Using AWT controls, Layout Managers, and Menus

- **Controls** are components that allow a user to interact with your application eg push button.
- **Layout manager** automatically positions components within a container. Thus, the appearance of a window is determined by a combination of the controls that it contains and the layout manager used to position them.
- **Menu bar :** Each entry in a menu bar activates a drop-down menu of options from which the user can choose. This constitutes the main menu of an application. As a general rule, a menu bar is positioned at the top of a window.

AWT Control Fundamentals

The AWT supports the following types of controls:

- Labels
- Push buttons
- Check boxes
- Choice lists
- Lists
- Scroll bars
- Text Editing

These controls are **subclasses of Component**.

Continued..

- To add a control in a window, create an instance of the desired control and then add it to a window by calling add( ), which is defined by Component class. One of the form is -

  Component add(Component compRef)

  compRef is a reference to an instance of the control that you want to add. A reference to the object is returned.

- To remove a control from a window call remove( ). This method is also defined by Container.

  void remove(Component compRef)

  compRef is a reference to the control you want to remove.

- To remove all controls, call

  removeAll( )

Responding to Controls

- Except for labels, which are passive, all other controls generate events when they are accessed by the user.

- When the user clicks on a push button, an event is generated that identifies the push button.

- The program simply implements the appropriate interface and then registers an event listener for each control that are needed to monitor.

Labels

- A label is an object of type Label, and it contains a string, which it displays. Label defines the following constructors:

  Label( ) throws HeadlessException
  Label(String str) throws HeadlessException
  Label(String str, int how) throws HeadlessException

- 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

Continued..

- To set or change the text in a label call

  void setText(String str)

- To return the current label

  String getText( )

- To set the alignment of the string within the label call

  void setAlignment(int how)

  how must be one of the alignment constants.

- To get the current alignment call

  int getAlignment( )

Example

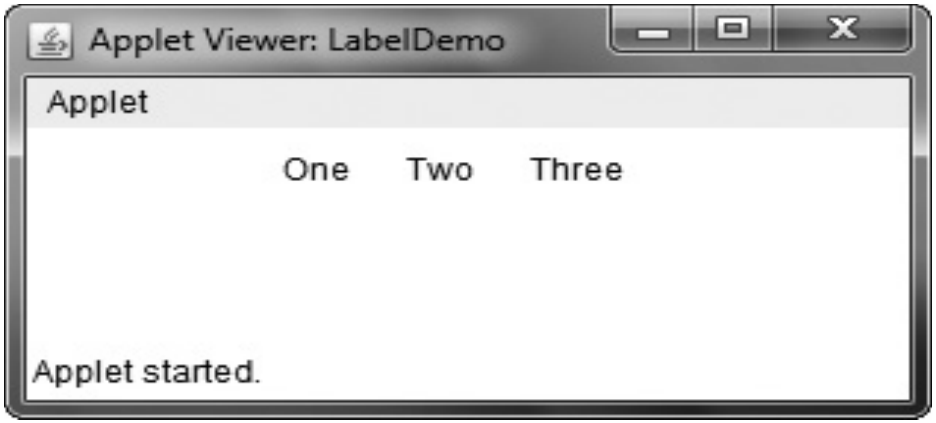
Create three Labels and add them to Applet-

```
// 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");
```

Continued..

```
// add labels to applet window
add(one);
add(two);
add(three);
}
}
```

Note: The labels are organized in the window by the default layout manager.



Using Buttons

- 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( ) throws HeadlessException

  Button(String str) throws HeadlessException

The first version creates an empty button. The second creates a button that contains str as a label.

- After a button has been created, one can set its label by

  void setLabel(String str)

Here, str becomes the new label for the button.

- To retrieve label of a button call

  String getLabel( )

Handling Buttons

- Each time a button is pressed, an **action event** is generated.
- This is sent to any listeners that previously registered an interest in **receiving action** event notifications from that component.
- Each listener implements the **ActionListener interface**. That interface defines the **actionPerformed( )** method, which is called when an event occurs.
- An **ActionEvent object** is supplied as the argument to this method. It contains both a reference to the button that generated the event and a reference to the action command string associated with the button.
- By default, **the action command string is the label of the button**.

Example

- Here is an example that creates three buttons labeled "Yes", "No", and "Undecided".
- Each time one is pressed, a message is displayed that reports which button has been pressed.
- In this version, **the action command of the button (is its label) is used to determine which button has been pressed.**
- The label is obtained by calling the **getActionCommand( )** method on the ActionEvent object passed to actionPerformed( ).

Code

```
// Demonstrate Buttons
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonDemo" width=250
height=150>
</applet>
*/
```

Continued..

```
public class ButtonDemo extends Applet
implements ActionListener {
String msg = "";
Button yes, no, maybe;
public void init() {
yes = new Button("Yes");  // create three
instances of Button class
no = new Button("No");
maybe = new Button("Undecided");
```

Continued..

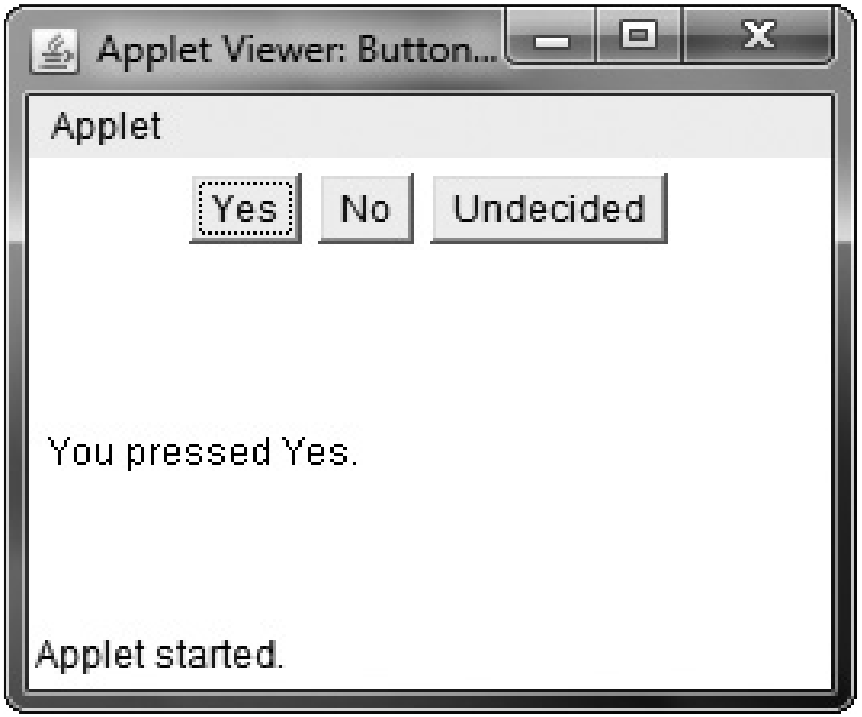
```
add(yes);    // returned reference to the added component is not used
add(no);
add(maybe);
yes.addActionListener(this);
no.addActionListener(this);
maybe.addActionListener(this);
}
```

Continued..

```
public void actionPerformed(ActionEvent ae) {
String str = ae.getActionCommand();
if(str.equals("Yes")) {
msg = "You pressed Yes.";
}
else if(str.equals("No")) {
msg = "You pressed No.";
}
else {
msg = "You pressed Undecided.";
}
repaint();
}
```

Continued..

```
public void paint(Graphics g) {
g.drawString(msg, 6, 100);
}
}
```



Alternate Code

- Instead of comparing button action command strings, one can also determine which button has been pressed by comparing the object obtained from the **getSource( )** method to the button objects that are added to the window.

- To do this, one must keep a list of the objects when they are added. The following applet shows this approach:

Code

```
// Recognize Button objects.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="ButtonList" width=250
height=150>
</applet>
*/
```

Code

```
public class ButtonList extends Applet
implements ActionListener {

String msg = "";
Button bList[] = new Button[3];
public void init() {
Button yes = new Button("Yes");
Button no = new Button("No");
Button maybe = new Button("Undecided");
```

Continued..

```
// store references to buttons as added
bList[0] = (Button) add(yes);  //return type of add is component
bList[1] = (Button) add(no);  //to tell that it is button type casting
bList[2] = (Button) add(maybe);  // is used
// register to receive action events
for(int i = 0; i < 3; i++) {
bList[i].addActionListener(this);
}   // end for
}   // end init()
```

Continued..

```
public void actionPerformed(ActionEvent ae) {
for(int i = 0; i < 3; i++) {
If (ae.getSource() == bList[i]) {
msg = "You pressed " + bList[i].getLabel();
}
}
repaint();
}
public void paint(Graphics g) {
g.drawString(msg, 6, 100);
}
}
```

Continued..

- In this version, the program stores each button reference in an array when the buttons are added to the applet window.
- Inside **actionPerformed( )**, this array is then used to determine which button has been pressed.
- For simple programs, it is usually easier to recognize buttons by their labels.
- **However, in situations in which you will be changing the label inside a button during the execution of your program, or using buttons that have the same label, it may be easier to determine which button has been pushed by using its object reference**.

Continued..

It is also possible to set the action command string associated with a button to something other than its label by calling **setActionCommand( ).**

This method changes the action command string, but does not affect the string used to label the button.

Thus, setting the action command enables the action command and the label of a button to differ.

Adding a Button to Frame in a Standalone GUI program

```
import java.awt.*;
import java.awt.event.*;
public class ButtonText {
        public static void main(String[] args) {
                Frame frame=new Frame("Button Frame");
                Button button = new Button("Submit");
                frame.add(button);
                frame.setLayout(new FlowLayout());
                frame.setSize(200,100);
                frame.setVisible(true);
                frame.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent e){
                                System.exit(0);
                        }
                });


        }
}
```