'null' if there are no jobs in the queue. So you will need to check if it is empty or not. If it is empty you should print a message stating that.

```
*** Next Job ***
Invoice ID: 1589
Invoice Date: 01-25-2023
Customer Name: Bruce Banner
Address: 890 5th Ave, Manhattan, New York 10451
Paint: Flat Dutch Boy
Color: Calming Gray
Total Cost: $1,026.75
```

or if no job

```
*** Next Job ***
*No Jobs in Queue*
```

Note: The job in this example may not be your job YET, as you haven't sorted the data.

Complete the viewTeamJobs() method

The viewTeamJobs() takes on the three queues created(one for each team) to assign jobs. When this method runs it will use the queue's peek() method and print out what the current job for each team is. If a team doesn't have a job it shouldn't print out 'null' but instead '*None*'. You should also print a header like *** Team 1 *** before each team's job get's printed. This way we know what each team is working on.

```
*** Team 1 ***
Invoice ID: 1589
Invoice Date: 01-25-2023
Customer Name: Bruce Banner
Address: 890 5th Ave, Manhattan, New York 10451
Paint: Flat Dutch Boy
Color: Calming Gray
Total Cost: $1,026.75

*** Team 2 ***
Invoice ID: 1234
Invoice Date: 02-15-2023
Customer Name: Tony Stark
Address: 10880 Malibu Point, Malibu, CA 90265
Paint: Semi-Gloss Sherwin Williams
Color: Hot Rod Red
Total Cost: $2,631.58

*** Team 3 ***
*NONE*
```

Complete the assignJob() method

Some of this method is already coded including a basic menu, input and the start of the switch. What you need to do is finish the switch statement. Set up cases for each 1, 2 and 3 corresponding to the teams. This method is to take the next job (dequeue) from the database's invoice. We don't have controls to not show a team if they already have a job SO you need to check for that. If someone tries to assign a job to a team that already has a current job then you should display a message stating you can't assign a job. There are some helping methods on the PaintJobDB class that you can use like assignNextJob() that does dequeue and returns the next Invoice object.

```
*** Assign Job ***
1. Team 1
2. Team 2
3. Team 3
Choose: 1

*** Happy Accidents Paint Job Invoices ***
1. View Next Job
```

```
2. See Team's Current Jobs
3. Assign Job to Team
4. Complete Job
5. Add New Job
6. Exit
Operation: 3

*** Assign Job ***
1. Team 1
2. Team 2
3. Team 3
Choose: 1

Team 1 already has a job, please mark complete before assigning a new one.
```

Complete the addNewJob() method

This method gets inputs for each property of an invoice

- id
- date
- lastName
- firstName
- address
- paint
- paint finish
- color
- cost

You need to use the scanner sent in the parameter(sc) to get inputs for each, create a new Invoice and add it into the the invoice queue through the db object.

When getting input you will need to setup some data validation/conversion from String paint finish to the enum PaintFinish data types. I would suggest having a default value so that if someone doesn't type something correctly you get some value to not throw an exception. Double check any other validation as required...maybe the date to make sure there are three values separated by a dash(-), etc. That is up to you.

```
*** Happy Accidents Paint Job Invoices ***
1. View Next Job
2. See Team's Current Jobs
3. Assign Job to Team
4. Complete Job
5. Add New Job
6. Exit
Operation: 5

Enter ID: 78569
Enter date: 03-21-2023
Enter name: Scott Lang
Enter address: 601 Buena Vista Avenue West, Buena Vista, CA 95640
Enter paint: Dutch Boy
Enter finish: Flat
Enter color: Dirt Brown
Enter cost: 635.26
```

Note: Again this is my example, on getting input you can do it how you want.

Complete the exit() method

So the exit method is handling our data saving. It is going to call upon the file write out methods that we have on the PaintJobDB object. BUT before we do that, we don't want to lose current jobs that a team might be working on. So before we do a whole write out we should dequeue each team1, team2 and team3 queues back to the invoice queue on the db object. THEN we can call the writeOutInvoices() and writeOutCompleted() methods.

```
*** Happy Accidents Paint Job Invoices ***
1. View Next Job
2. See Team's Current Jobs
3. Assign Job to Team
4. Complete Job
5. Add New Job
6. Exit
Operation: 6

Exiting...Invoices Saved Out.
```

Part 4: Sorting the complete invoices via name

One last thing, as far as completed invoices go. For ease of lookup in the future the paint company wants the completed invoices to be sorted via the customer's name. But we do have a problem as we have already setup the compareTo() to use the invoice data. So we will to set up a customer comparator to send in our Collection.sort() call.

Set up a lambda expression to compare via last name AND the first name (if the last names are the same).

You will find the Collections.sort() method call on the PaintJobDB line 202(ish) that needs to be coded.

Note that the items are stored in a stack and may be ordered opposite of what you want. Play with how you compare them and/or use another collection object to sort and put back into the completeInvoice stack.

Example Output(user input bolded):

Reference the following file for output: **1531-Assignment6-ExampleOutput**

Submission

1. Compress the IntelliJ project folder and submit it to this assignment.

   to compress: On Windows, right click -> send to -> compressed .zip file

   On Mac, right-click -> Compress

Hints/Tips (Before Submitting):

- *You can use your own verbiage for the error messages/output on the PaintJobSystem.java file, OR you can see what I have and code it exactly.*
- *Remember my notes on the Queue that the IDE likes to auto import sometimes for us. Delete those import java.util.Queue; lines at the top if they appear.*
- *It may seem like a lot of work but I have hopefully broke this down in order of things to do so that you can get rid of compiler errors and do each part of a time.*
- *Don't forget to have a header at the top of your file and include a* ***Resource statement****.*
- *Use comments throughout for full points.*
- *Follow all Java Styling Guides as covered*