Assignment No. 1 ->

```
public class NonRecursiveFibonacci {
    public static long nonRecursiveFibonacci(int n) {
        if (n <= 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            long[] fib = new long[n + 1];
            fib[0] = 0;
            fib[1] = 1;
            for (int i = 2; i <= n; i++) {
                fib[i] = fib[i - 1] + fib[i - 2];
            }
            return fib[n];
        }
    }
     public static void main(String[] args) {
        int n = 10;
        System.out.println("Fibonacci(" + n + ") = " + nonRecursiveFibonacci(n));
    }
}
```

```
java -cp /tmp/bDNOmtEXrM NonRecursiveFibonacci
Fibonacci(10) = 55
```

```
public class RecursiveFibonacci {
    public static long recursiveFibonacci(int n) {
        if (n <= 0) {
            return 0;
        } else if (n == 1) {
            return 1;
        } else {
            return recursiveFibonacci(n - 1) + recursiveFibonacci(n - 2);
        }
    }
public static void main(String[] args) {
        int n = 10;
        System.out.println("Fibonacci(" + n + ") = " + recursiveFibonacci(n));
    }
}
```

```
java -cp /tmp/bDNOmtEXrM RecursiveFibonacci
Fibonacci(10) = 55
```

Assignment No. 2 ->

```
import java.util.Comparator;
import java.util.PriorityQueue;
import java.util.Scanner;
class Huffman {
   public static void printCode(HuffmanNode root, String s) {
      if (root.left == null && root.right == null && Character.isLetter(root.c)) {
         System.out.println(root.c + ":" + s);
         return;
      }
      printCode(root.left, s + "0");
      printCode(root.right, s + "1");
   }
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);
      int n = 6;
      char[] charArray = {'a', 'b', 'c', 'd', 'e', 'f'};
      int[] charfreq = {5, 9, 12, 13, 16, 45};
      PriorityQueue<HuffmanNode> q = new PriorityQueue<HuffmanNode>(n, new MyComparator());
      for (int i = 0; i < n; i++) {
         HuffmanNode hn = new HuffmanNode();
         hn.c = charArray[i];
         hn.data = charfreq[i];
         hn.left = null;
         hn.right = null;
         q.add(hn);
      }
      HuffmanNode root = null;
      while (q.size() > 1) {
         HuffmanNode x = q.peek();
         q.poll();
         HuffmanNode y = q.peek();
```

```
            q.poll();
            HuffmanNode f = new HuffmanNode();
            f.data = x.data + y.data;
            f.c = '-';
            f.left = x;
            f.right = y;
            root = f;
            q.add(f);
        }
        printCode(root, "");
    }
}
class HuffmanNode {
    int data;
    char c;
    HuffmanNode left;
    HuffmanNode right;
}
class MyComparator implements Comparator<HuffmanNode> {
    public int compare(HuffmanNode x, HuffmanNode y) {
        return x.data - y.data;
    }
}
```

```
java -cp /tmp/7Ki6BysSbS Huffman
f:0
c:100
d:101
a:1100
b:1101
e:111
```

Assignment No. 3 ->

```
import java.util.Arrays;
import java.util.Comparator;

public class FractionalKnapSack {
   private static double getMaxValue(ItemValue[] arr, int capacity) {
      Arrays.sort(arr, new Comparator<ItemValue>() {
         @Override
         public int compare(ItemValue item1, ItemValue item2) {
            double cpr1 = new Double((double) item1.profit / (double) item1.weight);
            double cpr2 = new Double((double) item2.profit / (double) item2.weight);
            if (cpr1 < cpr2)
               return 1;
            else
               return -1;
         }
      });
      double totalValue = 0d;
      for (ItemValue i : arr) {
         int curWt = (int) i.weight;
         int curVal = (int) i.profit;
         if (capacity - curWt >= 0) {
            capacity = capacity - curWt;
            totalValue += curVal;
         } else {
            double fraction = ((double) capacity / (double) curWt);
            totalValue += (curVal * fraction);
            capacity = (int) (capacity - (curWt * fraction));
            break;
         }
      }
      return totalValue;
```

```
  }
   static class ItemValue {
      int profit, weight;

      public ItemValue(int val, int wt) {
         this.weight = wt;
         this.profit = val;
      }
   }
   public static void main(String[] args) {
      ItemValue[] arr = { new ItemValue(60, 10), new ItemValue(100, 20), new ItemValue(120, 30) };
      int capacity = 50;
      double maxValue = getMaxValue(arr, capacity);
      System.out.println("Total Cost of Knapsack – " + maxValue);
   }
}
```

```
java -cp /tmp/7Ki6BysSbS FractionalKnapSack
Total Cost of Knapsack - 240.0
```

Assignment No. 4 ->

```
public class KnapsackDP {
    public static int knapsack(int[] values, int[] weights, int capacity) {
        int n = values.length;
        int[][] dp = new int[n + 1][capacity + 1];
        for (int i = 0; i <= n; i++) {
            for (int w = 0; w <= capacity; w++) {
                if (i == 0 || w == 0) {
                    dp[i][w] = 0;
                } else if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(values[i - 1] + dp[i - 1][w - weights[i - 1]], dp[i - 1][w]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
        return dp[n][capacity];
    }
    public static void main(String[] args) {
        int[] values = {60, 100, 120};
        int[] weights = {10, 20, 30};
        int capacity = 50;

        int maxValue = knapsack(values, weights, capacity);
        System.out.println("Maximum value that can be obtained: " + maxValue);
    }
}
```

```
java -cp /tmp/OhOPtLRI1U KnapsackDP
Maximum value that can be obtained: 220
```

Assignment No. 5 ->

```
public class NQueenProblem {
   final int N = 4;

   void printSolution(int board[][]) {
      for (int i = 0; i < N; i++) {
         for (int j = 0; j < N; j++) {
            if (board[i][j] == 1)
               System.out.print("Q ");
            else
               System.out.print(". ");
         }
         System.out.println();
      }
   }

   boolean isSafe(int board[][], int row, int col) {
      int i, j;

      for (i = 0; i < col; i++)
         if (board[row][i] == 1)
            return false;

      for (i = row, j = col; i >= 0 && j >= 0; i--, j--)
         if (board[i][j] == 1)
            return false;

      for (i = row, j = col; j >= 0 && i < N; i++, j--)
         if (board[i][j] == 1)
            return false;
```

```
    return true;
}

boolean solveNQUtil(int board[][], int col) {
    if (col >= N)
        return true;

    for (int i = 0; i < N; i++) {
        if (isSafe(board, i, col)) {
            board[i][col] = 1;
            if (solveNQUtil(board, col + 1))
                return true;
            board[i][col] = 0;
        }
    }

    return false;
}

boolean solveNQ() {
    int board[][] = {{0, 0, 0, 0},
                     {0, 0, 0, 0},
                     {0, 0, 0, 0},
                     {0, 0, 0, 0}};

    if (solveNQUtil(board, 0) == false) {
        System.out.print("Solution does not exist");
        return false;
    }

    printSolution(board);
```

```
        return true;
    }

    public static void main(String args[]) {
        NQueenProblem Queen = new NQueenProblem();
        Queen.solveNQ();
    }
}
```

```
java -cp /tmp/yquhWrXcDC NQueenProblem
 . . Q .
 Q . . .
 . . . Q
 . Q . .
```