이벤트 처리

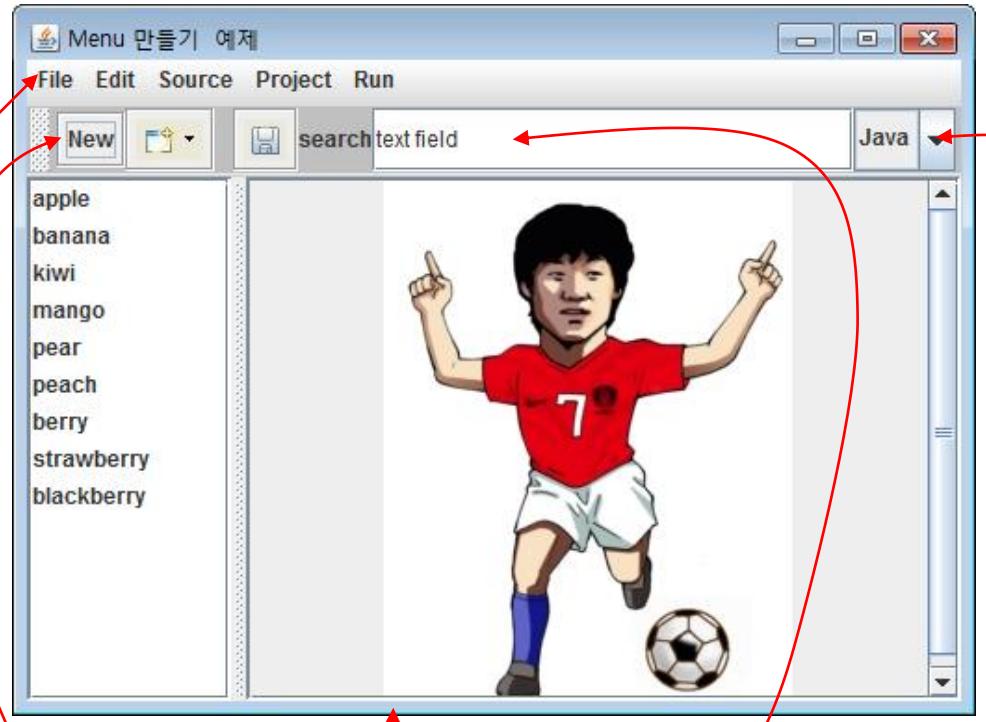
한성대학교 컴퓨터공학부

신 성

이벤트 기반 프로그래밍

- **이벤트 기반 프로그래밍(Event Driven Programming)**
  - 이벤트의 발생에 의해 프로그램 흐름이 결정되는 방식
    - 이벤트가 발생하면 이벤트를 처리하는 루틴(이벤트 리스너) 실행
    - 실행될 코드는 이벤트의 발생에 의해 전적으로 결정
  - 이벤트 종류
    - 사용자의 입력 : 마우스 드래그, 마우스 클릭, 키보드 누름 등
    - 센서로부터의 입력, 네트워크로부터 데이터 송수신
    - 다른 응용프로그램이나 다른 스레드로부터의 메시지
- **이벤트 기반 응용 프로그램의 구조**
  - 각 이벤트마다 처리하는 리스너 코드 보유
- **GUI 응용프로그램은 이벤트 기반 프로그래밍으로 작성됨**
  - GUI 라이브러리 종류
    - C++의 MFC, C# GUI, Visual Basic, X Window, Android 등
    - 자바의 AWT와 Swing

스윙 응용프로그램의 이벤트의 실제 예



마우스로 메뉴 선택
(Mouse Event, Action Event)

마우스로 버튼 클릭
(Mouse Event, Action Event)

마우스로 윈도우 크기 조절
(Mouse Event, Container Event)

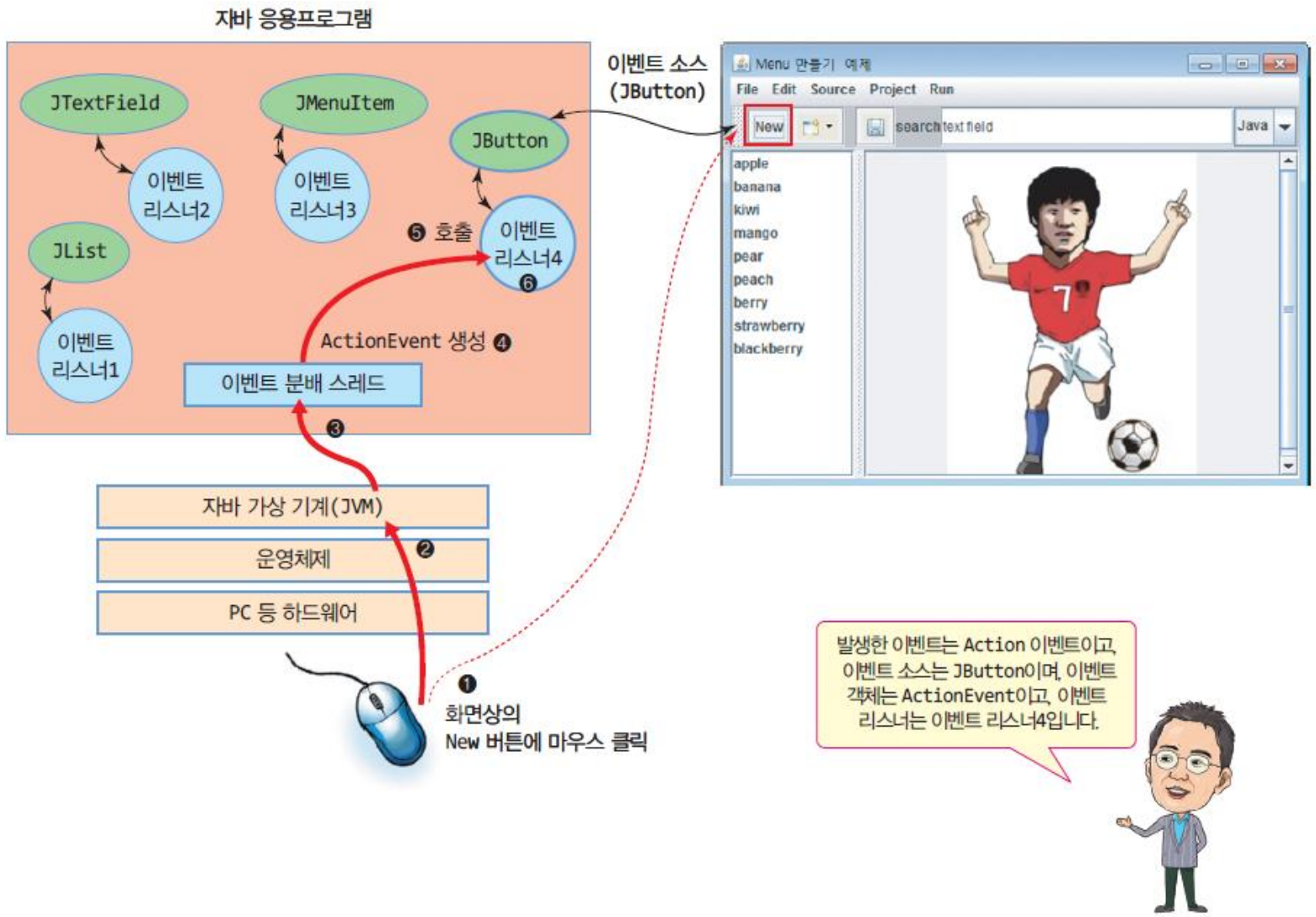
키보드 입력
(Key Event)

키보드로 메뉴 선택
(Key Event, Action Event)

자바 스윙 프로그램에서 이벤트 처리 과정

- **이벤트가 처리되는 과정**
  - 이벤트 발생
    - 예 :마우스의 움직임 혹은 키보드입력
  - 이벤트 객체 생성
    - 현재 발생한 이벤트에 대한 정보를 가진 객체
  - 응용프로그램에 작성된 이벤트 리스너 찾기
  - 이벤트 리스너 실행
    - 리스너에 이벤트 객체 전달
    - 리스너 코드 실행

자바의 이벤트 기반 스윙 응용프로그램의 구조와 이벤트 처리 과정



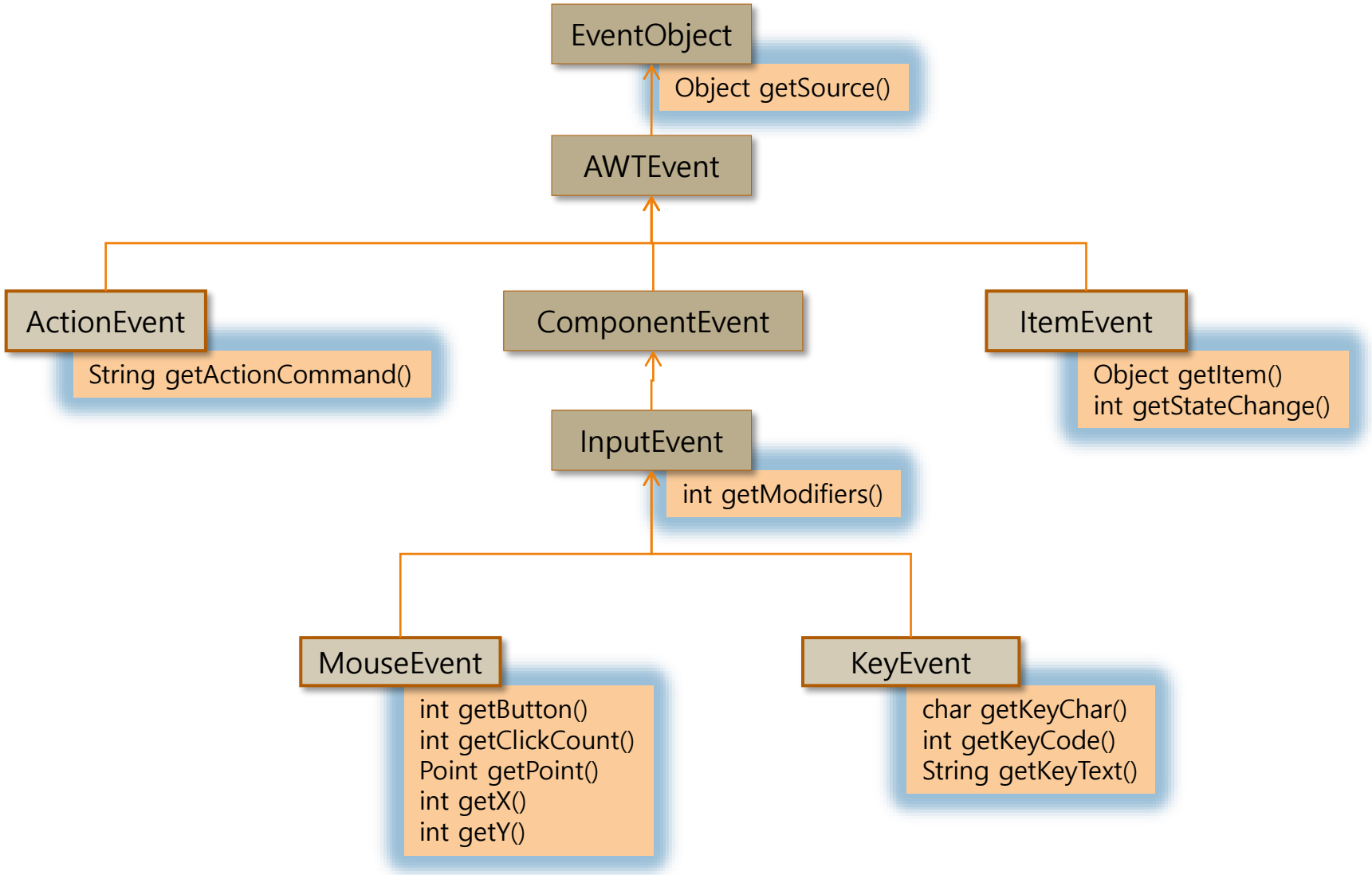
이벤트 객체

- **이벤트 객체**
  - 발생한 이벤트에 관한 정보를 가진 객체
  - 이벤트 리스너에 전달됨
    - 이벤트 리스너 코드가 발생한 이벤트에 대한 상황을 파악할 수 있게 함

- **이벤트 객체가 포함하는 정보**
  - 이벤트 종류와 이벤트 소스
  - 이벤트가 발생한 화면 좌표 및 컴포넌트 내 좌표
  - 이벤트가 발생한 버튼이나 메뉴 아이템의 문자열
  - 클릭된 마우스 버튼 번호 및 마우스의 클릭 횟수
  - 키의 코드 값과 문자 값
  - 체크박스, 라디오버튼 등과 같은 컴포넌트에 이벤트가 발생하였다면 체크 상태

- **이벤트 소스를 알아 내는 메소드 : 현재 발생한 이벤트의 소스 컴포넌트의 레퍼런스를 반환**
  - Object getSource()
    - 발생한 이벤트의 소스 컴포넌트 리턴
    - Object 타입으로 리턴하므로 캐스팅하여 사용
    - 모든 이벤트 객체는 EventObject의 getSource() 상속 받음

이벤트 객체와 이벤트 정보를 리턴하는 메소드



이벤트 객체, 이벤트 소스, 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	라디오 버튼의 선택 상태가 변할 때
	JCheckBoxMenuItem	체크박스 메뉴 아이템의 선택 혹은 해제
ListSelectionEvent	JList	리스트에 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
TextEvent	TextField	텍스트 변경
	TextArea	텍스트 변경
WindowEvent	Window	Window를 상속받는 모든 컴포넌트에 대해 윈도우 활성화, 비활성화, 아이콘화, 아이콘에서 복구, 윈도우 열기, 윈도우 닫기, 윈도우 종료

이벤트 소스 : Component

=> Component를 상속받은 모든 스윙 컴포넌트에 대해 이벤트 발생 가능

리스너 인터페이스

- **이벤트 리스너**
  - 이벤트를 처리하는 자바 프로그램 코드, 클래스로 작성
- **자바는 다양한 리스너 인터페이스 제공**
  - 예) ActionListener 인터페이스 – 버튼 클릭 이벤트를 처리하기 위한 인터페이스

```
interface ActionListener { // 아래 메소드를 개발자가 구현해야 함
  public void actionPerformed(ActionEvent e); // Action 이벤트 발생시 호출됨
}
```

  - 예) MouseListener 인터페이스 – 마우스 조작에 따른 이벤트를 처리하기 위한 인터페이스

```
interface MouseListener { // 아래의 5개 메소드를 개발자가 구현해야 함
  public void mousePressed(MouseEvent e); // 마우스 버튼이 눌러지는 순간 호출
  public void mouseReleased(MouseEvent e); // 눌러진 마우스 버튼이 떼어지는 순간 호출
  public void mouseClicked(MouseEvent e); // 마우스가 클릭되는 순간 호출
  public void mouseEntered(MouseEvent e); // 마우스가 컴포넌트 위에 올라가는 순간 호출
  public void mouseExited(MouseEvent e); // 마우스가 컴포넌트 위에서 내려오는 순간 호출
}
```

- **사용자의 이벤트 리스너 작성**
  - 자바의 리스너 인터페이스 (interface)를 상속받아 구현
  - 리스너 인터페이스의 모든 추상 메소드 구현

자바에서 제공하는 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
Text	TextListener	void textValueChanged(TextEvent)	텍스트가 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 아이콘화될 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
ListSelection	ListSelectionListener	void valueChanged(ListSelectionEvent)	Jlist에 선택된 아이템이 변경될 때

이벤트 리스너 작성 과정 사례

1. 이벤트와 이벤트 리스너 선택

- 버튼 클릭을 처리하고자 하는 경우
  - 이벤트 : Action 이벤트, 이벤트 리스너 : ActionListener

2. 이벤트 리스너 클래스 작성 : ActionListener 인터페이스 구현

```
class MyActionListener implements ActionListener {
  public void actionPerformed(ActionEvent e) { // 버튼이 클릭될 때 호출되는 메소드
    JButton b = (JButton)e.getSource();          // 사용자가 클릭한 버튼 알아내기
    if(b.getText().equals("Action"))             // 버튼의 문자열이 "Action"인지 비교
      b.setText("액션");                         // JButton의 setText() 호출. 문자열변경
    else
      b.setText("Action");                       // JButton의 setText() 호출. 문자열변경
  }
}
```

3. 이벤트 리스너 등록

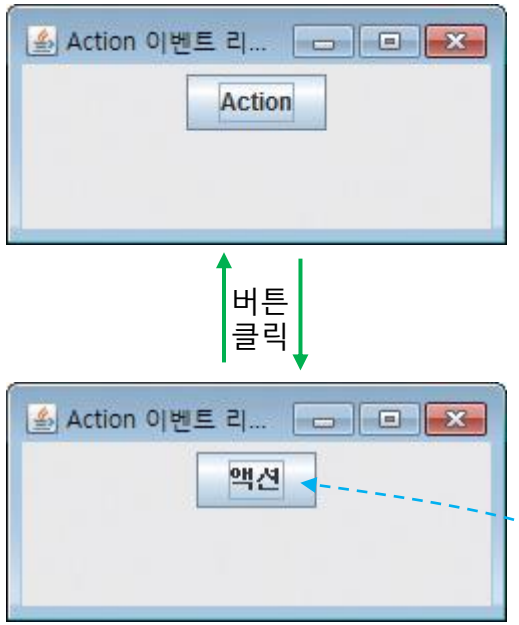
- 이벤트를 받아 처리하고자 하는 컴포넌트에 이벤트 리스너 등록
- component.addXXXListener(listener)
  - xxx : 이벤트 명, listener : 이벤트 리스너 객체

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

이벤트 리스너 작성 방법

- 3 가지 방법
  - 독립 클래스로 작성
    - 이벤트 리스너를 완전한 클래스로 작성
    - 이벤트 리스너를 여러 곳에서 사용할 때 적합
  - 내부 클래스(inner class)로 작성
    - 클래스 안에 멤버처럼 클래스 작성
    - 이벤트 리스너를 특정 클래스에서만 사용할 때 적합
  - 익명 클래스(anonymous class)로 작성
    - 클래스의 이름 없이 간단히 리스너 작성
    - 클래스 조차 만들 필요 없이 리스너 코드가 간단한 경우에 적합

독립 클래스로 Action 이벤트 리스너 만들기



- 독립된 클래스로 Action 이벤트 리스너 작성
- 이 클래스를 별도의 MyActionListener.java 파일로 작성하여도 됨

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

public class IndepClassListener extends JFrame {
    IndepClassListener() {
        setTitle("Action 이벤트 리스너 예제");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        JButton btn = new JButton("Action");
        btn.addActionListener(new MyActionListener());
        c.add(btn);
        setSize(250, 120);
        setVisible(true);
    }
    public static void main(String [] args) {
        new IndepClassListener();
    }
}
// 독립된 클래스로 이벤트 리스너를 작성한다.
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        JButton b = (JButton)e.getSource();
        if(b.getText().equals("Action"))
            b.setText("액션");
        else
            b.setText("Action");
    }
}
```

Action 이벤트 리스너 등록

Action 이벤트 리스너 구현

```
// 독립된 클래스로 이벤트 리스너를 작성한다.
class MyActionListener implements ActionListener {

    public void actionPerformed(ActionEvent e) {

        JButton b = (JButton) e.getSource();
        if(b.getText().equals("Action"))
            b.setText("액션");
        else
            b.setText("Action");

    }

}
```

object를 다운 캐스팅, 강제 형변환

[앞의 슬라이드] 이벤트 객체

- **이벤트 객체**
  - 발생한 이벤트에 관한 정보를 가진 객체
  - 이벤트 리스너에 전달됨
    - 이벤트 리스너 코드가 발생한 이벤트에 대한 상황을 파악할 수 있게 함

- **이벤트 객체가 포함하는 정보**
  - 이벤트 종류와 이벤트 소스
  - 이벤트가 발생한 화면 좌표 및 컴포넌트 내 좌표
  - 이벤트가 발생한 버튼이나 메뉴 아이템의 문자열
  - 클릭된 마우스 버튼 번호 및 마우스의 클릭 횟수
  - 키의 코드 값과 문자 값
  - 체크박스, 라디오버튼 등과 같은 컴포넌트에 이벤트가 발생하였다면 체크 상태

- **이벤트 소스를 알아 내는 메소드 : 현재 발생한 이벤트의 소스 컴포넌트의 레퍼런스를 반환**
  - Object getSource()
    - 발생한 이벤트의 소스 컴포넌트 리턴
    - Object 타입으로 리턴하므로 캐스팅하여 사용
    - 모든 이벤트 객체는 EventObject의 getSource() 상속 받음

Cont.

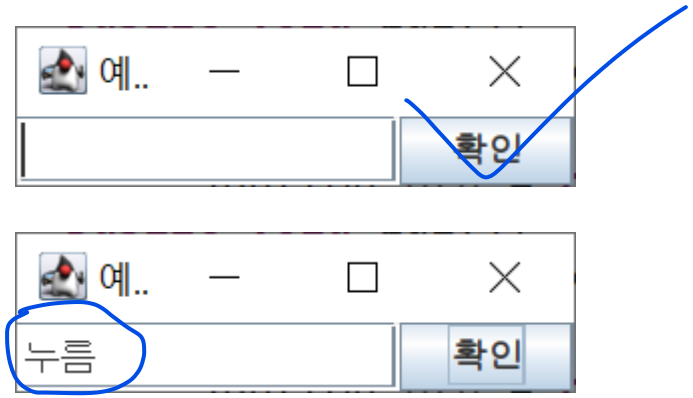
이벤트 처리를 위한 프로그래밍

- 1단계: 이벤트 소스 결정
  - 이벤트(E_D)를 발생시킬 컴포넌트(C_D) 결정
- 2단계: 이벤트 리스너 구현
  - 이벤트(E_D)를 처리할 객체의 클래스를 이벤트 리스너 인터페이스로 구현
    - 인터페이스의 추상 메서드를 오버라이딩
      - 이벤트 발생시 수행할 동작을 기술
- 3단계: 이벤트 소스와 이벤트 리스너 연결
  - `addXXXListener()` 메서드를 통해 C_D에 이벤트 리스너를 연결
    - 이벤트 발생시 이를 감지해 처리할 수 있도록

올려드린 ex1.txt

```
public class Test {
	JFrame mf;
	static JTextField text;

	public Test() {
		mf = new JFrame();
		mf.setTitle("예제");
		mf.setSize(200, 60);
		mf.setLocation(500, 300);
		mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		gui();

		mf.setVisible(true);
	}

	public void gui(){
		text = new JTextField();

		JButton btn = new JButton("확인");

		mf.add(text, BorderLayout.CENTER);
		mf.add(btn, BorderLayout.EAST);

		btn.addActionListener(new MyActionListener());
	}

	public static void main(String[] args) {
	    Test test = new Test();
	    test.gui();
	}
}
```

다른 클래스 인스턴스에 접근하기 위한 방법은?
(다음 페이지부터 몇 가지 소개)

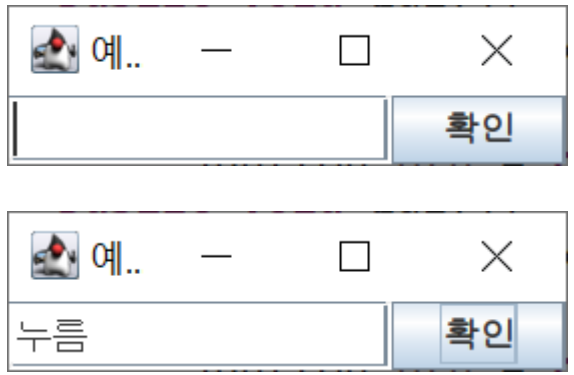



```
class MyActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
            Test.text.setText("누름");
    }
}
```

올려드린 ex2.txt

```
public class Test {
	JFrame mf;
	JTextField text;

	public Test() {
		mf = new JFrame();
		mf.setTitle("예제");
		mf.setSize(200, 60);
		mf.setLocation(500, 300);
		mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		gui();

		mf.setVisible(true);
	}

	public void gui(){
		text = new JTextField();

		JButton btn = new JButton("확인");

		mf.add(text, BorderLayout.CENTER);
		mf.add(btn, BorderLayout.EAST);

		btn.addActionListener(new MyActionListener(text));
	}

	public static void main(String[] args) {
	    Test test = new Test();
	    test.gui();
	}
}
```

생성자 함수 이용해 text 레퍼런스 전달



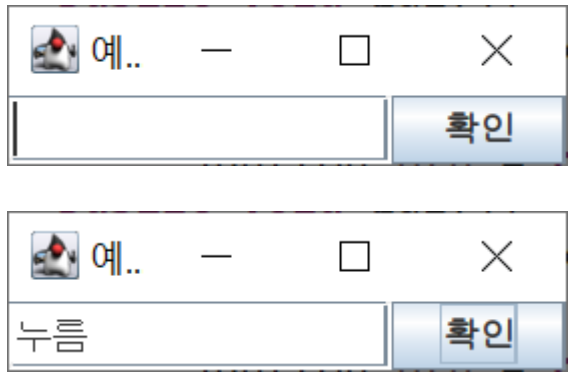

```
class MyActionListener implements ActionListener {

    JTextField t;

    MyActionListener(JTextField t){
        this.t=t;
    }

    public void actionPerformed(ActionEvent e) {
            t.setText("누름");
    }
}
```

올려드린 ex3.txt

```
public class Test {
	JFrame mf;
	JTextField text;

	public Test() {
		mf = new JFrame();
		mf.setTitle("예제");
		mf.setSize(200, 60);
		mf.setLocation(500, 300);
		mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		gui();

		mf.setVisible(true);
	}

	public void gui(){
		text = new JTextField();

		JButton btn = new JButton("확인");

		mf.add(text, BorderLayout.CENTER);
		mf.add(btn, BorderLayout.EAST);

		btn.addActionListener(new MyActionListener());
	}

	public static void main(String[] args) {
	    Test test = new Test();
	    test.gui();
	}

	class MyActionListener implements ActionListener {
		public void actionPerformed(ActionEvent e) {
			text.setText("누름");
		}
	 }
}
```

3 2 1

내부 클래스 이용



내부 클래스

한 클래스 정의 안에 정의된 또다른 클래스

- 내부 클래스의 메서드에서 외부 클래스의 멤버에 직접 접근이 가능
- 선언된 위치에 따라 변수와 동일한 유효 범위와 접근성을 가짐
- 코드의 복잡성을 줄일 수 있음

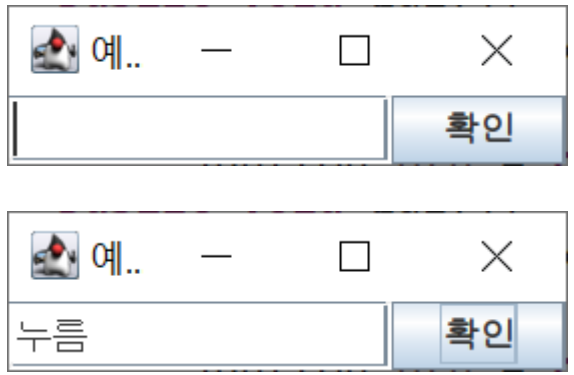
장점
- 외부 클래스 멤버를 직접 접근 가능
- 이벤트 리스너를 특정 클래스에서만 사용할 때 적합

단점
- 리스너 구현만 필요한 경우에는 여전히 별도의 클래스 선언 필요

올려드린 ex4.txt

```
public class Test {
	JFrame mf;
	JTextField text;

	public Test() {
		mf = new JFrame();
		mf.setTitle("예제");
		mf.setSize(200, 60);
		mf.setLocation(500, 300);
		mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		gui();

		mf.setVisible(true);
	}

	public void gui(){
		text = new JTextField();

		JButton btn = new JButton("확인");

		mf.add(text, BorderLayout.CENTER);
		mf.add(btn, BorderLayout.EAST);

		btn.addActionListener(new ActionListener(){
		public void actionPerformed(ActionEvent e) {
			text.setText("누름");
		}
	 });
	}

	public static void main(String[] args) {
	    Test test = new Test();
	    test.gui();
	}

}
```

익명 내부 클래스 임시 객체 이용



임시 객체

리스너 객체에 대한 참조변수 없이 임시 객체를 생성하여 리스너 등록시 인수로 사용

- 내부 클래스 객체를 리스너 지정 메서드에 바로 사용

- 장점
  - 리스너 객체가 단 한 번만 사용되는 경우에 유용
    - 각 이벤트 소스마다 별도의 리스너 지정시 적합

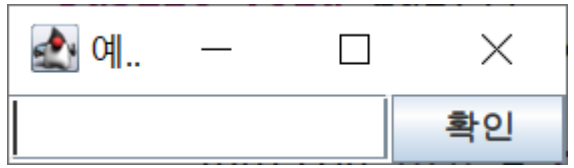
이름있는 객체 사용	임시 객체 사용

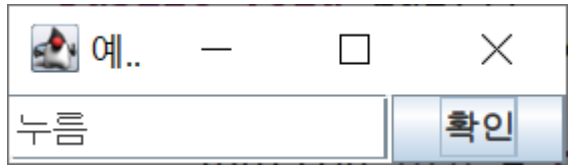
`Interface obj = new Interface() {`
`  // 메서드 구현`
`};`
`Method(obj);`
	`Method( new Interface() {`
	`  // 메서드 구현`
	`} );`

```java
public class Test implements ActionListener {
	JFrame mf;
	JTextField text;

	public Test() {
		mf = new JFrame();
		mf.setTitle("예제");
		mf.setSize(200, 60);
		mf.setLocation(500, 300);
		mf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

		gui();

		mf.setVisible(true);
	}

	public void gui(){
		text = new JTextField();

		JButton btn = new JButton("확인");

		mf.add(text, BorderLayout.CENTER);
		mf.add(btn, BorderLayout.EAST);

		btn.addActionListener(this);
	}

	public static void main(String[] args) {
	    Test test = new Test();
	    test.gui();
	}

	public void actionPerformed(ActionEvent e) {
	    text.setText("누름");
	}

}
```

올려드린 ex5.txt

현재 클래스에 리스너 인터페이스를 구현





현재 응용 프로그램 클래스가 리스너 인터페이스를 구현

- 리스너 연결시 리스너로부터 구현된 this 전달

- 장점
  - 별도의 클래스 작성 및 별도 객체 생성 불필요
  - 상대적으로 단순한 구성, 간단한 이벤트 처리에 적합

- 단점
  - 응용 프로그램이 직접 컴포넌트들의 핸들러를 제공한다는 점에서 구조적이지 못함

Q&A

담당교수 : 신성

E-mail : sihns@gachon.ac.kr

연구실 : IT대학 310호

휴대폰 번호 : 010-8873-8353

카카오톡 아이디: sihns929