COSC1095: Programming Principles 2J
Mid-Semester Test, Semester 1, 2004
SOLUTIONS

1. What will be assigned to y in the code segment below:

```
int x = 3;
y = x++ % ++x;
```

A. 0
B. 1
C. 2
D. 3
E. 4

ANSWER: D

2. How many '*' will the following program output to the screen:

```
int n = 5;

for (int i = 1; i <n; i++)  {
   System.out.print(" * ");
   for (int j = n; j >= 0; j--)  {
       if (j > 2) continue;
       System.out.println(" * ");
   }
}
```

A. 8
B. 12
C. 16
D. 20
E. 25
F. 30

ANSWER: C

3. Select the true statement(s) from the following

Within a class:

A. A private method may access a private instance variable
B. A private method may access a public instance variable
C. A public method may call a private method
D. A private method may call another private method
E. A public method may call another public method

ANSWER: A,B,C,D,E

4. What will be the output of the following program:

```
public class PassingIntArray{
  public static void addOne(int n[], int m) {
    n[2]++;
    n[1]++;
    m++;
  }
  public static void main (String[] args){
    int n[] = {10,20,30};
    int m = 40;
    addOne(n, m);
    System.out.println("values are " + n[0] + ","+ n[1]+ "," + n[2] + “,” + m);
  }
}
```

A. values are 10,20,30,40
B. values are 11,21,31,40
C. values are 11,21,31,41
D. values are 10,21,31,40

ANSWER: D

5. If class B extends class A and class C extends class B then which of the following statement is **false**?

A. an instance of class A may use all public methods of class A
B. an instance of class A may use all public methods of class B
C. an instance of class C may use all public methods of class A
D. an instance of class B may use all public methods of class C

ANSWER: B

6. Which of the following statement(s) will cause compilation errors?

```
int a;
long b;
double c;
```

A. a = b;
B. b = a;
C. c = a;

D. a = (int) c;

ANSWER: A

7. Assume that *Apple* is a subclass of *Fruit.* Which of the following assignment(s) will cause compilation errors?

```
Apple apple1 = new Apple(2.5);
Fruit fruit1 = new fruit(3.2);
Apple apple2;
Fruit fruit2;
```

A. apple2 = fruit1;
B. fruit2 = apple1;
C. apple2 = Apple fruit1;
D. apple2 = (Apple) fruit1;
E. fruit2 = Fruit apple1;
F. fruit1 = (Fruit) apple1;

ANSWER: A,C,E

8. Which of the statement(s) below attempting to override *method1()* in subclass B is/are valid?

```
class A {
  void method1(int amount) {
  }
}

class B extends A {
  // method1 to be overridden here
}
```

A. private void method1(int amount) { }
B. void method1(int amount) { }
C. protected void method1(int amount) { }
D. public void method1(int amount) { }

ANSWER: B,C,D

9. Which of the following statement in the ***add*** method is/are **invalid**?

```
public class Test  {
   public static void main(String[] args) {
```

```
        A a = new A();
        a.addOne(5);
    }
}
class A {
    public static void addOne(int x)     {
        int y=0;
        final int z = 0;
        x++;
        y++;
        z++;
        i++;
        j++;

    }
     private static int i;
     private int j;
}
```

A. x++;
B. y++;
C. z++
D. i++;
E. j++;

ANSWER: C,E

10. Which of the following method is not a static method?

A. readLine() of BufferedReader class
B. sqrt() method of Math class
C. parseInt() method of Integer class
D. parseDouble() method of Double class

ANSWER: A

Part B: Short coding problems (40 marks)

(a) (a) Simple coding (8 marks)
Complete a class called *ControlDemo* to allow a user to enter three integers, one per line. The average of these three numbers will be printed (see the sample output below)

- No exception handling is required
- No input validation is required
- ***ConsoleReader*** class is used (see the appendix)

A sample run of the program is:

```
>java ControlDemo
Enter three integers, one per line:
1
2
3
The average of the numbers is: “2.0”
```

```
//Sample Program
import java.text.*;

public class ControlDemo {
  public static void main (String [] args) {
    ConsoleReader console= new ConsoleReader (System.in);
    System.out.println("Enter three integers, one each line:");

    double sum = 0;

    for (int i =0; i <3 ; i ++)
       { sum += (double) console.readInt();
       }

    DecimalFormat df = new DecimalFormat("0.00");

    System.out.println("The average of the numbers is \"" +
                            df.format(sum/3) + "\"");
  }
}
```

(b) Simple coding II (12 marks)

Complete a class called *ControlDemo2* to allow a user to enter some words, one word per line. If an empty line is entered, then the program will finish and print all the input in a reversed order and in one line (see the sample output below).

- No exception handling is required
- No input validation is required
- You can use ***ConsoleReader*** Class (see the appendix)
- You can use string concatenation, eg stringA = stringB + stringC;

A sample run of the program is:

```
>java ControlDemo2
Enter some words, one word per line:
Hello
World
!

You entered: ! World Hello
```

```
public class ControlDemo2 {
  public static void main (String [] args) {
    ConsoleReader console= new ConsoleReader (System.in);
    System.out.println("Enter some words, one word per line:");

    String input = "";

    String newInput;

    do {
       newInput = console.readLine();

       input = newInput + " " + input;

    } while(newInput.length() != 0);


    System.out.println("You entered: " + input);

  }
}
```

(c) Class and methods coding (20 marks)

(i) 15 marks
Define a class **ClubMember**. A club member has a name (String) and an age (int). The annual fee of each employee is equal to *(basicFee + age* rate)*, where *basicFee* and *rate* are always $200 and $10. The class has one constructor which takes *name* and *age* as its parameters. It also have a method to return name, *getName()*, a method to return age, *getAge()*, and a method, *getFee()* to return his/her annual Fee.

(ii) 5 marks
Complete the class classed **TestClubMember** to produce the output as shown below:

- No input validation is required
- No exception handling is required
- You can use ***ConsoleReader*** class (see the appendix)
- No formatting of output is required.
- The output in bold are user input data.

```
>java TestClubMember
Enter the member's name: Geoff
Enter the member's age: 20
Geoff's annual fee is: $400
```

```
public class ClubMember{

   private static int basicFee = 200;
   private static int rate = 10;

   private String name;
   private int age;

   public ClubMember(String memName, int memAge)
   { name = memName;
     age = memAge;
   }

   public String getName()
   {return name;}

   public int age()
```

```
   {return age;}

   public int getFee()
   {return (basicFee + age* rate); }
}

public class TestClubMember
{  public static void main(String[] args)
   {  ConsoleReader console= new ConsoleReader (System.in);

      System.out.print("Enter the member's name: ");
      String name = console.readLine();

      System.out.print("Enter the member's age: ");
      int age = console.readInt();


      ClubMember one = new ClubMember(name,age);

      System.out.println(one.getName() + "'s annual fee is: $" + one.getFee());

   }
}
```

Part C: Inheritance and Polymorphism (40 marks)

You are to implement a simple student administration system using inheritance in Java. The super class is called ***Student*** and the sub class is called ***SeStudent*** class (senior student). A class called ***TestStudent*** is used to test the above classes.

(a) 12 marks
The ***Student*** class has the following properties:

- Accessor methods
  - *getID()*
  - *getName()*
  - *getCredit()*
- Mutator methods
  - *addCredit(parameters)* to add credit points after passing some courses.
  - *reduceCredit(parameters)* to deduct some credit points as some kinds of penalty. A student can't have a negative credit amount.
  - *compareCredit(parameters)* to compare the credits of two students.

- A constructor to identify a student: studentID, name and credit.

(b) 18 marks

The ***SeStudent*** class has the following properties:

- Accessor method
  - *getMinCredit()* to get the minimum credit points required for a senior student.
  - *getMaxCredit()* to get the maximum credit points that a senior student can have.
- Mutator methods
  - *addCredit(parameters)* to add credit points after passing some courses. A senior student cannot have more credit points can the maximum amount.
  - *reduceCredit(parameters)* to deduct some credit points as some kinds of penalty. A senior student can't have credit points less than the minimum amount.
  - *haveExemption(parameters)* to add extra credit points to a student. However a student can't have more credit points than the maximum amount.

- Two constructors; one to inherits from the Student class and include the minimum and maximum number of credit points; the other constructor is the default with the minimum set to 0 and the maximum set 200.

(c) 10 marks

The ***TestStudent*** class will make use of the above classes to produce the following output for two students:

- Sally is not a senior student. Her ID is 31203 and she has 12 credit points
- Harry is a senior student. His ID is 11656, he has 36 credit points while the minimum and maximum credit points for him are 30 and 112.
- Exception handling is not required

```
>java TestStudent
See if we can add 12 points for sally:
Yes, Sally now has 24 credit points.
See if we can deduct 8 points from Harry:
No, Harry must maintain 30 credit points.
Compare Sally and Harry:
Sally has less credit points than Harry: -12
```

```
class Student
{   private int studentID;
    private String name;
    private int credit;

    public Student(int sID, String sName, int sCredit)
    { studentID = sID;
      name = sName;
```

```
    credit = sCredit;
  }

  public int getID(){return studentID;}
  public String getName() {return name;}
  public int getCredit() {return credit;}

  public boolean addCredit(int addAmount)
  {  credit += addAmount;
     return true;
  }

  public boolean reduceCredit(int reduceAmount)
  { if (reduceAmount > credit) return false;

    credit = credit - reduceAmount;
    return true;
  }

  public int compareCredit(Student anotherStudent)
  { return (this.getCredit() - anotherStudent.getCredit());
  }

}

============================================================
class SeStudent extends Student
{   private int minCredit;
  private int maxCredit;

  public int getMinCredit(){return minCredit; }
  public int getMaxCredit(){return maxCredit; }

  public boolean addCredit(int addAmount)
  {  if ((getCredit() + addAmount) > maxCredit)
        return false;

     super.addCredit(addAmount);
     return true;
  }

  public boolean reduceCredit(int reduceAmount)
  { if ((getCredit() - reduceAmount) < minCredit)
        {
          return false;
        }
```

```
    super.reduceCredit(reduceAmount);
    return true;
  }


  public boolean haveExemption(int addAmount)
  {  if ((getCredit() + addAmount) > maxCredit)
        return false;

    super.addCredit(addAmount);
    return true;
  }


  public SeStudent(int sID, String sName, int sCredit, int sMinCredit, int
sMaxCredit)
  { super(sID, sName, sCredit);
  minCredit = sMinCredit;
  maxCredit = sMaxCredit;
  }

  public SeStudent(int sID, String sName, int sCredit)
  { this(sID, sName, sCredit,0,200);
  }
}
```

```
public class TestStudent
{  public static void main(String[] args)
  {  Student sally  = new Student (123, "Sally", 12);
    SeStudent harry  = new SeStudent (456, "Harry", 36, 30, 112);

    System.out.println("See if we can add 12 points for sally:");

    if (sally.addCredit(12))
        System.out.println("Yes, " + sally.getName() + " now has "
                + sally.getCredit() + " credit points");
    else System.out.println("No, something wrong");

    System.out.println("See if we can deduct 8 points from Harry:");

    if (harry.reduceCredit(8))
        System.out.println("Yes, " + harry.getName() + " now has "
                                    + harry.getCredit() + " credit points");
    else  System.out.println("No, " + harry.getName()
                + " must maintain " + harry.getMinCredit() + " credit points.");
```

```
        System.out.println("Compare Sally and Harry: ");

        int diff = sally.compareCredit(harry);

        if (diff <0)
            System.out.println(sally.getName() + " has less credit pionts than "
                                + harry.getName() + ": " + diff);
        else if (diff>0)
            System.out.println(sally.getName() + " has more credit pionts than "
                                + harry.getName() + ": " + diff);
        else
            System.out.println(sally.getName() + " and " + harry.getName() +
                                " have same credit pionts");


    }
}
```