Stack

```java
public class StackX {

    private int maxSize;
    private double[] stackArray;
    private int top;

    public StackX(int s) {

        maxSize = s;
        stackArray = new double[maxSize];
        top = -1;
    }

    public void push(double j) {

        if (top == (maxSize - 1)) {
            System.out.println("Queue is Full");
        }
        else {
            stackArray[++top] = j;
        }
    }

    public double pop() {

        if(top == -1) {
            System.out.println("Queue is Empty");
            return -99;
        }
        else {
            return stackArray[top--];
        }
    }
```

```java
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

```java
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++;
        }
    }
```

```java
    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;
        }
    }
```

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

Queue – Circular Queue

```java
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 if Full");
        }
        else {
            if(rear == (maxSize - 1)) {
                rear = -1;
            }
            queueArray[++rear] = j;
            noOfItems++;
        }
    }
```

```java
    public int remove() {

        if (noOfItems == 0) {
            System.out.println("Queue is empty");
            return -99;
        }
        else {
            int temp = queueArray[front++];

            if(front == maxSize) {
                front = 0;
            }

            noOfItems--;
            return temp;
        }
    }

    public int peakFront() {

        if (noOfItems == 0) {
            System.out.println("Queue is empty");
            return -99;
        }
        else {
            return queueArray[front];
        }
    }
```

```
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  }
```

LinkList

```java
public class Link {

    public int iData;
    public Link next;

    public Link(int s) {
        iData = s;
        next = null;
    }

    public void displayLink() {
        System.out.print(iData + " ");
    }

}
```

```java
public class LinkList {

    public Link first;

    public LinkList() {
        first = null;
    }

    public void displayList() {

        Link cur = first;

        while (cur != null) {
            cur.displayLink();
            cur = cur.next;
        }
    }

    public boolean findBoolean(int key) {

        Link cur = first;

        while(cur != null) {

            if(cur.iData == key) {
                return true;
            }
            else {
                cur = cur.next;
            }
        }
        return false;
    }
```

```
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      }
```

```java
    public boolean insertAfter(int key , int newData) {

        Link newLink = new Link(newData);

        Link cur = first;

        while(cur != null) {
            if(cur.iData == key) {
                newLink.next = cur.next;
                cur.next = newLink;
                return true;
            }
            else {
                cur = cur.next;
            }
        }
        return false;
    }

    public Link deleteFirst() {
        Link temp = first;
        first = first.next;
        return temp;
    }
```

```
    public boolean delete (int key) {
        Link cur = first;
        Link previous = first;

        while(cur != null)
        {
            if(cur.iData == key)
            {
                if(cur == first)//if first is deleting
                {
                    first = first.next;
                    return true;
                }
                else// if a middle link is deleting
                {
                    previous.next = cur.next;
                    return true;
                }
            }
            else
            {
                previous = cur;
                cur = cur.next;
            }
        }
        return false;
    }
}
```

Tree

```java
public class Node {

    public int iData;
    public double dData;
    public Node leftChild;
    public Node rightChild;

    public Node() {

    }

    public void displayNode() {
        System.out.println(iData + " , " + dData);
    }

}
```

```java
public class Tree {

    private Node root;

    public Tree() {

        root = null;
    }
```

```java
public void insert(int id , double dd) {

    Node newNode = new Node();

    newNode.iData = id;
    newNode.dData = dd;

    if(root == null) {
        root = newNode;
    }
    else
    {
        Node cur = root;
        Node previous = root;

        while(true) {

            previous = cur;

            if(id < cur.iData) {
                cur = cur.leftChild;

                if(cur == null) {
                    previous.rightChild = newNode;
                    return;
                }
            }
            else {
                cur = cur.rightChild;

                if(cur == null) {
                    previous.leftChild = newNode;
                    return;
                }
            }

        }
    }
}
```

```java
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);
    }
}
```

```
    @SuppressWarnings("unused")
    private void preOrder(Node localRoot) {

        if(localRoot != null) {

            localRoot.displayNode();
            preOrder(localRoot.leftChild);
            preOrder(localRoot.rightChild);
        }
    }

    @SuppressWarnings("unused")
    private void postOrder(Node localRoot) {

        if(localRoot != null) {

            postOrder(localRoot.leftChild);
            postOrder(localRoot.rightChild);
            localRoot.displayNode();
        }
    }
}
```