22. 이벤트처리

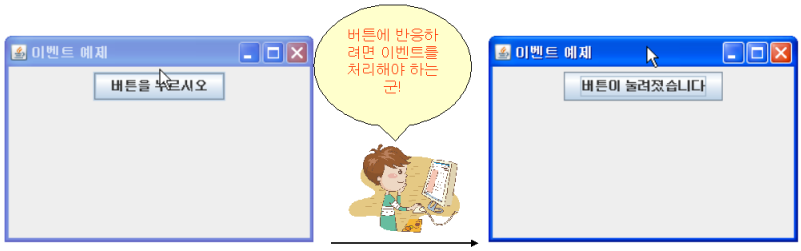
학습목표

- 버튼을 클릭할 수는 있었지만 아무 일도 일어나지 않았다. 그 이유는 버튼을 눌렀을 때 발생되는 이벤트(event)를 처리하지 않았기 때문이다.

1. 이벤트 처리의 개요

1) 이벤트란?

GUI 프로그래밍은 사용자의 입력을 기다리고 있다가 사용자가 특정 버튼을 클릭하면 작업이 진행된다. 이와 같이 프로그램의 실행이 이벤트에 의해 결정되는 방식을 **이벤트-구동 프로그래밍(event-driven programming)**이라고 한다. **이벤트(event)**는 마우스로 버튼을 클릭하는 것처럼 어떤 동작에 의하여 발생한다. 이벤트가 발생하면 해당 컴포넌트가 이벤트 객체를 생성한다. 이벤트 객체(event object)는 이벤트에 대한 여러 가지 정보를 가지고 있다. 또한 이벤트를 발생시킨 컴포넌트를 **이벤트 소스(event source)**라고 부른다.



발생된 이벤트 객체에 반응하여서 이벤트를 처리하는 객체를 **이벤트 리스너(event listener)**라고 한다. 만약 이벤트 소스가 이벤트 리스너에 연결되어 있다면 이벤트가 발생하였을 때 이벤트가 리스너 내부의 이벤트 처리 메소드로 전달된다. 따라서 이벤트가 발생했을 경우에 실행되어야 할 코드가 있다면 이벤트 처리 메소드에 넣으면 된다. 만약 연결된 리스너가 없다면 아무 일도 일어나지 않는다. 버튼을 눌렀을 때 아무 일도 일어나지 않았다면 이 버튼에 이벤트 리스너가 연결되지 않은 것이다.

이벤트 리스너 클래스를 작성하고 이 리스너 클래스의 인스턴스를 생성하여서 이것을 버튼 컴포넌트와 연결하여야 한다.

2) 이벤트 처리의 과정

① 이벤트를 발생하는 컴포넌트를 생성하여야 한다.

```
public class MyEvent extends JFrame {                // 프레임을 상속하여서 MyEvent 선언
        private JButton button;                      // 버튼을 참조하는 변수를 선언
                …
        public MyEvent() {                           // 생성자에서 컴포넌트를 생성하고 추가한다
```

```
                JPanel panel = new JPanel();          // 패널 생성
                button = new JButton("동작");         // 버튼 생성
                panel.add(button);                    // 버튼을 패널에 추가
                add(panel);                           // 패널을 프레임에 추가
                …
        }
        …
}
```

② 이벤트 리스너 클래스를 작성한다

이벤트 리스너가 되기 위해서는 리스너 인터페이스를 구현하여야 한다. 리스너 인터페이스는 어떤 클래스가 이벤트를 처리하기 위한 규격이다. 자바는 이벤트의 종류에 따라 여러 가지 리스너 인터페이스를 제공한다. 버튼의 이벤트는 액션이벤트(action event)이다. 액션이벤트를 처리하기 위한 리스너 인터페이스는 ActionListener 인터페이스이다. 따라서 버튼이 발생하는 이벤트를 처리하려면 ActionListener 인터페이스를 구현하여야 한다. ActionListener에서 actionPerformed()라는 메소드만을 구현하면 된다. 이메소드는 액션 이벤트가 발생할 때마다 호출된다.

```
class MyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                …                       // Action 이벤트를 처리하는 코드가 여기에 들어간다.
        }
}
```

③ 이벤트 리스너를 이벤트 소스에 등록한다

이벤트 리스너 객체를 이벤트 소스에 등록하는 단계이다. 각 컴포넌트는 이벤트 리스너를 등록할 수 있는 메소드를 제공한다. 버튼의 경우, addActionListener() 메소드이다.

```
public class MyEvent extends JFrame {              // 프레임을 상속하여서 MyEvent 선언
        private JButton button;                    // 버튼을 참조하는 변수를 선언
                …
        public MyEvent() {                         // 생성자에서 컴포넌트를 생성하고 추가한다
                …
                button = new JButton("동작");      // 버튼 생성
                button.addActionListener(new MyListener());      //이벤트 리스너 등록
                panel.add(button);                 // 버튼을 패널에 추가
                …
        }
        …
}
```

addActionListener()의 매개변수로 새로 생성된 MyListener 클래스의 인스턴스가 전달되었다.

3) 이벤트 객체

각 이벤트 리스너는 이벤트 객체를 매개변수로 가진다. 이벤트 객체는 발생한 이젠트에 대한 모든 정보를 리스너로 전달한다. 모든 이벤트 객체는 EventObject 클래스를 상속 받는다. EventObject 클래스는 getSource() 메소드만을 가지고 있다. getSource()는 발생한 이벤트 소스를 Object 타입으로 반환하므로, 필요한 타입으로 형변환해서 사용하면 된다.

```
java.lang.Object
   java.util.EventObject
      java.awt.AWTEvent
         java.awt.event.ComponentEvent
            java.awt.event.InputEvent
               java.awt.event.MouseEvent
```

MouseEvent 클래스

이벤트를 발생시킨 이벤트 소스 등의 여러 가지 정보를 제공한다.

```
public void actionPerformed(MouseEvent e) {
   button = (JButton)e.getSource();
      ...
}
```

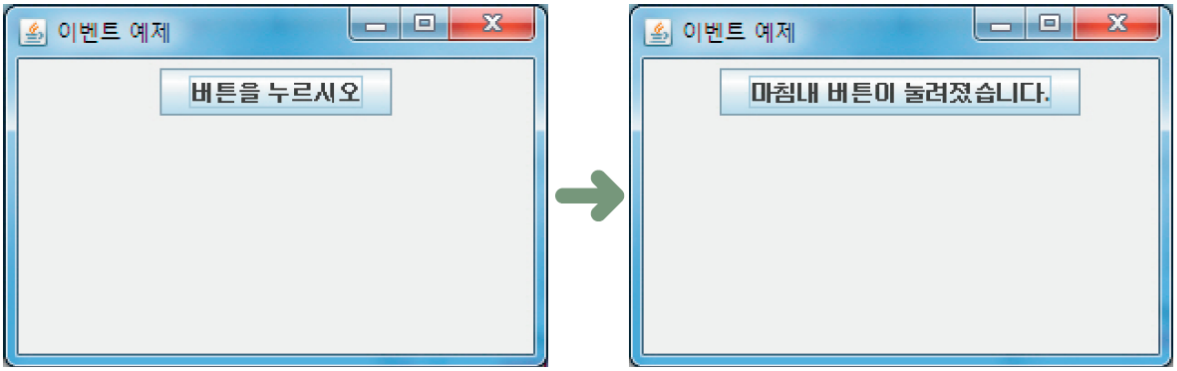
4) 리스너를 독립적인 클래스로 작성

버튼을 클릭하면 버튼의 텍스트를 변경하는 프로그램을 작성한다.

```
01  import javax.swing.*;
02  import java.awt.FlowLayout;
03  import java.awt.event.*;  <----------이벤트 처리를 위한 패키지
04
05  class MyListener implements ActionListener {
06     public void actionPerformed(ActionEvent e) {
07        JButton button = (JButton) e.getSource();  <---------- MyListener 클래스를 별도의 클래스로 정의한다. ActionListener 인터페이스를 구현한다.
08        button.setText("마침내 버튼이 눌려졌습니다.");
09     }
10  }
11  class MyFrame extends JFrame {
12     private JButton button;
13     public MyFrame() {
14        this.setSize(300, 200);
15        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
16        this.setTitle("이벤트 예제");
17        this.setLayout(new FlowLayout());
18        button = new JButton("버튼을 누르시오");
19        button.addActionListener(new MyListener());  <---------버튼에 이벤트 리스너 등록
20        this.add(button);
21        this.setVisible(true);
22     }
23  }
```

```
24  public class ActionEventTest {
25     public static void main(String[] args) {
26        MyFrame t = new MyFrame();
27     }
28  }
```

실행결과



5) 리스너 클래스를 내부 클래스로 사용하는 방법

MyListener라는 클래스를 별도의 클래스로 하면 MyFrame 안의 멤버변수들을 쉽게 사용할 수 없다. 따라서 이런 경우에는 일반적으로 MyListener 클래스를 내부 클래스로 만든다. 내부 클래스는 다른 클래스 안에 위치하는 클래스이다. 내부 클래스에서는 외부 클래스의 멤버변수들을 자유롭게 사용할 수 있다.

```
01  import javax.swing.*;
02  import java.awt.event.*; //이벤트 처리를 위한 패키지
03
04  class MyFrame extends JFrame {
05     private JButton button;
06     private JLabel label;
07
08     public MyFrame() {
09        this.setSize(300, 200);
10        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        this.setTitle("이벤트 예제");
12        JPanel panel = new JPanel();
13        button = new JButton("버튼을 누르시오");
14        label = new JLabel("아직 버튼이 눌려지지 않았습니다");
15        button.addActionListener(new MyListener());
16        panel.add(button);
17        panel.add(label);
18        this.add(panel);
19        this.setVisible(true);
20     }
21
```

button, label 변수가 멤버 변수로 선언되었다. 그 이유는 생성자와 actionPerformed() 메소드에서 사용하기 때문에 멤버 변수로 하여야 한다. 여기서는 전용 멤버로 선언하여도 된다. 내부 클래스는 전용 멤버에도 접근할 수 있다.

버튼에 이벤트 리스너 등록

MyListener 클래스는 MyFrame 클래스의 내부 클래스로 정의된다. MyListener 클래스는 Action 이벤트를 처리할 수도 있도록 ActionListener 인터페이스를 구현한다.

```
22      private class MyListener implements ActionListener {
23         public void actionPerformed(ActionEvent e) {
24            if (e.getSource() == button) {
25               label.setText("마침내 버튼이 눌려졌습니다.");
26            }
27         }
28      }
29   }
30   public class ActionEventTest {
31      public static void main(String[] args) {
32         MyFrame t = new MyFrame();
33      }
34   }
```

MyListener 클래스 안에서 actionPerformed() 메소드는 반드시 정의되어야 한다. 이 메소드는 사용자가 버튼을 누를 때마다 실행된다. 매개변수인 e는 버튼에 의하여 생성되는 이벤트 객체이다.

멤버인 label에 쉽게 접근할 수 있다.

6) MyFrame 클래스에서 이벤트를 처리하는 방법

더 많이 사용되는 방법은 MyFrame 클래스가 JFrame을 상속받으면서 동시에 ActionListener 인터페이스도 구현하는 경우이다.

MyFrameTest1.java

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

class MyEventListener extends JFrame implements ActionListener {
        private JButton button;

        public MyEventListener() {
                this.setSize(300, 200);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setTitle("이벤트 예제");
                JPanel panel = new JPanel();
                button = new JButton("버튼을 누르시오");
                button.addActionListener(this);
                panel.add(button);
                this.add(panel);
                this.setVisible(true);
        }
        public void actionPerformed(ActionEvent e) {
                if (e.getSource() == button) {
                        button.setText("버튼이 눌려졌습니다");
```

이벤트도 처리

현재 객체를 리스너로 등록

```
                }
        }
}
public class MyFrameTest1 {
        public static void main(String[] args) {
                MyEventListener mel = new MyEventListener();
        }
}
```

7) 무명 클래스를 사용하는 방법

무명 클래스는 이름없는 클래스를 작성하여 한 번만 사용하는 것이다. 이 방법은 코드를 읽기 쉽게 하고 무명 클래스가 정의되면서 바로 사용된다. 안드로이드 프로그래밍에서도 많이 사용된다.

```
01  class MyFrame extends JFrame {
02      ...
03      public MyFrame() {
04          ...
05          button = new JButton("버튼을 누르시오");
06          button.addActionListener(new ActionListener() {
07              public void actionPerformed(ActionEvent e) {
08                  if (e.getSource() == button) {
09                      label.setText("마침내 버튼이 눌려졌습니다.");
10                  }
11              }
12
13          });
14          ...
15      }
16
17  }
```

무명 클래스는 ActionListener 인터페이스를 구현한다. 무명 클래스의 객체도 동시에 생성된다.

무명 클래스를 정의한다. 무명 클래스 안에서 actionPerformed() 메소드를 정의한다.

8) EventHandler 클래스를 사용하는 방법(단점:이벤트 처리가 늦어지는 경우에는 프로그램이 멈춘다.)

EventHandler 클래스를 사용하면 하나의 문장으로 간단하게 이벤트 리스너를 생성할 수 있다. 하지만 아주 간단한 타입의 이벤트 리스너에 대해서만 유용하다. 예를 들어 버튼이 클릭되었을 때 toFront()라는 메소드를 호출하게 만드는 방법은 다음과 같다.

```
myButton.addActionListener(
        (ActionListener)EventHandler.create(ActionListener.class, frame, "toFront"));
```

만약 myButton이 클릭되면 frame.toFront()가 실행된다.

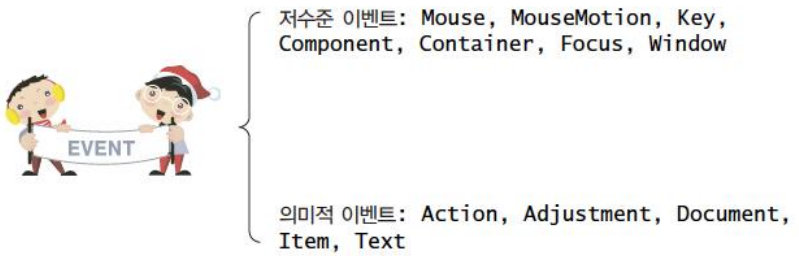
위 문장는 무명 클래스로 작성된 다음과 같은 코드와 동일한 효과를 가진다.

```
myButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        frame.toFront();
    }
});
```

2. 스윙 컴포넌트의 이벤트

1) 이벤트의 분류

이벤트는 저수준(low level) 이벤트와 의미적(semantic) 이벤트 2가지로 분류할 수 있다.



저수준(low level) 이벤트는 사용자의 직접적인 입력의 결과로 발생되는 이벤트를 의미한다. 의미적(semantic) 이벤트는 컴포넌트와 사용자간의 상호 작용의 결과로 생성되며 상위 수준의 이벤트를 말한다.

2) 모든 컴포넌트가 지원하는 이벤트

저수준 이벤트로 마우스와 키보드 이벤트이다.

이벤트 종류	설명
Component	컴포넌트의 크기나 위치가 변경되었을 경우 발생
Focus	키보드 입력을 받을 수 있는 상태가 되었을 때, 혹은 그반대의 경우에 발생
Container	컴포넌트가 컨테이너에 추가되거나 삭제될 때 발생
Key	사용자가 키를 눌렀을 때 키보드 포커스를 가지고 있는 객체에서 발생
Mouse	마우스 버튼이 클릭되었을 때, 또는 마우스가 객체의 영역으로 들어오거나 나갈 때 발생
MouseMotion	마우스가 움직였을 때 발생
MouseWheel	컴포넌트 위에서 마우스 휠을 움직이는 경우 발생
Window	윈도우에 어떤 변화가 있을 때 발생(열림, 닫힘, 아이콘화 등)

3) 일부 컴포넌트만 지원하는 이벤트

의미적 이벤트이고 액션 이벤트와 아이템 이벤트가 대표적이다. 가능하다면 저수준보다 의미적 이벤트를 사용하다.

이벤트 종류	설명
Action	사용자가 어떤 동작을 하는 경우에 발생
Caret	텍스트 삽입점이 이동하거나 텍스트 선택이 변경되었을 경우 발생
Change	일반적으로 객체의 상태가 변경되었을 경우 발생
Document	문서의 상태가 변경되는 경우 발생
Item	선택 가능한 컴포넌트에서 사용자가 선택을 하였을 때 발생
ListSelection	리스트나 테이블에서 선택 부분이 변경되었을 경우에 발생

컴포넌트	이벤트					
	Action	Caret	Change	Document	Item	ListSelection
버튼(button)	V		V		V	
체크 박스(check box)	V		V		V	
색상 선택(color chooser)			V			
콤보 박스(combo box)	V				V	
다이알로그(dialog)						
파일 선택(file chooser)	V					
프레임(frame)						
리스트(list)						V
메뉴 항목(menu item)	V		V		V	
진행바(progress bar)			V			
라디오 버튼(radio button)	V		V		V	
슬라이더(slider)			V			
스피너(spinner)			V			
테이블(table)						V
텍스트 영역(text area)		V		V		
펙스트 필드(text field)	V	V		V		

4) 리스너 인터페이스의 요약

리스너 인터페이스	어댑터 클래스	메소드
ActionListener	none	actionPerformed()
AdjustmentListener	none	adjustmentValueChanged()
ComponentListener	ComponentAdapter	componentHidden() componentMoved() componentResized() componentShown()
ContainerListener	ContainerAdapter	componentAdded() componentRemoved()
FocusListener	FocusAdapter	focusGained() focusLost()
ItemListener	none	itemStateChanged()
KeyListener	KeyAdapter	keyPressed() keyReleased() keyTyped()
MouseListener	MouseAdapter	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()
MouseMotionListener	MouseMotionAdapter	mouseDragged() mouseMoved()
TextListener	none	textValueChanged()
WindowListener	WindowAdapter	windowActivated() windowClosed() windowClosing() windowDeactivated() windowDeiconified() windowIconified() windowOpened()

3. 액션 이벤트

1) 액션 이벤트

액션 이벤트는 대표적인 이벤트로서 다음과 같은 경우에 발생한다.

- 사용자가 버튼을 클릭하는 경우
- 사용자가 메뉴 항목을 선택하는 경우
- 사용자가 텍스트 필드에서 엔터키를 누르는 경우

- ActionListener 인터페이스

메소드	설명
`void actionPerformed(ActionEvent e)`	사용자가 어떤 동작을 수행하는 경우 호출

- ActionEvent 클래스

메소드	설명
`Object getSource()`	이벤트를 생성한 객체 반환
`String getActionCommand()`	이벤트를 발생시킨 컴포넌트 객체와 관련된 문자열을 반환한다. 즉 버튼 컴포넌트의 경우에는 버튼의 텍스트가 반환된다.

`int getModifier()`	이벤트가 생성될 때의 키보드의 상태를 반환한다. 즉 SHIFT_MASK, CTRL_MASK, META_MASK, ALT_MASK 등을 이용하여 Shift 키, Ctrl 키, Meta 키, Alt 키 등의 상태를 알 수 있다. 이벤트가 발생하였을 시점에 Ctrl 키가 눌려져 있었는지를 알아내려면 다음과 같이 한다.
	`void actionPerformed(ActionEvent e){`
	`  if (e.getModifier() & ActionEvent.CTRL_MASK){`
	`  // Ctrl 키가 눌려져 있었을 경우의 처리`
	`  ...`
	`  }`
	`  }`
	`}`

2) 이벤트 발생원의 식별

ActionEvent와 같이 여러 개의 컴포넌트가 같은 이벤트를 발생시키는 경우, 이벤트의 발생원(source)을 찾는 작업이 필요하게 된다. 이벤트의 발생원을 찾기 위해서는 다음과 같은 메소드를 사용할 수 있다.

- getSource() 메소드를 이용하여 이벤트를 발생시킨 객체를 식별한다.
- getId() 메소드를 이용하여 이벤트의 타입을 식별한다.
- getActionCommand() 메소드를 이용하여 이벤트를 발생시킨 컴포넌트 이름을 식별한다.

(예제)

MyFrameTest2.java

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

```
class MyActionEvent extends JFrame {
	private JButton button1;
	private JButton button2;
	private JPanel panel;
	MyListener listener = new MyListener();    // 리스너 객체 생성

	public MyActionEvent() {
		this.setSize(300, 200);
		this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		this.setTitle("이벤트 예제");
		panel = new JPanel();
		button1 = new JButton("노란색");
		button1.addActionListener(listener);       // 액션 이벤트 처리 등록
		panel.add(button1);
		button2 = new JButton("핑크색");
		button2.addActionListener(listener);       // 액션 이벤트 처리 등록
		panel.add(button2);
		this.add(panel);
		this.setVisible(true);
	}
	private class MyListener implements ActionListener {       // 내부 클래스
		public void actionPerformed(ActionEvent e) {
			if (e.getSource() == button1) {            // 이벤트 소스가 button1이면
				panel.setBackground(Color.YELLOW);
			} else if (e.getSource() == button2) {     // 이벤트 소스가 button2이면
				panel.setBackground(Color.PINK);
			}
		}
	}
}
public class MyFrameTest2 {
	public static void main(String[] args) {
		MyActionEvent mae = new MyActionEvent ();
	}
}
```

4. Key 이벤트

Key 이벤트는 사용자가 키보드를 이용하여 입력을 하는 경우에 발생한다. 키를 누를 때도 발생하지만 키에 손을 떼는 경우에도 발생한다. KeyEvent가 발생하려면 반드시 포커스를 가지고 있어야 한다. 키보드 포커스를 얻으려면 requestFocus()라는 메소드를 사용한다. KeyEvent를 받기 위해서는 KeyListener를 구현하여야 한다.

- KeyListener 인터페이스

메소드	설명
keyTyped(KeyEvent e)	사용자가 글자를 입력했을 경우에 호출
keyPressed(KeyEvent e)	사용자가 키를 눌렀을 경우에 호출
keyReleased(KeyEvent e)	사용자가 키에서 손을 떼었을 경우에 호출

- KeyEvent 클래스

메소드	설명
int getKeyChar()	KeyEvent에 들어있는 글자(유니코드)를 반환한다.
int getKeyCode()	KeyEvent에 들어있는 키코드(keycode)를 반환한다. 키코드란 글자가 아니라 키보드 자판의 각각의 키를 가리키는 상수이다. 예를 들어 Escape 키의 키코드는 VK_ESCAPE로 정의되어 있다. 예를 들어 눌려진 키가 Enter 키인가를 판단하기 위해서는 다음과 같은 문장을 이용한다. `if (e.getKeyCode() == e.VK_ENTER) { // 처리할 내용 }`
boolean isActionKey()	이벤트를 발생시킨 키가 액션 키이면 true를 반환한다. 액션 키란 Cut, Copy, Paste, Page Up, Caps Lock, 화살표와 function 키를 의미한다.

- InputEvent 클래스

메소드	설명
int getID()	이벤트의 타입을 반환한다. 가능한 타입은 다음과 같다. MouseEvent.MOUSE_PRESSED, MouseEvent.MOUSE_RELEASED, MouseEvent.MOUSE_CLICKED.
getComponent()	이벤트를 일으킨 컴포넌트를 반환한다. getSource()를 사용하여도 된다.
int getWhen()	이벤트가 발생한 시각을 반환한다.
boolean isAltDown() boolean isControlDown() boolean isMetaDown() boolean isShiftDown()	이벤트가 발생한 시각에 키보드의 수식키들의 상태를 반환한다.

(예제)

키보드에서 문자가 입력되면 문자코드와 키코드, ALT나 SHIFT 키의 상태가 반환된다.

MyKeyEventTest.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
class MyKeyEvent extends JFrame implements KeyListener {
```

```
        public MyKeyEvent() {
                setTitle("이벤트 예제");
                setSize(300, 200);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JTextField tf = new JTextField(20);
                tf.addKeyListener(this);     // 현재의 객체를 이벤트 리스너로 추가

                add(tf);
                setVisible(true);
        }
        public void keyTyped(KeyEvent e) {  // 사용자가 키를 입력했을 경우에 호출
                display(e, "KeyTyped ");
        }
        public void keyPressed(KeyEvent e) {  // 사용자가 키를 누르는 순간에 호출
                display(e, "KeyPressed ");
        }
        public void keyReleased(KeyEvent e) {  // 사용자가 키에서 손을  떼는 순간에 호출
                display(e, "Key Released ");
        }
        protected void display(KeyEvent e, String s) {
                char c = e.getKeyChar();  // 눌러진 키의 아스키 코드값을 반환
                int keyCode = e.getKeyCode();
                String modifiers = e.isAltDown() + " " + e.isControlDown() + " "+ e.isShiftDown();
                System.out.println(s + " " + c + " " + keyCode + " " + modifiers);
        }
}
public class MyKeyEventTest{
        public static void main(String[] args) {
                MyKeyEvent mke = new MyKeyEvent ();
        }
}
```

키보드 이벤트 처리기를 붙인다.

[자동차 예제 1]

CarKeyTest.java

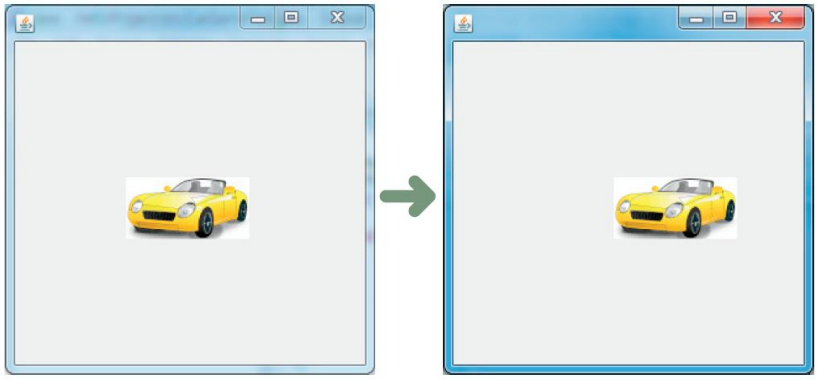
```
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
```

```
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

class CarPanel extends JPanel {
	BufferedImage img = null;
	int img_x = 100
	int img_y = 100;

	public CarPanel() {
		try {
			img = ImageIO.read(new File("car.gif"));	//60x39
		} catch (IOException e) {
			System.out.println("no image");
			System.exit(1);
		}
		addKeyListener(new KeyListener() {
			public void keyPressed(KeyEvent e) {
				int keycode = e.getKeyCode();
				switch (keycode) {
					case KeyEvent.VK_UP:	img_y -= 10;	break;
					case KeyEvent.VK_DOWN:	img_y += 10;	break;
					case KeyEvent.VK_LEFT:	img_x -= 10;	break;
					case KeyEvent.VK_RIGHT:	img_x += 10;	break;
				}
				repaint();
			}
			public void keyReleased(KeyEvent arg0) {		}
			public void keyTyped(KeyEvent arg0) {		}
		});
		this.requestFocus();
		setFocusable(true);
	}
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, img_x, img_y, null);
	}
}
public class CarKeyTest extends JFrame {
	public CarKeyTest() {
		setSize(300, 300);
```

```
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                add(new CarPanel());
                setVisible(true);
        }
        public static void main(String[] args) {
                CarKeyTest ckt = new CarKeyTest();
        }
}
```



키보드의 화살표 키를 이용하여서 화면 안의 자동차를 움직여 보자.

5. Mouse와 MouseMotion 이벤트

마우스 이벤트는 사용자가 마우스를 조작하는 경우에 발생한다. 구체적으로 마우스 이벤트는 커서가 컴포넌트의 영역에 들어가거나 빠져 나올 때, 사용자가 마우스 버튼을 누르거나 손을 뗄 때 발생한다. MouseEvent를 받기 위해서는 **MouseListener**를 구현하여야 한다. 다만 마우스 움직임을 추적하는 것은 상당히 시스템의 오버헤드를 증가시키므로 마우스가 이동하는 경우에는 따로 MouseMotion이라는 이벤트가 있으며 **MouseEventListener**를 구현해야 한다. 마우스 휠을 조작할 때는 MouseWheel 이벤트가 있다.

- MouseListener

메소드	설명
mouseClicked(MouseEvent e)	사용자가 컴포넌트를 클릭한 경우에 호출된다.
mouseEntered(MouseEvent e)	마우스 커서가 컴포넌트로 들어가면 호출된다.
mouseExited(MouseEvent e)	마우스 커서가 컴포넌트에서 나가면 호출된다.
mousePressed(MouseEvent e)	마우스가 컴포넌트위에서 눌려지면 호출된다.
mouseReleased(MouseEvent e)	마우스가 컴포넌트위에서 떼어지면 호출된다.

- MouseMotionListener

메소드	설명
mouseDragged(MouseEvent e)	마우스 드래그하면 호출된다.
mouseMoved(MouseEvent e)	마우스가 클릭되지 않고 이동하는 경우에 호출된다.

마우스의 버튼이 클릭된다면 다음과 같은 순서로 이벤트들이 발생한다.

실행결과

```
Mouse pressed (# of clicks: 1) X=118 Y=81   <-- 버튼을 눌렀을 때 발생
Mouse released (# of clicks: 1) X=118 Y=81  <-- 버튼에서 손을 떼면 발생
Mouse clicked (# of clicks: 1) X=118 Y=81   <-- 버튼이 한번 클릭되면 발생
```

마우스가 드래그된다면 위의 이벤트들이 중복해서 발생된다.

실행결과

```
Mouse pressed (# of clicks: 1) X=93 Y=47    <-- 버튼을 클릭하였을 때 발생
Mouse dragged X=93 Y=48
Mouse dragged X=94 Y=48
...                                         <-- 버튼을 클릭한채로 움직이면 발생
Mouse dragged X=117 Y=66
Mouse dragged X=118 Y=66
Mouse released (# of clicks: 1) X=118 Y=66  <-- 버튼에서 손을 떼면 발생
```

마우스 이벤트 객체

마우스 버튼의 클릭 횟수나 더블 클릭을 감시하며 클릭된 위치와 클릭된 버튼을 알 수 있다.

- MouseEvent 클래스

메소드	설명
`int getClickCount()`	빠른 연속적인 클릭의 횟수를 반환한다. 예를 들어 2이면 더블 클릭을 의미한다.

`int getX()`
`int getY()`
`Point getPoint()`
	이벤트가 발생했을 당시의 (x, y) 위치를 반환한다. 위치는 컴포넌트에 상대적이다.

`int getXOnScreen()`
`int getYOnScreen()`
`int getLocationOnScreen()`
	절대 좌표 값 (x, y)을 반환한다. 이들 좌표값은 가상 화면에 상대적이다.

`int getButton()`	어떤 마우스 버튼의 상태가 변경되었는지를 반환한다. NOBUTTON, BUTTON1, BUTTON2, BUTTON3 중의 하나이다.

MouseEvent도 많은 메소드를 InputEvent 클래스에서 상속받는다.

MyMouseEventTest.java

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

class MyMouseEvent extends JFrame implements MouseListener, MouseMotionListener {
        public MyCarMouseEvent() {
                setTitle("Mouse Event");
```

```
                setSize(300, 200);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                JPanel panel = new JPanel();
                panel.addMouseListener(this);
                panel.addMouseMotionListener(this);
                add(panel);
                setVisible(true);
        }
        public void mousePressed(MouseEvent e) {
                display("Mouse pressed (# of clicks: " + e.getClickCount() + ")", e);
        }
        public void mouseReleased(MouseEvent e) {
                display("Mouse released (# of clicks: " + e.getClickCount() + ")", e);
        }
        public void mouseEntered(MouseEvent e) {
                display("Mouse entered", e);
        }
        public void mouseExited(MouseEvent e) {
                display("Mouse exited", e);
        }
        public void mouseClicked(MouseEvent e) {
                display("Mouse clicked (# of clicks: " + e.getClickCount() + ")", e);
        }
        protected void display(String s, MouseEvent e) {
                System.out.println(s + " X=" + e.getX() + " Y=" + e.getY());
        }
        // MouseMotion 리스너 메소드 구현
        public void mouseDragged(MouseEvent e) {
                display("Mouse dragged", e);
        }
        public void mouseMoved(MouseEvent e) {
                display("Mouse moved", e);
        }
}
public class MyMouseEventTest {
        public static void main(String[] args) {
                MyMouseEvent mme = new MyMouseEvent();
        }
}
```

마우스 이벤트 처리기를 붙인다.

[자동차 예제 2]

MyCarMouseEventTest.java

```
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;
class MyCarMouseEvent extends JPanel {
	BufferedImage img = null;
	int img_x = 0;
	int img_y = 0;
	public MyCarMouseEvent() {
		try {
			img = ImageIO.read(new File("car.gif"));
		} catch (IOException e) {
			System.out.println("no image");
			System.exit(1);
		}
		addMouseListener(new MouseListener() {
			public void mousePressed(MouseEvent e) {
				img_x = e.getX();
				img_y = e.getY();
				repaint();
			}
			public void mouseReleased(MouseEvent e) { }
			public void mouseEntered(MouseEvent e) { }
			public void mouseExited(MouseEvent e) { }
			public void mouseClicked(MouseEvent e) { }
		});
	}
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, img_x, img_y, null);
	}
}
public class MyCarMouseEventTest extends JFrame {
	public MyCarMouseEventTest() {
```

```
                setSize(300, 300);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                add(new MyCarMouseEvent());
                setVisible(true);
        }
        public static void main(String[] args) {
                MyCarMouseEventTest mmet = new MyCarMouseEventTest();
        }
}
```

6. 어댑터 클래스

이벤트를 처리하기 위해서는 리스너 인터페이스에서 정의되어 있는 모든 메소드를 구현해야 한다. 따라서 프로그래머가 작성하기를 원하는 메소드는 하나뿐인 경우에도 인터페이스의 모든 메소드를 구현해야 하는 불편함이 따른다. 이러한 불편을 해소하기 위한 것이 각 리스너(Listener)에 대응하는 **어댑터 클래스(Adaptor Class)**이다. 어댑터 클래스(Adaptor Class)를 사용하면 원하는 메소드 만을 구현하는 것이 가능해진다. 한가지 주의할 것은 **리스너는 인터페이스**이고 **어댑터는 클래스**의 형태로 제공된다는 것이다. 의미적 이벤트에는 어댑터 클래스가 없는데 이것은 의미적 이벤트의 리스너 인터페이스는 메소드를 하나만 가지고 있기 때문이다.

인터페이스	어댑터 클래스
ComponentListener	ComponentAdapter
ContainerListener	ContainerAdapter
FocusListener	FocusAdater
KeyListener	KeyAdapter
MouseListener	MouseAdapter
MouseMotionListener	MouseMotionAdapter
WindowListener	WindowAdapter

- 리스너를 사용하는 방법

```
public class KeyEventTest implements KeyListener {
    ...
    p.addKeyListener(this);
    ...
    public void keyTyped(KeyEvent e){        // 작성하기 원하는 메소드
       if( e.getKeyChar() == 'x' ){
          ...
       }
    }
    public void keyPressed(KeyEvent e){      // 불필요한 메소드
    }
    public void keyReleased(KeyEvent e){     // 불필요한 메소드
    }
}
```

- 어댑터를 사용하는 방법

```
public class KeyEventTest extends KeyAdaptor {

    public void keyTyped(KeyEvent e){
        if( e.getKeyChar() == 'x' ){
            ...
        }
    }
}
```

필요한 메소드만 재정의한다.

어댑터를 사용하는 방법의 문제점은 자바에서는 다중 상속을 허용하지 않기 때문에 두 개의 클래스를 동시에 상속받을 수 없다는 점이다. 즉 키보드에서 입력을 받는 애플릿의 경우에 KeyAdaptor와 Applet 클래스를 동시에 상속받아야 하지만 자바에서는 불가능하다. 이 경우의 해결 방안은 KeyAdaptor를 내부 클래스(inner class)로 정의하여 사용하는 것이다.

내부 클래스를 이용하여 어댑터 클래스 문제를 해결하면 다음과 같다.

```
public class MyFrame extends JFrame {
    ...
    addKeyListener(new MyKeyAdapter());          // 내부 클래스 객체를 생성
    ...
        class MyKeyAdaptor extends KeyAdapter {  // 내부 클래스 정의
        public void keyTyped(KeyEvent e) {
        // 원하는 코드를 입력한다.
        }
    }
}
```

응용프로그램이 마우스 이벤트와 마우스 움직임 이벤트를 동시에 요구하면 MouseInputAdapter 클래스를 확장하는 것이 좋다. 이 클래스는 MouseInputListener 인터페이스를 구현하는데 이 인터페이스는 MouseListener와 MouseMotionListener 인터페이스를 동시에 구현하고 있다.

[자동차 예제 3]

마우스로 자동차 이동을 MouseAdapter를 사용한다.

CarMouseAdapterTest.java

```
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

class MyCarPanel extends JPanel {
        BufferedImage img = null;
```

```
	int img_x = 0;
	int img_y = 0;

	public MyCarPanel() {
		try {
			img = ImageIO.read(new File("car.gif"));
		} catch (IOException e) {
			System.out.println("no image");
			System.exit(1);
		}
		addMouseListener(new MouseAdapter() {
			public void mousePressed(MouseEvent e) {
				img_x = e.getX();
				img_y = e.getY();
				repaint();
			}
		});
	}
	public void paintComponent(Graphics g) {
		super.paintComponent(g);
		g.drawImage(img, img_x, img_y, null);
	}
}

public class CarMouseAdapterTest extends JFrame {
	public CarMouseAdapterTest() {
		setSize(300, 300);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		add(new MyCarPanel());
		setVisible(true);
	}
	public static void main(String[] args) {
		CarMouseAdapterTest cmat = new CarMouseAdapterTest();
	}
}
```

7. 컴포넌트, 컨테이너, 윈도우, 포커스 이벤트

1) 컴포넌트 이벤트

컴포넌트 이벤트는 컴포넌트가 이동되거나 가려지거나 크기가 변경되었음을 경우에 발생한다. 컴포넌트 이벤트를 받기 위해서는 ComponentListener 인터페이스의 다음과 같은 메소드를 구현하여야 한다.

• ComponentListener 인터페이스

메소드	설명
componentHidden(ComponentEvent e)	setVisible() 메소드가 호출되어 컴포넌트가 가려진 경우에 발생
componentMoved(ComponentEvent e)	컴포넌트가 이동된 경우에 발생
componentResized(ComponentEvent e)	컴포넌트의 크기가 변화한 경우에 발생
componentShown(ComponentEvent e)	컴포넌트가 화면에 나타났을 때에 발생

• ComponentEvent 클래스

메소드	설명
Component getComponent()	이벤트를 발생한 컴포넌트를 반환한다.

2) 포함 이벤트

컨테이너에 컴포넌트를 추가되거나 제거될 때에 발생한다. ContainmentListener 인터페이스의 다음과 같은 메소드를 구현하여야 한다.

• ContainmentListener 인터페이스

메소드	설명
componentAdded(ContainerEvent e)	컴포넌트가 컨테이너에 추가되는 경우에 발생
componentRemoved(ContainerEvent e)	컴포넌트가 컨테이너에서 제거되는 경우에 발생

3) 포커스 이벤트

윈도우 시스템에서는 화면상 동시에 여러 개의 컴포넌트가 존재하고 이중에서 오직 한 개의 컴포넌트만이 마우스나 키보드 입력을 받을 수 있다. 이 경우 마우스나 키보드 입력을 받을 수 있는 컴포넌트가 포커스(focus)를 가지고 있다고 한다. 컴포넌트가 포커스를 획득하거나 상실하게 되면 FocusEvent가 발생한다. FocusEvent를 받기 위해서는 FocusListener를 구현하여야 한다.

• FocusListener 인터페이스

메소드	설명
focusGained(FocusEvent e)	컴포넌트가 포커스를 획득하는 경우에 발생
focusLost(FocusEvent e)	컴포넌트가 포커스를 상실하는 경우에 발생

4) 윈도우 이벤트

윈도우 이벤트는 사용자가 윈도우 조작을 하는 경우에 발생한다. 윈도우를 열거나 닫는 경우, 아이콘화하거나, 원상복구하는 경우, 활성화하거나, 비활성화하는 경우에 발생한다. 윈도우 이벤트를 받기 위해서는 WindowListener 인터페이스를 구현하여야 한다. 윈도우 포커스 이벤트와 윈도우 상태 이벤트가 있다. 윈도우 포커스 이벤트는 윈도우가 포커스를 얻거나 잃었을 때 발생한다. 윈도우 상태 이벤트는 윈도우

의 상태가 변경되면 발생한다. 이들 이벤트를 처리하려면 WindowFocusListener 인터페이스나 WindowStateListener 인터페이스를 구현하여야 한다.

- WindowListener 인터페이스

메소드	설명
windowOpened(WindowEvent e)	윈도우를 여는 경우에 호출된다.
windowClosing(WindowEvent e)	윈도우를 닫으라는 요청을 받는 경우에 호출된다.
windowClosed(WindowEvent e)	윈도우를 닫은 후에 호출된다.
windowIconified(WindowEvent e)	윈도우가 아이콘화되는 경우에 호출된다.
windowDeiconified(WindowEvent e)	윈도우가 복귀되는 경우에 호출된다.
windowActivated(WindowEvent e)	윈도우가 활성화되는 경우에 호출된다.
windowDeactivated(WindowEvent e)	윈도우가 비활성화되는 경우에 호출된다.

- WindowFocusListener 인터페이스

메소드	설명
windowGainedFocus(WindowEvent) windowLostFocus(WindowEvent)	윈도우가 포커스를 얻거나 잃는 경우에 발생한다.

- WindowStateListener 인터페이스

메소드	설명
windowStateChanged(WindowEvent)	윈도우의 상태가 변경되면 호출된다. 윈도우의 상태는 아이콘화되거나 최대화, 최소화되는 경우에 변경된다.

(예제)

컴포넌트와 포커스 이벤트를 테스트하기 위하여 프레임만 생성하고 프레임에 컴포넌트와 포커스 리스너를 구현하여서 프레임을 이동하거나 최소화시켜보면 다음과 같은 출력이 콘솔에 나타난다.

MyComponentFocusTest.java

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

class MyComponentFocus extends JFrame implements ComponentListener, FocusListener {
        JTextArea display;

        public MyComponentFocus() {
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setSize(300, 200);
                setTitle("Window Event Test");
                addComponentListener(this);   // 프레임에 컴포넌트 리스너를 등록
                addFocusListener(this);       // 프레임에 컴포넌트 리스너를 등록
```

```
                setVisible(true);
        }
        public void componentHidden(ComponentEvent e) {   // 컴포넌트가 감춰질 때 호출
                display("componentHidden() 메소드 호출 ");
        }
        public void componentMoved(ComponentEvent e) {   // 컴포넌트가 이동될 때 호출
                display("componentMoved() 메소드 호출 ");
        }
        public void componentResized(ComponentEvent e) {   // 컴포넌트의 크기가 변경되면 호출
                display("componentResized() 메소드 호출 ");
        }
        public void componentShown(ComponentEvent e) {    // 호출화면에 표시되면
                display("componentShown() 메소드 호출 ");
        }
        public void focusGained(FocusEvent e) {                 // 포커스를얻으면 호출
                display("focusGained() 메소드 호출 ");
        }
        public void focusLost(FocusEvent e) {                    // 포커스를 잃으면 호출
                display("focusLost() 메소드 호출 ");
        }
        private void display(String s) {
                System.out.println(s);
        }
}

public class MyComponentFocusTest {
        public static void main(String[] args) {
                MyComponentFocus mcf = new MyComponentFocus();
        }
}
```

20. 배치관리자 초등학생 성적관리 문제에 이벤트를 적용한다.