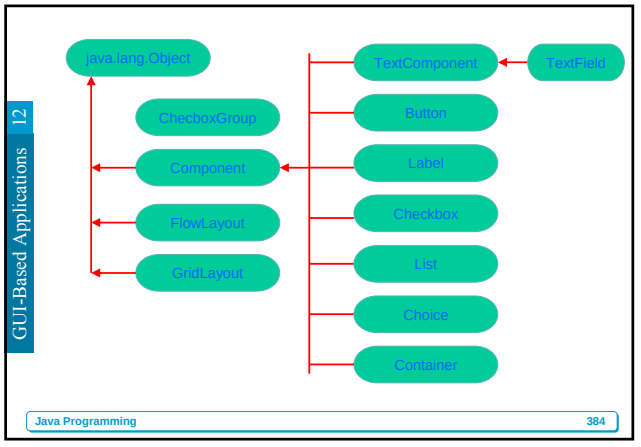
12 GUI-Based Applications

GUI-Based Applications

- Identify key AWT components
- Use AWT components to build user interfaces for real programs
- Control the colors and fonts used by an AWT component
- Use the Java printing mechanism



Label

- public Label()

  constructs an empty Label- text is not displayed

- public Label(String s)

  Constructs a Label that displays the text s with default left-justified alignment

- public Label(String s, int alignment)

  Label.LEFT, Label.CENTER, Label.RIGHT

- public String getText()
- public void setText(String s)
- public void setAlignment(int alignment)

TextField

- public TextField()

  constructs a TextField object

- public TextField(int columns)

  constructs an empty TextField object with specified number of columns

- public TextField(String s, int columns)
- public void setEchoChar(char c)
- public void setEditable(boolean b) // true == editable
- public String getText()
- public void setText(String s)


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

public class TextFieldApp implements ActionListener {
        private Frame myFrame      ;
        private Label myLabel      ;
        private Panel myPanel      ;
        private TextField password ;
        private TextField tf       ;

        public TextFieldApp ()         {
                myFrame = new Frame("Sample Application") ;
                myPanel = new Panel() ;
                // Label
                myLabel = new Label("Enter password") ;
                // TextField
                password = new TextField("") ;
                password.setEchoChar('*') ;
                password.addActionListener(this) ;
```

```
                // Panel
                myPanel.setLayout(new GridLayout(1,2)) ;
                myPanel.add(myLabel) ;
                myPanel.add(password) ;
                // Text Field
                tf = new TextField() ;
                tf.setEditable(false) ;
                myFrame.add(myPanel,BorderLayout.NORTH) ;
                myFrame.add(tf,BorderLayout.CENTER);
                // setSize and setVisible
                myFrame.setSize(500,256) ;
                myFrame.setVisible(true) ;
        }
        public void actionPerformed(ActionEvent e)    {
                String s ;
                password.setText("");
                s = e.getActionCommand() ;
                if(s.compareTo("keyword") == 0) tf.setEditable(true)    ;
                else  tf.setEditable(false)             ;

        }
```



```
public static void main(String[] args) {
                TextFieldApp tfa = new TextFieldApp() ;
        }
}
```


Choice

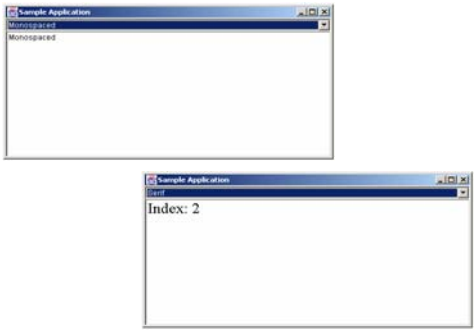
- public String getItem(int index)
- public synchronized void add(String s)
- public synchronized String getSelectedItem()
- public int getSelectedIndex()
- public synchronized String insert(String s, int index)
- public synchronized void remove(String s)


```
import java.awt.* ;
import java.awt.event.* ;
public class ChoiceApp implements ItemListener {
        private Frame myFrame      ;
        private TextField tf                    ;
        private Choice fonts       ;
        static int size = 10 ;
        public ChoiceApp () {
            myFrame = new Frame("Sample Application") ;
            // Choice
            fonts = new Choice() ;
            fonts.add( "Monospaced") ; // Courier
            fonts.add( "SansSerif") ; // Helvetica
            fonts.add( "Serif") ; // Times
            fonts.addItemListener(this) ;
```



```
                // TextField
                tf = new TextField(fonts.getItem(0), 30) ;
                myFrame.add(fonts,BorderLayout.NORTH) ;
                myFrame.add(tf,BorderLayout.CENTER);
                // setSize and setVisible
                myFrame.setSize(500,256) ;
                myFrame.setVisible(true) ;
        }
        public void itemStateChanged( ItemEvent e)              {
                tf.setText( "Index: " + fonts.getSelectedIndex()) ;
                tf.setFont(new Font(fonts.getSelectedItem(),
                        tf.getFont().getStyle(),ChoiceApp.size++)) ;
        }
```

```
    public static void main(String[] args) {
        ChoiceApp ca = new ChoiceApp() ;
    }
}
```

Checkbox and CheckboxGroup

- public Checkbox(String s)
- public Checkbox(String s, CheckboxGroup c, boolean state)
- public CheckboxGroup()


```
import java.awt.* ;
import java.awt.event.* ;
public class CheckboxApp implements ItemListener {
    private Frame myFrame      ;
    private TextField tf                ;
    private Checkbox bold,italic ;
    public CheckboxApp () {
        myFrame = new Frame("Sample Application") ;
        // Checkbox
        bold   = new Checkbox( "Bold" ) ;
        italic = new Checkbox( "Italic" ) ;
        bold.addItemListener(this) ;
        italic.addItemListener(this) ;
```



```
        // TextField
        tf = new TextField("", 30) ;
        myFrame.add(tf,BorderLayout.NORTH) ;
        myFrame.add(bold,BorderLayout.CENTER) ;
        myFrame.add(italic,BorderLayout.EAST);
        // setSize and setVisible
        myFrame.setSize(500,128) ;
        myFrame.setVisible(true) ;
    }
    public void itemStateChanged( ItemEvent e) {
      int valBold = (bold.getState() ? Font.BOLD : Font.PLAIN ) ;
      int valItalic = (italic.getState() ? Font.ITALIC : Font.PLAIN ) ;
tf.setFont(new Font("Serif",valBold+valItalic,18) );
    }
```



```
    public static void main(String[] args)
    {
        CheckboxApp ca = new CheckboxApp() ;
    }
}
```


```
import java.awt.* ;
import java.awt.event.* ;
public class GroupCheckboxApp implements ItemListener {
        private Frame myFrame      ;
        private TextField tf                    ;
        private Checkbox plain,bold,italic,boldItalic ;
        private CheckboxGroup fontStyle ;
        private Font boldFont,italicFont,plainFont,boldItalicFont ;
        public GroupCheckboxApp () {
          myFrame = new Frame("Sample Application") ;
          myFrame.setLayout(new FlowLayout());
          // Predefined Fonts
          boldFont = new Font("Serif",Font.BOLD,18) ;
          italicFont = new Font("Serif",Font.ITALIC,18) ;
          plainFont = new Font("Serif",Font.PLAIN,18) ;
boldItalicFont = new Font("Serif",Font.BOLD+Font.ITALIC,18) ;
```



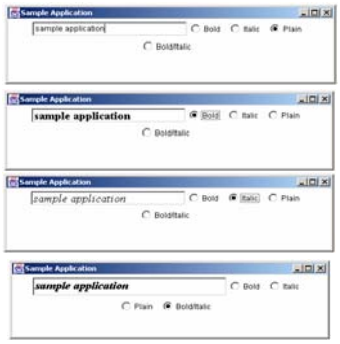
```
        // GroupCheckbox
        CheckboxGroup fontStyle = new CheckboxGroup() ;
        // Checkbox
        bold   = new Checkbox( "Bold" ,fontStyle,false) ;
        italic = new Checkbox( "Italic" ,fontStyle,false) ;
        plain  = new Checkbox( "Plain" ,fontStyle,true) ;
        boldItalic = new Checkbox( "Bold/Italic" ,fontStyle,false) ;
        // add ItemListener
        bold.addItemListener(this) ;
        italic.addItemListener(this) ;
        plain.addItemListener(this) ;
        boldItalic.addItemListener(this) ;
        // TextField
       tf = new TextField("", 30) ;
```



```
        myFrame.add(tf) ;
        myFrame.add(bold) ;
        myFrame.add(italic);
        myFrame.add(plain) ;
        myFrame.add(boldItalic);
        // setSize and setVisible
        myFrame.setSize(500,128) ;
        myFrame.setVisible(true) ;
       }
       public void itemStateChanged( ItemEvent e) {
          if( e.getSource() == plain )
            tf.setFont( plainFont);
          else  if( e.getSource() == bold )
            tf.setFont( boldFont);
          else  if( e.getSource() == italic )
            tf.setFont( italicFont);
          else if( e.getSource() == boldItalic )
            tf.setFont( boldItalicFont);
       }
```

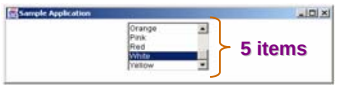


```
        public static void main(String[] args)
        {
                GroupCheckboxApp ca = new GroupCheckboxApp() ;
        }
}
```

Single-Selection List

colorList = new List(5 , false) ; // single-select




```
public class SingleSelectListApp implements ItemListener {
    private Frame myFrame      ;
    private List colorList     ;
    private String[] colorNames = {
       "Black" , "Blue", "Cyan", "Dark Gray", "Gray", "Green",
       "Light Gray","Magenta","Orange", "Pink","Red","White","Yellow" } ;
    private Color[] colors = {
        Color.black ,Color.blue,Color.cyan,Color.darkGray,Color.gray,
        Color.green,Color.lightGray,Color.magenta,Color.orange,
        Color.pink, Color.red, Color.white, Color.yellow  } ;
    public SingleSelectListApp () {
          myFrame = new Frame("Sample Application") ;
          myFrame.setLayout(new FlowLayout());
          // List
          colorList  = new List( 5 , false) ;
          // add ItemListener
          colorList.addItemListener(this) ;
          // add items to the list
          for(int i=0;i< colorNames.length;i++)
              colorList.add(colorNames[i]) ;
          myFrame.add(colorList)    ;
          // setSize and setVisible
          myFrame.setSize(500,128) ;
          myFrame.setVisible(true) ;
       }
```



```
    public void itemStateChanged( ItemEvent e) {
        myFrame.setBackground(colors[colorList.getSelectedIndex()]) ;
    }

    public static void main(String[] args)  {
        SingleSelectListApp ssla = new SingleSelectListApp() ;
    }
}
```


Multiple-Selection List

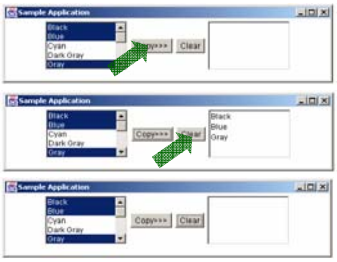
colorList = new List(5 , true) ; // multiple-select


```
public class MultipleSelectListApp implements ActionListener {
        private Frame myFrame      ;
        private List colorList     ;
        private List copyList      ;
        private Button copy        ;
        private Button clear       ;

        public MultipleSelectListApp () {
            myFrame = new Frame("Sample Application") ;
            myFrame.setLayout(new FlowLayout());
            // Lists
            colorList  = new List( 5 , true) ;
            copyList  = new List( 5 , false) ;
            // Button
            copy = new Button("Copy>>>") ;
            copy.addActionListener(this) ;
            clear = new Button("Clear") ;
            clear.addActionListener(this) ;
            // add items to the list
            for(int i=0;i< colorNames.length;i++)
                colorList.add(colorNames[i]) ;
```

```
                    myFrame.add(colorList)    ;
                    myFrame.add(copy)         ;
                    myFrame.add(clear)        ;
                    myFrame.add(copyList)     ;
                    // setSize and setVisible
                    myFrame.setSize(500,128) ;
                    myFrame.setVisible(true) ;
                }
                public void actionPerformed( ActionEvent e) {
                        if ( e.getSource() == copy ) {
                          String[] colors ;
                          // get the selected states
                          colors = colorList.getSelectedItems() ;
                          // copy them to copyList
                          for( int i=0 ; i < colors.length ; i++ )
                              copyList.add( colors[i] ) ;
                        }else{
                          copyList.clear();
                        }
                }
```



```
public static void main(String[] args)
{
  MultipleSelectListApp ssla = new MultipleSelectListApp() ;
}
}
```



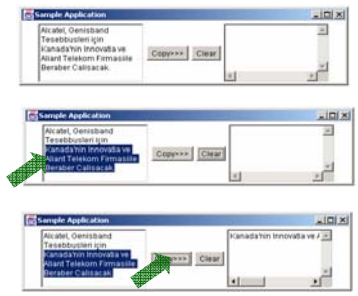
TextArea

- public TextArea( )
- public TextArea(int rows, int columns)
- public TextArea(String s)
- public TextArea(String s, int rows, int columns)
- public TextArea(String s, int rows, int columns, int scrollbars)


```
public class TextAreaApp implements ActionListener, TextListener {
        private Frame myFrame   ;
        private TextArea t1,t2  ;
        private Button copy     ;
        private Button clear    ;

        public TextAreaApp () {
          String s = "Alcatel, Genisband Tesebbusleri için " +
                     "Kanada'nin Innovatia ve Aliant Telekom Firmasi" +
                     "ile Beraber Calisacak." ;
          myFrame = new Frame("Sample Application") ;
          myFrame.setLayout(new FlowLayout());
          // TextArea
      t1  = new TextArea( s , 5 , 20 ,TextArea.SCROLLBARS_NONE ) ;
          t2  = new TextArea( 5 , 20 ) ;
          // Button
          copy = new Button("Copy>>>") ;
          copy.addActionListener(this) ;
          clear = new Button("Clear") ;
          clear.addActionListener(this) ;
          myFrame.add(t1)             ;
          myFrame.add(copy)                 ;
          myFrame.add(clear)                ;
          myFrame.add(t2)             ;
```



```
          // setSize and setVisible
          myFrame.setSize(500,128) ;
          myFrame.setVisible(true) ;
        }
        public void textValueChanged( TextEvent e )
        {
            TextComponent source = (TextComponent) e.getSource() ;
                t2.setText( source.getText() ) ;
        }
        public void actionPerformed( ActionEvent e){
                if ( e.getSource() == copy ) {
                  t2.setText( t1.getSelectedText() )    ;
                }else{
                  t2.setText("") ;
                }
        }

        public static void main(String[] args) {
                TextAreaApp taa = new TextAreaApp() ;
        }
}
```

Canvas

- A canvas is a dedicated drawing area that can also receive mouse events.
- Class Canvas inherits from Component
- The paint method for a Canvas must be overriden to draw on the Canvas
- Drawing on a Canvas is performed with coordinates that are measured from the upper-left corner (0,0) of the Canvas.

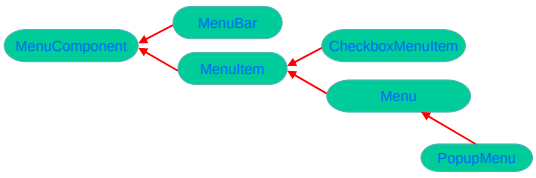

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
public class MyCanvas extends Canvas implements KeyListener {
        int index;
        Color[] colors =  {
        Color.black , Color.blue, Color.cyan, Color.darkGray, Color.gray,
        Color.green,Color.lightGray, Color.magenta, Color.orange,
        Color.pink, Color.red, Color.white,Color.yellow } ;
        public void paint(Graphics g) {
                g.setColor(colors[index]);
                g.fillRect(0,0,getSize().width,getSize().height);
        }
       public void keyTyped(KeyEvent ev) {
         index++;
         if (index == colors.length)
           index =0;
         repaint();
      }
    public void keyPressed(KeyEvent ev) {}
    public void keyReleased(KeyEvent ev) {}
```



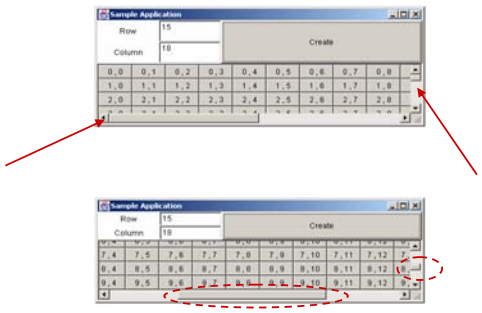
```
    public static void main(String args[]) {
            Frame f = new Frame("Canvas");
            MyCanvas mc = new MyCanvas();
            f.add(mc,BorderLayout.CENTER);
            f.setSize(150, 150);
            mc.requestFocus();
            mc.addKeyListener(mc);
           f.setVisible(true);
        }
}
```

Menus with Frames

- Menus are an integral part of GUIs
- Menus allow the user to perform actions without unneccessarily cluttering a graphical user interface with extra GUI components
- Menus can only be used with Frames
- PopupMenus can be used with any GUI component



ScrollPane



```
import java.awt.* ;
import java.awt.event.* ;
public class ScrollPaneApp implements ActionListener, TextListener {
        private Frame myFrame      ;
        private Panel np           ;
        private Panel innerPanel   ;
        private ScrollPane sp      ;
        private Button[] buttons   ;
        private Label rlabel,clabel   ;
        private TextField rtf,ctf     ;
        private Button create         ;
        private int row=10,column=10     ;
        private Dialog d              ;
```



```
        public ScrollPaneApp (){
            myFrame = new Frame("Sample Application") ;
            np = new Panel() ;
            np.setLayout(new GridLayout(2,2)) ;
            rlabel = new Label( "Row" , Label.CENTER ) ;
            clabel = new Label( "Column" , Label.CENTER ) ;
            rtf    = new TextField( 10 ) ;
            rtf.addTextListener(this)   ;
            ctf    = new TextField( 10 ) ;
            ctf.addTextListener(this)   ;
            np.add(rlabel)              ;
            np.add(rtf)                 ;
            np.add(clabel)              ;
            np.add(ctf)                 ;
```



```
            // ScrollPane
            sp = new ScrollPane() ;
            innerPanel = new Panel() ;
            sp.add(innerPanel)   ;
            // Button
            create = new Button("Create") ;
            create.addActionListener(this);
            // setSize and setVisible
            myFrame.add(np,BorderLayout.WEST) ;
            myFrame.add(create,BorderLayout.CENTER) ;
            myFrame.add(sp,BorderLayout.SOUTH) ;
            myFrame.setSize(500,200) ;
            myFrame.setVisible(true) ;
        }
```



```
        public void textValueChanged( TextEvent e ) {
            if ( e.getSource() == rtf ) {
                row = Integer.parseInt( rtf.getText() ) ;
            }else {
                column = Integer.parseInt( ctf.getText() ) ;
            }
        }
```



```
        public void actionPerformed( ActionEvent e) {
            if( (row>0) && (column>0) ) {
            innerPanel.setLayout(new GridLayout(row,column)) ;
            buttons = new Button[row*column] ;
            for(int i=0;i<row;i++)
              for(int j=0;j<column;j++) {
                buttons[i*column+j] = new Button( i + " , " + j) ;
                    innerPanel.add(buttons[i*column+j]);
              }
             myFrame.pack();
        }else {
```



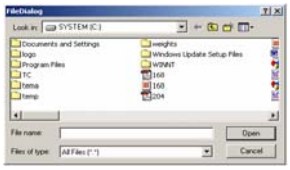
```
            d = new Dialog(myFrame, "Ooops Dialog", false);
            Button b = new Button("OK") ;
            b.addActionListener(this) ;
            d.add(new Label("Oooppps..."),BorderLayout.CENTER);
            d.add(b,BorderLayout.SOUTH);
            d.pack();
            d.setVisible(true);
          }
        }
        public static void main(String[] args) {
            ScrollPaneApp spa = new ScrollPaneApp() ;
        }
    }
```

Dialog


```
Dialog d = new Dialog(myFrame, "Dialog", false);
Button b = new Button("OK") ;
b.addActionListener(this) ;
d.add(new Label("Hello,I'm a Dialog"),BorderLayout.CENTER);
d.add(b,BorderLayout.SOUTH);
d.pack();
d.setVisible(true);
```

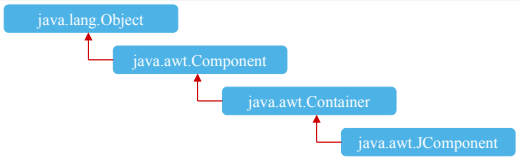
Creating a FileDialog


```
FileDialog d = new FileDialog(parentFrame, "FileDialog");

d.setVisible(true); // block here until OK selected

String fname = d.getFile();
```



A SWING

- Swing GUI components
  - Defined in package **javax.swing**
  - Original GUI components from Abstract Windowing Toolkit in **java.awt**
    - Heavyweight components - rely on local platform's windowing system for look and feel
  - Swing components are lightweight
    - Written in Java, not weighed down by complex GUI capabilities of platform
    - More portable than heavyweight components
  - Swing components allow programmer to specify look and feel
    - Can change depending on platform
    - Can be the same across all platforms



- **Component** defines methods that can be used in its subclasses (for example, **paint** and **repaint**)
- **Container** - collection of related components
  - When using **JFrames**, attach components to the content pane (a **Container**)
  - Method **add**
- **JComponent** - superclass to most Swing components
- Much of a component's functionality inherited from these classes

Swing Overview

- Some capabilities of subclasses of **JComponent**
  - Pluggable look and feel
  - Shortcut keys (mnemonics)
    - Direct access to components through keyboard
  - Common event handling
    - If several components perform same actions
  - Tool tips
    - Description of component that appears when mouse over it

JLabel

- Labels
  - Provide text instructions on a GUI
  - Read-only text
  - Programs rarely change a label's contents
  - Class **JLabel** (subclass of **JComponent**)
- Methods

```
18    label1 = new JLabel( "Label with text" );
```

  - **myLabel.setToolTipText("Text")**
    - Displays **"Text"** in a tool tip when mouse over label
  - **myLabel.setText("Text")**
  - **myLabel.getText()**

Swing A

Icon

- **Icon**
  - Object that implements interface **Icon**

  - One class is **ImageIcon** (**.gif** and **.jpeg** images)

```
Icon bug = new ImageIcon( "bug1.gif" );
```

  - Display an icon with **setIcon** method (of class **JLabel**)
    - **myLabel.setIcon(myIcon);**

```
label3.setIcon( bug );
```

Swing A

Alignment

- By default, text appears to right of image
- **JLabel** methods **setHorizontalTextPosition** and **setVerticalTextPosition**
  - Specify where text appears in label
  - Use integer constants defined in interface **SwingConstants** (**javax.swing**)
    - **SwingConstants.LEFT, RIGHT, BOTTOM, CENTER**
- Another **JLabel** constructor
  - **JLabel("Text", ImageIcon, Text_Alignment_CONSTANT)**

Swing A

JTextField, JPasswordField

- Single line areas in which text can be entered or displayed
- **JPasswordField**s show inputted text as an asterisk *
- **JTextField** extends **JTextComponent**
  - **JPasswordField** extends **JTextField**

- When **Enter** pressed
  - **ActionEvent** occurs
  - Currently active field "has the focus"

Swing A

Methods

- Constructors
  - **JTextField(10)**
    - Textfield with 10 columns of text
    - Takes average character width, multiplies by 10
  - **JTextField("Hi")**
    - Sets text, width determined automatically
  - **JTextField("Hi", 20)**
- **setEditable(boolean)**
  - If **false**, user cannot edit text
  - Can still generate events

- **getPassword**
  - Class **JPasswordField**
  - Returns password as an **array** of type **char**

Swing A

JButton

- Button
  - Component user clicks to trigger an action
  - Several types of buttons
    - Command buttons, toggle buttons, check boxes, radio buttons
- Command button
  - Generates **ActionEvent** when clicked
  - Created with class **JButton**
    - Inherits from class **AbstractButton**
    - Defines many features of Swing buttons
- **JButton**
  - Text on face called button label
  - Each button should have a different label
  - Support display of **Icon**s

Swing A

- Methods of class **JButton**
  - Constructors

```
JButton myButton = new JButton( "Label" );
JButton myButton = new JButton( "Label", myIcon );
```

  - **setRolloverIcon(myIcon)**
    - Sets image to display when mouse over button
- Class ActionEvent
  - **getActionCommand**
    - Returns label of button that generated event

Swing A

JCheckBox, JRadioButton

- State buttons
  - **JToggleButton**
    - Subclasses **JCheckBox**, **JRadioButton**
  - Have on/off (true/false) values
- Class **JCheckBox**

  - Text appears to right of checkbox
  - Constructor

```
JCheckBox myBox = new JCheckBox( "Title" );
```

Swing A

- When **JCheckBox** changes
  - **ItemEvent** generated
    - Handled by an **ItemListener**, which must define **itemStateChanged**
  - Register handlers with with **addItemListener**

```
private class CheckBoxHandler implements ItemListener {
        public void itemStateChanged( ItemEvent e ) {
    }
```

  - **getStateChange**
    - Returns **ItemEvent.SELECTED** or **ItemEvent.DESELECTED**
- **JTextField**
  - Method **setText(fontObject)**
    - **new Font(name, style_CONSTANT, size)**
    - **style_CONSTANT - FONT.PLAIN, BOLD, ITALIC**
      - Can add to get combinations

Swing A

- Radio buttons
  - Have two states: selected and deselected
  - Normally appear as a group
    - Only one radio button in the group can be selected at time
    - Selecting one button forces the other buttons off
  - Used for mutually exclusive options
  - **ButtonGroup** - maintains logical relationship between radio buttons
- Class **JRadioButton**
  - Constructor
    - **JRadioButton("Label", selected)**
    - If selected **true**, **JRadioButton** initially selected

Swing A

- Class **JRadioButton**
  - Generates **ItemEvents** (like **JCheckBox**)
- Class **ButtonGroup**
  - **ButtonGroup myGroup = new ButtonGroup();**
  - Binds radio buttons into logical relationship
  - Method **add**
    - Associate a radio button with a group

myGroup.add(myRadioButton)

Swing A

JComboBox

- Combo box (drop down list)
  - List of items, user makes a selection
  - Class **JComboBox**
    - Generate **ItemEvents**
- **JComboBox**

  - Constructor

JComboBox (arrayOfNames)

  - Numeric index keeps track of elements
    - First element added at index **0**
    - First item added is appears as currently selected item when combo box appears

Swing A

- **JComboBox** methods
  - **getSelectedIndex**
    - Returns the index of the currently selected item
    - **myComboBox.getSelectedIndex()**
  - **setMaximumRowCount(n)**
    - Set max number of elements to display when user clicks combo box
    - Scrollbar automatically provided
    - **setMaximumRowCount(3)**

JList

- List
  - Displays series of items, may select one or more
  - This section, discuss single-selection lists
- Class **JList**
  - Constructor **JList(arrayOfNames)**
    - Takes array of **Objects** (**Strings**) to display in list
  - **setVisibleRowCount(n)**
    - Displays **n** items at a time
    - Does not provide automatic scrolling

- **JScrollPane** object used for scrolling

```
c.add( new JScrollPane( colorList ) );
```

  - Takes component to which to add scrolling as argument
  - Add **JScrollPane** object to content pane
- **JList** methods
  - **setSelectionMode(selection_CONSTANT)**
  - **SINGLE_SELECTION**
    - One item selected at a time
  - **SINGLE_INTERVAL_SELECTION**
    - Multiple selection list, allows contiguous items to be selected
  - **MULTIPLE_INTERVAL_SELECTION**
    - Multiple-selection list, any items can be selected

- **JList** methods
  - **getSelectedIndex()**
    - Returns index of selected item
- Event handlers
  - Implement interface **ListSelectionListener** (**javax.swing.event**)
  - Define method **valueChanged**
  - Register handler with **addListSelectionListener**

Multiple Selection List

- Multiple selection lists
  - **SINGLE_INTERVAL_SELECTION**
    - Select a contiguous group of items by holding *Shift* key
  - **MULTIPLE_INTERVAL_SELECTION**
    - Select any amount of items
    - Hold *Ctrl* key and click each item to select
- **JList** methods
  - **getSelectedValues()**
    - Returns an array of **Objects** representing selected items
  - **setListData(arrayOfObjects)**
    - Sets items of **JList** to elements in **arrayOfObjects**

JPanel

- Complex GUIs
  - Each component needs to be placed in an exact location
  - Can use multiple panels
    - Each panel's components arranged in a specific layout
- Panels
  - Class **JPanel** inherits from **JComponent**, which inherits from **java.awt.Container**
    - Every **JPanel** is a **Container**
  - **JPanels** can have components (and other **JPanels**) added to them
    - **JPanel** sized to components it contains
    - Grows to accomodate components as they are added

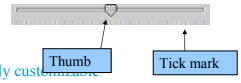
JTextArea

- **JTextArea**
  - Area for manipulating multiple lines of text
  - Like **JTextField**, inherits from **JTextComponent**
    - Many of the same methods
  - Does not have automatic scrolling
  - Methods
    - **getSelectedText**
      - Returns selected text (dragging mouse over text)
    - **setText(string)**
  - Constructor
    - **JTextArea(string, numrows, numcolumns)**
- **JScrollPane**
  - Provides scrolling for a component

- Initialize with component
  - **new JScrollPane(myComponent)**
- Can set scrolling policies (always, as needed, never)
- Methods **setHorizontalScrollBarPolicy**, **setVerticalScrollBarPolicy**
  - Constants:
  - **JScrollPane.VERTICAL_SCROLLBAR_ALWAYS**
  - **JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED**
  - **JScrollPane.VERTICAL_SCROLLBAR_NEVER**
    - Similar for **HORIZONTAL**
  - If set to **JScrollPane.HORIZONTAL_SCROLLBAR_NEVER**, word wrap

JSlider

- Select from a range of integer values



- Highly customizable
  - Snap-to ticks, major and minor ticks, labels
- When has focus (currently selected GUI component)
  - Use mouse or keyboard
    - Arrow or keys to move thumb, *Home*, *End*
- Have horizontal or vertical orientation
  - Minimum value at left/bottom, maximum at right/top
  - Thumb indicates current value

- Methods
  - Constructor
  - **JSlider(orientation_CONSTANT, min, max, initialValue)**
    - **orientation_CONSTANT**
      - **SwingConstants.HORIZONTAL**
      - **SwingConstants.VERTICAL**
    - **min, max** - range of values for slider
    - **initialValue** - starting location of thumb

```
diameter = new JSlider( SwingConstants.HORIZONTAL,
                        0, 200, 10 );
```


- Methods
  - **setMajorTickSpacing(n)**
    - Each tick mark represents **n** values in range
  - **setPaintTicks(boolean)**
    - **false** (default) - tick marks not shown
  - **getValue()**
    - Returns current thumb position
- Events
  - JSliders generates **ChangeEvent**s
    - **addChangeListener**
    - Define method **stateChanged**

JFrame

- **JFrame**
  - Inherits from **java.awt.Frame**, which inherits from **java.awt.Window**
  - **JFrame** is a window with a title bar and a border
    - Not a lightweight component - not written completely in Java
    - Window part of local platform's GUI components
      - Different for Windows, Macintosh, and UNIX
- **JFrame** operations when user closes window
  - Controlled with method **setDefaultCloseOperation**
    - Interface **WindowConstants** (**javax.swing**) has three constants to use
    - **DISPOSE_ON_CLOSE**, **DO_NOTHING_ON_CLOSE**, **HIDE_ON_CLOSE** (default)

- Windows take up valuable resources
  - Explicitly remove windows when not needed
  - Method **dispose** (of class **Window**, indirect superclass of **JFrame**)
    - Or, use **setDefaultCloseOperation**
  - **DO_NOTHING_ON_CLOSE** -
    - You determine what happens when user wants to close window
- Display
  - By default, window not displayed until method **show** called
  - Can display by calling method **setVisible(true)**
  - Method **setSize**
    - Set a window's size else only title bar will appear

Menu

- Menus
  - Important part of GUIs
  - Perform actions without cluttering GUI
  - Attached to objects of classes that have method **setJMenuBar**
    - **JFrame** and **JApplet**
  - **ActionEvent**s
- Classes used to define menus
  - **JMenuBar** - container for menus, manages menu bar
  - **JMenuItem** - manages menu items
    - Menu items - GUI components inside a menu
    - Can initiate an action or be a submenu
    - Method **isSelected**

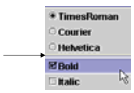
- Classes used to define menus (continued)
  - **JMenu** - manages menus
    - Menus contain menu items, and are added to menu bars
    - Can be added to other menus as submenus
    - When clicked, expands to show list of menu items
  - **JCheckBoxMenuItem** (extends **JMenuItem**)
    - Manages menu items that can be toggled
    - When selected, check appears to left of item
  - **JRadioButtonMenuItem** (extends **JMenuItem**)
    - Manages menu items that can be toggled
    - When multiple **JRadioButtonMenuItem**s are part of a group (**ButtonGroup**), only one can be selected at a time
    - When selected, filled circle appears to left of item

- Mnemonics
  - Quick access to menu items (File)
    - Can be used with classes that have subclass `javax.swing.AbstractButton`
  - Method `setMnemonic`

```
JMenu fileMenu = new JMenu( "File" )
fileMenu.setMnemonic( 'F' );
```

    - Press `Alt + F` to access menu
- Methods
  - `setSelected( true )`
    - Of class `AbstractButton`
    - Sets button/item to selected state

- Methods (continued)
  - **addSeparator()**
    - Of class **JMenu**
    - Inserts separator line into menu



- Dialog boxes
  - Modal - No other window can be accessed while it is open (default)
    - Modeless - other windows can be accessed

Dialogs

- **JOptionPane.showMessageDialog(parentWindow, text, title, messageType)**
- **parentWindow** - determines where dialog box appears
  - **null** - displayed at center of screen
  - Window specified - dialog box centered horizontally over parent

```
JOptionPane.showMessageDialog( MenuTest.this,
                               "This is an example\nof using menus",
                               "About", JOptionPane.PLAIN_MESSAGE );
```

Swing A

- Using menus
  - Create menu bar
    - Set menu bar for **JFrame**
      - **setJMenuBar(myBar);**
  - Create menus
    - Set Mnemonics
  - Create menu items
    - Set Mnemonics
    - Set event handlers
  - If using **JRadioButtonMenuItem**s
    - Create a group: **myGroup = new ButtonGroup();**
    - Add **JRadioButtonMenuItem**s to the group

Swing A

- Add menu items to appropriate menus
  - `myMenu.add( myItem );`
  - Insert separators if necessary: `myMenu.addSeparator();`
- If creating submenus, add submenu to menu
  - `myMenu.add( mySubMenu );`
- Add menus to menu bar
  - `myMenuBar.add( myMenu );`