Getting GUI:
A Very Graphic Story

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

국립한경대학교
HANKYONG NATIONAL UNIV.

모든 것은 윈도우에서 시작된다

Jframe은 화면 위의 윈도우를 나타내는 객체이다.

- ✓ 버튼, 체크박스, 텍스트 필드 같은 인터페이스와 관련된 모든 것들을 이곳에 집어넣는다.
- ✓ 메뉴 항목이 들어 있는 메뉴 막대도 집어넣을 수 있다.
- ✓ Jframe의 모양은 플랫폼에 따라 다르게 보일 수 있다.

A JFrame with a menu bar and two '**widgets**' (a button and a radio button)

윈도우에 위젯을 추가해보자

일단 Jframe을 만들고 나면 그 Jframe에 각종 컴포넌트(위젯)를 넣을 수 있다.

- ✓ 윈도우에 집어넣을 수 있는 스윙 컴포넌트는 헤아릴 수 없이 많다.
  (javax.swing 패키지에 포함되어 있다)
- ✓ 자주 쓰이는 것들로는 JButton, JRadioButton, JCheckBox, JLabel, JList, JScrollPane, JSlider, JTextArea, JTextField, JTable 등이 있다.
- ✓ 대부분 사용법이 간단하지만 JTable처럼 조금 복잡한 것도 있다.

Making a GUI is easy:

1. Make a frame (a JFrame)

```
JFrame frame = new JFrame();
```

2. Make a widget (button, text field, etc.)

```
JButton button = new JButton("click me");
```

3. Add the widget to the frame

```
frame.getContentPane().add(button);
```

프레임에 뭔가를 추가할 때 직접 추가하지는 않는다.
프레임은 윈도우를 둘러싸고 있는 테두리라고 생각하면 된다.
실제 위젯을 추가할 때는 **틀**(pane)에 추가해야 한다.

4. Display it (give it a size and make it visible)

```
frame.setSize(300,300);
frame.setVisible(true);
```

첫 번째 GUI: 프레임 위의 버튼

```
import javax.swing.*;

public class SimpleGui1 {
    public static void main (String[] args) {

        JFrame frame = new JFrame();
        JButton button = new JButton("click me");


        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);



        frame.getContentPane().add(button);


        frame.setSize(300,300);


        frame.setVisible(true);
    }
}
```

don't forget to import this swing package

make a frame and a button

(you can pass the button constructor the text you want on the button)

this line makes the program quit as soon as you close the window (if you leave this out it will just sit there on the screen forever)

add the button to the frame's content pane

give the frame a size, in pixels

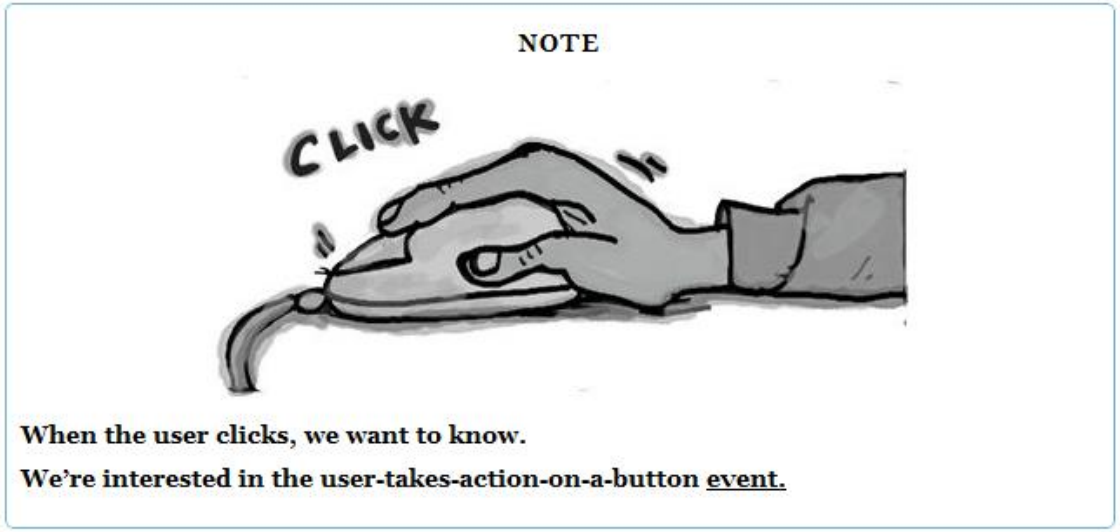
finally, make it visible!! (if you forget this step, you won't see anything when you run this code)

그런데 클릭해도 아무 반응이 없다

사용자가 버튼을 클릭했을 때 특정한 뭔가를 수행하도록 하려면 어떻게 해야 하나**?**

두 가지 일이 필요하다**:**

1. 사용자가 클릭했을 때 호출될 **메소드 (**버튼 클릭에 의해 일어나게 하고 싶은 것**)**
2. 그 메소드를 **언제** 실행시켜야 할 지를 알아내는 방법.
   즉**,** 사용자가 버튼을 클릭했는지를 알아내는 방법

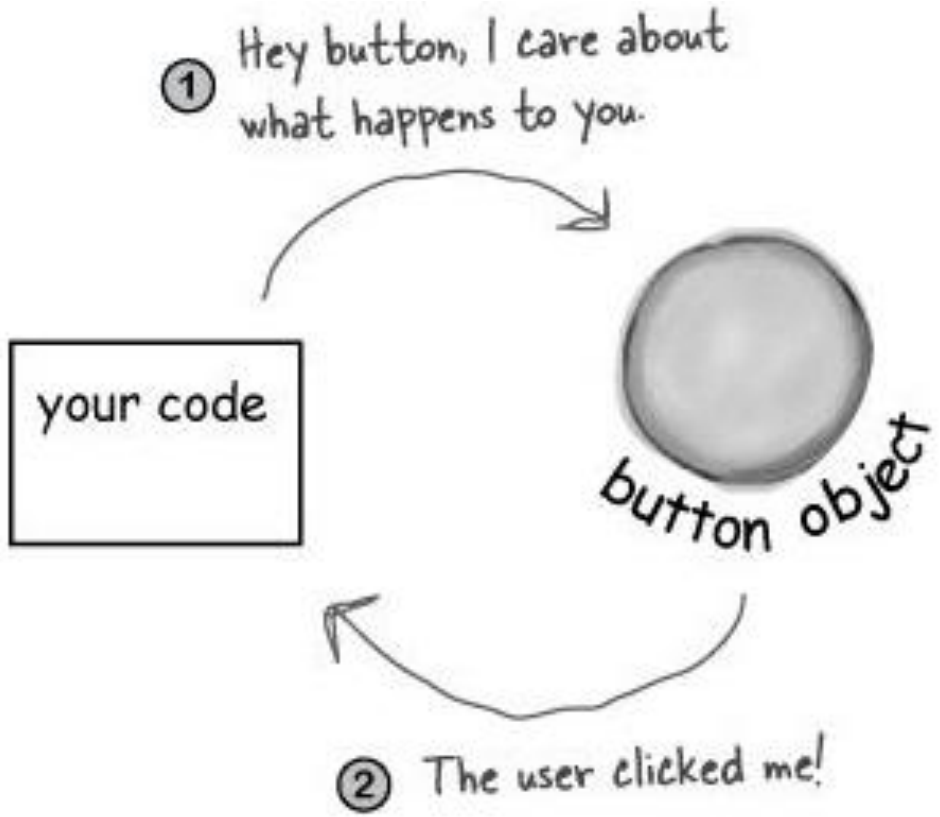


Getting a user event

사용자가 버튼을 클릭하면 버튼에 있는 텍스트가 click me에서 I've been clicked! 로 바뀌게 하고 싶다면?

```
public void changeIt() {
    button.setText("I've been clicked!");
}
```

- ✓ 이것 만으로는 부족하다. 이 메소드가 언제 실행되어야 하는지를 어떻게 알아낼 수 있을까? 즉, 버튼이 클릭되었는지 어떻게 알 수 있을까?
- ✓ 자바에서 사용자 이벤트를 받아서 처리하는 과정을 **이벤트 처리**(event handling)라고 한다.
- ✓ 자바에는 **다양한 이벤트 타입**이 있다.
- ✓ 사용자가 버튼을 클릭하는 것도 이벤트가 된다.



첫째, 버튼은 우리가 그 버튼에 대해 관심이 있다는 것을 알아야 한다.

둘째, 버튼은 버튼 클릭 이벤트가 발생했을 때 우리에게 콜백 해 줄 방법이 필요하다.

버튼의 이벤트에 관심이 있다면 “내가 너의 이벤트를 리스닝하고 있다”고 말해주는 인터페이스를 구현하라

리스너 인터페이스는 리스너(사용자)와 이벤트 소스(버튼)를 연결해주는 다리 역할을 해준다.

- ✓ 스윙 GUI 컴포넌트는 모두 이벤트 소스이다.
- ✓ 이벤트 소스란 사용자의 행동(마우스 클릭, 키 입력, 창 닫기 등)을 이벤트로 바꿔주는 객체를 뜻한다.

이벤트도 객체로 표현된다: 이벤트 클래스의 객체

- ✓ **java.awt.event** 패키지 참조: 모두 Event라는 이름을 가진다.
- ✓ MouseEvent, KeyEvent, WindowEvent, ActionEvent, and several others.

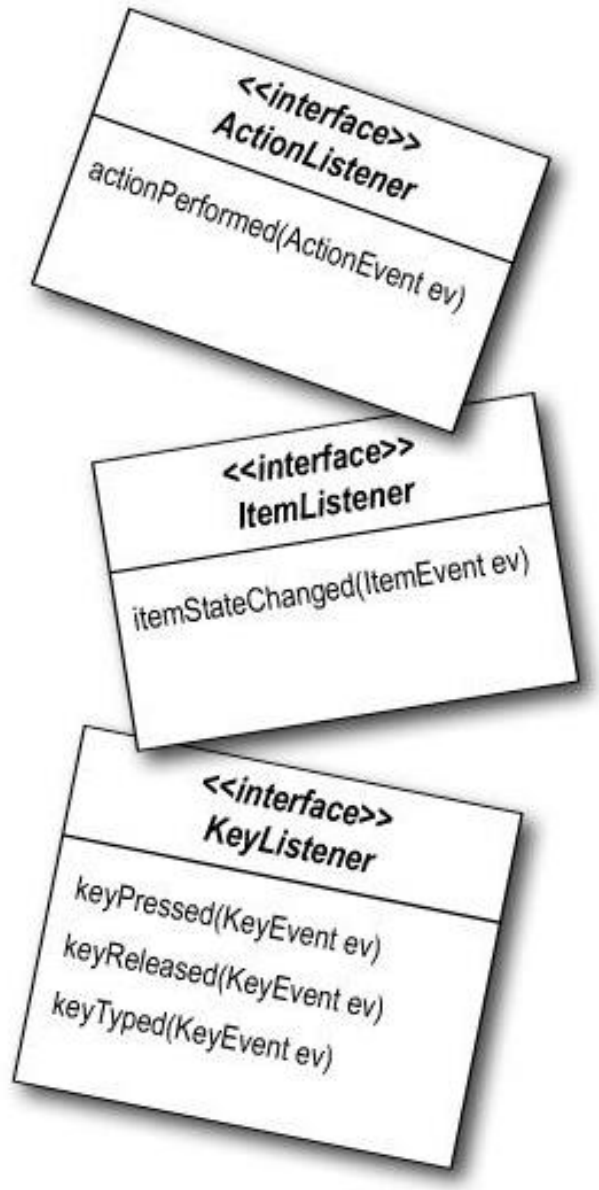
사용자가 리스너 인터페이스를 구현할 때 버튼에게 콜백할 방법을 제공해준다

인터페이스에 콜백 메소드가 선언되어 있다.

이벤트 소스(예: 버튼)는 사용자가 뭔가 의미있는 행동(예: 버튼 클릭)을 할 때 이벤트 객체를 생성한다.

작성하는 대부분의 코드는 이벤트를 생성하는 것보다는 이벤트를 받는 쪽이 될 것이다.

즉, 이벤트 소스보다는 이벤트 리스너에 대부분의 시간을 보낼 것이다.



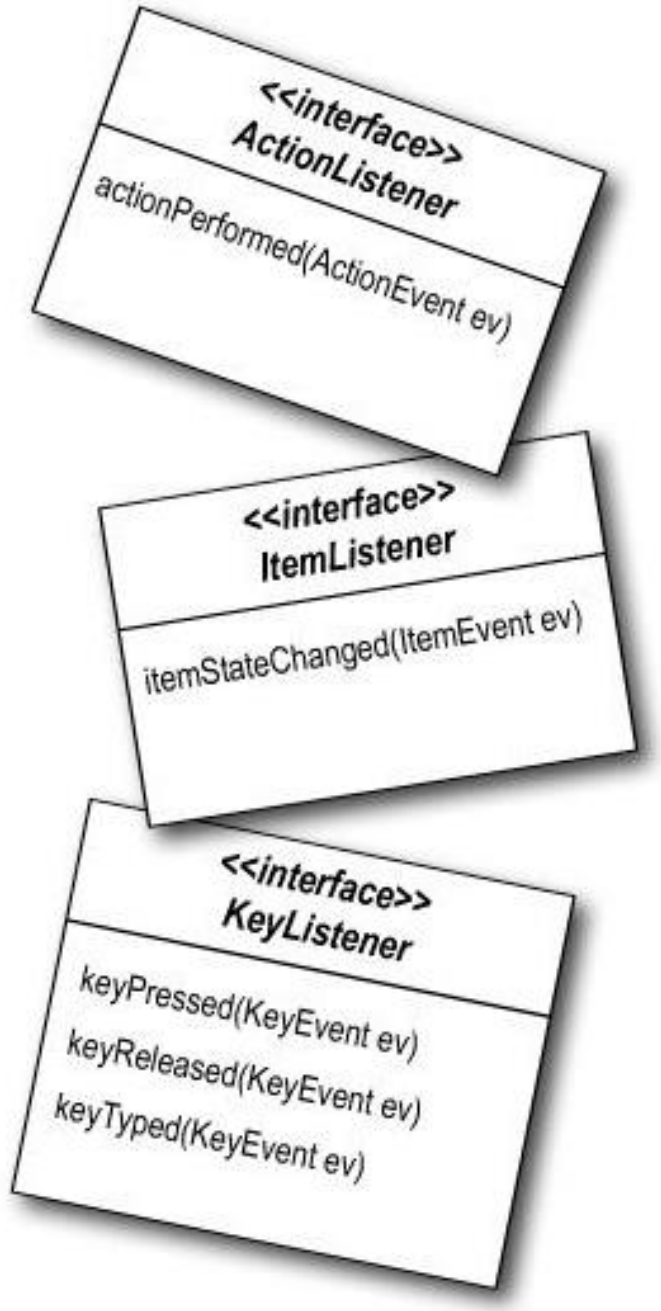
모든 이벤트 타입은 대응되는 리스너 인터페이스를 가지고 있다.

MouseEvent가 필요한가? => MouseListener를 구현

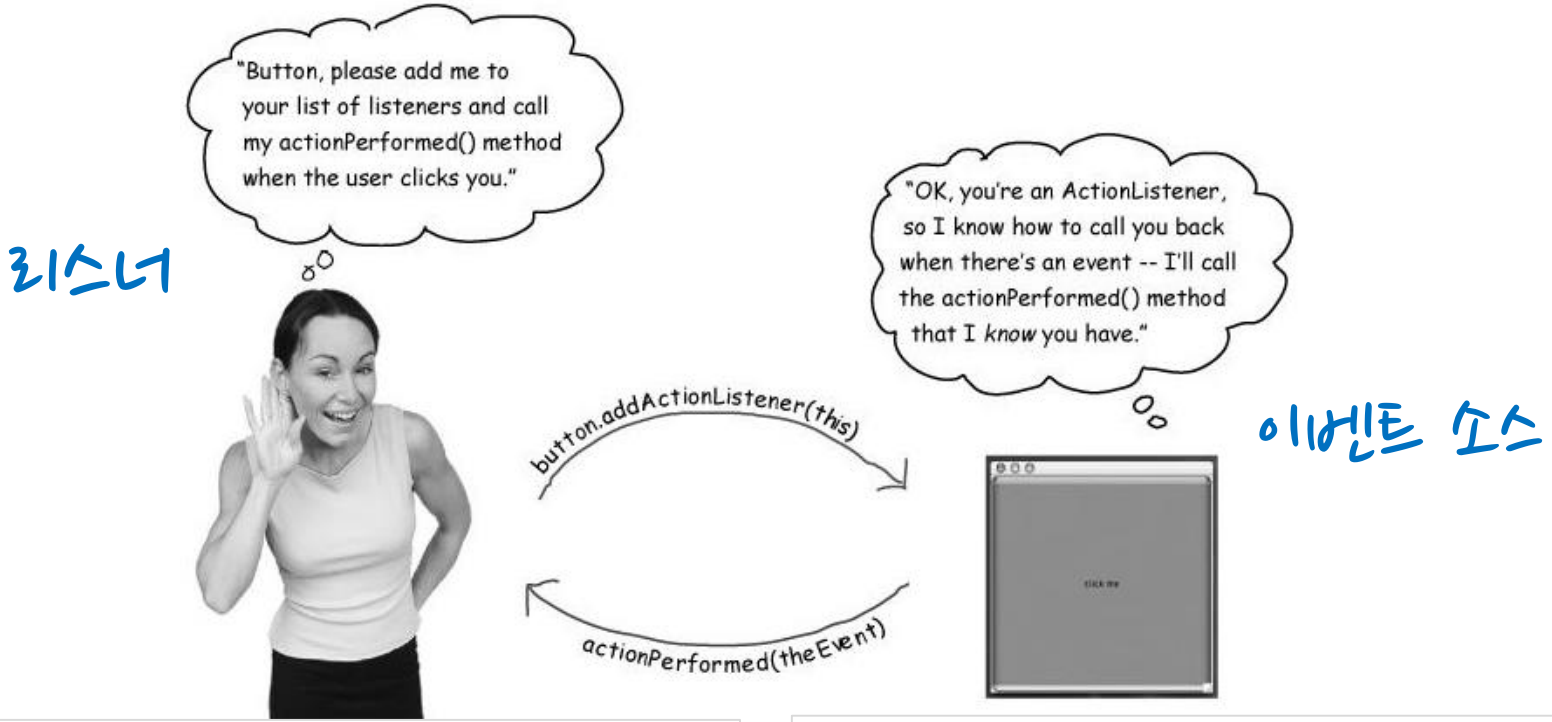
WindowEvent가 필요한가? => WindowListener를 구현

⇒ **Rule**: 인터페이스 안의 모든 메소드를 구현해야 한다.

어떤 메소드는 하나 이상의 메소드를 가진다.



리스너와 이벤트 소스 사이의 의사 소통 방법



클래스에서 어떤 버튼의 ActionEvent를 알고 싶다면 ActionListener 인터페이스를 구현하면 된다.

버튼이 누군가 관심이 있다는 것을 알아야 하기 때문에

⇒ addActionListener(this)를 호출하고

⇒ ActionListener 레퍼런스를 전달함으로써

버튼에게 등록한다.

버튼은 ActionEvent의 소스이므로 어느 객체가 관심 있는 리스너인지 알아야 한다.

버튼에는 **addActionListener()**라는 메소드가 있어서 그 버튼에 관심있는 객체(리스너)가 버튼에게 관심있다는 사실을 알려줄 방법을 제공해준다.

버튼의 ActionEvent를 얻는 방법

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

                                                    ①
public class SimpleGui1B implements ActionListener {
    JButton button;

    public static void main (String[] args) {
       SimpleGui1B gui = new SimpleGui1B();
       gui.go();
     }

    public void go() {
       JFrame frame = new JFrame();
       button = new JButton("click me");

②      button.addActionListener(this);

       frame.getContentPane().add(button);
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(300,300);
       frame.setVisible(true);
    }

③
    public void actionPerformed(ActionEvent event) {
       button.setText("I've been clicked!");
    }
}
```

ActionListener 인터페이스를 구현한다.

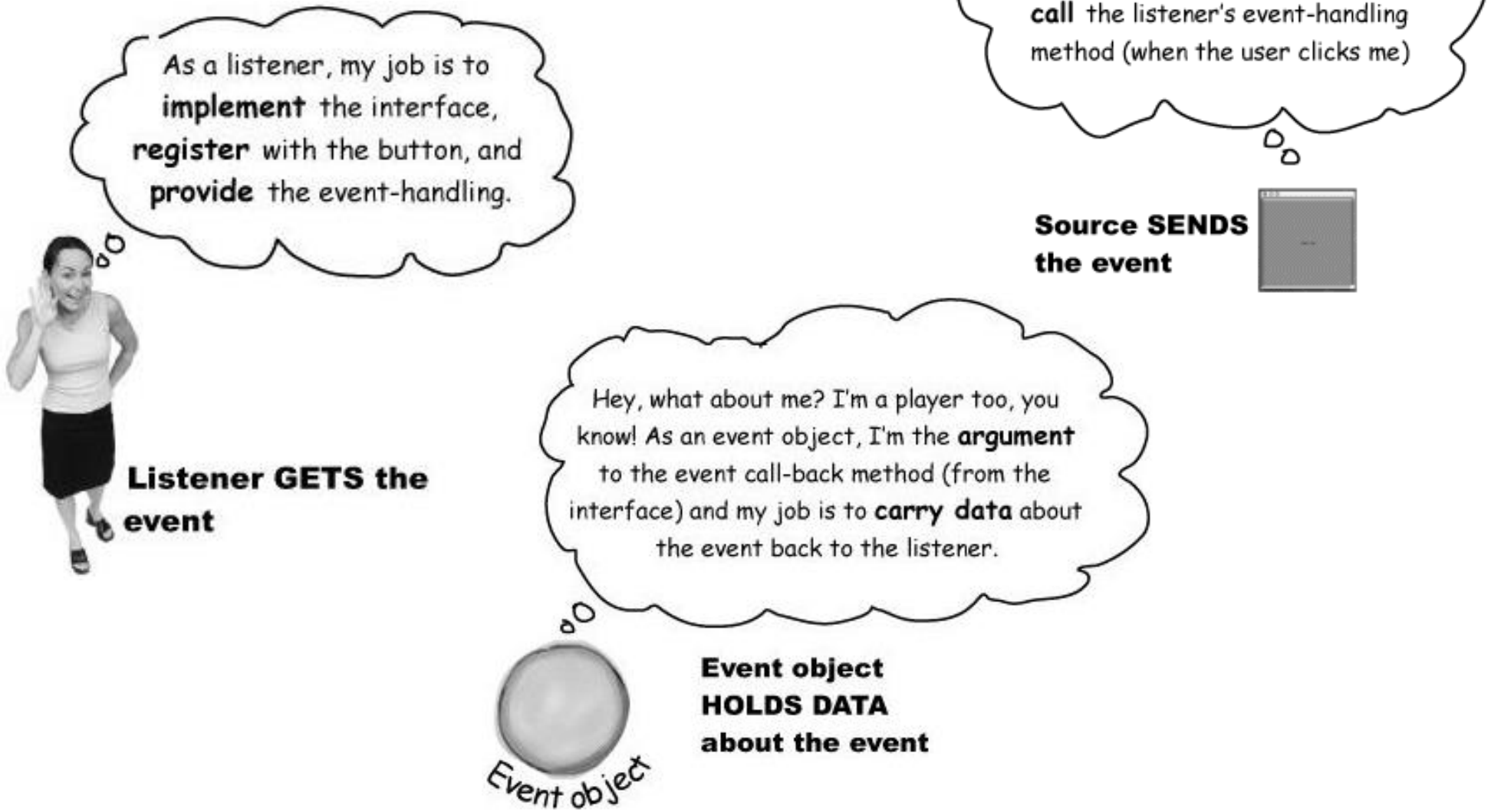
버튼에 등록한다. 즉 '리스너 목록에 나도 끼워줘!' 라고 말한다.

ActionListener 인터페이스의 actionPerformed() 메소드를 구현한다.

버튼에서는 이 메소드를 호출하여 이벤트가 일어났다는 것을 알려준다.

Listeners, Sources, and Events

Your job is to be a good listener.



Getting back to graphics...

GUI에 뭔가를 넣는 세 가지 방법:

1. **위젯을 프레임에 추가**
   버튼, 메뉴, 라디오 버튼 등을 추가한다.

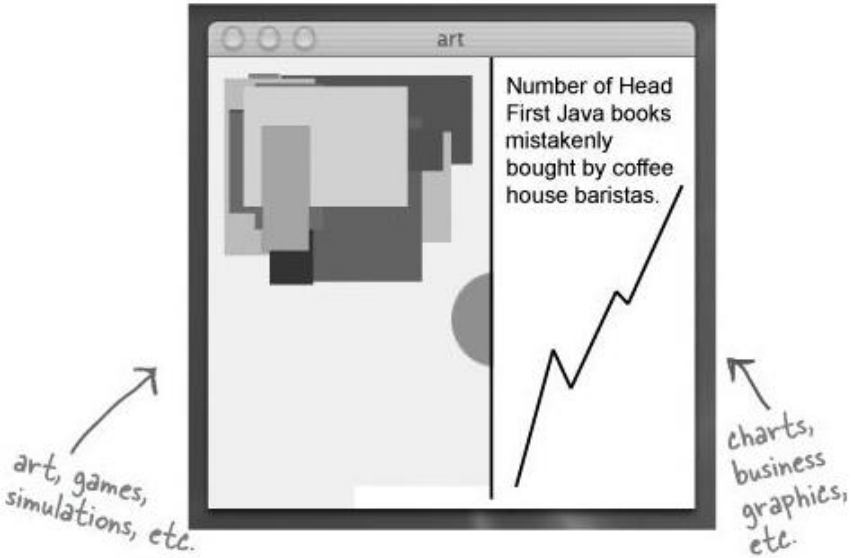
   ```
   frame.getContentPane().add(myButton);
   ```

   javax.swing 패키지에는 12개 이상의 위젯 타입이 있다.

2. **위젯에 2D 그래픽 그리기**
   그래픽 객체를 사용하여 모양을 그린다.

   ```
   graphics.fillOval(70,70,100,100);
   ```

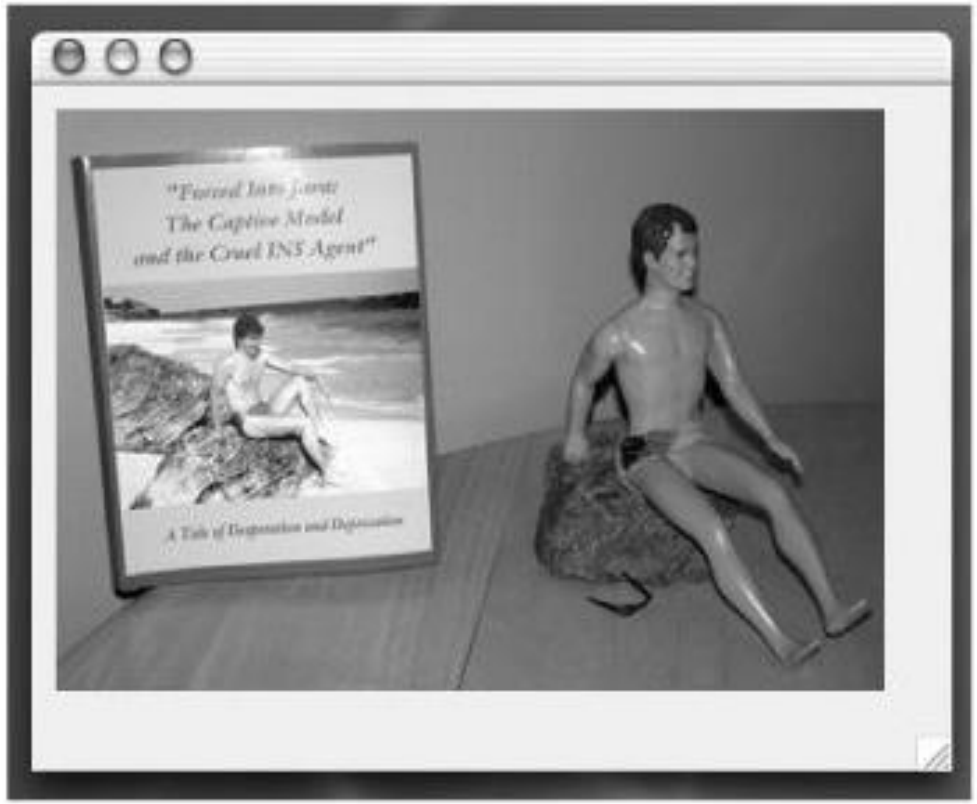
   박스나 원 이상의 것들을 그릴 수 있다; **Java2D API**는 재미있고 정교한 그래픽 메소드들로 가득 차 있다.



GUI에 뭔가를 넣는 세 가지 방법:

3. 위젯에 JPEG 넣기
   위젯에 자신의 이미지를 넣을 수 있다.

```
graphics.drawImage(myPic,10,10,this);
```



Make your own drawing widget

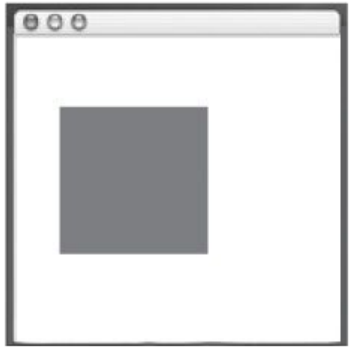
JPanel의 서브 클래스를 작성해, **paintComponent()** 메소드를 오버라이딩 한다.

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

class MyDrawPanel extends JPanel {

      public void paintComponent(Graphics g) {

         g.setColor(Color.orange);

         g.fillRect(20,50,100,100);
      }
}
```

사용자가 이 메소드를 직접 호출하는 일은 절대 없다!!

Graphics 객체

JVM은 화면 갱신이 필요할 때마다 paintcomponent()를 호출하여 그림을 다시 그린다



Fun things to do in paintComponent()

```
public void paintComponent(Graphics g) {

    Image image = new ImageIcon("catzilla.jpg").getImage();

    g.drawImage(image,3,4,this);

}
```

The x,y coordinates for where the picture's top left corner should go. This says "3 pixels from the left edge of the panel and 4 pixels from the top edge of the panel". These numbers are always relative to the widget (in this case your JPanel subclass), not the entire frame.

Paint a randomly-colored circle on a black background

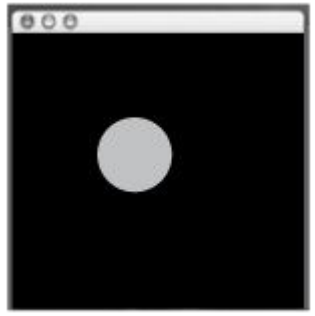
```
public void paintComponent(Graphics g) {

    g.fillRect(0,0,this.getWidth(), this.getHeight());

    int red = (int) (Math.random() * 256);
    int green = (int) (Math.random() * 256);
    int blue = (int) (Math.random() * 256);

    Color randomColor = new Color(red, green, blue);
    g.setColor(randomColor);
    g.fillOval(70,70,100,100);
}
```

fill the entire panel with black (the default color)

The first two args define the (x,y) upper left corner, relative to the panel, for where drawing starts, so 0, 0 means "start 0 pixels from the left edge and 0 pixels from the top edge." The other two args say, "Make the width of this rectangle as wide as the panel (this.width()), and make the height as tall as the panel (this.height)"

You can make a color by passing in 3 ints to represent the RGB values.

start 70 pixels from the left, 70 from the top, make it 100 pixels wide, and 100 pixels tall.



Graphics 레퍼런스 뒤에는 Graphics2D 객체가 있다

paintComponent()의 아규먼트는 Graphics(java.awt.Graphics) 타입으로 선언되어 있다:

```
public void paintComponent(Graphics g){}
```

- ✓ 파라메터 'g'는 IS-A Graphics 객체 관계: Graphics의 서브클래스가 될 수 있다는 것을 의미
- ✓ 사실, 파라메터 'g' 에 의해 참조되는 객체는 실제로는 Graphics2D 클래스의 인스턴스이다.

이 사실이 왜 중요한가?

- ✓ Graphics 레퍼런스로는 할 수 없지만 Graphics2D 레퍼런스로는 할 수 있는 것이 있기 때문
- ✓ Graphics2D 객체는 Graphics 객체보다 더 많은 것을 할 수 있다
- ✓ 사실 이 'g'는 Graphics 레퍼런스 뒤에 숨겨진 Graphics2D 객체

다형성을 기억해 보자:

- ✓ 컴파일러는 객체 타입이 아니라 레퍼런스 타입에 근거해서 어느 메소드를 호출할 수 있는 가를 결정한다.
- ✓ 만일 Animal 레퍼런스 변수에 의해 참조되는 Dog 객체를 가지고 있다면:

```
Animal a = new Dog();
```

- ✓ 다음처럼 bark() 메소드를 호출할 수 없다

```
a.bark();
```

컴파일러는 a를 Animal 타입으로 간주하고 Animal 클래스에 있는 bark() 메소드에 대한 리모컨 버튼을 찾아보지만 거기엔 없다.

그러나 다음처럼 캐스팅을 통해 Dog 객체에 접근할 수는 있다:

```
Dog d = (Dog)a;
d.bark();
```

- ✓ 따라서 Graphics 객체에서도 다음과 같은 규칙을 적용하면 된다:

  "Graphics2D 클래스의 메소드를 사용하고자 할 때 paintComponent 메소드의 파라메터 g를 곧바로 사용할 수는 없지만 새로운 Graphics2D 변수로 캐스팅해서 쓸 수는 있다."

```
Graphics2D g2d = (Graphics2D) g;
```

Note

Methods you can call on a **Graphics reference**:

- drawImage()
- drawLine()
- drawPolygon
- drawRect()
- drawOval()
- fillRect()
- fillRoundRect()
- setColor()

To cast the Graphics2D object to a Graphics2D reference:

Graphics2D g2d = (Graphics2D) g;

Methods you can call on a **Graphics2D reference**:

- fill3DRect()
- draw3DRect()
- rotate()
- scale()
- shear()
- transform()
- setRenderingHints()

(these are not complete method lists, check the API for more)

원을 단색이 아닌 그라디언트로 색칠할 수 있다

```
public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    GradientPaint gradient = new GradientPaint(70,70,Color.blue, 150,150, Color.orange);

    g2d.setPaint(gradient);

    g2d.fillOval(70,70,100,100);
}
```

cast it so we can call something that Graphics2D has but Graphics doesn't

starting Point, starting color, ending Point, ending color

this sets the virtual paint brush to a gradient instead of a solid color

the fillOval() method really means "fill the oval with whatever is loaded on your paintbrush (i.e. the gradient)"

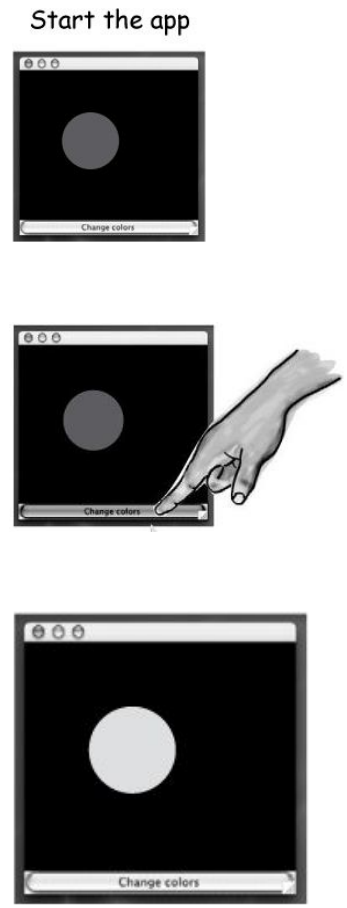
```
public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    int red = (int) (Math.random() * 255);
    int green = (int) (Math.random() * 255);
    int blue = (int) (Math.random() * 255);
    Color startColor = new Color(red, green, blue);

    red = (int) (Math.random() * 255);
    green = (int) (Math.random() * 255);
    blue = (int) (Math.random() * 255);
    Color endColor = new Color(red, green, blue);

    GradientPaint gradient = new GradientPaint(70,70,startColor, 150,150, endColor);
    g2d.setPaint(gradient);
    g2d.fillOval(70,70,100,100);
}
```

this is just like the one above, except it makes random colors for the start and stop colors of the gradient. Try it!

We can get an event. We can paint graphics. But can we paint graphics when we get an event?

사용자가 버튼을 클릭할 때마다 원의 색이 바뀌도록 해 보자:

1. 프레임을 만들어서 그 안에 두 개의 위젯(패널과 버튼)을 넣는다.
   리스너를 생성하고 버튼에 등록한다.
   그 다음은 프레임이 보여지고 사용자가 클릭하기만을 기다린다.
2. 사용자가 버튼을 클릭하면 버튼은 이벤트 객체를 생성하고 리스너의 이벤트 핸들러를 호출한다.
3. 이벤트 핸들러는 프레임의 **repaint()**를 호출한다.
   시스템은 드로잉 패널의 **paintComponent()**를 호출한다.
4. **paintComponent()**가 다시 실행되면서 무작위로 선택된 새로운 색으로 다시 칠해진다.



GUI 레이아웃: 프레임에 두 개 이상 위젯 넣는 방법

```
frame.getContentPane().add(button);
```

This isn't really the way you're supposed to do it (the one-arg add method).

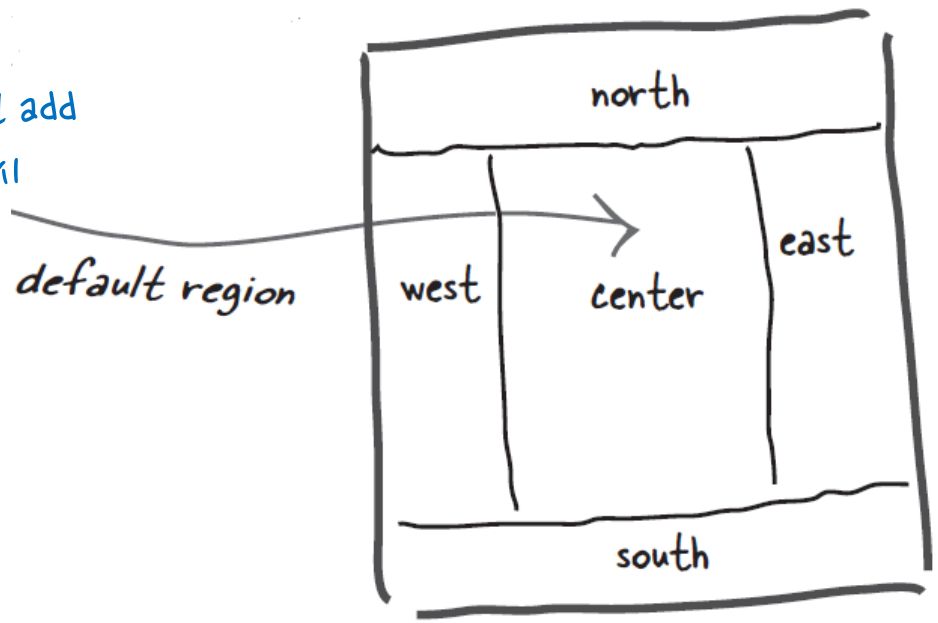
원래는 이렇게 하면 안된다!

프레임의 기본 틀에 위젯을 추가할 때는 항상 이렇게 두 개의 인자를 갖는 add 메소드를 사용하는 것이 바람직!!

```
frame.getContentPane().add(BorderLayout.CENTER, button);
```

지역(상수로 지정)과 그 지역에 추가할 위젯, 이렇게 두 개의 아규먼트를 받아들이는 add 메소드를 호출해야 한다.

인자 하나인 add 메소드 사용시



실습과제 12-1

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


public class SimpleGui3C implements ActionListener {

    JFrame frame;

    public static void main (String[] args) {
       SimpleGui3C gui = new SimpleGui3C();
       gui.go();
    }

    public void go() {
       frame = new JFrame();
       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       JButton button = new JButton("Change colors");
       button.addActionListener(this);

       MyDrawPanel drawPanel = new MyDrawPanel();

       frame.getContentPane().add(BorderLayout.SOUTH, button);
       frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
       frame.setSize(300,300);
       frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}
```

The custom drawing panel (instance of MyDrawPanel) is in the CENTER region of the frame.

Button is in the SOUTH region of the frame

Add the listener (this) to the button.

Add the two widgets (button and drawing panel) to the two regions of the frame.

When the user clicks, tell the frame to repaint() itself. That means paintComponent() is called on every widget in the frame!

```
class MyDrawPanel extends JPanel {

    public void paintComponent(Graphics g) {
      // Code to fill the oval with a random color
      // See page 367 for the code
    }

}
```

```
public void paintComponent(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;

    int red = (int) (Math.random() * 255);
    int green = (int) (Math.random() * 255);
    int blue = (int) (Math.random() * 255);
    Color startColor = new Color(red, green, blue);

    red = (int) (Math.random() * 255);
    green = (int) (Math.random() * 255);
    blue = (int) (Math.random() * 255);
    Color endColor = new Color(red, green, blue);

    GradientPaint gradient = new GradientPaint(70,70,startColor, 150,150, endColor);
    g2d.setPaint(gradient);
    g2d.fillOval(70,70,100,100);
}
```

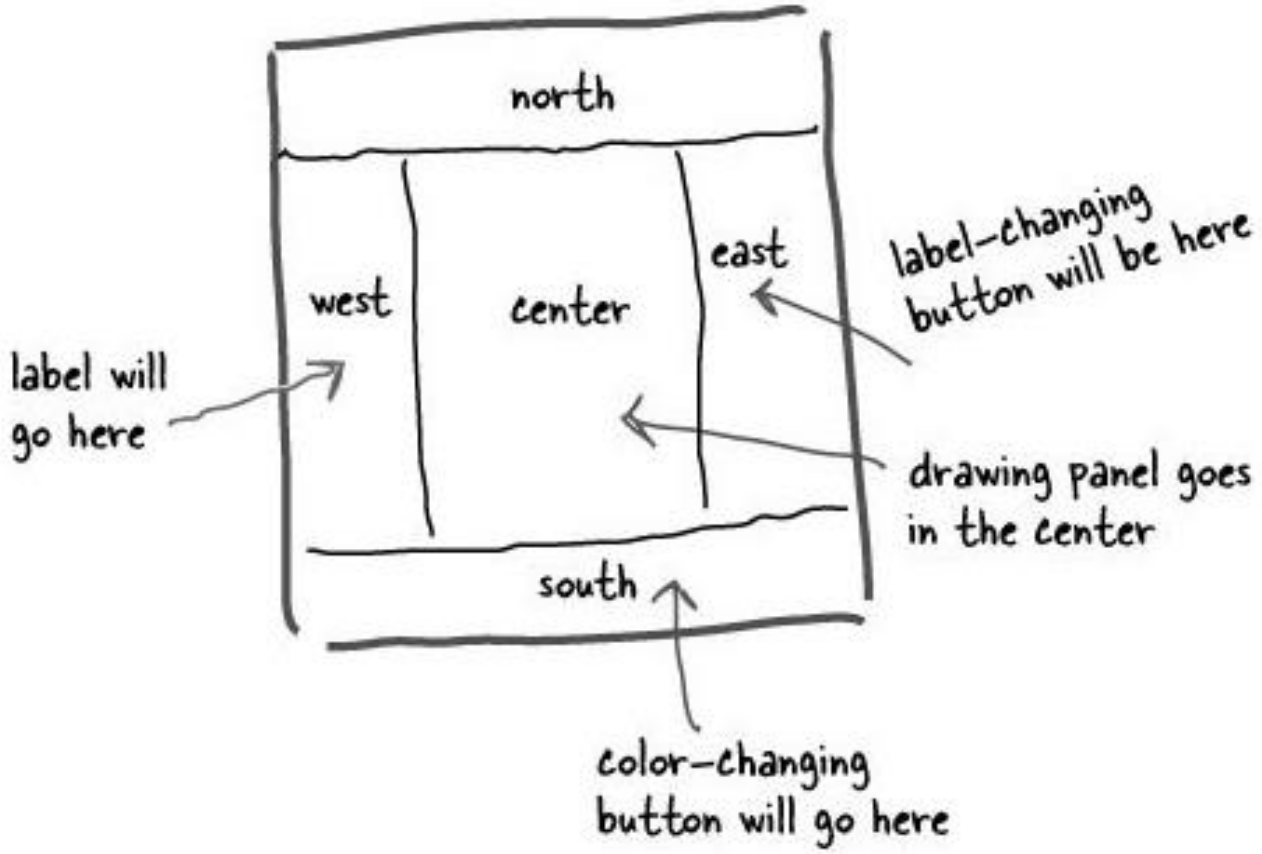
this is just like the one above, except it makes random colors for the start and stop colors of the gradient. Try it!

Let’s try it with TWO buttons

남쪽 지역 버튼은 지금까지처럼 단순히 **repaint()**를 호출한다.

동쪽 지역의 버튼은 레이블의 텍스트를 바꿀 것이다.

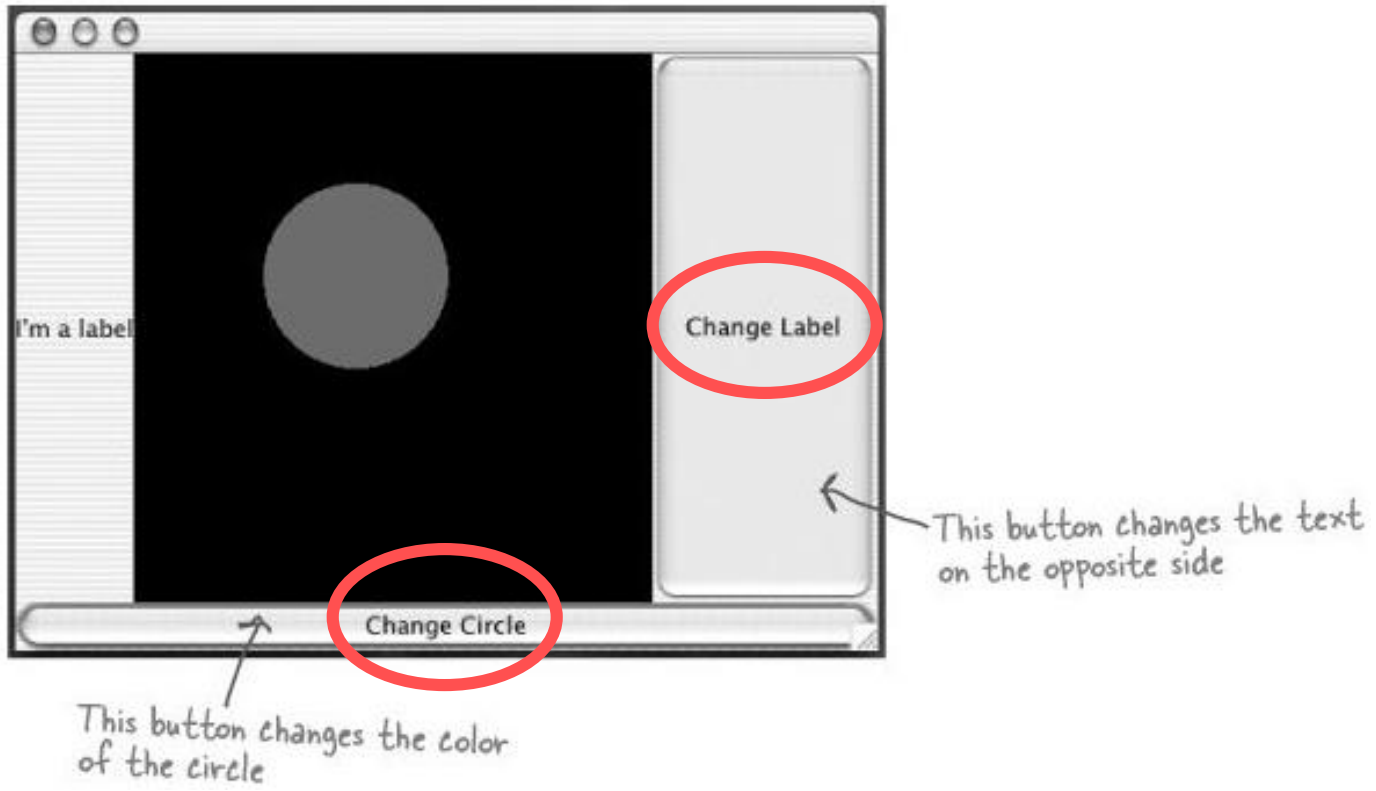
So now we need **FOUR** widgets



And we need to get **TWO** events

그게 가능한가?

actionPerformed() 메소드는 한 개 뿐인데 어떻게 두 개의 이벤트를 처리할 수 있나?

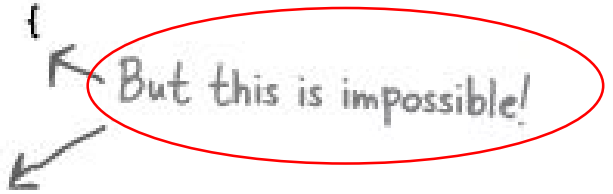


두 개의 다른 버튼이 뭔가 다르게 동작하기를 원할 때 각 버튼에 대한 액션 이벤트는 어떻게 얻을 수 있나?

option one

Implement two actionPerformed() methods

```
class MyGui implements ActionListener {
    // lots of code here and then:

    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }

    public void actionPerformed(ActionEvent event) {
        label.setText("That hurt!");
    }
}
```



문제점: 자바 클래스에서 같은 메소드를 두 번 구현할 수 없다!!

option two

Register the same listener with both buttons.

```
class MyGui implements ActionListener {
    // declare a bunch of instance variables here

    public void go() {
       // build gui
       colorButton = new JButton();
       labelButton = new JButton();
       colorButton.addActionListener(this);
       labelButton.addActionListener(this);
       // more gui code here ...
    }

   public void actionPerformed(ActionEvent event) {
        if (event.getSource() == colorButton) {
            frame.repaint();
        } else {
           label.setText("That hurt!");
        }
    }
}
```

Register the same listener with both buttons

Query the event object to find out which button actually fired it, and use that to decide what to do.

문제점: 돌아가기는 하나 객체지향적이지 못하다!!

option three

Create two separate ActionListener classes

```
class MyGui  {
   JFrame frame;
   JLabel label;
   void gui() {
      // code to instantiate the two listeners and register one
      // with the color button and the other with the label button
   }
} // close class
```

```
class ColorButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}
```

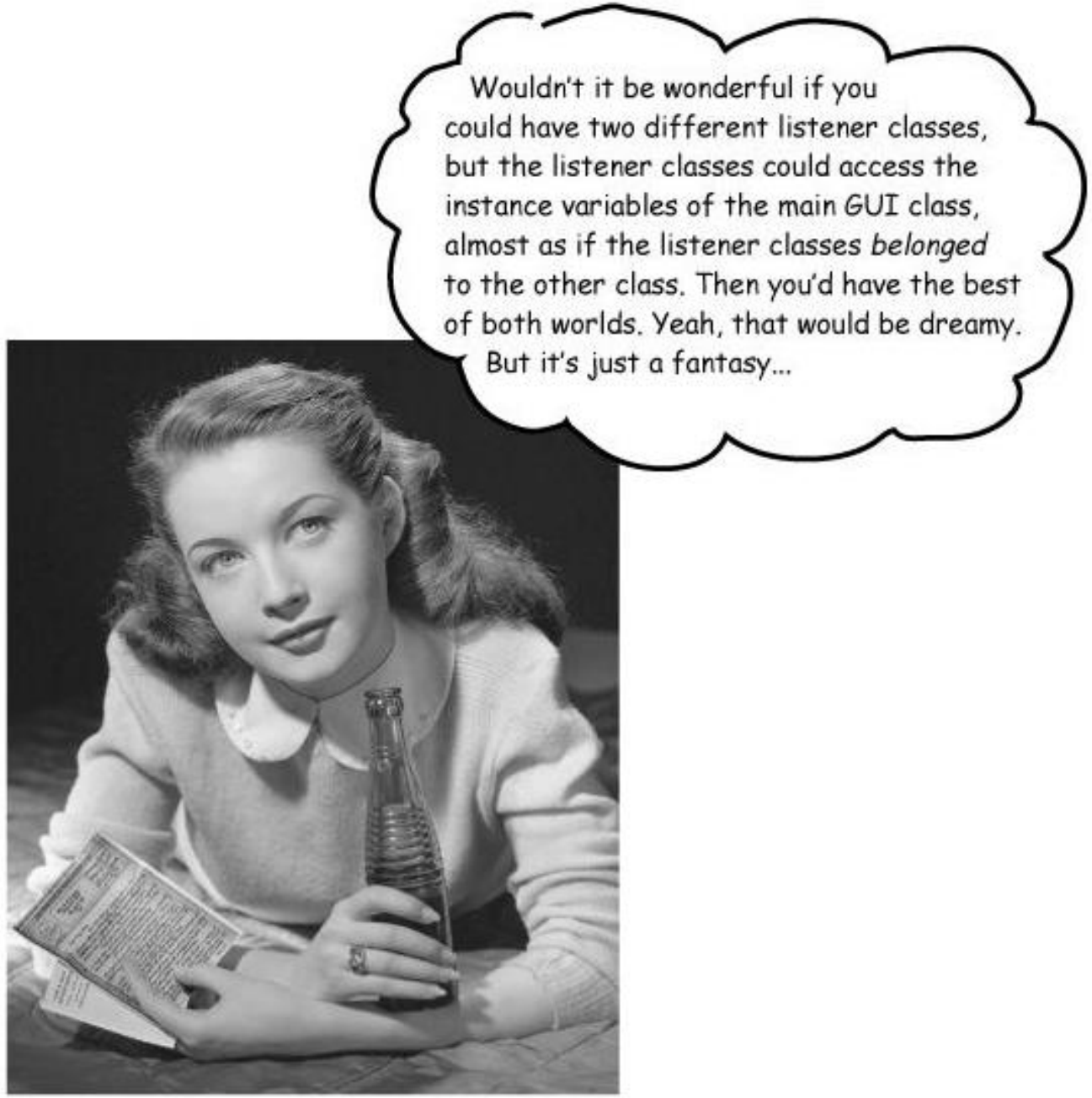
Won't work! This class doesn't have a reference to the 'frame' variable of the MyGui class

---

```
class LabelButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        label.setText("That hurt!");
    }
}
```

Problem! This class has no reference to the variable 'label'

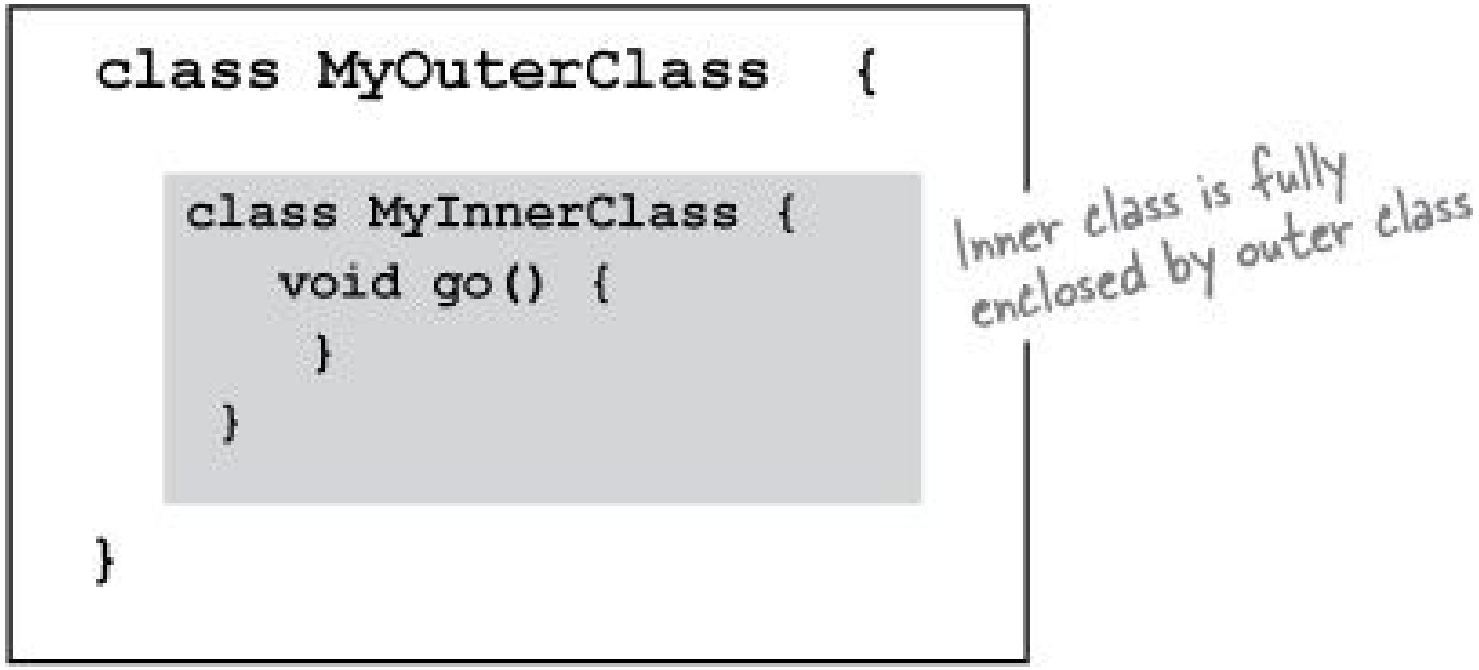
문제점: 두 클래스에서 'frame' 과 'label'이라는 변수에 접근할 수가 없다!!


Wouldn't it be wonderful if you could have two different listener classes, but the listener classes could access the instance variables of the main GUI class, almost as if the listener classes *belonged* to the other class. Then you'd have the best of both worlds. Yeah, that would be dreamy.
But it's just a fantasy...

해결책은 내부 클래스!

클래스 안에 또 다른 클래스를 가질 수 있다.

단순 내부 클래스(inner class):



내부 클래스에서는 외부 클래스의 모든 메소드와 변수를 사용할 수 있다.

심지어 외부 클래스에서 private로 선언된 변수와 메소드에도 접근할 수 있다.

마치 자신의 내부 클래스에서 선언한 것처럼.

외부 클래스 변수를 사용한 내부 클래스:

```
class MyOuterClass  {

   private int x;

  class MyInnerClass {
     void go() {
        x = 42;
     }
  } // close inner class

} // close outer class
```

use 'x' as if it were a variable of the inner class!

내부 클래스 인스턴스는 외부 클래스 인스턴스와 서로 묶여야 한다

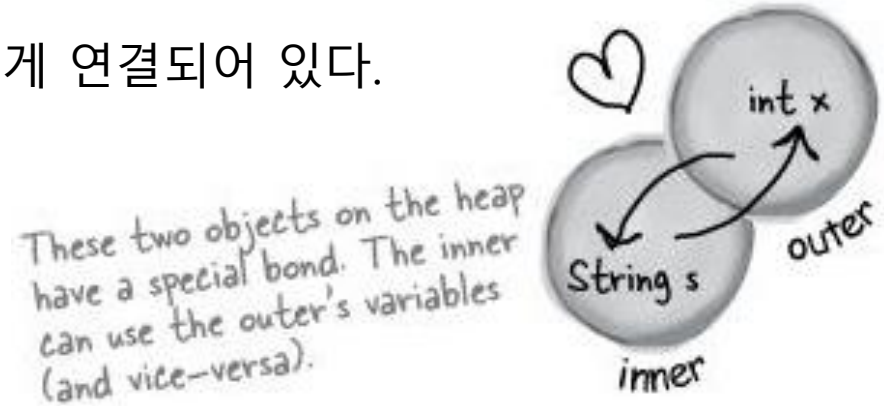
NOTE 내부 객체는 외부 객체와 특별한 결합을 공유한다.

내부 객체는 힙에 있는 특정 외부 객체와 엮여 있어야만 한다.

1. 외부 클래스의 인스턴스를 만든다.

2. 외부 클래스의 인스턴스를 사용하여 내부 클래스의 인스턴스를 만든다.

3. 외부와 내부 객체가 이제 밀접하게 연결되어 있다.



내부 클래스의 인스턴스를 만드는 방법

외부 클래스에서 자신의 내부 클래스의 인스턴스를 만들 수 있다.

다른 클래스에서와 똑같은 방식으로… **new MyInner()**

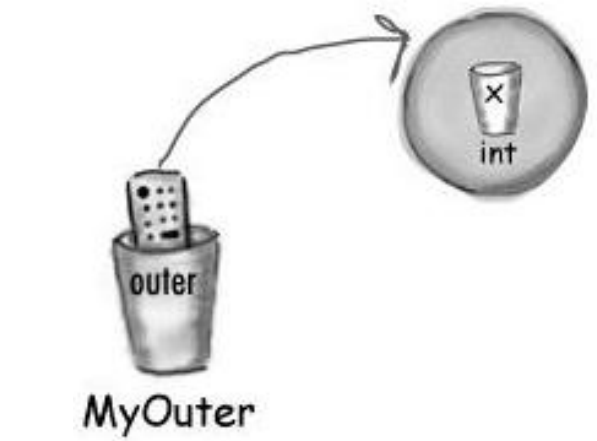
```
class MyOuter  {

   private int x;

   MyInner inner = new MyInner();

   public void doStuff() {
      inner.go();
   }

   class MyInner {
      void go() {
         x = 42;
      }
   } // close inner class

} // close outer class
```

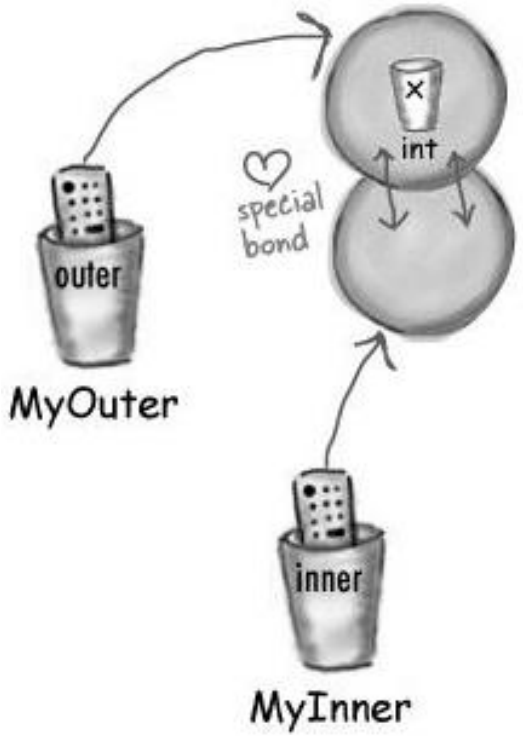
The outer class has a private instance variable 'x'

Make an instance of the inner class

call a method on the inner class

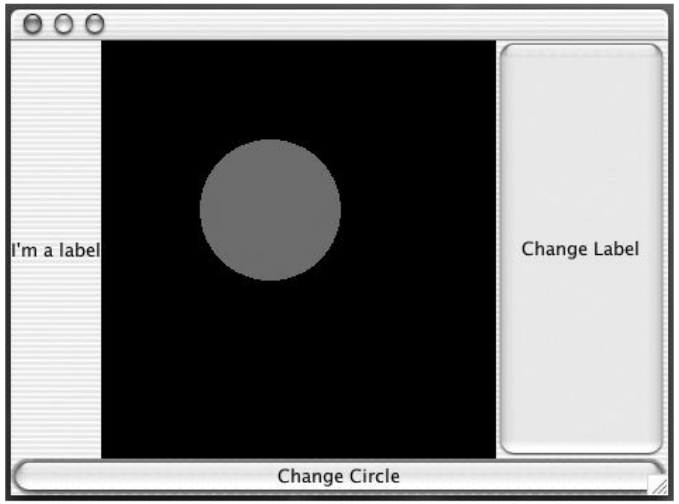
The method in the inner class uses the outer class instance variable 'x', as if 'x' belonged to the inner class.





실습과제 12-2

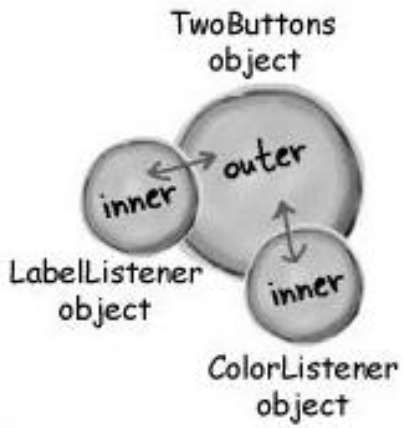
Now we can get the two-button code working




```
public class TwoButtons {          <- the main GUI class doesn't implement ActionListener now

    JFrame frame;
    JLabel label;

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

    public void go() {
        frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JButton labelButton = new JButton("Change Label");
        labelButton.addActionListener(new LabelListener());

        JButton colorButton = new JButton("Change Circle");
        colorButton.addActionListener(new ColorListener());
```

instead of passing (this) to the button's listener registration method, pass a new instance of the appropriate listener class.

```
    label = new JLabel("I'm a label");
    MyDrawPanel drawPanel = new MyDrawPanel();

    frame.getContentPane().add(BorderLayout.SOUTH, colorButton);
    frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
    frame.getContentPane().add(BorderLayout.EAST, labelButton);
    frame.getContentPane().add(BorderLayout.WEST, label);

    frame.setSize(300,300);
    frame.setVisible(true);
}

class LabelListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        label.setText("Ouch!");
    }
} // close inner class

class ColorListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}  // close inner class

}
```



Now we get to have TWO ActionListeners in a single class!

inner class knows about 'label'

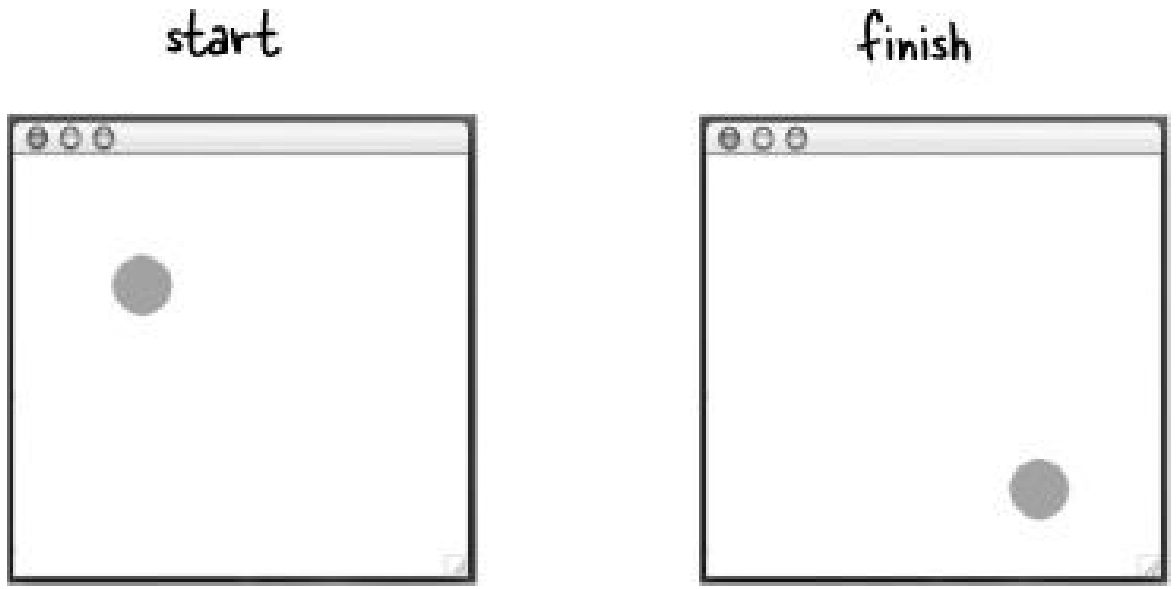
the inner class gets to use the 'frame' instance variable, without having an explicit reference to the outer class object.

Using an inner class for animation

이벤트 리스너를 만들 때 내부 클래스를 이용하면 매우 유용하다.

- ✓ 이벤트 핸들링 메소드를 한 번 이상 구현할 수 있기 때문에
- ✓ 이제 외부 클래스와 내부 클래스가 서로 다른 상속 트리에 있을 때 얼마나 유용한지 살펴보자

목적: 원(circle)이 화면 왼쪽 위에서 오른쪽 아래로 움직이는 단순한 애니메이션을 만드는 것



How simple animation works

1. 특정 x, y 좌표에 객체를 그린다.

```
g.fillOval(20,50,100,100);
```

20 pixels from the left, 50 pixels from the top

2. 다른 x, y 좌표에 객체를 다시 그린다.

```
g.fillOval(25,55,100,100);
```

25 pixels from the left, 55 pixels from the top

(the object moved a little down and to the right)

3. 애니메이션이 진행되는 동안 x, y 값을 바꿔가면서 위의 단계를 반복한다.

```
class MyDrawPanel extends JPanel {
    public void paintComponent(Graphics g) {
        g.setColor(Color.orange);
        g.fillOval(x,y,100,100);
    }
}
```

each time paintComponent() is called, the oval gets painted at a different location

The complete simple animation code

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


public class SimpleAnimation {

    int x = 70;
    int y = 70;

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

    public void go() {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        MyDrawPanel drawPanel = new MyDrawPanel();

        frame.getContentPane().add(drawPanel);
        frame.setSize(300,300);
        frame.setVisible(true);
```

make two instance variables in the main GUI class, for the x and y coordinates of the circle.

Nothing new here. Make the widgets and put them in the frame.

This is where the action is!

```
for (int i = 0; i < 130; i++) {

   x++;
   y++;

   drawPanel.repaint();

   try {
     Thread.sleep(50);
   } catch(Exception ex) { }
}
```

repeat this 130 times

increment the x and y coordinates

tell the panel to repaint itself (so we can see the circle in the new location)

Slow it down a little (otherwise it will move so quickly you won't SEE it move). Don't worry, you weren't supposed to already know this. We'll get to threads in chapter 15.

```
}// close go() method
```

내부 클래스

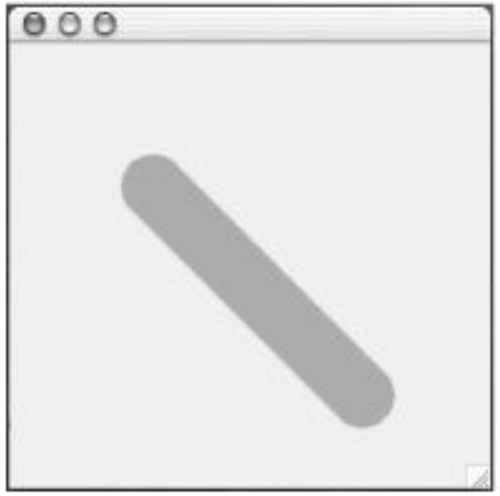
Now it's an inner class.

```
class MyDrawPanel extends JPanel {

   public void paintComponent(Graphics g) {
      g.setColor(Color.green);
      g.fillOval(x,y,40,40);

   }
  } // close inner class
} // close outer class
```

Use the continually-updated x and y coordinates of the outer class.

실습과제 12-3 Fix it!

We forgot to erase what was already there! So we got trails.



```
public void paintComponent(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0,0,this.getWidth(), this.getHeight());

    g.setColor(Color.green);
    g.fillOval(x,y,40,40);

}
```

getWidth() and getHeight() are methods inherited from JPanel.


Cyber BeatBox
Hand Clap
High Tom
Hi Bongo
Maracas
Whistle
Low Conga
Cowbell
Vibraslap
Low-mid Tom
High Agogo
Open Hi Conga

코드 키친

Listening for a non-GUI event

음악의 박자에 맞춰서 무작위로 생성된 그래픽을 표시해주는 프로그램을 만들어보자.

지금까지는 GUI 이벤트만 받아서 사용했지만 이제 특정 종류의 MIDI 이벤트를 받아와야 한다.

음악 비트를 listen하기 위한 가장 쉬운 방법은 실제 **MIDI 이벤트를 등록하여 listen하는 것**이다.

⇒ 그래서 시퀀서가 이벤트를 얻을 때마다 코드에서도 이벤트를 얻어서 그래픽을 그리면 될 것이다.

그러나 문제가 있다...

사실은 **버그**인데, 현재 우리가 만들고 있는 **MIDI 이벤트(NOTE ON 이벤트 등)를 listen할 수가 없다!!**

⇒ 그런데 MIDI 이벤트 중에는 우리가 listen할 수 있는 **ControllerEvent라는 이벤트**가 있다.

ControllerEvent 이벤트 사용

그래서 약간 우회하는 방법을 사용해야 한다

Listen할 수 있는 **ControllerEvent**라는 이벤트를 이용한다

즉, 모든 NOTE ON 이벤트에 대해 대응되는 **ControllerEvent**가 점화되도록 만든다.

ControllerEvent 이벤트를 어떻게 동시에 점화시킬 수 있나?

그냥 다른 이벤트처럼 트랙에 추가하기만 하면 된다:

BEAT 1 - NOTE ON, CONTROLLER EVENT

BEAT 2 - NOTE OFF

BEAT 3 - NOTE ON, CONTROLLER EVENT

BEAT 4 - NOTE OFF

and so on.

음악 아트 프로그램이 수행해야 하는 것들:

1. 피아노(임의의 악기) 상에서 랜덤 음표를 연주하도록 하기 위해 일련의 메시지&이벤트를 만든다.
2. 이벤트에 대한 리스너를 등록한다.
3. 시퀀서 연주를 시작한다.
4. 리스너의 이벤트 핸들러 메소드가 호출될 때마다 드로잉 패널에 랜덤 사각형을 그린 후, **repaint()** 메소드를 호출한다.

아래 세 가지 버전으로 구축해보자:

버전 1: MIDI 이벤트를 만들고 추가하는 것을 단순화시킨다.

버전 2: 그래픽 없이 이벤트를 만들고 listen한다. 각 비트마다 명령행에서 메시지를 프린트한다.

버전 3: 버전 2에 그래픽을 추가한다.

An easier way to make messages / events

NOTE ON 이벤트를 추가하기 위한 네 줄,
NOTE OFF 이벤트를 추가하기 위한 네 줄:

```
ShortMessage a = new ShortMessage();
a.setMessage(144, 1, note, 100);
MidiEvent noteOn = new MidiEvent(a, 1);
track.add(noteOn);

ShortMessage b = new ShortMessage();
b.setMessage(128, 1, note, 100);
MidiEvent noteOff = new MidiEvent(b, 16);
track.add(noteOff);
```

각 이벤트마다 발생해야만 하는 것들:

1. Make a message instance
   ShortMessage first = new ShortMessage();
2. Call setMessage() with the instructions
   first.setMessage(192, 1, instrument, 0)
3. Make a MidiEvent instance for the message
   MidiEvent noteOn = new MidiEvent(first, 1);
4. Add the event to the track
   track.add(noteOn);

메시지를 만들고 MidiEvent를 반환해 주는 static utility 메소드를 만들자.

```
public static MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
        ShortMessage a = new ShortMessage();
        a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);

    }catch(Exception e) { }
    return event;
}
```

the four arguments for the message

The event 'tick' for WHEN this message should happen

whoo! A method with five parameters.

make the message and the event, using the method parameters

return the event (a MidiEvent all loaded up with the message)

실습과제 12-4 makeEvent() 메소드 활용 방법

```
import javax.sound.midi.*;
public class MiniMusicPlayer1 {

   public static void main(String[] args) {

      try {

         Sequencer sequencer = MidiSystem.getSequencer();
         sequencer.open();


         Sequence seq = new Sequence(Sequence.PPQ, 4);
         Track track = seq.createTrack();

      for (int i = 5; i < 61; i+= 4) {

         track.add(makeEvent(144,1,i,100,i));
         track.add(makeEvent(128,1,i,100,i + 2));

       } // end loop

          sequencer.setSequence(seq);
          sequencer.setTempoInBPM(220);
          sequencer.start();
      } catch (Exception ex) {ex.printStackTrace();}
  } // close main


   public static MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
          MidiEvent event = null;
          try {
            ShortMessage a = new ShortMessage();
            a.setMessage(comd, chan, one, two);
           event = new MidiEvent(a, tick);

          }catch(Exception e) { }
         return event;
       }
} // close class
```

don't forget the import

make (and open) a sequencer

make a sequence and a track

make a bunch of events to make the notes keep going up (from piano note 5 to piano note 61)

call our new makeEvent() method to make the message and event, then add the result (the MidiEvent returned from makeEvent()) to the track. These are NOTE ON (144) and NOTE OFF (128) pairs

start it running

실습과제 12-5 ControllerEvent 등록/listen 방법

```
import javax.sound.midi.*;
public class MiniMusicPlayer2 implements ControllerEventListener {

    public static void main(String[] args) {
        MiniMusicPlayer2 mini = new MiniMusicPlayer2();
        mini.go();
    }
  public void go() {

     try {
        Sequencer sequencer = MidiSystem.getSequencer();
        sequencer.open();

        int[] eventsIWant = {127};
        sequencer.addControllerEventListener(this, eventsIWant);

        Sequence seq = new Sequence(Sequence.PPQ, 4);
        Track track = seq.createTrack();

        for (int i = 5; i < 60; i+= 4) {
           track.add(makeEvent(144,1,i,100,i));

           track.add(makeEvent(176,1,127,0,i));

           track.add(makeEvent(128,1,i,100,i + 2));
        } // end loop

        sequencer.setSequence(seq);
        sequencer.setTempoInBPM(220);
        sequencer.start();
     } catch (Exception ex) {ex.printStackTrace();}
  } // close
```

We need to listen for ControllerEvents, so we implement the listener interface

이 이벤트는 아무 것도 하지 않는다.
단, Note가 연주될 때마다 이벤트를 얻을 수 있게 해준다.

176 - controllerEvent

we insert
is the event
rgument for
event number #127. This event will do NOTHING! We put it in JUST so that we can get an event each time a note is played. In other words, its sole purpose is so that something will fire that WE can listen for (we can't listen for NOTE ON/OFF events). Note that we're making this event happen at the SAME tick as the NOTE ON. So when the NOTE ON event happens, we'll know about it because OUR event will fire at the same time.

```
public void controlChange(ShortMessage event) {
     System.out.println("la");
}
```

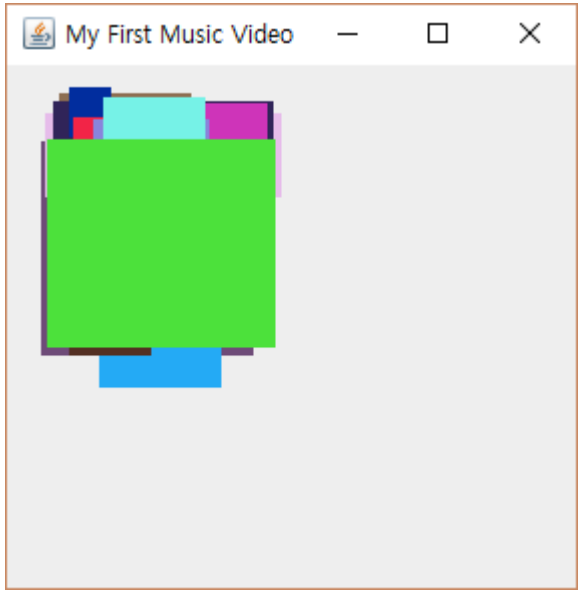
The event handler method (from the Controller-Event listener interface). Each time we get the event, we'll print "la" to the command-line.

```
public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
       MidiEvent event = null;
       try {
         ShortMessage a = new ShortMessage();
         a.setMessage(comd, chan, one, two);
         event = new MidiEvent(a, tick);

       }catch(Exception e) { }
      return event;
    }
} // close class
```

Code that's different from the previous version is highlighted in gray. (and we're not running it all within main() this time)

실습과제 12-6 음악에 맞춰서 그래픽 표시

```
import javax.sound.midi.*;
import java.io.*;
import javax.swing.*;
import java.awt.*;

public class MiniMusicPlayer3 {

     static JFrame f = new JFrame("My First Music Video");
     static MyDrawPanel ml;

     public static void main(String[] args) {
           MiniMusicPlayer3 mini = new MiniMusicPlayer3();
           mini.go();
     } // close method

     public void setUpGui() {
        ml = new MyDrawPanel();
        f.setContentPane(ml);
        f.setBounds(30,30, 300,300);
        f.setVisible(true);
    } // close method

    public void go() {
       setUpGui();

       try {

          Sequencer sequencer = MidiSystem.getSequencer();
          sequencer.open();
          sequencer.addControllerEventListener(ml, new int[] {127});
          Sequence seq = new Sequence(Sequence.PPQ, 4);
          Track track = seq.createTrack();
```



```
      int r = 0;
      for (int i = 0; i < 60; i+= 4) {

         r = (int) ((Math.random() * 50) + 1);
         track.add(makeEvent(144,1,r,100,i));
         track.add(makeEvent(176,1,127,0,i));
         track.add(makeEvent(128,1,r,100,i + 2));
      } // end loop

      sequencer.setSequence(seq);
      sequencer.start();
      sequencer.setTempoInBPM(120);
   } catch (Exception ex) {ex.printStackTrace();}
} // close method
public MidiEvent makeEvent(int comd, int chan, int one, int two, int tick) {
    MidiEvent event = null;
    try {
       ShortMessage a = new ShortMessage();
       a.setMessage(comd, chan, one, two);
        event = new MidiEvent(a, tick);

    }catch(Exception e) { }
    return event;
} // close method
```

The drawing panel inner class:

```
class MyDrawPanel extends JPanel implements ControllerEventListener {
     boolean msg = false;

     public void controlChange(ShortMessage event) {
        msg = true;
        repaint();
     }

     public void paintComponent(Graphics g) {
      if (msg) {

        int r = (int) (Math.random() * 250);
        int gr = (int) (Math.random() * 250);
        int b = (int) (Math.random() * 250);

        g.setColor(new Color(r,gr,b));

        int ht = (int) ((Math.random() * 120) + 10);
        int width = (int) ((Math.random() * 120) + 10);

        int x = (int) ((Math.random() * 40) + 10);
        int y = (int) ((Math.random() * 40) + 10);

        g.fillRect(x,y,ht, width);
        msg = false;

      } // close if
    } // close method
  } // close inner class

} // close class
```

실습과제 12-7

BE the compiler

The Java file on this page represents a complete source file. Your job is to play compiler and determine whether this file will compile. If it won't compile, how would you fix it, and if it does compile, what would it do?

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

class InnerButton {

  JFrame frame;
  JButton b;

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

  public void go() {
    frame = new JFrame();
    frame.setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);

    b = new JButton("A");
    b.addActionListener();

    frame.getContentPane().add(
                 BorderLayout.SOUTH, b);
    frame.setSize(200,100);
    frame.setVisible(true);
  }

  class BListener extends ActionListener {
    public void actionPerformed(ActionEvent e) {
      if (b.getText().equals("A")) {
        b.setText("B");
      } else {
        b.setText("A");
      }
    }
  }
}
```

실습과제 12-8 Pool Puzzle

```
import javax.swing.*;
import java.awt.*;
public class Animate {
    int x = 1;
    int y = 1;
    public static void main (String[] args) {
       Animate gui = new Animate ();
       gui.go();
   }
   public void go() {
       JFrame _________ = new JFrame();
       frame.setDefaultCloseOperation(
                    JFrame.EXIT_ON_CLOSE);
       ____________________________________;
       _________.getContentPane().add(drawP);
       ____________________________;
       _________.setVisible(true);
       for (int i=0; i<124; _______________) {
       ____________________;
       ____________________;
       try {
          Thread.sleep(50);
       } catch(Exception ex) { }
       }
   }
   class MyDrawP extends JPanel {
       public void paintComponent (Graphics
                                   _________) {
          ____________________________________;
          ____________________________________;
          ____________________________________;
          ____________________________________;
       }
   }
}
```