EXAMINATION QUESTION PAPER - SEMESTER 2, 2004/5

Module code:	**CSP032N**
Module title:	**Object - Oriented Software Design and Development**
Module leader:	**Dr. Q. DANG**

Date:	**Today**
Day / evening:	**Day**
Duration:	**2 hours**

Materials supplied:	**None**
Materials permitted:	**Pen;** **Blank paper**
Warning:	**Candidates are warned that possession of unauthorised materials in an examination is a serious assessment offence.**

Instructions to candidates:	**Answer any Four(4) of the SIX questions. Each question carries 25 marks. No credit will be given for attempting further questions.**

DO NOT TURN PAGE OVER UNTIL INSTRUCTED

QUESTION 1

Consider the development of a program that manages an appointment calendar to allow the user to add appointments and remove cancelled appointments.

(a) Determine and draw use case diagram(s) of the appointment calendar system.

(5 marks)

(b) List the relevant operations for each use case.

(5 marks)

(c) What are the classes that you would identify? Justify your answer. Draw a UML class diagram of the classes and their relationships.

(5 marks)

(d) Draw CRC cards for the classes from your answer to Question 1(c) to bring out class responsibilities and collaborators?

(5 marks)

(e) Draw a sequence diagram for adding an appointment to the system.

(5 marks)

QUESTION 2

(a) Polymorphism is a feature that characterises OO programming. Discuss how polymorphism is demonstrated in the implementation of the shapes' classes provided in Appendix A.

(8 marks)

(b) Inspect the Java source code provided in Appendix B, which makes use of the shape classes from Appendix A. Explain why the implementation of the `Comparator` interface (as in the class `ShapeComparatorByArea`) realises the form of polymorphism through *Interface Type* in Java?

(9 marks)

(c) In object-oriented programming, a class that makes use of services provided by another class is called client class. Explain the text *"Polymorphism supports the re-use of the client code as well as the extensibility of the service provider code"*, using the code examples in your answer to sub-questions (2a) and (2b).

(8 marks)

QUESTION 3

Inspect the source code of Java classes: Circle, Rectangle and Shape provided in Appendix A at the end of this paper.

(a) Square class is a subclass of the provided Rectangle class. Write code in Java to implement the Square class for it to inherit the area( ) and circumference( ) methods of the Rectangle class as they are, **without** being overridden.

(10 marks)

(b) Distinguish between *specifiers* private, protected and public in a Java program. Provide examples for each of them.

(5 marks)

(c) What is the purpose of declaring variables width and height as protected in the class Rectangle?

(5 marks)

(d) *"In object-oriented programming, a subclass not only inherits what has been built in its parent class(es) but can also be further extended with new functionality"*. Use the provided implementation of the shape's classes to explain the statement.

(5 marks)

QUESTION 4

(a) Inspect the code of a Java program in Appendix D. Write your code to add a JButton to the given program so that when the button is clicked at runtime the content of the textfield MyTextField will be reset (empty).

(9 marks)

(b) A software design pattern is a description of a problem and its solution that may be applied to many programming situations. Which design pattern does the program in your answer to sub-question 4(a) manifest? Justify your answer, using a table of the same format (but not content!) as the one given in sub-question 4(c) below.

(8 marks)

(c) Inspect the code given in Appendix B, which uses a `Comparator` object to sort the shape objects using the method `Arrays.sort(anArray, aComparator).` This is a manifestation of the STRATEGY pattern. Match the elements of the STRATEGY pattern with the actual pattern elements from the code using the table below (redraw it in your answer book).

(8 marks)

Name in Design Pattern	Actual Name in the code example
`Context`	?
`Strategy`	?
`ConcreteStrategy`	?
`DoWork()`	?

P.T.O.

QUESTION 5

(a) In the context of Object-oriented programming, concisely explain the concepts of Encapsulation and Information Hiding?

(8 marks)

(b) Inspect the code in Appendix A

(i) Briefly explain what are the *accessor* and *mutator* methods of a class.

(3 marks)

(ii) Name the accessor and mutator methods of the `Circle` class provided in Appendix A.

(3 marks)

(iii) Why is `Circle` class mutable whereas `Rectangle` class is not?

(3 marks)

(c) Look at the code and the screenshot of the program output provided in Appendix C. As shown in the appendix, the object `Child_1` created for Fred has changed after having been created. Is the class `Person` mutable? Justify your answer to this question.

(8 marks)

QUESTION 6

(a) Why would the use of encapsulation and information hiding result in more robust software? Use the code examples from Appendix C to illustrate your answer to this question.

(8 marks)

(b) Inspect the code of class `Person`, class `TestPerson`, and the screenshot of the `TestPerson` output at runtime provided in Appendix C at the end of this paper. Explain, at the code level, why the dates of birth of Fred and Tony are the same as displayed.

(8 marks)

(c) Modify the implementation of the `Person` class, but not the `TestPerson` class, in order for the output of the `TestPerson` class to display the correct dates of birth of Fred and Tony, applying the information hiding principle.

(9 marks)

APPENDIX A

```
/** the Shape abstract class */
public abstract class Shape
{
    public abstract double area();
    public abstract double circumference();

}//~public abstract class Sh...

/** Circle class */
public class Circle extends Shape
{
    private double r;
    public Circle(double radius)
    {
        r = radius;
    }
    public double area()
    {
        return 3.14*r*r;
    }
    public double circumference()
    {
        return 3.14*2*r;
    }
    public double getRadius()
    {
        return r;
    }
    public void setRadius(double radius)
    {
        r = radius;
    }
}//~public class Circle exte...

/** Rectangle class */
public class Rectangle extends Shape
{
    protected double width, height;
    public Rectangle(double w, double h)
    {
        width = w;
        height = h;
    }
    public double area()
    {
        return width * height;
    }
    public double circumference()
    {
        return 2*( width + height);
    }
}//~public class Rectangle e...

/** Square class */
public class Square extends Rectangle
{
      /** your code goes here */
}//~public class Square
```

APPENDIX B

```
/* This program demonstrates polymorphism thru use of Interface type in
 * Java, sorting using Comaparator interface
*/

import java.util.Arrays;
import java.util.*; //to use Comparator interface

/** implements Comparator for Shape objects */
class ShapeComparatorByArea implements Comparator
{
    public int compare(Object objA, Object objB)
    {
        //a method defined by Comparator interface
        Shape shapeA = (Shape) objA;
        Shape shapeB = (Shape) objB;
        return ((Double)shapeA.area()).compareTo((Double)shapeB.area());
    }
}//~class ShapeComparatorByC...


/** this class is to test the ShapeComparatorByArea */
public class  useOfComparableInterface
{
    public static void main(String[] args)
    {
        System.out.println("Sorting shapes by their areas");

        Object shapeArray[] = new Object[3];
        shapeArray[0]= new Circle(3.5);
        shapeArray[1]= new Rectangle(2.1, 3.4);
        shapeArray[2]= new Square(5.0);

        Comparator comp = new ShapeComparatorByArea();
        Arrays.sort(shapeArray, comp);

    }//~public static void main(...
}//~public class  useOfCompa...
```

APPENDIX C

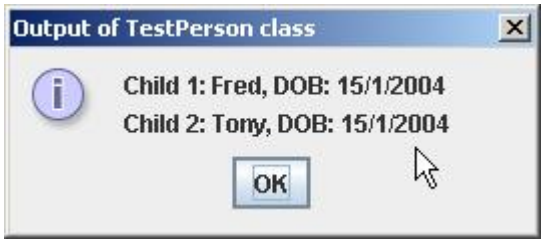
Start of code

```
/** Person Class */
class Person
{
    private String name;
    private Date DOB;
    Person (String aName, Date DateOfBirth)
    {
        name = aName;
        DOB= DateOfBirth;
    }

    public String toString()
    {
        return name + ", DOB: " + DOB.getDate() + "/" + DOB.getMonth() +
               "/"  + DOB.getYear();
    }
}//~class Person...
```

```
/** TestPerson Class */
import java.util.Date;
import javax.swing.JOptionPane;

public class TestPerson
{
    public static void main (String args[])
    {
        Date tempDate = new Date(2003, 1, 15);

        Person Child_1 = new Person("Fred", tempDate);

        tempDate.setTime((new Date(2004, 1, 15)).getTime());
            Person Child_2 = new Person("Tony", tempDate);

        String outputString = "Child 1: " + Child_1.toString() + "\n" +
                              "Child 2: " + Child_2.toString();
        JOptionPane.showMessageDialog(null, outputString,
                  "Output of TestPerson class",
                  JOptionPane.INFORMATION_MESSAGE);
    }//~public static void main ...
}//~public class TestPersonC...
```

end of code

The program output screenshot



APPENDIX D

```
//Start of code
import java.awt.event.*;
import javax.swing.*;
public class TextFieldTest extends JFrame
{
    private JTextField MyTextField = new JTextField("Hello World!");
    // set up GUI
    public TextFieldTest()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(MyTextField);

        setSize( 250, 100 );
        setVisible( true );
    }//~public TextFieldTest()...

    // execute application
    public static void main( String args[] )
    {
        TextFieldTest application = new TextFieldTest();
    }
}
```