INSTITUTE OF TECHNOLOGY BLANCHARDSTOWN

BACHELOR OF SCIENCE 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

January 2006
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)

Question 1

a) Design patterns were discovered by the Gang of Four.
Briefly describe what led to the discovery of design patterns and why they are so important in software projects.

[5 Marks]

b) Describe how the Proxy Design pattern works and give three situations where it might be used.

[5 Marks]

c) 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]

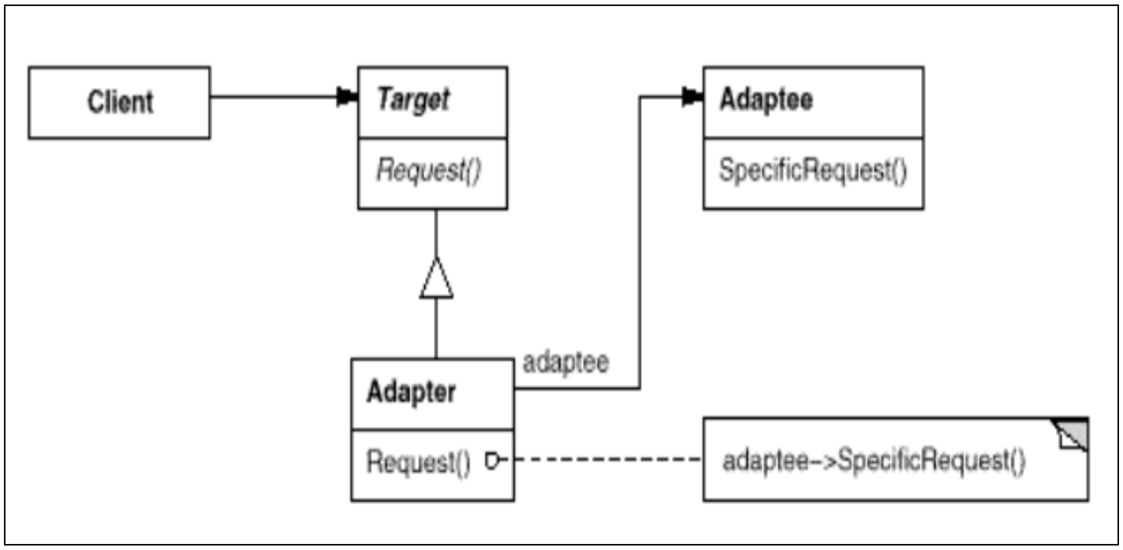
d) When using the Singleton pattern, one approach is to use a static variable and to make the constructor private.

Explain why you would do this in order to implement the Singleton pattern.

[5 Marks]

Question 1 (Contd.)

e) Given the following UML diagram, explain briefly with the aid of an example the role of each participant in the Adapter pattern.



[5 Marks]

f) Use an intuitive example to explain the intent of the Composite design pattern.

[5 Marks]

g) When using the Flyweight pattern, there are two states called intrinsic and extrinsic. With the aid of an example, explain the differences between these two states.

[5 Marks]

(Total Marks 25)

Section B
Candidates should attempt any 3 of the following questions.

Question 2

a) Even though the Singleton pattern is grouped with other creational patterns, why to some extent is it different?

[4 Marks]

b) Consider a system for processing ATM card transactions. A thief is using a stolen ATM card number to steal funds, concurrent with the legitimate user's withdrawing funds. How could you design a Singleton to reduce this risk?

[9 Marks]

c) There are various factory design patterns available for creating objects. Identify and describe three such patterns.

Illustrate each pattern with a real world example through the use of a UML diagram.

[12 Marks]

(Total Marks 25)

Question 3

a) The Chain of Responsibility pattern does not have to use a linear chain.
What does this statement mean?
What, if any, implications are there if a non-linear chain is used?

[4 Marks]

b) Compare the intent of the Facade and Adapter design patterns.

[6 Marks]

c) Read this case study and answer the questions that follow:

You have been asked to deploy an application that displays details about a person's email address book.
The address book contains two different user interfaces. One interface displays details about people in general such as their name, email address, phone number and company name.
The second interface displays details about groups of people such as the name of the group, its purpose, a list of its members and their email addresses.

i) Identify the design pattern that you would use to deploy this application.

ii) Explain why you would use this design pattern.

iii) Illustrate your answer with the aid of a UML diagram and include a description of the participants.

iv) Describe four consequences of using this pattern.

[15 Marks]

(Total Marks 25)

Question 4

a) In general a pattern has four essential elements.

Identify and describe the four essential elements.

[4 Marks]

b) A common pattern cited in early literature on programming frameworks is the Model-View-Controller (MVC) pattern.

Briefly describe with the aid of an example the role of the various participants in the MVC pattern.

[5 Marks]

c) When someone enters an incorrect value in a cell of a JTable, you might want to change the colour of the row to indicate the problem.

Identify a design pattern that could be used for this purpose.
Describe in detail how this design pattern could solve the given request.

[6 Marks]

d) When you build a Java user interface, you provide controls – menu items, buttons, check boxes, and so on – to allow the user to tell the program what to do. When a user selects one of these controls, the program receives an **ActionEvent** which it must trap by implementing the **ActionListener** interfaces (actionPerformed). This code can get quite cumbersome if there are many controls that make up the user interface.

Describe with the aid of some sample code one method that can be used to reduce the amount of coding in the **actionPerformed** method by forwarding specific commands to specific user interface controls.

[10 Marks]

(Total Marks 25)

Question 5

a) The Iterator pattern is one of the simplest and most frequently used of the design patterns.
Java supports the Iterator pattern by providing Enumerations for its Vector and Hashtable classes.

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.

So, for example, a test program for the Filter class might look as follows:

```
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();
}
}
```

[12 Marks]

b) The program given in code listing 1 (below) uses a simple Decorator pattern to create a cool button.

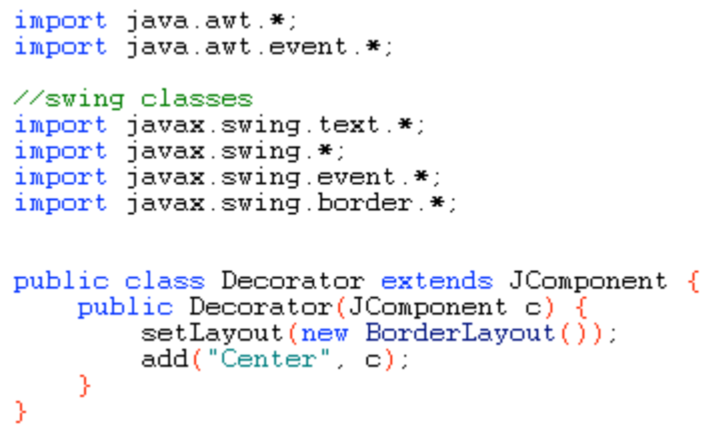
Describe in detail the role each class plays in order to implement the Decorator pattern, together with an explanation of each of the class methods.

[13 Marks]

(Total Marks 25)

Code Listing 1

Decorater.Java



```java
import java.awt.*;
import java.awt.event.*;

//swing classes
import javax.swing.text.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;


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

CoolDecorator.java

```
import java.awt.*;
import java.awt.event.*;

//swing classes
import javax.swing.text.*;
import javax.swing.*;
import javax.swing.event.*;
import javax.swing.border.*;


public class CoolDecorator extends Decorator {
    boolean mouse_over;
    JComponent thisComp;

    public CoolDecorator(JComponent c) {
        super(c);
        mouse_over = false;
        thisComp = this;
        c.addMouseListener(new MouseAdapter() {

               public void mouseEntered(MouseEvent e) {
                       mouse_over = true;
                       thisComp.repaint();
               }

              public void mouseExited(MouseEvent e) {
                       mouse_over = false;
                       thisComp.repaint();
               }
          });
    }



    public void paint(Graphics g) {
        super.paint(g);
        if (! mouse_over) {
            Dimension d = super.getSize();
            g.setColor(Color.lightGray);
            g.drawRect(0, 0, d.width-1, d.height-1);
            g.drawLine(d.width-2, 0, d.width-2, d.height-1);
            g.drawLine(0, d.height-2, d.width-2, d.height-2);
        }
    }
}
```

SlashDecorator.java

```java
import java.awt.*;
import java.awt.event.*;

//swing classes
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);
    }
}
```

slashWindow.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;

//swing classes
import javax.swing.text.*;
import javax.swing.*;
import javax.swing.event.*;


public class slashWindow extends JxFrame
   implements ActionListener
{
   JButton CButton, DButton, Quit;
   public slashWindow()
   {
      super ("Deco Button");
      JPanel jp = new JPanel();

      getContentPane().add(jp);

      jp.add( new CoolDecorator(
              CButton = new JButton("Cbutton")));

      jp.add( new SlashDecorator(new CoolDecorator(
              DButton = new JButton("Dbutton"))));


      jp.add(Quit = new JButton("Quit"));
      Quit.addActionListener(this);
      setSize(new Dimension(200,100));

      setVisible(true);
      Quit.requestFocus();
   }
   public void actionPerformed(ActionEvent e)
   {
      System.exit(0);
   }
   static public void main(String argv[])
   {
      new slashWindow();
   }
}
```