Chapter 10 GUI 이벤트 처리

10.1 이벤트 처리 개요

이벤트란?

- Event-driven programming(이벤트 구동 프로그래밍): 어떤 이벤트가 일어나는지를 감시하고 있다가 이벤트가 발생하면 적절한 처리를 해주는 프로그래밍 방식
- Event: 사용자가 버튼을 클릭하거나 마우스를 움직이거나 키를 누르면 발생한다.

Term	Description	Example
이벤트 객체	발생한 이벤트에 대한 정보를 가지고 있어 이벤트가 발생하면 이벤트 객체가 생성되는 객체	마우스 이벤트 객체라면 현재 위치, 버튼 눌림 상태 등을 가지고 있다.
이벤트 소스	이벤트를 생성하는 컴포넌트	버튼이 이벤트를 발생하였다면 그 버튼은 이벤트 소스다.
이벤트 리스너	발생된 이벤트 객체에 반응하여서 이벤트를 처리하는 객체. 이벤트를 처리하기 위하여 이벤트에 귀를 기울이고 있는 객체.	버튼에서 발생하는 이벤트를 처리하려면 이벤트 리스너를 버튼에 등록해야 한다.

이벤트 처리 과정

```
public class MyFrame extends JFrame {
    ...
    public MyFrame() {
        button = new JButton("동작");
        button.addActionListener(new MyListener()); // 2. 각 컴포넌트가 제공하는 이벤트를 등록하는 메소드를 이용해 이벤트 리스너를 컴포넌트에 등록한다.
        ...
    }

    class MyListener implements ActionListener {  // 1. 리스너 인터페이스(종류에 따라 이벤트를 처리하기 위한 규격)를 구현하여 클래스를 이벤트 리스너로 만든다.
        public void actionPerformed(ActionEvent e) {
            ... // Action 이벤트를 처리하는 코드
        }
    }
}
```

이벤트 객체

- 이벤트 객체는 발생된 이벤트에 대한 모든 정보를 리스너로 전달한다.
- 이벤트 객체는 getSource() 메소드를 가진다.

- getSource() 메소드: 이벤트가 발생한 이벤트 소스를 Object 타입으로 반환한다. 이것을 필요한 타입으로 형변환해서 사용해야 한다.

10.2 이벤트 처리 방법

내부 클래스가 이벤트를 처리하는 방법

1. 이벤트를 처리하는 클래스들이 모여 있는 패키지를 java.awt.event를 포함한다.
2. 리스너 인터페이스를 구현하는 클래스를 내부 클래스로 하여 작성한다.
    - 내부 클래스: 다른 클래스 안에 위치하는 클래스로서 외부 클래스의 필드들을 자유롭게 사용할 수 있다.
3. 이벤트 리스너 객체를 생성하고 addActionListener()를 이용하여 버튼에 리스너 객체를 등록한다.

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.BorderLayout;
import java.awt.event.*;

public class EventTest1 extends JFrame {
    private JButton button;
    private JLabel label;
    private int counter = 0;

    class MyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            counter++;
            label.setText("현재의 카운터 값: " + counter);
        }
    }

    public EventTest1() {
        setLocation(500, 500);
        setSize(400, 150);
        setTitle("내부 클래스가 이벤트를 처리하는 방법");
        setLayout(new BorderLayout());

        button = new JButton("증가");
        label = new JLabel("현재의 카운터 값: " + counter);

        button.addActionListener(new MyListener());

        add(label, "Center");
        add(button, "East");
        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String [] args) {
        EventTest1 t = new EventTest1();
    }
}
```

외부 클래스가 이벤트를 처리하는 방법

- 이벤트를 처리하는 외부 클래스에서 다른 클래스의 내부 객체에 접근할 수 없기 때문에 이벤트에 대응하기 어렵다.

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

class MyListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource();
        // counter++;
        // label.setText("현재의 카운터 값: " + counter);
    }
}

public class EventTest2 extends JFrame {
    private JButton button;
    protected JLabel label;
    int counter = 0;

    public EventTest2() {
        setLocation(300, 300);
        setSize(400, 150);
        setTitle("외부 클래스가 이벤트를 처리하는 방법");
        setLayout(new FlowLayout());

        button = new JButton("증가");
        label = new JLabel("현재의 카운터 값: " + counter);

        button.addActionListener(new MyListener());

        add(button);
        add(label);

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String [] args) {
        EventTest2 et = new EventTest2();
    }
}
```

프레임 클래스가 이벤트를 처리하는 방법

- 프레임 클래스가 JFrame을 상속받으면서 동시에 ActionListener 인터페이스도 구현하는 방법

- 비교적 많이 사용된다.

```
import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JLabel;
import java.awt.FlowLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class EventTest3 extends JFrame implements ActionListener {
    private JButton button;
    private JLabel label;
    private int counter;

    public void actionPerformed(ActionEvent e) {
        counter++;
        label.setText("현재 카운터 값은: " + counter);
    }

    public EventTest3() {
        setLocation(400, 150);
        setSize(400, 150);
        setTitle("프레임 클래스가 이벤트 처리를 하는 방법");
        setLayout(new FlowLayout());

        button = new JButton("증가");
        label = new JLabel("현재 카운터 값은 : " + counter);

        button.addActionListener(this);

        add(button);
        add(label);

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String [] args ) {
        EventTest3 et = new EventTest3();
    }
}
```

익명 클래스를 사용하는 방법

- 익명 클래스: 이름이 없는 클래스를 작성하여 바로 한 번만 사용한다.
- 많이 사용된다. 특히 안드로이드 프로그램에서 주로 사용된다.

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
```

```
import javax.swing.JLabel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class EventTest4 extends JFrame {
    private JPanel panel;
    private JButton button;
    private JLabel label;
    private int counter = 0;

    public EventTest4() {
        setLocation(1000, 1000);
        setSize(500, 500);
        setTitle("익명 클래스를 이용하는 방법");

        panel = new JPanel();
        button = new JButton("증가");
        label = new JLabel("현재 카운터 값은:   " + counter);

        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                counter++;
                label.setText("증가된 카운터 값은: " + counter);
            }
        });

        panel.add(button);
        panel.add(label);
        add(panel);

        setVisible(true);
    }

    public static void main(String [] args) {
        EventTest4 et = new EventTest4();
    }
}
```

람다식을 이용하는 방법

- Lambda expression: 이름이 없는 메소드
- Lambda expression은 한 번만 사용되고 메소드의 길이가 짧은 경우에 유용하다.

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JLabel;

public class EventTest5 extends JFrame {
    private JPanel panel;
    private JButton button;
```

```
    private JLabel label;
    private int counter;

    public EventTest5() {
        setLocation(400, 400);
        setSize(400, 500);
        setTitle("The way using lambda expression");

        panel = new JPanel();
        button = new JButton("증가");
        label = new JLabel("현재 카운터 값은: " + counter);

        button.addActionListener(e -> {
            counter++;
            label.setText("현재 카운터 값은: " + counter);
        });

        panel.add(button);
        panel.add(label);
        add(panel);

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String [] args) {
        EventTest5 t = new EventTest5();
    }
}
```

10.3 스윙 컴포넌트의 이벤트

- 거의 모든 스윙 컴포넌트들이 다양한 이벤트를 발생한다.
- JComboBox Class는 ActionEvent, ItemEvent, PopupManuEvent를 발생한다. 개발자는 자신이 처리하고 싶은 이벤트를 선택해 리스너 클래스를 작성하여 주면 된다.
- 스윙 컴포넌트가 발생하는 이벤트는 모든 컴포넌트가 공통적으로 지원하는 저수준 이벤트와 일부 컴포넌트만 지원하는 의미적 이벤트로 나눌 수 있다.

저수준 이벤트

- 저수준 이벤트는 모든 컴포넌트에서 발생한다.
- 마우스나 키보드로부터 발생된다.

low-level Event	Description
Component	컴포넌트의 크기나 위치가 변경되었을 경우 발생
Focus	키보드 입력을 받을 수 있는 상태가 되었을 때 혹은 그 반대의 경우에 발생
Container	컴포넌트가 컨테이너에 추가되거나 삭제될 때 발생
Key	사용자가 키를 눌럭을 때 또는 키보드 포커스를 가지고 있는 객체에서 발생

low-level Event	Description
Mouse	마우스 버튼이 클릭되었을 때 또는 마우스가 객체의 영역으로 돌어오거나 나갈 때 발생
MouseMotion	마우스가 움직였을 때 발생
MouseWheel	컴포넌트 위에서 마우스 휠을 움직이는 경우 발생
Window	윈도우에 어떤 변화(열림, 닫힘, 아이콘화 등)가 있을 때 발생

의미적 이벤트

- 일부 컴포넌트에서 발생한다.
- 가능하다는 저수준 이벤트보다는 의미적 이벤트를 처리하는 것이 코드를 강건하게 하고 이식성이 좋게 한다.
- 의미적 이벤트의 경우 각 컴포넌트에 따라 발생할 수 있는 이벤트의 종류가 달라진다.

Semantic Event	Description
Action	사용자가 어떤 동작을 하는 경우에 발생
Caret	텍스트 삽입점이 동하거나 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

액션 이벤트

- 버튼에서 발생하는 이벤트를 처리하려면 클래스를 정의할 때 ActionListener 인터페이스를 구현하여야 한다.
- ActionListener에는 actionPerformed()라는 추상 메소드만이 정의되어 있다.
- actionPerformed() 메소드는 액션 이벤트가 발생할 때마다 호출된다.
- 사용자가 버튼을 클릭하는 경우
- 사용자가 메뉴 항목을 선택하는 경우
- 사용자가 텍스트 필드에서 엔터키를 누르는 경우

예제 10-1

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ChangeBackground extends JFrame {
    private JPanel panel;
    private JButton buttonYellow;
    private JButton buttonPink;
```

```
    public ChangeBackground() {
        setLocation(500, 500);
        setSize(500, 200);
        setTitle("버튼으로 배경색 변경하기");

        panel = new JPanel();

        buttonYellow = new JButton("노랑색");
        buttonYellow.addActionListener(new MyListener());
        panel.add(buttonYellow);

        buttonPink = new JButton("분홍색");
        buttonPink.addActionListener(new MyListener());
        panel.add(buttonPink);

        add(panel);

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    private class MyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == buttonYellow) {
                panel.setBackground(Color.yellow);
            } else if (e.getSource() == buttonPink) {
                panel.setBackground(Color.pink);
            }
        }
    }

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

키패드 만들기

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.BorderLayout;
import java.awt.GridLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class KeyPad extends JFrame implements ActionListener {
    private JPanel panel;
    private JTextField txt;
```

```
    public KeyPad() {
        setLocation(600, 400);
        setSize(500, 500);
        setTitle("Key Pad");

        txt = new JTextField();
        add(txt, BorderLayout.NORTH);

        panel = new JPanel();
        panel.setLayout(new GridLayout(3, 3));
        add(panel, BorderLayout.CENTER);

        for ( int i = 1; i <= 9; i++) {
            JButton btn = new JButton("" + i);
            btn.addActionListener(this);
            btn.setPreferredSize(new Dimension(100, 30));
            panel.add(btn);
        }
        pack();

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public void actionPerformed(ActionEvent e) {
        String actionCommand = e.getActionCommand();
        txt.setText(txt.getText() + actionCommand);
    }

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

가위 바위 보 게임

```
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JButton;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.util.Random;

import java.awt.GridLayout;
import java.awt.BorderLayout;

public class RockPapperScissor extends JFrame implements ActionListener {
    final int SCISSOR = 0;
    final int ROCK = 1;
```

```
    final int PAPPER = 2;

    private JPanel panel;
    private JLabel information, output;
    private JButton rock, papper, scissor;

    public RockPapperScissor() {
        setLocation(600, 400);
        setSize(500, 500);
        setTitle("가위 바위 보");

        panel = new JPanel();
        panel.setLayout(new GridLayout(0, 3));

        information = new JLabel("아래의 버튼 중에서 하나를 클릭하시오!");
        output = new JLabel("Good Lurk");

        rock = new JButton("ROCK");
        papper = new JButton("PAPPER");
        scissor = new JButton("SCISSOR");

        rock.addActionListener(this);
        papper.addActionListener(this);
        scissor.addActionListener(this);

        panel.add(rock);
        panel.add(papper);
        panel.add(scissor);

        add(information, BorderLayout.NORTH);
        add(panel, BorderLayout.CENTER);
        add(output, BorderLayout.SOUTH);

        setVisible(true);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    }

    public static void main(String [] args) {
        RockPapperScissor f = new RockPapperScissor();
    }

    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton)e.getSource();
        int user = 0;

        if (b.getText().equals("ROCK"))
            user = ROCK;
        else if (b.getText().equals("PAPPER"))
            user = PAPPER;
        else if (b.getText().equals("SCISSOR"))
            user = SCISSOR;

        Random random = new Random();
        int computer = random.nextInt(3);
```

```
            if (user == computer)
                output.setText("무승부");
            else if (user == (computer + 1) % 3)
                output.setText("승리");
            else
                output.setText("패배");
        }
    }
```

10.4 키 이벤트

- key event(키 이벤트) : 사용자가 키보드를 이용하여 키를 누르거나 손을 떼는 경우에도 발생한다.
- keyPressed 아벤트: 사용자가 키를 누르면 이벤트가 발생한다.
- keyReleased 이벤트: 사용자가 키에서 손을 떼면 이벤트가 발생한다.
- keyTyped 이벤트 : 입력된 유니코드 문자가 전송된다.
- 사용자가 하나의 키를 누르면 3개의 키 이벤트가 순차적으로 발생한다.
- keyPressed() -> ketTyped() -> keyReleased() 순으로 호출된다.

포커스

- 컴포넌트가 키 이벤트를 받으려면 반드시 포커스를 가지고 있어야 한다.
- 포커스: 키 입력을 받을 권리

KeyListener 인터페이스

KeyEvent 클래스

getKeyChar()

getKeyCode()

isControlDown()

10.5 Mouse와 Mousemotion 이벤트

10.6 이벤트 클래스