19CSE212 Data Structures & Algorithms

Labsheet 7

Queue, Circular Queue, Deque

---

1. Declare a class (QueueInt.java) for integer QueueInt with three attributes:

(a) An array 'arr' of size 5.

(b) Two variables front and rear initialized to -1.

Test cases:

(a) Write a separate test driver class (Test.java) and create an object instance of QueueInt

class.

QueueInt qi = new QueueInt ();

Compile both the .java files and run Test.java. Ensure no errors.

(b) Now try to access the 'front' and ‘rear’ attribute of QueueInt from Test.java directly.

System.out.println("Queue Front is”+qi.front+ ” and Rear is " + qi.rear);

Compile and execute.

2. Add a default constructor that will create the queue of some standard size (say 10) in case user does not give the size.

```
QueueInt () {
arr = new int[10];
front = -1;
rear=-1;
}
```

Test cases:

```
QueueInt qi = new QueueInt ();
System.out.println(qi.arr.length);
```

3. Add a parameterized constructor that will create the queue with specified size.

```
QueueInt (int sz) {
arr = new int[sz];
front = -1;
rear=-1;
}
```

Test cases:

```
QueueInt qi2 = new QueueInt (15);
System.out.println(qi2.arr.length);
```

4. Add print function to QueueInt .java that will print the contents of the queue. It should be public.

```
public void print() {
```

// Your logic here

// Scan through the array from front to rear and the print the values with

tab space in between.

}

5. Implement enqueue() method first without checking the array length limit.

public void enquue(int item) {

// Your logic here

}

Test cases:

(a) Write test file and try invoking enqueue operations and check.

QueueInt qi = new QueueInt ();

qi.enqueue(100);

qi.print();

qi.enqueue(200);

qi.print();

(b) Write test file that tries to enqueue beyond Queue capacity

QueueInt qi = new QueueInt ();

.....,.

qi.enqueue(900);

qi.print();

qi.enqueue(300);

qi.print();

The execution will abort as soon as the last enqueue is invoked.

ArrayIndexOutOfBoundsException.

6. Implement the check to enqueue() method. Don't add an item to the array if rear= =n-1. Instead print "can't enqueue" message.

public void enqueue(int item) {

// Your enhanced logic here

}

Test cases:

(a) Run test file. No exception will be thrown this time around.

7. Add a getter method getFront() which returns the front element in the queue.

public int getFront() {

// return arr[front];

}

Test cases:

(a) Invoke getFront() from test file and print the top.

```
System.out.println(qi2.getFront());
```

8. Now implement dequeue() method that removes the front item in the queue and returns it. First without lower bound checking logic.

```
public int dequeue() {
        // Your logic here
}
```

Test cases:

(a) Write Test5.java to enqueue and dequeue few items to check it's working.

```
int item = qi.dequeue();
qi.print();
```

(b) Write test file to dequeue more items than what were enqueued.

```
int item1 = qi.dequeue();
qi.print();
....
```

The last call to dequeue() should throw Array out of bounds exception.

9. Now implement the check for front=rear=-1 and print "can't pop" message. Run test file. Note that no exception will be thrown this time.

10. You can't check if contents of two queues are same by using = =.

Test cases:

(a) Write the test file.

```
QueueInt qi1 = new QueueInt ();
QueueInt qi2 = new QueueInt ();
qi1.enqueue(100);
qi2.enqueue(100);
qi1.enqueue(200);
qi2.enqueue(200);
if (qi1 == qi2)
System.out.println("Both qi1 and qi2 are same");
else
System.out.println("Both qi1 and qi2 are not the same");
```

Run it and check. It will print qi1 and qi2 are not same. Why?

Because = = operator will only compare 2 addresses. Then how to check the contents?

11. Implement equals() method which will first compare the elements of two queues. If not same, return false. If same, scan through arrays of both queues to check if each item is one queue is same as an item in another queue. If so, return true. Else return false.

```
public boolean equals(Queue another) {
        // Your logic here
}
```

Test cases:

(a) Now run test file. It will print qi1 and qi2 are same since the contents are same.

(b) Now write test file to do same number of enqueues but contents different.

```
QueueInt qi1 = new QueueInt (5);
QueueInt qi2 = new QueueInt (5);
qi1.enqueue(100);
qi2.enqueue(100);
qi1.enqueue(200);
qi2.enqueue(300);
if (code)
System.out.println("Both qi1 and qi2 are same");
else
System.out.println("Both qi1 and qi2 are not the same");
```

12. Implement a circular queue by following the procedure given in question from 1 to 11.

13. Implement the following methods in the above circular queue:

(a) splitq(), to split a queue into two queues so that all items in odd positions are in one queue and those in even positions are in another queue.

(b) getminElement () to return the minimum element in a queue.

14. Implement the following operations on Deque using a circular array.

insertFront(): Adds an item at the front of Deque.

insertLast(): Adds an item at the rear of Deque.

deleteFront(): Deletes an item from front of Deque.

deleteLast(): Deletes an item from rear of Deque.

getFront(): Gets the front item from queue.

getRear(): Gets the last item from queue.

isEmpty(): Checks whether Deque is empty or not.

isFull(): Checks whether Deque is full or not.

display(): Display queue elements starting from front to rear

Test case:

Create a queue of size 5

insetFront1(10):

insertLast(20):

insetFront(30):

deleteFront():

deleteLast():

insertLast(25):

insetFront(40):

insetFront(50):

getRear():

getFront():

15. You are given a stack data structure with push and pop operations. Implement a queue using instances of stack data structure and operations on it