제 9 장 자바 GUI 기초, AWT와 스윙(SWING)

자바의 GUI(Graphical User Interface)

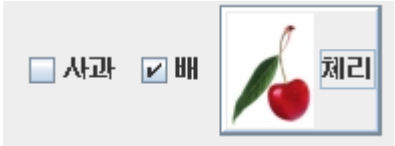
- GUI 목적
  - 그래픽 이용, 사용자에게 이해하기 쉬운 모양으로 정보 제공
  - 사용자는 마우스나 키보드를 이용하여 쉽게 입력
- 자바 GUI 특징
  - 강력한 GUI 컴포넌트 제공
  - 쉬운 GUI 프로그래밍
- 자바의 GUI 프로그래밍 방법
  - GUI 컴포넌트와 그래픽 이용
    - AWT 패키지와 Swing 패키지에 제공되는 메카니즘 이용
    - AWT - java.awt 패키지
    - Swing - javax.swing 패키지

AWT와 Swing 패키지

- AWT(Abstract Windowing Toolkit)
  - 자바가 처음 나왔을 때 함께 배포된 GUI 라이브러리
  - java.awt 패키지
    - Frame, Window, Panel, Button, Label, …
  - native(운영체제)와 응용프로그램 사이의 연결 라이브러리
    - 중량 컴포넌트(Heavy weight components)
      - *AWT 컴포넌트는 native(peer)에 의존적임*
      - *OS의 도움을 받아야 화면에 출력되며 동작하는 컴포넌트. 운영체제에 많은 부담. 오히려 처리 속도는 빠름*
      - *운영체제에 따라 서로 다른 모양으로 그려짐*
- Swing(스윙)
  - AWT 기술을 기반으로 작성된 자바 라이브러리
    - 모든 AWT 기능 + 추가된 풍부하고 화려한 고급 컴포넌트
    - AWT 컴포넌트에 J자가 덧붙여진 이름의 클래스
    - 그 외 J 자로 시작하는 클래스
  - 순수 자바 언어로 구현, JDK 1.1 부터 - javax.swing 패키지
    - JFrame, JWindow, JPanel, JButton, JLabel, …
  - Swing 컴포넌트는 native(peer 혹은 운영체제)에 의존하지 않음
    - 경량 컴포넌트(Light weight components)
    - 운영체제에 관계없이 항상 동일하게 작동하며 동일한 모양으로 그려짐

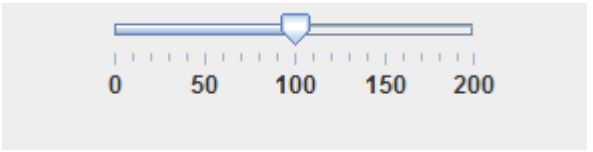
스윙 컴포넌트 예시

JButton

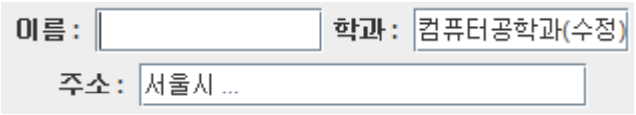


JCheckBox

JRadioButton



JSlider

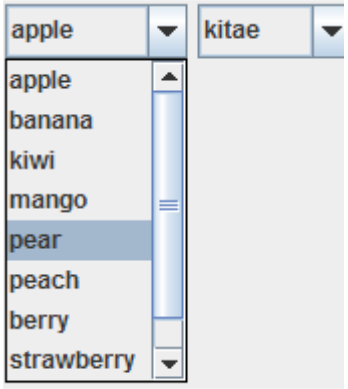


JTextField

JPasswordField

JSpinner

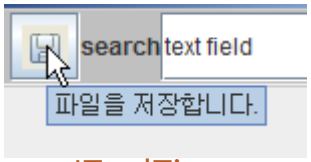
JTextArea



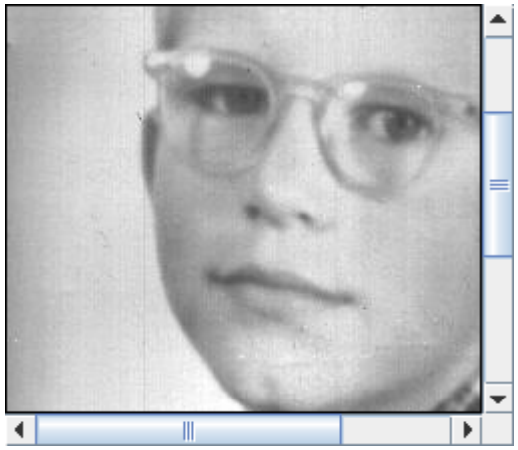
JComboBox

JList

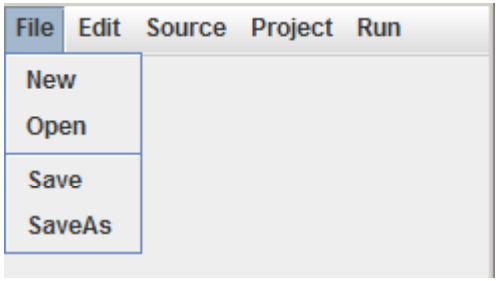
JProgressBar



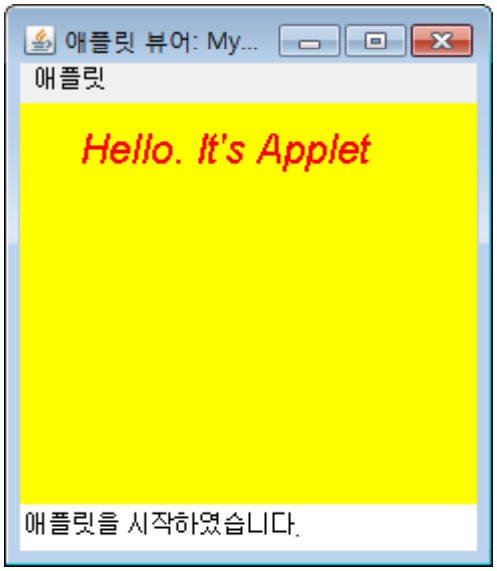
JToolTip



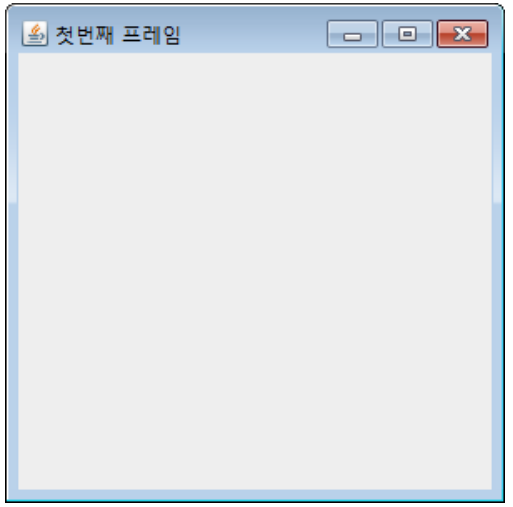
JScrollPane



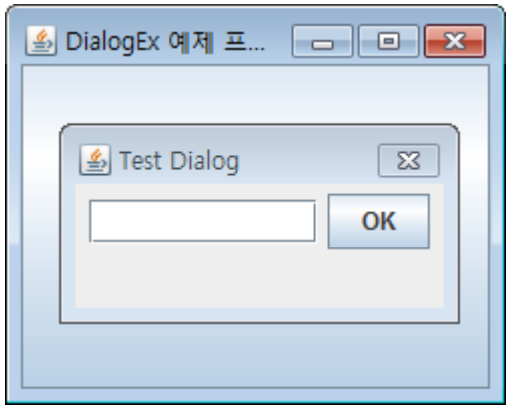
JMenu



JApplet



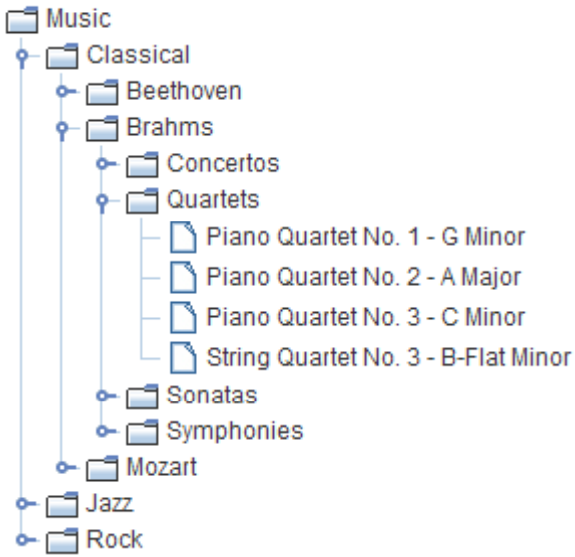
JFrame



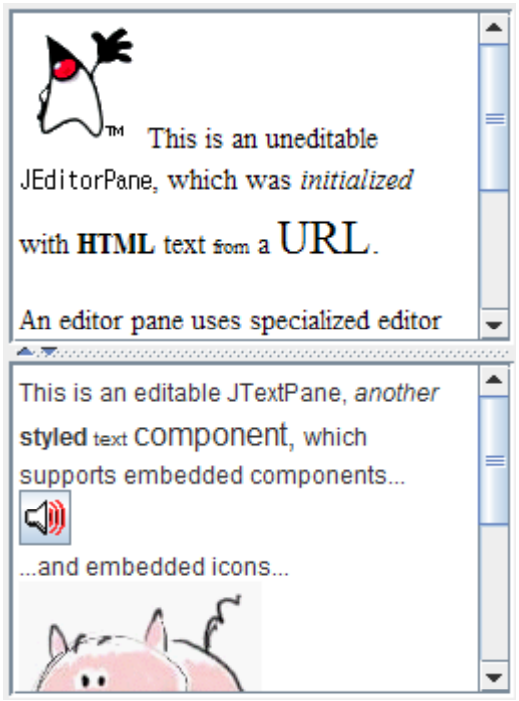
JDialog

First Name	Last Name	Favorite Color	Favorite Movie	Favorite Number	Favorite Food
Mike	Albers	Green	Brazil	44	
Mark	Andrews	Blue	Curse of the Dem...	3	
Brian	Beck	Black	The Blues Brothers	2.718	
Lara	Bunni	Red	Airplane (the whol...	15	
Roger	Brinkley	Blue	The Man Who Kn...	13	
Brent	Christian	Black	Blade Runner (Dir...	23	

JTable



JTree



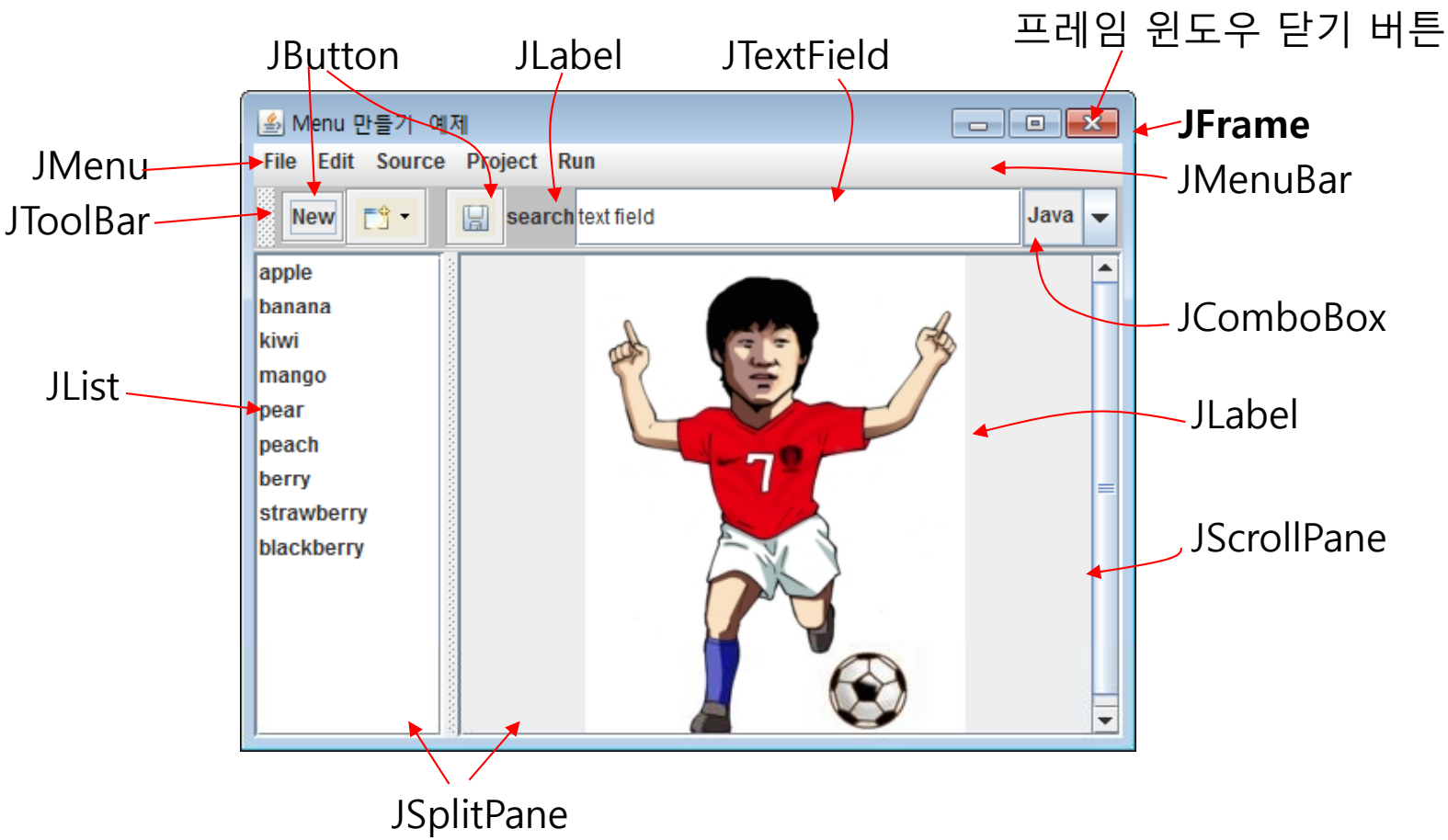
JEditorPane and JTextPane

JToolBar

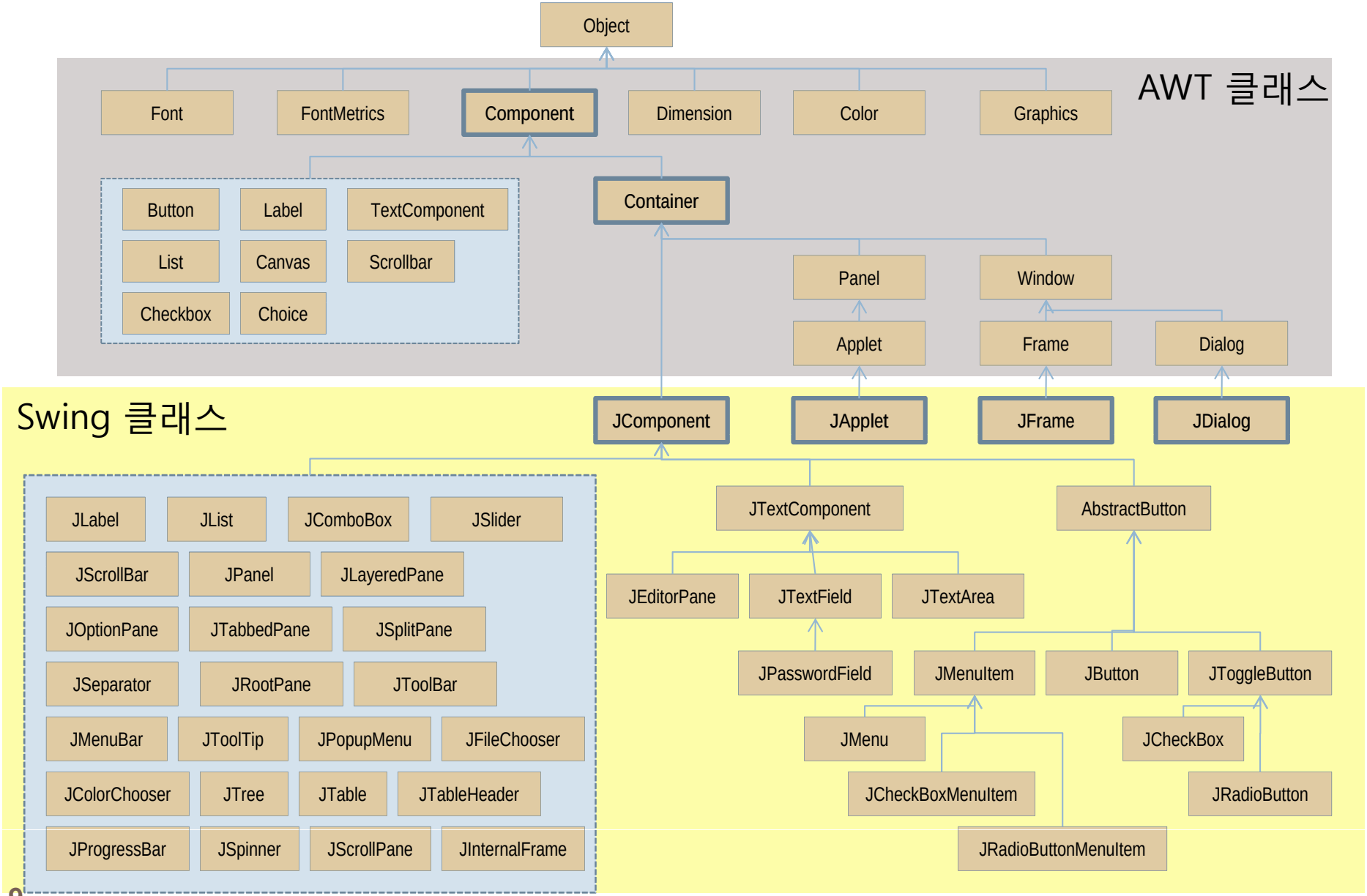
JTabbedPane

JSplitPane

Swing 으로 만든 GUI 프로그램 샘플



GUI 라이브러리 계층 구조



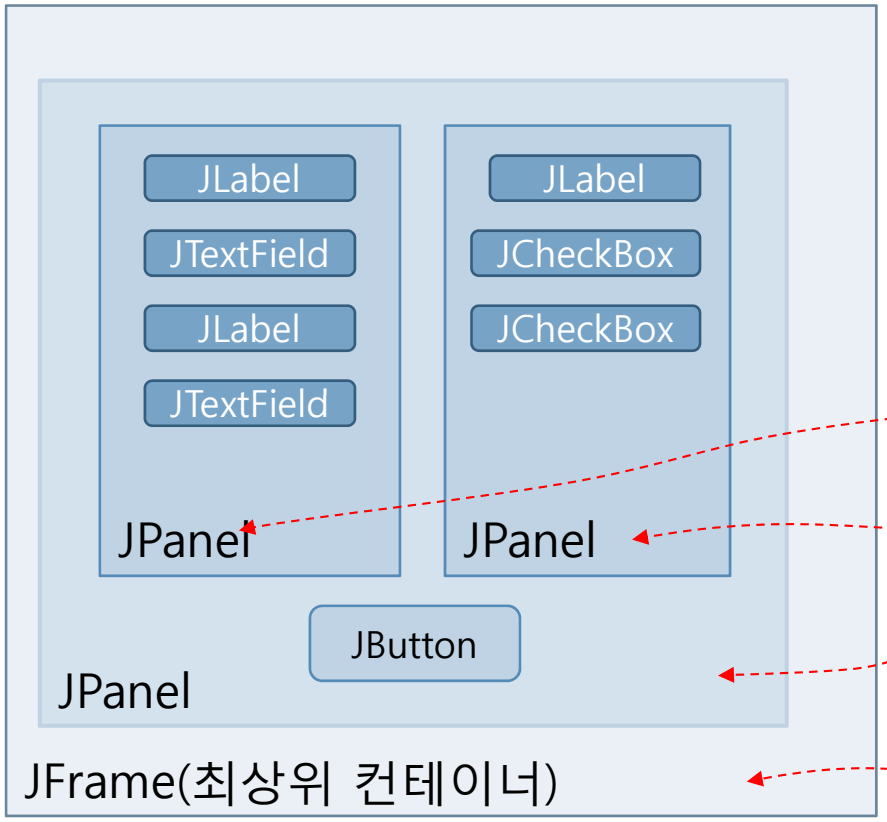
Swing 클래스의 특징

- 클래스 이름이 J 자로 시작
- 화려하고 다양한 컴포넌트로 쉽게 GUI 프로그래밍
- 스윙 컴포넌트는 2 가지 유형
  - Jcomponent를 상속받는 클래스
    - 대부분의 스윙 컴포넌트
  - AWT의 Container를 상속받는 몇 개의 클래스
    - JApplet, JDialog, JFrame 등
- JComponent
  - 매우 중요한 추상 클래스
    - 스윙 컴포넌트의 공통적인 속성 구현
    - ~~new JComponent()~~ 인스턴스를 생성할 수 없음
  - AWT의 Component를 상속받음

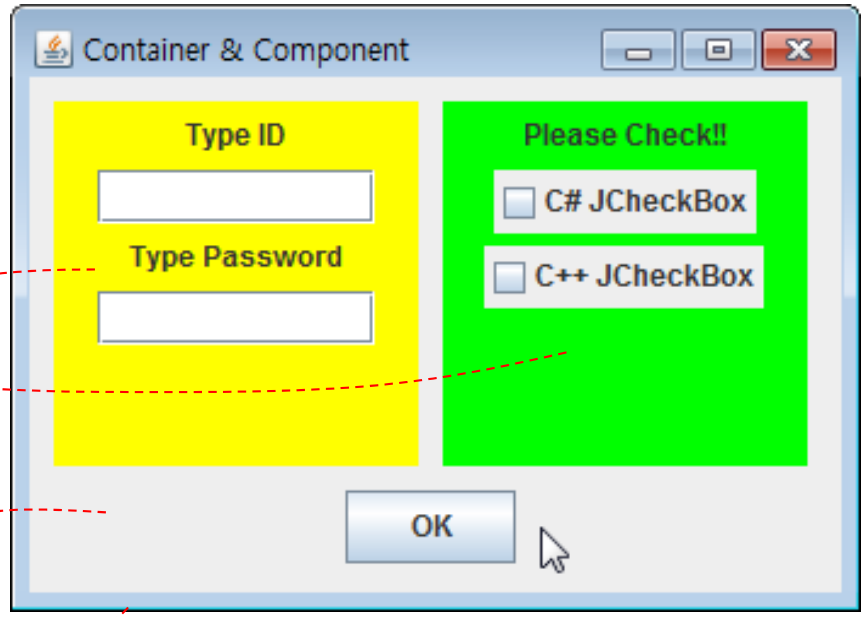
컨테이너와 컴포넌트

- 컨테이너
  - 다른 컴포넌트를 포함할 수 있는 GUI 컴포넌트
    - java.awt.Container를 상속받음
  - 다른 컨테이너에 포함될 수 있음
    - AWT 컨테이너
      - *Panel, Frame, Applet, Dialog, Window*
    - Swing 컨테이너
      - *JPanel JFrame, JApplet, JDialog, JWindow*
- 최상위 컨테이너
  - 다른 컨테이너에 속하지 않고 독립적으로 존재 가능한 컨테이너
    - 스스로 화면에 자신을 출력하는 컨테이너
    - JFrame, JDialog, JApplet
  - 모든 컴포넌트는 컨테이너에 포함되어야 화면에 출력 가능
- 컴포넌트
  - 컨테이너에 포함되어야 화면에 출력될 수 있는 GUI 객체
  - 모든 GUI 컴포넌트의 최상위 클래스
    - java.awt.Component
  - 스윙 컴포넌트의 최상위 클래스
    - javax.swing.JComponent

컨테이너와 컴포넌트의 포함관계



스윙의 컨테이너와 컴포넌트의 포함 관계



스윙 GUI 프로그램

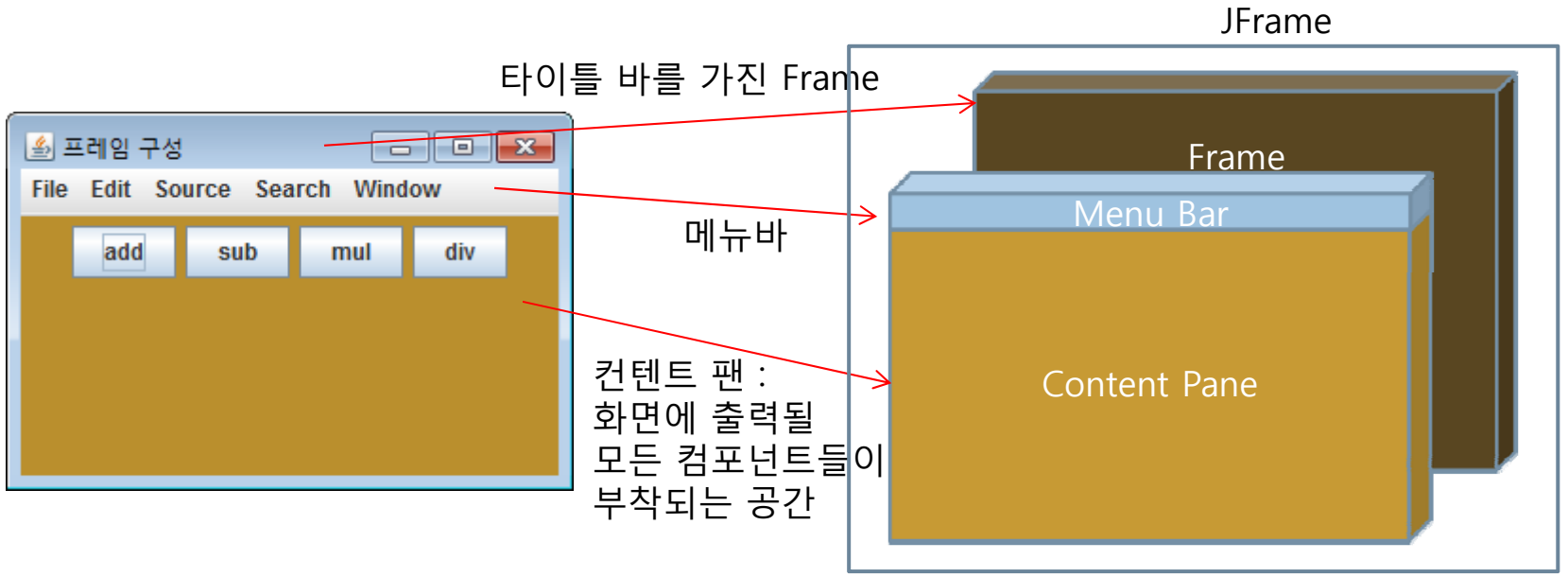
스윙 GUI 프로그램 만들기

1. 프레임 만들기
2. 프레임에 스윙 컴포넌트 붙이기
3. main() 메소드 작성

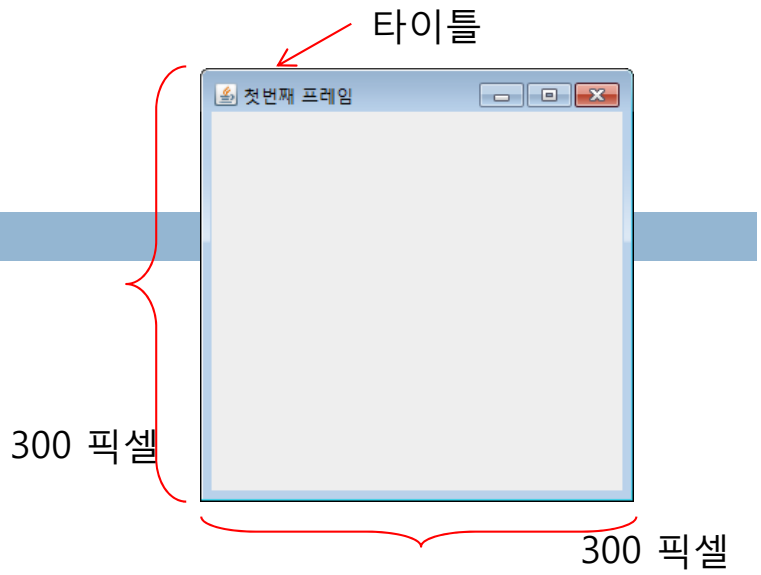
- 스윙 프로그램을 작성하기 위한 import문
  - import java.awt.*; // 그래픽 처리를 위한 클래스들의 경로명
  - import java.awt.event.*; // AWT 이벤트 사용을 위한 경로명
  - import javax.swing.*; // 스윙 컴포넌트 클래스들의 경로명
  - import javax.swing.event.*; // 스윙 이벤트를 위한 경로명

스윙 프레임

- 모든 스윙 컴포넌트를 담는 최상위 GUI 컨테이너
  - JFrame을 상속받아 구현
  - 컴포넌트가 화면에 보이려면 스윙 프레임에 부착되어야 함
  - 프레임을 닫으면 프레임 내의 모든 컴포넌트가 보이지 않게 됨
- 스윙 프레임(JFrame) 기본 구성
  - 프레임 – 스윙 프로그램의 기본 틀
  - 메뉴바 – 메뉴들이 부착되는 공간
  - 컨텐트 팬 – GUI 컴포넌트들이 부착되는 공간



프레임 만들기



두 가지 방법

- main() 메소드에서 JFrame 객체를 생성
- 확장성, 융통성 결여

```
import javax.swing.*;

public class MyApp {
    public static void main(String [] args) {
        JFrame f = new JFrame();
        f.setTitle("첫번째 프레임");
        f.setSize(300,300);
        f.setVisible(true);
    }
}
```

방법 1. main() 메소드에서 JFrame 객체 생성

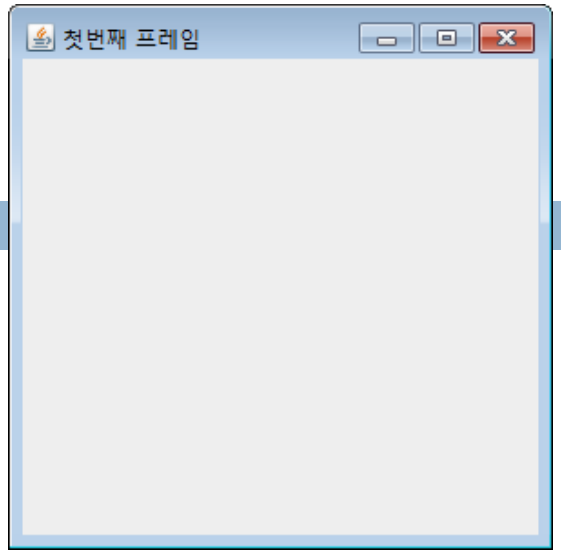
- JFrame 을 상속받은 프레임 클래스 이용
- main() 은 단순히 프레임 객체를 생성하는 역할

```
import javax.swing.*;

public class MyFrame extends JFrame {
    MyFrame() {
        setTitle("첫번째 프레임");
        setSize(300,300);
        setVisible(true);
    }

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

추천

방법 2. JFrame 을 상속받은 프레임 클래스 이용

main()의 위치



추천

```
import javax.swing.*;

public class MyFrame extends JFrame {
    MyFrame() {
        setTitle("첫번째 프레임");
        setSize(300,300);
        setVisible(true);
    }

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

main()을 프레임 클래스 내의 멤버로 작성

```
import javax.swing.*;

class MyFrame extends JFrame {
    MyFrame() {
        setTitle("첫번째 프레임");
        setSize(300,300);
        setVisible(true);
    }
}
public class MyApp {
    public static void main(String [] args) {
        MyFrame mf = new MyFrame();
    }
}
```

main()을 가진 다른 클래스 MyApp 작성

프레임에 컴포넌트 붙이기

타이틀 – 타이틀 바에 부착

```
// JFrame의 생성자 이용
JFrame frame = new JFrame("타이틀문자열");

 // JFrame의 setTitle() 메소드 호출
frame.setTitle("타이틀문자열");
```

컨텐트팬 알아내기

```
JFrame frame = new JFrame();

Container contentPane = frame.getContentPane();
```

스윙 컴포넌트 – 컨텐트 팬에 부착

컨텐트팬에 컴포넌트 달기

```
JFrame frame = new JFrame();
JButton b = new JButton("Click");
Container c = frame.getContentPane();
c.add(b);
```

컨텐트팬 변경

```
JPanel p = new JPanel();
frame.setContentPane(p);
```

Tip. 컨텐트팬에 대한 JDK1.5이후의 변경 사항

- JDK 1.5 이전
  - 프레임의 컨텐트팬을 알아내어 반드시 컨텐트팬에 컴포넌트 부착

```
JFrame frame = new JFrame();
JButton b = new JButton("Click");
Container c = frame.getContentPane();
c.add(b); // 버튼 b를 컨텐트팬에 부착
```

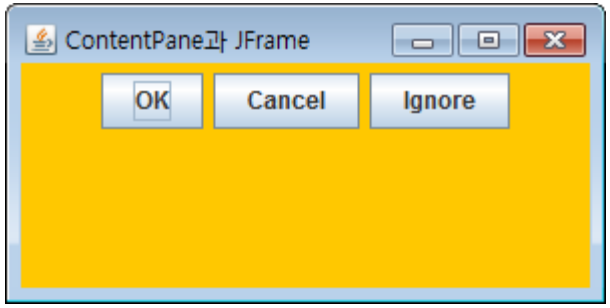
- JDK 1.5 이후
  - 프레임에 컴포넌트를 부착하면 프레임이 대신 컨텐트팬에 부착

```
JFrame frame = new JFrame();
JButton b = new JButton("Click");
frame.add(b); // 컨텐트팬에 대신 버튼 b 부착
```

- 저자의 결론
  - JDK 1.5 이전처럼 명료하게 컨텐트팬에 컴포넌트를 부착하는 것이 바람직함
    - 컨텐트팬에 접근하고 다루는 경우가 많기 때문

예제 9-1 : 컴포넌트를 부착한 프레임 예


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

public class ContentPaneEx extends JFrame {
   ContentPaneEx() {
      setTitle("ContentPane과 JFrame");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      Container contentPane = getContentPane();
      contentPane.setBackground(Color.ORANGE);
      contentPane.setLayout(new FlowLayout());
      contentPane.add(new JButton("OK"));
      contentPane.add(new JButton("Cancel"));
      contentPane.add(new JButton("Ignore"));

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

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



스윙 응용프로그램의 종료

- 응용프로그램 내에서 스스로 종료

```
System.exit(0);
```

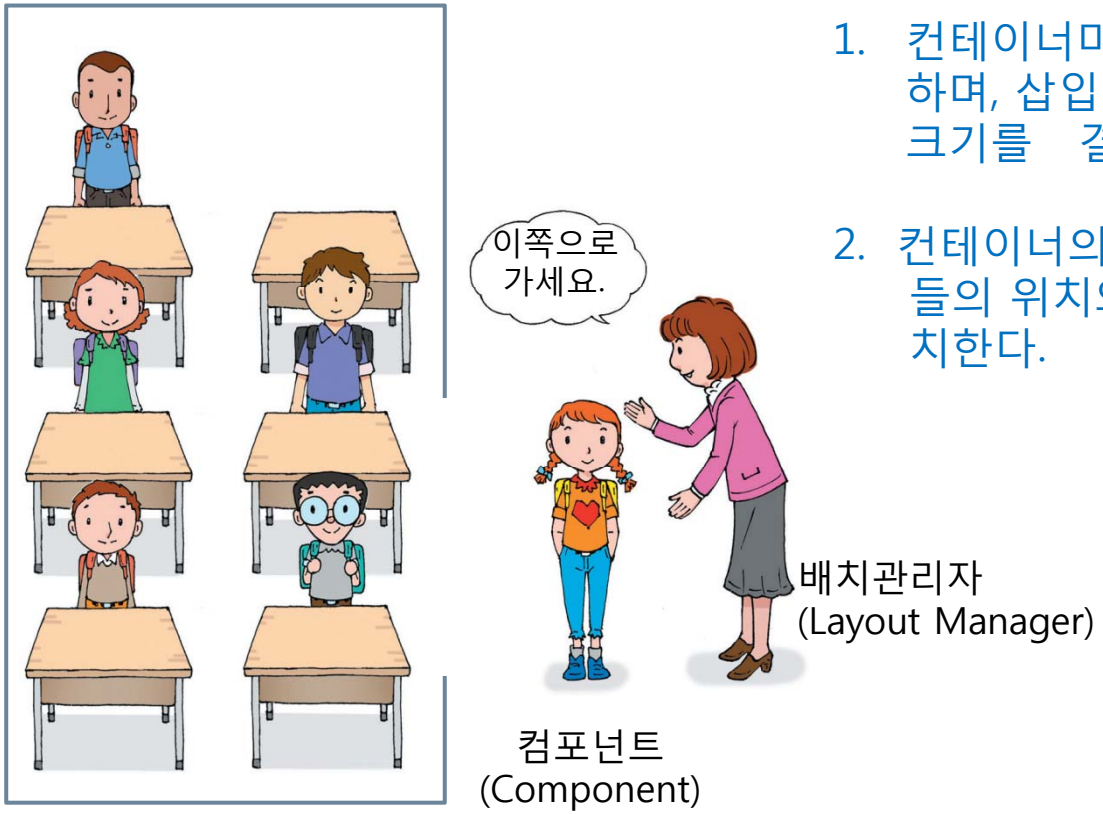
  - 언제 어디서나 무조건 종료
- **프레임 종료버튼(X)이 클릭되면 어떤 일이 일어나는가?**
  - 프레임을 종료하여 프레임 윈도우가 닫힘
    - 프레임이 화면에서 보이지 않게 되고 응용프로그램이 사라짐
  - 프레임이 보이지 않게 되지만 응용프로그램이 종료한 것 아님
    - 키보드나 마우스 입력을 받지 못함
    - 다시 setVisible(true)를 호출하면 보이게 되고 이전 처럼 작동함
- 프레임 종료버튼이 클릭될 때 프레임을 닫고 응용 프로그램이 종료하도록 하는 방법

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

main() 종료 뒤에도 프레임이 살아 있는 이유?

- 스윙 프로그램이 실행되는 동안 생성되는 스레드
  - 메인 스레드
    - main()을 실행하는 스레드
    - 자바 응용프로그램의 실행을 시작한 스레드
  - 이벤트 처리 스레드
    - 스윙 응용프로그램이 실행될 때 자동으로 실행되는 스레드
    - 이벤트 처리 스레드의 역할
      - 프레임과 버튼 등 GUI 화면 그리기
      - 키나 마우스 입력을 받아 이벤트를 처리할 코드 호출
- 자바 응용프로그램의 종료 조건
  - 실행 중인 사용자 스레드가 하나도 없을 때 종료
- 스윙 프로그램 main() 종료 뒤 프레임이 살아있는 이유
  - 메인 스레드가 종료되어도 이벤트 처리 스레드가 살아 있어 프레임 화면을 그리고 마우스나 키 입력을 받기 때문

컨테이너와 배치 개념



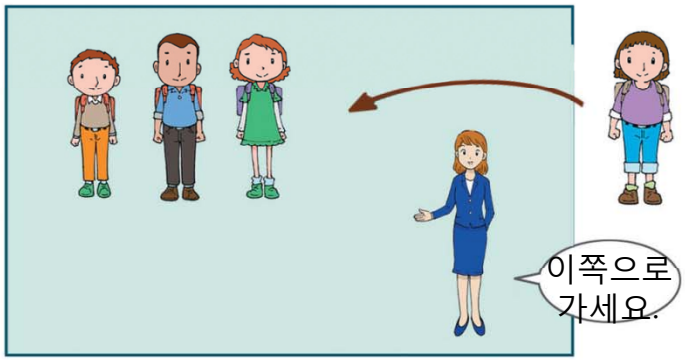
1. 컨테이너마다 하나의 배치관리자가 존재하며, 삽입되는 모든 컴포넌트의 위치와 크기를 결정하고 적절히 배치한다.

2. 컨테이너의 크기가 변하면 내부 컴포넌트들의 위치와 크기를 모두 재조절하고 재배치한다.

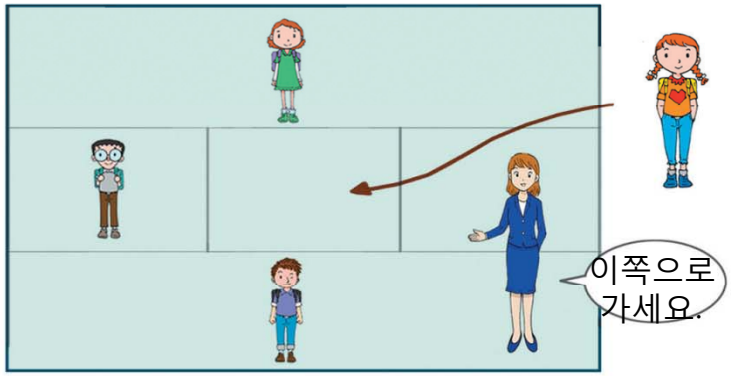
배치 관리자 대표 유형 4 가지

- java.awt 패키지에 구현되어 있음

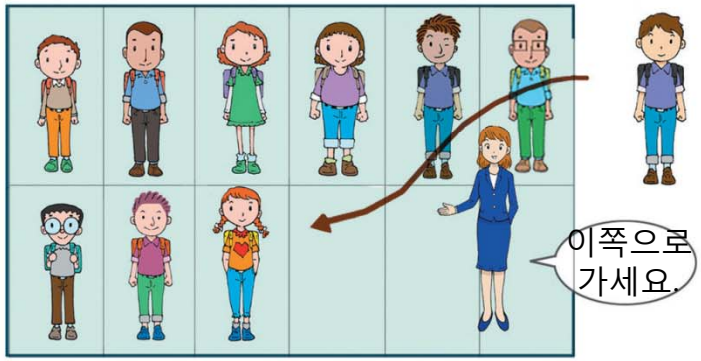
FlowLayout



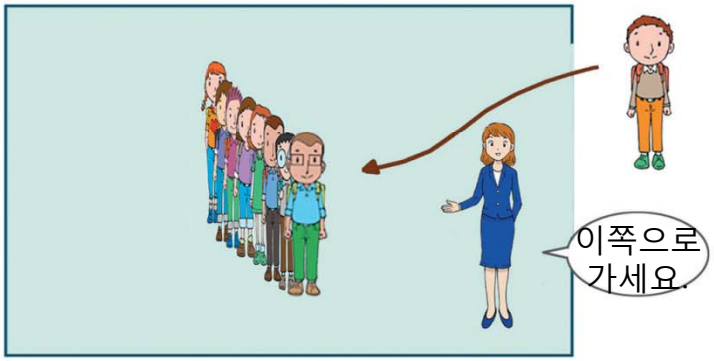
BorderLayout



GridLayout



CardLayout



컨테이너와 배치관리자

- 컨테이너의 디폴트 배치관리자
  - 컨테이너는 생성시 디폴트 배치관리자 설정

AWT와 스윙 컨테이너	디폴트 배치관리자
Window, JWindow	BorderLayout
Frame, JFrame	BorderLayout
Dialog, JDialog	BorderLayout
Panel, JPanel	FlowLayout
Applet, JApplet	FlowLayout

- 컨테이너에 새로운 배치관리자 설정
  - Container.setLayout(LayoutManager lm)
    - lm을 새로운 배치관리자로 설정

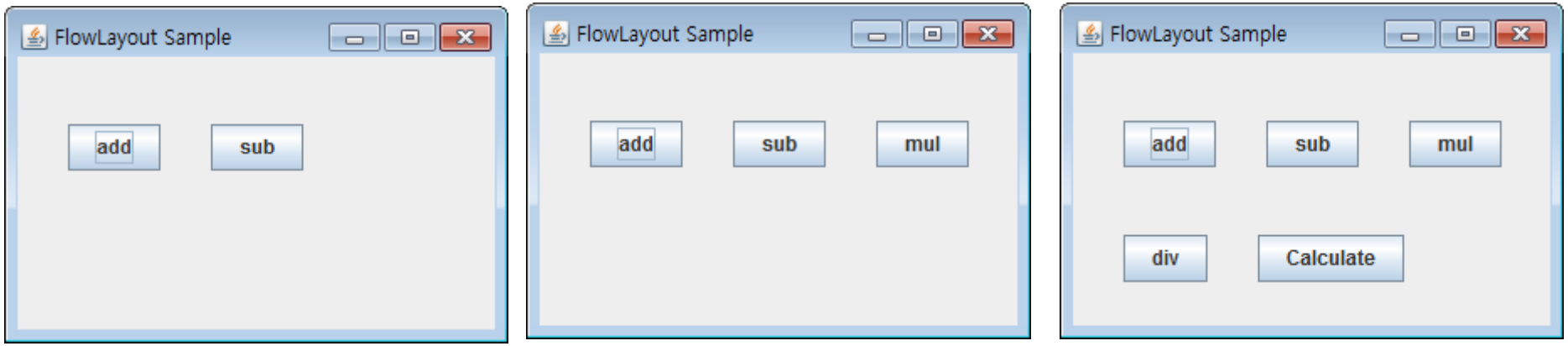
```
// JPanel 패널에 BorderLayout 배치관리자를 설정하는 예

JPanel p = new JPanel();
p.setLayout(new BorderLayout());
```

```
JFrame frame = new JFrame();
Container c = frame.getConentPane(); // 프레임의 컨텐트팬
c.setLayout(new FlowLayout()); // 컨텐트팬에 FlowLayout 설정

혹은

frame.setLayout(new FlowLayout()); // JDK 1.5 이후 버전에서
```

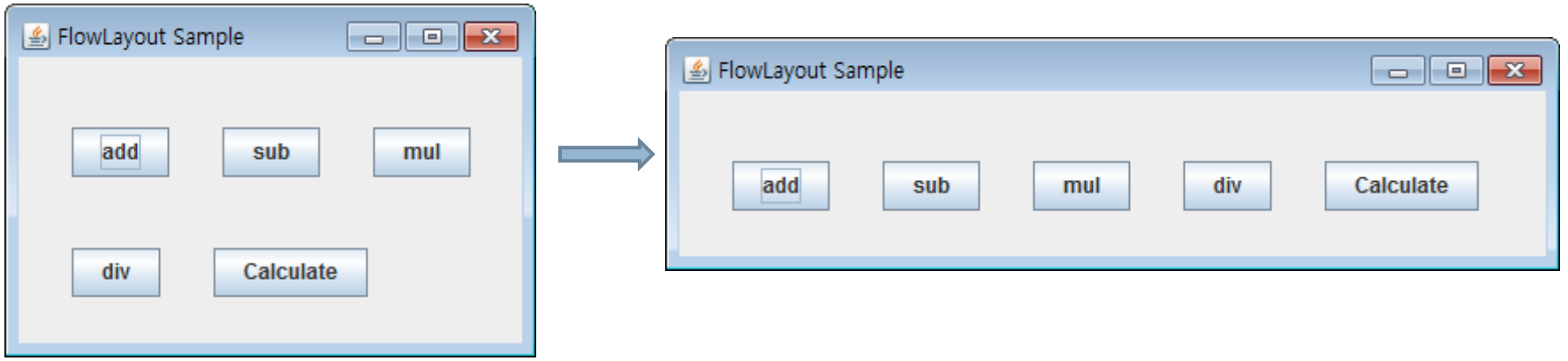
FlowLayout

- 배치방법
  - 컨테이너 공간 내에 왼쪽에서 오른쪽으로 배치
    - 다시 위에서 아래로 순서대로 컴포넌트를 배치한다.

```
container.setLayout(new FlowLayout());
container.add(new JButton("add"));
container.add(new JButton("sub"));
container.add(new JButton("mul"));
container.add(new JButton("div"));
container.add(new JButton("Calculate"));
```



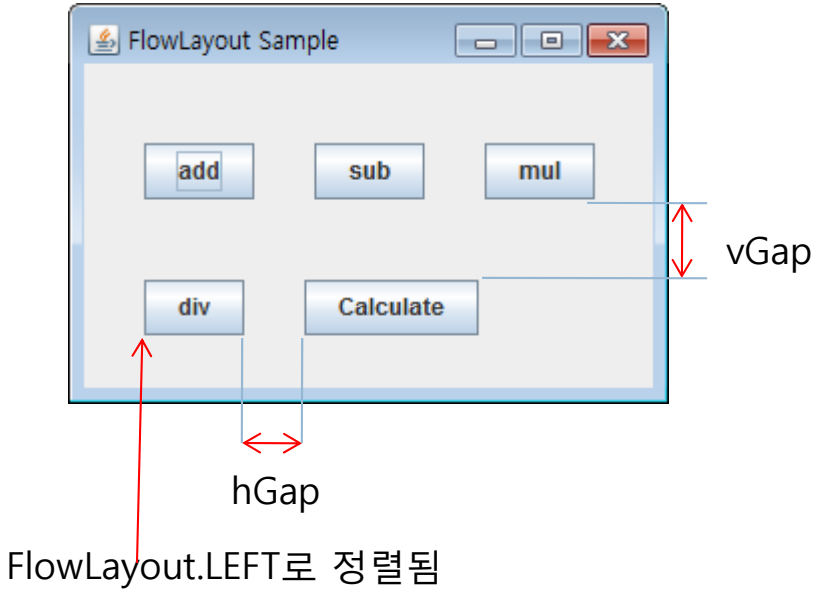
- 컨테이너의 크기가 변하면 배치 관리자에 의해 재배치됨



프레임의 크기를 바꾸면 배치도 변한다.

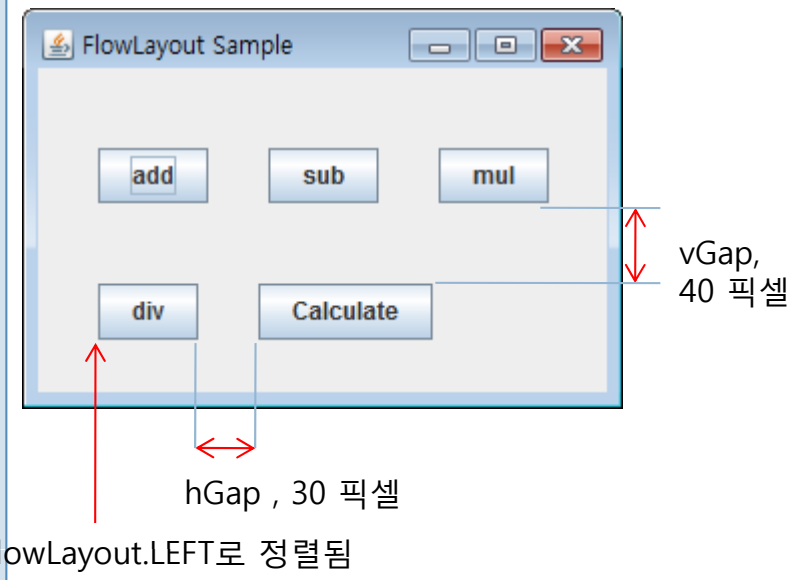
FlowLayout - 생성자와 속성

- 생성자
  - FlowLayout()
  - FlowLayout(int align)
  - FlowLayout(int align, int hGap, int vGap)
    - align : 컴포넌트의 정렬(5 가지중 많이 사용되는 3 가지)
      - *FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER(디폴트)*
    - hGap : 좌우 두 컴포넌트 사이의 수평 간격, 픽셀 단위(디폴트 : 5)
    - vGap : 상하 두 컴포넌트 사이의 수직 간격, 픽셀 단위(디폴트 : 5)



예제 9-2 : LEFT로 정렬되는 수평 간격이 30 픽셀, 수직 간격이 40 픽셀인 FlowLayout 사용 예

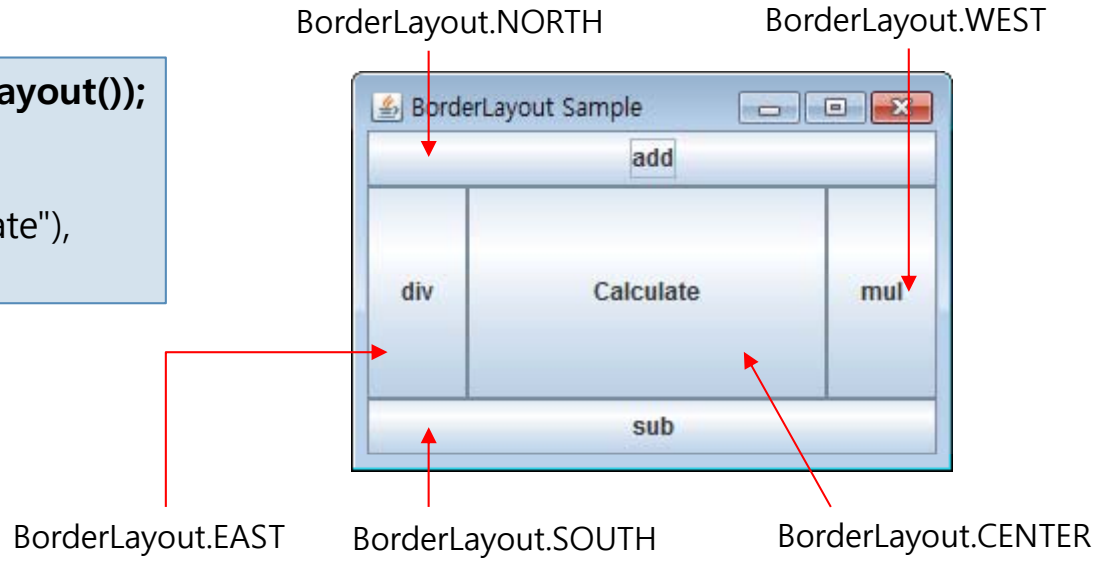

```
import javax.swing.*;
import java.awt.*;

public class FlowLayoutEx extends JFrame {
   FlowLayoutEx() {
      setTitle("FlowLayout Sample");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      setLayout(new FlowLayout(FlowLayout.LEFT, 30, 40));
      add(new JButton("add"));
      add(new JButton("sub"));
      add(new JButton("mul"));
      add(new JButton("div"));
      add(new JButton("Calculate"));

      setSize(300, 250);
      setVisible(true);
   }

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



BorderLayout

- 배치방법
  - 컨테이너 공간을 5 구역으로 분할, 배치
    - East, West, South, North, Center
  - 배치 방법
    - add(Component comp, int index)
      - *comp를 index의 공간에 배치*
  - 컨테이너의 크기가 변하면 재배치

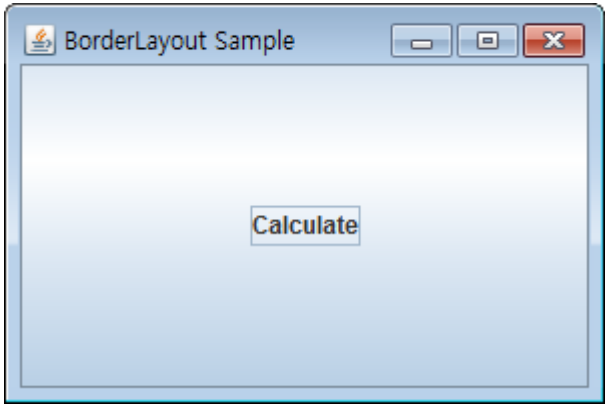
```
container.setLayout(new BorderLayout());
container.add(new JButton("div"),
              BorderLayout.WEST);
container.add(new JButton("Calculate"),
              BorderLayout.CENTER);
```



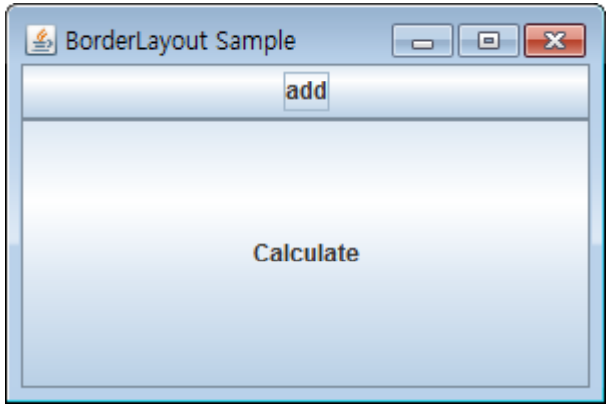
BorderLayout 생성자와 속성

- 생성자
  - BorderLayout()
  - BorderLayout(int hGap, int vGap)
    - hGap : 좌우 두 컴포넌트 사이의 수평 간격, 픽셀 단위(디폴트 : 0)
    - vGap : 상하 두 컴포넌트 사이의 수직 간격, 픽셀 단위(디폴트 : 0)

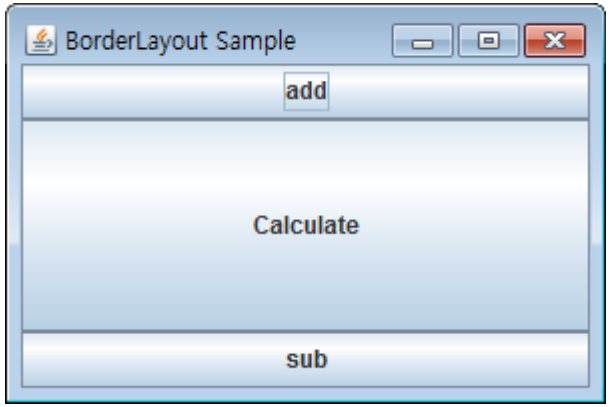
BorderLayout의 사용예



CENTER에 컴포넌트가 삽입될 때

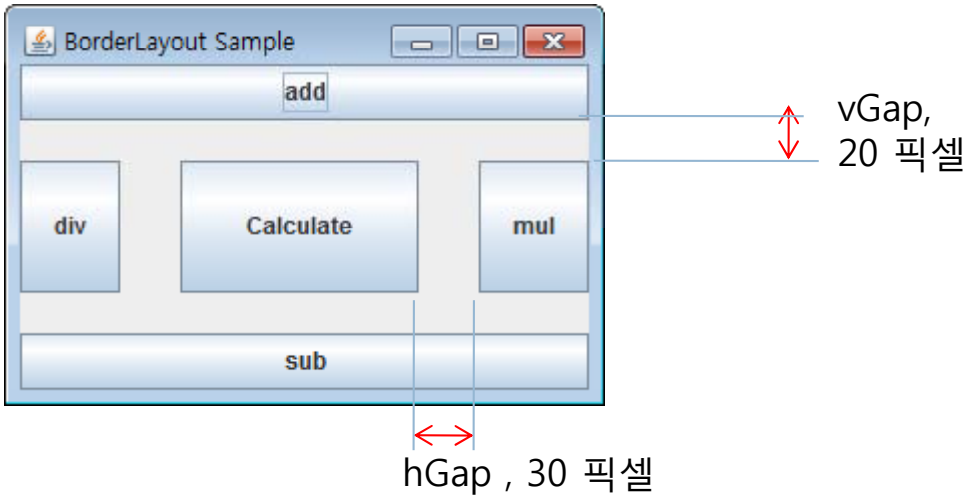


CENTER와 NORTH에 컴포넌트가 삽입될 때



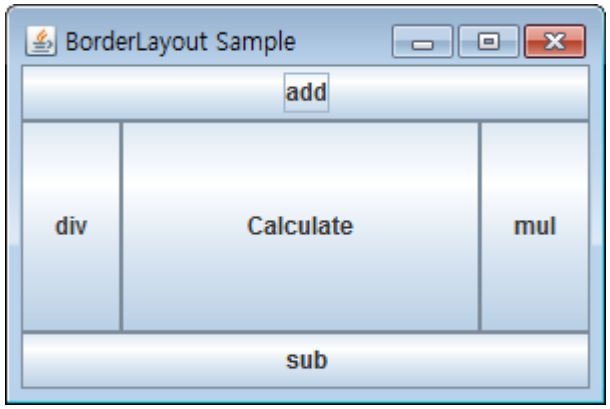
CENTER, NORTH, SOUTH에 컴포넌트가 삽입될 때

new BorderLayout(30,20);
으로 배치관리자를
생성하였을 때



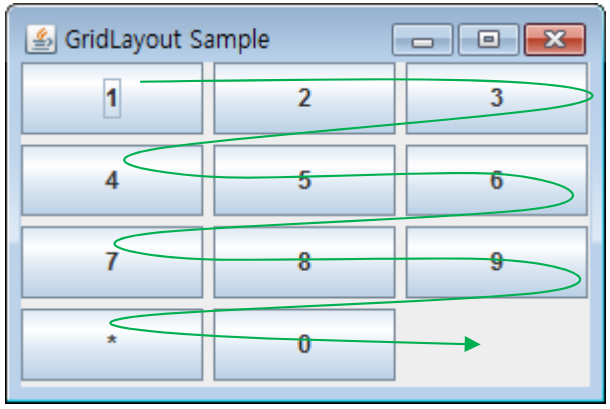
예제 9-3 : BorderLayout 사용 예


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

public class BorderLayoutEx extends JFrame {
   BorderLayoutEx() {
      setTitle("BorderLayout Sample");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      setLayout(new BorderLayout());
      add(new JButton("add"), BorderLayout.NORTH);
      add(new JButton("sub"), BorderLayout.SOUTH);
      add(new JButton("mul"), BorderLayout.EAST);
      add(new JButton("div"), BorderLayout.WEST);
      add(new JButton("Calculate"), BorderLayout.CENTER);

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

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



GridLayout

- 배치방법
  - 컨테이너 공간을 동일한 사각형 격자(그리드)로 분할하고 각 셀에 하나의 컴포넌트 배치
    - 격자 구성은 생성자에 행수와 열수 지정
    - 셀에 왼쪽에서 오른쪽으로, 다시 위에서 아래로 순서대로 배치




```
container.setLayout(new GridLayout(4,3,5,5)); // 4×3 분할로 컴포넌트 배치
container.add(new JButton("1")); // 상단 왼쪽 첫 번째 셀에 버튼 배치
container.add(new JButton("2")); // 그 옆 셀에 버튼 배치
```

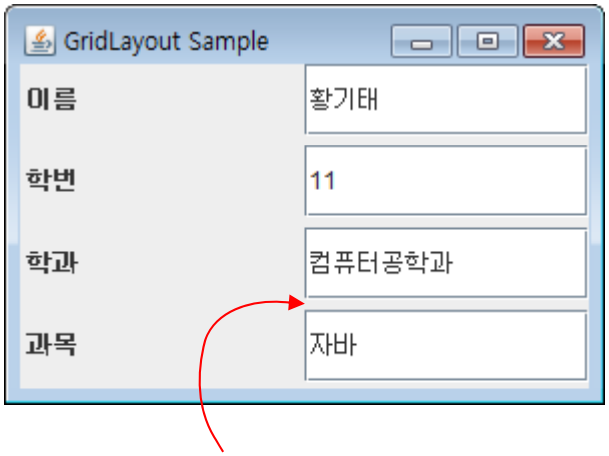
- 4x3 그리드 레이아웃 설정
- 총 11 개의 버튼이 순서대로 add 됨
- 수직 간격 vGap : 5 픽셀
- 수평 간격 hGap : 5 픽셀

  - 컨테이너의 크기가 변하면 재배치
    - 크기 재조정

GridLayout 생성자와 속성

- 생성자
  - GridLayout()
  - GridLayout(int rows, int cols)
  - GridLayout(int rows, int cols, int hGap, int vGap)
    - rows : 격자의 행수 (디폴트 : 1)
    - cols : 격자의 열수 (디폴트 : 1)
    - hGap : 좌우 두 컴포넌트 사이의 수평 간격, 픽셀 단위(디폴트 : 0)
    - vGap : 상하 두 컴포넌트 사이의 수직 간격, 픽셀 단위(디폴트 : 0)
    - rows x cols 만큼의 셀을 가진 격자로 컨테이너 공간을 분할, 배치

예제 9-4 : GridLayout으로 입력 폼 만들기



두 행 사이의 수직 간격
vGap이 5 픽셀로 설정됨

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

public class GridLayoutEx extends JFrame {
    GridLayoutEx() {
        setTitle("GridLayout Sample");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        GridLayout grid = new GridLayout(4, 2);
        grid.setVgap(5);
        setLayout(grid);
        add(new JLabel(" 이름"));
        add(new JTextField(""));
        add(new JLabel(" 학번"));
        add(new JTextField(""));
        add(new JLabel(" 학과"));
        add(new JTextField(""));
        add(new JLabel(" 과목"));
        add(new JTextField(""));

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

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

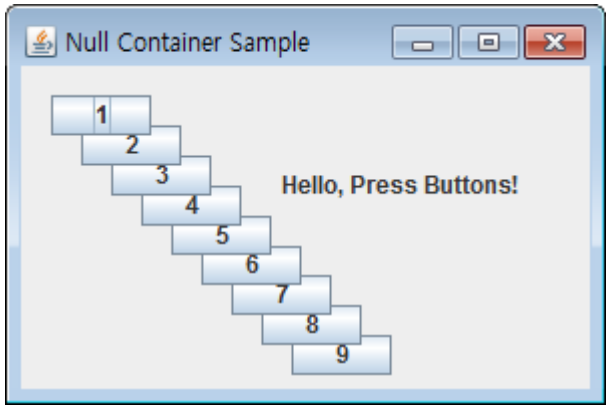
배치관리자 없는 컨테이너

- 배치관리자가 없는 컨테이너 개념
  - 응용프로그램에서 컴포넌트의 절대 크기와 절대 위치 결정
- 용도
  - 컴포넌트의 크기나 위치를 개발자 임의로 결정하고자 하는 경우
  - 게임 프로그램과 같이 시간이나 마우스/키보드의 입력에 따라 컴포넌트들의 위치와 크기가 수시로 변하는 경우
  - 여러 컴포넌트들이 서로 겹쳐 출력하고자 하는 경우
- 컨테이너의 배치 관리자 제거 방법
  - container.setLayout(null);

```
// JPanel의 배치관리자를 삭제하는 예

JPanel p = new JPanel();
p.setLayout(null);
```

- 컴포넌트의 크기와 위치 설정
  - 프로그램 내에서 이루어져야 함
  - 컴포넌트들이 서로 겹치게 할 수 있음
  - 다음 메소드 이용
    - 컴포넌트 크기 설정 : component.setSize(int width, int height);
    - 컴포넌트 위치 설정 : component.setLocation(int x, int y);
    - 컴포넌트 위치와 크기 동시 설정 : component.setBounds(int x, int y, int width, int height);

예제 9-5 : 배치관리자 없는 컨테이너에 컴포넌트 위치와 크기를 절대적으로 지정



원하는 위치에 원하는 크기로
컴포넌트를 마음대로
배치할 수 있다.

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

public class NullContainerEx extends JFrame {
    NullContainerEx() {
        setTitle("Null Container Sample");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);

        JLabel la = new JLabel("Hello, Press Buttons!");
        la.setLocation(130, 50);
        la.setSize(200, 20);
        add(la);
        for(int i=1; i<=9; i++) {
            JButton b = new JButton(Integer.toString(i));
            b.setLocation(i*15, i*15);
            b.setSize(50, 20);
            add(b);
        }
        setSize(300, 200);
        setVisible(true);
    }

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