BSCCS2005: Proctored Exam 1 Questions with Test Cases

1. Write a program that, given a string `s` as input, prints the string obtained from removing all duplicate characters from `s` (that is, retain only the first occurrence of each character). The characters should appear in the output in the same order as they appear in `s`.

   Note: `str.charAt( arg )` *: returns the character present at index arg in the string str*
   Sample input 1:
   `eerie`
   Output:
   `eri`

   Sample input 2:
   `abcd`
   Output:
   `abcd`

```
import java.util.*;
public class StringDisplay {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    String s = sc.nextLine();

    // Fill the code here

  }
}
```

   Public test case 1:
   Input:

```
abababa
```

   Output:

```
ab
```

   Public test case 2:
   Input:

```
aeeffdc
```

Output:

```
aefdc
```

Private test case 1:
Input:

```
axxyyeedc
```

Output:

```
axyedc
```

2. It has been decided to give promotion to one of the four employees in the Sales department of a company based on the following criteria. An employee with the maximum number of years of experience is considered the ideal candidate for promotion. If two employees have maximum experience, then the one who has taken the minimum number of days of leave will be considered for promotion. Assume that no two of them have taken the same number of days of leave.

   Class `Employee` must implement the interface `Comparable`, and should have the following members.

   - Instance variables: `ID` (employee ID), `experience` (number of years of experience), `nleaves` (number of days of leave taken so far)
   - Override the method `equals(Object o)` such that it returns true if the two employees being compared have the same `ID`; else, return false.
   - Override the method `compareTo(Object obj)` to compare two `Employee` objects, based on `experience` and `nleaves`.

   Class `Company` has two methods:

   - The method `main()` in Class `Company` accepts the inputs to instantiate four objects of `Employee[]` type. The input is accepted in the order `ID`, `experience`, and `nleaves`, for each employee. It then invokes the method `displayID(e)`, which returns the `ID` of the employee who gets the promotion.
   - `int displayID(Employee[] emo)` returns the `ID` of the `Employee` object who gets the promotion, by using the method `compareTo(Employee e)` inside the class `Employee`.

   Sample Input:

```
10 2 1
10 2 1
11 3 1
12 3 0
```

   Output:

```
12
```

```
import java.util.*;
class Employee implements Comparable<Employee>{
  int ID;
  int experience;
  int nleaves;

  public Employee(int i, int e, int l){
    ID = i;
    experience = e;
    nleaves = l;
  }

  // Override equals(Object o) here
  // Override compareTo(Object o) here

}

public class Company {
  public static int displayID(Employee[] emp){
      int max = 0;
      int n = 0;
      for(int i = 0; i < 4; i++){
        int count = 0;
        for(int j = 0; j < 4; j++){
          if(!emp[i].equals(emp[j])){
            count = count + emp[i].compareTo(emp[j]);
          }
        }
        if(count > max){
          max = count;
          n = emp[i].ID;
        }
      }
      return n;
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Employee[] e = new Employee[4];
    e[0] = new Employee(sc.nextInt(), sc.nextInt(), sc.nextInt());
    e[1] = new Employee(sc.nextInt(), sc.nextInt(), sc.nextInt());
    e[2] = new Employee(sc.nextInt(), sc.nextInt(), sc.nextInt());
    e[3] = new Employee(sc.nextInt(), sc.nextInt(), sc.nextInt());
    int id = displayID(e);
    System.out.println(id);
  }
}
```

Public test case 1:

```
10 2 1
10 2 1
```

```
11 3 1
12 3 0
```

Output:

```
12
```

Public test case 2:

```
20 2 1
21 3 2
22 4 2
23 2 4
```

Output:

```
22
```

Private test case 1:
Input:

```
100 2 3
101 2 2
102 2 1
103 2 0
```

Output:

```
103
```

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

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

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 `x1` and `y1` of point `p1(x1,y1)`. The second line of input will be `x2` and `y2` of point `p2(x2,y2)`. It then invokes the method `mid` of one of the objects.

Sample input 1:

```
3 4
5 6
```

Output:

```
(4,5)
```

Sample input 2:

```
-3 4
-5 6
```

Output:

```
(-4,5)
```

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

//Add your code for Class Point here

```
public class Test{
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int x1, y1;
        int x2, y2;
        x1 = sc.nextInt();
        x2 = sc.nextInt();
        y1 = sc.nextDouble();
        y2 = sc.nextDouble();
        Point p1 = new Point(x1, y1);
        Point p2 = new Point(x2, y2);
        Point p3 = p1.mid(p2);
        System.out.println(p3);
    }
}
```

Public test case 1:

```
4 5
2 1
```

Output:

```
(3,3)
```

Public test case 2:

```
13 13
12 12
```

Output:

```
(12,12)
```

Private test case 1:

```
-11 2
13 -4
```

Output:

```
(1,-1)
```

4. Write a program that prints the roll number and the marks of the student scoring the highest marks among all students in a college, and the roll number, the marks and the department of the student scoring highest marks among the undergraduate (UG) students in that college. Your program should define two types - `Student` and its subtype `UGStudent`. It should accept the roll number and total marks of 3 students, of type `Student`, and the roll number, total marks, and department of 3 UG students, of type `UGStudent`.

   - The class `StudentList` contains a generic array which can store instances of `Student`/ `UGStudent` type. It also provides the iterator to traverse through that array which is implemented using an inner class named `Iter`. Assume that the total marks for each student is unique. Implement the following to complete the program and obtain the specified output.
   - In the class `UGStudent`
     - Complete the definition of constructor.
   - In the class `FClass`
     - Define the generic function `printTopper()` that uses an iterator of `StudentIterator` type to find the `Student` (or `UGstudent`) who obtained the highest total marks, and prints the details by calling the `print()` method.
   - In the class `Iter`
     - Complete the definition of `has_next()` method.

   Sample input 1:

```
10 78
11 67
12 98
101 56 EE
102 87 ME
103 33 CE
```

   Output:

```
12 : 98
102 : 87 :  ME
```

   Sample input 2:

```
1 87
2 67
3 9
1101 56 CSE
1012 76 CSE
1033 78 CSE
```

Output:

```
1 : 87
1033 : 78 :  CSE
```

```
import java.util.*;

interface StudentIterator{
    public boolean has_next();
    public Student get_next();
}

class Student{
    private int rollno;
    private int totalmarks;
    public Student(int rollno, int totalmarks) {
        this.rollno = rollno;
        this.totalmarks = totalmarks;
    }
    public int get_totalmarks() {
        return totalmarks;
    }
    public void print() {
        System.out.print(rollno + " : " + totalmarks);
    }
}
class UGStudent extends Student{
    private String department;

    // Define the appropriate constructor

    public void print() {
        super.print();
        System.out.print(" : " + department);
    }
}
class StudentList<T extends Student>{
    private T s_arr[];
    public StudentList(T[] s_arr) {
        this.s_arr = s_arr;
    }
    public StudentIterator getIterator() {
        return new Iter();
    }
    private class Iter implements StudentIterator{
        private int i = -1;
        public boolean has_next() {
           // Complete the definition of this method.
        }
        public Student get_next() {
            i++;
            return s_arr[i];
        }
    }
}

class FClass{

    //Define function printTopper

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

            Student[] s1 = new Student[3];
            UGStudent[] s2 = new UGStudent[3];

            for(int i = 0; i < 3; i++) {
                int r = sc.nextInt();
                int t = sc.nextInt();
                s1[i] = new Student(r, t);
            }

            for(int i = 0; i < 3; i++) {
                int r = sc.nextInt();
                int t = sc.nextInt();
                String d = sc.nextLine();
                s2[i] = new UGStudent(r, t, d);
            }
            StudentList<Student> sList = new StudentList<Student>(s1);
            printTopper(sList);
            System.out.println();
            StudentList<UGStudent> uList = new StudentList<UGStudent>(s2);
            printTopper(uList);
        }
}
```

Public test case 1:
Input:

```
10 78
11 67
12 98
101 56 EE
102 87 ME
103 33 CE
```

Output:

```
12 : 98
102 : 87 :  ME
```

Public test case 1:
Input:

```
1 87
2 67
3 9
1101 56 CSE
1012 76 CSE
1033 78 CSE
```

Output:

```
1 : 87
1033 : 78 :  CSE
```

Private test case 1:
Input:

```
12 78
13 55
14 90
15 7 CSE
16 99 CSE
17 77 IT
```

Output:

```
14 : 90
16 : 99 :  CSE
```

5. Write a program that, given an array `arr` of `n` integers as input, prints the average of the positive integers in `arr`, followed by the average of the negative integers in `arr`.

   Sample input 1:
   ```
   5
   -10 12 12 -15 12
   ```
   Output: `12.0 -12.5`

   Sample input 2:
   ```
   4
   -10 10 -10 10
   ```
   Output: `10.0 -10.0`

   ```
   public class PosNegAvg {
       double posAvg=0,negAvg=0;
       double posCount=0,negCount=0;
       public void avg(int arr[]) {
       //write your code here

       }
       public static void main(String[] args) {
           Scanner scanner=new Scanner(System.in);
           PosNegAvg obj=new PosNegAvg();
           int length=scanner.nextInt();
           int arr[]=new int[length];
           for (int i = 0; i < arr.length; i++) {
               arr[i]=scanner.nextInt();
           }
           obj.avg(arr);
           System.out.println(obj.posAvg);
           System.out.println(obj.negAvg);
       }
   }
   ```

   Public test case 1:
   Input:

   ```
   5
   23
   45
   -67
   -89
   54
   ```

Output:

```
40.666666666666664
-78.0
```

Public test case 2:
Input:

```
4
-10
20
-10
20
```

Output:

```
20.0
-10.0
```

Private test case 1:
Input:

```
4
-10
-20
-45
6
```

Output:

```
6.0
-25.0
```

6. Write a program that, given the names and the ages of 3 persons, finds the person eligible for vaccination based on the following criteria. People with any comorbidity are given preference over people with no such issues. If two or more of them have any comorbidity, then the youngest among them is given preference over the others. Assume that no two persons have the same age.

   Class `Person` should implement the interface `Comparable`, and should have the following members.

   - `String name`, `int age` and `boolean comorbidity` as instance variables
   - A parameterized constructor to initialize the instance variables
   - Accessor methods to return the value of the instance variables
   - Override method `int compareTo(Object o)` to decide who gets preference based on the criteria given above.
   - Override method `boolean equals(Object o)` such that it returns true if the two persons being compared have the same `name`; else, return false.

   Class `Test` has two methods.

   - The main method accepts the name, the age and the comorbidity status (true/false) of 3 persons. It creates an object of `ArrayList` of type `Person` to store the input objects.
   - It then finds the eligible person using the method `int compareTo(Object o)` in class `Person`.
   - The method `displayPersons(ArrayList<Person>)` prints the name of the person eligible for the vaccination.

   Sample Input:

```
ram 55 true
sita 22 true
geetha 20 false
```

   Output:

```
sita
```

```
import java.util.*;
class Person implements Comparable<Person>{
  private String name;
  private int age;
  private boolean comorbidity;
  public Person(String n, int a, boolean b){
    this.name = n;
    this.age = a;
    this.comorbidity = b;
  }
  public String getName(){
    return name;
  }
  public int getAge(){
    return age;
  }
  public boolean getComorbidity(){
    return comorbidity;
  }

   // Define compareTo() here

  // Override equals() here

  }
}

public class Test {
  public static void displayPerson(ArrayList<Person> l){
    String name = "";
    int max = 0;
    for(Person p1 : l){
      int count = 0;
      for(Person p2 : l){
        if(!p1.equals(p2)){
          count = count +  p1.compareTo(p2);
        }
      }
      if(count > max){
        max = count;
        name = p1.getName();
      }
    }
    System.out.println(name);
  }
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    Person p1 = new Person(sc.next(), sc.nextInt(), sc.nextBoolean());
    Person p2 = new Person(sc.next(), sc.nextInt(), sc.nextBoolean());
    Person p3 = new Person(sc.next(), sc.nextInt(),sc.nextBoolean());
    ArrayList<Person> l1 = new ArrayList<Person>();
    l1.add(p1);
    l1.add(p2);
    l1.add(p3);
    displayPerson(l1);
  }
}
```

Public test case 1:
Input:

```
ram 55 true
sita 22 true
geetha 20 false
```

Output:

```
sita
```

Public test case 2:
Input:

```
ravi 33 false
gopi 44 true
mahesh 12 false
```

Output:

```
gopi
```

Private test case 1:
Input:

```
abc 77 true
xyz 55 true
qrs 60 true
```

Output:

```
xyz
```

7. Write a program that, given two rational numbers `r1(p1/q1)` and `r2(p2/q2)` as input, where `p1,p2,q1,q2` are integers, finds the product `r3(p3/q3)` of `r1` and `r2`. Assume that neither `q1` nor `q2` will be zero.

   Class `Rational` should have the following members.

   - Instance variables `p` and `q`
   - A constructor to initialize `p` and `q`
   - A method `public Rational product(Rational r)` to compute the product `r3`, where `p3 = p1 * p2`, and `q3 = q1 * q2`.
   - A method that overrides the method `toString()` to print the output as `p3/q3`.

   Class `Test` has the main method.

   - The method `main()` should accept inputs `p` and `q`. The first line of input will be `p1` and `q1`. The second line of input will be `p2` and `q2`.

   Sample input 1:

```
3 4
3 4
```

   Output:

```
9/16
```

```
import java.util.Scanner;
class Rational{
    private int p;
    private int q;

//Define constructor

//Override method toString()

//Define public Rational product(Rational r)

}
public class Test {
    public static void main(String[] args) {
        Scanner scanner=new Scanner(System.in);
        Rational r1=new Rational(scanner.nextInt(),scanner.nextInt());
        Rational r2=new Rational(scanner.nextInt(),scanner.nextInt());
        Rational r3=r1.product(r2);
        System.out.println(r3);
    }
}
```

Public test case 1:
Input:

```
11 4
5 6
```

Output:

```
55/24
```

Public test case 2:
Input:

```
5 4
6 7
```

Output:

```
30/28
```

Private test case 1:
Input:

```
12 20
34 2
```

Output:

```
408/40
```

8. Complete the following program, which models a voter-chart generator and should work as detailed below.

   - The program accepts voting requests/registrations in the following format:
     - Number of female voter requests.
     - The `voter_id` and `age` of each female voter
     - Number of male voter requests
     - The `voter_id` and `age` of each male voter
   - All the requests are forwarded to the `voterChart` method which, in turn, validates the requests, and print the details according to the following criteria.
     - Each valid voting request should have a unique `voter_id`. In case there are multiple voting requests with the same voter id, only the first request should be considered, and the rest should be discarded.
       Hint: Use the `LinkedHashSet` *collection, and override the* ***equals*** *method to achieve this feature. The methods* ***equals*** *and* ***hashCode*** *are defined inside the* ***Object*** *class.*
     - Once all the duplicate voting requests are filtered out and unique valid requests are registered, the program should sort the registered requests in descending order of `age` of the voter and then print the details of all registered requests, in the format shown in public test cases.
       Hint: Use the `TreeSet` *collection, and override the* ***compareTo*** *method to achieve this feature. The method* ***compareTo*** *is declared inside the* ***Comparable*** *interface.*

   What has to be done:

   - Override *equals* and *compareTo* methods
   - Complete the definition of *voterChart* method

   Method signatures :

   - `public int compareTo(T obj)`, where T is a generic type.
   - `public boolean equals(Object obj)`

   Sample input 1:

```
3
1001 56
1002 34
1001 28
2
2001 45
2001 39
```

Output:
```
Female Voter:1001, age:56
Male Voter:2001, age:45
Female Voter:1002, age:34
```

Sample input 2:
```
2
1001 50
1001 27
3
1001 42
1001 33
1002 22
```

Output:
```
Female Voter:1001, age:50
Male Voter:1002, age:22
```

```
import java.util.*;
abstract class Voter implements Comparable<Voter>{
    String voter_id;
    int age;
    public Voter(String id,int a){
        voter_id = id;
        age = a;
    }
    public int hashCode() {
        // overriding hashCode to generate the object's id/hash code only
        // on the basis of voter_id
        return Integer.parseInt(voter_id);
    }

    // override compareTo method here
    // override equals method here

}
class FemaleVoter extends Voter{
    public FemaleVoter(String voter_id, int age) {
        super(voter_id,age);
    }
    public String toString() {
        return "Female Voter:"+voter_id+", age:"+age;
    }
}
class MaleVoter extends Voter{
    public MaleVoter(String voter_id, int age) {
        super(voter_id,age);
    }
    public String toString() {
        return "Male Voter:"+voter_id+", age:"+age;
    }
}

public class Exam4 {

    // Define voterChart method here

    public static void main(String args[]) {
        Scanner s = new Scanner(System.in);
        ArrayList<Voter> registrations = new ArrayList<Voter>();

        //reading the number of Female voters
        int female_voters = s.nextInt();
        for(int i=1;i<= female_voters;i++) {
            //reading voter_id
            String id = s.next();
            //reading age
            int age = s.nextInt();
            FemaleVoter f1 = new FemaleVoter(id,age);
            registrations.add(f1);
        }

        //reading the number of Female voters
        int male_voters = s.nextInt();
        for(int i=1;i<= male_voters;i++) {
            //reading voter_id
            String id = s.next();
            //reading age
            int age = s.nextInt();
            MaleVoter m1 = new MaleVoter(id,age);
            registrations.add(m1);
        }

        voterChart(registrations);
    }
}
```

Public test case 1:
Input:

```
3
1001 56
1002 34
1001 28
2
2001 45
2001 39
```

Output:

```
Female Voter:1001, age:56
Male Voter:2001, age:45
Female Voter:1002, age:34
```

Public test case 2:
Input:

```
2
1001 50
1001 27
3
1001 42
1001 33
1002 22
```

Output:

```
Female Voter:1001, age:50
Male Voter:1002, age:22
```

Private test case 1:
Input:

```
1
1001 49
3
2001 23
1001 53
2002 36
```

Output:

```
Female Voter:1001, age:49
Male Voter:2002, age:36
Male Voter:2001, age:23
```