Week 2

Ppa1

Consider the program given below. Complete the program according to the instructions provided in the comments such that the program satisfies the given test cases.

```
import java.util.*;
class Rectangle{
   int w;    //width
   int h;    //height
//LINE-1: write the function setw(int) to initialize w
            public void setw(int a)
            {
                w=a;
            }
//LINE-2: write the function seth(int) to initialize h
            public void seth(int b)
            {
               h=b;
            }
//LINE-3: write the function area() to return area of rectangle
   public int area()
   {
      int a=w*h;
      return(a);
   }
}
public class FClass{
```

```
        public static void main(String[] args) {
      Scanner sc= new Scanner(System.in);
      int w = Integer.parseInt(sc.nextLine());
      int h = Integer.parseInt(sc.nextLine());
      Rectangle r = new Rectangle();
      r.setw(w);
      r.seth(h);
      int area = r.area();
      System.out.print(area);
   }
}
```

Ppa2

```
import java.util.*;
class FClass {

  public static void main(String[] args){
    Scanner sc = new Scanner(System.in);
    String s1 = sc.next();
    evenDisplay(s1);
  }
//Define evenDisplay(String) method here
  public static void evenDisplay(String s)
  {
     int l=s.length();
     for(int i=0;i<l;i+=2)
     {
        System.out.print(s.charAt(i));
     }
  }
}
```

Grpa1

Write a program to find the sum of the following series up to n terms.
$1^2 + (1^2 + 2^2) + (1^2 + 2^2 + 3^2) + ... + (1^2 + 2^2 + ... + n^2)$

```
import java.util.*;
public class SeriesSum {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
//Fill your code here
int sum=0;
for(int i=1;i<=n;i++)
{
   for(int j=1;j<=i;j++)
   {
      sum+=Math.pow(j,2);
   }
}
System.out.println(sum);

  }
}
```

Grpa2

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";
		}
//Define the required methods
public Employee(String name, String id, String dept) {
      this.ename = name;
      this.eid = id;
      this.edept = dept;
   }

   public void copyDept(Employee e) {
      this.edept = e.edept;
   }

   public void displayDetails() {
      System.out.println("ename : " + ename);
      System.out.println("eid : " + eid);
      System.out.println("edept : "+ 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();
            }
}
```

Grpa3

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.*;

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;
    }

}
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();
    }
}
```

Week3

Ppa1

Write a class named *Calculator* that has the following methods:

- *sum(double a, double b)* that prints the value of a + b
- *subtraction(double a, double b)* that prints the value of a - b
- *multiply(double a, double b)* that prints the value of a * b
- *division(double a, double b)* that prints the value of a / b

Write another class named *UpdatedCalculator* that inherits all the methods of *Calculator* and also has the following method:

- *remainder(double a, double b)* that prints the value of a % b

```
import java.util.*;
   class Calculator{
      // Fill the code
      public void sum(double a,double b)
      {
         System.out.println(a+b);
      }
      public void subtraction(double a,double b)
      {
         System.out.println(a-b);
      }
      public void multiply(double a,double b)
      {
         System.out.println(a*b);
      }
      public void division(double a,double b)
      {
         System.out.println(a/b);
      }
  }
   class UpdatedCalculator extends Calculator{
      // Fill the code
      public void remainder(double a,double b)
      {
         System.out.println(a%b);
      }
}
    public class CalculatorCheck{
      public static void main(String[] args) {
         Scanner sc = new Scanner(System.in);
```

```
        double n1 = sc.nextDouble();
        double n2 = sc.nextDouble();
        Calculator c = new Calculator();
        c.sum(n1, n2);
        c.subtraction(n1, n2);
        c.multiply(n1, n2);
        c.division(n1, n2);
        UpdatedCalculator uc = new UpdatedCalculator();
        uc.remainder(n1, n2);
      }

  }
```

Ppa2

Consider the following Java program.

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

```
import java.util.*;
class Point{
   private int x, y;
 // implement the constructor and
   // override the toString() and equals() methods
   public Point(int x1, int y1){
    this.x = x1;
    this.y = y1;
   }

   public String toString(){
```

```
        return "(" + this.x + ", " + this.y + ")";
    }
    // override the toString() and equals() methods
    public Boolean equals(Point p){
        if(p.x == this.x && p.y == this.y){
            return true;
        }else{
            return false;
        }
    }

}

class FClass{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x1 = sc.nextInt();
        int y1 = sc.nextInt();
        int x2 = sc.nextInt();
        int y2 = sc.nextInt();

        Point p1 = new Point(x1, y1);
        Point p2 = new Point(x2, y2);

        if(p1.equals(p2))
            System.out.println(p1 + "==" + p2);
        else
            System.out.println(p1 + "!=" + p2);
    }
}
```

Grpa1

```
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 name,long aadharno,double sal)
   {
      super(name,aadharno);
      this.salary=sal;
   }

   //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;
    double salary;

    //implement the constructor
    public ContactEmployee(String name,long aadharno,int contactHour)
    {
        super(name,aadharno,contactHour*hourlyPay);
        this.contactHour=contactHour;
    }

    //salary is computed as contactHour * hourlyPay
 public void print(){
        super.print();
 }

}

class FClass{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String nm1 = sc.nextLine();
        String nm2 = sc.nextLine();
        long adh1 = sc.nextLong();
        long adh2 = sc.nextLong();
        double sal = sc.nextDouble();
        int cont = sc.nextInt();
```

```
        Employee[] eArr = new Employee[2];
        eArr[0] = new Employee(nm1, adh1, sal);
        eArr[1] = new ContactEmployee(nm2, adh2, cont);
        for(Employee e : eArr)
            e.print();
    }
}
```

Grpa2

```
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 w,int h)
{
    this.w=w;
    this.h=h;
}
public int area()
{
    return(w*h);
}
```

```
}

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

}

class FClass{
   private static void caller(Shape s) {

      if(s instanceof Rectangle) {
         //check if s is of type Rectangle
         System.out.println(s.area());
      }
      else if(s instanceof Cube) {
         //check if s is of type 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));
    }
}
```

Grpa3

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 accno,String name,int bal)
{
   this.accountNumber=accno;
   this.name=name;
   this.balance=bal;
}
public void balance()
{
   System.out.println(balance);
}
public void deposit(int amount)
{
   balance=balance+amount;
}
public void withdraw(int amt)
{
   if(checkMinBalance(amt))
   {
      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();
        }

    }
```

Week4

Ppa1

```
import java.util.*;
interface Searchable{
    public int search(int start_index, Object key);
}
class Char{
    private char c;
    public Char(char c_) {
        c = c_;
```

```
    }
    public boolean equals(Object d) {
//implement equals() for Char
Char ch = (Char) d;
        if (c == ch.c ){
            return true;
        }else {
            return false;
        }
    }
}
class CharArray implements Searchable{
    private Char[] carr;
    public CharArray(Char[] carr_){
        carr = carr_;
    }
    public int search(int start_index, Object key) {
        //search the key in array carr from the index start_index

        for (int i = start_index; i < carr.length; i++){
            if (carr[i].equals(key)){
                return i;
            }
        }
        return -1;
        //if the key found, return index of the first occurrence of the key

        //else return -1
    }
}
```

```
class FrequencyCounter{
    public static int getFrequency(Searchable ob, Object key){
        int i=0;
        if (ob instanceof CharArray) {
            //count occurrences of the key in ob using search function
            int count = 0 , j = 0;
            j = ob.search(i, key);
            while (j > -1){
                if ( j != -1){
                    count += 1;
                }
                j = ob.search(j+1 ,key);
            }
            return count;

        }
        else
            return 0;
    }
}
class FClass{
    public static void main(String[] args) {
        String str;
        char c;
        Scanner sc = new Scanner(System.in);
        str = sc.nextLine();
        c = sc.next().charAt(0);
        Char key = new Char(c);
        Char[] cA = new Char[str.length()];
        for(int i = 0; i < str.length(); i++) {
            cA[i] = new Char(str.charAt(i));
```

```
        }
        CharArray caObj = new CharArray(cA);
        System.out.println(FrequencyCounter.getFrequency(caObj, key));
    }
}
```

Ppa2

Both the Voter and EVM classes must be created in such a way that they enforce the existence of only a single instance at a time. Each Voter object must be mapped with a unique EVM object and vice versa. A Voter must be allocated an EVM and then the voting process should start, once voting is completed that particular EVM should be freed and the next voter should be called.

Again a new EVM must be allocated to the next voter like previously and the process continues till all the voters cast their votes.

```
import java.util.Scanner;

class Voter{

// Define appropriate variables for implementing singleton behaviour
    // in accordance with the given coded parts and sample output
    static Voter new_voter;
    static int total_no_of_voters;
    static int current_voter_count;

    private Voter() {
        // System.out.println("1 Inside firstvoter " + "EVm no " + EVM.evm_count +
        //  " voterno " + Voter.current_voter_count);
                            current_voter_count++;
```

```
}

public static Voter getVoter() {
   //implement singleton behaviour
   // System.out.println("2 Inside firstvoter " + "EVm no " + EVM.evm_count +
   //  "voterno " + Voter.current_voter_count + new_voter);
   if(new_voter == null){
      new_voter = new Voter();
      return new_voter;
   }return null;
}

public void firstVoter(){
   // System.out.println("3 Inside firstvoter " + "EVm no " + EVM.evm_count +
   //  "voterno " + Voter.current_voter_count );
                        if(new_voter != null) {
                                EVM new_machine = EVM.getEVM(new_voter);
                                new_machine.startVoting();
                        }
}

public void callNewVoter() {
   // Write code to setup a new EVM object for the new voter
   // System.out.println("4 Inside firstvoter " + "EVm no " + EVM.evm_count +
   //  "voterno " + Voter.current_voter_count);
   if(Voter.current_voter_count < Voter.total_no_of_voters){
      Voter v = Voter.getVoter();
      EVM evm = EVM.getEVM(v);



   //Ignore the following 6 lines of code
```

```
        //but do not delete this code in your submission
//=========================================================================
        try {
                                EVM x = EVM.getEVM(null);
                                        x.startVoting();
                            }catch(NullPointerException e) {
                                        System.out.println("EVM is Singleton");
                    }
//=========================================================================
                    // Resume writing your code here
                evm.startVoting();
                 }    // Hint: Write code to start voting for the new user on the new EVM
    }
}

class EVM{

    // Define appropriate variables for implementing singleton behaviour
    // in accordance with the given coded parts and sample output
        static Voter current_voter;
        static EVM new_evm;
        static int evm_count;

        private EVM(Voter v) {
        //     System.out.println("a Inside firstvoter " + "EVm no " + EVM.evm_count +
        //  "voterno " + Voter.current_voter_count);
            current_voter = v;
                                evm_count++;
        }
        public static EVM getEVM(Voter v) {
            // Implement singleton behaviour
```

```
//     System.out.println("b Inside firstvoter " + "EVm no " + EVM.evm_count +
//  "voterno " + Voter.current_voter_count);
   if (new_evm == null){
      new_evm = new EVM(v);
      return new_evm;
   }
   return null;
}

public void startVoting() {
   // Complete voting for the current voter and call the next voter
   // Hint : Use callback here
//     System.out.println("c Inside firstvoter " + "EVm no " + EVM.evm_count +
//  "voterno " + Voter.current_voter_count);
   System.out.println("voting under process on EVM number "+EVM.evm_count);
   System.out.println("Voting completed for voter "+Voter.current_voter_count);

   Voter.new_voter = null;
   EVM.new_evm = null;

   EVM.current_voter.callNewVoter();
}
}


public class Election{
        public static void main(String args[]) {
                Scanner s = new Scanner(System.in);
                Voter.total_no_of_voters =  s.nextInt();
                // Assume input is always non zero
                Voter v = Voter.getVoter();
```

```
                    //Trying to create another voter when one voter is in the middle of
                    //voting process, students can ignore this try-catch block of code

                                try {
                                        Voter x = Voter.getVoter();
                                        x.callNewVoter();
                                } catch(NullPointerException e) {
                                        System.out.println("Voter is Singleton");
                                }

                    //Starting the first vote of the day
                      v.firstVoter();
            }
}
```

Grpa1

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 S = "";

        for (int i = 0; i < s.length(); i++){
            S = s.charAt(i) + S;
        }
        return (S);
    }
}
class UpdatedStrings extends StringReverse{
    public int vowelCount(String s){
        int count = 0;
        for (int j = 0; j < s.length(); j++){
            if (s.charAt(j) == 'a' || s.charAt(j) == 'e' || s.charAt(j) == 'i' || s.charAt(j) == 'o' || s.charAt(j) ==
'u'){
                count += 1;
            }
        }
        return count;
    }
}
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));
  }
}
```

Grpa2

```
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;
//implement the parameterized constructor to initialize size
    int p=0;
    int iter=0;
    public Sequence(int s){
        size=s;
        this.iArr=new int[s];
    }
//implement addTo(elem) to add an int elem to the sequence
public void addTo(int n){
        this.iArr[this.p]=n;
        this.p++;
    }
```

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

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() + ", ");
    }
}
```

Week 5

Ppa1

Given a class name as input, complete the Java code to print the count of public and declared methods, fields and constructors in the class. For each method in class `ClassStats` below, fill in the missing code as described in the comments. Each method takes the class name as input.

```
import java.lang.reflect.*;
import java.util.*;
class ClassStats{
public static int getPubMethodCount(String cname) {
        try {
           //add code to return the count of
           //public methods in the given class
           Class c = Class.forName(cname);
           Method[] m = c.getMethods();
           return m.length;
        }catch(Exception e) { return -1; }
    }
    public static int getAllMethodCount(String cname) {
        try {
```

```
        //add code to return the count of all
        //declared methods in the given class
       Class c = Class.forName(cname);
        Method[] meth = c.getDeclaredMethods();
        return meth.length;
    }catch(Exception e) { return -1; }
}
public static int getPubFieldCount(String cname) {
    try {
        //add code to return the count of
        //public fields (instance variables) in the given class
        Class c = Class.forName(cname);
        Field[] f = c.getFields();
        return f.length;
    }catch(Exception e) { return -1; }
}
public static int getAllFieldCount(String cname) {
    try {
        //add code to return the count of
        //all fields (instance variables) in the given class
         Class c = Class.forName(cname);
        Field[] fil = c.getDeclaredFields();
        return fil.length;
    }catch(Exception e) { return -1; }
}
public static int getPubContCount(String cname) {
    try {
        //add code to return the count of
        //public constructors in the given class
       Class c = Class.forName(cname);
        Constructor[] co = c.getConstructors();
```

```
            return co.length;
        }catch(Exception e) { return -1; }
    }
    public static int getAllContCount(String cname) {
        try {
            //add code to return the count of
            //all constructors in the given class
            Class c = Class.forName(cname);
            Constructor[] con = c.getDeclaredConstructors();
            return con.length;
        }catch(Exception e) { return -1; }
    }
}

class FClass{
    public static void main(String[] args) {
        String cname;
        Scanner sc = new Scanner(System.in);
        cname = sc.nextLine();
        System.out.println("Constructor: " +
                    ClassStats.getPubContCount(cname) + ", " +
                    ClassStats.getAllContCount(cname));
        System.out.println("Fields: " +
                    ClassStats.getPubFieldCount(cname) + ", " +
                    ClassStats.getAllFieldCount(cname));
        System.out.println("Methods: " +
                    ClassStats.getPubMethodCount(cname) + ", " +
                    ClassStats.getAllMethodCount(cname));
    }
}
```

Ppa2

Complete the Java code given below that takes as input a string array, where each string is assured to be either an integer or a double in string format. Your code must segregate the two types - integer and double - and print the double values followed by the integer values. For this, your code must iterate through the input array, and add each element to the appropriate array based on its type.

```
import java.util.Scanner;
class ConvertArrays{
        public Double doubleArr[]=new Double[3];
        public Integer intArr[]=new Integer[3];
        public int x=0,y=0,z=0;
        public void convert(String[] arr){
//loop through the arr and store each element
                    //in the appropriate array
                    for (String elem : arr){
                   if (elem.contains(".") ){
                    doubleArr[x] = Double.parseDouble(elem);
                    x++;
                   }
                   else {
                    intArr[y] = Integer.parseInt(elem);
                    y++;
                   }
                 }
}
        public <T> void display(T[] arr){
          for(T elements:arr)
```

```
                System.out.print(elements+" ");
            System.out.println();
        }
}
public class Programming {
        public static void main(String[] args) {
                Scanner scanner=new Scanner(System.in);
                String arr[]= new String[6];
                for (int i = 0; i < arr.length; i++) {
                        arr[i]=scanner.next();
                }
        ConvertArrays conArrays=new ConvertArrays();
        conArrays.convert(arr);
        System.out.println("===After conversion Arrays===");
        conArrays.display(conArrays.doubleArr);
        conArrays.display(conArrays.intArr);
    }
}
```

Grpa1

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.

```
import java.util.*;
//Add your code for ComplexNum here
class ComplexNum<T extends Number> {
   public Number real;
   public Number imag;

   public ComplexNum(Number real, Number imag) {
      this.real = real;
      this.imag = imag;
   }

   public <T extends Number> ComplexNum<T> add(ComplexNum<T> c) {
      return new ComplexNum<>(this.real.doubleValue() + c.real.doubleValue(),
this.imag.doubleValue() + c.imag.doubleValue());
   }
```

```
    public String toString() {
            return real.doubleValue() + " + " + imag.doubleValue() + "i";
    }
}
class FClass{
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n1, n2;
        double d1, d2;
        n1 = sc.nextInt();
        n2 = sc.nextInt();
        d1 = sc.nextDouble();
        d2 = sc.nextDouble();
        ComplexNum<Integer> c1 = new ComplexNum<Integer>(n1, n2);
        ComplexNum<Double> c2 = new ComplexNum<Double>(d1, d2);
        ComplexNum<Double> c3 = c1.add(c2);
        System.out.println(c1 + " + " + c2 + " = " + c3);
    }
}
```

Grpa2

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;

// Define constructor(s) as needed
   public ArrayExample(T[] a) {
                this.a = a;
        }
 // Define method display() that print the elements of array a
 public void display() {
                for (int i = 0; i < a.length; i++) {
                        System.out.print(a[i] + " ");
                }
                System.out.println();
```

```
        }
  // Define method elementCount(T x) that
  // counts the no. of times x is present in the array a
  public int elementCount(T x) {
                int count = 0;
                for (int i = 0; i < a.length; i++) {
                        if (a[i].equals(x)) {
                                count++;
                        }
                }
                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 the array
    Integer[] x = new Integer[len];
    for(int i = 0; i < len; i++){
        x[i] = sc.nextInt(); //Taking input for Integer array
    }

    //Write the code here to create an object obj for Integer array
    ArrayExample<Integer> obj = new ArrayExample<Integer>(x);

    int s1 = sc.nextInt(); //Taking input for the value to be counted
    String[] y = new String[len];
    for(int i = 0; i < len; i++){
        y[i] = sc.next(); //Taking input for String array
    }
```

```
    //Write the code here to create an object obj1 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));
  }
}
```

Week6

Grpa1

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.

```
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;
  }
}
public class Main {
// Define displayPlayers() method here

// Define displayPlayers() method here

public static void displayPlayers(ArrayList<CricketPlayer> bw, ArrayList<CricketPlayer> bt)
 {
    for(int i = 0; i < bw.size(); i++){
        System.out.print(bw.get(i).getName() + " ");
    }
    System.out.println("");
    for(int i = 0; i < bt.size(); i++){
        System.out.print(bt.get(i).getName() + " ");
    }
```

```
}

public static void main(String[] args) {
  Scanner sc = new Scanner(System.in);
  CricketPlayer p1 = new CricketPlayer(sc.next(), sc.nextInt(),
                    sc.nextInt(), sc.nextInt());
  CricketPlayer p2 = new CricketPlayer(sc.next(), sc.nextInt(),
                    sc.nextInt(), sc.nextInt());
  CricketPlayer p3 = new CricketPlayer(sc.next(), sc.nextInt(),
                    sc.nextInt(), sc.nextInt());
  CricketPlayer p4 = new CricketPlayer(sc.next(), sc.nextInt(),
                    sc.nextInt(), sc.nextInt());

  // Define ArrayList objects here

  // Batsman ArrayList

  ArrayList<CricketPlayer> bt = new ArrayList<CricketPlayer>();

  // Bowler ArrayList

  ArrayList<CricketPlayer> bw = new ArrayList<CricketPlayer>();

  // All Players ArrayList

  CricketPlayer[] all = {p1, p2, p3, p4};

  for(int i = 0; i < all.length; i++){
     if(all[i].avgRuns() > 25){
        bt.add(all[i]);
```

```
        }
        if(all[i].avgWickets() > 1){
            bw.add(all[i]);

        }
    }


 displayPlayers(bw, bt);
  }
}
```

Grpa2

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> brackets = new Stack<Character>();
        char[] left = { '{', '(', '[' };
        char[] right = { '}', ')', ']' };
        HashMap<Character, Character> map = new HashMap<>();

        map.put('(', ')');
        map.put('[', ']');
        map.put('{', '}');


        for (int i = 0; i < sequence.length(); i++) {
           char x = sequence.charAt(i);

           boolean leftContains = false;
           boolean rightContains = false;


           for(int j = 0; j < left.length; j++)
           {
              if(left[j] == x){
                 leftContains = true;
              }
           }

           for(int j = 0; j < right.length; j++)
           {
              if(right[j] == x){
                 rightContains = true;
              }
           }
```

```
            if (leftContains) {

                brackets.add(x);
            } else if (rightContains) {
                if(brackets.isEmpty()){
                    // System.out.println("Not balanced");
                    return false;
                }
                char y = brackets.pop();
                if (map.get(y) != x) {
                    // System.out.println("Not balanced");
                    return false;
                }
            }
        }

        // System.out.println("Balanced");
        if(brackets.isEmpty())
        {
            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");
        }
      }
   }
}
```

Live coding question

Wk2

Q1

Write the java program that creates 3 objects of students and prints the name of student with highest total. Implement the code as follows:

Define class `Student` with following members:

- `String name, double[] marks` (array length is 3) as instance variables which represents the name and marks of student
- Constructor to initialize instance variables
- Accessor method `getName()` that returns the name of student
- Method `findTotal( )` that returns the total marks of the student

Define class `Test` that has the following members:

- Method `getMax()` which takes `Student[]` as parameter and returns the name of the student with highest total.
- `main()` method, that creates 3 instances of `Student`, stores them in an array and invokes the method `getMax()` by passing that `Student[]`

```
import java.util.*;
class Student {
   private String name;
   private double marks [];
   public Student(String name, double[] marks) {
      this.name = name;
      this.marks = marks;
   }
   public String getName() {
      return (this.name);
   }
   public double findTotal() {
      double total = 0.0;
      for (double i : this.marks) {
         total = total + i;
      }
      return (total);
   }
}
public class Test {
   public static String getMax(Student[] student) {
      double max_marks = 0.0;
      String max_student = "";
      for (Student i : student) {
         double total_marks = i.findTotal();
         if (total_marks > max_marks) {
            max_student = i.getName();
            max_marks = total_marks;
```

```
        }
      }
      return max_student;
    }

    public static void main(String[] args) {
      Scanner sc = new Scanner(System.in);
      Student[] arr = new Student[3];
      for(int i = 0; i < 3; i++){
        String name = sc.next();
        double[] m = {sc.nextDouble(), sc.nextDouble(), sc.nextDouble()};
        arr[i] = new Student(name, m);
      }
      System.out.println(getMax(arr));
    }
}
```

Q2

`Employee e1` does a set of projects. `Employee e2` also does all the projects did by `e1` except the first project, in place of which `e2` does another project. Write a program that defines two classes `Employee` and `Test`. Define copy constructor to create `e2` from `e1` in such a way that changing the values of instance variables of either `e2` or `e1` should not affect the other one. The code takes name of `e2` and new project done by `e2` as input. Complete the program as specified below.

- Class `Employee` that has the following members.
  - Private instance variables `String name` and `String[] projects` to store name and projects respectively
  - Define required constructor(s)
  - Accessor methods `getName( )` and `getProject( )` to get name of employee and project at specific index.
  - Mutator methods `setName( )` and `setProject( )` to set name of employee and project at specific index.
- Class `Test` that has the method `main` which does the following.
  - Two objects of `Employee e1` and `e2` are created. `e2` is created using `e1`
  - name of `e2` and second item bought by `c2` are updated by taking the input
  - name of `e1`, `e2` and first project done by `e1` and `e2` are printed

```
import java.util.*;
class Employee{
   String name;
   String[] projects;
   public Employee(String n, String[] proj){
      name = n;
      projects = proj;
   }
   public Employee(Employee e){
      this.name = e.name;
      int l = e.projects.length;
      this.projects = new String[l];
      for(int i = 0; i < l; i++){
         this.projects[i] = e.projects[i];
      }
   }
   public void setName(String n) {
      name = n;
   }
   public void setProject(int index, String proj) {
      projects[index] = proj;
   }
   public String getName() {
      return name;
   }
   public String getProject(int indx) {
      return projects[indx];
   }
}
public class Test {
   public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        String[] proj =  {"PJ1", "PJ2", "PJ3"};
        Employee e1 = new Employee("Surya", proj);
        Employee e2 = new Employee(e1);
        e2.setName(sc.next());
        e2.setProject(0, sc.next());
        System.out.println(e1.getName() + ": " + e1.getProject(0));
        System.out.println(e2.getName() + ": " + e2.getProject(0));
    }
}
```

Wk3

Q1

Write the Java code as instructed.

- Class `Faculty` has the following members.
  - `String name`, `double salary` as private instance variables
  - Constructor to initialize the instance variables
  - Method `public double bonus(float percent)` that returns the bonus computed as (percent/100.0)*salary
  - You should define method `getDetails()` to display `name` and `salary` of an `Faculty`
  - You should overload method `getDetails()` as `getDetails(float percent)` to display the `name`, `salary` and `bonus` of an `Faculty`
- Class `Hod` extends class `Faculty` and has the following members.
  - `String personalAssistant` as private instance variable
  - Constructor to initialize the instance variables of `Hod` that includes the instance variables of `Faculty`
  - You should override method `public double bonus(float percent)` that returns the bonus of a `Hod` as 50 percent less bonus than a regular faculty
  - You should override method `getDetails()` to display the `name`, `salary` and `personalAssistant` of `Hod`.
  - You should override method `getDetails(float percent)` to display the `name`, `salary`, `personalAssistant` and `bonus` of a `Hod`
- Class `InheritanceTest` has the `main` method and the following functionality.
  - It creates objects of `Faculty` and `Hod` types, and also accepts the required input values.
  - Invokes methods `getDetails()` and `getDetails(float percent)` on `Faculty` and `Hod` objects.

```
import java.util.Scanner;
class Faculty{
  private String name;
  private double salary;
  public Faculty(String name, double salary) {
     this.name = name;
     this.salary = salary;
  }
  public double bonus(float percent){
     return (percent/100.0)*salary;
  }
   public String getDetails() {
      return name + ", " + salary;
   }
   public String getDetails(float percent ) {
      return getDetails()+ ", bonus = "+bonus(percent);
   }
}
class Hod extends Faculty{
   private String personalAssistant;
   public Hod(String name, double salary, String pa) {
      super(name, salary);
      this.personalAssistant = pa;
   }
   public double bonus(float percent){
      return 0.5*super.bonus(percent);
   }
   public String getDetails() {
      return super.getDetails()+", "+ personalAssistant;
   }
   public String getDetails(float percent ) {
```

```
        return getDetails()+", "+bonus(percent);
    }
}
public class InheritanceTest{
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        Faculty obj1 = new Faculty(sc.next(), sc.nextDouble());
        Faculty obj2 = new Hod(sc.next(), sc.nextDouble(), sc.next());
        System.out.println(obj1.getDetails());
        System.out.println(obj1.getDetails(10));
        System.out.println(obj2.getDetails());
        System.out.println(obj2.getDetails(10));
    }
}
```

Wk4

Q1

Write a Java program as instructed.

- Abstract class `UPIPayment` has two abstract methods: `abstract void payment()` and `abstract void rewards()`
- Class `PhonePay` extends class `UPIPayment` and has the following members.
  - `private int amount` as private instance variable
  - Constructor to initialize the instance variable
  - You should override method `public void payment()` such that the overridden method displays the message `"Phone pay:Payment success:"` and invokes method `rewards()`.
  - You should override method `public void rewards()` as follows.
    - If `amount < 500`, then display the message `"Sorry no rewards"`.
    - Else, if `amount >= 500`, then display the message `"10 off on next mobile rc"`.
- Class `Paytm` extends class `UPIPayment` and has the following members.
  - `private int amount` as private instance variable
  - Constructor to initialize the instance variable
  - You should override method `public void payment()` such that the overridden method displays the message `"Paytm:Payment success:"` and invokes method `rewards()`.
  - You should override method `public void rewards()` as follows.
    - If `amount < 500`, then display the message `"Sorry no rewards"`.
    - Else, if `amount >= 500`, then display message `"10 off on next DTH rc"`.
- Class `UPIUser` has the following method.
  - Method `public void transferAndGetRewards(UPIPayment obj)` that invokes method `payment()` using `obj`.
- Class `AbstractTest` has the `main` method, and has the following functionality.
  - Creates an object of `UPIUser`
  - Takes two amounts `a1` for PhonePay and `a2` for Paytm
  - Invokes method `transferAndGetRewards()` first by passing an object of `PhonePay` with amount `a1` as parameter, and then by passing an object of `Paytm` with amount `a2` as parameter.

```
import java.util.*;
abstract class UPIPayment{
   abstract void payment();
   abstract void rewards();
}
class PhonePay extends UPIPayment{
   private int amount;
   public PhonePay(int amount) {
      this.amount = amount;
   }
   public void payment() {
      System.out.println("Phone pay:Payment success:");
      rewards();
   }
   public void rewards() {
      if(amount < 500)
         System.out.println("Sorry no rewards");
      else if(amount >= 500)
         System.out.println("10 off on next mobile rc");
   }
}
class Paytm extends UPIPayment{
   private int amount;
   public Paytm(int amount) {
      this.amount = amount;
   }
   public void payment() {
      System.out.println("Paytm:Payment success:");
      rewards();
   }
   public void rewards() {
```

```
        if(amount < 500)
            System.out.println("Sorry no rewards");
        else if(amount >= 500)
            System.out.println("10 off on next DTH rc");
    }
}
class UPIUser{
    public void transferAndGetRewards(UPIPayment obj) {
        obj.payment();
    }
}
public class AbstractTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int a1 = sc.nextInt();
        int a2 = sc.nextInt();
        UPIUser u = new UPIUser();
        u.transferAndGetRewards(new PhonePay(a1));
        u.transferAndGetRewards(new Paytm(a2));
    }
}
```

Q2

Write Java code as instructed.

- Define an interface `Appraisable` that has the following members:
  - Default method `default void appraisal(Teacher t)` that increments the `salary` of an `Employee` by `(stuPassPer/100)*5000`.
  - Abstract method `public abstract void checkAndUpdateSalary()`
- Define an interface `SpecialAppraisable` that extends `Appraisable` and has the following members:
  - Default method `default void spAppraisal(Teacher t)` that increments the salary of an `Employee` by `(stuPassPer/100)*10000`.
- Class `Teacher` that implements the interface `SpecialAppraisable` and has the following members:
  - `String name`, `double salary` and `private double stuPassPer` as private instance variables
  - Constructor to initialize the instance variables
  - Mutator method to update `salary`
  - Accessor method to access `salary`
  - Accessor method to access `stuPassPer`
  - Override method `toString()` that returns `name`, `salary` and `stuPassPer` of the `Teacher` as a single concatenated string (each separated by a single space)
  - Overriding method `public void checkAndUpdateSalary()` that has the following functionality.
    - If `stuPassPer>=60 and stuPassPer<75` then invoke the `appraisal()` method from `Appraisable` interface
    - Else, if `stuPassPer>=75 and stuPassPer<=100` then invoke the`spAppraisal()` method from SpecialAppraisable interface
- Class `InterfaceTest` that has the following members:
  - You should define method `public static void printUpdatedTeachList(Teacher[] tList)` that has the following functionality
    - Iterate over array `tList` and invoke method `checkAndUpdateSalary()` on each `Teacher` object.
    - Then, iterate over `tList` and display each `Teacher` object.
  - `main` method has the following functionality
    - Creates and initializes an array `tArr` of three `Teacher` objects
    - Invokes method `printUpdatedTeachList(Teacher[] tList)` to print the updated details of each `Teacher` after the appraisal is applied

```
import java.util.*;
interface Appraisable{
   default void appraisal(Teacher t){
      t.setSalary(t.getSalary()+(t.getstuPassPer()/100)*5000);
   }
   public abstract void checkAndUpdateSalary();
}
interface SpecialAppraisable extends Appraisable{
```

```
    default void spAppraisal(Teacher t){
        t.setSalary(t.getSalary()+(t.getstuPassPer()/100)*10000);
    }
}
class Teacher implements SpecialAppraisable{
    private String name;
    private double salary;
    private double stuPassPer;
    public Teacher(String name, double salary, double stuPassPer) {
        this.name = name;
        this.salary = salary;
        this.stuPassPer = stuPassPer;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
    public double getstuPassPer() {
        return stuPassPer;
    }
    public String toString() {
        return name + ", " + salary + ", " + stuPassPer;
    }
    public void checkAndUpdateSalary() {
        if(stuPassPer >= 60 && stuPassPer < 75)
            appraisal(this);
        else if(stuPassPer >= 75 && stuPassPer <= 100)
            spAppraisal(this);
     }
```

```
}
public class InterfaceTest {
   public static void printUpdatedTeachList(Teacher[] tList) {
      for (int i = 0; i < tList.length; i++)
         tList[i].checkAndUpdateSalary();
      for (int i = 0; i < tList.length; i++)
         System.out.println(tList[i]);
   }
public static void main(String[] args) {
Scanner sc = new Scanner(System.in);
Teacher tArr[] = new Teacher[3];
for (int i = 0; i < tArr.length; i++)
tArr[i] = new Teacher(sc.next(), sc.nextDouble(), sc.nextDouble());
InterfaceTest.printUpdatedTeachList(tArr);
}
}
```

Q3

Write a Java program that accepts list of `student` objects and updates the regular fees, based on the number of backlogs that each student has.

- An interface `Generatable` that has the following member:
  - Abstract method `abstract void feeGenerate(Student s)`
- Class `Student` that has the following members:
  - `String name`, `double fee` and `int backlogs` as private instance variable.
  - Constructor to initialize the `name` and `backlogs`.
  - Mutator method to update the `fee`
  - Accessor method to access the `backlogs`
  - Override method `toString()` that returns `name`, `fee` and `backlogs` of the `Student` as a single concatenated string (each separated by comma(,))
- Class `ExamBranch` that has the following members:
  - A private inner class `Regular` that implements the interface `Generatable` and overrides method `public void feeGenerate(Student s)` that, in turn, initializes the regular student fee as 1500.
  - A private inner class `Supple` that implements the interface `Generatable` and overrides method `public void feeGenerate(Student s)` that, in turn, has the following functionality.
    - If the student `backlogs == 1`, then update the student fee to 2000.
    - Else, if the student `backlogs == 2`, then update the student fee to 2500.
    - Else, if the student `backlogs >= 3`, then update the student fee to 3500.
  - You should define method `getRegularFee()` that returns an object of `Regular` class.
  - You should define method `getSuppleFee()` that returns an object of `Supple` class.
- Class `PrivateClassTest` has the following members:
  - You should define method `public static Student[] getStudentsFee(Student sList[])` that does the following:
    - Iterates over array `sList` such that in each iteration, invoke method `feeGenerate(Student s)` on each `Student` object from `Regular` class, if student `backlogs == 0`. Else, invoke method `feeGenerate(Student s)` on each `Student` object from `Supple` class.
  - `main` method has the following functionality
    - Creates and initializes an array `sArr` of three `Student` objects
    - Invokes method `getStudentsFee(sArr)` to get the updated details of each `Student` after the fee is updated
    - Prints the updated list

```
import java.util.Scanner;

interface Generatable{

   abstract void feeGenerate(Student s);

}

class Student {
```

```
    private String name;
    private double fee;
    private int backlogs;
    public Student(String name, int backlogs) {
        this.name = name;
        this.backlogs = backlogs;
    }
    public void setFee(double fee) {
        this.fee = fee;
    }
    public int getBacklogs() {
        return backlogs;
    }
    public String toString() {
        return name + ", " + fee + ", " + backlogs;
    }
}
class ExamBranch{
    private class Regular implements Generatable{
        public void feeGenerate(Student s) {
            s.setFee(1500.00);
        }
    }
    private class Supple implements Generatable{
        public void feeGenerate(Student s) {
            if(s.getBacklogs() == 1)
                s.setFee(1500 + 500);
            else if(s.getBacklogs() == 2)
                s.setFee(1500 + 1000);
            else if(s.getBacklogs() >=3 )
                s.setFee(1500 + 2000);
```

```
    }
}
    public Generatable getRegularFee() {
        return new Regular();
    }
    public Generatable getSuppleFee() {
        return new Supple();
    }
}
public class PrivateClassTest {
    public static Student[] getStudentsFee(Student sList[]){
        ExamBranch obj;
        for (int i = 0; i < sList.length; i++) {
            if(sList[i].getBacklogs()==0) {
                obj=new ExamBranch();
                obj.getRegularFee().feeGenerate(sList[i]);
            }
            else {
                obj=new ExamBranch();
                obj.getSuppleFee().feeGenerate(sList[i]);
            }
        }
        return sList;
    }
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        Student[] sArr = new Student[3];
        for (int i = 0; i < sArr.length; i++) {
            sArr[i] = new Student(sc.next(), sc.nextInt());
        }
        sArr = PrivateClassTest.getStudentsFee(sArr);
```

```
    for (int i = 0; i < sArr.length; i++) {
      System.out.println(sArr[i]);
    }
  }
}
```

Wk5

Q1

Write a program that, given two integers, two doubles as `x` and `y` coordinates of two points `p1(x1,y1)` and `p2(x2,y2)` on a two-dimensional plane as input, finds the mid-point `p3(x3,y3)` of the line segment formed by `p1` and `p2` using the formula:

$$x3 = \frac{x1 + x2}{2} \text{ and } y3 = \frac{y1 + y2}{2}$$

Generic class `Point` has the following members.

- Instance variables `x` and `y`
- A constructor to initialize `x` and `y`
- A method `mid(Point p)` to return the mid-point of the line segment joining the current point to `p`
- A method that overrides the method `toString()` in the Object class to format the output

Class `Test` has the main method.

- The main method accepts the two input points. The first line of input will be two integers of point `p1`. The second line of input will be two doubles of point `p2`. It then invokes the method `mid` of one of the objects.

```
import java.util.*;
class Point<T extends Number>{
  T x, y;
  public Point(T x, T y) {
    this.x = x;
    this.y = y;
  }
```

```
    public Point<Double> mid(Point<?> p){
        Point<Double> pt = new Point<Double>(0.0, 0.0);
        pt.x = (this.x.doubleValue() + p.x.doubleValue()) / 2;
        pt.y = (this.y.doubleValue() + p.y.doubleValue()) / 2;
        return pt;
    }
    public String toString() {
        return "(" + x + "," + y + ")";
    }
}
public class Test{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Point<Integer> p1 = new Point<Integer>(sc.nextInt(), sc.nextInt());
        Point<Double> p2 = new Point<Double>(sc.nextDouble(), sc.nextDouble());
        Point<Double> p3 = p1.mid(p2);
        System.out.println(p3);
    }
}
```

Wk6

Q1

Write a Java program that takes as input 4 Shop objects and the list of `Shop` objects with attributes shop `name`, and number of items sold `nsold`. The program should add each customer name as key and number of items as value to the map object. After adding all objects to the map, display the shop name which has sold maximum number of items as shown in the test cases. Complete the program as specified below:

- Class `Shop` that has the following members:
  - `String name`, `int nsold` as private instance variable
  - Constructor to initialize the `name` and `nsold`
  - Accessor methods to all instance variables
- Class `MapTest` has the following members:
  - You should define method `public static void printShopName(ArrayList<Shop> sList)` that does the following:
    - Iterates over array `sList` such that in each iteration, add each customer name as key and number of items as value to the map object.
    - Print the shop name which has sold maximum number of items.
  - `main` method has the following functionality
    - Creates a list of 4 `Shop` objects.
    - Invokes method `printShopName(list)` to print the shop name which has sold maximum number of items.

```
import java.util.*;

class Shop{

   private String name;

   private int nsold;

   public Shop(String s, int ns){

         this.name = s;

         this.nsold = ns;

   }

   public String getName(){

         return name;

   }

   public int getItemSold(){

         return nsold;

   }
```

```
}
public class MapTest {
        public static void printShopName(ArrayList<Shop> sList) {
          Map<String, Integer> m = new LinkedHashMap<String, Integer>();
          String shop = "";
          int sold = 0;
for(Shop s: sList)
   m.put(s.getName(), m.getOrDefault(s.getName(),0)+s.getItemSold());

for (HashMap.Entry<String, Integer> entry : m.entrySet()){
  if(entry.getValue()> sold) {
        shop = entry.getKey();
     sold = entry.getValue();
  }
}
     System.out.println(shop+" : "+sold);
  }
  public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        ArrayList<Shop> list = new ArrayList<Shop>();
        for (int i = 0; i < 4; i++) {
                list.add(new Shop(sc.next(), sc.nextInt()));
        }
        printShopName(list);
  }
}
```