Datastrucutres through Java Lab manual is prepared according to the JNTUH Syllabus for M.Tech 1st Year 1st Semester CSE branch

LAB MANUAL

Datastructures Through JAVA

By

www.btechsmartclass.com

1. Write Java programs that use both recursive and non-recursive functions for implementing the following searching methods:
(a) Linear search
(b) Binary search

Linear search

(a): Iterative Linear search

```
class LinearSearchDemo {
     static Object[] a = { 89, 45, 175, 7, 50, 43, 126, 90 };
     static Object key = 126;
     public static void main(String args[])
     {
         if( linearSearch() )
             System.out.println(key + " found in the list");
         else
             System.out.println(key + " not found in the list");
     }
     static boolean linearSearch()
     {
        for( int i=0; i<a.length; i++)
           if(key == a[i])
               return true;
        return false;
     }
}
```

(b): Recursive Linear search

```
class RecursiveLinearSearchDemo{
     static Object[] a = { 89, 45, 175, 7, 50, 43, 126, 90 };
     static Object key = 43;
     public static void main(String args[])
     {
          if( linearSearch(a.length-1) )
              System.out.println(key + " found in the list");
          else
              System.out.println(key + " not found in the list");
     }
     static boolean linearSearch(int n)
     {
          if( n < 0 )
              return false;
          if(key == a[n])
              return true;
          else
              return linearSearch(n-1);
     }
}
```

Binary search

Iterative Binary search

```
class BinarySearchDemo {
      static Object[] a = { "AP", "KA", "MH", "MP", "OR", "TN", "UP", "WB"};
      static Object key = "UP";
      public static void main(String args[])
      {
            if( binarySearch() )
            System.out.println(key + " found in the list");
            else
            System.out.println(key + " not found in the list");
      }
      static boolean binarySearch()
      {
            int c, mid, low = 0, high = a.length-1;
            while( low <= high)
            {
               mid = (low + high)/2;

               c = ((Comparable)key).compareTo(a[mid]);

               if( c < 0) high = mid-1;
               else if( c > 0) low = mid+1;
               else return true;
            }
            return false;
      }
}
```

Recursive Binary search

```
class RecursiveBinarySearchDemo {
      static Object[] a = { "AP", "KA", "MH", "MP", "OR", "TN", "UP", "WB"};
      static Object key = "XP";
      public static void main(String args[]){
            if( binarySearch(0, a.length-1) )
                  System.out.println(key + " found in the list");
            else
                  System.out.println(key + " not found in the list");
      }
      static boolean binarySearch(int low, int high){
            if( low > high ) return false;

            int mid = (low + high)/2;

            int c = ((Comparable)key).compareTo(a[mid]);
            if( c < 0) return binarySearch(low, mid-1);
            else if( c > 0) return binarySearch(mid+1, high);
            else return true;
      }
}
```

2. Write Java programs to implement the List ADT using arrays and linked lists.

Array Implementation of List

```
public interface List {
        public void createList(int n);
        public void insertFirst(Object ob);
        public void insertAfterbject ob, Object pos);
        public Object deleteFirst();
        public Object deleteAfter(Object pos);
        public boolean isEmpty();
        public int size();
}
class ArrayList implements List {
        class Node {
                Object data;
                int next;
                Node(Object ob, int i) // constructor
                {
                        data = ob;
                        next = i;
                }
        }
        int MAXSIZE; // max number of nodes in the list
        Node list[]; // create list array
        int head, count; // count: current number of nodes in the list
        ArrayList( int s) // constructor
        {
                MAXSIZE = s;
                list = new Node[MAXSIZE];
        }
    public void initializeList() {
        for( int p = 0; p < MAXSIZE-1; p++ )
                list[p] = new Node(null, p+1);
        list[MAXSIZE-1] = new Node(null, -1);
  }
  public void createList(int n) // create 'n' nodes
  {
        int p;
        for( p = 0; p < n; p++ )
        {
          list[p] = new Node(11+11*p, p+1);
          count++;
        }
        list[p-1].next = -1; // end of the list
  }
```

```
public void insertFirst(Object item)   {
        if( count == MAXSIZE ) {
          System.out.println("***List is FULL");
          return;
        }
        int p = getNode();
        if( p != -1 ) {
          list[p].data = item;
          if( isEmpty() ) list[p].next = -1;
          else list[p].next = head;
          head = p;
          count++;
        }
}
public void insertAfter(Object item, Object x)    {
        if( count == MAXSIZE ) {
          System.out.println("***List is FULL");
          return;
        }
        int q = getNode(); // get the available position to insert new node
        int p = find(x); // get the index (position) of the Object x
        if( q != -1 ) {
          list[q].data = item;
          list[q].next = list[p].next;
          list[p].next = q;
          count++;
        }
}
public int getNode() // returns available node index
{
        for( int p = 0; p < MAXSIZE; p++ )
          if(list[p].data == null) return p;
        return -1;
}
public int find(Object ob) // find the index (position) of the Object ob
{
        int p = head;
        while( p != -1) {
          if( list[p].data == ob ) return p;
          p = list[p].next; // advance to next node
        }
        return -1;
}
```

```
  public Object deleteFirst()  {
        if( isEmpty() ) {
          System.out.println("List is empty: no deletion");
          return null;
        }
        Object tmp = list[head].data;
        if( list[head].next == -1 ) // if the list contains one node,
          head = -1; // make list empty.
        else
          head = list[head].next;
        count--; // update count
        return tmp;
  }
  public Object deleteAfter(Object x)   {
        int p = find(x);
        if( p == -1 || list[p].next == -1 ) {
          System.out.println("No deletion");
          return null;
        }
        int q = list[p].next;
        Object tmp = list[q].data;
        list[p].next = list[q].next;
        count--;
        return tmp;
  }
  public void display() {
        int p = head;
        System.out.print("\nList: [ " );
        while( p != -1) {
          System.out.print(list[p].data + " "); // print data
          p = list[p].next; // advance to next node
        }
        System.out.println("]\n");//
  }
  public boolean isEmpty()  {
        if(count == 0) return true;
        else return false;
  }
  public int size() {
        return count;
  }
}
```

```
class ArrayListDemo {
  public static void main(String[] args) {
        ArrayList linkedList = new ArrayList(10);
        linkedList.initializeList();
        linkedList.createList(4); // create 4 nodes
        linkedList.display(); // print the list
        System.out.print("InsertFirst 55:");
        linkedList.insertFirst(55);
        linkedList.display();
        System.out.print("Insert 66 after 33:");
        linkedList.insertAfter(66, 33); // insert 66 after 33
        linkedList.display();
        Object item = linkedList.deleteFirst(); System.out.println("Deleted node: " + item);
        linkedList.display();
        System.out.print("InsertFirst 77:");
        linkedList.insertFirst(77);
        linkedList.display();
        item = linkedList.deleteAfter(22); // delete node after node 22
        System.out.println("Deleted node: " + item);
        linkedList.display();
        System.out.println("size(): " + linkedList.size());
  }
}
```

Output:

```
List: [ 11 22 33 44 ]
InsertFirst 55:
List: [ 55 11 22 33 44 ]
Insert 66 after 33:
List: [ 55 11 22 33 66 44 ]
Deleted node: 55
List: [ 11 22 33 66 44 ]
InsertFirst 77:
List: [ 77 11 22 33 66 44 ]
Deleted node: 33
List: [ 77 11 22 66 44 ]
size(): 5
```

Linked Implementation of List

```
class LinkedList implements List {
   class Node {
     Object data; // data item
     Node next; // refers to next node in the list
     Node( Object d ) // constructor
     { data = d; } // 'next' is automatically set to null
   }
Node head; // head refers to first node
Node p; // p refers to current node
int count; // current number of nodes
public void createList(int n) // create 'n' nodes
{
   p = new Node(11); // create first node
   head = p; // assign mem. address of 'p' to 'head'
   for( int i = 1; i < n; i++ ) // create 'n-1' nodes
     p = p.next = new Node(11 + 11*i);
     count = n;
}
public void insertFirst(Object item) // insert at the beginning of list
{
   p = new Node(item); // create new node
   p.next = head; // new node refers to old head
   head = p; // new head refers to new node
   count++;
}
public void insertAfter(Object item,Object key){
   p = find(key); // get "location of key item"
   if( p == null )
     System.out.println(key + " key is not found");
   else{
     Node q = new Node(item); // create new node
     q.next = p.next; // new node next refers to p.next
     p.next = q; // p.next refers to new node
     count++;
   }
}
public Node find(Object key){
   p = head;
   while( p != null ) // start at beginning of list until end of list
   {
     if( p.data == key ) return p;
     p = p.next; // move to next node
   }
   return null; // if key search is unsuccessful, return null
}
```

```
public Object deleteFirst(){  // delete first node
    if( isEmpty() ){
      System.out.println("List is empty: no deletion");
      return null;
    }
    Node tmp = head; // tmp saves reference to head
    head = tmp.next;
    count--;
    return tmp.data;
}

public Object deleteAfter(Object key) // delete node after key item
{
    p = find(key); // p = "location of key node"
    if( p == null ){
      System.out.println(key + " key is not found");
      return null;
    }
    if( p.next == null ) // if(there is no node after key node)
    {
      System.out.println("No deletion");
      return null;
    }
    else{
      Node tmp = p.next; // save node after key node
      p.next = tmp.next; // point to next of node deleted
      count--;
      return tmp.data; // return deleted node
    }
}
public void displayList() {
    p = head; // assign mem. address of 'head' to 'p'
    System.out.print("\nLinked List: ");
    while( p != null ) // start at beginning of list until end of list
    {
      System.out.print(p.data + " -> "); // print data
      p = p.next; // move to next node
    }
    System.out.println(p); // prints 'null'
}
public boolean isEmpty() // true if list is empty
{
    return (head == null);
}
public int size() {
    return count;
}
```

} // end of LinkeList class

```
class LinkedListDemo {
   public static void main(String[] args){
     LinkedList list = new LinkedList(); // create list object
     list.createList(4); // create 4 nodes
     list.displayList();
     list.insertFirst(55); // insert 55 as first node
     list.displayList();
     list.insertAfter(66, 33); // insert 66 after 33
     list.displayList();
     Object item = list.deleteFirst(); // delete first node
     if( item != null ){
        System.out.println("deleteFirst(): " + item);
        list.displayList();
     }
     item = list.deleteAfter(22); // delete a node after node(22)
     if( item != null ){
        System.out.println("deleteAfter(22): " + item);
        list.displayList();
     }
     System.out.println("size(): " + list.size());
   }
}
```

Output:

```
Linked List: 11 -> 22 -> 33 -> 44 -> null
Linked List: 55 -> 11 -> 22 -> 33 -> 44 -> null
Linked List: 55 -> 11 -> 22 -> 33 -> 66 -> 44 -> null
deleteFirst(): 55
Linked List: 11 -> 22 -> 33 -> 66 -> 44 -> null
deleteAfter(22): 33
Linked List: 11 -> 22 -> 66 -> 44 -> null
size(): 4
```

3. Write Java programs to implement the following using an array.

(a) Stack ADT

(b) Queue ADT

```
public interfaceStack {
   public void push(Object ob);
   public Object pop();
   public Object peek();
   public boolean isEmpty();
   public int size();
}

public class ArrayStack implements Stack {
   private Object a[];
   private int top; // stack top
   public ArrayStack(int n) // constructor
   {
     a = new Object[n]; // create stack array
     top = -1; // no items in the stack
   }
   public void push(Object item) // add an item on top of stack
   {
     if(top == a.length-1){
        System.out.println("Stack is full");
        return;
     }
     top++; // increment top
     a[top] = item; // insert an item
   }
   public Object pop() // remove an item from top of stack
   {
     if( isEmpty() ){
        System.out.println("Stack is empty");
        return null;
     }
     Object item = a[top]; // access top item
     top--; // decrement top
     return item;
   }
   public Object peek(){ // get top item of stack
     if( isEmpty() ) return null;
     return a[top];
   }
   public boolean isEmpty(){ // true if stack is empty
     return (top == -1);
   }
   public int size(){ // returns number of items in the stack
     return top+1;
```

```
    }
}


class ArrayStackDemo {
   public static void main(String[] args){
     ArrayStack stk = new ArrayStack(4); // create stack of size 4
     Object item;
     stk.push('A'); // push 3 items onto stack
     stk.push('B');
     stk.push('C');
     System.out.println("size(): "+ stk.size());
     item = stk.pop(); // delete item
     System.out.println(item + " is deleted");
     stk.push('D'); // add three more items to the stack
     stk.push('E');
     stk.push('F');
     System.out.println(stk.pop() + " is deleted");
     stk.push('G'); // push one item
     item = stk.peek(); // get top item from the stack
     System.out.println(item + " is on top of stack");
   }
}
```

Output:

```
size(): 3
C is deleted
Stack is full
E is deleted
G is on top of stack
```

Queue ADT

```
public interface Queue {
   public void insert(Object ob);
   public Object remove();
   public Object peek();
   public boolean isEmpty();
   public int size();
}
class ArrayQueue implements Queue {
   private int maxSize; // maximum queue size
   private Object[] que; // que is an array
   private int front;
   private int rear;
   private int count; // count of items in queue (queue size)
   public ArrayQueue(int s) { // constructor
      maxSize = s;
      que = new Object[maxSize];
      front = rear = -1;
      count = 0;
   }
   public void insert(Object item) // add item at rear of queue
   {
      if( count == maxSize ){
         System.out.println("Queue is Full"); return;
      }
      if(rear == maxSize-1 || rear == -1) {
         que[0] = item;
         rear = 0;
         if( front == -1) front = 0;
      }
      else
         que[++rear] = item;
      count++; // update queue size
   }
   public Object remove(){ // delete item from front of queue
      if( isEmpty() ){
         System.out.println("Queue is Empty"); return 0;
      }
      Object tmp = que[front]; // save item to be deleted
      que[front] = null; // make deleted item's cell empty
      if( front == rear )
         rear = front = -1;
      else if( front == maxSize-1 )
         front = 0;
      else
         front++;
      count--; // less one item from the queue size
```

```
        return tmp;
    }

    public Object peek() // peek at front of the queue
    {
        return que[front];
    }
    public boolean isEmpty() // true if the queue is empty
    {
        return (count == 0);
    }
    public int size() // current number of items in the queue
    {
        return count;
    }
    public void displayAll(){
        System.out.print("Queue: ");
        for( int i = 0; i < maxSize; i++ )
            System.out.print( que[i] + " ");
        System.out.println();
    }
}

class QueueDemo {
    public static void main(String[] args){
        /* queue holds a max of 5 items */
        ArrayQueue q = new ArrayQueue(5);
        Object item;
        q.insert('A'); q.insert('B'); q.insert('C'); q.displayAll();
        item = q.remove(); // delete item
        System.out.println(item + " is deleted");
        item = q.remove();
        System.out.println(item + " is deleted");
        q.displayAll();
        q.insert('D'); // insert 3 more items
        q.insert('E');
        q.insert('F');
        q.displayAll();
        item = q.remove();
        System.out.println(item + " is deleted");
        q.displayAll();
        System.out.println("peek(): " + q.peek());
        q.insert('G');
        q.displayAll();
        System.out.println("Queue size: " + q.size());
    }
}
```

Output:

```
Queue: A B C null null
A is deleted
B is deleted
Queue: null null C null null
Queue: F null C D E
C is deleted
Queue: F null null D E
peek(): D
Queue: F G null D E
Queue size: 4
```

4. Write a java program that reads an infix expression, converts the expression to postfix form and then evaluates the postfix expression (use stack ADT).

```
class InfixToPostfix {
   java.util.Stack<Character> stk = new java.util.Stack<Character>();
   public String toPostfix(String infix){
      infix = "(" + infix + ")"; // enclose infix expr within parentheses
      String postfix = "";
      /* scan the infix char-by-char until end of string is reached */
      for( int i=0; i<infix.length(); i++){
         char ch, item;
         ch = infix.charAt(i);
         if( isOperand(ch) ) // if(ch is an operand), then
         postfix = postfix + ch; // append ch to postfix string
         if( ch == '(' ) // if(ch is a left-bracket), then
            stk.push(ch); // push onto the stack
         if( isOperator(ch) ) // if(ch is an operator), then
         {
            item = stk.pop(); // pop an item from the stack
            /* if(item is an operator), then check the precedence of ch and item*/
            if( isOperator(item) ) {
               if( precedence(item) >= precedence(ch) ){
                  stk.push(item);
                  stk.push(ch);
               }
               else{
                  postfix = postfix + item;
                  stk.push(ch);
               }
            }
            else {
               stk.push(item);
               stk.push(ch);
            }
         } // end of if(isOperator(ch))
         if( ch == ')' ) {
            item = stk.pop();
            while( item != '(' ){
               postfix = postfix + item;
               item = stk.pop();
            }
         }
      } // end of for-loop
      return postfix;
   } // end of toPostfix() method
```

```
   public boolean isOperand(char c) {
      return(c >= 'A' && c <= 'Z');
   }
   public boolean isOperator(char c){
      return( c=='+' || c=='-' || c=='*' || c=='/' );
   }

   public int precedence(char c) {
      int rank = 1; // rank = 1 for '*' or '/'
      if( c == '+' || c == '-' ) rank = 2;
         return rank;
   }
}

class InfixToPostfixDemo
{
   public static void main(String args[]){
      InfixToPostfix obj = new InfixToPostfix();
      String infix = "A*(B+C/D)-E";
      System.out.println("infix: " + infix );
      System.out.println("postfix:"+obj.toPostfix(infix));
    }
}
```

Output:

```
infix: A*(B+C/D)-E
postfix: ABCD/+*E-
```

6. Write a Java program that uses both stack and queue to test whether the given string is a palindrome.

(a): Testing whether the given string is a palindrome using stack

```
import java.util.Stack;
class Palindrome {
   public static void main(String args[]) {
      String str = "MALAYALAM";
      if( isPalindrome(str) )
         System.out.println( str + " is a Palindrome");
      else
         System.out.println( str + " is not a Palindrome");
   }
   static boolean isPalindrome(String str){
      Stack<Character> stk = new Stack<Character>();
      for( int i=0; i < str.length(); i++ )
         stk.push(str.charAt(i));
      for( int i=0; i < str.length()/2; i++ )
         if( str.charAt(i) != stk.pop() )
            return false;
      return true;
   }
}
```

(b): Testing whether the given string is a palindrome using queue

```
import java.util.LinkedList;
class Palindrome {
   public static void main(String args[]) {
      String str = "RADAR";
      if( isPalindrome(str) )
         System.out.println( str + " is a Palindrome");
      else
         System.out.println( str + " is not a Palindrome");
   }
   static boolean isPalindrome(String str){
      LinkedList<Character> que = new LinkedList<Character>();
      int n = str.length();

      for( int i=0; i < n; i++ )
         que.addLast(str.charAt(i));
      for( int i=n-1; i > n/2; i-- )
         if( str.charAt(i) != que.removeFirst())
           return false;
      return true;
   }
}
```

7. Write Java programs to implement the following using a singly linked list.

(a) Stack ADT
(b) Queue ADT

Program 21(a): Linked Implementation of a Stack

```
class Node {
   int data; // data item
   Node next; // next node in linked-stack
   Node( int d ){ // constructor
      data = d;
   } // next is automatically set to null
}
class LinkedStack {
   Node top; // top refers to top-node
   Node p; // p refers to current node
   public void push(int item){ // add item onto stack
      p = new Node(item); // create new node
      p.next = top; // new node refers to old top
      top = p; // top refers to new node
   }
   public Node pop() { // remove a node from the stack
      if( isEmpty() ) {
         System.out.println("Stack is empty");
         return null;
      }
      Node tmp = top; // tmp saves reference to top node
      top = tmp.next; // now, top refers to next node of old top

      return tmp; // return the popped item
   }
   public Node peek(){ // get top node from the stack, without deleting
      if( isEmpty() ) {
         System.out.println("Stack is empty");
         return null;
      }
      return top;
   }
   public void displayStack() {
      p = top; // p refers to top
      System.out.print("\nContents of Stack: [ ");
      while( p != null ) { // start printing from top of stack to bottom of stack
         System.out.print(p.data + " "); // print data
         p = p.next; // move to next node
      }
      System.out.println("]");
   }
```

```
   public boolean isEmpty(){  // true if stack is empty
       return (top == null);
   }
}
```

```
class LinkedStackDemo {
   public static void main(String[] args) {
      LinkedStack stk = new LinkedStack(); // create stack object
      Node item; // item stores popped node
      stk.push(20); // add 20, 35, 40 to stack
      stk.push(35);
      stk.push(40);
      stk.displayStack(); // print contents of stack
      item = stk.pop(); // remove a node from the top and print it
      if( item != null ) {
         System.out.println("Popped item: " + item.data);
         stk.displayStack();
      }
      stk.push(65); // insert 65, 70, 75
      stk.push(70);
      stk.push(75);
      stk.displayStack(); // display contents of stack
      item = stk.pop(); // remove a node from the top and display it
      if( item != null ){
         System.out.println("Popped item: " + item.data);
         stk.displayStack();
      }
      System.out.println("peek(): " + stk.peek());// get top item

      stk.push(90); // insert 90
      stk.displayStack();
   }
}
```

Output:

```
Contents of Stack: [ 40 35 20 ]
Popped item: 40
Contents of Stack: [ 35 20 ]
Contents of Stack: [ 75 70 65 35 20 ]
Popped item: 75
peek(): 70
Contents of Stack: [ 70 65 35 20 ]
Contents of Stack: [ 90 70 65 35 20 ]
```

(b): A LinkedQueue Class

```
public class LinkedQueue {
   class Node {
      Object data;
      Node next;
      Node(Object item) // constructor
      {
         data = item;
      }
   }
   Node front, rear;
   int count;
   public void insert(Object item) {
      Node p = new Node(item);
      if(front == null) // queue is empty; insert first item
      {
         front = rear = p;
         rear.next = null;
      }
      if(front == rear) // queue contains one item; insert second item
      {
         rear = p;
         front.next = rear;
         rear.next = null;
      }

      else { // queue contains 2 or more items
         rear.next = p; // old rear.next refers to p
         rear = p; // new rear refers to p
         rear.next = null;
      }
      count++; // increment queue size
   }
   public Object remove() {
      if(isEmpty()) {
         System.out.println("Q is empty"); return null;
      }
      Object item = front.data;
      front = front.next;
      count--; // decrement queue size
      return item;
   }
   public boolean isEmpty() {
       return (front == null);
   }
   public Object peek() {
       return front.data;
   }
```

```
   public int size() {
      return count;
   }
   public void display(){
      Node p = front;
      System.out.print("Linked Q: ");
      if(p == null) System.out.println("empty");
      while( p != null ){
         System.out.print(p.data + " ");
         p = p.next;
      }
      System.out.println();
   }
}

class LinkedQueueDemo {
   public static void main(String[] args) {
      LinkedQueue q = new LinkedQueue();
      q.display();
      q.insert('A');
      q.insert('B');
      q.insert('C');
      q.insert('D');
      q.display();
      System.out.println("delete(): " + q.remove());
      q.display();
      System.out.println("peek(): " + q.peek());
      q.insert('E');
      q.insert('F');
      System.out.println("delete(): " + q.remove());
      q.display();
      System.out.println("size(): " + q.size());
   }
}
```

Output:

```
Linked Q: empty
Linked Q: A B C D
remove(): A
Linked Q: B C D
peek(): B
remove(): B
Linked Q: C D E F
size(): 4
```

8. Write Java programs to implement the deque (double ended queue) ADT using
(a) Array
(b) Doubly linked list.

(a): An ArrayDeque Class

```
public class ArrayDeque {
   private int maxSize;
   private Object[] que;
   private int first;
   private int last;
   private int count; // current number of items in deque
   public ArrayDeque(int s) {   // constructor
      maxSize = s;
      que = new Object[maxSize];
      first = last = -1;
      count = 0;
   }
   public void addLast(Object item) {
      if(count == maxSize) {
         System.out.println("Deque is full"); return;
      }
      last = (last+1) % maxSize;
      que[last] = item;
      if(first == -1 && last == 0) first = 0;
      count++;
   }
   public Object removeLast() {
      if(count == 0) {
         System.out.println("Deque is empty"); return(' ');
      }
      Object item = que[last];
      que[last] = ' ';
      if(last > 0)
         last = (last-1) % maxSize;
      count--;
      if(count == 0)
         first = last = -1;
      return(item);
   }
   public void addFirst(Object item){
      if(count == maxSize){
         System.out.println("Deque is full"); return;
      }
      if(first > 0)
         first = (first-1) % maxSize;
      else if(first == 0)
         first = maxSize-1;
      que[first] = item;
      count++;
   }
```

```
   public Object removeFirst() {
      if(count == 0) {
         System.out.println("Deque is empty");
         return(' ');
      }
      Object item = que[first];
      que[first] = ' ';
      if(first == maxSize-1)
         first = 0;
      else
         first = (first+1) % maxSize;
      count--;
      if(count == 0)
         first = last = -1;
      return(item);
   }
   void display() {
      System.out.println("----------------------------");
      System.out.print("first:"+first + ", last:"+ last);
      System.out.println(", count: " + count);
      System.out.println(" 0 1 2 3 4 5");
      System.out.print("Deque: ");
      for( int i=0; i<maxSize; i++ )
         System.out.print(que[i]+ " ");
      System.out.println("\n----------------------------");
   }
   public boolean isEmpty(){ // true if queue is empty
      return (count == 0);
   }
   public boolean isFull(){ // true if queue is full
      return (count == maxSize);
   }
}
class ArrayDequeDemo {
   public static void main(String[] args) {
      ArrayDeque q = new ArrayDeque(6); // queue holds a max of 6 items
      q.insertLast('A'); /* (a) */
      q.insertLast('B');
      q.insertLast('C');
      q.insertLast('D');
      System.out.println("deleteFirst():"+q.deleteFirst());
      q.display();
      q.insertLast('E'); /* (b) */
      q.display(); /* (c) */
      System.out.println("deleteLast():"+q.deleteLast());
      System.out.println("deleteLast():"+q.deleteLast());
      q.display();
      q.insertFirst('P'); q.insertFirst('Q'); /* (d) */
      q.insertFirst('R'); q.display();
```

```
        q.deleteFirst(); q.display(); /* (e) */
        q.insertFirst('X'); q.display(); /* (f) */
        q.insertLast('Y'); q.display(); /* (g) */
        q.insertLast('Z'); q.display(); /* (h) */
    }
}
```

Output:

```
deleteFirst(): A
----------------------------
first:1, last:3, count: 3
0 1 2 3 4 5
Deque: B C D
----------------------------
first:1, last:4, count: 4
0 1 2 3 4 5
Deque: B C D E
----------------------------
deleteLast(): E
deleteLast(): D
----------------------------
first:1, last:2, count: 2
0 1 2 3 4 5
Deque: B C
----------------------------
first:4, last:2, count: 5
0 1 2 3 4 5
Deque: P B C R Q
----------------------------
first:5, last:2, count: 4
0 1 2 3 4 5
Deque: P B C Q
----------------------------
first:4, last:2, count: 5
0 1 2 3 4 5
Deque: P B C X Q
----------------------------
first:4, last:3, count: 6
0 1 2 3 4 5
Deque: P B C Y X Q
----------------------------
Deque is full
----------------------------
first:4, last:3, count: 6
0 1 2 3 4 5
Deque: P B C Y X Q
----------------------------
```

(b): A LinkedDeque class

```
public class LinkedDeque {
   public class DequeNode {
      DequeNode prev;
      Object data;
      DequeNode next;
       DequeNode( Object item ) // constructor
       {
          data = item;
       } // prev & next automatically refer to null
    }
    private DequeNode first, last;
    private int count;
    public void addFirst(Object item){
       if( isEmpty() )
          first = last = new DequeNode(item);
       else {
          DequeNode tmp = new DequeNode(item);
          tmp.next = first;
          first.prev = tmp;
          first = tmp;
       }
       count++;
    }
    public void addLast(Object item){
       if( isEmpty() )
          first = last = new DequeNode(item);
       else{
          DequeNode tmp = new DequeNode(item);
          tmp.prev = last;
          last.next = tmp;
          last = tmp;
       }
       count++;
    }
    public Object removeFirst(){
       if( isEmpty() ){
          System.out.println("Deque is empty");
          return null;
       }
       else {
          Object item = first.data;
          first = first.next;
          first.prev = null;
          count--;
          return item;
       }
    }
```

```
   public Object removeLast() {
      if( isEmpty() ){
         System.out.println("Deque is empty");
         return null;
      }
      else {
         Object item = last.data;
         last = last.prev;
         last.next = null;
         count--;
         return item;
      }
   }
   public Object getFirst(){
   if( !isEmpty() )
      return( first.data );
   else
      return null;
   }
   public Object getLast(){
      if( !isEmpty() )
         return( last.data );
      else
         return null;
   }
   public boolean isEmpty(){
      return (count == 0);
   }
   public int size(){
      return(count);
   }
   public void display(){
      DequeNode p = first;
      System.out.print("Deque: [ ");
      while( p != null ){
         System.out.print( p.data + " " );
         p = p.next;
      }
      System.out.println("]");
    }
}
```

```
public class LinkedDequeDemo {
   public static void main( String args[]){
      LinkedDeque dq = new LinkedDeque();
      System.out.println("removeFirst():" + dq.removeFirst());
      dq.addFirst('A');
      dq.addFirst('B');
      dq.addFirst('C');
      dq.display();
      dq.addLast('D');
      dq.addLast('E');
      System.out.println("getFirst():" + dq.getFirst());
      System.out.println("getLast():" + dq.getLast());
      dq.display();
      System.out.println("removeFirst():"+dq.removeFirst());
      System.out.println("removeLast():"+ dq.removeLast());
      dq.display();
      System.out.println("size():" + dq.size());
   }
}
```

Output:

```
Deque is empty
removeFirst(): null
Deque: [ C B A ]
getFirst(): C
getLast(): E
Deque: [ C B A D E ]
removeFirst(): C
removeLast(): E
Deque: [ B A D ]
size(): 3
```

9. Write a Java program to implement a priority queue ADT.

```
public class Node {
   String data; // data item
   int prn; // priority number (minimum has highest priority)
   Node next; // "next" refers to the next node
   Node( String str, int p ) { // constructor
      data = str;
      prn = p;
   } // "next" is automatically set to null
}

class LinkedPriorityQueue {
   Node head; // "head" refers to first node
   public void insert(String item, int pkey){ // insert item after pkey
      Node newNode = new Node(item, pkey); // create new node
      int k;
      if( head == null )
         k = 1;
      else if( newNode.prn < head.prn )
         k = 2;
      else
         k = 3;
      switch( k ){
         case 1: head = newNode; // Q is empty, add head node
                 head.next = null;
                 break;
         case 2: Node oldHead = head; // add one item before head
                 head = newNode;
                 newNode.next = oldHead;
                 break;
         case 3: Node p = head; // add item before a node
                 Node prev = p;
                 Node nodeBefore = null;
                 while( p != null ) {
                   if( newNode.prn < p.prn ){
                      nodeBefore = p; break;
                   }
                   else {
                      prev = p; // save previous node of current node
                      p = p.next; // move to next node
                   }
                 } // end of while
                 newNode.next = nodeBefore;
                prev.next = newNode;
      } // end of switch
   } // end of insert() method
```

```
public Node delete(){
   if( isEmpty() ){
      System.out.println("Queue is empty");
      return null;
   }
   else {
      Node tmp = head;
      head = head.next;
      return tmp;
   }
}
public void displayList() {
   Node p = head; // assign address of head to p
   System.out.print("\nQueue: ");
   while( p != null ) // start at beginning of list until end of list
   {
      System.out.print(p.data+"(" +p.prn+ ")" + " ");
      p = p.next; // move to next node
   }
   System.out.println();
}
public boolean isEmpty() // true if list is empty
{
    return (head == null);
}
public Node peek() // get first item
{
     return head;
}
```

```
class LinkedPriorityQueueDemo {
   public static void main(String[] args){
      LinkedPriorityQueue pq = new LinkedPriorityQueue();
      Node item;
      pq.insert("Babu", 3);
      pq.insert("Nitin", 2);
      pq.insert("Laxmi", 2);
      pq.insert("Kim", 1);
      pq.insert("Jimmy", 3);
      pq.displayList();
      item = pq.delete();
      if( item != null )
         System.out.println("delete():" + item.data + "(" +item.prn+")");
      pq.displayList();
      pq.insert("Scot", 2);
      pq.insert("Anu", 1);
      pq.insert("Lehar", 4);
      pq.displayList();
   }
}
```

Output:

```
Queue: Kim(1) Nitin(2) Laxmi(2) Babu(3) Jimmy(3)
delete(): Kim(1)
Queue: Nitin(2) Laxmi(2) Babu(3) Jimmy(3)
Queue: Anu(1) Nitin(2) Laxmi(2) Scot(2) Babu(3) Jimmy(3) Lehar(4)
```