Chapter 4

Queue

Queue

By: S. Hassan Adelyar

Queue

Queue

- A **queue** is a data structure that is somewhat like a stack, except that in a queue the **first item inserted** is the **first** to be **removed** (**First-In-First-Out,FIFO**), while in a **stack** the last item inserted is the first to be removed (**LIFO**).
- The **main rule** for queue is to **insert** and **delete** objects according to the **FIFO** principle.

Queue

- There are **various queues** quietly doing their job in your **computer's** (or the **network's**) **operating** system. There is **printer queue** where print jobs wait for the printer to be available.
- A queue also stores **keystroke** data as you type at type at the keyboard.
- To avoid moving objects once they are placed in Q, we define two variables **first** and **rear**, which has the following meanings:
- **first points** to the first element and **rear** points to the **last** element.
- **Initially first=rear=0**, which indicate that the Q is empty.

Queue

```
class Queue {
    private int maxSize;
    private long[] queArray;
    private int front;
    private int rear;
    private int nItems;
    public Queue(int s)  {
        maxSize = s;
        queArray = new long[maxSize];
        front = 0;
        rear = -1;
        nItems = 0;
    }
```

Queue

```
public void insert(long j) {
        if(rear == maxSize - 1)
            rear = -1;
        queArray[++rear] = j;
        nItems++;
    }
public long remove() {
        long temp = queArray[front++];
        if(front == maxSize)
            front = 0;
        nItems--;
        return temp;
    }
```

Queue

```
public long peekFront()  {
    return queArray[front];
}
public boolean isEmpty()  {
    return (nItems==0);
}
public boolean isFull()  {
    return (nItems==maxSize);
}
public int Size() {
    return nItems;
}
```

Queue

```
public static void main(String[] args)  {
    Queue theQueue = new Queue(5);
    theQueue.insert(10);
    theQueue.insert(20);
    theQueue.insert(30);
    theQueue.insert(40);
    theQueue.remove();
    theQueue.remove();
    theQueue.remove();
    theQueue.insert(50);
    theQueue.insert(60);
    theQueue.insert(70);
    theQueue.insert(80);
```

Queue

```
        while( !theQueue.isEmpty() )
        {
           long n = theQueue.remove();
           System.out.print(n);
           System.out.print(" ");
        }
        System.out.println(" ");
     }

  }
```

Priority queues

Queue

- A priority queue is a more **specialized** data structure than a stack or queue. Like an ordinary queue, a priority queue has a **front** and a **rear**, and items are **removed** from the **front**.
- However, in a priority queue, **items** are **ordered** by **key** value so that the item with the **lowest key** (or in some implementations the **highest key**) is always at the **front**. Items are inserted in the proper position to maintain the order.
- Like ordinary queues, priority queues are used in various ways in certain computer systems. In a **preemptive multitasking operating system** for example, programs may be placed in a priority queue so the highest-priority program is the next one to receive a **time-slice** that allow it to execute.

Priority Queue Example

Queue

```
class PriorityQ {
    private int maxSize;
    private long[] queArray;
    private int nItems;

    public PriorityQ(int s)  {
        maxSize = s;
        queArray = new long[maxSize];
        nItems = 0;
    }
```

Queue

```
public void insert(long item) {
        int j;
        if(nItems == 0)
            queArray[nItems++] = item;
        else  {
            for(j=nItems-1; j>=0; j--) {
                if(item > queArray[j] )
                    queArray[j+1] = queArray[j];
                else
                    break;
            }
            queArray[j+1] = item;
            nItems++;
        }
    }
```

Queue

```
public long remove() {
        return queArray[--nItems];
    }
 public long peekMin() {
        return queArray[nItems - 1];
    }
 public boolean isEmpty()   {
        return (nItems == 0);
    }
public boolean isFull() {
        return (nItems == maxSize);
    }
}
```

Queue

```
class PriorityQApp {
public static void main(String[] args) {
    PriorityQ thePQ = new PriorityQ(5);
    thePQ.insert(30);
    thePQ.insert(50);
    thePQ.insert(10);
    thePQ.insert(40);
    thePQ.insert(20);
    while( !thePQ.isEmpty()) {
        long item = thePQ.remove();
        System.out.print(item + " ");
    }
    System.out.println(" ");
  }
 }
```