# *Programming III (420-B31-HR)*

# *Lab 9 – The Queue Abstract Data Type*

Date assigned: Tuesday, October 25, 2065

**Objectives:**

At the completion of this lab, students will be able to

1. use the Queue ADT to enqueue and dequeue different types of elements onto a first-in, first-out (FIFO) collection.
2. compare the operations of a LIFO stack and a FIFO queue.
3. use composition to implement an ADT.
4. use the Priority Queue ADT to enqueue and dequeue different types of elements onto a highest-priority-in first-out (HPIFO) collection.

**Reference:** Notes: *Queues*

**To Start:**

1. Download the **B31\_L09\_Queues** folder from Moodle. Rename it to ***username*\_B31\_L09\_Queues**.
2. Start **Eclipse** and use your **420-B31\Labs** folder as your workspace.
3. Create a new **Java Project** called ***username*\_B31\_L09\_Queues**. Add Junit4 to your build path.
4. Rename **B31\_L09\_Queues\_Answers.docx** file to ***username*\_B31\_L09\_Queues\_Answers.docx**.

**To be handed in:**

1. Your ***username*\_B31\_L09\_Queues** folder should be zipped and uploaded to **Moodle**.

# The Queue ADT

**Objectives:** Learn to use a Queue.

**To Do:**

## Complete the table provided in ***username\_*B31\_L09\_Queues\_Answers.docx,** to understand how a queue works. Remember that the object state for a queue is the front, rear, size and the contents of the queue itself.

## Redo the palindrome problem from the last lab using both a stack and a queue. The idea is to store each character of the input string (skipping blanks and punctuation again) in both the stack and the queue. Since a stack and a queue will provide the stored characters in the opposite order, you can easily confirm or disprove that a string is a palindrome. Use the **PalindromeTest.java** junit Test case to test your method. The **Palindrome** class contains the code from the previous lab (lab 8) that uses only a stack.

# The Priority Queue ADT

## **Investigate**: Read the description of priority queues in Appendix A – PriorityQueues.

### Code the Java interface, **PriorityQueue**, based on the ADT specification.

### Complete the table in ***username\_*B31\_L09\_Queues\_Answers.docx** to map the methods of **PriorityQueue** to **Queue**. (Assume that the priority indicates the position in the queue – the element at the front of the queue has priority 1, the second element has priority 2 and so on. So if **enqueue(3,"new element")** is called, "new element" will be placed in the third position in the queue.)

### Code the implementation of the **PriorityQueue** using composition. (Refer to the class diagram below.) Call the class **CompositionPriorityQueue**. Use the **Queue** and **ListQueue** classes in the **gray. adts.queue** package in your implementation. Do not change anything in the **gray.adts.queue** package. If you are unsure about how the **enqueue(int, E)** method works, look at the test plan on the next page.



### Create a junit Test case to test your **CompositionPriorityQueue** using the test cases shown on the next page. Test and debug your class.

**Test Plan for enqueue(int priority, E element)**

| Method | Purpose | Object State | Expected  Result |
| --- | --- | --- | --- |
| PriorityQueue <String> pq = new CompositionPriorityQueue <String> (); | Instantiate a new priority queue | size = 0  front = null  rear = 1st queue slot | A PriorityQueue object for Strings is created |
| pq.enqueue("A"); | Enqueue an element with the lowest priority onto on empty queue | size = 1  "A"  front = "A"  rear = 2nd queue slot |  |
| pq.peek(); | To verify the queue state | size = 1  "A"  front = "A"  rear = 2nd queue slot | "A" |
| pq.enqueue(1,"B"); | Enqueue an element with a higher priority into a non-empty queue | size = 2  "B" "A"  front = "B"  rear = 3rd queue slot |  |
| pq.peek(); | To verify the queue state | size = 2  "B" "A"  front = "B"  rear = 3rd queue slot | "B" |
| pq.enqueue("C"); | Enqueue an element with lowest priority into a non-empty queue | size = 3  "B" "A" "C"  front = "B"  rear = 4th queue slot |  |
| pq.peek(); | To verify the queue state | size = 3  "B" "A" "C"  front = "B"  rear = 4th queue slot | "B" |
| pq.enqueue(2,"D"); | Enqueue an element with a higher priority into a non-empty queue | size = 4  "B" "D" "A" "C"  front = "B"  rear = 5th queue slot |  |
| pq.peek(); | To verify the queue state | size = 4  "B" "D" "A" "C"  front = "B"  rear = 5th queue slot | "B" |
| pq.enqueue(1,"E"); | Enqueue an element with a highest priority into a non-empty queue | size = 5  "E" "B" "D" "A" "C"  front = "E"  rear = 6th queue slot |  |
| pq.peek(); | To verify the queue state | size = 5  "E" "B" "D" "A" "C"  front = "E"  rear = 6th queue slot | "E" |
| pq.size(); | To verify the queue state | size = 5  "E" "B" "D" "A" "C"  front = "E"  rear = 6th queue slot | 5 |
| pq.dequeue(); | To verify the queue state | size = 4  "B" "D" "A" "C"  front = "B"  rear = 5th queue slot | "E" |
| pq.size(); | To verify the queue state | size = 4  "B" "D" "A" "C"  front = "B"  rear = 5th queue slot | 4 |
| pq.dequeue(); | To verify the queue state | size = 3  "D" "A" "C"  front = "D"  rear = 4th queue slot | "B" |
| pq.dequeue(); | To verify the queue state | size = 2  "A" "C"  front = "A"  rear = 3rd queue slot | "D" |
| pq.dequeue(); | To verify the queue state | size = 1  "C"  front = "C"  rear = 2nd queue slot | "A" |
| pq.isEmpty(); | To verify the queue state | size = 1  "C"  front = "A"  rear = 2nd queue slot | false |
| pq.dequeue(); | To verify the queue state | size = 0  front = null  rear = 1st queue slot | "C" |
| pq.isEmpty(); | To verify the queue state | size = 0  front = null  rear = 1st queue slot | True |

# Homework: Moodle Quiz

1. Log on to **Moodle**, go to the **Programming III** course page and complete the **Lab 9 Quiz by Friday October 28**.

**Marking Scheme**

|  |  |  |
| --- | --- | --- |
|  | **Mark** | **Out of** |
| **Part A: The Queue ADT** |  |  |
| * Queue ADT Table completion |  | 14 |
| * Palindrome class |  | 10 |
| **Part B: The Priority Queue ADT** |  |  |
| - PriorityQueue interface |  | 10 |
| - PriorityQueue operations mapped to Queue operations |  | 2 |
| - CompositionPriorityQueue |  | 15 |
| - junit class |  | 10 |
| **Total** |  | **61** |