Ahmadu Bello University
Department of Mathematics
First Semester Examinations – June 2014
COSC211: Introduction to Object Oriented Programming I

Attempt **Four** questions Time: 120 mins

1. Examine the following code for the class `Horse` and answer the questions that follow.

```
1. //Horse.java
2.
3. public class Horse{
4.    private static final double MAX_WT = 500.0;
5.    private static final double MIN_WT = 25.0;
6.    private static final double MAX_HT = 2.0;
7.    private static final double MIN_HT = 0.5;
8.    private static final String DEF_NAME = "Horse";
9.    private static final double DEF_WT = 25.0;
10.   private static final double DEF_HT = 0.5;
11.
12.   private String name;
13.   private double weight;  //kilograms
14.   private double height;  //metres
15.
16.   //getters
17.   public String getName(){
18.      return name;
19.   }
20.
21.   public double getWeight(){
22.      return weight;
23.   }
24.
25.   public double getHeight(){
26.      return height;
27.   }
28.
```

```
29.    //setters
30.    public void setName(String name){
31.      this.name = name;
32.    }
33.
34.    public void setWeight(double weight){
35.      if (weight > MAX_WT) weight = MAX_WT;
36.      if (weight < MIN_WT) weight = MIN_WT;
37.      this.weight = weight;
38.    }
39.
40.    public void setHeight(double height){
41.      if (height > MAX_HT) height = MAX_HT;
42.      if (height < MIN_HT) height = MIN_HT;
43.      this.height = height;
44.    }
45.
46.    //constructors
47.    public Horse(String name, double weight, double
           height){
48.      setName(name);
49.      setWeight(weight);
50.      setHeight(height);
51.    }
52.
53.    public Horse(String name){
54.      this(name, DEF_WT, DEF_HT);
55.    }
56.
57.    public Horse(){
58.      this(DEF_NAME);
59.    }
60.
61.    public String toString(){
62.      return String.format("Name: %s\nWeight: " +
                              "%8.2f\nHeight: %8.2f",
                              name, weight, height);
63.    }
64. }//end of class horse
```

First Semester Examinations – June 2014

(*a*) On lines 4 to 10 fields are declared to be both `static` and `final`. Explain the meanings of these two keywords in this context.

A field declared to be `static` is a class field.
It is shared by all the instances of the class and is stored once only..
It is accessible even if no instance of the class has been created.
It is accessed using the class name rather than the instance name: eg `Horse.MAX_WT` (2 marks)

A field declared to be `final` is a constant, so that its value, once assigned, cannot be changed. (1 mark)

(*b*) On lines 17 to 27 three *getters* are defined. What is their purpose? Why are they required?

A `public` getter allows the value of a field to be retrieved by a call made from outside the class containing the field. (2 marks)

This is necessary if the field is `private`. (1 mark)

(*c*) Two of the *setter* methods perform *data validation*. What is the purpose of performing data validation? Explain how the data validation in one of these setter methods works.

Data validation is performed to ensure that the value of a field is reasonable. (1 mark)
The validation for `weight` ensures that it lies between 25 kg and 500 kg (lines 35-36). (2 marks)

(*d*) Explain the usage of the `this` keyword on line 31.

It serves to distinguish between the *field* `weight` and the *parameter* `weight`. The first (preceded by `this`) is the field; the second (on its own) is the parameter. (3 marks)

(*e*) Explain what is meant by *constructor overloading*. Explain how this is carried out in the `Horse` class.

A constructor is overloaded when more than one is defined (with the same name) but each has a different list of parameter types. The different constructors are

distinguished by their parameter lists. (2 marks)

The `Horse` class has three constructors: `Horse(String, double, double)`, `Horse(String)` and `Horse()`. (2 marks)

(*f*) There is a `toString()` method defined on lines 61 to 63. Explain what it does and how it is used.

The `toString()` method returns a string representation of an object. (1 mark)

Whenever a string representation of an object is required the `toString()` method may be automattically inserted by the compiler, as in this example:

```
System.out.println(horse);  //becomes horse.toString()
```

where `horse` is an instance of `Horse`. (3 marks)

First Semester Examinations – June 2014

2. (*a*) Create a class named `Grade` that includes the following.

(*i*) Declarations for two *fields* named `name` and `score` of type `string` and `double` respectively.
(*ii*) Definitions of two *constructors* where the first one should has no parameter while the second contains parameters that initialize the fields declared in (*i*) above.
(*iii*) *Getters* and *setters* of the fields declared in (*i*) above.
(*iv*) A method called `isScoreValid()` that will returns `true` if `score` is neither greater than 100 nor less than 0, `false` otherwise.
(*v*) A method called `computeGrade()` that will returns the `grade` (`char`) as follows.

```
grade = 'A' if score>=70
grade = 'B' if 60 <= score < 70
grade = 'C' if 50 <= score < 60
grade = 'D' if 45 <= score < 50
grade = 'E' if 40 <= score < 45
grade = 'F' if score < 40
```

```
public class Grade{
   private String name;
   private double score;                          (2 marks)

   public Grade(){                                 (1 mark)
      this("Garba", 0);
   }// end of no-args constructor

   public Grade(String name, double score){   (1 mark)
      this.score = score;
      this.name = name;
   }// end of constructor

   public String getName(){                        (1 mark)
      return name;
   }// end of getName()

   public double getScore(){                       (1 mark)
      return score;
   }//end of getScore()

   public void setScore(double score){          (1 mark)
```

```
    this.score = score;
  }// end of setScore()

  public void setName(String name){          (1 mark)
    this.name = name;
  }// end of setName()

  public boolean isScoreValid(){             (2 marks)
    if (score < 0 || score > 100)
      return false;
    else
      return true;
  }// end of isScoreValid()

  public char computeGrade(){                (4 marks)
    char grade;

    if (score >= 70)
      grade = 'A';
    else if (score >= 60)
      grade = 'B';
    else if (score >= 50)
      grade = 'C';
    else if (score >= 45)
      grade = 'D';
    else if (score >= 40)
      grade = 'E';
    else
      grade = 'F';

    return grade;
  }//end of computeGrade()
}// end of Grade class
```

(*b*) Create a test class called `GradeTest` that will enable the user to enter a student's name and score and see the students grade. The output should look like

```
Fati's score is 62 and grade is B
```

or (if the score is more than 100 or less than zero)

```
The score is invalid
```

```
import java.util.Scanner;                          (½ mark)

public class GradeTest{
  public static void main(String[] args){
    Scanner input = new Scanner(System.in);
                                                   (½ mark)
    System.out.print("Enter the student's " +
      "name: ");
    String name = input.nextLine();                (1 mark)

    System.out.print("Enter the student's " +
      "score: ");
    double score = input.nextDouble();             (1 mark)

    Grade grade = new Grade(name, score);          (1 mark)
    if(grade.isScoreValid())                       (1 mark)
      System.out.printf("%s\'s score is "
           "%.2f and grade is %c",
           grade.getName(), grade.getScore,
           grade.computeGrade());
    else
      System.out.println("The score is invalid");
                                                   (1 mark)
  }
}//end of GradeTest class
```

---

3. Examine the following code and answer the questions that follow.

```
 1. //ProcessMarks
 2.
 3. import java.util.Scanner;
 4.
 5. public class ProcessMarks{
 6.    public static void main(String[] args){
 7.       Scanner input = new Scanner(System.in);
 8.       int sum = 0, num = 0;
 9.       System.out.print("Enter a student mark " +
                 "(-1 to exit): ");
10.       int mark = input.nextInt();
11.
12.       while(mark >= 0){
13.          if (mark >= 40){
14.             sum = sum + mark;
15.             num++;
16.          }
17.          System.out.print("Enter a " +
                       "student mark (-1 to exit): ");
18.          mark = input.nextInt();
19.       }
20.
21.       if (num > 0){
22.          double average = (double)sum / num;
23.          System.out.printf("The required " +
                    "average is %.1f\n", average);
24.          }
25.       else
26.          System.out.println("The are no " +
                    "marks to average");
27.    }//end of main()
28. }//end of class ProcessMarks
```

(*a*) What is the purpose of the `import` statement on line 3?

The `import` statement makes the `Scanner` class in the package `java.util` from the Java API available to the program. (3 marks)

(*b*) What is the most important difference between a *pre-condition* loop and a *post-condition* loop? What type of loop starts on line 12?

With a post-conditon loop the loop body will be traversed at least once; while with a pre-condition loop it is possible that the loop body may never be entered.

It is a pre-condition loop. (2 + 1 marks)

(*c*) What is the `sum` that is evaluated on line 14?

It is the `sum` of all those marks that are 40 or more (3 marks)

(*d*) Explain the purpose of `(double)` on line 22 where `average` is evaluated.

The `sum` is *cast* (converted) to a `double` before the division is performed. If this is not done then integer division will be performed and the resulting value of `average` will not be correct. (3 marks)

(*e*) When running this program a user enters 45, 56, 34, 68 and then –1 in response to the prompts. Show the resulting output.

```
(average = (45 + 56 + 68) / 3 = 56.333...)
The required average is 56.3
```
(4 marks)

(*f*) When running this program a user enters 20, 34, 39, 15 and then –1 in response to the prompts. Show the resulting output.

```
There are no marks to average
```
(4 marks)

COSC211: Introduction to Object Oriented Programming I

4. *Ahmadu Bello University Press* wants to know the maximum, minimum, total, and average profit gained from years 2005 to 2014. Besides that, they are interested in knowing the difference between the maximum and minimum profit and those profits that are above average. The profits are as follows.

Year	**Profit(₦)**
2005	5,000,000.34
2006	2,005.000.00
2007	3,020,000.97
2008	5,057,800.20
2009	4,500,000.67
2010	5,000,000.00
2011	3,048,900.56
2012	4,800,000.50
2013	2,980,000.71
2014	4,909,000.80

(*a*) Create a class called `Profit` that has the following members.

(*i*) Two array fields to store the profit and year named `profit` and `year` using appropriate data types.
(*ii*) A method called `getMaxProfit()` that will return the maximum profit.
(*iii*) A method called `getMinProfit()` that will return the minimum profit.
(*iv*) A method called `getSumOfProfit()` that will return the total profit.
(*v*) A method called `getAverageProfit()` that will return the average profit.
(*vi*) A method called `getRange()` that will return the difference between the maximum and minimum profits.
(*vii*) A method called `showProfitBelowAverage()` that will display all profits that are below average.

```
public class Profit{                                    (1 mark)
   int[] year = {2005, 2006, 2007,                      (1 mark)
                 2008, 2009, 2010,
                 2011, 2012, 2013,
                 2014};
   double[] profit = {5000000.34, 2005000.00,           (1 mark)
                      3020000.97, 5057800.20,
                      4500000.67, 5000000.00,
                      3048900.56, 4800000.50,
                      2980000.71, 4909000.80};
```

```
public double getMaxProfit(){            (2 marks)
   double maxProfit = profit[0];
   for(int i = 1; i < profit.length; i++){
     if (profit[i] > maxProfit)
       maxProfit = profit[i];
   }// end of for loop
   return maxProfit;
}//end of method getMaxProfit()

public double getMinProfit(){            (2 marks)
   double minProfit = profit[0];
   for(int i = 1; i < profit.length; i++){
     if (profit[i] < minProfit)
       minProfit = profit[i];
   }// end of for loop
   return minProfit;
}// end of method getMinProfit()

public double getSumOfProfit(){          (2 marks)
   double sum = 0;
   for(int i = 0; i < profit.length; i++){
     sum = sum + profit[i];
   }//end of for loop
   return sum;
}//end of method getSumOfProfit()

public double getAverageProfit(){        (3 marks)
   return getSumOfProfit() / profit.length;
}//end of method getAverageProfit

public double getRange(){                (2 marks)
   return getMaxProfit() - getMinProfit();
}//end of method getSumOfProfit()

public void showProfitBelowAverage(){    (3 marks)
   for(int i = 0; i < profit.length; i++){
     if (profit[i] < getAverageProfit()){
       System.out.printf("%.2f%s",
            profit[i], ", ");
```

```
        }// end of if
      }// end of for loop
    }//end of method showProfitBelowAverage()
}//end of class Profit
```

(*b*) Create a test class that will instantiate the `Profit` class and display the required results with suitable headings and labels.

```
public class ProfitTest{
  public static void main(String args[]){
    Profit profit = new Profit();          (1 mark)
    System.out.println("Ahmadu Bello " +
         "University Press");
    System.out.printf("The total profits " +
         "gained is %.2f \n",
         profit.getSumOfProfit());         (2 marks)
    System.out.print("The profits that " +
         "are below average are: ");
    profit.showProfitBelowAverage();
  }//end of method main
}// end of class ProfitTest
```

5. Examine the following code and answer the questions that follow.

```
1. //Order.java
2. /*An order is placed for a number of items, if
3.  *the number is big enough a discount is allowed
4.
5. public class Order{
6.    private static final double UNIT_PRICE = 1000;
7.    private static final double DISC_RATE = 0.05;
8.    private static final int MIN_DISC_QUANT = 100;
9.
10.   private int quantity;
11.
12.   public Order(int quantity){
13.     setQuantity(quantity)
14.   }//end of constructor
15.
16.   public getQuantity(){
17.     return quantity;
18.   }//end of method getQuantity()
19.
20.   public void setQuantity(quantity){
21.     if (quantity < 0);
22.       quantity = 0;
23.     this.quantity = quantity;
24.   }//end of method setQuantity()
25.
26.   public double calculateCost(){
27.     double totalCost = UNIT_PRICE * quantity;
28.
29.     if (quantity >= MIN_DISC_QUANT){
30.       discount = totalCost * DISC_RATE;
31.       totalCost -= discount;
32.     }//end of selection structure
33.
34.   }//end of method calculateCost()
35. }//end of class Order
```

(*a*) There are eight lines containing errors that would be detected by the Java compiler. Identify them and rewrite the lines with the errors corrected.

```
Line 4:  */
Line 6: private static final double UNIT_PRICE = 1000.0;
Line 13: setQuantity(quantity);
Line 16: public int getQuantity();
Line 20: public void setQuantity(int quantity){
Line 21: if(quantity < 0)
Line 30: double discount = totalCost * DISC_RATE;
Line 33: return totalCost;
```

(1½ each = 12 marks)

(*b*) Define a `toString()` method for the `Order` class that can be used to display the order details.

```
Public String toString(){
   return String.format("Quantity: %4d\tCost: %,.2f",
        quantity, calculateCost());
}                                                    (2 marks)
```

(*c*) Write an `OrderTest` class that will instantiate three objects from this class and display the order details.

```
public class OrderTest(
   public static void main(String[] args){
      Order order1 = new Order(20);
      Order order2 = new Order(100);
      Order order3 = new Order(120);

      System.out.println(order1);
      System.out.println(order2);
      System.out.println(order3);
   }//end of main()
}//end of class OrderTest                    (3 + 3 marks)
```

First Semester Examinations – June 2014

6. (*a*) Write an application that will prompt user the to enter the first term, common difference, and the number of terms of an arithmetic progression (AP). It should compute the *n*th term of the series and the sum of the first *n* terms. Your code should ensure that the number of terms, *n*, is positive.

[Hint: use $T_n = a + (n-1)d$, and $S_n = \frac{1}{2}n(a + T_n)$, where a is the first term, n is the number of terms, d is the common difference, T_n is the *n*th term of the series, and S_n is the sum of the first n terms.]

```
import java.util.Scanner;                          (½ mark)
public class Series{
   public static void main(String args[]){
      int n;
      double d;
      double a;                                    (2 marks)
      double term;
      double sum;

      Scanner input = new Scanner(System.in);  (½ mark)

      do{                                        (2 marks)
         System.out.print("Please enter the " +
              "number of terms: ");
         n = input.nextInt();
      }while (n < 0);

      System.out.print("Enter the first term: ");
      a = input.nextDouble();                      (1 mark)

      System.out.print("Enter the common " +
           difference: ");
      d = input.nextDouble();                      (1 mark)

      term = a + (n - 1) * d;                      (1 mark)
      sum = (n * (a + term)) / 2;                  (1 mark)

      System.out.printf("The sum of the " +
           series is : %.2f ", sum);               (1 mark)
   }//end of main method
}//end of class Series
```

(*b*) A quadratic equation of the form $ax^2 + bx + c = 0$ has roots $\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.

Write a fragment of code that efficiently determines the values of the roots (`root1` and `root2`) of the given equation. Assume that all the variables have been declared as type `double`. Note that the roots are real and distinct, real and equal, or complex according to whether the *discriminant* ($b^2 - 4ac$) is positive, zero or negative, respectively.

Format the output in the following ways (where `root1` and `root2` are the calculated roots of the equation).

(*i*) The roots are real and distinct : `root1`, `root2`.
(*ii*) The roots are real and equal: `root1`.
(*iii*) The roots are complex.

Note: the roots should not be displayed if they are complex.

```
import java.util.Scanner;                          (½ mark)

public class Quadratic{
  public static void main(String[] args){
    double a;
    double b;                                  (2 marks)
    double c;
    double root1,root2;

    Scanner input = new Scanner(System.in);  (½ mark)

    System.out.print("Enter the value of a: ");
    a = input.nextDouble();                    (1 mark)

    System.out.print("Enter the value of b: ");
    b = input.nextDouble();                    (1 mark)

    System.out.print("Enter the value of c: ");
    c = input.nextDouble();                    (1 mark)

    double disc = b * b - 4 * a * c;           (1 mark)
```

```
        if (disc > 0){                          (1 mark)
          root1 = (-b + Math.sqrt(disc)) / (2 * a);
          root2 = (-b - Math.sqrt(disc)) / (2 * a);
          System.out.printf("The roots are real " +
               "and distinct: %.2f, %.2f \n",
               root1, root2);
        }else if (disc == 0){                   (1 mark)
          root1 = -b / (2 * a);
          System.out.printf("The roots are real " +
               "and equal: %.2f\n", root1);
        }else                                   (1 mark)
          System.out.println("The roots are " +
               "complex");
    }//end of main method
}//end of class Quadratic
```