Using Swing: Work on Your Swing

Samkeun Kim <skim@hknu.ac.kr>

http://cyber.hankyong.ac.kr

스윙 컴포넌트 (Swing components)

컴포넌트

⇒ 우리가 지금까지 위젯이라고 불러왔던 것의 더 정확한 표현

⇒ GUI에 넣었던 것들

⇒ 사용자가 볼 수 있고 사용자와 상호작용하던 것들

⇒ 텍스트 필드, 버튼, 스크롤 가능한 리스트, 라디오 버튼 등이 모두 컴포넌트!

✓ 이들 모두 javax.swing.Jcomponent를 확장한다.

Components can be nested

스윙에서는 거의 모든 컴포넌트에 다른 컴포넌트를 집어넣을 수 있다.

그러나 버튼이나 대화형 리스트 같은 **대화형 컴포넌트**를 프레임이나 패널과 같은 **배경 컴포넌트**에 집어넣는 경우가 거의 대부분이다.

Jframe을 제외하고는 대화형 컴포넌트와 배경 컴포넌트 사이의 차이는 거의 없다.

예를 들어, **JPanel**의 경우 보통 여러 컴포넌트를 그룹지워 주는 배경 컴포넌트로 사용되지만 또한 대화형 컴포넌트로도 사용될 수 있다.

즉, 다른 컴포넌트와 마찬가지로 마우스 클릭 등을 포함하는 JPanel의 이벤트를 등록할 수 있다.

Components can be nested

Four steps to making a GUI (review)

1. Make a window (a JFrame)
```
JFrame frame = new JFrame();
```
2. Make a component (button, text field, etc.)
```
JButton button = new JButton("click me");
```
3. Add the component to the frame
```
frame.getContentPane().add(BorderLayout.EAST, button);
```
4. Display it (give it a size and make it visible)
```
frame.setSize(300,300);
frame.setVisible(true);
```

Components can be nested

Put **interactive** components:

Into **background** components:

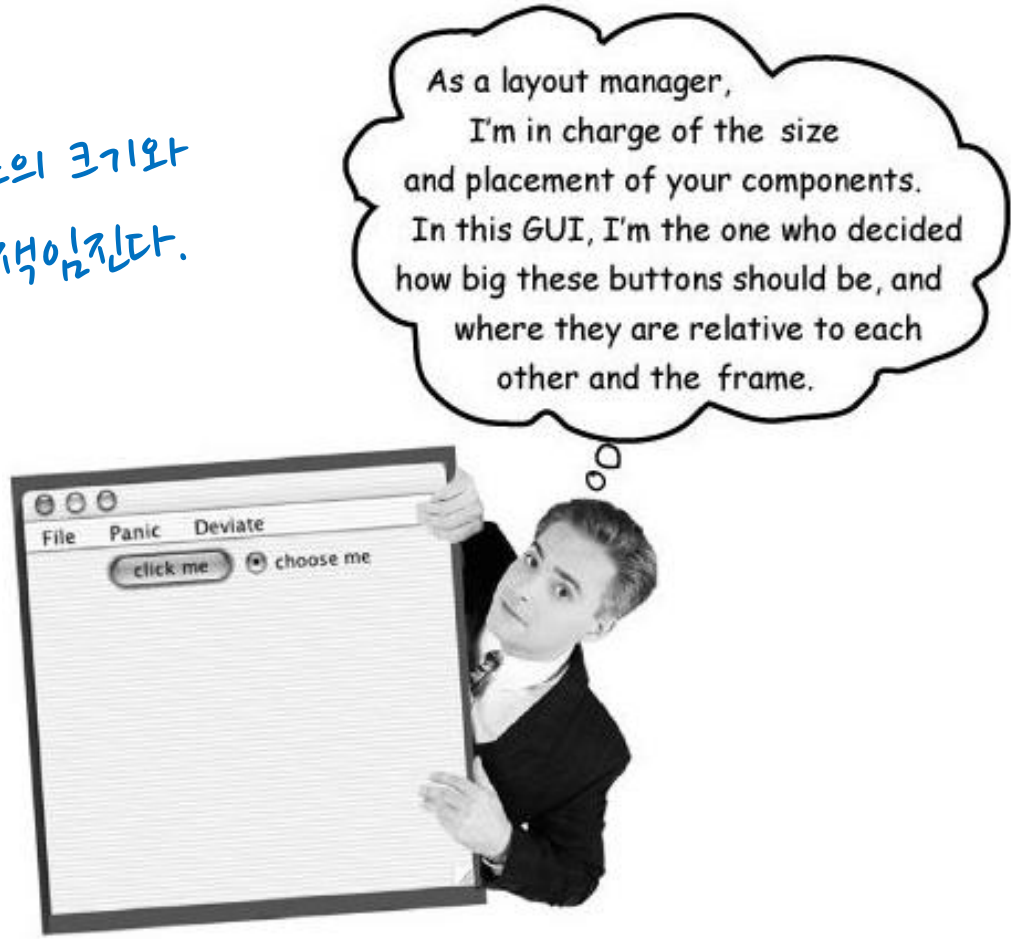
Layout Managers

레이아웃 매니저: 특정 컴포넌트(거의 항상 배경 컴포넌트)와 연관된 Java 객체

⇒ 자신의 레이아웃 매니저에 포함된 컴포넌트를 제어한다

즉, 프레임에 패널이 들어있고
그 패널에 버튼이 들어있다면
패널의 레이아웃 매니저는 버튼의
크기와 위치를 제어하고,
프레임의 레이아웃 매니저는
패널의 크기와 위치를 제어한다.

컴포넌트의 크기와
위치를 책임진다.

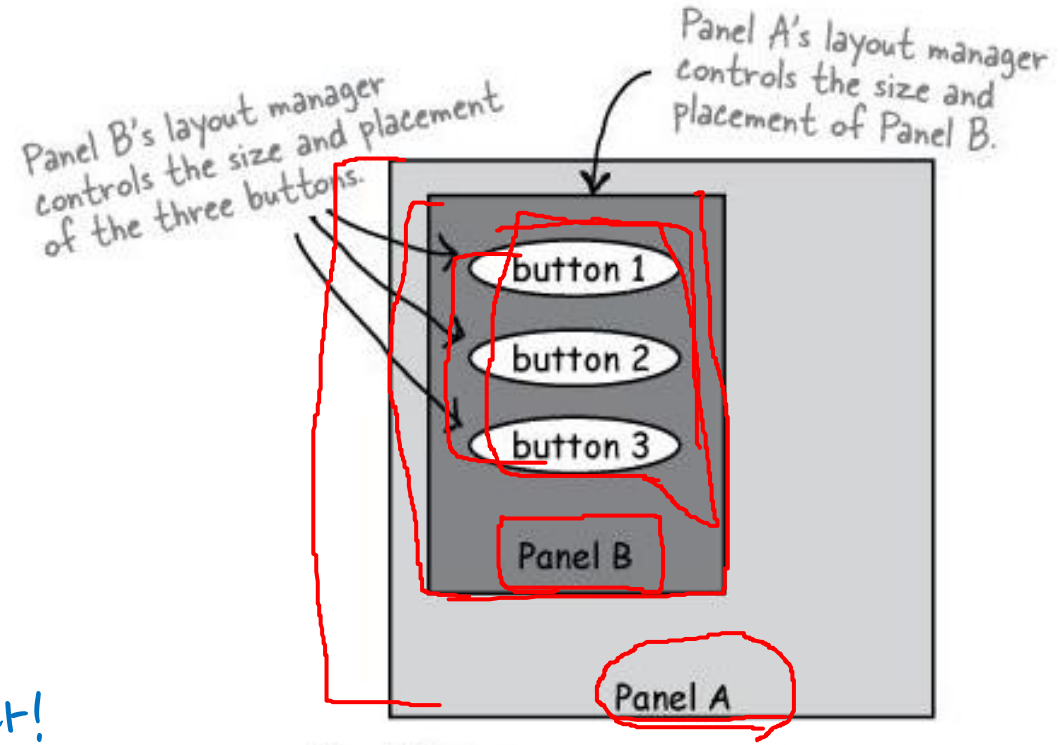


보통 아래처럼 버튼을 패널에 add하면 패널에 버튼이 '들어있다'라고 말한다.

```
myPanel.add(button);
```

```
JPanel panelA = new JPanel();
JPanel panelB = new JPanel();
panelB.add(new JButton("button 1"));
panelB.add(new JButton("button 2"));
panelB.add(new JButton("button 3"));
panelA.add(panelB);
```

Panel A 레이아웃 매니저는 세 개의 버튼에 대해서는 아무런 할 말이 없다!



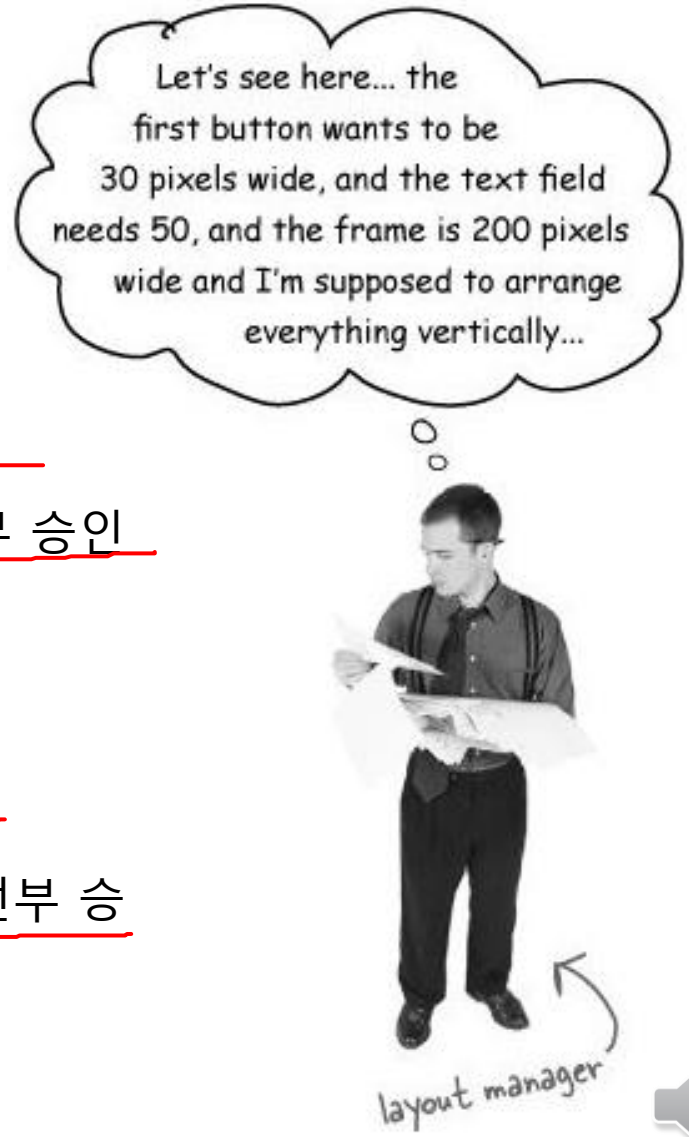
Panel A's layout manager has NOTHING to say about the three buttons. The hierarchy of control is only one level—Panel A's layout manager controls only the things added directly to Panel A, and does not control anything nested within those added components.

How does the layout manager decide?

레이아웃 매니저마다 컴포넌트를 **배열하는 정책**이 다르다.

A layout scenario:

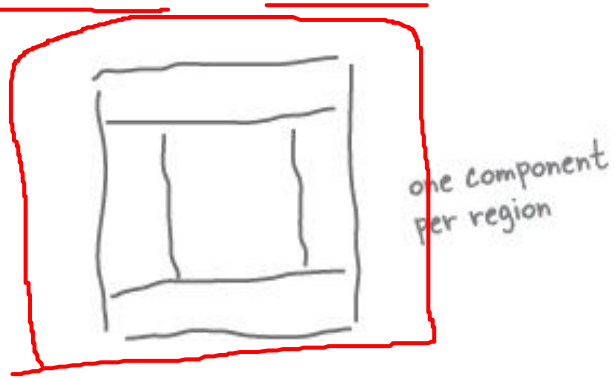
1. 패널에 세 개의 버튼을 추가한다
2. 패널의 레이아웃 매니저는 각 버튼에게 원하는 크기를 요청한다
3. 패널의 레이아웃 매니저는 **정책에 따라** 버튼의 요구사항을 [전부 승인/부분 승인/거절] 할 것인지를 결정한다.
4. 패널을 프레임에 추가한다
5. 프레임의 레이아웃 매니저는 패널에게 원하는 크기를 요청한다
6. 프레임의 레이아웃 매니저는 **정책에 따라** 패널의 요구사항을 [전부 승인/부분 승인/거절] 할 것인지를 결정한다



Big 3 layout managers: border, flow, and box

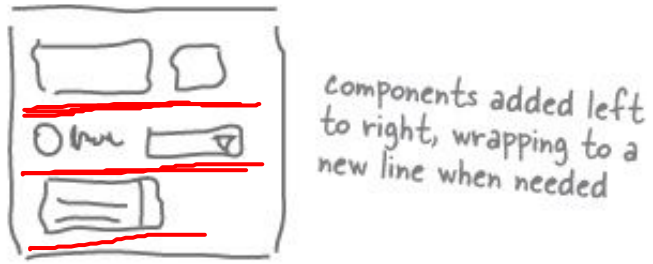
BorderLayout

- 5개의 지역으로 나눈다
- **프레임**의 디폴트 레이아웃 매니저



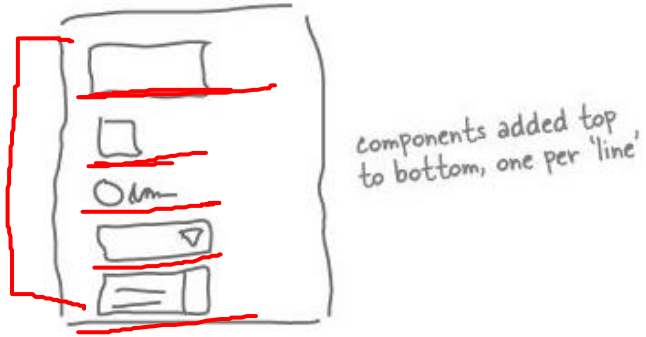
FlowLayout

- 워드 프로세서와 비슷하게 동작
- **원하는 크기만큼 왼쪽에서 오른쪽으로**
- 들어오는 순서대로 채워진다
- **패널**의 디폴트 레이아웃 매니저



BoxLayout

- 각 컴포넌트가 자신의 크기와 추가되는 순서대로 배치된다는 점에서 **FlowLayout과 유사**하다
- 그러나 **컴포넌트를 한 줄에 하나씩 수직방향으로** 쌓을 수 있다는 점에서는 다르다

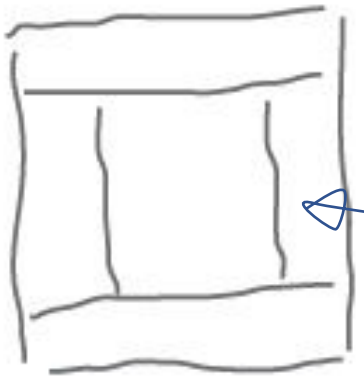


BorderLayout은 5개의 지역이 있다: east, west, north, south, and center

Let's add a button to the **EAST** region:

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

public class Button1 {

    public static void main (String[] args) {
        Button1 gui = new Button1();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        JButton button = new JButton("click me");
        frame.getContentPane().add(BorderLayout.EAST, button);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```

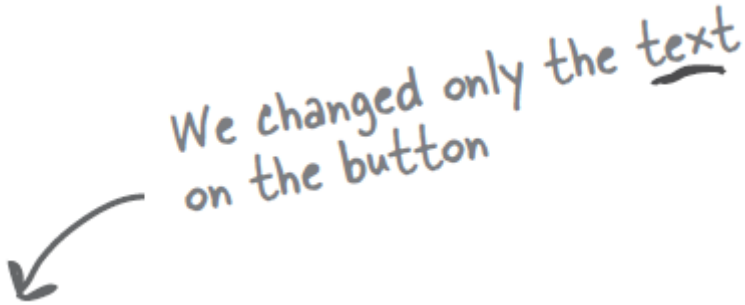
BorderLayout is in java.awt package



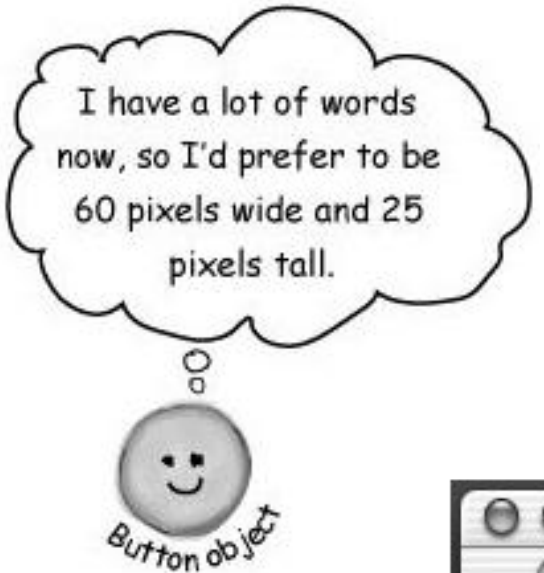
specify the region

버튼에 더 많은 글자를 넣으면 어떻게 될까?

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("click like you mean it");
    frame.getContentPane().add(BorderLayout.EAST, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```



버튼에 더 많은 글자를 넣으면 어떻게 될까?



The button gets its preferred **width**, but **not height.**

Let's try a button in the **NORTH** region

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("There is no spoon...");
    frame.getContentPane().add(BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```

The button is as tall as it wants to be, but as wide as the frame.

Now let's make the button ask to be taller

- ✓ 어떻게 버튼을 더 크게 만들 수 있을까?
- ✓ 버튼은 이미 프레임만큼 넓다 => 더 큰 글씨체를 줌으로써 더 키가 크도록 만들 수 있다.

```
public void go() {
    JFrame frame = new JFrame();
    JButton button = new JButton("Click This!");
    Font bigFont = new Font("serif", Font.BOLD, 28);
    button.setFont(bigFont);
    frame.getContentPane().add(BorderLayout.NORTH, button);
    frame.setSize(200,200);
    frame.setVisible(true);
}
```

A bigger font will force the frame to allocate more space for the button's height

The width stays the same, but now the button is taller. The north region stretched to accomodate the button's new preferred height.

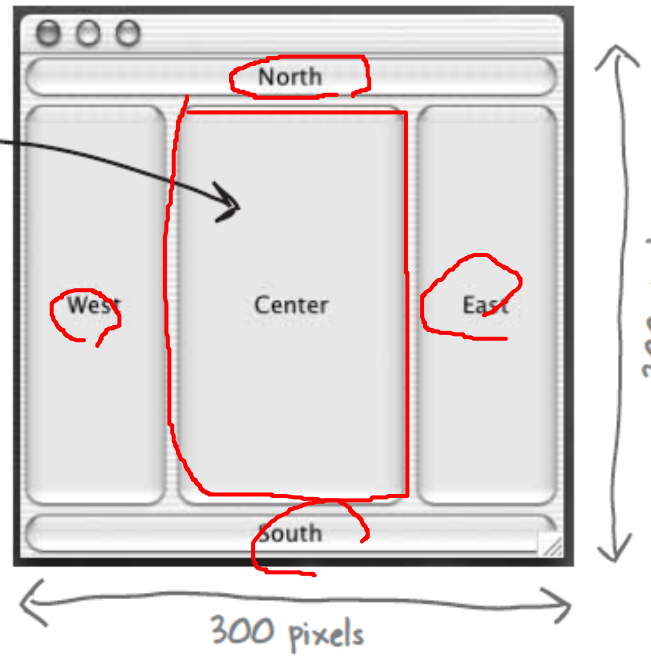
The center region gets whatever's left!

```
public void go() {
    JFrame frame = new JFrame();

    JButton east = new JButton("East");
    JButton west = new JButton("West");
    JButton north = new JButton("North");
    JButton south = new JButton("South");
    JButton center = new JButton("Center");

    frame.getContentPane().add(BorderLayout.EAST, east);
    frame.getContentPane().add(BorderLayout.WEST, west);
    frame.getContentPane().add(BorderLayout.NORTH, north);
    frame.getContentPane().add(BorderLayout.SOUTH, south);
    frame.getContentPane().add(BorderLayout.CENTER, center);

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

Components in the center get whatever space is left over, based on the frame dimensions (300 x 300 in this code).

Components in the east and west get their preferred width.

Components in the north and south get their preferred height.

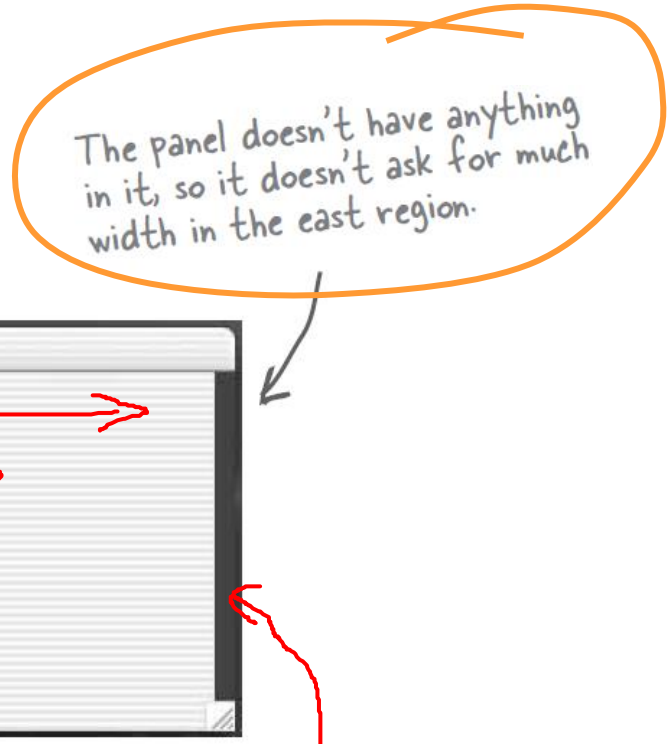


When you put something in the north or south, it goes all the way across the frame, so the things in the east and west won't be as tall as they would be if the north and south regions were empty.

FlowLayout에서는 컴포넌트를 순서대로 배치

왼쪽에서 오른쪽으로, 위에서 아래로 순서대로 추가된다.

- ✓ 동쪽 지역에 패널을 추가해 보자.
- ✓ **JPanel**의 디폴트 레이아웃 관리자는 **FlowLayout**이다.



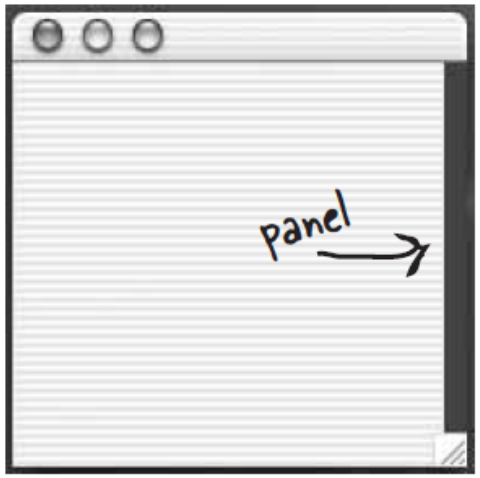

```
import javax.swing.*;
import java.awt.*;

public class Panel1 {

    public static void main (String[] args) {
        Panel1 gui = new Panel1();
        gui.go();
    }


    public void go() {
        JFrame frame = new JFrame();
        JPanel panel = new JPanel();
        panel.setBackground(Color.darkGray);
        frame.getContentPane().add(BorderLayout.EAST, panel);
        frame.setSize(200,200);
        frame.setVisible(true);
    }
}
```

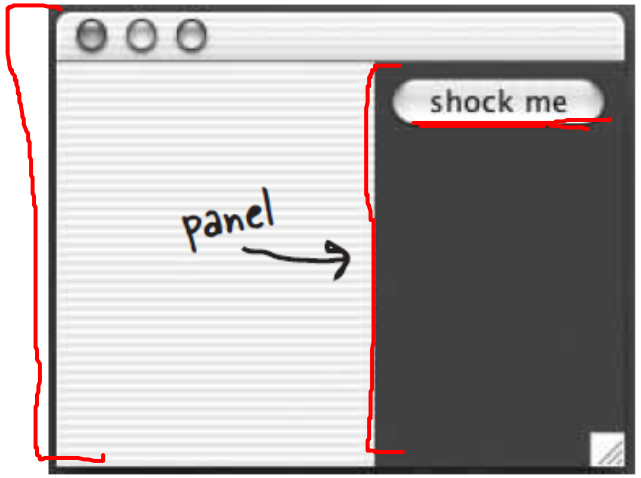
Make the panel gray so we can see where it is on the frame.

패널에 버튼을 추가하자.

```
public void go() {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);

      JButton button = new JButton("shock me");

      panel.add(button);
      frame.getContentPane().add(BorderLayout.EAST, panel);

      frame.setSize(250,200);
      frame.setVisible(true);
}
```

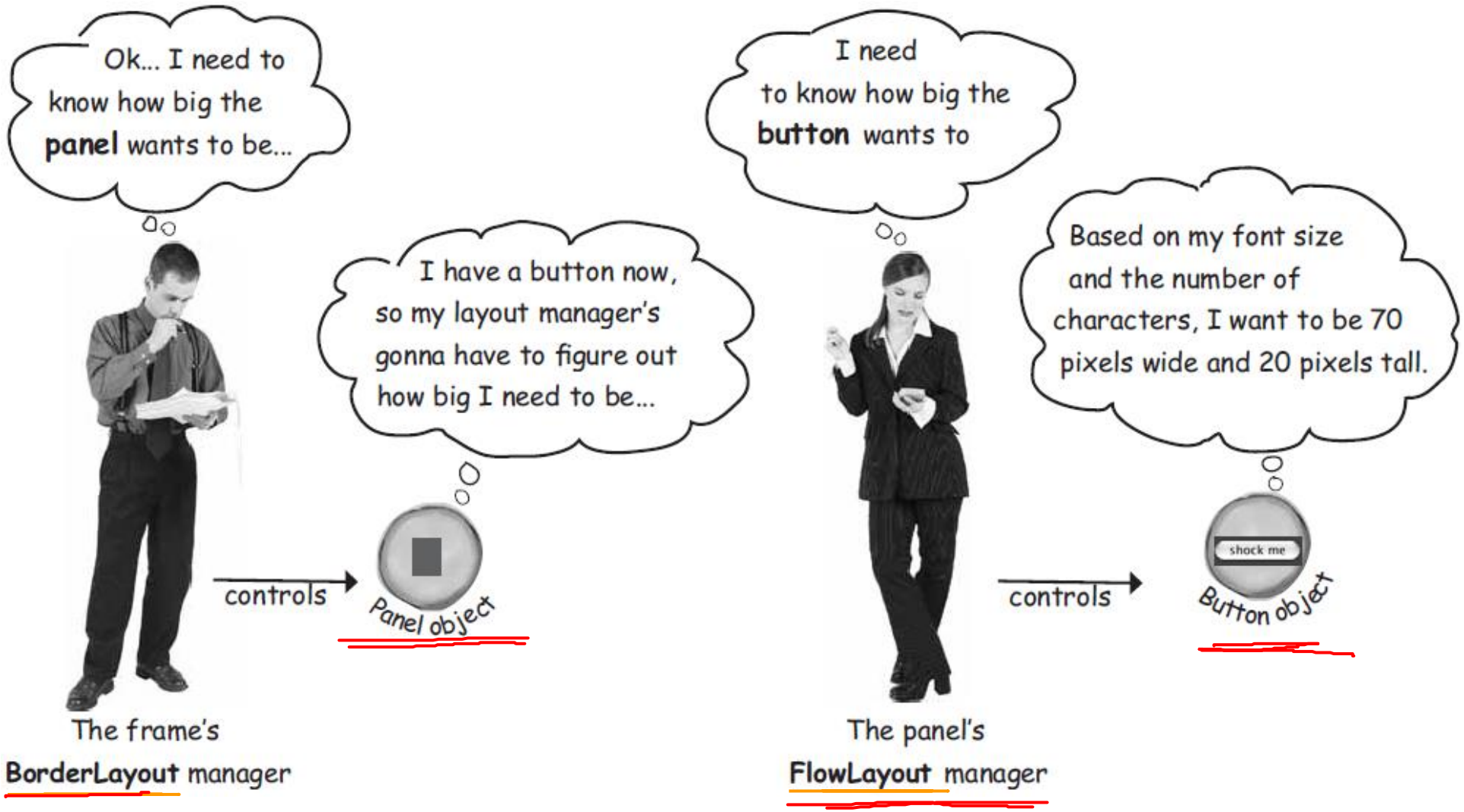
Add the button to the panel and add the panel to the frame. The panel's layout manager (flow) controls the button, and the frame's layout manager (border) controls the panel.





The panel expanded!

And the button got its preferred size in both dimensions, because the panel uses flow layout, and the button is part of the panel (not the frame).


Ok... I need to know how big the **panel** wants to be...
I have a button now, so my layout manager's gonna have to figure out how big I need to be...
I need to know how big the **button** wants to
Based on my font size and the number of characters, I want to be 70 pixels wide and 20 pixels tall.
controls
Panel object
controls
shock me
Button object
The frame's **BorderLayout** manager
The panel's **FlowLayout** manager

What happens if we add TWO buttons to the panel?

```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");

    panel.add(button);
    panel.add(buttonTwo);

    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

make TWO buttons

add BOTH to the panel

what we *wanted*:

We want the buttons stacked on top of each other

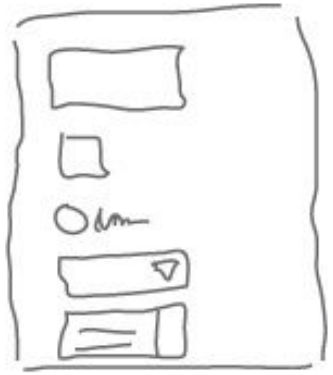
what we *got*:

BoxLayout이 해결책!!

나란히 놓을 수 있는 공간이 있어도 세로 방향으로 배치할 수 있다.

FlowLayout과는 달리 **BoxLayout**에서는 수평 방향으로 공간이 있어도 강제로 컴포넌트를 다음 줄로 넘길 수 있다.

한 줄에 하나씩!!



```
public void go() {
    JFrame frame = new JFrame();
    JPanel panel = new JPanel();
    panel.setBackground(Color.darkGray);

    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));


    JButton button = new JButton("shock me");
    JButton buttonTwo = new JButton("bliss");
    panel.add(button);
    panel.add(buttonTwo);
    frame.getContentPane().add(BorderLayout.EAST, panel);
    frame.setSize(250,200);
    frame.setVisible(true);
}
```

Change the layout manager to be a new instance of BoxLayout.

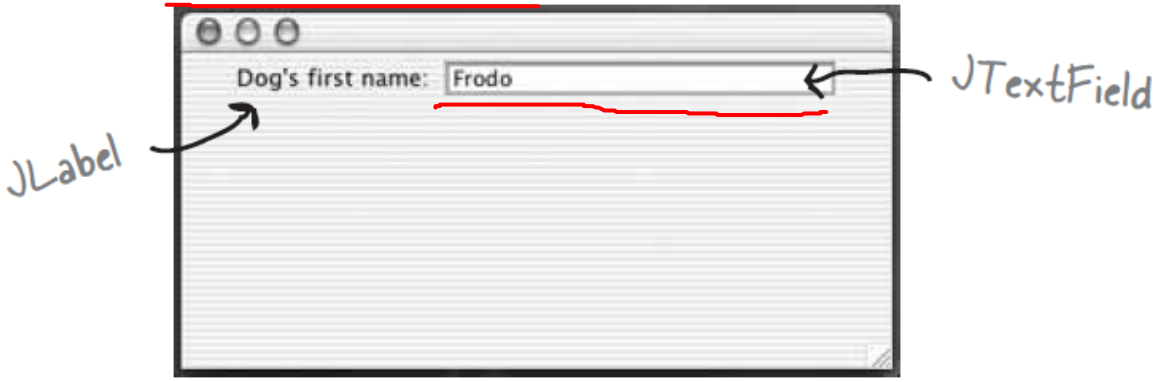
The BoxLayout constructor needs to know the component its laying out (i.e., the panel) and which axis to use (we use Y_AXIS for a vertical stack).

Notice how the panel is narrower again, because it doesn't need to fit both buttons horizontally. So the panel told the frame it needed enough room for only the largest button, 'shock me'.

20

Playing with Swing components

JTextField



Constructors

```
JTextField field = new JTextField(20);
```

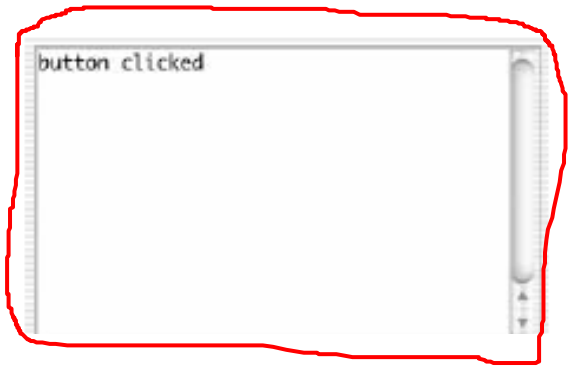
20 means 20 columns, not 20 pixels. This defines the preferred width of the text field.

```
JTextField field = new JTextField("Your name");
```

How to use it

1. 텍스트를 가져온다
```
System.out.println(field.getText());
```
2. Put text in it
```
field.setText("whatever");
field.setText("");  // This clears the field
```
3. Get an ActionEvent when the user presses return or enter
```
field.addActionListener(myActionListener);
```
4. Select / Highlight the text in the field
```
field.selectAll();
```
5. Put the cursor back in the field (so the user can just start typing)
```
field.requestFocus();
```

JTextArea



Constructor

```
JTextArea text = new JTextArea(10,20);
```

10 means 10 lines (sets the preferred height)

20 means 20 columns (sets the preferred width)

How to use it

1. 수직 스크롤 바만 가지도록 한다

```
JScrollPane scroller = new JScrollPane(text);
text.setLineWrap(true);

scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

panel.add(scroller);
```

Turn on line wrapping

Tell the scroll pane to use o a vertical scrollbar

2. 텍스트로 대치한다

```
text.setText("Not all who are lost are wandering");
```

3. 텍스트를 추가한다

```
text.append("button clicked");
```

4. 필드의 텍스트를 선택하거나 하이라이트 한다

```
text.selectAll();
```

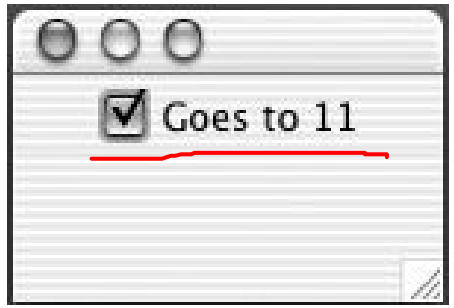
5. 커서를 필드에 둔다 (사용자가 바로 입력할 수 있도록)

```
text.requestFocus();
```

JCheckBox

생성자

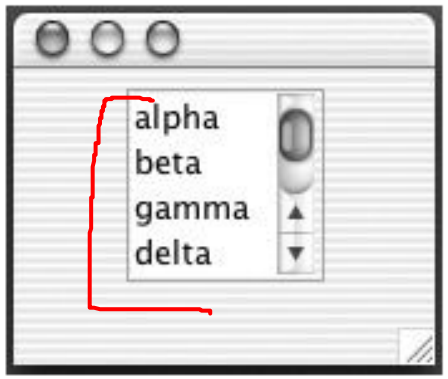
```
JCheckBox check = new JCheckBox("Goes to 11");
```



How to use it

1. Listen for an item event (when it's selected or deselected)
```
check.addItemListener(this);
```
2. Handle the event (and find out whether or not it's selected)
```
public void itemStateChanged(ItemEvent ev) {
    String onOrOff = "off";
    if (check.isSelected()) onOrOff = "on";
    System.out.println("Check box is " + onOrOff);
}
```
3. Select or deselect it in code
```
check.setSelected(true);
check.setSelected(false);
```

JList



Constructor

```
String [] listEntries = {"alpha", "beta", "gamma", "delta",
                         "epsilon", "zeta", "eta", "theta "};
JList list = new JList(listEntries);
```

How to use it

1. Make it have a vertical scrollbar

```
JScrollPane scroller = new JScrollPane(list);
scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
panel.add(scroller);
```

2. Set the number of lines to show before scrolling

```
list.setVisibleRowCount(4);
```

3. Restrict the user to selecting only ONE thing at a time

```
list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

4. Register for list selection events

```
list.addListSelectionListener(this);
```

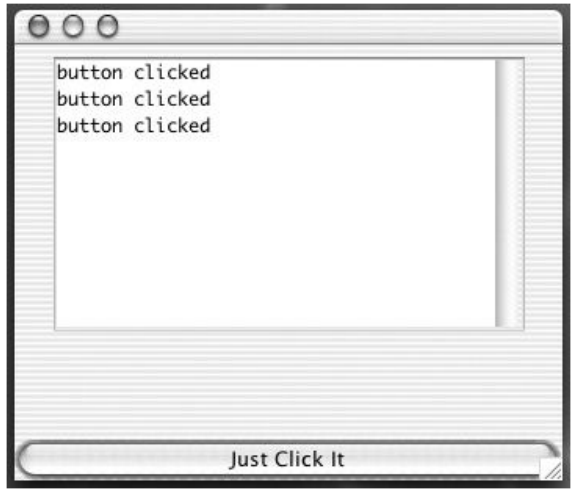
5. Handle events (find out which thing in the list was selected)

```
public void valueChanged(ListSelectionEvent lse) {
    if( !lse.getValueIsAdjusting()) {
      String selection = (String) list.getSelectedValue();
      System.out.println(selection);
    }
}
```

getSelectedValue() actually returns an Object. A list is limited to only String objec

실습과제 13-1 JTextArea example

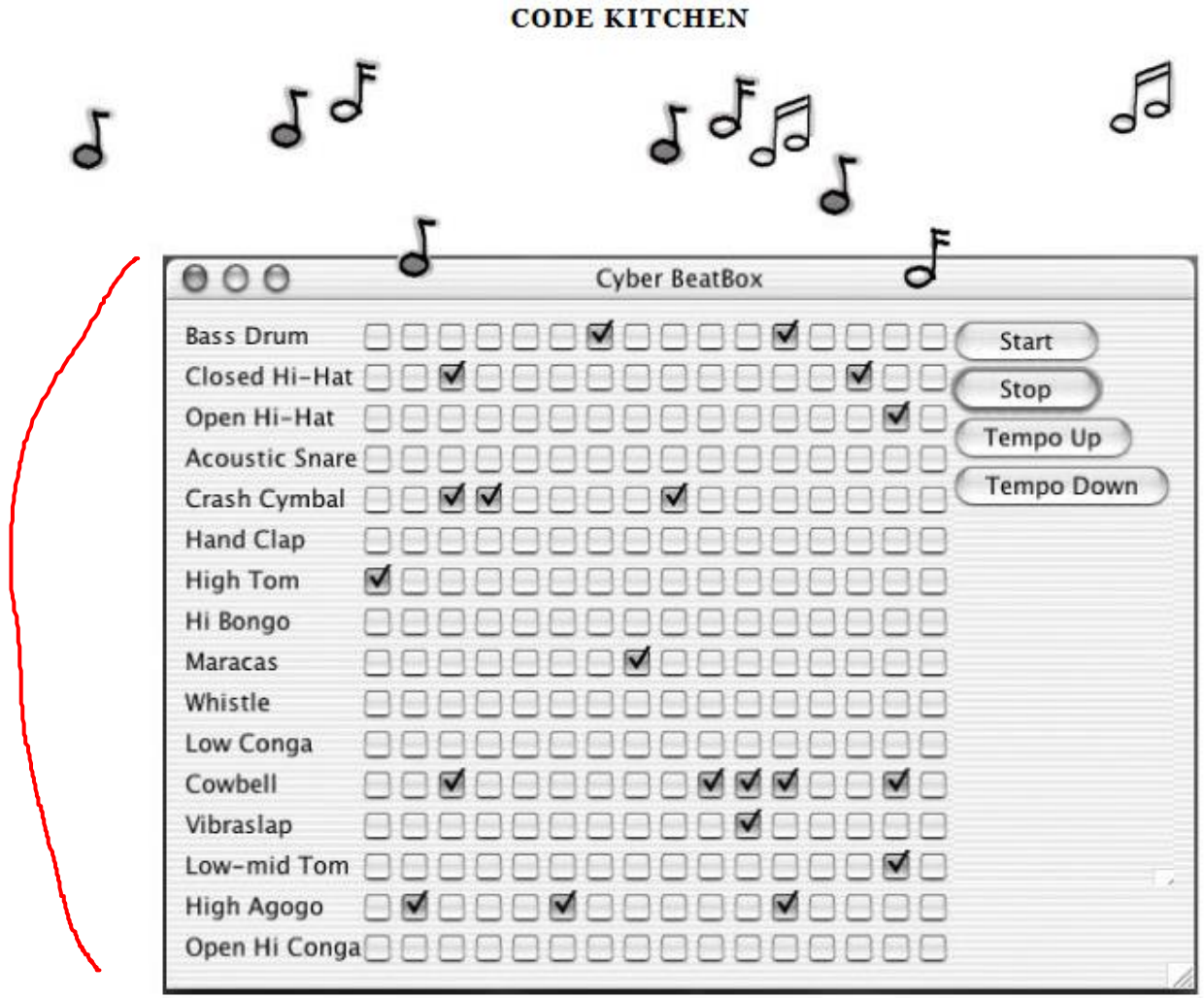
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;


public class TextArea1 implements ActionListener {

    JTextArea text;

    public static void main (String[] args) {
       TextArea1 gui = new TextArea1();
       gui.go();
    }


   public void go() {
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      JButton button = new JButton("Just Click It");
      button.addActionListener(this);
      text = new JTextArea(10,20);
      text.setLineWrap(true);

      JScrollPane scroller = new JScrollPane(text);
      scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
      scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

      panel.add(scroller);

      frame.getContentPane().add(BorderLayout.CENTER, panel);
      frame.getContentPane().add(BorderLayout.SOUTH, button);

      frame.setSize(350,300);
      frame.setVisible(true);
   }

   public void actionPerformed(ActionEvent ev) {
      text.append("button clicked \n ");
   }
}
```



Insert a new line so the words go on a separate line each time the button is clicked. Otherwise, they'll run together.

코드 키친

Making the BeatBox



Making the BeatBox

This is the full code listing for this version of the BeatBox, with buttons for starting, stopping, and changing the tempo. The code listing is complete, and fully-annotated, but here's the overview:

1. Build a GUI that has 256 checkboxes (JCheckBox) that start out unchecked, 16 labels (JLabel) for the instrument names, and four buttons.

2. Register an ActionListener for each of the four buttons. We don't need listeners for the individual checkboxes, because we aren't trying to change the pattern sound dynamically (i.e. as soon as the user checks a box). Instead, we wait until the user hits the 'start' button, and then walk through all 256 checkboxes to get their state and make a MIDI track.

3. Set-up the MIDI system (you've done this before) including getting a Sequencer, making a Sequence, and creating a track. We are using a sequencer method that's new to Java 5.0, setLoopCount( ). This method allows you to specify how many times you want a sequence to loop. We're also using the sequence's tempo factor to adjust the tempo up or down, and maintain the new tempo from one iteration of the loop to the next.

4. When the user hits 'start', the real action begins. The event-handling method for the 'start' button calls the buildTrackAndStart() method. In that method, we walk through all 256 checkboxes (one row at a time, a single instrument across all 16 beats) to get their state, then use the information to build a MIDI track (using the handy makeEvent() method we used in the previous chapter). Once the track is built, we start the sequencer, which keeps playing (because we're looping it) until the user hits 'stop'.

```
import java.awt.*;
import javax.swing.*;
import javax.sound.midi.*;
import java.util.*;
import java.awt.event.*;


public class BeatBox {


    JPanel mainPanel;
    ArrayList<JCheckBox> checkboxList;
    Sequencer sequencer;
    Sequence sequence;
    Track track;
    JFrame theFrame;

    String[] instrumentNames = {"Bass Drum", "Closed Hi-Hat",
       "Open Hi-Hat","Acoustic Snare", "Crash Cymbal", "Hand Clap",
       "High Tom", "Hi Bongo", "Maracas", "Whistle", "Low Conga",
       "Cowbell", "Vibraslap", "Low-mid Tom", "High Agogo",
       "Open Hi Conga"};
    int[] instruments = {35,42,46,38,49,39,50,60,70,72,64,56,58,47,67,63};


    public static void main (String[] args) {
        new BeatBox2().buildGUI();
    }

    public void buildGUI() {
        theFrame = new JFrame("Cyber BeatBox");
        theFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BorderLayout layout = new BorderLayout();
        JPanel background = new JPanel(layout);
        background.setBorder(BorderFactory.createEmptyBorder(10,10,10,10));

        checkboxList = new ArrayList<JCheckBox>();
        Box buttonBox = new Box(BoxLayout.Y_AXIS);

        JButton start = new JButton("Start");
        start.addActionListener(new MyStartListener());
        buttonBox.add(start);

        JButton stop = new JButton("Stop");
        stop.addActionListener(new MyStopListener());
        buttonBox.add(stop);

        JButton upTempo = new JButton("Tempo Up");
        upTempo.addActionListener(new MyUpTempoListener());
        buttonBox.add(upTempo);

        JButton downTempo = new JButton("Tempo Down");
```

checkBox를 ArrayList에 저장한다.

GUI 레이블을 만들 때 사용할 String 배열로 저장될 악기명.
(각 행 별로)

실제 드럼 건반(keys)을 나타낸다. 드럼 채널은 피아노의 각 'key'가 하나의 다른 드럼에 해당된다는 점을 제외하고는 피아노와 비슷하다. '35'번은 베이스에 대한 key를, '42'번은 Hi-Hat key를 의미하는 것처럼.

'empty border'는 패널의 에지와 컴포넌트가 놓여지는 공간 사이에 여백을 제공해준다.

Nothing special here, just lots of GUI code. You've seen most of it before.

```
    downTempo.addActionListener(new MyDownTempoListener());
    buttonBox.add(downTempo);

    Box nameBox = new Box(BoxLayout.Y_AXIS);
    for (int i = 0; i < 16; i++) {
       nameBox.add(new Label(instrumentNames[i]));
    }

    background.add(BorderLayout.EAST, buttonBox);
    background.add(BorderLayout.WEST, nameBox);

    theFrame.getContentPane().add(background);

    GridLayout grid = new GridLayout(16,16);
    grid.setVgap(1);
    grid.setHgap(2);
    mainPanel = new JPanel(grid);
    background.add(BorderLayout.CENTER, mainPanel);

    for (int i = 0; i < 256; i++) {
        JCheckBox c = new JCheckBox();
        c.setSelected(false);
        checkboxList.add(c);
        mainPanel.add(c);
    } // end loop

    setUpMidi();

    theFrame.setBounds(50,50,300,300);
    theFrame.pack();
    theFrame.setVisible(true);
} // close method


public void setUpMidi() {
  try {
    sequencer = MidiSystem.getSequencer();
    sequencer.open();
    sequence = new Sequence(Sequence.PPQ,4);
    track = sequence.createTrack();
    sequencer.setTempoInBPM(120);

  } catch(Exception e) {e.printStackTrace();}
} // close method
```

Still more GUI set-up code.
Nothing remarkable.

체크박스를 만들어서 'false'를 설정한다.
(처음에는 체크되지 않은 상태로 보인다)
그리고 그들을 ArrayList와 GUI 패널에 추가한다.

시퀀서, 시퀀스, 트랙을 얻기 위한 일반적인
MIDI 설정 관련 코드.
특별한 내용은 없다.

모든 것이 일어나는 곳!! 여기서 체크박스 상태를 MIDI 이벤트로 바꾸어서 그들을 트랙에 추가한다.

```
public void buildTrackAndStart() {
  int[] trackList = null;

  sequence.deleteTrack(track);
  track = sequence.createTrack();

   for (int i = 0; i < 16; i++) {
     trackList = new int[16];

     int key = instruments[i];

     for (int j = 0; j < 16; j++ ) {

        JCheckBox jc = (JCheckBox) checkboxList.get(j + (16*i));
        if ( jc.isSelected()) {
           trackList[j] = key;
        } else {
           trackList[j] = 0;
        }
     } // close inner loop

     makeTracks(trackList);
     track.add(makeEvent(176,1,127,0,16));
  } // close outer

  track.add(makeEvent(192,9,1,0,15));
  try {

     sequencer.setSequence(sequence);
     sequencer.setLoopCount(sequencer.LOOP_CONTINUOUSLY);
     sequencer.start();
     sequencer.setTempoInBPM(120);
  } catch(Exception e) {e.printStackTrace();}
} // close buildTrackAndStart method

public class MyStartListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        buildTrackAndStart();
    }
} // close inner class
```

각 악기에 대한 값을 저장하기 위한 16-원소 배열을 16 비트 모두에 걸쳐서 만들 것이다. 악기가 특정 비트에서 연주되어야 한다면 key 값을, 아니면 0값을 넣는다.

기존 트랙을 제거하고 새로운 트랙을 만든다.

16개 행의 각각에 대해 반복한다. (베이스, 콩고 등)

어떤 악기 (Bass, Hi-Hat 등) 인지를 나타내는 'key'를 설정한다. instruments 배열은 각 악기에 대한 실제 MIDI 번호를 저장한다.

Do this for each of the BEATS for this row

해당 비트의 체크박스가 체크된 상태인가? Yes이면 배열의 해당 slot에 key 값을 넣는다. No이면 이 악기는 현재 비트에서는 연주되지 않은 것으로 간주하여 0값을 넣는다.

이 악기에 대해 그리고 16개 모든 비트에 대해 이벤트를 만들어서 트랙에 추가한다.

16번째 비트(0~15)에는 이벤트가 반드시 있도록 확인한다. 그렇지 않으면 다시 시작하기 전에 16비트 모두 종료되지 않을 수 있다.

루프 반복 횟수를 지정한다. 이 경우 continuous looping.

NOW PLAY THE THING!!

버튼들을 위한 첫 번째 내부 클래스.

```
public class MyStopListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
        sequencer.stop();
    }
} // close inner class

public class MyUpTempoListener implements ActionListener {
    public void actionPerformed(ActionEvent a) {
       float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * 1.03));
    }
 } // close inner class

 public class MyDownTempoListener implements ActionListener {
     public void actionPerformed(ActionEvent a) {
       float tempoFactor = sequencer.getTempoFactor();
        sequencer.setTempoFactor((float)(tempoFactor * .97));
    }
} // close inner class
```

나머지 버튼들을 위한 내부 클래스 리스너들.

시퀀서의 템포를 주어진 배율로 변경한다. 디폴트 값은 1.00이고, 여기서는 클릭당 +/-3%씩 증감시킨다.

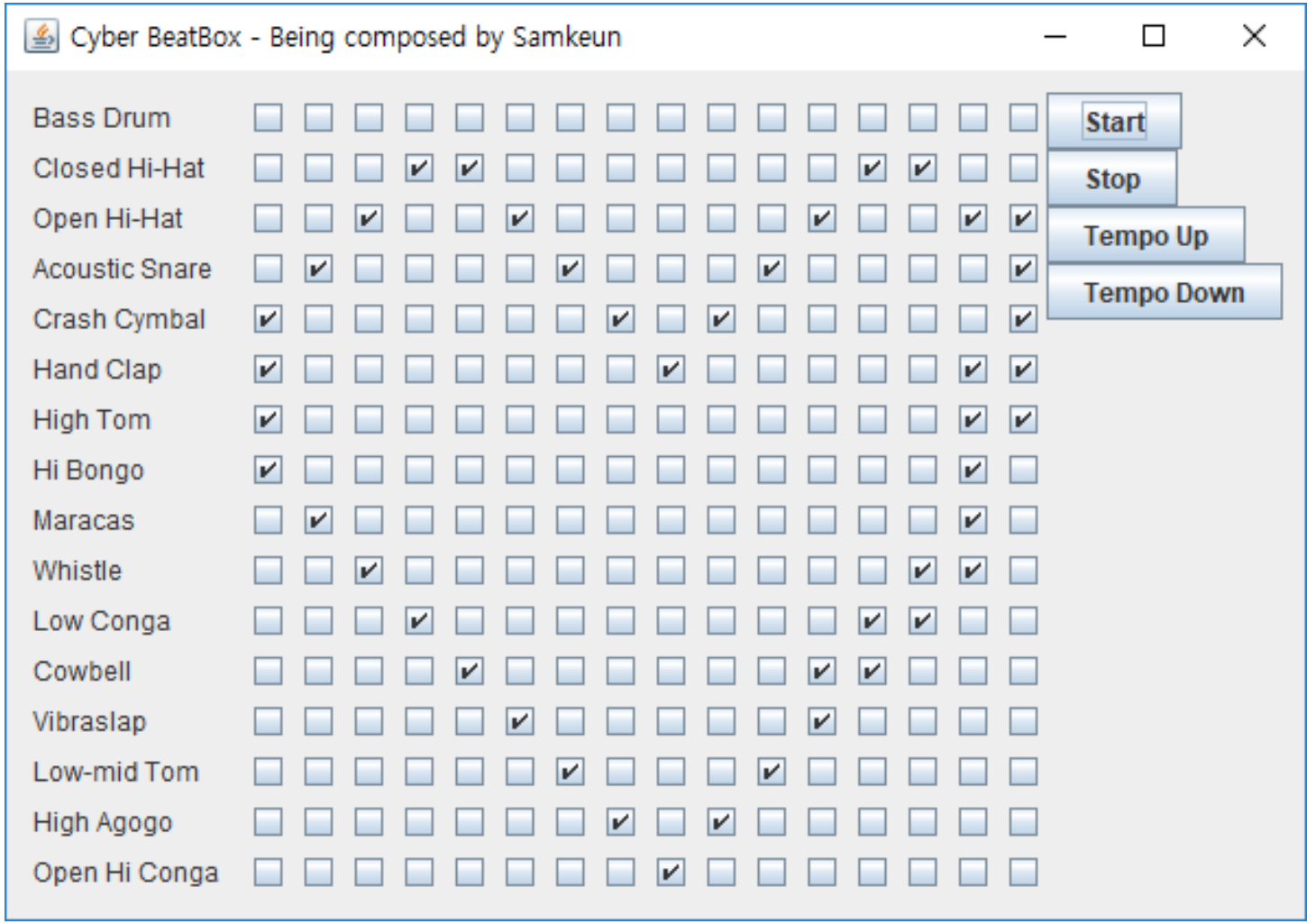
```
public void makeTracks(int[] list) {

   for (int i = 0; i < 16; i++) {
      int key = list[i];

      if (key != 0) {
         track.add(makeEvent(144,9,key, 100, i));
         track.add(makeEvent(128,9,key, 100, i+1));
      }
   }
}

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) {e.printStackTrace(); }
    return event;
}

} // close class
```

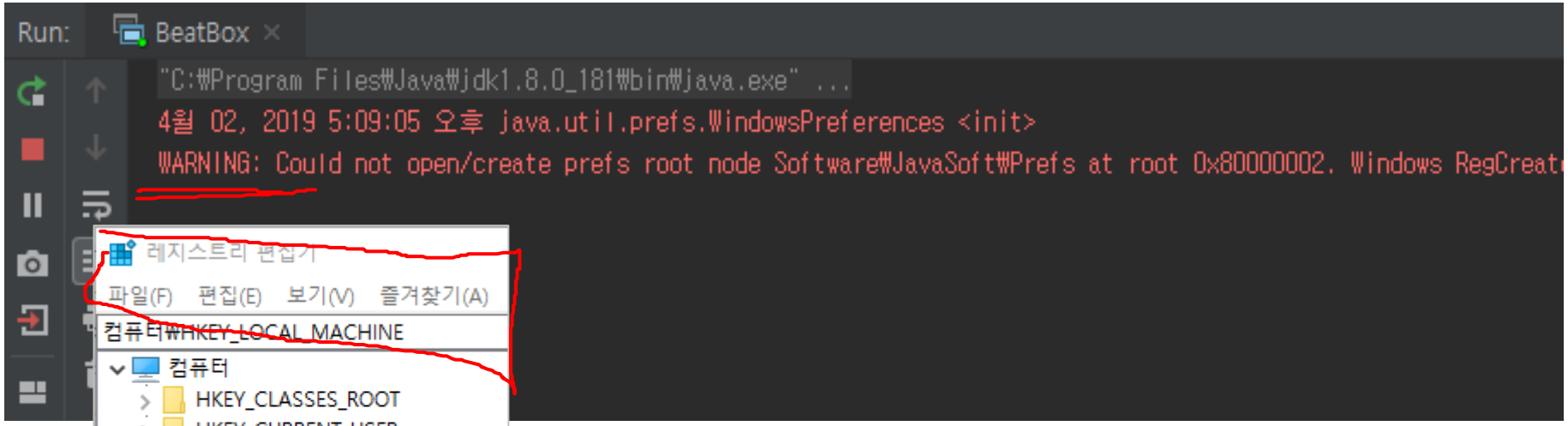
한 번에 하나의 악기의 16 비트 전체에 대하여 이벤트를 만든다. Bass 드럼을 위한 int[] 배열의 값을 수 있다. 배열의 각 인덱스는 Bass 드럼의 key 값 또는 0 값을 저장하게 된다. 0 값이면 해당 악기가 그 비트에서는 연주되지 않는다. 그렇지 않으면 이벤트를 만들어서 트랙에 추가한다.

Make the NOTE ON and NOT OFF events, and add them to the Track.

This is the utility method from last chapter's CodeKitchen. Nothing new.

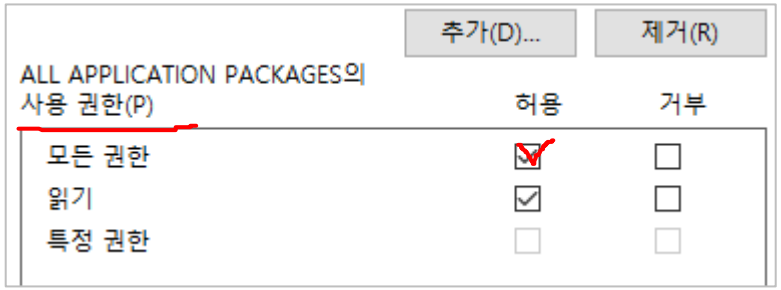
실습과제 13-2 Making the BeatBox





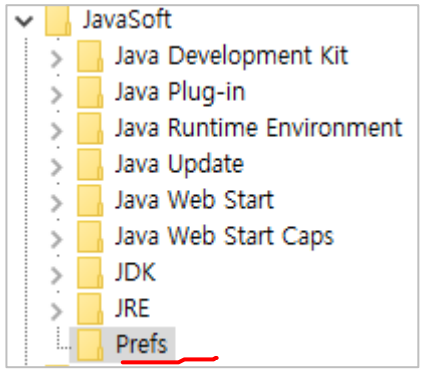
검색: regedit

HKEY_LOCAL_MACHINE > SOFTWARE > JavaSoft > 우클릭: 사용권한



한번 더 JavaSoft 우클릭 > 새로만들기 > 키(K) 선택

Prefs 생성



실습과제 13-3 슬롯 머신

다음과 같은 게임 프로그램이 작성되어 있다. 슬롯 머신처럼 버튼을 누르면 3개의 난수가 화면에 표시된다. 3개의 난수들이 일치하면 득점한다고 가정한다.

[Project 2: Virtual Math Tutor]

사이버캠퍼스 '과제' 참조!!