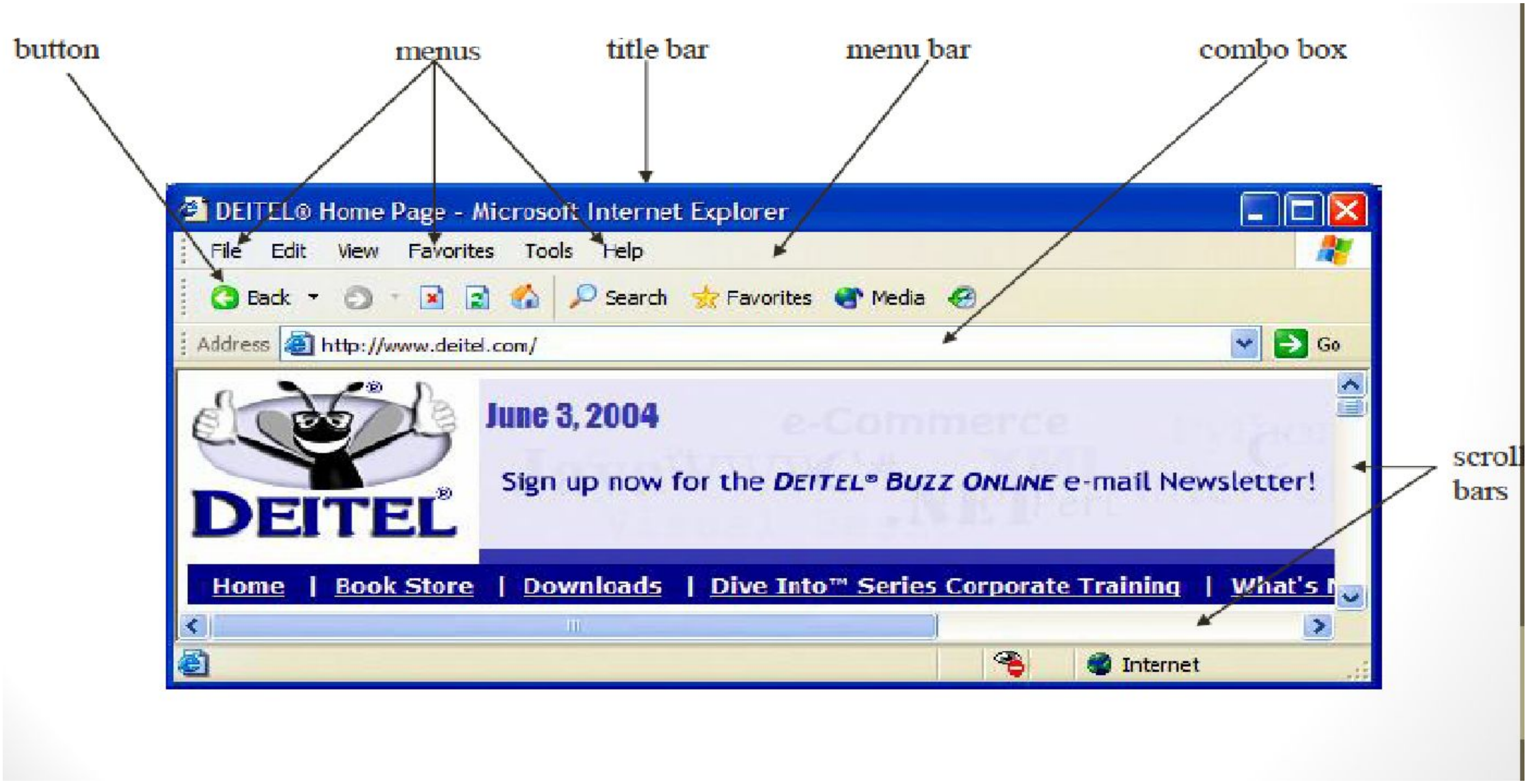
Abstract Windowing Toolkit
AWT

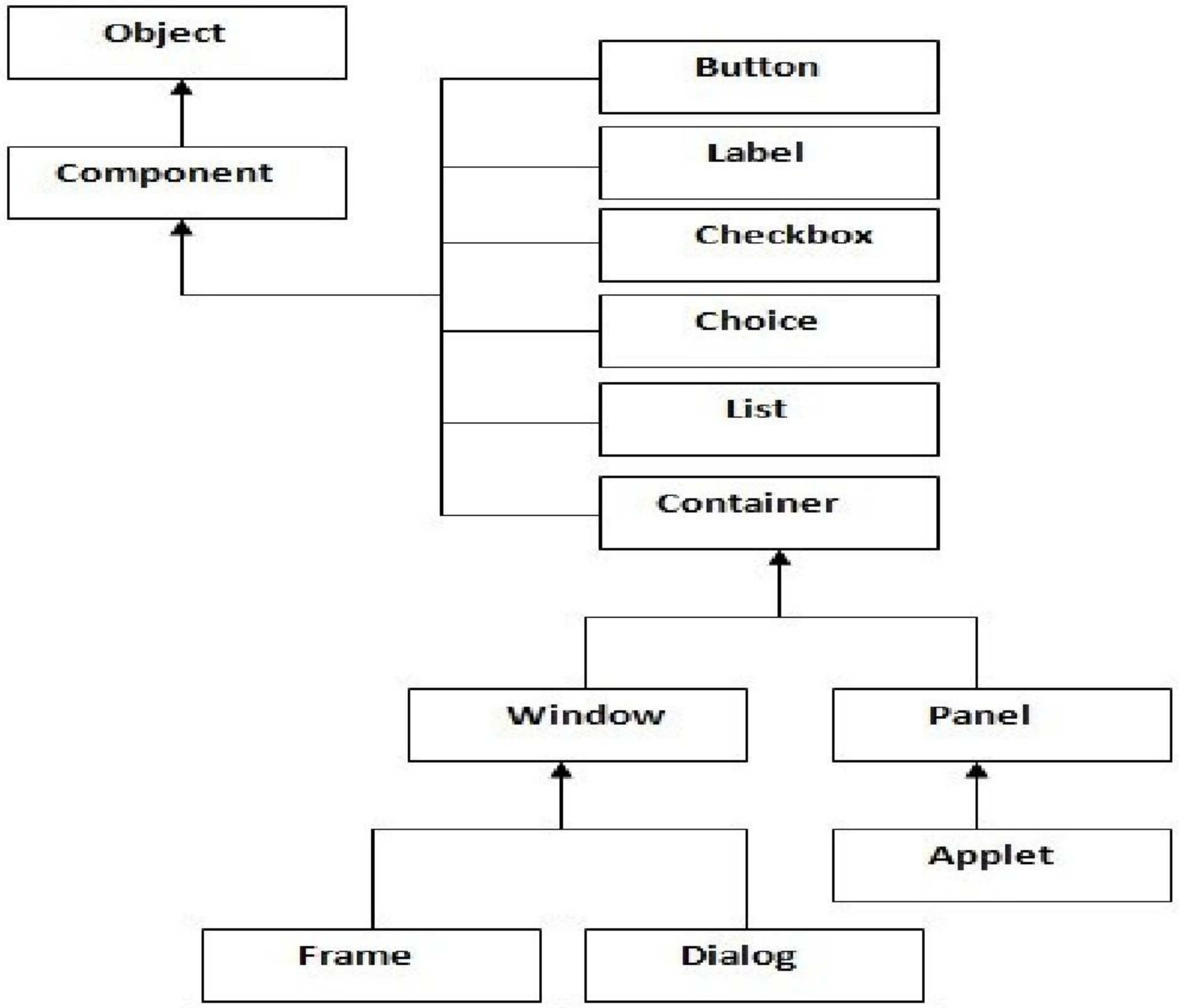
Introduction

- Java AWT (Abstract Windowing Toolkit) is an API to develop GUI or window-based application in java.
- Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system.
- AWT is **heavyweight** i.e. its components uses the resources of system.
- The java.awt package provides classes for AWT API such as **TextField**, **Label**, **TextArea**, **RadioButton, CheckBox, Choice, List** etc.

GUI


button
menus
title bar
menu bar
combo box
scroll bars
DEITEL® Home Page - Microsoft Internet Explorer
File
Edit
View
Favorites
Tools
Help
Back
Search
Favorites
Media
Address
http://www.deitel.com/
Go
June 3, 2004
Sign up now for the DEITEL® BUZZ ONLINE e-mail Newsletter!
DEITEL®
Home | Book Store | Downloads | Dive Into™ Series Corporate Training | What's N
Internet

AWT Hierarchy



Object

- The Object class is the top most class and parent of all the classes in java by default.
- Every class in java is directly or indirectly derived from the object class.

Component

- The Component is abstract class that encapsulates all the attributes of visual component.
- All User interface (UI) elements that are displayed on screen are subclasses of Component.

Component is responsible for remembering the current foreground and background color and the currently selected text font.

Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width, int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.
void remove(Component c)	Remove a component
void setBounds(int x,int y, int width, int height)	Set the location and size of single component and useful only with null layout.

Container

- The Container is a component in AWT that can contain another components like buttons, textfields, labels etc.
- The classes that extends Container class are known as container such as Frame, Dialog and Panel.
- Container is **responsible for laying out any components** that it contains through the use of layout managers.
- **Methods:**
    - void setFont (Font f)
    - void setLayout(LayoutManager mgr)

Panel

- Panel class is concrete class it doesn’t add new methods.
- The Panel is the container that **doesn't contain title bar and menu bars and Borders**.
- It can have other components like button, textfield etc.

An Applet is Panel is a Container

```
java.lang.Object
    | +----java.awt.Component
          | +----java.awt.Container
               | +----java.awt.Panel
                    | +----java.applet.Applet
```

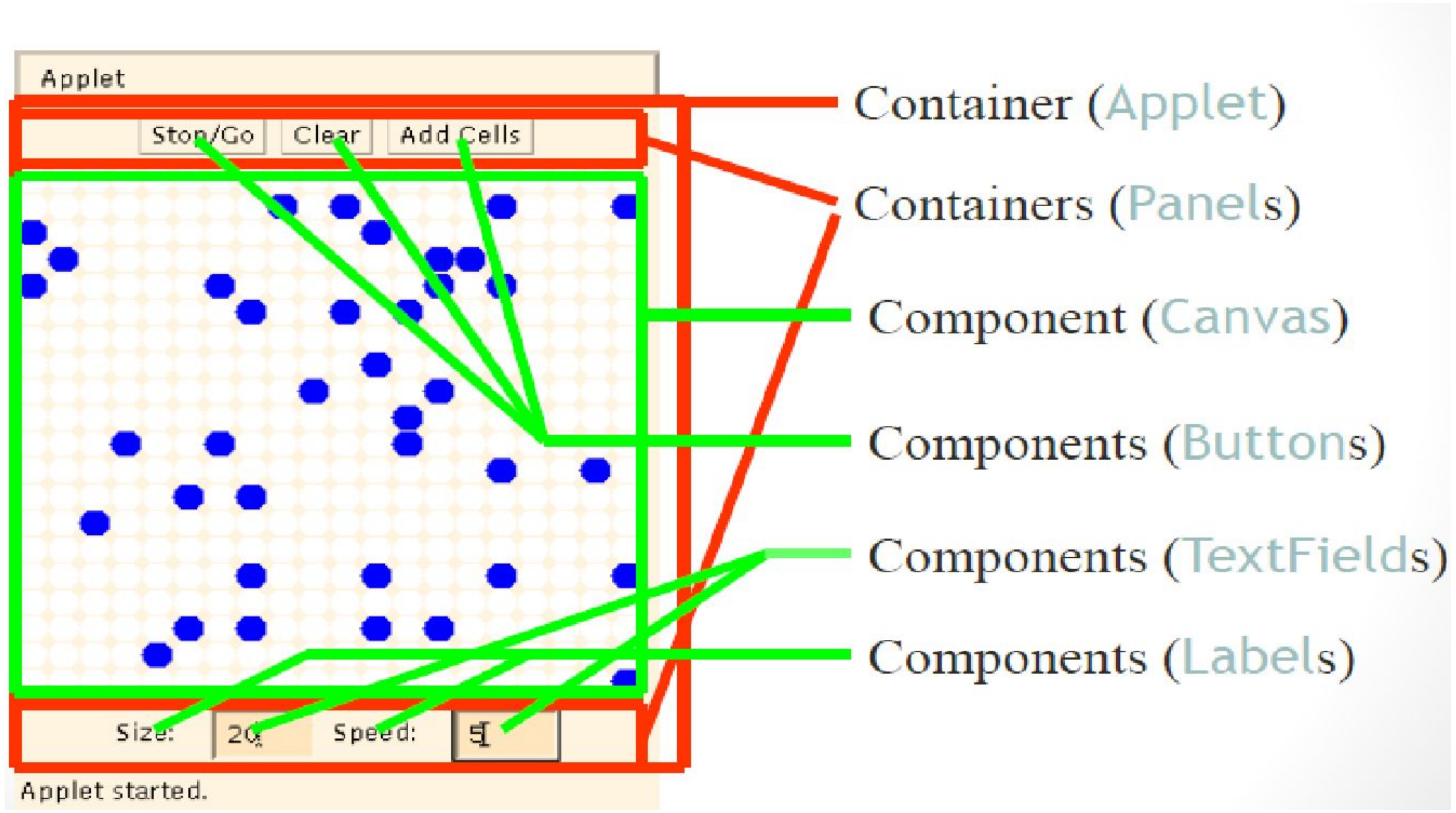
An Applet

- Applet is a public class which is predefined by **java.applet.Applet**
- There is no main() method in Applet like Application program. The main() method is defined by browser or Appletviewer for Applet.
- Life cycle methods: init, start, paint, stop, destroy
- Applet is one type of container and subclass of Panel.

To create an applet

- import java.applet.*;
- Import java.awt.*;
- Applet tag code in comment.
- Extends Applet class
- Life cycle method
- Class must be public

Applet Life Cycle



Window

- The window is the container that have **no borders and menu bars.**
- You must use frame, dialog or another window for creating a window.

Frame

- It is subclass of **Window**.
- The Frame is the container that contain title bar and can have menu bars,borders, and resizing corners.
- It can have other components like button, textfield, etc.
- Methods:
  - void setTitle(String title)
  - void setBackground(Color bgcolor)

Working with Frame Window

- Extends Frame class
- Constructor are:
    - Frame()
    - Frame(String title)
- Setting and Getting window size:
    - void setSize(int width, int height)
    - void setSize(Dimension newsize)
- Showing and Hiding Frame
    - void setVisible(boolean visibleFlag)

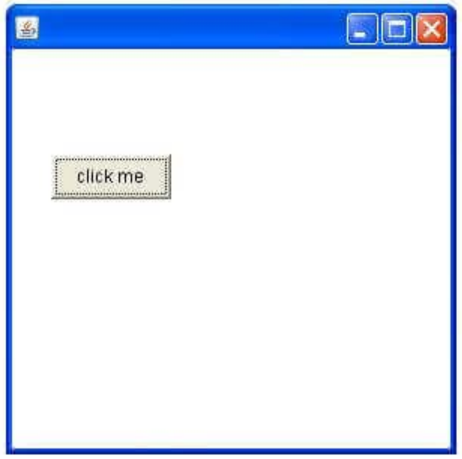
Frame Class

- We can create stand-alone AWT based applications.
- A Frame provides main window for the GUI application.
- There are two ways to create a Frame :
  1. By instantiating Frame Class
  2. By extending Frame class

Program using Frames

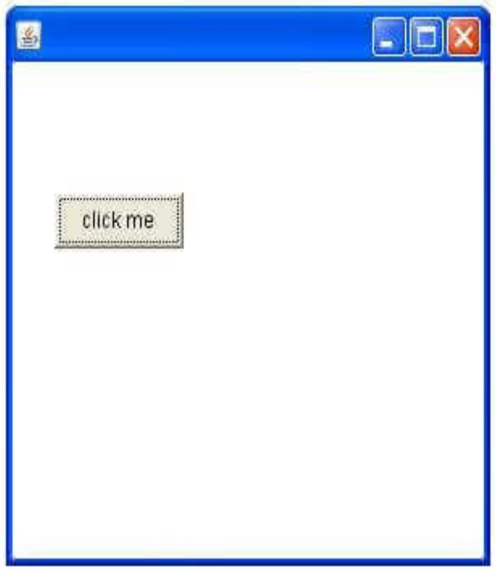
```
import java.awt.*;
class FirstFrame{
FirstFrame(){
Frame f=new Frame();
Button b=new Button("click me
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true); }
```

```
public static void main(String args[]){

FirstFrame f=new FirstFrame();
}
}
```



Program using Frames

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);
add(b);
setSize(300,300);
setLayout(null);
setVisible(true);
}
public static void main
(String args[]){
First f=new First();
}
}
```



Controls

- Labels
- Buttons
- Checkbox
- CheckboxGroup
- Textfield
- TextFieldArea
- ScollBar

Label

- The easiest control to use is a label.
- A label is an object of type Label, and it contains a string, which it displays.
- Labels are passive controls that do not support any interaction with the user.

Labels

- **Label defines the following constructors:**
- **Label( )**
- **Label(String str)**
- **Label(String str, int how)**
- The first version creates a blank label.
- The second version creates a label that contains the string specified by *str*. This string is left-justified.
- The third version creates a label that contains the string specified by *str* using the alignment specified by *how*. The value of how must be one of these three constants: Label.LEFT, Label.RIGHT, or Label.CENTER.

Label

- Methods
- void setText(String str)
- String getText( )
- void setAlignment(int how)
- int getAlignment( )

Controls

```
// Demonstrate Labels
import java.awt.*;
import java.applet.*;
/*
<applet code="LabelDemo"
width=300 height=200>
</applet>
*/
public class LabelDemo
extends Applet
{
public void init()
{
Label one = new Label("One");
Label two = new Label("Two");
Label three = new Label("Three");
// add labels to applet window
add(one);
add(two);
add(three);
}
}
```

Buttons

- The most widely used control is the push button.
- A push button is a component that contains a label and that generates an event when it is pressed.
- Push buttons are objects of type Button.
- Button defines these two constructors:
- **Button( )**
- **Button(String str)**

Buttons

- String getLabel()
- void setLabel(String str)
- void setEnabled(Boolean enable)
- Void addActionListener(ActionListener l)
- void removeActionListener(ActionListener l)
- String getActionCommand()
- void setActionCommand(String Cmd)

```
// Demonstrate Buttons
import java.awt.*;
import java.applet.*;
/*
<applet code="ButtonDemo"
width=250 height=150>
</applet>
*/
public class ButtonDemo extends
Applet
{
String msg = "";
Button yes, no, maybe;
public void init()
{
yes = new Button("Yes");
no = new Button("No");
maybe = new Button("Understand");
add(yes);
add(no);
add(maybe);
}
public void paint(Graphics g)
{
g.drawString(msg, 6, 100);
}
}
```

Check Boxes

- A check box is a control that is used to turn an option on or off.
- It consists of a small box that can either contain a check mark or not.
- There is a label associated with each check box that describes what option the box represents.
- We change the state of a check box by clicking on it. Check boxes can be used individually or as part of a group.

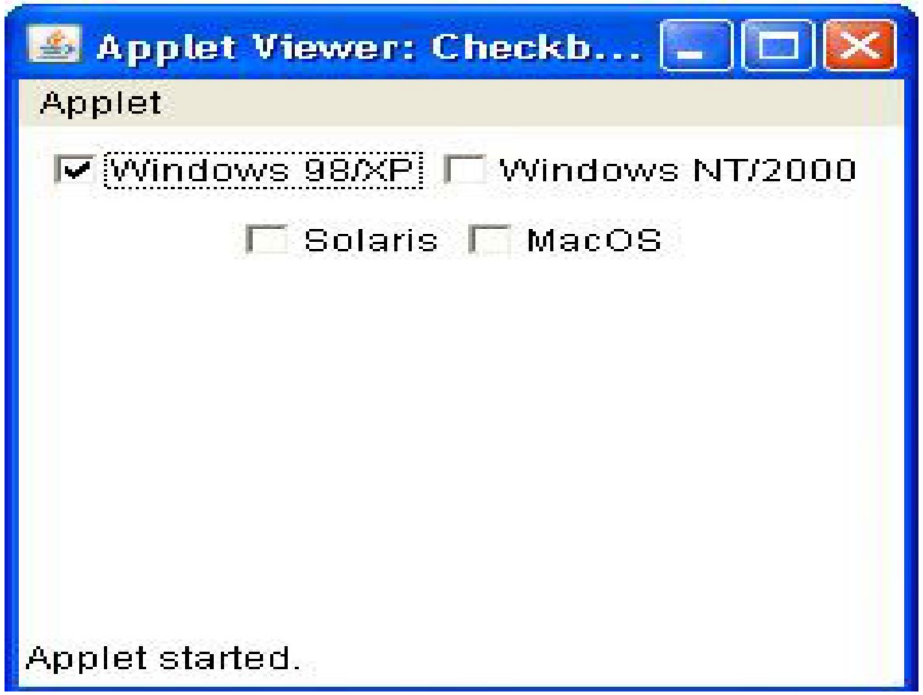
Checkbox constructors:

- Checkbox( )
- Checkbox(String str)
- Checkbox(String str, boolean on)
- Checkbox(String str, boolean on, CheckboxGroup cbGroup)
- Checkbox(String str, CheckboxGroup cbGroup, boolean on)

Methods

- boolean getState( )
- void setState(boolean on)
- String getLabel( )
- void setLabel(String str)
- void addItemListener(ItemListener l)
- void removeItemListener(ItemListener l)

```
// Demonstrate check boxes.
import java.awt.*;
import java.applet.*;
/*
<applet code="CheckboxDemo"
width=250 height=200>
</applet>
*/
public class CheckboxDemo extends
Applet
{
String msg = “Hello";
Checkbox Win98, winNT, solaris, mac;
public void init()
{
Win98 = new Checkbox("Windows 98/XP",
null, true);
winNT = new Checkbox("Windows
NT/2000");
solaris = new Checkbox("Solaris");
mac = new Checkbox("MacOS");
add(Win98);
add(winNT);
add(solaris);
add(mac);
}
public void paint(Graphics g)
{g.drawString( msg, 100,100);}
}
```


Applet Viewer: Checkb...
Applet
[✓] Windows 98/XP
[ ] Windows NT/2000
[ ] Solaris
[ ] MacOS
Applet started.

Checkbox Group

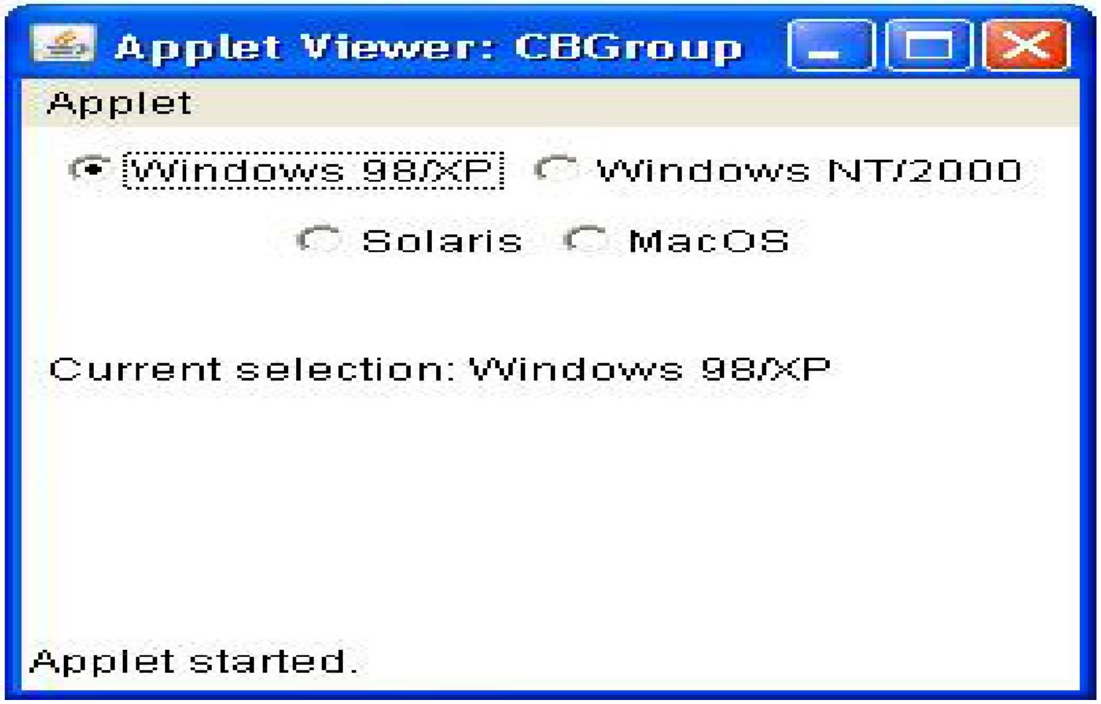
- It is possible to create a set of mutually exclusive check boxes in which one and only one check box in the group can be checked at any one time.
- These check boxes are often called radio button.
- Check box groups are objects of type **CheckboxGroup**.
- Only the default constructor is defined, which creates an empty group.

Methods

Checkbox getSelectedCheckbox( )

void setSelectedCheckbox(Checkbox wh)

```
import java.awt.*;
import java.applet.*;
/*
<applet code="CBGroup"
width=250 height=200>
</applet>
*/
public class CBGroup extends Applet
{
String msg = "";
Checkbox Win98, winNT,
solaris, mac;
CheckboxGroup cbg;
public void init()
{
cbg = new CheckboxGroup();
Win98 = new Checkbox("Windows 98/XP", cbg,
true);
winNT = new Checkbox("Windows NT/2000",
cbg, false);
solaris = new Checkbox("Solaris", cbg,
false);
mac = new Checkbox("MacOS", cbg, false);
add(Win98); add(winNT);
add(solaris); add(mac);}
public void paint(Graphics g)
{
msg = "Current selection: ";
msg +=
cbg.getSelectedCheckbox().getLabel();
g.drawString(msg, 6, 100);
}}
```



(● Windows 98/XP) (Windows NT/2000) (Solaris) (MacOS)

Current selection: Windows 98/XP

Applet started.

Choice Controls

- The Choice class is used to create a pop-up list of items from which the user may choose.
- Thus, a Choice control is a form of menu.
- Each item in the list is a string that appears as a left justified label in the order it is added to the Choice object.
- Like a list but only one option can be selected from list.

Methods

void add(String name)

String getSelectedItem( )

int getSelectedIndex( )

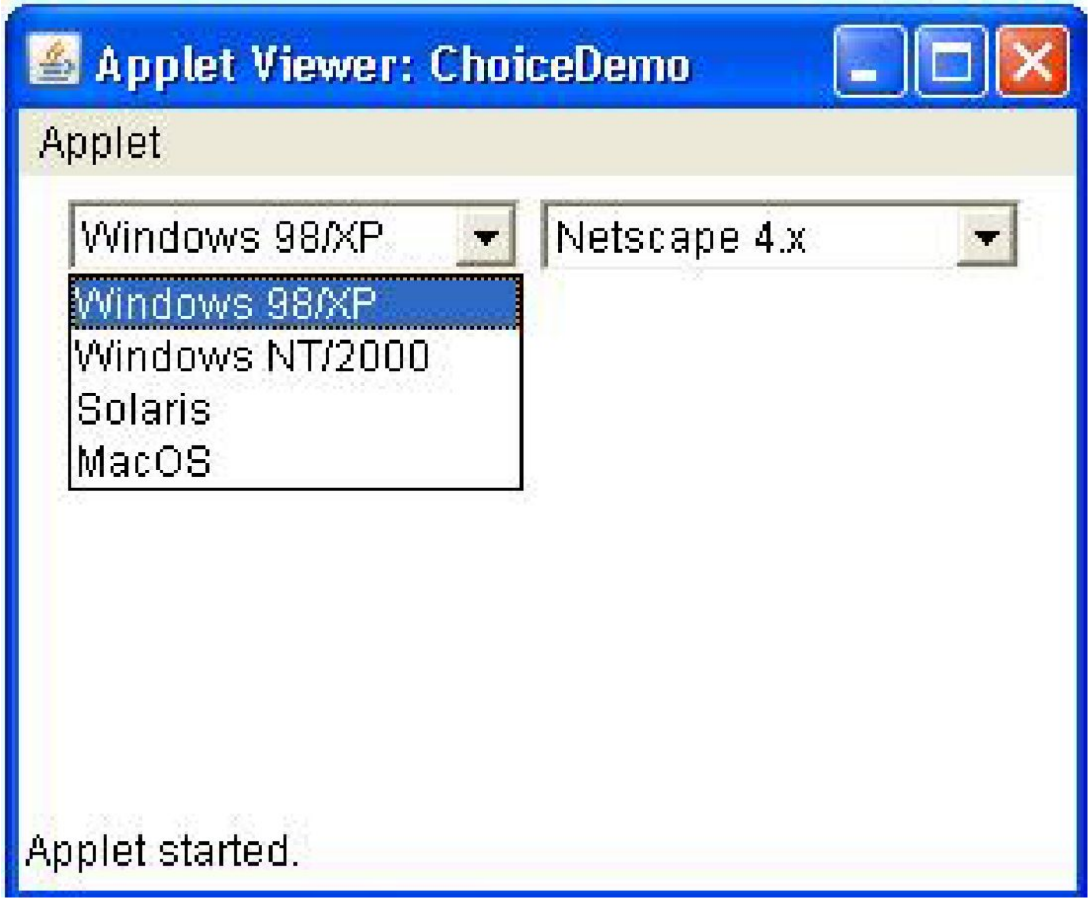
int getItemCount( )

void select(int index)

void select(String name)

String getItem(int index)

```
import java.awt.*;
import java.applet.*;
/*
<applet code="ChoiceDemo"
width=300 height=180>
</applet>
*/
public class ChoiceDemo extends
Applet
{
Choice os, browser;
String msg = "";
public void init()
{
os = new Choice();
browser = new Choice();
os.add("Windows 98/XP");
os.add("Windows NT/2000");
os.add("Solaris");
os.add("MacOS");
browser.add("Netscape 3.x");
browser.add("Netscape 4.x");
browser.add("Netscape 5.x");
browser.add("Netscape 6.x");
browser.add("Internet Explorer 4.0");
browser.add("Internet Explorer 5.0");
browser.add("Internet Explorer 6.0");
browser.add("Lynx 2.4");
browser.select("Netscape 4.x");
add(os);
add(browser);
}
public void paint(Graphics g)
{}}
```


Applet Viewer: ChoiceDemo
Applet
Windows 98/XP
Netscape 4.x
Windows 98/XP
Windows NT/2000
Solaris
MacOS
Applet started.

Lists

- The List class provides a compact, multiple-choice, scrolling selection list.
- Unlike the Choice object, which shows only the single selected item in the menu, a List object can be constructed to show any number of choices in the visible Window.
- It can also be created to allow multiple selections.

List

- **List( )**
- **List(int numRows)**
- **List(int numRows, boolean multipleSelect)**

- The first version creates a List control that allows only one item to be selected at any one time.
- In the second form, the value of *numRows* specifies the number of entries in the list that will always be visible (others can be scrolled into view as needed).
- In the third form, if *multipleSelect* is true, then the user may select two or more items at a time.

Methods

void add(String name)

void add(String name, int index)

String getSelectedItem( )

int getSelectedIndex( )

String[ ] getSelectedItems( )

int[ ] getSelectedIndexes( )

int getItemCount( )

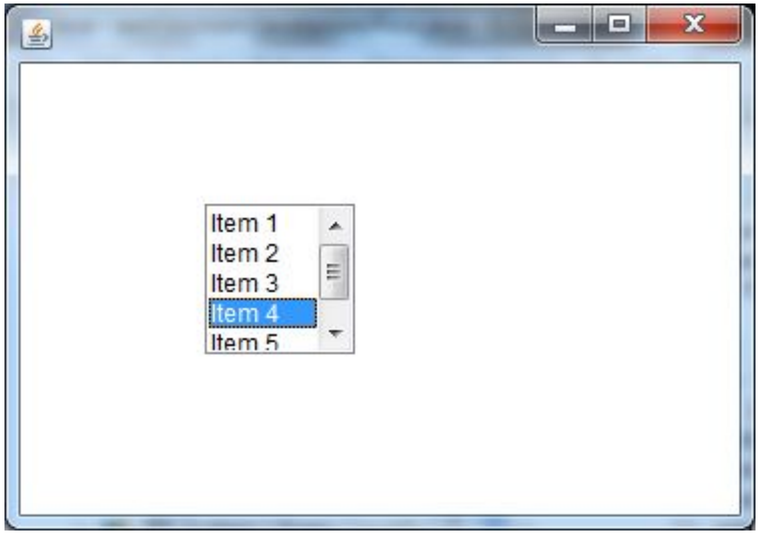
void select(int index)

String getItem(int index)

List Example

```
/ importing awt class
import java.awt.*;

public class ListExample1
{
   // class constructor
    ListExample1() {
    // creating the frame
      Frame f = new Frame();
     // creating the list of 5 rows
      List l1 = new List(5);

      // setting the position of list component
      l1.setBounds(100, 100, 75, 75);

      // adding list items into the list
      l1.add("Item 1");
      l1.add("Item 2");
      l1.add("Item 3");
      l1.add("Item 4");
      l1.add("Item 5");

      // adding the list to frame
      f.add(l1);
```

```
  / setting size, layout and visibility of frame
   f.setSize(400, 400);
   f.setLayout(null);
   f.setVisible(true);
  }

// main method
public static void main(String args[])
{
  new ListExample1();
}
}
```


Item 1
Item 2
Item 3
Item 4
Item 5

ScrollBars

- Scroll bars are used to select continuous values between a specified minimum and maximum.
- Scroll bars may be **oriented horizontally or vertically.**
- A scroll bar is actually a composite of several individual parts.
    - slider box (or thumb) for the scroll bar.
- The slider box can be dragged by the user to a new position, this action translates into some form of **page up and page down.**

Constructors

- **Scrollbar( )**
- **Scrollbar(int style)**
- **Scrollbar(int style, int iValue, int tSize, int min, int max)**

- The first form creates a **vertical scroll bar**.
- The second and third forms allow us to specify *style* **Scrollbar.VERTICAL, Scrollbar.HORIZONTAL**.
- In the third form, the **initial value** of the scroll bar is passed in ***iValue*.** The number of units represented by the **height of the thumb** is passed in ***tSize***. The minimum and maximum values for the scroll bar are specified by min and max.

Methods

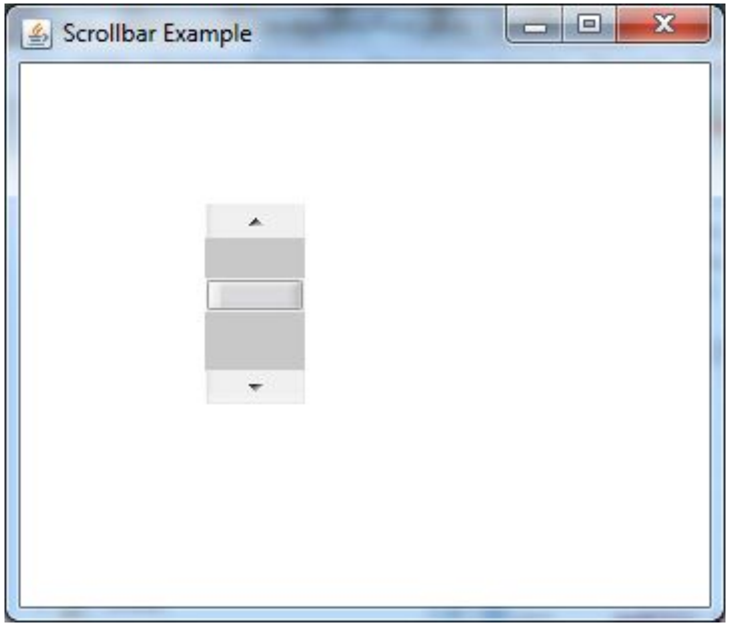
```
void setValues(int iValue, int tSize, int min, int max)
int getValue( )
void setValue(int newValue)
int getMinimum( )
int getMaximum( )
void setUnitIncrement(int newIncr)
void setBlockIncrement(int newIncr)
```

Scrollbar Example

```
// importing awt package
import java.awt.*;
 public class ScrollbarExample1 {
 // class constructor
ScrollbarExample1() {

    // creating a frame
      Frame f = new Frame("Scrollbar Example");

    // creating a scroll bar
      Scrollbar s = new Scrollbar();

    // setting the position of scroll bar
      s.setBounds (100, 100, 50, 100);

    // adding scroll bar to the frame
      f.add(s);

    // setting size, layout and visibility of frame
      f.setSize(400, 400);
      f.setLayout(null);
      f.setVisible(true);
}
```

```
   // main method
public static void main(String args[]) {

    new ScrollbarExample1();
}
}
```

Output:



TextField

- The TextField class implements a single-line text-entry area, called an **edit control**.
- Text fields allow the user to enter strings and to edit the text using the arrow keys, cut and paste keys, and mouse selections.
- TextField is a subclass of **TextComponent.**

TextField Constructors

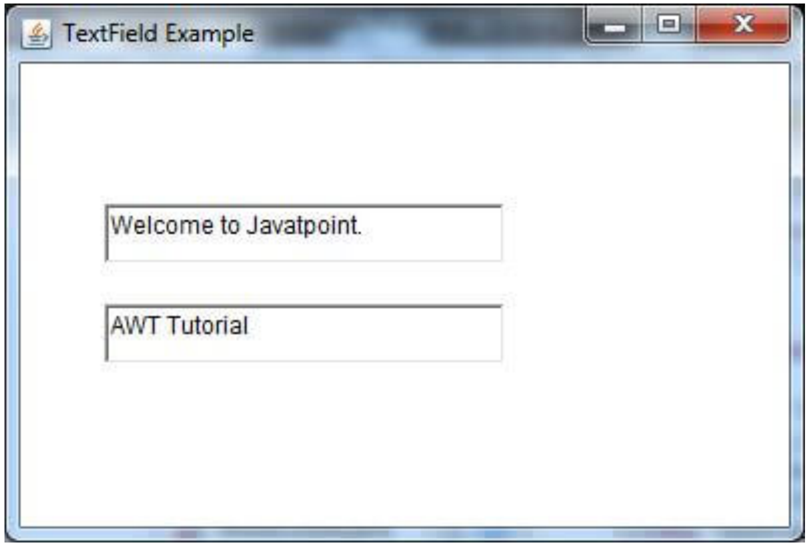
- TextField( )
- TextField(int numChars)
- TextField(String str)
- TextField(String str, int numChars)

TextField Methods

- String getText( )
- void setText(String str)
- String getSelectedText( )
- void select(int startIndex, int endIndex)
- boolean isEditable( )
- void setEditable(boolean canEdit)
- void setEchoChar(char ch)
- boolean echoCharIsSet( )
- char getEchoChar( )

Text field Example

```
// importing AWT class
import java.awt.*;
public class TextFieldExample1 {
   // main method
   public static void main(String args[]) {
   // creating a frame
   Frame f = new Frame("TextField Example");

   // creating objects of textfield
   TextField t1, t2;
   // instantiating the textfield objects
   // setting the location of those objects in the frame

   t1 = new TextField("Welcome to Javatpoint.");
   t1.setBounds(50, 100, 200, 30);
   t2 = new TextField("AWT Tutorial");
   t2.setBounds(50, 150, 200, 30);

  // adding the components to frame
   f.add(t1);
   f.add(t2);

     // setting size, layout and visibility of frame

    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```



Welcome to Javatpoint.

AWT Tutorial

TextArea

- Sometimes a single line of text input is not enough for a given task. To handle these situations, the AWT includes a simple multiline editor called TextArea.
- Following are the constructors for TextArea:
    - TextArea( )
    - TextArea(int numLines, int numChars)
    - TextArea(String str)
    - TextArea(String str, int numLines, int numChars)
    - TextArea(String str, int numLines, int numChars, int sBars)

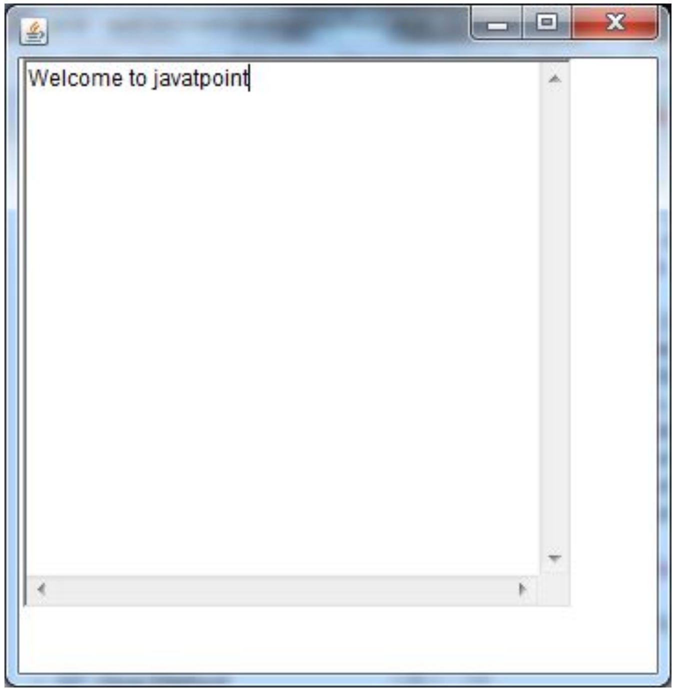
sBars must be one of these values: **SCROLLBARS_BOTH, SCROLLBARS_NONE,SCROLLBARS_HORIZONTAL_ONLY, SCROLLBARS_VERTICAL_ONLY**

Methods

- TextArea is a subclass of TextComponent.
- Therefore, it supports the getText( ), setText( ), getSelectedText( ), select( ), isEditable( ), and setEditable( ) methods as of TextField.
- TextArea adds the following methods:
- **void append(String str)**
- **void insert(String str, int index)**
- **void replaceRange(String str, int startIndex, int endIndex)**

Text Area Example

```
//importing AWT class
import java.awt.*;
public class TextAreaExample
{
// constructor to initialize
   TextAreaExample() {
// creating a frame
      Frame f = new Frame();
// creating a text area
         TextArea area = new TextArea("Welcome to ja
vatpoint");
// setting location of text area in frame
      area.setBounds(10, 30, 300, 300);
// adding text area to frame
      f.add(area);
// setting size, layout and visibility of frame
      f.setSize(400, 400);
        f.setLayout(null);
        f.setVisible(true);
      }
// main method
public static void main(String args[])
{
  new TextAreaExample();
}
}
```



Welcome to javatpoint

Layout Managers

- Layout means arranging the components within the container.
- The task of lay outing can be done automatically by the Layout manager.
- The layout manager is set by the setLayout( ) method.
- If no call to setLayout( ) is made, then the default layout manager is used.
- Whenever a container is resized (or sized for the first time), the layout manager is used to position each of the components within it.

- The setLayout( ) method has the following general form:
- void setLayout(LayoutManager layoutObj)
- Here, *layoutObj* is a reference to the desired layout manager.
- If we wish to disable the layout manager and position components manually, pass null for *layoutObj*.

LayoutManager

- **LayoutManager** is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:
- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

FlowLayout

- FlowLayout is the default layout manager.
- FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
- Components are laid out from the upper-left corner, left to right and top to bottom.
- When no more components fit on a line, the next one appears on the next line.
- A small space is left between each component, above and below, as well as left and right.

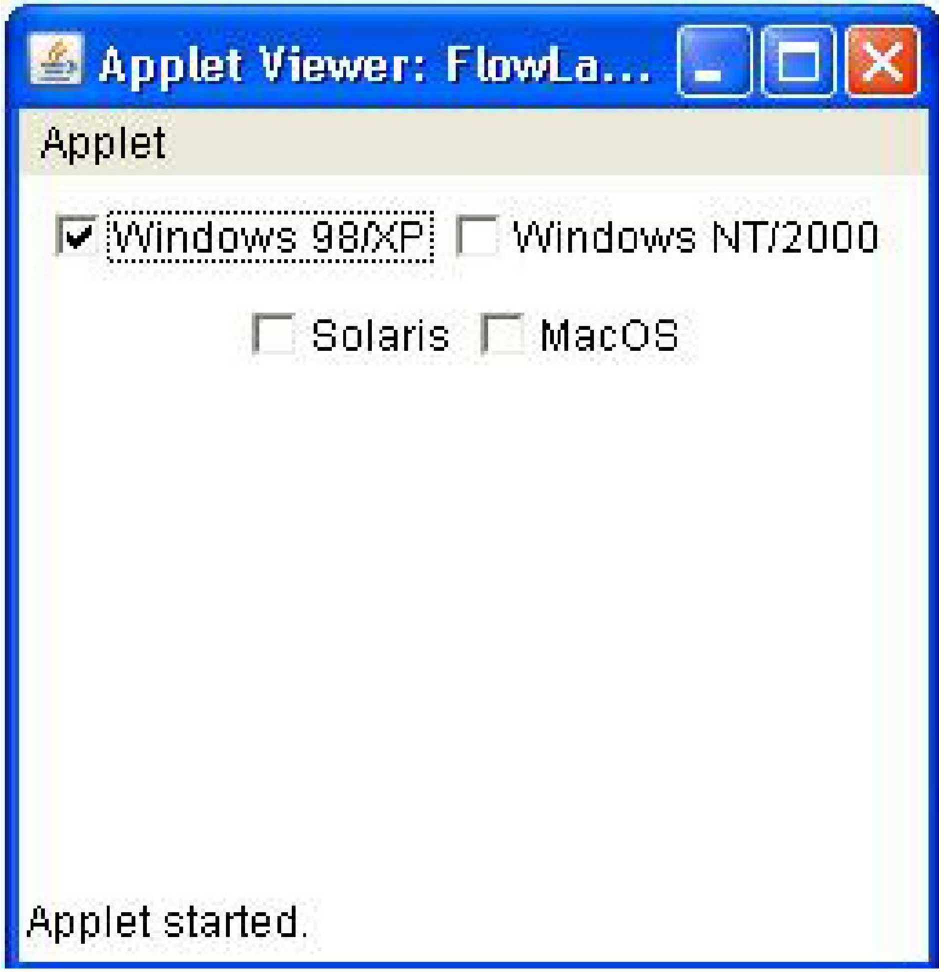
FlowLayout Constructors

- **FlowLayout( )**
- **FlowLayout(int how)**
- **FlowLayout(int how, int horz, int vert)**
- The first is default, which centers components and leaves **five pixels** of space between each component.
- The second form lets us specify how each line is aligned. Valid values for how are as follows:
    - FlowLayout.LEFT
    - FlowLayout.CENTER
    - FlowLayout.RIGHT
- The third form allows us to specify the horizontal and vertical space left between components

FlowLayout Methods

- int getAlignment()
- int getHgap()
- int getVgap()
- int setAlignment(int align)
- int setHgap(int hgap)
- int setVgap(int vgap)

```
public class FlowLayoutDemo extends Applet
{
Checkbox Win98, winNT, solaris, mac;
public void init()
{
Win98 = new Checkbox("Windows 98/XP", null, true);
winNT = new Checkbox("Windows NT/2000");
solaris = new Checkbox("Solaris");
mac = new Checkbox("MacOS");
setLayout(new FlowLayout(FlowLayout.CENTER));
add(Win98); add(winNT);add(solaris);add(mac);
}}
```



Applet started.

BorderLayout

- The BorderLayout class implements a common layout style for top-level windows.
- It has four narrow, fixed-width components at the edges and one large area in the center.
- The four sides are referred to as
    - north,
    - south,
    - east, and
    - west.
    - The middle area is called the center.

BorderLayout Constructors

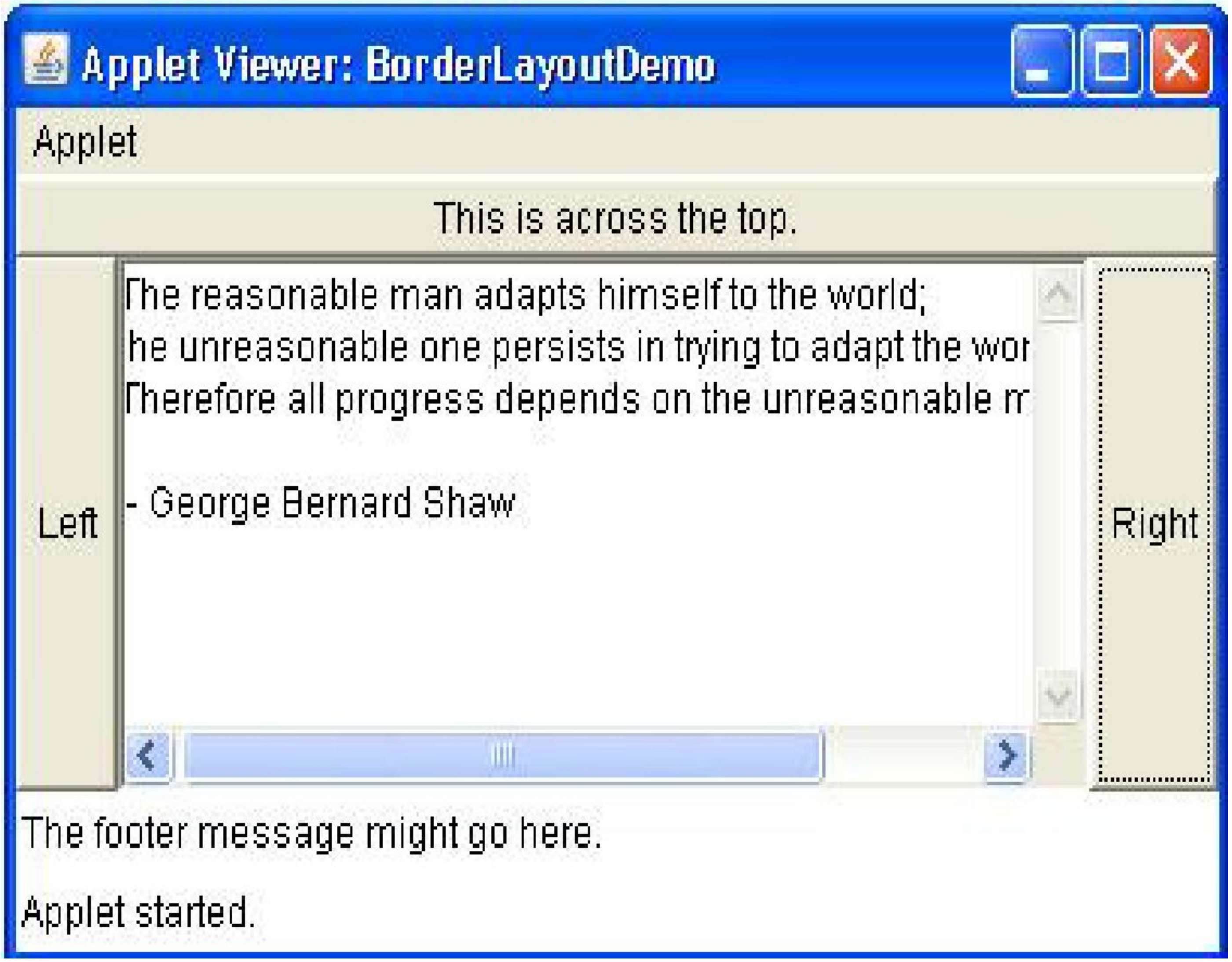
- BorderLayout( )
- BorderLayout(int horz, int vert)

- The first form creates a default border layout.
- The second allows us to specify the horizontal and vertical space left between components in *horz* and *vert*, respectively.

BorderLayout

- BorderLayout defines the following constants that specify the regions:
- **BorderLayout.CENTER**
- **BorderLayout.SOUTH**
- **BorderLayout.EAST**
- **BorderLayout.WEST**
- **BorderLayout.NORTH**
- To add components, we use these constants with the following form of add( ), which is defined by Container:
- **void add(Component compObj, Object region);**
- Here, *compObj* is the component to be added, and region specifies where the component will be added.

```
public class BorderLayoutDemo extends Applet {
public void init() {
setLayout(new BorderLayout());
add(new Button("This is across the top."), BorderLayout.NORTH);
add(new Label("The footer message."), BorderLayout.SOUTH);
add(new Button("Right"), BorderLayout.EAST);
add(new Button("Left"), BorderLayout.WEST);
String msg = "The reasonable man adapts himself to the world;\n" +
"the unreasonable one persists in trying to adapt the world to
himself.\n" +
"Therefore all progress depends on the unreasonable man.\n\n" + " -
George Bernard Shaw\n\n";
add(new TextArea(msg), BorderLayout.CENTER);
}
}
```



```
The reasonable man adapts himself to the world;
he unreasonable one persists in trying to adapt the wor
Therefore all progress depends on the unreasonable m

- George Bernard Shaw
```

The footer message might go here.

Applet started.

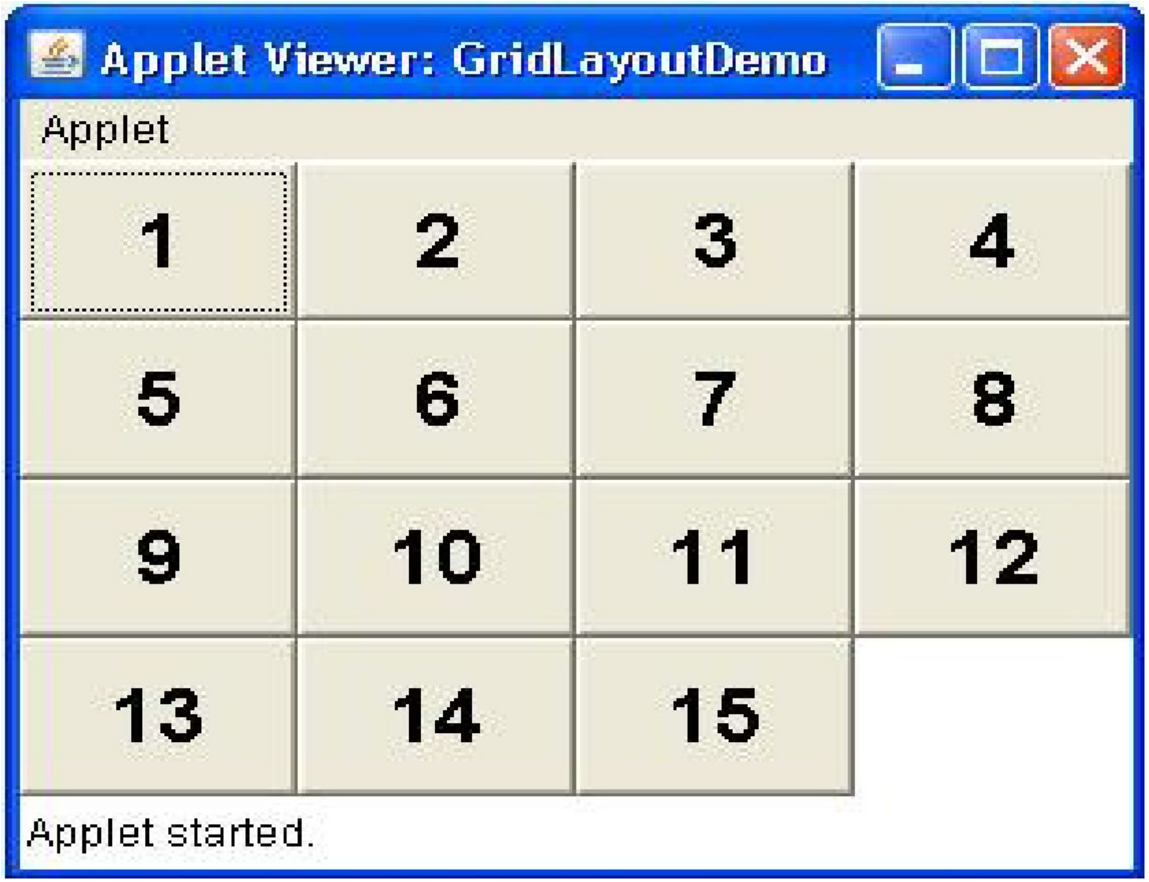
GridLayout

- GridLayout lays out components in a two-dimensional grid.
- When we instantiate a GridLayout, we define the number of rows and columns.

GridLayout constructors

- GridLayout( )
- GridLayout(int numRows, int numColumns)
- GridLayout(int numRows, int numColumns, int horz, int vert)
- The first form creates a single-column grid layout.
- The second creates a grid layout with specified number of rows and columns.
- **Either numRows or numColumns can be zero**.
- Specifying *numRows* as zero allows for unlimited-length columns.
- Specifying *numColumns* as zero allows for unlimited-length rows.

```
public class GridLayoutDemo extends Applet {
static final int n = 4;
public void init(){
setLayout(new GridLayout(n, n));
setFont(new Font("SansSerif", Font.BOLD, 24));
for(int i = 0; i < n; i++){
for(int j = 0; j < n; j++){
int k = i * n + j;
if(k > 0)
add(new Button("" + k));
} }}}
```


Applet Viewer: GridLayoutDemo
Applet
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Applet started.

CardLayout

- The CardLayout class is unique among the other layout managers in that it stores several different layouts.
- Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on top at a given time.
- This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.
- We can prepare the other layouts and have them hidden, ready to be activated when needed.

- CardLayout provides these two constructors:
- **CardLayout( )**
- **CardLayout(int horz, int vert)**
- The first form creates a default card layout.
- The second form allows us to specify the horizontal and vertical space left between components.

Methods

- void add(Component panelObj, Object name);
- Here **name** is a string that specifies the name of the card whose panel is specified by *panelObj*. After we have created a deck, our program activates a card by calling one of the following methods:
  - void first(Container deck)
  - void last(Container deck)
  - void next(Container deck)
  - void previous(Container deck)
  - void show(Container deck, String cardName)
- *deck* is a reference to the container (usually a panel) that holds the cards, and *cardName* is the name of a card.

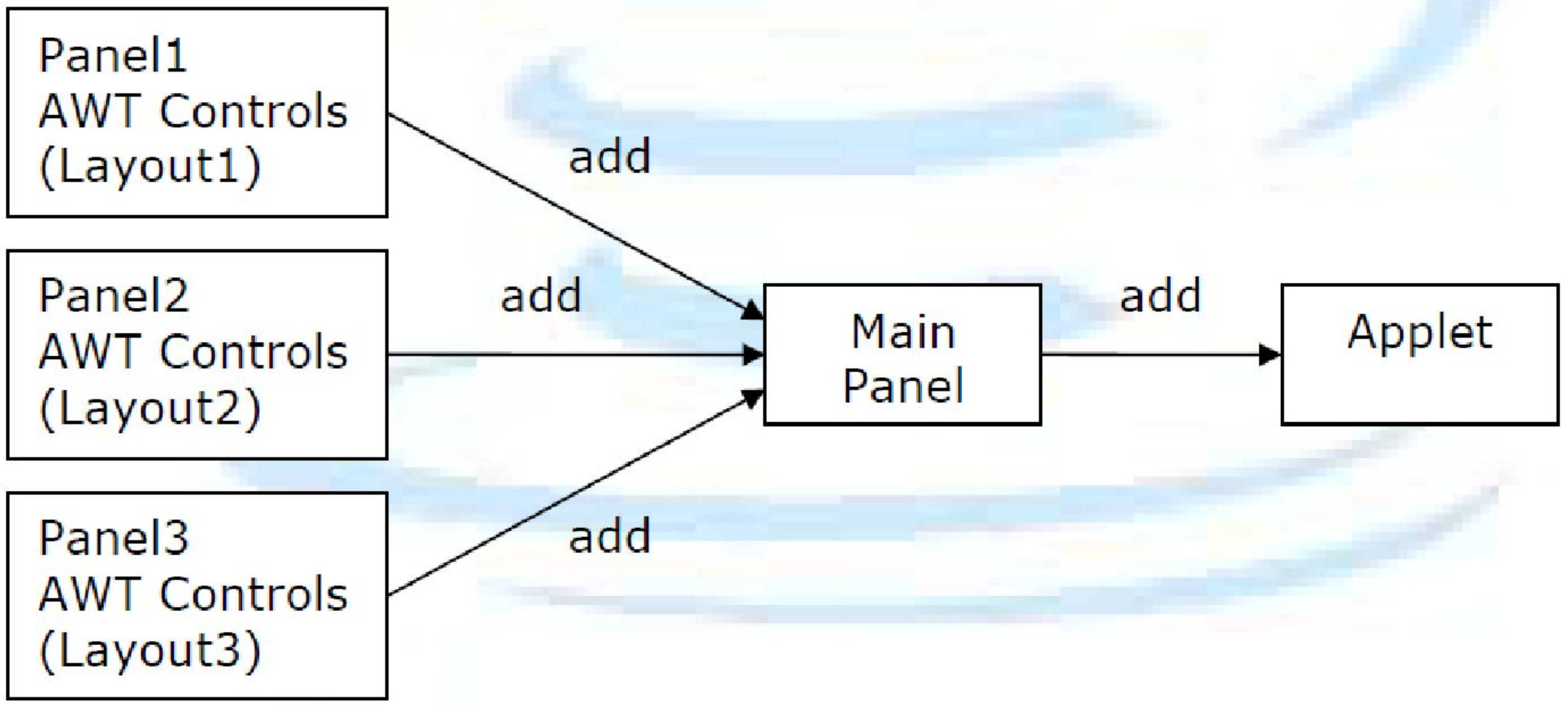
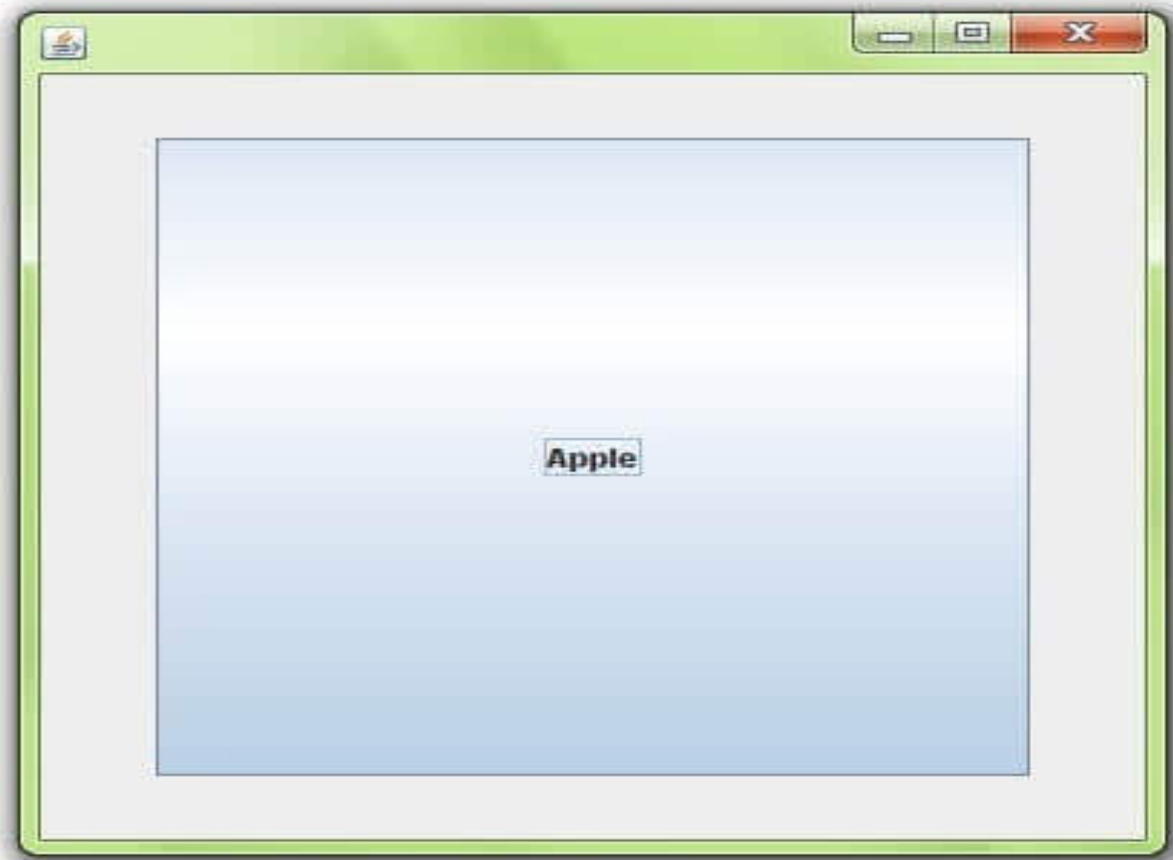


Fig. Creation of card layout

- // Demonstrate CardLayout.

```
import java.awt.*;
import java.awt.event.*;
public class CardLayoutExample extends Frame implement
  s ActionListener{
CardLayout card;
Button b1,b2,b3;
   CardLayoutExample(){
     card=new CardLayout(40,30);
      setLayout(card);
b1=new Button("Apple");
    b2=new Button("Boy");
    b3=new Button("Cat");
    b1.addActionListener(this);
    b2.addActionListener(this);
    b3.addActionListener(this);
    add(b1,”card1”); add(b2,”card2”); add(b3,”card3”);     }
  public void actionPerformed(ActionEvent e) {
  card.next(this);
  }
  public static void main(String[] args) {
    CardLayoutExample cl=new CardLayoutExample();
    cl.setSize(400,400);
    cl.setVisible(true);
      }  }
```


Apple

GridBagLayout

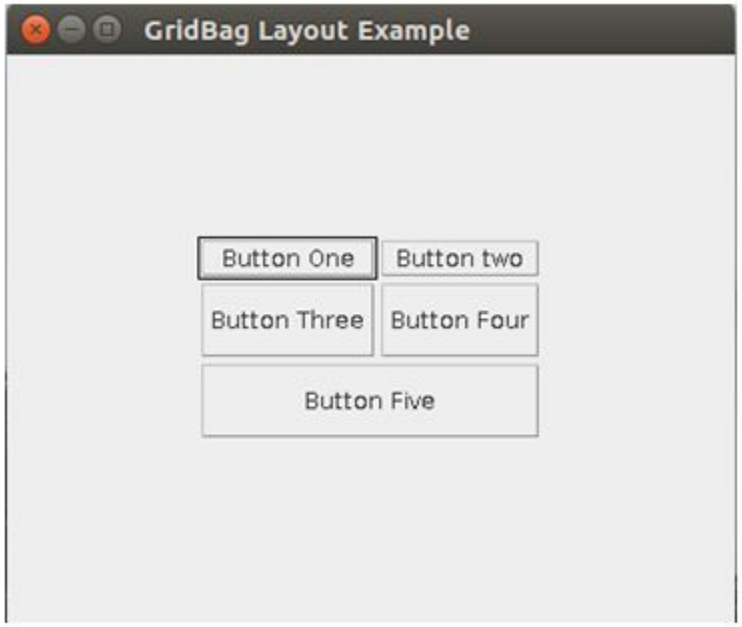
- Each GridBagLayout object maintains a dynamic rectangular grid of cells, with each component occupying one or more cells, called its *display area*.
- Each component managed by a grid bag layout is associated with an instance of GridBagConstraints that specifies how the component is laid out within its display area.

- For customize a GridBagConstraints object by setting one or more of its instance variables:
- gridx, gridy: Specifies the cell at the upper left of the component's display area, where the upper-left-most cell has address gridx = 0, gridy = 0.
- gridwidth, gridheight: Specifies the number of cells in a row (for gridwidth) or column (for gridheight) in the component's display area. The default value is 1.
- fill: Used when the component's display area is larger than the component's requested size to determine whether (and how) to resize the component.

```
• import java.awt.Button;
• import java.awt.GridBagConstraints;
• import java.awt.GridBagLayout;
• import javax.swing.*;
• public class GridBagLayoutExample extends JFrame{
•   public static void main(String[] args) {
•           GridBagLayoutExample a = new GridBagLayoutExample();
•       }
•       public GridBagLayoutExample() {
•   GridBagLayoutgrid = new GridBagLayout();
•           GridBagConstraints gbc = new GridBagConstraints();
•           setLayout(grid);
•           setTitle("GridBag Layout Example");
•           GridBagLayout layout = new GridBagLayout();
•   this.setLayout(layout);
• gbc.fill = GridBagConstraints.HORIZONTAL;
•     gbc.gridx = 0;
•     gbc.gridy = 0;
•     this.add(new Button("Button One"), gbc);
•     gbc.gridx = 1;
•     gbc.gridy = 0;
•     this.add(new Button("Button two"), gbc);
•     gbc.fill = GridBagConstraints.HORIZONTAL;
•     gbc.ipady = 20;
•     gbc.gridx = 0;
•     gbc.gridy = 1;
•     this.add(new Button("Button Three"), gbc);
•     gbc.gridx = 1;
•     gbc.gridy = 1;
•     this.add(new Button("Button Four"), gbc);
•
```

- gbc.gridx = 0;
-     gbc.gridy = 2;
-     gbc.fill = GridBagConstraints.HORIZONTAL;
-     gbc.gridwidth = 2;
-     **this**.add(**new** Button("Button Five"), gbc);
-             setSize(300, 300);
-             setPreferredSize(getSize());
-             setVisible(**true**);
-             setDefaultCloseOperation(EXIT_ON_CLOSE);
-               }
-
- }

- Output


GridBag Layout Example
Button One
Button two
Button Three
Button Four
Button Five

Menu Bars and Menus

- A menu bar displays a list of top-level menu choices. Each choice is associated with a dropdown menu.
- This concept is implemented in Java by the following classes:
    - **MenuBar**, **Menu**, and **MenuItem**.
- In general, a menu bar contains one or more Menu objects. Each Menu object contains a list of MenuItem objects. Each MenuItem object represents something that can be selected by the user.

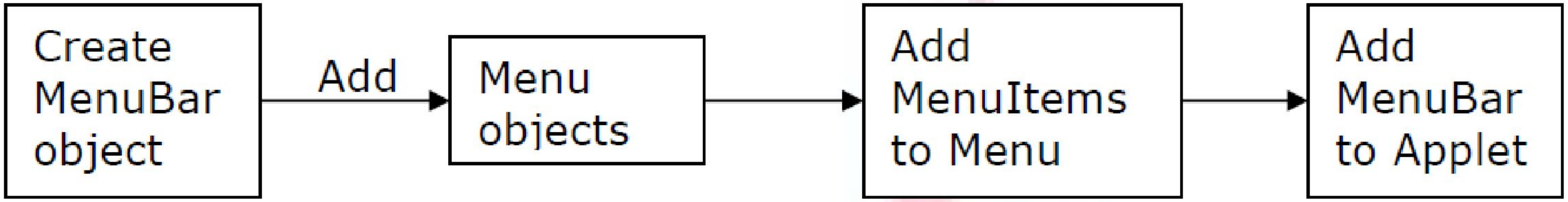


Fig. Creating a menu on Frame

- MenuBar Class Defines only default constructor.
- Menu Class Constructors
  - Menu( )
  - Menu(String optionName)
  - Menu(String optionName, boolean removable)
- Here, *optionName* specifies the name of the menu selection.
- Individual menu items constructors:
  - MenuItem( )
  - MenuItem(String *itemName*)
  - MenuItem(String *itemName*, MenuShortcut *keyAccel*)

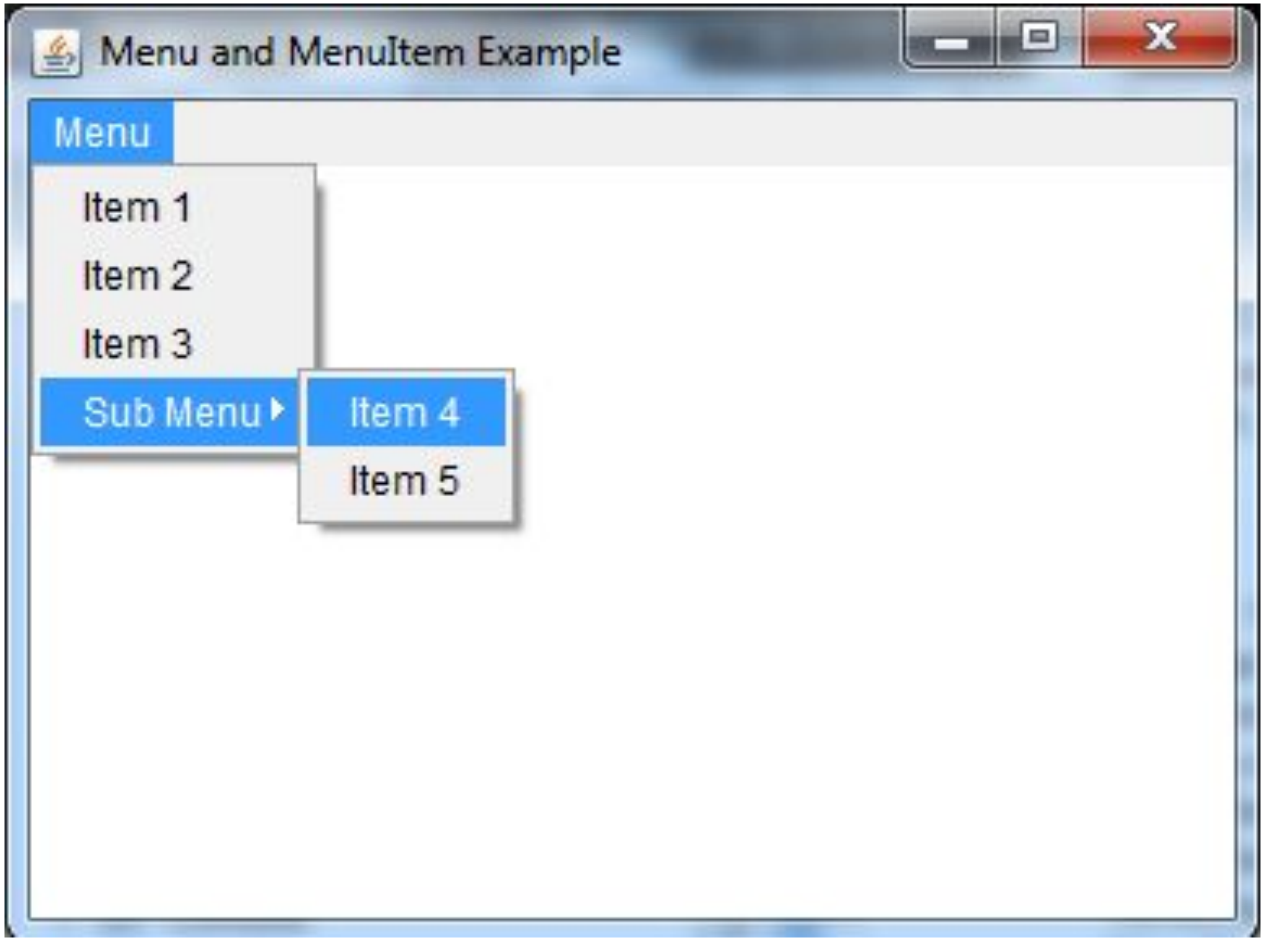
Methods

- Disable or enable a menu item by using:
    - void setEnabled(boolean *enabledFlag*)
    - boolean isEnabled( )
- Label set and get using:
    - void setLabel(String *newName*)
    - String getLabel( )
- Checkable menu item by using a subclass of **MenuItem** called **CheckboxMenuItem**. :
    - CheckboxMenuItem( )
    - CheckboxMenuItem(String *itemName*)
    - CheckboxMenuItem(String *itemName*, boolean *on*)

Methods

- Status about checkable MenuItem:
  - boolean getState( )
  - void setState(boolean *checked*)
- For add MenuItem:
  - MenuItem add(MenuItem *item*)
- For add MenuBar
  - Menu add(Menu *menu*)
- To get Item from Menu:
  - Object getItem( )

```
import java.awt.*;
class MenuExample
{
    MenuExample(){
      Frame f= new Frame("Menu Example");

      MenuBar mb=new MenuBar();
      Menu menu=new Menu("Menu");
      Menu submenu=new Menu("Sub Menu");

      MenuItem i1=new MenuItem("Item 1");
      MenuItem i2=new MenuItem("Item 2");
      MenuItem i3=new MenuItem("Item 3");
      MenuItem i4=new MenuItem("Item 4");
      MenuItem i5=new MenuItem("Item 5");
menu.add(i1);
menu.add(i2);  menu.add(i3);
submenu.add(i4);
submenu.add(i5);
menu.add(submenu);
mb.add(menu);
f.setMenuBar(mb);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String
 args[])  {
new MenuExample();
}  }
```

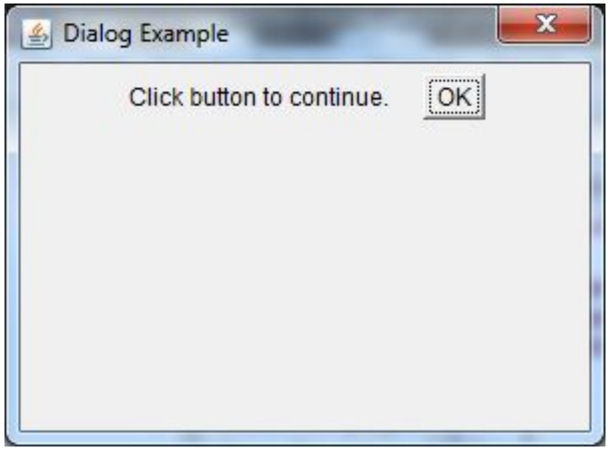

Menu and MenuItem Example
Menu
Item 1
Item 2
Item 3
Sub Menu
Item 4
Item 5

DialogBox

- Dialog boxes are primarily used to obtain user input.
- They are similar to frame windows, except that dialog boxes are always child windows of a top-level window.
- Dialog boxes don’t have menu bars.
- In other respects, dialog boxes function like frame windows.
- Dialog boxes may be modal or modeless.
- When a *modal* dialog box is active, all input is directed to it until it is closed.
- When a *modeless* dialog box is active, input focus can be directed to another window in your program.

- *Constructors:*.
  - Dialog(Frame *parentWindow*, boolean *mode*)
  - Dialog(Frame *parentWindow*, String *title*, boolean *mode*)
- To create Dialog Box:
  - Create Frame or Applet
  - Create another class which extends Dialog class.
  - Call this new class from Frame/Applet class.
  - In constructor of Extended Dialog class, use super method and pass vales to constructor of Dialog

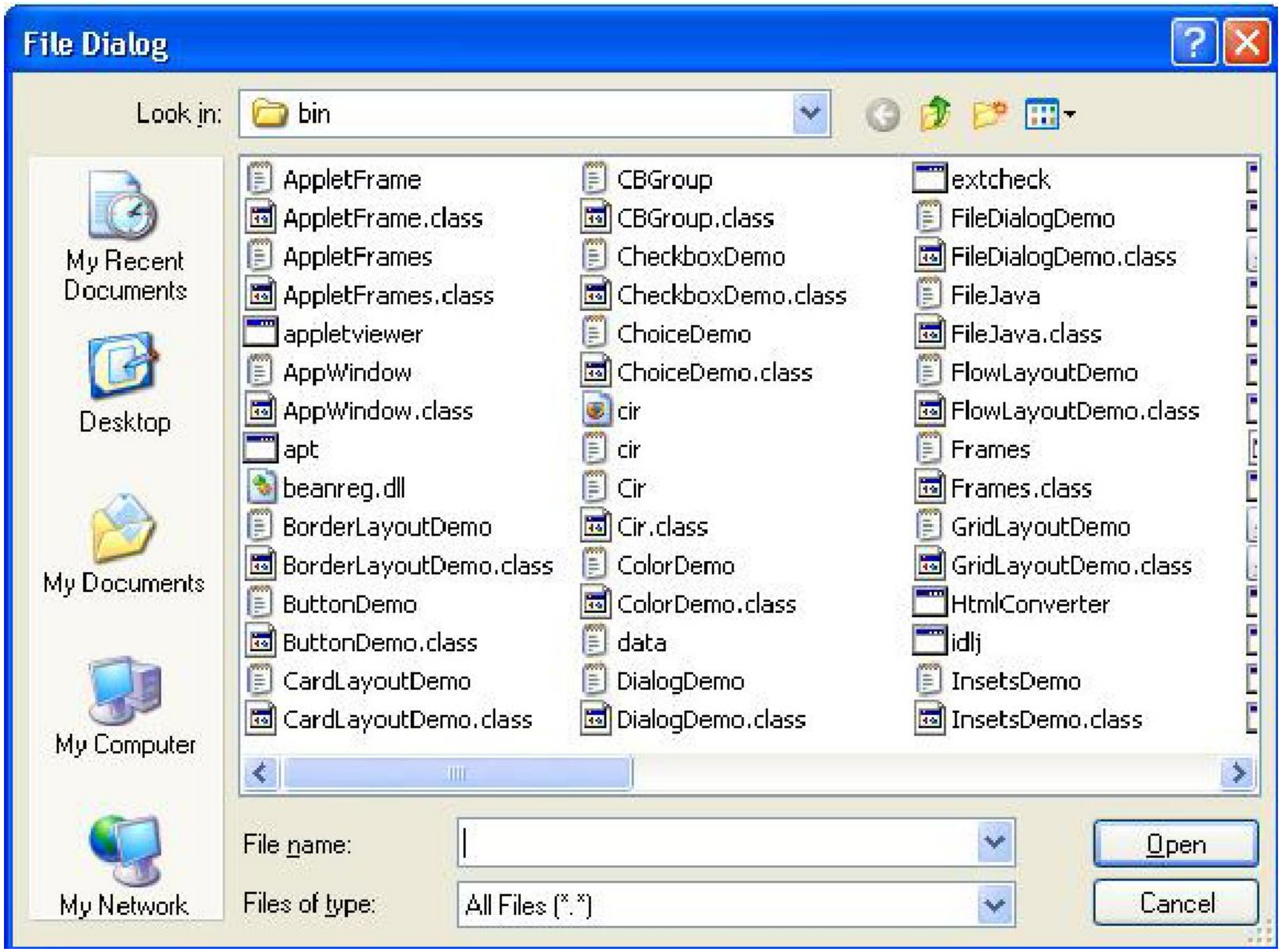
```
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
  private static Dialog d;
  DialogExample() {
    Frame f= new Frame();
    d = new Dialog(f , "Dialog Example", true);
    d.setLayout( new FlowLayout() );
    Button b = new Button ("OK");
    b.addActionListener ( new ActionListener()
    {  public void actionPerformed( ActionEvent e )
      { DialogExample.d.setVisible(false); }
    });
    d.add( new Label ("Click button to continue."));
    d.add(b);        d.setSize(300,300);
    d.setVisible(true);     }
```



FileDialog

- Java provides a built-in dialog box that lets the user specify a file.
- To create a file dialog box, instantiate an object of type **FileDialog**.
- Constructor:
  - FileDialog(Frame *parent*, String *boxName*)
  - FileDialog(Frame *parent*, String *boxName*, int *how*)
  - FileDialog(Frame *parent*)
- Int how: **FileDialog.LOAD, FileDialog.SAVE**
- **Methods:**
  - String getDirectory( )
  - String getFile( )

```
import java.awt.*;
class SampleFrame extends Frame
{
SampleFrame(String title){
super(title); }}
class FileDialogDemo
{public static void main(String args[]){
Frame f = new SampleFrame("File Dialog Demo");
f.setVisible(true);
f.setSize(100, 100);
FileDialog fd = new FileDialog(f, "File Dialog");
fd.setVisible(true);
}}
```



AppletFrame	CBGroup	extcheck
AppletFrame.class	CBGroup.class	FileDialogDemo
AppletFrames	CheckboxDemo	FileDialogDemo.class
AppletFrames.class	CheckboxDemo.class	FileJava
appletviewer	ChoiceDemo	FileJava.class
AppWindow	ChoiceDemo.class	FlowLayoutDemo
AppWindow.class	cir	FlowLayoutDemo.class
apt	cir	Frames
beanreg.dll	Cir	Frames.class
BorderLayoutDemo	Cir.class	GridLayoutDemo
BorderLayoutDemo.class	ColorDemo	GridLayoutDemo.class
ButtonDemo	ColorDemo.class	HtmlConverter
ButtonDemo.class	data	idlj
CardLayoutDemo	DialogDemo	InsetsDemo
CardLayoutDemo.class	DialogDemo.class	InsetsDemo.class

File name:

Files of type: All Files (*.*)