Chapter 14. 자바 GUI와 이벤트 처리

학습목표

- 자바의 GUI를 구성하는 주요 클래스의 기능과 구조에 관해 학습합니다.
- GUI 구성을 위한 프레임, 배치관라자, 컨테이너에 관해 학습합니다.
- 이벤트의 개요와 이벤트 모델에 관해 학습합니다.
- 이벤트의 종류와 관련 클래스에 관해 학습합니다.
- 전형적인 이벤트 처리 절차와 방법에 관해 학습합니다.

목차

Section 1.
AWT와 스윙

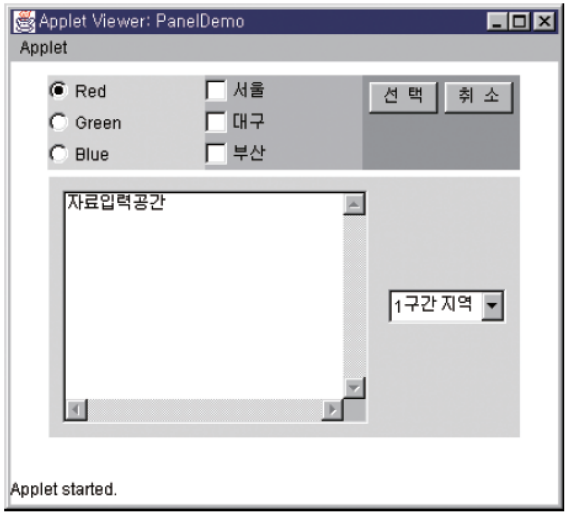
- **사용자 인터페이스(User Interface)**
  - 일반적인 응용 프로그램이나 웹 프로그램에서 사용자가 접속하는 부분

- **GUI(Graphical User Interface)**
  - 사용자 인터페이스에 다양한 그래픽을 적용한 것
  - "goo-ee"라고 부름

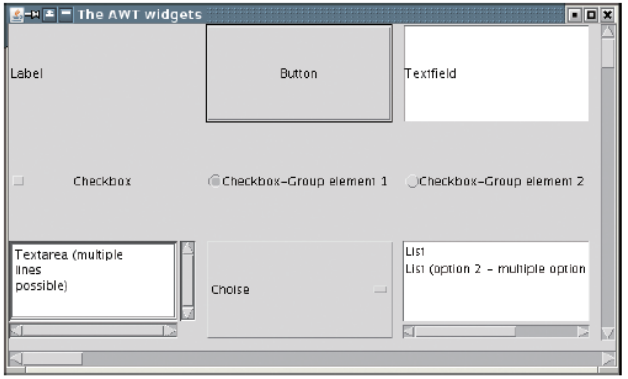
1-1 AWT(Abstract Window Toolkit)

AWT(Abstract Window Toolkit)

- 자바 언어는 초기에 GUI를 제공하기 위해 패키지를 제공



(a) 윈도 시스템에서의 AWT 컴포넌트



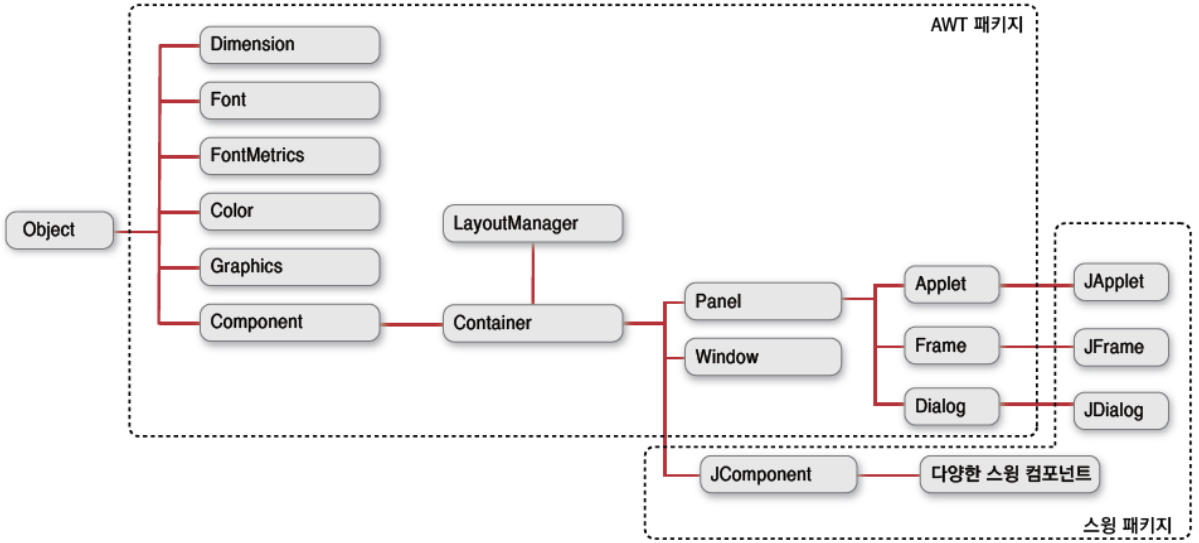
(b) 리눅스 시스템에서의 AWT 컴포넌트

- AWT 컴포넌트의 단점
  - 운영체제가 제공하는 그래픽 컴포넌트와 일대일로 연결되어 작동하여야 하는 부담과 실행되는 시스템에 따라 겉모양이 다르게 보인다

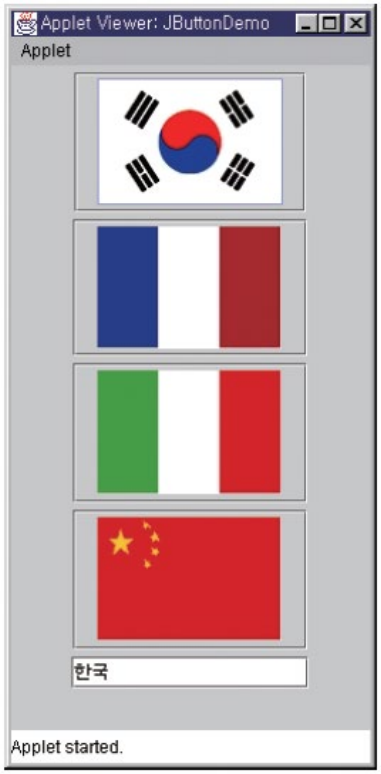
1-2 스윙(Swing)

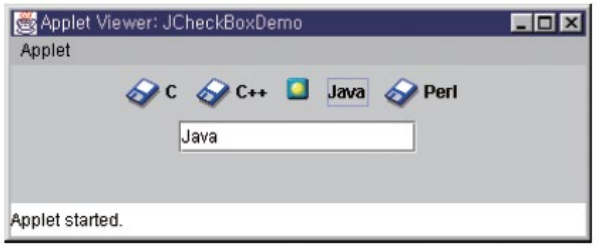
스윙(Swing)

- 선(sun)사가 넷스케이프사와 공동으로 개발한 새로운 그래픽 툴킷(toolkit)
- 자바의 표준 클래스인 JFC(Java Foundation Classes)로 제공되고 있으며 JDK에 포함
- 스윙은 AWT와 달리 그래픽을 JVM이 자체적으로 처리



1-2 스윙(Swing)


Applet Viewer: JButtonDemo
Applet
한국
Applet started.


Applet Viewer: JCheckBoxDemo
Applet
C
C++
Java
Perl
Java
Applet started.

Section 2.

AWT 패키지와 주요 클래스

2-1 java.awt 패키지

java.awt 패키지

- GUI 구축을 위한 컴포넌트 클래스뿐만 아니라 다른 클래스들도 많이 제공

유형 구분	클래스 명
컴포넌트의 배치와 관련된 클래스	BorderLayout, FlowLayout, GridLayout, GridBagLayout, GridBagConstraints, CardLayout
GUI 구성과 관련된 클래스	Button, Label, Canvas, Checkbox, Choice, Scrollbar, Component, List, Menu, TextComponent, TextArea, MenuBar, MenuItem, TextField, CheckboxGroup, CheckboxMenuItem, MenuComponent
그래픽 출력과 관련된 클래스	Color, Font, FontMetrics, Rectangle, Point, Polygon, Graphics, Image
컨테이너 클래스	Frame, Panel, Window, Container, FileDialog, Dialog
그 외의 클래스	Insets, Dimension, Toolkit, Event, MediaTracker

2-2 Component 클래스

Component 클래스

- 추상 클래스
- 그래피컬 사용자 인터페이스(GUI) 구축에 필요한 많은 하위 클래스를 가지고 있음
- 약 100여 개의 메소드가 제공

표 18-2 Component 클래스의 주요 메소드

메소드	설명
Image createImage(int width, int height)	width, height 크기의 Image 객체를 생성하여 반환
Font getFont()	현재 설정된 폰트를 폰트 객체로 반환
FontMetrics getFontMetrics(Font font)	font 폰트에 관한 FontMetrics 객체를 반환
Color getForeground()	전경색(foreground)을 Color 객체로 반환
Dimension getSize()	현재 컴포넌트의 크기를 Dimension 객체로 반환
void paint(Graphics g)	현재의 컴포넌트에 Graphics 객체 g를 이용하여 그림을 그리는 메소드
void repaint()	JVM에 update() 메소드 호출을 요청한다.
void setBackground(Color c)	배경색을 c로 설정
void setFont(Font f)	폰트를 f로 설정
void setForeground(Color c)	전경색을 c로 설정
void setSize(int x, int y)	컴포넌트의 크기를 x, y 값으로 설정한다. x, y 값은 픽셀 단위이다.
void setVisible(boolean b)	b가 true일 경우 컴포넌트를 화면에 나타낸다.
public void update(Graphics g)	컴포넌트를 배경색으로 채우고 paint() 메소드를 호출한다.

2-3 Container 클래스

Container 클래스

- Component 클래스의 하위 클래스로서 컴포넌트를 담는 쟁반 역할을 하는 클래스
- 사용되는 쟁반 역할을 하는 클래스는 JFrame, JPanel, Applet, JApplet 클래스

메소드	설명
Component add(Component c)	컴포넌트 c를 현재의 컨테이너에 추가하고 c를 반환
Component add(Component c, int position)	컴포넌트 c를 position이 지정하는 위치에 추가
void remove(Component c)	컴포넌트 c를 현재의 컨테이너에서 제거
void setLayout(LayoutManager lm)	lm으로 지정된 배치관리자를 현재 컨테이너의 배치관리자로 설정
void addContainerListener(ContainerListener cl)	이벤트를 받아들이기 위한 이벤트 리스너를 등록
void removeContainerListener(ContainerListener cl)	이벤트 리스너의 등록을 해제

Section 3.

프레임(Frame)

3 프레임(Frame)

- **응용 프로그램에서 GUI를 구축하기 위해서는 프레임 클래스를 이용**
  - GUI를 구축할 때는 먼저 프레임을 선택한 다음,
  - 프레임에 연관된 판, 즉 컨테이너를 생성하여 원하는 컴포넌트를 컨테이너에 추가한다.

- **JFrame 클래스**
  - Component → Container → Window → Frame → JFrame 형태로 상속된 클래스

[형식] JFrame 클래스의 생성자

```
JFrame()
JFrame(String title)
```

title : 프레임의 제목에 나타낼 문자열

3 프레임(Frame)

메소드	설명
public Container getContentPane()	현재의 프레임에 대한 Container 객체를 반환
void setDefaultCloseOperation (int operation)	프레임의 오른쪽 상단에 묵시적으로 "close" 버튼을 만든다. operation은 상수를 의미하며, 일반적으로 EXIT_ON_CLOSE를 지정하여 프레임을 종료시킨다.
void setJMenuBar (JMenuBar menubar)	프레임에 menubar를 설정한다.
void setTitle(String title)	프레임의 타이틀바에 title로 지정된 문자열을 나타낸다. 이 메소드는 java.awt 패키지인 Frame 클래스로부터 상속된 클래스이다.

3 프레임(Frame)

예제 14.1 FrameTest1.java와 FirstFrame2.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class FirstFrame1 {
05:   public FirstFrame1(){                                  ← 생성자
06:     JFrame jf = new JFrame("첫 번째 GUI 프로그램");      ← 프레임 객체 생성(제목 지정)
07:     Container ct = jf.getContentPane();                  ← 프레임으로부터 판(컨테이너) 생성
08:     JButton jb = new JButton("테스트버튼");              ← 버튼 컴포넌트 생성
09:     ct.add(jb);                                          ← 버튼을 판(컨테이너)에 추가
10:     jf.setSize(400,300);                                 ← 프레임 크기 설정
11:     jf.setVisible(true);                                 ← 프레임을 화면에 출력
12:   }
13: }
14: public class FrameTest1{
15:   public static void main(String[] args) {
16:     new FirstFrame1();
17:   }
18: }
```

실행 결과

첫 번째와 두 번째 프로그램의 실행결과는 프레임의 제목만 다르고 같다.

3 프레임(Frame)

예제 14.2 FrameTest2.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class FirstFrame3 extends JFrame  {   <-- JFrame을 상속하여 클래스 생성
05:   public FirstFrame3(){
06:     Container ct = getContentPane();   <-- 컨테이너 객체 생성
07:     JButton jb = new JButton("테스트버튼");
08:     ct.add(jb);
09:     setTitle("세 번째 GUI 프로그램");   <-- 프레임의 제목 설정
10:     setSize(400,300);
11:     setVisible(true);
12:   }
13: }
14: public class FrameTest2{
15:   public static void main(String[] args) {
16:     new FirstFrame3();
17:   }
18: }
```

실행 결과

Section 4.

배치관리자(Layout Manager)

4 배치관리자(Layout Manager)

- **컴포넌트들은 그릇 역할을 하는 컨테이너(또는 패널)에 담기게 되며 자바에서는 컨테이너에 컴포넌트를 배치하는 다양한 방법을 배치관리자 클래스로 제공**
  - FlowLayout, BorderLayout, GridLayout 클래스 제공
  - Container 클래스에서 배치관리자를 설정하는 setLayout() 메소드 제공(표18-3)

```
01: ........
02: class LayoutTest extends JFrame{ <-------- 프레임 상속 클래스 생성
03: public LayoutTest() {
04:   Container ct = getContentPane(); <-------- 프레임의 컨테이너(쟁반) 객체 생성
05:   ........
06:   ct.setLayout(...); <-------- 컨테이너에 배치관리자 설정
07:   ........
08:   ct.add(....); <--------
09:   ct.add(....); <-------- 설정된 배치관리자의 규칙에 따라 배치
10:   ........
11: ........
```

4-1 FlowLayout 클래스

- 컨테이너에 컴포넌트를 수평 방향으로 배치

[형식] FlowLayout 클래스의 생성자

```
FlowLayout()
FlowLayout(int align)
FlowLayout(int align, int hgap, int vgap)
```

align : 정렬방식을 지정하는 상수(LEFT, CENTER, RIGHT 사용 가능)

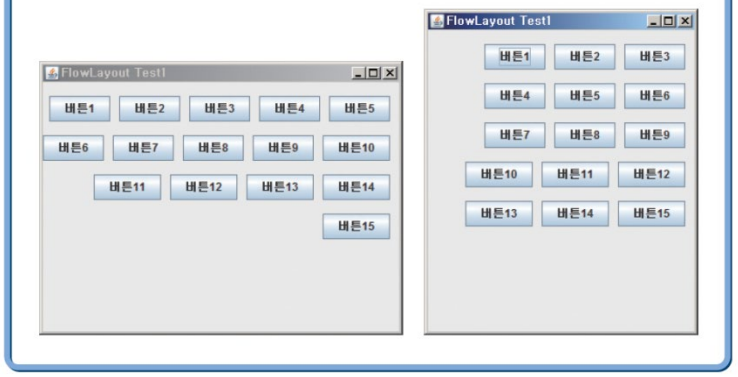
hgap, vgap : 컴포넌트 사이의 수직, 수평 간격을 지정하는 픽셀 값

4-1 FlowLayout 클래스

예제 14.3 FlowLayoutTest1.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class FlowLayout1 extends JFrame{
05:   public FlowLayout1(){
06:     Container ct = getContentPane();          ← 프레임으로부터 컨테이너 객체 생성
07:     FlowLayout fl = new FlowLayout(FlowLayout.RIGHT, 10, 15);   ← 배치관리자 객체 생성
08:     ct.setLayout(fl);                         ← 컨테이너에 배치관리자 설정
09:     for ( int i = 1; i<=15 ; i++)             ← 15개의 버튼 컴포넌트 생성하여 추가
10:           ct.add(new JButton("버튼"+i));
11:     setTitle("FlowLayout Test1");
12:     setSize(400,300);
13:     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14:     setVisible(true);
15:   }
16: }
17: public class FlowLayoutTest1 {
18:   public static void main(String[] args) {
19:     new FlowLayout1();
20:   }
21: }
```

실행 결과



4-2 BorderLayout 클래스

- 컨테이너에 컴포넌트를 추가할 때 방향을 지정하여 추가
  - 지정할 수 있는 방향 : "East", "West", "South", "North", "Center"

[형식] BorderLayout 클래스의 생성자

```
BorderLayout()
BorderLayout(int hgap, int vgap)
```

hgap, vgap : 컴포넌트 사이의 수직, 수평 간격을 지정하는 픽셀 값

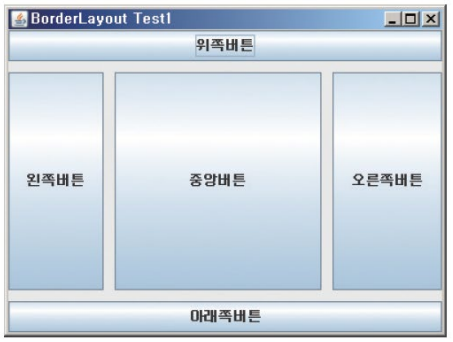
4-2 BorderLayout 클래스

예제 14.4 BorderLayoutTest1.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class BorderLayout1 extends JFrame{
05:   public BorderLayout1() {
06:     Container ct = getContentPane();
07:     BorderLayout bl = new BorderLayout(10, 10);   <-- BorderLayout 배치관리자 객체 생성
08:     ct.setLayout(bl);                             <-- 컨테이너에 배치관리자 설정
09:     ct.add(new JButton("오른쪽버튼"), BorderLayout.EAST);
10:     ct.add(new JButton("왼쪽버튼"), BorderLayout.WEST);
11:     ct.add(new JButton("위쪽버튼"), BorderLayout.NORTH);
12:     ct.add(new JButton("아래쪽버튼"), BorderLayout.SOUTH);
13:     ct.add(new JButton("중앙버튼"), BorderLayout.CENTER);
14:     setTitle("BorderLayout Test1");
15:     setSize(400,300);
16:     setVisible(true);
17:   }
18: }
19: public class BorderLayoutTest1 {
20:   public static void main(String[] args) {
21:     new BorderLayout1();
22:   }
23: }
```

09~13: 버튼 컴포넌트 생성하여 추가

실행 결과



4-3 GridLayout 클래스

- 컨테이너에 컴포넌트를 추가할 때 행과 열을 가진 배열 형태로 배치

[형식] GridLayout 클래스의 생성자

```
GridLayout()
GridLayout(int rows, int cols)
GridLayout(int rows, int cols, int hgap, int vgap)
```

rows, cols : 배치할 행과 열을 지정

hgap, vgap : 컴포넌트 사이의 수직, 수평 간격을 지정하는 픽셀 값

4-3 GridLayout 클래스

예제 14.5 GridLayoutTest1.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class GridLayout1 extends JFrame{
05:   public GridLayout1(){
06:     Container ct = getContentPane();
07:     GridLayout gl = new GridLayout(4,5,10,10);
08:     ct.setLayout(gl);
09:     for ( int i = 1; i<=20 ; i++)
10:             ct.add(new JButton("버튼"+i));
11:     setTitle("GridLayout Test1");
12:     setSize(800,800);
13:     setVisible(true);
14:   }
15: }
16: public class GridLayoutTest1 {
17:   public static void main(String[] args) {
18:     new GridLayout1();
19:   }
20: }
```

Grid 배치 관리자 객체 생성

컨테이너에 배치관리자 설정

20개의 버튼을 생성하여 추가

실행 결과

Section 5.

컨테이너(Container)

● 그릇 역할을 하는 클래스 : Container 클래스와 JPanel 클래스

- Container클래스는 JFrame 객체의 getContentPane() 메소드에 의해 생성하여 사용
- JPanel 클래스는 직접 생성하여 사용

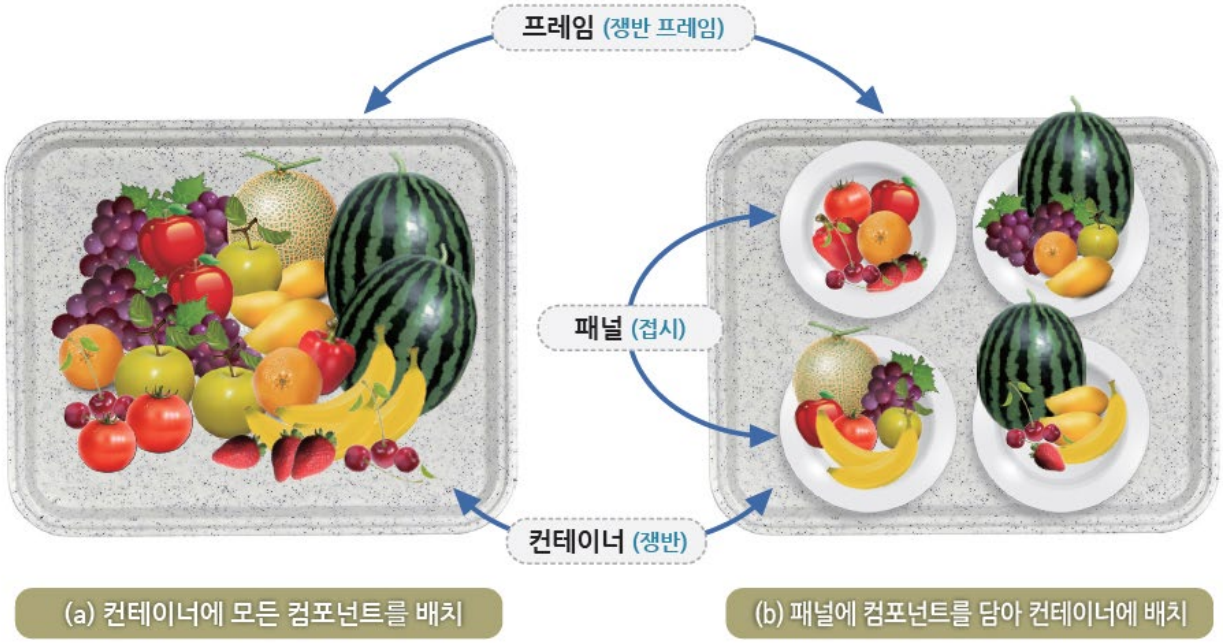


그림 18-4 프레임과 컨테이너, 패널의 관계

5 컨테이너(Container)

- 프레임으로부터 쟁반을 생성하여 컴포넌트 추가하는 예

```
01: ........
02: class ContainerTest extends JFrame {      <-- 프레임 상속 클래스 생성
03:   Container c = getContentPane();         <-- 프레임으로부터 컨테이너 객체 생성
04:   c.setLayout(...);                       <-- 배치관리자 설정
05:   c.add(...);                             <-- 컨테이너에 컴포넌트 추가
06:   c.add(...);                             <-- 컨테이너에 컴포넌트 추가
07: ........
```

5 컨테이너(Container)

- 접시(Jpanel)에 컴포넌트를 추가한 다음 접시를 쟁반에 배치하는 예

```
01: ........
02: class ContainerTest extends JFrame {      <-- 프레임 상속 클래스 생성
03:   Container c = getContentPane();         <-- 프레임으로부터 컨테이너 객체 생성
04:   c.setLayout(...);                       <-- 컨테이너 배치관리자 설정
05:   JPanel jp1 = new JPanel()               <-- 패널 객체 생성
06:   jp1.setLayout(...);                     <-- 패널 배치관리자 설정
07:   jp1.add(...);                           <--
08:   jp1.add(...);                           <-- 패널에 컴포넌트 추가
09:   ........
10:   JPanel jp2 = new JPanel()               <-- 패널 객체 생성
11:   jp2.setLayout(...);                     <-- 패널 배치관리자 설정
12:   jp2.add(...);                           <--
13:   jp2.add(...);                           <-- 컨테이너에 패널 추가
14:   ........
15:   c.add(jp1);
16:   c.add(jp2);
17: ........
```

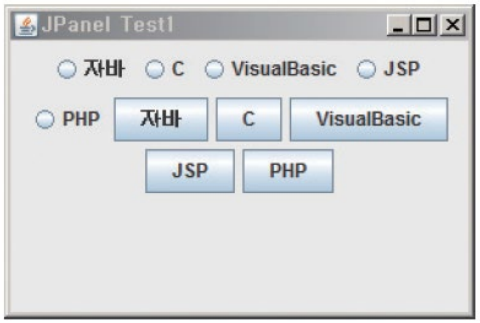
5 컨테이너(Container)

예제 1.4.6 JPanelTest1.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class JPanel1 extends JFrame{
05:   public JPanel1() {
06:     Container ct = getContentPane();
07:     ct.setLayout(new FlowLayout());
08:     ct.add(new JRadioButton("자바"));
09:     ct.add(new JRadioButton("C"));
10:     ct.add(new JRadioButton("VisualBasic"));
11:     ct.add(new JRadioButton("JSP"));
12:     ct.add(new JRadioButton("PHP"));
13:     ct.add(new JButton("자바"));
14:     ct.add(new JButton("C"));
15:     ct.add(new JButton("VisualBasic"));
16:     ct.add(new JButton("JSP"));
17:     ct.add(new JButton("PHP"));
18:     setTitle("JPanel Test1");
19:     setSize(300,200);
20:     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
21:     setVisible(true);
22:   }
23: }
24:  public class JPanelTest1 {
25:   public static void main(String[] args) {
26:     new JPanel1();
27:   }
28: }
```

배치관리자 설정

컨테이너에 컴포넌트 추가

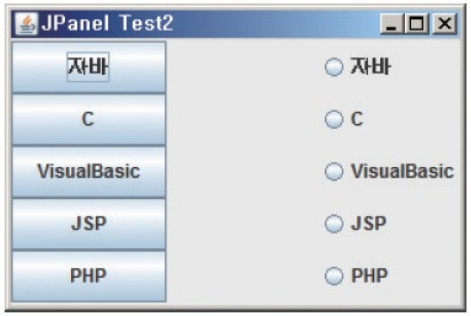


5 컨테이너(Container)

예제 14.7 JPanelTest2.java

```
01: import javax.swing.*;
02: import java.awt.*;
03:
04: class JPanel2 extends JFrame{
05:   public JPanel2() {
06:     Container ct = getContentPane();
07:     ct.setLayout(new BorderLayout(5,5));
08:     JPanel jp1 = new JPanel();
09:     jp1.setLayout(new GridLayout(5,1));
10:     jp1.add(new JRadioButton("자바"));
11:     jp1.add(new JRadioButton("C"));
12:     jp1.add(new JRadioButton("VisualBasic"));
13:     jp1.add(new JRadioButton("JSP"));
14:     jp1.add(new JRadioButton("PHP"));
15:     JPanel jp2 = new JPanel();
16:     jp2.setLayout(new GridLayout(5,1));
17:     jp2.add(new JButton("자바"));
18:     jp2.add(new JButton("C"));
19:     jp2.add(new JButton("VisualBasic"));
20:     jp2.add(new JButton("JSP"));
21:     jp2.add(new JButton("PHP"));
22:     ct.add(jp1, BorderLayout.EAST);
23:     ct.add(jp2, BorderLayout.WEST);
24:     setTitle("JPanel Test2");
25:     setSize(300,200);
26:     setVisible(true);
27:   }
28: }
29: public class JPanelTest2 {
30:   public static void main(String[] args) {
31:     new JPanel2();
32:   }
33: }
```

07: 컨테이너에 Border 배치 관리자 설정
08: 패널 객체 생성
09: 패널에 Grid 배치 관리자 설정
10~14: 패널에 컴포넌트 추가
15: 패널 객체 생성
16: 패널에 Grid 배치 관리자 설정
17~21: 패널에 컴포넌트 추가
22~23: 컨테이너에 패널 추가



Section 6.

이벤트 개요

- **이벤트(event)**
  - 사용자의 입력, 즉 키보드, 마우스 등의 장치로부터 발생하는 모든 사건을 의미

- **이벤트 지향(event-driven) 프로그램**
  - 프로그램은 무한 루프를 돌면서 사용자의 이벤트가 발생하기를 기다림
  - 사용자의 이벤트가 발생하면, 그 이벤트를 처리하고 다시 무한 루프에 빠지게 됨

● 위임형 이벤트 모델(delegation event model)

- 자바의 이벤트 처리 모델

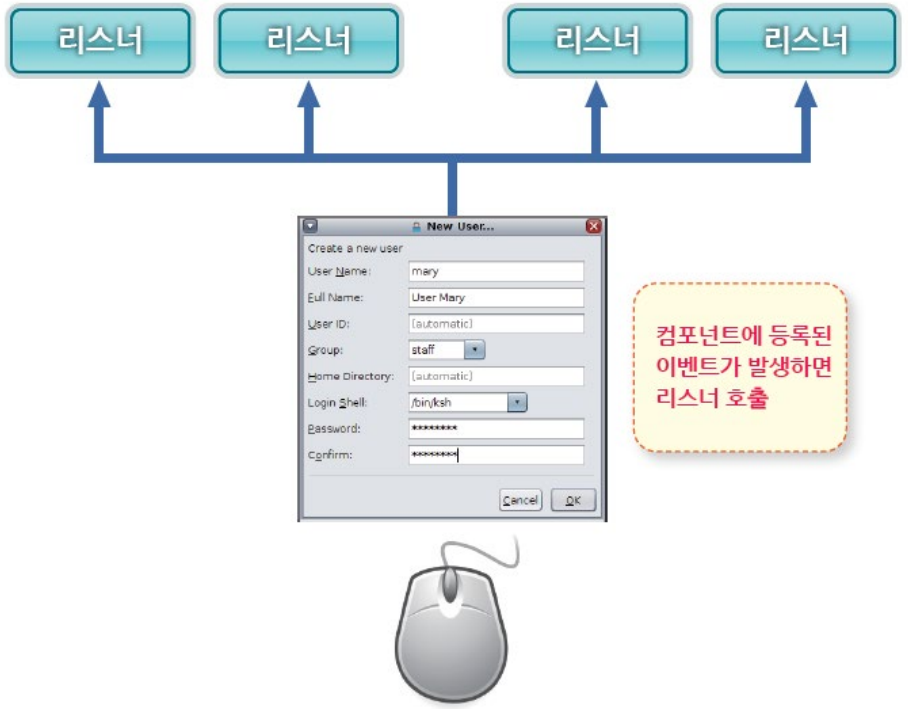


그림 18-5 위임형 이벤트 모델

Section 7.

이벤트 관련 컴포넌트

- 자바에서의 대부분의 이벤트는 GUI 관련 컴포넌트에서 발생
- 주요 이벤트를 발생시키는 컴포넌트들의 클래스 계층 구조

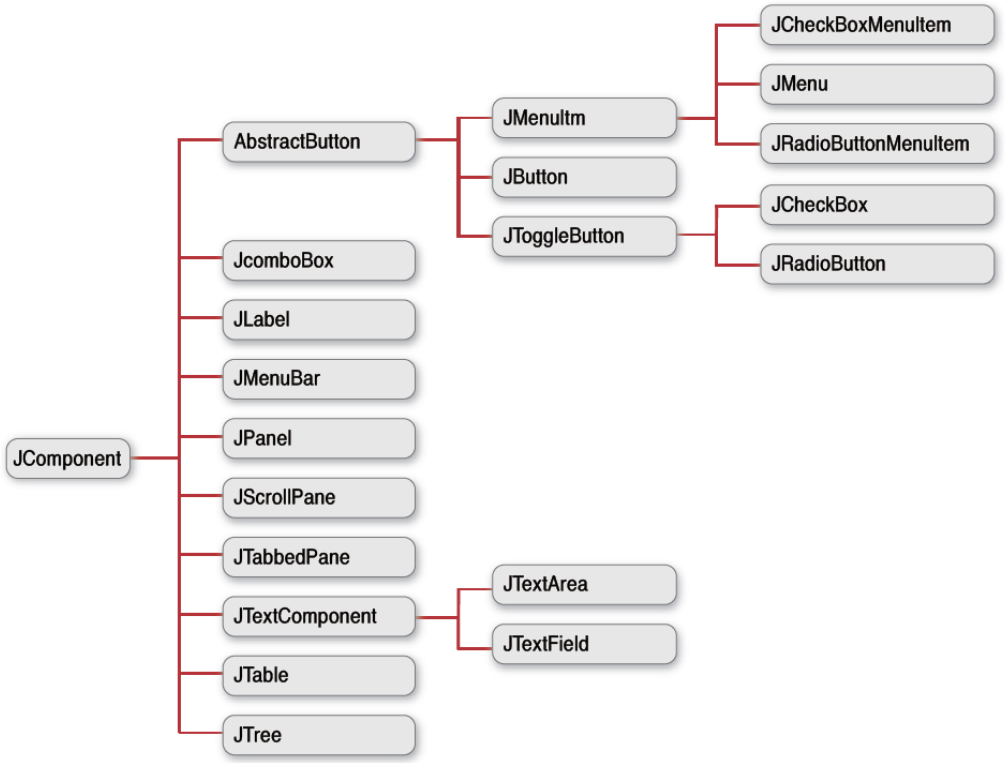


그림 18-6 이벤트를 발생시키는 스윙 컴포넌트들의 클래스 계층 구조

Section 8.

이벤트 종류

8 이벤트 종류

- 자바에서는 이벤트를 객체로 처리
- 각 컴포넌트들은 다양한 종류의 이벤트를 발생
- 이벤트 관련 클래스

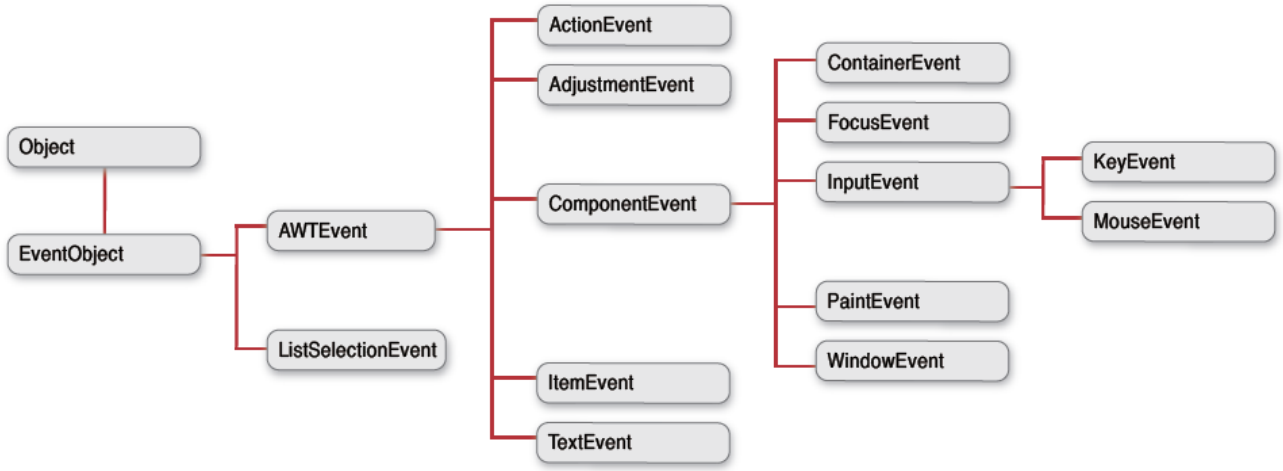


그림 18-7 이벤트 관련 클래스(java.awt.event 패키지)

- 자바에서는 이벤트와 연관된 클래스들을 java.awt.event 패키지로 제공

8 이벤트 종류

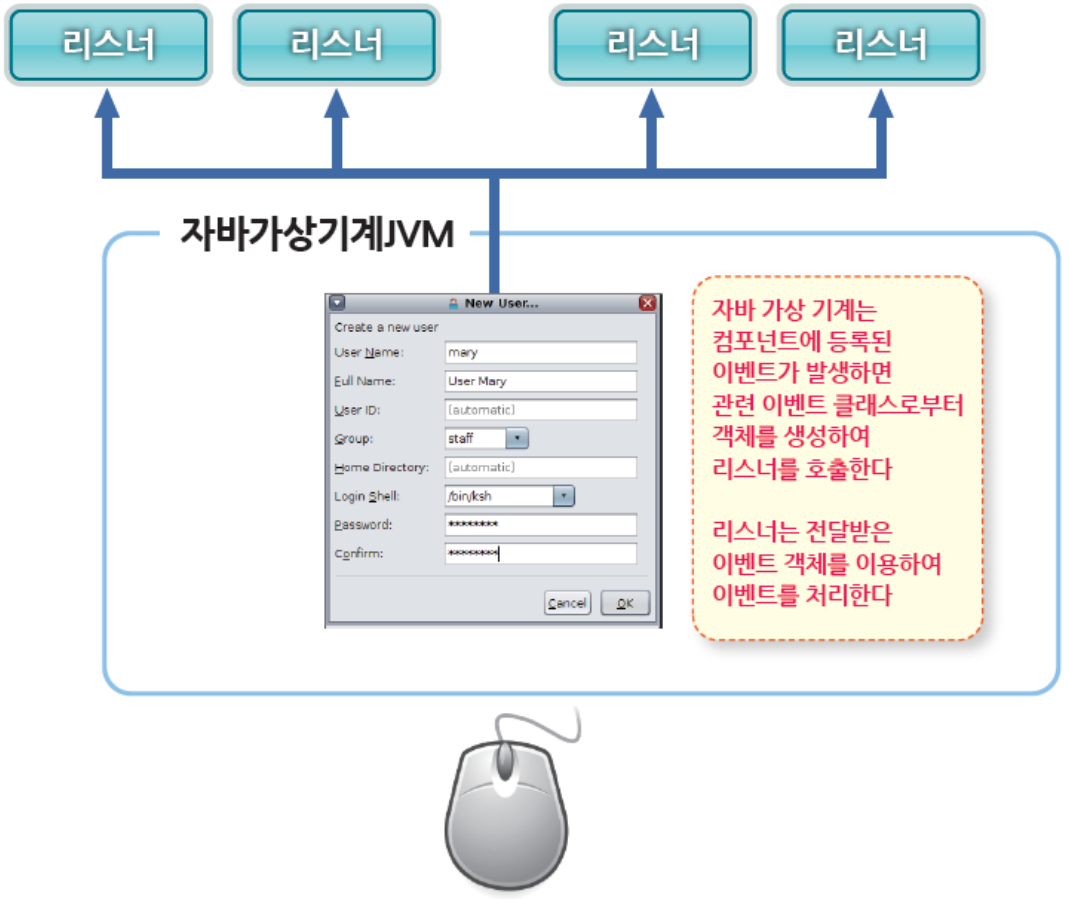


그림 18-8 자바 가상 기계의 이벤트 전달

8 이벤트 종류

컴포넌트	사용자의 작동	관련 이벤트
JButton	버튼을 누를 때	ActionEvent
JTextField	텍스트 필드에서 리턴키를 칠 때	ActionEvent
JComboBox	새로운 항목을 선택할 때	ItemEvent ActionEvent
JList	항목들을 선택할 때	ListSelectionEvent
JCheckBox	체크박스를 선택할 때	ItemEvent ActionEvent
JRadioButton	라디오 버튼을 선택할 때	ItemEvent ActionEvent
JMenuItem	메뉴 항목을 선택할 때	ActionEvent
JScrollbar	스크롤 바를 움직일 때	AdjustmentEvent
Window	윈도를 열거나 닫을 때, 아이콘화할 때, 아이콘화된 윈도를 다시 활성화할 때	WindowEvent
Container	컴포넌트가 컨테이너(패널, 프레임 등)에 추가되거나 삭제될 때	ContainerEvent
Component	컴포넌트를 움직이거나, 크기를 조정할 때, 다른 윈도 뒤로 감춰지거나, 다시 나타날 때	ComponentEvent
Component	컴포넌트가 마우스의 초점(focus)을 가지거나, 벗어날 때	FocusEvent
Component	Key가 눌러지거나 띄어질 때	KeyEvent
Component	마우스 버튼이 눌러지고, 띄어지고 마우스 초점이 컴포넌트 영역에 들어가고 나올 때	MouseEvent
Component	마우스가 이동(move)하고 눌러진 상태에서 끌 때(dragged)	MouseEvent

8-1 EventObject 클래스와 AWTEvent 클래스

● EventObject 클래스

- 대부분의 주요 이벤트 클래스들이 EventObject 클래스로부터 상속

[형식] EventObject 클래스의 생성자

```
EventObject(Object src)
```

src : 이벤트를 생성하는 객체

표 18-6 EventObject 클래스의 주요 메소드

메소드	설명
Object getSource( )	이벤트를 발생시킨 객체를 반환
String toString( )	이벤트 이름을 문자열로 반환

8-1 EventObject 클래스와 AWTEvent 클래스

● AWTEvent 클래스

- EventObject 클래스로부터 상속된 클래스

표 18-7 AWTEvent 클래스의 메소드

메소드	설명
int getID()	이벤트의 형을 반환한다

8-2 ActionEvent 클래스

Action 이벤트

- 버튼이 눌려졌거나, 리스트 항목이 선택되었을 때 ,메뉴의 한 항목이 선택되었을 때 발생
- 가장 많이 사용되는 이벤트 클래스

[형식] ActionEvent 클래스 생성자

```
ActionEvent(Object src, int type, String cmd)
ActionEvent(Object src, int type, String cmd, int modifiers)
```

src : 이벤트를 발생한 객체
type : 이벤트의 타입
cmd : 이벤트를 발생시킨 컴포넌트의 레이블
modifiers : 이벤트가 발생할 때 같이 사용된 수정키를 의미하는 상수

8-2 ActionEvent 클래스

- **이벤트가 발생되었을 때 같이 사용된 수정자(modifier)키를 구분하기 위한 4개의 상수를 제공**
  - ALT_MASK : 수정자 키로 ALT 키를 사용
  - CTRL_MASK : 수정자 키로 CTRL 키를 사용
  - META_MASK : 수정자 키로 META 키를 사용
  - SHIFT_MASK : 수정자 키로 SHIFT 키를 사용

표 18-8 ActionEvent 클래스의 주요 메소드

메소드	설명
String getActionCommand( )	ActionEvent를 발생시킨 객체의 이름을 반환한다. 예를 들어 버튼이 눌러져서 이벤트가 발생하였다면, 이 메소드는 버튼의 레이블을 문자열로 반환한다.
int getModifiers( )	이벤트 발생 시 같이 사용된 수정자 키를(ALT, CTRL, META, SHIFT) 나타내는 상숫값을 반환한다.

8-3 AdjustmentEvent 클래스

Adjustment 이벤트

- 스크롤 바의 위치가 이동되면 발생

5가지 유형

- BLOCK_DECREMENT : 스크롤 바의 값을 감소시키는 경우
- BLOCK_INCREMENT : 스크롤 바의 값을 증가시키는 경우
- TRACK : 스크롤 바를 드래그(drag) 하는 경우
- UNIT_INCREMENT : 스크롤 바의 값을 상향 버튼을 사용하여 증가시키는 경우
- UNIT_DECREMENT : 스크롤 바의 값을 하향 버튼을 사용하여 감소시키는 경우

8-3 AdjustmentEvent 클래스

[형식] Adjustment 클래스 생성자

```
AdjustmentEvent(Adjustable src, int id, int adjustment_type, int data)
```

src : 이벤트를 발생시킨 객체
id : 이벤트의 유형
type : 스크롤 바가 움직인 유형을 의미하며 5가지 종류의 상수로 표현
data : 스크롤 바의 이동 값

표 18-9 AdjustmentEvent 클래스의 주요 메소드

메소드	설명
Adjustable getAdjustable( )	이벤트를 발생시킨 객체를 반환
int getAdjustmentType( )	이벤트의 유형을 반환
int getValue( )	스크롤의 이동 값을 반환

8-4 ItemEvent 클래스

- **ItemEvent**
  - 체크박스나 리스트 항목이 선택되었을 때 또는 메뉴의 한 항목이 선택되었거나 선택된 항목이 해제될 때 발생

- **이벤트의 유형을 구분하기 위한 두 개의 상수를 제공**
  - SELECTED : 한 항목이 선택되었을 때
  - DESELECTED : 선택된 항목이 해제되었을 때

8-4 ItemEvent 클래스

[형식] ItemEvent 클래스 생성자

```
ItemEvent(ItemSelectable src, int type, Object entry, int state)
```

src : 이벤트를 발생시킨 항목
type : 이벤트의 유형
entry : 이벤트 발생 시 전달하고자 하는 특수한 item 객체
state : item의 현재 상태

표 18-10 ItemEvent 클래스의 주요 메소드

메소드	설명
Object getItem( )	이벤트를 발생시킨 객체를 반환
ItemSelectable getItemSelectable( )	이벤트를 발생시킨 ItemSelectable 객체를 반환한다. 리스트나 선택 박스 등은 ItemSelectable 인터페이스를 이용하여 구현한다.
int getStateChange( )	이벤트의 발생으로 변환된 상태를 상수로 반환

8-5 KeyEvent 클래스

- **KeyEvent**
  - 키보드로부터 입력이 일어날 때 발생

- **이벤트의 유형을 구분하기 위해 3개의 상수를 사용**
  - KEY_PRESSED : 키가 눌러졌을 때
  - KEY_RELEASED : 키가 띄어졌을 때
  - KEY_TYPED : 키에 의해 문자가 생성되었을 때

8-5 KeyEvent 클래스

[형식] KeyEvent 클래스 생성자

```
KeyEvent(Component src, int type, long when, int modifiers, int code)
KeyEvent(Component src, int type, long when, int modifiers, int code, char ch)
```

src : 이벤트를 발생시킨 컴포넌트
type : 이벤트의 유형
modifiers : 이벤트 발생 시 같이 사용된 수정자 키
code : 함수키와 같은 특수한 키
ch : 입력된 문자

표 18-11 KeyEvent 클래스의 주요 메소드

메소드	설명
char getKeyChar()	입력된 문자 값을 반환한다

8-6 MouseEvent 클래스

- **MouseEvent**
  - 매우 다양한 형태로 발생
  - 사용자가 마우스를 이용하여 작동하는 대부분의 행동에 이벤트가 발생

- **이벤트의 유형을 구분하기 위해 7개의 상수를 사용**
  - MOUSE_CLICKED : 마우스 클릭
  - MOUSE_DRAGGED : 마우스 드래그
  - MOUSE_ENTERED : 마우스가 컴포넌트 영역에 진입
  - MOUSE_EXITED : 마우스가 컴포넌트 영역에서 나왔을 때
  - MOUSE_MOVED : 마우스가 이동할 때
  - MOUSE_PRESSED : 마우스가 눌러졌을 때
  - MOUSE_RELEASED : 마우스가 띄어졌을 때

8-6 MouseEvent 클래스

[형식] MouseEvent 클래스 생성자

MouseEvent(Component src, int type, long when, int modifiers, int x, int y, int clicks, boolean triggersPopup)

src : 이벤트를 발생시킨 컴포넌트
int : 이벤트의 유형
modifiers : 이벤트가 발생하였을 때 같이 사용된 수정자 키
x, y : 이벤트가 일어난 위치
clicks : 마우스가 눌러진 횟수
triggersPopup : true이면 이벤트가 팝업popup 메뉴에서 발생

표 18-12 MouseEvent 클래스의 주요 메소드

메소드	설명
int getX( )	이벤트가 발생한 위치의 x 값을 반환
int getY( )	이벤트가 발생한 위치의 y 값을 반환
Point getPoint()	이벤트가 발생한 위치를 Point 객체로 반환. Point 클래스는 java.awt 패키지에 제공되는 클래스로 이 클래스의 객체는 이벤트가 발생한 위치의 x, y 값을 가진다.
void translatePoint(int x, int y)	이벤트의 발생 위치를 x, y 값으로 변환
int getClickCount()	마우스가 눌러진 횟수를 반환

Section 9.

이벤트 처리 : 리스너 인터페이스

- **이벤트 리스너 인터페이스를 이용하여 이벤트를 처리하기 위한 순서**
  - 처리할 이벤트의 종류를 결정
  - 이벤트에 적합한 이벤트 리스너 인터페이스를 사용하여 클래스를 작성
  - 이벤트를 받아들일 각 컴포넌트에 리스너를 등록
  - 리스너 인터페이스에 선언된 메소드를 오버라이딩하여 이벤트 처리 루틴 작성

- **ava.awt.event 패키지**
  - 각각의 이벤트 클래스와 연관된 이벤트 리스너 인터페이스를 제공

● 이벤트 클래스와 인터페이스, 인터페이스에 선언된 메소드

이벤트 클래스	리스너 인터페이스	리스너 인터페이스에 선언된 메소드
ActionEvent	ActionListener	actionPerform(ActionEvent ae)
AdjustmentEvent	AdjustmentListener	adjustmentValueChanged(AdjustmentEvent ae)
ComponentEvent	ComponentListener	componentHidden(ComponentEvent ce) componentMoved(ComponentEvent ce) componentResized(ComponentEvent ce) componentShown(ComponentEvent ce)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent ce) componentRemoved(ContainerEvent ce)
FocusEvent	FocusListener	focusGained(FocusEvent fe) focusLost(FocusEvent fe)
ItemEvent	ItemListener	itemStateChanged(ItemEvent ie)

● 이벤트 클래스와 인터페이스, 인터페이스에 선언된 메소드

이벤트 클래스	리스너 인터페이스	리스너 인터페이스에 선언된 메소드
KeyEvent	KeyListener	keyPressed(KeyEvent ke) keyReleased(KeyEvent ke) keyTyped(KeyEvent ke)
ListSelectionEvent	ListSelectionListener	valueChange(ListSelectionEvent lse)
MouseEvent	MouseListener	mouseClicked(MouseEvent me) mouseEntered(MouseEvent me) mouseExited(MouseEvent me) mousePressed(MouseEvent me) mouseReleased(MouseEvent me)
MouseEvent	MouseMotionListener	mouseDragged(MouseEvent me) mouseMoved(MouseEvent me)
WindowEvent	WindowListener	windowActivated(WindowEvent we) windowClosed(WindowEvent we) windowClosing(WindowEvent we) windowDeactivated(WindowEvent we) windowDeiconified(WindowEvent we) windowIconified(WindowEvent we) windowOpened(WindowEvent we)

9 이벤트 처리 : 리스너 인터페이스(Listener Interface)

- 이벤트 리스너 등록

addActionListener(ActionListener l)

addChangeListener(ChangeListener l)

addItemListener(ItemListener l)

addFocusListener(FocusListener l)

addKeyListener(KeyListener l)

addMouseListener(MouseListener l)

addMouseMotionListener(MouseMotionListener l)

- 이벤트 리스너 제거

removeActionListener(ActionListener l)

removeChangeListener(ChangeListener l)

removeItemListener(ItemListener l)

removeFocusListener(FocusListener l)

removeKeyListener(KeyListener l)

removeMouseListener(MouseListener l)

removeMouseMotionListener(MouseMotionListener l)

```
01: ........
02: ct.setLayout(new FlowLayout());
03: JButton jb = new JButton("버튼");
04: jl = new JLabel("버튼을 누르세요");
05: jb.addActionListener(this); <---------- 버튼 객체에 ActionListener를 등록.
                                            매개 변수로 이벤트를 처리할 객체를 지정.
                                            자신의 클래스에서 처리하는 경우 this로 지정.
06: ........
```

● 자바에서는 다양한 방법으로 이벤트를 처리 가능

① GUI를 제공하는 클래스에서 직접이벤트를 처리하는 방법

② 이벤트 처리를 위한 클래스를 따로 만들어 사용하는 방법

③ 내부 클래스를 사용하여 이벤트를 처리하는 방법

④ 무명(anonymous) 클래스를 이용하는 방법

⑤ 람다식(java 1.8)을 이용하는 방법

- 이 책에서는 가장 많이 사용하는 세 가지 방법에 관해서만 기술

9-1 GUI 클래스에서 이벤트 처리

- **GUI를 제공하는 클래스 내에서 처리하는 방법**
  - ① 처리가 요구되는 이벤트를 결정
  - ② 관련된 리스너 인터페이스를 포함하여 클래스 작성
  - ③ 인터페이스에 선언된 메소드를 오버라이딩하여 이벳트 처리 루틴 작성

9-1 GUI 클래스에서 이벤트 처리

예제 18.8 EventTest1.java

```
01: import javax.swing.*;
02: import java.awt.*;
03: import java.awt.event.*;
04:
05: class Event1 extends JFrame implements ActionListener{
06:   JLabel jl;
07:   public Event1() {
08:     Container ct = getContentPane();
09:     ct.setLayout(new FlowLayout());
10:     JButton jb = new JButton("버튼");
11:     jl = new JLabel("버튼을 누르세요");
12:     jb.addActionListener(this);
13:     ct.add(jb);
14:     ct.add(jl);
15:     setTitle("Event Test1");
16:     setSize(200,100);
17:     setVisible(true);
18:   }
19:   public void actionPerformed(ActionEvent e) {
20:     jl.setText("잘 하셨어요");
21:   }
22: }
23: public class EventTest1 {
24:   public static void main(String[] args) {
25:     new Event1();
26:   }
27: }
```

버튼에 동작하는 이벤트를 처리하기 위해 ActionListener를 포함

JLabel 객체를 속성으로 등록

버튼 객체 생성

라벨 객체 생성

버튼 객체에 리스너를 등록
(this는 자체 처리를 의미)

이벤트 처리 메소드 작성(표18-13 참조)

라벨의 값을 새로운 값으로 설정

9-2 내부 클래스를 이용한 이벤트 처리

- **자바에서는 클래스 내부에 새로운 클래스를 포함하여 사용하는 것을 허용**
  - 이러한 클래스를 내부inner 클래스라고 한다
  - 내부 클래스는 다른 용도로도 사용할 수 있지만, 이벤트를 처리할 때 유용

```
01: class OutClass {
02:    ........
03:    class InnerClass {
04:       ........
05:    }
06:    ........
07: }
```

내부 클래스 선언

외부 클래스의 모든 요소 사용 가능

9-2 내부 클래스를 이용한 이벤트 처리

예제 18.9 EventTest2.java

```
01: import javax.swing.*;
02: import java.awt.*;
03: import java.awt.event.*;
04:
05: class Event2 extends JFrame {
06:   JLabel jl;
07:   JButton jb1, jb2;
08:   public Event2() {
09:     Container ct = getContentPane();
10:     ct.setLayout(new FlowLayout());
11:     jb1 = new JButton("사랑합니다");
12:     jb2 = new JButton("행복합니다");
13:     jl = new JLabel("버튼을 누르세요");
14:     jb1.addActionListener(new EventProcess());
15:     jb2.addActionListener(new EventProcess());
16:     ct.add(jb1);
17:     ct.add(jb2);
18:     ct.add(jl);
19:     setTitle("Event Test2");
20:     setSize(250,150);
21:     setVisible(true);
22:   }
23:   private class EventProcess implements ActionListener {
24:     public void actionPerformed(ActionEvent e) {
25:             jl.setText(e.getActionCommand());
26:     }
27:   }
28: }
29: public class EventTest2 {
30:   public static void main(String[] args) {
31:     new Event2();
32:   }
33: }
```

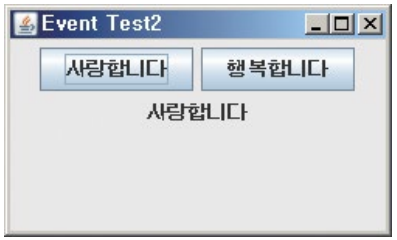
버튼과 라벨을 클래스의 속성으로 지정

두 개의 버튼에 리스너를 등록.
처리 객체로 내부 클래스 지정

내부 클래스로 인터페이스를 포함하는 클래스 작성

이벤트 처리 루틴 작성

외부 클래스의 속성인 라벨을 직접 사용.
ActionEvent 객체를 이용하여
버튼의 라벨 값을 추출



9-3 독립된 이벤트 처리 클래스를 이용한 이벤트 처리

- 이벤트 처리를 위한 클래스를 별로로 작성하여 이벤트를 처리하는 방법

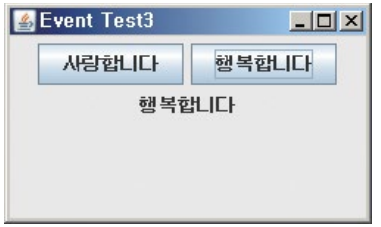
예제 18.10 EventTest3.java

```
01: import javax.swing.*;
02: import java.awt.*;
03: import java.awt.event.*;
04:
05: class EventClass implements ActionListener {   <-- 별도의 이벤트 처리 클래스 생성
06:   JLabel jl;
07:   public EventClass(JLabel jl){   <-- 생성자를 통해 JLabel 객체를 공유
08:     this.jl = jl;
09:   }
10:   public void actionPerformed(ActionEvent e) {
11:     jl.setText(e.getActionCommand());
12:   }
13: }
14: class Event3 extends JFrame {
15:   JLabel jl;
16:   JButton jb1, jb2;
17:   public Event3() {
18:     Container ct = getContentPane();
19:     ct.setLayout(new FlowLayout());
20:     jb1 = new JButton("사랑합니다");
```

9-3 독립된 이벤트 처리 클래스를 이용한 이벤트 처리

```
21:     jb2 = new JButton("행복합니다");
22:     jl = new JLabel("버튼을 누르세요");
23:     jb1.addActionListener(new EventClass(jl));
24:     jb2.addActionListener(new EventClass(jl));
25:     ct.add(jb1);
26:     ct.add(jb2);
27:     ct.add(jl);
28:     setTitle("Event Test3");
29:     setSize(250,150);
30:     setVisible(true);
31:   }
32:
33: }
34: public class EventTest3 {
35:   public static void main(String[] args) {
36:     new Event3();
37:   }
38: }
```

리스너를 등록.
처리 객체를 지정할 때 JLabel 객체 전달



Thank You!