

Stack

```
3 public class StackX {
4
5     private int maxSize;
6     private double[] stackArray;
7     private int top;
8
9     public StackX(int s) {
10
11         maxSize = s;
12         stackArray = new double[maxSize];
13         top = -1;
14     }
15
16     public void push(double j) {
17
18         if (top == (maxSize - 1)) {
19             System.out.println("Queue is Full");
20         }
21         else {
22             stackArray[++top] = j;
23         }
24     }
25
26     public double pop() {
27
28         if(top == -1) {
29             System.out.println("Queue is Empty");
30             return -99;
31         }
32         else {
33             return stackArray[top--];
34         }
35     }
36 }
```

```
public double peek() {  
    if(top == -1) {  
        System.out.println("Queue is Empty");  
        return -99;  
    }  
    else {  
        return stackArray[top];  
    }  
}  
  
public boolean isEmpty() {  
    if(top == -1) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}  
  
public boolean isFull() {  
    if (top == (maxSize - 1)) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Queue – Linear Queue

```
public class QueueX {  
  
    private int maxSize;  
    private int[] queueArray;  
    private int front;  
    private int rear;  
    private int noOfItems;  
  
    public QueueX(int s) {  
  
        maxSize = s;  
        queueArray = new int[maxSize];  
        front = 0;  
        rear = -1;  
        noOfItems = 0;  
    }  
  
    public void insert(int j) {  
  
        if(noOfItems == (maxSize - 1)) {  
            System.out.println("Queue is Full");  
        }  
        else {  
            queueArray[++rear] = j;  
            noOfItems++;  
        }  
    }  
}
```

```
public int remove() {  
  
    if(noOfItems == 0) {  
        System.out.println("Queue is empty");  
        return -99;  
    }  
    else {  
        noOfItems--;  
        return queueArray[front++];  
    }  
}  
  
public int peakFront() {  
  
    if (noOfItems == 0) {  
        System.out.println("Queue is empty");  
        return -99;  
    }  
    else {  
        return queueArray[front];  
    }  
}  
  
public boolean isFull() {  
    if(noOfItems == (maxSize - 1)) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

```
public boolean isEmpty() {  
    if (noOfItems == 0) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```

Queue – Circular Queue

```
public class QueueCircle {  
  
    private int maxSize;  
    private int[] queueArray;  
    private int front;  
    private int rear;  
    private int noOfItems;  
  
    • public QueueCircle(int s) {  
  
        maxSize = s;  
        queueArray = new int[maxSize];  
        front = 0;  
        rear = -1;  
        noOfItems = 0;  
    }  
  
    • public void insert(int j) {  
  
        if (noOfItems == maxSize) {  
            System.out.println("Queue is Full");  
        }  
        else {  
            if(rear == (maxSize - 1)) {  
                rear = -1;  
            }  
            queueArray[++rear] = j;  
            noOfItems++;  
        }  
    }  
}
```

```
33
34● public int remove() {
35
36     if (noOfItems == 0) {
37         System.out.println("Queue is empty");
38         return -99;
39     }
40     else {
41         int temp = queueArray[front++];
42
43         if(front == maxSize) {
44             front = 0;
45         }
46
47         noOfItems--;
48         return temp;
49     }
50 }
51
52● public int peakFront() {
53
54     if (noOfItems == 0) {
55         System.out.println("Queue is empty");
56         return -99;
57     }
58     else {
59         return queueArray[front];
60     }
61 }
62
```

```
62
63● public boolean isFull() {
64     if(noOfItems == (maxSize - 1)) {
65         return true;
66     }
67     else {
68         return false;
69     }
70 }
71
72● public boolean isEmpty() {
73     if (noOfItems == 0) {
74         return true;
75     }
76     else {
77         return false;
78     }
79 }
80 }
```

LinkedList

```
3 public class Link {  
4  
5     public int iData;  
6     public Link next;  
7  
8     public Link(int s) {  
9         iData = s;  
10        next = null;  
11    }  
12  
13    public void displayLink() {  
14        System.out.print(iData + " ");  
15    }  
16  
17 }  
18
```



```
3 public class LinkedList {
4
5     public Link first;
6
7     public LinkedList() {
8         first = null;
9     }
10
11     public void displayList() {
12
13         Link cur = first;
14
15         while (cur != null) {
16             cur.displayLink();
17             cur = cur.next;
18         }
19     }
20
21     public boolean findBoolean(int key) {
22
23         Link cur = first;
24
25         while(cur != null) {
26
27             if(cur.iData == key) {
28                 return true;
29             }
30             else {
31                 cur = cur.next;
32             }
33         }
34         return false;
35     }
36 }
```

```
36
37● public Link findLink(int key) {
38
39     Link cur = first;
40
41     while(cur != null) {
42
43         if(cur.iData == key) {
44             return cur;
45         }
46         else {
47             cur = cur.next;
48         }
49     }
50     return null;
51 }
52
53● public void insertFirst(int key) {
54
55     Link newLink = new Link(key);
56     newLink.next = first;
57     first = newLink;
58 }
```

```
59
60● public boolean insertAfter(int key , int newData) {
61
62     Link newLink = new Link(newData);
63
64     Link cur = first;
65
66     while(cur != null) {
67         if(cur.iData == key) {
68             newLink.next = cur.next;
69             cur.next = newLink;
70             return true;
71         }
72         else {
73             cur = cur.next;
74         }
75     }
76     return false;
77 }
78
79● public Link deleteFirst() {
80     Link temp = first;
81     first = first.next;
82     return temp;
83 }
84
```

```

84
85● public boolean delete (int key) {
86     Link cur = first;
87     Link previous = first;
88
89     while(cur != null)
90     {
91         if(cur.iData == key)
92         {
93             if(cur == first)//if first is deleting
94             {
95                 first = first.next;
96                 return true;
97             }
98             else// if a middle link is deleting
99             {
100                 previous.next = cur.next;
101                 return true;
102             }
103         }
104         else
105         {
106             previous = cur;
107             cur = cur.next;
108         }
109     }
110     return false;
111 }
112 }

```

Tree

```
2
3 public class Node {
4
5     public int iData;
6     public double dData;
7     public Node leftChild;
8     public Node rightChild;
9
10    public Node() {
11
12    }
13
14    public void displayNode() {
15        System.out.println(iData + " , " + dData);
16    }
17
18 }
19
```

```
2
3 public class Tree {
4
5     private Node root;
6
7    public Tree() {
8
9        root = null;
10    }
11
```

```

12 public void insert(int id , double dd) {
13
14     Node newNode = new Node();
15
16     newNode.iData = id;
17     newNode.dData = dd;
18
19     if(root == null) {
20         root = newNode;
21     }
22     else
23     {
24         Node cur = root;
25         Node previous = root;|
26
27         while(true) {
28
29             previous = cur;
30
31             if(id < cur.iData) {
32                 cur = cur.leftChild;
33
34                 if(cur == null) {
35                     previous.rightChild = newNode;
36                     return;
37                 }
38             }
39             else {
40                 cur = cur.rightChild;
41
42                 if(cur == null) {
43                     previous.leftChild = newNode;
44                     return;
45                 }
46             }
47         }
48     }
49 }
50
51

```

```
public boolean delete (int id) {  
    return false;  
}  
  
public Node find(int key) {  
    Node cur = root;  
  
    while (cur.iData != key) {  
        if (key < cur.dData) {  
            cur = cur.leftChild;  
        }  
        else {  
            cur = cur.rightChild;  
        }  
        if (cur == null) {  
            return null;  
        }  
    }  
  
    return cur;  
}  
  
@SuppressWarnings("unused")  
private void inOrder(Node localRoot) {  
    if(localRoot != null) {  
        inOrder(localRoot.leftChild);  
        localRoot.displayNode();  
        inOrder(localRoot.rightChild);  
    }  
}
```

```
86
87● @SuppressWarnings("unused")
88 private void preOrder(Node localRoot) {
89
90     if(localRoot != null) {
91
92         localRoot.displayNode();
93         preOrder(localRoot.leftChild);
94         preOrder(localRoot.rightChild);
95     }
96 }
97
98● @SuppressWarnings("unused")
99 private void postOrder(Node localRoot) {
100
101     if(localRoot != null) {
102
103         postOrder(localRoot.leftChild);
104         postOrder(localRoot.rightChild);
105         localRoot.displayNode();
106     }
107 }
108 }
109
```