Pages 284-294

1. True or False
2. A queue is a “first in, first out” structure. **T**
3. The element that has been in a queue the longest is at the “rear” of the queue. **F**
4. If you *enqueue* five elements into an empty queue and then *dequeue* five elements, the queue will be empty again. **T**
5. If you *enqueue* five elements into an empty queue and then perform the *isEmpty* operation five times, the queue will be empty again. **F**
6. The *enqueue* operation should be classified as a “transformer.” **T**
7. The *isEmpty* operation should be classified as a “transformer.” **F**
8. The *dequeue* operation should be classified as an “observer.” **F**
9. If we first enqueue *elementA* into an empty queue and then enqueue *elementB*, the front of the queue is *elementA*. **T**

3. Discuss the two approaches for the Empty QUEUE.

The two methods to check if the queue is empty are *isEmpty(),* and *size().* The *isEmpty()* method returns true if the queue is empty. The *size()* method returns the number of elements the queue has. It depends on whether the programmer wants a boolean or an Integer value returned.

8. Draw the internal representation of the queue q for each step of the following code sequence:

*ArrayBoundedQueue<String> q = new ArrayBoundedQueue<String>(5);*

*q.enqueue(“X”);*

*q.enqueue(“M”);*

*q.dequeue():*

*q.enqueue(“T”);*

Front: 0

Rear: 0

Elements: 1

Front: 1

Rear: 1

Elements: 1

Front: 0

Rear: 1

Elements: 2

0 1 2 3 4

T

X

Front: 0

Rear: 1

Elements: 2

Front: 0

Rear: 0

Elements: 1

M

0 1 2 3 4

M

0 1 2 3 4

0 1 2 3 4

M

X

X

0 1 2 3 4

19. Draw the internal representation of the queue q for each step of the following code sequence:

*LInkedQueue<String> q;*

*q = new LinkedQueue<String>();*

*q.enqueue(“X”);*

*q.enqueue(“M”);*

*q.dequeue();*

*q.enqueue(“T”);*

Front:

Rear:

Info: T

Link:

Info: M

Link:

Front:

Rear:

Info: M

Link:

Front:

Rear:

Info: M

Link:

Info: X

Link:

Front:

Rear:

Info: X

Link:

39. Complete the table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer | Arrival Time | Service Time | Finish Time | Wait Time |
| 1 | 0 | 10 | 10 | 0 |
| 2 | 8 | 3 | 13 | 2 |
| 3 | 8 | 10 | 23 | 5 |
| 4 | 9 | 40 | 63 | 14 |
| 5 | 20 | 15 | 78 | 43 |
| 6 | 32 | 18 | 96 | 46 |

One queue average waiting time: 18.33

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer | Arrival Time | Service Time | Finish Time | Wait Time |
| 1 | 0 | 10 | 10 | 0 |
| 2 | 8 | 3 | 11 | 0 |
| 3 | 8 | 10 | 20 | 2 |
| 4 | 9 | 40 | 51 | 2 |
| 5 | 20 | 15 | 35 | 0 |
| 6 | 32 | 18 | 69 | 19 |

Two queue average waiting time: 3.83

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Customer | Arrival Time | Service Time | Finish Time | Wait Time |
| 1 | 0 | 10 | 10 | 0 |
| 2 | 8 | 3 | 11 | 0 |
| 3 | 8 | 10 | 18 | 0 |
| 4 | 9 | 40 | 50 | 1 |
| 5 | 20 | 15 | 35 | 0 |
| 6 | 32 | 18 | 50 | 0 |

Three queue average waiting time: 0.16

43. Potential interference problems may cause the same ticket being sold to two different people. A solution could be to use different threads for each way for customers to buy a ticket, and make sure some sort of synchronization method is used to avoid selling the same ticket twice.