CSC 360 Programming Assignment #9: JCFs
Due Date: Wednesday, November 19

We will implement a wacky simulation to demonstrate the difference between various types of JCF containers. The simulation will be of NKU's Office of the Registrar. This simulation will be somewhat like the bank simulation covered in class except that there will be 4 waiting lines that an entering Student can wait in but each line uses a different form of access, as opposed to the strictly first-in-first-out access of a queue. You will use four different JCF classes for the lines.

The first line will use a LinkedList where Students will be inserted at the end using offer and removed from the front using remove (that is, this line is a queue). The second line will use a PriorityQueue where Students will be inserted using offer and be placed based on highest priority first, and Students will be removed from the front using remove. The priority for each Student will be based on the Student's number of credit hours earned. The third line will be a Stack where Students are added using push and removed using pop. The fourth line will be implemented using an ArrayList. In this line, Students are always removed from the front but are added at a random index that you generate each time a new Student is to enter the line. From simulation results, the four lines operate fairly similarly with line 1 usually providing the best performance (lowest wait time).

The simulation class will have several instance data: the four lines, an array of Registrars, int values for total wait time, total students, number of registrars, length of simulation, current time, next line to add a Student, next line to remove a Student, and two int arrays storing for each line the line's total wait time and total students. For instance, lineTotalWait[i] is the total wait time of the Students to this point of the simulation for line i.

The simulation will work like this. The simulation will run for a specified number of registrars and a specified number of minutes. It will continue to run after that the time limit has elapsed until all of the lines are empty. For every minute, there is a chance that Students can enter a line. The chances are as follows:

39% no Students	34% 1 Student	20% 2 Students
5% 3 Students	2% 4 Students	

The next Student to enter will join the *next* line. The first Student to enter goes in the first line, the next Student (whenever that is) will join the second line, and so forth through the fourth line and then the next Student to enter will join the first line. Use the instance datum next line to add a Student to indicate the line that the next Student will be added to. This variable will range between 0 and 3 (or 1 and 4, your choice). Upon a Student entering a line, create a new Student object. Student objects will be like Customer objects except that they will also have a number of credit hours earned. The credit hours earned for the Student will be randomly assigned as follows. Use a random number from 0 to 200. If the number is between 0-50 then credit hours earned is 0 hours otherwise credit hours earned is the random number – 50 (so that 199 would become 149). The Student class needs to implement Comparable which will compare two Students on Credit hours (on a tie, use the time the Student entered the line as a tie-breaker). Comparable is only used for Students who are waiting in the Priority Queue line.

For each new Student, as you add it to a line, store the current simulation time that the Student was added to the line, and generate the Student's serviceTime (the time the Student will need with a Registrar). This value is generated randomly as follows:

35% 1 minute	42% 2 minutes	17% 3 minutes	5% 4 minutes	1% 5 minutes

Once the simulation time has elapsed, no new Students will join any line but the Registrar's office will remain open until all Students have been served.

Each Registrar will be an object much like the Teller. A Registrar will have a single instance datum, a Student. When a Registrar is available, it will remove a Student from the next line. The next line will be selected based on the variable next line to remove a Student from. Note that this is a separate variable from the one used to indicate where a Student is added to. This number will also range between 0 and 3 (or 1 and 4).

Upon removing the Student from a line, the simulation computes this Student's wait time (current time – time in line) and then updates total wait time and total wait time for the given line. Add 1 to the total number of Students serviced (or add 1 to total number of Students when Student is created, your choice).

If a Registrar currently has a Student (the student instance datum is not null), the Registrar will service the Student for 1 minute. Decrement the Student's randomly generated serviceTime. Upon this value reaching 0, the Student is done and will leave the Registrar, set the Student instance datum for this Registrar to null. In the next minute, the next Student will be selected from a line.

You will write four classes. The simulation program will be a class without a main. The constructor should initialize all instance data appropriately. It will receive two parameters, the number of Registrars for this run and the duration of the simulation in minutes. A run method will run the simulation. The simulation class should also have accessor methods to return the totalWaitTime, totalStudentsServiced, and to return the int arrays for total students in each line and total wait time of each line. You can also compute average wait times here in this class or separately in the User class. You will also need a Student and a Registrar class. These classes can be nested inside of your Simulation class, or in a separate file, your choice.

Your User class will have a main method and will use nested for loops to iterate through the number of Registrars and minutes to simulate. Each inner loop, create a new Simulation object, run it and obtain the relevant statistics to output (number of Registrars, time, total wait time, total Students serviced, average wait time, total wait time for each of the four lines and the total number of Students per line). Output one Simulation run result per line. Iterate for 2-6 Registrars and for 60-480 minutes in 60 minute increments (i.e., 60, 120, 180, 240, 300, 360, 420, 480). The output should be nicely formatted. The following is an excerpt from my program for 2 and 3 registrars.

Reg	Time	Students	Wait	Avg	L0 Wait	L1 Wait	L2 Wait	L3 Wait
2	60	64	1173	18.328	17.13	18.19	18.75	19.25
2	120	114	3219	28.237	27.86	28.52	28.29	28.29
2	180	160	4448	27.800	27.33	27.70	27.90	28.28
2	240	227	13551	59.696	59.25	59.39	60.16	60.00
2	300	288	18564	64.458	63.76	64.14	64.68	65.25
2	360	368	32013	86.992	86.29	86.70	87.20	87.78
2	420	433	44502	102.776	102.93	102.17	102.94	103.06
2	480	463	48069	103.821	103.34	103.78	104.36	103.80
3	60	41	23	0.561	0.73	0.40	0.50	0.60
3	120	129	807	6.256	6.00	6.16	6.44	6.44
3	180	157	224	1.427	1.65	1.26	1.26	1.54
3	240	222	562	2.532	2.46	2.45	2.65	2.56
3	300	277	1413	5.101	5.10	5.04	5.14	5.12
3	360	342	1286	3.760	3.71	3.93	3.65	3.75
3	420	402	1890	4.701	4.73	4.72	4.64	4.71
3	480	487	2526	5.187	5.04	5.25	5.30	5.16

Hand in your four classes (commented) and the output from your simulation run.