P10 Prioritized Task Manager

Overview

Part of P07 included implementing a sorted list to help manage and prioritize Tasks in a to-do list, which required *linear* time complexity to add Tasks to the correct spot in the list. (If you hadn't put the pieces together yet, that was a rudimentary priority queue.)

Now that you know about heaps, we can manage that sorted list with a better runtime complexity! In this assignment, you will create a priority queue using a **heap** that can prioritize Tasks using any of 3 different criteria: title, priority level, or estimated completion time. You'll also be able to change the priority sorting on the fly in a much more efficient manner than before.

Friends don't let friends maintain priority queues using insertion sort!!

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **04/28**.
+5%	**Bonus Points**: students whose ***final*** submission to Gradescope is before **5:00 PM Central Time** on **WED 05/01** *and* who pass ALL immediate tests will receive an additional 2.5 points toward this assignment, **up to a maximum total of 50 points**.
15 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.
30 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.
50 points	**MAXIMUM TOTAL SCORE**

Learning Objectives

After completing this assignment, you should be able to:

- **Implement** a priority queue using an array-based max-heap.
- **Explain** the difference between the runtime of two different priority queue implementations.
- Continue to **practice** developing and writing unit tests.
- **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` in `TaskQueue` and `TaskQueueTester`
    - `java.util.Arrays` in `TaskQueue` and `TaskQueueTester`
- 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 the `TaskQueueTester` class 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 a file called `log.txt` containing the full transcript of your usage of the tool. 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/398763/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 Prioritized Task 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 three (3) Java source files from the assignment page on Canvas:
   a. **Task.java** (object, does NOT include a main method)
   b. **CompareCriteria.java** (enum - Javadocs)
   c. **PriorityLevel.java** (enum - Javadocs)

3. Create two (2) Java source files within that project's src folder:
   a. **TaskQueue.java** (does NOT include a main method)
   b. **TaskQueueTester.java** (includes a main method)

1. Testing

We may sound like a broken record at this point in the semester but testers are an important skill and should be done ***in conjunction with*** writing other code. There will not always be an "all knowing" oracle to let you know that code you wrote is correct.[1]

In a move that may upset some of you, all of the immediate tests on Gradescope will be on your testers, *and we're not going to tell you* ***what's broken*** *about the broken implementations*. 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!

- ➢ The list of **required** tester methods can be found in these Javadocs.

2. Prioritizing Tasks

To accomplish ordering our tasks in some manner but having BETTER runtime in doing so, you will make a priority queue using an array-based max-heap implementation. You have the option to implement the percolate methods iteratively OR recursively. (If you are looking for more recursion practice, this is a great opportunity.)

Tasks with HIGHER priority should come out of the queue first. How we decide which Task has higher priority can vary, based on which comparison criterion is selected for the priority queue:

1. TITLE - Compare and prioritize the tasks based on their titles in ***reverse*** **lexicographical order**. This means that the title "a task" > "b task", and "A task" > "a task". Tasks that are considered greater reverse-lexicographically[2] will have higher priority.

2. LEVEL - Compare and prioritize the tasks based on their priorityLevel. The ordering of the levels is OPTIONAL < LOW < MEDIUM < HIGH < URGENT.

3. TIME - Compare and prioritize the tasks based on their estimated completion times. Tasks that take longer to complete are considered larger and thus have higher priority.

The `Task.java` file we provided (Javadocs) does not have a `compareTo()` method implemented! Implement it now, using the ordering described above. Note that this method is ***NOT*** the Comparable interface's required method, as it has a second parameter communicating **how** to compare the two Tasks.

The list of methods you need to complete for the `TaskQueue.java` class can be found in these Javadocs.

---

[1] In many courses, there are no autograders available to you. This is also common in industry.

[2] Standard lexicographical ordering *generally* follows alphabetical ordering, with a caveat. If letters are NOT in the same case, uppercase letters **ALL** appear *before* lowercase letters (so Z < a in standard lexicographical ordering, and Z > a in reverse lexicographical ordering). See String.compareTo() for more information.

2.1 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.

3. OPTIONAL: Writing a Driver

You will notice that we have not provided a driver of any sort for this assignment. We decided to not provide one for 1) the sake of keeping this assignment simple given it's the end of the semester and 2) it is a great opportunity for something you can do to start building on this program out as part of your projects portfolio for future internships! If you want to create a driver in your (copious) free time, you are more than welcome to do so!

Assignment Submission

Hooray, you've finished the FINAL 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, submit your source code through Gradescope.

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

- Task.java
- TaskQueue.java
- TaskQueueTester.java

Additionally, if you used any generative AI at any point during your development, you ***must*** include the full transcript of your interaction in a file called

- log.txt

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