CSCI 241, Summer 2017, Filip Jagodzinski
Programming Assignment 4
Check Canvas for Due Date and Time

WESTERN WASHINGTON UNIVERSITY
Computer Science

For this programming assignment, you will implement a priority queue using a min-heap. Your task is to prioritize airplane departures which request takeoff at different times from the Seattle's airport. Read the ENTIRETY of this assignment before you begin, because it contains optional (extra credit) components.

I. Program Behavior, User's perspective

requests.txt

```
DL717 SEA NRT 12:05 317
WN766 SEA SJC 12:06 140
AA399 SEA JFK 12:07 268
EK230 SEA DXB 12:08 385
DL2635 SEA BOS 12:11 240
AS922 SEA DCA 12:12 275
```

- The program is named TakeoffQueue, in the file *TakeoffQueue.java.*
- The program is invoked via the command line, and requires 2 command line arguments. The first is the name of a file, and the second is the number 1, 2 or 3 specifying which prioritization schedule to use (see below). A sample file is shown above right. A sample invocation:

  Sample Invocation 1 : `java TakeoffQueue requests.txt 1`
  Sample Invocation 2 : `java TakeoffQueue requests.txt 2`
  Sample Invocation 3 : `java TakeoffQueue requests.txt 3`

  The program reads the plain text file. Each line has the following format:

  ```
  <Flight Num> <From> <To> <Takeoff Request Time> <Num Passengers>
  ```

  The `From` and `To` are 3 letter airport codes. The `Takeoff Request Time` is when the airplane informs the control tower that it is ready to take off.

  The program grants takeoff requests using a prioritization formula, depending on which schedule is used (1, 2 or 3). For schedules 2 and 3, prioritization is based on the count of passengers; the higher the count of passengers, the higher the priority. Only 1 airplane can be taking off at any one time (the flight that is at the top of the min heap!). Scheme 3 also takes into account the request time AND the amount of time that an airplane needs to take off once permission has been granted. The time to takeoff once permission is granted is `Num Passengers /2`, where `/` is the integer division operator, ceiling rounded (next highest) to a multiple of 60. The units is seconds.

- The second argument determines whether a simple (1), intermediate (2), or full (3) prioritization schedule is followed. The three choices also indicate what is printed to the screen.
    - **Simple prioritization (1)** : the airplanes are granted permission to take off based on the order in which the requests are made. In this scenario the count of passengers is not taken into account when determining takeoff order; nor is the time it takes to take off (you are assuming that takeoff is instant). The output is only the order of takeoffs.
    - **Intermediate prioritization (2)** : the airplanes are granted permission to take off based on the count of passengers. However, the time when takeoff requests are made is not considered; <u>assume that all takeoff requests arrive all at once</u>. The time that it takes for a flight to take off also is assumed to be instantaneous. The output includes the flight numbers in the correct order, and passenger counts.
    - **Full prioritization (3)**: the airplanes are granted permission to take off based on the count of passengers, and the time when the takeoff request is made. In this scenario, the time that it takes an airplane to take off is also taken into account, which may or may not induce an airplane at the top of the min-heap to have to wait to take off. While planes are waiting to take off, a NEW take off request might be received, which would cause the min-heap to be reordered. The output is the order of takeoffs, and their times. **This (full prioritization) is the extra credit portion of this assignment. Attempt it ONLY after you've completed (1) and (2).**

Sample invocations.

Invocation	Output

`java TakeoffQueue requests.txt 1`	`DL717`
	`WN766`
	`AA399`
	`EK230`
	`DL2635`
	`AS922`

`java TakeoffQueue requests.txt 2`	`EK230 385`
	`DL717 317`
	`AS922 275`
	`AA399 268`
	`DL2635 240`
	`WN766 140`

`java TakeoffQueue requests.txt 3`	`DL717 departed at 12:08`
	`EK230 departed at 12:12`
	`AS922 departed at 12:15`
	`AA399 departed at 12:18`
	`DL2635 departed at 12:20`
	`WN766 departed at 12:22`

The first invocation is simply the order of takeoffs in which the requests arrived (yes, that is the same as the order of flights in the file). The second invocation indicates that takeoff occurs based on the passenger count prioritization. The higher the count of passengers, the higher the priority. Thus, the highest passenger count flight (EK230) takes off first, followed by the next highest passenger count flight (DL717), etc.

In the third invocation, DL717 made a takeoff request at 12:05pm. Because at that instance it was the ONLY flight with a request in the queue (min heap), it was granted permission to take off. DL717 has 320 passengers, thus the time to takeoff is 320 / 2 = 160 which rounds to 180, which is 3 minutes. Thus DL717 completed takeoff at 12:08.00. By that time, WN766, AA399, and EK230 had made takeoff requests, and prioritization based on passengers grants EK230 to takeoff, which needs 240 seconds = 4 minutes. Thus EK230 completed takeoff by 12:12.00. By that time, all of the flights in the file had requested takeoff, and based on their ordering (AS922 > AA399 > DL2635 > WN766), AS922 takes off next, and completes takeoff at 12:15, AA399 completes takeoff at 12:18, DL2635 at 12:20, and WN766 at 12:20.

II. Program Specifics, Back-end perspective

- Error catching is NOT required. You do not have to check if a user specifies an incorrect number of command line arguments, if the file exists, etc.
- The order requests (times) in the file are already ordered for you in terms of sequence. The first (top-most) line in the file has the earliest takeoff request time. The second-from-the-top line has the next earliest takeoff request, etc.
- The file contains takeoff times that are all PM.
- You MUST implement a min-heap.
- For implmenting (3) : the order of events at <u>each</u> second of time should be
    - Process just-received takeoff request and update min-heap
    - Check if runway is in empty
    - Grant takeoff request to highest priority item in min-heap IF runway is empty; that flight will IMMEDIATELY begin takeoff

III. Hints and Notes

- Use the sample plain-text file on the course website to get started.
- Once (1) and (2) prioritization schedules are implemented, implementing (3) requires only a handful of lines of code more … THINK before you start coding.

IV. Submission and Rubric

Submit your .java file via Canvas.

You earn points for implementing correctly the asked-for program such that it works as intended (correctness) and so that the program runs in a reasonable amount of time (efficiency). Points are deducted for errors in commenting, style, etc. (clarity).

Code : Correctness	
Program parses input plain-text file	+1 point
Min-heap is implemented	+4 points
Choice 1 is correctly implemented USING the min-heap, correct output	+8 points
Choice 2 is correctly implemented USING the min-heap, correct output	+5 points

Code : Efficiency	
On input of a file with *n* flight details, *n* < 100, the run-time of the program should be seconds, including building the min-heap and deleting from it. When *n* > 1,000 – and even up to 1,000,000 – the run-time should also be reasonable (less than 1 minute).	+2 points

Code : Clarity, deductions	
Inadequate commenting: your code should have a block comment at the top of the file that specifies the author, date, and program's purpose. Each function should be accompanied by a brief comment, and precede code sections in a function by a comment that specifies what that section does. Code sections are logical chunks of code, usually 5-10 lines. Your code should be compilable from the command line using *javac* and/or compiled using a standard IDE. If special libraries, or compile flags are needed, state that in the comment block at the top of the file. File must be correctly named.	-2 points
Inadequate variable and function names: variables should be descriptive (`a` is bad variable name, but `anArray` is fine); CS standard one-letter variables such as `i`, `j`, `k`, etc. for loop iterator variables are fine.	-2 points
Inadequate style: each function should be written concisely, and not be too lengthy (number of lines of code); if a function is 100+ lines, it should be broken up into separate functions. Alternatively, a function should not be so terse that it is cryptic.	-2 points

Code : Extra credit	
Choice 3 is correctly implemented, correct output … all or none … no partial credit	+5 points