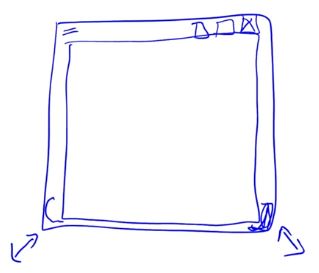
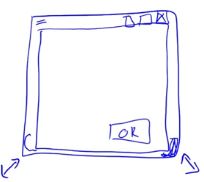
**GRAPHICAL USER INTERFACE (GUI)**

Whenever you do the gui, this is how you do it:

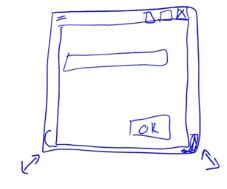
* There is a frame class, it will give you frame.



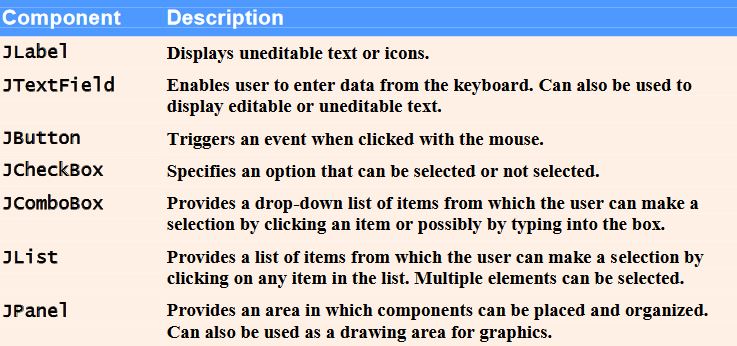
* You can add stuff for example OK button in it. OK button is another class. You instantiate it and you add it to frame bc frame has a method that accepts this kind of adds.



* Or you can add a field to read data from the user.



This is basically how you do GUI. You make your frame, take your frame, extend the frame class if you like and you keep adding button kinda components in it.



**GUI and Graphics Case Study: Creating Simple Drawings**

• Java’s coordinate system

* Defined by x-coordinates and y-coordinates

• Also known as horizontal and vertical coordinates

• Are measured along the x-axis and y-axis

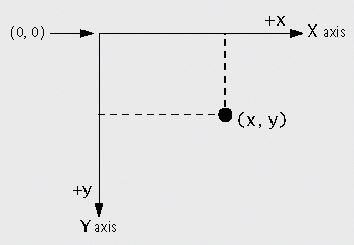
* Coordinate units are measured in pixels

• Graphics class from the java.awt package

* Provides methods for drawing text and shapes

• JPanel class from the javax.swing package

* Provides an area on which to draw



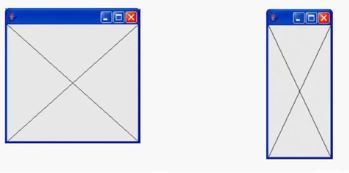
**CHECK DrawPanel.java, DrawPanelTest.java in order**

The JPanel Class

* Every JPanel has a paintComponent method
  + paintComponent is called whenever the system needs to display the JPanel
* getWidth and getHeight methods
  + Return the width and height of the JPanel, respectively
* drawLine method
  + Draws a line from the coordinates defined by its first two arguments to the coordinates defined by its second two arguments

JFrame class from the javax.swing package

* Allows the programmer to create a window
* setDefaultCloseOperation method
  + Pass JFrame.EXIT\_ON\_CLOSE as its argument to set the application to terminate when the user closes the window
* add method
  + Attaches a JPanel to the JFrame
* setSize method
  + Sets the width (first argument) and height (second argument) of the JFrame



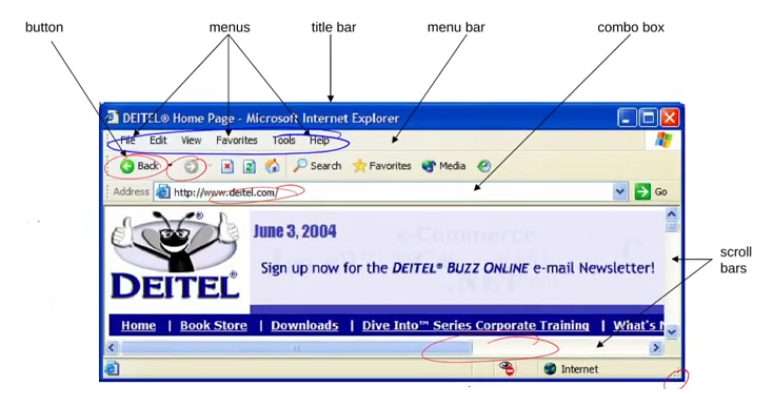
You can move the window around or resize it.

When you resize it, cross is properly drown bc OS knows this window has to be redrawn when you resize it.

It calls paintComponent method of your class through polymorphism. All the members of a frame are used to call paintComponent.

**Event Based Programming** : If this event occurs (pressing the button (I don’t care who is pressing my button, how it is pressed (by mouse click, pen, finger, keyboard…), when it is pressed) ), call this method. That method is automatically run by the whole java system (GUI management libraries). I don’t care about those libraries as a user.

**GUI COMPONENTS**



**Simple GUI-Based Input/Output with JOptionPane**

Dialog boxes

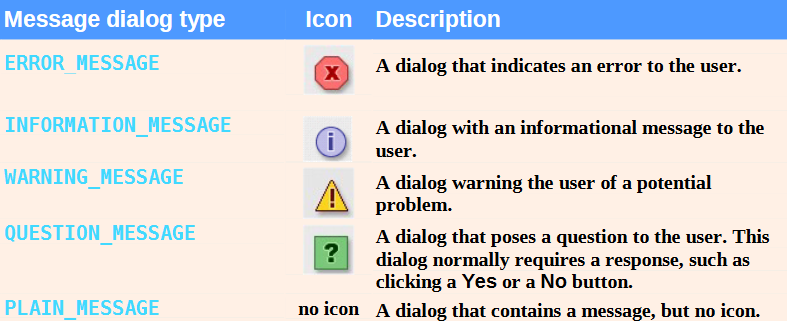
* Used by applications to interact with the user
* Provided by Java’s JOptionPane class
  + Contains input dialogs and message dialogs

**CHECK Addition.java**

We have used already designed window just for now.

Sample dialogue for Addition.java:





**Overview of Swing Components**

Swing GUI components

* Declared in package javax.swing
* Most are pure Java components
* Part of the Java Foundation Classes (JFC)

Swing components are implemented in Java, so they are more portable and flexible than the original Java GUI components from package java.awt, which were based on the GUI components of the underlying platform. For this reason, Swing GUI components are generally preferred.

• Lightweight components

* Not tied directly to GUI components supported by underlying platform

• Heavyweight components

* Tied directly to the local platform
* AWT components
* Some Swing components

**Superclasses of Swing’s Lightweight GUI Components**

• Class Component (package java.awt)

– Subclass of Object

– Declares many behaviors and attributes common to GUI components

• Class Container (package java.awt)

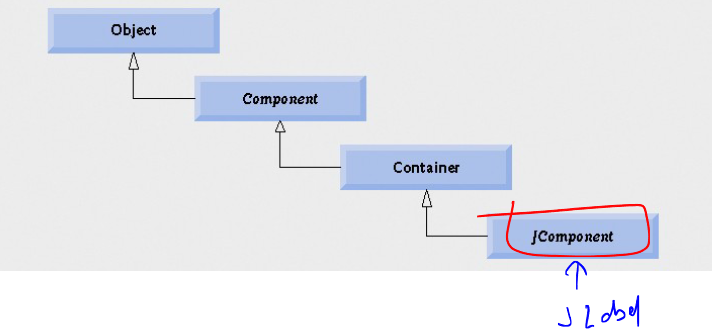
– Subclass of Component

– Organizes Components

• Class JComponent (package javax.swing)

– Subclass of Container

– Superclass of all lightweight Swing components



*Common superclasses of many of the Swing components.*

• Common lightweight component features

– Pluggable look-and-feel to customize the appearance of components

– Shortcut keys (called mnemonics)

– Common event-handling capabilities

– Brief description of component’s purpose (called tool tips)

– Support for localization

* localization is: When you develop a program for Turkish, but you like to sell the same program in USA. So you have to change all the strings to English. Each time you don’t have to modify the program thanks to localization.
* Localization does this: You write your program independent of the language. Later after you are done with your program, you may add your strings, whether they are Turkish or English or German, to your program later. So whenever you need to port your program to some other language, you don’t modify your program, just change your localization string.

**• Class JFrame**

– Most windows are an instance or subclass of this class

– Provides title bar

– Provides buttons to minimize, maximize and close the application

**• Label**

– Text instructions or information stating the purpose of each component

– Created with class JLabel

• Laying out containers

– Determines where components are placed in the container

– Done in Java with layout managers

• One of which is class FlowLayout

– Set with the setLayout method of class JFrame

**CHECK LabelFrame.java, LabelTest.java**



Creating and Attaching label1

• Method setToolTipText of class JComponent

– Specifies the tool tip

• Method add of class Container

– Adds a component to a container

*If you do not explicitly add a GUI component to a container, the GUI component will not be displayed when the container appears on the screen.*

Creating and Attaching label2

• Interface Icon

– Can be added to a JLabel with the setIcon method

– Implemented by class ImageIcon

• Interface SwingConstants

– Declares a set of common integer constants such as those used to set the alignment of components

– Can be used with methods setHorizontalAlignment and setVerticalAlignment

Creating and Attaching label3

• Other JLabel methods

– getText and setText

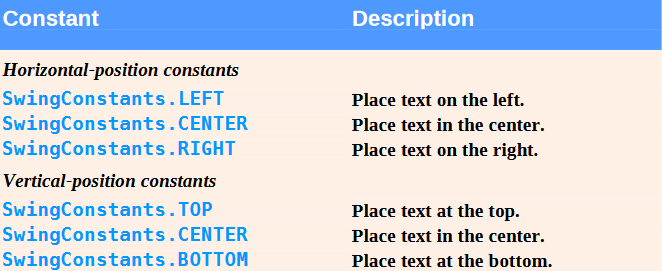
• For setting and retrieving the text of a label

– getIcon and setIcon

• For setting and retrieving the icon displayed in the label

– getHorizontalTextPosition and setHorizontalTextPosition

• For setting and retrieving the horizontal position of the text displayed in the label



**Creating and Displaying a LabelFrame Window**

• Other JFrame methods

– setDefaultCloseOperation

• Dictates how the application reacts when the user clicks the close button

– setSize

• Specifies the width and height of the window

– setVisible

• Determines whether the window is displayed (true) or not (false)

**Text Fields and an Introduction to Event Handling with Nested Classes**

• GUIs are event-driven (getEvents(), dispatchEvents() 🡪 do these 2 with infinite loop when you run a GUI program)

– A user interaction creates an event

• Common events are clicking a button, typing in a text field, selecting an item from a menu, closing and window and moving the mouse

– OS get those events and asks your dispatcher of your GUI: “Can anybody handle these events?”. If there is sb handling that event, that component handles the event.

– The event causes a call to a method called an event handler

• Class JTextComponent

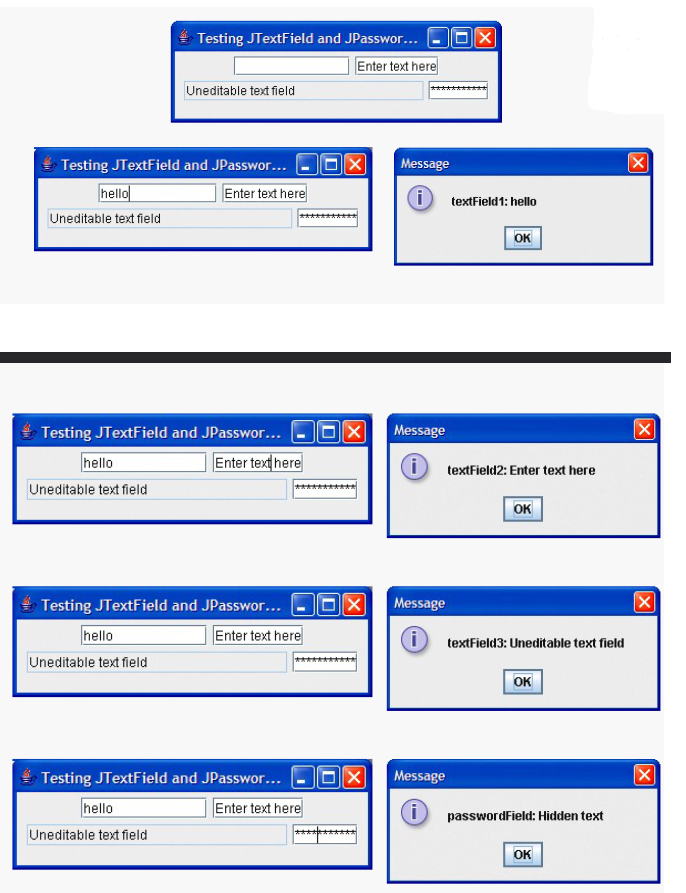
– Superclass of JTextField

• Superclass of JPasswordField

– Adds echo character to hide text input in component

– Allows user to enter text in the component when component has the application’s focus

**CHECK TextFieldFrame.java, TextFieldTest.java**



**Steps Required to Set Up Event Handling for a GUI Component**

• Several coding steps are required for an application to respond to events

– Create a class for the event handler

– Implement an appropriate event-listener interface

– Register the event handler

**Using a Nested Class to Implement an Event Handler**

• Top-level classes

– Not declared within another class

• Nested classes

– Declared within another class

– Non-static nested classes are called inner classes

• Frequently used for event handling

• An inner class is allowed to directly access its top-level class’s variables and methods, even if they are private.

• JTextFields and JPasswordFields

– Pressing enter within either of these fields causes an ActionEvent

• Processed by objects that implement the ActionListener interface

**Registering the Event Handler for Each Text Field**

• Registering an event handler

– Call method addActionListener to register an ActionListener object

– ActionListener listens for events on the object

*The event listener for an event must implement the appropriate event-listener interface.*

*Forgetting to register an event-handler object for a particular GUI component’s event type causes events of that type to be ignored.*

**Details of Class TextFieldHandler’s actionPerformed Method**

• Event source

– Component from which event originates

– Can be determined using method getSource

– Text from a JTextField can be acquired using getActionCommand

– Text from a JPasswordField can be acquired using getPassword

**Common GUI Event Types and Listener Interfaces**

• Event types

– All are subclasses of AWTEvent

– Some declared in package java.awt.event

– Those specific to Swing components declared in javax.swing.event

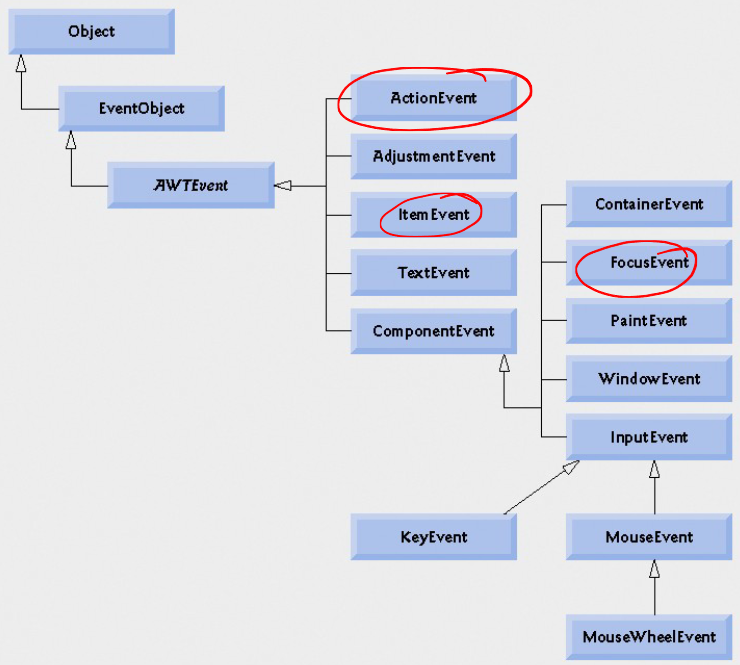
• Delegation event model

– Event source is the component with which user interacts

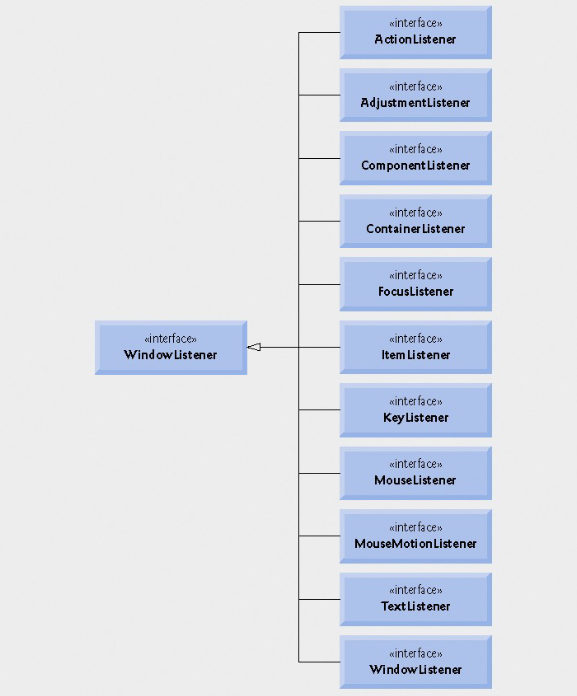
– Event object is created and contains information about the event that happened

– Event listener is notified when an event happens

Some event classes of package java.awt.event



Some common event-listener interfaces of package java.awt.event



If you need to handle MouseEvents you need to implement MouseListener, etc.

You should read the documentation for the listener type.

**How Event Handling Works?**

• Remaining questions

– How did the event handler get registered?

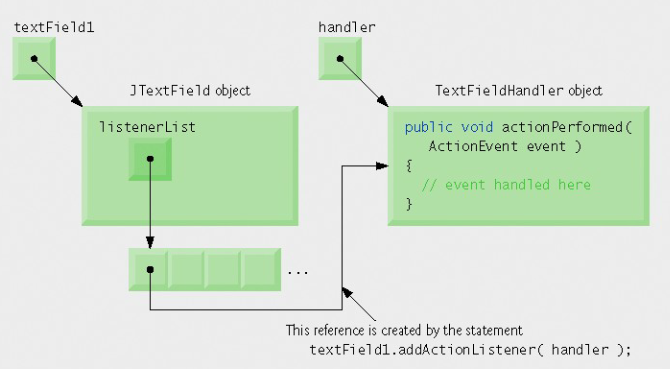
– How does the GUI component know to call actionPerformed rather than some other event- handling method?

**Registering Events**

• Every JComponent has instance variable listenerList

– Object of type EventListenerList

– Maintains references to all its registered listeners



Each component has a listenerList. When you register an event handler, it is added to this list. If sth happens with that compenent this list is scanned to find if a handler exist. If a handler exists, that handler is called. Otherwise that event is ignored.

**Event-Handler Invocation**

• Events are dispatched to only the event listeners that match the event type

– Events have a unique event ID specifying the event type

•MouseEvents are handled by MouseListeners and MouseMotionsListeners

•KeyEvents are handled by KeyListeners

**JButton**

• Button

– Component user clicks to trigger a specific action

– Can be command button, check box, toggle button or radio button

– Button types are subclasses of class AbstractButton

• Command button

– Generates an ActionEvent when it is clicked

– Created with class JButton

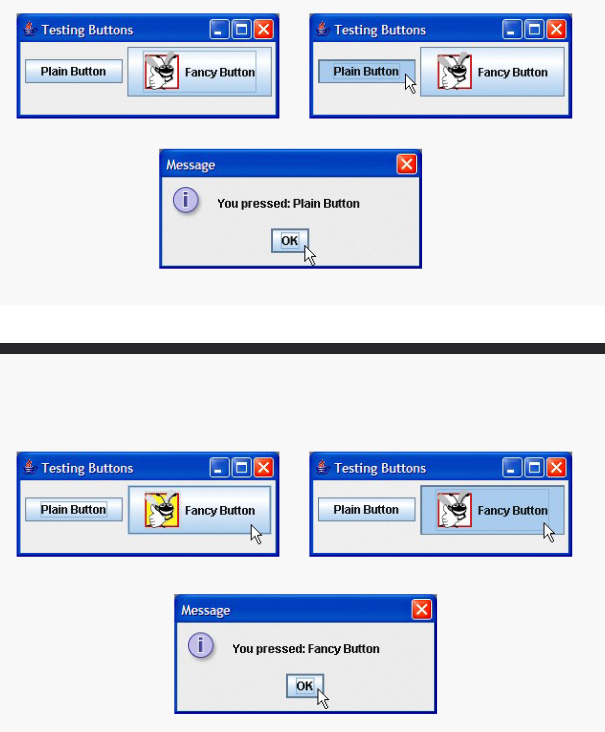
– Text on the face of the button is called button label

*Having more than one JButton with the same label makes the JButtons ambiguous to the user. Provide a unique label for each button.*

*Diagram

Description automatically generated*

**CHECK ButtonFrame.java, ButtonTest.java**



• JButtons can have a rollover icon

– Appears when mouse is positioned over a button

– Added to a JButton with method setRolloverIcon

*Because class AbstractButton supports displaying text and images on a button, all subclasses of AbstractButton also support displaying text and images.*

*Using rollover icons for JButtons provides users with visual feedback indicating that when they click the mouse while the cursor is positioned over the button, an action will occur.*

*When used in an inner class, keyword this refers to the current inner-class object being manipulated. An inner-class method can use its outer-class object’s this by preceding this with the outer-class name and a dot, as in ButtonFrame.this.*

**Buttons That Maintain State**

• State buttons

– Swing contains three types of state buttons

– JToggleButton, JCheckBox and JRadioButton

– JCheckBox and JRadioButton are subclasses of JToggleButton

**JCheckBox**

• JCheckBox

– Contains a check box label that appears to right of check box by default

– Generates an ItemEvent when it is clicked

• ItemEvents are handled by an ItemListener

• Passed to method itemStateChanged

– Method isSelected returns whether check box is selected (true) or not (false)

**CHECK CheckBoxFrame.java, CheckBoxTest.java**

Graphical user interface, application

Description automatically generated

**JRadioButton**

• JRadioButton

– Has two states – selected and unselected

– Normally appear in a group in which only one radio button can be selected at once

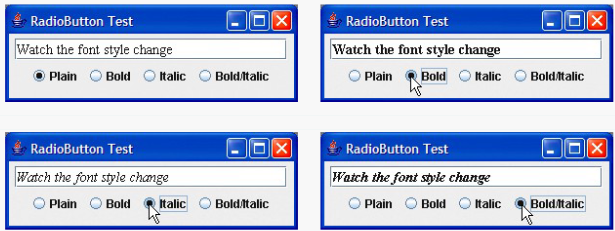
• Group maintained by a ButtonGroup object

– Declares method add to add a JRadioButton to group

– Usually represents mutually exclusive options

*Adding a ButtonGroup object (or an object of any other class that does not derive from Component) to a container results in a compilation error.*

**CHECK RadioButtonFrame.java, RadioButtonTest.java**



**JComboBox and Using an Anonymous Inner Class for Event Handling**

• Combo box

– Also called a drop-down list

– Implemented by class JComboBox

– Each item in the list has an index

– setMaximumRowCount sets the maximum number of rows shown at once

– JComboBox provides a scrollbar and up and down arrows to traverse list

Set the maximum row count for a JComboBox to a number of rows that prevents the list from expanding outside the bounds of the window in which it is used. This configuration will ensure that the list displays correctly when it is expanded by the user.

**Using an Anonymous Inner Class for Event Handling**

• Anonymous inner class

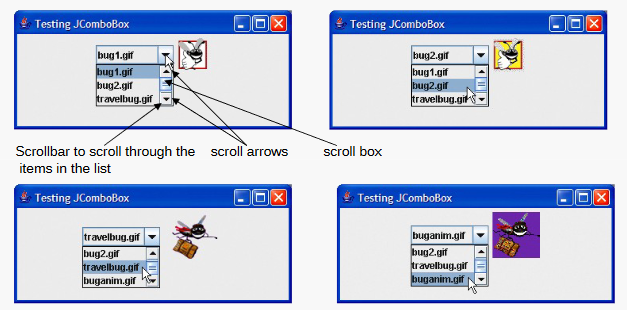
– Special form of inner class

– Declared without a name

– Typically appears inside a method call

– Has limited access to local variables

**CHECK ComboBoxFrame.java, ComboBoxTest.java**



An anonymous inner class declared in a method can access the instance variables and methods of the top-level class object that declared it, as well as the method’s final local variables, but cannot access the method’s non-final variables.

Like any other class, when an anonymous inner class implements an interface, the class must implement every method in the interface.

**JList**

• List

– Displays a series of items from which the user may select one or more items

– Implemented by class JList

– Allows for single-selection lists or multiple-selection lists

– A ListSelectionEvent occurs when an item is selected

• Handled by a ListSelectionListener and passed to method valueChanged

**CHECK ListFrame.java, ListTest.java**

Graphical user interface

Description automatically generated

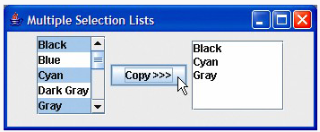
**Multiple-Selection Lists**

• Multiple-selection list

– Enables users to select many items

– Single interval selection allows only a continuous range of items

– Multiple interval selection allows any set of elements to be selected



**CHECK MultipleSelectionFrame.java, MultipleSelectionTest.java**

**Mouse Event Handling**

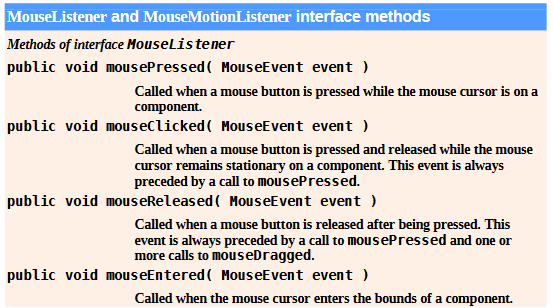
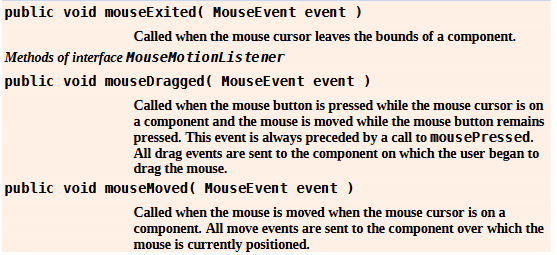
• Mouse events

– Create a MouseEvent object

– Handled by MouseListeners and MouseMotionListeners

– MouseInputListener combines the two interfaces

– Interface MouseWheelListener declares method mouseWheelMoved to handle MouseWheelEvents



Method calls to mouseDragged and mouseReleased are sent to the MouseMotionListener for the Component on which a mouse drag operation started. Similarly, the mouseReleased method call at the end of a drag operation is sent to the MouseListener for the Component on which the drag operation started.

**CHECK MouseTrackerFrame.java, MouseTracker.java**

A screenshot of a computer

Description automatically generated with medium confidence

**Adapter Classes**

• Adapter class

– Implements event listener interface

– Provides default implementation for all event-handling methods

When a class implements an interface, the class has an “is a” relationship with that interface. All direct and indirect subclasses of that class inherit this interface. Thus, an object of a class that (e.g., an object of a subclass of MouseAdapter is a MouseListener).

Since there are 5 methods to implement in MouseListener, it is too much typing if you just use mouseEntered. I don’t like the fact that I have to implement all 5 methods. To prevent this from happening, they developed adapter classes. For example they implement all the mouse handling methods by providing empty methods.

**Extending MouseAdapter**

• MouseAdapter

– Adapter class for MouseListener and MouseMotionListener interfaces

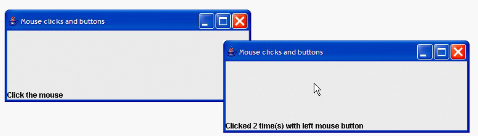
– Extending class allows you to override only the methods you wish to use

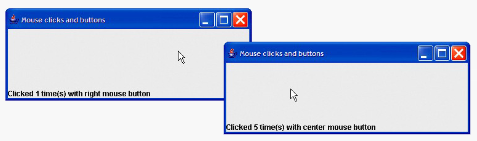
If you extend an adapter class and misspell the name of the method you are overriding, your method simply becomes another method in the class. This is a logic error that is difficult to detect, since the program will call the empty version of the method inherited from the adapter class.



*Event-adapter classes and the interfaces they implement in package java.awt.event.*

**CHECK MouseDetailsFrame.java, MouseDetails.java**

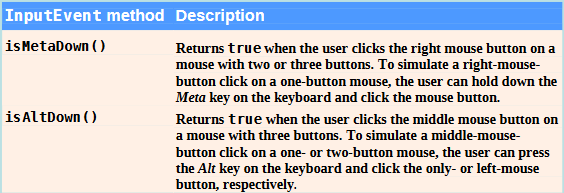




**JPanel Subclass for Drawing with the Mouse**

• Overriding class JPanel

– Provides a dedicated drawing area



*InputEvent methods that help distinguish among left-, center- and right-mouse-button clicks.*

**Method paintComponent**

• Method paintComponent

– Draws on a Swing component

– Overriding method allows you to create custom drawings

– Must call superclass method first when overridden

*Most Swing GUI components can be transparent or opaque. If a Swing GUI component is opaque, its background will be cleared when its paintComponent method is called. Only opaque components can display a customized background color. JPanel objects are opaque by default.*

*In a JComponent subclass’s paintComponent method, the first statement should always be a call to the superclass’s paintComponent method to ensure that an object of the subclass displays correctly.*

*If an overridden paintComponent method does not call the superclass’s version, the subclass component may not display properly. If an overridden paintComponent method calls the superclass’s version after other drawing is performed, the drawing will be erased.*

**Defining the Custom Drawing Area**

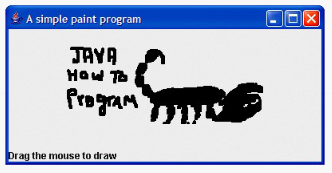
• Customized subclass of JPanel

– Provides custom drawing area

– Class Graphics is used to draw on Swing components

– Class Point represents an x-y coordinate

**CHECK PaintPanel.java, Painter.java**



Calling repaint for a Swing GUI component indicates that the component should be refreshed on the screen as soon as possible. The background of the GUI component is cleared only if the component is opaque. JComponent method setOpaque can be passed a boolean argument indicating whether the component is opaque (true) or transparent (false).

Drawing on any GUI component is performed with coordinates that are measured from the upper-left corner (0, 0) of that GUI component, not the upper-left corner of the screen.

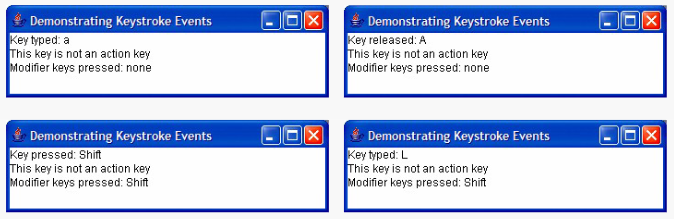
**Key-Event Handling**

•KeyListener interface

– For handling KeyEvents

– Declares methods keyPressed, keyReleased and keyTyped

**CHECK KeyDemoFrame.java, KeyDemo.java**

Graphical user interface, application

Description automatically generated

**Layout Managers**

• Layout managers

– Provided to arrange GUI components in a container

– Provide basic layout capabilities

– Implement the interface LayoutManager

Most Java programming environments provide GUI design tools that help a programmer graphically design a GUI; the design tools then write the Java code to create the GUI. Such tools often provide greater control over the size, position and alignment of GUI components than do the built-in layout managers.

It is possible to set a Container’s layout to null, which indicates that no layout manager should be used. In a Container without a layout manager, the programmer must position and size the components in the given container and take care that, on resize events, all components are repositioned as necessary. A component’s resize events can be processed by a ComponentListener.

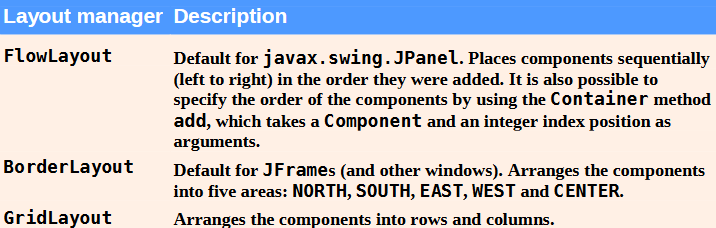
**FlowLayout**

• FlowLayout

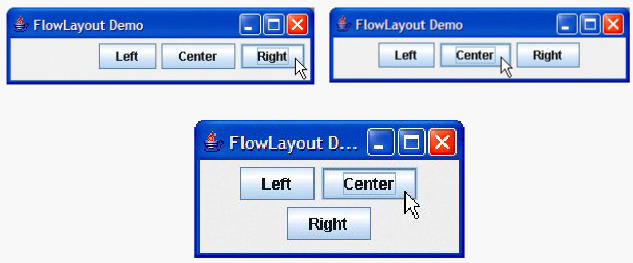
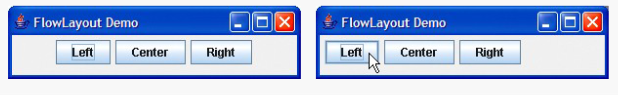
– Simplest layout manager

– Components are placed left to right in order they are added

– Components can be left aligned, centered or right aligned



**CHECK FlowLayoutFrame.java, FlowLayoutDemo.java**



**BorderLayout**

• BorderLayout

– Arranges components into five regions – north, south, east, west and center

– Implements interface LayoutManager2

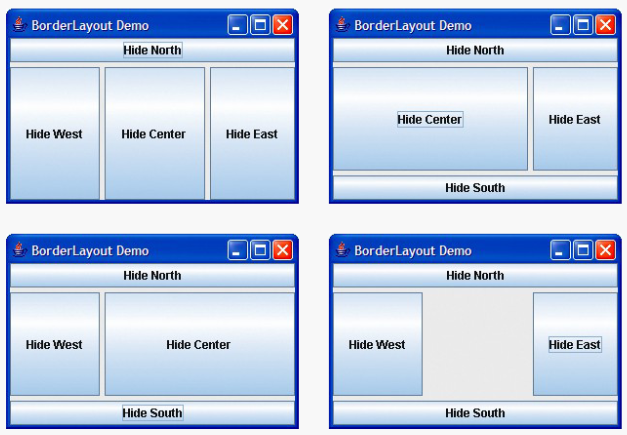
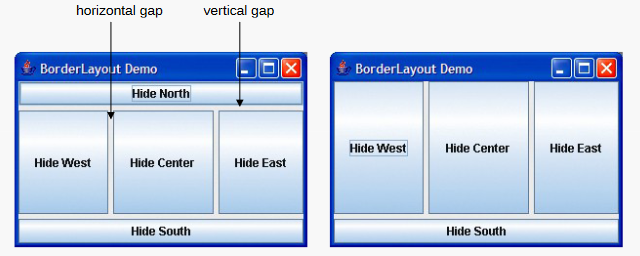
– Provides horizontal gap spacing and vertical gap spacing

Each container can have only one layout manager. Separate containers in the same application can use different layout managers.

If no region is specified when adding a Component to a BorderLayout, the layout manager assumes that the Component should be added to region BorderLayout.CENTER.

When more than one component is added to a region in a BorderLayout, only the last component added to that region will be displayed. There is no error that indicates this problem.

**CHECK BorderLayoutFrame.java, BorderLayoutDemo.java**



**GridLayout**

• GridLayout

– Divides container into a grid

– Every component has the same width and height

**CHECK GridLayoutFrame.java, GridLayoutDemo.java**



**Using Panels to Manage More Complex Layouts**

• Complex GUIs often require multiple panels to arrange their components properly

**CHECK PanelFrame.java, PanelDemo.java**



**JTextArea**

• JTextArea

– Provides an area for manipulating multiple lines of text

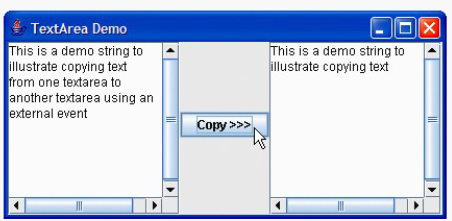
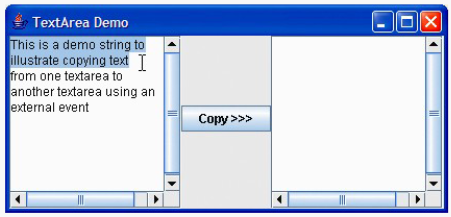
• Box container

– Subclass of Container

– Uses a BoxLayout layout manager

To provide line-wrapping functionality for a JTextArea, invoke JTextArea method setLine-Wrap with a true argument.

**CHECK TextAreaFrame.java, TextAreaDemo.java**



**JScrollPane Scrollbar Policies**

• JScrollPane has scrollbar policies

**– Horizontal policies**

• Always (HORIZONTAL\_SCROLLBAR\_ALWAYS)

• As needed (HORIZONTAL\_SCROLLBAR\_AS\_NEEDED)

• Never (HORIZONTAL\_SCROLLBAR\_NEVER)

**– Vertical policies**

• Always (VERTICAL\_SCROLLBAR\_ALWAYS)

• As needed (VERTICAL\_SCROLLBAR\_AS\_NEEDED)

• Never (VERTICAL\_SCROLLBAR\_NEVER)