Ahmadu Bello University
Department of Mathematics
First Semester Examinations – Month 2013
COSC211: Introduction to Object Oriented Programming I

Attempt **ALL** the questions   Time: **120** mins

1. Examine the following Java code and answer the questions that follow.
[Note that line numbers are not part of the code and are for reference only.]

```
 1.   //Tree.java
 2.
 3.   public class Tree{
 4.      private double girth;
 5.      private double height;
 6.
 7.      public Tree(double girth, double height){
 8.         setGirth(girth);
 9.         setHeight(height);
10.      }
11.
12.      public Tree(){
13.         this(0.01, 0.05);
14.      }
15.
16.      public double getGirth(){
17.         return girth;
18.      }
19.
20.      public double getHeight(){
21.         return height;
22.      }
23.
24.      public void setGirth(double girth){
25.         if(girth < 0.01) girth = 0.01;
26.         this.girth = girth;
27.      }
28.
29.      public double setHeight(double height){
30.         if(height < 0.05) height = 0.05;
31.          this.height = height;
32.       }
33.
34.       public String toString(){
35.          return String.format("Girth %8.2f\t" +
                "Height %8.2f", girth, height);
36.       }
37.    }//end of class Tree
```

(*a*) The class has two fields, both *private*. Explain why such fields are normally kept private.

Ensures that the fields cannot be accessed directly except from within the class. (1)
The fields can be used in a controlld way using public methods that are members of the class. (1)

(*b*) There are two *constructors*, one on lines 7 to 10 and one on lines 12 to 14. Describe two characteristics that these constructor methods have.

They have the same name as the class (including capitalization. (1)
They have no return value (not even void). (1)

(*c*) When there is more than one constructor defined in a class how does the JVM distinguish between them? Explain the term *signature*.

Although they have the same name they must have a different signature – that is their lists of parameter types must be different. (1)
The signature of a method is the name combined with the ordered list of their parameter types. (1)

(*d*) There are two *getter methods*. Identify them and explain what they are for.

The getter methods are `getGirth()` (lines 16-18) and `getHeight()` (lines 20-22). (1)
They return the values of the fields `girth` and `height`, respectively. (1)

(*e*) There are two cases of *data validation* performed by this code. What lines are they on and how do they work?

Data validation occurs on lines 25 and 30. (2)
Line 25 ensures that the value of `girth` cannot be less than 0.1. (1)
Line 30 ensures that the value of `height` cannot be less than 0.5. (1)

(*f*) The keyword `this` is used in two different senses. First on line 13 it is used in a constructor. Second on lines 26 and 30 it is used in a mutator. Explain each of these usages fully.

On line 13 `this` is used as a call to another constructor, identified by the parameter type list. In this case the call is to the first constructor. (2)
On lines 26 and 31 `this` is used to distinguish between the field (`this.girth` and `this.height`) and the parameters (`girth` and `height`). (2)

(*g*) There is a `toString()` method. What is it used for? What characteristics must a `toString()` method have?

The toString() method is used to return a string representation of a Tree instance. (2)
The `toString()` method must (i) return a `String`, (ii) have no parameter, (iii) be named '`toString`', and (iv) be `public`. (2)

(*h*) Write the Java source code for a program that will instantiate two `Tree` objects and display their field values.

```
//TreeTest.java
public class TreeTest{
    public static void main(String[] args)
        Tree tree = new Tree(0.5, 7.5);
        Tree seedling = new Tree();

        System.out.println(tree);
        System.out.println(seedling);
    }//end of method main()
}//end of class TreeTest
```

(5)

2. The following code, stored in the files *Worker.java* and *WorkerPay.java*, has ***eight*** errors in it.
[Note that line numbers are not part of the code and are for reference only.]

```
 1.  //Worker.java
 2.
 3.  public Worker{
 4.     private static final double HOURLY_PAY =
            2000.00;
 5.     private static final int BASE_HOURS = 40;
 6.     private hours;
 7.
 8.     public Worker(int hours){
 9.        setHours(hours);
10.     }
11.
12.     public void setHours(int hours){
13.        if(hours > 70) hours = 70
14.        this.hours = hours;
15.     }
16.
17.     public double calculatePay(){
18.        double pay;
19.        if hours > BASE_HOURS {
20.           pay = BASE_HOURS * HOURLY_PAY;
21.           pay += (hours - BASE_HOURS) *
                 HOURLY_PAY * 1.5;
22.        }else{
23.           pay = hours * HOURLY_PAY;
24.        return pay;
25.
26.  }//end of class Worker
```

```
 1.  //WorkerPay.java
 2.
 3.  public class WorkerPay{
 4.      public void main(String[] args){
 5.          Worker worker = new Worker(56);
 6.          System.out.printf("Pay is %.2f\n",
                 worker.calculatePay());
```

```
7.      }
8.  }//end of class WorkerPay
```

(*a*) Locate each of the ***eight*** lines that contain an error and rewrite it correctly.

Line 3: `public class worker{   (2)`
Line 6: `private int hours;   (2)`
Line 15: `if (hours > 70) hours = 70;   (2)`
Line 19: `if (hours > BASE_HOURS){   (2)`
Line 21: `} else   (2)`
Line 25: `}   (2)`
Line 4: `public static void main(String[] args){`
Line 6: `...`

```
          worker.calculatePay());   (2)
```

(*b*) Describe the purpose of the method `calculatePay()` on lines 17 to 25 and explain how it works. (In answering this refer to the code you have corrected since the original may have errors.)

The method calculates and returns an hourly-rated workers pay. (1)
If the worker does more than BASE_HOURS then his overtime is calculted (at time and a half) and added to his basic pay (4)
otherwise his basic pay is calculated. (2)

(*c*) Reproduce the screen display created when running the program *WorkerPay.class*, (compiled after the corrections have been applied).

```
Pay is 128000.00
```
(2)

5

3. (*a*) Explain ***in full*** the effect of the following lines of code if they appeared in a `main()` method..

(*i*) Dog myDog = new Dog();

`myDog` is declared to be an instance of the class `Dog`.
The constructor creates and initialises the instance fields on the heap.
`myDog` is a reference to the data on the heap. (3)

(*ii*) int myNum = 27;

`myNum` is declared to be an `int` and 4 bytes are reserved for it in memory.
The value 27 is stored at that location. (2)

(*iii*) String myName = "Fatima Abubakar";

`myName` is declared to be an instance of class `String`.
The constructor creates and initializes the instance fields on the heap with the length of the string "Fatima Abubakar" and its characters.
`myName` is a reference to the data on the heap. (2)

(*iv*)
```
if(height >= 2.0)
     System.out.println("You are tall.\n");
else
     System.out.println("You aren't tall.\n");
```

If the `boolean height >= 2.0` evaluates to `true` "You are tall" is displayed, otherwise "Ypu aren't tall" is displayed. (3)

(b) There are two principal data types in the Java Programming Language: *primitive data types* and *objects*. Distinguish carefully between them.

Primitive data types are built-in data types that have a fixed storage requirement. (2)
Their values will be stored on the stack if they are declared in a method; their values will be stored on the heap if they are declared as fields. (2)
Objects are created by a class constructor. Their method code is loaded into memory with the class file and their instance variables are stored on the heap. Their names reference these field values. (3)

(c) How many bits of storage do the following data types require: `int`, `byte`, `short`, `long`, `float`, `double`.

int	32 bits	4 bytes
byte	8 bits	1 byte
short	16 bits	2 bytes
long	64 bits	8 bytes
float	32 bits	4 bytes
double	64 bits	8 bytes

6)

4. (a) Write the code for a class called `Course` that has the following private fields: `code` (`String`); `title` (`String`), `level` (`int`) and `semester` (`int`).

There should be a constructor that sets the field values, passed as arguments. There should also be a *no-args* constructor that gives each field a suitable default value.

There should be *accessor* and *mutator* methods for each field. The mutator methods for `level` and `semester` should include suitable *validation checks*.

There should be an appropriate `toString()` method.

```
public class course{
     private String code;
     private String title;
     private int level;
     private int semester;    (2)

     //constructor
     public Course(String code, String title,
        int level, int semester){
         setCode(code);
         setTitle(title);
         setLevel(level);
         setSemester(semester);
     }//end of constructor    (2)

     //no-args constructor
     public Course(){
         this("code", "title", 100, 1);
     }//end of no-args constructor    (2)

     //accessor method for code
     public String getCode(){
         return code;
     }//end of getCode()    (1)

     //accessor method for title
     public String getTitle(){
```

```
    return title;
}//end of getTitle()    (1)

//accessor method for level
public int getLevel(){
    return level;
}//end of getLevel()    (1)

//accessor method for semester
public int getSemester(){
    return semester;
}//end of getSemester()    (1)

//mutator method for code
public void setCode(String code){
    this.code = code;
}//end of setCode()    (1)

//mutator method for title
public void setTitle(String title){
    this.title = title;
}//end of setTitle()    (1)

//mutator method for level
public void setLevel(int level){
    level = (level / 100) * 100;
    if (level < 100) level = 100;
    if (level > 400) level = 400;
    this.level = level;
}//end of setLevel()    (2)

//mutator method for semester
public void setSemester(semester){
    if (semester < 1) semester = 1;
    if (semester > 2) semester = 2;
    this.semester = semester;
}//end of setSemester()    (1)

public String toString(){
    return String.format("%s:%s " +
        "Level: %d Semester %d\n",
        code, title, level, semester);
}//end of toString() method    (2)
}//end of class Course    (2)
```

(b) Now write the code for a program that will instantiate the class you created in (a) above. The field values should be "COSC211", "An Introduction to object-oriented programming I", 200 and 1, repectively. The program should display these field values.

```
    public class CourseTest(
        public static void main(String[] args){
            Course course = new Course("COSC211",
                "An Introduction to Object-Oriented " *
                "Programming", 200, 1);
            System.out,println(course);
        }//end of method main()
    }//end of class CourseTest    5
```

5. (*a*) Some methods return a value, others do not. Describe the differences between them in (*i*) how the methods are defined, and (*ii*) how the methods are called.

(i) The method header is marked `void` if there is no return method. (2)
Thus

```
public void myMethod(){
    statements;
}     (2)
The method header is marked with the return type if
there is a return value. (2)
Thus
public int myMethod(){
    int myInt;
    statements;
    return myInt;
}     (2)
```

(ii) A method with no return value is called by writing it on its own as a statement (with a fully qualified name).

```
    myClass.myMethod();    (2)
```

A method with a return value must be incorporated into a suitable expression (possibly an assignment).

```
    int number = myClass.myMethod();    (2)
```

(*b*) Design a method that will take three integer arguments and return the smallest.

```
  public int min(int x, int y, int z){
      if (x < y)
          if (x < z)
              return x;
          else
              return z;
      else
          if (y < z)
              return y;
          else
              return z;
  }    (10)
```

(c) Why is the `main()` method declared to be static.

The `main()` method is called before an instance of its class can be created and therefore it must be `static`. (3)

6. (*a*) The following statements appear at the very start of a Java file.

```
import javax.swing.JFrame;
import static java.lang.Math.*;
```

Explain what they are for.

The program requires access to the `JFrame` class in the `javax.swing` package. (3)
It can access the class without the need for a fully-qualified name. (2)
The program can use any of the static members of the `Math` class from the `java.lang` package. (3)
It can access them without the need for a fully-qualified name. (2)

(*b*) Write a Java program that will prompt the user for the size of the angles A and B, and the length of the side BC of a triangle ABC. The program should calculate and display the length of the side AC. [Sine rule.]

```
//SineRule.java

 import static java.lang.Math.*;
 import java.util.Scanner;

 public class SineRule
     private static final double DEG2RAD =
         PI /180.0;
     private static void main(String[] args){
         Scanner input = new Scanner(System.in);
         System.out.print("Enter the side BC: ");
         double lenBC = input.nextDouble();
         System.out.print("Enter the angle A " +
             "(degrees): ");
         double angleA = input.nextDouble();
         angleA = angleA * DEG2RAD;
         System.out.print("Enter the angle B " +
             "(degrees): ");
         double angleB = input.nextDouble();
         angleB = angleB * DEG2RAD;
         double lenAC = lenBC * sin(angleB) /
             sin(angleA);
```

```
            System.out.printf("AC has length %f\n",
              lenAC);
        }//end of method main()
    }//end of class SineRule
```

Synrax (5)

Code (10)