CN208 Introduction to Computer Programming

Lecture #13
Graphical User Interface (GUI)

Pimarn Apipattanamontre
Email: ***pimarn@pimarn.com***

Frame Windows

- Every graphical program requires one or more frame windows.
- A frame window is designed as an area to put graphical user interface components.
- To show a frame, we use the *JFrame* class in the *javax.swing* package.
- To create a frame window object, use

```
JFrame f = new Jframe( );
```

  Constructs a new frame with a size of 0 x 0 pixels.

- To set the frame with a specific size (*width* x *height* pixels), use

```
f.setSize(int width, int height);
```

- To set the title of the frame, use

```
f.setTitle(String title);
```

- To show the frame, use

```
f.show();
```

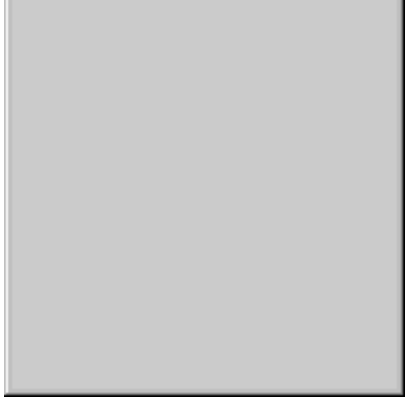
- To set the frame to be the size that can fit all graphical interface components, use

```
f.pack();
```

Empty Frame

File FrameTest1.java

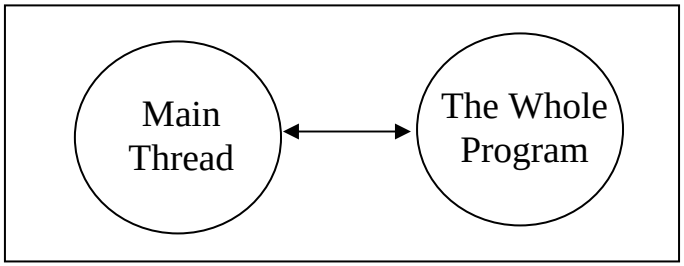
```
import javax.swing.JFrame;

public class FrameTest1
{  public static void main(String[] args)
   {  JFrame f = new JFrame();
      f.setSize(200, 300);
      f.setTitle("Empty Frame");
      f.show();
   }
}
```



- Unlike applets, graphical applications have a *main( )* method.
- The *main( )* method of the *FrameTest1.java* program constructs a *JFrame* object, sets up the title of the object and calls the *show( )* method to pop up an empty frame.
- Although the *main( )* method is finished, the user interface component (the frame window) is still running. Thus, the program is still running.
- Although we close the frame window, the program is still running.
- We can terminate (kill) the program by typing Ctrl+C in the console window.

Single-Threaded Applications

- There is an important difference between console programs and graphical programs.
- In console programs:
  - There is only one unit of virtual processor (called a *thread*).
  - We can view a thread as a processor that has an environment set up appropriately to run a particular part of the program.
  - When the console program is executed, a so-called *main thread* is set up and starts to execute the program sequentially from beginning to end.
  - When the program is completely executed, the *main thread* is finished.



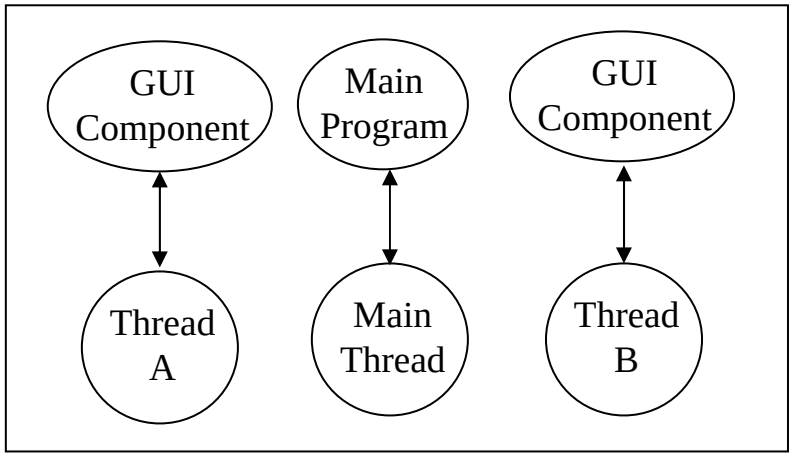
Single-Threaded Applications (Console Programs)

Multi-Threaded Applications

- In graphical programs:
  - There can be more than one thread while the program is being executed
  - The main thread executes instructions at the main program (do the main processing part).
  - Whenever a user interface component (for example, a frame window object) is created, there will be a corresponding thread constructed and set up for taking care of that frame window.
  - At a particular time, there can be many threads executing parts of the program *concurrently*.
  - When the main program is finished, the main thread is completed.
  - The program does not terminate since there are still other threads running.
  - To terminate the program (kill all the threads), use

```
System.exit(0)
```

Multi-Threaded Applications



Multi-Threaded Applications (Graphical Programs)

GUI = Graphical User Interface

WindowListener Interface

- To listen frame-window events, we must implement the *WindowListener* interface or define the subclass of the *WindowAdapter* class.

File WindowListener.java

```
public interface WindowListener
{
   void windowOpened(WindowEvent event);
   // Called when opening the window at the first time.
   void windowClosed(WindowEvent event);
   // Called when the window is closed by the dispose( ) method.
   void windowActivated(WindowEvent event);
   // Called when a user clicks inside the window.
   void windowDeactivated(WindowEvent event);
   // Called when a user clicks another window.
   void windowIconified(WindowEvent event);
   // Called when a user clicks on the “minimize” icon in the title bar
   // of the window.
   void windowDeIconified(WindowEvent event);
   // Called when a user clicks on the icon while the window is
      minimized.
   void windowClosing(WindowEvent event);
   // Called when a user clicks on the “close” icon in the title bar of the
   // window.
}
```

File WindowAdapter.java

```
public class WindowAdapter implements WindowListener
{
   void windowOpened(WindowEvent event) { }
   void windowClosed(WindowEvent event) { }
   void windowActivated(WindowEvent event) { }
   void windowDeactivated(WindowEvent event) { }
   void windowIconified(WindowEvent event) { }
   void windowDeIconified(WindowEvent event) { }
   void windowClosing(WindowEvent event) { }
}
```

Empty Frame

- When a user clicks on the “close” icon in the title bar of a frame window, we would like to close the frame and release all system resources reserved for that particular frame window.
- Also, when all the frame windows are closed, we would like to terminate the program.

File EmptyFrame.java

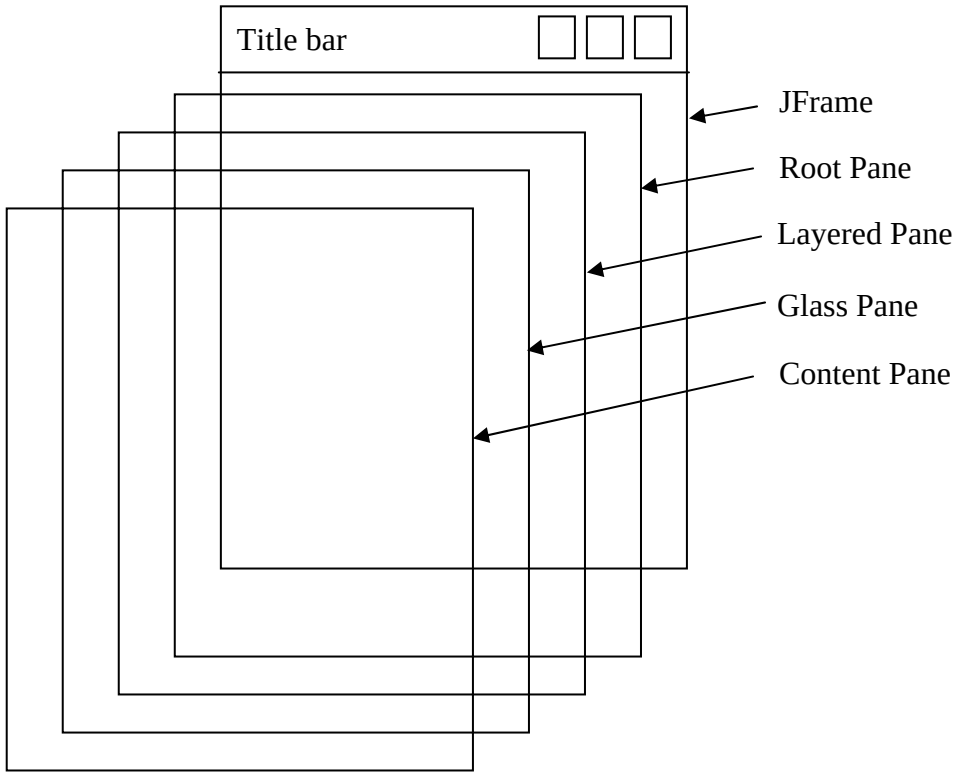
```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class EmptyFrame extends JFrame
{  public EmptyFrame( )
   {  final int FRAME_WIDTH = 300;
      final int FRAME_HEIGHT = 300;
      setSize(FRAME_WIDTH, FRAME_HEIGHT);

      WindowCloser listener = new WindowCloser();
      addWindowListener(listener);
   }

   private class WindowCloser extends WindowAdapter
   { public void windowClosing(WindowEvent event)
     {  dispose(); // release all system resources of the window object.
     }
   }

}
```

File FrameTest2.java

```
public class FrameTest2
{  public static void main(String[] args)
   {  EmptyFrame2 f1 = new EmptyFrame2();
      f1.setTitle("Empty Frame: One");
      f1.show();

      EmptyFrame2 f2 = new EmptyFrame2();
      f2.setTitle("Empty Frame: Two");
      f2.show();
  }
}
```

The Structure of the JFrame Object

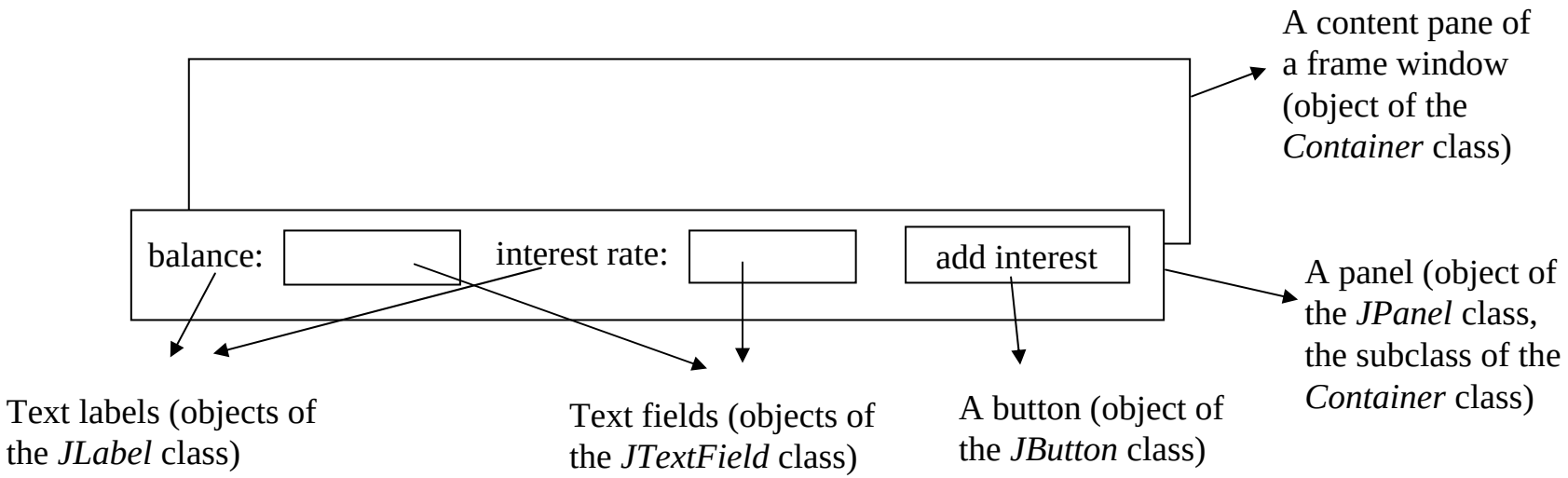
- The structure of a JFrame object consists of four panes: the root pane, the layered pane, the glass pane and content pane.



  - The *root pane* holds the layered pane and the glass pane.
  - The *layered pane* holds the menu bar and the content pane.
  - The *glass pane* captures mouse events in the frame window.
  - The *content pane* holds user interface components we want to add to the frame window.

Container Class

- We put graphical user interface components such as text labels, text fields, buttons into an object of the *Container* class.
- *Container* objects can also be placed into larger a *Container* object.
- Thus, an object of the *Container* class (or its subclass) are where we can put graphical user interface components or *Container* objects into.



Container Class

- There are two types of the Containers:

  (1) A content pane of a frame window (object of the *Container* class).

  - To get the content pane of a frame window, we use the *getContentPane( )* method of the *JFrame* class.

```
JFrame frame = new JFrame();
Container contentPane = frame.getContentPane();
```

  (2) A panel object created from the *JPanel* class (*JPanel* is a subclass of the *Container* class).

  - To create a *JPanel* object, we use:

```
JPanel panel = new JPanel( );
```

- We can add graphical user interface components or *Container* objects into a *Container* object by using *add( )* method but we need to know the concepts of layouts, which control the arrangement of its components.

JTextField Class

- Use *JTextField* components to provide space for the user input
- When we construct an object of the *JTextField* class, we need to specify the width (the number of characters we expect a user to type):

```
JTextField balanceField = new JTextField(5);
```

  The user can type additional characters but a part of the contents of the text field becomes invisible.
- To get the text contained in the text field, use

```
String balanceStr = balanceField.getText();
```

- To set the text field with the specified text, use

```
balanceField.setText("10000");
```

- To indicate whether or not the text field should be editable, use

```
balanceField.setEditable(false); // balanceField will be for display only
```

JLabel and Jbutton Classes

- We usually place a *JLabel* component next to each text field (for field description).
- To construct an object of the *JLabel* class, we supply a label string:

```
JLabel balanceLabel = new JLabel("balance: ");
JLabel rateLabel = new JLabel("interest rate: ");
```

- We use a *JButton* component for a button.
- To construct an object of the *JButton* class, we supply a label string, an icon, or both:

```
JButton addInterestRateButton = new JButton("add interest");
JButton moveButton = new JButton(new ImageIcon("hand.gif"));
JButton moveButton = new JButton("move", new ImageIcon("hand.gif"));
```

- To get the label of the button, use:

```
String buttonLabel = addInterestRateButton.getText();
```

JPanel Class

- We use a *JPanel* component to collect user interface components together.

```
JLabel balanceLabel = new Jlabel(“balance: ”);
JTextField balanceField = new JTextField(5);
JLabel rateLabel = new Jlabel(“interest rate: ”);
JTextField rateField = new JTextField(5);
JButton addInterestRateButton = new JButton(“add interest”);

JPanel panel = new JPanel();
panel.add(balanceLabel); panel.add(balanceField);
panel.add(rateLabel); panel.add(rateField);
panel.add(addInterestRateButton);
```

Layout Types

- Each container has a layout manager that controls the arrangement of its components.
- There are three types of layouts:
  - Border layout
  - Flow layout
  - Grid layout
- By default, each container object has its own layout type when it is constructed.

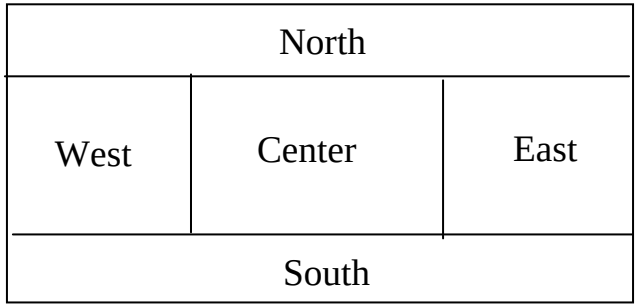
  For example: Every object of the *JPanel* class uses the flow layout by default.

  We can change the layout type by using the *setLayout( )* method.

```
JPanel panel = new JPanel(); // uses flow layout by default
panel.setLayout(new BorderLayout());
```

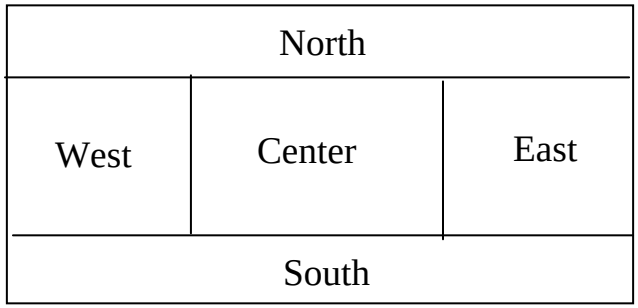
Border Layout

- Border layout sets up five areas in which graphical user interface components can be placed.



- We can only put one component in each area and the border layout expands that component to fill all available space in its area.
- To add two or more components in an area, we need to first place them in a *JPanel* object and then place the *JPanel* object into that area.
- The content pane of the *JFrame* object has a border layout by default.

Border Layout



- When we add a component into a *Container* object using the border layout, we specify the area using predefined constants: BorderLayout.NORTH, BorderLayout.WEST, BorderLayout.EAST, BorderLayout.CENTER, BorderLayout.SOUTH.

```
JFrame frame = new JFrame();
Container contentPane = frame.getContentPane();
JButton aButton = new JButton("add interest");
contentPane.add(aButton, BorderLayout.SOUTH);
```

Border layout expands a component (in this case, a button) to fill all available space in the area that the components are placed.

Flow Layout

- Flow layout arranges its component from left to right and starts a new row when there is no more room in the current row.

- Flow layout leaves each graphical user interface component at its natural (preferred) size.
- *JPanel* objects use the flow layout by default.
- Within each row, the components are centered by default.
- We can also specify other alignments (left-justified or right-justified) when we create flow layout object.

```
JPanel panel = JPanel();
panel.setLayout(new FlowLayout(Flowlayout.RIGHT));//set alignment to be right-justified
```

Grid Layout & Change Default Layout

- Grid layout arranges graphical user interface components in a grid with a fixed number of rows and columns, resizing each component so that they all have the same size.

- We can change the default layout type of a *Container* object by using the *setLayout( )* method.

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout()); // change from flow layout to border layout
panel.setLayout(new GridLayout(3, 3)); // change to 3x3 grid layout
```

- What if we set

```
panel.setLayout(null); // We do not want to use any layout type with this panel.
```

ActionListener Interface

- When a user takes actions on some user interface components, action events are generated.
- The possible actions that the user takes could be:
  - Hitting the Enter key inside a text field.
  - Selecting a particular menu item from a menu.
  - Clicking a button.
- To listen these action events, we have to implement the *ActionListener* interface.
- The *ActionListener* interface has only the *actionPerformed( )* method so there is no need for *ActionAdapter* class.
- After an action listener is installed, when an action event occurs, the *actionPerformed( )* method is called.

File ActionListener.java

```
public interface ActionListener
{
   void actionPerformed(ActionEvent event);
   // Called when a user takes an action on the user interface component.
}
```

Example of Event Handling Problem

File FindNewBalanceFrame.java

```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class FindNewBalanceFrame extends JFrame
{  public FindNewBalanceFrame( )
   {  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JLabel balanceLabel = new JLabel("balance: ");
      balanceTextField = new JTextField(10);
      JLabel intRateLabel = new JLabel("interest rate: ");
      intRateTextField = new JTextField(10);
      JButton addIntRateButton = new JButton("add interest");
      ButtonClickListener buttonListener = new ButtonClickListener( );
      addIntRateButton.addActionListener(buttonListener);
      controlPanel = new JPanel();
      controlPanel.add(balanceLabel); controlPanel.add(balanceTextField);
      controlPanel.add(intRateLabel);  controlPanel.add(intRateTextField);
      controlPanel.add(addIntRateButton);
      Container contentPane = getContentPane( );
      contentPane.add(controlPanel, BorderLayout.CENTER);
      pack( );
   }
  JTextField balanceTextField;
  JTextField intRateTextField;
  JPanel controlPanel;
  private class ButtonClickListener implements ActionListener
  {   public void actionPerformed(ActionEvent event)
      {  double balance = Double.parseDouble(balanceTextField.getText( ));
         double intRate = Double.parseDouble(intRateTextField.getText( ));
         balance = balance + (balance * intRate) / 100;
         String balanceString = Double.toString(balance);
         balanceTextField.setText(balanceString);

      }
   }
}
```

call *System.exit(0)* when the frame is closed.

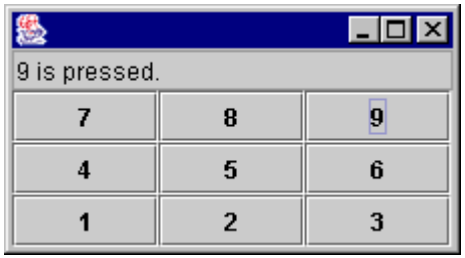
File FindNewBalanceTest.java

```
public class FindNewBalanceTest
{ public static void main(String[ ] args)
   {  FindNewBalanceFrame frame = new FindNewBalanceFrame( );
      frame.show( );
   }
}
```

Example of Event Handling Problem

File ButtonFrame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class ButtonFrame extends JFrame
{ public ButtonFrame( )
  { setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    String[] numberButton = { "7", "8", "9", "4", "5", "6", "1", "2", "3"};

    JPanel numberPanel = new JPanel( );
    numberPanel.setLayout(new GridLayout(3,3));

    for (int i=0; i<numberButton.length; i++)
    { JButton aButton = new JButton(numberButton[i]);
      ButtonClickListener buttonListener = new ButtonClickListener( );
      aButton.addActionListener(buttonListener);
      numberPanel.add(aButton);
    }
    shownText = new JTextField(20);
    shownText.setEditable(false);

    Container contentPane = getContentPane( );
    contentPane.add(shownText, BorderLayout.NORTH);
    contentPane.add(numberPanel, BorderLayout.CENTER);
    pack( );
  }
  JTextField shownText;
  private class ButtonClickListener implements ActionListener
    {  public void actionPerformed(ActionEvent event)
       {  JButton ClickedButton = (JButton) event.getSource( );
          shownText.setText(ClickedButton.getText( ) + " is pressed.");
       }
    }
}
```



File ButtonFrameTest.java

```
public class ButtonFrameTest
{
  public static void main(String[] args)
  {
    ButtonFrame frame = new ButtonFrame( );
    frame.show( );
  }
}
```

Use the *getSource( )* method of the *ActionEvent* Class to get a reference of the *Object* class indicating the event source.

JTextArea Class

- We use the *JTextField* class for a single line of text.
- To display multiple lines of text, we use the *JTextArea* class.
- Since both *JTextField* and *JTextArea* classes are subclasses of the *JTextComponent* class, they have some common methods inherited from the *JTextComponent* class.
- When we construct an object of the *JTextArea* class, we need to specify the number of rows and columns of the area:

```
JTextArea aTextArea = new JTextArea(10, 20);
```

- To get all the texts contained in the text area, use

```
String allTexts = aTextArea.getText();
```

- To set the text area with the specified string *s*, use

```
aTextArea.setText(s);
```

- To disallow the user to edit the contents of the text area, use

```
aTextArea.setEditable(false); // aTextArea will be used for display only.
```

JTextArea Class

- We can set the font of a text component (an object of the JTextArea or JTextField classes) by first constructing a font object of the *Font* class (located in *java.awt* package):

```
Font myFont = new Font(String fontFacename,int fontStyle,int fontSize);
```

  - *fontFacename* can be one of "Serif", "SanSerif", "Monospaced", "Dialog" or "DialogInput".
  - *fontStyle* can be one of Font.PLAIN, Font.ITALIC, Font.BOLD, or (Font.ITALIC + Font.BOLD)
  - *fontSize* is the size of the font in points (72 points = 1 inch).
- Then we can use the *setFont( )* method as follows:

```
aTextArea.setFont(myFont);
```

JScrollPane Class

- When we construct an object of the *JTextArea* class,

```
JTextArea aTextArea = new JTextArea(10, 20);
```

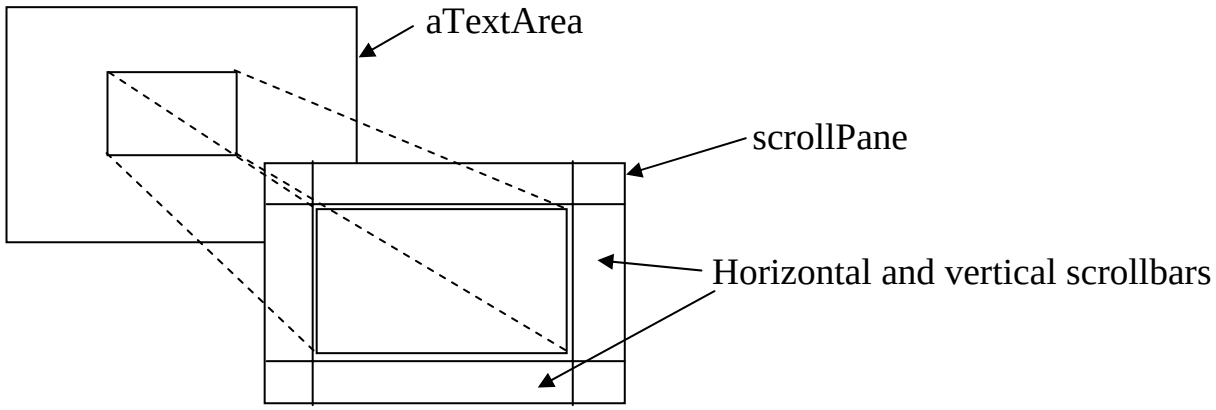
  This will create a JTextArea object with the 10x20 pixel area.

  The user can type additional characters outside the area but those characters are invisible.

- To see the invisible characters, we need scrollbars for a *JTextArea* object. This can be done by putting the *JTextArea* object into a *JScrollPane* object.

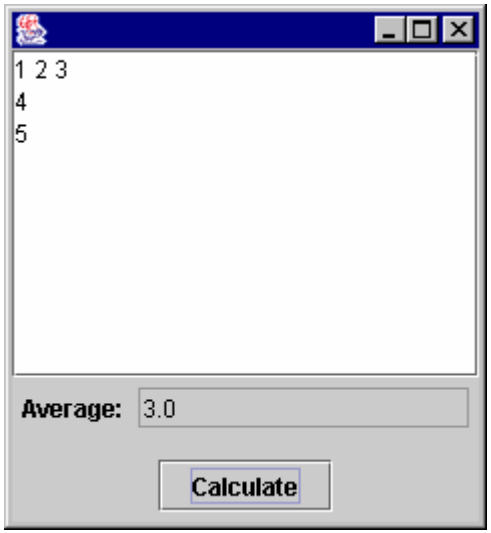
```
JScrollPane scrollPane = new JScrollPane(aTextArea);
```

  This creates a pane that displays the contents of *aTextArea*, where the vertical and horizontal scrollbars appear on the pane when the contents are larger than the area of *aTextArea*.



Example of Event Processing

- A user can enter integers into the text area and then click on the “Calculate” button to show the average of the numbers in the text field.

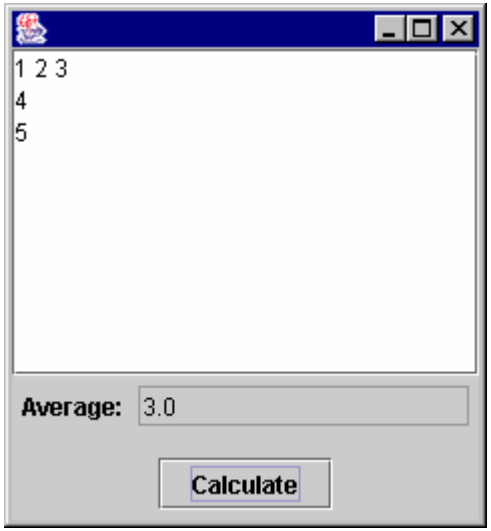


- One possible way to lay out components into the frame window.
  - Put *a* text area into the Center area of the content pane.
  - Create the so-called south panel and set the layout to be 2x1 grid layout.
  - Put the south panel into the South area of the content pane.
  - Create each component of the south panel: the avg panel and the button panel.
  - The avg panel contains one label and one text field.
  - The button panel contains only one button.

Example of Event Processing

File CalAvgFrame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.StringTokenizer;

public class CalAvgFrame extends JFrame
{ public CalAvgFrame( )
  { setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    dataTextArea = new JTextArea(10, 20);
    JScrollPane scrollPane = new JScrollPane(dataTextArea);

    JLabel avgLabel = new JLabel("Average: ");
    avgTextField = new JTextField(15); avgTextField.setEditable(false);
    JPanel avgPanel = new JPanel( );
    avgPanel.add(avgLabel); avgPanel.add(avgTextField);

    JButton calculateButton = new JButton("Calculate");


    JPanel buttonPanel = new JPanel( ); buttonPanel.add(calculateButton);

    JPanel southPanel = new JPanel( );
    southPanel.setLayout(new GridLayout(2,1));
    southPanel.add(avgPanel); southPanel.add(buttonPanel);

    Container contentPane = getContentPane( );
    contentPane.add(scrollPane, BorderLayout.CENTER);
    contentPane.add(southPanel, BorderLayout.SOUTH);
    pack( );
  }
  JTextArea dataTextArea;
  JTextField avgTextField;

}
```



File CalAvgFrameTest.java

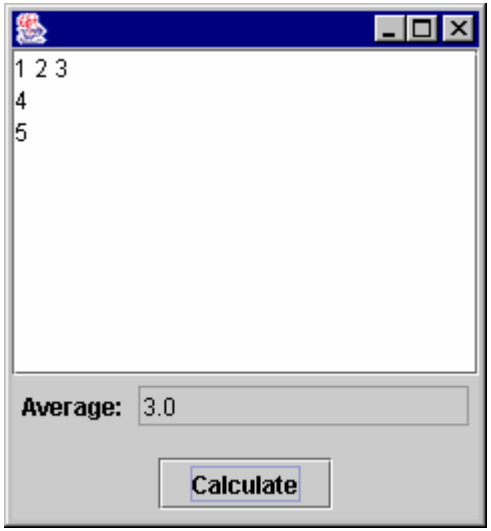
```
public class CalAvgFrameTest
{   public static void main(String[ ] args)
    {  CalAvgFrame f = new CalAvgFrame( );
       f.show( );
    }
}
```

Example of Event Processing

File CalAvgFrame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.StringTokenizer;

public class CalAvgFrame extends JFrame
{ public CalAvgFrame( )
  { setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    dataTextArea = new JTextArea(10, 20);
    JScrollPane scrollPane = new JScrollPane(dataTextArea);

    JLabel avgLabel = new JLabel("Average: ");
    avgTextField = new JTextField(15); avgTextField.setEditable(false);
    JPanel avgPanel = new JPanel( );
    avgPanel.add(avgLabel); avgPanel.add(avgTextField);

    JButton calculateButton = new JButton("Calculate");
    ButtonListener buttonListener = new ButtonListener( );
    calculateButton.addActionListener(buttonListener);
    JPanel buttonPanel = new JPanel( ); buttonPanel.add(calculateButton);

    JPanel southPanel = new JPanel( );
    southPanel.setLayout(new GridLayout(2,1));
    southPanel.add(avgPanel); southPanel.add(buttonPanel);

    Container contentPane = getContentPane( );
    contentPane.add(scrollPane, BorderLayout.CENTER);
    contentPane.add(southPanel, BorderLayout.SOUTH);
    pack( );
  }
  JTextArea dataTextArea;
  JTextField avgTextField;
}
```

add ButtonListener as an inner class.



File CalAvgFrame.java (Continued)

```
private class ButtonListener implements ActionListener
{  public void actionPerformed(ActionEvent event)
   {  String input = dataTextArea.getText( );
      StringTokenizer tokenizer = new StringTokenizer(input);
      int numInput = tokenizer.countTokens( );
      double sum = 0; double avg;
      for (int i=1; i<=numInput; i++)
      {  String currentToken = tokenizer.nextToken( );
         double currentNumber = Double.parseDouble(currentToken);
         sum = sum + currentNumber;
      }
      if (numInput == 0)  avg = 0;  else avg = sum/numInput;
      avgTextField.setText(Double.toString(avg));
   }
}
```

Choices

- In this lecture, we will learn how to create three types of choices:
  - Radio Buttons (objects in the *JRadioButton* class)
  - Combo Boxes (objects in the *JComboBox* class)
  - Check Boxes (objects in the *JCheckBox* class)

JRadioButton Class

- In a radio button set, the choices are *mutually exclusive*; that is only one button can be selected at one time.
- When the user selects another button in the same set, the previously selected button is automatically turned off.
- These buttons are called *radio buttons* because they work like buttons of the station selector in a car radio.
- To create a set of radio buttons, we first create each *JRadioButton* object individually and then add them to a *ButtonGroup* object.

```
JRadioButton smallButton = new JRadioButton(“Small”);
JRadioButton mediumButton = new JRadioButton(“medium”);
JRadioButton largeButton = new JRadioButton(“Large”);
ButtonGroup bg = new ButtonGroup();
bg.add(smallButton); bg.add(mediumButton); bg.add(largeButton);
```

- Note that adding radio buttons into a ButtonGroup means that only one button in the group can be one at any time.

JRadioButton Class

- We can set one of the radio buttons to be “selected” as a default choice by using the *setSelected( )* method.

```
smallButton.setSelected(true)
```

- We can also use the *isSelected( )* method to find out whether a button is currently selected.

```
smallButton.isSelected(); //returns true if smallButton is currently selected.
```

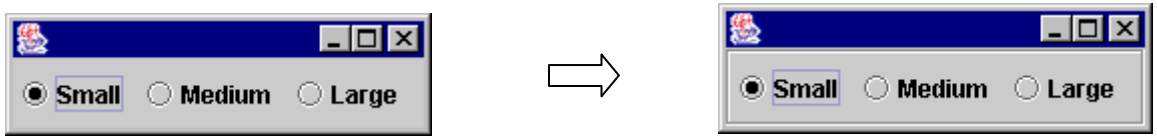
- To show the radioButton set, we still need to put them into a *Container* object first.

```
JPanel radioButtonPanel = new JPanel();
radioButtonPanel.add(smallButton);
radioButtonPanel.add(mediumButton);
radioButtonPanel.add(largeButton);
```

JRadioButton Class

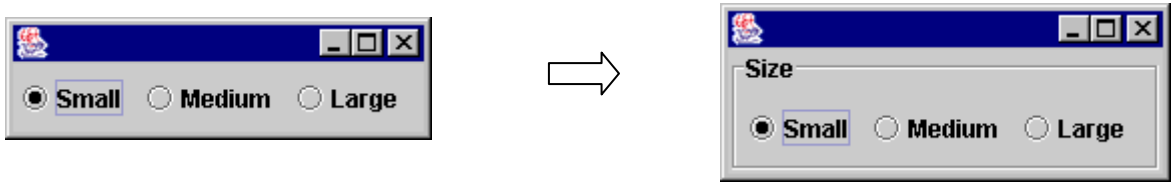
- We can add a border to a panel.
- There are a lot of border types defined as Java classes (located in *javax.swing.border* package).
- To add a border to *radioButtonPanel*, we created an object of that border class and use the *setBorder( )* method.

```
radioButtonPanel.setBorder(new EtchedBorder());
```



- We can also add a title to the border by using the *TitledBorder* class as follows:

```
radioButtonPanel.setBorder(new TitledBorder(new EtchedBorder(), “Size”));
```



JComboBox Class

- If we have a large number of mutually exclusive choices, it would take a lot of space to represent these choices by using a radio button set.
- Instead, we can use an object of the *JComboBox* class.

- A *JComboBox* object is a combination of a list and a text field.
- The text field displays the name of the current selection.
- When we click on the arrow to the right of the text field, a list of selections drops down and one of them can be chosen from the list.

JComboBox Class

- To use a combo box, we construct an object of the *JComboBox* class:

```
JComboBox facenameCombo = new JComboBox();
```

- Then each item can be added to the *JComboBox* object by using the *addItem( )* method:

```
facenameCombo.addItem("Serif");
facenameCombo.addItem("SansSerif");
facenameCombo.addItem("Monospaced");
```

- The *JComboBox* object can be set to be editable for typing our own choice:

```
facenameCombo.setEditable(true);
```

- To get the selected item from the *JComboBox* object, use:

```
String selectedFacename = (String) facenameCombo.getSelectedItem();
```

  The *getSelectedItem( )* method returns a reference of type *Object*.

- To set an item as a default choice for user, use:

```
facenameCombo.setSelectedItem("Serif");
```

JCheckBox Class

- Choices are sometimes not mutually exclusive. We can select one, more than one, or none.
- This type of choice can be implemented as a separate box by using the *JCheckBox* class.

- We construct an object of the JCheckBox class by:

```
JCheckBox italicCheckBox = new JCheckBox(“Italic”);
JCheckBox boldCheckBox = new JCheckBox(“Bold”);
```

- Since each *JCheckBox* object can be independently chosen, there is no concept of a button group.
- To find out whether each check box is selected, use the *isSelected( )* method:

```
      italicCheckBox.isSelected(); //returns true if smallButton is currently
selected.
```

Example of Processing Choices

- Radio buttons, check boxes, and combo boxes generate an *ActionEvent* object when a user selects an item.
- We need to define a class implementing the *ActionListener* interface.
- A frame window contains various types of choices. When a choice is selected, the text in the text field changes its appearance according to the selected choice.

Example of Event Processing

File ChoiceFrame.java

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

public class ChoiceFrame extends JFrame
{ public ChoiceFrame( )
  {  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setFrameComponents( );
     setSampleFont( );
  }

  private void setFrameComponents( ) { }

  private void setSampleFont( ) { }

  // instance variables
  JTextArea sampleText;    JComboBox facenameBox;
  JRadioButton smallButton;   JRadioButton largeButton;
  JCheckBox italicBox;   JCheckBox boldBox;




}
```

File ChoiceFrame.java (Continued)

```
private void setFrameComponents( )
{  sampleText = new JTextArea(2, 20); sampleText.setText("Java OOP Course");
   sampleText.setEditable(false);



   facenameBox = new JComboBox( );
   facenameBox.addItem("Serif"); facenameBox.addItem("Monospaced");
   facenameBox.setEditable(false);

   JPanel facenamePanel = new JPanel( ); facenamePanel.add(facenameBox);

   smallButton = new JRadioButton("Small"); largeButton = new JRadioButton("Large");

   largeButton.setSelected(true);
   ButtonGroup sizeGroup = new ButtonGroup( );
   sizeGroup.add(smallButton); sizeGroup.add(largeButton);
   JPanel sizePanel = new JPanel( );
   sizePanel.add(smallButton); sizePanel.add(largeButton);
   sizePanel.setBorder(new TitledBorder(new EtchedBorder( ), "Size"));

   italicBox = new JCheckBox("Italic");
   boldBox = new JCheckBox("Bold");
   JPanel stylePanel = new JPanel( );
   stylePanel.add(italicBox); stylePanel.add(boldBox);
   stylePanel.setBorder(new TitledBorder(new EtchedBorder( ), "Style"));

   JPanel southPanel = new JPanel( );
   southPanel.setLayout(new GridLayout(3,1));
   southPanel.add(facenamePanel); southPanel.add(sizePanel); southPanel.add(stylePanel);

   Container contentPane = getContentPane( );
   contentPane.add(sampleText, BorderLayout.CENTER);
   contentPane.add(southPanel, BorderLayout.SOUTH);

   pack( );
 }
```

Example of Event Processing

File ChoiceFrame.java

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

public class ChoiceFrame extends JFrame
{ public ChoiceFrame( )
  {  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

     setFrameComponents( );
     setSampleFont( );

  }

  private void setFrameComponents( ) { }

  private void setSampleFont( ) { }

  // instance variables
  JTextArea sampleText;    JComboBox facenameBox;
  JRadioButton smallButton;   JRadioButton largeButton;
  JCheckBox italicBox;   JCheckBox boldBox;









}
```

File ChoiceFrame.java (Continued)

```
private void setSampleFont( )
{  String facename = (String) facenameBox.getSelectedItem( );

   int style = 0;
   if (italicBox.isSelected( )) style = style + Font.ITALIC;
   if (boldBox.isSelected( )) style = style + Font.BOLD;

   final int SMALL_SIZE = 16; final int LARGE_SIZE = 24;
   int size = 0;
   if (smallButton.isSelected( )) size = SMALL_SIZE;
   else if (largeButton.isSelected( )) size = LARGE_SIZE;

   sampleText.setFont(new Font(facename, style, size));
   sampleText.setText(sampleText.getText( ));
}
```

File ChoiceFrameTest.java

```
public class ChoiceFrameTest
{ public static void main(String[ ] args)
  {  ChoiceFrame cf = new ChoiceFrame( );
     cf.show( );
  }
}
```

Example of Event Processing

File ChoiceFrame.java

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

public class ChoiceFrame extends JFrame
{ public ChoiceFrame( )
  {  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     setFrameComponents( );
     setSampleFont( );
  }

  private void setFrameComponents( ) { }

  private void setSampleFont( ) { }

  // instance variables
  JTextArea sampleText;    JComboBox facenameBox;
  JRadioButton smallButton;   JRadioButton largeButton;
  JCheckBox italicBox;   JCheckBox boldBox;

  private class ChoiceListener implements ActionListener
  {  public  void actionPerformed(ActionEvent event)
     {   setSampleFont();
     }
  }

}
```

File ChoiceFrame.java (Continued)

```
private void setFrameComponents( )
{  sampleText = new JTextArea(2, 20); sampleText.setText("Java OOP Course");
   sampleText.setEditable(false);

   ChoiceListener listener = new ChoiceListener( );

   facenameBox = new JComboBox( );
   facenameBox.addItem("Serif"); facenameBox.addItem("Monospaced");
   facenameBox.setEditable(false);
   facenameBox.addActionListener(listener);
   JPanel facenamePanel = new JPanel(); facenamePanel.add(facenameBox);

   smallButton = new JRadioButton("Small"); largeButton = new JRadioButton("Large");
   smallButton.addActionListener(listener); largeButton.addActionListener(listener);
   largeButton.setSelected(true);
   ButtonGroup sizeGroup = new ButtonGroup( );
   sizeGroup.add(smallButton); sizeGroup.add(largeButton);
   JPanel sizePanel = new JPanel( );
   sizePanel.add(smallButton); sizePanel.add(largeButton);
   sizePanel.setBorder(new TitledBorder(new EtchedBorder( ), "Size"));

   italicBox = new JCheckBox("Italic"); italicBox.addActionListener(listener);
   boldBox = new JCheckBox("Bold");  boldBox.addActionListener(listener);
   JPanel stylePanel = new JPanel( );
   stylePanel.add(italicBox); stylePanel.add(boldBox);
   stylePanel.setBorder(new TitledBorder(new EtchedBorder( ), "Style"));

   JPanel southPanel = new JPanel( );
   southPanel.setLayout(new GridLayout(3,1));
   southPanel.add(facenamePanel); southPanel.add(sizePanel); southPanel.add(stylePanel);

   Container contentPane = getContentPane();
   contentPane.add(sampleText, BorderLayout.CENTER);
   contentPane.add(southPanel, BorderLayout.SOUTH);

   pack( );
}
```

Menus

- We can create a menu by first constructing an object of the *JMenuBar* class and then attaching it to the layered pane of a frame window (a *JFrame* object) by using the *setJMenuBar( )* method.

```
public class MyFrame extends JFrame
{ . . .
  JMenuBar menuBar = new JMenuBar();
  setJMenuBar(menuBar); //attach menuBar to the layered pane of this JFrame object
  . . .
}
```

- Then we can add menus (objects of the *JMenu* class) to the menu bar:

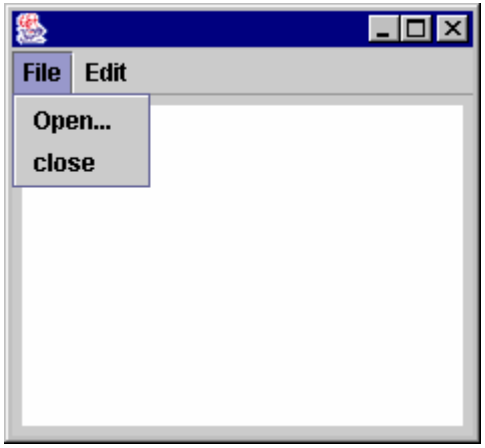
```
JMenu fileMenu = new JMenu(“File”); menuBar.add(fileMenu);
JMenu editMenu = new JMenu(“Edit”); menuBar.add(editMenu);
```

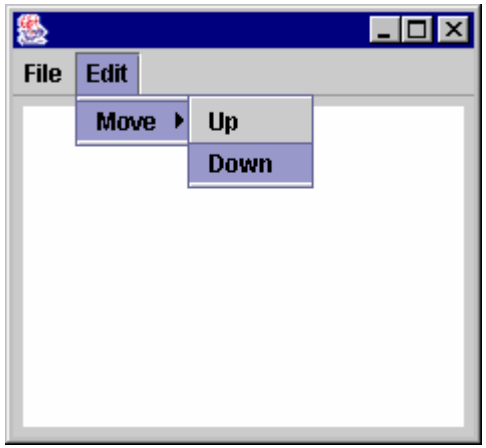
- Finally we can add menu items (objects of the *JMenuItem* class) or submenus (objects of the *JMenu* class) to the particular menu:

```
JMenuItem openMenuItem = new JMenuItem(“Open...”);
fileMenu.add(openMenuItem);
JMenu moveSubMenu = new JMenu(“Move”); editMenu.add(moveSubMenu);
JMenuItem upMenuItem = new JMenuItem(“Up”); moveSubMenu.add(upMenuItem);
JMenuItem downMenuItem = new JMenuItem(“Down”); moveSubMenu.add(downMenuItem);
```

Menus

- When a user selects a particular menu item, an action event is generated.
- We can install a listener only to menu items, not to menus or a menu bar.
- To construct a menu-selection listener, we define its class by implementing the *ActionListener* interface.





Example of Event Processing

File MenuFrame.java

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

public class MenuFrame extends JFrame
{  public MenuFrame( )
   {  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setFrameComponents( );
   }

   private void setFrameComponents( ) { }

   JTextArea shownText;
   JMenuItem openMenuItem;
   JMenuItem closeMenuItem;
   JMenuItem upMenuItem;
   JMenuItem downMenuItem;

   private class MenuItemListener implements ActionListener
   { public void actionPerformed(ActionEvent event)
     {  if (event.getSource( ) == openMenuItem)
        {  shownText.setText("File-Open menu item is selected.");
        }
        else if (event.getSource( ) == closeMenuItem)
        {  shownText.setText("File-Close menu item is selected.");
        }
        else if (event.getSource( ) == upMenuItem)
        {  shownText.setText("Edit-Move-Up menu is selected.");
        }
        else if (event.getSource( ) == downMenuItem)
        {  shownText.setText("Edit-Move-Down menu is selected.");
        }

     }
   }
}
```

File MenuFrame.java (Continued)

```
private void setFrameComponents( )
{  JMenuBar menuBar = new JMenuBar( );
   setJMenuBar(menuBar);
   MenuItemListener listener = new MenuItemListener( );

   JMenu fileMenu = new JMenu("File");
   menuBar.add(fileMenu);
   openMenuItem = new JMenuItem("Open...");
   openMenuItem.addActionListener(listener);
   closeMenuItem = new JMenuItem("Close");
   closeMenuItem.addActionListener(listener);
   fileMenu.add(openMenuItem); fileMenu.add(closeMenuItem);

   JMenu editMenu = new JMenu("Edit");
   menuBar.add(editMenu);
   JMenu moveMenu = new JMenu("Move");
   upMenuItem = new JMenuItem("Up");
   upMenuItem.addActionListener(listener);
   downMenuItem = new JMenuItem("Down");
   downMenuItem.addActionListener(listener);
   moveMenu.add(upMenuItem); moveMenu.add(downMenuItem);
   editMenu.add(moveMenu);

   JPanel panel = new JPanel( );
   shownText = new JTextArea(10, 20);  shownText.setEditable(false);
   panel.add(shownText);

   Container contentPane = getContentPane( );
   contentPane.add(panel, BorderLayout.CENTER);
   pack( );
}
```