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 2005
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) Describe briefly what is meant by the term ‘Design Pattern’ and in particular why they are used in software projects.

[5 Marks]

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

Briefly describe the role of the various participants in the **MVC** pattern.

[5 Marks]

c) What is the intent of the **Builder** pattern?

Illustrate your answer with a simple example.

[5 Marks]

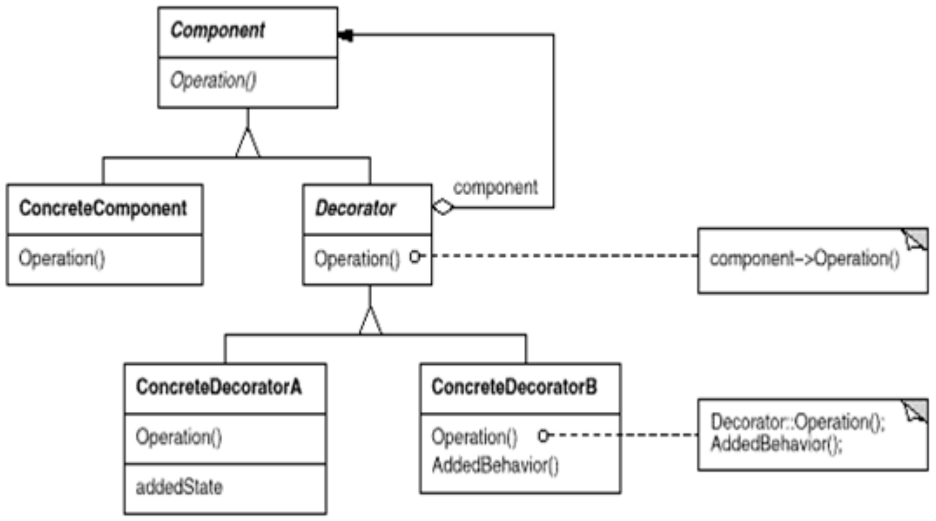
d) When using the **Singleton** pattern, one approach is to use a static variable and to make the constructor private.

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 the role of each participant in the **Decorator** pattern.



[5 Marks]

f) Design patterns can fall into one of three categories.
What are these categories?
Provide a brief description of each category and **two** examples of design patterns that fall within each category.

[5 Marks]

g) 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]

(Total Marks 25)

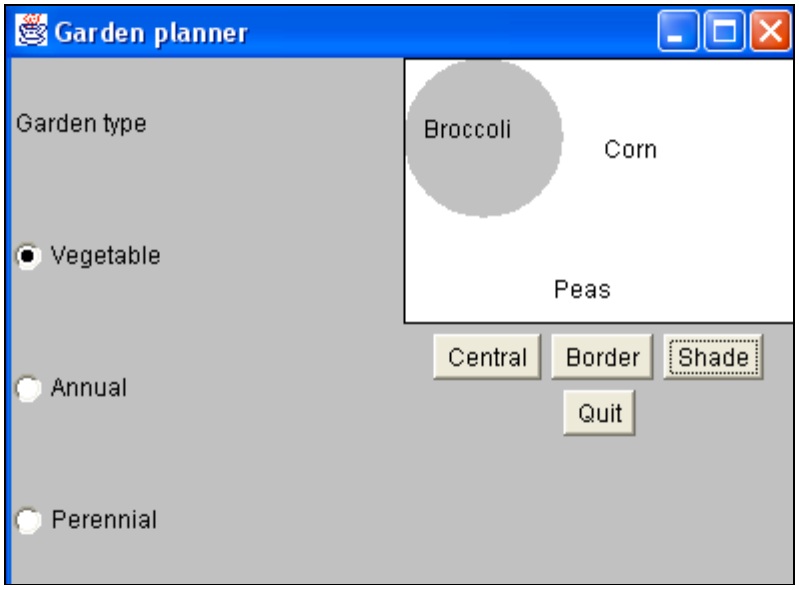
Section B
Candidates should attempt any 3 of the following questions.

Question 2

Read this case study and answer the questions that follow:

Assume that you are working as a Java Software Designer. You have been requested to update the Garden Planner software system contained in **code listing 1**. The current system allows a gardener to select a particular garden and a garden area. The plant for that particular area will then be displayed. For example the gardener could select the vegetable garden and then display the plant that is suitable to a shade area. Currently the GUI operates so that the gardener simply selects the garden type, clicks on the appropriate area button (ie central, border or shade) and a suitable plant for the given garden and area appears on the screen.

The following diagram shows the system in operation.



You have been asked to modify the system so that additional gardens can be incorporated into the Garden Planner. The new types of gardens are a Bonsai garden and a Rose garden. In addition to this the garden designer would also like to see the height and watering frequency of each plant that is displayed.

Question 2 (Contd.)

a) Draw an accurate UML diagram that illustrates the structure of the **Abstract Factory** pattern when applied to the updated Garden Planner System.

[5 Marks]

b) Taking the above requirements into account, write new classes for the Plant, BonsaiGarden and RoseGarden participants. Assume that the other garden participants have been already modified.

[8 Marks]

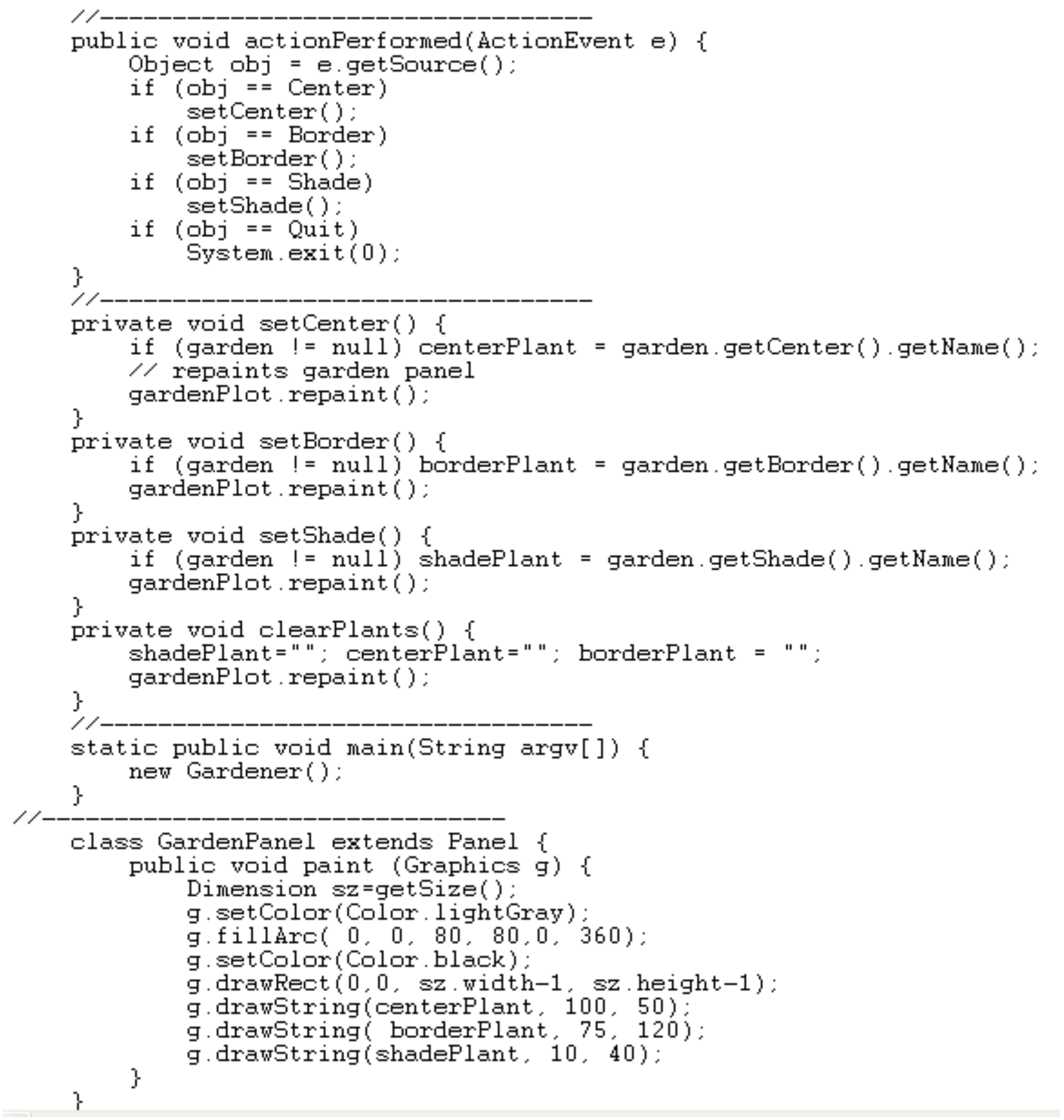
c) Describe in a step-by-step fashion, how you would edit the current Garden Planner (GUI) (Gardener.java) so that a Rose or Bonsai garden can be selected and in addition to the plant name, the height and watering frequency can also be displayed.

[12 Marks]

(Total Marks 25)

Code Listing 1
Gardener.java

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

//illustrates use of Abstract Factory pattern
public class Gardener extends Frame
implements ActionListener {
    private Checkbox Veggie, Annual, Peren;
    private Button Center, Border, Shade, Quit;
    private Garden garden = null;
    private GardenPanel gardenPlot;
    private String borderPlant = "", centerPlant = "", shadePlant = "

    public Gardener() {
        super("Garden planner");
        setGUI();
    }
    //---------------------------------
    private void setGUI() {
        setBackground(Color.lightGray);
        setLayout(new GridLayout(1,2)); // 1 row 2 columns
        Panel left = new Panel();
        add(left);
        Panel right= new Panel();
        add(right);

        //create label and 3 radio buttons on left side

        left.setLayout(new GridLayout(4, 1)); // 4 rows and 1 column
        left.add(new Label("Garden type"));
        CheckboxGroup grp= new CheckboxGroup();
        Veggie = new Checkbox("Vegetable", grp, false);
        Annual = new Checkbox("Annual", grp, false);
        Peren = new Checkbox("Perennial", grp, false);

        left.add(Veggie);
        left.add(Annual);
        left.add(Peren);

        Veggie.addItemListener(new VeggieListener());
        Peren.addItemListener(new PerenListener());
        Annual.addItemListener(new AnnualListener());



        //now create right side
        right.setLayout(new GridLayout(2,1)); // 2 rows 1 column
        gardenPlot = new GardenPanel(); // defined below
        gardenPlot.setBackground(Color.white);
        Panel botRight = new Panel();

        right.add(gardenPlot);
        right.add(botRight);
        Center = new Button("Central");
        Border =  new Button("Border");
        Shade = new Button("Shade");
        Quit = new Button("Quit");
        // add buttons to panel
        botRight.add(Center);
        Center.addActionListener(this);
        botRight.add(Border);
        Border.addActionListener(this);
        botRight.add(Shade);
        Shade.addActionListener(this);
        botRight.add(Quit);
        Quit.addActionListener(this);
        setBounds(200,200, 400,300);
        setVisible(true);

    }
```



```
    //-----------------------------------
    public void actionPerformed(ActionEvent e) {
        Object obj = e.getSource();
        if (obj == Center)
            setCenter();
        if (obj == Border)
            setBorder();
        if (obj == Shade)
            setShade();
        if (obj == Quit)
            System.exit(0);
    }
    //-----------------------------------
    private void setCenter() {
        if (garden != null) centerPlant = garden.getCenter().getName();
        // repaints garden panel
        gardenPlot.repaint();
    }
    private void setBorder() {
        if (garden != null) borderPlant = garden.getBorder().getName();
        gardenPlot.repaint();
    }
    private void setShade() {
        if (garden != null) shadePlant = garden.getShade().getName();
        gardenPlot.repaint();
    }
    private void clearPlants() {
        shadePlant=""; centerPlant=""; borderPlant = "";
        gardenPlot.repaint();
    }
    //-----------------------------------
    static public void main(String argv[]) {
        new Gardener();
    }
//--------------------------------
    class GardenPanel extends Panel {
        public void paint (Graphics g) {
            Dimension sz=getSize();
            g.setColor(Color.lightGray);
            g.fillArc( 0, 0, 80, 80,0, 360);
            g.setColor(Color.black);
            g.drawRect(0,0, sz.width-1, sz.height-1);
            g.drawString(centerPlant, 100, 50);
            g.drawString( borderPlant, 75, 120);
            g.drawString(shadePlant, 10, 40);
        }
    }
    //------------------------------
    class VeggieListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            // creating an instance of garden as VeggieGarden
            garden = new VeggieGarden();
            clearPlants();
        }
    }
    //------------------------------
    class PerenListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            // creating an instance of garden as PerennialGarden
            garden = new PerennialGarden();
            clearPlants();
        }
    }
    //------------------------------
    class AnnualListener implements ItemListener {
        public void itemStateChanged(ItemEvent e) {
            garden = new AnnualGarden();
            clearPlants();
        }
    }


}     //end of Gardener class
```

Garden.java

```
public abstract class Garden {
    public abstract Plant getShade();
    public abstract Plant getCenter();
    public abstract Plant getBorder();
}
```

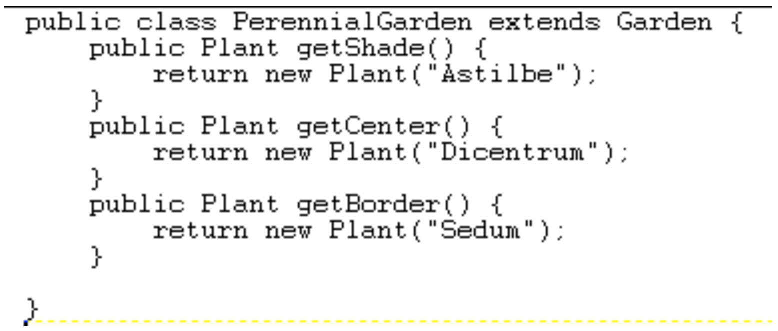
Plant.java

```
public class Plant {
    private String name;
    public Plant(String pname) {
        name = pname;      //save name
    }
    public String getName() {
        return name;
    }
}
```

AnnualGarden.java

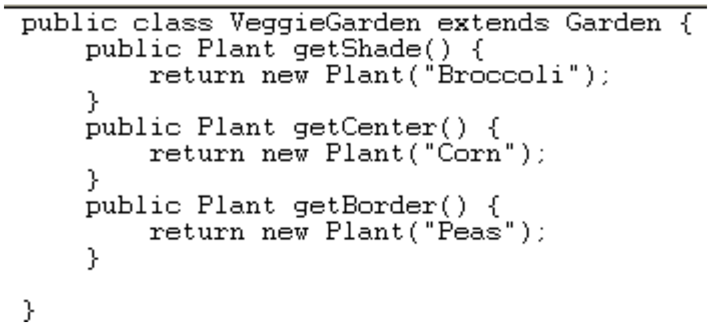
```
public class AnnualGarden extends Garden {
    public Plant getShade() {
        return new Plant("Coleus");
    }
    public Plant getCenter() {
        return new Plant("Marigold");
    }
    public Plant getBorder() {
        return new Plant("Alyssum");
    }

}
```

PerennialGarden.java



```
public class PerennialGarden extends Garden {
    public Plant getShade() {
        return new Plant("Astilbe");
    }
    public Plant getCenter() {
        return new Plant("Dicentrum");
    }
    public Plant getBorder() {
        return new Plant("Sedum");
    }

}
```

VeggieGarden.java



```
public class VeggieGarden extends Garden {
    public Plant getShade() {
        return new Plant("Broccoli");
    }
    public Plant getCenter() {
        return new Plant("Corn");
    }
    public Plant getBorder() {
        return new Plant("Peas");
    }

}
```

Question 3

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

[6 Marks]

b) With the aid of a UML diagram, describe the components of the **Composite** design pattern

[6 Marks]

c) The program code in **code listing 2** uses a **Flyweight** pattern to create folders. Basically the system draws an icon for each folder with each person's name displayed under the folder.

Describe in detail the role each class plays in order to implement the **Flyweight** pattern. Within each class, provide an explanation of the class methods.

[13 Marks]

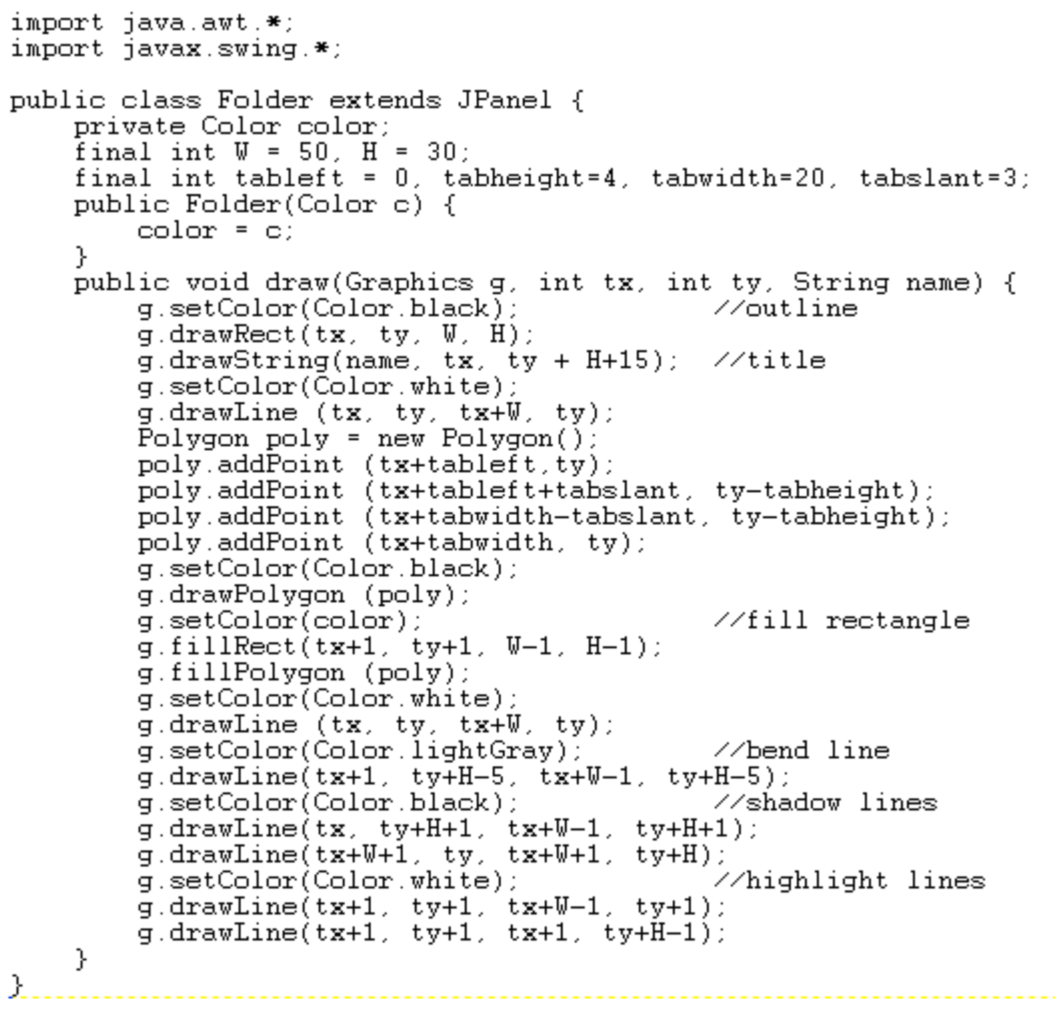
(Total Marks 25)

Code Listing 2

FolderFactory.java

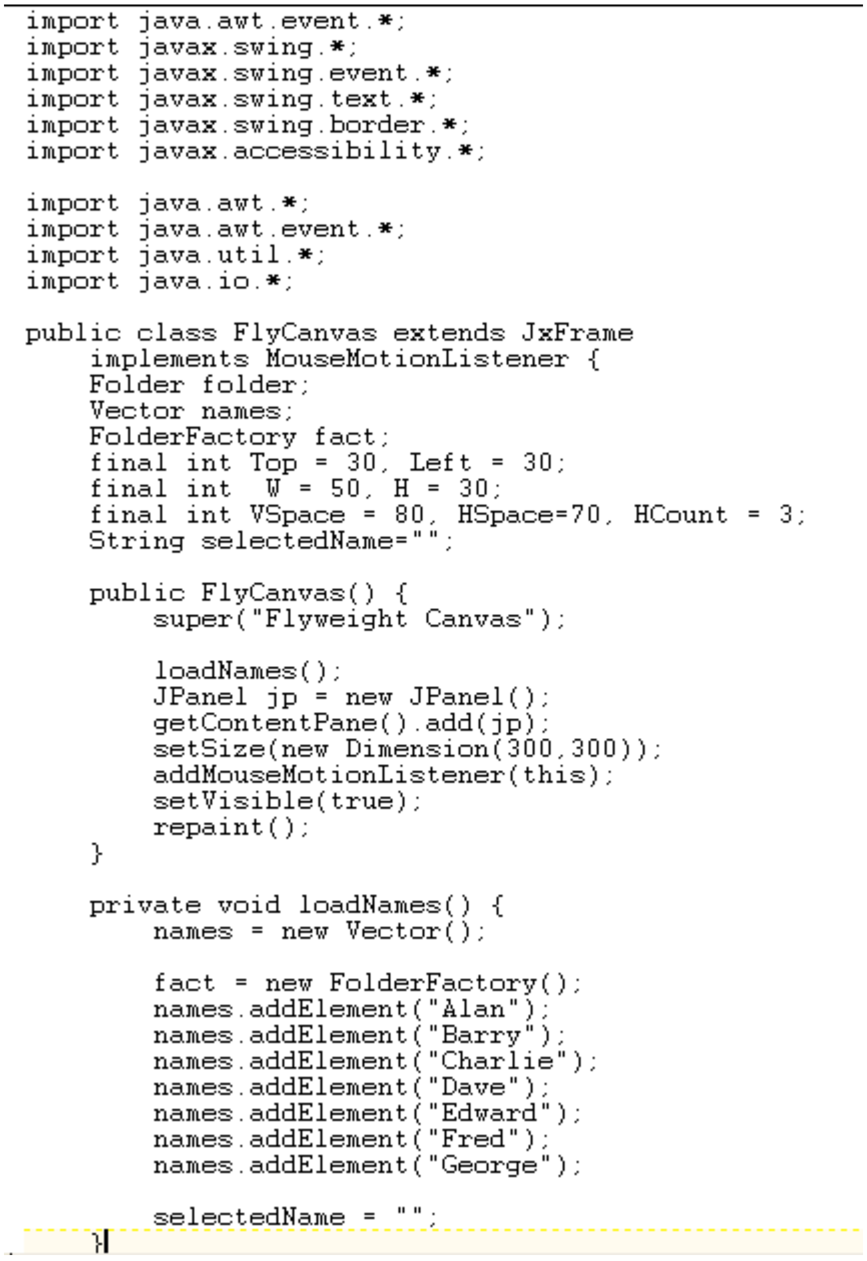
```
import java.awt.*;

public class FolderFactory {
    Folder unSelected, Selected;
    public FolderFactory() {
        Color brown = new Color(0x5f5f1c);
        Selected =  new Folder(brown);
        unSelected = new Folder(Color.yellow);
    }

    public Folder getFolder(boolean isSelected) {
        if (isSelected)
            return Selected;
        else
            return unSelected;
    }
}
```

Folder.java



```
import java.awt.*;
import javax.swing.*;

public class Folder extends JPanel {
    private Color color;
    final int W = 50, H = 30;
    final int tableft = 0, tabheight=4, tabwidth=20, tabslant=3;
    public Folder(Color c) {
        color = c;
    }
    public void draw(Graphics g, int tx, int ty, String name) {
        g.setColor(Color.black);            //outline
        g.drawRect(tx, ty, W, H);
        g.drawString(name, tx, ty + H+15);  //title
        g.setColor(Color.white);
        g.drawLine (tx, ty, tx+W, ty);
        Polygon poly = new Polygon();
        poly.addPoint (tx+tableft,ty);
        poly.addPoint (tx+tableft+tabslant, ty-tabheight);
        poly.addPoint (tx+tabwidth-tabslant, ty-tabheight);
        poly.addPoint (tx+tabwidth, ty);
        g.setColor(Color.black);
        g.drawPolygon (poly);
        g.setColor(color);                  //fill rectangle
        g.fillRect(tx+1, ty+1, W-1, H-1);
        g.fillPolygon (poly);
        g.setColor(Color.white);
        g.drawLine (tx, ty, tx+W, ty);
        g.setColor(Color.lightGray);        //bend line
        g.drawLine(tx+1, ty+H-5, tx+W-1, ty+H-5);
        g.setColor(Color.black);            //shadow lines
        g.drawLine(tx, ty+H+1, tx+W-1, ty+H+1);
        g.drawLine(tx+W+1, ty, tx+W+1, ty+H);
        g.setColor(Color.white);            //highlight lines
        g.drawLine(tx+1, ty+1, tx+W-1, ty+1);
        g.drawLine(tx+1, ty+1, tx+1, ty+H-1);
    }
}
```

FlyCanvas.java



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

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

public class FlyCanvas extends JxFrame
    implements MouseMotionListener {
    Folder folder;
    Vector names;
    FolderFactory fact;
    final int Top = 30, Left = 30;
    final int  W = 50, H = 30;
    final int VSpace = 80, HSpace=70, HCount = 3;
    String selectedName="";

    public FlyCanvas() {
        super("Flyweight Canvas");

        loadNames();
        JPanel jp = new JPanel();
        getContentPane().add(jp);
        setSize(new Dimension(300,300));
        addMouseMotionListener(this);
        setVisible(true);
        repaint();
    }

    private void loadNames() {
        names = new Vector();

        fact = new FolderFactory();
        names.addElement("Alan");
        names.addElement("Barry");
        names.addElement("Charlie");
        names.addElement("Dave");
        names.addElement("Edward");
        names.addElement("Fred");
        names.addElement("George");

        selectedName = "";
    }
    public void paint(Graphics g) {
        Folder f;
        String name;

        int j = 0;      //count number in row
        int row = Top;  //start in upper left
        int x = Left;

        //go through all the names and folders
        for (int i = 0; i< names.size(); i++) {
            name = (String)names.elementAt(i);
            if (name.equals(selectedName))
                f = fact.getFolder(true);
            else
                f = fact.getFolder(false);
            //have that folder draw itself at this spot
            f.draw(g, x, row, name);

            x = x + HSpace;    //change to next posn
            j++;
            if (j >= HCount) { //reset for next row
                j = 0;
                row += VSpace;
                x = Left;
            }
        }
    }

    public void mouseMoved(MouseEvent e) {
        int j = 0;      //count number in row
        int row = Top;  //start in upper left
        int x = Left;

        //go through all the names and folders
        for (int i = 0; i< names.size(); i++) {
            //see if this folder contains the mouse
            Rectangle r = new Rectangle(x,row,W,H);
            if (r.contains(e.getX(), e.getY())) {
                selectedName=(String)names.elementAt(i);
                repaint();
            }
            x = x + HSpace;      //change to next posn
            j++;
            if (j >= HCount) {   //reset for next row
                j = 0;
                row += VSpace;
                x = Left;
            }
        }
    }

    public void mouseDragged(MouseEvent e) {
    }

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

// End of FlyCanvas

Question 4

a) Using some simple UML diagrams together with real-world examples, describe the difference between the **Factory** pattern and the **Factory Method** pattern.

[8 Marks]

b) Explain how the **Proxy** pattern works.
Illustrate your answer with the aid of three examples.

**[6 Marks]
]**

c) The program given in **code listing 3 (on next page)** creates a simple user interface that allows the user to select menu items, File|Open and File|Exit,. When the user clicks on a button labelled 'Blue', the background colour of the window turns to blue.

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 *actionPerformed* code can get pretty unwieldy.

Using a simple **Command Pattern** re-write the appropriate sections of this program so that the conditional block in the *actionPerformed* code is removed.

[11 Marks]

(Total Marks 25)

Code Listing 3

```
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 5

a) Explain using a simple example how the **Facade** pattern can simplify the use of a complex system for clients.

[5 Marks]

b) What is the intent of the **Chain of Responsibility** pattern?

Describe two consequences of applying the **Chain of Responsibility** pattern to a software design problem.

[8 Marks]

c) The following example allows the user to display points, lines and squares.
In designing the system, the designer decided to include these shapes in a higher level concept that could be called a 'displayable shape'.
To accomplish this, the designer created a shape class and then derived from it the classes that represented points, lines and squares (see fig below).

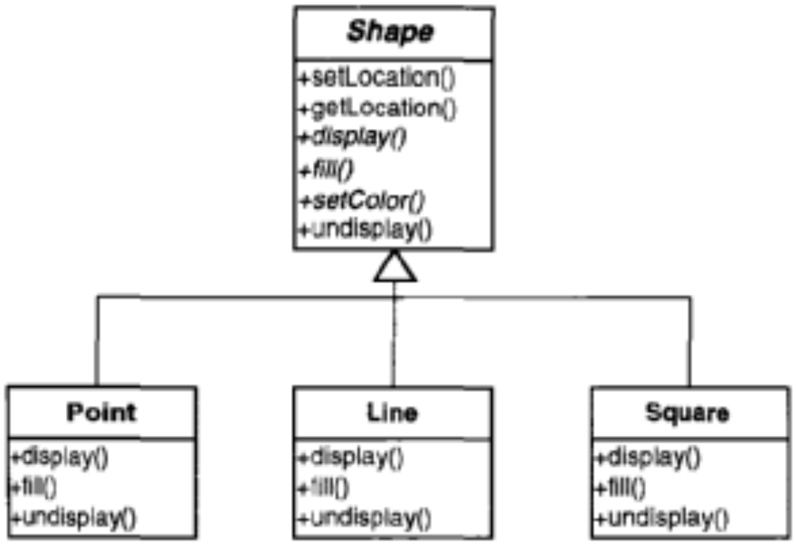


Figure 7-3 Points, Lines, and Squares showing methods.

After implementing the above system, the designer was then asked to include a class that could draw circles. This class should also contain all of the required methods such as display(), fill() and undisplay().
The designer then discovered that a colleague had written a circle class that perfomed the exact tasks required. Unfortunately the methods within this class were called displayIt(), fillIt(), undisplayIt().

I. Using the **Adapter** pattern, explain in a step-by-step fashion how you would incorporate the given 'circle' class into the above system.

[4 Marks]

II. Draw an accurate UML diagram that reflects the updated system.

[4 Marks]

III. Provide a Java code fragment for the class 'circle' that implements the **Adapter** pattern.

[4 Marks]

(Total Marks 25)