JAVA PROGRAMMING – EXERCISE 10

In this lab, we'll be dealing with multiple classes, constructors, and data fields.

COURSE MARKER

1) Start Course Marker by selecting
Start>All Programs>EPS >EEE >Course Marker 4

After a few moments you will see the login window –
NOTE: User Name is your usual username (mch…), and the password is your student ID number:

2) Once you're in, access the relevant assignment by selecting:
EEEN10035 – Java Programming>unit10 – OOP:

3) Click the Setup button on top, and then click OK.
Clicking the Setup button copies the files needed for this exercise to the folder
P:>mchXXYY> EEEN10035u10exercises10 (where mchXXYY is replaced with your username).

The files copied are:
Circle.java, and CircleTester.java
These are the java files you'll be working on this time.

4) That's all we need from Course Marker, so now you can close it.

5) Everything we do from here on out will involve *ConTEXT*.

Open ConTEXT via,
Start>All Programs> Start>All Programs> EPS >EEE> ConTEXT
(this path may have changed –ask for help if necessary)

In the resulting window navigate to the folder,
P:>mchXXYY> EEEN10035u10exercises10

You should see the following files in the folder: Circle.java, and CircleTester.java

Before we move on ahead, we shall review data fields and constructors in classes.

DATA FIELDS

A 'data field' is just another name for an 'instance' variable (unless its declared **static** –see Lecture 15 notes). Observe the example below, where 'g' is the data field in the class called **Something** :

Anything.java

```
public class Anything
{
    public static void main (String args[])  {
        Something s = new Something();
        s.printMessage ();
        s.printSum ();
        System.out.println (s.g);
    }

}
```

Something.java

```
public class Something {
        int g=10;
        public void printMessage  () {
           System.out.println ("Hello");
        }
        public void printSum ()  {
            int sum=2+1+5;
            System.out.println (sum);
        }
}
```

Similar to how you call methods, you can call data fields by this syntax: ObjectName.VariableName;

To use the methods and data fields belonging to our second class **Something**, we need to declare it first.
NOTE: s is the variable name here, also called an 'object reference variable'.

In the example above, **main** creates an instance of **Something**, then uses the reference variable to call printMessage () and printSum (), before finally printing out the value g. Note that the methods in **Something** are non-static, meaning that you *have to* create an instance of the class in order to call them.

CONSTRUCTORS

Constructors are used to initialize the data fields. In the above example there isn't a constructor, but in the one is included in the slightly different example below :

Anything.java

```
public class Anything  {
    public static void main(String args [ ] )   {
        Something s=new Something ();
        s.PrintMessage ();
        System.out.println (s.g+" "+s.h);
    }
}
```

the output of this program will be,
Hello
10 15.5

Something.java

```
public class Something {
        int g;
        double h;
        public Something () {
          g=10;
          h=15.5;
        }
        public void PrintMessage () {
          System.out.println("Hello");
        }
}
```

This is the constructor, which initializes the data fields g and h with values.
Notice the constructor name is **exactly** the same as the class name. This is an example of an explicit constructor.

NOTES:

- The constructor should have the same *name* as the class.
- The constructor is automatically executed when you declare a new object.
- Although a constructor looks and behaves like a method there are two crucial differences: you can't call it directly, and it doesn't have a return data type (or void) in its declaration.
- It is possible to pass values through parameters to the constructor. These are usually used to initialize data fields. If data fields are not initialised they are given a default value depending on their type, e.g. 0 for type int, 0.0 for type double, etc.
- If you don't write a constructor, a default constructor is automatically used by the compiler. Default constructors do not have any statements or parameters.

TASKS

1) Open Circle.java and define a data field called radius of type double.

2) Write the Circle constructor, such that it takes a single parameter of type double, that is used to initialize the radius.

3) Complete the method called calcArea that calculates and returns the circle's area.

4) Complete the main method in CircleTester.java to create an instance of Circle with a radius of 1.0, and print out its area using the calcArea method defined above.

5) String variables can be assigned using a statement such as
String colour = "red";

Add a field to Circle called colour, of type String. Amend the constructor so that it is initialized to a colour of the caller's choice (the constructor now has two parameters).

6) Make changes to CircleTester, to test the modified constructor.

7) Add a method to Circle to return the colour.

8) Add statement(s) to CircleTester to print out both the color and radius instance variables. These variables can be accessed using **circle1.radius** and **circle1.color**.
Note: you will be asked to remove these statements in a later step.

9) As things stand, the value of **radius** is set by the Circle constructor. However, it is also possible for **radius** to be changed directly by using the reference variable and the dot operator. For example, the following statement, placed in CircleTester, sets the value of **radius** to 3.5:

```
circle1.radius = 3.5;
```

Test that this is true by adding a similar statement to CircleTester and using the statement(s) you added in step 8 above. Check that CircleTester compiles OK, and test the program.

10) Step 9 demonstrated that it is possible to directly change the value of an instance variable. Instance variables effectively determine an object's *state*: if the value of **radius** is changed, a Circle instance is changed, i.e. its state will change. As things stand, the radius instance variable can be externally accessed and set to any value desired. For example, it is possible to write

```
circle1.radius = -3.5;
```

-i.e. setting the data to a negative value.

This is nonsensical, but entirely possible, and we want to avoid such mistakes.

Similarly, the color instance variable is also directly changeable.

The common solution is to *hide* the data using the **private** visibility modifier:

```
public class Circle {

        //define fields

        private double radius;
```

```
        private String color;

        //rest of class

        //…

}
```

11) Modify Circle so that radius is defined as a private field as above, and make sure it compiles OK. Next, try compiling CircleTester again *(it should fail –why?).* Make a note of the compiler error message.

12) Remove the statement(s) you added to CircleTester in step 11.

13) But what if we still wanted to change the value of radius, but in a controlled manner, so we can check for and avoid problems like setting nonsensical values?
The answer lies in creating a new method, called a 'setter' method. For example :

```
public void setRadius(double r) {
        if (r < 1) {
                System.out.println( "Error ! illegal value for radius");
        }
        else {
                radius = r;
                System.out.println( "radius set to  " + radius);
        }
}
```

The above is an example of a 'setter' method, since its purpose is to set, or change, the value of an instance variable. Notice we are not returning anything, hence the return 'type' is **void**, and no **return** statement is required.

14) Add the setRadius method to Circle and make sure it compiles without error. Next, modify CircleTester by adding statement(s) that call setRadius. Make sure that both classes compile without error, and experiment by calling setRadius with legal, and then illegal values of rad.

15) Add a method to Circle called getRadius() that *returns* the value of **radius**, and add statement(s) to CircleTester to test it, e.g. print out its value. Such a method is termed a 'getter' method, because it 'gets' the value of a field.

16) Add a method called printColour that prints out the colour. The method does not return anything and its signature is as follows:

public void printColour ()

17) Add statement(s) to CircleTester to print out the color instance variable.

18) The above steps have demonstrated how to protect the data from direct modification by declaring it as private, and forcing other code to use the 'getter' and 'setter' methods instead –this is an important concept known as **encapsulation**. There is a rule of thumb regarding encapsulation:

declare instance variables (i.e. fields) **private**

declare 'getter' and 'setter' methods **public**

19) Applying the principle of encapsulation, write a class called Rectangle that has data fields named length and width (use type double). The class should have two methods named getPerimeter and getArea, to calculate and return the perimeter and area of the rectangle, respectively, a setter method named setData, to set the length and width to new values, and a getter method named showData to display length and width.

20) Write a class called RectangleTester to test the methods in your Rectangle class.

21) Write a class called Shape according to the following specification. The class has two data fields, both arrays, one called **circles** of type Circle, and the other called **rectangles**, of type Rectangle. The constructor has two integer parameters that are used to initialize size of the arrays. A method called addCircle should be added, to enable a circle to be added to **circles**. The method should not allow more circles to be added than the array has space for. Similarly, add a method called addRectangle to add a rectangle to **rectangles**, with the same restriction. Finally add two methods called printCircles and printRectangles, that print out the contents of the arrays, as in the following example:

printCircles ()
circle 1: radius = 1.0, area = 3.14, colour = yellow
circle 2: radius = 2.0, area = 12.56, colour = red

printRectangles ()

rectangle 1: perimeter = 4.0, area = 1.0
rectangle 2: perimeter = 8.0, area = 4.0

22) Finally, write a class called TestShape to create an instance of Shape, and two instances each of Circle and Rectangle, to test the above methods.

END