Assignment 5

Objectives

- More practice implementing an interface in Java
- Exposure to the Priority Queue ADT
- Practice implementing a Heap
- Exposure to the Comparable<E> interface

Introduction

This assignment has two parts:

Part 1 asks you to implement the `PriorityQueue` interface using an array-based Heap data structure that will store `Comparable` objects (objects that implement the Comparable interface). A reference-based list implementation of `PriorityQueue` is provided for you (`LinkedPriorityQueue.java` and `ComparableNode.java`) so you can run the `Part1Tester` and compare running times of the two implementations.

Part 2 asks you to implement a small application modeling a transportation boarding gate (i.e. an airport boarding gate) in which you will use your `HeapPriorityQueue`.

Part I

1. Download the files: `A5Tester.java`, `PriorityQueue.java`, `HeapPriorityQueue.java`, `LinkedPriorityQueue.java`, `ComparableNode.java`, `HeapEmptyException.java` and `HeapFullException.java`.
2. Read the comments in `HeapPriorityQueue.java` carefully
   a. **Add the constructors and required method stubs according to the interface in order to allow the tester to compile.**
3. Compile and run the test program `A5Tester.java` with `LinkedPriorityQueue.java` to understand the behavior of the tester:
   ```
   javac A5Tester.java
   java A5Tester linked
   ```
4. Compile and run the test program `A5Tester.java` with `HeapPriorityQueue.java` and repeat step a and b below
   ```
   javac A5Tester.java
   java A5Tester
   ```
   a. Uncomment one of the tests in the tester
   b. Implement one of the methods in `HeapPriorityQueue.java`
   c. Once all of the tests have passed move on to Part II.
5. You can run the tester on the `LinkedPriorityQueue` by running as follows:
   ```
   java A5Tester linked
   ```
   Notice how much slower it is! It even skips a test and it still SLOW!

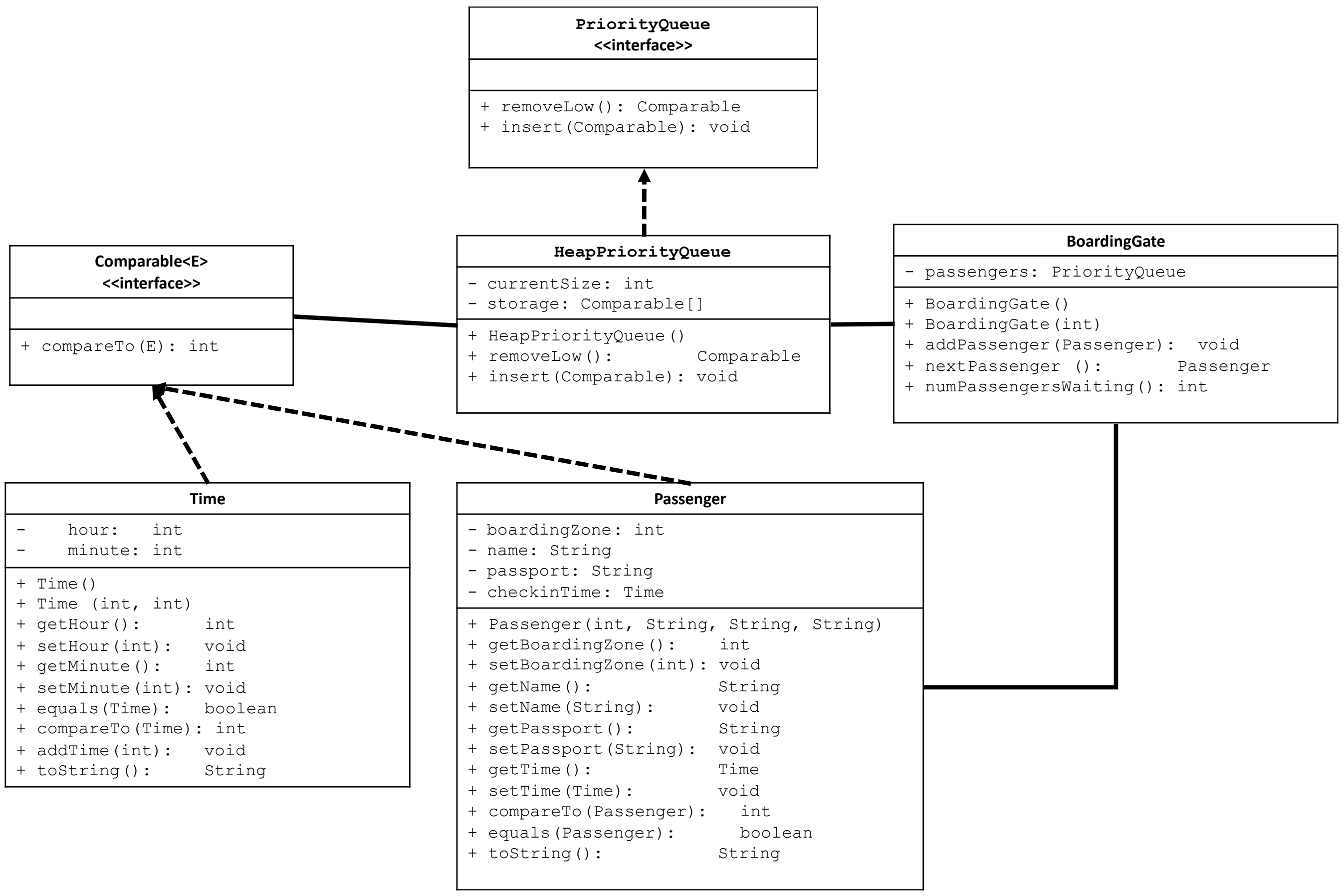
Part II

For this part of the assignment, you will be creating an application to support the modelling of a simple boarding gate (i.e. an airplane gate) in a transit facility (i.e. an Airport). You are asked to write the software that will manage the boarding of passengers based on their `boardingZone` and their `arrivalTime`.

Imagine… you are checking people in for their flight and assigning them a position in line. Passengers arrive continuously, but boarding cannot begin until the plane is ready.

- A passenger (A) enters, they are boarding in zone 3, you add them to the queue.
- A passenger (B) enters, they are boarding in zone 1, you add them to the queue ahead of the previous person.
- Another passenger (C) enters, they are boarding in zone 1, you add them to the queue ahead of person A in boarding zone 1 but behind the person C in boarding zone 1 because they arrived at a later time.
- The plane begins boarding, person B gets their passport checked to board
- A passenger (D) enters, they are boarding in zone ~~3~~ 2, you add them to the queue ahead of passenger A but behind passenger C.

The given a tester `A5Tester.java` that will test the functionality of your `BoardingGate` implementation and mimics a scenario similar to the one the above. The classes involved are represented in the UML diagram below. Read the tester and documentation carefully to help you understand how the classes you will be writing will be used.



Submission

Submit only your `HeapPriorityQueue.java`, `Passenger.java` and `BoardingGate.java` via conneX.

Please be sure you submit your assignment, not just save a draft. **ALL late** and **incorrect** submissions will be given a **ZERO** grade.

Even if you chose to not complete the assignment, you must **submit ALL files** and they **MUST compile with the full, uncommented tester** or you will receive a **ZERO** grade.

If you submit files that do not compile, or that do not use the correct method names you will receive a **zero grade** for the assignment. It is your responsibility to ensure you follow the specification and submit the correct files.

Your code must **not** be written to specifically pass the test cases in the testers, instead, it must work on all valid inputs. We will change the input values, add extra tests and we will inspect your code for hard-coded solutions.

A reminder that it is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution. We will use plagiarism detection software on your assignment submissions.