**CS 1181 Programming Assignment 6  
Queueing Systems**

Queue

Queue

The purpose of this lab is to give you a chance to see how queues can be used in a typical application. The diagram below illustrates the system you will be implementing. The Request Generator bubble represents a program component that is simulating the arrival of a series of service requests. Those requests are placed into an input queue by the Request Generator. The three Request Server bubbles are duplicates of a program component that knows how to service a request. Each of the Request Servers will attempt to get a service request from the input queue. When successful it will simulate processing the request and then place a

notification into the output queue indicating that it completed processing a request. The Output Server bubble takes notifications from the output queue and prints them to the console.

This is a multithreaded application. You are being provided with a NetBeans project containing the classes needed to implement this system. Since you do not know anything about multithreading, the parts of this project having to do with multithreading are provided to you. Your primary job is to implement the queues that are needed to make this system work. There are some other pieces of code you need to complete to have a project that works.   
The following describes the various classes that are part of this system in more detail.

**The Delay Class (provided)**

This class is used to create delays of random length in the program. Since the program has lots of different activities running at the same time, it is necessary for each activity to take a break when there is no work to do or when the activity needs to simulate the passage of time spent processing a request. This class provides methods which when called, will place the calling activity (thread) into a sleep state thereby allowing some other activity in the program to run. The constructor and the methods of this class are documented in the starter project provided with this lab.

**The RequestQueue Interface (provided)**

This is the interface that all the other program components expect to be able to use in order to access the input and output queues described above. The interface defines the following method signatures:

void enqueue (String input); // Add a String to the queue  
String dequeue () throws Exception; // Get a String from the queue. If empty, throw exception  
int getMaxLength (); // Report the maximum length the queue reached

**The MyRequestQueue Class (you must implement and test this)**

This class implements the RequestQueue interface. This class is implemented using your own code for linking nodes together in a *linked queue data structure*. You need to define a Node class inside of this class. You need to specify all member variables needed to maintain your queue and implement all methods of the RequestQueue interface. Note that you will be storing Strings into your linked list. DO NOT use any generic stuff for this.

**The RequestGenerator Class (partial – you must add code to this)**

The purpose of this class is to place strings into a RequestQueue object at random times. When objects of this class are created, the constructor is given the RequestQueue object to be used, and the number of requests to be generated. These are stored into member variables by the constructor which you must provide.

Since this class is to be executed by a thread as a separate activity, you must implement all the other code for this class inside the method called ***run***. The comments inside the ***run*** method explain what you need to do. The main things are as follows:

* Create a delay object used to put this activity to sleep
* Make a loop that executes the number of requests times
  + Each time thru the loop use the delay object to wait a random amount
  + Place a string on the input queue

**The RequestServer Class (mostly provided)**

The purpose of this class is to process requests from the incoming request queue and then send a notification to the output queue when processing has completed. You will need to provide the constructor for this class. The code in the run method to process requests from the input queue and place notifications into the output queue has been provided.

**The CS1181Lab5 Class (mostly provided)**

The main method of this class creates all the objects needed for this queueing system to run. It creates the threads needed to execute the request generator, and the request server objects. You need to add code to create the necessary objects and to interact with the user to find out how many requests to process and how many servers to run. There are 5 lines which have been commented out to avoid compiler errors. They are marked with //\*\* at the start of the lines. After you have added the constructors to the above classes those 5 lines can have the comments removed. Once the threads are running, the main method calls the processOutput method to take the notifications off the queue and output the strings to the console. After all notifications are processed, the main method reports on the maximum lengths attained by the input and output queues. There are comments in this class file with specific instructions on what you need to do.

**The Simulation**

Once you have all the classes implemented, run your program. Try running your program with user input of 100 requests and 3 servers and observe the output. It will take a while for the program to complete. A partial sample output appears below.

1: Request # 94

2: Request # 95

1: Request # 96

3: Request # 97

1: Request # 98

2: Request # 99

3: Request # 100

RequestServer 1 served 31 requests - terminating!

RequestServer 3 served 34 requests - terminating!

RequestServer 2 served 35 requests - terminating!

Output processing is done

Queue Stats: Input - MaxLength = 8, Output - MaxLength = 5

Finished all threads

**Development Strategy**

Start by implementing the MyRequestQueue class. Use the QueueTester class that has been added to the starter project to thoroughly test the operation of your MyRequestQueue class. When you can properly add and remove things from the queue, make sure that the maxLength is working correctly. Document each test case with comments.

**For your project to be successful, it is VERY IMPORTANT that you fully test your MyRequestQueue class BEFORE you attempt to use it with the other classes provided!**

Once your MyRequestQueue class is working properly, add the code needed to complete the other classes as described above. Run the simulation and observe the output.

**Rubric (60 pts)**

**If it does not compile, you receive 0 points.**

1. **MyRequestQueue class (20 pts)  
    enqueue method – must be based on linked list and properly add to back of list  
    dequeue method – must be based on linked list and properly delete from the front  
    maxLength must be correctly tracked**
2. **QueueTester class (8 pts)**

**Thoroughly tests the three methods of the MyRequestQueue class**

**Each test case is documented with comments**

1. **RequestGenerator class (10 pts)  
    Constructor handles initialization properly  
    Run method properly waits, then places correct number of strings on the queue**
2. **RequestServer class (5 pts)  
    Constructor handles initialization properly**
3. **CS1181Lab5 class (10 pts)  
    Main method creates required objects and gets input from user**

**Simulation runs and produces expected results**

1. **All code follows appropriate style guidelines. (7 pts)**

**Remove all comments telling you what you need to do.**

**Add your own Javadoc comments for every method that needs them.**

**Run the Javadoc tool to generate the final Javadoc documentation**

**What to submit**

**The NetBeans project in one ZIP file. Copy the code for your MyRequestQueue class, your QueueTester class and your RequestGenerator class into a single text document. Only 2 files are submitted to the drop box.**