JAVA NOTES

GRAPHICAL USER INTERFACES

Terry Marris July 2001

9 LAYOUT MANAGERS

9.1 LEARNING OUTCOMES

By the end of this lesson the student should be able to

- use flow, border, grid and grid bag layout managers to achieve a variety of screen formats

9.2 INTRODUCTION

In §3..§8 we looked at GUI components such as text fields and areas, buttons and drop down lists. We see how to arrange these components in a frame.

9.3 FLOW LAYOUT

A flow layout arranges components in a left-to-right flow, much like lines of text in a paragraph. Flow layouts are typically used to arrange buttons in a panel, but can be used for any component such as labels, text boxes and lists.

Components are added in a line. When the line becomes full, a new line is started.

Just like a paragraph, the components can be left aligned, right aligned or centre aligned.

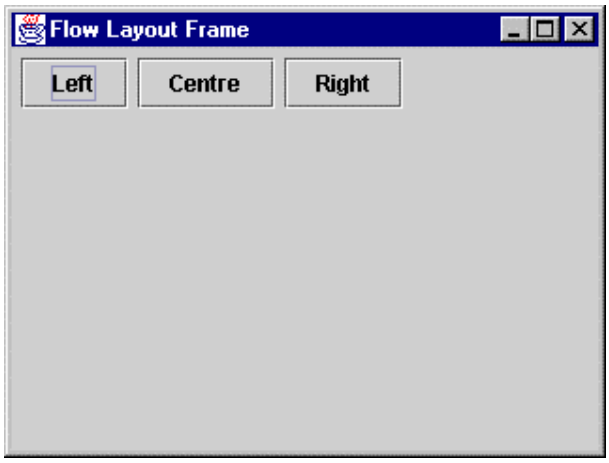


Figure 9.1 *Flow Layout Left Aligned*

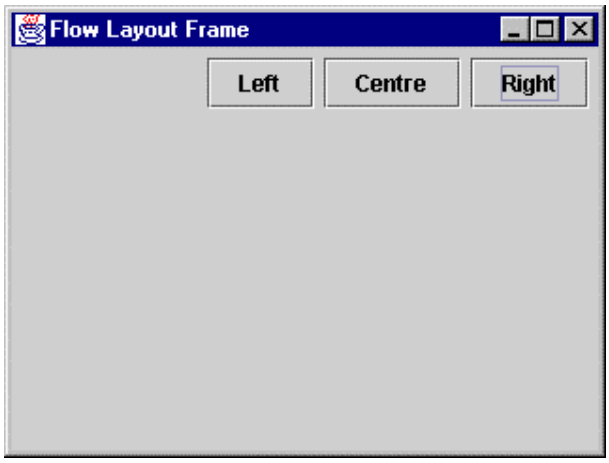


Figure 9.2 *Flow Layout Right Aligned*

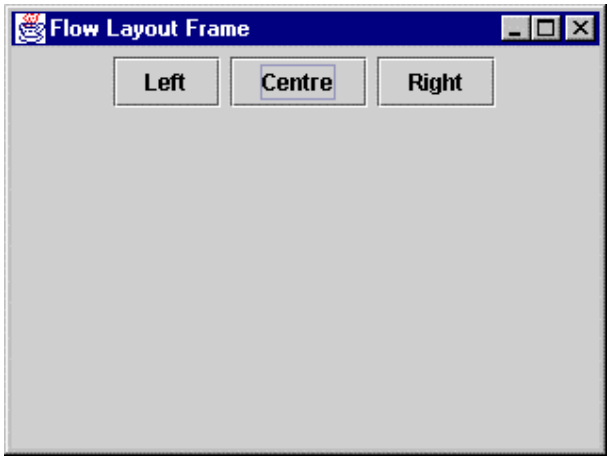


Figure 9.3 *Flow Layout Centre Aligned*

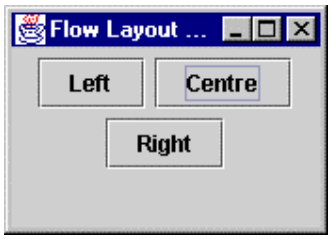


Figure 9.4 *Flow Layout Starts a New Line*

A flow layout lets each component assume its natural (preferred) size.

A flow layout also allows you to set the horizontal and vertical gap between components. By default this is five pixels.

The program shown below demonstrates the principles.

```
/* FlowLayoutFrame.java
   Terry Marris  8 July 2001
*/

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

class FlowLayoutPanel extends JPanel implements ActionListener
{
  private JButton leftButton, centreButton, rightButton;

  public FlowLayoutPanel()
  {
    leftButton = new JButton("Left");
    leftButton.addActionListener(this);
    add(leftButton);

    centreButton = new JButton("Centre");
    centreButton.addActionListener(this);
    add(centreButton);

    rightButton = new JButton("Right");
    rightButton.addActionListener(this);
    add(rightButton);
  }


  public void actionPerformed(ActionEvent e)
  {
    Object source = e.getSource();
    if (source == leftButton)
      setLayout(new FlowLayout(FlowLayout.LEFT));
    else if (source == centreButton)
      setLayout(new FlowLayout(FlowLayout.CENTER));
    else if (source == rightButton)
      setLayout(new FlowLayout(FlowLayout.RIGHT));
    validate();  // recomputes the size and layout of all
  }              // components in this container
}
```

```
public class FlowLayoutFrame extends JFrame {
  public FlowLayoutFrame()
  {
    setTitle("Flow Layout Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    Container contentPane = getContentPane();
    contentPane.add(new FlowLayoutPanel());
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new FlowLayoutFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of program *FlowLayoutFrame.java* shown above.

java.awt.FlowLayout

Fields		
static int	CENTER	indicates that each row of components should be centred.
static int	LEFT	indicates that each row of components should be left justified.
static int	RIGHT	indicates that each row of component should be right justified.

Constructors	
FlowLayout()	initialises a new Flow Layout with a centred alignment and a default five pixel horizontal and vertical gap.
FlowLayout(int align)	initialises a new Flow Layout with the given alignment (FlowLayout.LEFT, CENTER or RIGHT) and a default five pixel gap between components.
FlowLayout(int align, int hgap, int vgap)	initialises a new Flow Layout with the given alignment and horizontal and vertical gap between components.

Methods		
void	setAlignment(int align)	sets the alignment of component sin this flow Layout with the given alignment, which is one of FlowLayout.LEFT, CENTER or RIGHT.
void	setHgap(int hgap)	sets the horizontal gap between components
void	setVGap(int vgap)	sets the vertical gap between components

9.4 BORDER LAYOUT

A border layout places its components in five regions: NORTH, SOUTH, WEST, EAST and CENTER.

The components are laid out according to their natural (preferred) sizes and the size of their container.

The NORTH and SOUTH components are stretched horizontally to fill their regions.

The WEST and EAST components are stretched vertically to fill their regions.

The CENTER component is stretched both vertically and horizontally to fill any space left over.

A region may not contain more than one component.

In Figure 9.5 we show the result of placing a button in each region.

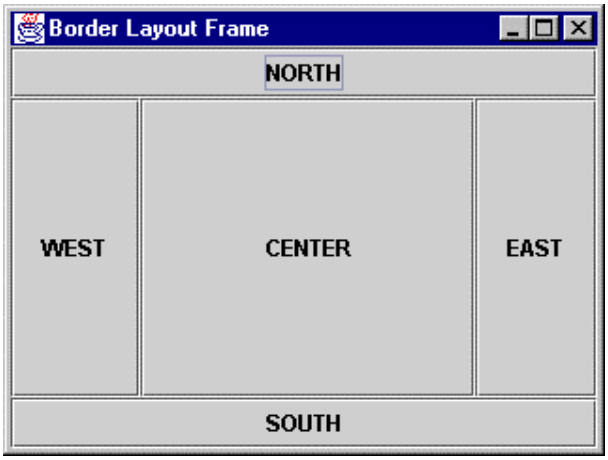


Figure 9.5 *Components are stretched to fill their regions*

We add a component, such as a button, by specifying the region as a parameter in an *add()* message to a container.

```
JButton northButton = new JButton("NORTH");
panel.add(northButton, BorderLayout.NORTH);
```

If the region is not specified, *BorderLayout.CENTER* is assumed.

The program that produced Figure 9.5 is shown below.

```
/* BorderLayoutFrame.java
   Terry Marris  8 July 2001
*/

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

class BorderLayoutPanel extends JPanel {
  private JButton northButton, southButton,
                  westButton, eastButton,
                  centreButton;

  public BorderLayoutPanel()
  {
    setLayout(new BorderLayout());

    northButton = new JButton("NORTH");
    add(northButton, BorderLayout.NORTH);

    southButton = new JButton("SOUTH");
    add(southButton, BorderLayout.SOUTH);

    westButton = new JButton("WEST");
    add(westButton, BorderLayout.WEST);

    eastButton = new JButton("EAST");
    add(eastButton, BorderLayout.EAST);

    centreButton = new JButton("CENTER");
    add(centreButton, BorderLayout.CENTER);
  }
}
```

```
public class BorderLayoutFrame extends JFrame {
  public BorderLayoutFrame()
  {
    setTitle("Border Layout Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);
    setSize(screen.width/2, screen.height/2);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    Container contentPane = getContentPane();
    contentPane.add(new BorderLayoutPanel());
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new BorderLayoutFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of program *BorderLayoutFrame.java* shown above.

java.awt.BorderLayout

Fields		
static String	CENTER	middle of container
static String	EAST	right side of container
static String	NORTH	top of container
static String	SOUTH	bottom of container
static String	WEST	left side of container

Constructors	
BorderLayout()	initialises a new border layout with no gaps between components.
BorderLayout(int hgap, int vgap)	initialises a new border layout with the given horizontal and vertical gaps between components.

9.5 GRID LAYOUT

The grid layout arranges all components in rows and columns, just like a spreadsheet. But, unlike a spreadsheet, the cells are always the same size.

The size of a cell is automatically calculated from the components it contains.

In the constructor of the grid layout object you specify how many rows and columns you need.

```
panel.setLayout(new GridLayout(5, 4);
```

specifies a grid with five rows and four columns.

You can specify the horizontal and vertical gaps you want.

```
panel.setLayout(new GridLayout(5, 4, 3, 2);
```

specifies a horizontal gap between components of three pixels and a vertical gap of two pixels.

You add components one by one along a row. When a row is filled, the next row is started. Each *add()* message places the component in the next cell in sequence.

Small grids are useful for organising parts of a window. If you want to have a row of buttons of identical size, place them inside a panel governed by grid layout with a single row. You will need to set the gap size so that the buttons have some space between them.

java.awt.GridLayout

Constructors	
GridLayout(int rows, int cols)	initialises a new GridLayout object with the given number of rows and columns.
GridLayout(int rows, int cols, int hgap, int vgap)	initialises a new GridLayout object with the given number of rows and columns, with the given horizontal and vertical gaps (in pixels) between components.

Figure 9.6 *Check Boxes in a grid of one column*

Here is the coding.

```
/* GridFrame.java
   Terry Marris  11 July 2001
*/

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

class TextPanel extends JPanel {
  private JTextField exampleField;

  public TextPanel()
  {
    exampleField = new JTextField(
           "The quick brown fox jumps over the lazy dog", 30);
    exampleField.setFont(new Font(
                         "SansSerif", Font.PLAIN, 12));
    exampleField.setEditable(false);
    exampleField.setBackground(Color.white);
    add(exampleField);
  }


  public void updateText(int style, int size)
  {
    exampleField.setFont(new Font("SansSerif", style, size));
    repaint();
  }
}
```

```
class GridPanel extends JPanel {
  private JCheckBox boldCheckBox, italicCheckBox,
                    largeCheckBox;

  public GridPanel()
  {
    setLayout(new GridLayout(3, 1, 2, 1));

    boldCheckBox = new JCheckBox("Bold");
    // We want the label to precede its check box
    boldCheckBox.setHorizontalTextPosition(
                              SwingConstants.LEFT);
    // We want each check box right aligned in its column
    boldCheckBox.setHorizontalAlignment(SwingConstants.RIGHT);
    add(boldCheckBox);

    italicCheckBox = new JCheckBox("Italic");
    italicCheckBox.setHorizontalTextPosition(
                               SwingConstants.LEFT);
    italicCheckBox.setHorizontalAlignment(
                               SwingConstants.RIGHT);
    add(italicCheckBox);

    largeCheckBox = new JCheckBox("Large");
    largeCheckBox.setHorizontalTextPosition(
                              SwingConstants.LEFT);
    largeCheckBox.setHorizontalAlignment(
                              SwingConstants.RIGHT);
    add(largeCheckBox);
  }


  public JCheckBox getBoldCheckBox()
  {
    return boldCheckBox;
  }


  public JCheckBox getItalicCheckBox()
  {
    return italicCheckBox;
  }


  public JCheckBox getLargeCheckBox()
  {
    return largeCheckBox;
  }
}
```

```
class FontDialogPanel extends JPanel implements ActionListener
{
  private GridPanel gridPanel;
  private TextPanel textPanel;

  public FontDialogPanel()
  {
    setLayout(new BorderLayout());

    gridPanel = new GridPanel();
    add(gridPanel, BorderLayout.WEST);

    gridPanel.getBoldCheckBox().addActionListener(this);
    gridPanel.getItalicCheckBox().addActionListener(this);
    gridPanel.getLargeCheckBox().addActionListener(this);

    textPanel = new TextPanel();
    add(textPanel, BorderLayout.SOUTH);
  }


  public void updateFont()
  {
    int style = 0;

    if (gridPanel.getBoldCheckBox().isSelected())
      style = Font.BOLD;

    if (gridPanel.getItalicCheckBox().isSelected())
      style += Font.ITALIC;

    int size = 12;
    if (gridPanel.getLargeCheckBox().isSelected())
      size = 14;

    textPanel.updateText(style, size);
  }


  public void actionPerformed(ActionEvent e)
  {
    updateFont();
  }
}
```

```
public class GridFrame extends JFrame {
  public GridFrame()
  {
    setTitle("Grid Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    Container contentPane = getContentPane();
    contentPane.add(new FontDialogPanel());
    pack();
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new GridFrame();
    aFrame.show();
  }
}
```

Exercise: Explain each line of the program *GridFrame.java* shown above.

9.6 GRID BAG LAYOUT

A grid bag layout has a rectangular grid of cells. Its components (buttons, lists, text, etc) are not all the same size. A component can occupy one or more cells.

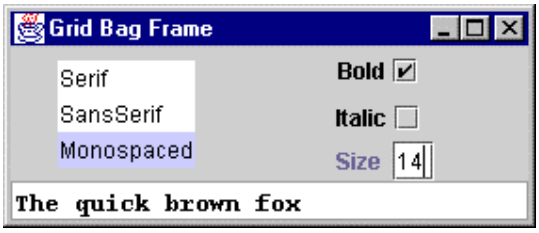


Figure 9.7 *Grid Bag Layout*

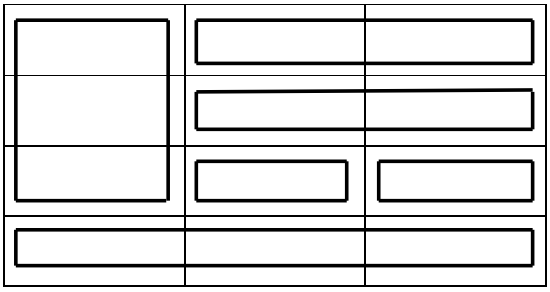


Figure 9.8 *Grid Bag Design*

We start with a grid comprising four rows and three columns. A single cell is just large enough for the smallest component. We lay out the components on the grid. The list box spans three rows. Each check box spreads over two columns. The label and size fields each occupy just one cell. The sample text is spread over three columns.

Having a clear idea of the required layout, we go on to implement it.

We

- create a *GridBagLayout* object. The number or rows and columns of the underlying grid are automatically calculated from the information we give later
- set this *GridBagLayout* object to the layout manager for this panel
- create a *GridBagConstraints* object. The *GridBagConstraints* object specifies how components are set out in the grid
- for each component we set the grid bag constraints that specify where it is to be placed on the grid, how many rows and columns it occupies, and whether it is to be resized to fully occupy its display area

We look at the *GridBagConstraints* class in more detail.

java.awt.GridBagConstraints

Fields		
int	anchor	used when the component is smaller than its display area. Anchors the component to a region of its display area. Used with GridBagConstraints constants NORTH, CENTER, EAST, etc. Default is GridBagConstraints.CENTER.
static int	BOTH	resize the component both horizontally and vertically to fill its display area.
static int	CENTER	places the component in the centre of its display area.
static int	EAST	places the component on the right side of its display area.
int	fill	used when the component is smaller than its display area. Determines whether to resize the component, and, if so, how. Used with GridBagConstraints constants NONE, HORIZONTALLY, VERTICALLY and BOTH. Default is GridBagConstraints.NONE.
int	gridheight	specifies the number of cells in a column for the component's display area.
int	gridwidth	specifies the number of cells in a row for the components display area.
int	gridx	specifies the column where the display area starts. The first column is numbered zero.
int	gridy	specifies the row where the display area starts. The first row is numbered zero.
static int	HORIZONTAL	resize the component horizontally but not vertically.
static int	NONE	do not resize this component.
static int	NORTH	put the component at the top of its display area.
static int	NORTHEAST	put the component at the top right of its display area.
static int	NORTHWEST	put the component at the top left of its display area.
static int	SOUTH	put the component at the bottom of its display area.
static int	SOUTHEAST	put the component at the bottom right of its display area.
static int	SOUTHWEST	put the component at the bottom left of its display area.
static int	VERTICAL	resize the component vertically but not horizontally.
int	weightx	specifies how to distribute extra horizontal space. Perhaps best set at 100 to begin with.
int	weighty	specifies how to distribute extra vertical space. Perhaps best set at 100 to begin with.
static int	WEST	put the component at the left of its display area.

Perhaps the easiest way to determine how to set the *GridBagConstants* is to list all the constants and components in a table. Then for each component decide what the constants should be.

	fontList	bold Check Box	italic Check Box	sizeLabel	sizeField	example Text
anchor	✓	✓	✓	✓	✓	✓
BOTH						
CENTER	✓	✓	✓			✓
EAST				✓		
fill	✓	✓	✓	✓	✓	✓
gridheight	3	1	1	1	1	1
gridwidth	1	2	2	1	1	3
gridx	0	1	1	1	2	0
gridy	0	0	1	2	2	3
HORIZONTAL						✓
NONE	✓	✓	✓	✓	✓	
NORTH						
NORTHEAST						
NORTHWEST						
SOUTH						
SOUTHEAST						
SOUTHWEST						
VERTICAL						
weightx	100	100	100	100	100	100
weighty	100	100	100	100	100	100
WEST					✓	

So, for example, the *fontList* is going to be anchored in the centre of its display area, it will be spread over three rows and one column, it will not be resized to fill its entire display area and its top left hand corner will start in row 0, column 0.

The entire program is shown below. A helper method, *add(Component, GridBagConstraints, int x, int y, int w, int h)* factors out the repetitive code.

```
/* GridBagFrame.java
   Terry Marris  10 July 2001
*/

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

import javax.swing.*;
import javax.swing.event.*;

class GridBagPanel extends JPanel
                   implements ActionListener,
                              ListSelectionListener {
  private JList fontList;
  private JCheckBox boldCheckBox, italicCheckBox;
  private JTextField sizeField, exampleField;


  public void add(Component c, GridBagConstraints gbc,
              int x, int y, int w, int h)
  {
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = w;
    gbc.gridheight = h;
    add(c, gbc);
  }
```

```
public GridBagPanel()
{
  String fontStringList[] = {
                   "Serif", "SansSerif", "Monospaced" };
  fontList = new JList(fontStringList);
  fontList.setSelectedIndex(0);
  fontList.addListSelectionListener(this);

  boldCheckBox = new JCheckBox("Bold");
  boldCheckBox.setHorizontalTextPosition(
                             SwingConstants.LEFT);
  boldCheckBox.addActionListener(this);

  italicCheckBox = new JCheckBox("Italic");
  italicCheckBox.setHorizontalTextPosition(
                               SwingConstants.LEFT);
  italicCheckBox.addActionListener(this);

  JLabel sizeLabel = new JLabel("Size  ");
  sizeField = new JTextField("12", 2);
  sizeField.addActionListener(this);

  exampleField = new JTextField(25);
  exampleField.setEditable(false);
  exampleField.setBackground(Color.white);
  exampleField.setText("The quick brown fox");

  setLayout(new GridBagLayout());

  GridBagConstraints gbc = new GridBagConstraints();
  gbc.fill = GridBagConstraints.NONE;

  gbc.anchor = GridBagConstraints.CENTER;
  gbc.weightx = 100;
  gbc.weighty = 100;

  add(fontList, gbc, 0, 0, 1, 3);
                    // gridx, gridy, gridwidth, gridheight
  add(boldCheckBox, gbc, 1, 0, 2, 1);
  add(italicCheckBox, gbc, 1, 1, 2, 1);

  gbc.anchor = GridBagConstraints.EAST;
  add(sizeLabel, gbc, 1, 2, 1, 1);

  gbc.anchor = GridBagConstraints.WEST;
  add(sizeField, gbc, 2, 2, 1, 1);

  gbc.anchor = GridBagConstraints.CENTER;
  gbc.fill = GridBagConstraints.HORIZONTAL;
  add(exampleField, gbc, 0, 3, 3, 1);
}
```

```
  public void updateFont()
  {
    int style = 0;

    if (boldCheckBox.isSelected())
      style = Font.BOLD;

    if (italicCheckBox.isSelected())
      style += Font.ITALIC;

    int size = Integer.parseInt(sizeField.getText());

    String fontName = (String)fontList.getSelectedValue();

    exampleField.setFont(new Font(fontName, style, size));
    repaint();
  }


  public void actionPerformed(ActionEvent e)
  {
    updateFont();
  }


  public void valueChanged(ListSelectionEvent e)
  {
    updateFont();
  }
}
```

```
public class GridBagFrame extends JFrame {
  public GridBagFrame()
  {
    setTitle("Grid Bag Frame");

    Toolkit tk = Toolkit.getDefaultToolkit();
    Dimension screen = tk.getScreenSize();
    setLocation(screen.width/4, screen.height/4);

    addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          System.exit(0);
        }
      });

    Container contentPane = getContentPane();
    contentPane.add(new GridBagPanel());

    pack();
  }


  public static void main(String[] s)
  {
    JFrame aFrame = new GridBagFrame();
    aFrame.show();
  }
}
```

9.6 FURTHER READING

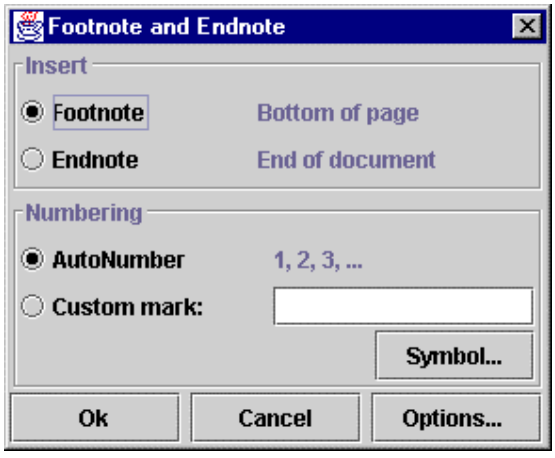
ARNOW & WEISS *Introduction to Programming Using Java* pp 132
HORSTMANN & CORNELL *Core Java 2 Volume 1* pp 380
www.java.sun.com/docs/books/tutorial

In the next lesson we see how to construct menus.

9.7 REVIEW

9.8 EXERCISES

The frame shown below is inspired by Microsoft Word Insert Footnote frame.



Write and test a Java frame that implements this layout. The user should not be able to re-size the frame. No action should be attached to the buttons; they are there just for show.