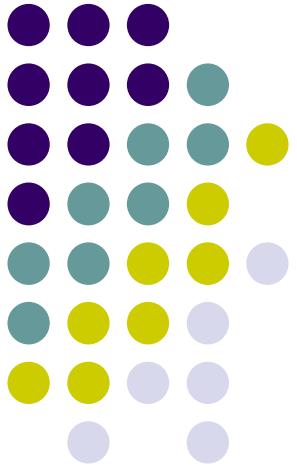
第十九章
AWT的其它物件

學習選擇表單與下拉選單

學習功能表的製作

學習捲軸的製作

學習各種對話方塊與其相關應用



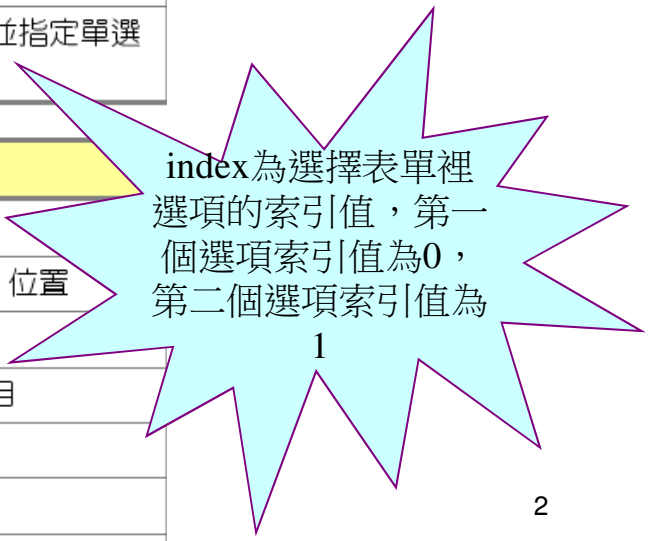
建立選擇表單 (1/2)

- 選擇表單（list）可用來顯示多個選項
- 下表列出List類別的建構元與常用的method

表 19.1.1　java.awt.List 的建構元與 method

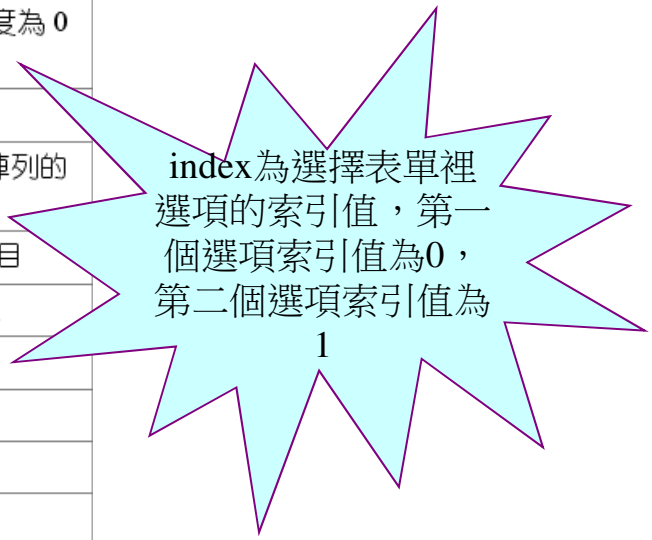
建構元	主要功能
List()	建立選擇表單物件，預設的顯示列數為 4 列
List(int rows)	建立可顯示 rows 列的表單物件
List(int rows, boolean multipleMode)	建立可顯示 rows 列的表單物件，並指定單選或複選

method	主要功能
void add(String item)	加入一個選項到選擇表單中
void add(String item, int index)	加入一個選項到選擇表單中的 index 位置
void addActionListener(ActionListener l)	將傾聽者加入選擇表單
void addItemListener(ItemListener l)	將傾聽者加入選擇表單裡特定的項目
void deselect(int index)	取消選擇表單裡第 index 項的選取
String getItem(int index)	取得選擇表單裡的第 index 項目
int getItemCount()	取得選擇表單裡選項的數目



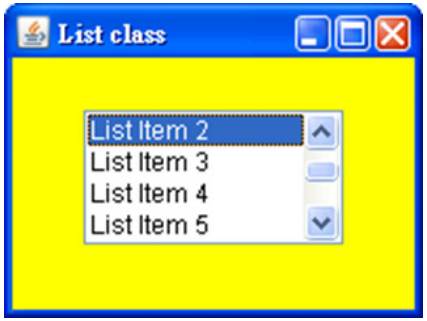
建立選擇表單 (2/2)

method	主要功能
String[] getItems()	取得選擇表單裡選項的所有項目，並以字串陣列的形式傳回
int getRows()	取得可顯示的列數
int getSelectedIndex()	傳回被選取項目的 index 值。若未有項目被選取，或者是多個項目被選取，則傳回-1
int[] getSelectedIndexes()	以整數陣列的型式傳回所有被選取項目的 index 值。若沒有項目被選取，則傳回長度為 0 的陣列
String getSelectedItem()	傳回被選取項目的名稱
String[] getSelectedItems()	取得所有被選取項目的名稱，並以字串陣列的格式傳回
int getVisibleIndex()	取得由 makeVisible() method 所設定的項目
boolean isIndexSelected(int index)	查詢第 index 個選項是否為被選取的狀態
boolean isMultipleMode()	查詢選擇表單是否為複選
void makeVisible(int index)	設定第 index 個選項顯示在選擇表單裡
void remove(int position)	移去位置為 position 的項目
void remove(String item)	移去名稱為 item 的項目
void removeAll()	移去所有的 item 項目
void replaceItem(String str, int index)	將位置為 index 的項目用 str 來取代
void select(int index)	設定第 index 項為被選取的狀態
void setMultipleMode(boolean b)	設定選擇表單是否允許複選



選擇表單的範例

- app19_1是建立選擇表單的簡單範例




```
01  // app19_1, 建立選擇表單
02  import java.awt.*;
03  public class app19_1 extends Frame
04  {
05     static app19_1 frm=new app19_1();
06     static List lst=new List();          // 建立選擇表單物件 lst
07     public static void main(String args[])
08     {
09        frm.setLayout(new FlowLayout(FlowLayout.CENTER,10,25));
10        frm.setTitle("List class");
11        for(int i=0;i<=9;i++)             // 利用 for 迴圈加入選項
12           lst.add("List Item "+i);
13        lst.select(2);                    // 選取索引值為 2 的選項
14        frm.setSize(200,150);
15        frm.add(lst);
16        frm.setBackground(Color.yellow);
17        frm.setVisible(true);
18        System.out.println("lst.getRows()= "+lst.getRows());
19        System.out.println("lst.getItemCount()= "+lst.getItemCount());
20     }
21  }
```

```
/* app19_1 OUTPUT-----

lst.getRows()= 4
lst.getItemCount()= 10
----------------------*/
```

選擇表單的事件處理 (1/2)

- 選取選擇表單內的選項時
  - ItemEvent事件會被觸發
  - 利用addItemListener() 把事件傾聽者向List類別的物件註冊
  - 事件處理的程式碼要撰寫在itemStateChanged() method
- app19_2是選擇表單的事件處理範例

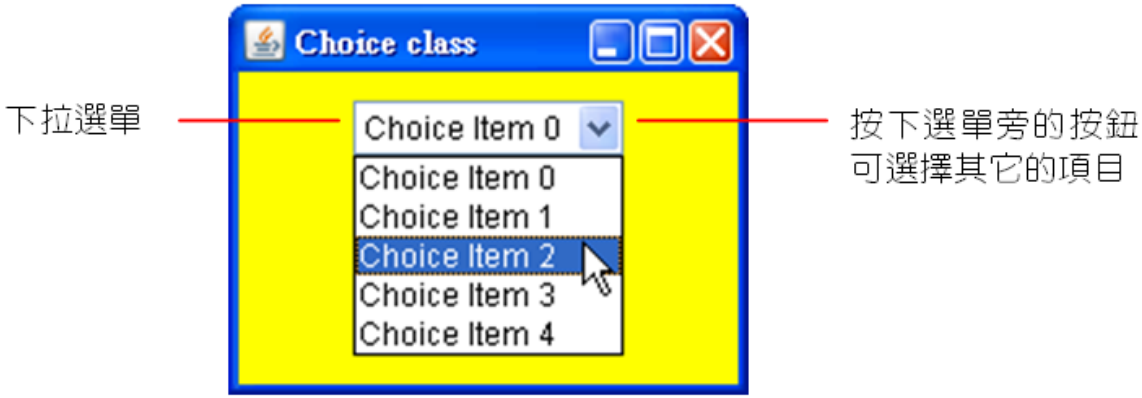

```
01  // app19_2, 選擇表單的事件處理範例
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app19_2 extends Frame implements ItemListener
05  {
06     static app19_2 frm=new app19_2();
07     static List lst=new List();                    // 建立選擇表單物件 lst
08     public static void main(String args[])
09     {
```

選擇表單的事件處理 (2/2)

```
10          frm.setLayout(new FlowLayout(FlowLayout.CENTER,10,25));
11          frm.setTitle("Select a color");
12          lst.add("yellow");                          // 加入選項到 lst 內
13          lst.add("orange");
14          lst.add("pink");
15          lst.add("cyan");
16          lst.addItemListener(frm);                   // 設定frm為lst的傾聽者
17          frm.add(lst);
18          frm.setSize(200,150);
19          frm.setVisible(true);
20      }
21      public void itemStateChanged(ItemEvent e)     // 事件處理的程式碼
22      {
23          String color=lst.getSelectedItem();       // 取得被選取之選項名稱
24          if(color=="yellow")                       // 如果選項名稱為 yellow
25              frm.setBackground(Color.yellow);
26          else if(color=="orange")                  // 如果選項名稱為 orange
27              frm.setBackground(Color.orange);
28          else if(color=="pink")                    // 如果選項名稱為 pink
29              frm.setBackground(Color.pink);
30          else if(color=="cyan")                    // 如果選項名稱為 cyan
31              frm.setBackground(Color.cyan);
32          frm.setTitle("you select "+color);        // 設定視窗 frm 的標題
33      }
34  }
```

下拉選單的認識

- 下拉選單（choice）
  - 提供多個選項以供選取
  - 只能選擇單一的項目
  - 下圖為下拉選單示意圖：



Choice類別的建構元與method

- 下表列出Choice類別的建構元與常用的method：

表 19.2.1　java.awt.Choice 的建構元與 method

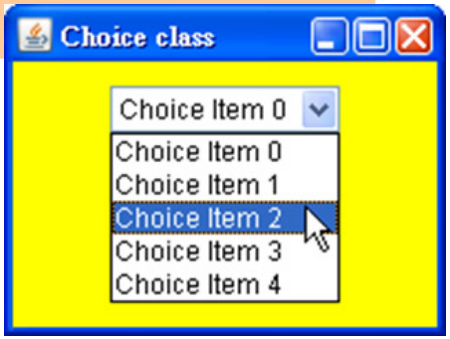
建構元	主要功能
Choice()	建立下拉選單物件

method	主要功能
void add(String item)	加入 item 到選單中
void addItemListener(ItemListener l)	加入事件傾聽者
String getItem(int index)	取出第 index 項目的名稱
int getItemCount()	傳回下拉選單中選項的總數
int getSelectedIndex()	傳回下拉選單中，被選取項目的 index 值
String getSelectedItem()	傳回下拉選單中，被選取項目的名稱
void insert(String item, int index)	在 index 的位置加下 item 選項
void remove(int position)	移除 position 位置的選項
void remove(String item)	移除名稱為 item 的選項
void removeAll()	移除所有的項目
void select(int pos)	設定 pos 位置的項目為被選取的狀態
void select(String str)	設定名稱為 str 的項目為被選取的狀態

建立下拉選單

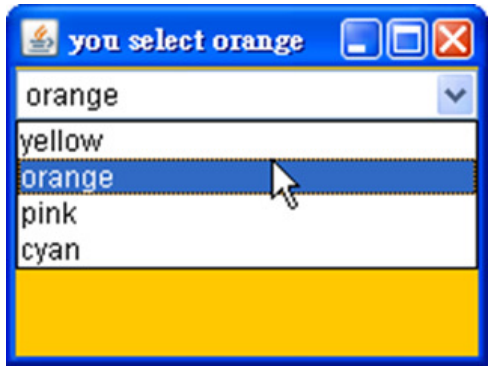
- app19_3以Choice下拉選單來顯示選項：

```
01  // app19_3, 建立選擇表單
02  import java.awt.*;
03  public class app19_3 extends Frame
04  {
05     static app19_3 frm=new app19_3();
06     static Choice chc=new Choice();      // 建立下拉選單物件 chc
07     public static void main(String args[])
08     {
09        frm.setLayout(new FlowLayout(FlowLayout.CENTER,10,10));
10        frm.setTitle("Choice class");
11        for(int i=0;i<=4;i++)              // 利用 for 迴圈加入選項
12           chc.add("Choice Item "+i);
13        frm.add(chc);
14        frm.setSize(200,150);
15        frm.setBackground(Color.yellow);
16        frm.setVisible(true);
17     }
18  }
```



下拉選單的事件處理 (1/2)

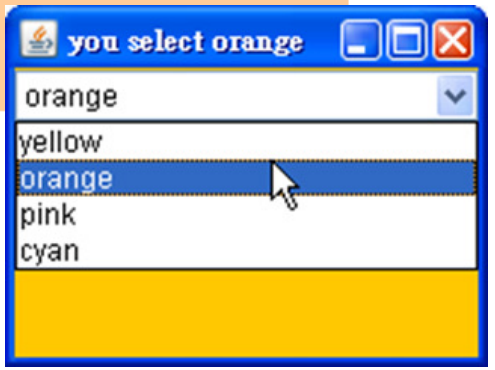
- 選取下拉選單內的選項時會觸發ItemEvent事件
- app19_4是下拉選單的事件處理範例




```
01  // app19_4, 下拉選單的事件處理範例
02  import java.awt.*;
03  import java.awt.event.*;
04  class app19_4 extends Frame implements ItemListener
05  {
06     static app19_4 frm=new app19_4();
07     static Choice chc=new Choice();   // 建立下拉表單物件 chc
08     public static void main(String args[])
09     {
10        BorderLayout br=new BorderLayout();
11        frm.setLayout(br);
12        frm.setTitle("Select a color");
13        chc.add("yellow");              // 加入選項到下拉表單物件 chc 裡
14        chc.add("orange");
15        chc.add("pink");
16        chc.add("cyan");
17        chc.addItemListener(frm);       // 設定 frm 為 chc 的事件傾聽者
```

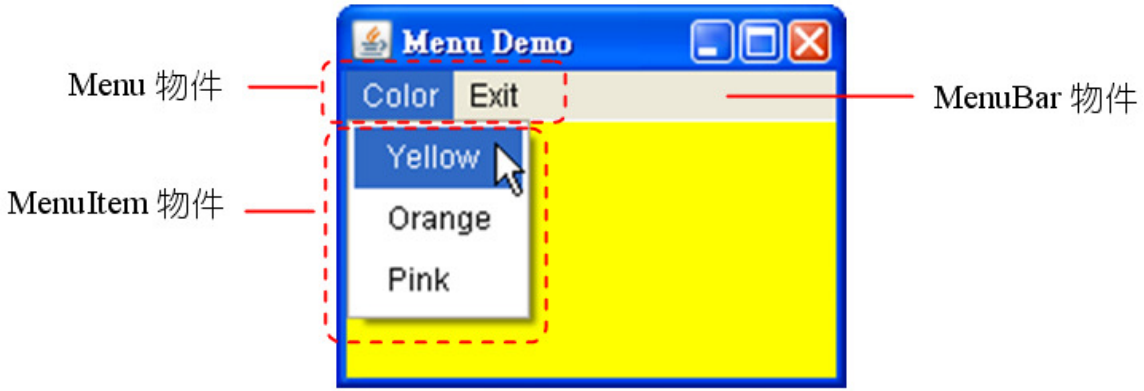
下拉選單的事件處理 (2/2)

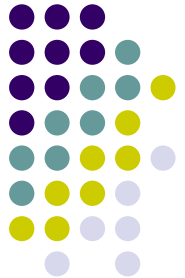
```
18          frm.setSize(200,150);
19          frm.add(chc,br.NORTH);
20          frm.setVisible(true);
21       }
22       public void itemStateChanged(ItemEvent e)
23       {
24          String color=chc.getSelectedItem();
25          if(color=="yellow")              // 選擇 yellow 選項
26             frm.setBackground(Color.yellow);
27          else if(color=="orange")         // 選擇 orange 選項
28             frm.setBackground(Color.orange);
29          else if(color=="pink")           // 選擇 pink 選項
30             frm.setBackground(Color.pink);
31          else if(color=="cyan")           // 選擇 cyan 選項
32             frm.setBackground(Color.cyan);
33          frm.setTitle("you select "+color);
34       }
35    }
```



認識功能表

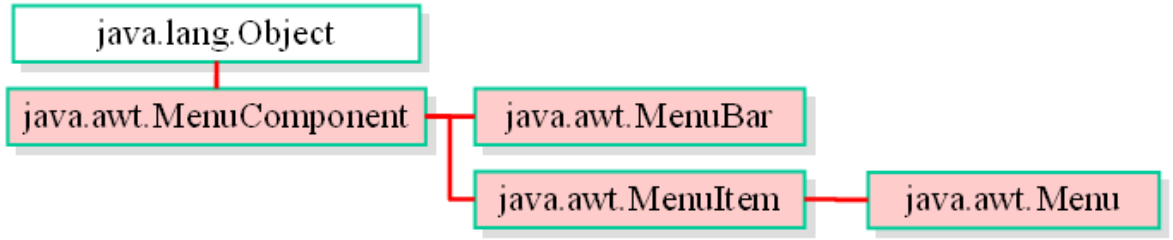
- 功能表可將每項功能分門別類存放，方便使用者選取
- 完整的功能表是由三種功能表類別所建立
  - 「MenuBar」
  - 「Menu」
  - 「MenuItem」
- 下圖說明這三種類別的物件於功能表裡扮演的角色：

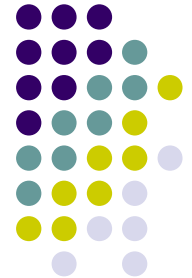




功能表類別的繼承關係

- MenuItem與MenuBar繼承自MenuComponent類別
- Menu繼承MenuItem類別
- 功能表類別的繼承關係圖：





MenuBar類別的建構元與method

- MenuBar類別提供的建構元與method：

表 19.3.1　java.awt.MenuBar 的建構元與 method

建構元	主要功能
MenuBar()	建立 MenuBar 物件

method	主要功能
Menu add(Menu m)	將 Menu 物件 m 加到 MenuBar 中
Menu getMenu(int i)	取得指定位置的 Menu 物件
int getMenuCount()	取得 MenuBar 中 Menu 物件的總數
void remove(int index)	移除指定位置的 Menu 物件
void remove(MenuComponent m)	移除 MenuComponent 物件 m

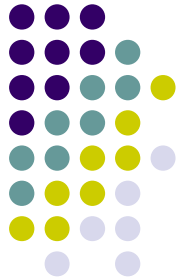
Menu類別的建構元與method

- Menu類別提供的建構元與method：

表 19.3.2　java.awt.Menu 的建構元與 method

建構元	主要功能
Menu()	建立 Menu 物件
Menu(String label)	建立標題為 label 的 Menu 物件

method	主要功能
MenuItem add(MenuItem mi)	增加 MenuItem 物件 mi 到 Menu 中
void add(String label)	增加標題為 label 的 MenuItem 物件到 Menu 中
void addSeparator()	在目前的位置增加一行分隔線
MenuItem getItem(int index)	傳回指定位置的 MenuItem 物件
int getItemCount()	傳回目前的 Menu 物件裡，MenuItme 的總數
void insert(MenuItem mi, int index)	在 index 位置插入 MenuItem 物件 mi
void insert(String label, int index)	在 index 位置增加標題為 label 的 MenuItem 物件
void insertSeparator(int index)	在 index 位置增加一行分隔線
void remove(int index)	移除 index 位置的 MenuItem 物件
void removeAll()	移除 Menu 中所有的 MenuItem 物件



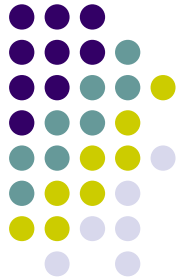
MenuItem類別的建構元與method

- MenuItem類別提供的建構元與method：

表 19.3.3　java.awt.MenuItem 的建構元與 method

建構元	主要功能
MenuItem()	建立 MenuItem 物件
MenuItem(String label)	建立標題為 label 的 MenuItem 物件

method	主要功能
void addActionListener(ActionListener l)	加入傾聽者物件
String getLabel()	取得 MenuItem 的標題
boolean isEnabled()	查詢 MenuItem 是否可以使用
void setEnabled(boolean b)	設定 MenuItem 可以使用
void setLabel(String label)	設定 MenuItem 的標題為 label



建立功能表

- 要建立完整的功能表，步驟為

  (1) 分別建立MenuBar、Menu與MenuItem物件

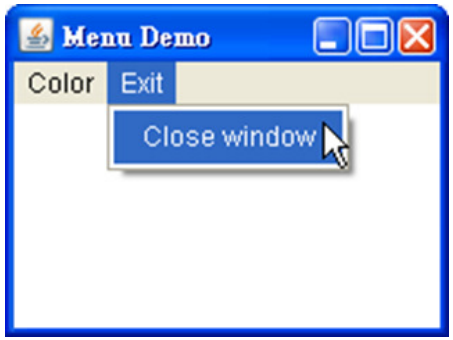
  (2) 用add() method把Menu物件加到MenuBar物件

  (3) 把MenuItem物件加到Menu物件中

建立功能表的範例

app19_5是建立功能表的範例

```
01  // app19_5, 建立功能表
02  import java.awt.*;
03  public class app19_5
04  {
05     static Frame frm=new Frame("Menu Demo");
06     static MenuBar mb=new MenuBar();              // 建立 MenuBar 物件
07     static Menu menu1=new Menu("Color");          } 建立 Menu 物件
08     static Menu menu2=new Menu("Exit");           }
09     static MenuItem mi1=new MenuItem("Yellow");   }
10     static MenuItem mi2=new MenuItem("Orange");   } 建立 MenuItem
11     static MenuItem mi3=new MenuItem("Pink");     } 物件
12     static MenuItem mi4=new MenuItem("Close window");
13
14     public static void main(String args[])
15     {
16        mb.add(menu1);                             // 將 menu1 加入 mb 中
17        mb.add(menu2);                             // 將 menu2 加入 mb 中
18        menu1.add(mi1);                            // 將 mi1 加入 menu1 中
19        menu1.add(mi2);                            // 將 mi2 加入 menu1 中
20        menu1.add(mi3);                            // 將 mi3 加入 menu1 中
21        menu2.add(mi4);                            // 將 mi4 加入 menu2 中
22        frm.setMenuBar(mb);                        // 設定 frm 的功能表為 mb
23        frm.setSize(200,150);
24        frm.setVisible(true);
25     }
26  }
```



功能表的事件處理 (1/2)

- 功能表
  - 觸發最簡單的事件 -- ActionEvent
  - 把要處理的事情撰寫在actionPerformed() method裡
- app19_6是處理功能表事件的範例


```
01  // app19_6, 功能表的事件處理範例
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app19_6 extends Frame implements ActionListener
05  {
06     static app19_6 frm=new app19_6();
07     static Label lab=new Label("Tastes differ.",Label.CENTER);
08     static MenuBar mb=new MenuBar();              // 建立 MenuBar 物件
09     static Menu menu1=new Menu("Style");
10     static Menu menu2=new Menu("Exit");
11     static MenuItem mi1=new MenuItem("Plane");
12     static MenuItem mi2=new MenuItem("Bold");
13     static MenuItem mi3=new MenuItem("Italic");
14     static MenuItem mi4=new MenuItem("Close window");
15     public static void main(String args[])
16     {
```

建立 Menu 物件

建立 MenuItem 物件

功能表的事件處理 (2/2)

```
17        frm.setTitle("Menu Demo");
18        mb.add(menu1);
19        mb.add(menu2);
20        menu1.add(mi1);
21        menu1.add(mi2);
22        menu1.add(mi3);
23        menu2.add(mi4);
24        mi1.addActionListener(frm);   // 設定 frm 為 mi1 的事件傾聽者
25        mi2.addActionListener(frm);   // 設定 frm 為 mi2 的事件傾聽者
26        mi3.addActionListener(frm);   // 設定 frm 為 mi3 的事件傾聽者
27        mi4.addActionListener(frm);   // 設定 frm 為 mi4 的事件傾聽者
28        lab.setFont(new Font("Dialog",Font.PLAIN,24));
29        frm.add(lab);
30        frm.setSize(280,150);
31        frm.setMenuBar(mb);
32        frm.setVisible(true);
33     }
34     public void actionPerformed(ActionEvent e) // 事件處理的程式碼
35     {
36        MenuItem mi=(MenuItem) e.getSource();     // 取得觸發事件的物件
37        if(mi==mi1)                               // mi1 觸發事件
38           lab.setFont(new Font("Dialog",Font.PLAIN,24));
39        else if(mi==mi2)                          // mi2 觸發事件
40           lab.setFont(new Font("Dialog",Font.BOLD,24));
41        else if(mi==mi3)                          // mi3 觸發事件
42           lab.setFont(new Font("Dialog",Font.ITALIC,24));
43        else if(mi==mi4)                          // mi4 觸發事件
44           frm.dispose();                         // 關閉視窗
45     }
46  }
```

將型態強制轉換成MenuItem型態

認識捲軸

- 捲軸包含
  - 兩個捲軸箭號（位於捲軸兩端）
  - 一個捲軸盒（用來拖曳捲軸）
  - 捲軸列（用來放置捲軸盒）
- Scrollbar類別
  - 處理捲軸相關的功能
  - 下表列出Scrollbar類別的建構元

表 19.4.1　java.awt.Scrollbar 的建構元與 method

建構元	主要功能
Scrollbar()	建立垂直方向的捲軸
Scrollbar(int orientation)	建立捲軸，並指定方向
Scrollbar(int orientation, int value, int visible, int minimum, int maximum)	建立捲軸，並指定方向、初始值、捲軸盒的可視大小、捲軸的最小與最大值

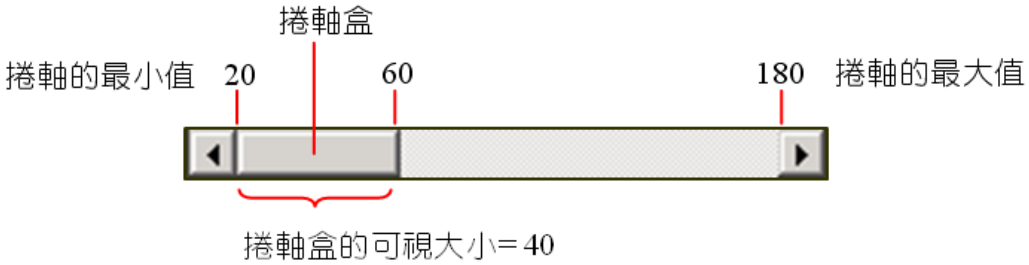
Scrollbar類別的method

- 下表列出Scrollbar類別的method：

method	主要功能
void addAdjustmentListener(AdjustmentListener l)	加入 AdjustmentEvent 事件傾聽者
int getMaximum()	取得捲軸的最大值
int getMinimum()	取得捲軸的最小值
int getOrientation()	取得捲軸的方向
int getValue()	取得目前捲軸的數值
int getVisibleAmount()	取得捲軸盒的可視大小
void setMaximum(int newMaximum)	設定捲軸的最大值
void setMinimum(int newMinimum)	設定捲軸的最小值
void setOrientation(int orientation)	設定捲軸的方向
void setValue(int newValue)	設定目前捲軸的數值
void setValues(int value, int visible, int minimum, int maximum)	設定捲軸的數值、捲軸盒的可視大小、捲軸的最小值與最大值
void setVisibleAmount(int newAmount)	設定捲軸盒的可視大小

捲軸的方向與大小

- 捲軸的方向
  - 水平
  - 垂直
- 捲軸盒的可視大小
  - 利用setVisibleAmount() 設定
  - 下圖說明捲軸盒的大小與傳回值的關係



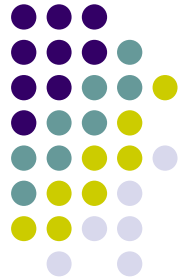
- Scrollbar類別使用的是AdjustmentListener介面，此介面只定義一個method：

```
void adjustmentValueChanged(adjustmentEvent e)
```

捲軸的實例

app19_7是捲軸的實例應用

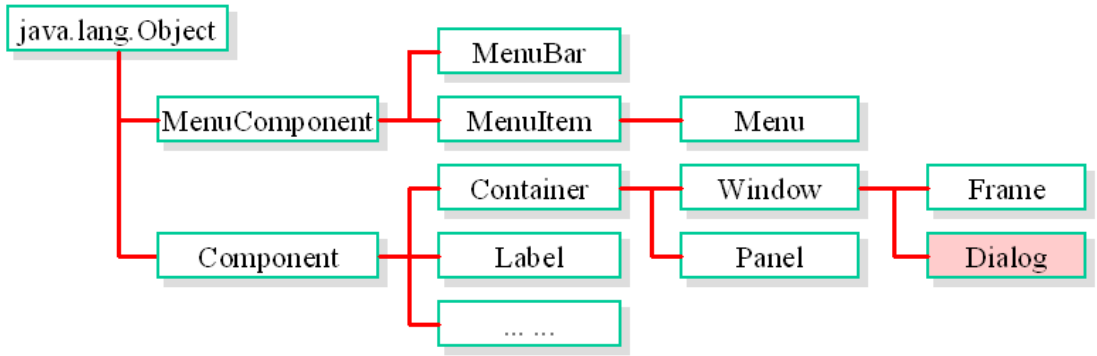
```
01  // app19_7, 捲軸的實例應用
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app19_7 extends Frame implements AdjustmentListener
05  {
06     static app19_7 frm=new app19_7();
07     static Scrollbar scr=new Scrollbar();  // 建立垂直捲軸 scr
08     static Label lab1=new Label("Silence is golden.",Label.CENTER);
09     static Label lab2=new Label("size=20",Label.CENTER);
10
11     public static void main(String args[])
12     {
13        BorderLayout br=new BorderLayout(5,5);
14        frm.setTitle("Scrollbar Demo");
15        frm.setSize(300,150);
16        scr.addAdjustmentListener(frm);   // 以 frm 當成 scr 的傾聽者
17        scr.setValues(20,4,12,40);        // 設定 scr 的相關數值
18        frm.add(scr,br.EAST);
19        frm.add(lab1,br.CENTER);
20        frm.add(lab2,br.SOUTH);
21        lab1.setFont(new Font("Dialog",Font.PLAIN,20));
22        lab2.setBackground(Color.orange);
23        frm.setVisible(true);
24     }
25     public void adjustmentValueChanged(AdjustmentEvent e)
26     {
27        int size=scr.getValue();          // 取得 scr 的數值
28        lab1.setFont(new Font("Dialog",Font.PLAIN,size));  // 設定字型樣式
29        lab2.setText("size="+size);       // 顯示字體大小
30     }
31  }
```



認識對話方塊

- 對話方塊
  - 以Dialog類別建立對話方塊
  - 通常利用對話方塊（dialog box）處理簡單的訊息，如下圖：

- 下面為Dialog類別的繼承圖：



Dialog類別的建構元

- Dialog類別可放置AWT的物件，如按鈕、選單
- 下表列出Dialog類別的建構元：

表 19.5.1　java.awt.Dialog 的建構元與 method

建構元	主要功能
Dialog(Dialog owner)	建立主控對話方塊，指定其擁有者為另一對話方塊
Dialog(Dialog owner, String title)	同上，並加上標題
Dialog(Dialog owner, String title, boolean modal)	建立對話方塊，指定其擁有者為另一對話方塊，並可設定 modal 來指定是否要主控
Dialog(Frame owner)	建立主控對話方塊，指定其擁有者為一視窗
Dialog(Frame owner, boolean modal)	同上，但可設定 modal 來指定是否要主控
Dialog(Frame owner, String title)	建立主控對話方塊，指定其擁有者為一視窗，並可設定標題
Dialog(Frame owner, String title, boolean modal)	建立對話方塊，指定其擁有者為一視窗，設定標題，並可設定 modal 來指定是否要主控

Dialog類別的method

- 下表列出Dialog類別常用的method：

method	主要功能
void dispose()	銷毀對話方塊物件
String getTitle()	取得對話方塊的標題
void hide()	隱藏對話方塊
boolean isModal()	測試對話方塊是否為主控
boolean isResizable()	測試對話方塊是否可改變大小
void setModal(boolean b)	設定對話方塊為主控
void setResizable(boolean resizable)	設定對話方塊是否可以改變大小
void setTitle(String title)	設定對話方塊的標題
void show()	顯示對話方塊

使用對話方塊需注意的事項

- 對話方塊必須指定其擁有者（owner）
- 擁有者
  - 可以是一個視窗
  - 也可以是另一個對話方塊
- 有兩點必須注意：

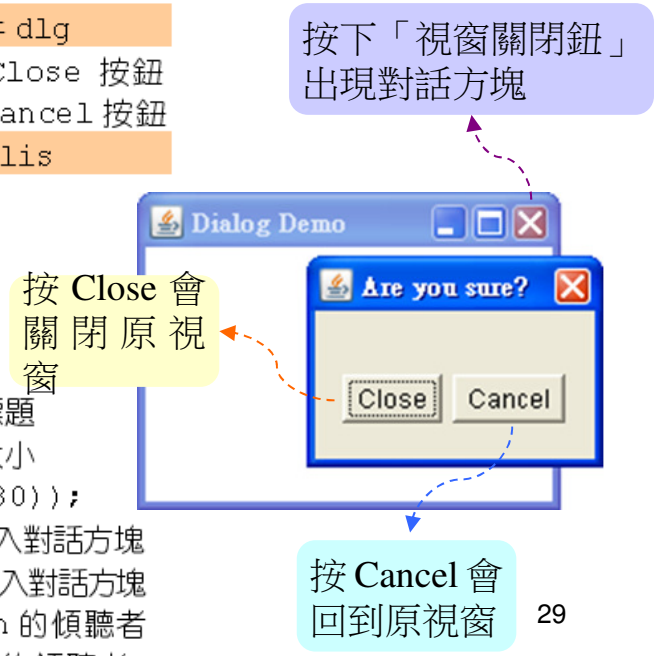
(1) 對話方塊有「主控」與「非主控」之分，所謂的主控對話方塊（modal dialog box），是指使用者一定要處理完主控對話方塊的事情之後，才能回到它的擁有者視窗裡繼續執行，但非主控對話方塊則無此限制。

(2) 對話方塊並不會自動顯示，而是要呼叫 setVisible(true) method 才能顯示它。如果要隱藏已顯示的對話方塊，可用 setVisible(false) method。

使用對話方塊 (1/2)

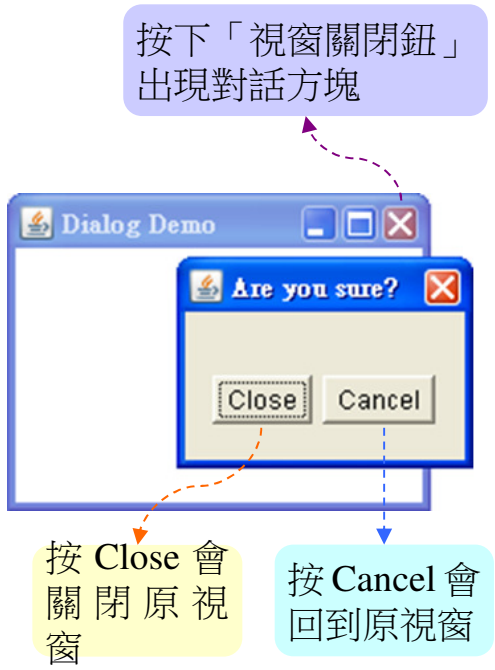
- app19_8是對話方塊的使用範例

```
01  // app19_8, 對話方塊的實例應用
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app19_8 extends Frame implements ActionListener
05  {
06     static app19_8 frm=new app19_8();
07     static Dialog dlg=new Dialog(frm);   // 建立 Dialog 物件 dlg
08     static Button Close_btn=new Button("Close");     // Close 按鈕
09     static Button Cancel_btn=new Button("Cancel");   // Cancel 按鈕
10     static WinLis wlis=new WinLis();     // 建立傾聽者物件 wlis
11
12     public static void main(String args[])
13     {
14        frm.setTitle("Dialog Demo");
15        frm.setSize(200,150);
16        dlg.setTitle("Are you sure?");    // 設定對話方塊的標題
17        dlg.setSize(140,100);             // 設定對話方塊的大小
18        dlg.setLayout(new FlowLayout(FlowLayout.CENTER,5,30));
19        dlg.add(Close_btn);               // 將 Close_btn 加入對話方塊
20        dlg.add(Cancel_btn);              // 將Cancel_btn 加入對話方塊
21        Cancel_btn.addActionListener(frm); // 設定 Cancel_btn 的傾聽者
22        Close_btn.addActionListener(frm); // 設定 Close_btn 的傾聽者
```



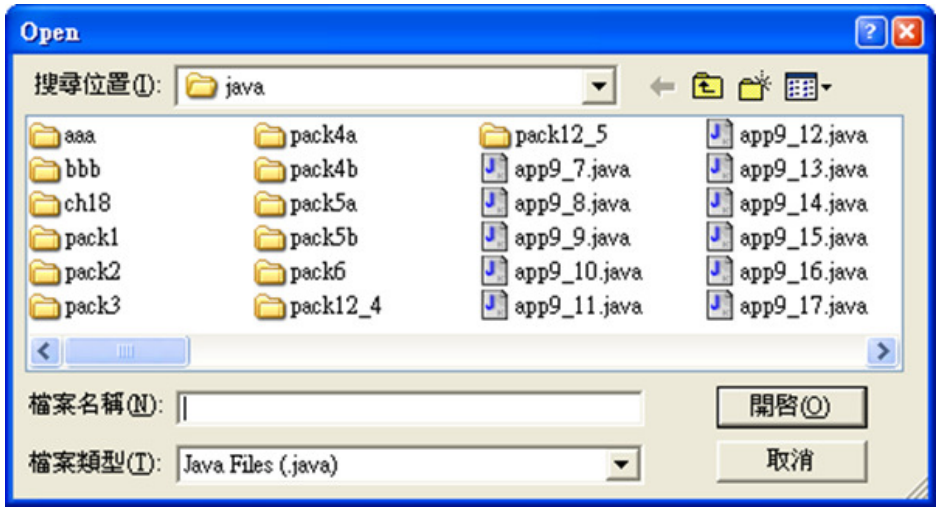
使用對話方塊 (2/2)

```
23          frm.addWindowListener(wlis);        // 設定 frm 的傾聽者
24          frm.setVisible(true);
25       }
26
27       static class WinLis extends WindowAdapter
28       {
29          public void windowClosing(WindowEvent e)   // 按下視窗關閉鈕時
30          {
31             dlg.setLocation(80,30);          // 設定對話方塊的位置
32             dlg.show();                      // 顯示對話方塊
33          }
34       }
35
36       public void actionPerformed(ActionEvent e) // 按下對話方塊上的按鈕時
37       {
38          Button btn=(Button) e.getSource(); // 取得被按下的按鈕
39          if(btn==Close_btn)                 // 如果是 Close 按鈕被按下
40          {
41             dlg.dispose();                  // 關閉對話方塊
42             frm.dispose();                  // 關閉視窗
43          }
44          else if (btn==Cancel_btn)          // 如果是 Cancel 按鈕被按下
45             dlg.hide();                     // 隱藏對話方塊
46       }
47    }
```



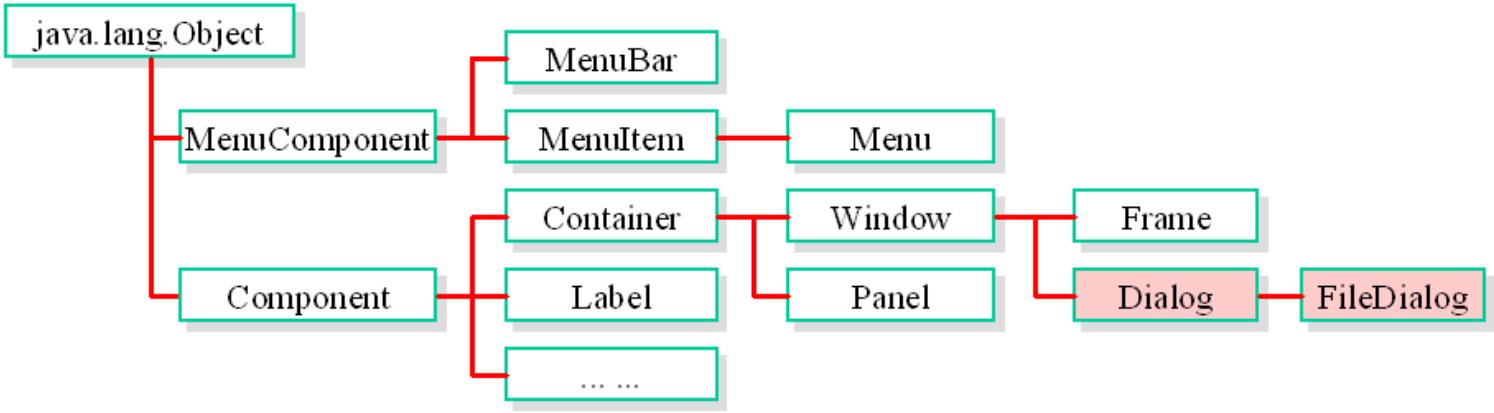
檔案對話方塊

- 檔案對話方塊
  - 以FileDialog類別來建立
  - 專門用來處理檔案存取等相關事務
  - 下圖為一典型的檔案對話方塊：



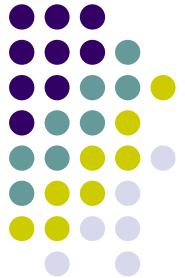
FileDialog類別的繼承關係

- FileDialog類別的繼承關係圖：



- 下表列出FileDialog的資料成員：

資料成員	主要功能
static int LOAD	檔案對話方塊為讀取檔案用
static int SAVE	檔案對話方塊為寫入檔案用



FileDialog類別的建構元與method

- 下表列出FileDialog常用的建構元與method：

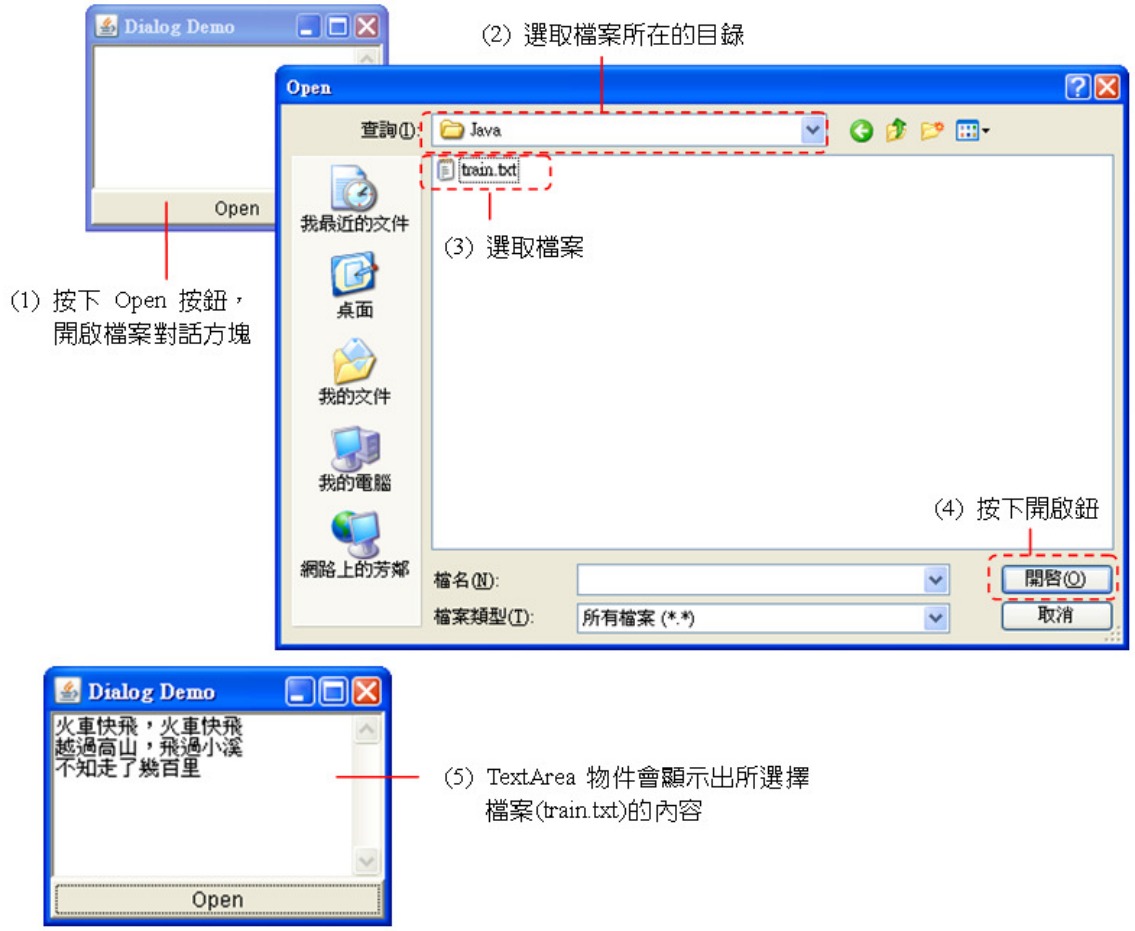
表 19.6.1　java.awt.FileDialog 的建構元與 method

建構元	主要功能
FileDialog(Frame parent)	建立讀取用的檔案對話方塊
FileDialog(Frame parent, String title)	建立讀取用的檔案對話方塊，並建立標題
FileDialog(Frame parent, String title, int mode)	建立檔案對話方塊，建立標題，並可指定是寫入或讀取用，若 mode 為 LOAD，則為讀取用，若為 SAVE，則為儲存用

method	主要功能
String getDirectory()	取得檔案對話方塊內所選取的目錄
String getFile()	取得檔案對話方塊內所選取的檔案名稱
int getMode()	取得檔案對話方塊的模式為 LOAD 或 SAVE
void setDirectory(String dir)	設定檔案對話方塊的預設開啟目錄
void setFile(String file)	設定檔案對話方塊的預設開啟檔案
void setMode(int mode)	設定檔案對話方塊的模式

檔案對話方塊的使用 (1/3)

- app19_9是使用FileDialog類別的範例，下圖為程式執行流程



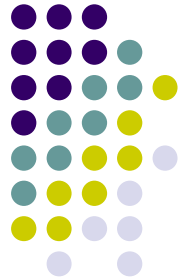
檔案對話方塊的使用 (2/3)

- app19_9的程式：

```
01  // app19_9, FileDialog 類別的使用
02  import java.awt.*;
03  import java.awt.event.*;
04  import java.io.*;
05  public class app19_9 extends Frame implements ActionListener
06  {
07     static app19_9 frm=new app19_9();
08     static FileDialog fdlg=new FileDialog(frm,"Open"); // 建立 fdlg 物件
09     static Button btn=new Button("Open");
10     static TextArea txa=new TextArea();
11
12     public static void main(String args[])
13     {
14        BorderLayout br=new BorderLayout();
15        frm.setTitle("Dialog Demo");
16        frm.setLayout(br);
17        frm.setSize(200,150);
18        frm.add(txa,br.CENTER);
19        frm.add(btn,br.SOUTH);
20        btn.addActionListener(frm);   // 設定 frm 為 btn 的事件傾聽者
21        frm.setVisible(true);
22     }
```

檔案對話方塊的使用 (3/3)

```
23      // 當 Open 按鈕按下時，執行下列的程式碼
24      public void actionPerformed(ActionEvent e)
25      {
26         fdlg.setVisible(true);          // 顯示檔案對話方塊
27         // 以下的程式碼是按下檔案對話方塊內的「開啟」鈕之後，才會執行
28         String fname=fdlg.getDirectory()+fdlg.getFile(); // 取得路徑與名稱
29         try
30         {
31            FileInputStream fi=new FileInputStream(fname);
32            byte ba[]=new byte[fi.available()];
33            fi.read(ba);                 // 讀入檔案內容到 byte 陣列裡
34            txa.setText(new String(ba)); // 將 byte 陣列的內容寫到 txa 物件裡
35            fi.close();
36         }
37         catch(IOException ioe){};
38      }
39   }
```



-The End-