Parctical No. 2

Roll No. 27

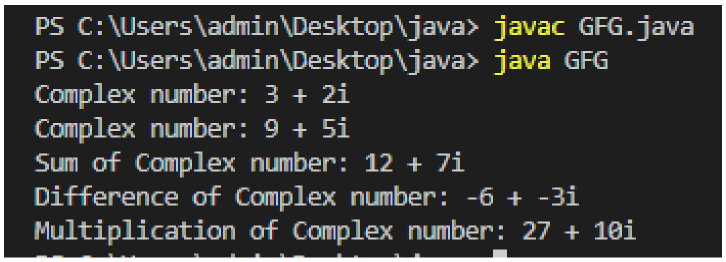
1)Program to perform complex Number addition, multiplication and substraction using constructors overloading.

Theory : Constructor overloading is a technique of having more than one constructor with different parameters lists. The complex numbers will be initialized with the help of constructor. Fucntion will be called with the help of another class.

Code :

```
import java.util.*;
class Complex {
   int real, imaginary;
   Complex() {}
   Complex(int tempReal, int tempImaginary)
   {
      real = tempReal;
      imaginary = tempImaginary;
   }
   Complex addComp(Complex C1, Complex C2)
   {
      Complex temp = new Complex();
      temp.real = C1.real + C2.real;
      temp.imaginary = C1.imaginary + C2.imaginary;
      return temp;
   }
   Complex subtractComp(Complex C1, Complex C2)
   {
      Complex temp = new Complex();
      temp.real = C1.real - C2.real;
      temp.imaginary = C1.imaginary - C2.imaginary;
      return temp;
   }
   Complex MultiplyComp(Complex C1, Complex C2)
   {
      Complex temp =  new Complex();
      temp.real = C1.real * C2.real;
      temp.imaginary = C1.imaginary * C2.imaginary;
      return temp;
   }
   void printComplexNumber()
   {
      System.out.println("Complex number: " + real + " + " + imaginary + "i");
   }
}
public class GFG {
    public static void main(String[] args)
   {
      Complex C1 = new Complex(3, 2);
      C1.printComplexNumber();
      Complex C2 = new Complex(9, 5);
      C2.printComplexNumber();
      Complex C3 = new Complex();
      C3 = C3.addComp(C1, C2);
      System.out.print("Sum of ");
      C3.printComplexNumber();
      C3 = C3.subtractComp(C1, C2);
      System.out.print("Difference of ");
      C3.printComplexNumber();
      C3 = C3.MultiplyComp(C1, C2);
      System.out.print("Multiplication of ");
      C3.printComplexNumber();
   }
        }
```

Output :



```
PS C:\Users\admin\Desktop\java> javac GFG.java
PS C:\Users\admin\Desktop\java> java GFG
Complex number: 3 + 2i
Complex number: 9 + 5i
Sum of Complex number: 12 + 7i
Difference of Complex number: -6 + -3i
Multiplication of Complex number: 27 + 10i
```

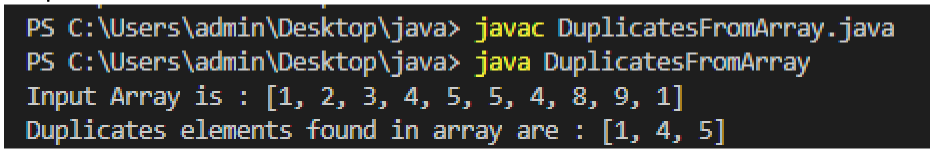
2)To check if the array has any duplicate element.

Theory : Array is passed to findDuplicates() in which the duplicates elements are found with binary search logic. After duplicates element found get stored in hash table in stored form by Set

Code :

```
import java.util.Arrays;
import java.util.Set;
import java.util.HashSet;
public class DuplicatesFromArray {
    public static void main(String args[]) {
      int[] arrayElements = {1, 2, 3, 4, 5, 5, 4, 8, 9, 1};
      Set<Integer> result = findDuplicates(arrayElements);
      System.out.println("Input Array is : " + Arrays.toString(arrayElements));
      System.out.println("Duplicates elements found in array are : " + result);
    }
    public static Set<Integer> findDuplicates(int[] input) {
        Set<Integer> result = new HashSet<Integer>();
        for(int i=0; i < input.length; i++) {
            for (int j=1; j < input.length; j++) {
                if(input[i] == input[j] && i != j) {
                    result.add(input[i]);
                    break;
                }
            }
        }
        return result;
    }
}
```

Output :



```
PS C:\Users\admin\Desktop\java> javac DuplicatesFromArray.java
PS C:\Users\admin\Desktop\java> java DuplicatesFromArray
Input Array is : [1, 2, 3, 4, 5, 5, 4, 8, 9, 1]
Duplicates elements found in array are : [1, 4, 5]
```

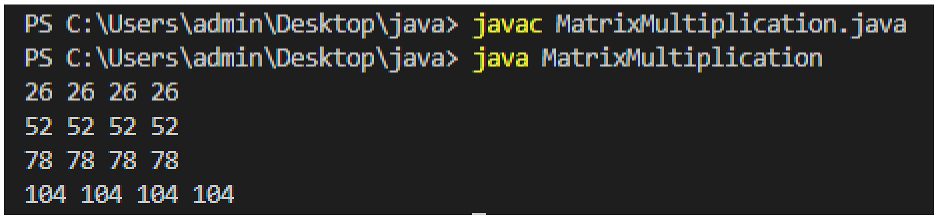
3) A. Matrix Multiplication

Theory : The new matrix result is created to store the product of two matrix. First row of matrix1[i][k] multiply with the first column of matrix2[k][j] and sum of all get stored in result[i][j].

Code :

```
import java.io.*;
public class MatrixMultiplication {
    static int N = 4;
    static void multiply(int matrix1[][], int matrix2[][], int result[][])
    {
        int i, j, k;
        for(i=0; i<N; i++)
        {
            for(j=0; j<N; j++)
            {
                result[i][j]=0;
                for(k=0; k<N; k++)
                {
                    result[i][j] += matrix1[i][k] * matrix2[k][j];
                }
            }
        }
    }
    public static void main(String args[])
    {
        int matrix1[][] = {{1, 1, 1, 1}, {2, 2, 2, 2}, {3, 3, 3, 3}, {4, 4, 4, 4}};
        int matrix2[][] = {{5, 5, 5, 5}, {6, 6, 6, 6}, {7, 7, 7, 7}, {8, 8, 8, 8}};
        int result[][] = new int[N][N];
        int i,j;
        multiply(matrix1, matrix2, result);
        for(i=0; i<N; i++)
        {
            for(j=0; j<N; j++)
            {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output :



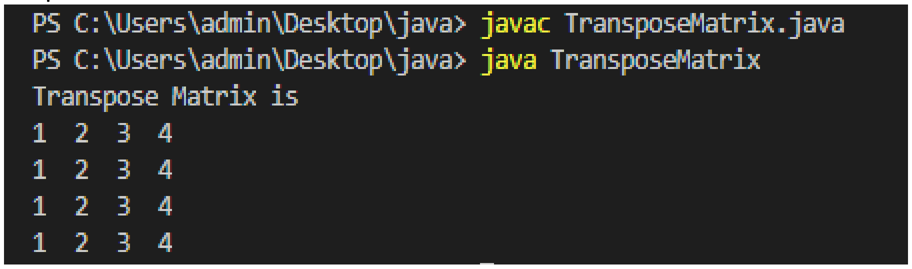
```
PS C:\Users\admin\Desktop\java> javac MatrixMultiplication.java
PS C:\Users\admin\Desktop\java> java MatrixMultiplication
26 26 26 26
52 52 52 52
78 78 78 78
104 104 104 104
```

B. Transpose Matrix

Theory : matrix2[][] is formed to store the result. In below program the transpose is done by storing the matrix1[j][i] value to matrix2[i][j] place.

Code :

```
class TransposeMatrix {
    static final int N = 4;
    static void transpose(int matrix1[][], int matrix2[][]) {
        int i, j;
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
                matrix2[i][j] = matrix1[j][i];
            }
        }
    }
    public static void main (String[] args) {
        int matrix1[][] = { {1, 1, 1, 1},
                            {2, 2, 2, 2},
                            {3, 3, 3, 3},
                            {4, 4, 4, 4}};
        int matrix2[][] = new int[N][N], i, j;
        transpose(matrix1, matrix2);
        System.out.print("Transpose Matrix is \n");
        for (i = 0; i < N; i++) {
            for (j = 0; j < N; j++) {
            System.out.print(matrix2[i][j] + "  ");
            }
            System.out.print("\n");
        }
    }
}
```
Output :



```
PS C:\Users\admin\Desktop\java> javac TransposeMatrix.java
PS C:\Users\admin\Desktop\java> java TransposeMatrix
Transpose Matrix is
1  2  3  4
1  2  3  4
1  2  3  4
1  2  3  4
```