JAVA GUI

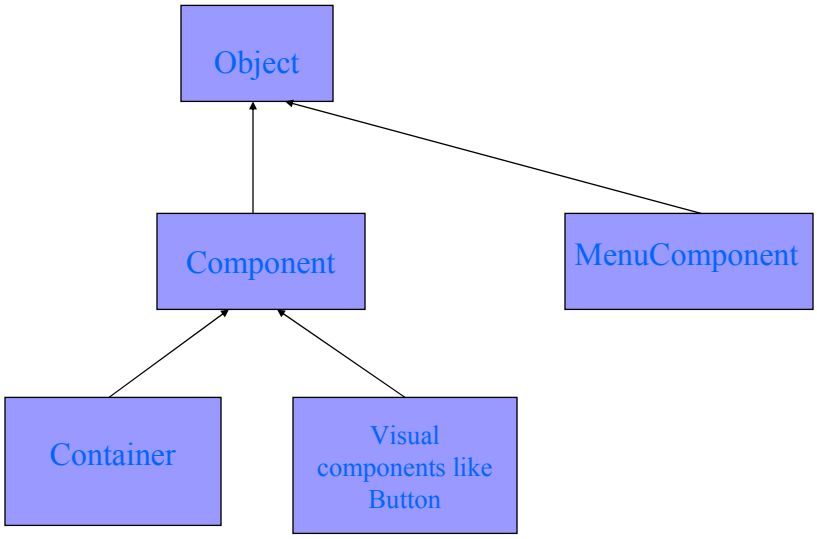
GUI Packages

- java.awt
- javax.swing

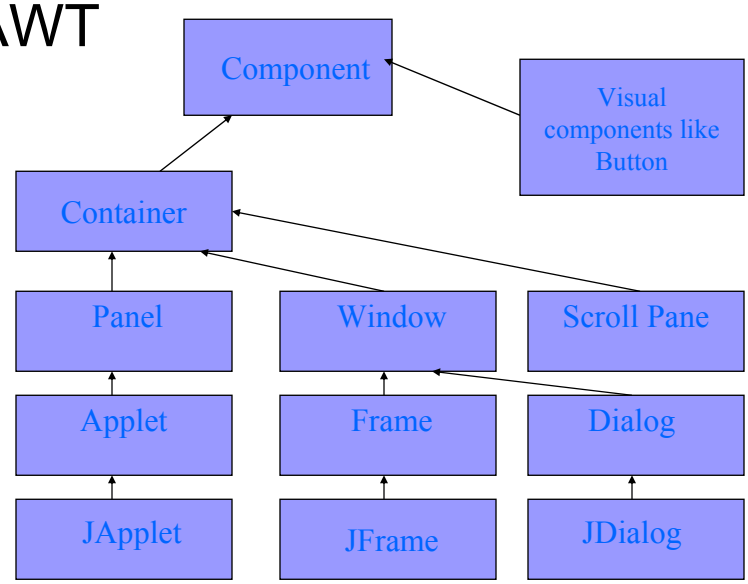
Abstract Window Toolkit

- The java.awt package was the primary repository for classes that are used to create a GUI in java but many of the classes it defines have been superseded in Java 2 by javax.swing
- However the Swing classes are generally derived from, and depend on, fundamental classes within java.awt, so these cant be ignored.

AWT



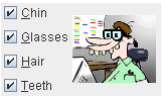
AWT



Components

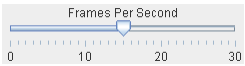
- **A component represents a graphical entity of one kind or another that can be displayed on screen**
- **All container and visual components inherit from java.awt.Component**
- **Key Classes: Window, JFrame, JDialog, JApplet etc**

JButton

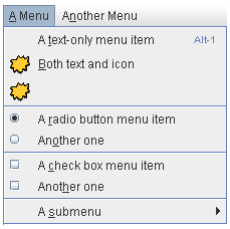


JCheckBox

JPasswordField

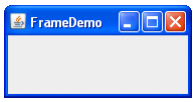


JSlider



JMenu

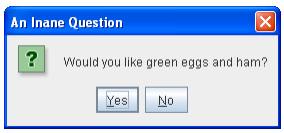
JList



JFrame

JApplet

JSpinner



JDialog

Framing a Window Example

```
import javax.swing.JFrame;

public class TryWindow {
  // The window object
  static JFrame aWindow = new JFrame("This is the Window Title");

  public static void main(String[] args) {
    int windowWidth = 400;                    // Window width in pixels
    int windowHeight = 150;                   // Window height in pixels
    aWindow.setBounds(50, 100,                // Set position
                windowWidth, windowHeight);   // and size
    aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    aWindow.setVisible(true);                 // Display the window
  }
}
```

Containers

- All the classes derived from the Container can contain other objects of the classes derived from Component
- The exception is Window Class as it cant be contained in other containers

Components

- **There are several methods that are available for components**
- **setForeground( ) and setBackground( )**
  - Each method takes a single argument which is an instance of java.awt.Color
  - Generally foreground color controls the text while background color is the non-text areas of a component
  - If the component is in a container and you don't specify the fore and background colors then it uses these colors from the container

Components

- **setFont( ) method--determines the font that a component will use for dealing with text.**
  - It takes a single argument which is an instance of java.awt.Font
  - If you don't explicitly set a components font, the component uses the font of its container
- **setEnabled( )—takes a single argument of type boolean.**
  - if this argument is true then the component has its normal appearance
  - If it is false then that component is grayed out and does not respond to user input

Components

- **setSize( ) and setBounds( ) method—These methods sets a components geometry**
  - The setSize( ) method takes two arguments, width and height
  - The setBounds( ) methods takes four arguments , x, y , width and height
  - These methods only work for frames
- **setVisible( )—takes a single argument of type boolean.**
  - if this argument is true then the component can be seen on the screen
  - If it is false then that component is not visible
  - This method works for frames only

Container Components

- **The Container class is abstract and most commonly used concrete subclasses are JApplet, JFrame, JDialog and Panel**
- **Containers are components capable of holding other components within their boundaries, A container is responsible for laying out any components that it contains. It does this through the use of various layout managers**
- **Panel—is a concrete class of Container and doesn't add any new method.**
- **A panel is a window that does not contain a title bar,menu bar or border, that is why you don't see these when your browser is executing the applet**
- **When you run an applet using an applet viewer, the applet viewer provides the title and the border**
- **Panel is the super class for applet**

Container Components

- **Frame– is a sub class of window and has title bar, menu bar, borders and resizing corners**
- **A frame is capable of being moved around on the screen independent of other GUI windows**
- **If you create a frame object from within an applet, it will contain a warning message to the user that an applet window has been created, while when a frame window is created by an application, a normal window is created**
- **Generally you will create a subclass of Frame to use frames**
- **There are only two forms of the Frame constructor**
  - Frame( )
  - Frame(String title)

Container Components

- When a frame is constructed it has no size and is not displayed on the screen
- Use inherited methods setSize( ), setBounds( ) to give it a size and then you can display it by calling setVisible(true)
- To remove an unwanted frame from the screen you can call setVisible(false)
- Frame is not destroyed by calling setVisible(false) you can redisplay it by again calling setVisible(true)
- You call the dispose( ) method to release the non-memory resources of a frame

Layout managers

- Java GUI reside in applets or in frames
- For applets you put your components in your applet while for applications you put your components in your frames
- Where those components end up in the applets and frames is dictated by the layout managers
- There are several layout manager classes in the AWT and swing
- java.awt.LayoutManager is an interface not a class

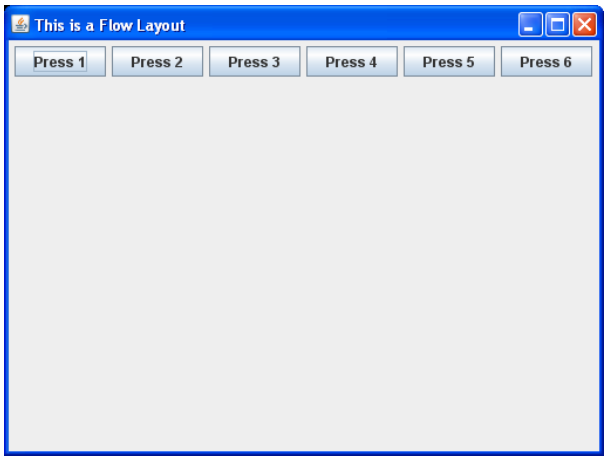
Layout managers

- **Flow Layout Manager—It is the default layout manager for panels and applets**
- **It always arranges the components in horizontal rows while honoring each components preferred size**
- **The components always appear left to right in the order in which they were added to their container**
- **Within every row the components are evenly spaced and the cluster of components is centered**
- **To change this default behavior , you can use setLayout( )**
  - setLayout needs a parameter of type object of Layout Manager
  - setLayout(new FlowLayout(FlowLayout.RIGHT))
- **By default it leaves a gap of 5 pixels between components in both horizontal and vertical directions**

FlowLayout Example

```
import javax.swing.JFrame;
import javax.swing.JButton;

import java.awt.Toolkit;
import java.awt.Dimension;
import java.awt.Container;
import java.awt.FlowLayout;

public class TryFlowLayout {
  // The window object
  static JFrame aWindow = new JFrame("This is a Flow Layout");

  public static void main(String[] args) {
    Toolkit theKit = aWindow.getToolkit();         // Get the window toolkit
    Dimension wndSize = theKit.getScreenSize();    // Get screen size
    // Set the position to screen center & size to half screen size
    aWindow.setBounds(wndSize.width/4, wndSize.height/4,   // Position
                      wndSize.width/2, wndSize.height/2);  // Size
    aWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
FlowLayout flow = new FlowLayout();            // Create a layout manager
    Container content = aWindow.getContentPane();  // Get the content pane
    content.setLayout(flow);                       // Set the container layout mgr

    // Now add six button components
    for(int i = 1; i <= 6; i++)
      content.add(new JButton("Press " + i));      // Add a Button to content pane

    aWindow.setVisible(true);                      // Display the window
  }
}
```



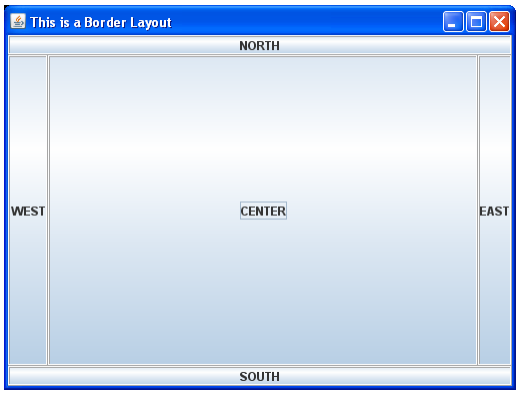
Layout managers

- Grid Layout Manager—It always ignores a component's preferred size
- It divides the whole region into a matrix of rows and columns
- The number of rows and columns are specified as parameters to the constructor
- Every component in the applet in this case is exactly the same size and they appear in the order in which they are added from left to right row by row
- They behave strangely if you put lesser components than number of rows times number of columns or more components. How?

```
GridLayout grid = new GridLayout(3,4,30,20);          // Create a layout manager

JButton button;                                       // Stores a button
   for(int i = 1; i <= 10; i++) {
     content.add(button = new JButton(" Press " + i));    // Add a Button
     button.setBorder(edge);                              // Set the border
   }
```

Layout managers

- **Border Layout Manager—It is the default layout manager for frames**
- **It divides its territory in five regions, North, South, East, West and Center**
- **Each region can contain at the most one component, It may be empty though**
- **This layout manager honors the preferred height of the North and South components and forces them to be exactly as wide as the container**
- **The East and West regions are opposite of North and South. In East and west, a component gets its preferred width but has its height constrained**



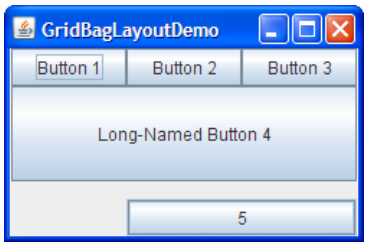

```
JButton button;
    content.add(button = new JButton("EAST"), BorderLayout.EAST);
    button.setBorder(edge);
    content.add(button = new JButton("WEST"), BorderLayout.WEST);
    button.setBorder(edge);
    content.add(button = new JButton("NORTH"), BorderLayout.NORTH);
    button.setBorder(edge);
    content.add(button = new JButton("SOUTH"), BorderLayout.SOUTH);
    button.setBorder(edge);
    content.add(button = new JButton("CENTER"), BorderLayout.CENTER);
    button.setBorder(edge);
```

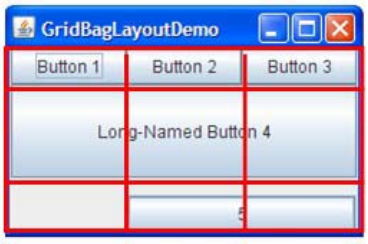
Layout managers

- A component at east or west extends vertically up to the bottom of the North component(if there is one) or to the top of the container(if there is no North component)
- This component extends down to the top of the South component (if there is one) or to the bottom of the container(if there is no South component)
- The border layout is not affected by the order in which you add the components, because you specify which region has to receive this component(by default it is centre)
- You use the overloaded version of add( ) that takes two arguments, one the component to add, other where to add
- You can pass either a string like “North”, Center” or you can use defined constants like BorderLayout.NORTH, BorderLayout.CENTER etc

Layout managers

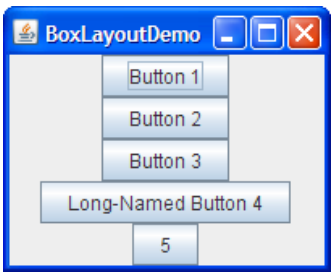
- GridBag Layout Manager—It divides its container into an array of cells but different cells can have different heights and different widths and a component can occupy a single cell or it can span a number of cells





Layout managers

- The Swing packages include a general purpose layout manager named BoxLayout. BoxLayout either stacks its components on top of each other or places them in a row — your choice. You might think of it as a version of Flow Layout, but with greater functionality
- As the box layout arranges components, it takes the components' alignments and minimum, preferred, and maximum sizes into account.



Visual components

- **Button Checkbox Choice**

  FileDialog Label List ScrollPane

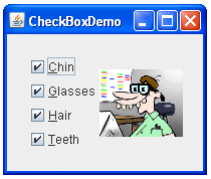
  Scrollbar TextArea TextField
- **To use one of these components you first create an instance by calling the appropriate constructor**
- **Then you add the component to the container**

Visual components

- **Button– Two constructors are available, One with no argument other that takes the label as argument**
  - setLabel( ) and getLabel( ) methods are available in the class Button
- **Checkbox—is a two state button**
  - There are two constructors available

    Checkbox(String label)

    Checkbox(String label, boolean initialState)
  - True state means checked
  - If you don't specify any state the default is false(unchecked)



Visual components

- **Methods available in Checkbox are**
  - boolean getState( )

    void setState(boolean state)
- **Checkboxes can be grouped into check-box groups which have radio behavior**
- **With radio behavior, only one member of a checkbox group can be true at a time**
- **To use check-box group, you need to create an instance of class CheckboxGroup and then pass this instance as parameter to the Checkbox constructor**
  - CheckboxGroup cbg = new checkboxGroup( );

    add(new Checkbox("Hello",true,cbg));
- **Class CheckboxGroup is not a component**

Visual components

- **Label – does not respond to user input**
  - Label( )

    Label(String text)

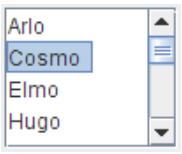
    Label(String text, int alignment)
- **The default alignment for labels is to the left, to set the alignment, pass one of the following to third constructor**
  - Label.LEFT
  - Label.CENTER
  - Label.RIGHT
- **For reading and setting the text of a label**
  - String getText( )
  - Void setText(String newText)

Visual components

- **List – A list is a collection of text items, arranged vertically**
- **If the list contains more items than it can display, it acquires a vertical scroll bar**
  - List( )

    List(int NoofVisibleRows)

    List(int NoofVisibleRows, boolean multiSelectOk)
- **If Multiselect is off, then selecting a new item causes the old selected item to be deselected**
  - List l1=new List(2,true)
  - L1. AddItem(“One”);

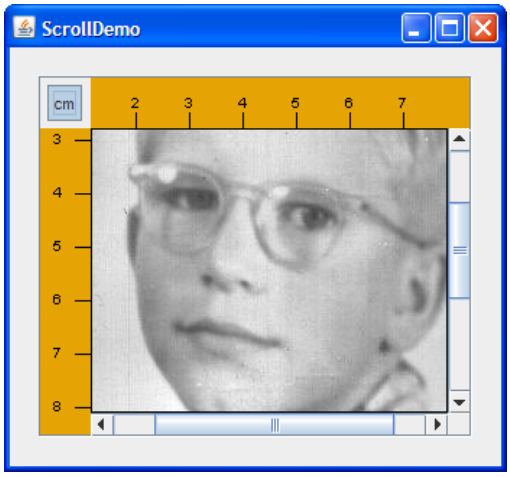


Visual components

- **The List class provides a large number of support methods**
  - Void addItem(String text)
  - void addItem(String text, int index)
  - String getItem(int index)
  - int getItemCount( )
  - int getSelectedIndex( )
  - int[ ] getSelectedIndexes( )
  - Object getSelectedValue( )
  - Object[] getSelectedValues( )

Visual components

- **ScrollPane—can contain a single component which may be taller or wider than the scroll pane itself**
- **If the contained component is larger than the scroll pane, then the default behavior of the scroll pane is to acquire horizontal and/or vertical scroll bars as needed**
- **There are four constructors for this class**
  - JScrollPane( )
  - JScrollPane(Component view)

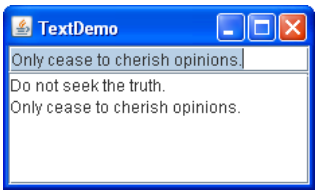


Visual components

- **ScrollBar—there are three constructors for the scrollbar class**
  - **JScrollBar**()
  - **JScrollBar**(int orientation)
    Creates a scrollbar with the specified orientation
  - **JScrollBar**(int orientation, int value, int extent, int min, int max)

Visual components

- **Textfield and TextArea classes extends from the TextComponent Class**
- **They represent one dimensional and two dimensional components for text input, display and editing**
- **The TextField class have the following constructors**
  - JTextField()
    JTextField(String)
    JTextField(String, int)
    JTextField(int)



Visual components

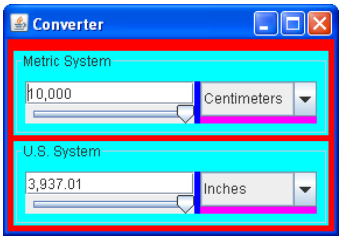
- **The number of columns is a measure of width in terms of columns of text as rendered in a particular font**
- **What happens when a user types beyond the rightmost character column in both the classes. ?**
- **In both cases, the visual text scrolls to the left**
- **Both classes inherit some methods from their common parent class TextComponent**
  - String getSelectedText( )
  - String getText( )
  - void setText(String text)
  - void setEditable(boolean editable)

Visual components

- **The TextArea class have the following constructors**
- **JTextArea()**
  JTextArea(String)
  JTextArea(String, int, int)
  JTextArea(int, int)

Panels

- The JPanel class provides general-purpose containers for lightweight components.
- By default, panels do not add colors to anything except their own background; however, you can easily add borders to them and otherwise customize their painting.



Panel

- As the Converter example demonstrates, panels are useful for grouping components, simplifying component layout, and putting borders around groups of components.
- Like other containers, a panel uses a layout manager to position and size its components. By default, a panel's layout manager is an instance of FlowLayout

```
JPanel p = new JPanel(new BorderLayout());
 JPanel p = new JPanel();
 p.setLayout(new BoxLayout(p, BoxLayout.PAGE_AXIS));
```

- When you add components to a panel, you use the add method. Exactly which arguments you specify to the add method depend on which layout manager the panel uses. When the layout manager is FlowLayout, BoxLayout, GridLayout, or SpringLayout, you will

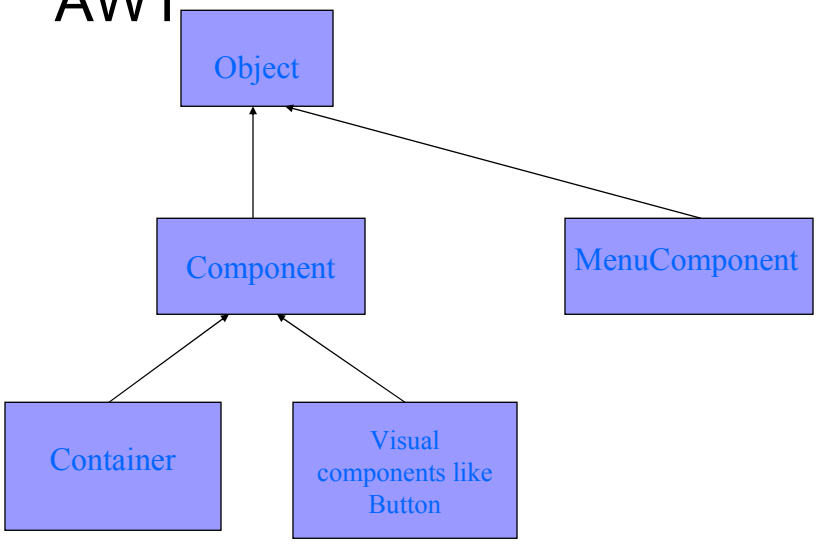
  typically use the one-argument add method, like this:

```
aFlowPanel.add(aComponent);
aFlowPanel.add(anotherComponent);
```

- When the layout manager is BorderLayout, you need to provide an argument specifying the added component's position within the panel. For example:

```
aBorderPanel.add(aComponent, BorderLayout.CENTER);
aBorderPanel.add(anotherComponent, BorderLayout.PAGE_END);
```

AWT



Menu components

- **Concept of using menu is implemented in java using MenuBar, Menu and MenuItem classes**
- **A menu bar displays a list of menu choices**
- **Menu bars may only appear in frames**
- **To create a frame with a menu bar containing drop down menu,**
  - Create a menu bar and attach it to the frame
  - Create and populate Menu
  - Attach menu to the menu bar
- **To create a menu bar, create an instance of MenuBar which only has default constructor**

Menu components

- **Now create instances of Menu that will define the selections,displayed on the bar**
- **Constructors for Menu are**

  JMenu( )

  JMenu(String optionName)
- **Individual menu items are of type MenuItem**
- **Constructors for MenuItem are**

  JMenuItem( )

  JMenuItem(String itemName)

  JMenuItem(String itemName, MenuShortCut keyAccel)

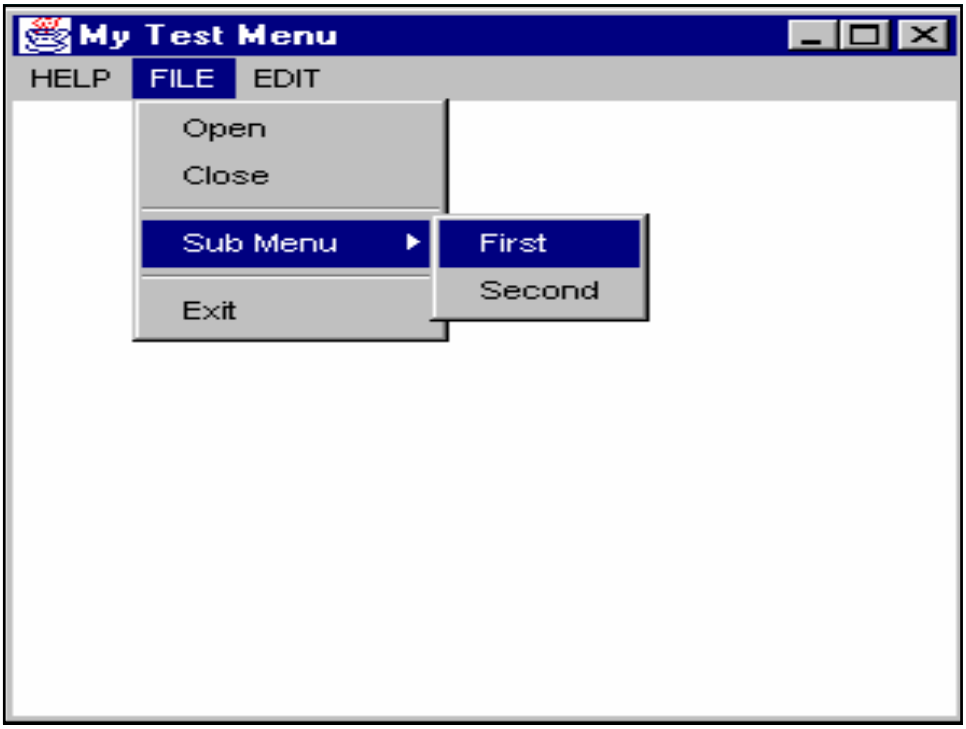
Menu components

- **Methods available for JMenuItem are**
  - void setEnabled(boolean Flag)
  - boolean isEnabled( )
  - void setLabel(String newName)
  - String getLabel( )
- **You can also populate your menu with JCheckboxMenuItem that is a subclass of MenuItem, Constructors for JCheckboxMenuItem are**

  JCheckboxMenuItem( )

  JCheckboxMenuItem(String itemName)

  JCheckboxMenuItem(String optionName, boolean on)
- **Additional methods available for JCheckboxMenuItem are**
  - void setState(boolean Flag) and boolean getState( )



Using Graphics

- **The AWT supports a rich collection of graphics methods.**
- **All graphics are drawn relative to window**
- **The origin of each window is at the top-left corner and is 0,0 and all coordinates are specified in pixels**
- **All output to a window takes place through a graphics context which is an instance of the Graphics class**
- **Major operations provided by the Graphics class are**

  selecting a color

  selecting a font

  Drawing and filling

Using Graphics

- **To use Graphics class you simply provide the argument of paint method the Graphics context**
  - public void paint(Graphics g){

    g.setColor(Color.red);

    g.fillRect(0,0,300,300);

    }

Using Graphics

- **Selecting a color**
- **Colors are selected by calling the setColor( ) method**
- **The argument of setColor( ) is an instance of Color class**
- **Calling setColor( ) only affects subsequent operations**
- **There are 13 pre-defined colors accessible as static final variables of Color class**
  - Color.red
  - Color.blue etc
- **If you want a color that is not in the list than you can construct your own instance of the Color class**
- **You would be using the constructor**

  Color(int redLevel, int greenLevel, int blueLevel)

Using Graphics

- **Selecting a font**
- **fonts are selected by calling the setFont( ) method**
- **The argument of setFont( ) is an instance of Font class**
- **Calling setFont( ) only affects subsequent operations**
- **The constructor for the Font class is**

  Font(String fontName, int style, int size)
- **The style parameter should be on of the following**
  - Font.PLAIN
  - Font.BOLD
  - Font.ITALIC
- **Font f=new Font("Serif", Font.BOLD, 24);**

  g.setFont(f);

Using Graphics

- **Drawing and filling**
- **drawLine(int startX,int startY,int endX,int endY)**
- **drawRect(int top, int left, int width, int height)**
- **fillRect(int top, int left, int width, int height)**
- **drawRoundRect(int top, int left, int width, int height, intxDiam, int yDiam)**
- **fillRoundRect(int top, int left, int width, int height, intxDiam, int yDiam)**
- **drawOval(int top, int left, int width, int height)**
- **fillOval(int top, int left, int width, int height)**

Using Graphics

- **drawArc(int top, int left, int width, int height, int startAngle, int sweepAngle)**
- **fillArc(int top, int left, int width, int height, int startAngle, int sweepAngle)**
- **drawPolygon(int x[ ],int y [ ],int noPoints)**
- **fillPolygon(int x[ ],int y [ ],int noPoints)**
- **drawPolyline(int x[ ],int y [ ],int noPoints)**
- **drawString(string s, int x, int y)**

```
public class MoveBall extends Applet
{ int xpos=10; int ypos=50; int xinc=2; int yinc=2;
int xtop=20; int ytop=20; int xbot=370; int ybot=270; Color color;
public void init()  { color=Color.red;              }
public Color genColor() {
return new Color((float)Math.random(),(float)Math.random(),(float)Math.random()); }
public void paint(Graphics g){
                        if (xpos>=xbot){   color=genColor(); xinc= -2; }
                        else if(xpos<=xtop) { color=genColor(); xinc=2; }
                        if (ypos>=ybot) { color=genColor(); yinc=-2;  }
                        else if(ypos<=ytop) { color=genColor(); yinc=2; }
                        xpos+=xinc; ypos+=yinc;
                        g.setColor(color);
                        g.fillOval(xpos,ypos,15,15);
                        repaint(10); } }
```