

NATIONAL DIPLOMA IN COMPUTING
(Information Technology)

Object Orientation with Design Patterns
CM302

Semester I

Internal Examiner(s): Ms. Orla McMahon

External Examiner(s): Mr John Dunnion
Prof. Gerard Parr

August 2004
Time of examination here

Instructions to candidates:

- 1) Section A: Attempt any five parts.**
- 2) Section B: Answer any 3 Questions.**
- 3) All questions carry equal marks.**

DO NOT TURN OVER THIS PAGE UNTIL YOU ARE TOLD TO DO SO

Section A

Attempt any 5 parts of this question

(5 marks each)

a)	Describe the difference between Class Adapters and Object Adapters . [5 Marks]
b)	Graphical representations of design patterns only capture the end product of the design process. Why? List and briefly describe four essential elements that can be used to describe a design pattern. [5 Marks]
c)	Any object in Java can be queried to see what methods and parameters make up its interface. Write a code sample that demonstrates how you would query a Java object for its methods only. [5 Marks]
d)	Describe with the aid of a code sample how you can easily determine that you are dealing with two identical instances of a Flyweight class. [5 Marks]
e)	Briefly describe the difference between the Factory Method Pattern and the Abstract Factory Pattern . [5 Marks]
f)	Briefly describe the Template Pattern and the four types of methods that can be found in a template class. [5 Marks]
g)	What is a “Design Pattern”? What are the advantages of using Design Patterns? [5 Marks]

(25 marks)

Section B

Candidates should attempt any 3 of the following questions.

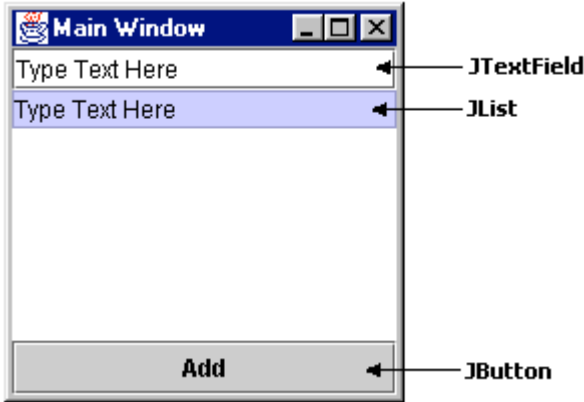
Question 2

2a)	<p>What are Creational Patterns? Name four.</p> <p>[6 Marks]</p>
2b)	<p>What is the Singleton Pattern and when would you use it?</p> <p>Write a simple Singleton Class that uses exception handling to force the programmer to deal with the possibility of null pointers.</p> <p>[8 Marks]</p>
2c)	<p>The following UML class diagram describes a Simple Factory Pattern that is used to create two different kinds of NumberList objects based on a value passed to the getNumberList method.</p> <pre>classDiagram class NumberList { <abstract> #double_list: double #int_list: int display() getDoubleList(): double getIntList(): int sum(): String } class DoubleList { -size: int DoubleList(String) } class IntList { -size: int IntList(String) } class NumberFactory { getNumberList(String): NumberList } NumberList < -- DoubleList NumberList < -- IntList NumberFactory --> NumberList : <<creates>></pre> <p>Using the following test code as a reference, implement each of the classes shown in the UML class diagram above.</p> <pre>String list1 = new String("1 2 3 4 5 6 7 8 9 10"); String list2 = new String("1.1 2.2 3.3 4.4 5.5 6.6 7.7 8.8 9.9 10.1"); NumberFactory nfactory = new NumberFactory(); nfactory.getNumberList(list1).display(); NumberList numberlist2 = nfactory.getNumberList(list2);</pre>

	<pre>numberlist2.display(); System.out.println("Sum of list 2 : " + numberlist2.sum());</pre> <p>[11 Marks]</p>
--	---

(25 marks)

Question 3

3a)	<p>The Chain of Responsibility Pattern helps to keep separate the knowledge of what each object in a program can do. That is it reduces the coupling between objects so that they can act independently.</p> <p>Describe four situations where this pattern might be used.</p> <p style="text-align: right;">[4 Marks]</p>
3b)	<p>The program given in code listing 1 (on next page) creates a simple user interface that allows the user to select menu items, File Open and File Exit, and click on a button labelled Blue that turns the background of the window blue.</p> <p>As long as there are only a few menu items and buttons this approach works fine, but when there are several menu items and buttons the <i>actionPerformed</i> code can get pretty unwieldy.</p> <p>Using a simple Command Pattern re-write the appropriate sections of this program so that the conditional block in the <i>actionPerformed</i> code is removed.</p> <p style="text-align: right;">[10 Marks]</p>
3c)	<p>Write a program that allows the user to add items to a JList control by typing text into a JTextField and clicking on an add button. Your program should take advantage of the Model View Controller (MVC) architecture.</p> <p>In developing your program you must incorporate the following classes</p> <ul style="list-style-type: none"> • MainWindow extends JFrame • ListData extends AbstractListModel <div style="text-align: center;">  </div> <p style="text-align: right;">[11 Marks]</p>

(25 marks)

Code Listing 1

```
public class SimpleApp extends Frame implements ActionListener
{
    Menu mnuFile;
    MenuItem mnuOpen, mnuExit;
    Button btnBlue;
    Panel p;

    public SimpleApp()
    {
        super("Simple App");

        //Create a new menu bar for the frame
        MenuBar mbar = new MenuBar();
        setMenuBar(mbar);

        // Create the menu items and add them to the bar
        mnuFile = new Menu("File", true);
        mbar.add(mnuFile);
        mnuOpen = new MenuItem("Open...");
        mnuFile.add(mnuOpen);
        mnuExit = new MenuItem("Exit");
        mnuFile.add(mnuExit);

        // Add an actionlistener for each menu item
        // actions will be handled by this class
        mnuOpen.addActionListener(this);
        mnuExit.addActionListener(this);

        // Create a button for the frame
        btnBlue = new Button("Blue");

        // Create a panel and add the button
        p = new Panel();
        add(p);
        p.add(btnBlue);

        // Add an actionlistener for the button
        // actions will be handled by this class
        btnBlue.addActionListener(this);

        setBounds(100,100,200,100);
        setVisible(true);
    }

    // Handle actions from the menu items and button
    public void actionPerformed(ActionEvent e)
    {
        // Determine the source of the action and
```

```

        // carry out the appropriate action
        Object obj = e.getSource();
        if(obj == mnuOpen)
            fileOpen();
        if (obj == mnuExit)
            exitClicked();
        if (obj == btnBlue)
            redClicked();
    }

    // Called from actionPerformed when exit selected
    private void exitClicked()
    {
        System.exit(0);
    }
    // Called from actionPerformed when Open selected
    private void fileOpen()
    {
        FileDialog fDlg = new FileDialog(this, "Open a
            file",FileDialog.LOAD);
        fDlg.show();
    }
    // Called from actionPerformed when button clicked
    private void redClicked()
    {
        p.setBackground(Color.blue);
    }

    static public void main(String argv[])
    {
        new SimpleApp();
    }
}

```

Question 4

4a)	<p>Define the terms Observer and Subject with regard to the Observer Pattern.</p> <p>Write an abstract class called Observer and an abstract class called Subject that could be used to implement the Observer Pattern.</p> <p style="text-align: right;">[6 Marks]</p>
4b)	<p>The following UML class diagram describes a commonly used creational pattern. Name the pattern and describe how each of the classes in the diagram contributes to the patterns implementation.</p> <div data-bbox="365 598 1230 1171" data-label="Diagram"> <pre> classDiagram class Garden { +Garden() +getBorder() Plant +getCenter() Plant +getShade() Plant } class Gardener { -clearPlants -setBorder -setCenter -setGUI -setShade } class Plant { +Plant +getName } class AnnualGarden { +getBorder +getCenter +getShade } class PerennialGarden { +getBorder +getCenter +getShade } class VeggieGarden { +getBorder +getCenter +getShade } Garden "0..1" -- "1" Gardener : -garden Garden < -- AnnualGarden Garden < -- PerennialGarden Garden < -- VeggieGarden </pre> </div> <p style="text-align: right;">[8 Marks]</p>

4c)	<p>The Iterator Pattern is one of the simplest and most frequently used of the design patterns.</p> <p>Java supports the Iterator pattern by providing Enumerations for its Vector and Hashtable classes.</p> <p>Given a Vector containing a list of names, write a class which implements the Enumeration interface so as to provide a filter that will only iterate through names that begin with some prefix.</p> <p>So, for example, a test program for the Filter class might look as follows:</p> <pre> class MainApp { private Vector data; public MainApp() { data = new Vector(); data.addElement("Alan"); data.addElement("Conor"); data.addElement("Joanne"); data.addElement("David"); data.addElement("John"); data.addElement("Martin"); } public void filterNames() { Filter filter = new Filter(data.elements(), "Jo"); while(filter.hasMoreElements()) { String s = (String)filter.nextElement(); System.out.println(s); } } public static void main(String[] args) { MainApp app = new MainApp(); app.filterNames(); } } </pre> <p style="text-align: right;">[11 Marks]</p>
-----	---

(25 marks)

Question 5

5a)	Compare the intent of the Facade and Adapter design patterns. [6 Marks]
5b)	Use an intuitive example to explain the intent of the Composite design pattern. [6 Marks]
5c)	<p>Examine the classes in code listing 2 (on next page), then answer the following questions:</p> <p>i) Create a new Decorator class called CrazyDecorator. The CrazyDecorator class must change a JComponent's background colour to red when the mouse enters the JComponent. The JComponent's background colour must be reset to its original colour when the mouse exits the JComponent. [9 Marks]</p> <p>ii) Create a simple test program call DecoratorTester which will decorate a JButton with the SlashDecorator <u>AND</u> the CrazyDecorator. [4 Marks]</p>

(25 marks)

Code Listing 2

Decorator.java

```
public class Decorator extends JComponent
{
    public Decorator(JComponent c)
    {
        setLayout(new BorderLayout());
        add("Center", c);
    }
}
```

SlashDecorator.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.text.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;

public class SlashDecorator extends Decorator {
    int x1, y1, w1, h1;

    public SlashDecorator(JComponent c) {
        super(c);
    }
    public void setBounds(int x, int y, int w, int h) {
        x1 = x; y1 = y;
        w1 = w; h1 = h;
        super.setBounds(x, y, w, h);
    }
    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
        g.drawLine(0, 0, w1, h1);
    }
}
```