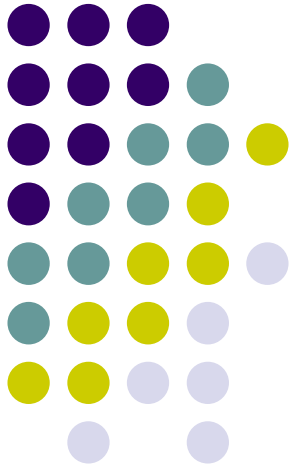
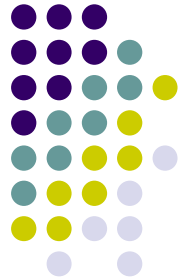
第十八章
事件處理

認識Java的委派事件模式

認識並學習使用各種事件處理類別

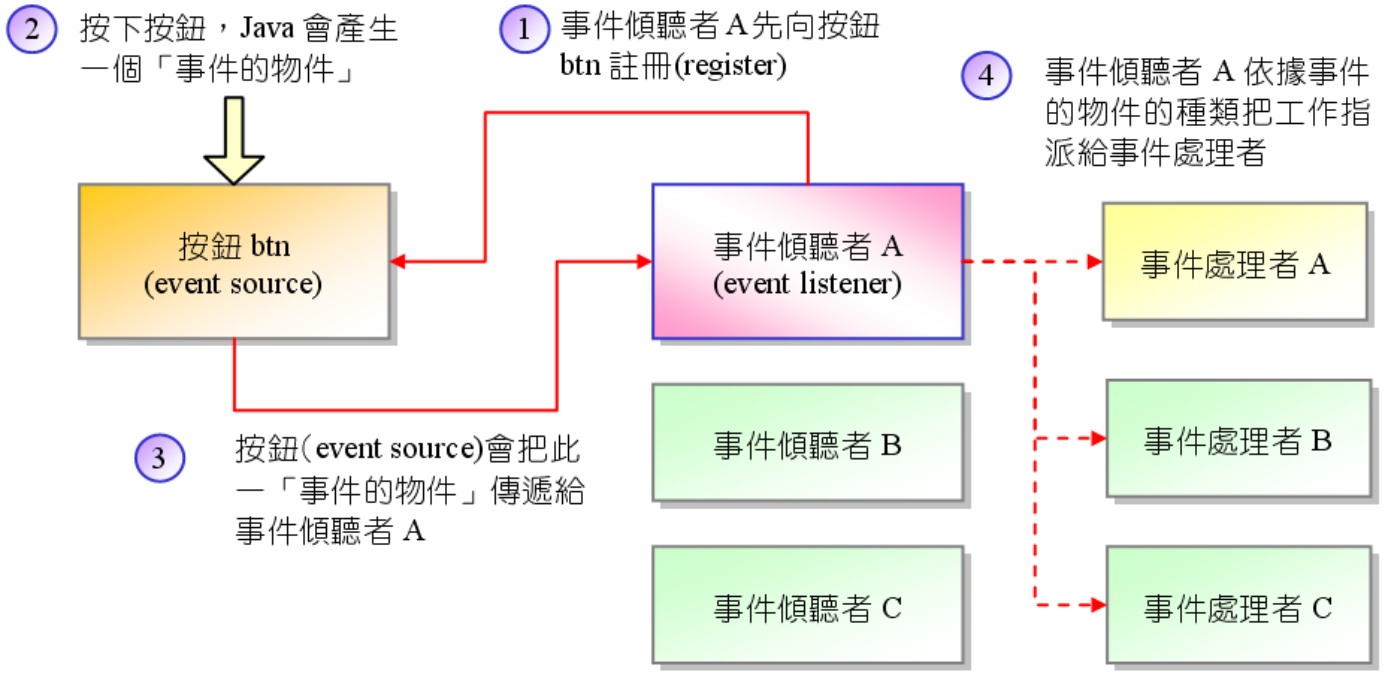
學習各種物件的事件處理





委派事件

- 指當事件發生時，產生事件的物件會把「訊息」轉給「事件傾聽者」（event listener）處理
- 下圖說明「委派事件模式」的運作流程：



簡單的範例

- 視窗內加入按鈕，尚未加入事件的處理之程式：

```
01  // app18_1, 簡單的事件處理範例(未加入事件處理)
02  import java.awt.*;
03  public class app18_1 extends Frame // 設定 app18_1 類別繼承自 Frame 類別
04  {
05     static app18_1 frm=new app18_1(); // 建立 app18_1 類別的物件 frm
06     static Button btn=new Button("Click Me");
07
08     public static void main(String args[])
09     {
10        frm.setLayout(new FlowLayout());
11        frm.setTitle("Action Event");
12        frm.setSize(200,150);
13        frm.add(btn);
14        frm.setVisible(true);
15     }
16  }
```

視窗內有一按鈕，當此按鈕按下時，視窗的顏色會有變化，下圖說明此範例的執行流程

傾聽者與註冊

- 傾聽者通常會讓包含「事件來源者」的物件來擔任
- 我們不能把app18_1的第5行撰寫成下面的敘述，又讓frm充當傾聽者：

```
static Frame frm=new Frame();                    // 產生 Frame 類別的物件 frm
```

  - 按鈕觸發事件由ActionListener介面傾聽，第3行修改成：

```
public class app18_1 extends Frame implements ActionListener
```

類別 app18_1 實作 ActionListener 介面

  - 決定事件來源者與傾聽者後，接著向事件來源者註冊：

```
btn.addActionListener(frm);
```

傾聽者

事件來源者

撰寫事件處理的程式碼

- 本例的事件處理是把視窗的底色改成黃色，因此可以撰寫出如下的程式：

```
public void actionPerformed(ActionEvent e)    // 事件發生的處理動作
{
   frm.setBackground(Color.yellow);           // 把視窗的底色改成黃色
}
```

  - actionPerformed() method會接收ActionEvent類別型態的物件，正是按鈕按下後傳過來的物件
  - 由於會用到ActionEvent類別，必須載入包含此類別的類別庫：

```
import java.awt.event.*;     // 載入 java.awt.event 類別庫裡的所有類別
```

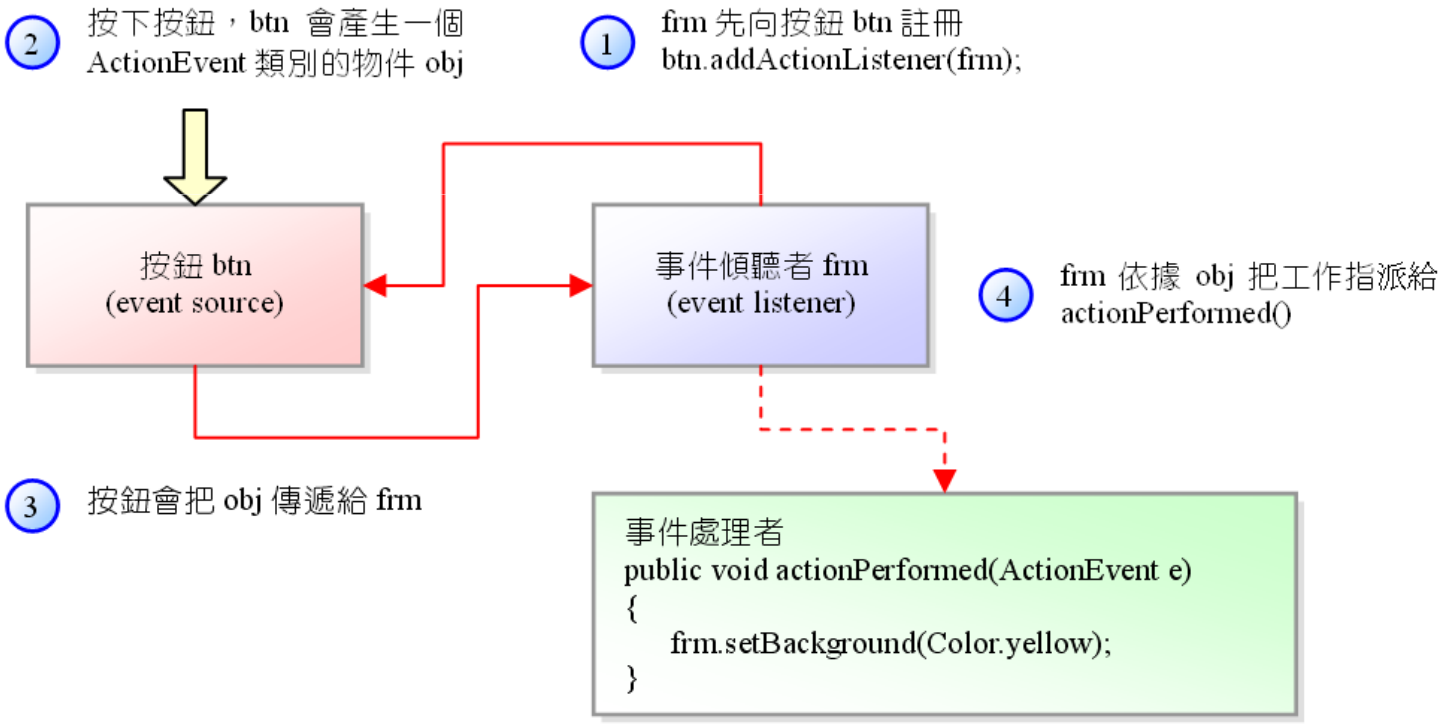
最後的完成工作 (1/2)

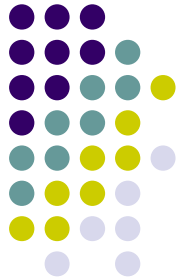
- 重新整理，可得到下面的程式碼：

```
01  // app18_2, 簡單的事件處理範例(已加入事件處理)
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_2 extends Frame implements ActionListener
05  {
06     static app18_2 frm=new app18_2();
07     static Button btn=new Button("Click Me");
08
09     public static void main(String args[])
10     {
11        btn.addActionListener(frm);              // 把 frm 向 btn 註冊
12        frm.setLayout(new FlowLayout());
13        frm.setTitle("Action Event");
14        frm.setSize(200,150);
15        frm.add(btn);
16        frm.setVisible(true);
17     }
18
19     public void actionPerformed(ActionEvent e) // 事件發生的處理動作
20     {
21        frm.setBackground(Color.yellow);
22     }
23  }
```

最後的完成工作 (2/2)

- app18_2的執行流程：



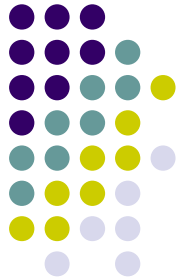


內部類別當成傾聽者

- 下面的程式是定義內部類別後，將它當成傾聽者：

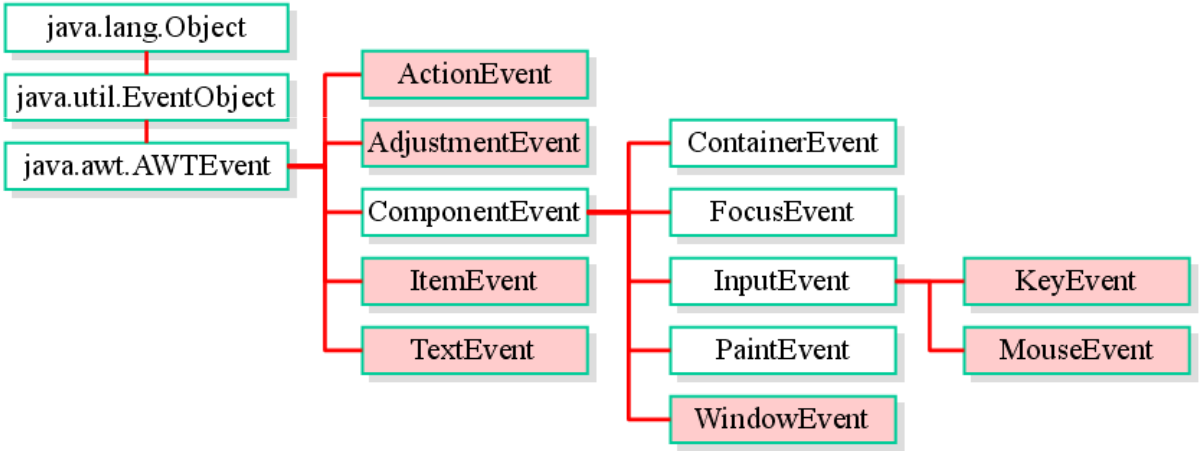
```
01  // app18_3, 定義內部類別當成傾聽者
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_3          // 主類別，注意此類別不需繼承 Frame 類別
05  {
06     static Frame frm=new Frame("Action Event");
07     static Button btn=new Button("Click Me");
08
09     public static void main(String args[])
10     {
11        btn.addActionListener(new ActLis());
12        frm.setLayout(new FlowLayout());
13        frm.setSize(200,150);
14        frm.add(btn);
15        frm.setVisible(true);
16     }
17     // 定義內部類別 ActLis，並實作 ActionListener 介面
18     static class ActLis implements ActionListener
19     {
20        public void actionPerformed(ActionEvent e)   // 事件發生的處理動作
21        {
22           frm.setBackground(Color.yellow);
23        }
24     }
25  }
```

宣告成static是因為在外部類別的「類別函數」內，不能建立內部類別物件



事件類別

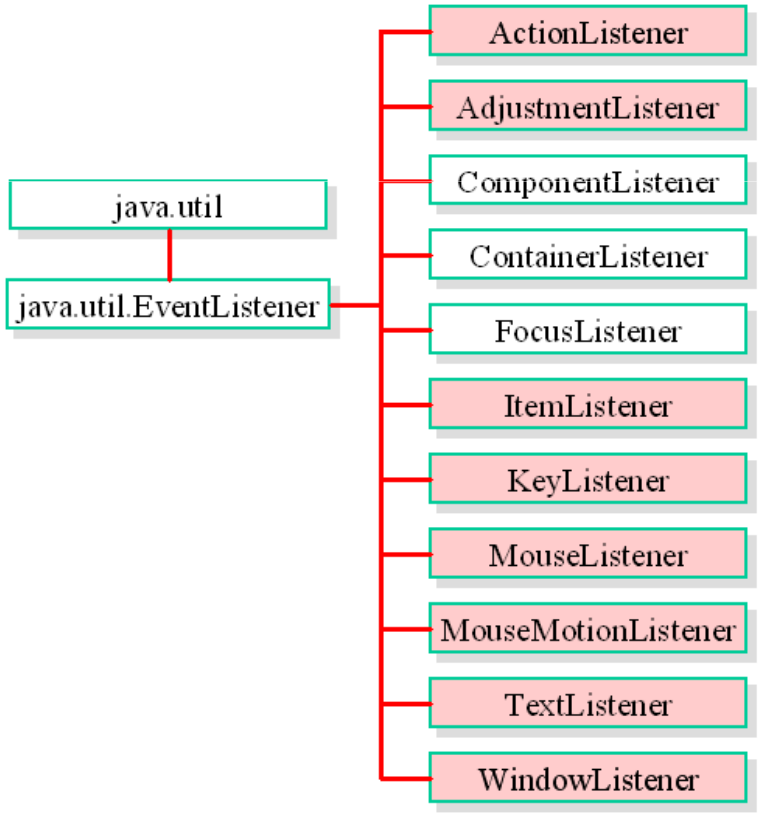
- AWTEvent類別是所有事件類別的最上層
  - 下圖為事件類別的繼承關係圖：



- 事件類別大致分為兩種：
  - 語意事件（semantic events）
  - 低階事件（low-level events）

事件傾聽者的繼承關係

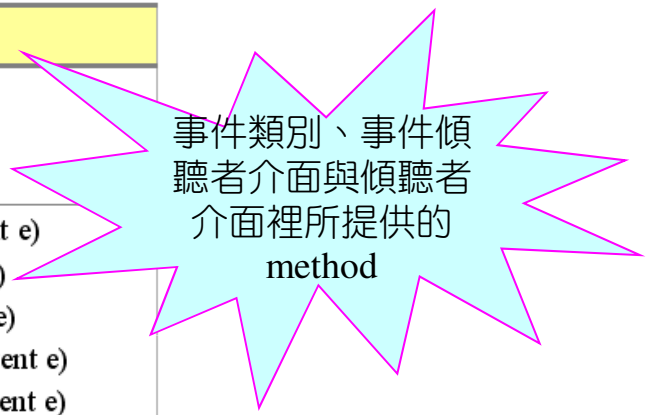
- 下圖顯示出事件傾聽者之間的繼承關係：



事件與傾聽者類別的method

表 18.2.1　事件類別、事件傾聽者介面與傾聽者介面裡所提供的 method

事件類別	傾聽者介面	傾聽者介面所提供的事件處理者
KeyEvent	KeyListener	void keyTyped(KeyEvent e) void keyPressed(KeyEvent e) void keyReleased(KeyEvent e)
WindowEvent	WindowListener	void windowActivated(WindowEvent e) void windowClosed(WindowEvent e) void windowClosing(WindowEvent e) void windowDeactivated(WindowEvent e) void windowDeiconified(WindowEvent e) void windowIconified(WindowEvent e) void windowOpened(WindowEvent e)
TextEvent	TextListener	void textValueChanged(TextEvent e)
ActionEvent	ActionListener	void actionPerformed(ActionEvent e)
AdjustmentEvent	AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent e)
ItemEvent	ItemListener	void itemStateChanged(ItemEvent e)
MouseEvent	MouseListener	void mouseClicked(MouseEvent e) void mouseEntered(MouseEvent e) void mouseExited(MouseEvent e) void mousePressed(MouseEvent e) void mouseReleased(MouseEvent e)
	MouseMotionListener	void mouseDragged(MouseEvent e) void mouseMoved(MouseEvent e)



可能觸發的事件整理

- 物件與可能觸發事件類別的對應關係，整理成下表：

表 18.2.2　AWT 的物件可能產生事件的對應關係表

事件來源者	產生事件的類別型態
Button	ActionEvent
CheckBox	ActionEvent、ItemEvent
Component	ComponentEvent、FocusEvent、KeyEvent、MouseEvent
MenuItem	ActionEvent
Scrollbar	AdjustmentEvent
TextArea	TextEvent
TextField	TextEvent
Window	WindowEvent

某些物件可能會觸發多個事件，只需針對所要的事件撰寫程式碼

觸發動作事件

- 觸發事件的物件把ActionEvent類別的物件傳送給向它註冊的傾聽者，請它負責處理
  - getSource() method可傳回事件來源物件
  - getModifiers() method可取得事件發生時所按下的按鍵
  - 按鍵被按下的常數定義在InputEvent類別，列表如下：

表 18.3.1　java.awt.event.InputEvent 的資料成員

資料成員	說 明
static int ALT_DOWN_MASK	用來表示鍵盤上的 ALT 按鍵被按下，其值為 512
static int CTRL_DOWN_MASK	用來表示鍵盤上的 CTRL 按鍵被按下，其值為 128
static int SHIFT_DOWN_MASK	用來表示鍵盤上的 SHIFT 按鍵被按下，其值為 64

加入關閉視窗的按鈕 (1/2)

- app18_4是ActionEvent類別的使用範例

```
01  // app18_4, 加入可關閉視窗的按鈕
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_4 extends Frame implements ActionListener
05  {
06     static app18_4 frm=new app18_4();
07     static Button btn1=new Button("Yellow");
08     static Button btn2=new Button("Green");
09     static Button btn3=new Button("Exit");
10
11     public static void main(String args[])
12     {
13        btn1.addActionListener(frm);     // 把事件傾聽者 frm 向 btn1 註冊
14        btn2.addActionListener(frm);     // 把事件傾聽者 frm 向 btn2 註冊
15        btn3.addActionListener(frm);     // 把事件傾聽者 frm 向 btn3 註冊
16
```

加入關閉視窗的按鈕 (2/2)


```
17          frm.setTitle("Action Event");
18          frm.setLayout(new FlowLayout(FlowLayout.CENTER));
19          frm.setSize(200,150);
20          frm.add(btn1);
21          frm.add(btn2);
22          frm.add(btn3);
23          frm.setVisible(true);
24      }
25
26      public void actionPerformed(ActionEvent e)
27      {
28          Button btn=(Button) e.getSource();      // 取得事件來源的物件
29          if(btn==btn1)                           // 如果是按下 btn1 按鈕
30              frm.setBackground(Color.yellow);
31          else if(btn==btn2)                      // 如果是按下 btn2 按鈕
32              frm.setBackground(Color.green);
33          else                                    // 如果是按下 btn3 按鈕
34              System.exit(0);
35      }
36  }
```

getSource() 可能會傳回其父類別的物件，因此必須先將它強制型態轉換成Button類別

exit(0) 代表正常結束，其它整數代表非正常結束

ItemEvent的成員

- 視窗中的選項物件被選取時，會觸發「選項事件」
- ItemEvent事件的傾聽者須實作ItemListener介面 ，該介面定義itemStateChanged() method：

```
public void itemStateChanged(ItemEvent e)
```

- ItemEvent類別提供一些成員列表如下：

表 18.4.1　ItemEvent 的資料成員與 method

資料成員	主要功能
static int DESELECTED	代表 Item 物件的狀態沒有被選取
static int SELECTED	代表 Item 物件的狀態被選取

method	主要功能
Object getItem()	取得觸發事件的 Item 物件
int getStateChange()	傳回 Item 物件改變的狀態 (DESELECTED 或 SELECTED)

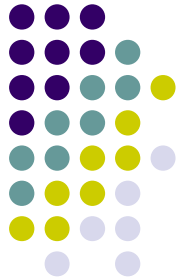
選項事件的範例 (1/2)

- app18_5的物件配置圖與程式碼如下所示：

```
01  // app18_5, ItemEvent 類別的使用範例
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_5 extends Frame implements ItemListener
05  {
06     static app18_5 frm=new app18_5();
07     static Checkbox ckb1=new Checkbox("Epson 5900L");
08     static Checkbox ckb2=new Checkbox("HP LaserJet 4p");
09     static Label lab=new Label(" Select a printer ");
10
11     public static void main(String args[])
12     {
13        CheckboxGroup grp=new CheckboxGroup();
14        frm.setSize(200,150);
15        frm.setTitle("Item Event");
16        frm.setLayout(new FlowLayout(FlowLayout.LEFT));
17        ckb1.setCheckboxGroup(grp);        // 將 ckb1 設為單選
18        ckb2.setCheckboxGroup(grp);        // 將 ckb2 設為單選
19        lab.setBackground(Color.orange);
```

選項事件的範例 (2/2)

```
20          ckb1.addItemListener(frm);          // 讓 frm 當成 ckb1 的傾聽者
21          ckb2.addItemListener(frm);          // 讓 frm 當成 ckb2 的傾聽者
22          frm.add(lab);
23          frm.add(ckb1);
24          frm.add(ckb2);
25          frm.setVisible(true);
26       }
27       // ItemEvent 事件發生時的處理動作
28       public void itemStateChanged(ItemEvent e)
29       {
30          if(ckb1.getState()==true)           // 如果是 ckb1 被選擇
31             lab.setText(" Epson 5900L");
32          else if(ckb2.getState()==true)      // 如果是 ckb2 被選擇
33             lab.setText(" HP LaserJet 4p");
34       }
35    }
```



文字事件

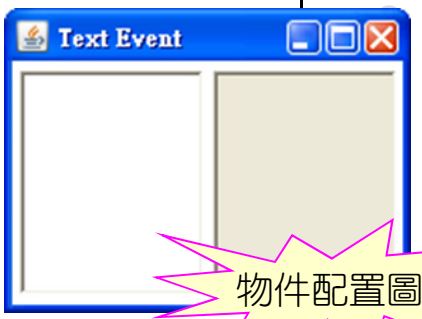
- 「文字事件」（text event）
  - 當TextField或TextArea物件裡的文字改變時所觸發的事件
  - TextEvent類別處理文字事件
  - TextListener為傾聽TextEvent事件的介面
    - TextListener介面定義textValueChanged() method：

```
public void textValueChanged(TextEvent e)
```

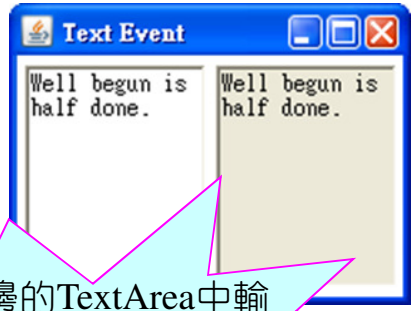
文字事件的範例

TextEvent類別的使用範例

```
01  // app18_6, TextEvent 類別的使用範例
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_6 extends Frame implements TextListener
05  {
06     static app18_6 frm=new app18_6();
07     static TextArea txa1;
08     static TextArea txa2;
09
10    public static void main(String args[])
11    {
12       txa1=new TextArea("",6,10,TextArea.SCROLLBARS_NONE);
13       txa2=new TextArea("",6,10,TextArea.SCROLLBARS_NONE);
14       frm.setSize(200,150);
15       frm.setTitle("Text Event");
16       frm.setLayout(new FlowLayout(FlowLayout.CENTER));
17       txa1.addTextListener(frm);          // 將 frm 設為 txa1 的傾聽者
18       txa2.setEditable(false);            // 將 txa2 設定為不可編輯
19       frm.add(txa1);
20       frm.add(txa2);
21       frm.setVisible(true);
22    }
23    // 當 txa1 物件裡的文字改變時，執行下列的程式碼
24    public void textValueChanged(TextEvent e)
25    {
26       txa2.setText(txa1.getText());
27    }
28  }
```

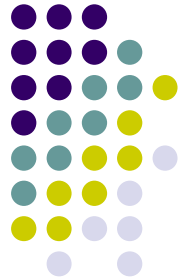


物件配置圖



在左邊的TextArea中輸入文字，右邊的TextArea會跟著顯示相同的文字

按鍵事件類別



- 按鍵事件類別（KeyEvent class）
  - 繼承自InputEvent類別
  - 屬於低階層的事件類別

- 下表列出KeyEvent類別常用的method：

表 18.6.1　KeyEvent 類別裡定義的 method

method	主要功能
char getKeyChar()	傳回按下的字元
int getKeyCode()	傳回字元碼
public boolean isActionKey()	判別所按下的按鍵是否為 Action Key，所謂的 Action Key 是指方向鍵、End、Insert、Home、PgDn 與 PgUp 與 F1~F12、Num Lock、Caps Lock 等按鍵

- KeyListener介面的事件處理method
  - keyPressed()
  - keyReleased()
  - keyTyped()

事件轉接器

- 「事件轉接器」（adapter）類別
  - 類別內定義的是 "空的" method
  - 只須針對相關的事件處理撰寫程式碼
- 處理KeyEvent事件的有2種
  - KeyListener介面
  - KeyAdapter類別

KeyListener介面

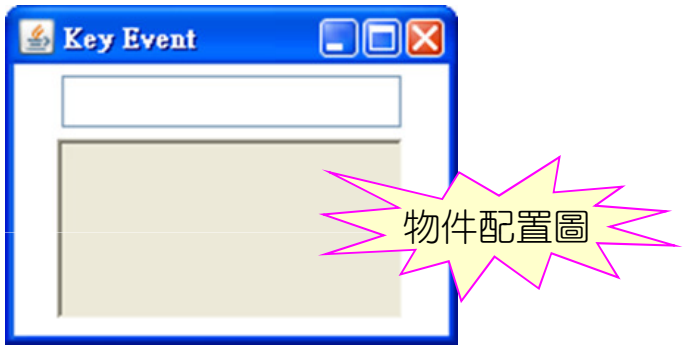
- 須以類別實作KeyListener介面
- KeyListener介面裡宣告三個method，列表如下：

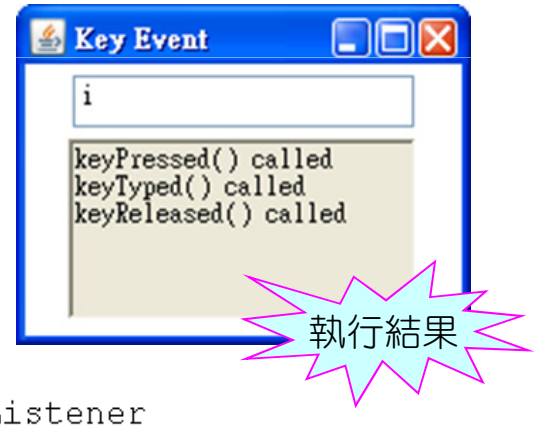
表 18.6.2　KeyListener 介面裡定義的 method

KeyListener 介面的 method	功能說明
void keyPressed(KeyEvent e)	按下按鍵事件
void keyReleased(KeyEvent e)	放開按鍵事件
void keyTyped(KeyEvent e)	字元輸入事件，即按下按鍵與放開按鍵這一整個事件，但不包括鍵入 Action Key

KeyEvent使用的範例 (1/2)

- app18_7的物件配置與程式碼如下所示：

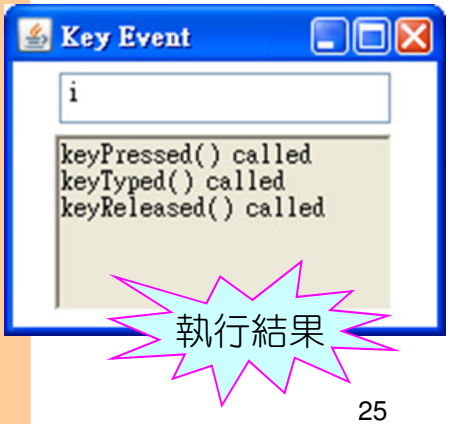





```
01  // app18_7, 以 KeyListener 介面處理 KeyEvent 事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_7 extends Frame implements KeyListener
05  {
06     static app18_7 frm=new app18_7();
07     static TextField txf=new TextField(18);
08     static TextArea txa=new TextArea("",4,19,TextArea.SCROLLBARS_NONE);
09
```

KeyEvent使用的範例 (2/2)

```
10      public static void main(String args[])
11      {
12         frm.setSize(200,150);
13         frm.setTitle("Key Event");
14         frm.setLayout(new FlowLayout(FlowLayout.CENTER));
15         txf.addKeyListener(frm);           // 將 frm 設為 txf 的傾聽者
16         txa.setEditable(false);            // 將 txa 設為不可編輯
17         frm.add(txf);
18         frm.add(txa);
19         frm.setVisible(true);
20      }
21      // 當 txf 物件觸發 KeyEvent 事件時，依事件種類執行下列的程式碼
22      public void keyPressed(KeyEvent e)   // 當按鍵按下時
23      {
24         txa.setText("");                   // 清空 txa 裡的文字
25         txa.append("keyPressed() called\n");
26      }
27      public void keyReleased(KeyEvent e) // 當按鍵放開時
28      {
29         txa.append("keyReleased() called\n");
30      }
31      public void keyTyped(KeyEvent e)     // 鍵入文字時
32      {
33         txa.append("keyTyped() called\n");
34      }
35   }
```



KeyAdapter類別事件

- KeyAdapter類別
  - 事實上是 "空的"，也就是沒有任何敘述的method實作KeyListener介面
  - 若只針對keyPressed() 事件撰寫程式碼，此時的keyPressed()即利用「改寫」（overriding）的技術，覆蓋定義於KeyAdapter裡空的keyPressed() method

KeyAdapter的範例

以 KeyAdapter 類別事件處理 KeyEvent 事件的範例

```
01  // app18_8, 以 KeyAdapter 類別事件處理 KeyEvent 事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_8 extends Frame   // 不需實作 KeyListener 介面
05  {
06     static app18_8 frm=new app18_8();
07     static TextField txf=new TextField(18);
08     static TextArea txa=new TextArea("",4,19,TextArea.SCROLLBARS_NONE);
09     public static void main(String args[])
10     {
11        frm.setSize(200,150);
12        frm.setTitle("Key Event");
13        frm.setLayout(new FlowLayout(FlowLayout.CENTER));
14        txf.addKeyListener(new KeyLis());
15        txa.setEditable(false);
16        frm.add(txf);
17        frm.add(txa);
18        frm.setVisible(true);
19     }
20     // 定義 KeyLis 為 static 類別，並繼承自 KeyAdapter 類別
21     static class KeyLis extends KeyAdapter
22     {
23        public void keyPressed(KeyEvent e)
24        {
25           txa.setText("");          // 清空 txa 裡的文字
26           if(e.isActionKey())       // 如果是 Action key
27              txa.append("Action key is pressed\n");
28           else                      // 如果不是 Action key，則印出字元
29              txa.append(e.getKeyChar()+" is pressed\n");
30        }
31     }
32  }
```

先產生KeyLis類別的物件，再把它當成是txf的事件傾聽者

必須宣告成static才能被main() method存取

取得鍵入的字元

某些按鍵如 Shift、Ctrl 與 Alt 等，並沒有被歸類為 Action key，但又不屬於可列印字元，諸如此類的按鍵會以「□」來顯示

滑鼠事件類別的method

- 滑鼠事件類別（MouseEvent class）
  - 繼承自InputEvent類別
- 觸發滑鼠事件的動作
  - 滑鼠的按鈕按下
  - 滑鼠指標進入或移出事件來源物件
  - 移動、拖曳滑鼠等
- 下表是MouseEvent類別裡常用的method：

表 18.7.1　MouseEvent 類別常用的 method

method	主要功能
Point getPoint()	取得滑鼠按鍵按下之點的坐標，並以 Point 類別型態的物件傳回
int getX()	取得滑鼠按鍵按下之點的 x 坐標
int getY()	取得滑鼠按鍵按下之點的 y 坐標

MouseListener介面

- 滑鼠事件的傾聽者
  - MouseListener介面
  - MouseMotionListener介面
- MouseListener介面主要傾聽下列5項事件：
  - 按一下滑鼠按鈕（包括滑鼠左鍵或右鍵）
  - 滑鼠的指標移到事件的來源物件（如按鈕）的上方
  - 滑鼠的指標從事件的來源物件（如按鈕）的上方移出
  - 按下滑鼠的任一個按鍵
  - 放開滑鼠被按下的按鍵

MouseListener介面的method

- MouseListener介面裡定義的method：

表 18.7.2　MouseListener 介面裡定義的 method

MouseListener 介面的 method	功能說明
void mouseClicked(MouseEvent e)	在事件來源物件的上方按一下滑鼠按鍵（此動作包括按下與放開兩個程序）
void mouseEntered(MouseEvent e)	滑鼠的指標進入事件的來源物件
void mouseExited(MouseEvent e)	滑鼠的指標移出事件的來源物件
void mousePressed(MouseEvent e)	按下滑鼠的任一個按鍵
void mouseReleased(MouseEvent e)	放開滑鼠被按下的按鍵

MouseMotionListener介面

- MouseMotionListener介面用來傾聽下列事件的發生：
  - 當滑鼠在事件的來源物件上方移動（move）時
  - 當滑鼠在事件的來源物件上方拖曳（drag）時
- MouseMotionListener介面裡定義的method：

表 18.7.3　MouseMotionListener 介面裡定義的 method

MouseMotionListener 介面的 method	功能說明
void mouseDragged(MouseEvent e)	當滑鼠在事件的來源物件上方拖曳
void mouseMoved(MouseEvent e)	當滑鼠在事件的來源物件上方移動

- 滑鼠事件的處理可利用
  - MouseAdapter類別
  - MouseMotionAdapter類別

使用MouseListener介面 (1/2)

- app18_9的物件配置與程式碼如下所示：

```
01  // app18_9, 以 MouseListener 介面處理 MouseEvent 事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_9 extends Frame implements MouseListener
05  {
06     static app18_9 frm=new app18_9();
07     static Button btn=new Button(" Click Me!! ");
08     static TextArea txa=new TextArea("",2,5,TextArea.SCROLLBARS_NONE);
09
10     public static void main(String args[])
11     {
12        BorderLayout br=new BorderLayout(2,5);
13        frm.setSize(200,150);
14        frm.setTitle("Mouse Event");
15        frm.setLayout(br);
16        btn.addMouseListener(frm);          // 設定 frm 為 btn 的傾聽者
17        txa.setEditable(false);
18        frm.add(btn,br.WEST);
19        frm.add(txa,br.CENTER);
20        frm.setVisible(true);
21     }
```

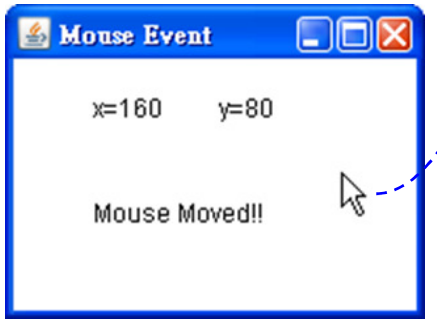
使用MouseListener介面 (2/2)

```
22
23      public void mouseEntered(MouseEvent e) // 滑鼠的指標進入 btn 上方
24      {
25         txa.setText("Mouse entered\n");
26      }
27      public void mouseClicked(MouseEvent e) // 按下並放開滑鼠按鈕
28      {
29         txa.append("Mouse clicked\n");
30      }
31      public void mouseExited(MouseEvent e)   // 滑鼠的指標移開 btn 上方
32      {
33         txa.append("Mouse exited\n");
34      }
35      public void mousePressed(MouseEvent e) // 按下滑鼠按鈕
36      {
37         txa.append("Mouse pressed\n");
38      }
39      public void mouseReleased(MouseEvent e) // 放開滑鼠按鈕
40      {
41         txa.append("Mouse released\n");
42      }
43   }
```

MouseMotionListener介面 (1/2)

- MouseMotionListener介面用來傾聽的事件
  - 滑鼠移動
  - 拖曳
- 利用addMouseMotionListener() method註冊事件傾聽者
- app18_10是MouseMotionListener介面使用的範例

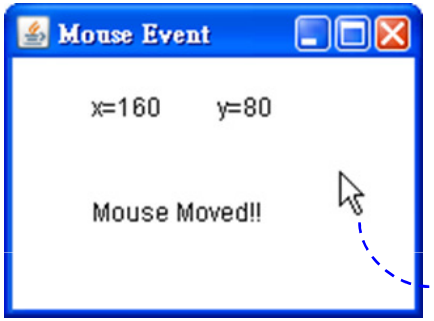
```
01  // app18_10, 以 MouseMotionListener 介面處理 MouseEvent 事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_10 extends Frame implements MouseMotionListener
05  {
06     static app18_10 frm=new app18_10();
07     static Label labx=new Label();
08     static Label laby=new Label();
09     static Label lab=new Label();
10     public static void main(String agrs[])
11     {
```



當滑鼠拖曳或移動時，此處便會顯示出滑鼠指標的座標

MouseMotionListener介面 (2/2)

```
12          frm.setLayout(null);
13          frm.addMouseMotionListener(frm);  // 設定 frm 為事件的傾聽者
14          labx.setBounds(40,40,40,20);
15          laby.setBounds(100,40,40,20);
16          lab.setBounds(40,80,100,40);
17          frm.setSize(200,150);
18          frm.setTitle("Mouse Event");
19          frm.add(labx);
20          frm.add(laby);
21          frm.add(lab);
22          frm.setVisible(true);
23      }
24      public void mouseMoved(MouseEvent e)// 當滑鼠移動時
25      {
26          labx.setText("x="+e.getX());       // 顯示 x 座標
27          laby.setText("y="+e.getY());       // 顯示 y 座標
28          lab.setText("Mouse Moved!!");      // 顯示"Mouse Moved!!"字串
29      }
30      public void mouseDragged(MouseEvent e)   // 當滑鼠拖曳時
31      {
32          labx.setText("x="+e.getX());       // 顯示 x 座標
33          laby.setText("y="+e.getY());       // 顯示 y 座標
34          lab.setText("Mouse Dragged!!");    // 顯示"Mouse Dragged!!"字串
35      }
36  }
```



當滑鼠拖曳或移動時，此處便會顯示出滑鼠指標的座標

MouseAdapter類別

以MouseAdapter類別事件處理MouseEvent事件的範例

```
01  // app18_11, 以 MouseAdapter 類別事件處理 MouseEvent 事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_11 extends Frame
05  {
06     static app18_11 frm=new app18_11();
07     static Button btn=new Button(" Click Me!! ");
08     static TextArea txa=new TextArea("",2,5,TextArea.SCROLLBARS_NONE);
09     static MouseLis mlis=new MouseLis();  // 建立 MouseLis 類別的物件
10     public static void main(String args[])
11     {
12        BorderLayout br=new BorderLayout(2,5);
13        frm.setSize(200,150);
14        frm.setTitle("Mouse Event");
15        frm.setLayout(br);
16        btn.addMouseListener(mlis);      // 以 mlis 物件做為 btn 的傾聽者
17        txa.setEditable(false);
18        frm.add(btn,br.WEST);
19        frm.add(txa,br.CENTER);
20        frm.setVisible(true);
21     }
22     // 定義 MouseLis 為 static 類別，並繼承自 MouseAdapter 類別
23     static class MouseLis extends MouseAdapter
24     {
25        public void mouseClicked(MouseEvent e)
26        {
27           int x=e.getX();                  // 取得 x 座標
28           int y=e.getY();                  // 取得 y 座標
29           txa.append("clicked at ("+x+","+y+")\n");
30        }
31     }
32  }
```

必須宣告成static才能被main() method存取

使用MouseMotionAdapter類別

- MouseMotionAdapter類別
  - 已實作MouseMotionListener介面
- 處理MouseEvent的事件的步驟
  - 以繼承MouseMotionAdapter的方式建立新類別
  - 再以此類別的物件當成傾聽者

WindowEvent類別的method

- 視窗事件類別（WindowEvent class）
  - 屬於低階的事件類別
- 觸發視窗事件的動作
  - 視窗的建立
  - 視窗縮小至工具列
  - 關閉視窗
- WindowEvent類別提供的method：

表 18.8.1　WindowEvent 的 method

method	主要功能
Window getWindow()	取得觸發事件的視窗
String paramString()	取得觸發事件之視窗的引數

WindowListener介面的method

- 下表列出WindowListener介面定義的method：

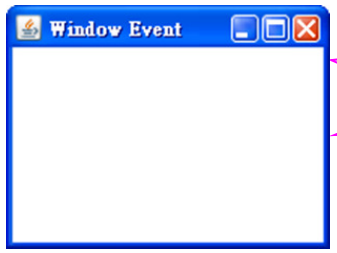
表 18.8.2　WindowListener 介面裡定義的 method

WindowListener 介面的 method	事件說明
void windowActivated(WindowEvent e)	視窗由「非作用中視窗」變成「作用中視窗」
void windowClosed(WindowEvent e)	視窗已被關閉
void windowClosing(WindowEvent e)	使用者企圖關閉視窗時。此事件是發生在按下視窗關閉鈕時，因此通常會利用這個時機讓使用者確定是否要關閉
void windowDeactivated(WindowEvent e)	視窗由「作用中視窗」變成「非作用中視窗」
void windowDeiconified(WindowEvent e)	視窗由最小化狀態變成一般狀態
void windowIconified(WindowEvent e)	視窗由一般狀態變成最小化狀態
void windowOpened(WindowEvent e)	視窗開啟時

使用WindowAdapter類別 (1/2)

- app18_12以WindowAdapter類別來處理視窗事件

```
01  // app18_12, WindowAdapter 類別來處理視窗事件
02  import java.awt.*;
03  import java.awt.event.*;
04  public class app18_12 extends Frame
05  {
06     static app18_12 frm=new app18_12();
07     static WinLis wlis=new WinLis();     // 產生 WinLis 類別的物件 wlis
08
09     public static void main(String args[])
10     {
11        frm.setSize(200,150);
12        frm.setTitle("Window Event");
13        frm.addWindowListener(wlis);      // 設定 wlis 為 frm 的事件傾聽者
14        frm.setVisible(true);
15     }
16     // 定義 WinLis 為 static，並繼承自 WindowAdapter 類別
17     static class WinLis extends WindowAdapter
18     {
19        public void windowClosing(WindowEvent e)   // 按下視窗關閉鈕
20        {
21           System.out.println("windowClosing() called");
22           System.out.println("Closing window...");
23           frm.dispose();                          // 關閉視窗並釋放資源
```



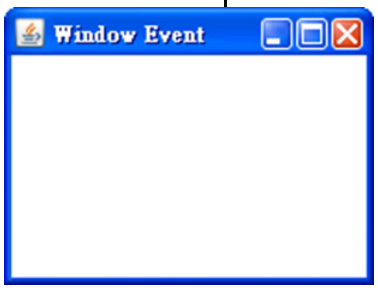
執行結果會因觸發視窗事件而定

```
/* app18_12 OUTPUT-------
windowActivated() called
windowOpened() called
windowIconified() called
windowDeactivated() called
windowActivated() called
windowDeiconified() called
windowActivated() called
windowClosing() called
Closing window...
window closed...
windowClosed() called
-----------------------*/
```

此處也可使用 `System.exit(0);` 以此方式關閉視窗，資源不會被釋放

使用WindowAdapter類別 (2/2)

```
24            System.out.println("window closed...");
25         }
26         public void windowClosed(WindowEvent e)    // 關閉視窗
27         {
28            System.out.println("windowClosed() called");
29         }
30         public void windowDeactivated(WindowEvent e)   // 變成非作用中視窗
31         {
32            System.out.println("windowDeactivated() called");
33         }
34         public void windowActivated(WindowEvent e)   // 變成作用中視窗
35         {
36            System.out.println("windowActivated() called");
37         }
38         public void windowDeiconified(WindowEvent e)   // 視窗還原
39         {
40            System.out.println("windowDeiconified() called");
41         }
42         public void windowIconified(WindowEvent e)     // 視窗最小化
43         {
44            System.out.println("windowIconified() called");
45         }
46         public void windowOpened(WindowEvent e)        // 開啟視窗
47         {
48            System.out.println("windowOpened() called");
49         }
50      }
51   }
```




```
/* app18_12 OUTPUT-------
windowActivated() called
windowOpened() called
windowIconified() called
windowDeactivated() called
windowActivated() called
windowDeiconified() called
windowActivated() called
windowClosing() called
Closing window...
window closed...
windowClosed() called
-----------------------*/
```

執行結果會因觸發視窗事件而定