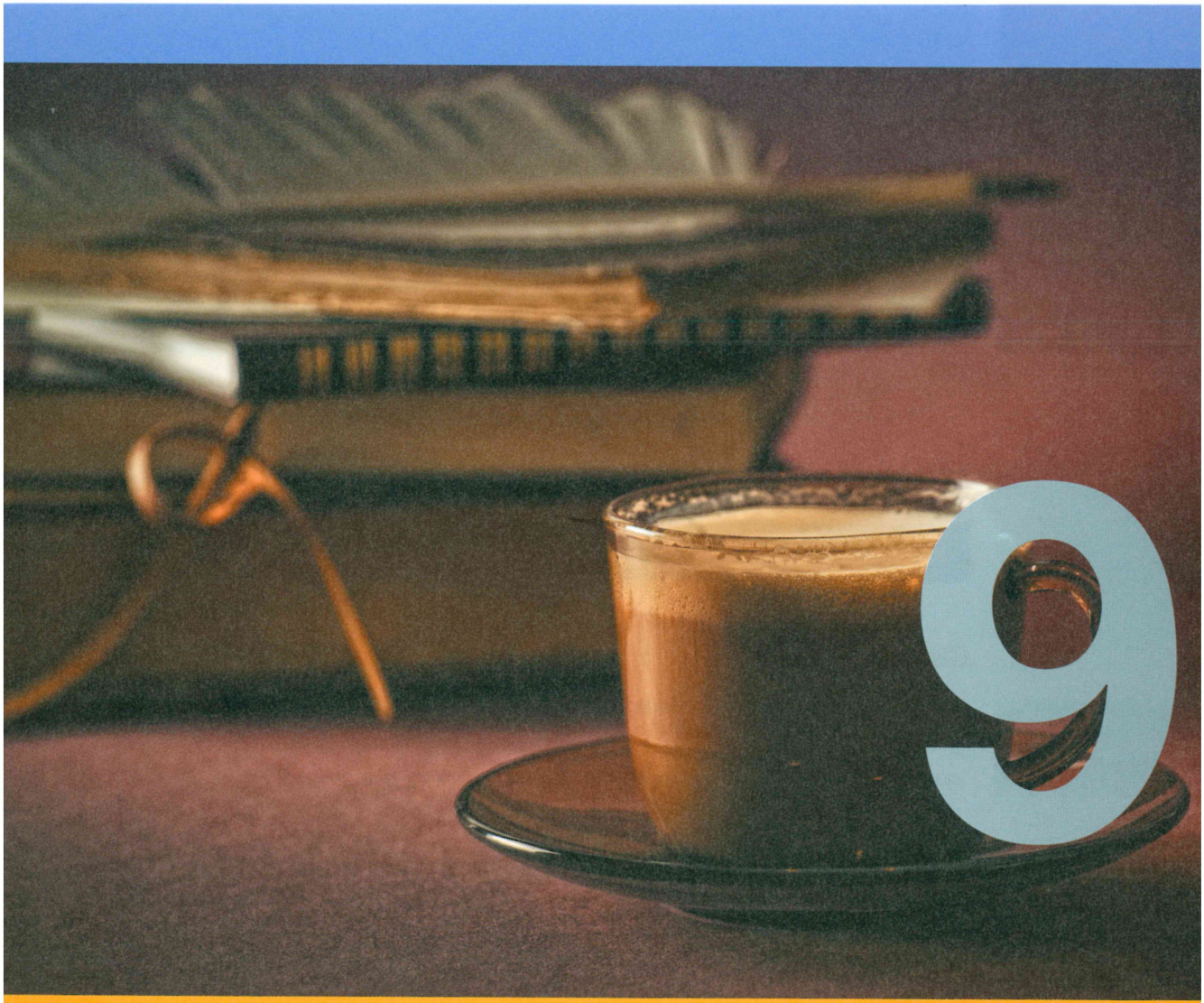
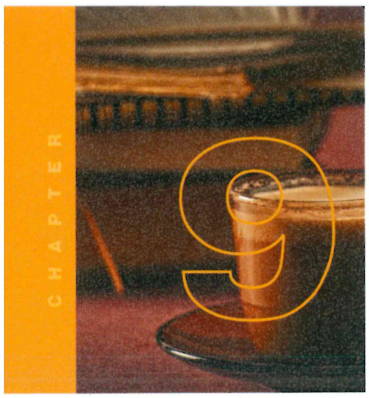


9

자바 GUI 기초, AWT와 스윙(Swing)

Objectives

- 자바 GUI 응용프로그램의 특징을 알고 작성하는 방법을 안다.
- AWT와 스윙 패키지에 대한 개요와 차이점을 안다.
- 스윙 패키지에 속하는 클래스의 계층 구조를 안다.
- 컨테이너와 컴포넌트의 상호 관계를 이해한다.
- 스윙 프로그램 작성 과정을 알고 그에 따라 만들어본다.
- JFrame으로 프레임을 작성해본다.
- 스윙 응용프로그램을 종료하는 방법을 안다.
- 컨테이너와 배치, 배치관리자의 관계를 안다.
- 4가지 유형의 배치관리자를 안다.
- FlowLayout 배치관리자를 사용하여 프로그램을 작성해본다.
- BorderLayout 배치관리자 사용하여 프로그램을 작성해본다.
- GridLayout 배치관리자 사용하여 프로그램을 작성해본다.
- 배치관리자 없는 컨테이너를 만들고 응용해본다.



JAVA PROGRAMMING

자바 GUI 기초, AWT와 스윙(Swing)

9.1 자바의 GUI

GUI 응용프로그램이란?

GUI
AWT와 Swing 패키지

GUI란 Graphical User Interface의 약자로서 이미지 혹은 그래픽을 이용하여 메뉴 등을 포함하는 화면을 구성하고, 키보드 외 마우스 등의 편리한 입력 도구를 이용하여 사용자가 입력하기 편하도록 만들어진 사용자 인터페이스이다.

소프트웨어 개발자들은 프로그래밍 언어를 선택함에 있어 GUI를 작성하기에 편한 정도와 다양한 기능 여부를 중요한 기준으로 삼는다. 자바는 AWT와 Swing 패키지 등 어떤 언어보다 강력한 GUI 라이브러리를 제공하여, 소프트웨어 개발자들이 다양한 GUI를 쉽게 구성하도록 해 준다.

일반적으로 GUI 응용프로그램과 그렇지 않은 콘솔 기반 응용프로그램은 작성 방식이 서로 다르다. 자바도 마찬가지이다. 8장까지는 텍스트를 출력하고 키보드로 입력을 받는 방식으로 사용자 인터페이스를 작성해왔다. 9장부터는 GUI 기반으로 응용프로그램을 작성하는 방법을 알아본다.

AWT와 Swing 패키지

GUI 컴포넌트

자바 언어는 GUI 응용프로그램을 쉽게 작성할 수 있도록 다양한 GUI 컴포넌트(GUI Component)를 제공한다. 자바의 GUI 컴포넌트는 AWT 컴포넌트와 Swing 컴포넌트로 구분되며, 이들을 각각 java.awt 패키지와 javax.swing 패키지를 통해 공급된다.

● AWT

중량 컴포넌트

AWT(Abstract Windowing Toolkit)는 자바가 처음 나왔을 때 함께 배포된 패키지로서 많은 GUI 컴포넌트를 포함한다. Frame, Window, Panel, Dialog, Button, Label, TextField, Checkbox, Choice 등 AWT의 컴포넌트들은 중량 컴포넌트(heavy weight component)로 불리는데, 이 컴포넌트들은 운영체제(native OS)의 도움을 받아 화면

에 출력되기 때문에 운영체제의 자원을 많이 소모하여 운영체제에 많은 부담을 준다. 예를 들어, AWT Button은 윈도우 운영체제(Windows Operating System)에서 윈도우 버튼(Windows Button)에 의해 구현되고, 맥 운영체제(Macintosh Operating System)에서는 맥 버튼(Macintosh Button)에 의해 구현된다. 그러므로 AWT Button은 실행되는 운영체제에 따라 다른 모양으로 그려진다.

● 스윙(Swing)

경량 컴포넌트

스윙(Swing)은 AWT와 달리 순수 자바 언어로 작성되었다. 운영체제의 도움을 받지 않기 때문에 스윙 컴포넌트들은 경량 컴포넌트(light weight component)라고 불린다. 그러므로 스윙 컴포넌트들은 운영체제와 관계없이 항상 동일하게 작동하며 동일한 모양으로 그려진다. 모든 AWT 컴포넌트들이 100% 호환되도록 스윙 컴포넌트로 다시 작성되었으며, 스윙 컴포넌트의 이름은 AWT 컴포넌트와 구분하기 위해 모두 대문자 J로 시작한다. 또한 선마이크로시스템즈사는 AWT에는 없는 풍부한 고급 스윙 컴포넌트들을 추가적으로 개발하였다. AWT보다 스윙 컴포넌트를 이용하면 보다 화려하며 다양한 모양의 GUI 응용프로그램을 쉽게 개발할 수 있다. 스윙은 AWT에 구현된 이벤트 처리나 GUI 컴포넌트의 기본 원리를 바탕으로 작성되었기 때문에, 스윙을 사용하기 위해 AWT 패키지가 필요하다.

스윙은 [그림 9-1]에서 볼 수 있듯이 JFrame, JWindow, JPanel, JButton, JLabel, JList, JTree, JScrollPane 등 다양한 컴포넌트들을 포함한다.

● AWT보다 스윙 사용 권장

개발자는 AWT와 스윙 중 택일하여 GUI 응용프로그램을 개발할 수 있다. 이들 두 라이브러리는 동작하는 방식이 다르고, 응용프로그램을 작성하는 방법에도 차이가 있다. 일반적으로 다양하고 모양이 예쁜 스윙 컴포넌트를 더 많이 사용한다. 최근에는 거의 모든 GUI 응용프로그램이 스윙 기반으로 작성되고 있다.



잠깐!

스윙과 AWT는 작동하는 내부 구조가 서로 다르게 구현되어 있기 때문에 스윙 컴포넌트와 AWT 컴포넌트를 혼용해서는 안 된다. 예를 들어, JButton과 Button을 한 프로그램에서 동시에 사용해서는 안 된다.



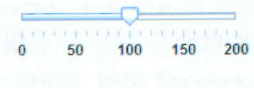
JButton



JCheckBox



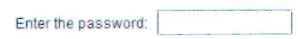
JRadioButton



JSlider



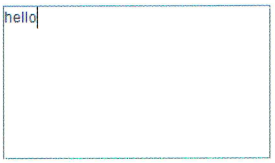
JTextField



JPasswordField



JSpinner



JTextArea



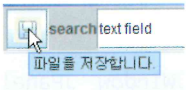
JComboBox



JList



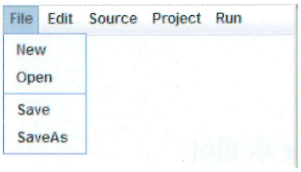
JProgressBar



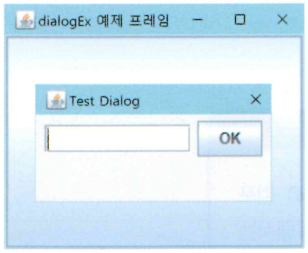
JToolTip



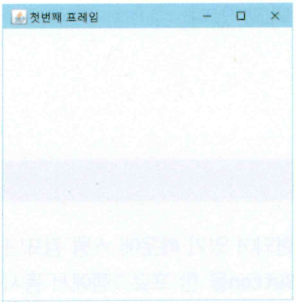
JScrollPane



JMenu



JDialog



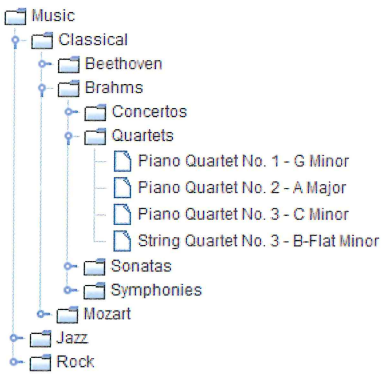
JFrame



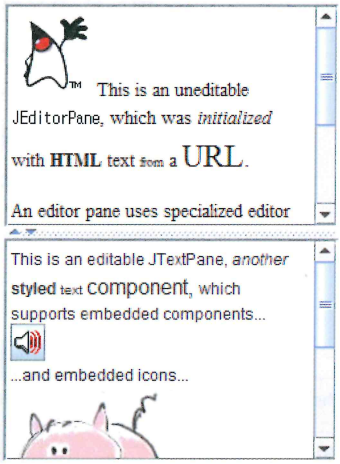
JColorChooser

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Dem...	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whol...	15	
Roger	Brinkley	Blue	The Man Who Kn...	13	
Brent	Christian	Black	Blade Runner (Dir...	23	

JTable



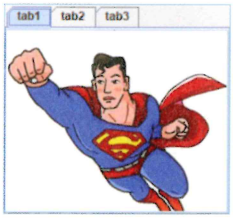
JTree



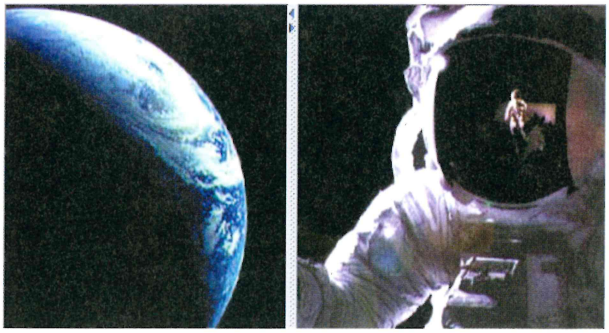
JEditorPane and JTextPane



JToolBar



JTabbedPane



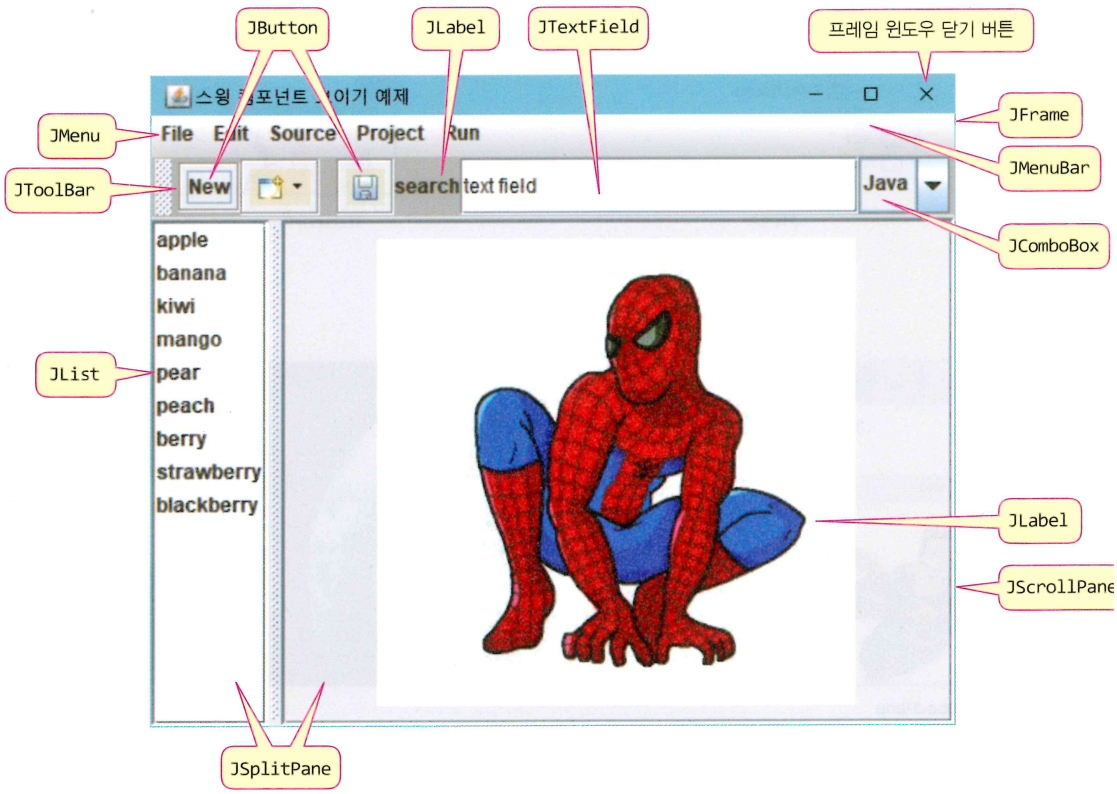
JSplitPane

[그림 9-1] 스윙 컴포넌트

스윙 기반의 GUI 응용프로그램 샘플

[그림 9-2]는 스윙 기반 GUI 응용프로그램의 샘플을 보여준다. 이 응용프로그램은 다
한 스윙 컴포넌트들로 구성되어 있다. 스윙 응용프로그램은 마치 레고 블록으로 조
하듯이 스윙 컴포넌트들을 조립하여 작성된다. [그림 9-2]에 있는 이 컴포넌트들의
할을 간단히 설명하면 다음과 같으며, 이들에 대해서는 10, 11, 14장에서 자세히
룬다.

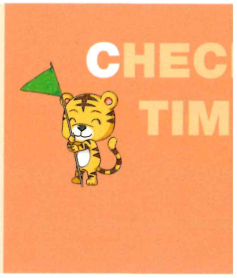
- 응용프로그램의 전체 컴포넌트를 담는 JFrame
- 모든 메뉴를 담는 JMenuBar
- 메뉴로 작동하는 JMenu
- 툴바로 작동하는 JToolBar
- 버튼으로 작동하는 JButton. 툴바에 부착됨
- 문자열을 출력하는 JLabel. 툴바에 부착됨
- 이미지 버튼으로 작동하는 JButton. 툴바에 부착됨



[그림 9-2] 스윙 기반의 GUI 응용프로그램과 스윙 컴포넌트들

- 한 줄 문자열을 입력 받는 창 JTextField. 툴바에 부착됨
- 푸시다운 버튼으로 작동하는 JComboBox. 툴바에 부착됨
- 두 개의 분리된 팬으로 작동하는 JSplitPane. JFrame에 부착됨
- 리스트를 출력하는 JList. JSplitPane의 왼쪽에 부착됨
- 이미지를 출력하는 JLabel. JSplitPane의 오른쪽에 부착됨
- JLabel에 출력된 이미지나 텍스트를 스크롤 가능하게 만드는 JScrollPane

1. AWT와 스윙 중 먼저 세상에 나온 것은 어떤 것인가?
2. 응용프로그램의 GUI 모양이 맥(Mac), 윈도우(Windows), 리눅스(Linux) 등 운영체제에 따라 달라지는 것은 AWT와 스윙 중 어떤 것인가?
3. AWT와 스윙 중 실행 속도가 빠른 것은 어떤 것인가? 그 이유가 무엇이라고 생각되는가?



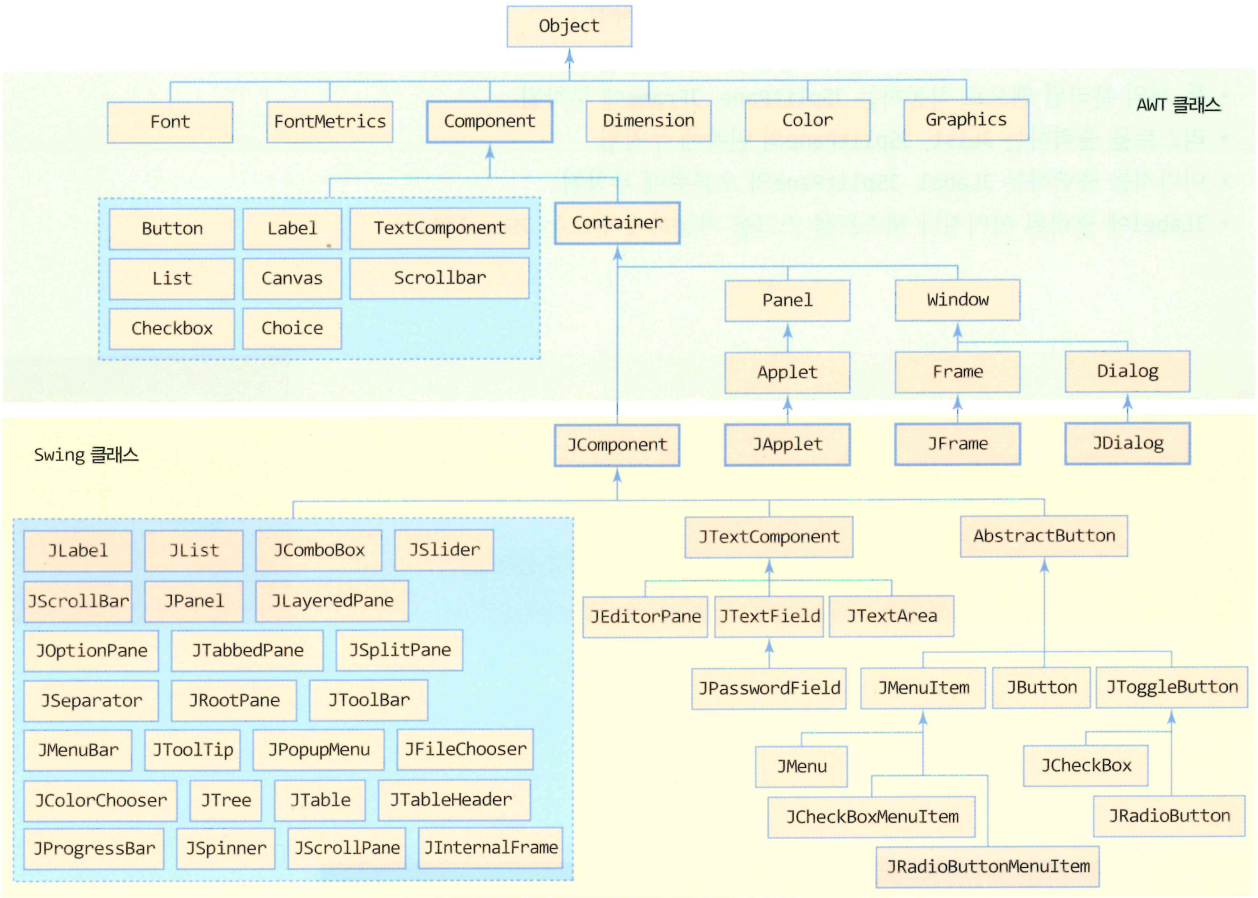
9.2 자바 GUI 패키지

GUI 패키지 계층 구조

Component
JComponent

GUI 응용프로그램을 사용하기 위해 필요한 주요 클래스들의 상속 관계를 [그림 9-3]에 나타내었다. 모든 GUI 컴포넌트들은 Component 클래스를 반드시 상속받으며, 스윙 컴포넌트의 클래스 명은 모두 J로 시작한다. AWT 컴포넌트는 Button, Label 등과 같이 Component를 직접 상속받는 것들과 Panel, Frame 등과 같이 Container를 상속받는 것들이 있다. 그리고 JApplet, JFrame, JDialog를 제외한 모든 스윙 컴포넌트들은 JComponent를 상속받는다.

Font, Dimension, Color, Graphics 등은 컴포넌트가 아니지만, 문자의 폰트 설정, 색, 도형 그리기 등 그래픽 작업 시 반드시 필요하다.



[그림 9-3] AWT와 스윙 클래스의 상속 관계

컨테이너와 컴포넌트

자바의 GUI 응용프로그램은 GUI 컴포넌트들로 구성되며, GUI 컴포넌트들은 다른 컴포넌트를 포함할 수 있는지 여부에 따라 순수 컴포넌트와 컨테이너로 분류된다. 빈 판 위에 레고 블록을 쌓아 가듯이, GUI 컴포넌트들을 쌓아 GUI 응용프로그램을 구성한다.

● 컨테이너

컨테이너
Container

컨테이너란 다른 GUI 컴포넌트를 포함할 수 있는 컴포넌트이다. 그러므로 컨테이너는 컴포넌트이면서 동시에 컨테이너이다. 컨테이너가 되기 위해서는 java.awt.Container 클래스를 상속받아야만 한다. Container 클래스는 [그림 9-3]과 같이 java.awt.Component를 상속받기 때문에 컨테이너가 컴포넌트이기도 한 것이다. 컨테이너는

다른 컨테이너에 컴포넌트로 포함될 수도 있다. 컨테이너로 작동하는 클래스는 다음과 같다.

```
Frame, Panel, Applet, Dialog, Window // AWT 컨테이너
JFrame, JPanel, JApplet, JDialog, JWindow // 스윙 컨테이너
```

● 컴포넌트

컴포넌트
Component

한편, 컴포넌트란 컨테이너와 달리 다른 컴포넌트를 포함할 수 없으며, 컨테이너에 포함되어야 비로소 화면에 출력될 수 있는 GUI 객체이다. AWT나 스윙의 모든 컴포넌트들은 java.awt.Component를 상속받기 때문에, Component 클래스에는 모든 컴포넌트들의 공통적인 속성과 기능이 작성되어 있다. 컴포넌트의 크기, 모양, 위치, 색, 폰트 등에 관한 정보를 관리하는 멤버 변수와 메소드, 컴포넌트 그리기, 이동, 삭제, 이벤트 처리에 관한 메소드 등 다양한 기능을 제공한다. 또한 순수 스윙 컴포넌트들은 모두 javax.swing.JComponent를 상속받으며, JComponent에는 스윙 컴포넌트들의 공통적인 기능이 작성되어 있다(10장 예제 10-1 참고).

● 최상위 컨테이너

최상위 컨테이너
JFrame
JDialog
JApplet

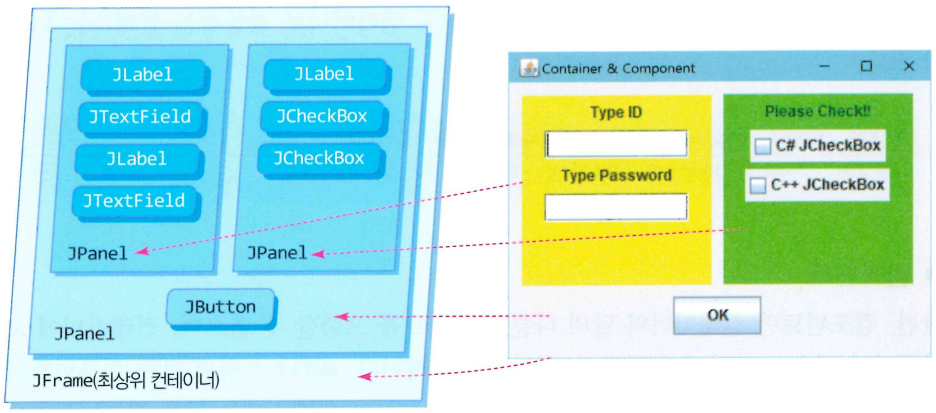
컨테이너 중에서 다른 컨테이너에 속하지 않고도 독립적으로 화면에 출력될 수 있는 컨테이너를 최상위 컨테이너(Top Level Container)라고 한다. JFrame, JDialog, JApplet의 3가지가 이에 속한다. 그러나 이들을 제외한 나머지 컨테이너나 컴포넌트들은 다른 컨테이너에 부착되어야 하고, 종국에는 최상위 컨테이너에 부착되어야만 화면에 출력된다.

● 컨테이너와 컴포넌트의 포함 관계

[그림 9-4]는 최상위 컨테이너와 컨테이너, 컴포넌트의 포함 관계를 보여준다. 그림의 오른쪽은 스윙으로 작성된 GUI 응용프로그램의 실제 모습을, 왼쪽은 응용프로그램을 구성하는 스윙 컴포넌트들의 관계를 보여준다. 맨 바깥에 최상위 컨테이너인 JFrame 컨테이너가 있고, 그 안에 JPanel 컨테이너가 하나 부착되어 있다. 그리고 이 컨테이너에 다시 두 개의 JPanel 컨테이너와 1개의 버튼 컴포넌트가 부착되어 있다. 두 JPanel 컨테이너에는 다시 여러 개의 스윙 컴포넌트들이 부착되어 있다.

자식 컴포넌트

자바 스윙 응용프로그램은 이렇게 JFrame과 같은 최상위 컨테이너 위에 컨테이너와 컴포넌트들이 마치 레고 블록을 쌓는 것처럼 컨테이너와 컴포넌트의 계층 구조로 구성되어, 컨테이너에 부착된 컴포넌트들을 자식 컴포넌트라고 부른다.



스윙의 컨테이너와 컴포넌트의 포함 관계 스윙 GUI 프로그램

[그림 9-4] 스윙 응용프로그램에서 컨테이너와 컴포넌트의 포함 관계

9.3 스윙 GUI 프로그램 만들기

스윙으로 GUI 응용프로그램을 만들기 위해서는 다음의 세 과정이 필요하다.

- 스윙 프레임 작성
- main() 메소드 작성
- 프레임에 스윙 컴포넌트 붙이기

스윙 패키지 사용을 위한 import 문

스윙 패키지를 이용하기 위해서는 스윙 컴포넌트의 클래스 파일들이 존재하는 경로명 javax.swing.*를 import 해야 한다.

```
import javax.swing.*;
```

대부분의 스윙 응용프로그램은 이벤트 처리, 이미지나 도형을 그리는 부분을 필수적으로 동반하므로, 다음과 같은 import 문이 필요한 경우가 많다.

```
import java.awt.*;                  // 폰트 등 그래픽 처리를 위한 클래스들의 경로명
import java.awt.event.*;            // 이벤트 처리에 필요한 기본 클래스들의 경로명
import javax.swing.*;               // 스윙 컴포넌트 클래스들의 경로명
import javax.swing.event.*;         // 스윙 이벤트 처리에 필요한 클래스들의 경로명
```

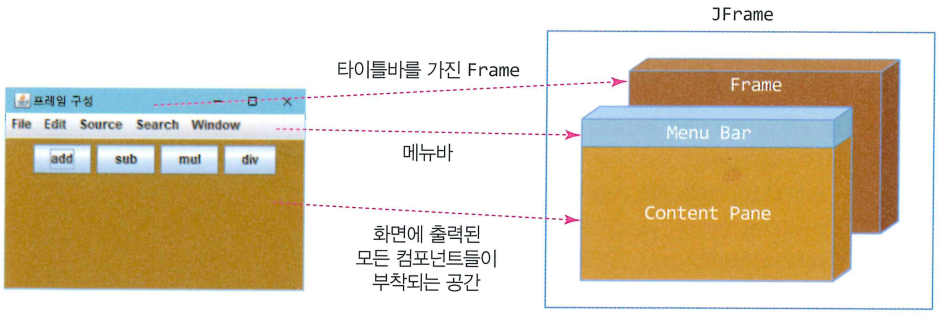
스윙 프레임과 컨텐트팬

JFrame
컨텐트팬

스윙 프레임은 모든 스윙 컴포넌트들을 담는 최상위 컨테이너(Top Level Container)이다. 스윙 프레임이 출력될 때, 스윙 프레임 내에 부착된 모든 컴포넌트들이 화면에 출력된다. 컴포넌트들은 스윙 프레임 없이 독립적으로 화면에 출력될 수 없다. 스윙 프레임이 닫히면 프레임 내의 모든 컴포넌트들도 프레임과 함께 화면에서 사라진다.

스윙에서 프레임의 역할을 수행하는 클래스가 JFrame이다. JFrame 객체가 생성되면 [그림 9-5]와 같이 구성된다. JFrame 객체는 Frame(java.awt.Frame), 메뉴바(Menu Bar), 컨텐트팬(Content Pane)의 3공간으로 구성된다. Frame은 AWT 패키지에 있는 클래스로서 JFrame이 [그림 9-3]과 같이 상속받기 때문에 당연히 그 속성들이 존재하며, 메뉴바는 메뉴들을 부착하는 공간이고, 컨텐트팬은 메뉴를 제외한 모든 GUI 컴포넌트들을 부착하는 공간이다. 스윙 응용프로그램 개발자는 메뉴를 만들어 JFrame의 메뉴바에 부착하여야 하며, 화면에 출력하고자 하는 GUI 컴포넌트들은 모두 컨텐트팬에 부착하여야 한다.

[그림 9-5]의 왼쪽은 JFrame 객체가 출력된 사례로서 5개의 메뉴가 메뉴바에 부착되어 있고, 컨텐트팬에는 add, sub, mul, div 이름의 버튼 컴포넌트들이 부착되어 있다.



[그림 9-5] 스윙 프레임(JFrame)의 구성

프레임 만들기, JFrame 클래스 상속

스윙 응용프로그램의 프레임은 JFrame을 상속받아 만들면 된다. 다음은 JFrame을 상속받아 MyFrame 클래스를 만든 사례이다.

```
public class MyFrame extends JFrame {
.........................
}
```

MyFrame의 객체가 생성되면 스윙 프레임의 역할을 하게 된다. 예제 9-1은 JFrame을 상속받아 프레임을 만든 사례를 보여준다. MyFrame 클래스에 main() 메소드를 두고 main()에서 다음과 같이 MyFrame의 객체를 생성하면 스윙 프레임이 탄생한다.

```
MyFrame frame = new MyFrame(); // 스윙 프레임 생성
```

프레임의 크기와 타이틀은 MyFrame() 생성자에서 다음과 같이 설정하면 된다.

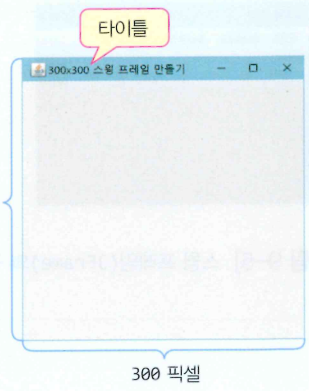
```
setTitle("300x300 스윙 프레임 만들기"); // 프레임 타이틀 설정
setSize(300, 300); // 폭 300, 높이 300 크기로 프레임 크기 설정
setVisible(true); // 프레임이 출력되도록 지시. false의 경우 프레임이 숨겨짐
```



예제 9-1 300x300 크기의 스윙 프레임 만들기

300×300 크기의 스윙 프레임을 작성하라.

```
1  import javax.swing.*;
2
3  public class MyFrame extends JFrame {
4     public MyFrame() {
5        setTitle("300x300 스윙 프레임 만들기");
6        setSize(300,300); // 프레임 크기 300x300
7        setVisible(true); // 프레임 출력
8     }
9
10    public static void main(String[] args) {
11       MyFrame frame = new MyFrame();
12    }
13 }
```



setTitle(), setSize(), setVisible() 메소드는 모두 JFrame의 멤버이다. 이 코드에서 setSize(300, 300)을 호출하지 않으면 프레임의 크기가 0×0이 되어 화면에 출력되지 않는다. 또한 setVisible(true)를 호출하지 않는다면 프레임의 디폴트 속성이 invisible(보이지 않는 상태)이므로 역시 화면에 출력되지 않는다.

MyFrame 객체가 생성되면 스윙 프레임이 화면에 출력되고, 사용자로부터 키나 마우스 입력을 받으면서 스스로 동작한다.

스윙 응용프로그램에서 main() 메소드의 기능과 위치

스윙 응용프로그램에서 main()의 기능은 최소화하는 것이 좋다. main()에는 스윙 응용프로그램이 실행되는 시작점으로서 프레임을 생성하는 코드 정도만 만들고, 나머지 기능은 프레임 클래스에 작성하는 것이 좋다.

main() 메소드는 [그림 9-6]의 (a)와 같이 MyFrame 클래스에 두거나, (b)와 같이 main() 메소드만을 가진 새로운 클래스에 둘 수도 있다. 어떤 방법으로 해도 큰 차이는 없다. 굳이 두 번째 방법처럼 새로운 클래스를 만들 필요는 없을 것 같다.

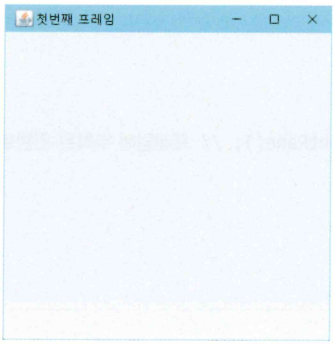
권장

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        setTitle("첫번째 프레임");
        setSize(300, 300);
        setVisible(true);
    }

    public static void main(String [] args) {
        MyFrame mf = new MyFrame();
    }
}
```

(a) 방법 1: main() 메소드를 MyFrame의 멤버로 작성

```
import javax.swing.*;

public class MyFrame extends JFrame {
    public MyFrame() {
        setTitle("첫번째 프레임");
        setSize(300, 300);
        setVisible(true);
    }
}
public class MyApp {
    public static void main(String [] args) {
        MyFrame mf = new MyFrame();
    }
}
```

(b) 방법 2: main() 메소드를 가진 새로운 MyApp 클래스 작성



(c) 두 가지 방법으로 실행해도 동일한 프레임 화면 생성



[그림 9-6] main() 메소드의 기능과 위치

프레임에 컴포넌트 붙이기

프레임은 응용프로그램을 구성하는 바탕 틀이다. 지금부터 프레임 안에 스윙 컴포넌트들을 부착해보자.

● 타이틀 달기

super()
setTitle()

프레임에 타이틀을 달기 위해서는 다음과 같이 super()를 이용하여 JFrame의 생성자를 호출하거나, JFrame 클래스의 setTitle() 메소드를 이용한다.

```
public MyFrame() { // 생성자
    super("타이틀문자열"); // JFrame("타이틀문자열") 생성자를 호출하여 타이틀 달기
    setTitle("타이틀문자열"); // 메소드를 호출하여 타이틀 달기
}
```

타이틀은 스윙 응용프로그램의 타이틀 바에 출력된다.

● 메뉴 붙이기

메뉴를 작성하기 위해서는 메뉴바를 만들고, 메뉴를 붙이고, 메뉴에는 여러 개의 메뉴 아이템을 붙인다. 메뉴바를 JFrame의 메뉴바 영역에 붙이면 화면에 메뉴가 출력된다. 메뉴에 대해서는 14장에서 자세히 설명한다.

● 컨텐트팬에 컴포넌트 달기

컨텐트팬
getContentPane()

스윙에서는 컨텐트팬(content pane)에만 컴포넌트를 부착할 수 있다. JFrame 객체가 생길 때 컨텐트팬이 자동으로 생성된다. 그러므로 현재 프레임에 붙어 있는 컨텐트팬을 알아내기 위해서는, 다음과 같이 JFrame 클래스의 getContentPane() 메소드를 호출한다. 컨텐트팬은 Container 타입이다.

```
public class MyFrame extends JFrame {
    public MyFrame() {
        ...
        Container contentPane = getContentPane(); // 프레임에 부착된 컨텐트팬을 알아낸다.

    }
    ...
}
```

컨텐트팬에 컴포넌트를 붙이는 것은 비교적 간단하다. 컨텐트팬은 컨테이너이기 때문에 다음과 같이 add() 메소드를 이용하여 간단히 컴포넌트를 부착하면 된다.

```
JButton button = new JButton("Click"); // 버튼 컴포넌트 생성
contentPane.add(button); // 컨텐트팬에 버튼 부착
```

컨텐트팬에 부착된 컴포넌트들은 프레임이 출력될 때 함께 화면에 출력된다.

● 컨텐트팬의 변경

JFrame 클래스의 setContentPane() 메소드를 이용하면 프레임에 부착된 컨텐트팬을 제거하고 새로운 컨텐트팬을 붙일 수 있다. 컨텐트팬은 Container 타입이므로 Container를 상속받은 어떤 컨테이너도 컨텐트팬이 될 수 있다. 다음 코드는 JPanel을 상속받은 MyPanel 객체를 생성하여 컨텐트팬으로 붙이는 예이다.

setContentPane()

```
class MyPanel extends JPanel {
    // JPanel을 상속받은 패널을 작성한다.
}

frame.setContentPane(new MyPanel()); // 프레임의 컨텐트팬을 MyPanel 객체로 변경
```



Tip 컨텐트팬과 JDK 1.5 버전 이후의 변경 사항

스윙 프레임의 초기 버전에서는 개발자가 컨텐트팬에 직접 컴포넌트를 부착하도록 하는 다소 부담스러운 제약이 있었다. 하지만, 자바 초보자들이 컨텐트팬의 존재를 모른 채 코딩하다 오류를 범하는 경우가 많아지자, JDK 1.5부터는 JFrame의 add(Component comp)를 호출해도 이 add() 메소드가 컴포넌트를 컨텐트팬에 대신 부착하도록 수정하였다. 이제, 개발자들은 컨텐트팬을 의식하지 않고 다음과 같이 JFrame에 컴포넌트를 바로 추가할 수 있다.

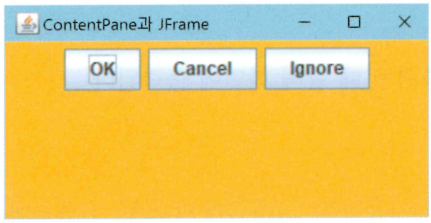
```
JFrame frame = new JFrame();
JButton b1 = new JButton("Click");
frame.add(b1); // 프레임의 add()가 대신 컨텐트팬에 button 부착. JDK 1.5 부터
```

그렇다 하더라도 컨텐트팬이 없어진 것이 아니다. 저자는 이런 방식으로 프로그램을 작성하는 것을 결코 권하지 않는다. 가능하면 컨텐트팬에 직접 컴포넌트를 붙이는 코딩 습관을 가지는 것이 좋다. 수준이 올라갈수록 JFrame의 구조와 컨텐트팬에 대해 정확히 알지 못하면 낭패를 당하는 경우가 종종 있다.

예제 9-2 3개의 버튼 컴포넌트를 가진 스윙 프레임 만들기

다음 그림과 같이 컨텐트팬의 배경색을 오렌지색으로 하고, 이곳에 OK, Cancel, Ignore 버튼들을 부착한 스윙 프로그램을 작성하라.

이 예제의 코드에 나오는 JButton, FlowLayout은 뒤에서 다루므로, 일단 넘어가도록 한다.




```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class ContentPaneEx extends JFrame {
5     public ContentPaneEx() {
6        setTitle("ContentPane과 JFrame"); // 프레임 타이틀 달기
7        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 프레임 윈도우를 닫으면
                                                         프로그램을 종료하도록 설정
8
9        Container contentPane = getContentPane(); // 컨텐트 팬을 알아낸다.
10       contentPane.setBackground(Color.ORANGE); // 컨텐트팬의 색을 오렌지색으로
11       contentPane.setLayout(new FlowLayout()); // 컨텐트팬에 FlowLayout 배치관
                                                    리자 달기
12
13       contentPane.add(new JButton("OK")); // OK 버튼 달기
14       contentPane.add(new JButton("Cancel")); // Cancel 버튼 달기
15       contentPane.add(new JButton("Ignore")); // Ignore 버튼 달기
16
17       setSize(300, 150); // 프레임 크기 300x150 설정
18       setVisible(true); // 화면에 프레임 출력
19    }
20
21    public static void main(String[] args) {
22       new ContentPaneEx();
23    }
24 }
```

배치관리자는 9.4절 참고

스윙 응용프로그램의 종료

자바에서 프로그램을 종료하려면 어떤 상황에서나 다음 코드를 사용하면 된다.

```
System.exit(0); // 자바 프로그램을 종료한다.
```

한편 프레임 윈도우의 오른쪽 상단에 있는 '프레임 종료 버튼(X)'은 프레임 윈도우를 닫는 버튼이지 프로그램을 종료시키는 버튼은 아니다. 사용자가 스윙 프로그램을 종료하기 위해 '프레임 종료 버튼(X)'을 클릭하면 프레임이 화면에서 사라지지만 프로그램은 종료되지 않고 살아 있다. 프레임 윈도우가 닫힐 때 스윙 프로그램도 함께 종료시키고자 한다면 다음 코드를 삽입하여야 한다.

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
// 프레임이 닫힐 때 프로그램도 함께 종료한다.
```

main() 메소드가 종료한 뒤에도 프레임이 살아 있는 이유는 무엇인가?

8장까지 다룬 콘솔 응용프로그램에서는 main() 메소드가 종료하면 자바 응용프로그램이 종료되었다. 그러나 스윙 응용프로그램에서는 main() 메소드가 종료되어도 프레임 윈도우가 살아서 작동한다. 그 이유는 무엇일까?

자바 응용프로그램이 시작되면 자바 가상 기계는 main 스레드를 만들고 main()을 실행시킨다(스레드는 13장에서 자세히 설명). 응용프로그램이 스레드를 만들지 않는 경우, main()이 종료하면 main 스레드도 종료되며, 더 이상 살아 있는 스레드가 없기 때문에 이 응용프로그램은 종료하게 되는 것이다.

main 스레드

그러나 스윙에서 JFrame 객체가 생성되면 main 스레드 외에, 입력되는 키와 마우스의 움직임을 컴포넌트에게 이벤트로 전달하는 이벤트 처리(분배) 스레드(Event Dispatch Thread)가 자동으로 추가 생성된다. 그러므로 main 스레드가 종료하더라도 이벤트 처리 스레드가 살아 있기 때문에, 자바 응용프로그램은 종료되지 않고 사용자로부터 키와 마우스 입력을 계속 처리하게 되는 것이다.

이벤트 처리(분배) 스레드

1 다음 중 최상위 컨테이너가 아닌 것은?

① JPanel ② JDialog ③ JFrame ④ JApplet

2 JFrame 내에 컴포넌트가 부착되는 특별한 영역을 무엇이라고 부르는가? 이 영역에 "Hello" 문자열을 가진 JButton 컴포넌트를 삽입하는 간단한 코드를 보여라.



9.4 컨테이너(Container)와 배치(Layout)

컨테이너와 배치 개념

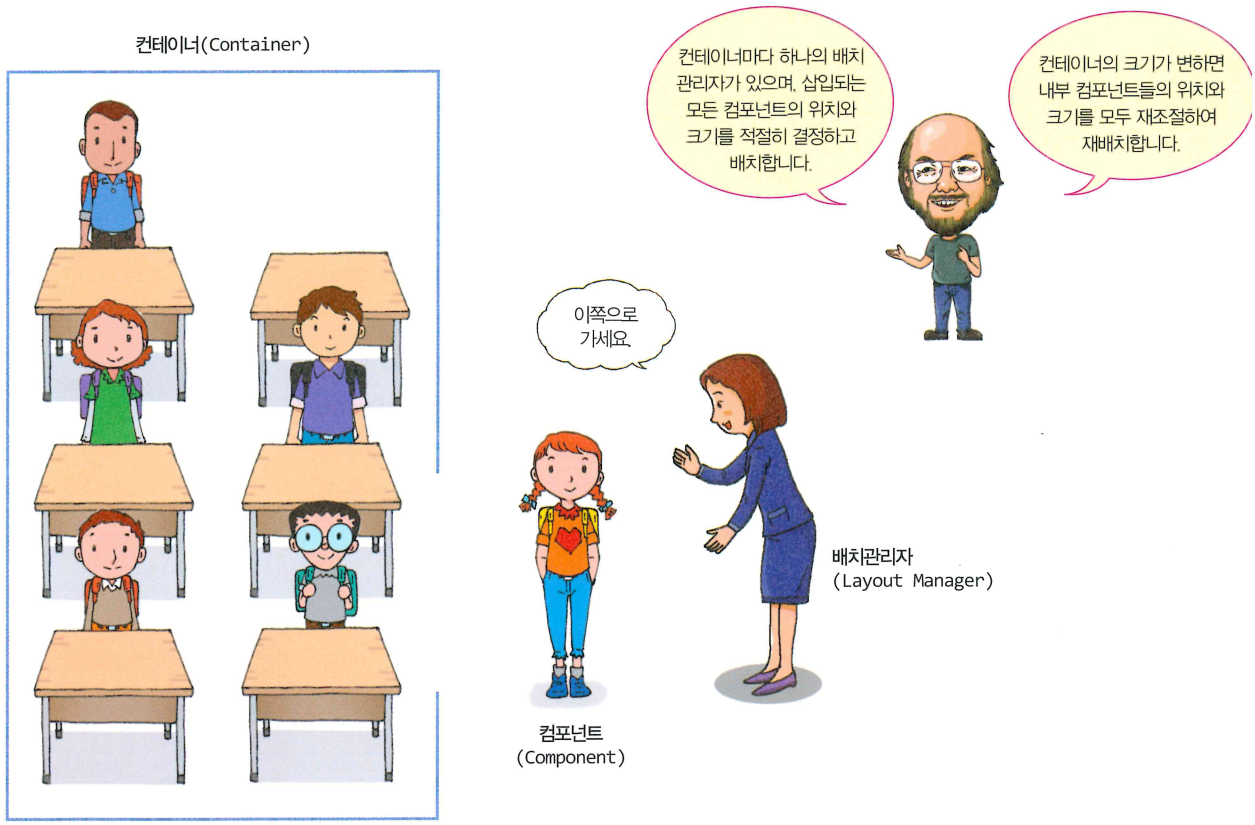
컨테이너
배치관리자

컨테이너에 부착되는 컴포넌트들의 위치와 크기는 컨테이너 내부에 있는 배치관리자(Layout Manager)에 의해 결정된다. [그림 9-7]은 컨테이너와 배치관리자 사이의 관계를 보여준다. 안내원(배치관리자)은 강의실(컨테이너)에 들어오는 각 학생들(컴포넌트)에게 위치를 지정(배치)해주고 있다.

배치와 관련하여 AWT나 스윙의 컨테이너는 다음과 같은 특징을 가진다.

컴포넌트의 위치와 크기 결정

- 컨테이너마다 배치관리자가 하나씩 있다(하지만, 배치관리자를 삭제할 수도 있다).
- 배치관리자는 컨테이너에 컴포넌트가 부착되는 시점에 컴포넌트의 위치와 크기를 결정한다.



[그림 9-7] 컨테이너와 배치, 배치관리자 사이의 관계

- 컨테이너의 크기가 변경되면 배치관리자는 컨테이너 내부의 모든 컴포넌트들의 위치와 크기를 재조정한다.

배치관리자의 종류

자바는 여러 종류의 배치관리자를 지원한다. [그림 9-8]은 대표적인 4가지 배치관리자가 컴포넌트를 배치하는 형태를 보여준다. 배치관리자는 java.awt 패키지에 존재하는 클래스들이며, 이들을 사용하기 위해서는 다음 import 문이 필요하다.

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

4가지 대표적인 배치관리자는 다음과 같다.

● FlowLayout

컨테이너에 부착되는 순서대로 왼쪽에서 오른쪽으로 컴포넌트를 배치하며, 오른쪽에 더 이상 배치할 공간이 없으면 아래로 내려와서 다시 왼쪽에서 오른쪽으로 배치한다. 컴포넌트의 크기는 화면에 출력될 수 있는 적당한 크기로 설정한다.

왼쪽에서 오른쪽으로

● BorderLayout

컨테이너의 공간을 동(EAST), 서(WEST), 남(SOUTH), 북(NORTH), 중앙(CENTER)의 5개의 영역으로 나누고 응용프로그램에서 지정한 영역에 컴포넌트를 배치한다. 응용프로그램에서는 5개의 영역 중 하나를 반드시 지정하여야 한다. 영역을 지정하지 않으면 중앙에 배치된다. 컴포넌트의 크기는 영역의 크기에 맞추어 설정된다.

5개의 영역

● GridLayout

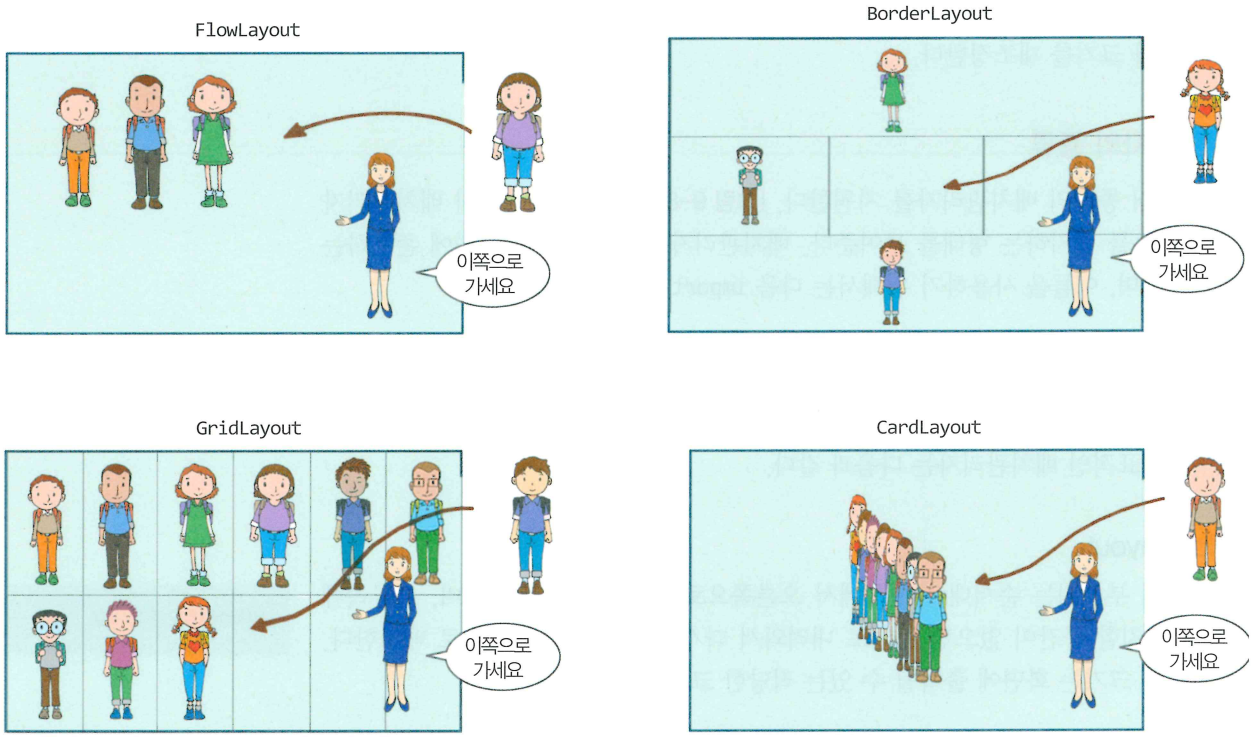
컨테이너의 공간을 응용프로그램에서 설정한 동일한 크기의 2차원 격자로 나누고, 컴포넌트가 삽입되는 순서대로 좌에서 우로, 다시 위에서 아래로 배치한다. 컴포넌트의 크기는 셀의 크기와 동일하게 설정한다.

2차원 격자

● CardLayout

컨테이너의 공간에 카드를 쌓아 놓은 듯이 컴포넌트를 포개어 배치한다. 컴포넌트의 크기는 컨테이너의 크기와 동일하게 설정한다.

카드



[그림 9-8] 대표적인 4가지 배치관리자

컨테이너의 디폴트 배치관리자

컨테이너가 생성될 때 자동으로 디폴트 배치관리자가 생성되며 이들은 〈표 9-1〉과 같다.

〈표 9-1〉
컨테이너의 디폴트 배치관리자

AWT와 스윙 컨테이너	디폴트 배치관리자
Window, JWindow	BorderLayout
Frame, JFrame	BorderLayout
Dialog, JDialog	BorderLayout
Panel, JPanel	FlowLayout
Applet, JApplet	FlowLayout

컨테이너에 새로운 배치관리자 설정, setLayout() 메소드

setLayout()

컨테이너에는 새로운 배치관리자를 마음대로 설정할 수 있다. 이를 위해 다음과 같이 Container 클래스의 setLayout() 메소드를 호출한다.

```
Container.setLayout(LayoutManager lm); // lm을 새로운 배치관리자로 설정한다.
```

예를 들어, JPanel에 BorderLayout 배치관리자를 설정하려면 다음과 같다.

```
JPanel p = new JPanel();
p.setLayout(new BorderLayout()); // 패널에 BorderLayout 배치관리자 설정
```

또한 컨텐트팬의 배치관리자를 FlowLayout으로 변경하려면 다음과 같이 한다.

```
Container contentPane = frame.getContentPane(); // 프레임의 컨텐트팬
contentPane.setLayout(new FlowLayout()); // FlowLayout 배치관리자 설정
```

다음과 같이 배치 관리자 객체를 생성하는 new를 빠트리지 않도록 주의하도록 하라.

오류

```
contentPane.setLayout(FlowLayout); // 오류
```

9.5 FlowLayout 배치관리자

배치 방법

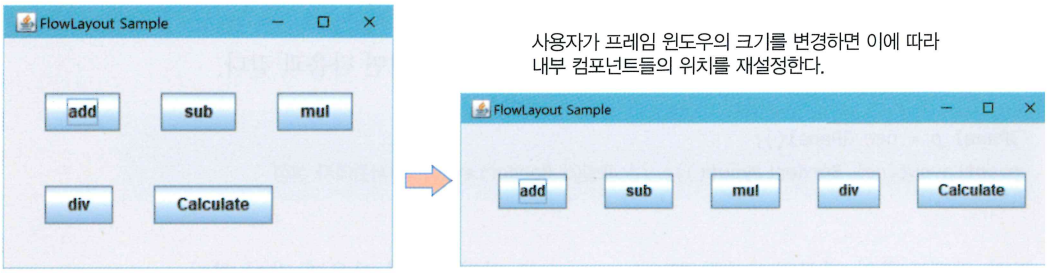
FlowLayout

add()

FlowLayout 배치관리자를 가진 컨테이너에 컴포넌트를 부착하는 방법은 다음과 같이 간단히 add() 메소드를 이용하면 된다.

```
container.setLayout(new FlowLayout());
container.add(new JButton("add"));
container.add(new JButton("sub"));
container.add(new JButton("mul"));
container.add(new JButton("div"));
container.add(new JButton("Calculate"));
```

FlowLayout 배치관리자는 컴포넌트를 왼쪽에서 오른쪽으로 배치하고, 더 이상 오른쪽 공간이 없으면 다시 아래로 내려와서 왼쪽에서 오른쪽으로 배치한다. [그림 9-9]는 5개 버튼 컴포넌트가 순서대로 부착된 모습을 보여준다. 컨테이너의 크기가 변하면, FlowLayout 배치관리자에 의해 컨테이너 크기에 맞도록 컴포넌트가 재배치된다.



[그림 9-9] FlowLayout 배치관리자를 가진 프레임의 크기가 변경될 때 재배치되는 모양

FlowLayout의 생성자와 속성

FlowLayout 배치관리자의 생성자는 다음과 같이 여러 개 있으며, 생성자에 컴포넌트 사이의 간격과 정렬 방식을 지정할 수 있으며 예제 9-3은 FlowLayout 배치관리자의 align, hGap, vGap 속성을 각각 보여준다.

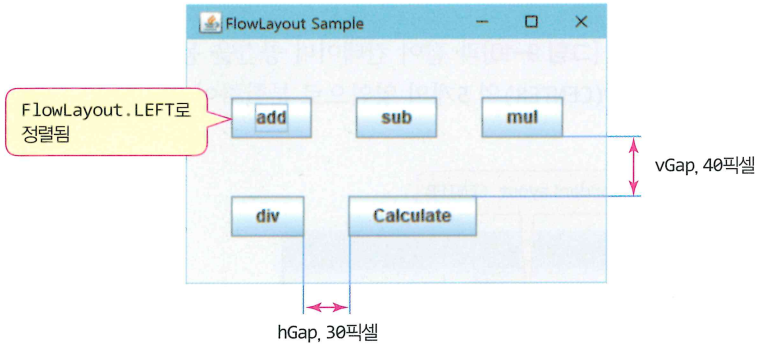
```
FlowLayout()
FlowLayout(int align)
FlowLayout(int align, int hGap, int vGap)
• align: 컴포넌트의 정렬 방법. 왼쪽 정렬(FlowLayout.LEFT), 오른쪽 정렬(FlowLayout.
  RIGHT), 중앙 정렬(FlowLayout.CENTER(디폴트))
• hGap: 좌우 컴포넌트 사이의 수평 간격, 픽셀 단위. 디폴트는 5
• vGap: 상하 컴포넌트 사이의 수직 간격, 픽셀 단위. 디폴트는 5
```

FlowLayout의 생성자를 사용하는 예를 들면 다음과 같다.

```
new FlowLayout(); // 중앙 정렬과 hGap=5, vGap=5인 배치관리자
new FlowLayout(FlowLayout.LEFT); // 왼쪽 정렬과 hGap=5, vGap=5인 배치관리자
new FlowLayout(FlowLayout.LEFT, 10, 20); // 중앙 정렬. hGap=10, vGap=20인 배치관리자
```

FlowLayout 배치관리자 활용 예제 9-3

수평 간격이 30, 수직 간격을 40픽셀로 하고 LEFT로 정렬 배치하는 FlowLayout 배치관리자를 가진 컨텐트팬에 5개의 버튼 컴포넌트를 부착한 스윙 응용프로그램을 작성하라.




```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class FlowLayoutEx extends JFrame {
5     public FlowLayoutEx() {
6        setTitle("FlowLayout Sample");
7        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8        Container c = getContentPane();
9
10       // 컨텐트팬에 FlowLayout 배치관리자 설정
11       c.setLayout(new FlowLayout(FlowLayout.LEFT, 30, 40));
12       c.add(new JButton("add"));
13       c.add(new JButton("sub"));
14       c.add(new JButton("mul"));
15       c.add(new JButton("div"));
16       c.add(new JButton("Calculate"));
17
18       setSize(300, 200); // 프레임 크기 300x200 설정
19       setVisible(true); // 화면에 프레임 출력
20    }
21    public static void main(String[] args) {
22       new FlowLayoutEx();
23    }
24 }
```

9.6 BorderLayout 배치관리자

배치 방법

BorderLayout

동(EAST)

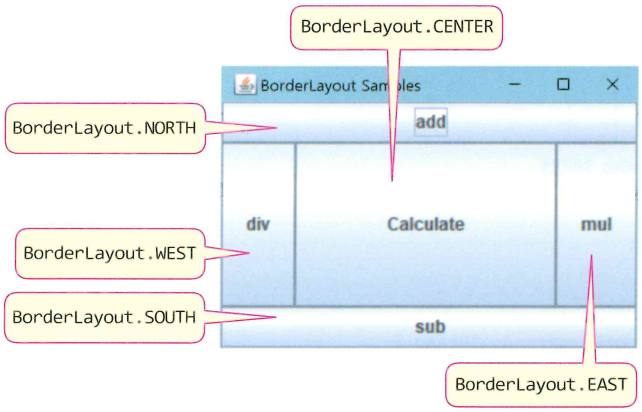
서(WEST)

남(SOUTH)

북(NORTH)

중앙(CENTER)

BorderLayout 배치관리자는 [그림 9-10]과 같이 컨테이너 공간을 동(EAST), 서(WEST), 남(SOUTH), 북(NORTH), 중앙(CENTER)의 5개의 영역으로 분할하여 배치한다.



[그림 9-10] BorderLayout 배치관리자가 설치된 컨테이너의 공간 분할

add()

다른 배치관리자와는 달리 BorderLayout 배치관리자를 가진 컨테이너에 컴포넌트를 삽입하기 위해서는, 다음과 같이 add() 메소드에 컴포넌트의 위치를 명확히 정해주어야 한다.

```
void add(Component comp, int index) comp 컴포넌트를 index 위치에 삽입한다.
• comp: 컨테이너에 삽입되는 컴포넌트
• index: 컴포넌트의 위치
      동: BorderLayout.EAST
      서: BorderLayout.WEST
      남: BorderLayout.SOUTH
      북: BorderLayout.NORTH
    중앙: BorderLayout.CENTER
```

add() 메소드를 사용하는 예를 들어보자.

```
container.setLayout(new BorderLayout());
container.add(new JButton("div"), BorderLayout.WEST);
container.add(new JButton("Calculate"), BorderLayout.CENTER);
```

컨테이너의 크기가 변하면 BorderLayout 배치관리자 역시 새로운 크기에 맞도록 컴포넌트의 크기를 재조정한다. BorderLayout 배치관리자를 사용하는 컨테이너에는 일차적으로 5개의 컴포넌트밖에 붙일 수가 없다. BorderLayout 컨테이너에 5개 이상의 컴포넌트를 부착하고자 하면, 한 영역에 JPanel 등 다른 컨테이너를 부착하고 이곳에 컴포넌트들을 부착하면 된다.

BorderLayout의 생성자와 속성

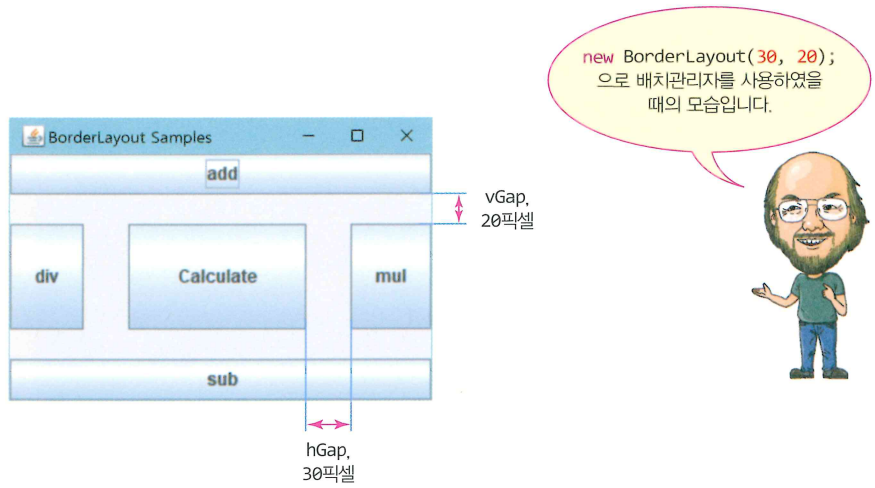
BorderLayout 배치관리자의 생성자는 다음과 같으며, 생성자를 이용하면 컴포넌트 사이의 간격으로 조절할 수 있다.

BorderLayout()
BorderLayout(int hGap, int vGap)
- hGap: 좌우 두 컴포넌트 사이의 수평 간격, 픽셀 단위. 디폴트는 0
- vGap: 상하 두 컴포넌트 사이의 수직 간격, 픽셀 단위. 디폴트는 0

BorderLayout의 생성자를 사용하는 예를 들어보자.

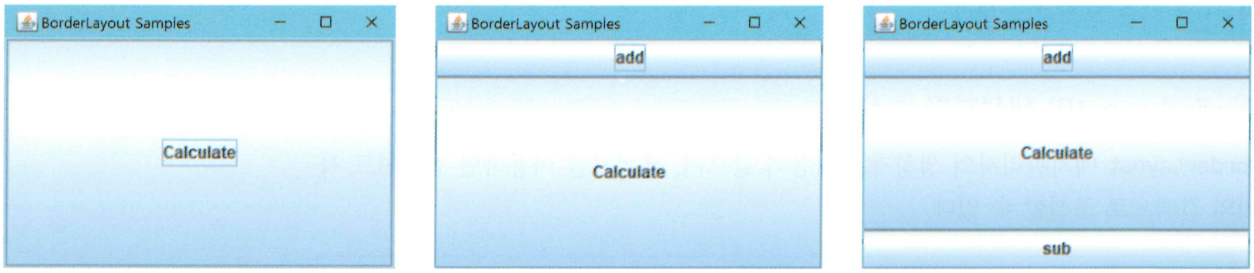
```
new BorderLayout(); // hGap=0, vGap=0인 BorderLayout 배치관리자 생성
new BorderLayout(30, 20); // hGap=30, vGap=20인 BorderLayout 배치관리자 생성
```

[그림 9-11]은 BorderLayout 배치관리자를 사용하는 사례를 보여준다.



[그림 9-11] BorderLayout 배치관리자의 사용 사례

CENTER에 컴포넌트가 부착될 때 다른 컴포넌트가 없으면 컨테이너 전체 크기로 배치된다. [그림 9-12]는 BorderLayout 컨테이너에 컴포넌트가 부착되는 과정을 보여준다.



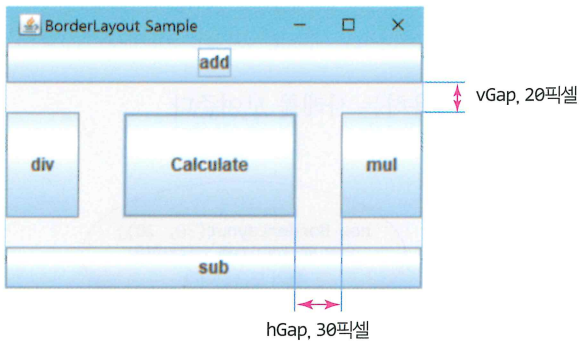
(a) CENTER에 컴포넌트가 삽입될 때 (b) CENTER와 NORTH에 컴포넌트가 삽입될 때 (c) CENTER, NORTH, SOUTH에 컴포넌트가 삽입될 때

[그림 9-12] BorderLayout 배치관리자를 가진 컨테이너에 컴포넌트 삽입



예제 9-4 BorderLayout 배치관리자를 사용하는 예

BorderLayout 배치관리자를 사용하여 컨텐트팬에 다음과 같이 5개의 버튼 컴포넌트를 삽입하라.




```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class BorderLayoutEx extends JFrame {
5     public BorderLayoutEx() {
6        setTitle("BorderLayout Sample");
7        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8        Container c = getContentPane();
9
10       // 컨텐트팬에 BorderLayout 배치관리자 설정
11       c.setLayout(new BorderLayout(30, 20));
```

```
12
13        c.add(new JButton("Calculate"), BorderLayout.CENTER);
14        c.add(new JButton("add"), BorderLayout.NORTH);
15        c.add(new JButton("sub"), BorderLayout.SOUTH);
16        c.add(new JButton("mul"), BorderLayout.EAST);
17        c.add(new JButton("div"), BorderLayout.WEST);
18
19        setSize(300, 200); // 프레임 크기 300×200 설정
20        setVisible(true); // 프레임을 화면에 출력
21     }
22     public static void main(String[] args) {
23        new BorderLayoutEx();
24     }
25  }
```

9.7 GridLayout 배치관리자

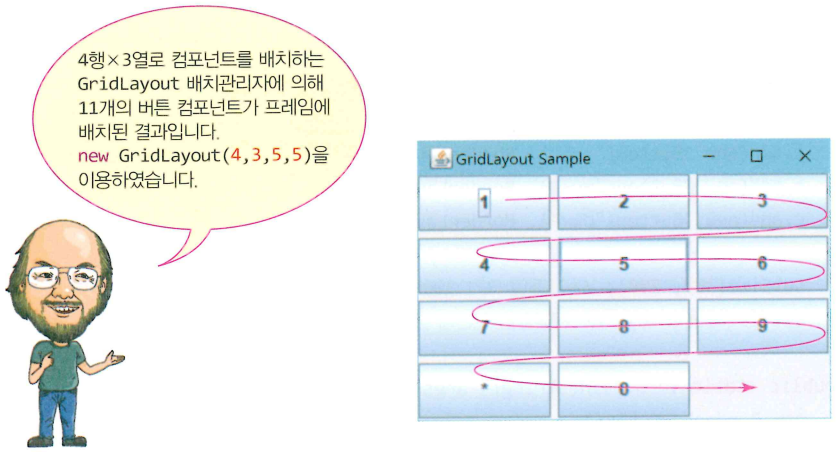
배치 방법

GridLayout

GridLayout은 컨테이너 공간을 그리드(격자) 모양으로 분할하여 각 셀에 하나씩 컴포넌트를 배치하는 방법이다. GridLayout은 부착되는 순서대로 컴포넌트를 셀에 배치한다. [그림 9-13]은 GridLayout에 의해 4×3 그리드로 분할된 프레임에 11개의 버튼이 부착된 상황을 보여준다. 여기서 붉은색 선은 컴포넌트를 배치하는 셀 순서이다. 12번째 새로운 버튼 컴포넌트가 추가되면 오른쪽 하단의 빈 셀에 배치된다.

컴포넌트를 삽입하는 방법은 다른 배치관리자와 별반 다르지 않으며 다음과 같다.

```
container.setLayout(new GridLayout(4,3)); // 4×3 분할로 컴포넌트 배치
container.add(new JButton("1")); // 상단 왼쪽 첫 번째 셀에 버튼 배치
container.add(new JButton("2")); // 그 옆 셀에 버튼 배치
```



[그림 9-13] 4×3 분할의 GridLayout 배치관리자에 의해 11개의 버튼 컴포넌트가 배치된 예

GridLayout의 생성자와 속성

GridLayout 배치관리자의 생성자는 다음과 같다.

```
GridLayout()
GridLayout(int rows, int cols)
GridLayout(int rows, int cols, int hGap, int vGap)
```
- rows: 그리드의 행 수, 디폴트는 1
- cols: 그리드의 열 수, 디폴트는 1
- hGap: 좌우 컴포넌트 사이의 수평 간격, 픽셀 단위. 디폴트는 0
- vGap: 상하 컴포넌트 사이의 수직 간격, 픽셀 단위. 디폴트는 0

GridLayout의 생성자를 사용하는 예를 들면 다음 코드와 같다.

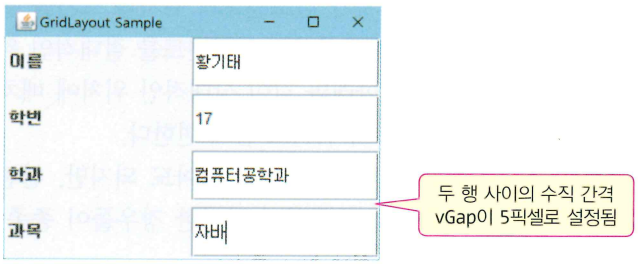
```
new GridLayout(); // 1×1 그리드에 배치하는 GridLayout 배치관리자 생성
new GridLayout(4, 3, 5, 5); // 4×3 그리드에 hGap=5, vGap=5의 간격으로 컴포넌트를 배치
                            하는 GridLayout 배치관리자 생성
```

셀 수보다 많은 컴포넌트가 추가되면 생성자에 주어진 행 수(rows)와 열 수(cols)가 지켜지지 않으며, 모든 컴포넌트를 수용하도록 행과 열의 수가 적당히 변형된다.

GridLayout으로 입력 폼 만들기 예제 9-5

아래 화면과 같이 사용자로부터 입력받는 폼을 스윙 응용프로그램을 작성하라.

GridLayout을 사용하여 콘텐트팬을 4×2 그리드로 분할하고 JLabel 컴포넌트와 JTextField 컴포넌트를 부착하면 된다. 두 행 사이의 수직 간격은 5픽셀로 하라.




```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class GridLayoutEx extends JFrame {
5     public GridLayoutEx() {
6        setTitle("GridLayout Sample");
7        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8
9        // 4x2 격자의 GridLayout 배치관리자 생성
10       GridLayout grid = new GridLayout(4, 2);
11       grid.setVgap(5); // 격자 사이의 수직 간격을 5 픽셀로 설정
12
13       Container c = getContentPane();
14       c.setLayout(grid); // grid를 컨텐트팬의 배치관리자로 지정
15       c.add(new JLabel(" 이름"));
16       c.add(new JTextField(""));
17       c.add(new JLabel(" 학번"));
18       c.add(new JTextField(""));
19       c.add(new JLabel(" 학과"));
20       c.add(new JTextField(""));
21       c.add(new JLabel(" 과목"));
22       c.add(new JTextField(""));
23
24       setSize(300, 200);
25       setVisible(true);
26    }
27    public static void main(String[] args) {
28       new GridLayoutEx();
29    }
30 }
```

9.8 배치관리자 없는 컨테이너

배치관리자 없는 컨테이너가 필요한 경우

모든 컨테이너는 배치관리자를 가지고, 배치관리자가 컴포넌트의 위치와 크기를 결정하도록 하는 것이 자바 GUI의 기본 정신이다. 배치관리자는 컴포넌트를 절대적인 위치에 배치하지 않고, 다른 컴포넌트의 오른쪽이나 아래와 같이 상대적인 위치에 배치한다. 그러므로 컨테이너의 크기가 변하면 컴포넌트의 위치도 함께 변한다.

배치관리자를 사용하면 컴포넌트의 위치에 대해 신경 쓰지 않아도 되지만, 한편으로 컴포넌트의 위치를 자동 결정하는 배치관리자가 오히려 불편한 경우들이 종종 있다. 어떤 경우인지 구체적으로 살펴보자.

- 컴포넌트의 크기나 위치를 개발자가 결정하고자 하는 경우
- 게임 프로그램과 같이 시간이나 마우스/키보드의 입력에 따라 컴포넌트들의 위치와 크기가 수시로 변하는 경우
- 여러 컴포넌트들이 겹치는 효과를 연출하고자 하는 경우

이런 경우, 컨테이너의 배치관리자를 없애고 응용프로그램에서 직접 컴포넌트의 위치와 크기를 마음대로 설정할 수 있어야 한다.

컨테이너의 배치관리자 제거

setLayout()
배치관리자 제거

Container의 setLayout() 메소드를 이용하여 컨테이너의 배치관리자를 제거할 수 있다.

```
container.setLayout(null); // container의 배치관리자 제거
```

다음 코드는 JPanel에 현재 부착된 배치관리자를 삭제한다.

```
JPanel p = new JPanel();
p.setLayout(null);
```

그 결과 컨테이너 p에는 배치관리자가 없으므로 부착되는 컴포넌트의 크기나 위치가 자동 조절되지 않는다. 다시 말해 배치가 일어나지 않는다. 그러므로 컨테이너 p 안에 삽입된 컴포넌트들은 모두 0×0 크기로 설정되어 보이지 않는다.

예를 들면, 다음 코드에서 2개의 버튼은 화면에 전혀 보이지 않는다.

```
JPanel p = new JPanel();
p.setLayout(null);

// 배치관리자가 없으므로 아래 2개의 버튼은 배치되지 않는다.
p.add(new JButton("click")); // 폭과 높이가 0인 상태로 화면에 보이지 않는다.
p.add(new JButton("me!")); // 폭과 높이가 0인 상태로 화면에 보이지 않는다.
```

컴포넌트의 절대 위치와 절대 크기 설정

절대 위치와 크기

배치관리자가 없는 컨테이너에 컴포넌트를 삽입할 때는 응용프로그램에서 컴포넌트의 크기와 위치를 직접 설정하여야 한다. 컴포넌트의 절대 위치와 크기를 설정하기 위해서는 java.awt.Component 클래스의 다음 메소드를 이용한다.

```
void setSize(int width, int height) 컴포넌트를 width x height 크기로 설정
void setLocation(int x, int y) 컴포넌트의 왼쪽 상단 모서리 좌표를 (x, y)로 설정
void setBounds(int x, int y, int width, int height) 크기와 위치 동시 설정
```

예를 들어보자. 다음 코드는 버튼을 100×40 크기로 설정하고 JPanel의 (50, 50) 위치에 삽입한다.

```
Panel p = new JPanel();
p.setLayout(null); // 패널 p의 배치관리자 제거

JButton clickButton = new JButton("Click");
clickButton.setSize(100, 40); // 버튼의 크기를 100×40으로 지정
clickButton.setLocation(50, 50); // 버튼의 위치를 (50, 50)으로 지정
p.add(clickButton); // 패널에 버튼 부착. 패널 내 (50, 50) 위치에 100×40 크기의 버튼 출력
```

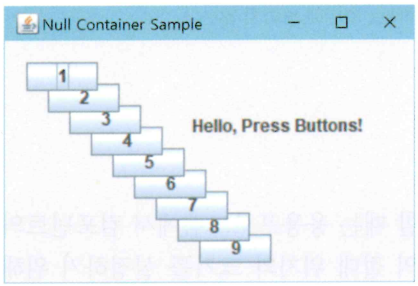
겹치도록 배치

CardLayout을 제외한 나머지 배치관리자는 컴포넌트들이 서로 겹치지 않도록 배치한다. 그러나 배치관리자가 없는 컨테이너를 만들면 컴포넌트를 절대 위치에 절대 크기로 배치할 수 있기 때문에 컴포넌트들이 서로 겹치도록 배치할 수 있다.

예제 9-6 배치관리자 없는 컨테이너에 컴포넌트를 절대 위치와 절대 크기로 지정

다음 그림과 같이 컨텐트팬에 배치관리자를 삭제하고 9개의 버튼과 하나의 문자열을 출력하는 프로그램을 작성하라.

"Hello, Press Buttons!" 문자열은 JLabel 컴포넌트를 이용하고, 그 위치를 (130, 50)에, 크기는 200×20으로 한다. 버튼의 크기는 50×20으로 한다.




```
1  import javax.swing.*;
2  import java.awt.*;
3
4  public class NullContainerEx extends JFrame {
5     public NullContainerEx() {
6        setTitle("Null Container Sample"); // 프레임의 타이틀 달기
7        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); // 프레임 윈도우를 닫으면
                                                          프로그램 종료
8        Container c = getContentPane();
9        c.setLayout(null); // 컨텐트팬의 배치관리자 제거
10
11       // JLabel 컴포넌트 생성하고 위치와 크기를 직접 지정한다.
12       JLabel la = new JLabel("Hello, Press Buttons!");
13       la.setLocation(130, 50); // la를 (130,50) 위치로 지정
14       la.setSize(200, 20); // la를 200x20 크기로 지정
15       c.add(la); // la를 컨텐트팬에 부착
16
17       // 9개의 버튼 컴포넌트를 생성하고 모두 동일한 크기로 설정한다.
18       // 위치는 서로 다르게 설정
19       for(int i=1; i<=9; i++) {
20          JButton b = new JButton(Integer.toString(i)); // 버튼 생성
21          b.setLocation(i*15, i*15); // 버튼의 위치 설정
22          b.setSize(50, 20); // 9 개의 버튼 크기는 동일하게 50x20
23          c.add(b); // 버튼을 컨텐트팬에 부착
24       }
25
26       setSize(300, 200);
```

컨텐트팬의 배치관리자를 제거하여 직접 컴포넌트의 위치와 크기를 설정할 수 있다.

Integer.toString(i)는 정수 i를 문자열로 변환한다.

```
27        setVisible(true);
28     }
29     public static void main(String[] args) {
30        new NullContainerEx();
31     }
32  }
```

CHECK TIME



1. 컨테이너가 컴포넌트와 다른 점은 무엇인가?
2. container에 컴포넌트 사이의 수평, 수직 간격을 각각 10, 20 픽셀로 배치하는 FlowLayout 배치관리자를 지정하는 코드를 작성하라.
3. 컨테이너 container에 설치된 배치관리자를 제거하는 한 줄의 코드를 작성하라.
4. 배치관리자가 없는 컨테이너에 컴포넌트를 배치할 때 응용프로그램에서 반드시 컴포넌트의 크기와 위치를 지정해야 하는 이유는 무엇인가?



SUMMARY

요약

- GUI란 Graphical User Interface의 약자로서 이미지 혹은 그래픽 등을 이용하여 메뉴 등을 포함한 화면을 구성하고, 키보드, 마우스 등의 편리한 입력 도구를 이용하여 사용자의 입력을 편리하게 하도록 작성된 사용자 인터페이스이다.
- 자바 언어에서 GUI를 구성하기 위한 기본 패키지는 AWT와 스윙(Swing)의 두 가지가 있으며, 스윙은 AWT와는 달리 light weight component로서 운영체제의 자원에 의존하지 않고 작동하도록 구현되었다.
- 스윙은 AWT 컴포넌트 외에 많은 고급 컴포넌트를 가지고 있기 때문에 GUI 응용프로그램을 작성하는 데 우수하다.
- AWT 컴포넌트는 java.awt 패키지에 스윙 컴포넌트는 javax.swing 패키지에 구현되어 있다.
- 컨테이너는 여러 컴포넌트를 포함할 수 있는 특별한 GUI 컴포넌트이다. 컨테이너도 다른 컨테이너에 포함될 수 있는 컴포넌트이다.
- 스윙에서 프레임을 구성하기 위해서는 JFrame을 상속받은 새로운 클래스를 만들어 사용하는 것이 좋다.
- JFrame은 내부에 컨텐트팬을 가지고 있으며, JFrame에 컴포넌트를 붙이기 위해서는 컨텐트팬에 컴포넌트를 부착하여야 한다.
- 스윙에서 JFrame의 윈도우 닫기 메뉴를 선택하여 프레임 윈도우를 닫을 수는 있어도 응용프로그램을 종료시킬 수는 없다. 응용프로그램을 종료시키기 위해서는 다음의 코드가 필요하다.

```
JFrame frame;
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

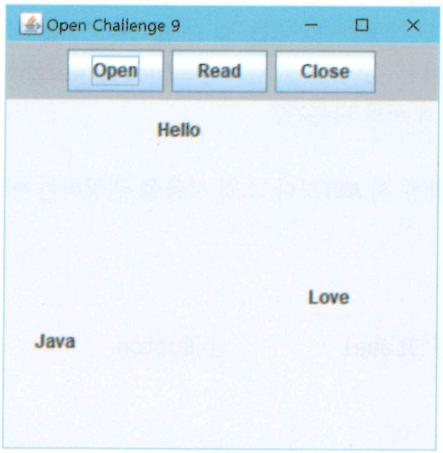
- 컨테이너는 내부에 반드시 하나의 배치관리자(Layout Manager)를 두고 있으며, 이 배치관리자는 컨테이너에 부착되는 모든 컴포넌트들의 위치와 크기를 상대적으로 설정한다.
- FlowLayout 배치관리자는 컴포넌트가 들어오면 순서대로 왼쪽에서 오른쪽으로 배치하며, 더 이상 오른쪽에 배치할 공간이 없으면 다시 아래로 내려와서 왼쪽으로 오른쪽으로 배치한다.
- BorderLayout 배치관리자는 컨테이너의 공간을 동(EAST), 서(WEST), 남(SOUTH), 북(NORTH), 중앙(CENTER)의 5개 영역으로 나누고 응용프로그램에서 지정한 영역에 컴포넌트를 배치한다.
- GridLayout 배치관리자는 컨테이너의 공간을 응용프로그램에서 설정한 2차원 그리드로 나누고, 컴포넌트가 들어오는 순서대로 좌에서 우로 배치하고, 다시 위에서 아래로 배치한다.
- CardLayout 배치관리자는 카드를 쌓아놓은 듯이 컴포넌트를 포개어 배치한다.
- 컨테이너에서 배치관리자를 삭제하고 응용프로그램에서 컴포넌트의 절대 위치와 크기를 설정하여 컴포넌트를 원하는 위치와 크기로 출력되게 할 수 있다.

Open Challenge 간단한 스윙 응용프로그램 만들기



목적
스윙 프레임 만들기, JPa
상속받기, 여러 패널을 이
여 컴포넌트 붙이기 연습

간단한 스윙 응용프로그램을 만들어보자. 그림과 같이 프레임의 컨텐트팬에 BorderLayout 배치관리자를 설치하고, NORTH 영역과 CENTER 영역에는 JPanel을 상속받은 패널을 만들어 붙인다. NORTH 영역의 패널에는 3개의 버튼을 컴포넌트를, CENTER 영역의 패널에는 3개의 JLabel을 이용하여 각각 "Hello", "Java", "Love"를 출력한다. 문자열이 출력되는 위치는 독자가 정하라. 난이도 5



힌트 Hint

- 컨텐트팬은 디폴트로 BorderLayout 배치관리자를 가지고 있다.
- 컨텐트팬의 NORTH와 CENTER 영역에 다음과 같이 JPanel을 상속받는 패널을 만들어 붙인다.

```
class NorthPanel extends JPanel { // NORTH에 붙일 패널
}
class CenterPanel extends JPanel { // CENTER에 붙일 패널
}
```

- NorthPanel에는 FlowLayout 배치관리자를 이용하여 3개의 버튼을 붙이면 된다.
- CenterPanel에는 배치관리자를 삭제하고, 3개의 JLabel 컴포넌트를 사용하여 문자열을 붙인다. 그리고 setSize()와 setLocation()을 이용하여 위치와 크기를 지정한다.
- NorthPanel 패널의 배경색을 밝은 회색(light gray)으로 하려면 다음과 같이 하면 된다.

```
class NorthPanel extends JPanel {
    public NorthPanel() {
        setBackground(Color.LIGHT_GRAY);
    }
}
```

연습문제

EXERCISE



이론문제

• 홀수 문제는 정답이 공개됩니다

1. 자바에서는 AWT 컴포넌트와 스윙 컴포넌트를 제공한다. 이들 중 어떤 것이 경량 컴포넌트(light weight component)이고, 어떤 것이 중량 컴포넌트(heavy weigh component)인가? 그리고 이들은 어떤 점에서 서로 다른가?

2. 데스크톱에서 실행되는 GUI 응용프로그램 작성 시 AWT보다 스윙 사용을 권장하는 유는 무엇인가?

3. 다음 중 스윙 컴포넌트가 아닌 것은?
   ① JPanel ② JTextField ③ JLabel ④ Button

4. 다음 중 컴포넌트는?
   ① JFrame ② Font ③ Color ④ Graphics

5. 컴포넌트와 컨테이너에 대해 잘못 말한 것은?
   ① 컨테이너는 컴포넌트를 담을 수 있는 컴포넌트이다.
   ② JPanel은 컨테이너로서 여러 개의 JButton 컴포넌트를 가질 수 있다.
   ③ JFrame은 최상위 컨테이너이다.
   ④ 컴포넌트들은 컨테이너 없이도 출력된다.

6. 배치관리자에 대해 잘못 말한 것을 모두 찾아라.
   ① 배치관리자를 가지는 것은 컨테이너만의 고유 기능이며, 컴포넌트는 가질 수 없다
   ② 배치관리자는 자신이 소속된 컨테이너의 크기를 조절한다.
   ③ 컨테이너가 생성될 때 배치관리자가 없는 상태이므로 배치관리자를 설정해 한다.
   ④ 한 컨테이너는 여러 개의 배치관리자를 가질 수 있다.
   ⑤ 배치관리자의 기능은 컨테이너에 포함된 컴포넌트들의 위치와 크기를 설정하는 이다.
   ⑥ 개발자는 자바에서 주어진 것 외의 새로운 배치관리자를 만들어 사용할 수 있다.
   ⑦ 컨테이너가 배치관리자를 가지지 않도록 할 수 없다.

7. 다음은 200×300 크기의 스윙 프레임을 만든 코드이다. 빈칸에 필요한 코드를 채워라.

```
_____ // 이곳에 필요한 import 문을 삽입하라.
public class MyFrame _____ {
    public MyFrame() {
        _____ // 프레임 타이틀로 "hello" 문자열 출력
        _____ // 프레임 크기를 200x300으로 설정
        _____ // 프레임 화면 출력
    }
    public static void main(String [] args) {
        MyFrame frame = _____
    }
}
```

8. 다음은 스윙 프레임을 작성한 코드이다. 빈칸을 채워라.

```
_____ // 이곳에 필요한 import 문을 삽입하라.
_____ // 이곳에 필요한 import 문을 삽입하라.
public class MyFrame _____ {
    public MyFrame() {
        _____ ; // 컨텐트팬 알아내기
        _____ ; // 컨텐트팬에 FlowLayout 배치관리자 설정
        _____ ; // 컨텐트팬 배경색을 노란색으로 설정
        _____ ; // 컨텐트팬에 "click" 버튼 달기
        setSize(300,300);
        setVisible(true);
    }
}
```

9. 다음 지시에 따라 컨테이너 c에 배치관리자를 설정하는 코드를 작성하라.

```
Container c;
c. _____
```

(1) 컴포넌트 사이의 수평 수직 간격이 각각 3, 4픽셀인 BorderLayout

(2) 컴포넌트 사이의 수평 수직 간격이 각각 5, 6픽셀이고 오른쪽 정렬하는 FlowLayout

(3) 컴포넌트 사이의 수평 수직 간격이 각각 7, 8픽셀이고 행수 5, 열수 2인 GridLayout

(4) 현재 등록된 배치관리자 삭제

10. 버튼 컴포넌트를 절대 위치에 배치하고자 한다. 주석을 참고하여 빈칸을 완성하라.

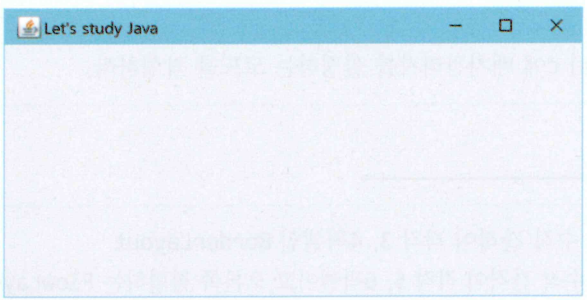
```
_____ // 이곳에 필요한 import 문을 삽입하라.
_____ // 이곳에 필요한 import 문을 삽입하라.
public class MyFrame _____ {
    public MyFrame() {
        _____ ; // 컨텐트팬 알아내기
        _____ ; // 컨텐트팬에 배치관리자 제거
        _____ ; // "click" 문자열의 버튼 컴포넌트 생성
        _____ ; // 버튼의 크기를 100x30으로 설정
        _____ ; // 버튼의 위치를 (50, 70)으로 설정
        _____ ; // 컨텐트팬에 버튼 달기
        setSize(300,300);
        setVisible(true);
    }
    public static void main(String [] args) {
        new MyFrame();
    }
}
```

실습문제

• 홀수 문제는 정답이 공개됩니다.

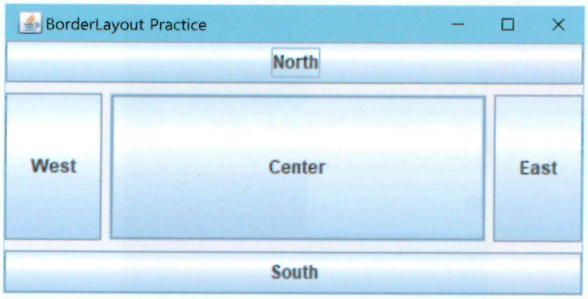
목적 스윙 프레임 만들기

1. 다음 그림과 같이 "Let's study Java"라는 문자열을 타이틀로 가지고 프레임의 크기가 400×200인 스윙 프로그램을 작성하라. 난이도 3



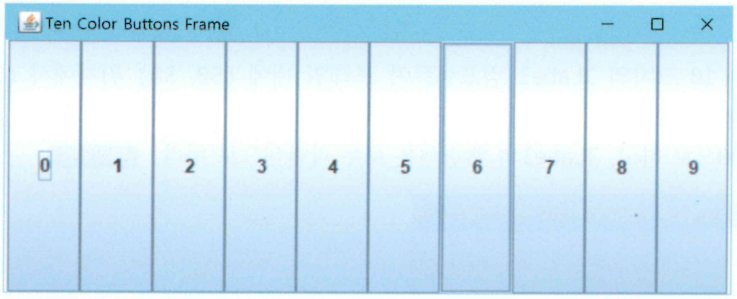
2. BorderLayout을 사용하여 컴포넌트 사이의 수평 수직 간격이 각각 5픽셀, 7픽셀이 되도록 스윙 응용프로그램을 작성하라. 난이도 3

목적 컨텐트팬과 BorderLayout 활용



3. GridLayout을 사용하여 다음 그림과 같이 한 줄에 10개의 버튼을 동일한 크기로 배치하는 스윙 프로그램을 작성하라. 난이도 3

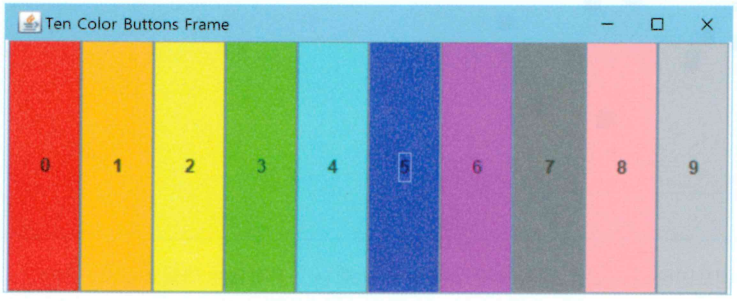
목적 GridLayout 활용



4. 문제 3을 수정하여 다음 결과와 같이 각 버튼의 배경색을 서로 다르게 설정하라.

난이도 4

목적 GridLayout과 JL 컴포넌트 색 활용

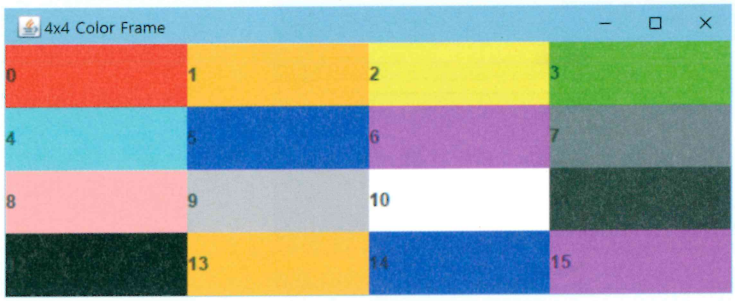


힌트 Hint

컴포넌트의 배경색을 노란색으로 설정하려면 comp.setBackground(Color.YELLOW);로 하면 된다.

목적 GridLayout과 JLabel 컴포넌트 활용

5. GridLayout을 이용하여 다음 그림과 같이 Color.WHITE, Color.GRAY, Color.RED 등 16개의 색을 배경색으로 하는 4×4 바둑판을 구성하라. 난이도 5

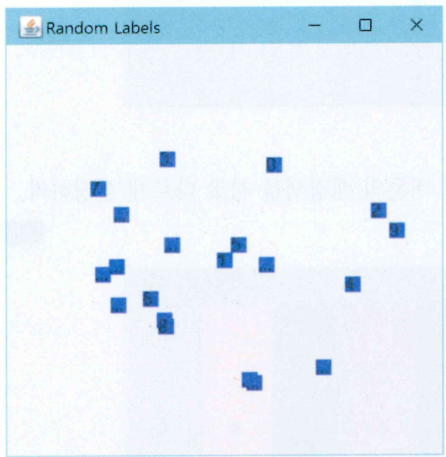


힌트 Hint

16개의 JLabel 컴포넌트를 생성하고 각 레이블 컴포넌트의 배경색을 칠한 다음 하나씩 GridLayout을 가진 컨테이너에 붙이면 된다.

목적 배치관리자 삭제와 임의의 위치에 컴포넌트 붙이기

6. 20개의 10×10 크기의 JLabel 컴포넌트가 프레임 내에 (50, 50) 위치에서 (250, 250) 영역에서 랜덤한 위치에 출력되도록 스윙 프로그램을 작성하라. 프레임의 크기를 300×300으로 하고, JLabel의 배경색은 모두 파란색으로 하라. 난이도 6



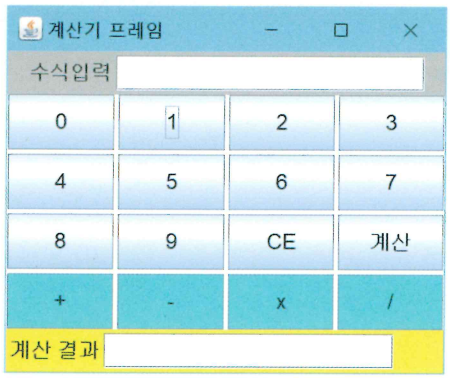
힌트 Hint

JLabel 컴포넌트의 위치를 랜덤하게 설정하기 위해 (x, y) 좌표는 다음과 같이 구한다.

```
int x = (int)(Math.random()*200) + 50; // 50~250
int y = (int)(Math.random()*200) + 50; // 50~250
label.setLocation(x,y); // label을 (x,y)에 배치
label.setSize(10,10); // label 크기를 10×10으로 설정
label.setOpaque(true); // label에 배경색이 보이게 함
```

7. 다음과 같은 GUI 모양을 가진 스윙 프레임을 작성하라. Open Challenge의 힌트나 정답을 참고하라. 버튼은 JButton, 텍스트는 JLabel, 입력창은 JTextField를 사용하면 된다. 난이도 7

목적 여러 개의 컴포넌트러 개의 패널을 가진 스윙임 만들기

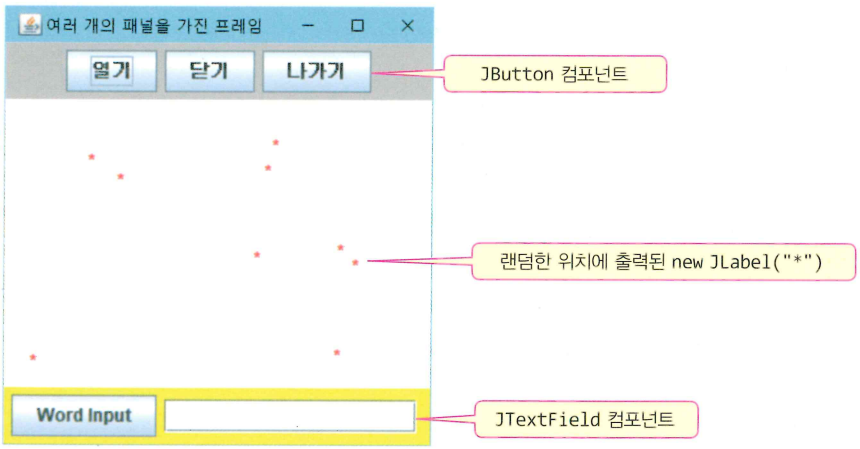


힌트 Hint

컨텐트팬의 NORTH, CENTER, SOUTH에 각각 JPanel을 상속받은 패널을 붙이고 그 곳에 버튼이나 JLabel을 붙이면 된다.

8. 다음과 같은 GUI 모양을 가진 스윙 프레임을 작성하라. Open Challenge의 힌트나 정답을 참고하라. 10개의 '*' 문자는 10개의 JLabel을 이용하여 랜덤한 위치에 출력하라. 난이도 7

목적 여러 개의 컴포넌트러 개의 패널을 가진 스윙임 만들기



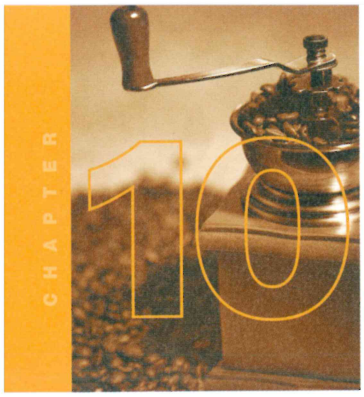


10

자바의 이벤트 처리

Objectives

- 자바의 이벤트 기반 프로그래밍의 구조를 이해한다.
- 자바에서 이벤트의 종류를 알고 이벤트 발생에 대해 살펴본다.
- 자바에서 제공하는 이벤트 리스너 종류를 알고 리스너 메소드를 이해한다.
- 자바에서 이벤트 리스너 작성 방법을 안다.
- 이벤트 객체에 대해 이해하고 이벤트 객체 메소드를 활용한다.
- 어댑터 클래스의 필요성을 알고 어댑터 클래스를 활용한다.
- Key 이벤트와 KeyListener 인터페이스를 이해하고 키 이벤트 리스너를 작성한다.
- Mouse 이벤트, MouseListener 인터페이스, MouseMotionListener 인터페이스, MouseWheelListener를 이해하고 마우스 이벤트 리스너를 작성할 수 있다.



JAVA PROGRAMMING

자바의 이벤트 처리

10.1 이벤트 기반 프로그래밍(Event Driven Programming)

이벤트 기반 프로그래밍

이벤트 기반 프로그래밍
이벤트
이벤트 리스너

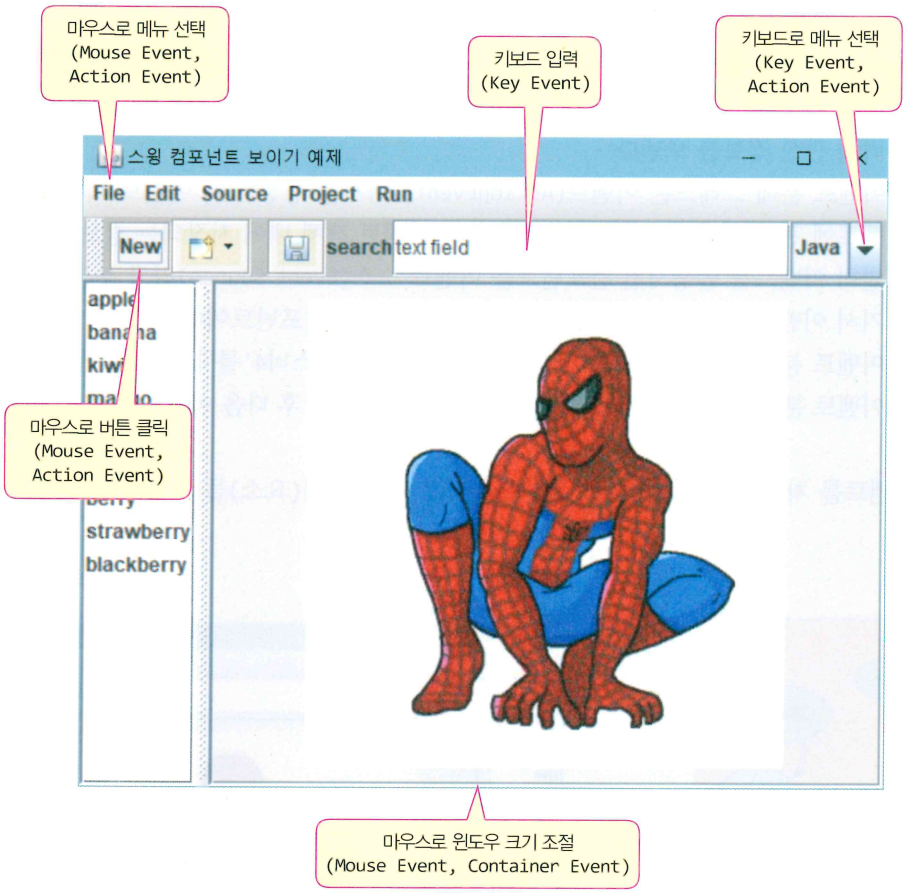
이벤트 기반 프로그래밍은 이벤트의 발생에 의해 프로그램 실행 흐름이 결정되는 방식의 프로그래밍 패러다임이다. 이벤트는 키 입력, 마우스 클릭, 마우스 드래그 등 사용자의 액션이나, 센서 등 외부 장치로부터의 입력, 네트워크를 통한 데이터 수신, 다른 스레드나 프로그램으로부터의 메시지 수신 등에 의해 발생한다. 이벤트 기반 프로그래밍과 대조되는 개념은 프로그램 작성자에 의해 프로그램의 흐름이 결정되는 방식이다. 이벤트 기반 응용프로그램은 각 이벤트를 처리하는 이벤트 리스너(event listener)들을 보유하며, 이벤트가 발생할 때마다 리스너가 실행된다. 그러므로 프로그램 내의 어떤 코드가 언제 실행될지는 전적으로 이벤트의 발생에 달려있다.

GUI 응용프로그램은 대부분 이벤트 기반 프로그래밍 기법으로 작성한다. Visual C++의 MFC, C#, Visual Basic, X window, Android는 모두 이벤트 기반 GUI 응용프로그램을 작성할 수 있는 라이브러리를 제공한다. 자바는 AWT나 스윙을 이용하여 어떤 언어보다 쉽고 훌륭한 이벤트 기반 GUI 프로그램을 작성할 수 있다.

[그림 10-1]은 스윙 기반 GUI 응용프로그램에서 이벤트 발생 사례를 보여준다. 키보드로 메뉴 선택, 텍스트 필드 창에 문자열 입력, 마우스로 메뉴 클릭, New 버튼 클릭, 마우스로 프레임 윈도우의 크기 조절, 체크박스나 라디오버튼에 마우스나 키보드로 체크 혹은 해제하는 등의 행위는 사실상 모두 응용프로그램에게 이벤트를 발생시킨다.

자바의 이벤트 기반 GUI 응용프로그램 구조

[그림 10-2]는 스윙 응용프로그램에서 발생한 이벤트가 처리되는 과정을 보여준다. 그림의 오른쪽은 스윙 응용프로그램의 프레임 윈도우와 GUI 컴포넌트들이 출력된 모습을 보여주며, 왼쪽은 응용프로그램의 내부 구조를 보여준다. 예를 들면, 오른쪽 화면의 New 버튼은 왼쪽의 JButton 컴포넌트가 출력된 이미지이며, 오른쪽 화면의



[그림 10-1] 스윙 응용프로그램에서 이벤트 발생 사례

'apple', 'banana', 'kiwi' 등으로 구성된 리스트는 왼쪽의 JList 컴포넌트가 출력된 이미지이다. 사실 사용자들은 오른쪽 화면을 보고 있지만 응용프로그램 내부에서는 왼쪽의 객체들이 활동하고 있다. 사용자들은 오른쪽 화면을 보면서 마우스를 움직이거나 클릭하지만, 이러한 움직임과 클릭 정보는 모두 이벤트 형태로 응용프로그램 내에 생성된 컴포넌트 객체에게 전달된다.

이벤트 리스너

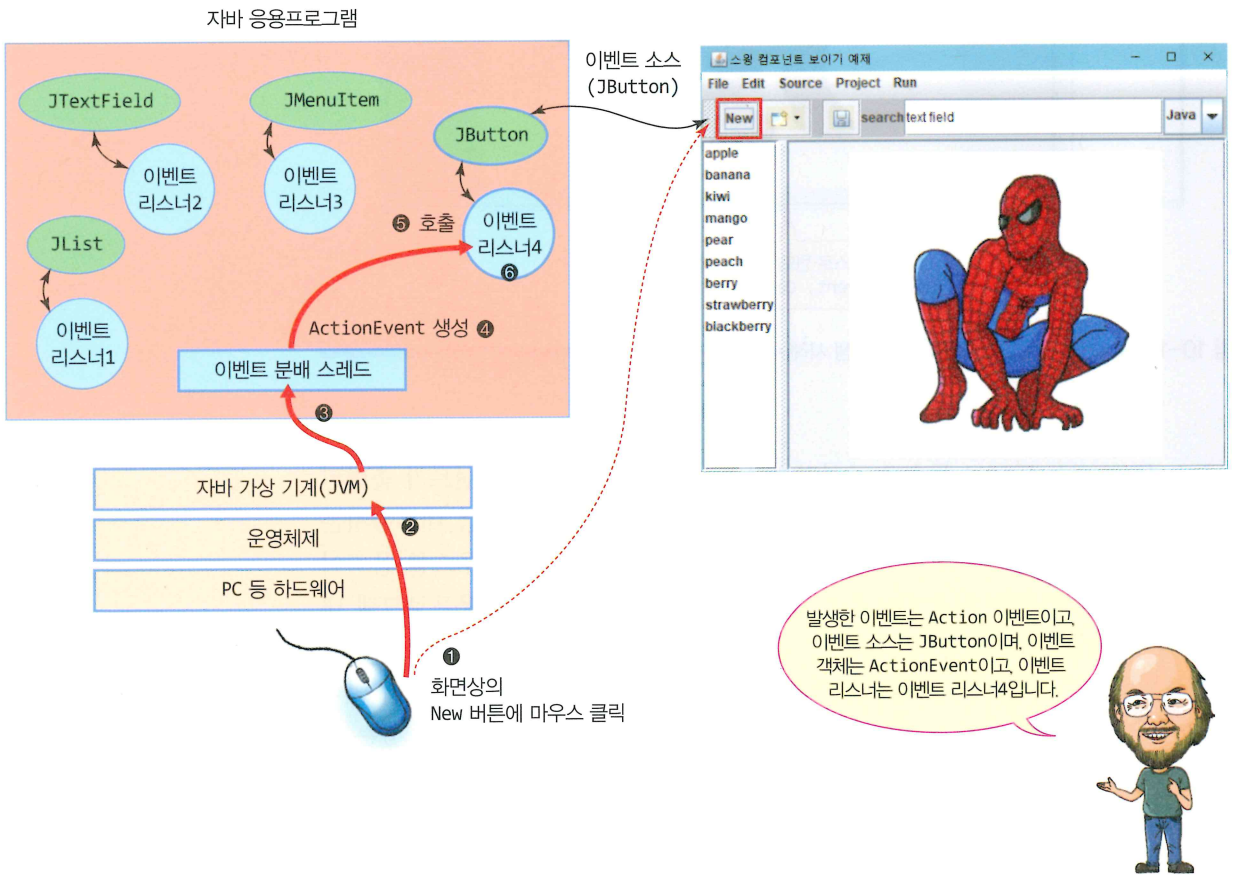
이벤트 리스너는 이벤트를 처리하는 프로그램 코드로서 컴포넌트에 연결되어 있어야 작동된다. 왼쪽 그림에서 각 컴포넌트들이 이벤트 리스너를 하나씩 가지고 있는 것을 볼 수 있다.

그림에서 'New' 버튼이 클릭되어 이벤트가 발생하고 처리되는 과정을 자세히 묘사하면 다음과 같다.

이벤트 분배 스레드
이벤트 객체
이벤트 소스
이벤트 리스너

① 사용자가 마우스로 화면의 New 버튼을 클릭한다.
② 버튼 클릭은 운영체제의 마우스 드라이버를 거쳐 자바 가상 기계에 전달된다.
③ 자바 가상 기계는 이벤트 분배 스레드(Event Dispatch Thread)에게 마우스 클릭에 관한 정보를 보낸다.
④ 이벤트 분배 스레드는 이벤트(ActionEvent) 객체를 생성한다. 이벤트 객체는 이벤트에 관한 여러 정보를 담은 객체이다. 이벤트 객체 내에 저장되는 정보 중, 특별히 이벤트를 발생시킨 컴포넌트를 이벤트 소스(Event Source)라고 부른다. 여기서 이벤트 소스는 'New' 글자가 새겨진 JButton 컴포넌트이다.
⑤ 이벤트 분배 스레드는 JButton에 연결된 '이벤트 리스너4'를 찾아 호출한다.
⑥ 이벤트 분배 스레드는 '이벤트 리스너4'로부터 리턴한 후 다음 이벤트를 기다린다.

이벤트를 처리하는 과정에서 등장하는 이벤트 관련 용어(요소)들을 정리해보자.



[그림 10-2] 자바 스윙 GUI 응용프로그램의 이벤트 처리 과정

● 이벤트 소스

이벤트를 발생시킨 GUI 컴포넌트이다.

● 이벤트 객체

발생한 이벤트에 대한 정보(이벤트 종류, 이벤트 소스, 화면 좌표, 마우스 버튼 종류, 눌러진 키)를 담는 객체로서, 이벤트에 따라 서로 다른 정보가 저장된다.

● 이벤트 리스너(Event Listener)

이벤트를 처리하는 코드로서 컴포넌트에 등록되어야 작동 가능하다.

● 이벤트 분배 스레드(Event Dispatch Thread)

이벤트 기반 프로그래밍의 핵심 요소로서 무한 루프를 실행하는 스레드이다. 자바 가상 기계로부터 이벤트의 발생을 통지받아, 이벤트 소스와 이벤트 종류를 결정하고 이에 따라 적절한 이벤트 객체를 생성하고 이벤트 리스너를 찾아 호출한다.



잠깐!

자바에서 이벤트는 이벤트 분배 스레드에 의해 하나씩 도착하는 순서대로 처리된다. 하나의 이벤트에 대한 처리가 완전히 종료된 후 다음 이벤트가 처리되므로, 한 이벤트 리스너의 처리 시간이 오래 걸리거나 리스너 내에서 잠을 자는 등 시간 지연이 과다하게 되면 다음 이벤트의 처리가 늦어지게 된다. 이로 인해 마우스의 움직임이나 키 입력이 지연되는 일이 일어날 수 있기 때문에 이벤트 리스너는 가능하면 짧게 작성하여야 하며, 길어지게 되는 경우 따로 스레드를 만들어 해결하여야 한다. 스레드는 13장에서 자세히 설명한다.



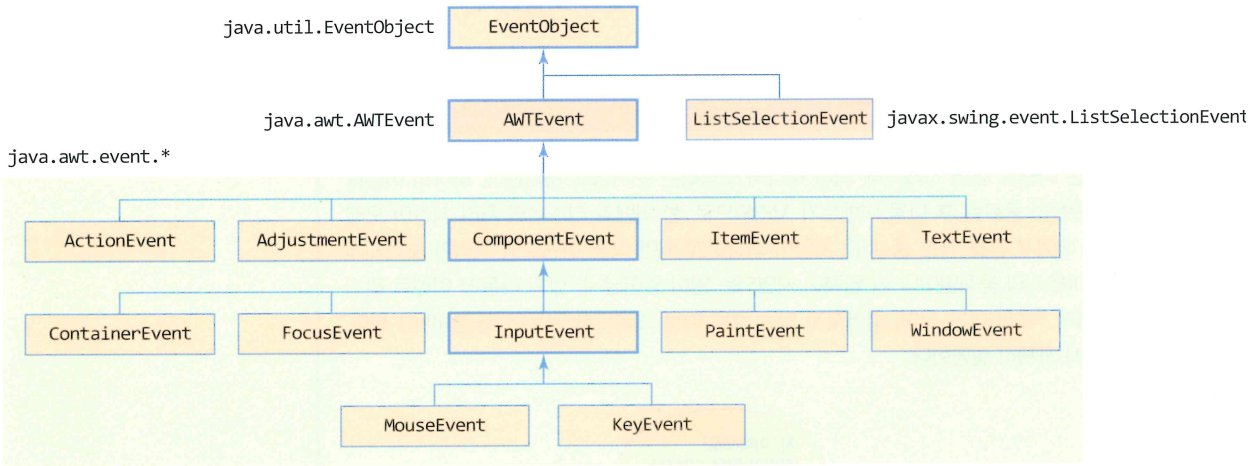
10.2 이벤트 객체

이벤트 객체란?

이벤트 객체

이벤트 객체는 현재 발생한 이벤트에 관한 정보를 가진 객체이며, 이벤트 리스너에게 전달된다. 이벤트 객체의 종류는 [그림 10-3]과 같이 다양하며, 모든 이벤트 객체들은 java.util.EventObject 클래스를 상속받으며, java.awt.event와 javax.swing.event 패키지에 구현되어 있다. 응용프로그램은 이벤트를 처리하기 위해 반드시 다음 import 문을 포함해야 한다.

```
import java.awt.event.*; // 이벤트 처리가 필요한 모든 소스에 포함
import javax.swing.event.*; // 스윙 이벤트를 다루는 경우에 추가 포함
```

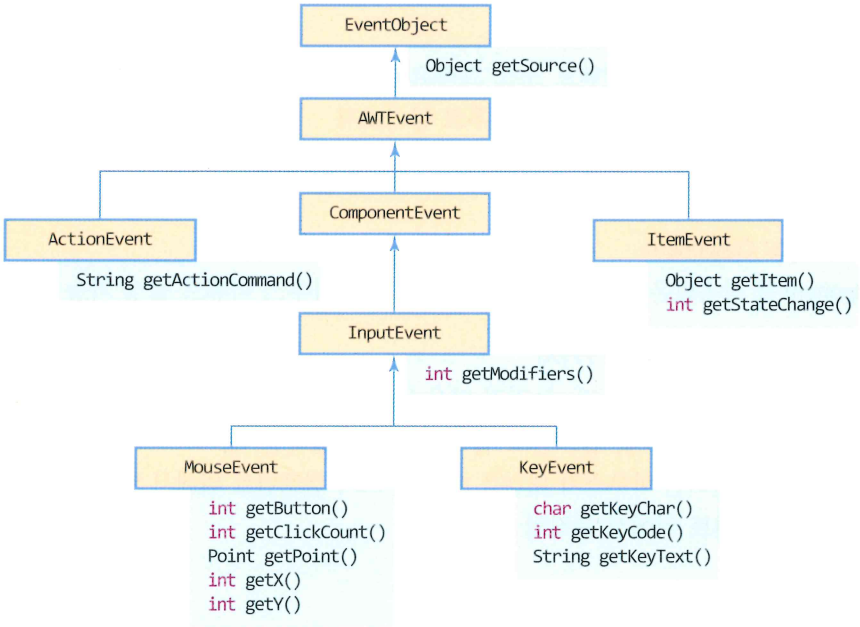


[그림 10-3] 이벤트 객체의 종류 및 상속 관계

이벤트 객체 정보

이벤트 객체는 발생한 이벤트에 따라 조금씩 다르지만, 대략 다음 정보를 담는다.

- 이벤트 종류
- 이벤트 소스
- 화면 내 이벤트가 발생한 마우스 좌표
- 컴포넌트 내 이벤트가 발생한 마우스 좌표
- 버튼이나 메뉴 아이템에 이벤트가 발생한 경우 버튼이나 메뉴 아이템의 문자열
- 클릭된 마우스 버튼 번호



[그림 10-4] 이벤트 객체의 메소드

- 마우스의 클릭 횟수
- 키가 눌러졌다면 키의 코드 값과 문자 값
- 체크박스, 라디오버튼에 이벤트가 발생하였다면 체크 상태

이벤트 객체는 메소드를 통해 이벤트 정보를 제공한다. [그림 10-4]에 이벤트 객체가 가진 일부 메소드를 소개한다. MouseEvent 객체의 메소드 예를 살펴보자. 눌러진 마우스 버튼 번호(getButton()), 마우스 클릭 횟수(getClickCount()), 마우스 포인터 좌표(getPoint(), getX(), getY()) 등을 알려주는 메소드가 있다.

이 중 모든 이벤트가 객체에 있고, 이벤트 리스너에서 가장 많이 사용되는 getSource() 메소드를 알아보자.

● Object getSource()

getSource()는 현재 발생한 이벤트의 소스 컴포넌트의 레퍼런스를 리턴한다. getSource()의 리턴 타입이 Object이므로 캐스팅해서 사용하는데, 예를 들어 버튼이 눌러진 경우라면 다음과 같이 캐스팅하면 된다.

getSource()

```
// event는 이벤트 객체
JButton b = (JButton)event.getSource(); // b는 이벤트가 발생한 버튼의 레퍼런스
```

이벤트 객체와 이벤트 소스

〈표 10-1〉에는 이벤트 객체와 이벤트 소스 그리고 이벤트가 발생하는 경우를 요약하
다. 이 표를 읽는 방법을 MouseEvent의 경우로 알아보자.

- 이벤트 객체: MouseEvent
- 이벤트 소스: Component

  Component를 상속받은 모든 스윙 컴포넌트에 대해 마우스 이벤트 발생 가능
- 이벤트가 발생하는 경우: 총 7가지

  마우스 버튼이 눌러지는 순간, 눌러진 마우스 버튼이 떼어질 때,
  마우스 버튼이 클릭될 때, 컴포넌트 위에 마우스가 올라갈 때,
  컴포넌트 위에 올라간 마우스가 내려올 때, 마우스가 컴포넌트 위에서 드래그될 때
  마우스가 컴포넌트 위에서 움직일 때

〈표 10-1〉
이벤트 객체와 이벤트 소스
그리고 이벤트가 발생하는 경우

이벤트 객체	이벤트 소스	이벤트가 발생하는 경우
ActionEvent	JButton	마우스나 <Enter> 키로 버튼 선택
	JMenuItem	메뉴 아이템 선택
	JTextField	텍스트 입력 중 <Enter> 키 입력
ItemEvent	JCheckBox	체크박스의 선택 혹은 해제
	JRadioButton	라디오버튼의 선택 상태가 변할 때
	JCheckBoxMenuItem	체크박스 메뉴 아이템의 선택 혹은 해제
ListSelectionEvent	JList	리스트에서 선택된 아이템이 변경될 때
KeyEvent	Component	키가 눌러지거나 눌러진 키가 떼어질 때
MouseEvent	Component	마우스 버튼이 눌러지거나 떼어질 때, 마우스 버튼이 클릭될 때, 컴포넌트 위에 마우스가 올라갈 때, 올라간 마우스가 내려올 때, 마우스가 드래그될 때, 마우스가 단순히 움직일 때
FocusEvent	Component	컴포넌트가 포커스를 받거나 잃을 때
WindowEvent	Window	Window를 상속받는 모든 컴포넌트에 대해 윈도우 활성화, 비활성화, 아이콘화, 아이콘에서 복구, 윈도우 열기, 윈도우 닫기, 윈도우 종료
AdjustmentEvent	JScrollBar	스크롤바를 움직일 때
ComponentEvent	Component	컴포넌트가 사라지거나, 나타나거나, 이동, 크기 변경 시
ContainerEvent	Container	Container에 컴포넌트의 추가 혹은 삭제

10.3 이벤트 리스너

리스너 인터페이스

이벤트 리스너

ActionListener

이벤트 리스너란 이벤트를 처리하는 자바 프로그램 코드로서 클래스로 만든다. JDK는 〈표 10-2〉와 같은 이벤트 리스너 인터페이스(interface)를 제공하며, 개발자는 이 인터페이스를 상속받고 추상 메소드를 모두 구현하여 이벤트 리스너를 작성한다.

ActionListener 인터페이스의 경우를 예로 들어보자. ActionListener 인터페이스는 자바 패키지에 다음과 같이 선언되어 있다.

```
interface ActionListener {
    public void actionPerformed(ActionEvent e); // Action 이벤트 발생 처리
}
```

actionPerformed(Actio
Event e)

버튼을 누르는 Action 이벤트가 발생하면 actionPerformed(ActionEvent e) 메소드가 호출되고, 이때 ActionEvent 객체가 인자로 전달된다.

MouseListener

MouseEvent

또한 MouseListener 인터페이스는 다음과 같이 5개의 메소드를 가지고 있으며, 각 메소드는 마우스의 조작에 따라 발생하는 이벤트를 처리한다. 메소드에는 MouseEvent 객체가 인자로 전달된다.

```
interface MouseListener {
    public void mousePressed(MouseEvent e);  // 마우스 버튼이 눌러지는 순간
    public void mouseReleased(MouseEvent e); // 눌러진 마우스 버튼이 떼어지는 순간
    public void mouseClicked(MouseEvent e);  // 마우스가 클릭되는 순간
    public void mouseEntered(MouseEvent e);  // 마우스가 컴포넌트 위에 올라가는 순간
    public void mouseExited(MouseEvent e);   // 마우스가 컴포넌트 위에서 내려오는 순간
}
```

리스너 인터페이스들은 java.awt.event 패키지에 구현되어 있으므로 이벤트 리스너를 작성하기 위해서는 import java.awt.event.*; 문이 필요하다.



잠깐!

- 개발자는 처리하고자 하는 이벤트에 대해 자바의 리스너 인터페이스를 상속받아 클래스를 작성한 뒤 인터페이스의 모든 메소드를 구현하여야 한다.
- 이벤트가 발생하는 각 경우에 따라 어떤 메소드가 호출되는지 숙지하는 것이 꼭 필요하다.

〈표 10-2〉 이벤트 리스너 인터페이스

이벤트 종류	리스너 인터페이스	리스너의 추상 메소드	메소드가 호출되는 경우
Action	ActionListener	void actionPerformed(ActionEvent)	Action 이벤트가 발생하는 경우
Item	ItemListener	void itemStateChanged(ItemEvent)	Item 이벤트가 발생하는 경우
Key	KeyListener	void keyPressed(KeyEvent)	모든 키에 대해 키가 눌러질 때
		void keyReleased(KeyEvent)	모든 키에 대해 눌러진 키가 떼어질 때
		void keyTyped(KeyEvent)	유니코드 키가 입력될 때
Mouse	MouseListener	void mousePressed(MouseEvent)	마우스 버튼이 눌러질 때
		void mouseReleased(MouseEvent)	눌러진 마우스 버튼이 떼어질 때
		void mouseClicked(MouseEvent)	마우스 버튼이 클릭될 때
		void mouseEntered(MouseEvent)	마우스가 컴포넌트 위에 올라올 때
		void mouseExited(MouseEvent)	컴포넌트 위에 올라온 마우스가 컴포넌트를 벗어날 때
Mouse	MouseMotionListener	void mouseDragged(MouseEvent)	마우스를 컴포넌트 위에서 드래그할 때
		void mouseMoved(MouseEvent)	마우스가 컴포넌트 위에서 움직일 때
Focus	FocusListener	void focusGained(FocusEvent)	컴포넌트가 포커스를 받을 때
		void focusLost(FocusEvent)	컴포넌트가 포커스를 잃을 때
ListSelection	ListSelectionListener	void valueChanged(ListSelectionEvent)	JList에 선택된 아이템이 변경될 때
Window	WindowListener	void windowOpened(WindowEvent)	윈도우가 생성되어 처음으로 보이게 될 때
		void windowClosing(WindowEvent)	윈도우의 시스템 메뉴에서 윈도우 닫기를 시도할 때
		void windowIconfied(WindowEvent)	윈도우가 아이콘화 될 때
		void windowDeiconfied(WindowEvent)	아이콘 상태에서 원래 상태로 복귀할 때
		void windowClosed(WindowEvent)	윈도우가 닫혔을 때
		void windowActivated(WindowEvent)	윈도우가 활성화될 때
		void windowDeactivated(WindowEvent)	윈도우가 비활성화될 때
Adjustment	AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent)	스크롤바를 움직일 때
Component	ComponentListener	void componentHidden(ComponentEvent)	컴포넌트가 보이지 않는 상태로 될 때
		void componentShown(ComponentEvent)	컴포넌트가 보이는 상태로 될 때
		void componentResized(ComponentEvent)	컴포넌트의 크기가 변경될 때
		void componentMoved(ComponentEvent)	컴포넌트의 위치가 변경될 때
Container	ContainerListener	void componentAdded(ContainerEvent)	컴포넌트가 컨테이너에 추가될 때
		void componentRemoved(ContainerEvent)	컴포넌트가 컨테이너에서 삭제될 때

이벤트 리스너 작성 과정

이벤트 리스너는 전형적으로 다음 3단계로 작성한다.

1. 이벤트와 이벤트 리스너 선택 - 목적에 적합한 이벤트와 리스너 인터페이스 선택
2. 이벤트 리스너 클래스 작성 - 리스너 인터페이스를 상속받는 클래스를 작성하고 추상 메소드 모두 구현
3. 이벤트 리스너 등록 - 이벤트를 받을 스윙 컴포넌트에 이벤트 리스너 등록

지금부터 버튼을 클릭하면 버튼 문자열을 "Action"과 "액션"으로 번갈아 바꾸는 이벤트 리스너를 만들어보자(예제 10-1 완성된 코드).

● 이벤트와 이벤트 리스너 선택

〈표 10-1〉과 〈표 10-2〉를 참고하면, 버튼을 클릭(선택)할 때 발생하는 이벤트와 이벤트 리스너 인터페이스, 그리고 이벤트 객체는 다음과 같다.

- 이벤트: Action 이벤트
- 이벤트 리스너: ActionListener
- 이벤트 객체: ActionEvent

● 이벤트 리스너 클래스 작성

ActionListener 인터페이스를 상속받은 MyActionListener 클래스를 선언하고, 한 개뿐인 추상 메소드 actionPerformed(ActionEvent e)를 다음과 같이 작성한다.

```
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) { // 버튼 클릭 시 호출되는 메소드
        JButton b = (JButton)e.getSource(); // 클릭된 버튼 알아내기
        if(b.getText().equals("Action")) // 버튼의 문자열이 "Action"인지 비교
            b.setText("액션"); // 문자열 변경
        else
            b.setText("Action"); // 문자열 변경
    }
}
```

if(e.getActionCommand().equals("Action"))으로 해도 됨

이 코드에서 사용자가 클릭한 버튼 컴포넌트(이벤트 소스)는 다음과 같이 알아낸다.

```
JButton b = (JButton)e.getSource(); // 이벤트 소스를 알아낸다.
```

그러고 나서 다음 코드를 이용하여 클릭된 버튼의 문자열을 "액션"으로 변경한다.

```
b.setText("액션");
```

● 이벤트 리스너 등록

이벤트 리스너가 작동하기 위해서는 MyActionListener의 객체를 생성하여 이벤트
처리할 버튼 컴포넌트에 등록해야 한다. 다음 코드와 같다.

```
MyActionListener listener = new MyActionListener(); // 리스너 객체 생성
btn.addActionListener(listener); // 리스너 등록
```

일반적으로 컴포넌트 component에 이벤트 리스너를 등록하기 위해서는 다음 메
드를 사용한다.

```
component.addXXXListener(listener);
```

여기서 XXX는 이벤트 이름에 해당한다. 예를 들면, addActionListener(
addFocusListener(), addMouseListener() 등이다. 그러나 모든 컴포넌트가 모든
벤트 리스너를 가지고 있는 것은 아니다. JButton에는 Item 이벤트가 발생하지 않
때문에, JButton 클래스는 addItemListener() 메소드를 가지고 있지 않다.

오류
```
JButton b = new JButton("test");
b.addItemListener(listener); // 오류. JButton 클래스에 addItemListener() 없음
```

또한 JScrollBar 컴포넌트에도 Action 이벤트가 발생하지 않으므로(<표 10-1> 참
조), JScrollBar 클래스 역시 addActionListener() 메소드를 가지지 않는다.

오류
```
JScrollBar s = new JScrollBar();
s.addActionListener(listener); // 오류. JScrollBar 클래스에 addActionListener() 없음
```

반면에 JList는 Mouse 이벤트를 받을 수 있기 때문에, 다음과 같이 addMous
Listener() 메소드를 가진다. 완성된 코드는 예제 10-1에서 확인할 수 있다.

```
JList list = new JList();
list.addMouseListener(listener); // 정상
```



잠깐!

- 이벤트 리스너는 컴포넌트에 등록되어야 이벤트가 발생할 때 호출된다.
- 모든 컴포넌트가 모든 이벤트를 처리할 수 있는 것은 아니다. 어떤 컴포넌트가 어떤 이벤트 리스너를 가질 수 있는지 〈표 10-1〉, 〈표 10-2〉의 표를 보면서 판단하여야 한다.



Tip 이벤트 리스너를 등록하는 메소드 이름 addXXXListener()에 add를 사용한 이유는 무엇일까?

한 컴포넌트는 다른 종류의 이벤트에 대해 각각 리스너를 가질 수 있다. 예를 들어, JButton은 다음과 같이 이벤트 리스너를 등록하는 서로 다른 메소드를 가지고 있다.

```
button.addActionListener(myActionListener); // Action 이벤트 리스너 등록
button.addKeyListener(myKeyListener);       // Key 이벤트 리스너 등록
button.addFocusListener(myFocusListener);   // Focus 이벤트 리스너 등록
button.addMouseListener(myMouseListener);   // Mouse 이벤트 리스너 등록
```

그런데, setBackground()처럼 많은 메소드들이 set으로 시작하는데, 특별히 이들이 add로 시작하는 이유는 무엇일까? 그것은 컴포넌트가 동일한 이벤트에 대해 여러 개의 이벤트 리스너를 가질 수 있기 때문이다. 다음과 같이 하면 JButton은 3개의 서로 다른 Action 이벤트 리스너를 가지게 된다.

```
button.addActionListener(new MyActionListener1());
button.addActionListener(new MyActionListener2());
button.addActionListener(new MyActionListener3());
```

이런 경우 button에 Action 이벤트가 발생하면 3개의 리스너가 모두 실행되며, 이들은 **등록된 반대순으로 실행**된다.

등록된 반대순으로 실행

이벤트 리스너 작성 방법

이 절에서는 이벤트 리스너 클래스를 작성하는 3가지 방법을 소개한다.

● 독립 클래스로 이벤트 리스너 작성

이벤트 리스너 클래스를 독립적으로 작성하는 방법으로서, 가장 전형적인 방법이다. 예제 10-1은 Action 이벤트 리스너를 독립된 클래스 MyActionListener로 작성한 사례이다. 이 방법은 리스너 코드가 길 때 적합하나, MyActionListener에서 다른 클래스의 멤버에 접근하기 어려운 단점이 있다. 예를 들어, 예제 10-1에서 actionPerformed()가 프레임 윈도우(IndepListenerClass 클래스)의 멤버에 접근할 수 없다.

예제 10-1 독립 클래스로 Action 이벤트의 리스너 작성

버튼을 클릭할 때 발생하는 Action 이벤트를 처리할 MyActionListener 클래스를 독립 클래스로 작성하라. 클릭할 때마다 버튼 문자열이 "Action"과 "액션"으로 번갈아 변하도록 하라.




```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class IndepClassListener extends JFrame {
6     public IndepClassListener() {
7        setTitle("Action 이벤트 리스너 예제");
8        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9        Container c = getContentPane();
10       c.setLayout(new FlowLayout());
11       JButton btn = new JButton("Action");
12       btn.addActionListener(new MyActionListener()); // Action 이벤트 리스너 달기
13       c.add(btn);
14
15       setSize(350, 150);
16       setVisible(true);
17    }
18    public static void main(String [] args) {
19       new IndepClassListener();
20    }
21 }
22 // 독립된 클래스로 이벤트 리스너를 작성한다.
23 class MyActionListener implements ActionListener {
24    public void actionPerformed(ActionEvent e) {
25       JButton b = (JButton)e.getSource(); // 이벤트 소스 버튼 알아내기
26       if(b.getText().equals("Action")) // 버튼의 문자열이 "Action"인지 비교
27          b.setText("액션"); // 버튼의 문자열을 "액션"으로 변경
28       else
29          b.setText("Action"); // 버튼의 문자열을 "Action"으로 변경
30    }
31 }
```

MyActionListener.java 파일로 따로 작성해도 됨

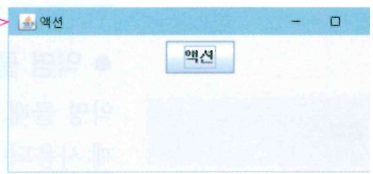
● 내부 클래스(inner class)로 이벤트 리스너 작성

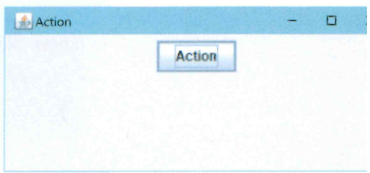
이벤트 리스너를 내부 클래스(inner class)로 작성하는 방법이다. 예제 10-2는 예제 10-1의 리스너를 내부 클래스로 작성된 사례이다. 내부 클래스는 자신을 내포한 클래스의 멤버와 같으므로, 예제 10-2에서 MyActionListener는 InnerClassListener 클래스나 슈퍼 클래스인 JFrame의 멤버를 마음대로 접근할 수 있다.

내부 클래스로 Action 이벤트 리스너 만들기 예제 10-2

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class InnerClassListener extends JFrame {
6     public InnerClassListener() {
7        setTitle("Action 이벤트 리스너 예제");
8        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9        Container c = getContentPane();
10       c.setLayout(new FlowLayout());
11       JButton btn = new JButton("Action");
12       btn.addActionListener(new MyActionListener());
13       c.add(btn);
14
15       setSize(350, 150);
16       setVisible(true);
17
18    }
19    // 내부 클래스로 Action 리스너를 작성한다.
20    private class MyActionListener implements ActionListener {
21       public void actionPerformed(ActionEvent e) {
22          JButton b = (JButton)e.getSource();
23          if(b.getText().equals("Action"))
24             b.setText("액션");
25          else
26             b.setText("Action");
27
28          // InnerClassListener의 멤버나 JFrame의 멤버를 호출할 수 있음
29          InnerClassListener.this.setTitle(b.getText()); // 프레임의 타이틀에 버
                                                           튼 문자열을 출력한다.
30       }
31    }
32    public static void main(String [] args) {
33       new InnerClassListener();
34    }
35 }
```

버튼의 문자열을 타이틀에 출력





Action 이벤트 리스너 달기

이 클래스는 내부 클래스이므로 자신을 둘러싼 외부 클래스인 InnerClassListener나 상속받은 JFrame의 모든 멤버에 접근할 수 있다.

InnerClassListener.thi
에 유의하라.

이 기능을 이용하여 소스의 29 라인에 다음 코드를 작성하여 이벤트가 발생할 때마다 버튼 이름을 프레임 윈도우의 타이틀에 출력하도록 작성해보았다.

```
InnerClassListener.this.setTitle(b.getText()); // JFrame의 setTitle() 호출. 그냥
                                                setTitle(b.getText())로 해도 됨
```

이 코드는 JFrame의 멤버 setTitle()을 호출하는 것으로 MyActionListener가 내부 클래스이기 때문에 가능하다. 한편 이 예제에서는 MyActionListener 클래스를 private 속성으로 선언하여 InnerClassListener 외부에서 사용할 수 없도록 하였다.

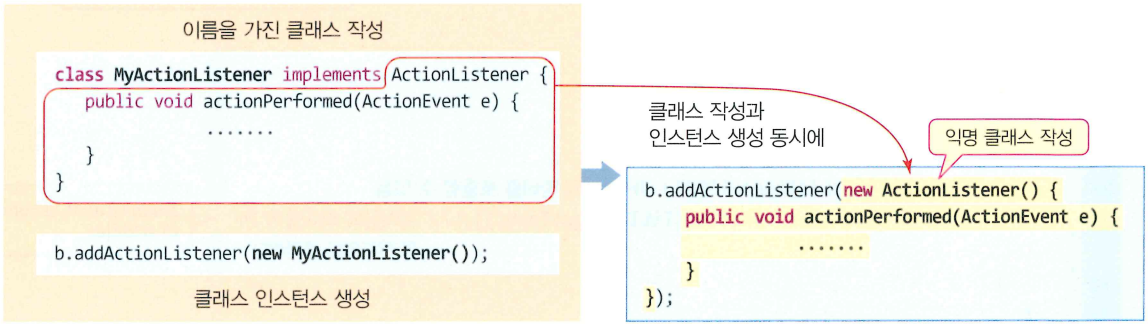
● 익명 클래스(anonymous class)로 이벤트 리스너 작성

익명 클래스

익명 클래스는 이름 없이 만들어진 클래스이다. 익명 클래스는 다음과 같이 new와 함께 사용되어 바로 객체를 생성하는 데 사용된다.

```
new 익명클래스의슈퍼클래스(생성자인자들) {
    // 멤버 구현
}
```

익명 클래스를 만드는 사례는 [그림 10-5]와 같다. [그림 10-5](a)는 MyActionListener 클래스를 작성한 사례로, ActionListener를 상속받아 클래스를 작성하고, new로 객체를 생성하는 부분으로 나뉘어져 있다. 이 코드를 익명 클래스로 작성하면, [그림 10-5](b)와 같이 ActionListener 인터페이스를 상속받으면서 바로 actionPerformed() 메소드를 구현한 코드 블록, 즉 익명 클래스를 작성하고 new로 생성하면 된다.



(a) 이름을 가진 클래스를 작성하고 클래스 인스턴스 생성하는 경우

(b) ActionListener를 상속받고 바로 메소드 작성, 동시에 new로 인스턴스를 생성하는 경우

[그림 10-5] MyActionListener 클래스를 익명 클래스로 작성

예제 10-3은 익명 클래스를 이용하여 예제 10-2를 재작성한 사례이다. 익명 클래스의 작성은 까다롭기는 하지만 금방 익숙해진다. 코드가 짧고 한 군데에서만 사용하는 경우, 익명 클래스로 작성하는 것이 더 편리하기 때문에 많은 개발자들이 사용하고 있다. 익명 클래스도 일종의 내부 클래스이므로 외부 클래스의 멤버에 마음대로 접근할 수 있다. 그러나 익명 클래스의 코드가 긴 경우 new 문장의 가독성이 떨어지므로 피하는 것이 좋다.

익명 클래스로 Action 이벤트 리스너 만들기 예제 10-3

예제 10-2를 익명 클래스를 이용하여 재작성하라.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class AnonymousClassListener extends JFrame {
6     public AnonymousClassListener() {
7        setTitle("Action 이벤트 리스너 작성");
8        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9        Container c = getContentPane();
10       c.setLayout(new FlowLayout());
11       JButton btn = new JButton("Action");
12       c.add(btn);
13       btn.addActionListener(new ActionListener() {
14          public void actionPerformed(ActionEvent e) {
15             JButton b = (JButton)e.getSource();
16             if(b.getText().equals("Action"))
17                b.setText("액션");
18             else
19                b.setText("Action");
20
21             // AnonymousClassListener의 멤버나 JFrame의 멤버를 호출할 수 있음
22             setTitle(b.getText());
23          }
24       });
25
26       setSize(350, 150);
27       setVisible(true);
28    }
29    public static void main(String [] args) {
30       new AnonymousClassListener();
31    }
32 }
```

익명 클래스로 Action 리스너 작성

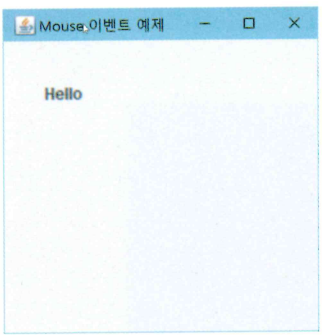
AnonymousClassListener.this.setTitle(b.getText());
로도 할 수 있음



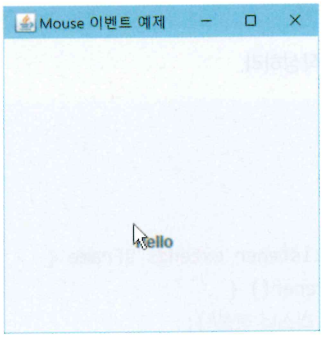
예제 10-4 마우스로 문자열 이동시키기 – 마우스 이벤트 연습

컨텐트팬의 아무 위치에 마우스 버튼을 누르면 마우스 포인트가 있는 위치로 "hello" 문자열을 옮기는 스윙 응용프로그램을 작성하라.

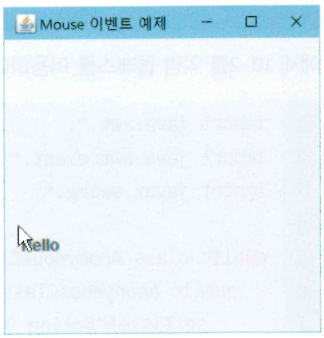
MouseListener를 상속받아 MyMouseListener를 작성하고 mousePressed()에 코드를 작성하면 된다. 나머지 4개의 메소드는 단순 리턴하도록 만든다. "hello" 문자열은 JLabel 컴포넌트를 사용하고, 컨텐트팬의 배치관리자를 null로 설정하여 마우스가 클릭된 위치로 JLabel 컴포넌트를 이동시킨다.



초기 화면



마우스를 다른 곳에 클릭한 경우

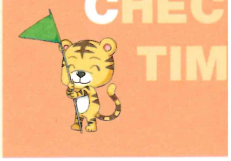


마우스를 다른 곳에 클릭한 경우

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class MouseListenerEx extends JFrame {
6     private JLabel la = new JLabel("Hello"); // "Hello"를 출력하기 위한 레이블
7
8     public MouseListenerEx() {
9        setTitle("Mouse 이벤트 예제");
10       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11       Container c = getContentPane();
12       c.addMouseListener(new MyMouseListener()); // 컨텐트팬에 이벤트 리스너 달기
13
14       c.setLayout(null); // 컨텐트팬의 배치관리자 삭제
15       la.setSize(50, 20); // 레이블의 크기 50x20 설정
16       la.setLocation(30, 30); // 레이블의 위치 (30,30)으로 설정
17       c.add(la); // 레이블 컴포넌트 삽입
18
19       setSize(250, 250);
20       setVisible(true);
21    }
22
23    // Mouse 리스너 구현
```

```
24      class MyMouseListener implements MouseListener {
25         public void mousePressed(MouseEvent e) {
26            int x = e.getX(); // 마우스 클릭 좌표 x
27            int y = e.getY(); // 마우스 클릭 좌표 y
28            la.setLocation(x, y); // 레이블의 위치를 (x,y)로 이동
29         }
30         public void mouseReleased(MouseEvent e) {}
31         public void mouseClicked(MouseEvent e) {}
32         public void mouseEntered(MouseEvent e) {}
33         public void mouseExited(MouseEvent e) {}
34      }
35
36      public static void main(String [] args) {
37         new MouseListenerEx();
38      }
39   }
```

마우스 버튼이 눌려진 위치 (x, y)를 알아내어 "hello" 레이블의 위치를 (x, y)로 이동시킨다.



1. MouseListener를 상속받는 마우스 리스너와 ActionListener를 상속받는 리스너 중 익명 클래스를 사용하면 더 효과적인 것은? 그리고 그 이유는?
2. 이벤트 리스너를 독립된 클래스로 작성하는 것보다 내부 클래스로 작성하는 장점은 무엇인가?

10.4 어댑터(Adapter) 클래스

리스너 인터페이스를 상속받아 이벤트 리스너를 구현할 때 리스너 인터페이스의 메소드를 모두 구현하여야 하는 부담이 있다. 예를 들어, 예제 10-4에서 마우스가 눌러지는 순간만 처리하려면 mousePressed()만 필요하지만, MouseInterface의 나머지 4개의 메소드도 모두 작성할 수밖에 없었다.

어댑터 클래스

자바의 JDK에는 이런 부담을 줄여주기 위해 리스너 인터페이스를 미리 구현해 놓은 클래스를 제공하는데 이것이 바로 어댑터 클래스(Adapter)이다. 〈표 10-3〉은 자바에서 제공하는 어댑터 클래스들을 보여주며, 예를 들어 MouseAdapter 클래스는 모든 메소드가 단순 리턴하도록 다음과 같이 작성되어 있다.

```
class MouseAdapter implements MouseListener, MouseMotionListener, MouseWheelListener {
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseClicked(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
    public void mouseDragged(MouseEvent e) { }
    public void mouseMoved(MouseEvent e) { }
    public void mouseWheelMoved(MouseWheelEvent e) { }
}
```

MouseListener 메소드

MouseMotionListener 메소드

MouseWheelListener 메소드

개발자들은 이벤트 리스너를 작성할 때, 리스너 인터페이스 대신 어댑터 클래스를 이용하는 것이 훨씬 편하다. [그림 10-6]은 MouseAdapter로 마우스 리스너를 구현한 코드 예를 보여준다.

〈표 10-3〉
리스너 인터페이스와 대응하는 어댑터 클래스

리스너 인터페이스	대응하는 어댑터 클래스
ActionListener	없음
ItemListener	없음
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter 혹은 MouseAdapter
FocusListener	FocusAdapter
WindowListener	WindowAdapter
AdjustmentListener	없음
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter



잠깐!

〈표 10-3〉에는 ActionListener와 같이 어댑터가 제공되지 않는 리스너가 있다. 리스너 인터페이스에 메소드가 하나뿐인 경우 굳이 어댑터 클래스를 만들 필요가 없기 때문이다.

```
JLabel la;
contentPane.addMouseListener(new MyMouseListener());

class MyMouseListener implements MouseListener {
   public void mousePressed(MouseEvent e) {
      int x = e.getX();
      int y = e.getY();
      la.setLocation(x, y);
   }
   public void mouseReleased(MouseEvent e) {}
   public void mouseClicked(MouseEvent e) {}
   public void mouseEntered(MouseEvent e) {}
   public void mouseExited(MouseEvent e) {}
}
```

MouseListener를 이용한 경우

```
JLabel la;
contentPane.addMouseListener(new MyMouseAdapter());

class MyMouseAdapter extends MouseAdapter {
   public void mousePressed(MouseEvent e) {
      int x = e.getX();
      int y = e.getY();
      la.setLocation(x, y);
   }
}
```

MouseAdapter를 이용한 경우

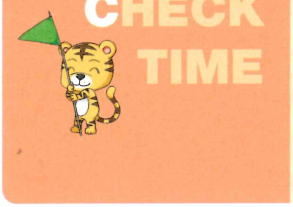
[그림 10-6] MouseListener를 MouseAdapter로 변경한 경우

예제 10-5 MouseAdapter 사용하기

MouseAdapter를 이용하여 예제 10-4를 수정하라.

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4
5  public class MouseAdapterEx extends JFrame {
6     private JLabel la = new JLabel("Hello"); // "Hello"을 출력하기 위한 레이블
7
8     public MouseAdapterEx() {
9        setTitle("Mouse 이벤트 예제");
10       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11       Container c = getContentPane();
12       c.addMouseListener(new MyMouseAdapter()); // 컨텐트팬에 Mouse 이벤트 리스
                                                   너 달기
13
14       c.setLayout(null); // 컨텐트팬의 배치관리자 삭제
15       la.setSize(50, 20); // 레이블의 크기 50x20 설정
16       la.setLocation(30, 30); // 레이블의 위치 (30,30)으로 설정
17       c.add(la); // 레이블 컴포넌트 삽입
18
19       setSize(250, 250);
20       setVisible(true);
21    }
22
23    // MouseAdapter를 상속받아 리스너 구현
```

```
24      class MyMouseAdapter extends MouseAdapter {
25         public void mousePressed(MouseEvent e) {
26            int x = e.getX(); // 마우스 클릭 좌표 x
27            int y = e.getY(); // 마우스 클릭 좌표 y
28            la.setLocation(x, y); // 레이블의 위치를 (x,y)로 이동
29         }
30      }
31
32      public static void main(String [] args) {
33         new MouseAdapterEx();
34      }
35   }
```



1. 어댑터는 인터페이스인가, 클래스인가?
2. 어댑터를 이용하는 것이 리스너를 이용하는 것보다 편리한 점이 무엇인가?
3. ActionListener와 ItemListener에 대응되는 ActionAdapter와 ItemAdapter가 존재하지 않는 이유는 무엇인가?

10.5 KeyEvent와 KeyListener

Key 이벤트와 포커스

Key 이벤트
포커스
키 입력의 독점권

Key 이벤트와 포커스의 관계를 잘 몰라 키 입력이 되지 않는다고 고생하는 개발자들을 본 적이 있다. Key 이벤트는 사용자가 키를 입력할 때 발생하는 이벤트이며, 모든 컴포넌트가 Key 이벤트를 받을 수 있다. 그러나 응용프로그램 내에 포커스(focus)를 가진 컴포넌트가 키 입력을 독점하기 때문에, 현재 포커스를 가진 컴포넌트에만 Key 이벤트가 전달된다. 포커스란 키 입력의 독점권을 뜻한다. 버튼을 누르기 위해 <Enter> 키를 입력하더라도 버튼이 포커스를 가지고 있지 않다면 Key 이벤트를 받을 수 없다.

어떤 컴포넌트에게 키를 입력하고자 하면 <Tab> 키나 마우스 클릭으로 포커스를 그 컴포넌트에게 이동시켜야 한다. 스윙 응용프로그램에서는 강제로 임의의 컴포넌트에 포커스를 주기 위해 다음 두 코드가 모두 필요하다.

```
component.setFocusable(true); // component가 포커스를 받을 수 있도록 설정한다.
component.requestFocus(); // component에게 포커스를 주어 키 입력을 받을 수 있게 함
```

component.requestFocus()만으로 component가 키 입력을 받을 수 있는 경우가 대부분이지만, 독자 컴퓨터의 설치 상황(실행 환경)에 따라 component.setFocusable(true)가 없으면 키 입력을 받지 못하는 경우가 있으므로 반드시 두 코드를 이용하기 바란다.

컴포넌트에 포커스 주기

키 입력을 받기 위해서는 포커스를 주는 시점 또한 중요하다. 첫째, 스윙 프레임이 만들어질 때, 어떤 컴포넌트에게 포커스를 주고자 한다면, JFrame의 setVisible(true) 코드를 실행한 후 컴포넌트의 requestFocus() 메소드를 호출해야 한다. JFrame의 setVisible(true) 메소드는 프레임을 출력한 후 포커스를 임의로 움직이기 때문이다.

```
setVisible(true); // 스윙 프레임 출력
component.setFocusable(true);
component.requestFocus(); // component 컴포넌트에게 포커스 설정
```

둘째, 사용자가 마우스로 컴포넌트를 클릭하면 그때 컴포넌트가 포커스를 얻도록 하는 방법으로 다음 코드를 작성하면 된다.

```
component.addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        Component c = (Component)e.getSource(); // 마우스가 클릭된 컴포넌트
        c.setFocusable(true);
        c.requestFocus(); // 마우스가 클릭된 컴포넌트에게 포커스 설정
    }
}); // 예제 10-8에서 활용하였음
```



잠깐!

만일 실행 중인 응용프로그램에 키 입력이 잘 되지 않는다면, 키를 받아야 하는 컴포넌트에 포커스가 설정되어 있는지 확인해보는 것이 좋다.

Key 이벤트와 KeyListener

이제 키를 입력하면 어떤 일들이 발생하며, 입력된 키를 어떻게 판별하는지 알아보자. 키 하나를 입력하는 동안 [그림 10-7]과 같이 순서대로 이벤트가 발생한다. 모든 키에 대해서 '키를 누르는 순간', '누른 키를 떼는 순간'에 각각 Key 이벤트가 한 번씩 발생한다. 특별히 문자 키(유니코드 키)인 경우에는 '누른 키를 떼는 순간'에 Key 이벤트가 한 번 더 발생하여 총 3번의 Key 이벤트가 발생하며, 각 경우에 [그림 10-7]과 같이 KeyListener의 메소드가 호출된다.

keyPressed()
keyReleased()
keyTyped()

KeyListener 인터페이스는 〈표 10-4〉, [그림 10-7]과 같이 3개의 추상 메소드로 구성된다. keyPressed()는 키를 누르는 순간에, keyReleased()는 누른 키를 떼는 순간에 호출되며, 문자 키(유니코드)인 경우에는 누른 키가 떼어지는 순간 keyTyped()가 가적으로 호출된다.

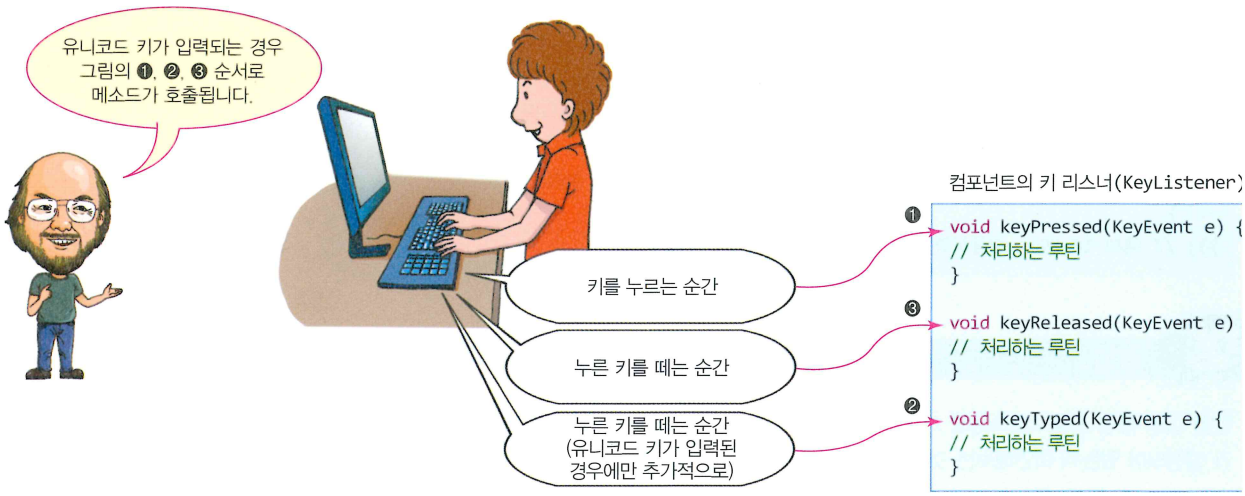
유니코드
제어 키

유니코드(Unicode)란 전 세계의 모든 문자를 컴퓨터에서 일관되게 표현하고 다루고자 설계된 국제 산업 표준이다. 유니코드는 한글, 영어 등 세계 여러 나라들의 문자들에 대해 코드 값을 정의하고 있다. 그러므로 A~Z, a~z, 0~9, !, @, & 등은 유니코드 값이 정의되어 있지만, <Function> 키, <Home> 키, <Up> 키 등 문자가 아닌 제어 키에 대해서는 유니코드 값을 정의하지 않는다.

유니코드 키가 입력될 때, KeyListener의 메소드가 호출되는 순서는 다음과 같다.

```
keyPressed(), keyTyped(), keyReleased()
```

예를 들어, 키 f를 입력하면 keyPressed(), keyTyped(), keyReleased() 순서대로 호출되며, <Home> 키를 입력하면 keyPressed(), keyReleased()만 순서대로 호출된다.



[그림 10-7] Key 이벤트가 발생하는 경우와 호출되는 KeyListener 메소드

키 이벤트 리스너 달기

컴포넌트에 키 이벤트 리스너를 등록하기 위해서는 다음과 같이 addKeyListener() 메소드를 이용한다.

```
component.addKeyListener(myKeyListener);
```

〈표 10-4〉 Key 이벤트가 발생하는 경우 호출되는 KeyListener의 메소드

Key 이벤트가 발생하는 경우	KeyListener의 메소드	리스너
키를 누르는 순간	void keyPressed(KeyEvent e)	KeyListener
누른 키를 떼는 순간	void keyReleased(KeyEvent e)	KeyListener
누른 키를 떼는 순간 (유니코드 키 경우에만 적용)	void keyTyped(KeyEvent e)	KeyListener

입력된 키 판별

키 이벤트가 발생하면 입력된 키 정보가 KeyEvent 객체에 담겨져 리스너에게 전달된다. KeyEvent 객체의 다음 2개의 메소드를 통해 입력된 키를 판별할 수 있다.

● char KeyEvent.getKeyChar()

getKeyChar()

입력된 키의 문자코드(유니코드 값)를 리턴하며, 유니코드 키가 아닌 경우 KeyEvent.CHAR_UNDEFINED를 리턴한다. 다음은 q 키가 눌러지는 순간 프로그램을 종료하는 코드이다. getKeyChar()의 리턴 값과 문자 'q'를 서로 비교하면 된다.

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyChar() == 'q')
        System.exit(0); // 키 q가 눌러지면 프로그램을 종료한다.
}
```

이런 방법으로 어떤 문자가 입력되었는지 판별할 수 있다.

● int KeyEvent.getKeyCode()

getKeyCode()
가상 키
VK_

이 메소드는 유니코드 키를 포함한 모든 키에 대해 정수형의 키 코드(key code) 값을 리턴한다. 키 코드는 운영체제나 하드웨어에 따라 서로 다를 수 있기 때문에, 입력된 키를 판별하기 위해서는 반드시 getKeyCode()가 리턴한 키 코드와 가상 키(Virtual Key) 값을 비교해야 한다. 가상 키는 KeyEvent 클래스에 VK_로 시작하는 static 상수로 선언되어 있다. 〈표 10-5〉는 KeyEvent 클래스에 선언된 가상 키를 보여준다. 다음 예는 <F5> 키가 눌러지면 프로그램을 종료하는 코드이다.

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_F5)
        System.exit(0); // <F5> 키가 눌러지면 프로그램을 종료한다.
}
```

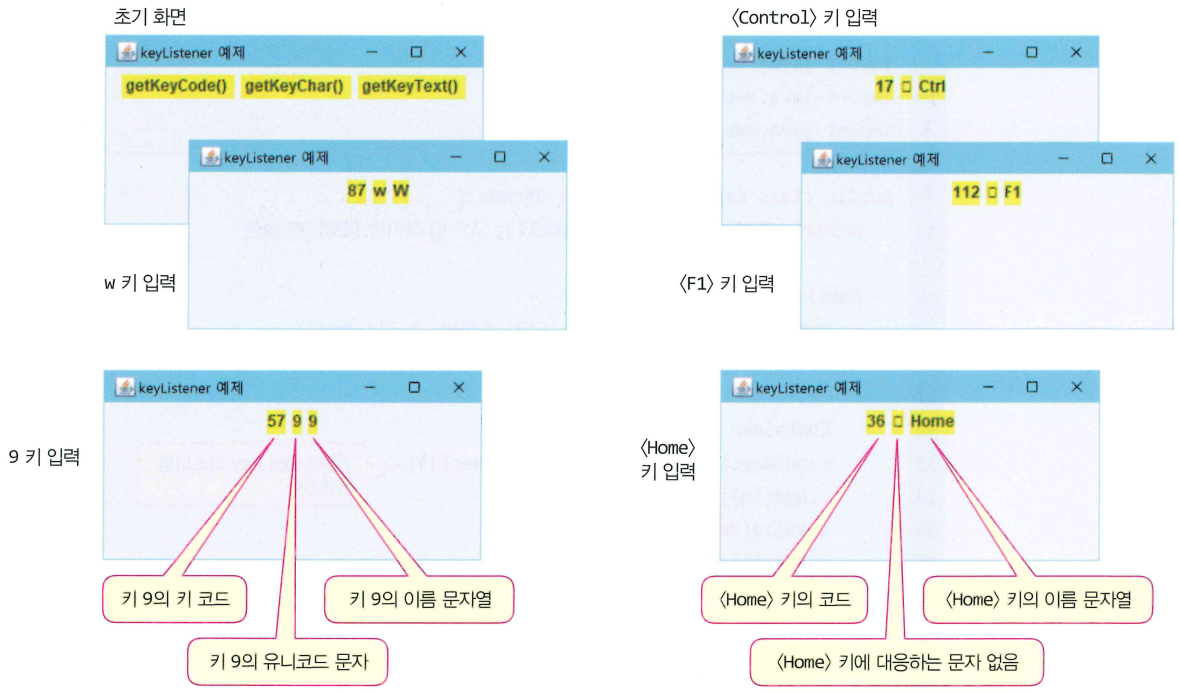
[그림 10-8]은 KeyEvent 클래스의 getKeyChar()와 getKeyCode() 메소드의 사용 예를 보여준다.

〈표 10-5〉 KeyEvent 클래스에 정의된 가상 키

가상 키	설명	가상 키	설명
VK_0 ~ VK_9	0에서 9까지의 키, '0'~'9'까지의 유니코드 값과 동일	VK_LEFT	왼쪽 방향 키
VK_A ~ VK_Z	A에서 Z까지의 키, 'A'~'Z'까지의 유니코드 값과 동일	VK_RIGHT	오른쪽 방향 키
VK_F1 ~ VK_F24	<F1>~<F24>까지의 키 코드	VK_UP	<Up> 키
VK_HOME	<Home> 키	VK_DOWN	<Down> 키
VK_END	<End> 키	VK_CONTROL	<Control> 키
VK_PGUP	<Page Up> 키	VK_SHIFT	<Shift> 키
VK_PGDN	<Page Down> 키	VK_ALT	<Alt> 키
VK_UNDEFINED	입력된 키의 코드 값을 알 수 없음	VK_TAB	<Tab> 키



[그림 10-8] KeyEvent의 getKeyChar()과 getKeyCode() 메소드의 사용 예



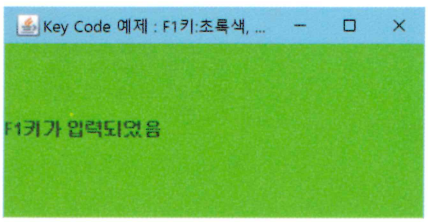
[그림 10-9] 예제 10-6의 실행 화면

〈F1〉 키를 입력받으면 컨텐트팬의 배경을 초록색으로, % 키를 입력받으면 노란색으로 변경 — 예제 10-7

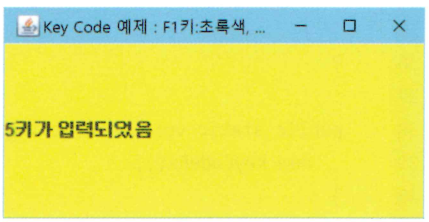


다음 그림과 같이 <F1> 키를 입력받으면 콘텐트팬의 배경을 초록색으로, % 키를 입력받으면 노란색으로 변경하는 응용프로그램을 작성하라.

사용자가 키를 입력하면 입력된 키의 이름 문자열을 화면에 출력한다. 아래의 오른쪽 그림에서 % 키는 사실상 <Shift> 키 입력 후 5 키를 입력한 경우로서, 최종적으로 입력된 키는 5 키이므로 오른쪽 화면에 "5키가 입력되었음"이라고 출력되어 있다. <F1> 키를 받기 위해 KeyEvent의 getKeyCode() 메소드를 이용하여야 하며, % 키는 getKeyChar() 메소드를 이용하면 된다.



〈F1〉 키가 입력되면 초록색 배경



% 키가 입력되면 노란색 배경
(% 키나 5 키는 동일한 키)

스윙 컴포넌트의 배경색을 설정하기 위해 미리 setOpaque(true) 호출

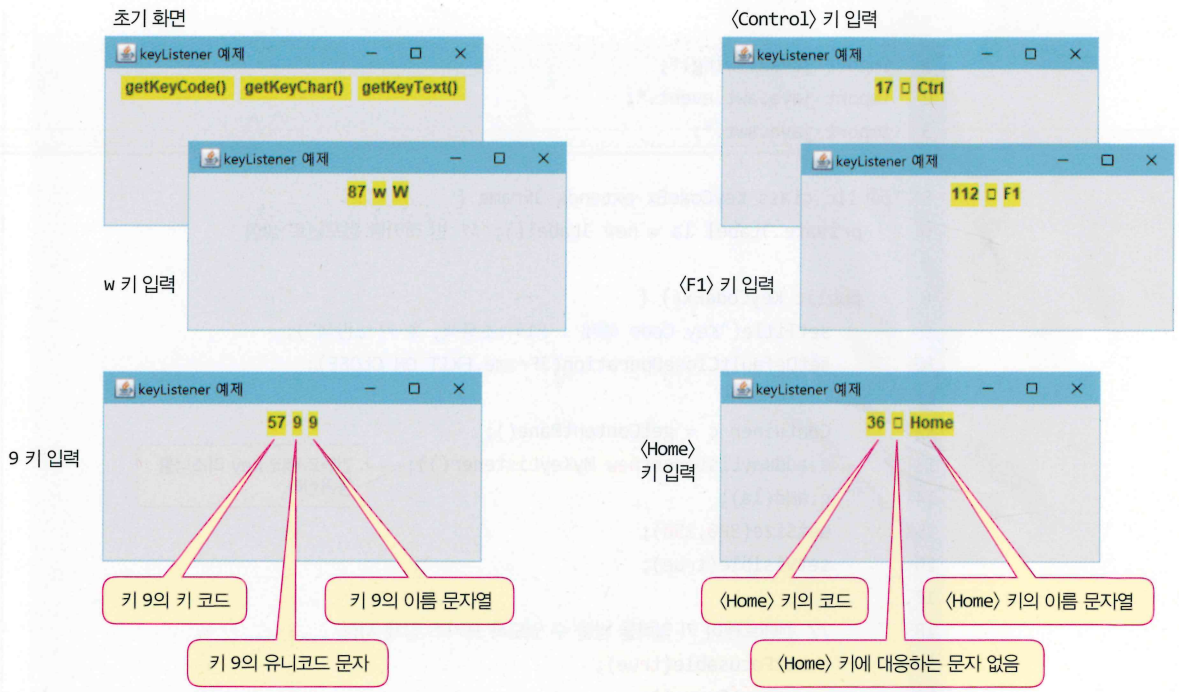
KeyEvent 객체로부터 입력된 키의 키 코드 값을 알아냄

KeyEvent 객체로부터 입력된 키의 유니코드 문자 값을 알아냄

키 코드 값으로부터 키 이름 문자열을 알아내어 레이블에 출력

System.out.println()은 콘솔에 출력

```
23          for(int i=0; i<keyMessage.length; i++) {
24              c.add(keyMessage[i]);
25              keyMessage[i].setOpaque(true); // 배경색이 보이록 불투명 속성 설정
26              keyMessage[i].setBackground(Color.YELLOW); // 배경색을 CYAN 색으로 변경
27          }
28
29          setSize(300,150);
30          setVisible(true);
31
32          // 컨텐트팬이 키 입력을 받을 수 있도록 포커스 강제 지정
33          c.setFocusable(true);
34          c.requestFocus();
35      }
36
37      // Key 리스너 구현
38      class MyKeyListener extends KeyAdapter {
39          public void keyPressed(KeyEvent e) {
40              int keyCode = e.getKeyCode(); // 키 코드 알아내기
41              char keyChar = e.getKeyChar(); // 키 문자 값 알아내기
42              keyMessage[0].setText(Integer.toString(keyCode)); // 키 코드 출력
43              keyMessage[1].setText(Character.toString(keyChar)); // 키 문자 출력
44              keyMessage[2].setText(KeyEvent.getKeyText(keyCode)); // 키 이름 문자
                                                                     열 출력
45              System.out.println("KeyPressed"); // 콘솔창에 메소드 이름 출력
46          }
47          public void keyReleased(KeyEvent e) {
48              System.out.println("KeyReleased"); // 콘솔창에 메소드 이름 출력
49          }
50          public void keyTyped(KeyEvent e) {
51              System.out.println("KeyTyped"); // 콘솔창에 메소드 이름 출력
52          }
53      }
54
55      public static void main(String [] args) {
56          new KeyListenerEx();
57      }
58  }
```



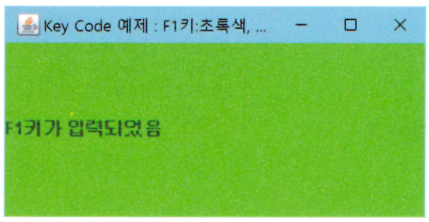
[그림 10-9] 예제 10-6의 실행 화면

〈F1〉 키를 입력받으면 컨텐트팬의 배경을 초록색으로, % 키를 입력받으면 노란색으로 변경 — 예제 10-7

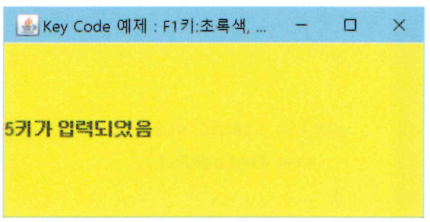


다음 그림과 같이 <F1> 키를 입력받으면 콘텐트팬의 배경을 초록색으로, % 키를 입력받으면 노란색으로 변경하는 응용프로그램을 작성하라.

사용자가 키를 입력하면 입력된 키의 이름 문자열을 화면에 출력한다. 아래의 오른쪽 그림에서 % 키는 사실상 <Shift> 키 입력 후 5 키를 입력한 경우로서, 최종적으로 입력된 키는 5 키이므로 오른쪽 화면에 "5키가 입력되었음"이라고 출력되어 있다. <F1> 키를 받기 위해 KeyEvent의 getKeyCode() 메소드를 이용하여야 하며, % 키는 getKeyChar() 메소드를 이용하면 된다.



〈F1〉 키가 입력되면 초록색 배경



% 키가 입력되면 노란색 배경
(% 키나 5 키는 동일한 키)

```java
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class KeyCodeEx extends JFrame {
   private JLabel la = new JLabel(); // 빈 레이블 컴포넌트 생성

   public KeyCodeEx() {
      setTitle("Key Code 예제 : F1키:초록색, % 키 노란색");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      Container c = getContentPane();
      c.addKeyListener(new MyKeyListener());
      c.add(la);
      setSize(300,150);
      setVisible(true);

      // 컨텐트팬이 키 입력을 받을 수 있도록 포커스 강제 지정
      c.setFocusable(true);
      c.requestFocus();
   }

   // Key 리스너 구현
   class MyKeyListener extends KeyAdapter {
      public void keyPressed(KeyEvent e) {
         Container contentPane = (Container)e.getSource();

         // la에 입력된 키의 키 이름 문자열을 출력하여 사용자에게 보고함
         la.setText(KeyEvent.getKeyText(e.getKeyCode())+"키가 입력되었음");

         if(e.getKeyChar() == '%') // 입력된 키가  % 인 경우
            contentPane.setBackground(Color.YELLOW);
         else if(e.getKeyCode() == KeyEvent.VK_F1) // 입력된 키가 <F1> 인가
            contentPane.setBackground(Color.GREEN);
      }
   }

   public static void main(String [] args) {
      new KeyCodeEx();
   }
}
```

컨텐트팬에 Key 리스너를 등록한다.

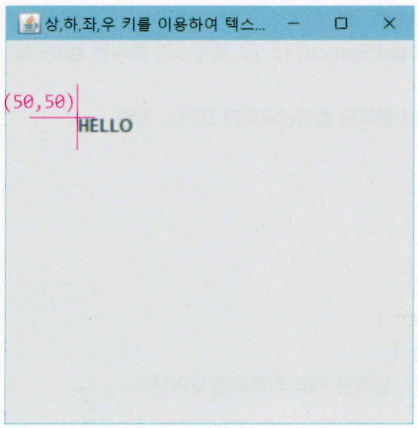
% 키는 유니코드 키이므로 getKeyChar() 메소드를 이용하여 리턴된 키의 문자 값과 '%' 문자를 비교한다.

<F1> 키는 유니코드 키가 아니므로 getKeyCode() 메소드를 이용하여 리턴된 키코드와 VK_F1 값을 비교한다.

상(UP), 하(DOWN), 좌(LEFT), 우(RIGHT) 키로 "HELLO" 문자열을 마음대로 움직이기 예제 10-8

상, 하, 좌, 우 키를 이용하여 "HELLO" 문자열을 움직이는 응용프로그램을 작성하라.

"HELLO" 문자열은 JLabel 컴포넌트로 만들어 컨텐트팬에 부착하고 상, 하, 좌, 우 키를 움직이면 키 방향으로 한 번에 10픽셀씩 움직인다. 이를 위해 컨텐트팬의 배치관리자를 삭제하여야 한다. "HELLO" 문자열을 초기에 (50, 50) 위치에 출력하라.



초기 상태



상, 하, 좌, 우 키를 여러 번 입력하여 "HELLO"를 움직인 상태

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  public class FlyingTextEx extends JFrame {
6     private final int FLYING_UNIT = 10; // 레이블이 한 번 움직이는 단위는 10픽셀
7     private JLabel la = new JLabel("HELLO"); // 키 입력에 따라 움직일 레이블 컴포넌트
8
9     public FlyingTextEx() {
10       setTitle("상,하,좌,우 키를 이용하여 텍스트 움직이기");
11       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13       Container c = getContentPane();
14       c.setLayout(null); // 컨텐트팬의 배치관리자 삭제
15       c.addKeyListener(new MyKeyListener()); // 컨텐트팬에 Key 리스너 달기
16       la.setLocation(50,50); // la의 초기 위치는 (50,50)
17       la.setSize(100,20);
18       c.add(la);
19
20       setSize(300,300);
21       setVisible(true);
```

"HELLO" 레이블이 컨텐트팬상의 임의의 위치로 마음대로 움직이기 위해서 컨텐트팬의 배치관리자를 삭제한다.

```
22
23          c.setFocusable(true);
24          c.requestFocus(); // 컨텐트팬이 키 입력을 받을 수 있도록 포커스 강제 지정
25
26          // 다음 코드는 컨텐트팬에 포커스를 잃은 경우 마우스를 클릭하면 다시 포커스를 얻게 함
27          c.addMouseListener(new MouseAdapter() {
28             public void mouseClicked(MouseEvent e) {
29                Component com = (Component)e.getSource(); // 마우스가 클릭된 컴포넌트
30                com.setFocusable(true);
31                com.requestFocus(); // 마우스가 클릭된 컴포넌트에게 포커스 설정
32             }
33          });
34       }
35
36       // Key 리스너 구현
37       class MyKeyListener extends KeyAdapter {
38          public void keyPressed(KeyEvent e) {
39             int keyCode = e.getKeyCode(); // 입력된 키의 키코드를 알아낸다.
40
41             // 키 코드 값(keyCode)에 따라 상,하,좌,우 키를 판별하고 la의 위치를 이동시킨다.
42             switch(keyCode) {
43                case KeyEvent.VK_UP: // UP 키
44                   la.setLocation(la.getX(), la.getY()-FLYING_UNIT); break;
45                case KeyEvent.VK_DOWN: // DOWN 키
46                   la.setLocation(la.getX(), la.getY()+FLYING_UNIT); break;
47                case KeyEvent.VK_LEFT: // LEFT 키
48                   la.setLocation(la.getX()-FLYING_UNIT, la.getY()); break;
49                case KeyEvent.VK_RIGHT: // RIGHT 키
50                   la.setLocation(la.getX()+FLYING_UNIT, la.getY()); break;
51             }
52          }
53       }
54
55       public static void main(String [] args) {
56          new FlyingTextEx();
57       }
58    }
```

getKeyCode()를 사용하는 이유는 상, 하, 좌, 우 키는 모두 유니코드 키가 아니기 때문이다.

<Up> 키인 경우 la의 현재 위치에서 FLYING_UNIT 픽셀만큼 위로 이동시킨다.



CHEC
TIM

1 KeyEvent의 메소드 중 <Delete> 키를 인식하는 데 사용되는 것은 어떤 메소드인가?
① getKeyChar() ② getKeyCode() ③ getKeyText()

2 # 키가 입력되면 프로그램을 종료시키는 MyKeyListener 클래스를 작성하라.

3 <Home> 키를 입력하였을 때 호출되지 않는 메소드는?
① keyPressed() ② keyReleased() ③ keyTyped()

4 컴포넌트가 키 이벤트를 받기 위해 갖추어야 할 조건이 아닌 것은?
① 컴포넌트에 Key 이벤트 리스너가 등록되어 있어야 한다.
② 컴포넌트에 포커스가 지정되어 있어야 한다.
③ KeyListener의 keyPressed() 메소드가 반드시 구현되어 있어야 한다.

5 컴포넌트 component에 강제로 포커스를 주기 위해 필요한 코드를 모두 골라라.
① component.requestFocus();
② component.setVisible(true);
③ component.setFocusable(true);
④ component.addKeyListener(listener);

10.6 MouseEvent와 MouseListener, MouseMotionListener, MouseWheelListener

Mouse 이벤트

Mouse 이벤트는 사용자의 마우스 조작에 따라 〈표 10-6〉과 같이 총 8가지 경우에 발생한다. 이 중에서 5가지 경우는 MouseListener의 메소드가 호출되고, 2가지 경우는 MouseMotionListener의 메소드가, 나머지 1가지는 MouseWheelListener의 메소드가 호출된다. 모든 스윙 컴포넌트가 Mouse 이벤트를 받을 수 있으며, Mouse 이벤트가 발생하면 〈표 10-6〉과 같이 MouseEvent 객체나 MouseWheelEvent 객체가 리스너의 메소드에 전달된다.

Mouse 이벤트
8가지 경우

〈표 10-6〉
Mouse 이벤트가 발생하는 경우와 리스너, 메소드

Mouse 이벤트가 발생하는 경우	리스너의 메소드	리스너
마우스가 컴포넌트 위에 올라갈 때	void mouseEntered(MouseEvent e)	MouseListener
마우스가 컴포넌트에서 내려올 때	void mouseExited(MouseEvent e)	MouseListener
마우스 버튼이 눌러졌을 때	void mousePressed(MouseEvent e)	MouseListener
눌러진 버튼이 떼어질 때	void mouseReleased(MouseEvent e)	MouseListener
마우스로 컴포넌트를 클릭하였을 때	void mouseClicked(MouseEvent e)	MouseListener
마우스가 드래그되는 동안	void mouseDragged(MouseEvent e)	MouseMotionListener
마우스가 움직이는 동안	void mouseMoved(MouseEvent e)	MouseMotionListener
마우스 휠이 구르는 동안	void mouseWheelMoved (MouseWheelEvent e)	MouseWheelListener

마우스가 눌러진 위치에서 그대로 떼어지면 다음 순서로 메소드가 호출된다.

```
mousePressed(), mouseReleased(), mouseClicked()
```

하지만 마우스가 드래그되면 다음 순서로 호출된다.

```
mousePressed(), mouseDragged(), ..., mouseDragged(), mouseReleased()
```

드래그되는 동안 mouseDragged()는 반복적으로 여러 번 호출되며, 눌러진 위치와 떼어진 위치가 달라 mouseClicked()는 호출되지 않는다(Tip 참조).



Tip mouseClicked()와 mouseReleased()는 어떻게 다른가?

mouseClicked()와 mouseReleased()는 둘 다 마우스 버튼이 떼어지는 순간에 호출되는 메소드이지만 서로 다르다. mouseClicked()는 마우스가 눌러진 위치에서 그대로 떼어질 때 호출된다. 하지만 mouseReleased()는 마우스가 눌러진 위치와 떼어진 위치가 같든 다르든 항상 호출된다. 하물며 마우스가 눌러진 컴포넌트를 벗어나는 경우에도 호출된다. 컴포넌트 A에 마우스를 누른 상태로 컴포넌트 B 혹은 자바 프로그램의 영역을 벗어나서 마우스를 떼어도 mouseReleased()는 호출되며, 이벤트 소스 역시 컴포넌트 A이다.

마우스 리스너 달기

마우스 리스너를 작성하고 달 때 조심해야 하는 점이 있다. 〈표 10-6〉의 처음 5가지 경우(mouseEntered(), mouseExited(), mousePressed(), mouseReleased(), mouseClicked())의 Mouse 이벤트를 처리하고자 한다면, Mouse 리스너를 작성하여 컴포넌트에 다음과 같이 등록한다.

```
component.addMouseListener(myMouseListener);
```

그러나 동일한 컴포넌트가 마우스 드래깅(mouseDragged())과 마우스 무브(mouseMoved()) 이벤트도 함께 처리하고자 하면, 다음과 같이 MouseMotion 리스너를 따로 등록해야 한다([그림 10-10] 코드 참조).

```
component.addMouseMotionListener(myMouseMotionListener);
```

MouseEvent 객체

MouseEvent 객체는 Mouse 이벤트나 MouseMotion 이벤트 정보를 제공하는 객체이다. MouseEvent 객체의 메소드를 이용하여 이벤트 정보를 얻어 보자.

● 마우스 포인터의 위치

다음 메소드는 컴포넌트 안에 마우스가 놓인 좌표를 제공한다.

```
int getX() 마우스 포인터의 x 위치 리턴
int getY() 마우스 포인터의 y 위치 리턴
Point getPoint() 마우스 포인터의 위치를 Point 객체로 리턴. point 객체에 x,y 정보 있음
```

마우스의 위치 (x, y)는 다음과 같이 알아내며, (x, y)는 마우스 이벤트가 발생한 컴포넌트 내 상대 좌표이다.

```
public void mousePressed(MouseEvent e) {
    int x = e.getX();
    int y = e.getY();
}
```

● 마우스 클릭 횟수

더블클릭을 인식할 때 이용되는 것으로 클릭 횟수를 리턴하는 메소드는 다음과 같다.

int getClickCount() 마우스의 클릭 횟수 리턴

마우스의 더블클릭은 다음 코드로 판단한다.

```
public void mouseClicked(MouseEvent e) {
    if(e.getClickCount() == 2) {
        // 더블클릭을 처리하는 코드
    }
}
```

● 마우스 버튼

눌러진 마우스 버튼을 리턴하는 메소드는 다음과 같다.

int getButton() 눌러진 마우스 버튼의 번호를 리턴한다.
- 리턴 값은 NOBUTTON, BUTTON1, BUTTON2, BUTTON3 중의 하나이며 마우스 왼쪽 버튼은 BUTTON1이고 오른쪽 버튼은 BUTTON3이다.

getButton() 메소드를 이용하여 마우스 왼쪽 버튼이 눌러졌는지 다음과 같이 판단할 수 있다.

```
public void mousePressed(MouseEvent e) {
    if(e.getButton() == MouseEvent.BUTTON1)
        System.out.println("Left Button Pressed");
}
```

Mouse 이벤트 처리 예

이 절에서는 7가지의 Mouse 이벤트를 처리하는 예를 보인다. 컨텐트팬에 텍스트 정보를 출력하기 위해 JLabel 컴포넌트를 부착하고 [그림 10-10]와 같이 마우스가 눌러지는 순간, 떼어지는 순간, 움직이는 동안, 드래그하는 동안, 리스너 메소드와 마우스 포인트 좌표 값을 출력한다. 마우스가 컨텐트팬에 올라가면 배경색을 CYAN으로, 컨텐트팬에서 내려가면 YELLOW 색으로 변경한다. 다음은 예제 코드이다.

마우스와 마우스 모션 이벤트 활용 예제 10-9

이 예제는 7개의 마우스 이벤트를 처리하는 코드 사례를 보여준다.

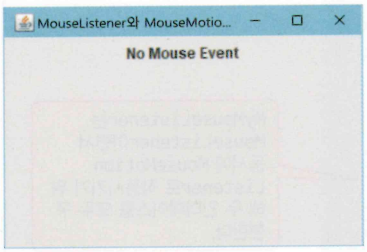
```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  public class MouseListenerAllEx extends JFrame {
6      private JLabel la = new JLabel("No Mouse Event"); // 메시지 출력 레이블 컴포넌트
7
8      public MouseListenerAllEx() {
9          setTitle("MouseListener와 MouseMotionListener 예제");
10         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11
12         Container c = getContentPane();
13         c.setLayout(new FlowLayout());
14
15         MyMouseListener listener = new MyMouseListener(); // 리스너 객체 생성
16         c.addMouseListener(listener); // MouseListener 리스너 등록
17         c.addMouseMotionListener(listener); // MouseMotionListener 리스너 등록
18
19         c.add(la);
20         setSize(300,200);
21         setVisible(true);
22     }
23
24     // Mouse 리스너와 MouseMotion 리스너를 모두 가진 리스너 작성
25     class MyMouseListener implements MouseListener, MouseMotionListener {
26         // MouseListener의 5개 메소드 구현
27         public void mousePressed(MouseEvent e) {
28             la.setText("mousePressed ("+e.getX()+","+e.getY()+")"); // 마우스
                                                    가 눌러진 위치 (x,y) 점을 출력
29         }
30         public void mouseReleased(MouseEvent e) {
31             la.setText("MouseReleased("+e.getX()+","+e.getY()+")"); // 마우스
                                                    가 떼어진 위치 (x,y) 점을 출력
32         }
33         public void mouseClicked(MouseEvent e) {}
34         public void mouseEntered(MouseEvent e) {
35             Component c = (Component)e.getSource(); // 마우스가 올라간 컴포넌트를 알아낸다.
36             c.setBackground(Color.CYAN);
37         }
38         public void mouseExited(MouseEvent e) {
39             Component c = (Component)e.getSource(); // 마우스가 내려간 컴포넌트를 알아낸다.
```

MyMouseListener를 Mouse 리스너와 MouseMotion 리스너로 로 등록하여야 한다.

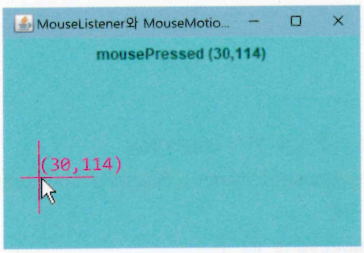
MyMouseListener는 MouseListener이면서 동시에 MouseMotion Listener로 작동시키기 해 두 인터페이스를 모두 구현한다.

e.getX()는 마우스가 클 된 x 좌표를, e.getY()는 좌표를 각각 리턴한다.

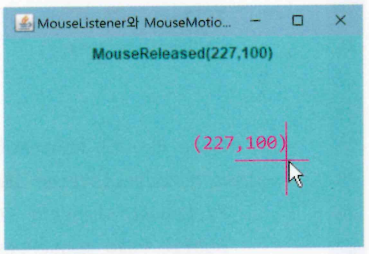
```
40              c.setBackground(Color.YELLOW);
41          }
42
43          // MouseMotionListener의 2개 메소드 구현
44          public void mouseDragged(MouseEvent e) { // 마우스가 드래깅되는 동안 계속 호출
45              la.setText("MouseDragged ("+e.getX()+","+e.getY()+")");
46          }
47          public void mouseMoved(MouseEvent e) { // 마우스가 움직이는 동안 계속 호출
48              la.setText("MouseMoved ("+e.getX()+","+e.getY()+")");
49          }
50      }
51
52      public static void main(String [] args) {
53          new MouseListenerAllEx();
54      }
55  }
```



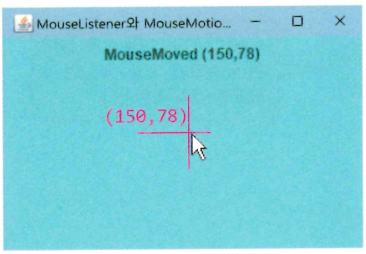
초기 화면



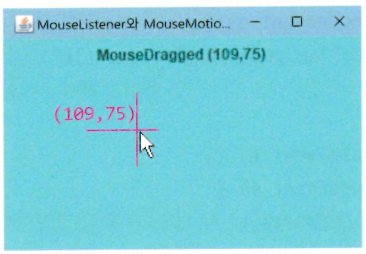
mouseEntered()에 의해 배경색 변경.
마우스 버튼이 눌러진 순간



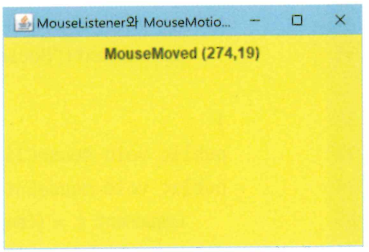
눌러진 마우스 버튼이 떼어진 순간



마우스가 패널 위에 이동하는 동안



마우스가 패널 위에 드래그하는 동안



마우스가 패널 바깥으로 나가면
mouseExited()에 의해 배경색 변경

[그림 10-10] 예제 10-9에서 7가지 마우스 이벤트를 처리하는 예

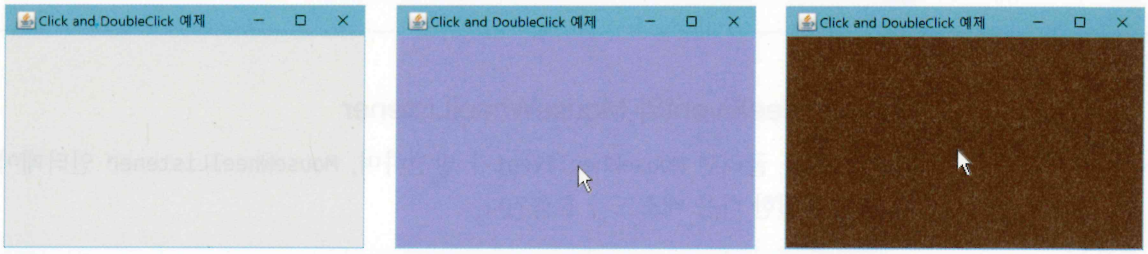
예제 10-10 더블클릭할 때마다 컨텐트팬의 배경색이 랜덤하게 변하는 예

더블클릭이 발생할 때마다 컨텐트팬의 배경색이 랜덤하게 변하도록 스윙 응용프로그램을 작성하라.

임의의 색을 생성하는 코드는 다음과 같이 작성하면 된다.

```
int r = (int)(Math.random()*256);
int g = (int)(Math.random()*256);
int b = (int)(Math.random()*256);
Color c = new Color(r,g,b); // c는 랜덤한 색
```

여기서 r, g, b는 색을 구성하는 Red, Green, Blue 성분의 양으로 [0, 255] 사이의 정수 값이다. 랜덤한 색을 얻기 위해 랜덤한 r, g, b 값을 얻어야 한다. 먼저 0~1 사이(1보다 작은)의 double 형의 난수를 리턴하는 Math.random()을 호출하고, 여기에 256을 곱하면 0에서 256보다 작은 실수 값이 형성된다. 이 값을 int 형으로 변환하면 소수점이 떨어져 나가고 [0, 255] 사이의 정수 값을 얻을 수 있다.




```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  public class ClickAndDoubleClickEx extends JFrame {
6     public ClickAndDoubleClickEx() {
7        setTitle("Click and DoubleClick 예제");
8        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9
10       Container c = getContentPane();
11       c.addMouseListener(new MyMouseListener()); // Mouse 리스너 달기
12       setSize(300,200);
13       setVisible(true);
14    }
15
16    // MouseAdapter를 상속받아 Mouse 리스너 구현
17    class MyMouseListener extends MouseAdapter {
18       public void mouseClicked(MouseEvent e) {
19          if(e.getClickCount() == 2) { // 더블클릭의 경우
20             // [0,255] 사이의 랜덤한 r,g,b 정수 값 얻기
21             int r = (int)(Math.random()*256);
```

```
            int g = (int)(Math.random()*256);
            int b = (int)(Math.random()*256);

            Component c = (Component)e.getSource(); // 마우스가 클릭된 컴포넌트
            c.setBackground(new Color(r,b,g)); // 배경을 r,g,b 색으로 칠한다.
        }
    }
  }

  public static void main(String [] args) {
    new ClickAndDoubleClickEx();
  }
}
```

MouseWheelEvent와 MouseWheelListener

마우스 휠을 굴리면 MouseWheelEvent가 발생하며, MouseWheelListener 인터페이스가 가진 유일한 다음 메소드가 호출된다.

```
public void mouseWheelMoved(MouseWheelEvent e)
```

다음은 마우스 휠 이벤트를 처리하는 리스너를 익명 클래스로 만든 사례이다.

```
component.addMouseWheelListener(new MouseWheelListener() {
    public void mouseWheelMoved(MouseWheelEvent e) {
        // 휠의 움직임에 따라 이벤트를 처리한다.
    }
});
```

MouseWheelEvent 처리는 실습문제 10-7의 힌트를 참고하라(정답 공개).



1. 마우스의 더블클릭은 MouseEvent 객체의 getClickCount()의 리턴 값이 2인지 비교하여 판단한다. 이를 위해 mousePressed()나 mouseReleased() 대신 mouseClicked()를 사용해야 하는 이유는 무엇인가?

2. JButton 컴포넌트를 컨텐트팬에 삽입하고 버튼에 마우스 왼쪽 버튼이 눌러지면 버튼의 배경색을 빨간색으로, 오른쪽 버튼이 눌러지면 파란색으로 바꾸는 응용프로그램을 작성하라.

요약



SUMMARY

- 이벤트 기반 프로그래밍은 이벤트의 발생에 의해 프로그램 실행 흐름이 결정되는 방식의 프로그래밍 패러다임이며, AWT나 스윙 등 자바의 GUI 프로그래밍은 이벤트 기반 프로그래밍이다.
- 이벤트는 키 입력, 마우스 클릭, 마우스 드래그 등의 사용자 액션이나, 센서 등 외부 장치로부터의 입력, 네트워크상의 다른 컴퓨터로부터의 데이터 송수신, 혹은 다른 스레드나 프로그램으로부터의 메시지 송수신 등에 의해 발생한다.
- 이벤트 소스란 이벤트를 발생시킨 컴포넌트이다.
- 이벤트 리스너란 이벤트를 처리하는 코드를 담은 객체이다.
- 자바에서 다양한 이벤트 리스너 인터페이스를 제공하며, 개발자는 이벤트 리스너를 만들기 위해 이벤트 리스너 인터페이스의 모든 멤버를 반드시 구현하여야 한다.
- 이벤트 객체란 발생한 이벤트에 관한 여러 정보를 가진 객체로서 이벤트 발생 시 이벤트 리스너에게 전달된다.
- 하나의 컴포넌트는 여러 종류의 이벤트에 대해 리스너를 가질 수 있으며, 하나의 이벤트에 대해 여러 개의 서로 다른 이벤트 리스너를 가질 수 있다.
- 이벤트 객체로부터 이벤트 소스 정보를 알아내기 위해 이벤트 객체의 getSource()를 호출하면 된다.
- 이벤트 리스너를 작성하는 방법은 독립된 클래스로 작성하는 방법, 내부 클래스(inner class)로 작성하는 방법, 익명 클래스(anonymous class)로 작성하는 방법 등 3가지 방법이 있다.
- 내부 클래스로 이벤트 리스너를 작성하면 자신을 둘러싼 외부 클래스의 멤버를 마음대로 호출할 수 있기 때문에 유용하다.
- 이벤트 리스너 객체를 여러 컴포넌트에 등록하여 사용하는 경우가 아니라면 이벤트 리스너를 익명 클래스로 작성하는 것이 좋다.
- 어댑터 클래스(Adapter)는 이벤트 리스너의 모든 멤버가 단순 리턴하도록 구현한 클래스로서 리스너의 모든 메소드를 구현할 필요가 없는 경우에 상속받아 사용하면 편리하다.
- Key 이벤트는 모든 컴포넌트에 발생 가능하다. 그러나 오직 현재 포커스를 가진 하나의 컴포넌트만 이벤트를 받는다. 즉, 키 입력은 포커스를 가진 컴포넌트의 키 이벤트 리스너에 의해 처리된다.
- 컴포넌트에게 포커스를 강제로 주기 위해 다음 2개의 코드가 모두 필요하다.

```
component.setFocusable(true);
component.requestFocus();
```

- KeyListener는 다음 3개의 메소드를 가지고 있다.



SUMMARY

요약

```
void keyPressed(KeyEvent e)   // 키를 누른 순간에 발생하는 Key 이벤트 처리
void keyReleased(KeyEvent e)  // 눌러진 키를 놓는 순간에 발생하는 Key 이벤트 처리
void keyTyped(KeyEvent e)     // 유니코드 키가 입력된 경우, 눌러진 키를 놓는 순간
```

- KeyEvent는 Key 이벤트 객체이다.
- KeyEvent는 유니코드 키가 입력된 경우, 키의 문자 코드를 리턴하는 다음 메소드를 가지고 있다.

```
char getKeyChar()
```

- KeyEvent는 모든 키에 대해 입력된 키의 코드 값을 리턴하는 다음 메소드를 가지고 있다.

```
int getKeyCode()
```

문자 키가 아닌 제어 키의 경우 입력된 키를 판별하기 위해서는 입력된 키 코드 값과 가상 키(Virtual Key) 값을 비교하여야 한다. 가상 키는 KeyEvent 클래스에 상수로 정의되어 있다.

- Mouse 이벤트는 모든 컴포넌트에서 처리 가능하며 마우스 조작에 따라 7가지 경우에 발생한다. 처음 5개 MouseListener에서, 뒤의 2개는 MouseMotionListener에서 정의된다.

```
void mouseEntered(MouseEvent e)   // 마우스가 컴포넌트 위에 올라갈 때
void mouseExited(MouseEvent e)    // 마우스가 컴포넌트에서 내려올 때
void mousePressed(MouseEvent e)   // 마우스 버튼이 눌러졌을 때
void mouseReleased(MouseEvent e)  // 마우스 버튼이 떼어질 때
void mouseClicked(MouseEvent e)   // 마우스로 컴포넌트를 클릭하였을 때
void mouseDragged(MouseEvent e)   // 마우스가 드래그되는 동안
void mouseMoved(MouseEvent e)     // 마우스가 움직이는 동안
```

- MouseListener의 mouseClicked()는 동일한 컴포넌트에 대해 마우스 버튼이 눌러지고 떼어지는 정상적인 마우스 클릭의 경우에 호출되는 메소드이다.
- MouseEvent는 Mouse 이벤트 객체이다.
- MouseEvent는 마우스 포인터의 위치를 알려주는 다음 메소드를 가지고 있다.

```
int getX(), int getY(), Point getPoint()
```

- MouseEvent는 클릭된 마우스 버튼 정보를 알려주는 다음 메소드를 가지고 있다.

```
int getButton()
```

- MouseEvent는 마우스 클릭 횟수를 알려주는 다음 메소드를 가지고 있다.

```
int getClickCount()
```

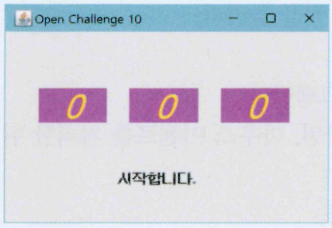
Open Challenge

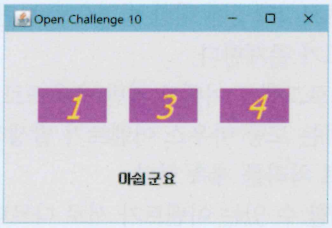
갬블링 게임 만들기

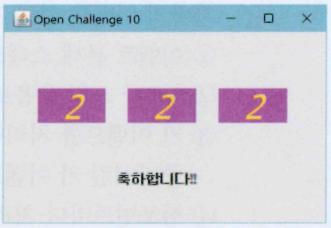


스윙으로 간단한 갬블링 게임을 만들어보자. 아래 그림과 같이 3개의 레이블 컴포넌트가 있다. 이 컴포넌트에는 각각 0~4까지의 한 자릿수가 출력된다. 〈Enter〉 키를 입력할 때마다 3개의 수를 랜덤하게 발생시켜 각 레이블에 출력한다. 그러고 나서 모두 동일한 수이면 "축하합니다!!"를, 아니면 "아쉽군요"를 출력한다. 난이도 6

목적
스윙 컴포넌트 활용, 키 리
작성







힌트 Hint

- JPanel을 상속받은 새로운 GamePanel 클래스를 작성하여 컨텐트팬으로한다. GamePanel의 배치관리자를 null로 설정하여, 3개의 레이블과 결과 레이블의 크기와 위치를 임의로 지정한다.
- GamePanel이 <Enter> 키를 받을 수 있도록 setFocusable(true)와 requestFocus() 메소드를 호출하여 포커스를 설정한다.
- 0~4 범위의 랜덤 수를 발생시키기 위해서는 다음과 같이 한다.

```
int x = (int)(Math.random()*5);
```

- Key 리스너를 작성하고 GamePanel에 등록한다. Key 리스너에는 keyPressed()만 구현하고 <Enter> 키가 입력되었는지 다음과 같이 검사한다.

```
if(e.getKeyChar() == '\n')
```

그러고 나서 0~4까지 랜덤한 수를 3개 발생시켜 각각 레이블 컴포넌트에 출력한다. 3개의 수가 같은지 검사하고 결과 레이블에 "축하합니다!!" 혹은 "아쉽군요"를 출력한다.

연습문제 EXERCISE



이론문제

• 홀수 문제는 정답이 공개됩니다.

1. 자바의 이벤트 기반 프로그래밍에 대한 설명으로 틀린 것을 골라라.
   ① 이벤트 분배 스레드가 존재한다.
   ② AWT나 스윙 응용프로그램은 이벤트 기반 응용프로그램이다.
   ③ 키 이벤트를 처리하는 도중 마우스 이벤트가 발생하면, 마우스 이벤트를 처리한 뒤 중단시킨 키 이벤트 처리를 계속 한다.
   ④ 컴포넌트마다 처리할 수 있는 이벤트가 서로 다르다.

2. MouseEvent 객체가 제공하지 않는 정보는 무엇인가?
   ① 이벤트 소스 ② 마우스 클릭된 화면 좌표
   ③ 클릭된 마우스 버튼 번호 ④ 마우스 드래깅 길이

3. 다음 프로그램 코드를 익명 클래스를 이용하여 다시 작성하라.

```
JButton btn = new JButton("Hello");
btn.addActionListener(new MyActionListener());

class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Click");
    }
}
```

4. 다음 프로그램 코드를 익명 클래스를 이용하여 다시 작성하라.

```
JButton btn = new JButton("Hello");
btn.addKeyListener(new MyKeyListener());

class MyKeyListener extends KeyAdapter {
    public void keyReleased(KeyEvent e) {
        System.out.println("Key Released");
    }
}
```

5. 다음 프로그램에서 틀린 부분을 찾아 수정하라.

(1)

```
class MyActionListener extends ActionAdapter {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Click");
}
```

(2)

```
class MyMouseListener implements MouseListener {
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse Pressed");
    }
}
```

(3)

```
class MyKeyListener extends KeyAdapter {
    public void keyTyped(ActionEvent e) {
        System.out.println("Key Pressed");
    }
}
```

6. 다음 Action 이벤트 리스너 코드가 있다.

```
class MyActionListener implements ActionListener {
    private String msg;
    public MyActionListener(String msg) { this.msg = msg; }
    public void actionPerformed(ActionEvent e) { System.out.println(msg); }
}
```

"Hello" 버튼에 다음과 같이 리스너를 등록하고, 버튼을 클릭하면 실행 결과는 무엇인가?

```
JButton btn = new JButton("Hello");
btn.addActionListener(new MyActionListener("1"));
btn.addActionListener(new MyActionListener("2"));
btn.addActionListener(new MyActionListener("3"));
```

① 3 2 1이 순서대로 출력된다.
② 1 2 3이 순서대로 출력된다.
③ 3만 출력된다.
④ 1만 출력된다.

7. 다음 중 어댑터 클래스를 가지지 않은 리스너는 무엇인가?

① ItemListener ② KeyListener
③ FocusListener ④ WindowListener

8. component가 포커스를 가지고 있어야 키 이벤트를 처리할 수 있다. component에 포커스를 강제로 설정하는 코드는 무엇인가?

① component.setFocus(); ② component.focusRequest();
③ component.requestFocus(); ④ component.focus();

9. 다음 중에서 유니코드 키가 아닌 것을 모두 골라라.

```
a  <Alt>  9  %  <Tab>  @  <Delete>  ;  <Shift>  ~  <Help>
```

10. 다음 코드는 a 키가 눌러졌는지 판별하기 위한 처리 코드이다.

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyChar() == _____) {
        System.exit(0);
    }
}
```

(1) 빈칸을 채워라.
(2) 2번째 라인을 <ALT> 키가 눌러졌는지 판별하는 코드로 수정하라.

11. 키 이벤트가 발생할 때 KeyListener의 keyPressed(), keyReleased() keyTyped() 메소드가 호출되는 순서에 대해 답하라.

(1) 사용자가 a 키를 입력할 때 (2) 사용자가 <Esc> 키를 입력할 때

12. 다음 마우스 리스너가 있다.

```
class MyMouseListener extends MouseAdapter {
    private int count=0;
    public void mousePressed(MouseEvent e) { count++; }
    public void mouseReleased(MouseEvent e) { System.out.print(count); }
    public void mouseDragged(MouseEvent e) { count++; }
}
```

그리고 컨텐트팬에 다음과 같이 Mouse 리스너와 MouseMotion 리스너를 등록하였다.

```
contentPane.addMouseListener(new MyMouseListener());
contentPane.addMouseMotionListener(new MyMouseListener());
```

(1) 컨텐트팬에 마우스를 한 번 누르고 놓았을 때 화면에 출력되는 내용은?
(2) 컨텐트팬에 마우스를 누른 후 드래그하고 마우스를 놓았을 때 출력되는 내용은?
(그동안 mouseDragged()가 10번 실행되었다.)
(3) 마우스 리스너 등록 코드를 다음과 같이 바꾸었을 때 (1), (2)번에 대한 결과는?

```
MyMouseListener ml = new MyMouseListener();
contentPane.addMouseListener(ml);
contentPane.addMouseMotionListener(ml);
```

13. 스윙 컴포넌트 component에 대해 다음 문장의 의미는 각각 무엇인가?

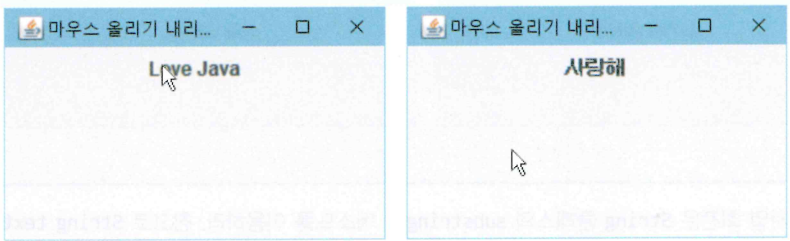
```
component.repaint();
component.revalidate();
```

실습문제

• 홀수 문제는 정답이 공개됩니다.

1. JLabel 컴포넌트는 Mouse 이벤트를 받을 수 있다. JLabel 컴포넌트에 마우스를 올리면 "Love Java"가, 내리면 "사랑해"가 출력되도록 스윙 응용프로그램을 작성하라.

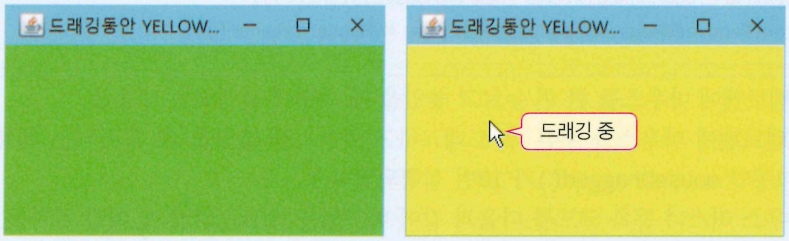
난이도 4



목적 간단한 Mouse 이벤트 리스너 작성

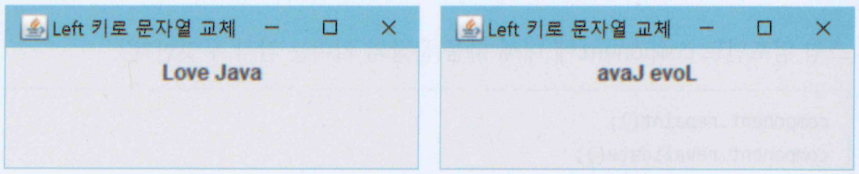
목적 Mouse 이벤트와 MouseMotion 이벤트 동시 처리

2. 컨텐트팬의 배경색은 초록색으로 하고 마우스를 드래깅하는 동안만 노란색으로 유지하는 스윙 응용프로그램을 작성하라. 난이도 5



목적 Key 리스너 작성과 포커스 연습

3. JLabel을 활용하여 "Love Java"를 출력하고 왼쪽 화살표 키(<Left> 키)를 입력할 때마다 "avaJ evoL"와 "Love Java"를 번갈아 출력하는 스윙 프로그램을 작성하라. 난이도 5

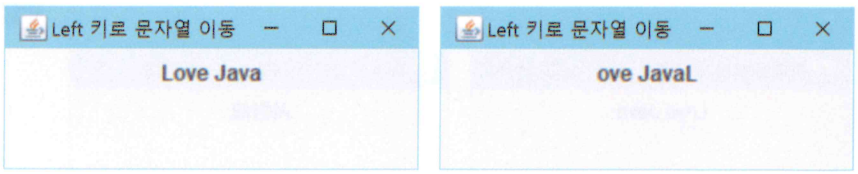


힌트 Hint

JLabel에 포커스를 설정하고 키 리스너를 작성한다. 그리고 JLabel에 출력되는 문자열을 StringBuffer를 이용하여 담고 reverse() 메소드를 호출하면 문자열을 거꾸로 바꿀 수 있다. 아니면 컨텐트팬에 리스너를 달아 처리해도 된다.

목적 Key 리스너 작성과 포커스 연습

4. JLabel을 활용하여 "Love Java"를 출력하고 왼쪽 화살표 키(<Left> 키)를 입력할 때마다 "ove JavaL", "ve JavaLo", "e JavaLov"와 같이 한 문자씩 왼쪽으로 회전하는 프로그램을 작성하라. 난이도 5

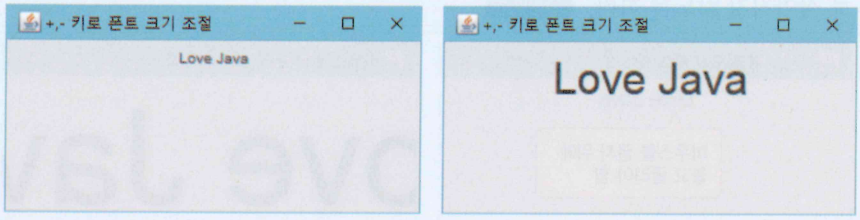


힌트 Hint

문자열 회전은 String 클래스의 substring() 메소드를 이용하라. 참고로 String text = "Love Java"에서, text.substring(0, 1)은 "L"을 리턴하고, text.substring(1)은 "ove Java"를 리턴한다. JLabel에 포커스를 설정해야 키 입력을 받을 수 있다는 것도 잊지 말라.

5. JLabel 컴포넌트로 "Love Java"를 출력하고, 키 리스너를 작성하여 + 키를 치면 폰트 크기를 5픽셀씩 키우고, - 키를 치면 폰트 크기를 5픽셀씩 줄이는 스윙 응용프로그램을 작성하라. 5픽셀 이하로 작아지지 않도록 하라. 난이도 6

목적 Key 리스너, 포커스 다루기 연습



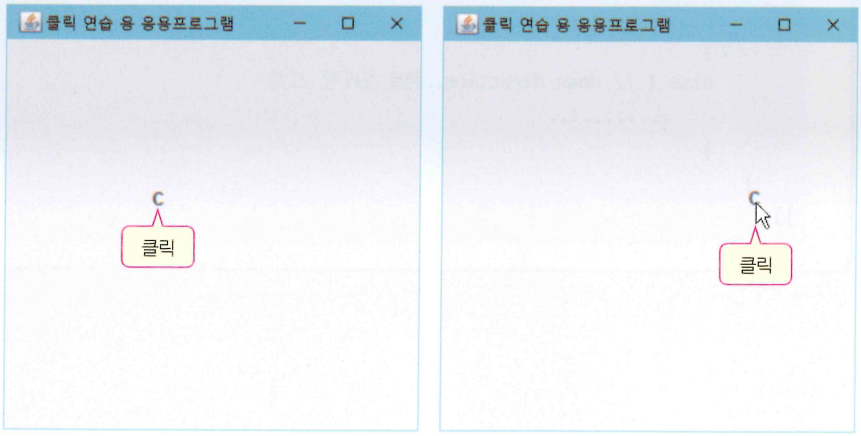
힌트 Hint

JLabel 컴포넌트에 폰트를 설정하는 방법은 다음과 같다.

```
JLabel la = new JLabel("Love Java");
la.setFont(new Font("Arial", Font.PLAIN, 10)); // Arial 폰트로 10픽셀 크기
Font f = la.getFont();
int size = f.getSize();
la.setFont(new Font("Arial", Font.PLAIN, size+5)); // 15픽셀 크기
```

6. 클릭 연습용 스윙 응용프로그램을 작성하라. "C"를 출력하는 JLabel을 하나 만들고 초기 위치를 (100, 100)으로 하고, "C"를 클릭할 때마다 컨텐트팬 내에 랜덤한 위치로 움직이게 하라. 난이도 7

목적 Mouse 리스너와 레이아웃 관리자 없는 컨텐트팬에 컴포넌트 이동 연습

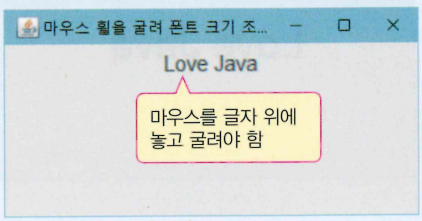


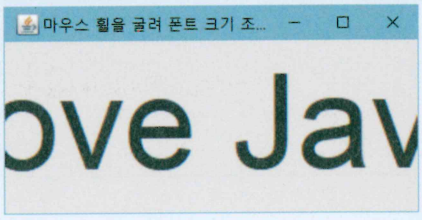
힌트 Hint

배치관리자를 삭제하면 레이블이 임의의 위치로 움직일 수 있다.

목적 MouseWheel 리스너 작성 연습

7. JLabel을 활용하여 "Love Java"를 출력하고, "Love Java" 글자 위에 마우스를 올려 마우스 휠을 위로 굴리면 글자가 작아지고, 아래로 굴리면 글자가 커지도록 프로그램을 작성하라. 폰트 크기는 한 번에 5픽셀씩 작아지거나 커지도록 하고, 5픽셀 이하로 작아지지 않도록 하라. 난이도 7





힌트 Hint

- 마우스 휠이 굴려질 때마다 MouseWheelEvent가 발생하며 MouseWheelListener의 mouseWheelMoved(MouseWheelEvent e) 메소드가 호출된다. 이때 매개 변수 e 객체를 다음과 같이 이용하면 마우스 휠이 위로 굴려졌는지 아래로 굴려졌는지 알 수 있다.

```
int n = e.getWheelRotation();
```

- n이 음수이면 마우스 휠이 위로 굴려졌고, 양수이면 아래로 굴려졌다. 실제 n은 -1 혹은 1 중 하나이다. 핵심적인 코드는 다음과 같다.

```
label.addMouseWheelListener(new MouseWheelListener() {
    public void mouseWheelMoved(MouseWheelEvent e) {
        int n = e.getWheelRotation();
        if(n < 0) { // up direction. 폰트 5픽셀 작게
            ..........
        }
        else { // down direction. 폰트 5픽셀 크게
            ..........
        }
    }
});
```