<https://way2java.com/awt-components/java-awt-menu-%e2%80%93-multiple-pull-down-lists/>

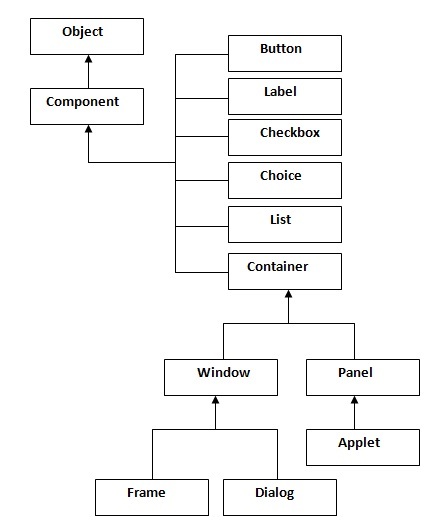
Abstract Windowing Toolkit-I

Introduction

The Graphical User Interface (GUI) of an applet can include a variety of components, such as buttons, check boxes , text fields, scrollbars, menus, and dialog boxes. A user can interact with an applet through these components. Each one of these components is represented as a separate class in Java. All these classes are available in a special package called Abstract windowing toolkit(AWT).

AWT is rarely used now days because of its platform dependent and heavy-weight nature. AWT components are considered heavy weight because they are being generated by underlying operating system (OS). For example if you are instantiating a text box in AWT that means you are actually asking OS to create a text box for you.

### Java AWT Hierarchy



To have everything placed on a screen to a particular position, we have to add them to a container. A container is like a screen wherein we are placing components like buttons, text fields, checkbox etc. In short a container contains and controls the layout of components. A container itself is a component (shown in the above hierarchy diagram) thus we can add a container inside container.

There are four types of containers available in AWT: Window, Frame, Dialog and Panel.

**Window:** An instance of the Window class has no border and no title  
**Dialog:** Dialog class has border and title. An instance of the Dialog class cannot exist without an associated instance of the Frame class.  
**Panel:** Panel does not contain title bar, menu bar or border. It is a generic container for holding components. An instance of the Panel class provides a container to which to add components.  
**Frame:** A frame has title, border and menu bars. It can contain several components like buttons, text fields, scrollbars etc. This is most widely used container while developing an application in AWT.

**Canva**s: Canvas component represents a blank rectangular area of the screen onto which the application can draw. Application can also trap input events from the use from that blank area of Canvas component

Examples of GUI based Applications

Following are some of the examples for GUI based applications.

* Automated Teller Machine (ATM)
* Airline Ticketing System
* Information Kiosks at railway stations
* Mobile Applications
* Navigation Systems

**Useful 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. |

**Event**:

An **event** is generated whenever a user interacts with an applet by way of pressing a button, dragging a mouse, selecting a menu item or by performing some such activity. Applets are event-driven programs. Event handling is the most important activity that often takes place in applet programming. In java , a source raises an event and sends it to one or more listeners. A **listene**r waits until it receives an appropriate event. Once it receives such an event, the listener ensures that the corresponding event-handling method is executed.

EVENTS

An event is generated when an user does something with a source. For example an event is generated when a user presses a button or drags a mouse or performs some similar actions. A set of **classes** are provided in **java.awt** package to represent the various types of events.

The primary event classes and the situations in which their objects are generated are described below:

Event classes

|  |  |
| --- | --- |
| Name of event class | Situations in which an event object is created |
| ActionEvent | When a button is pressed, a list item is double-clicked, a menu item is selected or the ENTER key is pressed in a textfield. |
| AdjustmentEvent | When a scroll bar is manipulated |
| ComponentEvent | When a component is hidden, moved, resized, or when it becomes visible |
| ContainerEvent | When a component is added or removed from a container |
| FocusEvent | When a component gains or loses keyboard focus |
| InputEvent | When a mouse or key event occurs |
| ItemEvent | When a check box or list item is selected, a choice selection is made, or a checkable menu item is selected or deselected |
| KeyEvent | When the input is received from the keyboard |
| MouseEvent | When the mouse is dragged, moved, clicked, pressed, released, the mouse enters a component or it exits a component |
| TextEvent | When the value inside a text area or text field is changed |
| WindowEvent | When a window is activated, closed, deactivated, deiconified, iconified, opened or exited |

Listeners

A listener receives event notifications. It should implement the appropriate listener interface. A listener interface declared the methods that are required to process some specified types of events. All of these listeners interfaces are specified in the java.awt.event package.

Listener interface

|  |  |
| --- | --- |
| Name of event class | Name of the corresponding Listener Interface |
| ActionEvent | ActionListener |
| AdjustmentEvent | AdjustmentListener |
| ComponentEvent | ComponentListener |
| ContainerEvent | ContainerListener |
| FocusEvent | FocusListener |
| ItemEvent | ItemListener |
| KeyEvent | KeyListener |
| MouseEvent | MouseListener, MouseMotionListener |
| TextEvent | TextListener |
| WindowEvent | WindowListener |

Event handling methods

A listener interface specifies the methods that are required to process the specified events. Such methods are known as event handling methods. Each one of these methods handles a particular type of event.

|  |  |
| --- | --- |
| Name of event class | Name of the corresponding Listener Interface |
| ActionEvent | actionPerformed() |
| AdjustmentEvent | adjustmentValueChanged() |
| ComponentEvent | componentHidden()  componentMoved()  componentResized()  componentShown() |
| ContainerEvent | componentAdded()  componentRemoved() |
| FocusEvent | focusGained()  focusLost() |
| ItemEvent | itemStateChanged() |
| KeyEvent | keyPressed()  keyReleased()  keyTyped() |
| MouseEvent | mouseClicked()  mouseEntered()  mouseExited()  mousePressed()  mouseReleased()  mouseDragged()  mouseMoved() |
| TextEvent | textValueChanged() |
| WindowEvent | windowActivated()  windowClosed()  windowClosing()  windowDeactivated()  windowDeiconified()  windowIconified()  windowOpened() |

### Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

* By extending Frame class (inheritance)
* By creating the object of Frame class (association)

Labels

A label contains a string, such as “Enter the value”. A label is often placed before a textfield.

Constructors of Label class are:

Label()

Label(String s)

Label(String s, int align)

s- is the text for label

align – left or center or right justification (LEFT, CENTER,RIGHT)

important methods are:

int getAlignment()

void setAlignment(int align)

String getText()

void setText(String s)

Example 1

to create a label and place it on applet’s screen

|  |
| --- |
| //Label demo  import java.applet.\*;  import java.awt.\*;  public class labeldemo extends Applet  {  public void init()  {  String s="enter the radius";  Label l=new Label(s, Label.LEFT);  add(l);  }  } |

Output

|  |
| --- |
| E:\javaprgs\applets>javac -cp . labeldemo.java  E:\javaprgs\applets>appletviewer labeldemo.html  Applet Viewer: labeldemo.class |

Button control

This has the constructor as follows:

Button()

Button(String s)

An event is generated when the user presses the button.

The listener can register and unregister to receive the click event of the button, by calling the following methods that are available in java.awt.Button class.

void addActionListener(ActionListener a1)

void removeActionListener(ActionListener a1)

here a1 is a reference to the ActionListener object

we can read and write caption on a button by the methods:

String getLabel()

void setLabel(String s)

within the actionPerformed() method, the caption on a button can be obtained by

String getActionCommane()

The method for setting label is

void setActionPerformed()

Example 2:

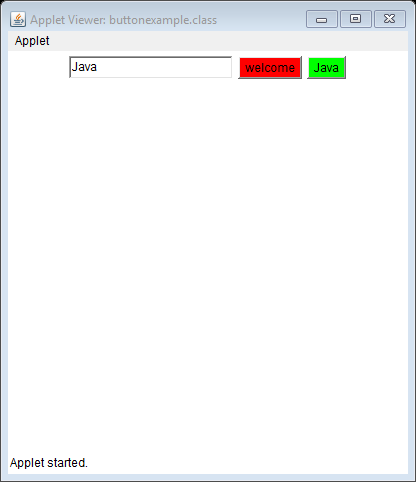
this program illustrates to create and use of two buttons. When the first button is pressed the message welcome is displayed. When the second button is pressed the message java is displayed within a textbox.

|  |
| --- |
| //creation and use of buttons  import java.applet.Applet;  import java.awt.\*;  import java.awt.event.\*;  public class buttonexample extends Applet implements ActionListener  {  TextField text1;  Button b1,b2;  public void init()  {  text1=new TextField(20);  add(text1);  b1=new Button("welcome");  b1.setBackground(Color.red);  add(b1);  b1.addActionListener(this);  b2=new Button("Java");  b2.setBackground(Color.green);  add(b2);  b2.addActionListener(this);  }  public void actionPerformed(ActionEvent e)  {  if(e.getSource()==b1)  {  text1.setText("welcome");  }  if(e.getSource()==b2)  {  text1.setText("Java");  }  }  } |

Output

|  |
| --- |
| E:\javaprgs\applets>javac -cp . buttonexample.java  E:\javaprgs\applets>appletviewer buttonexample.html |

Applet output



Checkbox control

Constructors are

Checkbox()

Checkbox(String s)

Checkbox(String s, boolean state)

Checkbox(String s, boolean state, CheckboxGroup g)

Checkbox(String s, CheckboxGroup g ,boolean state)

s- is the caption associated with the Checkbox object

if state is true , a check mark appears

an **item event** is generated when the state of checkbox changes

listeners are

void addItemListener(ItemListener i1)

void removeItemListener(ItemListener i1)

i1 is the ItemListener object

we can read and write the state of a checkbox by

boolean getState()

void setState(boolean value)

to read and write the caption

String getLabel()

void setLabel(String s)

The source(i.e checkbox) that generated the item event can be obtained by using the getItemSelectable() method as shown below:

ItemSelectable getItemSelectable()

The state change can be obtained by getStateChange() method

int getStateChange()

Example 3: the following program explains the method of creating and using three checkboxes. When one of these checkboxes is selected, the event-handler itemStateChanged() is invoked. Depending on the checkbox selected by the user, one of the three colors, namely Red,Green and Blue is displayed within a text box.

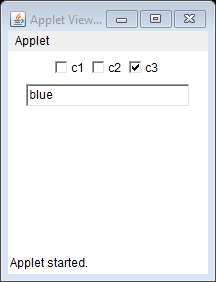
checkboxexample.java

|  |
| --- |
| //creation and use of buttons  import java.applet.Applet;  import java.awt.\*;  import java.awt.event.\*;  public class checkboxexample extends Applet implements ItemListener  {  Checkbox c1,c2,c3;  TextField t1;  public void init()  {  c1=new Checkbox("c1");  add(c1);  c1.addItemListener(this);  c2=new Checkbox("c2");  add(c2);  c2.addItemListener(this);  c3=new Checkbox("c3");  add(c3);  c3.addItemListener(this);  t1=new TextField(20);  add(t1);  }  public void itemStateChanged(ItemEvent e)  {  if(e.getItemSelectable()==c1)  {  t1.setText("Red");  }  if(e.getItemSelectable()==c2)  {  t1.setText("green");  }  if(e.getItemSelectable()==c3)  {  t1.setText("blue");  }  }  } |

checkboxexample.html

|  |
| --- |
| <html  <body>  <applet code="checkboxexample.class" width=200 height=200>  </applet>  </body>  </html> |

Output



Radio buttons

The constructor

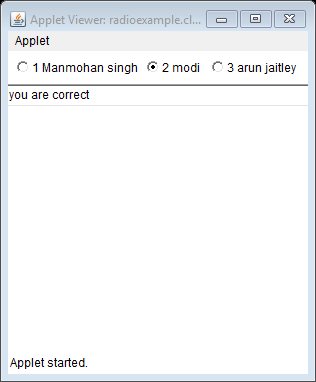
CheckboxGroup()

radioexample.java

|  |
| --- |
| //creation and use of buttons  import java.applet.Applet;  import java.awt.\*;  import java.awt.event.\*;  public class radioexample extends Applet implements ItemListener  {  Label l1;  CheckboxGroup cg1;  Checkbox c1,c2,c3;  TextField t1;  public void init()  {  l1=new Label("who is the prime minister of India");  cg1=new CheckboxGroup();  c1=new Checkbox("1 Manmohan singh",false,cg1);  add(c1);  c1.addItemListener(this);  c2=new Checkbox("2 modi ",false,cg1);  add(c2);  c2.addItemListener(this);  c3=new Checkbox("3 arun jaitley",false,cg1);  add(c3);  c3.addItemListener(this);  t1=new TextField(40);  add(t1);  }  public void itemStateChanged(ItemEvent e)  {  if(e.getItemSelectable()==c1)  {  t1.setText("you clicked 1 and is false");  }  if(e.getItemSelectable()==c2)  {  t1.setText("you are correct");  }  if(e.getItemSelectable()==c3)  {  t1.setText("you clicked 3 and is false");  }  }  } |

radioexample.html

|  |
| --- |
| <html>  <body>  <applet code="radioexample.class" width=300 height=300>  </applet>  </body>  </html> |



Choice control

It displays the first choice among all the allowed choices. When a user clicks on a choice, this control displays all the allowed choices. At this time, the user can select any one of those allowed choices. Once the user selects a choice, the choice control will return to its normal state. In this state, it will display only the current selection.

Constructor

Choice()

An itemevent is generated when a user selects one of the entries listed in a choice control

The listener can register and unregister to recive the ItemEvent object by invoking the following methods:

void addItemListener(ItemListener i1)

void removeItemListener(ItemListener i1)

we can add item to a Choice control through the following methods:

void addItem(String str)

we shall remove an existing item from Choice control by:

void remove(int i)

this method will remove the item at index i

the source that generated an item event can be obtained through the getItemSelectable() method as below:

ItemSelectable getItemSelectable()

The item that has been selected by the user can be known by invoking the method

String getSelectedItem()

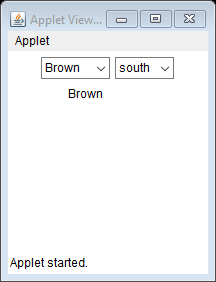
Example 5: Choiceexample.java

|  |
| --- |
| //creation and use of buttons  import java.applet.Applet;  import java.awt.\*;  import java.awt.event.\*;  public class choiceexample extends Applet implements ItemListener  {  Label l1;  Choice c1,c2;  TextField t1;  public void init()  {  c1=new Choice();  c1.addItem("Green");  c1.addItem("Brown");  c1.addItem("White");  c1.addItem("Red");  c1.addItem("Yellow");  c1.addItem("Orange");  c1.addItemListener(this);  add(c1);  c2=new Choice();  c2.addItem("East");  c2.addItem("west");  c2.addItem("south");  c2.addItem("north");  c2.addItemListener(this);  add(c2);  l1=new Label(" ");  add(l1);  }  public void itemStateChanged(ItemEvent e)  {  Choice c=(Choice) e.getItemSelectable();  l1.setText(c.getSelectedItem());  }  } |

Choiceexample.html

|  |
| --- |
| <html>  <body>  <applet code="choiceexample.class" width=300 height=300>  </applet>  </body>  </html> |

Output



List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

Constructors of LIST

List()

List(int rows)

List(int rows, Boolean multiple)

rows- represents the number of items that are visible to a user at a time.

Multiple- when this is true, the user can select two or more items in a list.

An **ItemEvent** object is generated when a user selects or deselects one of the items listed in a List control. An **ActionEvent** object is generated when a user double-clicks on an item in a List control.

An **ItemListener** object can be registered or unregistered to receive the item event by invoking the following methods:

**void addItemListener(ItemListener i1)**

**void removeItemListener(ItemListener i1)**

similarly, an **ActionListener** object can be registered or unregistered to receive the action event by invoking the following methods:

**void addActionListener(ActionListener a1)**

**void removeActionListener(ActionListener a1)**

we can add an item to a List control by using one of the following methods:

**void add(String str)**

**void add(String str, int i)**

we shall remove an existing item from a List control by suing one of the following methods:

**void remove(String str)**

**void remove(int i)**

**void removeAll()**

Here **str** is the name of the item to be removed and **i** is the index.

We can select the item at index **i** through the program with the help of the **select()** method:

void select(int i) throws IllegalArgumentException

to deselect the item at index i.

void deselect(int i)

sometimes a user may wish to select multiple items. In such a case, he has to set **flag** to **true** by using the **setMultipleMode()** method. If flag is **false**, he can select only one item.

void setMultipleMode(boolean flag)

Example 6:

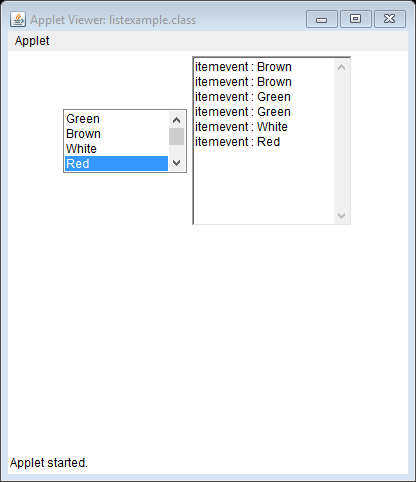
program name: listexample.java

|  |
| --- |
| //creation and use of buttons  import java.applet.Applet;  import java.awt.\*;  import java.awt.event.\*;  public class listexample extends Applet implements ItemListener,ActionListener  {  TextArea txtarea;  List list1;  public void init()  {  list1=new List();  list1.add("Green");  list1.add("Brown");  list1.add("White");  list1.add("Red");  list1.add("Yellow");  list1.add("Orange");  list1.addItemListener(this);  add(list1);  txtarea=new TextArea(10,20);  add(txtarea);  }  public void actionPerformed(ActionEvent ae)  {  txtarea.append("actionevent : "+ae.getActionCommand()+"\n");  }  public void itemStateChanged(ItemEvent e)  {  List l1=(List) e.getItemSelectable();  txtarea.append("itemevent : "+l1.getSelectedItem()+"\n");  }  } |

Program name: listexample.html

|  |
| --- |
| <html>  <body>  <applet code="listexample.class" width=300 height=300>  </applet>  </body>  </html> |

Output



FLOW LAYOUT

While a component is added to the applet window, it is possible to specify a particular position on the window at which that component is to be displayed. But doing so is not preferred as the code will no longer be platform independent. Instead , the placement of various components at specific positions on the applet window is handled by the Layout Managers.

The benefits of Layout Managers

1. No need to compute the coordinates at which the component is to be displayed. This is handled by Layout Manager
2. If the applet window is resized, the Layout Manager dynamically adjusts teh placement of the components.
3. This makes platform-independency possible

Three important layouts are

1. Flow Layout
2. Border Layout
3. Grid Layout

Any one of above can be done by the following method:

**void setLayout(LayoutManager lm)**

**lm** is the reference to an object that implements LayoutManager interface.

The current layout can be known by

LayoutManager getLayout()

FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel..

The constructor is

FlowLayout()

FlowLayout(int align)

FlowLayout(int align, int hgap, int vgap)

Here align indicates how the components are aligned in a row (LEFT, CENTER, RIGHT)

hgap & vgap - horizontal and vertical gap in pixels between components

Example 7:

The program shows the impact of flow layout .

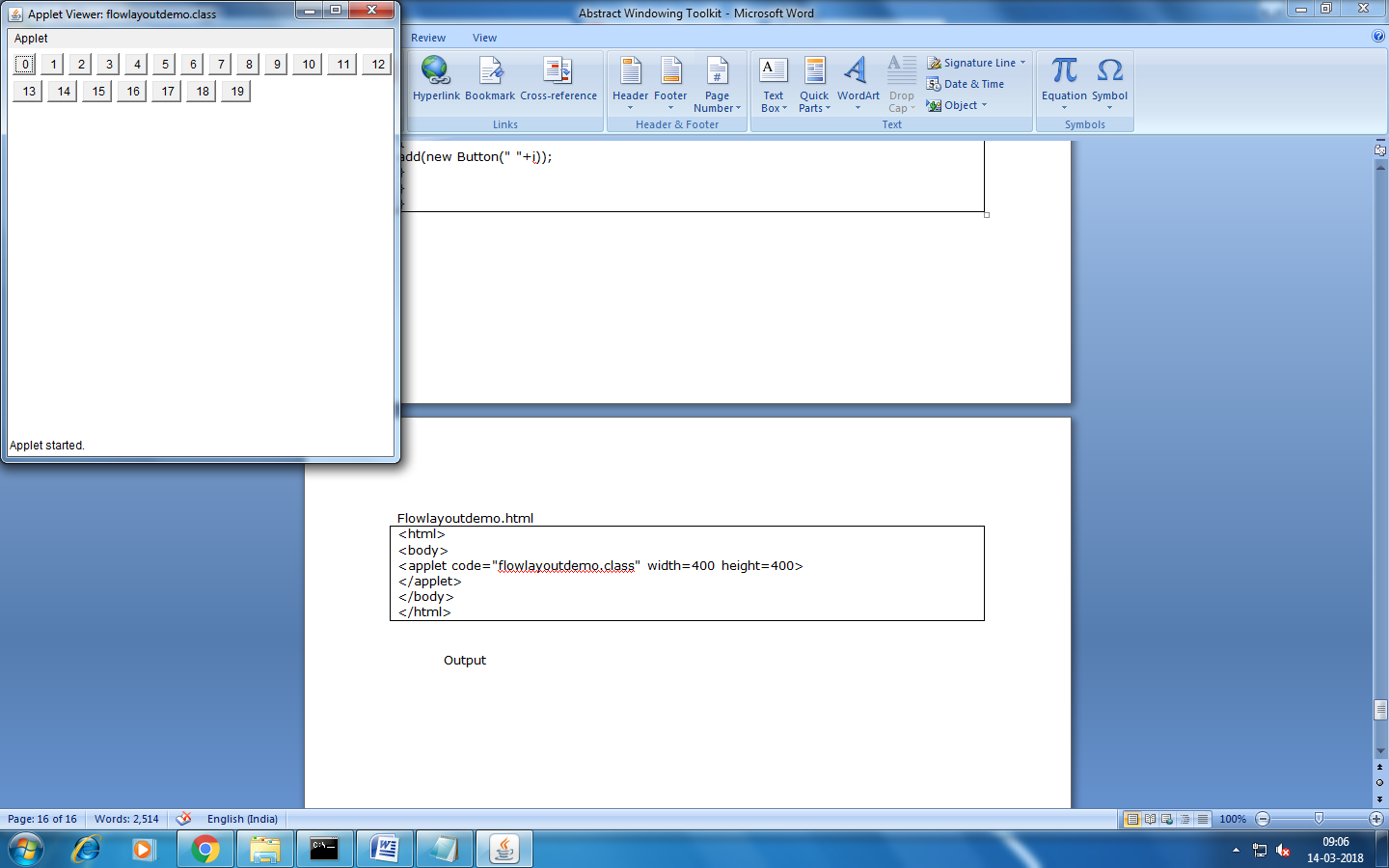
Flowlayoutdemo.java

|  |
| --- |
| import java.awt.\*;  import java.applet.Applet;  public class flowlayoutdemo extends Applet  {  public void init()  {  setLayout(new FlowLayout(FlowLayout.LEFT,5,5));  for(int i=0;i<20;i++)  {  add(new Button(" "+i));  }  }  } |

Flowlayoutdemo.html

|  |
| --- |
| <html>  <body>  <applet code="flowlayoutdemo.class" width=400 height=400>  </applet>  </body>  </html> |

Output



Border layout

This layout enables us to place some of the components around the periphery(i.e borders) of the applet

Constructor is

BorderLayout()

BorderLayout(int hgap, int vgap)

add(() method is used to add the components to applet

**void add(Component c, Object obj)**

here c is the component object to be added to the applet’s window and obj is a string that identifies its position via one of the following geographic terms:

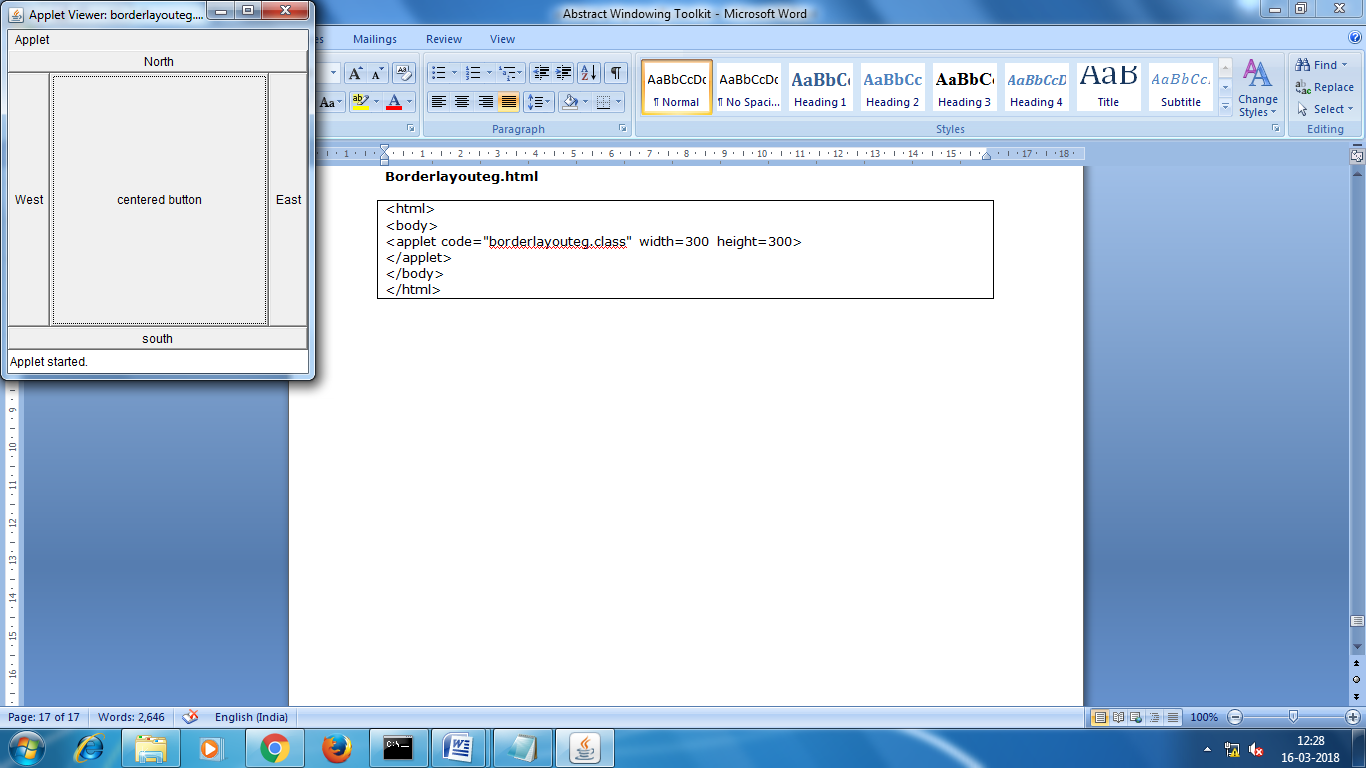
**“West”, “East”, “North”, “South” and “Center”**

**Program name: borderlayouteg.java**

|  |
| --- |
| **import java.awt.\*;**  **import java.applet.\*;**  **public class borderlayouteg extends Applet**  **{**  **Button but1, but2,but3,but4,but5;**  **public void init()**  **{**  **setLayout(new BorderLayout());**  **but1=new Button("centered button");**  **but2=new Button(" North");**  **but3=new Button("south");**  **but4=new Button("East");**  **but5=new Button("West");**  **add(but1, "Center");**  **add(but2, "North");**  **add(but3, "South");**  **add(but4, "East");**  **add(but5, "West");**  **}**  **}** |

**Borderlayouteg.html**

|  |
| --- |
| <html>  <body>  <applet code="borderlayouteg.class" width=300 height=300>  </applet>  </body>  </html> |



GridLayout

Here the Insets() method sets the top, left , bottom and right margins around the

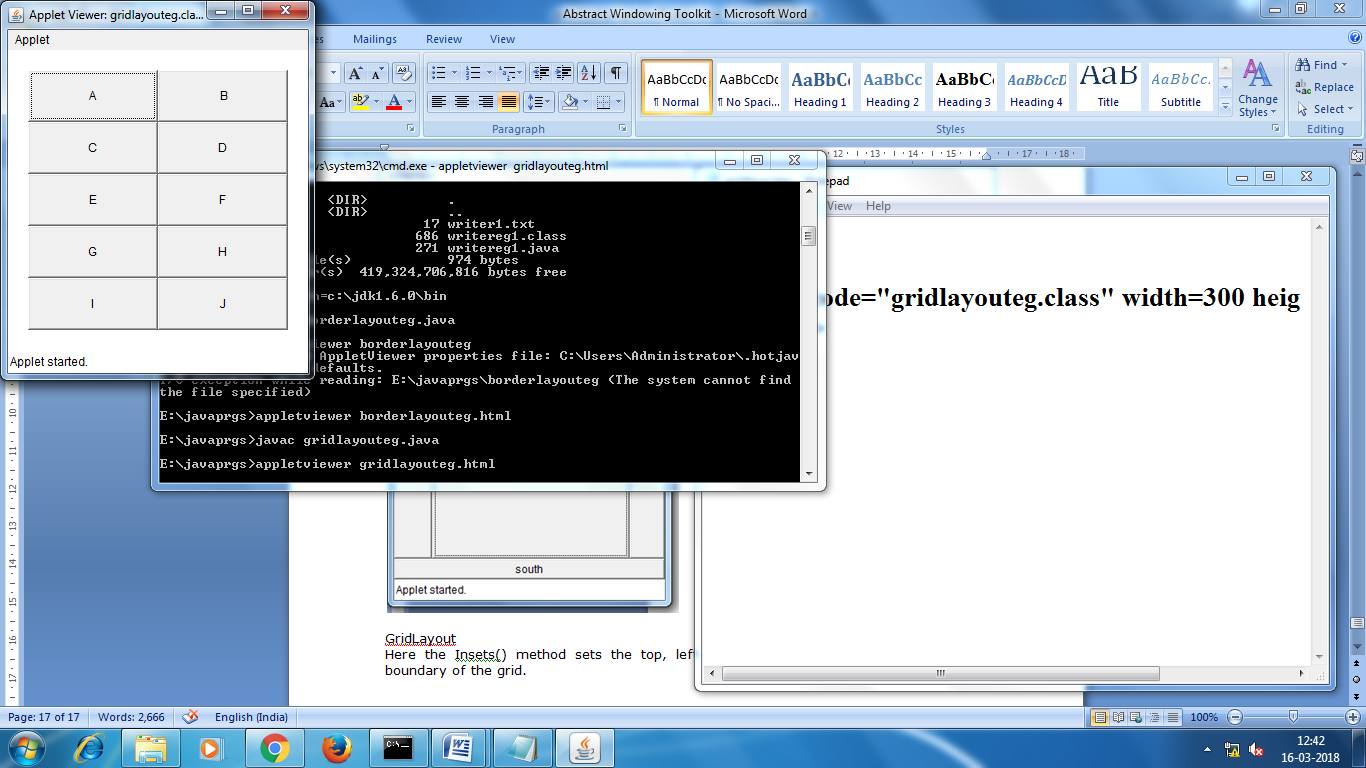
boundary of the grid.

Program name: gridlayouteg.java

|  |
| --- |
| import java.awt.\*;  import java.applet.\*;  import java.awt.event.\*;  public class gridlayouteg extends Applet  {  Button b1, b2,b3,b4,b5,b6,b7,b8,b9,b10;  public void init()  {  setLayout(new GridLayout(5,2));  b1=new Button("A");  b2=new Button(" B");  b3=new Button("C");  b4=new Button("D");  b5=new Button("E");  b6=new Button("F");  b7=new Button("G");  b8=new Button("H");  b9=new Button("I");  b10=new Button("J");  add(b1);  add(b1);  add(b2);  add(b3);  add(b4);  add(b5);  add(b6);  add(b7);  add(b8);  add(b9);  add(b10);  }  public Insets getInsets()  {  return new Insets(20,20,20,20);  }  } |

Gridlayout.java

|  |
| --- |
| <html>  <body>  <applet code="gridlayouteg.class" width=300 height=300>  </applet>  </body>  </html> |



Panels

Java AWT Panels play a very important role in layout management. Panels in combination with layout managers render the desired layout to the programmer. The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class. It doesn't have title bar.

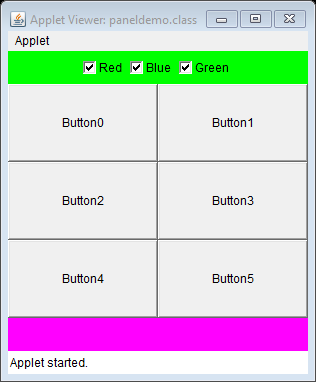
In the following program three panels are created as pn1, pn2 and pn3. In the first panel pn1 three checkboxes namely cb1, cb2 and cb3 are created with border layout. In the panel pn2 6 buttons are added with grid layout. In the panel pn3 a label is added. Whenever a check box is clicked some text are made to display.

Program name: paneldemo.java

|  |
| --- |
| //creating and using panels  import java.awt.\*;  import java.awt.event.\*;  import java.applet.Applet;  public class paneldemo extends Applet implements ItemListener  {  Label lb1;  Checkbox cb1,cb2,cb3;  public void init()  {  setLayout(new BorderLayout());  Panel pn1=new Panel();  cb1=new Checkbox("Red", true);  pn1.add(cb1);  cb2=new Checkbox("Blue", true);  pn1.add(cb2);  cb3=new Checkbox("Green", true);  pn1.add(cb3);  cb1.addItemListener(this);  cb2.addItemListener(this);  cb3.addItemListener(this);  pn1.setBackground(Color.green);  add(pn1,"North");  Panel pn2=new Panel();  pn2.setLayout(new GridLayout(3,2));  for(int i=0;i<6;i++)  pn2.add(new Button("Button"+i));  add(pn2,"Center");  Panel pn3=new Panel();  lb1=new Label(" ");  pn3.add(lb1);  pn3.setBackground(Color.magenta);  add(pn3,"South");  }  public void itemStateChanged(ItemEvent e)  {  if(e.getItemSelectable()==cb1)  {  lb1.setText("checkbox1 selected");  }  if(e.getItemSelectable()==cb2)  {  lb1.setText("checkbox2 selected");  }  if(e.getItemSelectable()==cb3)  {  lb1.setText("checkbox3 selected");  }  }  } |

Program name: paneldemo.html

|  |
| --- |
| <html>  <body>  <applet code="paneldemo.class" width=300 height=300>  </applet>  </body></html> |



Windows and frames

The **window** class extends the **container** class.

WindowListener object can be registered or unregistered to receive the window events with the help of the following methods:

void addWindowListener(WindowListener w1)

void removeWindowListener(WindowListener w1)

here **w1** is an object of WindowListener

the pack() method is invoked to lay out the components in a window and set its initial size.

A new window is initially invisible . to make it visible , the **show()** method is to be invoked.

When we finish using a window, we should relinquish its resources by invoking the **dispose()** method.

A WindowEvent object is generated when a window is activated, closed, aborted to be closed, deactivated, deiconified, iconified or opened. The WindowListener interface declares the following methods:

void windowActivated(WindowEvent we)

void windowClosed(WindowEvent we)

void windowClosing(WindowEvent we)

void windowDeactivated(WindowEvent we)

void windowDeiconified(WindowEvent we)

void windowIconified(WindowEvent we)

void windowOpened(WindowEvent we)

The Frame class extends the Window class. It provides title and menu bars.

Constructors are

Frame()

Frame(String title)

Example 1:

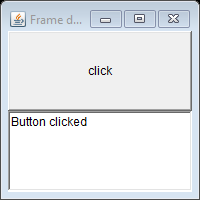
A frame is created with one button and one text field . when the button is clicked the message Button Clicked is reported in a text field.

Program name: framedemo.java

|  |
| --- |
| import java.awt.\*;  import java.awt.event.\*;  import java.applet.Applet;  public class framedemo extends Applet implements ActionListener  {  Button b1;  TextField tf;  public void init()  {  Frame f=new Frame("Frame demo");  f.setLayout(new GridLayout(2,2));  f.setSize(200,200);  b1=new Button("click");  f.add(b1);  b1.addActionListener(this);  tf=new TextField(20);  f.add(tf);  f.show();  }  public void actionPerformed(ActionEvent e)  {  if(e.getSource()==b1)  {  tf.setText("Button clicked");  }  }  } |

Program name:framedemo.html

|  |  |
| --- | --- |
| <html>  <body>  <applet code="framedemo.class" width=300 height=300>  </applet>  </body></html> |  |



Menus

To create menus, the java.awt package comes with mainly four classes – MenuBar, Menu, MenuItem and CheckboxMenuItem. All these four classes are not AWT components as they are not subclasses of java.awt.Component class. Infact, they are subclasses of java.awt.MenuComponent which is is no way connected in the hierarchy with Component class.

* **MenuBar:** MenuBar holds the menus. MenuBar is added to frame with **setMenuBar()** method. Implicitly, the menu bar is added to the north (top) of the frame. MenuBar cannot be added to other sides like south and west etc.
* **Menu:** Menu holds the menu items. Menu is added to frame with **add()** method. A sub-menu can be added to Menu.
* **MenuItem:** MenuItem displays the actual option user can select. Menu items are added to menu with method **addMenuItem()**. A dull-colored line can be added in between menu items with **addSeparator()** method. The dull-colored line groups (or separates from other) menu items with similar functionality like cut, copy and paste.
* **CheckboxMenuItem:** It differs from MenuItem in that it appears along with a checkbox. The selection can be done with checkbox selected.

With this menus can be constructed. The container for the top-level menu item is called a menubar. In Java , a menu bar is positional inside a frame. The menu bar contains menus. A menu is a collection of menu items and more menus(sub menus)

The **MenuBar** class encapsulates the functionality of a menu bar. The constructor is

**MenuBar()**

The Menu objects can be added to the **MenuBar** by using the **add()** method

**Menu add(Menu m)**

Here **m** is the Menu to be added to **MenuBar**

Constructor for Menu is

**Menu(String str)**

**str** is the string that is to be displayed for the Menu.

The font used for a menu-related string may be set via the **setFont()** method shown here.

void setFont(Font f)

MenuItem objects can be added to a Menu via the add() method

MenuItem add(MenuItem m1)

The constructor is

MenuItem(String str)

MenuItem objects generated ActionEvent objects when they are selected or deselected.

An ActionListener object can be registered.

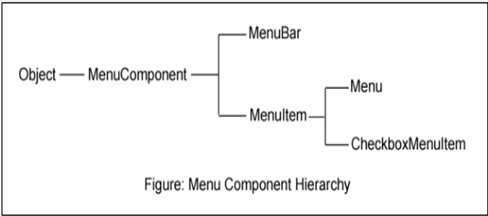
void addActionListener(ActionListener a1)

void removeActionListener(ActionListener a1)

here a1 is a reference to the ActionListener object.

A menu item can be disabled or enabled by using setEnabled() method

void setEnabled(Boolean flag)

[](http://way2java.com/wp-content/uploads/2011/04/ss9.bmp)

**Steps of Creating Java AWT Menu**

Creation of menus involves many steps to be followed in an order. Following are the steps.

1. Create menu bar
2. Add menu bar to the frame
3. Create menus
4. Add menus to menu bar
5. Create menu items
6. Add menu items to menus
7. Event handling

In the following program, three menus are created and populated with menu items. User's selected menu item information is displayed in the **text area**.

**Just using frame**

**Program name: menujavaexample.java**

|  |
| --- |
| **import java.awt.\*;**  **class MenuExample extends Frame**  **{**  **MenuExample()**  **{**  **MenuBar mbar=new MenuBar();**  **setMenuBar(mbar);**  **Menu mFile=new Menu("File");**  **Menu mEdit=new Menu("Edit");**  **Menu mView=new Menu("View");**  **mbar.add(mFile);**  **mbar.add(mEdit);**  **mbar.add(mView);**  **MenuItem iOpen=new MenuItem("Open");**  **MenuItem iSave=new MenuItem("Save");**  **MenuItem iExit=new MenuItem("Exit");**  **mFile.add(iOpen);**  **mFile.add(iSave);**  **mFile.add(iExit);**  **MenuItem iCopy=new MenuItem("Copy");**  **mEdit.add(iCopy);**  **}**  **}**  **public class menujavaexample**  **{**  **public static void main(String args[])**  **{**  **MenuExample me=new MenuExample();**  **me.setTitle("menu in java example");**  **me.setSize(350,250);**  **me.setResizable(false);**  **me.setVisible(true);**  **}**  **}** |

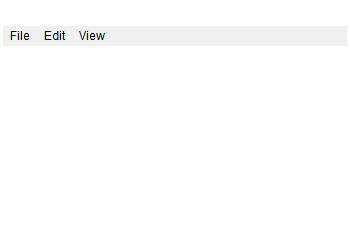
**Program name:menujavaexample.html**

|  |
| --- |
| <html>  <body>  <applet code="menujavaexample.class" width=400 height=400>  </applet>  </body>  </html> |

**To execute**

|  |
| --- |
| **E:\javaprgs\applets>javac -cp . menujavaexample.java**  **E:\javaprgs\applets>java -cp . menujavaexample** |

**Output**

****

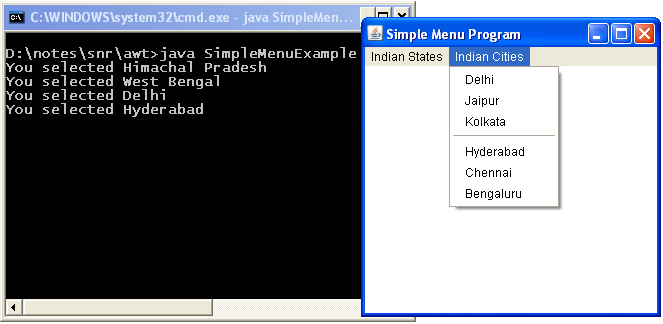
**Following is a simple Menus Java program with just two menus added with few menu items. User selection is displayed at DOS prompt for simplicity to get the concept. A big program we see later.**

**Program name:SimpleMenuExample.java**

|  |
| --- |
| import java.awt.\*;  import java.awt.event.\*;  public class SimpleMenuExample extends Frame implements ActionListener  {  Menu states, cities;  public SimpleMenuExample()  {  MenuBar mb = new MenuBar(); // begin with creating menu bar  setMenuBar(mb); // add menu bar to frame  states = new Menu("Indian States"); // create menus  cities = new Menu("Indian Cities");  mb.add(states); // add menus to menu bar  mb.add(cities);  states.addActionListener(this); // link with ActionListener for event //handling  cities.addActionListener(this);  states.add(new MenuItem("Himachal Pradesh"));  states.add(new MenuItem("Rajasthan"));  states.add(new MenuItem("West Bengal"));  states.addSeparator(); // separates from north Indian states from //south Indian  states.add(new MenuItem("Andhra Pradesh"));  states.add(new MenuItem("Tamilnadu"));  states.add(new MenuItem("Karnataka"));  cities.add(new MenuItem("Delhi"));  cities.add(new MenuItem("Jaipur"));  cities.add(new MenuItem("Kolkata"));  cities.addSeparator();// separates north Indian cities from south Indian  cities.add(new MenuItem("Hyderabad"));  cities.add(new MenuItem("Chennai"));  cities.add(new MenuItem("Bengaluru"));  setTitle("Simple Menu Program"); // frame creation methods  setSize(300, 300);  setVisible(true);  }  public void actionPerformed(ActionEvent e)  {  String str = e.getActionCommand(); // know the menu item selected by //the user  System.out.println("You selected " + str);  }  public static void main(String args[])  {  new SimpleMenuExample();  }  } |

**Program name:SimpleMenuExample.html**

|  |
| --- |
| <html>  <body>  <applet code="SimpleMenuExample.class" width=400 height=400>  </applet>  </body>  </html> |



Dialogs