1.

```
class LinearSearch {

    static int search(int arr[], int n, int x)
    {
        for (int i = 0; i < n; i++) {
            if (arr[i] == x)
                return i;
        }

        return -1;
    }

    public static void main(String[] args)
    {
        int[] arr = { 13, 14, 11, 17, 15 };
        int n = arr.length;

        int x = 4;
  int index = search(arr, n, x);
        if (index == -1)
            System.out.println("Element is not present in the array");
        else
            System.out.println("Element found at position " + index);
    }
}
```

2.

```
class BinarySearch{
 int binarySearch(int arr[], int l, int r, int x)
 {
  while (l <= r) {
   int mid = (l + r) / 2;


   if (arr[mid] == x) {
    return mid;

   } else if (arr[mid] > x) {
    r = mid - 1;


   } else {
   l = mid + 1;
   }
  }


  return -1;
 }

 public static void main(String args[])
 {
  BinarySearch ob = new BinarySearch();
```

```
  int arr[] = { 21, 33, 4, 10, 40 };
  int n = arr.length;
  int x = 10;
  int result = ob.binarySearch(arr, 0, n - 1, x);

  if (result == -1)
   System.out.println("Element not present");
  else
   System.out.println("Element found at index "
      + result);
 }
}
```

3.

```
import java.util.*;

public class SortElementsByFrequency {
    public static List<Integer> sortByFrequency(int[] arr) {
        Map<Integer, Integer> frequencyMap = new HashMap<>();

        for (int num : arr) {
            frequencyMap.put(num, frequencyMap.getOrDefault(num, 0) + 1);
        }

        List<Integer> sortedList = new ArrayList<>(frequencyMap.keySet());
        Collections.sort(sortedList, (a, b) -> {
            int freqCompare = frequencyMap.get(b).compareTo(frequencyMap.get(a));
            if (freqCompare != 0) {
                return freqCompare;
            } else {
                return a.compareTo(b);
            }
        });

        return sortedList;
    }

    public static void main(String[] args) {
        int[] arr = {3, 3, 1, 1, 1, 5, 5, 5, 5, 2, 2, 4, 4, 4, 4, 4};
        List<Integer> sortedList = sortByFrequency(arr);
        System.out.println("Sorted elements by frequency: " + sortedList);
    }
}
```

4. Sort an array of 0s, 1s and 2s

```
public static void sort012(int[] arr) {
        int low = 0;
        int high = arr.length - 1;
        int mid = 0;
        int temp;
        while (mid <= high) {
            switch (arr[mid]) {
                case 0: {
                    temp = arr[low];
```

```
                arr[low] = arr[mid];
                arr[mid] = temp;
                low++;
                mid++;
                break;
            }
            case 1: {
                mid++;
                break;
            }
            case 2: {
                temp = arr[mid];
                arr[mid] = arr[high];
                arr[high] = temp;
                high--;
                break;
            }
        }
}
```

5. Java Program to Check for balanced parenthesis by using Stacks

```
 Stack<Character> stack = new Stack<>();
        for (char ch : str.toCharArray()) {
            if (ch == '(' || ch == '[' || ch == '{') {
                stack.push(ch);
            } else if (ch == ')' && !stack.isEmpty() && stack.peek() == '(') {
                stack.pop();
            } else if (ch == ']' && !stack.isEmpty() && stack.peek() == '[') {
                stack.pop();
            } else if (ch == '}' && !stack.isEmpty() && stack.peek() == '{') {
                stack.pop();
            } else {
                return false;
            }
        }
        return stack.isEmpty();
}
```

6. Java Program to Implement Stack

```
public class StackImplementation {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public StackImplementation(int size) {
        this.maxSize = size;
        this.stackArray = new int[maxSize];
        this.top = -1;
    }

    public void push(int value) {
        if (top + 1 < maxSize) {
            stackArray[++top] = value;
        } else {
```

```
            System.out.println("Stack Overflow");
        }
    }

    public int pop() {
        if (!isEmpty()) {
            return stackArray[top--];
        } else {
            throw new EmptyStackException();
        }
    }

    public int peek() {
        if (!isEmpty()) {
            return stackArray[top];
        } else {
            throw new EmptyStackException();
        }
    }

    public boolean isEmpty() {
        return (top == -1);
    }
}
```

7. Java Program to Implement Queue

```
public class Queue {
 int size =5;
 int Q[]=new int[size];
 int rear, front;

 public Queue() {
  front= rear= -1;
 }boolean isEmpty(){
  if(front==-1)
   return true;
  else
   return false;
 }boolean isfull(){
  if(front==-1 && rear == size-1)
   return true;
  else
   return false;
 }
 void enqueue(int x) {
  if(isfull()) {
   System.out.println("Queue is full.");
  }else {
   if(front == -1)
    front = 0;
   rear++;
   Q[rear]=x;
   System.out.println(x+" : Inserted ");
  }
 }int dequeue(){
```

```
  int x;
  if (isEmpty()){
   System.out.println("Queue is Empty !");
   return -1;
  }else {
   x=Q[front];
   if(front >= rear) {
    front =-1;
    rear  =-1;
   }else {
    front++;
   }System.out.println(x+" : Deleted");
   return x;
  }
 }
 void display() {
  if(isEmpty()) {
   System.out.println("Queue is Empty !");
  }else {
   for(int i =front;i<=rear;i++) {
    System.out.println(Q[i]);
   }
  }
}
```

8. Java Program to Implement Dequeue.

```
   public static void main(String[] args) {
        Deque<Integer> deque = new ArrayDeque<>();
        deque.addFirst(1);
        deque.addLast(2);
        deque.addLast(3);
        System.out.println(deque.removeFirst());
        System.out.println(deque.peekLast());
    }
```

9. Java Program to Implement Stack Using Two Queues

```
public class StackUsingQueues {
    private Queue<Integer> q1 = new LinkedList<>();
    private Queue<Integer> q2 = new LinkedList<>();
    private int top;

    public void push(int x) {
        q2.add(x);
        top = x;
        while (!q1.isEmpty()) {
            q2.add(q1.remove());
        }
        Queue<Integer> temp = q1;
        q1 = q2;
        q2 = temp;
    }

    public int pop() {
        int popValue = q1.remove();
```

```
        if (!q1.isEmpty()) {
            top = q1.peek();
        }
        return popValue;
    }

    public int peek() {
        return top;
    }

    public boolean empty() {
        return q1.isEmpty();
    }
}
```

10. Java Program to Implement Queue Using Two Stacks

```
public class QueueUsingStacks {
    private Stack<Integer> stack1 = new Stack<>();
    private Stack<Integer> stack2 = new Stack<>();

    public void enqueue(int x) {
        stack1.push(x);
    }

    public int dequeue() {
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.pop();
    }

    public int peek() {
        if (stack2.isEmpty()) {
            while (!stack1.isEmpty()) {
                stack2.push(stack1.pop());
            }
        }
        return stack2.peek();
    }

    public boolean empty() {
        return stack1.isEmpty() && stack2.isEmpty();
    }
}
```