Queues Class Exercise

View the following clip for a moment: <http://www.csanimated.com/animation.php?t=Queue>

**Answers for part A and B are on question sheet.**

**Part A:**

Queue interface must provide the following operations

1. Enqueue - add to end of the list
2. Dequeue – remove from the beginning of the list

Queue interface may also provide the following additional operations:

1. Peek - view the item at the beginning of the list
2. isEmpty()
3. isFull() - only applicable if the data is stored in a data structure with a finite set of elements such as an array
4. dequeueAll() - remove all items in the list

The code below implements a Queue interface:

Study the code and answer the following questions.

1. What data structure is being used to implement the queue?
2. What type of data can be stored using this queue?
3. When will the queue run out of space?
4. Assuming the queue is instantiated with the no args constructor, how many items will the queue be able to store?
5. Assume someone is executing the following main(), demonstrate pictorially what will be occurring internally within myList.

**public** **class** UseArrayQueue {

**public** **static** **void** main(String[] args) {

Queue<String> myList;

myList = **new** Queue<String>();

**try**{

myList.enqueue("Rachel");

myList.enqueue("Gabi");

myList.enqueue("Tzippy");

myList.enqueue("Rivki");

myList.enqueue("Avigayl");

myList.enqueue("Nechama");

}

**catch**(QueueFullException e){

**try**{ **while** (**true**){

System.*out*.println("Serving " + myList.peek());

myList.dequeue();

}

}

**catch**(QueueEmptyException qe){

System.*out*.println("end queue application");

}

}

}}

1. Now suppose I run the following sequence of statements what will be happening internally within myList? Specifically, in which element will Tzippy be placed? Why?

**public** **class** UseArrayQueue2 {

**public** **static** **void** main(String[] args) {

Queue<String> myList = **new** Queue<String>();

myList.enqueue("Shifra");

myList.enqueue("Devora");

myList.enqueue("Esti");

myList.enqueue("Shayna");

myList.enqueue("MPlonczak");

myList.dequeue();

myList.enqueue("Tzippy"); }

}

1. What purpose is the % operator serving in determining where to place an item within the array?
2. Why is it impossible to simply increment by 1 to determine the next available position in the array?
3. What type of array is the Queue using to store its data? How would you describe it?

Part B:

1. Implement the Queue using an ArrayList
2. Implement the Queue using a LinkedList

* What is the specific challenge?
* How would you modify the LinkedList implementation to make it easier to use it to implement the Queue? Write the code to modify the LinkedList and then use the modified LinkedList to implement the Queue.



Part C:

Study the classes that implement the simulation of a Call Center and answer the following questions

**public** **static** **void** main(String[] args) {

// **TODO** Auto-generated method stub

SimulationValues myCustomerService = **new** SimulationValues();

Scanner input = **new** Scanner(System.*in*);

myCustomerService.setValues(input);

myCustomerService.run();

}

The main() method has only 4 statements.

1. What is the purpose of each of the statements? What will each statement accomplish?

Line 1: creates a new instance of SimulationValues, and sets up all the class variables, timer, linked list of calls, random generator..

Line 2: sets up a scanner from the keyboard

Line 3: connects the scanner “input” to the setValues method of SimulationValues instance myCustomerService, which sets up the number of calls perhour, and the percent of calls that are serviced in a certain amount of time and enters the sum into an array of servicePercent. It also asks the user how long to run the simulation and sets a timer for that amount of time.

Line 4: Calls the run method, and while there is still time left, and they finish a call, and there are more calls, take the next call to service, and then check for a new call while the time is passing. When there is no more time, then don’t accept any more calls and just service who is left on the queue and display results

SimulationValues class

1. Which variable is referencing the Queue in which the calls that arrive at the center will be stored?

myIncomingCalls

1. What type of data structure is actually being used to implement the Queue?

linkedList

1. What data is the user providing upon which to base the simulation?

The arrival rate of calls, the percentage of time the calls take, and the amount of time to run the simulation for

1. What data is the SimulationValues class compiling as it runs?

The timer calculates the time the calls took, and time remaining. The randomGenerator also generates random numbers as the program is executing.

1. What statistics does it provide once it has completed executing?

The average waiting time per call

1. Which object is the SimulationValues class using to keep track of the current simulated time? What methods does this object provide?

The Timer object, setTimer(), timeRemaining(), tick(), hasTimeLeft()

1. Which of the methods actually executes the simulation?

Run()

1. Study the code in the run() method
   1. What is the difference between busyTimeRemaining and myTimer.timeRemaining()?

BusyTimeRemaining is referring to the time left that the service is busy servicing a call

myTimer.timeRemaining() is referring to the amount of time that the service has left to run until it won’t accept any more calls to enter the queue

* 1. What controls how many times the loop will execute? How was that value set?

In the first loop, it will take more calls as long as there is more time left in myTimer() which was set in the setValues() method by user input

In the second loop, it will loop as long as there are more calls on the queue myIncomingCalls which are added in the checkForNewCall() method. There is a method that will return if the queue is empty and will execute as long as in is not empty

* 1. In general terms, what happens each time the loop executes?

It will take a call, and then check for a new call and take off a minute for the call and execute again as long as the myTimer() has more time. When there is no more time, it will print out “Not accepting more calls….” And then will just service any calls that were waiting on the queue until there are no more left and take off a minute for each call it services. Once the simulation is finished, it will display the number of calls processed and the average waiting time…

1. Study the code in the service() method?
   1. How does the method determine that a Call is still being serviced?

BusyTimeRemaining>0

* 1. What does the method do if a Call is still being serviced?

It decrements a minute from the busyTimeRemaining and then returns to the run method, where it will check for new calls and decrement a minute from the timer, and continue servicing calls (either that same call if there is still more busyTimeRemaining, or take the next incoming call

* 1. What does the method do if a Call is no longer being serviced?

If there is a call waiting, it will take the next call, and remove it from the queue and set the busyTimeRemaining to the amount of time the call will take, and update the totalWaiting time of all the calls to include the waiting time of this call.

* 1. How does the method determine how long it will take to service a new Call? What does the method do with this information?

busyTimeRemaining=nextCall.getServiceTime() the method returns the time the call will take and sets it to busyTimeRemaining

* 1. How does the method determine the amount of time a particular Call had to wait on the Queue before it got serviced?

The number of remaining minutes from when the call is taken is subtracted from the number of minutes remaining for the simulation when the call was added to the queue.

1. Study the code in the CheckForNewCall() method.
   1. How does the method determine whether another Call has arrived at this given moment?

It takes a random number and if it is less than the arrival rate (percentage of calls per hour) \*100 then a call arrived

* 1. How does the method determine the estimated time it will take to service the Call that has arrived (if one has indeed arrived)?

It creates a random percentage number of time that the call will take and while the percentage is greater than the percentage bracket of servicePercent based on serviceTime, serviceTime is incremented by one minute, until the servicePercent variable holds the correct percentage amount of time that the call will take and calculated the correct amount of minutes it will take to service the call in that servicePercent bracket.

* 1. What does this method do with the Call that has arrived?

It creates a new call to add to the queue with the time it was added, and the number of minutes it will take to service the call, so it can later calculate the amount of time the call waited.