JAVA – GrPA Solutions

Week – 2

Write a program to find the sum of the following series up to n terms.

$$1^2+(1^2+2^2)+(1^2+2^2+3^2)+\ldots\ldots+(1^2+2^2+\ldots.+n^2)$$

```java
import java.util.*;
public class SeriesSum {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
int sum = 0 ;
for(int i=1;i<=n;i++)
{
    for (int j=1;j<=i;j++)
    {
        sum = sum+(j*j);
    }
}
System.out.println(sum);

  }
}
```

JAVA – GrPA Solutions

Complete the definition of the given class by defining appropriate constructors and member functions such that it is in coherence with the given main method and produce the required output.

```
import java.util.Scanner;

class Employee{
        String ename;
        String eid;
        String edept;

        public Employee(){
                ename = "guest";
        }
public Employee(String name, String id, String dept) {
                ename = name;
                eid = id;
                edept = dept;
        }
        public void copyDept(Employee e){
                this.edept = e.edept;
        }
        public void displayDetails() {
                System.out.println("ename : "+this.ename);
                System.out.println("eid : "+this.eid);
                System.out.println("edept : "+this.edept);
        }
```

```
}
public class FClass
{
        public static void main(String args[])
        {
                Scanner s = new Scanner(System.in);
                Employee e1 = new Employee();

                //Enter name of the employee
                String name = s.nextLine();

                //Enter id of the employee
                String id = s.nextLine();

                //Enter department of the employee
                String dept = s.nextLine();

                Employee e2 = new Employee(name,id,dept);

                e1.copyDept(e2);
                //Copies the department name of e2 into e1's department name.

                e1.displayDetails();
        }
}
```

JAVA – GrPA Solutions

Complete the definition of the given class by defining appropriate constructors and member functions such that it is in coherence with the given main method and produce the required output

```java
import java.util.*;

class Employee
{
    String eid;
    String ename;
    String eprojects[];
 //Define all the required methods here
 public Employee(String id, String name, String[] project)
    {
        this.eid = id;
        this.ename = name;
        this.eprojects = project;
    }

public Employee(Employee e)
    {
        this.eid = e.eid;
        this.ename = e.ename;
        this.eprojects = e.eprojects;
    }

 public void display()
    {
        System.out.println("id:" + eid);
        System.out.println("name:" + ename);
        System.out.println("projects:");
        eprojects[0] = "P001";
        for (int i = 0; i < eprojects.length; i++) {
            System.out.print(eprojects[i] + ":");
        }
    }
public void mutator()
    {
        this.ename = "Mr "+ this.ename;
        this.eprojects[0] = null;
    }

}
```

```java
public class FClass
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
           String project[] = {"P001","P002","P003"};
        //Enter the id of employee
        String id = s.nextLine();
        //Enter the name of employee
        String name = s.nextLine();

        Employee e1 = new Employee(id,name,project);
        Employee e2 = new Employee(e1);
        //The copy constructor must copy all the data members.

        e1.mutator();

        e2.display();
    }
}
```

JAVA – GrPA Solutions

Week – 3

Implement the code as instructed in the comment, such that it satisfies the given test cases and is in coherence with the given ***main*** method

```
import java.util.*;
class Person{
    private String name;
    private long aadharno;
    public Person(String name, long aadharno){
        this.name = name;
        this.aadharno = aadharno;
    }
    public void print() {
        System.out.println("name : " + name);
        System.out.println("aadharno : " + aadharno);
    }
}

class Employee extends Person{
    private double salary;

    //implement the constructor
public Employee(String nm, long adhano, double salary){
        super(nm, adhano);
        this.salary = salary;
    }
    //override print method
        public void print(){
        super.print();
        System.out.println("salary : " + salary);
    }
}


class ContactEmployee extends Employee{
    final private static double hourlyPay = 100.00;
    private int contactHour;

    //implement the constructor
     public ContactEmployee(String nam, long aadhano, int con){
        super(nam, aadhano, con * hourlyPay);
    }
    //salary is computed as contactHour * hourlyPay
    public void print(){
        super.print();
}
}
```

JAVA – GrPA Solutions

```
44
45  class FClass{
46      public static void main(String[] args) {
47          Scanner sc = new Scanner(System.in);
48          String nm1 = sc.nextLine();
49          String nm2 = sc.nextLine();
50          long adh1 = sc.nextLong();
51          long adh2 = sc.nextLong();
52          double sal = sc.nextDouble();
53          int cont = sc.nextInt();
54          Employee[] eArr = new Employee[2];
55          eArr[0] = new Employee(nm1, adh1, sal);
56          eArr[1] = new ContactEmployee(nm2, adh2, cont);
57          for(Employee e : eArr)
58              e.print();
59      }
60  }
```

JAVA – GrPA Solutions

```java
import java.util.*;
class Shape{
    public int area() {
        return 0;
    }
    public int volume() {
        return 0;
    }
}

class Rectangle extends Shape{
    private int w, h;
//implement Rectangle class
 public Rectangle(int wid, int hei){
        this.w = wid;
        this.h = hei;
    }

    public int area(){
        return w * h;
    }
}

class Cube extends Shape{
    private int a;
    //implement Cube class
     public Cube(int len){
        this.a = len;
    }

    public int volume(){
        return a*a*a;
    }
}
```

JAVA – GrPA Solutions

```
class FClass{
    private static void caller(Shape s) {
        //check if s is of type Rectangle
              if (s instanceof Rectangle){
            System.out.println(s.area());
        }
        //check if s is of type Cube
  if (s instanceof Cube){
            System.out.println(s.volume());
        }

    }
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int w = sc.nextInt();
        int h = sc.nextInt();
        int a = sc.nextInt();
        caller(new Rectangle(w, h));
        caller(new Cube(a));
    }
}
```

JAVA – GrPA Solutions

Create *BankAccount* class that has the following instance variables and methods:

Instance variables:

accountNumber

name

balance

final variable: minBalance

Private method:

checkMinBalance(amount) - returns false if *balance - amount <= minBalance* else returns true

Public methods:

balance( ) - prints the balance

deposit(amount) - updates *balance = balance + amount*

withdraw(amount) - calls the *checkMinBalance(amount)* method,
if it returns true update *balance = balance - amount* else prints Transaction failed

```
import java.util.*;
    class BankAccount{
        int accountNumber;
        String name;
        int balance;
        final int minBalance = 100;
        private boolean checkMinBalance(int amount){
            if(balance - amount <= minBalance){
                return false;
            }
            else{
              return true;
            }
        }
```

```
    //Fill the code here
    public BankAccount(int acc,String n,int bal)
        {
          accountNumber = acc;
          name = n;
          balance = bal;
        }
        public void balance()
        {
          System.out.println(balance);
        }
        public void deposit(int amt)
        {
          balance = balance + amt;
        }
        public void withdraw(int amt)
        {
          if(checkMinBalance(amt) == true)
          {
            balance = balance - amt;
          }
          else
          {
            System.out.println("Transaction failed");
          }
        }
```

```
}
    class AccountCheck{
        public static void main(String[] args) {
            Scanner sc = new Scanner(System.in);
            int amnt = sc.nextInt( );
            int amnt1 = sc.nextInt( );
            BankAccount b = new BankAccount(1890, "rahul", 1000);
            b.deposit(amnt);
            b.balance();
            b.withdraw(amnt1);
            b.balance();
        }

    }
```

JAVA – GrPA Solutions

Week – 4

Create an abstract class **StringOperations** that has the following abstract methods:

String *reverse(String s)*

int *vowelCount(String s)*

Create **StringReverse** class that extends **StringOperations** class but defines only *String reverse(String s)* method. It reverses the string which is passed as parameter and returns the reversed string.

Create **UpdatedStrings** class that extends **StringReverse** class and defines int *vowelCount(String s)* method. It counts the vowels in the string which is passed as parameter and returns the count.

```
import java.util.*;
abstract class StringOperations{
  public abstract String reverse(String s);
  public abstract int vowelCount(String s);
}
//Fill the code here
abstract class StringReverse extends StringOperations
{
    public String reverse(String s)
    {
        String str = "";
        for(int i = 0;i<s.length();i++)
        {
            char ch = s.charAt(i);
            str = ch + str;
        }
        return(str);
    }
}

class UpdatedStrings extends StringReverse
{
    int k=0;
    String str = "AEIOUaeiou";
    public int vowelCount(String s)
    {
        for(int i=0;i<s.length();i++)
        {
            for(int j = 0;j<str.length();j++)
            {
                if(s.charAt(i)==str.charAt(j))
                {
                    k++;
                }
            }
        }
        return(k);
    }
}
class Example {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.next();
    UpdatedStrings str = new UpdatedStrings();
    System.out.println("Reverse of "+ s + " is "+ str.reverse(s));
    System.out.println("Vowel count of "+ s + " is "+ str.vowelCount(s));
  }
}
```

JAVA – GrPA Solutions

```java
import java.util.*;

interface Iterator{
    public boolean has_next();
    public Object get_next();
}

class Sequence{
    private final int maxLimit = 80;
    private SeqIterator _iter = null;
    int[] iArr;
    int size;
public Sequence(int size_) {
        iArr = new int[80];
        size = 0;
    }

    public void addTo(int elem) {
        iArr[size] = elem;
        size++;
    }

    public Iterator get_Iterator() {
        _iter = new SeqIterator();
        return _iter;
    }
    private class SeqIterator implements Iterator{
        int indx;
        public SeqIterator(){
            indx = -1;
        }
        public boolean has_next() {
            if(indx < size - 1)
                return true;
            return false;
        }
        public Object get_next() {
            return iArr[++indx];
        }
    }
}
```

```java
class FClass{
    public static void main(String[] args) {
        Sequence sObj = new Sequence(5);
        Scanner sc = new Scanner(System.in);
        for(int i = 0; i < 5; i++) {
            sObj.addTo(sc.nextInt());
        }
        Iterator i = sObj.get_Iterator();
        while(i.has_next())
            System.out.print(i.get_next() + ", ");
    }
}
```

JAVA – GrPA Solutions

Week – 5

Given as input two integers *n_1,n_2* and two double values *d_1,d_2* complete the Java code to form two complex numbers *c_1* and *c_2*, as described below, and print their sum.

The real parts of *c_1* and *c_2* are *n_1* and *d_1* respectively, whereas their imaginary parts are *n_2* and *d_2*, respectively.

Define a generic class **`ComplexNum`** with the following members.

Instance variables *r* and *i*

A constructor to initialize *r* and *i*

A method **`add()`** to return the sum of the two instances of generic type **`ComplexNum`**

A method that overrides the **`toString()`** method in the **`Object`** class so that the format of the output is in accordance with those in the test cases.

```
1  import java.util.*;
2  class ComplexNum<T extends Number>{
3      private T r, i;
4      public ComplexNum(T r, T i) {
5          this.r = r;
6          this.i = i;
7      }
8      public ComplexNum<Double> add(ComplexNum<?> c){
9          ComplexNum<Double> dc = new ComplexNum<Double>(0.0, 0.0);
10         dc.r = this.r.doubleValue() + c.r.doubleValue();
11         dc.i = this.i.doubleValue() + c.i.doubleValue();
12         return dc;
13     }
14     public String toString() {
15         return r.doubleValue() + " + " + i.doubleValue() + "i";
16     }
17 }
18 class FClass{
19 public static void main(String[] args) {
20         Scanner sc = new Scanner(System.in);
21         int n1, n2;
22         double d1, d2;
23         n1 = sc.nextInt();
24         n2 = sc.nextInt();
25         d1 = sc.nextDouble();
26         d2 = sc.nextDouble();
27         ComplexNum<Integer> c1 = new ComplexNum<Integer>(n1, n2);
28         ComplexNum<Double> c2 = new ComplexNum<Double>(d1, d2);
29         ComplexNum<Double> c3 = c1.add(c2);
30         System.out.println(c1 + " + " + c2 + " = " + c3);
31     }
32 }
```

JAVA – GrPA Solutions

Write a Java code that takes as input a positive number (length of an array here), and two arrays of that length - one of integers and another of strings. The code must also take an integer and a String as input, and print the number of occurrences of the integer and the string in the integer array and the string array, respectively.

Format of input:

Length of the arrays

Elements in the integer array (in separate lines)

Element to count in the integer array

Elements in the string array (in separate lines)

Element to count in the string array

Variables used in the code:

len - represents length of array

s1 - represents an element to be counted for in Integer array

s2 - represents an element to be counted for in String array

```
import java.util.*;
class ArrayExample <T>{
  T[] a;

public ArrayExample(T[] arr){
        a = arr;
      }
      public void display(){
        for(int i = 0; i < a.length; i++){
          System.out.print(a[i] + " ");
        }
        System.out.println();
      }
      public int elementCount(T x){
        int count = 0;
        for(int i = 0; i < a.length; i++){
          if(a[i].equals(x)){
            count = count + 1;
          }
        }
        return count;
      }
}
public class ArrayObject{
      public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        int len = sc.nextInt(); //Taking input for length of array
        Integer[] x = new Integer[len];
        for(int i = 0; i < len; i++){
          x[i] = sc.nextInt();
        }
        ArrayExample<Integer> obj = new ArrayExample<Integer>(x);
        int s1 = sc.nextInt();
        String[] y = new String[len];
        for(int i = 0; i < len; i++){
          y[i] = sc.next(); //Taking input for String array
        }
        ArrayExample<String> obj1 = new ArrayExample<String>(y);
    String s2 = sc.next(); //Taking input for the value to be counted
    obj.display();
    System.out.println(obj.elementCount(s1));
    obj1.display();
    System.out.println(obj1.elementCount(s2));
  }
}
```

JAVA – GrPA Solutions

Week – 6

Given as input a set of four objects of class **`CricketPlayer`** complete the Java code to segregate the players represented by these objects into batsmen and bowlers.

Create an **`ArrayList`** object to store the four objects of **`CricketPlayer.`** Segregate them as batsmen and bowlers based on the following criteria:

A player is termed as a batsman if his/her average runs per match are greater than 25.

A player is termed as a bowler if his/her average wickets per match are greater than 1.

Create **`ArrayList`** ***`bt`*** to store the batsmen and **`ArrayList`** ***`bw`*** to store the bowlers. Observe that the same player could belong to both the lists.

Print the list of bowlers in a line, followed by the list of batsmen in the next line, using the **`displayPlayers(ArrayList<CricketPlayer> bw, ArrayList<CricketPlayer> bt)`** method.

```java
import java.util.*;
class CricketPlayer{
  private String name;
  private int wickets;
  private int runs;
  private int matches;
  public CricketPlayer(String s, int w, int r, int m){
    this.name = s;
    this.wickets = w;
    this.runs = r;
    this.matches = m;
  }
  public String getName(){
    return name;
  }
  public int getWickets(){
    return wickets;
  }
  public int getRuns(){
    return runs;
  }
  public double avgRuns(){
    return runs/matches;
  }
  public double avgWickets(){
    return wickets/matches;
  }
}
```

JAVA – GrPA Solutions

```
29 public class Main {
30 public static void displayPlayers(ArrayList<CricketPlayer> BW,
31                                    ArrayList<CricketPlayer> BT){
32     for(CricketPlayer p : BW){
33       System.out.print(p.getName()+ " ");
34     }
35     System.out.println();
36     for(CricketPlayer p : BT){
37       System.out.print(p.getName()+ " ");
38     }
39     System.out.println();
40   }
41   public static void main(String[] args) {
42     Scanner sc = new Scanner(System.in);
43     CricketPlayer p1 = new CricketPlayer(sc.next(), sc.nextInt(),
44                                          sc.nextInt(), sc.nextInt());
45     CricketPlayer p2 = new CricketPlayer(sc.next(), sc.nextInt(),
46                                          sc.nextInt(), sc.nextInt());
47     CricketPlayer p3 = new CricketPlayer(sc.next(), sc.nextInt(),
48                                          sc.nextInt(), sc.nextInt());
49     CricketPlayer p4 = new CricketPlayer(sc.next(), sc.nextInt(),
50                                          sc.nextInt(), sc.nextInt());

52     ArrayList<CricketPlayer> temp = new ArrayList<CricketPlayer>();
53     ArrayList<CricketPlayer> bt = new ArrayList<CricketPlayer>();
54     ArrayList<CricketPlayer> bw = new ArrayList<CricketPlayer>();
55     temp.add(p1);
56     temp.add(p2);
57     temp.add(p3);
58     temp.add(p4);
59     for(CricketPlayer p: temp){
60       if(p.avgRuns() > 25){
61         bt.add(p);
62       }
63       if(p.avgWickets() > 1){
64         bw.add(p);
65       }
66     }
67  displayPlayers(bw, bt);
68   }
69 }
```

JAVA – GrPA Solutions

Write a program that checks for balanced parentheses in an expression i.e. whether the pairs and the order of "{", "}","(",")","[","]" are correct in the given input.

The program should keep taking expressions as input one after the other, until the user enters the word `done' (not case-sensitive). After all the expressions are input, for each input, the program should print whether the given expression is balanced or not (the order of the output should match the order of the input). If an input expression is balanced, print **Balanced** else print **Not Balanced**

```
import java.util.*;

public class Test3{
    public static boolean balanceCheck(String sequence) {
//Write your code here
  Stack<Character> stack = new Stack<Character>();
        for (int i=0; i < sequence.length(); i++) {
            char c = sequence.charAt(i);
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            }
            else if (c == ')' || c == ']' || c == '}') {
                if (stack.empty()) {
                    return false;
                }
                char ele = stack.peek();
                if ( (ele == '(' && c == ')') || (ele == '[' && c == ']') || (ele == '{' && c == '}') ) {
                    stack.pop();
                    continue;
                }
                break;
            }
        }
        if (stack.empty()) {
            return true;
        }
        return false;
}
```

```
    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);

        ArrayList<String> expr_arr= new ArrayList<String>();
        String inp=null;

        do {
            inp = s.nextLine();
            if(!inp.equalsIgnoreCase("Done"))
                expr_arr.add(inp);
        }while(!inp.equalsIgnoreCase("Done"));

        for(String expr : expr_arr) {
            if(balanceCheck(expr)) {
                System.out.println("Balanced");
            }
            else {
                System.out.println("Not Balanced");
            }
        }
    }
}
```

JAVA – GrPA Solutions

Week – 7

Recall that Java throws an **`ArithmeticException`** if there is an attempt to divide by zero. Similar to this, we can generate an exception if there is a division by three.

Given two integers as input, complete the Java code given below to generate such an exception.

Create a class **`DivisionException`** that extends the class **`Exception`**.

Override the **`toString()`** method to return "Division by 3 is not allowed".

In the class Test, define **`divide(int a, int b)`** to return **a/b**, if the value of b is not equal to 3. If the value of b is 3, then throw an instance of **`DivisionException`**.

Inside the method main(), invoke **`divide(x, y)`**, and handle any possible exception by printing the said message.

```java
import java.util.*;
class DivisionException extends Exception{
  public String toString(){
    return "Division by 3 is not allowed";
  }
}
public class Test {
  public static int divide(int a, int b) throws DivisionException{
    if(b == 3){
      throw new DivisionException();
    }
    else {
      return a/b;
    }
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int x = sc.nextInt();
    int y = sc.nextInt();
    try{
      int c = divide(x, y);
      System.out.println(c);
    }
    catch(DivisionException e){
      System.out.println(e);
    }
  }
}
```

JAVA – GrPA Solutions

Write a Java program that accepts as input an array of 5 integers. Instead of accepting elements in the order of indices (from 0 to 4), it accepts the array as 5 pairs of integers, where each pair is an index-value pair. The first integer in a pair represents the array index (or position), with accepted values ranging from 0 to 4. The second integer is the value at that index inside the array. Note that the input may not be in the order of the indices.

If any of the given index is out of range, then your code must throw appropriate exceptions, as shown in the test cases. If all indices are within the permissible range, then the code must print the values of the array in a single line (each value followed by a space).

Define a checked exceptionInvalidInputEx.

Define a class IntList having the following:

An integer array as an instance variable to store the 5 values.

A method set_value with two arguments - one for the index and the other for the value - that stores the value at the given index of the array. If an index is < 0 or >4, handle the appropriate exception and re-throw exception InvalidInputEx (which would be handled in main). Set the original exception as cause of the new exception, and then throw the new exception.

A methodgetArray to return the integer array.

```
import java.util.*;

class InvalidInputEx extends Exception{
    public InvalidInputEx(String msg) {
        super(msg);
    }
}

class IntList{
    private int[] i_arr = new int[5];
    public void set_value(int i, int v) throws InvalidInputEx {
        try {
            i_arr[i] = v;
        }catch(ArrayIndexOutOfBoundsException e1) {
            InvalidInputEx e2 = new InvalidInputEx("invalid index input");
            e2.initCause(e1);
            throw e2;
        }
    }
    public int[] getArray() {
        return i_arr;
    }
}
```

```
class FClass{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        IntList ilist = new IntList();
        try {
            for(int i = 0; i < 5; i++) {
                int n = sc.nextInt();
                int m = sc.nextInt();
                ilist.set_value(n, m);
            }
        }
        catch(InvalidInputEx e) {
            System.out.println(e.getMessage());
            Throwable ori = e.getCause();
            System.out.println(ori.getMessage());
        }
        int[] i_arr = ilist.getArray();
        for(int i = 0; i < i_arr.length; i++)
            System.out.print(i_arr[i] + " ");
    }
}
```

JAVA – GrPA Solutions

Week – 8

The program stores a list of Employee objects, each of which has name, department and salary as instance variables. A user can query the list to find the Employees who belong to a specific department and have salary greater than or equal to the input salary. Complete the program as specified.

Define a class Employee as follows:

Add the instance variables to represent name, department and salary

Implement the required constructor(s) and accessors.

Override the method toString() so that the format of the output is in accordance with those in the test cases.

Define a function query that takes a list of employees, a department and a salary as input. It returns a stream comprising the Employee objects that have the same department and have salary greater and equal to the given salary.

```java
import java.util.*;
import java.util.stream.*;

class Employee{
    private String name;
    private String dept;
    private int salary;
    public Employee(String n, String d, int s) {
        name = n;
        dept = d;
        salary = s;
    }
    public String get_name() {
        return name;
    }
    public String get_dept() {
        return dept;
    }
    public int get_salary() {
        return salary;
    }
    public String toString() {
        return name + " : " + dept + " : " + salary;
    }
}
```

JAVA – GrPA Solutions

```
class FClass{
    public static Stream<Employee> query(
            List<Employee> eList, String d, double s){
        Stream<Employee> st = eList.stream()
            .filter(n -> (n.get_dept().equals(d)
                && n.get_salary() >= s));
        return st;
    }
 public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        var eList = new ArrayList<Employee>();
        eList.add(new Employee("Jack", "HR", 30000));
        eList.add(new Employee("Aria", "HR", 40000));
        eList.add(new Employee("Nora", "IT", 50000));
        eList.add(new Employee("Bella", "IT", 60000));
        eList.add(new Employee("Jacob", "IT", 70000));
        eList.add(new Employee("James", "HR", 80000));
        String d = sc.next();        //read department
        double s = sc.nextInt();     //read salary

        var st = query(eList, d, s);
        st.forEach(n -> System.out.println(n + " "));
    }
}
```

JAVA – GrPA Solutions

Naresh (aka Customer c1) buys a set of items from a shop. Suresh (aka Customer c2) also buys all items bought by Naresh except the first item, in place of which Suresh buys another item. Write a program that defines two classes ***Items*** and ***Customer***, and clones the object of class ***Customer*** to model the scenario given above. Classes ***Items*** and ***Customer*** should be cloneable, and must have the functionality to clone (deep copy) c2 from c1. You are given as input the number of items bought by Naresh, the names of the items, and the new item that Suresh will be buying. The code to change the first item and the name in the second customer object after the cloning, has been provided in the given code. You should complete the program as specified below.

Define a class ***Items*** that implements interface ***Cloneable***, and has the following members.

– A public instance variable item of type String[] to store the item names

– Constructor(s) and accessors to, respectively, initialize and access the instance variable

– Override the method ***clone***

– Override the method ***toString*** so that the format of the output is in accordance with those in the test cases

Define a class ***Customer*** that implements interface ***Cloneable***, and has the following members.

– Instance variable name of type String to store the name of the customer

– Instance variable of type ***Items*** to store the items purchased by the customer

– Implement the constructor(s), the accessor ***getItems***() to return the object of Items, and the mutator ***setName(String s)*** to update the name of the customer.

– Override the method ***clone***

– Override the method ***toString*** so that the format of the output is in accordance with those in the test cases.

JAVA – GrPA Solutions

```java
import java.util.*;
class Items implements Cloneable{
  public String[] item;
  public Items(String[] a){
    item = a.clone();
  }
  public Items clone() throws CloneNotSupportedException{
    Items it = (Items) super.clone();
    it.item = (String[]) item.clone();
    return it;
  }
  public String toString(){
    String s = "";
    for(int i = 0; i < item.length; i++){
      s = s + item[i]+ " ";
    }
    return s;
  }
}
class Customer implements Cloneable{
  String name;
  Items i;
  public Customer(String n, Items i){
    this.name = n;
    this.i = i;
  }
  public Items getItems(){
    return this.i;
  }
  public void setName(String s){
    name = s;
  }
  public Customer clone() throws CloneNotSupportedException{
    Customer c = (Customer)super.clone();
    Items nitem = c.getItems().clone();
    c.i = nitem;
    return c;
  }
  public String toString(){
    return name + " " + i;
  }
}
```

JAVA – GrPA Solutions

```java
public class Order {
  public static void main(String[] args) throws CloneNotSupportedException{
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt(); // number of items
    String[] itm = new String[n];
    for(int i = 0; i < n; i++){
      itm[i] = sc.next(); // list of items
    }
    var c1 = new Customer("naresh", new Items(itm));
    Customer c2 = c1.clone();
    c2.getItems().item[0] = sc.next();   //Update first item of c2
    c2.setName("suresh"); //Update name of c2
    System.out.println(c1);
    System.out.println(c2);
  }
}
```