09

자바의 이벤트 처리

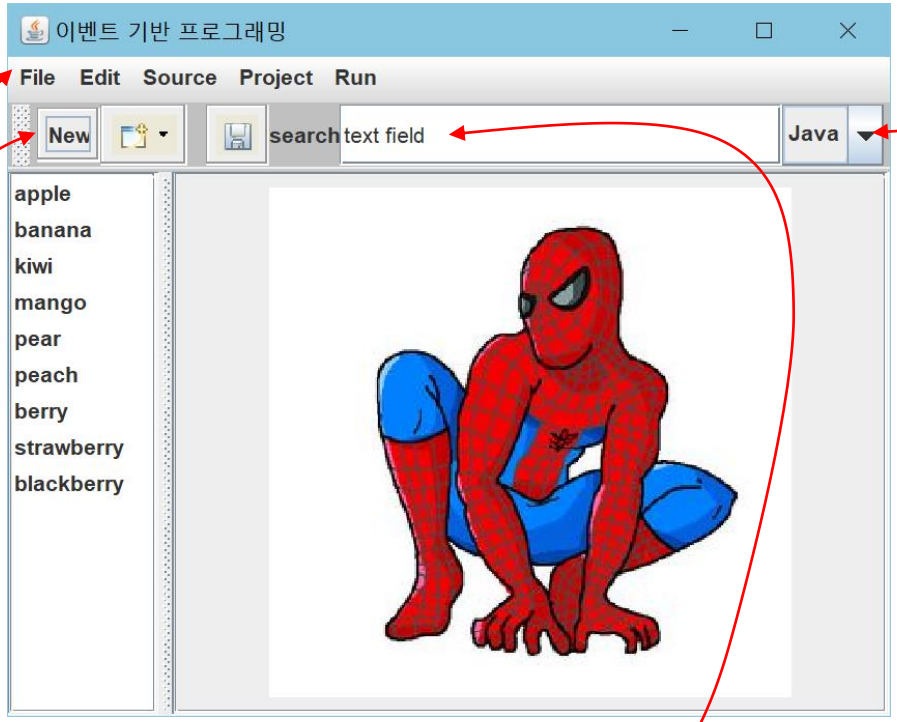
학습 목표

1. 자바의 이벤트 기반 GUI 프로그램 구조 이해
2. 이벤트와 이벤트 객체
3. 이벤트 리스너 작성
4. 어댑터 클래스로 리스너 작성
5. Key 이벤트로 키 입력 받기
6. Mouse 이벤트로 마우스 동작 인식

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
  - 반대되는 개념 : 배치 실행(batch programming)
    - 프로그램의 개발자가 프로그램의 흐름을 결정하는 방식
  - 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)

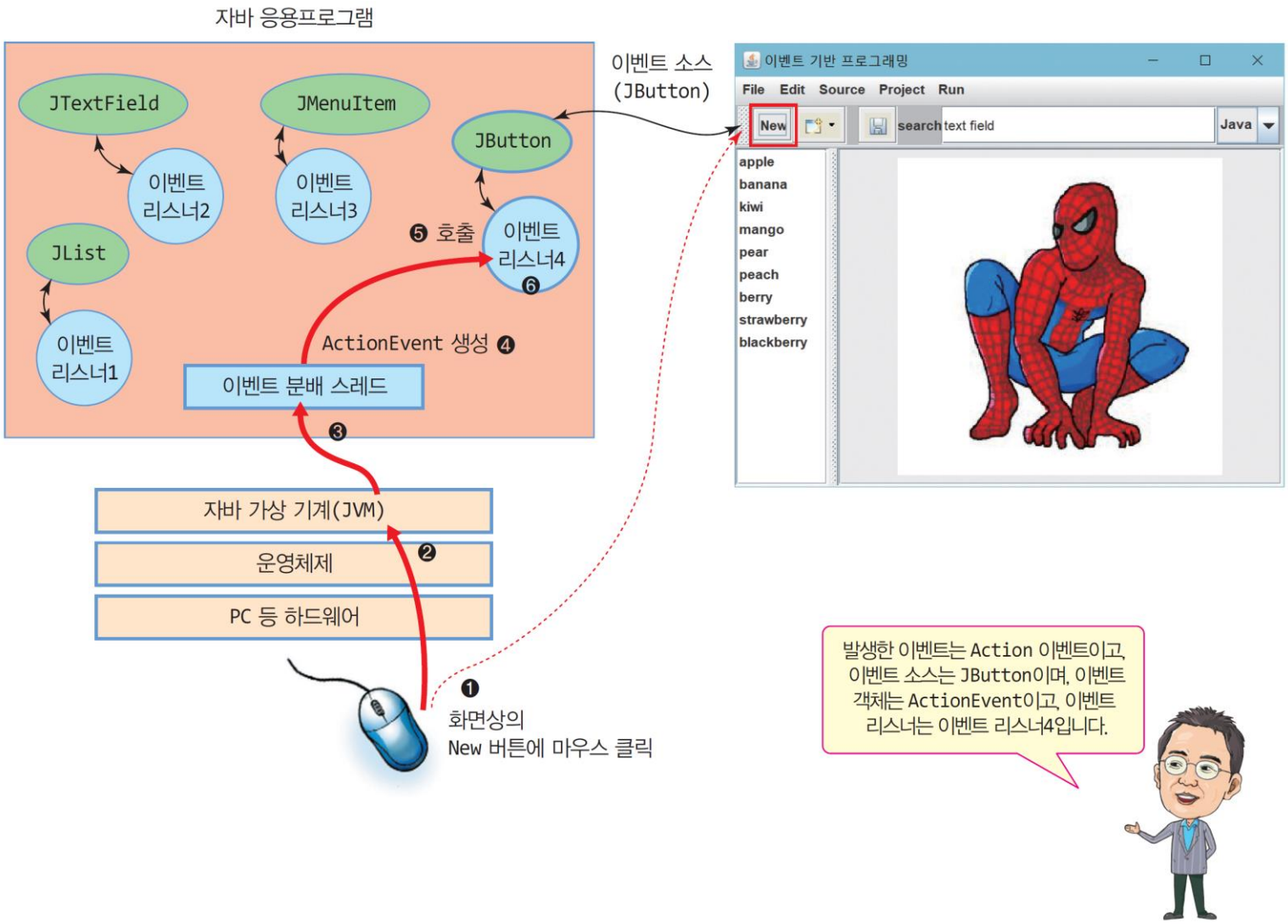
키보드 입력
(Key Event)

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



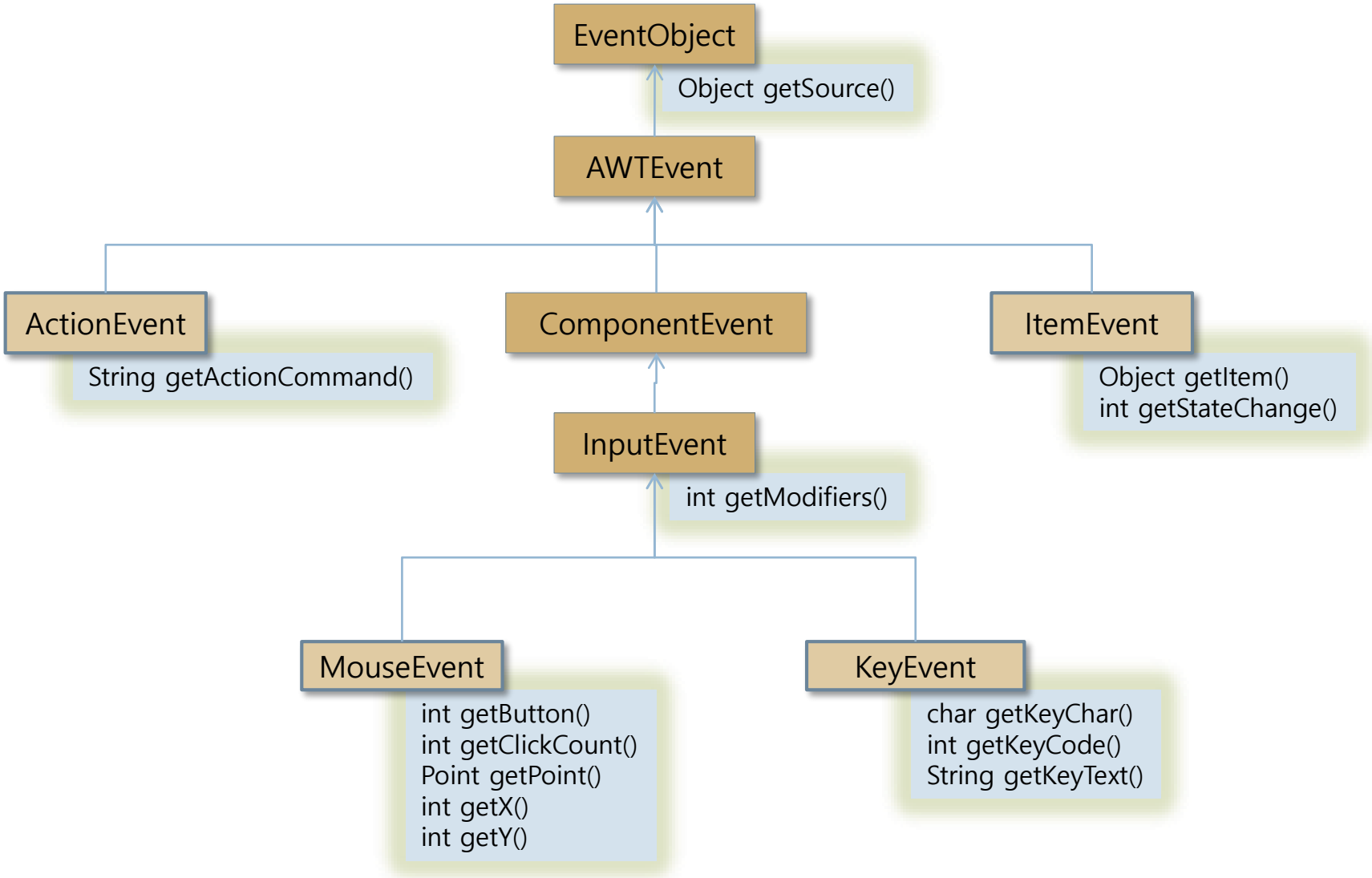
이벤트 객체

- 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
  - Object getSource()
    - 발생한 이벤트의 소스 컴포넌트 리턴
    - Object 타입으로 리턴하므로 캐스팅하여 사용
    - 모든 이벤트 객체에 대해 적용

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
WindowEvent	Window	Window를 상속받는 모든 컴포넌트에 대해 윈도우 활성화, 비활성화, 아이콘화, 아이콘에서 복구, 윈도우 열기, 윈도우 닫기, 윈도우 종료
ComponentEvent	Component	컴포넌트가 사라지거나, 나타나거나, 이동, 크기 변경 시
ContainerEvent	Container	Container에 컴포넌트 추가 혹은 삭제 시

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
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
    }
}
```

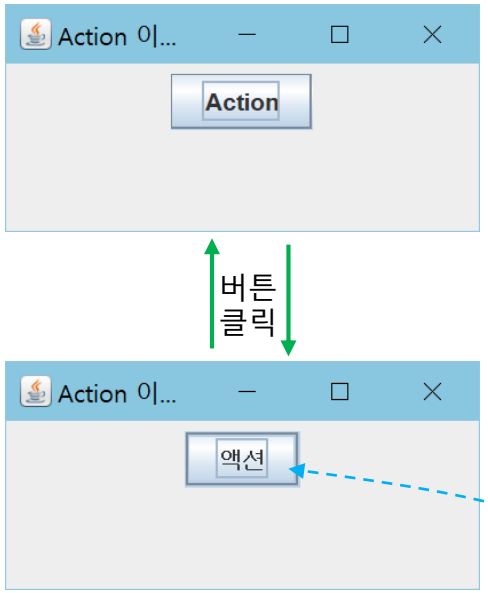
3. 이벤트 리스너 등록
   - 이벤트를 받아 처리하고자 하는 컴포넌트에 이벤트 리스너 등록
   - component.addXXXListener(listener)
     - xxx : 이벤트 명, listener : 이벤트 리스너 객체

```
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

예제 9-1 : 독립 클래스로 Action 이벤트 리스너 만들기



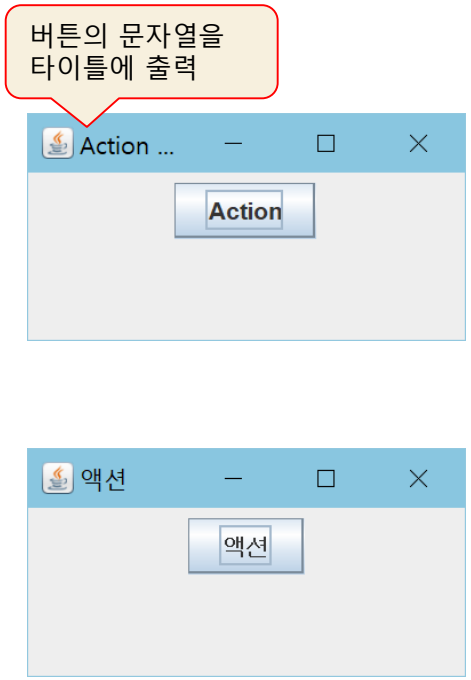

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class IndepClassListener extends JFrame {
    public 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 Action 이벤트 리스너 작성
- 이 클래스를 별도의 MyActionListener.java 파일로 작성하여도 됨

예제 9-2 : 내부 클래스로 Action 이벤트 리스너 만들기



- Action 이벤트 리스너를 내부 클래스로 작성
- private으로 선언하여 외부에서 사용할 수 없게 함

- 리스너에서 InnerClassListener와 JFrame 멤버에 대한 접근 용이

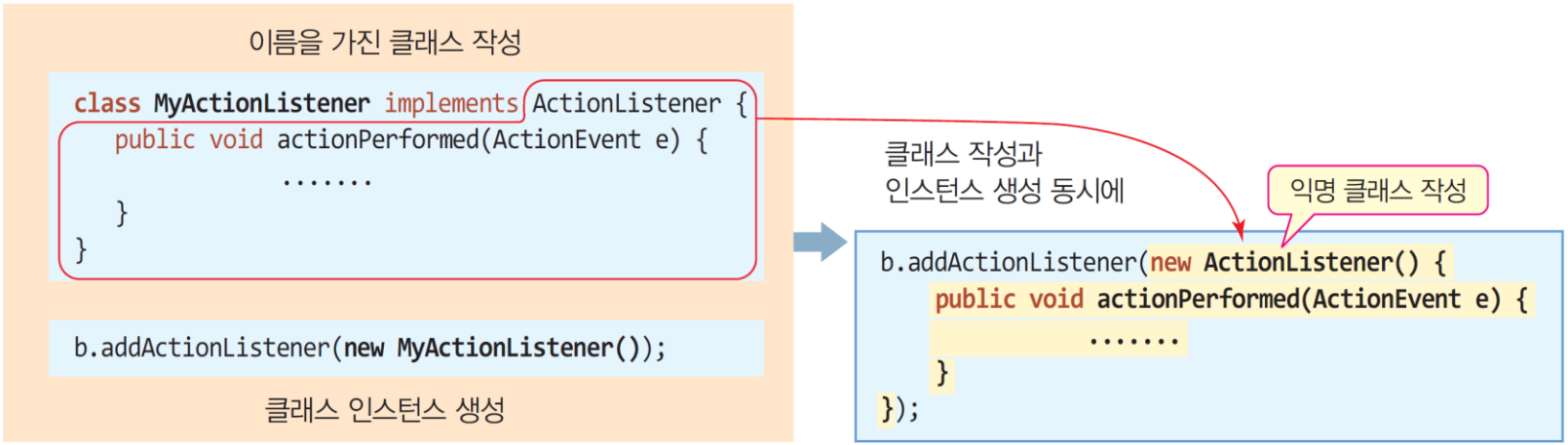
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class IndepClassListener extends JFrame {
    public 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);
    }
    private 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");
            // InnerClassListener의 멤버나 JFrame의 멤버를 호출
            InnerClassListener.this.setTitle(b.getText()); // 타이틀에
                                                버튼 문자열을 출력
        }
    }
    public static void main(String [] args) {
        new IndepClassListener();
    }
}
```

익명 클래스로 이벤트 리스너 작성

- 익명 클래스(anonymous class) : 이름 없는 클래스
  - (클래스 선언 + 인스턴스 생성)을 한번에 달성

```
new 익명클래스의슈퍼클래스이름(생성자인자들) {
    .....
    익명클래스의 멤버 구현
    .....
};
```

  - 간단한 리스너의 경우 익명 클래스 사용 추천
    - 메소드의 개수가 1, 2개인 리스너(ActionListener, ItemListener)에 대해 주로 사용
- ActionListener를 구현하는 익명의 이벤트 리스너 작성 예



(a) 이름을 가진 클래스를 작성하고 클래스 인스턴스 생성하는 경우

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

예제 9-3 : 익명 클래스로 Action 이벤트 리스너 만들기

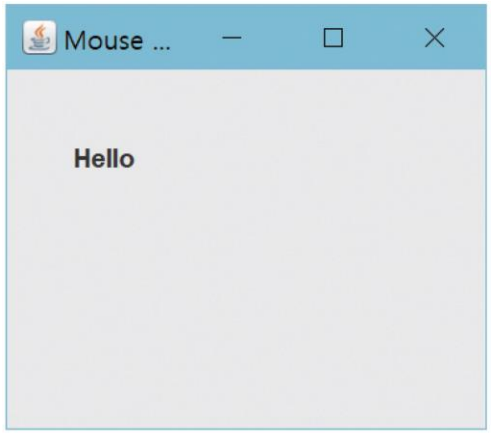
익명 클래스로 Action 리스너 작성

AnonymousClassListener의 멤버나 JFrame의 멤버를 호출할 수 있음

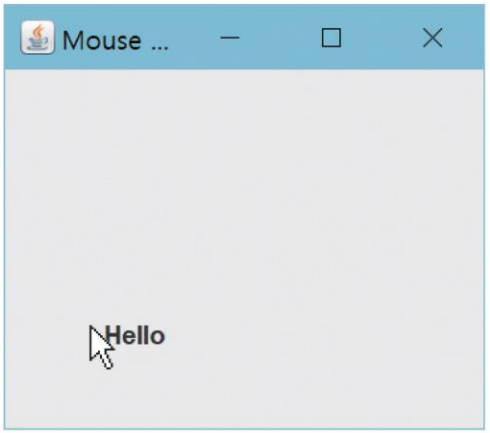
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class AnonymousClassListener extends JFrame {
    public AnonymousClassListener() {
        setTitle("Action 이벤트 리스너 작성");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container c = getContentPane();
        c.setLayout(new FlowLayout());
        JButton btn = new JButton("Action");
        c.add(btn);

        btn.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JButton b = (JButton)e.getSource();
                if(b.getText().equals("Action"))
                    b.setText("액션");
                else
                    b.setText("Action");
                setTitle(b.getText());
            }
        });

        setSize(350, 150);
        setVisible(true);
    }

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

예제 9-4 : 마우스 이벤트 리스너 작성 연습 - 마우스로 문자열 이동시키기

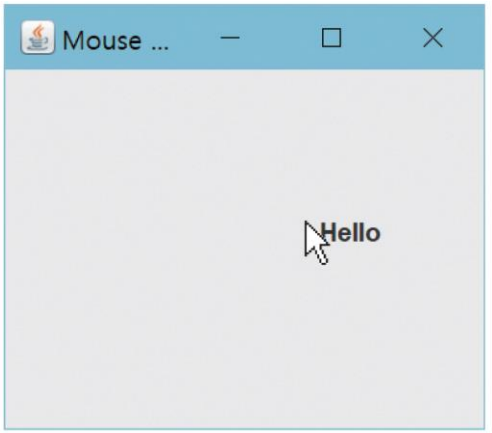
아래 실행 화면과 같이 프레임의 임의의 위치에 마우스 버튼을 누르면 마우스 포인터가 있는 위치에 "Hello" 문자열을 출력하는 프로그램을 작성하라.



초기 화면



마우스를 다른 곳에 클릭한 경우



마우스를 다른 곳에 클릭한 경우

- 마우스 버튼을 누르면 마우스가 있는 위치로 "Hello" 문자열을 이동시킨다.
- 이벤트와 리스너 : MouseEvent와 MouseListener
- 이벤트 소스 : 컨텐트팬
- 컨텐트팬의 배치관리자 : 배치관리자 삭제
- 구현할 리스너의 메소드 : mousePressed()
- "Hello" 문자열 : JLabel 컴포넌트 이용

예제 9-4의 정답

마우스 버튼이 눌러진 위치를 알아내어 "Hello" 를 옮긴다.

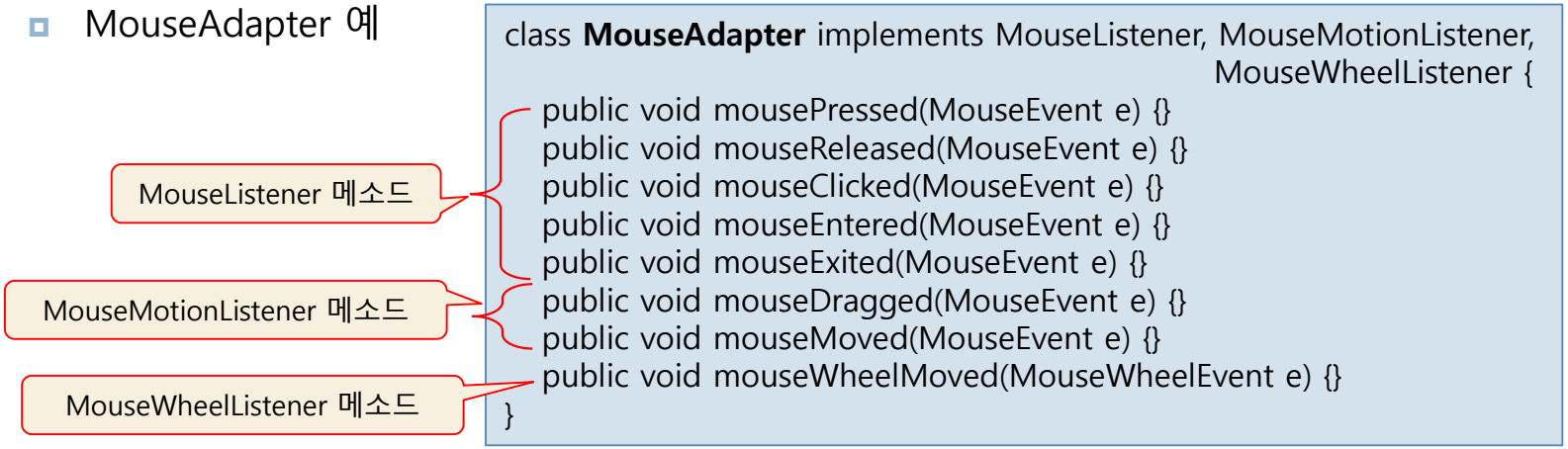
MouseListener의 5개 메소드를 모두 구현한다.

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

public class MouseListenerEx extends JFrame {
   private JLabel la = new JLabel("Hello"); // "Hello" 레이블

   public MouseListenerEx() {
      setTitle("Mouse 이벤트 예제");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      c.addMouseListener(new MyMouseListener());

      c.setLayout(null);
      la.setSize(50, 20); // 레이블의 크기 50x20 설정
      la.setLocation(30, 30); // 레이블의 위치 (30,30)으로 설정
      c.add(la);

      setSize(200, 200);
      setVisible(true);
   }
```

```
   class MyMouseListener implements MouseListener {
      public void mousePressed(MouseEvent e) {
         int x = e.getX(); // 마우스의 클릭 좌표 x
         int y = e.getY(); // 마우스의 클릭 좌표 y
         la.setLocation(x, y); // (x,y) 위치로 레이블 이동
      }

      public void mouseReleased(MouseEvent e) {}
      public void mouseClicked(MouseEvent e) {}
      public void mouseEntered(MouseEvent e) {}
      public void mouseExited(MouseEvent e) {}
   }

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

어댑터 클래스

- 이벤트 리스너 구현에 따른 부담
  - 리스너의 추상 메소드를 모두 구현해야 하는 부담
  - 예) 마우스 리스너에서 마우스가 눌러지는 경우(mousePressed())만 처리하고자 하는 경우에도 나머지 4 개의 메소드를 모두 구현해야 하는 부담
- 어댑터 클래스(Adapter)
  - 리스너의 모든 메소드를 단순 리턴하도록 만든 클래스(JDK에서 제공)
  - MouseAdapter 예



  - 추상 메소드가 하나뿐인 리스너는 어댑터 없음
    - ActionAdapter, ItemAdapter 클래스는 존재하지 않음

MouseListener대신 MouseAdapter를 사용한 예


```
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

JDK에서 제공하는 어댑터 클래스

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

예제 9-5 : MouseAdapter로 마우스 리스너 작성

MouseAdapter를 이용하여 예제 9-4를 수정하라.

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

public class MouseAdapterEx extends JFrame {
   private JLabel la = new JLabel("Hello"); // "Hello" 레이블

   public MouseListenerEx() {
      setTitle("Mouse 이벤트 예제");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      c.addMouseListener(new MyMouseAdapter());

      c.setLayout(null);
      la.setSize(50, 20); // 레이블의 크기 50x20 설정
      la.setLocation(30, 30); // 레이블의 위치 (30,30)으로 설정
      c.add(la);

      setSize(200, 200);
      setVisible(true);
   }
```

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

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

Key 이벤트와 포커스

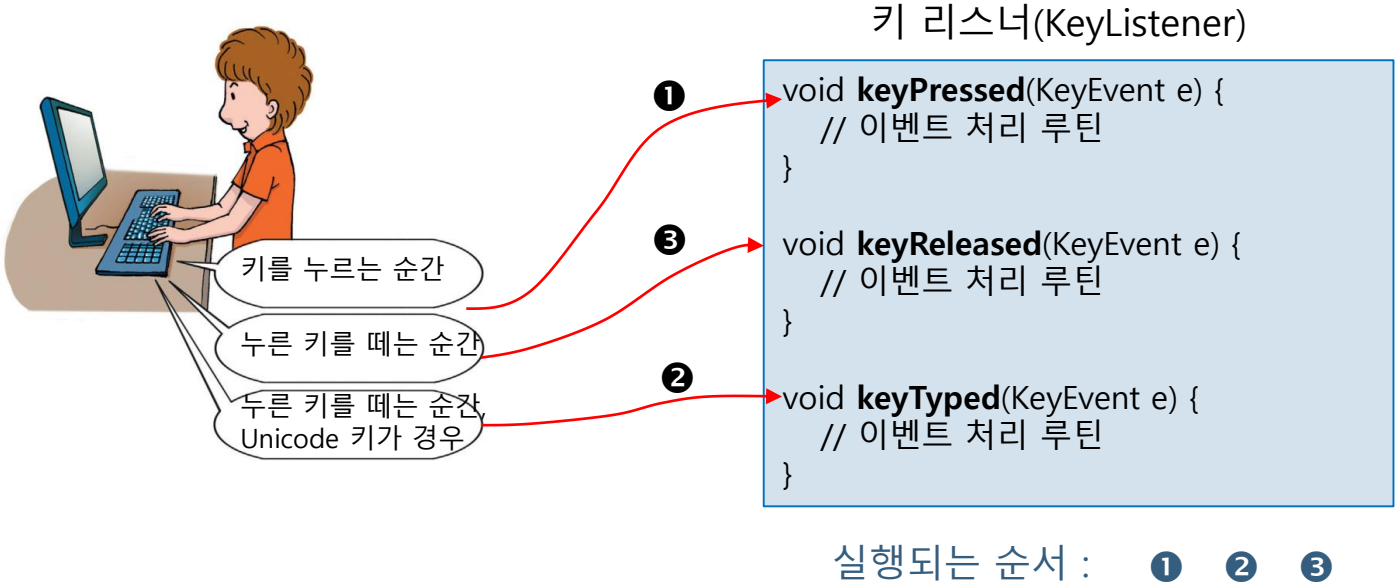
- 키 입력 시, 다음 세 경우 각각 Key 이벤트 발생
  - 키를 누르는 순간
  - 누른 키를 떼는 순간
  - 누른 키를 떼는 순간(Unicode 키의 경우에만)
- 키 이벤트를 받을 수 있는 조건
  - 모든 컴포넌트
  - 현재 포커스(focus)를 가진 컴포넌트가 키 이벤트 독점
- 포커스(focus)
  - 컴포넌트나 응용프로그램이 키 이벤트를 독점하는 권한
  - 컴포넌트에 포커스 설정 방법 : 다음 2 라인 코드 필요

```
component.setFocusable(true); // component가 포커스를 받을 수 있도록 설정
component.requestFocus(); // componen에 포커스 강제 지정
```

    - 자바플랫폼마다 실행 환경의 초기화가 서로 다를 수 있기 때문에 다음 코드가 필요함
      component.setFocusable(true);

KeyListener

- 응용프로그램에서 KeyListener를 상속받아 키 리스너 구현
- KeyListener의 3 개 메소드



- 컴포넌트에 키 이벤트 리스너 달기

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

유니코드(Unicode) 키

- 유니코드 키의 특징
  - 국제 산업 표준
  - 전 세계의 문자를 컴퓨터에서 일관되게 표현하기 위한 코드 체계
  - 문자들에 대해서만 키 코드 값 정의
    - A~Z, a~z, 0~9, !, @, & 등
  - 문자가 아닌 키 경우에는 표준화된 키 코드 값 없음
    - <Function> 키, <Home> 키, <Up> 키,<Delete> 키, <Control> 키, <Shift> 키, <Alt> 등은 플랫폼에 따라 키 코드 값이 다를 수 있음
- 유니코드 키가 입력되는 경우
  - keyPressed(), keyTyped(), keyReleased() 가 순서대로 호출
- 유니코드 키가 아닌 경우
  - keyPressed(), keyReleased() 만 호출됨

가상 키와 입력된 키 판별

- KeyEvent 객체
  - 입력된 키 정보를 가진 이벤트 객체
  - KeyEvent 객체의 메소드로 입력된 키 판별

- KeyEvent 객체의 메소드로 입력된 키 판별
  - char KeyEvent.getKeyChar()
    - 키의 유니코드 문자 값 리턴
    - Unicode 문자 키인 경우에만 의미 있음
    - 입력된 키를 판별하기 위해 문자 값과 비교하면 됨

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

q 키가 누르면 프로그램 종료

  - int KeyEvent.getKeyCode()
    - 유니코드 키 포함
    - 모든 키에 대한 정수형 키 코드 리턴
    - 입력된 키를 판별하기 위해 가상키(Virtual Key) 값과 비교하여야 함
    - 가상 키 값은 KeyEvent 클래스에 상수로 선언

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_F5)
        System.exit(0);
}
```

F5 키를 누르면 프로그램 종료

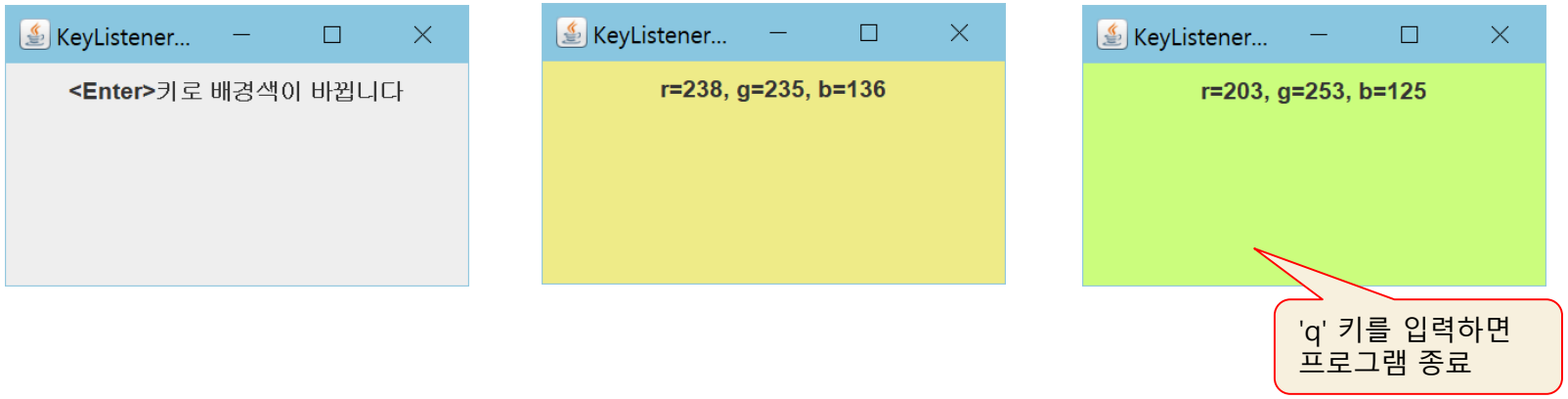
가상 키(Virtual Key)

- 가상 키는 KeyEvent 클래스에 상수로 선언
- 가상 키의 일부 소개

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

예제 9-6 : KeyListener 활용 – 입력된 문자 키 판별

컨텐트팬에 <Enter> 키를 입력할 때마다 배경색을 랜덤하게 바꾸고, 'q' 키를 입력하면 프로그램을 종료시켜라.



- 컨텐트팬에 키 리스너를 달고, 포커스를 주어, 키 입력을 받도록 해야 한다.
- 색은 new Color(int r, int g, int b)로 생성한다.
  r(red), g(green), b(blue)는 색의 3요소로서 0~255 사이의 값이다.

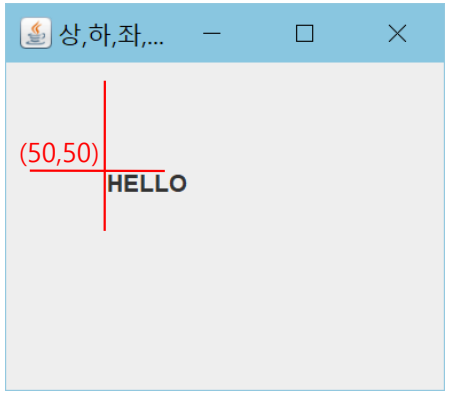
예제 9-6 정답


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

public class KeyCharEx extends JFrame {
   private JLabel la =
      new JLabel("<Enter>키로 배경색이 바뀝니다");

   public KeyCharEx() {
      super("KeyListener의 문자 키 입력 예제");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      c.setLayout(new FlowLayout());
      c.add(la);
      c.addKeyListener(new MyKeyListener());
      setSize(250, 150);
      setVisible(true);
      c.setFocusable(true); // 컨텐트팬이 포커스를 받을 수 있도록 설정
      c.requestFocus(); // 컨텐트팬에 포커스 설정
   }
```

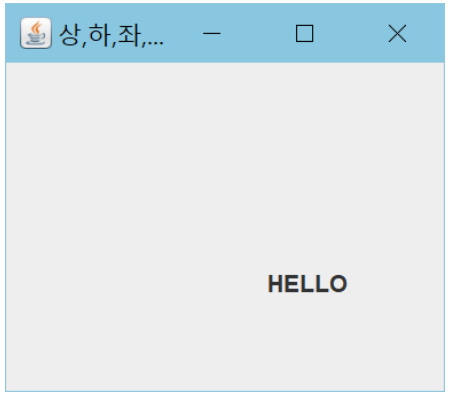
```
   class MyKeyListener extends KeyAdapter {
      public void keyPressed(KeyEvent e) {
         // 임의의 색을 만들기 위해 랜덤하게 r, g, b 성분 생성
         int r = (int) (Math.random() * 256); // red 성분
         int g = (int) (Math.random() * 256); // green 성분
         int b = (int) (Math.random() * 256); // blue 성분

         switch(e.getKeyChar()) { // 입력된 키 문자
            case '₩n': // <Enter> 키 입력
               la.setText("r=" + r + ", g=" + g + ", b=" + b);
               getContentPane().setBackground(
                                          new Color(r, g, b));
               break;
            case 'q':
               System.exit(0);
         }
      }
   }
   public static void main(String[] args) {
      new KeyCharEx();
   }
}
```

예제 9-7 : KeyListener 활용 – 상, 하, 좌, 우 키로 문자열 움직이기

상, 하, 좌, 우 키를 입력하면, 다음 그림과 같이 "HELLO" 문자열이 10픽셀씩 이동하는 프로그램을 작성하라.



초기 상태



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

상,하,좌,우 키를 움직이면 한 번에 10픽셀씩 "HELLO" 텍스트는 상,하,좌,우로 이동한다. 이 텍스트는 프레임의 영역을 벗어나서 움직일 수 있다.

예제 9-7 정답


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

public class FlyingTextEx extends JFrame {
    private JPanel contentPane = new JPanel();
    private JLabel la = new JLabel("HELLO");

    public FlyingTextEx() {
        super("상,하,좌,우 키를 이용하여 텍스트 움직이기");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        Container c = getContentPane();
        c.setLayout(null);
        c.addKeyListener(new MyKeyListener());

        la.setLocation(50, 50);
        la.setSize(100, 20);
        c.add(la);

        setSize(200, 200);
        setVisible(true);

        c.setFocusable(true); // 컨텐트팬이 포커스를 받을 수 있도록 설정
        c.requestFocus(); // 포커스 지정
    }
```

```
    class MyKeyListener extends KeyAdapter {
        public void keyPressed(KeyEvent e) {
            int keyCode = e.getKeyCode(); // 입력된 키코드
            switch(keyCode) {
                case KeyEvent.VK_UP:
                    la.setLocation(la.getX(), la.getY() - 10); break;
                case KeyEvent.VK_DOWN:
                    la.setLocation(la.getX(), la.getY() + 10); break;
                case KeyEvent.VK_LEFT:
                    la.setLocation(la.getX() - 10, la.getY()); break;
                case KeyEvent.VK_RIGHT:
                    la.setLocation(la.getX() + 10, la.getY()); break;
            }
        }
    }

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

Mouse 이벤트와 MouseListener, MouseMotionListener

- Mouse 이벤트 : 사용자의 마우스 조작에 따라 발생하는 이벤트

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

  - mouseClicked() : 마우스가 눌러진 위치에서 그대로 떼어질 때 호출
  - mouseReleased() : 마우스가 눌러진 위치에서 그대로 떼어지든 아니든 항상 호출
  - mouseDragged(): 마우스가 드래그되는 동안 계속 여러번 호출
- 마우스가 눌러진 위치에서 떼어지는 경우 메소드 호출 순서

  mousePressed(), mouseReleased(), mouseClicked()
- 마우스가 드래그될 때 호출되는 메소드 호출 순서

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

마우스 리스너 달기와 MouseEvent 객체 활용

- 마우스 리스너 달기
  - 마우스 리스너는 컴포넌트에 다음과 같이 등록

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

  - 컴포넌트가 마우스 무브(mouseMoved())나 마우스 드래깅(mouseDraggecd())을 함께 처리하고자 하면, MouseMotion 리스너 따로 등록

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

- MouseEvent 객체 활용
  - 마우스 포인터의 위치, 컴포넌트 내 상대 위치
    - int getX(), int getY()

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

  - 마우스 클릭 횟수
    - int getClickCount()

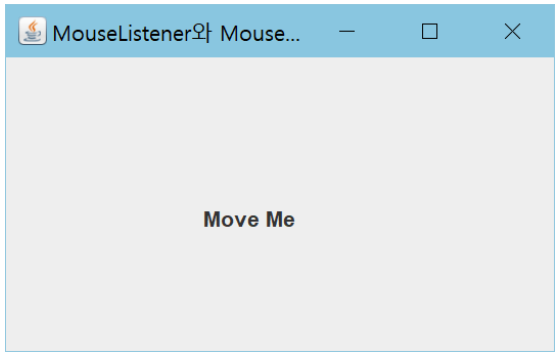
```
public void mouseClicked(MouseEvent e) {
    if(e.getClickCount() == 2) {
        ... // 더블클릭 처리 루틴
    }
}
```

마우스 이벤트 처리 예 : MouseListener와 MouseMotionListener

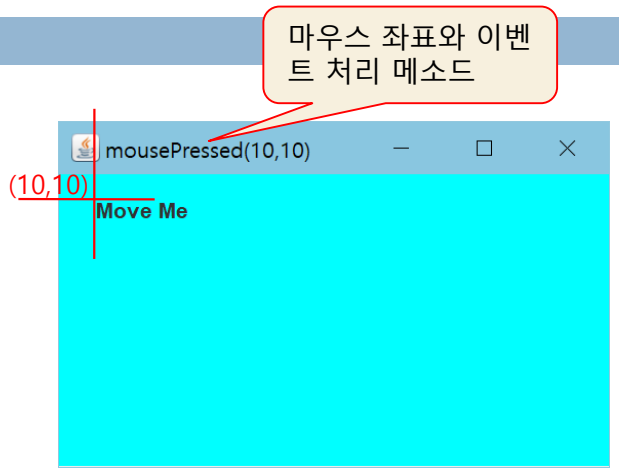
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;

public class MouseEventAllEx extends JFrame {
   private JLabel la = new JLabel(" Move Me");

   public MouseEventAllEx() {
      setTitle("MouseListener와 MouseMotionListener 예제");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      MyMouseListener listener =
                              new MyMouseListener();
      c.addMouseListener(listener);
      c.addMouseMotionListener(listener);

      c.setLayout(null);

      la.setSize(80,20);
      la.setLocation(100,80);
      c.add(la); // 레이블 컴포넌트 삽입

      setSize(300,200);
      setVisible(true);
   }
```

```
   class MyMouseListener implements MouseListener,
                  MouseMotionListener {
      public void mousePressed(MouseEvent e) {
         la.setLocation(e.getX(), e.getY());
         setTitle("mousePressed("+e.getX()+","+e.getY()+")");
      }
      public void mouseReleased(MouseEvent e) {
         la.setLocation(e.getX(), e.getY());
         setTitle("mouseReleased("+e.getX()+","+e.getY()+")");
      }
      public void mouseClicked(MouseEvent e) {}
      public void mouseEntered(MouseEvent e) {
         Component comp = (Component)e.getSource();
         comp.setBackground(Color.CYAN);
      }
      public void mouseExited(MouseEvent e) {
         Component comp = (Component)e.getSource();
         comp.setBackground(Color.YELLOW);
         setTitle("mouseExited("+e.getX()+","+e.getY()+")");
      }
      public void mouseDragged(MouseEvent e) {
         la.setLocation(e.getX(), e.getY());
         setTitle("mouseDragged("+e.getX()+","+e.getY()+")");
      }
      public void mouseMoved(MouseEvent e) {
         la.setLocation(e.getX(), e.getY());
         setTitle("mouseMoved ("+e.getX()+","+e.getY()+")");
      }
   }
   public static void main(String [] args) {
      new MouseEventAllEx();
   }
}
```

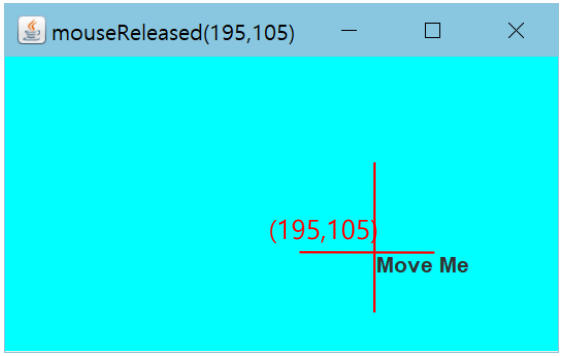
마우스 이벤트 처리 실행



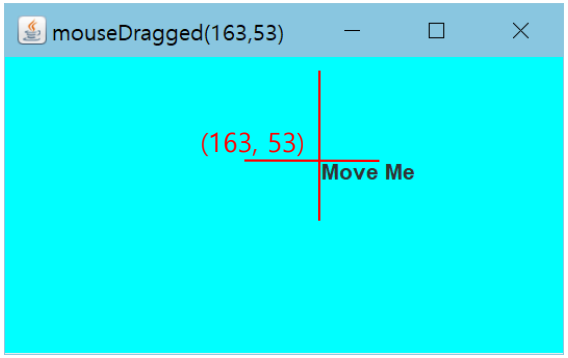
초기화면



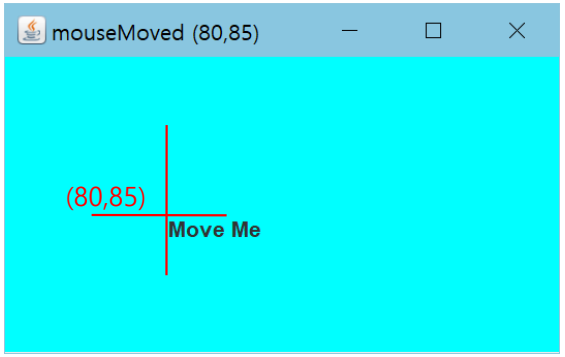
mouseEntered()에 의해 배경색 변경.
마우스 버튼이 눌러진 순간



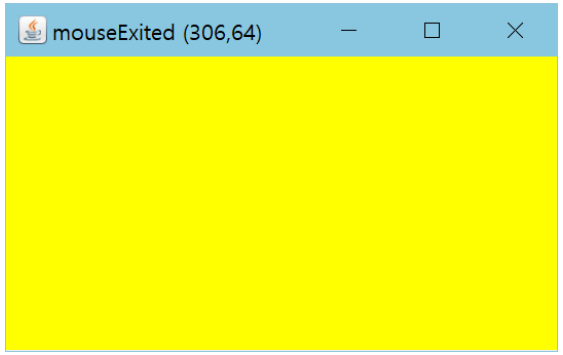
눌러진 마우스 버튼이 떼어진 순간



마우스가 컨텐트팬 위에
드래깅하는 동안



마우스가 컨텐트팬 위에
이동하는 동안



마우스가 컨텐트팬을 벗어나면
mouseExited()에 의해 배경색 변경

<참고> Lambda Expressions

- 기본문법
  - `(param1, param2, ...) -> { /* codes */ };`
    - 매개변수를 받아 code블럭을 처리
      - *param의 타입은 생략 가능*
      - *param이 없을 때는 ( )만 기술*
      - *param이 1개일 때는 ( ) 생략 가능*
      - *code가 1개의 문장일 때는 { } 생략 가능*
- 예1
  - (int x) -> { return x * x; } // x -> x * x;
  - (x, y) -> { System.out.println(x + " " + y) }
  - ( ) -> { System.out.println("Hello") }
- 예2

```
btn.setOnAction( (ActionEvent event) ->
                    { System.out.println("Hello World"); }  );
btn.setOnAction( (event) ->
                    System.out.println("Hello World") );
btn.setOnAction( event ->
                    System.out.println("Hello World") );
```

Functional Interface

- Target Type
  - 람다식(메소드)를 포함하는 인터페이스
    - 함수적인터페이스 변수명 = 람다식;

인터페이스	메소드	특징	비고
Consumer	accept()	Argments(O), Return(X)	인수만…
Supplier	get()	Argments(X), Return(O)	반환만…
Function	apply()	Argments(O), Return(O)	인수를 리턴(타입변환)
Operator	applyXXX()	Argments(O), Return(O)	연산결과를 리턴
Predicate	test()	Argments(O), Return(B)	리턴타입은 Boolean

- FunctionalInterface
  - 하나의 추상 method만 가진 인터페이스

- Consumer
  - 인수는 있고, 반환은 없는 함수적 인터페이스

인터페이스	메소드	비고
Consumer<T>	void accept(T t)	
BiConsumer<T,U>	void accept(T t, U u)	
DoubleConsumer	void accept(double n)	
IntConsumer	void accept(int n)	
LongConsumer	void accept(long n)	
ObjDoubleConsumer<T>	void accept(T t, double n)	
ObjIntConsumer<T>	void accept(T t, int n)	
ObjLongConsumer<T>	void accept(T t, long n)	

```
import java.util.function.*;

public class FuncInterfaceEx
{
     public static void main(String[] args)
     {
          Consumer<String> stringConsumer
                              = str -> System.out.println(str);
          BiConsumer<String,String> biConsumer
                              = (t,u) -> System.out.println(t + u);
          DoubleConsumer doubleConsumer
                              = d -> System.out.println("java " + d);
          ObjDoubleConsumer<String> objDoubleConsumer
                              = (t, d) -> System.out.println(t + d);

          stringConsumer.accept("Hello!");
          biConsumer.accept("Hello ", "Java!");
          doubleConsumer.accept(8.0);
          objDoubleConsumer.accept("Java ", 8.0);
     }
}
```

```
Hello!
Hello Java!
java 8.0
Java 8.0
```

- Supplier
  - 인수는 없고, 반환은 있는 함수적 인터페이스

인터페이스	메소드	비고
Supplier<T>	T get()	T 리턴
BooleanSupplier	boolean getAsBoolean()	
DoubleSupplier	double getAsDouble()	
IntSupplier	int getAsInt()	
LongSupplier	long getAsLong()	

- Function (타입변환 T, U → R)
  - 인수도 있고, 반환도 있는 함수적 인터페이스

인터페이스	메소드
Function<T,R>	R apply(T t)
BiFunction<T,U,R>	R apply(T t, U u)
DoubleFunction<R>	R apply(double n)
IntFunction<R>	R apply(int n)
IntToDoubleFunction	double applyAsDouble(int n)
IntToLongFunction	long applyAsLong(int n)
LongToDoubleFunction	double applyAsDouble(long n)
LongToIntFunction	int applyAsInt(long n)
ToDoubleBiFunction<T,U>	double applyAsDouble(T t, U u)

- Operator
  - 인수도 있고, 반환도 있는 함수적 인터페이스
  - 인수의 연산결과를 같은 타입으로 반환

인터페이스	메소드
BinaryOperator<T>	BiFunction의 파생 인터페이스
UnaryOperator<T>	Function의 파생 인터페이스
DoubleBinaryOperator	double applyAsDouble(double,double)
DoubleUnaryOperator	double applyAsDouble(double)
IntBinaryOperator	int applyAsInt(int,int)
IntUnaryOperator	int applyAsInt(int)
LongBinaryOperator	long applyAsLong(long,long)
LongUnaryOperator	long applyAsLong(long)

- Predicate
  - 인수 있고, 반환은 boolean인 함수적 인터페이스

인터페이스	메소드	비고
Predicate<T>	boolean test(T t)	true/false
BiPredicate<T,U>	boolean test(T t, U u)	
DoublePredicate	boolean test(double n)	
IntPredicate	boolean test(int n)	
LongPredicate	boolean test(long n)	

Method References

- 목적: method 참조를 통해 lambda 식에서 불필요한 매개변수를 제거
  - 기존 method를 단순히 호출만 하는 경우 등
  - (람다식의 예)
    - (num1, num2) -> Math.max(num1, num2);
  - (메소드참조의 예)
    - Math :: max
  - (생성자참조의 예)
    - Classname :: new

```
import java.util.function.IntBinaryOperator;

class Calc
{
    public static int add(int x, int y) { return x + y; }
}

public class MethodRefEx1
{
    public static void main(String[] args)
    {
        IntBinaryOperator op1, op2;

        op1 = (x, y) -> Calc.add(x, y);
        op2 = Calc :: add;

        System.out.println(op1.applyAsInt(3, 7));
        System.out.println(op2.applyAsInt(3, 7));
    }
}
```

```
class Person
{
    private String name;
    private int age;

    public Person(String name) {
        this.name = name;
    }

    public Person(String name, int age) {
        this(name);
        this.age = age;
    }
}

public class MethodRefEx
{
    public static void main(String[] args)
    {
        Function<String, Person> func1 = Person :: new;
        Person man = func1.apply("Kim");

        BiFunction<String, Integer, Person> func2 = Person :: new;
        Person woman = func2.apply("Kim", 24);
    }
}
```