Graphic User Interface (GUI) History , Development and Application

NAME: TARUN GOYAL

ROLL NO.: 164130

DATE OF PRESENTATION: 27-FEB-2020

Contents

GUI Introduction

Development of GUI (History)

Building GUI with JAVA

GUI Component in JAVA

Event Driven GUI

Event Listener

Layout Manager

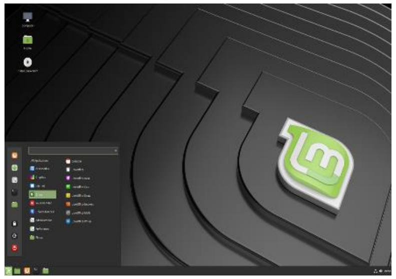
GUI Introduction

- The graphical user interface (GUI) is a form of user interface that allows users to interact with electronic devices through graphical icons and audio indicator such as primary notation, instead of text-based user interfaces, typed command labels or text navigation. GUI’s were introduced in reaction to the perceived steep learning curve of command-line interfaces (CLI’s),which require commands to be typed on a computer keyboard.

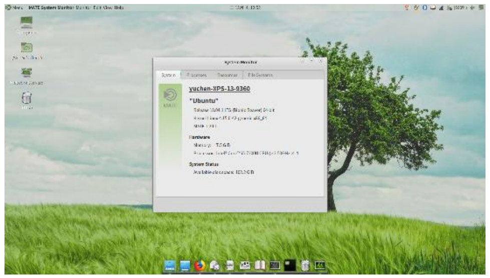
- The actions in a GUI are usually performed through direct manipulation of the graphical elements.

- Beyond computers, GUI’s are used in many handheld mobile devices such as mp3 players, portable media players, gaming devices, smartphones and smaller household, office and industrial controls.

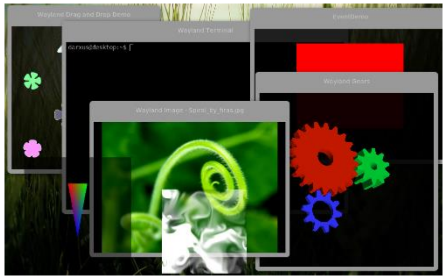
GUI Examples



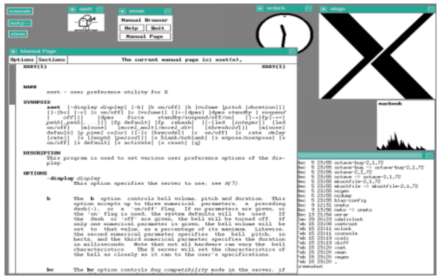
Cinnamon



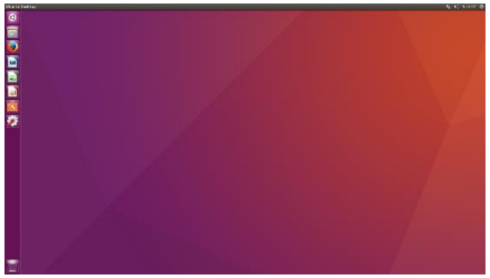
Mate



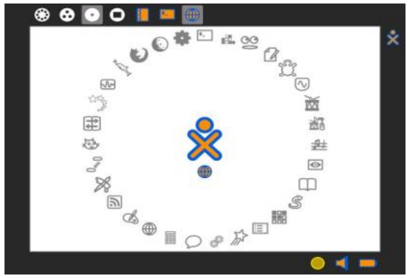
Windows



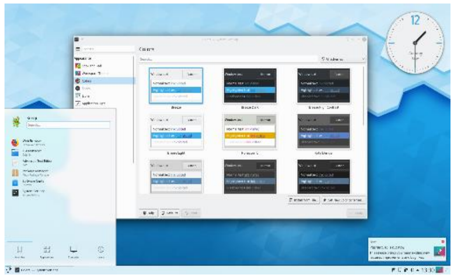
X Window System



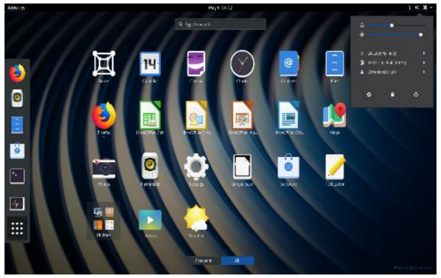
Ubuntu



Sugar



KDE Plasma



GNOME Shell

Why GUI is better than CLI

Topic	CLI	GUI
Ease	Due to a higher degree of memorization and familiarity needed	users tend to learn how to use a GUI faster than a CLI
Control	Users have a good bit of control over both the file and operating systems	A GUI offers a lot of access to files, software features, and the OS
Multi-Tasking	Although CLI are capable of multitasking, do not offer the same ease	have windows that enable view, control, manipulate, and toggle through multiple
Speed	Command line users only need to utilize a keyboard to navigate the interface, often resulting in faster speed	taking hand to the keyboard to use the mouse pointer is slower than using CLI
Resources	A computer that is only using the command line takes a lot less of the computer's system resources than GUI	requires more system resources because of the elements that require loading iocns
Scripting	mostly requires users to already know scripting commands and syntax, making it difficult for new	Creating scripts become much easier with the help of programming software
Remote Access	accessing another computer or device over a network, a user can manipulate the device or its files	access another computer or server is easy to navigate with little experience
Diversity	After you've learned how to navigate and use a command line it's not going to change as much as GUI	has a different design and structure when it comes to performing different tasks
Strain	is often very basic and can be more of a strain on a user's vision	use of shortcut keys and more frequent movement of hand positions

Development of GUI

1945

- **RADAR**
- First data(values) to graphic converter
- Non- Interactive

1963

- **SKETCHPAD**
- Developed by Ivan Sutherland, MIT
- First Completely Graphical User Interface
- Object Oriented Programming

Development of GUI (Contd...)

1968

- **THE MOTHER OF ALL DEMOS**
  - NLS : On-Line System
  - Word Processing
  - Multiple Windows
  - Network Collaboration

1973

- **Xerox PARC**
  - First operational Alto Computer
  - 3 button mouse
  - Bit-mapped display
  - The use of graphical windows
  - Ethernet network

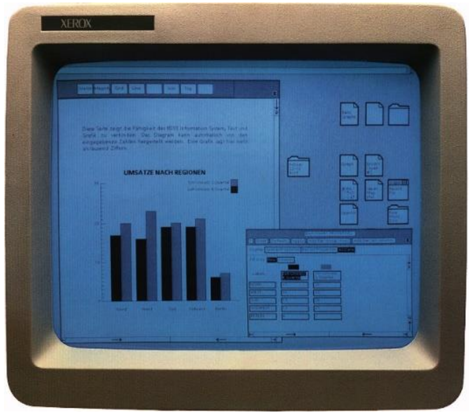
Development of GUI (Cont..)

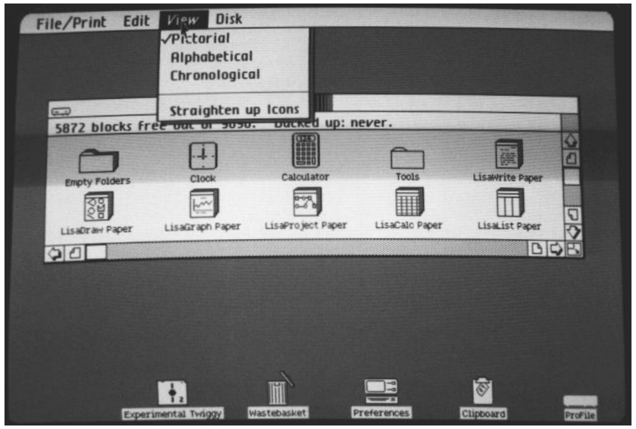
1981

- **XEROX STAR**
- Double-clickable icons
- Overlapping windows
- Dialog boxes
- A 1024*768 monochrome display

1983

- **Apple LISA**
- File Finder
- Pull down menus and menu bars
- Icons of all Files
- Layered Windows





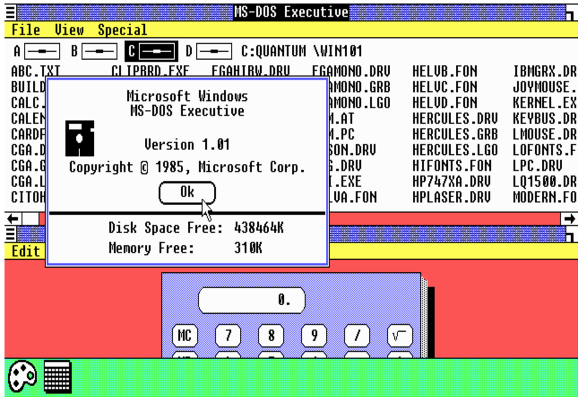
Development of GUI (Contd...)

1984

- **<u>Apple Macintosh (Mac)</u>**
  - Drag & Drop
  - Double -clicking
  - Commercial version

1985

- **<u>Microsoft Windows</u>**
  - The first version of Windows
  - Windows can not be overlapped, but are instead "tiled".
  - Windows are not allowed to cover an area at the bottom of the screen that is reserved for "iconized" programs.



```
ABC.TXT        CLIPBRD.EXE   EGAHIRW.DRV   EGAMONO.DRV   HELVB.FON      IBMGRX.DR
BUILD                                      AMONO.GRB     HELVC.FON      JOYMOUSE.
CALC.                                      AMONO.LGO     HELVD.FON      KERNEL.EX
CALEN                                      M.AT          HERCULES.DRV   KEYBUS.DR
CARDF                                      M.PC          HERCULES.GRB   LMOUSE.DR
CGA.D                                      SON.DRV       HERCULES.LGO   LOFONTS.F
CGA.G                                      G.DRV         HIFONTS.FON    LPC.DRV
CGA.L                                      I.EXE         HP747XA.DRV    LQ1500.DR
CITOH                                      LVA.FON       HPLASER.DRV    MODERN.FO
```

```
Microsoft Windows
MS-DOS Executive

Version 1.01
Copyright © 1985, Microsoft Corp.

Disk Space Free:  438464K
Memory Free:      310K
```

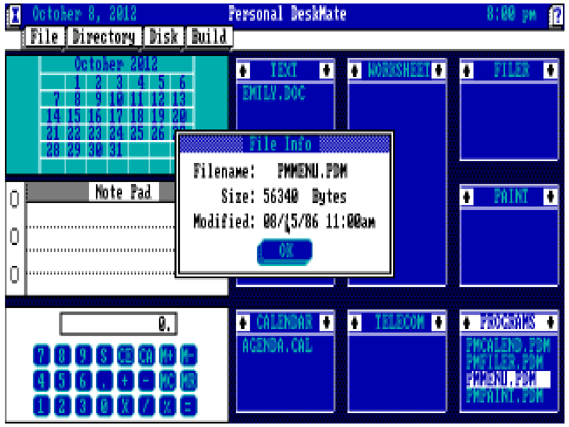
Development of GUI (Contd…)

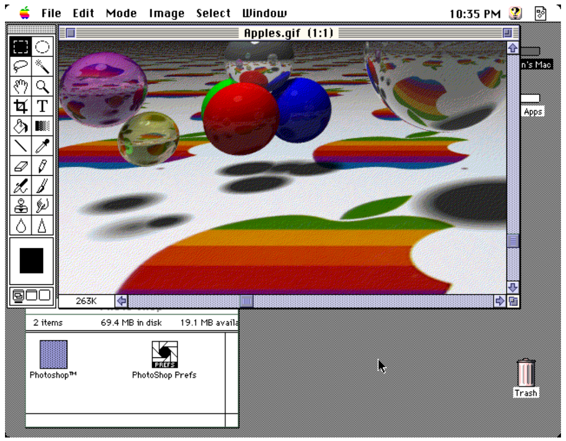
1986

- **Personal DeskMate**
- Released by TANDY for TANDY 1000 EX
- This is the first graphical version of Tandy's previously text-based integrated office package

1987

- **Apple Macintosh 2**
- The first color macintosh
- 640*480*256 color with 24 bit color card available.

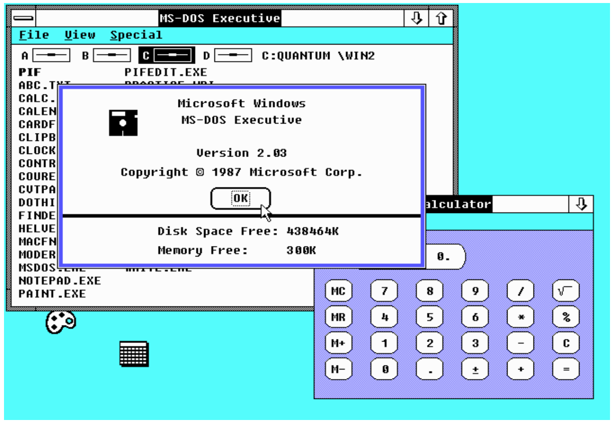




Development of GUI (Contd...)

1987

- **Windows 2.03**
- Microsoft releases the second version of Windows, version 2.03
- Finally has resizable / overlapping windows and new windowing controls



Microsoft Windows
MS-DOS Executive

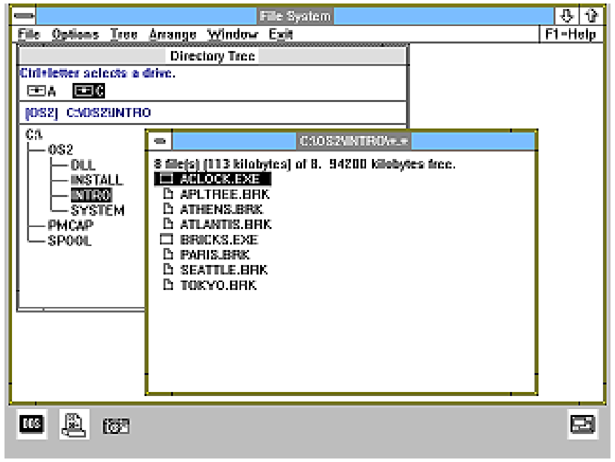
Version 2.03
Copyright © 1987 Microsoft Corp.

OK

Disk Space Free: 438464K
Memory Free: 300K

1988

- **IBM OS/2 1.1**
- Standard Edition (SE) which added a graphical user interface called Presentation Manager
- Written by Microsoft and looked like Windows 2

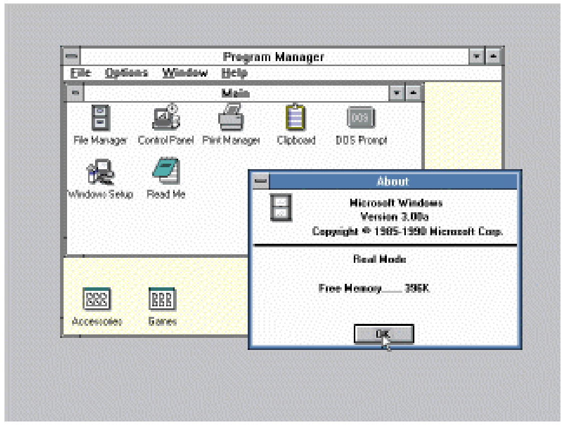


8 file(s) (113 kilobytes) of 8. 94200 kilobytes free.
ACLOCK.EXE
APLTREE.BRK
ATHENS.BRK
ATLANTIS.BRK
BRICKS.EXE
PARIS.BRK
SEATTLE.BRK
TOKYO.BRK

Development of GUI (Contd...)

1990

- **<u>Windows 3</u>**
- Program Manager shell



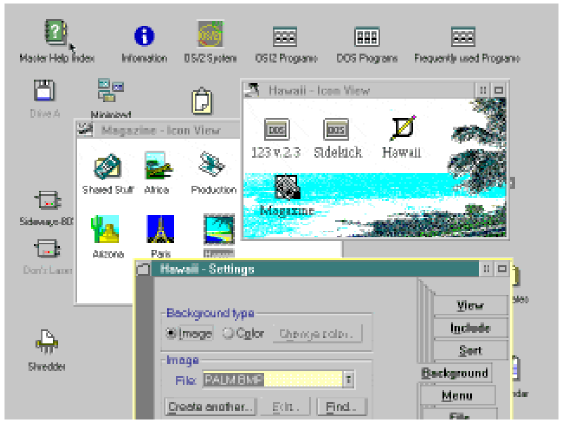
Microsoft Windows
Version 3.00a
Copyright © 1985-1990 Microsoft Corp.

Real Mode

Free Memory 396K

1992

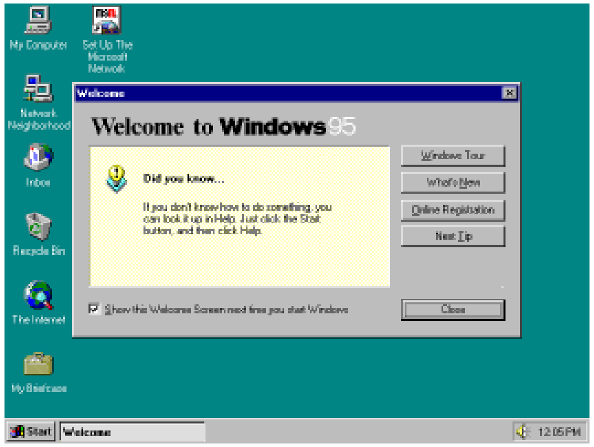
- **<u>IBM OS/2 2.0</u>**
- a true 32-bit OS
- Features a new "Workplace Shell"
- An object oriented user interface that is heavily integrated with the rest of the OS



Development of GUI (Contd...)

1995

- **<u>Windows 95</u>**
- A complete operating system rather than a GUI running on top of MS-DOS
- Task bar and start menu
- Build-in network support with dial-up for different protocols


Welcome to Windows 95

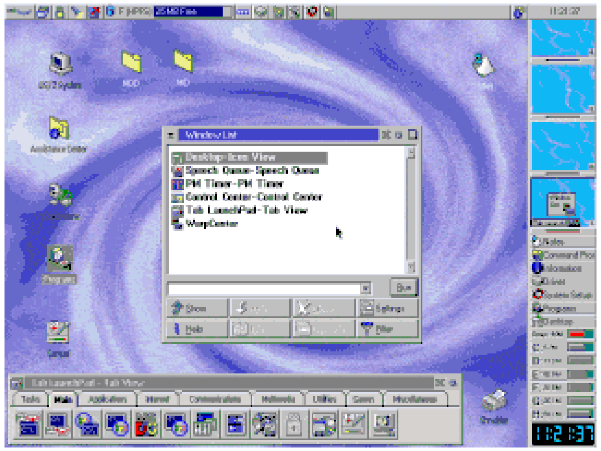
Did you know...

If you don't know how to do something, you can look it up in Help. Just click the Start button, and then click Help.

Show this Welcome Screen next time you start Windows

1996

- **<u>IBM OS/2 Warp 4</u>**
- Very stable, protected processes from each other
- Interchangeable file systems by **I**nstallable **F**ile **S**ystems in form of GUI



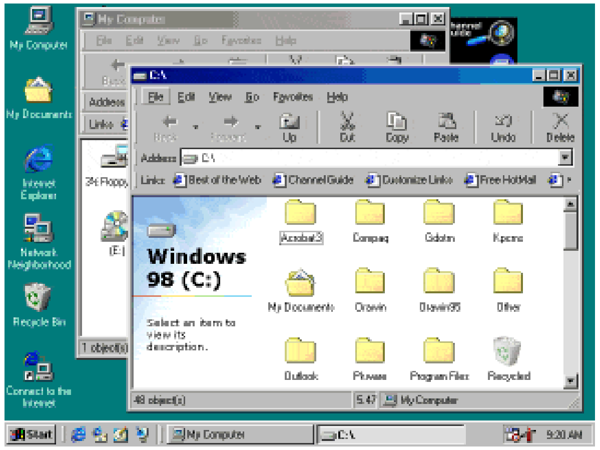
Development of GUI (Contd…)

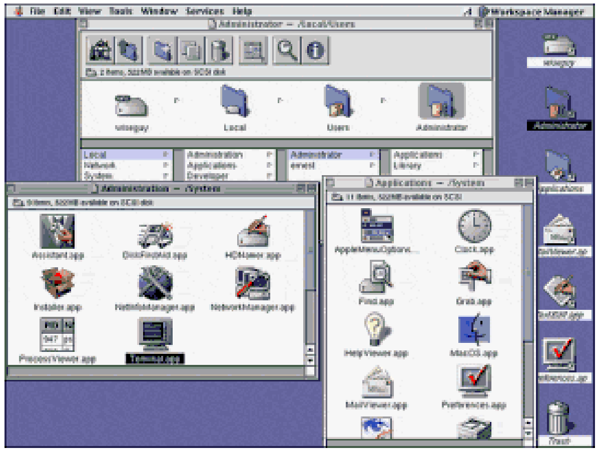
1998

- **Windows 98**
- Internet Explorer Web browser application takes over the role of the Windows shell, advertising right on the desktop, entire help system replaced by Internet Explorer

1999

- **Apple Mac OS X Server**
- A Unix based OS
- Macintosh GUI





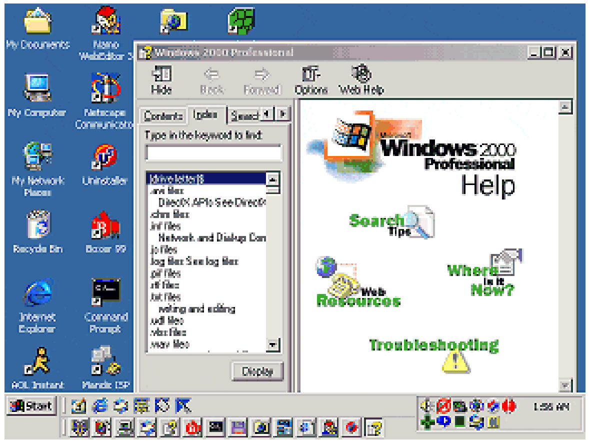
Development of GUI (Contd...)

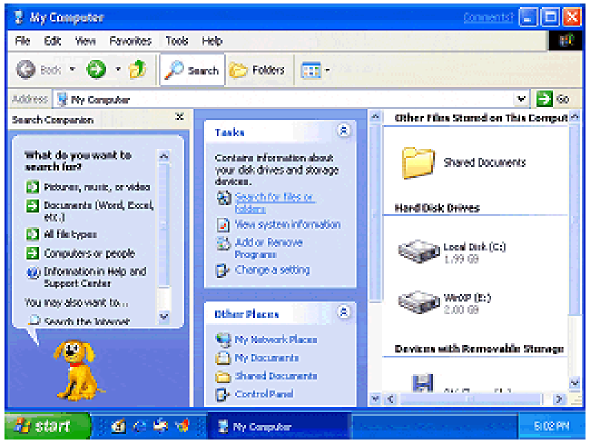
2000

- **<u>Microsoft Windows 2000</u>**
- AKA Windows NT 5
- The Internet Explorer web browser application finally takes over the Windows NT UI

2001

- **<u>Microsoft Windows XP</u>**
- "Tons of eye candy"
- "Product Activation" tethers XP to the existence of the Microsoft corporation.
- The dog from Microsoft Bob




My Computer
Search Companion
Tasks
Other Places
Other Files Stored on This Computer
Hard Disk Drives
Devices with Removable Storage
start

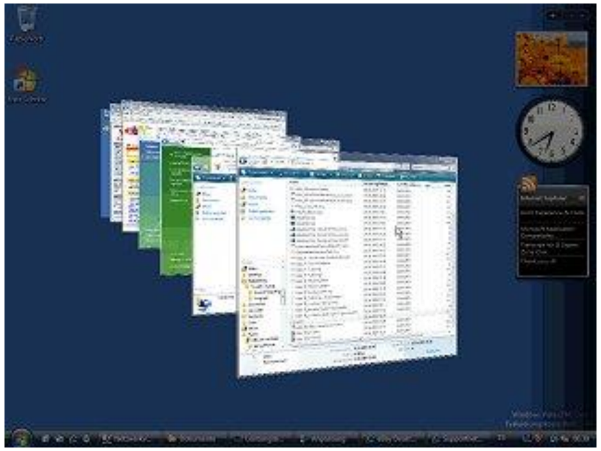
Development of GUI (Contd...)

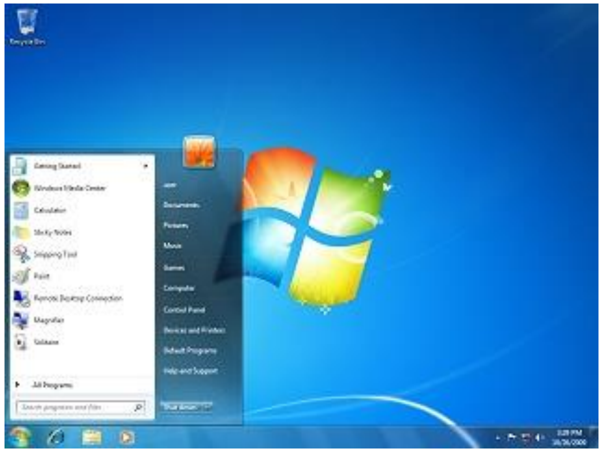
2007

- **<u>Microsoft Windows Vista</u>**
- 3D hardware-rendered user interface.
- Bundles IE 7,. irremovable as always.
- Increased Digital Restrictions Management that tries to prevent playback or duplicity of audio and video.

2009

- **<u>Microsoft Windows 7</u>**
- Relatively little difference over Vista
- You can "pin" icons to the Taskbar.
- Ribbons replace menus in some applications

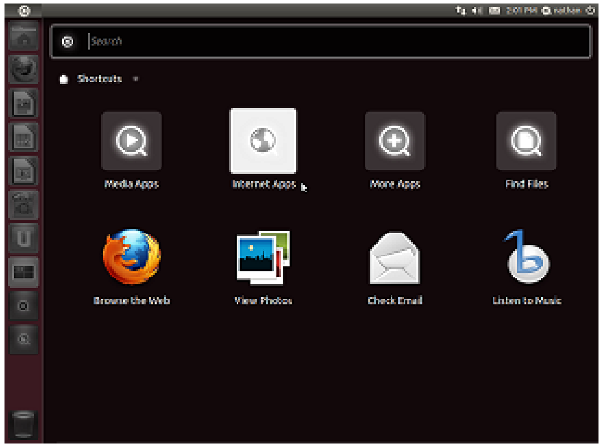




Development of GUI (Contd...)

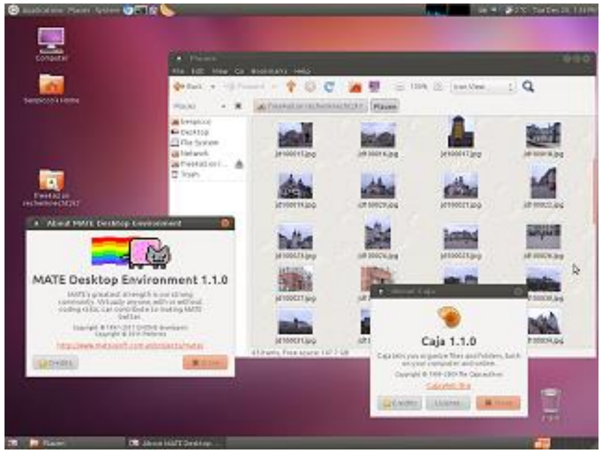
2011

- **Ubuntu 11.4**
- Removes the task bar.
- Launcher merges the role of the taskbar and the top panel's shortcuts.
- The Home Button and the "Dash" replace the Applications Menu



2011

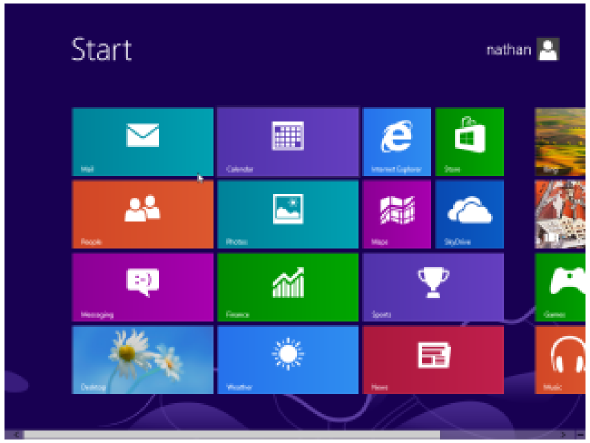
- **Mate**
- a fork of the GNOME 2
- A standard desktop user interface
- All the flexibility and configurability of GNOME 2.x



Development of GUI (Contd...)

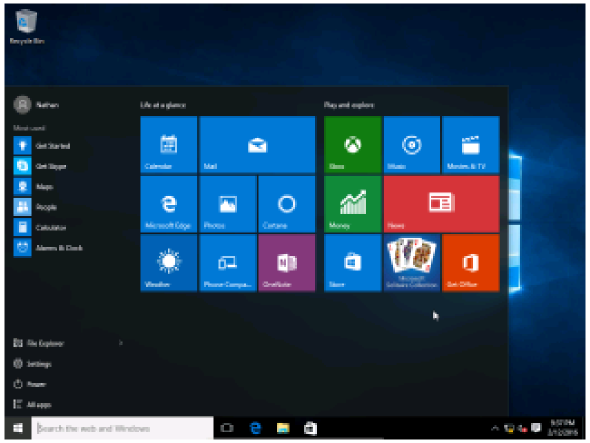
2012

- **<u>Microsoft Windows 8</u>**
  - "Apps" that run in a full screen DOS like mode ironically called "Modern UI"
  - Heavily promotes impractical "touch" navigation of the UI on desktop systems.
  - Removes the "Start" menu from desktop.



2015

- **<u>Microsoft Windows 10</u>**
  - The Start menu includes the contents of the Windows 8 Start Screen
  - Microsoft uses Windows 10 for advertising and bundles "telemetry" spyware



Building GUI with JAVA

- Abstract Windowing Toolkit (AWT): Sun's initial effort to create a set of cross-platform GUI classes (JDK 1.0 - 1.1)
  - Maps general Java code to each operating system's real GUI system
  - Limited to lowest common denominator; clunky to use
- Swing: A newer GUI library written from the ground up that allows much more powerful graphics and GUI construction (JDK 1.2+)
  - Paints GUI controls itself pixel-by-pixel rather than handing off to OS
  - Better features, better compatibility, better design
- For example, the AWT `Button` class corresponds to a more versatile Swing class called `JButton`
- However, Swing does not generally replace the AWT; we still use AWT events and the underlying AWT event processing model

java.awt - The Abstract Windowing Toolkit

Introduced with Java 1.0

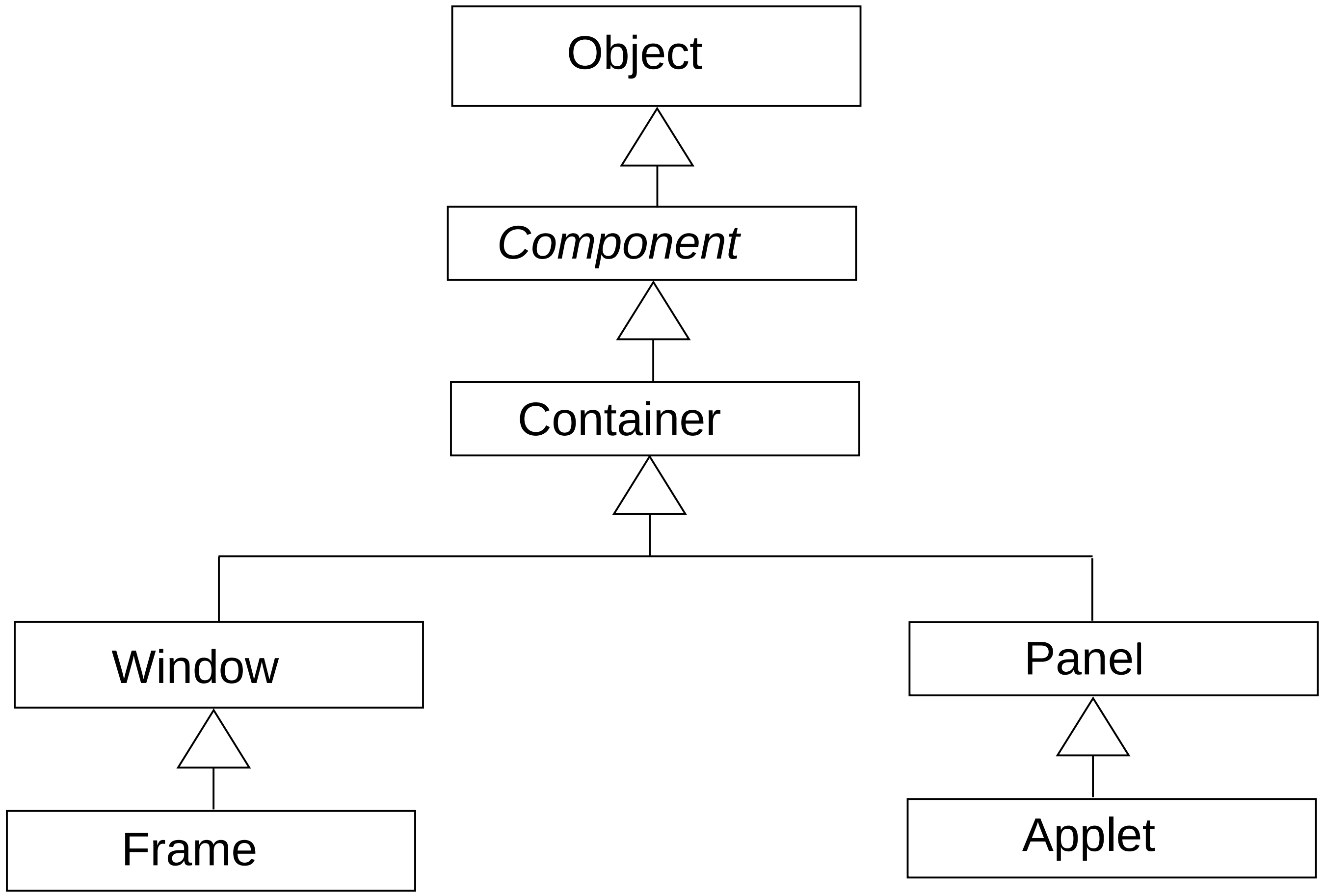
Classes are divided into 3 main categories:

- graphics (colours, fonts, shapes, etc.)
- components (GUI components: windows, buttons, menus, etc.)
- layout managers (control the positioning of components on the screen)

Each component corresponds to a “peer” component provided by the native GUI toolkit on the target platform (Windows, Sun Solaris, etc.)

Here is a (small) subset of the AWT class hierarchy:

AWT Hierarchy



java.awt - The Abstract Windowing Toolkit

`Component`

- an abstract class
- superclass of all GUI components except menu components and class CheckboxGroup()

`Container`

- the superclass for all components that contain other components
- defines `add()`, for adding components to a container

java.awt - The Abstract Windowing Toolkit

`Window`

- a top-level window with no border or menu bar
- rarely instantiated (its subclasses are more useful)

`Frame`

- a window with a title bar
- can have a menu bar
- top-level window for Java AWT-based applications
  - typically, `main()` creates an instance of an object that *is a* `Frame` as its top-level application window, then adds GUI components to it

java.awt - The Abstract Windowing Toolkit

`Panel`

- a container that must be contained within another container
- does not have its own window

`Applet`

- a subclass of `Panel`
- actually part of the `java.applet` package, not the AWT

AWT Limitations

“look and feel” of AWT-based programs differs slightly across platforms, because of differences in the underlying native GUI elements

AWT components limited to those that are available on all platforms (lowest common denominator)

javax.swing - The Swing Toolkit

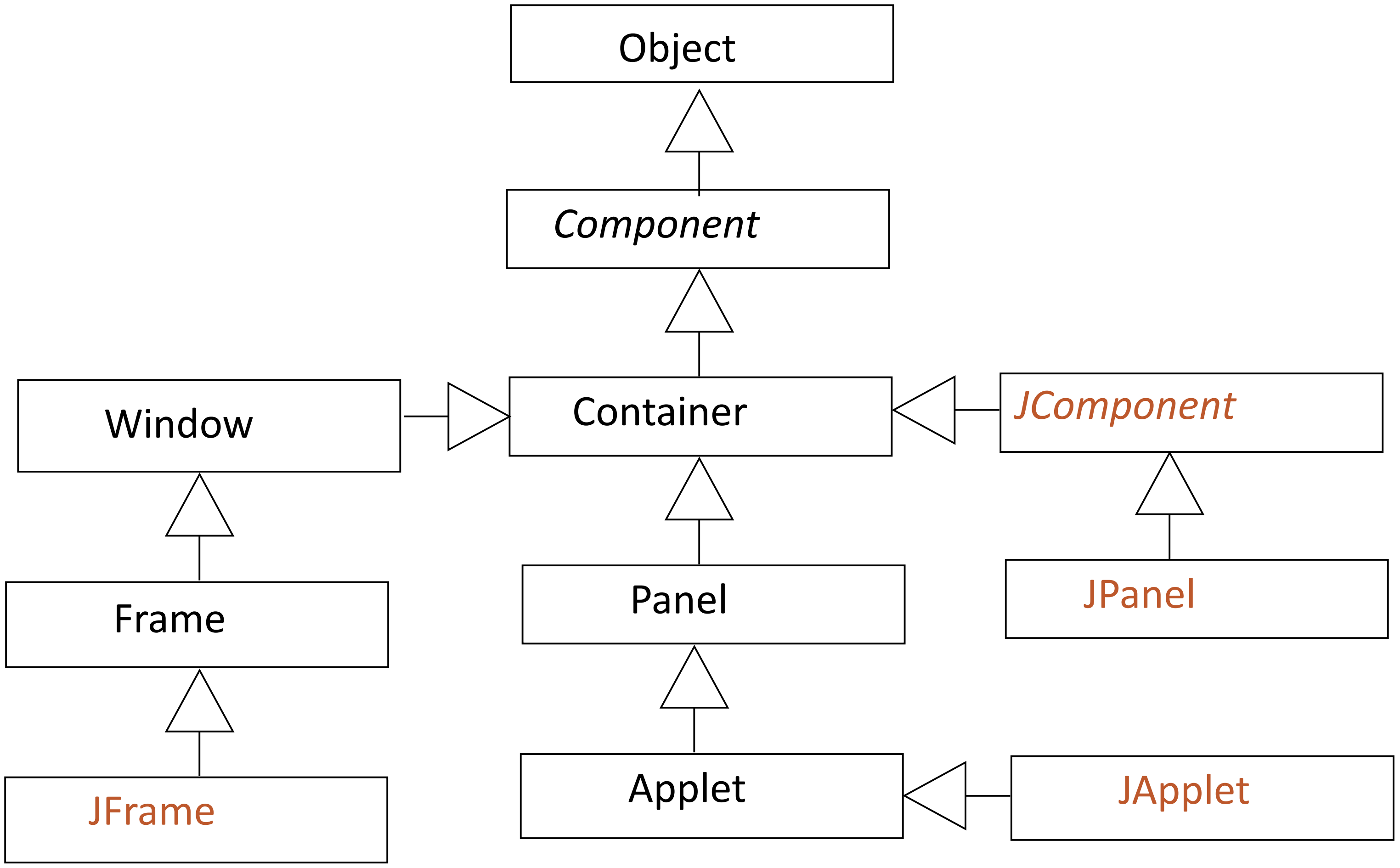
In response, Netscape developed the Internet Foundation Classes, which evolved into the Swing toolkit that is part of Sun’s Java Foundation Classes (JFC)

Swing components do not require native peer components

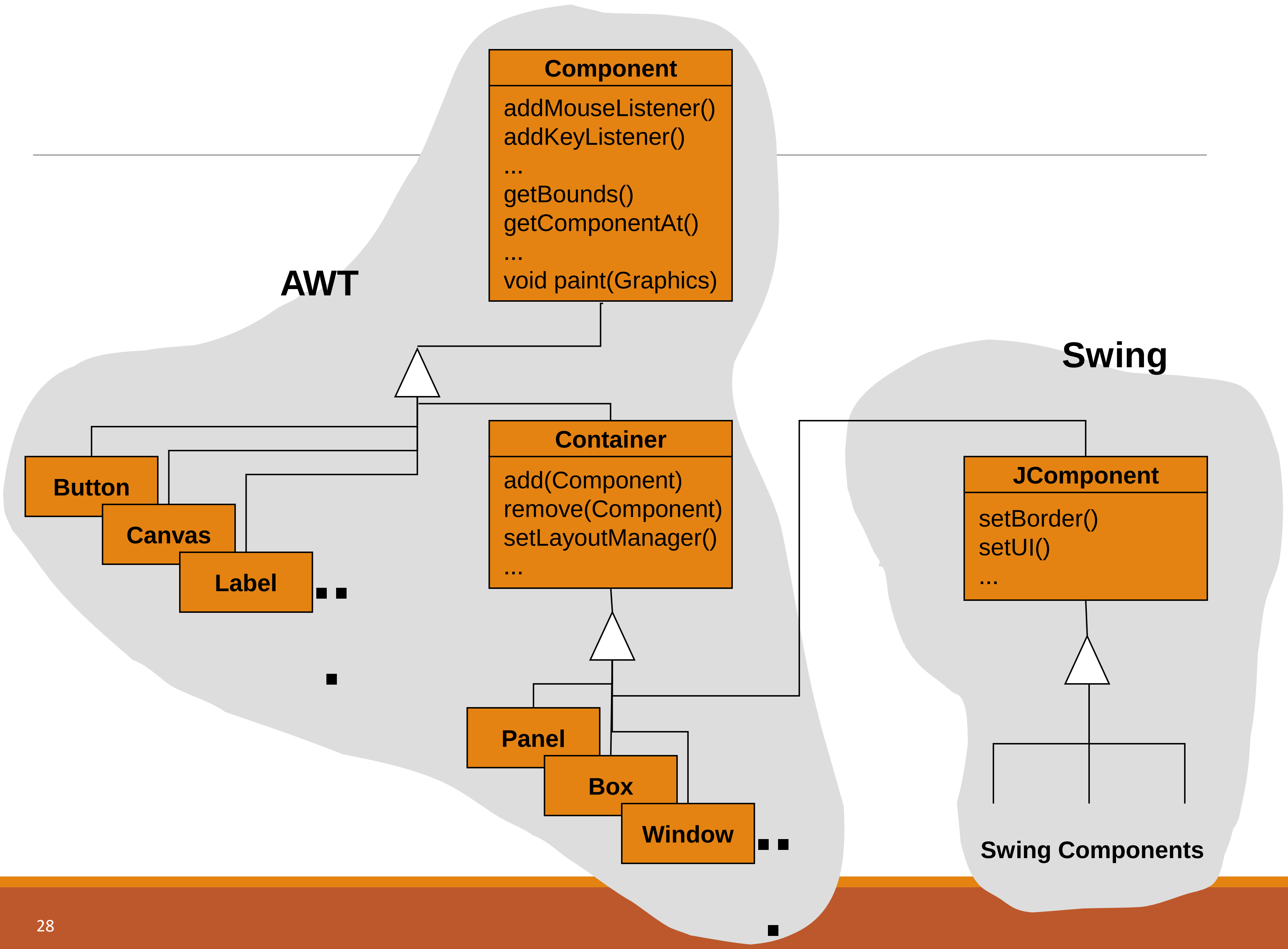
Each Swing UI component is painted onto a blank window

Only peer functionality required is the ability to display windows and paint on them

Relationship of Swing and AWT



Inherited Functions



Swing Components

There are various Swing GUI components that we can incorporate into our software:

- labels (including images)
- text fields and text areas
- buttons
- check boxes
- radio buttons
- menus
- combo boxes
- and many more…

Using the proper components for the situation is an important part of GUI design

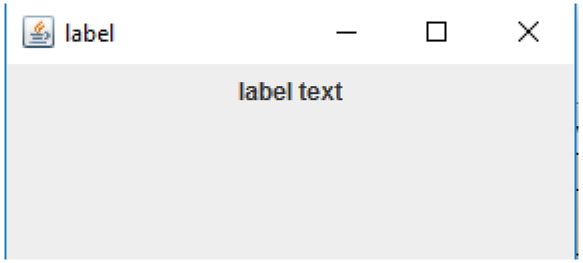
Labels and Image Icons

A label is used to provide information to the user or to add decoration to the GUI

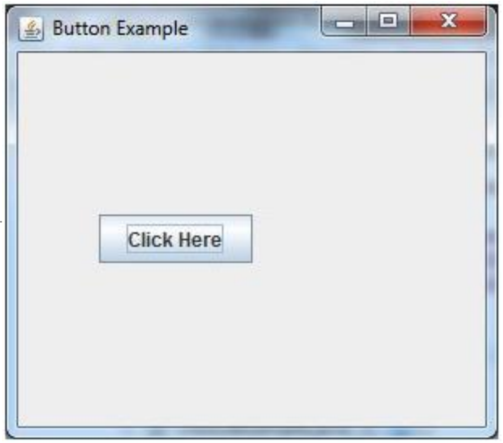
A Swing label is defined by the `JLabel` class

It can incorporate an image defined by the `ImageIcon` class (later)

The alignment and relative positioning of the text and image of a label can be explicitly set



Buttons



GUI buttons fall into various categories:

- push button – a generic button that initiates some action
- check box – a button that can be toggled on or off
- radio buttons – a set of buttons that provide a set of mutually exclusive options

Radio buttons must work as a group; only one can be toggled on at a time

Radio buttons are grouped using the `ButtonGroup` class

Push buttons and radio buttons generate action events when pushed or toggled

Check boxes generate *item state changed* events when toggled

Radio Buttons

Declaration and initialization:

```
JRadioButton radioButtonSmall, radioButtonLarge ;
radioButtonSmall = new JRadioButton (“small”, false);
radioButtonLarge = new JRadioButton (“large”, false);
this.add (radioButtonSmall);
this.add (radioButtonLarge);
radioButtonSmall.addActionListener (this);
radioButtonLarge.addActionListener (this);
```

Action Listener may be assigned to each button

Radio Buttons

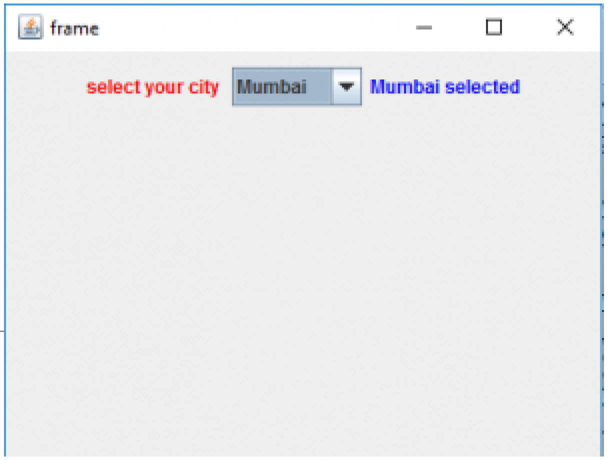
Action performed method use is same as regular buttons

But one needs to put them in a group so they become mutually exclusive

```
ButtonGroup buttonGroup = new
ButtonGroup();

buttonGroup.add(small);

buttonGroup.add(large);
```



Combo Boxes

A *combo box* displays a particular option with a pull down menu from which the user can choose a different option

The currently selected option is shown in the combo box

A combo box can be *editable*, so that the user can type their option directly into the box

itemStateChanged() fires on a mouse click in a ComboBox

Combo Box

What goes in the constructor:

```
//          Combo Box

comboBoxItem = new JComboBox();

this.add (comboBoxItem );

comboBoxItem.addItemListener (this);

comboBoxItem.addItem ("10 Computer");

comboBoxItem. addItem ("20 Monitor ");
comboBoxItem. addItem ("30 Printer ");


comboBoxItem.removeAllItems ();
```

A Combo Box can not be emptied unless it has something in it

Text Field

Constructor: new JTextField(20) // 20 columns

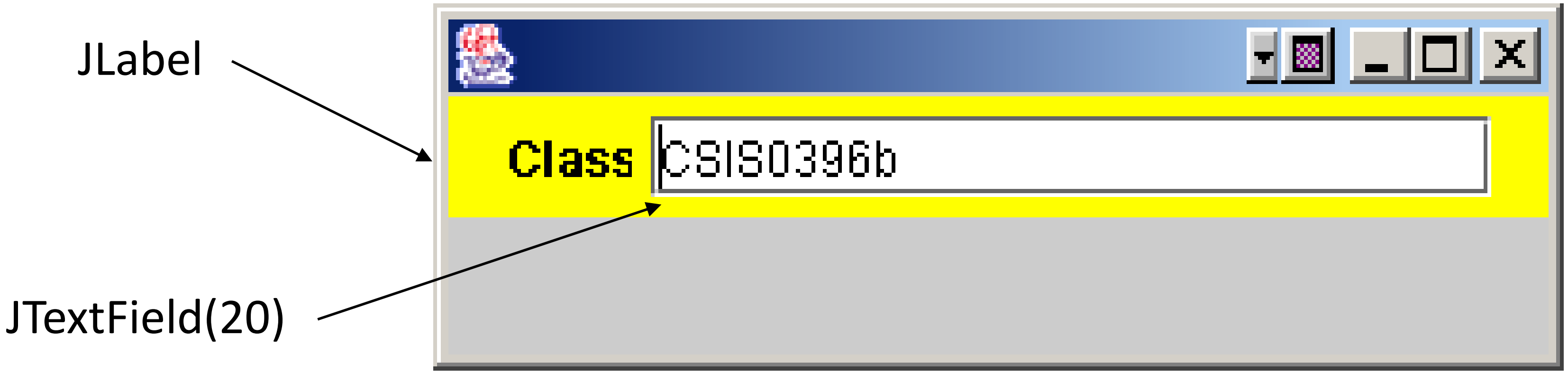
Get text out: field.getText()

Put text in: field.setText(“whatever”);

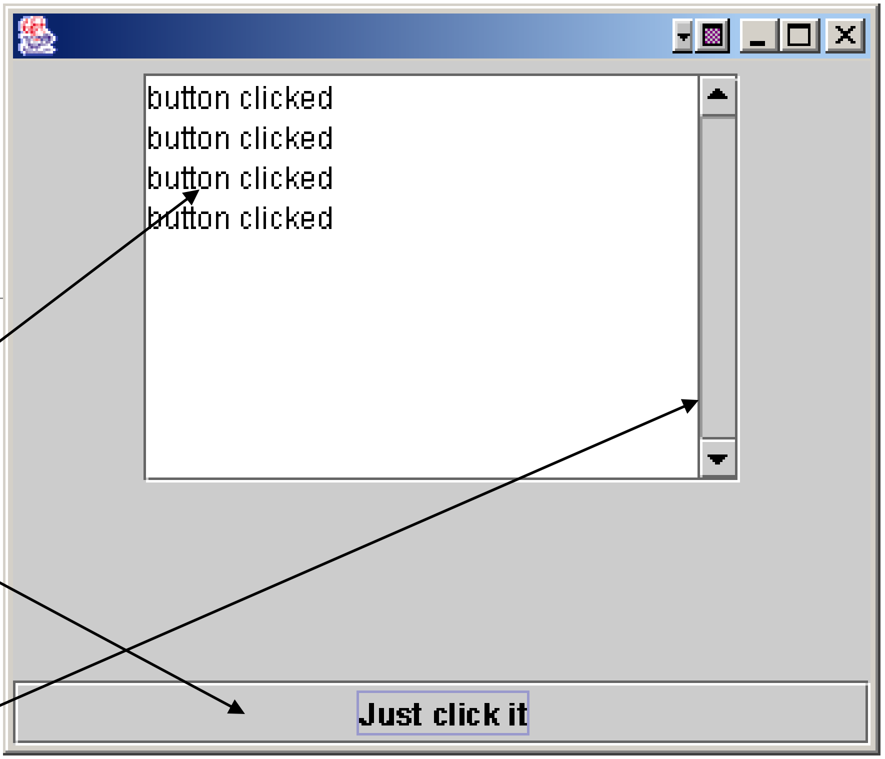
Get an ActionEvent when the user presses return or enter:
field.addActionListener(myActionListener);

Select the text in the field: field.selectAll();

Put the curson back in the field: field.requestFocus();



Text Area



```
button clicked
button clicked
button clicked
button clicked
```

```
JFrame jf = new JFrame();
JPanel jp = new JPanel();
JButton jb = new JButton("Just click it");
jb.addActionListener(this);
text = new JTextArea(10,20);
text.setLineWrap(true);
JScrollPane scroller = new JScrollPane(text);
jp.add(scroller);
jf.getContentPane().add(BorderLayout.CENTER,jp);
jf.getContentPane().add(BorderLayout.SOUTH,jb);
```

Create a JScrollPane and give it the text area that it is going to scroll for

JCheckBox

Constructor: new JCheckBox(“checkbox 1”);

Listen for an item event (when it is selected or deselected):

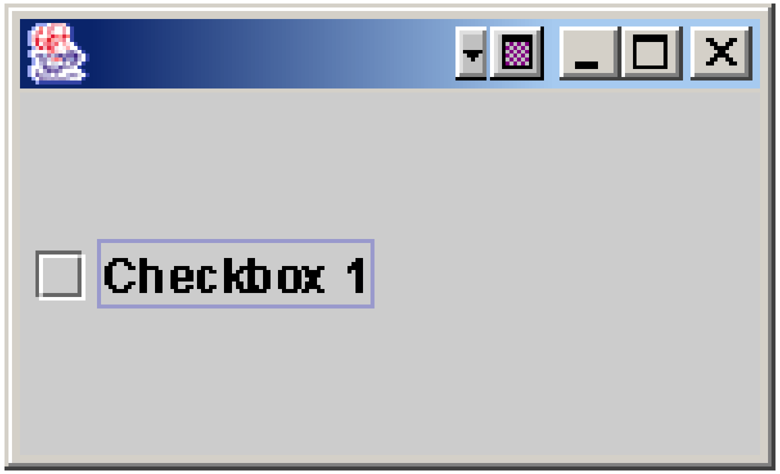
- check.addItemListener(this);

Handle the event:

-
  ```
  public void itemStateChanged(ItemEvent e) {
          String onOff = “off”;
          if (check.isSelected() ) onOff = “on”;
  }
  ```

Select or deselect it in code:

- check.setSelected(true);
- check.setSelected(false);



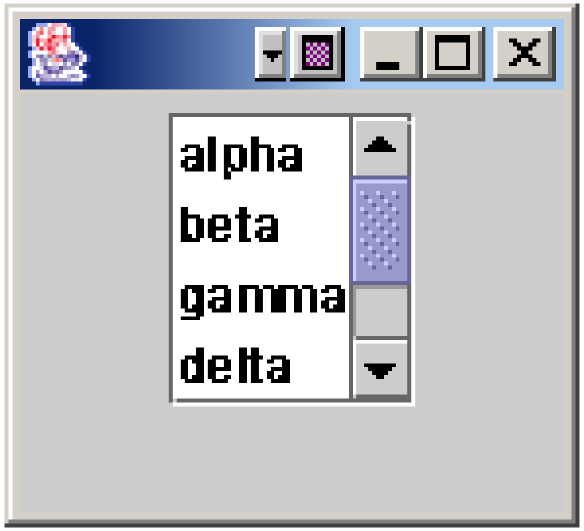
JList

```
String[] entries = {“alpha”, “beta”, “gamma”, “delta”, “zeta”, “eta” };
JPanel jp = new JPanel();
JList list = new JList(entries);

JScrollPane scroller = new JScrollPane(list);
    jp.add(scroller);

list.setVisibleRowCount(4);
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

jf.getContentPane().add(jp);
```



GUI Input via the Interface

- InputStreams are employed to get input from the user via the keyboard
  - it waits for the user to input data and press enter
- In a GUI-based program, there can be several sources of input; e.g., clicking a button, moving a slider, typing characters in a text field, etc.
- Invoking methods to wait for user input via a specific component won't work, if multiple input source points
  - ❖ the interface can have multiple input points
    - ➢ e.g. Microsoft Word can get input from multiple sources
      - ▪ Menu, Toolbar, Text input, Alt-keystroke
  - ❖ we can't predict where the next input will come from

Event-Driven GUI

- GUI-based Java programs use the Java Event Model
- The Java Virtual Machine watches for the user’s actions on components; e.g.,
  - mouse movement/dragging, clicks on components
  - key presses
- The JVM creates event objects that correspond to these actions, and sends these event objects to listeners which are provided by the program to handle the events

Event-Driven GUI

`java.awt.AWTEvent` extends `EventObject` and is the superclass of all Java 1.1 AWT events

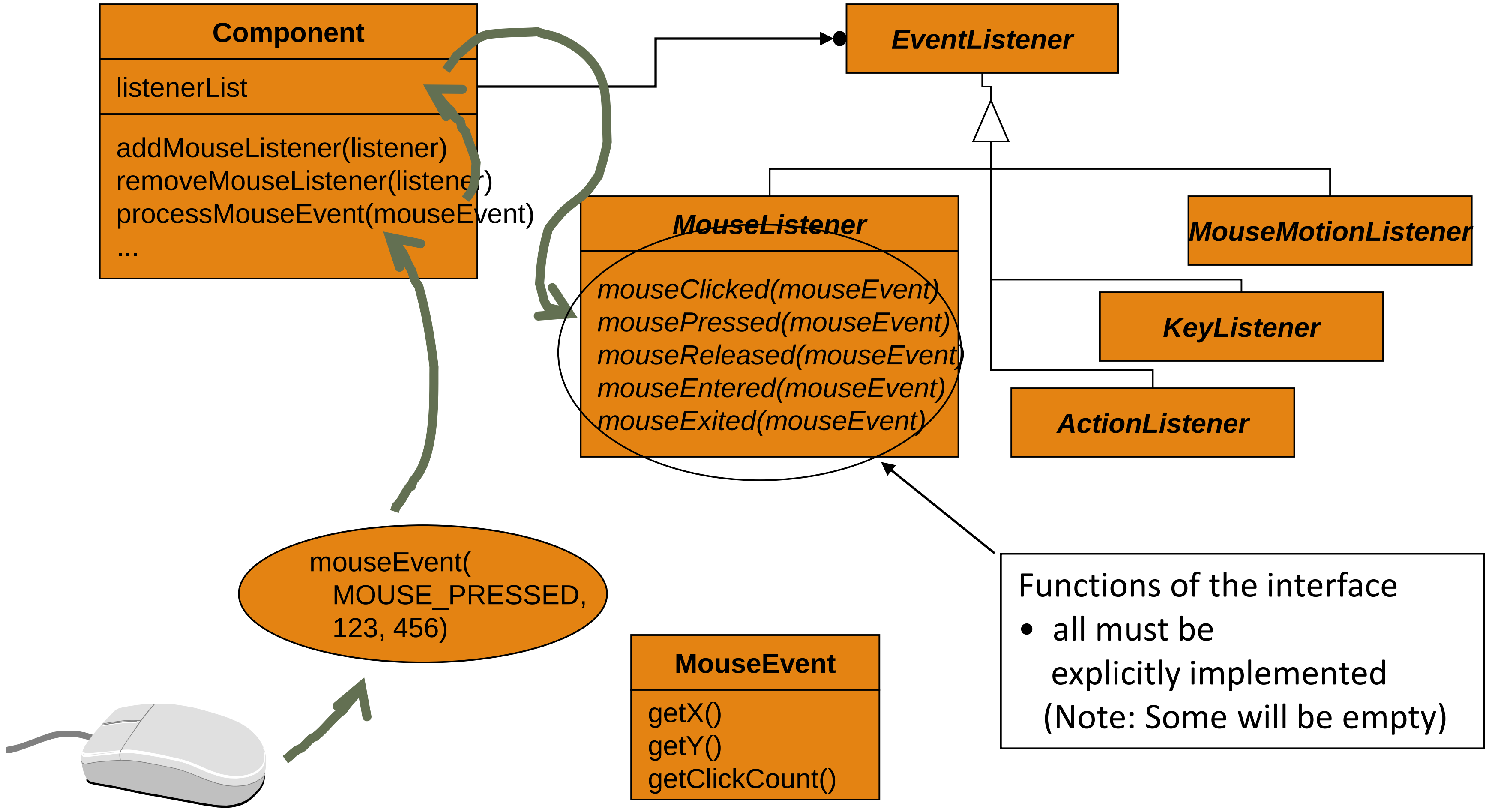
Each AWT event is represented by its own subclass; e.g.,

- `java.awt.event.WindowEvent`
- `java.awt.event.MouseEvent`

Swing adds additional event classes, but uses the Java 1.1 event model

Listening is done via Listener Interfaces, whose functions fire when events occur, dispatching an event object to the functions, which must be implemented

Mouse Event and Listener with Other Listeners



Functions of the interface
- all must be explicitly implemented (Note: Some will be empty)

Example - Creating a Frame

```
import javax.swing.*;
public class FrameExample1
{
   public static void main(String args[])
   {
      JFrame f = new JFrame("Frame Example 1");
      f.setSize(400, 300);
      f.show();
   }
}
```

Example - Creating a Frame

By default, a frame is sized at at 0x0 pixels, so `setSize()` is sent to the frame to change it size to 400 pixels wide x 300 pixels high

If the `setSize()` message is not sent, only the title bar is displayed

`setSize()` inherited from `Component`

The `show()` message is sent to the frame to make it visible and bring it to the front (if it is behind another window)

`show()` inherited from `Window`

Example - A Closeable Frame

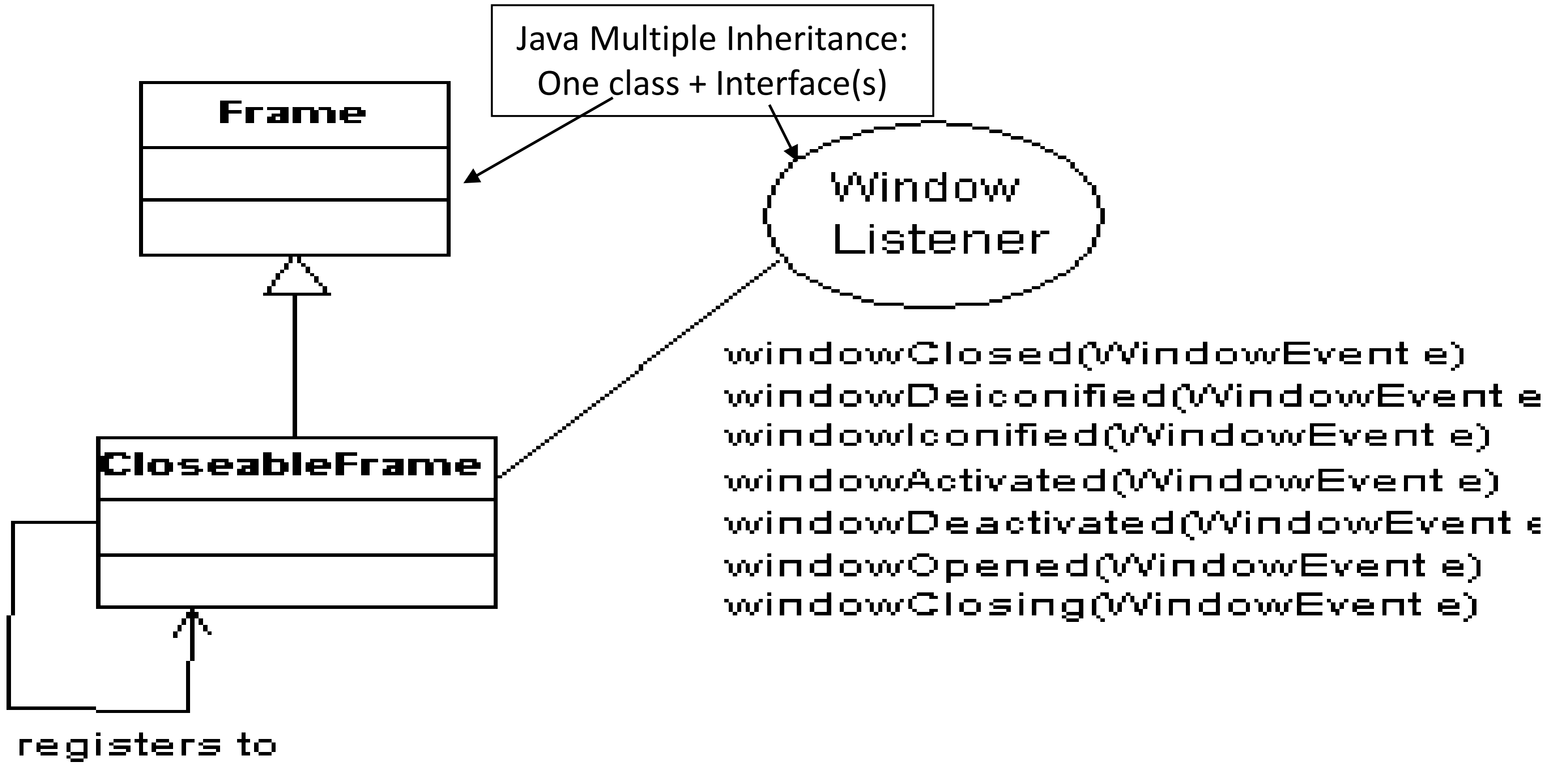
This frame has a deficiency that limits its usefulness as a basis for developing applications

- clicking on the Close button or selecting Close from the left-most drop-down menu hides the frame but does not close the application
- if the Java interpreter was run from a command line, to stop the program we must close the console window (NT) or issue a kill command(Unix).

This is because nobody is *listening* to the closing of the window…

Class diagram for ClosableFrame

- We can now design a subclass of `JFrame` called `CloseableFrame` that behaves as expected when the Close button is clicked


Java Multiple Inheritance:
One class + Interface(s)
Frame
Window Listener
CloseableFrame
windowClosed(WindowEvent e)
windowDeiconified(WindowEvent e
windowIconified(WindowEvent e)
windowActivated(WindowEvent e)
windowDeactivated(WindowEvent e
windowOpened(WindowEvent e)
windowClosing(WindowEvent e)
registers to

Example - A Closeable Frame

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

public class CloseableFrame extends Jframe implements WindowListener
{
   public CloseableFrame()
   {
      super();
      addWindowListener(this);
   }

   public CloseableFrame(String str)
   {
      super(str);
      addWindowListener(this);
   }
   public void windowClosed(WindowEvent event)
   {} // Do nothing!

   public void windowDeiconified(WindowEvent event)
   {}

   public void windowIconified(WindowEvent event)
   {}
```

Example - A Closeable Frame

```
public void windowActivated(WindowEvent event)
{}  // Do nothing!
public void windowDeactivated(WindowEvent event)
{}
public void windowOpened(WindowEvent event)
{}
public void windowClosing(WindowEvent event)
{
   // dispose of the JFrame object
   dispose();
   // terminate the program
   System.exit(0);
}

public static void main(String args[])

{

   JFrame f = new CloseableFrame ("Closeable Frame");

   f.setSize(400, 300);

   f.show();

}

}
```

`CloseableFrame` Constructors

`JFrame` has two useful constructors:

- `JFrame()` creates an untitled `JFrame` object
- `JFrame(String title)` creates a `JFrame` object with the specified title

Our `CloseableFrame` class provides similar constructors

When a window is opened, closing, closed, activated, deactivated, iconified, or deiconified, it sends a `WindowEvent` object to its `WindowListener` object

`CloseableFrame` is-a `JFrame` is-a `Frame` is-a `Window`, so a `CloseableFrame` object will send `WindowEvents`

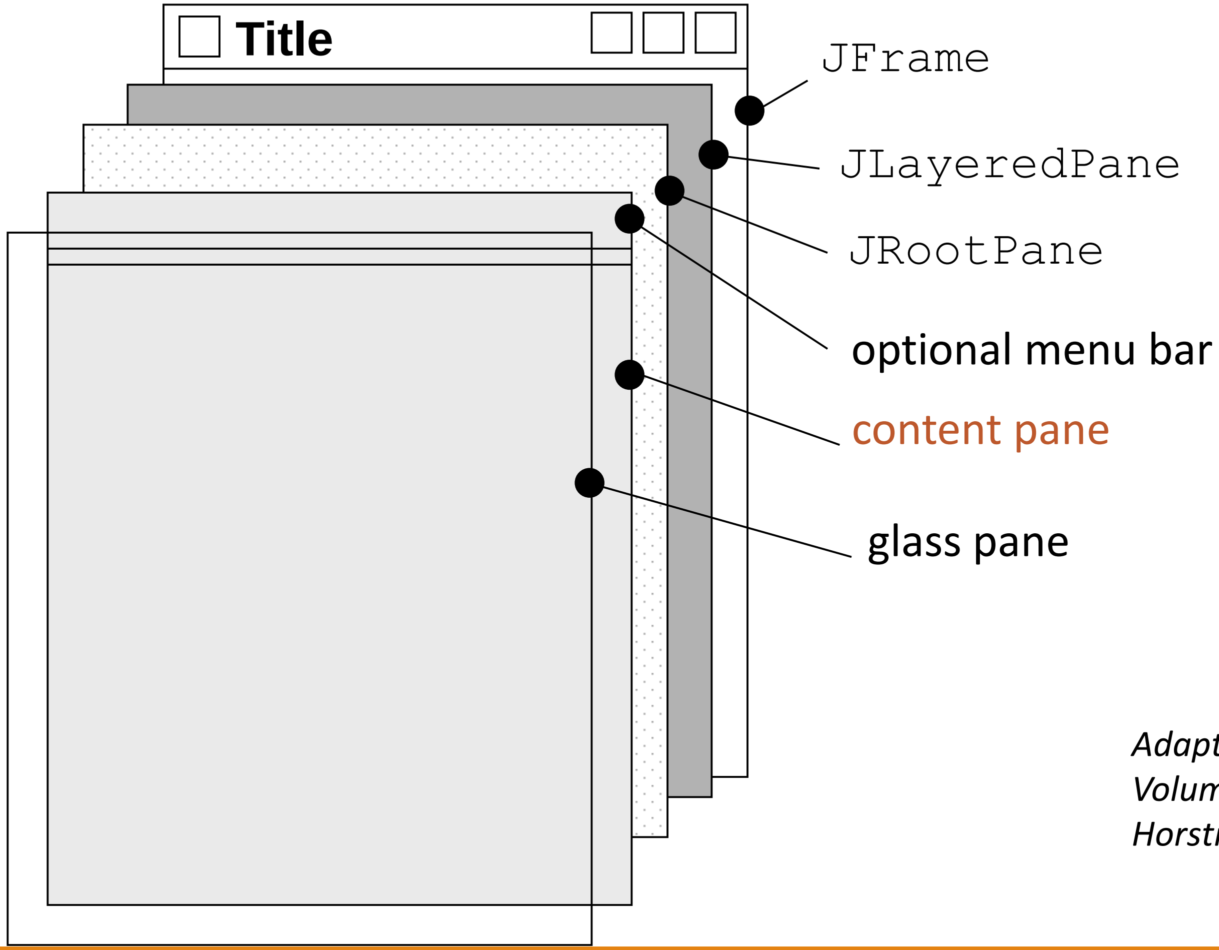
When the Close button is clicked, it notifies the listener object by invoking the object's `windowClosing()` method

Closing the Window

```
JFrame frame = new JFrame(“My GUI”);
frame.setDefaultCloseOperation(JFrame.EXIT_ON CLOSE);
```

Alternating method to make closeableFrame

Structure of a `JFrame` object



Adapted from Core Java 1.2, Volume 1 - Fundamentals, Horstmann & Cornell

JFrame

Methods for manipulating these parts

```
public Container getContentPane()
public Component getGlassPane()
public JMenuBar getJMenuBar()
public JLayeredPane getLayeredPane()
public JRootPane getRootPane()

public void setContentPane(…)
public void setGlassPane(…)
public void setJMenuBar(…)
public void setLayeredPane(…)
public void setRootPane(…)
```

Layout Manager

A layout manager is a Java object associated with a particular component (usually with a background component)

The layout manager controls the components contained within the component the layer manager is associated with

E.g. if a frame holds a panel, and the panel holds a button, the panel’s layout manager control s the size and placement of the button, and the frame’s layout manager controls the size and placement of the panel, and the button does not need a layout manager

A panel holds 5 things (north, east, south, west, center): even if the 5 things each have their own layout managers, the size and location of the 5 things in the panel are controlled by the panel’s layout manager

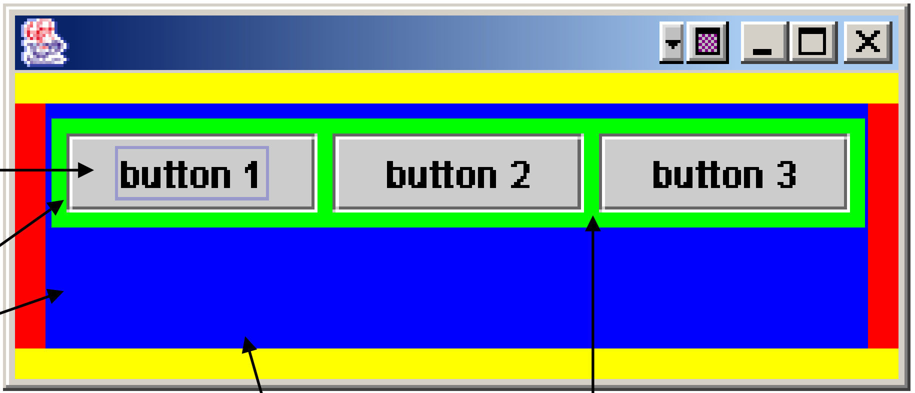
Different background components have different types of layout managers, different layout managers have different layout policies

Commonly used layout managers:

- BorderLayout: default for a frame
- FlowLayout: default for a panel
- BoxLayout

Nested Layouts

```
import javax.swing.*;
import java.awt.event.*;
public class SimpleGUI1 implements ActionListener {
  …
  JPanel panelA = new JPanel();
  JPanel panelB = new JPanel();
  panelB.add(new JButton(“button 1”));
  panelB.add(new JButton(“button 2”));
  panelB.add(new JButton(“button 3”));
  panelA.add(panelB);
  panelA.setColor(Color.blue);
  panelB.setColor(Color.green);
  …
}
```



Both panelA and panelB have their own layout managers.

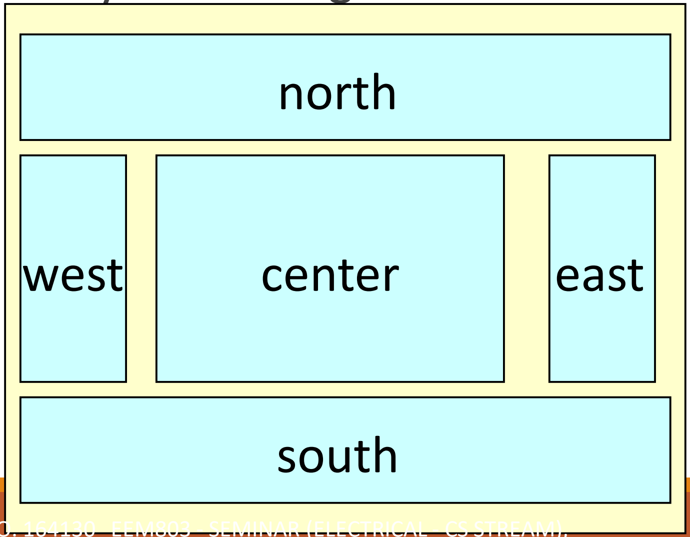
Border Layout

A BorderLayout manager divides a background component into 5 regions

You can only add one component per region

Components layed out by this manager usually don't get to have their preferred size

BorderLayout is the default layout manager for a frame



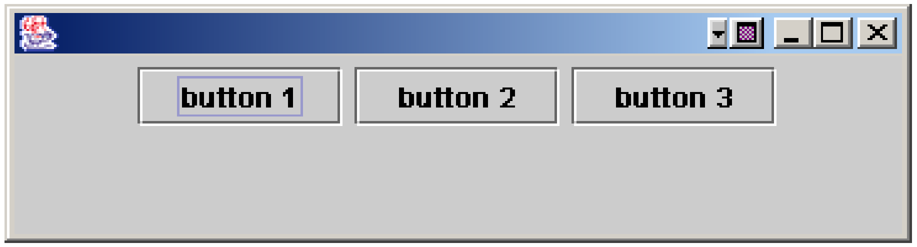
Flow Layout

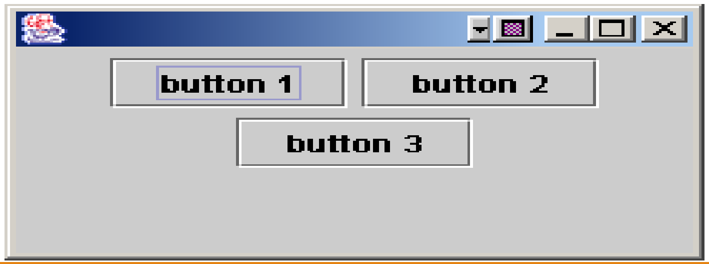
A FlowLayout manager acts kind of like a word processor, except with components instead of words

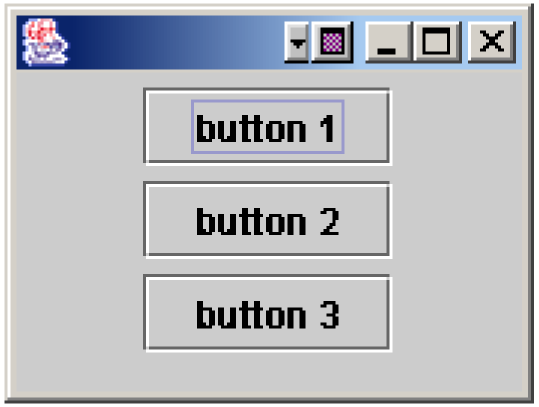
Each component is the size it wants to be, and they are laid out left to right in the order that they are added, with “word-wrap” turned on

When a component won’t fit horizontally, it drops to the next “line” in the layout

Default layout manager for a panel



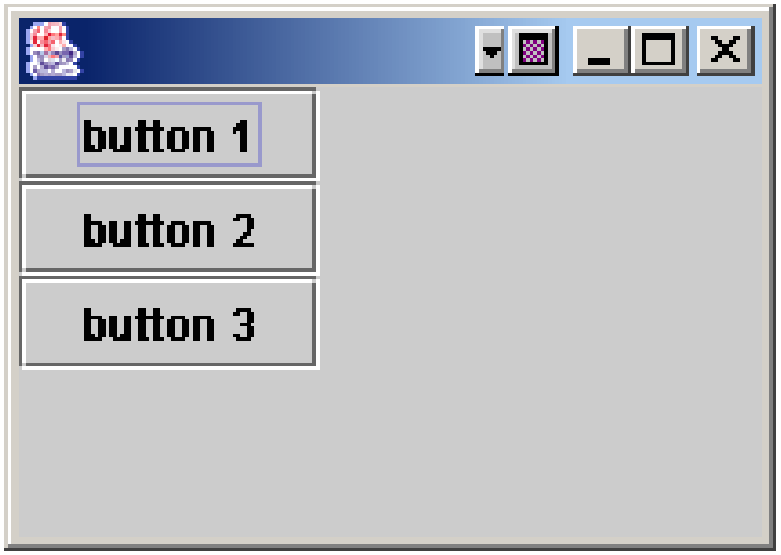




Box Layout

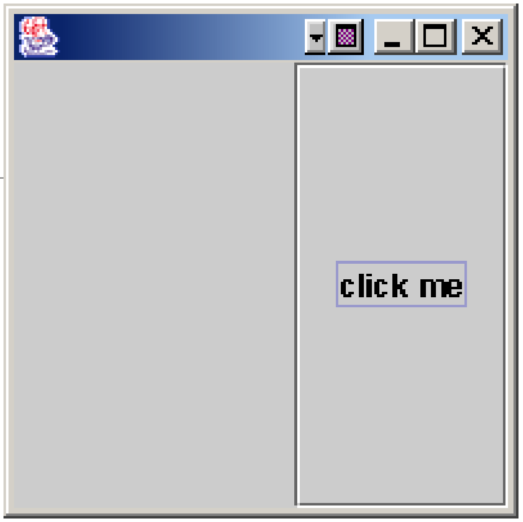
A BoxLayout manager is like FlowLayout in that each component gets to have its own size, and the components are placed in the order they are added

A BoxLayout can stack the components vertically or horizontally (instead of automatic 'component wrapping')

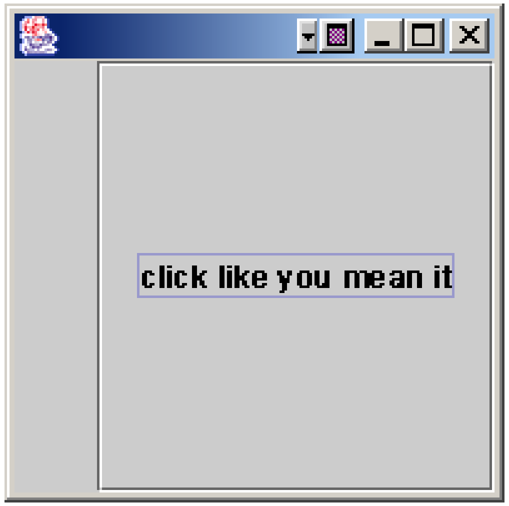


How is a BorderLayout laid out?

```
…
JButton jb = new JButton("click me");
jf.getContentPane().add(BorderLayout.EAST,jb);
jf.setSize(200,200);
…
```



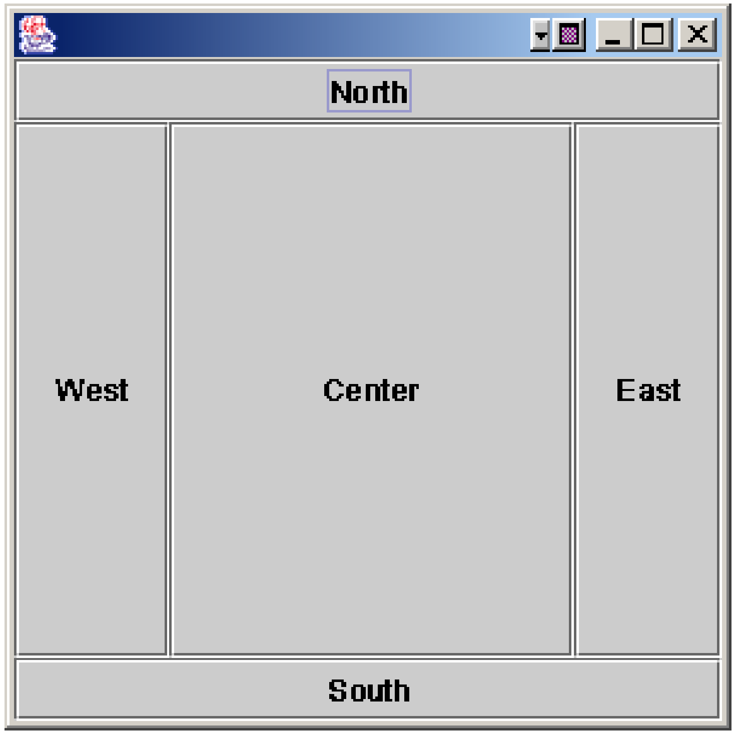
```
…
JButton jb = new JButton("click like you mean it");
jf.getContentPane().add(BorderLayout.EAST,jb);
jf.setSize(200,200);
…
```



- Since it is the east region or a border layout, the preferred width is respected, but the height will be as tall as the frame.

How BorderLayout works ?

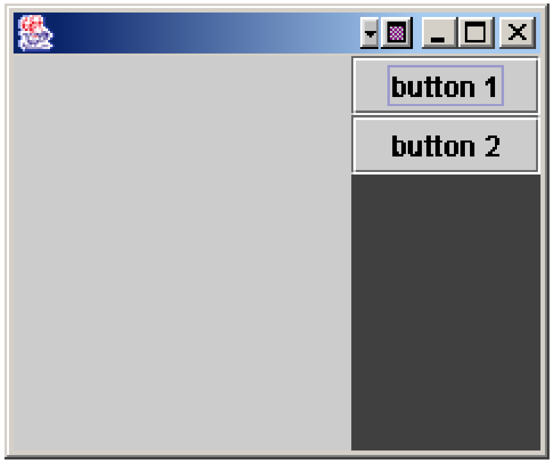
- When something is put in the north or south, it goes all the way across the frame, so the things in the east and west will not be as tall as the frame
- Components in the east and west get their preferred width
- Components in the north and south get their preferred height
- Components in the center get whatever space is left over, based on the frame dimensions

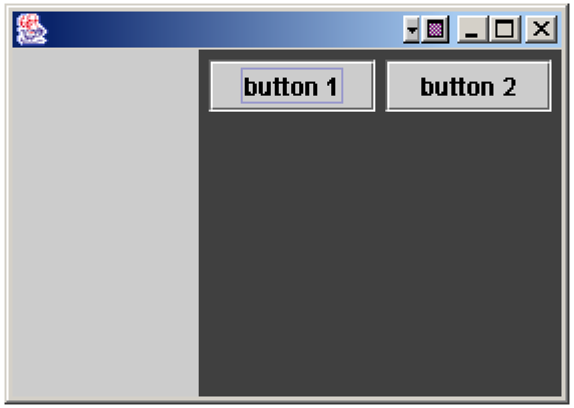


BorderLayout and FlowLayout together

```
JFrame jf = new JFrame();
JPanel jp = new JPanel();
jp.setBackground(Color.darkGray);
JButton jb1 = new JButton("button 1");
JButton jb2 = new JButton("button 2");
jp.add(jb1);
jp.add(jb2);
jf.getContentPane().add(BorderLayout.EAST,jp);
jf.setSize(250,200);
jf.setVisible(true);
```

- Which layout on the left corresponds to the above code?
- How to get the other one?





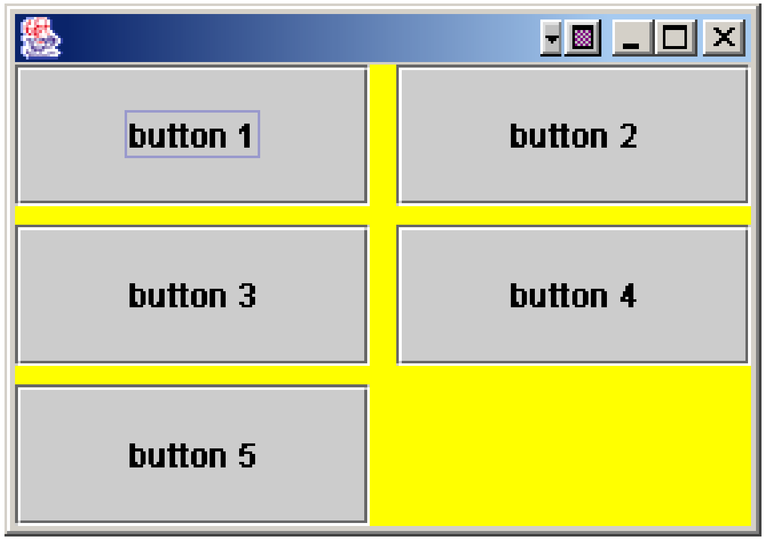
GridLayout

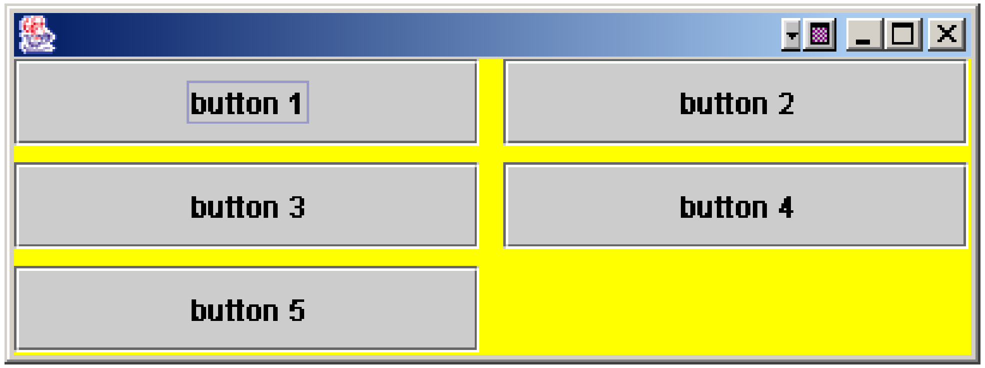
Sets up a grid (each cell is the same size)
(m x n) layout

```
frame.getContentPane().setLayout(new GridLayout(3,2,10,7));
```

$\underbrace{\hphantom{3,2,10,7}}$

(Row, Col,
Horizontal Gap,
Vertical Gap)





Layout Managers

Predefined layout managers defined in the Java standard class libraries:

Flow Layout **Border Layout** **Card Layout** **Grid Layout** **GridBag Layout**	**Defined in the AWT**
Box Layout **Overlay Layout**	**Defined in Swing**

Layout Managers

Every container has a default layout manager, but we can also explicitly set the layout manager for a container

Each layout manager has its own particular rules governing how the components will be arranged

Some layout managers pay attention to a component's preferred size or alignment, and others do not

The layout managers attempt to adjust the layout as components are added and as containers are resized

Special Features

Swing components offer a variety of other features

Tool tips provide a short pop-up description when the mouse cursor rests momentarily on a component

Borders around each component can be stylized in various ways

Keyboard shortcuts called *mnemonics* can be added to graphical objects such as buttons

Check the Sun Tutorial for more info