# 9 Java Buttons and Menus

## 9.1 Introduction

One of the features of Java is that it supports many different types of menus items, these are:

- Buttons. Simple buttons which are pressed to select an item.
- Pop-up menus (or pull-down menus). Used to select from a number of items. The selected item is displayed in the pop-up menu window.
- List boxes.
- Dialog boxes. Used to either load or save files.
- Checkboxes. Used to select/deselect an item.
- Radio buttons. Used to select from one or several items.
- Menu bars. Used to display horizontal menus with pull-down menus.

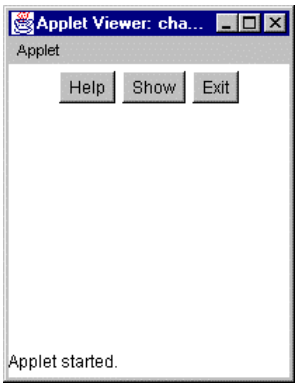
These are used with event handlers to produce event-driven options.

## 9.2 Buttons and events

Java applet 9.1 creates three `Button` objects. These are created with the `add()` method which displays the button in the applet window.

Java applet 9.1

```
import java.awt.*;
import java.applet.*;

public class chap9_01 extends Applet
{
  public void init()
  {
    add(new Button("Help"));
    add(new Button("Show"));
    add(new Button("Exit"));
  }
}
```



An alternative approach to creating buttons is to declare them using the `Button` type. For example the following applet is equivalent to Java applet 8.1. The names of the button objects, in this case, are `button1`, `button2` and `button3`.

```
import java.applet.*;
import java.awt.*;

public class chap9_01 extends Applet
{
Button button1, button2, button3;

    public void init()
    {
        button1= new Button("Help");
        button2= new Button("Show");
        button3= new Button("Exit");
        add(button1);
        add(button2);
        add(button3);
    }
}
```

## 9.3 Action with Java 1.0

In Java 1.0, the `action` method is called when an event occurs, such as a keypress, button press, and so on. The information on the event is stored in the `Event` parameter. Its format is:

```
public boolean action(Event evt, Object obj)
```

where `evt` is made with the specified target component, time stamp, event type, x and y coordinates, keyboard key, state of the modifier keys and argument. These are:

- `evt.target` is the target component
- `evt.when` is the time stamp
- `evt.id` is the event type
- `evt.x` is the x coordinate
- `evt.y` is the y coordinate
- `evt.key` is the key pressed in a keyboard event
- `evt.modifiers` is the state of the modifier keys
- `evt.arg` is the specified argument

Java applet 9.2 contains an example of the `action` method. It has two buttons (named `New1` and `New2`). When any of the buttons is pressed the `action`

method is called. Figure 9.1 shows the display when either of the buttons are pressed. In the left hand side of Figure 9.1 the `New1` button is pressed and the right-hand side shows the display after the `New2` button is pressed. It can be seen that differences are in the `target`, `arg` parameter and the `x`, `y` co-ordinate parameters.

📖 Java applet 9.2 (↯Java 1.0)

```
import java.applet.*;
import java.awt.*;

public class chap9_02 extends Applet
{
String Msg1=null, Msg2, Msg3, Msg4;
Button new1,new2;

    public void init()
    {
        new1=new Button("New 1");
        new2=new Button("New 2");
        add (new1); add(new2);
    }

    public boolean action(Event evt, Object obj)
    {
        Msg1= "Target= "+evt.target;
        Msg2= "When= " + evt.when + " id=" + evt.id +
              " x= "+ evt.x + " y= " + evt.y;
        Msg3= "Arg= " + evt.arg + " Key= " + evt.key;
        Msg4= "Click= " + evt.clickCount;
        repaint();
        return true;
    }

    public void paint(Graphics g)
    {
        if (Msg1!=null)
        {
             g.drawString(Msg1,30,80);
             g.drawString(Msg2,30,100);
             g.drawString(Msg3,30,120);
             g.drawString(Msg4,30,140);
        }
    }
}
```


**Figure 9.1** Sample runs

Thus to determine the button that has been pressed the `evt.arg` string can be tested. Java applet 9.3 shows an example where the `evt.arg` parameter is tested for its string content.

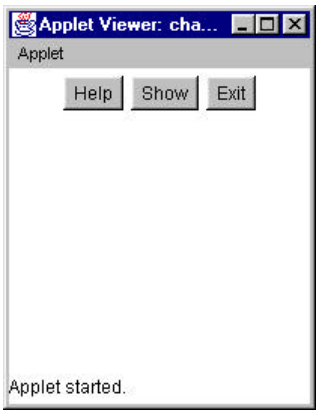
📖 Java applet 9.3 (↯Java 1.0)

```
import java.applet.*;
import java.awt.*;

public class chap9_03 extends Applet
{
String   Msg=null;
Button   new1, new2;
   public void init()
   {
      new1=new Button("New 1");     new2=new Button("New 2");
      add (new1); add(new2);
   }

   public boolean action(Event evt, Object obj)
   {
      if (evt.arg=="New 1") Msg= "New 1 pressed";
      else if (evt.arg=="New 2") Msg= "New 2 pressed";
      repaint();
      return true;
   }

   public void paint(Graphics g)
   {
      if (Msg!=null)
         g.drawString(Msg,30,80);
   }
}
```

Java applet 9.4 uses the `action` method which is called when an event occurs. Within this method the `event` variable is tested to see if one of the buttons caused the event. This is achieved with:

```
if (event.target instanceof Button)
```

If this test is true then the `Msg` string takes on the value of the object, which holds the name of the button that caused the event.




📖 Java applet 9.4 (↯Java 1.0)

```
import java.awt.*;
import java.applet.*;

public class chap9_04 extends Applet
{
String  Msg=null;

  public void init()
  {
   add(new Button("Help"));
   add(new Button("Show"));
   add(new Button("Exit"));
  }

  public boolean action(Event event,
```

```
    Object object)
  {
   if (event.target instanceof Button)
   {
     Msg = (String) object;
     repaint();
   }
   return(true);
  }

  public void paint(Graphics g)
  {
   if (Msg!=null)
   g.drawString("Button:" + Msg,30,80);
  }
}
```

## 9.4 Action Listener in Java 1.1

As with mouse events, buttons, menus and textfields are associated with an action listener (named `ActionListener`). When an event associated with these occurs then the `actionPerformed` method is called. Its format is:

```
public void actionPerformed(ActionEvent evt)
```

where `evt` defines the event. The associated methods are:

- `getActionCommand()` is the action command
- `evt.getModifiers()` is the state of the modifier keys
- `evt.paramString()` is the parameter string

Java applet 9.5 contains an example of the `action` method. It has two buttons (named `New1` and `New2`). When any of the buttons is pressed the action method is called. Each of the buttons has an associated listener which is initiated with:

```
button1.addActionListener(this);
button2.addActionListener(this);
```

Figure 9.2 shows the display when either of the buttons are pressed. In the left-hand side of Figure 9.2 the `New1` button is pressed and the right-hand side shows the display after the `New2` button is pressed.

📖 Java applet 9.5 (↯Java 1.1)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class chap9_05 extends Applet implements ActionListener
```

```
{
Button   button1, button2;
String   Msg1=null, Msg2, Msg3;

   public void init()
   {

      button1 = new Button("New 1");
      button2 = new Button("New 2");
      add(button1); add(button2);
      button1.addActionListener(this);
      button2.addActionListener(this);
   }

   public void actionPerformed(ActionEvent evt)
   {
      Msg1= "Command= "+evt.getActionCommand();
      Msg2= "Modifiers= " + evt.getModifiers();
      Msg3= "String= " + evt.paramString();
      repaint();
   }

   public void paint(Graphics g)
   {
      if (Msg1!=null)
      {
         g.drawString(Msg1,30,80);
         g.drawString(Msg2,30,100);
         g.drawString(Msg3,30,120);
      }
   }
}
```

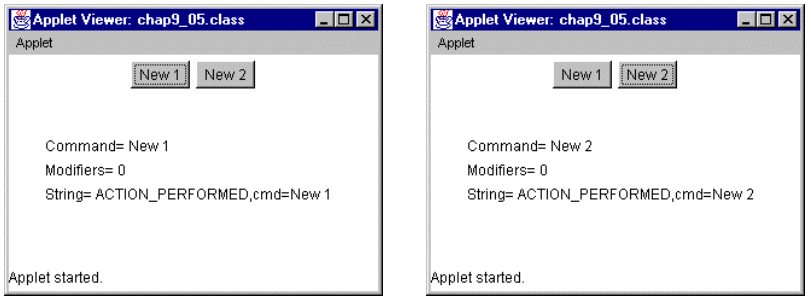



**Figure 9.2** Sample run

Thus to determine the button that has been pressed the `getActionCommand()` method is used. Java applet 9.6 shows an example where the `getActionCommand()` method is tested for its string content. Figure 9.3 shows a sample run.

### Java applet 9.6 (≠Java 1.1)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class chap9_06 extends Applet implements ActionListener
{

Button button1, button2;
```

```
String Msg=null;

    public void init()
    {

        button1 = new Button("New 1");
        button2 = new Button("New 2");
        add(button1); add(button2);
        button1.addActionListener(this);
        button2.addActionListener(this);
    }

    public void actionPerformed(ActionEvent evt)
    {
    String command;

        command=evt.getActionCommand();

        if (command.equals("New 1")) Msg="New 1 pressed";
        if (command.equals("New 2")) Msg="New 2 pressed";

        repaint();
    }

    public void paint(Graphics g)
    {
        if (Msg!=null)
        {
                g.drawString(Msg,30,80);
        }
    }
}
```

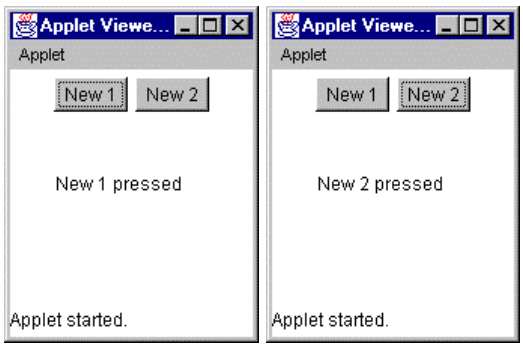



Figure 9.3 Sample run

## 9.5 Checkboxes

Typically checkboxes are used to select from a number of options. Java applet 9.7 shows how an applet can use checkboxes. As before, the `action` method is called when a checkbox changes its state and within the method the `event.target` parameter is tested for the checkbox with:

```
if (event.target instanceof Checkbox)
```

If this is true, then the method `DetermineCheckState()` is called which tests `event.target` for the checkbox value and its state (true or false).

Java applet 9.7 (↯Java 1.0)


```
import java.awt.*;
import java.applet.*;
public class chap9_07 extends Applet
{
String  Msg=null;
Checkbox fax, telephone, email, post;

 public void init()
 {
  fax=new Checkbox("FAX");
  telephone=new Checkbox("Telephone");
  email=new Checkbox("Email");
  post=new Checkbox("Post",null,true);
  add(fax); add(telephone);
  add(email); add(post);
 }

 public void DetermineCheckState(
     Checkbox Cbox)
 {
  Msg=Cbox.getLabel()+" "+ Cbox.getState();
  repaint();
 }

 public boolean action(Event event,
     Object object)
 {
  if (event.target instanceof Checkbox)
    DetermineCheckState(
      (Checkbox)event.target);
  return(true);
 }
 public void paint(Graphics g)
 {
  if (Msg!=null)
   g.drawString("Check box:" + Msg,30,80);
 }
}
```

## 9.6 Item listener in Java 1.1

As with mouse events, checkboxes and lists are associated with an item listener (named `ItemListener`). When an event associated with these occur then the `itemStateChanged` method is called. Its format is:

```
public void itemStateChanged(ItemEvent event)
```

where `event` defines the event. The associated methods are:

- `getItem()` is the item selected
- `getStateChange()` is the state of the checkbox
- `paramString()` is the parameter string

Java applet 9.8 contains an example of checkboxes and Figure 9.4 shows a sample run. Each of the checkboxes has an associated listener which is initiated in the form:

```
chbox.addItemListener(this);
```

📖 Java applet 9.8 (↯Java 1.1)

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class chap9_08 extends Applet implements ItemListener
{
String      Msg1=null,Msg2,Msg3;
Checkbox    fax, telephone, email,post;

    public void init()
    {
      fax=new Checkbox("FAX");
      telephone=new Checkbox("Telephone");
      email=new Checkbox("Email");
      post=new Checkbox("Post",null,true);
      add(fax);
      add(telephone);
      add(email);
      add(post);

      fax.addItemListener(this);
      email.addItemListener(this);
      telephone.addItemListener(this);
      post.addItemListener(this);
   }

   public void itemStateChanged(ItemEvent event)
   {
      Msg1=""+event.getItem();
      Msg2=""+event.getStateChange();
      Msg3=event.paramString();
      repaint();
   }

   public void paint(Graphics g)
   {
      if (Msg1!=null)
      {
         g.drawString(Msg1,30,80);
         g.drawString(Msg2,30,110);
         g.drawString(Msg3,30,150);
      }
   }
}
```

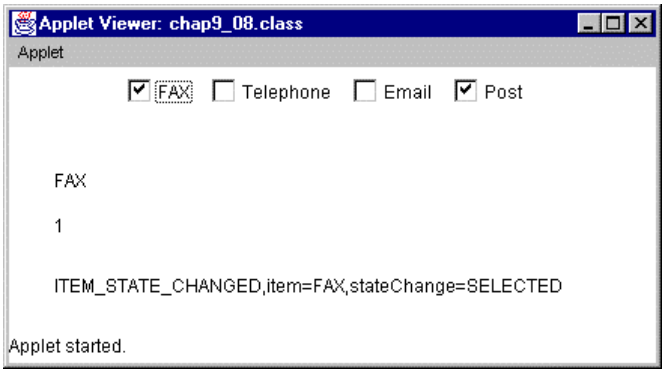



Figure 9.4 Sample run

## 9.7 Radio buttons

The standard checkboxes allow any number of options to be selected. A radio button allows only one option to be selected at a time. The program is changed by:

- Adding checkbox names (such as fax, tele, email and post).
- Initialising the checkbox with `CheckboxGroup()` to a checkbox group identifier.
- Add the identifier of the checkbox group to the `Checkbox()` method.
- Testing the target property of the event to see if it equals a checkbox name.

Java applet 9.9 shows how this is achieved.

📖 Java applet 9.9 (↯Java 1.0)

```
import java.awt.*;
import java.applet.*;

public class chap9_09 extends Applet
{
String      Msg=null;
Checkbox    fax, tele, email, post;

   public void init()
  {

   CheckboxGroup RadioGroup = new CheckboxGroup();

      add(fax=new Checkbox("FAX",RadioGroup,false));
      add(tele=new Checkbox("Telephone",RadioGroup,false));
      add (email=new Checkbox("Email",RadioGroup,false));
      add (post=new Checkbox("Post",RadioGroup,true));
```

```
  }

  public boolean action(Event event, Object object)
  {
     if (event.target.equals(fax)) Msg="FAX";
     else if (event.target.equals(tele)) Msg="Telephone";
     else if (event.target.equals(email)) Msg="Email";
     else if (event.target.equals(fax)) Msg="FAX";
     repaint();
     return(true);
  }

  public void paint(Graphics g)
  {

     if (Msg!=null) g.drawString("Check box:" + Msg,30,80);
  }
}
```

Java applet 9.10 show the Java 1.1 equivalent.

📖 Java applet 9.10 (↯Java 1.1)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class chap9_10 extends Applet implements ItemListener
{
String  Msg=null;
Checkbox fax, tele, email, post;

 public void init()
  {

   CheckboxGroup RadioGroup = new CheckboxGroup();

      add(fax=new Checkbox("FAX",RadioGroup,true));
      add(tele=new Checkbox("Telephone",RadioGroup,false));
      add (email=new Checkbox("Email",RadioGroup,false));
      add (post=new Checkbox("Post",RadioGroup,false));
      fax.addItemListener(this);
      tele.addItemListener(this);
      email.addItemListener(this);
      post.addItemListener(this);
  }

  public void itemStateChanged(ItemEvent event)
  {
  Object obj;
      obj=event.getItem();

      if (obj.equals("FAX")) Msg="FAX";
      else if (obj.equals("Telephone")) Msg="Telephone";
      else if (obj.equals("Email")) Msg="Email";
      else if (obj.equals("Post")) Msg="Post";
      repaint();
   }

   public void paint(Graphics g)
   {
      if (Msg!=null)    g.drawString("Check box:" + Msg,30,80);
   }
}
```

This sets the checkbox type to `RadioGroup` and it can be seen that only one of the checkboxes is initially set (that is, 'FAX'). Figure 9.5 shows a sample run. It should be noted that grouped checkboxes use a round circle with a dot (⊙), whereas ungrouped checkboxes use a square box with a check mark (☑).


**Figure 9.5** Sample run

## 9.8 Pop-up menu choices

To create a pop-up menu the `Choice` object is initially created with:

```
Choice mymenu = new Choice();
```

After this the menu options are defined using the `addItem` method. Java applet 9.11 shows an example usage of a pop-up menu.

Java applet 9.11 (↯Java 1.0)

```
import java.awt.*;
import java.applet.*;

public class chap9_11 extends Applet
{
String  Msg=null;
Choice  mymenu= new Choice();

  public void init()
  {
    mymenu.addItem("FAX");
    mymenu.addItem("Telephone");
    mymenu.addItem("Email");
    mymenu.addItem("Post");
    add(mymenu);
  }
  public void DetermineCheckState(
     Choice mymenu)
```

```
   {
     Msg=mymenu.getItem(
        mymenu.getSelectedIndex());
     repaint();
   }
   public boolean action(Event event,
       Object object)
   {
    if (event.target instanceof Choice)
     DetermineCheckState(
        (Choice)event.target);
    return(true);
   }

   public void paint(Graphics g)
   {
   if (Msg!=null)
    g.drawString("Menu select:"+Msg,30,120);
   }
 }
```

As before the `arg` property of the event can also be tested as shown in Java applet 9.12. Java applet 9.13 gives the Java 1.1 equivalent.

📖 Java applet 9.12 (↯Java 1.0)

```
import java.awt.*;
import java.applet.*;

public class chap9_12 extends Applet
{
String  Msg=null;
Choice  mymenu= new Choice();

  public void init()
  {
    mymenu.addItem("FAX");
    mymenu.addItem("Telephone");
    mymenu.addItem("Email");
    mymenu.addItem("Post");
    add(mymenu);
  }
  public boolean action(Event event, Object object)
  {

   if (event.arg=="FAX") Msg="FAX";
   else if (event.arg=="Telephone") Msg="Telephone";
   else if (event.arg=="Email") Msg="Email";
   else if (event.arg == "Post") Msg="Post";
   repaint();
   return(true);
  }
  public void paint(Graphics g)
  {
    if (Msg!=null)
      g.drawString("Menu select:" + Msg,30,120);
  }
}
```

📖 Java applet 9.13 (↯Java 1.1)

```
import java.awt.*;
```

```
import java.awt.event.*;
import java.applet.*;

public class chap9_13 extends Applet implements ItemListener
{
String  Msg=null;
Choice  mymenu= new Choice();

  public void init()
  {
    mymenu.addItem("FAX");
    mymenu.addItem("Telephone");
    mymenu.addItem("Email");
    mymenu.addItem("Post");
    add(mymenu);
    mymenu.addItemListener(this);
  }
  public void itemStateChanged(ItemEvent event)
  {
  Object obj;

   obj=event.getItem();

   if (obj.equals("FAX")) Msg="FAX";
   else if (obj.equals("Telephone")) Msg="Telephone";
   else if (obj.equals("Email")) Msg="Email";
   else if (obj.equals("Post")) Msg="Post";
   repaint();
  }
  public void paint(Graphics g)
  {
    if (Msg!=null)
      g.drawString("Menu select:" + Msg,30,120);
  }
}
```

## 9.9 Other pop-up menu options

The `java.awt.Choice` class allows for a pop-up menu. It includes the following methods:

| | |
|---|---|
| `public void` **`addItem`**`(String item);` | Adds a menu item to the end. |
| `public void` **`addNotify`**`();` | Allows the modification of a list's appearance without changing its functionality. |
| `public int` **`countItems`**`();` | Returns the number of items in the menu. |
| `public String` **`getItem`**`(int index);` | Returns the string of the menu item at that index value. |
| `public int` **`getSelectedIndex`**`();` | Returns the index value of the selected item. |

| | |
|---|---|
| `public String` **`getSelectedItem`**`();` | Returns the string of the selected item. |
| `protected String` **`paramString`**`();` | Returns the parameter String of the list. |
| `public void` **`select`**`(int pos);` | Select the menu item at a given index. |
| `public void` **`select`**`(String str);` | Select the menu item with a given string name. |

The `countItems` method is used to determine the number of items in a pop-up menu, for example:

```
Msg= "Number of items is " + mymenu.countItems()
```

The `getItem(int index)` returns the string associated with the menu item, where the first item has a value of zero. For example:

```
Msg= "Menu item number 2 is " + mymenu.getItem(2);
```

Java applet 9.14 uses the `select` method to display the second menu option as the default and the `getItem` method to display the name of the option.

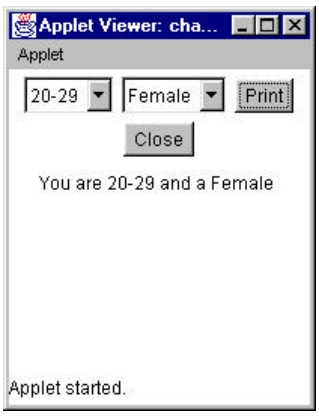
📖 Java applet 9.14 (↯Java 1.1)


```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class chap9_14 extends Applet
     implements ItemListener
{
String  Msg=null;
Choice  mymenu= new Choice();

 public void init()
 {
   mymenu.addItem("FAX");
   mymenu.addItem("Telephone");
   mymenu.addItem("Email");
   mymenu.addItem("Post");
   add(mymenu);
   mymenu.addItemListener(this);
   mymenu.select(1);
       // Select item 1 (Telephone)
}

public void itemStateChanged(ItemEvent evt)
{
Object obj;

 obj=evt.getItem();

 if (obj.equals("FAX"))
     Msg=mymenu.getItem(0);
 else if (obj.equals("Telephone"))
     Msg=mymenu.getItem(1);
```

```
 else if (obj.equals("Email"))
     Msg=mymenu.getItem(2);
 else if (obj.equals("Post"))
     Msg=mymenu.getItem(3);
 repaint();
}
public void paint(Graphics g)
{
 if (Msg!=null)
   g.drawString("Menu select:"+Msg,30,120);
}
}
```

## 9.10 Multiple menus

Multiple menus can be created in a Java applet and the `action` event can be used to differentiate between the menus. Java applet 9.15 has two pull-down menus and two buttons (`age`, `gender`, `print` and `close`). The event method `getItem` is then used to determine which of the menus was selected. In this case the `print` button is used to display the options of the two pull-down menus and `close` is used to exit from the applet.




Java applet 9.15 (↯Java 1.1)

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class chap9_15 extends Applet
  implements ItemListener, ActionListener
{
Choice age = new Choice();
Choice gender = new Choice();
Button print= new Button("Print");
Button close= new Button("Close");
String gendertype=null, agetype=null;

String Msg, Options[];

  public void init()
  {
      age.addItem("10-19");
      age.addItem("20-29");
      age.addItem("30-39");
      age.addItem("40-49");
      age.addItem("Other");
      add(age);

      gender.addItem("Male");
      gender.addItem("Female");
      add(gender);
      add(print);
      add(close);

      age.addItemListener(this);
      gender.addItemListener(this);
      print.addActionListener(this);
```

```
      close.addActionListener(this);
   }

  public void itemStateChanged(ItemEvent evt)
  {
   int i;
   Object obj;

      obj=evt.getItem();

   if (obj.equals("10-19")) agetype="10-19";
   else if (obj.equals("20-29"))
      agetype="20-29";
   else if (obj.equals("30-39"))
     agetype="30-39";
   else if (obj.equals("40-49"))
     agetype="40-49";
   else if (obj.equals("Other"))
     agetype="Other";
   else if (obj.equals("Male"))
     gendertype="Male";
   else if (obj.equals("Female"))
     gendertype="Female";
   }

 public void actionPerformed(ActionEvent evt)
 {
 String str;

   str=evt.getActionCommand();
   if (str.equals("Print"))  repaint();
   else if (str.equals("Close"))
     System.exit(0);
 }

 public void paint(Graphics g)
 {
  if ((agetype!=null) && (gendertype!=null))
   Msg="Your are " + agetype + " and a "
                   + gendertype;
  else Msg="Please select age and gender";

  if (Msg!=null) g.drawString(Msg,20,80);
 }
 }
```

## 9.11 Menu bar

Menu bars are now familiar in most GUIs (such as Microsoft Windows and Motif). They consist of a horizontal menu bar with pull-down submenus.

The `java.awt.MenuBar` class contains a constructor for a menu bar. Its format is:

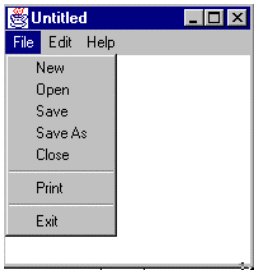
```
public MenuBar();
```

and the methods which can be applied to it are:

| | |
|---|---|
| `public Menu add(Menu m);` | Adds the specified menu to the menu bar. |
| `public void addNotify();` | Allows a change of appearance of the menu bar without changing any of the menu bar's functionality. |
| `public int countMenus();` | Counts the number of menus on the menu bar. |
| `public Menu getHelpMenu();` | Gets the help menu on the menu bar. |
| `public Menu getMenu(int i);` | Gets the specified menu. |
| `public void remove(int index);` | Removes the menu located at the specified index from the menu bar. |
| `public void remove(`<br>`  MenuComponent m);` | Removes the specified menu from the menu bar. |
| `public void removeNotify();` | Removes notify. |
| `public void setHelpMenu(`<br>`  Menu m);` | Sets the help menu to the specified menu on the menu bar. |

Java program 9.16 gives an example of using a menu bar. Initially the menu bar is created with the `MenuBar()` constructor, and submenus with the `Menu` constructors (in this case, `mfile`, `medit` and `mhelp`). Items are added to the submenus with the `MenuItem` constructor (such as `New`, `Open`, and so on). A `handleEvent()` method has been added to catch a close window operation. The `addSeparator()` method has been added to add a line between menu items. Note that this program is not an applet so that it can be run directly with the Java interpreter (such as `java.exe`).

Java standalone program 9.16 (↯Java 1.0)




```
import java.awt.*;

public class gomenu extends Frame
{
MenuBar mainmenu = new MenuBar();
Menu mfile = new Menu("File");
Menu medit = new Menu("Edit");
Menu mhelp = new Menu("Help");

    public gomenu()
    {
        mfile.add(new MenuItem("New"));
        mfile.add(new MenuItem("Open"));
        mfile.add(new MenuItem("Save"));
        mfile.add(new MenuItem("Save As"));
```

```
        mfile.add(new MenuItem("Close"));
        mfile.addSeparator();
        mfile.add(new MenuItem("Print"));
        mfile.addSeparator();
        mfile.add(new MenuItem("Exit"));

        mainmenu.add(mfile);

        medit.add(new MenuItem("Cut"));
        medit.add(new MenuItem("Copy"));
        medit.add(new MenuItem("Paste"));
        mainmenu.add(medit);

        mhelp.add(new MenuItem("Commands"));
        mhelp.add(new MenuItem("About"));
        mainmenu.add(mhelp);

        setMenuBar(mainmenu);
    }

    public boolean action(Event evt, Object obj)
    {
        if (evt.target instanceof MenuItem)
        {
             if (evt.arg=="Exit") System.exit(0);
        }
        return true;
    }

    public boolean handleEvent(Event evt)
    {
        if (evt.id == Event.WINDOW_DESTROY)
               System.exit(0);
        return true;
    }

    public static void main(String args[])
    {
        Frame f = new gomenu();
        f.resize(400,400);
        f.show();
    }
}
```

## 9.12 List box

A `List` component creates a scrolling list of options (where in a pull-down menu only one option can be viewed at a time). The `java.awt.List` class contains the `List` constructor which can be used to display a list component., which is in the form:

```
public List();
public List(int rows, boolean multipleSelections);
```

where `row` defines the number of rows in a list and `multipleSelections` is true when the user can select a number of selections, else it is false.

The methods that can be applied are:

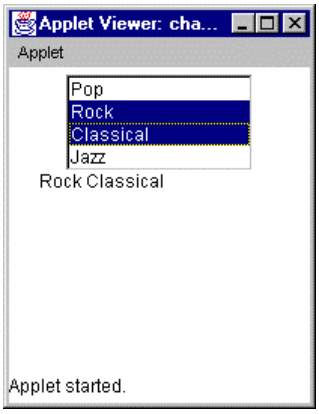
| | |
|---|---|
| `public void addItem(String item);` | Adds a menu item at the end. |
| `public void addItem(String item, int index);` | Add a menu item at the end. |
| `public void addNotify();` | Allows the modification of a list's appearance without changing its functionality. |
| `public boolean allowsMultipleSelections();` | Allows the selection of multiple selections. |
| `public void clear();` | Clears the list. |
| `public int countItems();` | Returns the number of items in the list. |
| `public void delItem(int position);` | Deletes an item from the list. |
| `public void delItems(int start, int end);` | Deletes items from the list. |
| `Public void deselect(int index);` | Deselects the item at the specified index. |
| `Public String getItem(int index);` | Gets the item associated with the specified index. |
| `public int getRows();` | Returns the number of visible lines in this list. |
| `public int getSelectedIndex();` | Gets the selected item on the list. |
| `public int[] getSelectedIndexes();` | Gets selected items on the list. |
| `public String getSelectedItem();` | Returns the selected item on the list as a string. |
| `public String[] getSelectedItems();` | Returns the selected items on the list as an array of strings. |
| `public int getVisibleIndex();` | Gets the index of the item that was last made visible by the method `makeVisible`. |
| `public boolean isSelected( int index);` | Returns true if the item at the specified index has been selected. |
| `public void makeVisible(int index);` | Makes a menu item visible. |

| | |
|---|---|
| `public Dimension` **`minimumSize`**`();` | Returns the minimum dimensions needed for the list. |
| `public Dimension` **`minimumSize`**`(int rows);` | Returns the minimum dimensions needed for the amount of rows in the list. |
| `protected String` **`paramString`**`();` | Returns the parameter String of the list. |
| `public Dimension` **`preferredSize`**`();` | Returns the preferred size of the list. |
| `public Dimension` **`preferredSize`**`(int rows);` | Returns the preferred size of the list. |
| `public void` **`removeNotify`**`();` | Removes notify. |
| `public void` **`replaceItem`**`(String newValue, int index);` | Replaces the item at the given index. |
| `Public void` **`select`**`(int index);` | Selects the item at the specified index. |
| `public void` **`setMultipleSelections`**`(boolean v);` | Allows multiple selections. |

Java program 9.17 shows an example of a program with a list component. Initially the list is created with the `List` constructor. The `addItem` method is then used to add the four items ("Pop", "Rock", "Classical" and "Jazz"). Within `actionPerformed` the program uses the `Options` array of strings to build up a message string (`Msg`). The `Options.length` parameter is used to determine the number of items in the array.

📖 Java program 9.17 (↯Java 1.1)

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class chap9_17 extends Applet
                  implements ActionListener
{
List lmenu = new List(4,true);
String Msg, Options[];

    public void init()
    {
        lmenu.addItem("Pop");
        lmenu.addItem("Rock");
        lmenu.addItem("Classical");
        lmenu.addItem("Jazz");
        add(lmenu);
        lmenu.addActionListener(this);
```



```
 }

public void actionPerformed(
                      ActionEvent evt)
{
   int i;
   String str;

    str=evt.getActionCommand();
    Options=lmenu.getSelectedItems();
    Msg="";
    for (i=0;i<Options.length;i++)
           Msg=Msg+Options[i] + " ";
    repaint();
}

public void paint(Graphics g)
{

  if (Msg!=null) g.drawString(Msg,20,80);
 }
}
```

## 9.13 File dialog

The `java.awt.Filedialog` class contains the `FileDialog` constructor which can be used to display a dialog window. To create a dialog window the following can be used:

```
public FileDialog(Frame parent, String title);
public FileDialog(Frame parent, String title, int mode);
```

where the `parent` is the owner of the dialog, `title` is the title of the dialog window and the `mode` is defined as whether the file is to be loaded or save. Two fields are defined for the mode, these are:

```
public final static int LOAD;
public final static int SAVE;
```

The methods that can be applied are:

| | |
|---|---|
| `public void` **`addNotify`**`();` | Allows applications to change the look of a file dialog window without changing its functionality. |
| `public String` **`getDirectory`**`();` | Gets the initial directory. |
| `public String` **`getFile`**`();` | Gets the file that the user specified. |
| `public FilenameFilter` **`getFilenameFilter`**`();` | Sets the default file filter. |

| | |
|---|---|
| `public int` **`getMode`**`();` | Indicates whether the file dialog box is for file loading from or file saving. |
| `protected String` **`paramString`**`();` | Returns the parameter string representing the state of the file dialog window. |
| `public void` **`setDirectory`**`(String dir);` | Gets the initial directory. |
| `public void` **`setFile`**`(String file);` | Sets the selected file for this file dialog window to be the specified file. |
| `public void setFilenameFilter( FilenameFilter filter);` | Sets the filename filter for the file dialog window to the specified filter. |

## 9.14 Exercises

**9.14.1** Modify Applet 9.14 so that the following are added:

(i) a `countItems()` method which displays the number of items in the menu.
(ii) a `getSelectIndex()` method which displays the item value of the item selected.
(iii) the `select()` method so that that email item is shown as the default. Note that it can be selected either with a string or the position.

**9.14.2** Implement the two Java applets in Figure 9.6. The `Show` button should display all the selected options.

**9.14.3** Add a `Reset` button to the applets developed in Exercise 9.14.2. This button should set the selected items back to their initial values.

**9.14.4** Implement a Java applet which displays the button on a basic calculator. An example is shown in Figure 9.7.

**9.14.5** Implement a Java applet, with checkboxes, which prompts a user for their personal information such as height, weight, and so on.

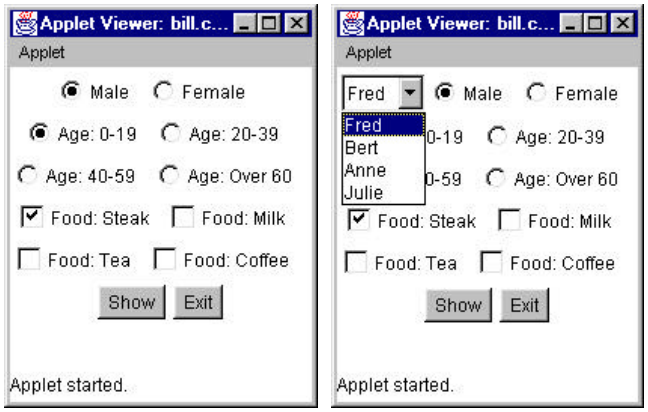



Figure 9.6 Sample runs

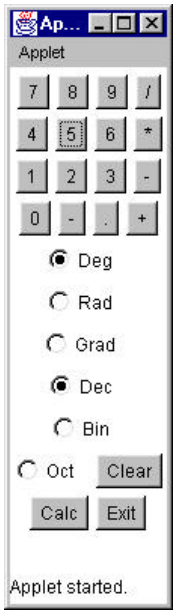



Figure 9.7 Sample runs