Sashi Bhusan Maharana
CSE, CUTM

Session 4

LAYOUT MANAGERS

SESSION OBJECTIVES

At the end of this session, you will be able to –

- Define and identify the functions of a Layout Manager
- List the types of Layouts
- Describe the applications of Layout Managers
- Explain how to set Layouts
- Discuss the FlowLayout
- Explain the BorderLayout
- Describe the GridLayout
- Describe the CardLayout
- Explain the GridBagLayout

4.1 Introduction

In the previous session we described the *java.applet.Applet* class and learnt various aspects of programming with applets. Event handling with applets was also discussed with a few examples. The Graphics, Font and FontMetrics classes were also explored. We appreciated the use of the Color class with respect to Java applications and applets.

In this session, we will look at the various layout managers that are available and identify the layout managers that are appropriate to the application.

4.2 The Layout Manager

Screen components on a user interface may arranged in various ways. For instance, they can be horizontally arranged in a serial fashion or arranged in a grid-like manner. Each of these schemes could be referred to as the *'layout'* of components. To manage these layouts we need to have layout managers or standard schemes that will set the design for a layout. Whenever the screen has to be resized or any item on the screen has to be redrawn, the layout manager will come into picture and arrange the items accordingly. Different languages have their own different ways of achieving this.

For instance, in certain RAD (Rapid Application Development) tools we have a drag and drop approach to create GUI elements and position them.

Procedure to arrange layouts would be-

1) Create GUI elements

2) Position them individually or set the desired layout scheme

4.2.1 Layout Manager in Java

We now know that the Java programming language is implemented on many different platforms. Each of these platforms has their own way of displaying screen components. There must be a way to define GUI elements independent of the platform being used such that resizing the window does not create havoc on the screen elements.

The AWT provides a group of classes known as layout managers, or layouts, that handle the layout management. All layouts implement the LayoutManager interface. A layout manager automatically positions components within a container

4.2.2 Types of Layouts

The various layouts available in Java are as follows:

- Flow Layout
- Border Layout
- Card Layout
- Grid Layout
- GridBag Layout

Can a single container have two layouts at the same time?

4.2.3 Applications of Layout Managers

Each different layout manager has its own particular use. A certain layout manager that may suit a certain user interface may not be applicable for some other user interface. If our needs are specific then only a certain layout manager will satisfy those needs.

For instance, in a Java program, if our requirements were to display a few components of same size in rows and columns, only the GridLayout would be suitable. Similarly, if we needed to display a component in maximum possible space, we would have to choose between the BorderLayout and the GridBagLayout since these would be the only two applicable for such a requirement.

4.2.4 How to set layouts?

When a component is first created, it uses its default layout manager. For example, the default layout of an applet is the *'FlowLayout'*. All the components are placed in a container and are arranged according to the associated layout manager. A new layout manager can be set using the method called *'setLayout( )'*.

4.2.5 When does the layout manager come into picture?

Each time, a container needs to change its appearance or whenever a new component is added, the layout manager is consulted. If the size of a component is changed even indirectly, say by changing its font, for example, the component should automatically resize and repaint itself. If that does not happen, we should invoke the *revalidate ( )* method of the component.

Let us now look at the details of each layout.

4.3 The Flow Layout Manager

The '*FlowLayout*' is the default layout manager for applets and panels. The components are arranged serially from the upper left corner to the right bottom corner. When there are a number of components, they are arranged row wise and left to right. The constructors for the *FlowLayout* are as follows:

```
FlowLayout mylayout = new FlowLayout();// constructor
// constructor with alignment specified
FlowLayout exLayout=new FlowLayout(FlowLayout.RIGHT);
setLayout(exlayout); // setting the layout to Flowlayout
```

Controls can be aligned to the left, right and center. To align the controls to the right, the following syntax is used:

```
setLayout(new  FlowLayout(FlowLayout.RIGHT))
```

Example 1 shows how to set the layout in a frame.

Example 1

```
import  java.awt.*;
public  class  FlowDemo  extends  Frame{
Label  lname=new  Label("Name");
TextField  tname=new  TextField(20);
public  FlowDemo(String  as)
{
super(as);
setLayout(new  FlowLayout());
add(lname);
add(tname);
}
public  static  void  main(String  args[])
   {
FlowDemo  ff=new  FlowDemo("FlowLayout  Demo");
ff.setSize(500,500);
ff.setVisible(true);
   }
}
```

Here we create a subclass of Frame called FlowDemo. This class defines two components- a label and a textfield. Since the default layout of the frame is BorderLayout and we wish to have a FlowLayout, we use the *setLayout( )* method to change the layout. The components are then added to the screen of the frame. In the main( ) method, we create an instance of FlowDemo and set its size and visibility respectively.

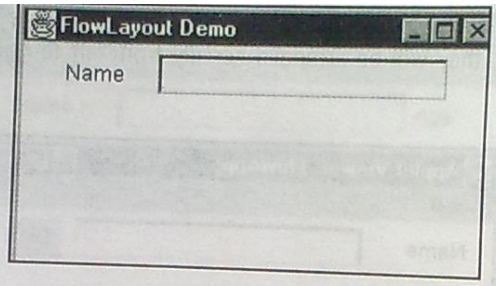


Figure 4.1

The code in example 2 illustrates the same example using an applet in place of a frame. Here the layout need not be set explicitly since it is the default layout for an applet.

Example 2

```
import java.awt.*;
import java.applet.*;
public class FlowApp extends Applet{

   public void init( )
   {
   TextField t1=new TextField(20);
   Label l1=new Label("Name :");
   Button b=new Button("OK");
   add(l1);
   add(t1);
   add(b);
   }
}
/* <applet code=FlowApp width=500 height=500>
</applet>
*/
```

We create a label, textfield and a button and add them to the screen. The default layout being FlowLayout for an applet, they will be arranged serially from left to right. The output will be as illustrated in Fig. 4.2.

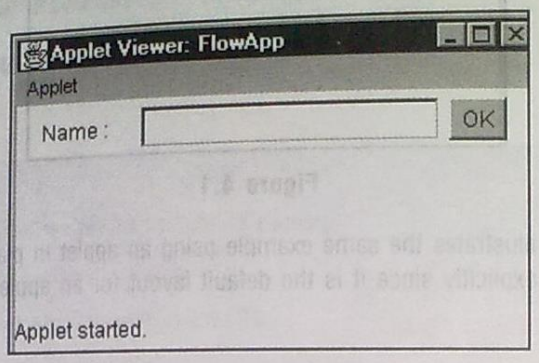


Figure 4.2

? How do you indicate where a component will be positioned using Flowlayout?

1) North, South,East,West
2) Assign a row/column grid reference
3) Pass a X/Y percentage parameter to the add method
4) Do nothing; the FlowLayout will position the component

If there are only a few components on a container, the FlowLayout works perfectly fine. But if there are a large number of components the screen will look disorganized or messy as shown in Figure 4.3. In such cases, it is better to opt for any other layout that can suitable arrange the components.

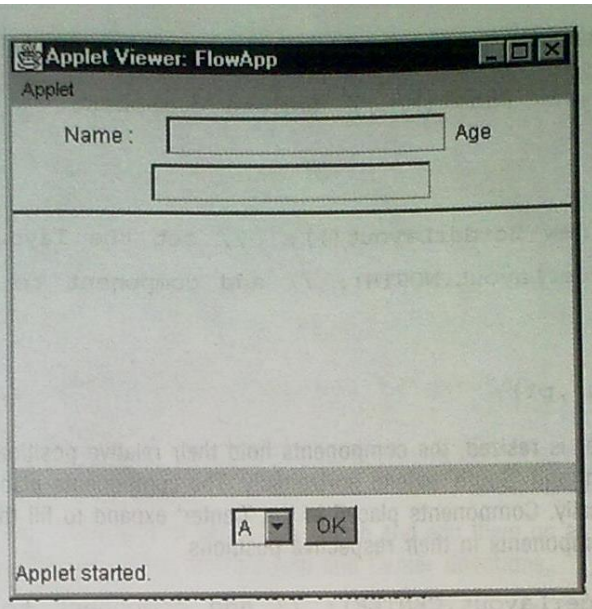


Figure 4.3

4.4 The BorderLayout Manager

The '*BorderLayout*' is the default layout manager for objects of type '*Window*', '*Frame*' and '*Dialog*'. Components can be assigned to North, South, East, West or Center position of the container using this layout.

The BorderLayout class defines several constant values to enable this.

- NORTH – corresponds to the top of the container.
- EAST – corresponds to the right of the container.
- SOUTH – corresponds to the bottom of the container.
- WEST – corresponds to the left of the container.
- CENTER – corresponds to the center of the container.

For instance, in the project, to add a Panel to a Frame we will be using the following code-

```
Panel p1= new Panel(); // declare component
. . .
. . .
setLayout(new BorderLayout());   // set the layout
add(p1,BorderLayout.NORTH); // add component to the layout
```

or

```
add("North",p1);
```

Even if the container is resized, the components hold their relative positions. The components placed in the 'North' and 'South' extend horizontally. The components placed in the 'East' and 'West' extend vertically. Components placed in the 'Center' expand to fill the area that remains after placing the components in their respective positions.

```
add(b2,BorderLayout.CENTER); // add component to the Center
```

Example 3 demonstrates the use of BorderLayout.

Example 3

```
import java.awt.*;
import java.applet.*;
public class BorderApp extends Applet{

  public void init( )
  {
     setLayout(new BorderLayout( ));
  Button b1=new Button("EAST");
Button b2=new Button("WEST");
Button b3=new Button("NORTH");
Button b4=new Button("SOUTH");
Button b5=new Button("CENTER");
```

```
add("East",b1);
add("West",b2);
add("North",b3);
add("South",b4);
add("Center",b5);
   }
}
/* <applet code=BorderApp width=500 height=500>
</applet>
*/
```

Here we create 5 components each of which is a button and add them on an applet. Then we add them respectively to the East, West, North South and Center directions.

The output will be as shown in Fig. 4.4.

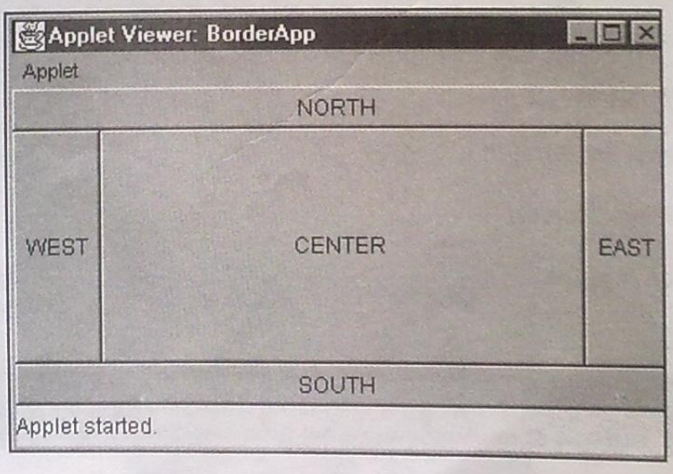


Figure 4.4

To remove any layout settings, set the layout to null. The syntax is: 'setLayout(null)'

?

What happens when you insert two components at the same position in a BorderLayout?

```
add("South",b1);
add("South",b2);
```

The border layout can indirectly contain more than five components. To achieve this, we can use panels of various layouts to contain components and then place those panels in the border layout.

For example, in the Figure 4.5, the panel in the *North* contains a header with a welcome message and a timer.The panel in the center contains the question and the answers. Finally the panel at the bottom consists of three buttons. Note that each of these panels have different layouts. And finally the main frame itself has BorderLayout.

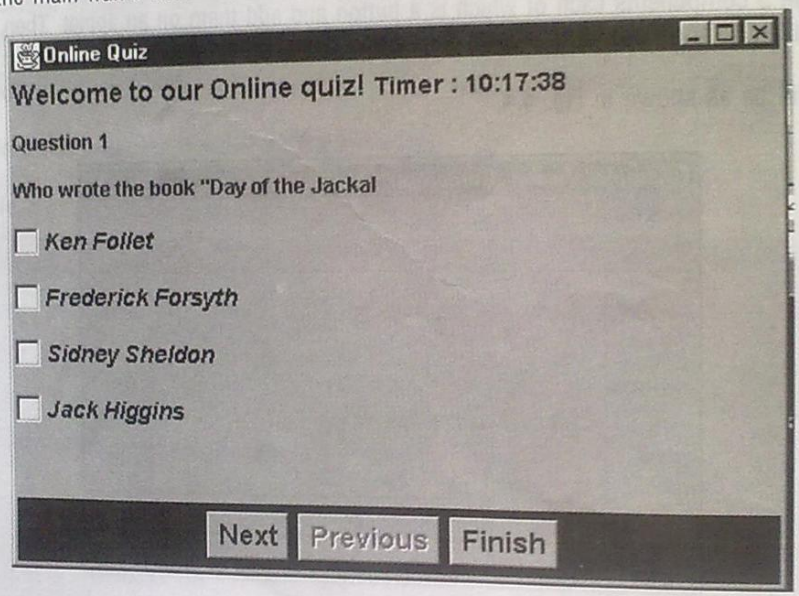


Figure 4.5

4.5 The GridLayout Manager

The '*GridLayout*' layout helps us to divide the container area into a rectangular grid. The components are arranged in rows and columns. This layout is used when all the components are of the same size.

The *GridLayout* constructor is as follows:

```
GridLayout gl=new GridLayout(4,3) ;
```

where 4 represents the number of rows and 3 represents the number of columns.

The following code snippet demonstrates the grid layout

```
. . .
Button btn[ ];
// declare button array
String str[ ]={"1","2","3","4","5","6","7","8","9"};
setLayout(new GridLayout(3,3));
btn=new Button[str.length]; // creates button array
for(int i=0;i<str.length;i++)
{
   btn[i]=new Button(str[i]);
   add(btn[i]);
}
. . .
```

The grid layout will appears as shown in Figure 4.6

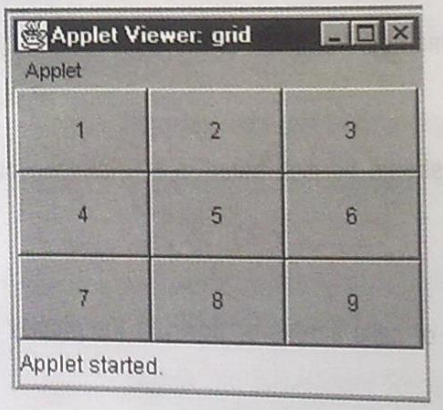


Figure 4.6

In this layout, the components need not be of the same size. It is similar to the *GridLayout* manager since the components are arranged in a grid of rows and columns. However, the order of placing the components is not 'left-to-right' and 'top-to-bottom'.

A container can be given the GridBagLayout using the following syntax.

```
GridBagLayout gb = new GridBagLayout()
ContainerName.setLayout(gb);
```

To use this layout, information must be provided on the size and layout of each component. The class '*GridBagConstraints*' holds all the information required by the class *GridBagLayout* to position and size each component. The *GridBagContraints* class can be thought of as a helper class to *GridBagLayout*.

The steps to use this layout are summarized below-

- Create an instance of GridBagLayout and define it as the current layout manager.
- Create an instance of the helper class GridBagConstraints.
- Set up the constraints for a component.
- Inform (through a method) the layout manager about the component and its constraints.
- Add the componet to the container.
- Repeat these steps for every component that is to be displayed.

The following is a list of member variables of the class *GridBagConstraints*:

- weightx, weighty

  Specifies how space should be distributed in a GridBagLayout. The default values for these variables are 0.

- gridwidth, gridheight

  Specifies the number of cells across or down in the display area of a component.

- ipadx, ipady

  Specifies by what amount to change the minimum height and width of the component. It adds 2*ipadx to the minimum width and 2*ipady to the minimum height of the component. The defaults are 0 for both.

- anchor

  Specifies where to place the components in the cell.The default is to place it in the centre of the cell. The following static data members can be used

  GridBagConstraint.NORTH

  GridBagConstraint..EAST

  GridBagConstraint..WEST

  GridBagConstraint.SOUTH

  GridBagConstraint.NORTHEAST

  GridBagConstraint.SOUTHEAST

  GridBagConstraint.NORTHWEST

  GridBagConstraint.SOUTHWEST

- gridx, gridy

  Specifies in which cell to place the component. It uses 'GridbagConstraints.RELATIVE' option which is the default value to indicate that the component should be placed to thr right of or below the component that was added previous to the this component.

- fill

  Specifies how a component fills a cell if the cell is larger than the component. The default is leave the size of the component unchanged. The following is a list of the *static* data members that are values for the *fill* variable.

GridBagConstraint.NONE	Default, does not change component size.
GridBagConstraint.HORIZONTAL	Increases the width of the component to make it fill the display area horizontally.
GridBagConstraint.VERTICAL	Increases the height of the component to make it fill the display area vertically.
GridBagConstraint.BOTH	Makes the component fill the display area entirely.
Insets	Specifies the top, bottom, left and right gap between the components. Default value is zero.

The constraints for each component can be set using the method 'setConstraints()'. For example:

```
gblay.setConstraints(lbl, gbc);
```

where 'gblay' is an object of the class *GridBagLayout*, lbl is a 'Label' component and 'gbc' is a *GridBagConstraints* object.

The '*GridBagLayout*' is more flexible than any other layout. Components can be placed using this layout more precisely. But it is also the most complex layout manager available.

Example 4 illustrates an example of *GridBagLayout* and *GridBagConstraints.*

Example 4

```
import java.awt.*;
import java.applet.Applet;
public class Mygridbag extends Applet
{
   TextArea ta;
   TextField tf;
   Button b1,b2;
   CheckboxGroup cbg;
   Checkbox cb1,cb2,cb3,cb4;
   GridBagLayout gb;
   GridBagConstraints gbc;
   public void init()
   {
      gb=new GridBagLayout();
      setLayout(gb);
      gbc=new GridBagConstraints();
      ta=new TextArea("Textarea ",5,10);
      tf=new TextField("enter your name");
      b1=new Button("TextArea");
      b2=new Button("TextField");
```

```
        cbg=new  CheckboxGroup();
        cb1=new  Checkbox("Bold",cbg,false);
        cb2=new  Checkbox("Italic",cbg,false);
        cb3=new  Checkbox("Plain",cbg,false);
        cb4=new  Checkbox("Bold/Italic",cbg,true);

        gbc.fill=GridBagConstraints.BOTH;
        addComponent(ta,0,0,4,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(b1,0,1,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(b2,0,2,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb1,2,1,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb2,2,2,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb3,3,1,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(cb4,3,2,1,1);
        gbc.fill=GridBagConstraints.HORIZONTAL;
        addComponent(tf,4,0,1,3);
    }
public void addComponent(Component c, int row, int col, int nrow,
int ncol)
    {
        gbc.gridx=col;
        gbc.gridy=row;
```

```
    gbc.gridwidth=ncol;
    gbc.gridheight=nrow;
    gb.setConstraints(c,gbc);
    add(c);
    }
}
```

When the container is resized and when additional space is available, components with larger weight values occupy more space than the ones with lower weight values.

The output of this program is as shown in Figure 4.7

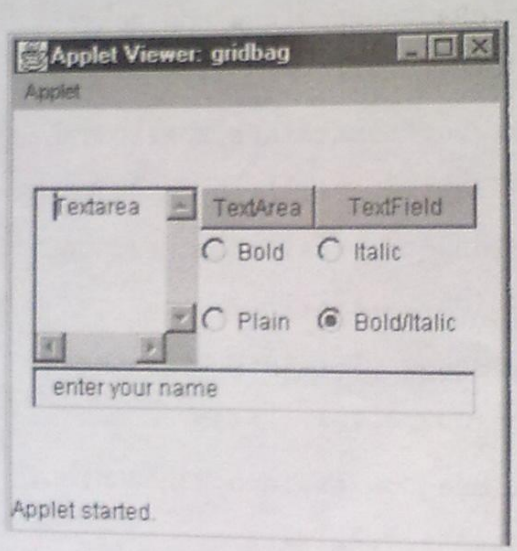


Figure 4.7

The functionality of the code is now explained in detail.

```
gbc.fill=GridBagConstraints.BOTH;
```

The member *fill* of the class *GridBagConstraints* is used to specify that the component can expand in both ways, horizontally and vertically. The following syntax makes the component expand only horizontally as shown below:

```
gbc.fill=GridBagConstraints.HORIZONTAL;
```

The component *TextArea* is added with the number of rows and columns it needs to occupy with the syntax given.

```
addComponent(ta,0,2,4,1);
```

0 – start from the 0th row

2 – start from the 2nd column

4 – ta occupies 4 rows

1 – ta occupies 1 column

To place components in a particular row and column,

```
gbc.gridx=col;
gbc.gridy=row;
```

Where:

gridx is the column where the component is to be placed

gridy is the row where the component is to be placed

To specify the number of columns and rows components should occupy

```
gbc.gridwidth=ncol;
gbc.gridheight=nrow;
```

Where:

gridwidth specifies the number of columns component should occupy

gridheight specifies the number of rows component should occupy

4.7 The CardLayout Manager

The CardLayout can store a stack of several layouts. Each layout is like a card in a deck. The card is usually a Panel object. The card to be displayed on top is controlled by a separate component such as a button.

This layout is used whenever we need number of panels each with a different layout to be displayed one by one. First, the set of required components are positoned on the respective panels. A different layout is set for each panel. For example :

```
panelTwo.setLayout(new GridLayout(2,1));
```

A main panel will contain these panels. The layout of the main panel is set to *CardLayout*. This is done as shown:

```
CardLayout card = new CardLayout();
panelMain.setLayout(card);
```

The next step is to add the other panels to the main panel. The following code snippet shows how.

```
panelMain.add("Red Panel", panelOne);
panelMain.add("Blue Panel", panelTwo);
```

The method 'add()' takes two parameters. The first is a *String* for panel title and the second is the *Panel* object name.

Usually a button is used to control which panel will be displayed. An *actionListener* is added to the button. The *actionPerformed()* methodcan define which card to display. This is decided by the methods given below:

```
card.next(panelMain);
card.previous(panelMain);
card.first(panelMain);
card.last(panelMain);
card.show(panelMain, "Red Panel"); // A particular panel
```

Figures 4.8 and 4.9 show the output when different cards are selected.

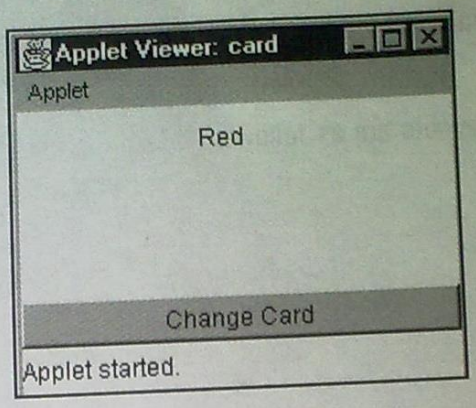


Figure 4.8

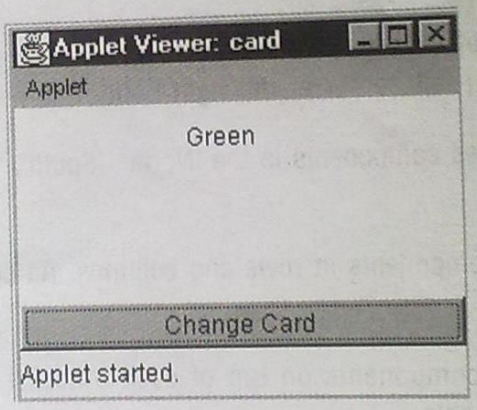


Figure 4.9

The Session in Brief

- The layout manager class provides a means for controlling the physical layout of GUI elements
- The different types of layouts are as follows:
  - FlowLayout
  - BorderLayout
  - CardLayout
  - GridLayout
  - GridBagLayout
- A layout is set with the method 'setLayout()'
- The Flowlayout is the default Layout Manager for applets and panels. The components are arranged from the upper left corner to the right bottom corner.
- The Borderlayout arranges components in the 'North', 'South', 'East', 'West' and 'Center' of a container.
- The Gridlayout places components in rows and columns. All components are of the same size.
- The Cardlayout places components on top of each other. It creates a stack of several components, usually panels.
- The GridBaglayout places components more precisely than any other layout manager.It is similar to the grid layout. The only difference is that here the components need not be of the same size and can be placed in any row or column.

Check Your Progress

1. Which LayoutManager arranges components left to right, then top to bottom, centering each row as it moves to the next?

   Select the one right answer.

   a) BorderLayout
   b) FlowLayout
   c) GridLayout
   d) CardLayout
   e) GridBagLayout

2. A component can be resized horizontally, but not vertically, when it is placed in which region of a BorderLayout?

   Select the one right answer.

   a) North or South
   b) East or West
   c) Center
   d) North, South, or Center
   e) any region

3. When using the GridBagLayout manager, each new component requires a new instance of the GridBagConstraints class. **True/False**

4. What most closely matches the appearance when this code is executed?

```
import java.awt.*;
public class MyClass extends Frame{
    public static void main(String args[]){
        MyClass cl = new MyClass();
    }
    MyClass( ){
```

```
    Panel p = new Panel( );
    p.setBackground(Color.pink);
    p.add(new Button("One"));
    p.add(new Button("Two"));
    p.add(new Button("Three"));
    add(p, BorderLayout.NORTH);
    setLayout(new FlowLayout());
    setSize(300,300);
    setVisible(true);
  }
}
```

a) The buttons will run from left to right along the bottom of the Frame

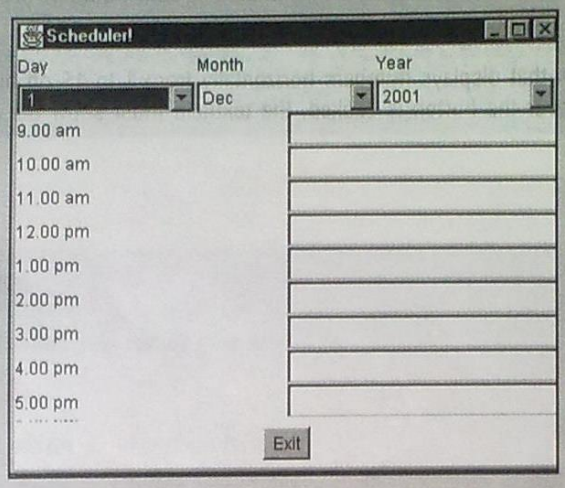
b) The buttons will run from left to right along the top of the frame

c) The buttons will not be displayed

d) Only button three will show occupying all of the frame

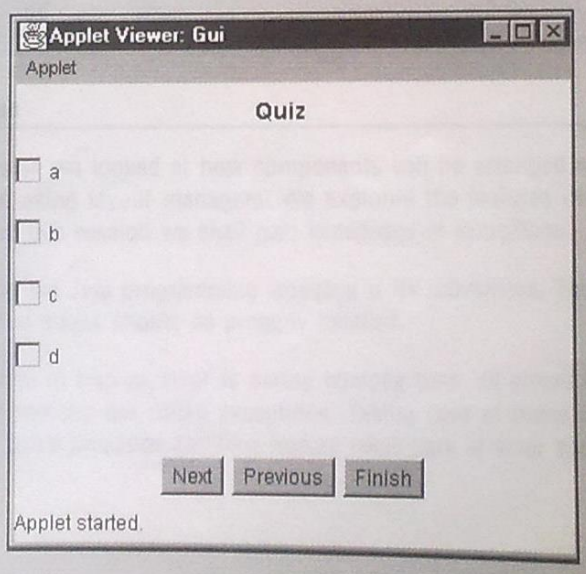
Do It Yourself

1. Write a program to create an applet as follows-



(Hint: The first panel uses GridLayout with 2 rows and 3 columns, the second uses GridLayout with 9 rows and 2 columns and the third panel uses its default layout manager. The three panels are arranged on the frame using BorderLayout.)

2. Write a program that creates the following applet-



(Hint : Use 3 Panels – one for each row, use GridLayout for second Panel and overa applet should have Borderlayout.) The buttons need not perform any action.

3. Write a program to display a calculator-like screen with numbers from 0-9, and specia symbols like '+', '-', etc.

4. Write a program that displays numbers horizontally from 1 to 15 on button and a textfiel below it. Whenever the button is clicked, the textfield must show the number selected.