P10 - Bank Transaction Manager

Overview

Banks need to efficiently handle numerous transactions every day, such as processing withdrawals, deposits, and loan applications. Many of these tasks involve prioritizing certain transactions over others. For example:
- A high-value loan application may need immediate attention.
- A request from a customer with a large balance may need to be given high priority.

In this assignment, you will simulate a bank transaction manager by implementing a priority queue using a heap-based data structure. Your program will handle tasks based on their priority, ensuring the most important transactions are addressed first.

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **12/03**.
+5%	**Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **Friday 12/06** *and* who pass ALL immediate tests will receive an additional 2.5 points toward this assignment, **up to a maximum total of 50 points**.
30 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
15 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
0 points	**Manual Grading Feedback:** There will not be any manual grading for this assignment.
50 points	**MAXIMUM TOTAL SCORE**

Learning Objectives

After completing this assignment, you should be able to:

- **Understand** the functionality of heaps and priority queues.
- **Implement** a heap-based priority queue in Java.
- **Simulate** real-world bank transaction management using heap operations.
- Continue to **practice** developing and writing tester methods.
- **Develop** and implement a nontrivial comparison method for comparison of objects along different axes depending on inputs, including a comparison based on values of an enum.

Additional Assignment Requirements and Notes

Keep in mind:

- Pair programming is **NOT ALLOWED** for this assignment. You must complete and submit P10 individually.

- The ONLY external libraries you may use in your program are:

  java.util.NoSuchElementException

- Use of *any* other packages (outside of java.lang) is NOT permitted.

- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables or constants beyond those specified in the write-up.

- You are allowed to define additional **private** helper methods.

- Only **BankManagerTester** may contain a main method.

- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- All other sources must be cited explicitly in your program comments, in accordance with the Appropriate Academic Conduct guidelines.

- Any use of ChatGPT or other large language models ***must be cited*** AND your submission MUST include **screenshots** of your interactions with the tool ***clearly showing all prompts and***

responses in full. Failure to cite or include your logs is considered academic misconduct and will be handled accordingly.

- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

Need More Help?

Check out the resources available to CS 300 students here:
https://canvas.wisc.edu/courses/427315/pages/resources

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Appropriate Academic Conduct, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - How do I cite my sources?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P10 Bank Transaction Manager**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.
2. Download five (5) Java source file(s) from the assignment page on Canvas:
   a. **TransactionHeap.java** (does NOT include a main method)
   b. **BankManager.java** (does NOT include a main method)
   c. **Account.java** (does NOT include a main method)
   d. **Tester.java** (includes a main method)
   e. **Transaction.java** (does NOT include a main method)

1. Account

We provide the file **Account.java** to help you out in implementing your code. It stores some basic information about accounts including the account number and their balance.

2. Heap Implementation

We provide the **TransactionHeap.java** skeleton to help you out in implementing your code. It is related to max-heap implementation. It uses an array-based structure to represent the binary heap. The main heap operations include:

- insert: Adds a new transaction while maintaining the heap property.

- remove: Removes the highest-priority transaction while maintaining the heap property.

- heapify: Adjusts the heap to maintain its property after insertions or removals.

3. Transaction class

In the file **Transaction.java**, you need the following fields:

- **type**: “Loan Application”, “Withdrawal Request” and “Deposit”.

- **priority**: such as “Low”, “Normal”, “High” and “Urgent”.

- **amount**: the amount of money to be processed.

- **user**: the Account object associated with this transaction

We also provide certain priority rules based the type of transaction:

- **Loan application**: Banks **prioritize processing loan applications** over other transactions, unless the difference between your loan amount and your balance amount is too great. We assume that up to **3 times is acceptable**. Once this amount is exceeded, the loan application is considered **unreasonable** and will be given the **lowest priority**.

- **Deposit:** The bank prioritizes depositing over **withdrawal requests** and **unreasonable loan applications**.
- **Withdrawal request:** Withdrawal request is **only better** than the **unreasonable loan application**.

You need to help the bank assign the priorities according to the type of transaction. For the same priority, you can further **prioritize the transactions by the account balance**.

Transaction.java makes use of enums, please read the paragraph below for more details.

3.1 A brief aside on comparing with Enums

Enums as a construct in Java implement the Comparable interface, meaning they already have a compareTo() written for them. (See Enum Type and java.lang.Enum.) They are compared based on their **ordinal values**. That is, the enum value with the *ordinal value* 0 is less than the enum value with the ordinal value 1. Enums are assigned an ordinal value, usually implicitly, when declared: the first listed enum value gets an ordinal value of 0, the second gets an ordinal value of 1, and so forth. You can experiment with the example here to better understand comparing enums and ordinal values.

4. Manager program

We provide the file **BankManager.java** to help you out in establishing the transaction manager program. Here you need to add various transactions with different amounts and priorities. Don't forget that customer balances are important. The manager program should also be able to display the current pending transaction queues in order.

5. Testing

In the file **BankManagerTester.java**, you need to verify:

- Correct implementation of the heap function.

- Priorities are correctly assigned according to given rules.

- The manager program is working.

All of the immediate tests on Gradescope will be on your testers. Knowing that you wrote a good tester is a good way to ensure that your other code is correct, especially without any immediate feedback from us. You've been practicing this skill ALL semester, you got this!

Assignment Submission

Hooray, you've finished this CS 300 programming assignment!

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, make a final submission of your source code to Gradescope.

For full credit, please submit the following files (**source code**, *not* .class files):

- **TransactionHeap.java**
- **BankManager.java**
- **BankManagerTester.java**
- **Transaction.java**

Additionally, if you used generative AI at any point during your development, ***you must include screenshots*** showing your FULL interaction with the tool(s).

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Students whose final submission (which must pass ALL immediate tests) is made before 5pm on the Wednesday before the due date will receive an additional 5% bonus toward this assignment. Submissions made after this time are NOT eligible for this bonus, but you may continue to make submissions until 10:00PM Central Time on the due date with no penalty.

Copyright notice

This assignment specification is the intellectual property of Blerina Gkotse, Hobbes LeGault, Caleb Raschka and Yuhao Liu and the University of Wisconsin–Madison and ***may not*** be shared without express, written permission.

Additionally, students are ***not permitted*** to share source code for their CS 300 projects on *any* public site.