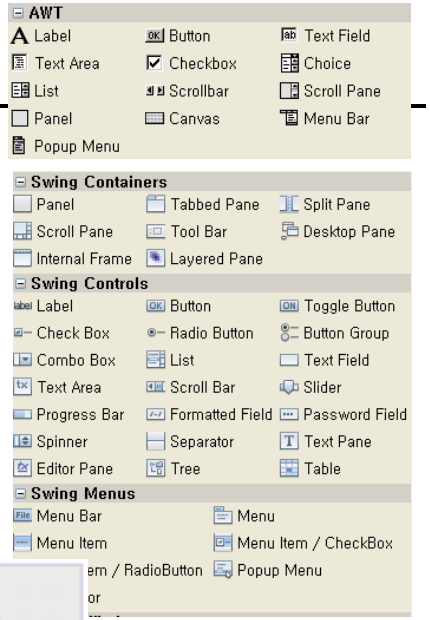
자바 GUI & 이벤트 처리

514770-1
2017년 봄학기
4/19/2017
박경신

자바에서 GUI의 종류

- AWT(Abstract Windows Toolkit)
  - 운영 체제가 제공하는 자원을 이용하여서 컴포넌트를 생성한다.
- SWING
  - 스윙 컴포넌트가 자바로 작성되어 있기 때문에 어떤 플랫폼에서도 일관된 화면을 보여줄 수 있다.



AWT

- AWT(Abstract Windows Toolkit)
  - 자바가 처음 나왔을 때 함께 배포된 GUI 라이브러리
  - java.awt 패키지
  - 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

SWING

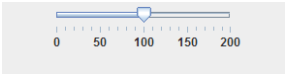
- Swing(스윙)
  - AWT 기술을 기반으로 작성된 자바 라이브러리
    - 모든 AWT 기능 + 추가된 풍부하고 화려한 고급 컴포넌트
    - AWT 컴포넌트에 J자가 덧붙여진 이름의 클래스
    - 그 외 J 자로 시작하는 클래스
  - 순수 자바 언어로 구현, JDK 1.1 부터 - javax.swing 패키지
  - Swing 컴포넌트는 native(peer 혹은 운영체제)에 의존하지 않음
    - 경량 컴포넌트(Light weight components)

SWING Components

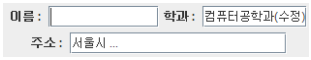
JButton

JCheckBox

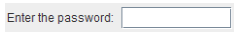
JRadioButton



JSlider



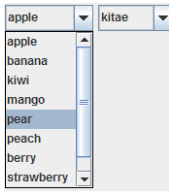
JTextField



JPasswordField

JSpinner

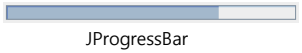
JTextArea



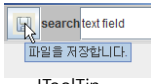
JComboBox

JList

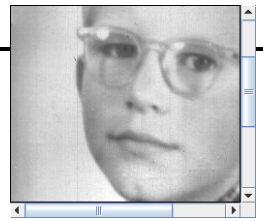
SWING Components



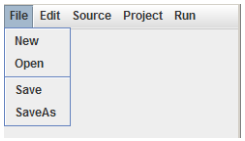
JProgressBar



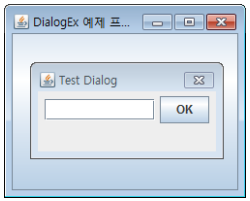
JToolTip



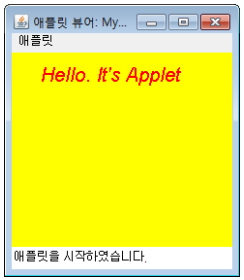
JScrollPane



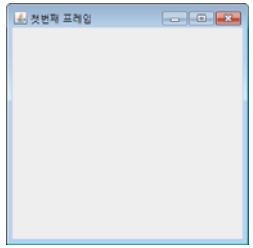
JMenu



JDialog



JApplet

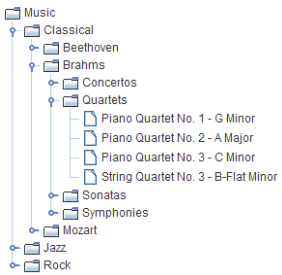


JFrame

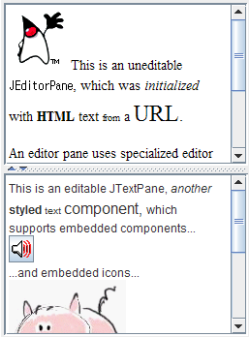
SWING Components

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



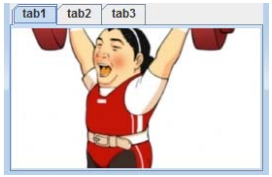
JTree



JEditorPane and JTextPane

SWING Components

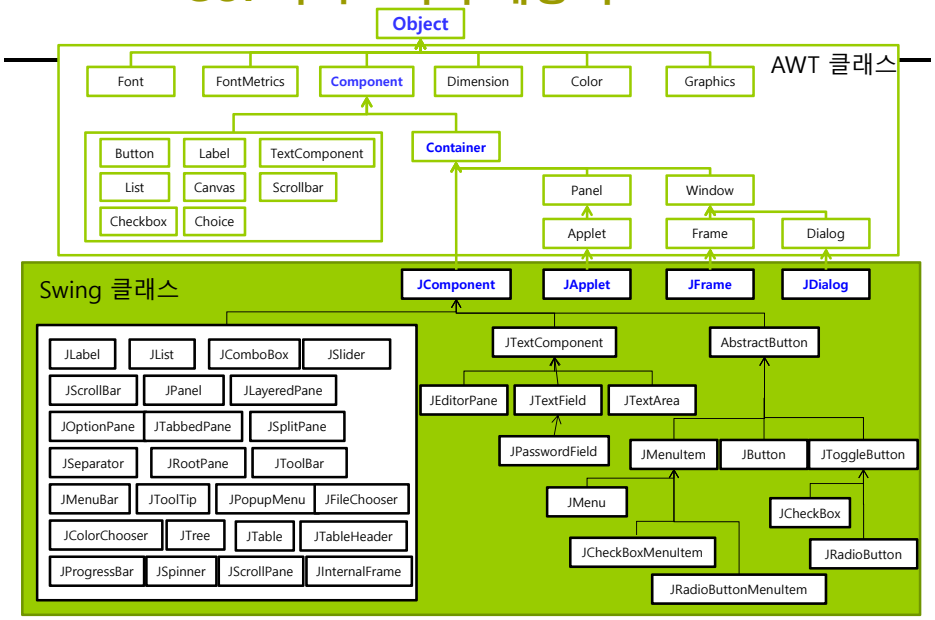
JToolBar



JTabbedPane

JSplitPane

GUI 라이브러리 계층 구조



Swing 클래스의 특징

- 클래스 이름이 J 자로 시작
- 화려하고 다양한 컴포넌트로 쉽게 GUI 프로그래밍
- 스윙 컴포넌트는 2 가지 유형
  - JComponent는 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
    - JComponent() 인스턴스를 생성할 수 없음
  - AWT의 Component를 상속받음

컨테이너와 컴포넌트

- **기본 컴포넌트**
  - 컨테이너에 포함되어야 화면에 출력될 수 있는 GUI 객체
    - JButton, JLabel, JCheckbox, JChoice, JList, JMenu, JTextField, JScrollbar, JTextArea, JCanvas 등이 여기에 속한다.
  - 모든 GUI 컴포넌트의 최상위 클래스
    - java.awt.Component
  - 스윙 컴포넌트의 최상위 클래스
    - javax.swing.JComponent
- **컨테이너 컴포넌트**
  - 다른 컴포넌트를 안에 포함할 수 있는 컴포넌트
    - JFrame, JDialog, JApplet, JPanel, JScrollPane 등이 여기에 속한다.

컨테이너의 종류

- **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
- **일반 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

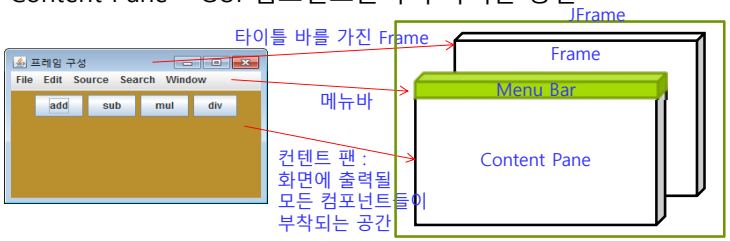
스윙 GUI 프로그램 만들기

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

JFrame 클래스

- 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
  - Frame – 스윙 프로그램의 기본 틀
  - Menu – 메뉴들이 부착되는 공간
  - Content Pane – GUI 컴포넌트들이 부착되는 공간

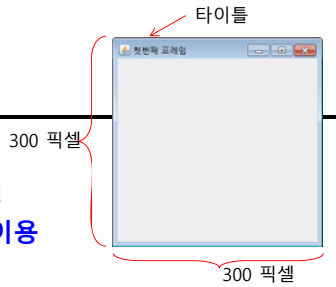


JFrame 클래스

- *setLocation(x, y) , setBounds(x, y, width, height), setSize(width, height)*
  - 프레임의 위치와 크기를 설정한다.
- *setIconImage(IconImage)*
  - 윈도우 시스템에 타이틀 바, 태스크 스위처에 표시할 아이콘을 알려준다.
- *setTitle()*
  - 타이틀 바의 제목을 변경한다.
- *setResizable(boolean)*
  - 사용자가 크기를 조절할 수 있는지를 설정한다.

프레임 만들기

- 프레임 만들기 두 가지 방법
  1. main() 메소드에서 JFrame 객체 생성
  2. **JFrame을 상속받은 프레임 클래스 이용**



방법 1. main() 메소드에서 JFrame 객체 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 2. JFrame 을 상속받은 프레임 클래스 이용
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

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);
```

스윙 응용프로그램의 종료

- 응용프로그램 내에서 스스로 종료

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

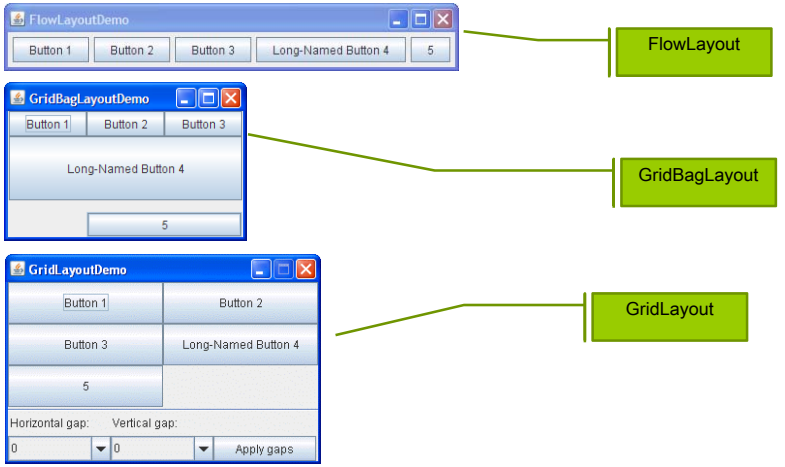
  - 언제 어디서나 무조건 종료
- **프레임 종료버튼(X)이 클릭되면 어떤 일이 일어나는가?**
  - 프레임을 종료하여 프레임 윈도우가 닫힘
    - 프레임이 화면에서 보이지 않게 되고 응용프로그램이 사라짐
  - 프레임이 보이지 않게 되지만 응용프로그램이 종료한 것 아님
    - 키보드나 마우스 입력을 받지 못함
    - 다시 setVisible(true)를 호출하면 보이게 되고 이전 처럼 작동함
- 프레임 종료버튼이 클릭될 때 프레임을 닫고 응용 프로그램이 종료하도록 하는 방법

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

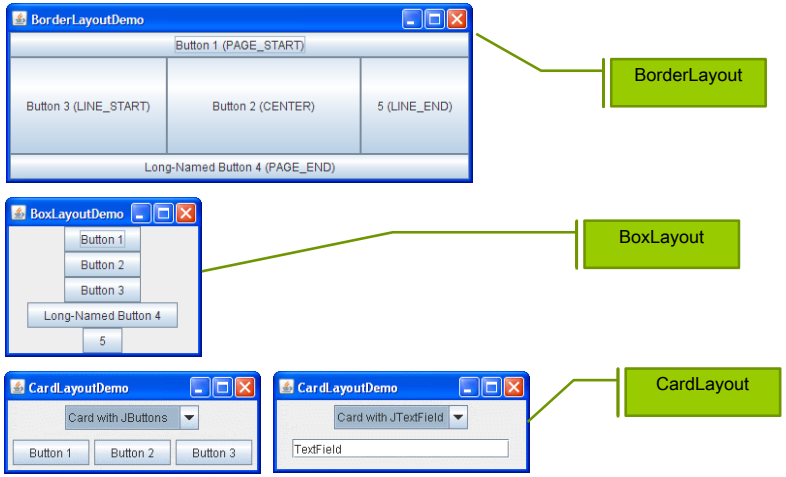
main() 종료 뒤에도 프레임이 살아 있는 이유?

- 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

배치 관리자(Layout)의 종류



배치 관리자(Layout)의 종류



배치 관리자의 설정

- 배치관리자 설정 방법
  1. 생성자를 이용하는 방법

     JPanel panel = new JPanel(new BorderLayout());
  2. setLayout() 메소드 이용

     panel.setLayout(new FlowLayout());
- 프로그래머가 컴포넌트의 크기와 힌트를 배치 관리자에게 주고 싶은 경우에는 setMinimumSize(), setPreferredSize(), setMaximumSize() 메소드를 사용
  - button.setMaximumSize(**new** Dimension(300, 200)); // 최대 크기 힌트
  - button.setAlignmentX(JComponent.*CENTER_ALIGNMENT*); // 중앙 정렬 힌트

컨테이너와 기본 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

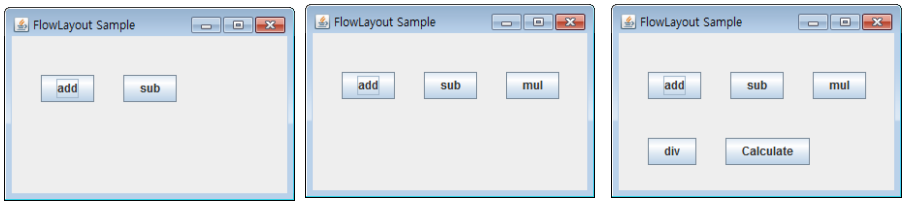
```
// JPanel 패널에 BorderLayout 설정
JPanel p = new JPanel();
p.setLayout(new BorderLayout());
```

```
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

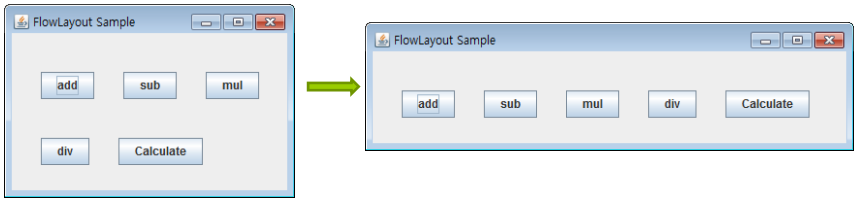
- 배치방법
  - 컨테이너 공간 내에 왼쪽에서 오른쪽으로 배치
    - 다시 위에서 아래로 순서대로 컴포넌트를 배치한다.

```
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"));
```



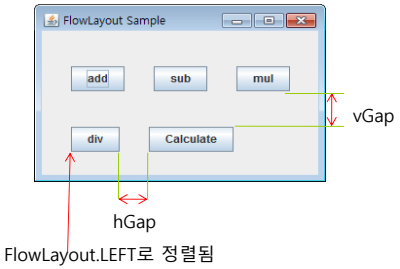
FlowLayout

- 컨테이너의 크기가 변화면 배치 관리자에 의해서 재배치
  - 프레임의 크기를 바꾸면 배치도 변한다.



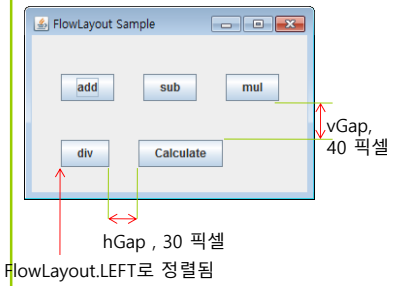
FlowLayout

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

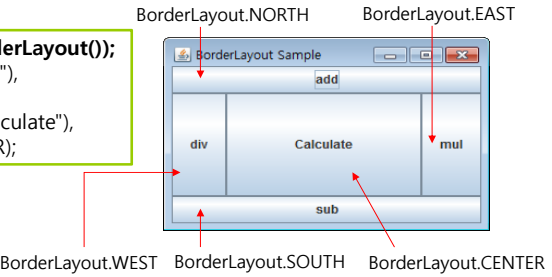
```
import javax.swing.*;
import java.awt.*;

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

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

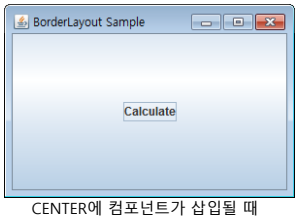


BorderLayout

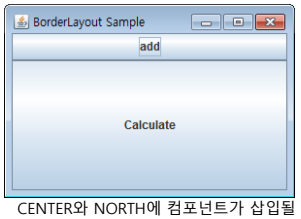
- 생성자
  - BorderLayout()
  - BorderLayout(int hGap, int vGap)
    - hGap : 좌우 두 컴포넌트 사이의 수평 간격, 픽셀 단위(디폴트 : 0)
    - vGap : 상하 두 컴포넌트 사이의 수직 간격, 픽셀 단위(디폴트 : 0)

생성자 또는 메소드	설명
BorderLayout(int hgap, int vgap)	컴포넌트 사이의 수평 간격 hgap과 수직 간격 vgap을 을 가지는 BorderLayout 객체 생성
setHgap(int)	컴포넌트 사이의 수평 간격 설정(단위는 픽셀)
setVgap(int)	컴포넌트 사이의 수직 간격 설정

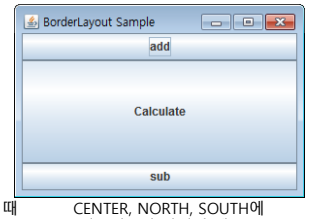
BorderLayout의 사용예



CENTER에 컴포넌트가 삽입될 때

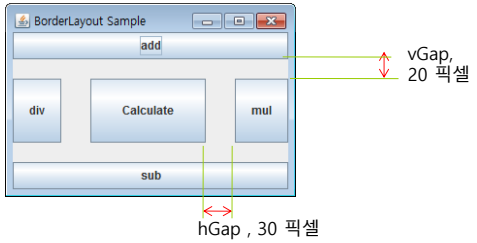


CENTER와 NORTH에 컴포넌트가 삽입될 때



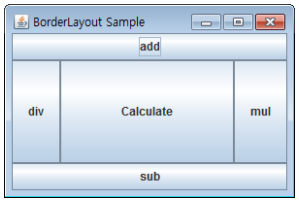
CENTER, NORTH, SOUTH에 컴포넌트가 삽입될 때

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



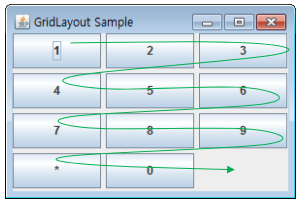
예제 : BorderLayout 사용 예

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

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

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




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

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

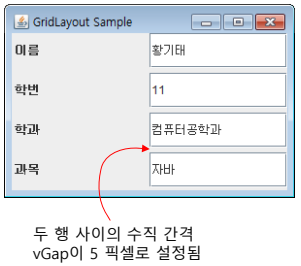
  - 컨테이너의 크기가 변하면 재배치
    - 크기 재조정

GridLayout

- 생성자
  - 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

생성자	설명
GridLayout(int rows, int cols)	rows 행과 cols 열을 가지는 GridLayout 객체를 생성한다. 만약 rows나 cols가 0이면 필요한 만큼의 행이나 열이 만들어진다.
GridLayout(int rows, int cols, int hgap, int vgap)	rows 행과 cols 열을 가지는 GridLayout 객체를 생성한다. hgap과 vgap은 컴포넌트 사이의 수평 간격과 수직 간격으로 단위는 픽셀이다.

예제 : GridLayout으로 입력 폼 만들기




```
import javax.swing.*;
import java.awt.*;
public class GridLayoutEx extends JFrame {
   GridLayoutEx() {
      setTitle("GridLayout Sample");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      GridLayout grid = new GridLayout(4, 2);
      grid.setVgap(5);
      c.setLayout(grid);
      c.add(new JLabel(" 이름"));
      c.add(new JTextField(""));
      c.add(new JLabel(" 학번"));
      c.add(new JTextField(""));
      c.add(new JLabel(" 학과"));
      c.add(new JTextField(""));
      c.add(new JLabel(" 과목"));
      c.add(new JTextField(""));
      setSize(300, 200);
      setVisible(true);
   }
   public static void main(String[] args) {
      new GridLayoutEx();
   }
}
```

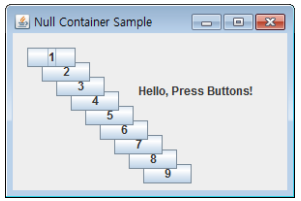
배치관리자 없는 컨테이너

- **배치관리자가 없는 컨테이너 개념**
  - 응용프로그램에서 컴포넌트의 절대 크기와 절대 위치 결정
  - 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 크기와 위치 설정을 해야 함
  - 컴포넌트들이 서로 겹치게 할 수 있음
    - 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 : 배치관리자 없는 컨테이너에 컴포넌트 위치와 크기를 절대적으로 지정



원하는 위치에 원하는 크기로 컴포넌트를 마음대로 배치할 수 있다.

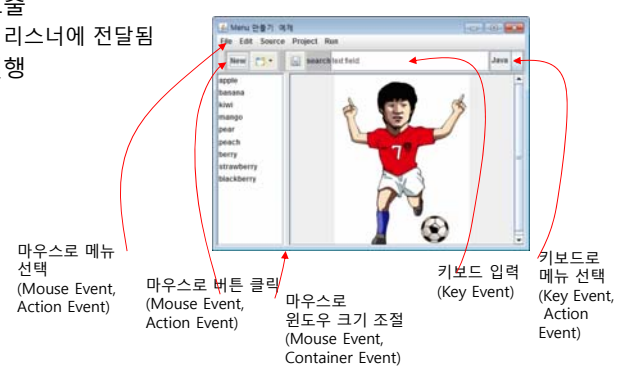
```
import javax.swing.*;
import java.awt.*;
public class NullContainerEx extends JFrame {
   NullContainerEx() {
      setTitle("Null Container Sample");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container c = getContentPane();
      c.setLayout(null);
      JLabel la = new JLabel("Hello, Press Buttons!");
      la.setLocation(130, 50);
      la.setSize(200, 20);
      c.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);
         c.add(b);
      }
      setSize(300, 200);
      setVisible(true);
   }
   public static void main(String[] args) {
      new NullContainerEx();
   }
}
```

이벤트 기반 프로그래밍

- **이벤트-구동 프로그래밍(event-driven programming):**
  - 프로그램의 실행이 이벤트의 발생에 의하여 결정되는 방식
    - 이벤트가 발생하면 이벤트를 처리하는 루틴(이벤트 리스너) 실행
    - 프로그램 내의 어떤 코드가 언제 실행될 지 아무도 모름, 이벤트의 발생에 의해 전적으로 결정
  - 반대되는 개념 : 배치 실행(batch programming)
    - 프로그램의 개발자가 프로그램의 흐름을 결정하는 방식
- 이벤트 종류
  - 마우스 드래그, 마우스 클릭, 키보드 누름 등 사용자 입력
  - 센서로부터의 입력, 네트워크로부터 데이타 송수신
  - 다른 응용프로그램이나 다른 스레드로부터의 메시지
- 이벤트 기반 프로그램의 구조
  - 이벤트 처리 리스너 들의 집합

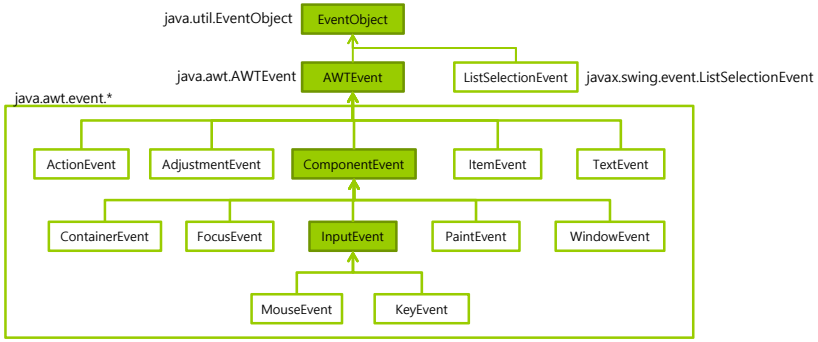
이벤트 처리 순서

- 이벤트 처리 순서
  - 이벤트 발생(예 :마우스나 키보드의 움직임 혹은 입력)
  - 이벤트 객체 생성
    - 현재 발생한 이벤트에 대한 정보를 가진 객체
  - 이벤트 리스너 찾기
  - 이벤트 리스너 호출
    - 이벤트 객체가 리스너에 전달됨
  - 이벤트 리스너 실행



이벤트 객체

- 이벤트 객체란?
  - 이벤트가 발생할 때, 발생한 이벤트에 관한 정보를 가진 객체
  - 이벤트 리스너에 전달됨
    - 이벤트 리스너 코드에서 이벤트가 발생한 상황을 파악할 수 있게 함
- 이벤트 객체의 종류



이벤트 객체에 포함된 정보

- 이벤트 객체가 포함하는 정보
  - 이벤트 종류, 이벤트 소스
  - 이벤트가 발생한 화면 좌표, 이벤트가 발생한 컴포넌트 내 좌표
  - 버튼이나 메뉴 아이템에 이벤트가 발생한 경우 버튼이나 메뉴 아이템의 문자열
  - 클릭된 마우스 버튼 번호, 마우스의 클릭 횟수
  - 키가 눌러졌다면 키의 코드 값과 문자 값
  - 체크박스, 라디오버튼 등과 같은 컴포넌트에 이벤트가 발생하였다면 체크 상태
- 이벤트에 따라 조금씩 다른 정보 포함
  - **ActionEvent** 객체 : 액션 문자열
  - **MouseEvent** 객체 : 마우스의 위치 정보, 마우스 버튼, 함께 눌러진 키 정보 등
  - **ItemEvent** 객체 : 아이템의 체크 상태
- 이벤트 소스 알아 내기
  - Object EventObject.getSource()
    - 발생한 이벤트의 소스 컴포넌트 리턴
    - Object 타입으로 리턴하므로 캐스팅하여 사용
    - 모든 이벤트 객체에 대해 적용

이벤트 객체와 이벤트 소스

이벤트 객체	이벤트 소스	이벤트가 발생하는 경우
ActionEvent	JButton	마우스나 키로 버튼 선택
	JList	리스트 아이템을 더블클릭하여 리스트 아이템 선택
	JMenuItem	메뉴 아이템 선택
	JTextField	텍스트 입력 중 〈Enter〉 키 입력
ItemEvent	JCheckBox	체크박스의 선택 혹은 해제
	JRadioButton	라디오버튼의 선택 상태가 변할 때
	JCheckBoxMenuItem	체크박스 메뉴 아이템의 선택 혹은 해제
	JList	리스트 아이템 선택
KeyEvent	Component	키가 눌러지거나 놀러진 키가 떼어질 때
MouseEvent	Component	마우스 버튼이 눌러지거나 떼어질 때, 마우스 버튼이 클릭될 때, 컴포넌트 위에 마우스가 올라갈 때, 올라간 마우스가 내려올 때, 마우스가 드래그될 때, 마우스가 단순히 움직일 때
FocusEvent	Component	컴포넌트가 포커스를 받거나 잃을 때
TextEvent	TextField	텍스트 변경
	TextArea	텍스트 변경
WindowEvent	Window	Window를 상속받는 모든 컴포넌트에 대해 윈도우 활성화, 비활성화, 아이콘화, 아이콘에서 복구, 윈도우 열기, 윈도우 닫기, 윈도우 종료
AdjustmentEvent	JScrollBar	스크롤바를 움직일 때
ComponentEvent	Component	컴포넌트가 사라지거나, 나타나거나, 이동, 크기 변경 시
ContainerEvent	Container	Container에 컴포넌트의 추가 혹은 삭제

이벤트 리스너(Event Listener)

- 이벤트 리스너란?
  - 발생된 이벤트 객체에 반응하여서 이벤트를 처리하는 객체를 이벤트 리스너(event listener)라고 한다.
- 이벤트 리스너 작성을 위한 interface 제공
  - 개발자가 리스너 인터페이스의 추상 메소드 구현
    - 이벤트가 발생하면 자바 플랫폼은 리스너 인터페이스의 추상 메소드 호출
    - 예) ActionListener, MouseListener 인터페이스

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

```
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
}
```

이벤트 리스너 등록

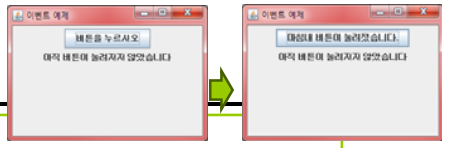
- 이벤트 리스너 등록
  - 이벤트를 받아 처리하고자 하는 컴포넌트에 이벤트 리스너 등록
- 이벤트 리스너 등록 메소드
  - Component.addXXXListener(listener)
    - xxx : 이벤트 명
    - listener : 이벤트 리스너 객체
    - 예) addMouseListener(), addActionListener(), addFocusListener() 등
- 이벤트 리스너가 등록된 컴포넌트에만 이벤트 전달
  - 이벤트 리스너가 등록된 컴포넌트만 이벤트 리스너 코드 작동

이벤트 종류	리스너 인터페이스	리스너의 추상 메소드	메소드가 호출되는 경우
Action	ActionListener	void actionPerformed(ActionEvent)	Action 이벤트가 발생하는 경우
Item	ItemListener	void itemStateChanged(ItemEvent)	Item 이벤트가 발생하는 경우
Key	KeyListener	void keyPressed(KeyEvent)	모든 키에 대해 키가 눌러질 때
		void keyReleased(KeyEvent)	모든 키에 대해 눌러진 키가 떼어질 때
		void keyTyped(KeyEvent)	유니코드 키가 입력될 때
Mouse	MouseListener	void mousePressed(MouseEvent)	마우스 버튼이 눌러질 때
		void mouseReleased(MouseEvent)	눌러진 마우스 버튼이 떼어질 때
		void mouseClicked(MouseEvent)	마우스 버튼이 클릭될 때
		void mouseEntered(MouseEvent)	마우스가 컴포넌트 위에 올라올 때
		void mouseExited(MouseEvent)	컴포넌트 위에 올라온 마우스가 컴포넌트를 벗어날 때
Mouse	MouseMotionListener	void mouseDragged(MouseEvent)	마우스를 컴포넌트 위에서 드래그할 때
		void mouseMoved(MouseEvent)	마우스가 컴포넌트 위에서 움직일 때
Focus	FocusListener	void focusGained(FocusEvent)	컴포넌트가 포커스를 받을 때
		void focusLost(FocusEvent)	컴포넌트가 포커스를 잃을 때
Text	TextListener	void textValueChanged(TextEvent)	텍스트가 변경될 때
Window	WindowListener	void windowOpened(WindowEvent)	윈도우가 생성되어 처음으로 보이게 될 때
		void windowClosing(WindowEvent)	윈도우의 시스템 메뉴에서 윈도우 닫기를 시도할 때
		void windowIconfied(WindowEvent)	윈도우가 아이콘화될 때
		void windowDeiconfied(WindowEvent)	아이콘 상태에서 원래 상태로 복귀할 때
		void windowClosed(WindowEvent)	윈도우가 닫혔을 때
		void windowActivated(WindowEvent)	윈도우가 활성화될 때
		void windowDeactivated(WindowEvent)	윈도우가 비활성화될 때
Adjustment	AdjustmentListener	void adjustmentValueChanged(AdjustmentEvent)	스크롤바를 움직일 때
Component	ComponentListener	void componentHidden(ComponentEvent)	컴포넌트가 보이지 않는 상태로 될 때
		void componentShown(ComponentEvent)	컴포넌트가 보이는 상태로 될 때
		void componentResized(ComponentEvent)	컴포넌트의 크기가 변경될 때
		void componentMoved(ComponentEvent)	컴포넌트의 위치가 변경될 때
Container	ContainerListener	void componentAdded(ContainerEvent)	컴포넌트가 컨테이너에 추가될 때
		void componentRemoved(ContainerEvent)	컴포넌트가 컨테이너에서 삭제될 때

이벤트 처리기 작성 방법

1. 독립적인 클래스로 이벤트 처리기를 작성
2. 내부 클래스(inner class)로 이벤트 처리기를 작성
3. 프레임 클래스에 이벤트 처리를 구현
4. 무명 클래스(anonymous class)를 사용하는 방법
5. 람다식(lambda)을 이용하는 방법

1.독립적인 클래스 작성



```
import javax.swing.*;
import java.awt.event.*;
class MyListener implements ActionListener {
  public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource();
        button.setText("마침내 버튼이 눌려졌습니다.");
  }
}
class MyFrame extends JFrame {
  private JButton button; private JLabel label;
  public MyFrame() {
        this.setSize(300, 200);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setTitle("이벤트 예제");
        JPanel panel = new JPanel();
        button = new JButton("버튼을 누르시오");
        label = new JLabel("아직 버튼이 눌려지지 않았습니다");
        button.addActionListener(new MyListener());
        panel.add(button);
        panel.add(label);
        this.add(panel);
        this.setVisible(true);
  }
}
```

Action 이벤트를 처리하는 코드 작성.

이벤트 리스너를 컴포넌트에 붙인다.

```
public class ActionEventTest1 {
    public static void main(String[] args) {
        MyFrame t = new MyFrame();
    }
}
```

2.내부 클래스 방법

- 만약 MyListener라는 클래스를 별도의 클래스로 하면 MyFrame 안의 멤버 변수들을 쉽게 사용할 수 없다.
- **일반적으로 MyListener 클래스를 내부 클래스로 만든다.**

```
class MyFrame extends JFrame {
…
    private class MyListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            if (e.getSource() == button) {
                label.setText("마침내 버튼이 눌려졌습니다.");
            }
        }
    }
}
public class ActionEventTest {
    public static void main(String[] args) {
        MyFrame t = new MyFrame();
    }
}
```

내부 클래스 label 에 접근할 수 있다.

3.MyFrame에서 이벤트 처리 구현 방법

- 더 많이 사용되는 방법은 MyFrame 클래스가 JFrame을 상속받으면서 동시에 ActionListener 인터페이스도 구현하는 경우이다.

```
…
class MyFrame extends JFrame implements ActionListener {
    …
    public MyFrame() {
        …
        button = new JButton("버튼을 누르시오");
        label = new JLabel("아직 버튼이 눌려지지 않았습니다");
        button.addActionListener(this);
        …
    }
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == button) {
            label.setText("마침내 버튼이 눌려졌습니다.");
        }
    }
}
…
```

이벤트도 처리

4.무명 클래스를 사용하는 방법

```
class MyFrame extends JFrame {
        ...
        public MyFrame() {
                ...
                button = new JButton("버튼을 누르시오");
                button.addActionListener(new ActionListener() {
                        public void actionPerformed(ActionEvent e) {
                                if (e.getSource() == button) {
                                        label.setText("마침내 버튼이
                                                        눌려졌습니다.");
                                }
                        }
                });
                ...
        }
}
```

5.람다식을 이용하는 방법

```
import javax.swing.*;
class MyFrame extends JFrame {
        private JButton button;
        private JLabel label;
        public MyFrame() {
                this.setSize(300, 200);
                this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                this.setTitle("이벤트 예제");
                JPanel panel = new JPanel();
                button = new JButton("버튼을 누르시오");
                label = new JLabel("아직 버튼이 눌려지지 않았습니다");
                button.addActionListener(e -> {
                        label.setText("마침내 버튼이 눌려졌습니다.");
                });
                panel.add(button);
                panel.add(label);
                this.add(panel);
                this.setVisible(true);
        }
}
```