



# java 17강 - AWT

---

양 명 속

[[now4ever7@gmail.com](mailto:now4ever7@gmail.com)]



# 목차

---

- AWT 란
  - 컴포넌트
  - 컨테이너
- AWT의 주요 컴포넌트
- 메뉴 만들기
- 레이아웃 매니저



# AWT

---

## ■ AWT란

- AWT(Abstract Window Toolkit)
- Window프로그래밍 (GUI 프로그래밍)을 하기 위한 도구
- GUI 어플리케이션의 개발에 필요한 여러 개의 관련 패키지와 클래스의 집합으로 구성
- GUI(Graphic User Interface) – 사용자가 그래픽을 통해서 하드웨어와 상호작용하는 환경을 말함
  - 마우스 등을 이용하여 화면의 메뉴 중에서 하나를 선택하여 작업을 지시함
- cf. 텍스트 기반 어플리케이션 : 키보드로 명령을 내리고 결과가 문자로 표시되는 단순한 형태



# AWT

---

- AWT 로 작성된 GUI 어플리케이션 역시 플랫폼 독립적이어서 여러 종류 OS에서 코드를 수정하지 않고도 실행이 가능하지만, 윈도우나 버튼과같은 GUI 컴포넌트들을 직접 구현하지 않고 해당 OS의 컴포넌트(native component)를 사용하기 때문에 AWT로 작성된 GUI 어플리케이션의 외양이 실행되는 OS마다 달라질 수 있음
- 여러 종류의 GUI 기반의 OS 들이 공통적으로 가지고 있는 컴포넌트만으로 구성해야하기 때문에 AWT가 제공할 수 있는 GUI 컴포넌트의 수가 제한적임
- 대신 OS에서 제공하는 컴포넌트는 해당 OS에 최적화되어 있기 때문에 자바로 구현한 컴포넌트보다 속도가 더 빠름



# AWT

---

- 자바의 성능과 하드웨어 환경이 크게 향상되면서 자바의 단점으로 지적되던 속도문제가 점점 해결되기 시작
- 그래서 새롭게 등장한 것이 **Swing**
  - AWT를 확장한 것
  - AWT와는 달리 순수한 자바로 이루어져 있어서 AWT보다 다양하고 풍부한 기능의 컴포넌트를 제공함으로써 진정한 GUI 어플리케이션을 개발하기 위한 도구로 사용되고 있음
  - **Swing**의 클래스들이 **AWT**의 클래스들을 기반으로 만들어진 **자손** 클래스들이므로 AWT와 Swing은 컴포넌트의 종류와 사용법만 조금 다를 뿐 나머지는 거의 같다



# AWT의 구성

## ■ AWT 관련 패키지 - java.awt로 시작

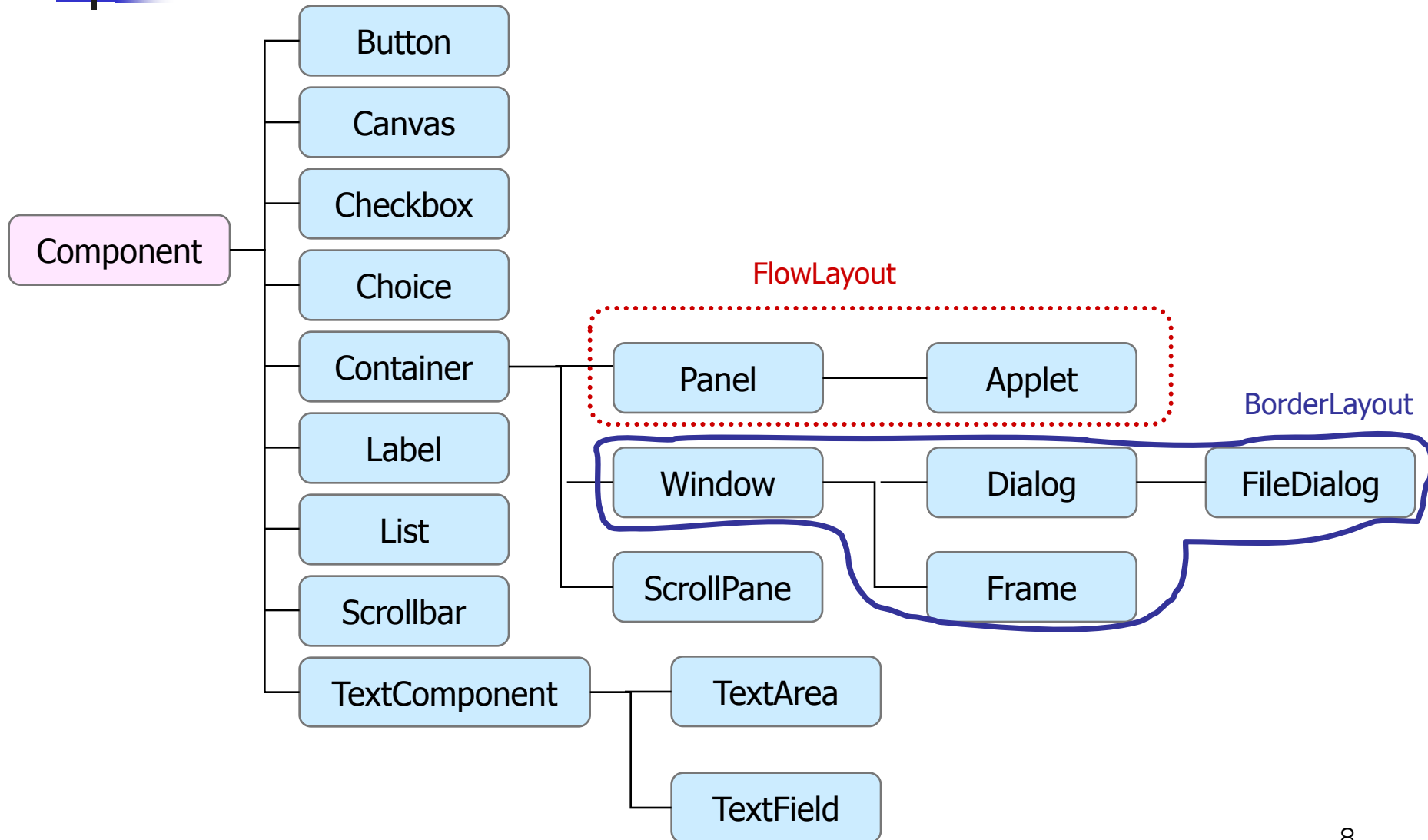
패키지명	설명
java.awt	AWT를 이용한 GUI 어플리케이션을 작성하는데 필요한 <b>기본적인 클래스와 컴포넌트</b> 를 제공
java.awt.event	GUI 어플리케이션에서 발생하는 <b>이벤트를 처리</b> 하는데 필요한 클래스와 인터페이스를 제공
java.awt.datatransfer	여러 어플리케이션 사이의, 또는 단일 어플리케이션 내에서의 데이터 전송을 구현하는데 필요한 클래스와 인터페이스를 제공함
java.awt.dnd	GUI의 장점 중의 하나인 끌어놓기(Drag and Drop)기능을 구현하는데 필요한 클래스들을 제공함
java.awt.font	폰트와 관련된 클래스와 인터페이스를 제공함
java.awt.image	이미지를 생성하거나 변경하는데 사용되는 클래스를 제공함
java.awt.print	출력에 관련된 클래스와 인터페이스를 제공함



# AWT 컴포넌트의 상속계층도

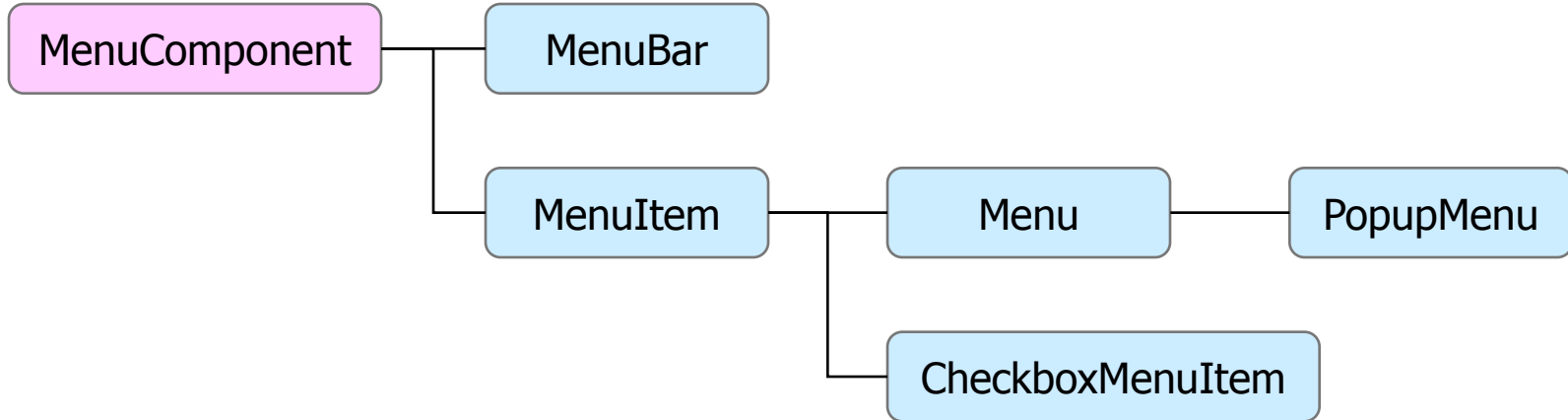
- 메뉴와 관련된 컴포넌트들을 제외한 모든 컴포넌트의 조상은 **Component 클래스**
- Component 클래스의 조상은 Object클래스
- Component 클래스와 그 자손 클래스들은 윈도우, 스크롤바, 버튼 등 GUI 응용 프로그램의 화면을 구성하는데 사용되는 클래스들임
- 이들을 AWT 컴포넌트 또는 줄여서 컴포넌트라고 부름
- 컴포넌트
  - 일반적인 컴포넌트 - 버튼, 체크박스 등
    - 일반적인 컴포넌트의 최상위 조상 - Component
  - 메뉴 컴포넌트 - 메뉴와 관련된 컴포넌트
    - 메뉴관련 컴포넌트의 최상위 조상 - **MenuComponent**

# AWT 컴포넌트의 상속계층도





# AWT 메뉴 컴포넌트의 상속계층도





# 컴포넌트

---

- Component는 MenuComponent를 제외한 AWT의 모든 컴포넌트의 조상이고 추상 클래스임
- Component에는 컴포넌트들이 가져야 할 공통적인 메서드들을 정의해놓고 있음



# 컨테이너(Container)

- 컨테이너 - Component의 자손들 중에 Container와 그 자손들
  - 다른 컴포넌트들을 포함할 수 있어서 Button, Label 과 같은 컴포넌트들을 포함할 수 있음
  - 컨테이너가 컨테이너를 포함할 수도 있음
  - 컨테이너에는 여러 개의 오버로딩된 **add 메서드**들이 정의되어 있어서, 컨테이너에 단순히 add메서드를 사용하는 것만으로 다른 컴포넌트들을 포함시킬 수 있음
  - 컨테이너에 포함된 컴포넌트들은 기본적으로 컨테이너에 설정된 배경색, 전경색, 글자체 등의 설정을 그대로 따르게 됨
- 독립적인 컨테이너
  - 독립적으로 사용될 수 있으며, 다른 컴포넌트나 종속적인 컨테이너를 포함할 수 있음
  - Frame, Window, Dialog
- 종속적인 컨테이너
  - 독립적으로 사용될 수 없으며, 다른 컨테이너에 포함되어야만 함
  - Panel, ScrollPane



# 컨테이너(Container)

컨테이너	설명
Frame	가장 일반적인 컨테이너로 윈도우와 모양이 같다. titlebar와 크기 조절버튼, 닫기 버튼을 가지고 있다. 메뉴를 추가할 수 있다
Window	Frame의 조상 경계선, titlebar, 크기 조절버튼, 닫기 버튼이 없으며, 메뉴도 추가할 수 없다 단지 컴포넌트를 담을 수 있는 평면 공간만을 가짐
Dialog	Frame 처럼 titlebar, 닫기 버튼을 갖고 있지만, 메뉴는 가질 수 없으며 기본적으로 크기를 변경할 수 없다. 주로 프로그램 사용자에게 메시지를 보여 주거나, 응답을 받는데 사용
Panel	평면 공간으로 Frame과 같이 여러 컴포넌트를 담을 수 있으며 단독적으로 사용될 수는 없다
ScrollPane	Panel과 같은 평면 공간이지만, Panel과는 달리 <b>단 하나의 컴포넌트만 포함</b> 할 수 있으며 자신보다 큰 컴포넌트가 포함되면 스크롤바가 자동적으로 나타남

# AWT의 주요 컴포넌트



## ■ Frame

- 생성자 : `Frame(String title)`
- 메서드 : `getTitle`, `setTitle`, `setState`, `getState`, `setResizable`

```
import java.awt.*;
```

```
class FrameTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Login"); // Frame객체를 생성한다.  
        f.setSize(300, 200);          // Frame의 크기를 설정한다.  
        f.setVisible(true);           // 생성한 Frame을 화면에 보이도록 한다.  
    }  
}
```

- 크기조절도 되고, **titlebar**를 드래그하면 이동, 윈도우즈의 윈도우와 같은 기능
- 닫기 버튼 - 이벤트 처리 이용해서 기능 추가해주어야 함



# Frame

---

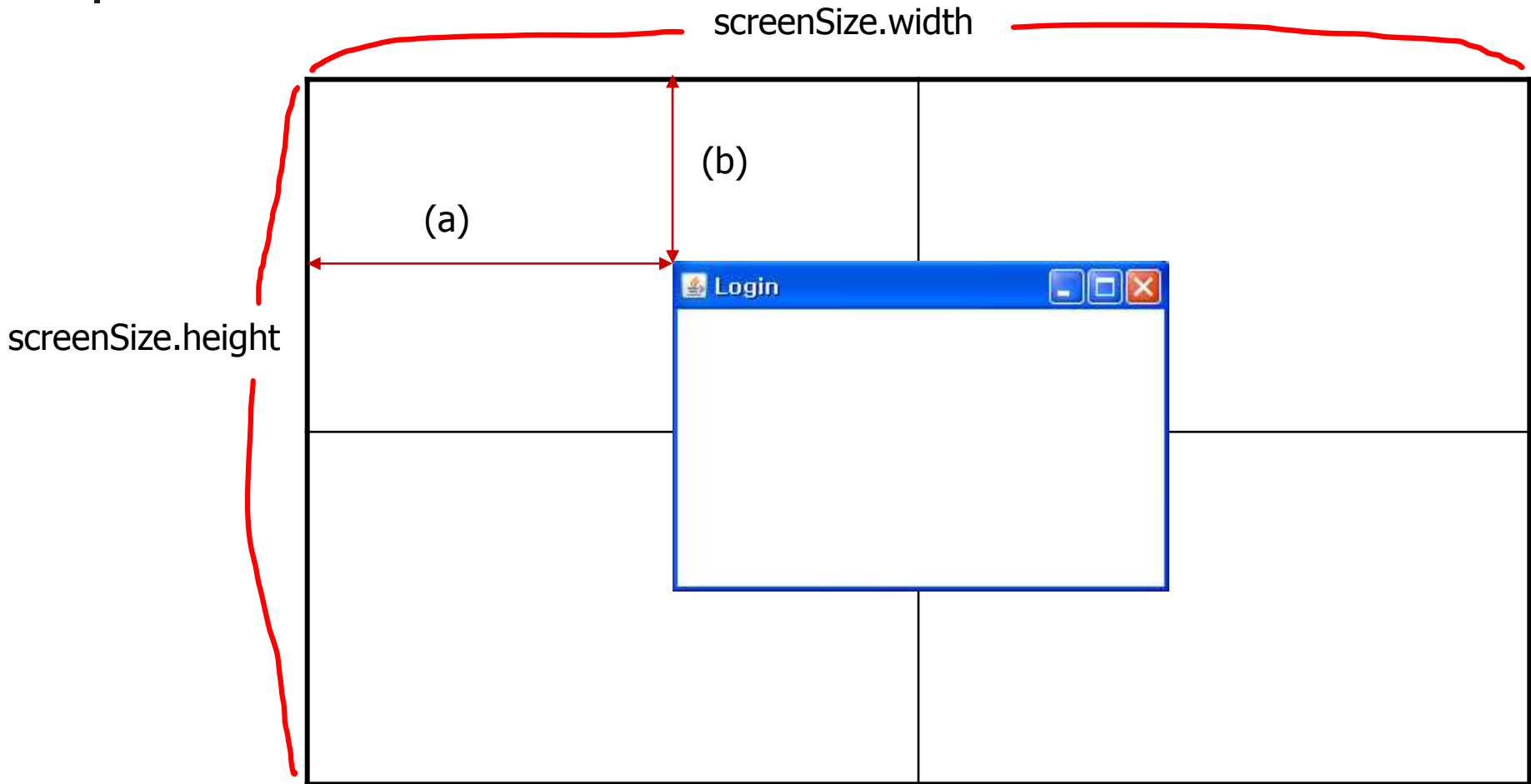
```
import java.awt.*;
```

```
class FrameTest2 {  
    public static void main(String args[]) {  
        Frame f = new Frame("Login");  
        f.setSize(300, 200);  
  
        Toolkit tk = Toolkit.getDefaultToolkit(); // 구현된 Toolkit객체를 얻는다.  
        Dimension screenSize = tk.getScreenSize(); // 화면의 크기를 구한다.  
  
        // 화면크기의 절반값에서 Frame크기의 절반값을 뺀 위치로 하면  
        // Frame이 화면 가운데 위치하게 된다.  
        f.setLocation(screenSize.width/2 - 150, screenSize.height/2 - 100);  
        f.setVisible(true); // 생성한 Frame을 화면에 보이도록 한다.  
    }  
}
```

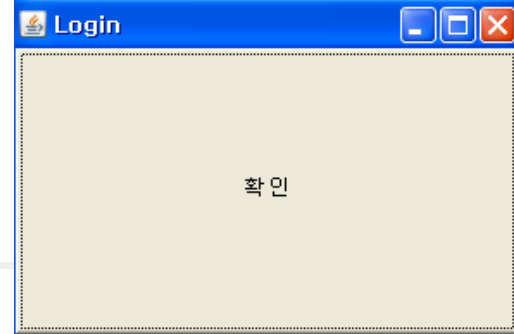
■ Toolkit 클래스의 `getScreenSize()`를 이용해서 Frame이 화면의 중앙에 나타나게 함

# Frame

- (a)  $\text{screenSize.width}/2 - \text{Frame너비}/2$
- (b)  $\text{screenSize.height}/2 - \text{Frame높이}/2$



# Button



- Button – 사용자가 클릭했을 때, 어떤 작업이 수행되도록 할 때 사용됨
- 메서드 : getLabel, setLabel
- 생성자 : Button(String label)

```
import java.awt.*;
```

```
class ButtonTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Login");  
        f.setSize(300, 200);  
  
        Button b = new Button("확 인"); // Button위에 "확 인"이라는 글자가 나타난다.  
        b.setSize(100, 50);             // Button의 크기를 설정한다.  
  
        f.add(b);                        // 생성된 Button을 Frame에 포함시킨다.  
        f.setVisible(true);  
    }  
}
```

- button이 Frame에 가득 차 있고, Frame의 크기를 변경하면 Button의 크기도 같이 변경됨
- Frame에 설정되어 있는 레이아웃 매니저 때문 (기본적으로 BorderLayout으로 설정)
- 컨테이너에는 레이아웃 매니저를 선택적으로 설정할 수 있는데, 레이아웃 매니저는 컨테이너의 크기가 변경될 때마다 컨테이너에 포함된 컴포넌트들의 크기와 위치를 자동적으로 관리해 줌
- 컨테이너 내의 컴포넌트 배치에 대해 신경쓰지 않아도 됨



# Button

```
import java.awt.*;
```

```
class ButtonTest2 {  
    public static void main(String args[]) {  
        Frame f = new Frame("Login");  
        f.setSize(300, 200);  
        f.setLayout(null);           // 레이아웃 매니저의 설정을 해제한다.  
  
        Button b = new Button("확 인");  
        b.setSize(100, 50);         // Button의 크기를 설정한다.  
        b.setLocation(100, 75);     // Frame내에서의 Button의 위치를 설정한다.  
  
        f.add(b);  
        f.setVisible(true);  
    }  
}
```



- 기본적으로 설정된 레이아웃 매니저를 없애고, **Button**의 크기와 위치를 수동적으로 설정
- **Frame**의 크기를 변경해도 **Button**의 크기와 위치에는 전혀 변화가 없다
- 대신 **Button**의 크기와 위치를 수동적으로 직접 지정해주어야 함



# Choice

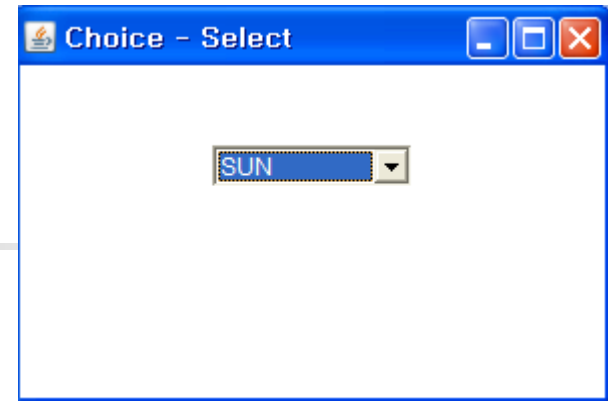
---

- Choice - 여러 개의 item이 있는 목록을 보여주고, 그 중에서 한 가지를 선택하도록 할 때 사용
  - 기존 GUI : 콤보박스, drop-down Listbox 라고 부름
  - 메서드 : add, remove, removeAll, insert, getItem, getItemCount, getSelectedIndex, getSelectedItem

# Choice

```
import java.awt.*;
```

```
class ChoiceTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Choice - Select");  
        f.setSize(300, 200);  
        f.setLayout(null);  
  
        Choice day = new Choice();           // Choice객체를 생성한 다음  
        day.add("SUN");                     // Choice의 목록에 나타날 값들을 추가한다.  
        day.add("MON");  
        day.add("TUE");  
        day.add("WED");  
        day.add("THU");  
        day.add("FRI");  
        day.add("SAT");  
  
        day.setSize(100, 50);  
        day.setLocation(100, 70);  
  
        f.add(day);  
        f.setVisible(true);  
    }  
}
```





# List

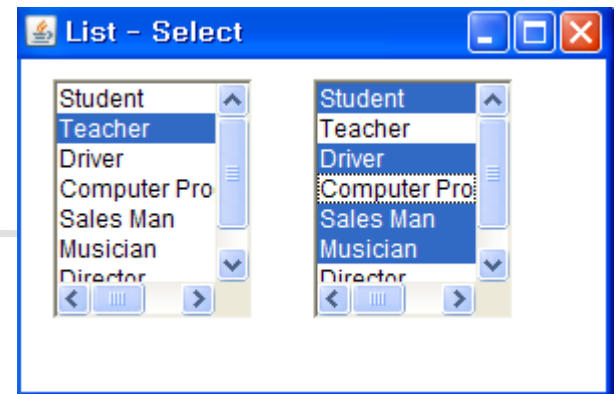
---

- List- 목록에서 원하는 항목을 선택할 수 있도록 할 때 사용됨
- 처음부터 모든 item 목록을 보여주며, 목록의 item 중에서 하나 또는 여러 개를 선택할 수 있음
- 생성자
  - List(int rows, boolean multipleMode) : multipleMode는 true이면 여러 개의 item 선택가능
  - List(int rows) : multipleMode는 false, 하나의 item만 선택가능
  - List() : rows의 값은 기본값인 4로 지정
- 메서드 : add, replaceItem, remove, removeAll, getRows, getItem, getItems, getItemCount, select, deselect, getSelectedIndex, getSelectedIndexes, getSelectedItem, getSelectedItems, isIndexSelected, setMultipleMode

# List

```
import java.awt.*;
```

```
class ListTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("List - Select");  
        f.setSize(300, 200);  
        f.setLayout(null);  
  
        List selectOne = new List(6); // 6개 목록을 보여줄 수 있는 크기의 List를 만든다.  
        selectOne.setLocation(20,40);  
        selectOne.setSize(100, 120);  
        selectOne.add("Student");  
        selectOne.add("Teacher");  
        selectOne.add("Driver");  
        selectOne.add("Computer Programmer");  
        selectOne.add("Sales Man");  
        selectOne.add("Musician");  
        selectOne.add("Director");  
    }  
}
```





# List

---

// 생성자의 두번째 인자값을 true로 설정해서 List의 목록에서 여러 개를 선택할 수 있게 했다.

```
List selectMany = new List(6, true); //다중 선택
```

```
selectMany.setLocation(150, 40);
```

```
selectMany.setSize(100,120);
```

```
selectMany.add("Student");
```

```
selectMany.add("Teacher");
```

```
selectMany.add("Driver");
```

```
selectMany.add("Computer Programmer");
```

```
selectMany.add("Sales Man");
```

```
selectMany.add("Musician");
```

```
selectMany.add("Director");
```

```
f.add(selectOne);
```

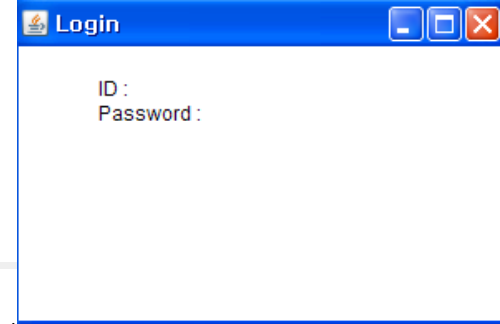
```
f.add(selectMany);
```

```
f.setVisible(true);
```

```
}
```

```
}
```

# Label



- Label – 화면에 글자를 표시, 설명이나 메시지를 화면에 나타내는데 주로 사용
- 생성자
  - Label(String text, int alignment) : alignment – Label.LEFT, Label.CENTER, Label.RIGHT
  - Label(String text) : 기본값인 Label.LEFT로 설정
- 메서드 : getText, setText, setAlignment

```
import java.awt.*;
class LabelTest {
    public static void main(String args[]) {
        Frame f = new Frame("Login");
        f.setSize(300, 200);
        f.setLayout(null);

        Label id = new Label("ID :");    // Label을 생성하고 크기와 위치를 지정한다.
        id.setBounds(50, 50, 30, 10);    // 50, 50위치에 크기가 가로 30, 세로 10

        Label pwd = new Label("Password :");
        pwd.setBounds(50, 65, 100, 10);

        f.add(id); // 생성한 Label을 Frame에 포함시킨다.
        f.add(pwd);
        f.setVisible(true);
    }
}
```

```
id.setSize(30, 10);
id.setLocation(50, 50);
```



# Checkbox

---

- Checkbox – boolean과 같이 true/false 또는 on/off 와 같이 둘 중의 한 값을 가질 수 있는 컴포넌트
- CheckboxGroup을 이용하면, 여러 가지 값들 중에서 한 가지를 선택할 수 있는 radio button도 만들 수 있음
- 생성자
  - Checkbox(String text, boolean state) : state – true이면 선택된 상태로 생성
  - Checkbox(String text) : 선택해제된 상태로
  - Checkbox()
  - Checkbox(String text , CheckboxGroup group, boolean state)
    - group – CheckboxGroup 객체의 참조
    - CheckboxGroup을 이용해서 radio button으로 만든다
- 메서드 : getLabel, setLabel, getState, setState



# Checkbox

```
import java.awt.*;
```

```
class CheckboxTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Questions");  
        f.setSize(280, 300);
```

```
// Frame의 LayoutManager를 FlowLayout으로 설정한다.  
f.setLayout(new FlowLayout());
```

```
Label q1 = new Label("1. 당신의 관심 분야는?(여러개 선택가능);  
Checkbox news = new Checkbox("news", true); // 선택된 상태로 생성  
Checkbox sports = new Checkbox("sports");  
Checkbox movies = new Checkbox("movies");  
Checkbox music = new Checkbox("music");
```

```
f.add(q1); f.add(news); f.add(sports); f.add(movies); f.add(music);
```

```
Label q2 = new Label("2. 얼마나 자주 극장에 가십니까?");  
CheckboxGroup group1 = new CheckboxGroup();  
Checkbox movie1 = new Checkbox("한 달에 한 번 갑니다.", group1, true);  
Checkbox movie2 = new Checkbox("일주일에 한 번 갑니다.", group1, false);  
Checkbox movie3 = new Checkbox("일주일에 두 번 갑니다.", group1, false);
```

Questions

1. 당신의 관심 분야는?(여러개 선택가능)  
☒ news ☐ sports ☐ movies ☐ music

2. 얼마나 자주 극장에 가십니까?  
☒ 한 달에 한 번 갑니다.  
☐ 일주일에 한 번 갑니다.  
☐ 일주일에 두 번 갑니다.

3. 하루에 얼마나 컴퓨터를 사용하십니까?  
☒ 5시간 이하 ☐ 10시간 이하 ☐ 15시간 이상



# Checkbox

---

```
f.add(q2); f.add(movie1); f.add(movie2); f.add(movie3);
```

```
Label q3 = new Label("3. 하루에 얼마나 컴퓨터를 사용하십니까?");
```

```
CheckboxGroup group2 = new CheckboxGroup();
```

```
Checkbox com1 = new Checkbox("5시간 이하 ", group2, true);
```

```
Checkbox com2 = new Checkbox("10시간 이하", group2, false);
```

```
Checkbox com3 = new Checkbox("15시간 이상", group2, false);
```

```
f.add(q3); f.add(com1); f.add(com2); f.add(com3);
```

```
f.setVisible(true);
```

```
}
```

```
}
```

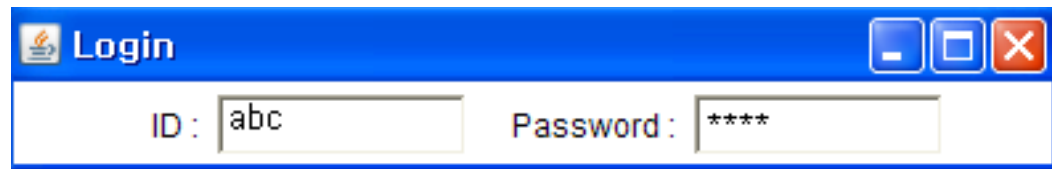


# TextField

---

- TextField – 사용자로부터 값을 입력받을 수 있는 컴포넌트
- 편집이 가능, 한 줄만 입력할 수 있어서 이름, id, 비밀번호 등 비교적 짧은 값의 입력에 사용됨
- 생성자
  - TextField(String text, int col) : col – 입력받을 글자의 수
    - col의 값에 따라서 TextField의 크기가 결정됨
  - TextField(int col)
  - TextField(String text)
  - TextField()
- 메서드 : setEchoChar, getColumns, setText, getText, select, selectAll, getSelectedText, setEditable

# TextField



```
import java.awt.*;
```

```
class TextFieldTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Login");  
        f.setSize(400, 65);  
        f.setLayout(new FlowLayout()); // LayoutManager를 FlowLayout으로 한다.  
  
        Label lid = new Label("ID :", Label.RIGHT); // 정렬을 오른쪽으로.  
        Label lpwd = new Label("Password :", Label.RIGHT);  
  
        TextField id = new TextField(10); //약 10개의 글자를 입력할 수 있는 TextField 생성  
        TextField pwd = new TextField(10);  
        pwd.setEchoChar('*'); // 입력한 값 대신 '*'가 보이도록 한다.  
  
        f.add(lid); // 생성한 컴포넌트들을 Frame에 포함시킨다.  
        f.add(id);  
        f.add(lpwd);  
        f.add(pwd);  
        f.setVisible(true);  
    }  
}
```

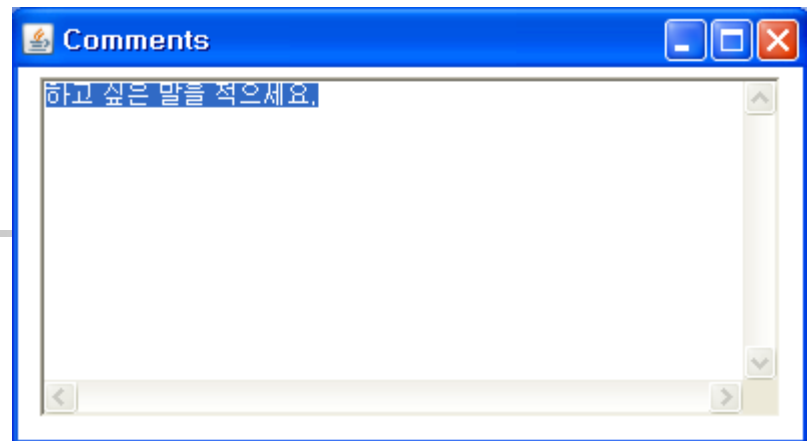


# TextArea

---

- TextArea – 여러 줄의 text를 입력하거나 보여줄 수 있는 편집가능한 컴포넌트
- 스크롤바를 이용해서 실제화면에 보이는 것보다 많은 양의 text를 담을 수 있음
- 생성자
  - TextArea(String text, int row, int col, int scrollbar)
    - scrollbar : scrollbar의 종류와 사용여부 지정
      - TextArea.SCROLLBARS\_BOTH
      - TextArea.SCROLLBARS\_NONE
      - TextArea.SCROLLBARS\_HORIZONTAL\_ONLY
      - TextArea.SCROLLBARS\_VERTICAL\_ONLY
  - TextArea(int row, int col) : scrollbar 는 수평, 수직 모두 갖는다
- 메서드
  - getRows, getColumns, setRows, setColumns, append, insert, replaceRange, setText, getText, select, selectAll, getSelectedText, setEditable

# TextArea



```
import java.awt.*;
```

```
class TextAreaTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Comments");  
        f.setSize(400, 220);  
        f.setLayout(new FlowLayout());
```

```
        TextArea comments = new TextArea("하고 싶은 말을 적으세요.", 10, 50);
```

```
        f.add(comments);  
        comments.selectAll(); // TextArea의 text 전체가 선택 되도록 한다.  
        f.setVisible(true);
```

```
    }  
}
```



# Scrollbar

---

- Scrollbar – 사용자가 정해진 범위 내에 있는 값을 쉽게 선택할 수 있도록 해주는 컴포넌트
- 주로 볼륨설정이나, 속도조절, 색상 선택 등에 사용됨
- 생성자
  - Scrollbar(int orientation, int init, int buttonSize, int min, int max)
  - orientation : Scrollbar의 종류 지정
    - Scrollbar.VERTICAL, Scrollbar.HORIZONTAL
  - init : Scrollbar의 초기값
  - buttonSize : Scroll버튼(bubble)의 크기
  - min : Scrollbar가 가질 수 있는 최소값
  - max : Scrollbar가 가질 수 있는 최대값
- 메서드 : getValue, setValue

# Scrollbar

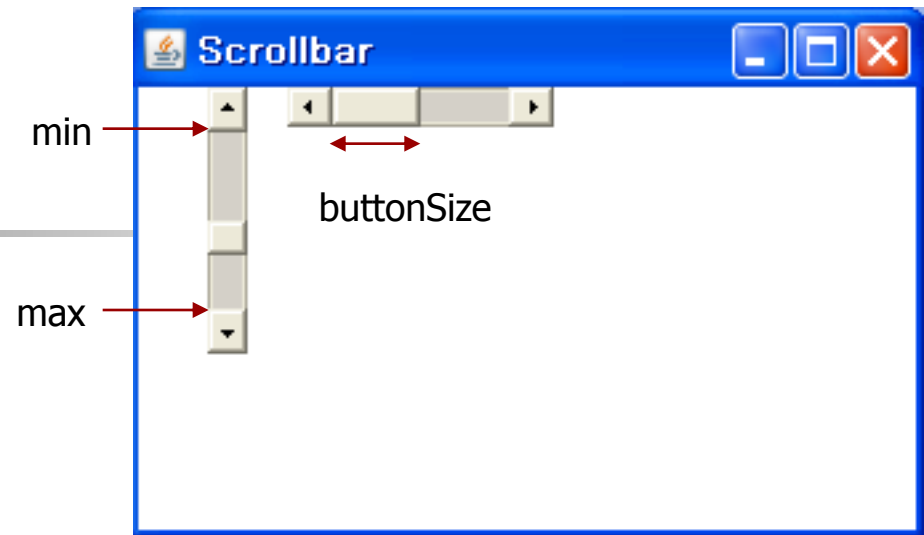
```
import java.awt.*;
```

```
class ScrollbarTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Scrollbar");  
        f.setSize(300, 200);  
        f.setLayout(null);
```

```
        //Scrollbar(int orientation, int init, int buttonSize, int min, int max)  
        Scrollbar hor = new Scrollbar(Scrollbar.HORIZONTAL, 0, 50, 0, 100);  
        hor.setSize(100, 15);  
        hor.setLocation(60, 30);  
        Scrollbar ver = new Scrollbar(Scrollbar.VERTICAL, 50, 20, 0, 100);  
        ver.setSize(15, 100);  
        ver.setLocation(30, 30);
```

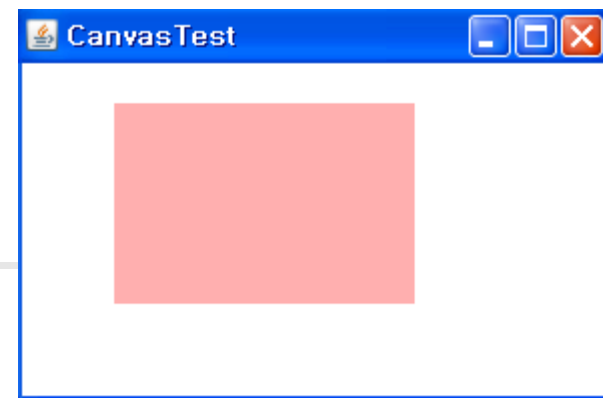
```
        f.add(hor);  
        f.add(ver);  
        f.setVisible(true);
```

```
    }  
}
```





# Canvas



- Canvas – 빈 평면 공간을 제공하는 컴포넌트
  - 여기에 그림을 그릴수도 있고, 글자를 적을 수도 있음
  - 주로 그림을 그리거나 이미지를 위한 공간으로 사용되며, 사용자 정의 컴포넌트를 만들 때도 사용됨

```
import java.awt.*;
```

```
class CanvasTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("CanvasTest");  
        f.setSize(300, 200);  
        f.setLayout(null);                // Frame의 Layout Manager설정을 해제한다.  
  
        Canvas c = new Canvas();  
        c.setBackground(Color.pink);    // Canvas의 배경을 분홍색(pink)으로 한다.  
        c.setBounds(50, 50, 150, 100);  
  
        f.add(c);                        // Canvas을 Frame에 포함시킨다.  
        f.setVisible(true);  
    }  
}
```

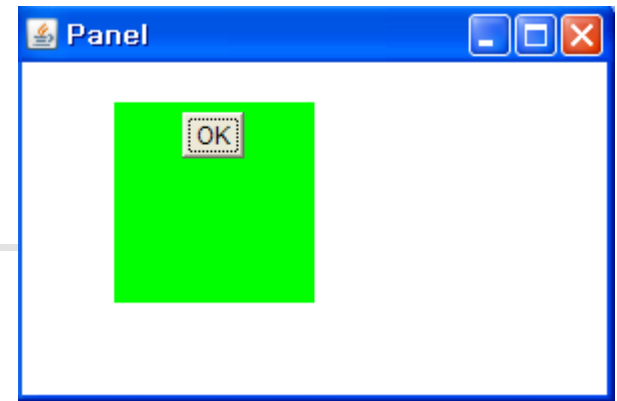


# Panel

---

- Panel – Frame과 같이 다른 컴포넌트를 자신의 영역 내에 포함시킬 수 있는 컨테이너
  - 동시에 Panel 자신이 다른 컨테이너에 포함될 수 있기도 함
  - Panel 이 Panel 에 포함되는 것도 가능함
  - Frame과 달리 titlebar나 버튼도 없고, 단지 비어있는 평면 공간만을 갖는다
  - 컨테이너라 자신만의 레이아웃을 유지할 수 있어서, Panel을 이용하면 Frame 내에서 컴포넌트들의 배치를 다양하게 할 수 있음
- 생성자
  - `public Panel()`
  - `public Panel(LayoutManager layout)`

# Panel




```
import java.awt.*;
```

```
class PanelTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Panel");  
        f.setSize(300, 200);  
        f.setLayout(null);                // Frame이 Layout Manager를 사용하지 않도록 한다.  
  
        Panel p = new Panel();  
        p.setBackground(Color.green);    // Panel의 배경을 녹색으로 한다.  
        p.setSize(100, 100);  
        p.setLocation(50, 50);  
  
        Button ok = new Button("OK");  
  
        p.add(ok);                        // Button을 Panel에 포함시킨다.  
        f.add(p);                        // Panel을 Frame에 포함시킨다.  
        f.setVisible(true);  
    }  
}
```

**Panel**은 기본적으로 **FlowLayout** 을 레이아웃 매니저로 사용하므로 **Panel**안에 포함된 버튼의 위치와 크기는 지정해주지 않아도 됨

# 실습

 회원가입

이름  비밀번호

직업  결혼여부 ☒ 미혼 ☐ 기혼

멤버십 정보 ☐ SKT ☐ KT ☐ LG U+ 자기 소개



# ScrollPane

- ScrollPane – 컨테이너이므로 다른 컴포넌트를 포함시킬 수 있으나, 다른 컨테이너들과는 달리 **단 하나의 컴포넌트만을 포함**시킬 수 있음
- 제한된 공간에서 크기가 큰 컴포넌트를 화면에 보여줄 수 있도록 하는데 사용됨
- 포함된 컴포넌트의 크기가 ScrollPane 자신보다 큰 경우 스크롤바를 이용해서 볼 수 있게 해줌
- 생성자
  - ScrollPane(int scrollbarDisplayPolicy)
    - scrollbarDisplayPolicy : 아래 값중 하나 지정
      - SCROLLBARS\_ALWAYS : 스크롤바가 항상 보이게
      - SCROLLBARS\_AS\_NEEDED : 필요할 때만 보이게
      - SCROLLBARS\_NEVER : 항상 보이지 않도록
  - ScrollPane()

# ScrollPane

```
import java.awt.*;
```

```
class ScrollPaneTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("ScrollPaneTest");  
        f.setSize(300, 200);  
  
        ScrollPane sp = new ScrollPane();  
        Panel p = new Panel();  
        p.setBackground(Color.green); // Panel의 배경을 green으로 한다.  
        p.add(new Button("첫번째")); // Button을 Panel에 포함시킨다.  
        p.add(new Button("두번째"));  
        p.add(new Button("세번째"));  
        p.add(new Button("네번째"));  
  
        sp.add(p); // Panel을 ScrollPane에 포함시킨다.  
        f.add(sp); // ScrollPane을 Frame에 포함시킨다.  
        f.setVisible(true);  
    }  
}
```



- 4개의 Button을 가진 Panel을 ScrollPane에 넣고, ScrollPane을 Frame에 넣었음
- Frame의 크기를 줄였을 때, 자동적으로 스크롤바가 나타나서 Frame의 크기를 변경하지 않고도 스크롤바를 통해서 가려진 컴포넌트들을 볼 수 있음



# Dialog

---

- Dialog - 주로 화면에 메시지창을 보여주는데 사용됨
  - 프로그램의 실행 중에 사용자에게 에러가 발생했음을 알린다던가, 파일을 삭제하기 전에 사용자로 부터 응답을 받아야 한다던가 하는데 사용
  - 다른 컴포넌트들을 포함할수 있는 컨테이너, Frame과 유사한 모양
  - 메서드 : show, hide, dispose, getTitle, setModal, setResizable
    - hide - Dialog가 화면에 보이지만 않게
    - dispose - Dialog를 닫음(메모리에서도 제거, 다시 사용하지 않을 때)
  - 생성자
    - Dialog(Frame parent, String title, boolean modal)
    - Dialog(Frame parent, String title)
    - 하나의 Frame이 Dialog의 부모로 지정되어야 함
    - 단순히 Dialog가 어느 Frame과 관련된 것인지를 지정해 주기 위한 것
    - modal(필수응답)인 경우
      - 부모로 지정된 Frame은 Dialog가 사라지기 전까지는 사용할 수 없게 됨
      - 주로 사용자로 부터 중요한 확인을 받아야 할 경우 사용

# Dialog

```
import java.awt.*;
```

```
class DialogTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Parent");  
        f.setSize(300, 200);  
  
        // parent Frame을 f로 하고, modal을 true로 해서 필수응답 Dialog로 함.  
        Dialog info = new Dialog(f, "Information", true);  
        info.setSize(140, 90);  
        info.setLocation(50, 50);           // parent Frame이 아닌, 화면이 위치의 기준이 된다.  
        info.setLayout(new FlowLayout());  
  
        Label msg = new Label("This is modal Dialog", Label.CENTER);  
        Button ok = new Button("OK");  
        info.add(msg);  
        info.add(ok);  
  
        f.setVisible(true);  
        info.setVisible(true); // Dialog를 화면에 보이게 한다.  
    }  
}
```







# Dialog – OK 버튼 누르면 닫히도록

```
import java.awt.*;
import java.awt.event.*;          // 이벤트를 처리하기 위해 추가해야 한다.
class DialogTest2 {
    public static void main(String args[]) {
        Frame f = new Frame("Parent");
        f.setSize(300, 200);

        final Dialog info = new Dialog(f, "Information", true);
        info.setSize(140, 90);
        info.setLocation(50, 50);
        info.setLayout(new FlowLayout());

        Label msg = new Label("This is modal Dialog", Label.CENTER);
        Button ok = new Button("OK");

        ok.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { // OK 버튼을 누르면 수행됨.
                // info.setVisible(false); // Dialog를 안보이게 한다.
                info.dispose();           // Dialog를 메모리에서 없앤다.
            }
        });
        info.add(msg);
        info.add(ok);
        f.setVisible(true);
        info.setVisible(true); // Dialog를 화면에 보이게 한다.
    }
}
```



# FileDialog

---

- FileDialog – 파일을 열거나 저장할 때 사용되는 Dialog
- 생성자
  - FileDialog(Frame parent, String title, int mode)
    - mode : FileDialog.LOAD, FileDialog.SAVE 중 하나
  - FileDialog(Frame parent, String title)
    - mode를 생략하면 디폴트로 LOAD가 사용됨
- 메서드
  - getFile, getDirectory, setFile, setDirectory

# FileDialog

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

```
class FileDialogTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Parent");  
        f.setSize(300, 200);
```

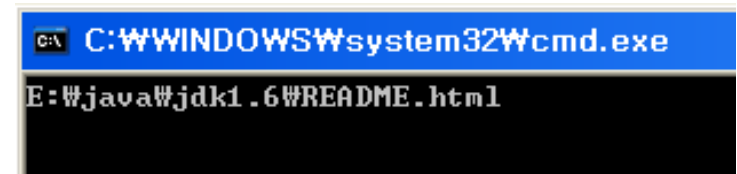
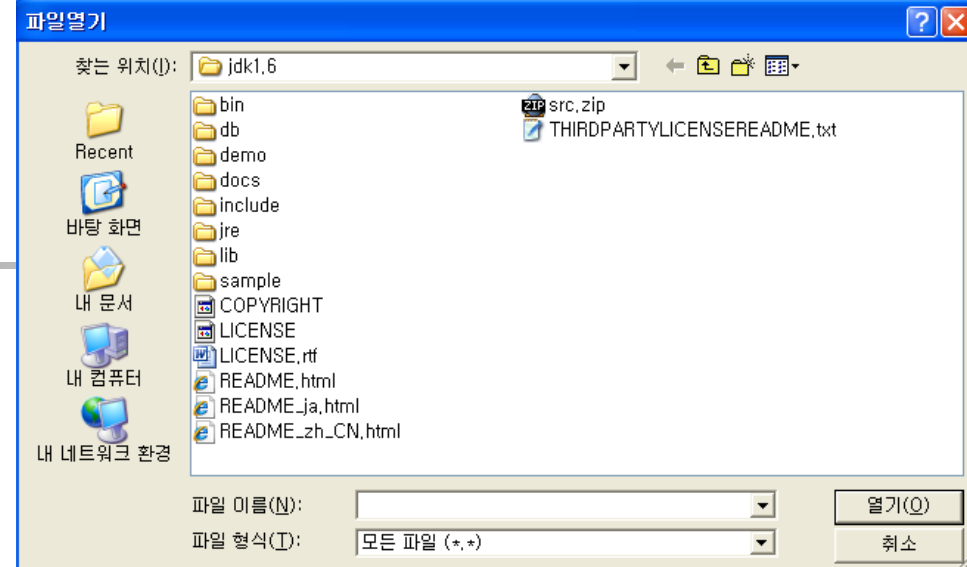
```
        FileDialog fileOpen = new FileDialog(f, "파일열기", FileDialog.LOAD);
```

```
        f.setVisible(true);  
        fileOpen.setDirectory("E:\\WWjava\\WWjdk1.6");  
        fileOpen.setVisible(true);
```

```
        //파일을 선택한 다음, FileDialog의 열기버튼을 누르면,  
        // getFile()과 getDirectory()를 이용해서 파일이름과 위치한 디렉토리를 얻을 수 있다.  
        System.out.println(fileOpen.getDirectory() + fileOpen.getFile());
```

```
    }  
}
```

취소버튼을 누르면 null을 반환함





# 그 외의 AWT 클래스

---

## ■ Font

- Component의 setFont(Font f)를 이용하면, 컴포넌트에 사용되는 text의 글자체를 원하는 것으로 지정가능
- 먼저 Font의 인스턴스를 생성해야 함
- 생성자
  - Font(String name, int style, int size)
    - 폰트의 이름
    - 폰트의 스타일:Font.PLAIN(보통체), Font.BOLD, Font.ITALIC, Font.BOLD+Font.ITALIC(굵은 기울임체)
      - Font의 static 멤버인 정수형 상수, 덧셈연산 가능
    - 폰트의 크기
- JDK에서는 5가지 폰트를 기본적으로 제공
  - Serif,
  - SansSerif, Dialog, DialogInput, Monospaced
- 그 외의 폰트를 사용하려면 컴퓨터에 설치되어 있어야 함
- GraphicsEnvironment클래스 - 현재 사용중인 시스템에 설치된 Font의 리스트를 얻기 위해 사용



# Font

```
import java.awt.*;
```

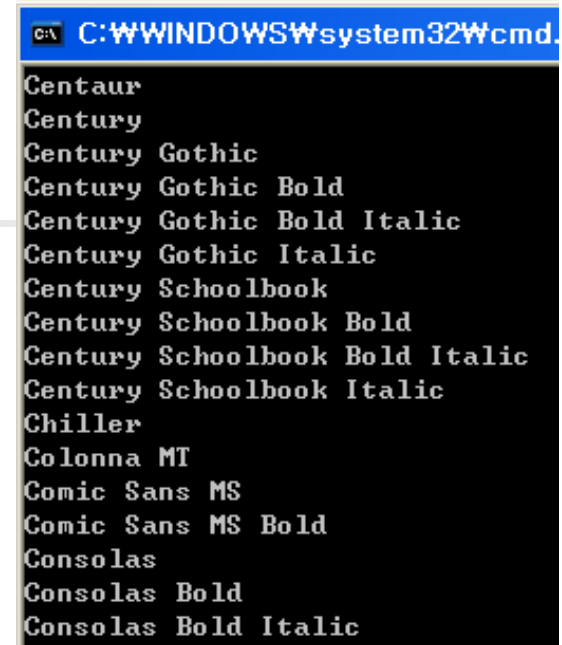
```
class FontList  
{
```

```
    public static void main(String[] args)  
    {
```

```
        GraphicsEnvironment ge =  
        GraphicsEnvironment.getLocalGraphicsEnvironment();  
        Font[] fonts = ge.getAllFonts();
```

```
        for(int i=0; i < fonts.length; i++) {  
            System.out.println(fonts[i].getFontName());  
        }
```

```
    }  
}
```

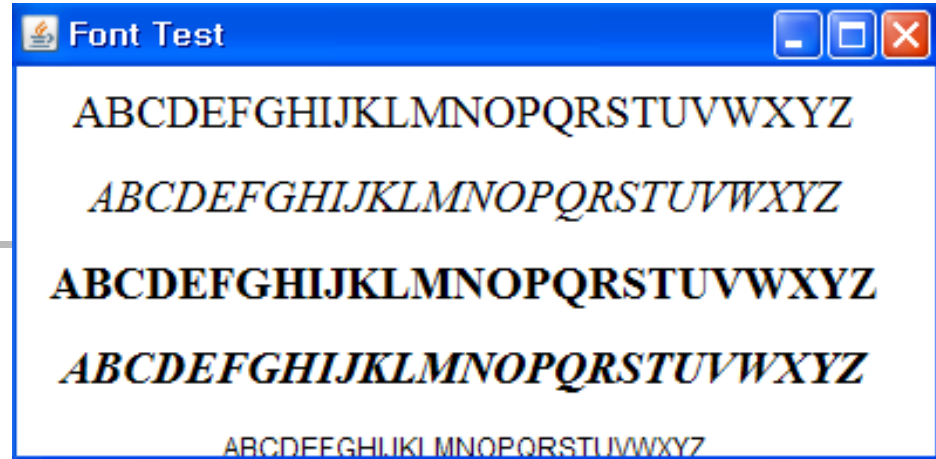


```
C:\WINDOWS\system32\cmd.exe  
Centaur  
Century  
Century Gothic  
Century Gothic Bold  
Century Gothic Bold Italic  
Century Gothic Italic  
Century Schoolbook  
Century Schoolbook Bold  
Century Schoolbook Bold Italic  
Century Schoolbook Italic  
Chiller  
Colonna MT  
Comic Sans MS  
Comic Sans MS Bold  
Consolas  
Consolas Bold  
Consolas Bold Italic
```

# Font

```
import java.awt.*;
```

```
class FontTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Font Test");  
        String abc = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
  
        // A 부터 Z를 내용으로 갖는 Label들을 생성한다.  
        Label abc1 = new Label(abc);  
        Label abc2 = new Label(abc);  
        Label abc3 = new Label(abc);  
        Label abc4 = new Label(abc);  
        Label abc5 = new Label(abc);  
  
        // Serif체이며, 크기가 20인 Font  
        Font f1 = new Font("Serif", Font.PLAIN, 20);           // 보통체  
        Font f2 = new Font("Serif", Font.ITALIC, 20);          // 기울임체  
        Font f3 = new Font("Serif", Font.BOLD, 20);            // 굵은체  
        Font f4 = new Font("Serif", Font.BOLD+Font.ITALIC, 20); // 굵은 기울임체  
  
        abc1.setFont(f1);           // Label에 새로운 Font를 적용한다.  
        abc2.setFont(f2);  
        abc3.setFont(f3);  
        abc4.setFont(f4);  
    }  
}
```





# Font

---

```
f.setLayout(new FlowLayout());  
f.add(abc1);  
f.add(abc2);  
f.add(abc3);  
f.add(abc4);  
f.add(abc5);  
  
f.setSize(400, 200);  
f.setVisible(true);  
}  
}
```



# Color

---

- **Color** – 색을 표현하기 위해 사용되는 클래스
- 원하는 색의 RGB 값만 알고 있으면 그 색을 표현할 수 있는 객체를 생성하여 사용할 수 있음
- Color 클래스내에는 blue, red, green, yellow 등 자주 쓰이는 색 13가지가 static 멤버변수로 정의되어있어서 색의 RGB 값을 몰라도 쉽게 사용할 수 있음
- 주로 컴포넌트의 배경색과 전경색을 설정하는 `setForeground(Color c)`, `setBackground(Color c)` 와 같은 메서드의 매개변수로 사용됨
- 생성자
  - `Color(int r, int g, int b)` : r, g, b 모두 0~255사이의 정수값
  - `Color(float r, float g, float b)` : r, g, b 모두 0.0~1.0사이의 실수값
  - `Color(int r, int g, int b, int a)` : a – alpha 값으로 0~255사이의 정수값
  - `Color(float r, float g, float b, float a)` : a – alpha 값으로 0.0~1.0사이의 실수값
    - alpha – 색의 불투명도, 값이 클수록 색이 불투명해짐



# Color

```
import java.awt.*;
```

```
class ColorTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Color Test");  
        f.setLayout(new GridLayout(14, 2));  
        Panel p1 = new Panel();      p1.setBackground(Color.black);  
        Panel p2 = new Panel();      p2.setBackground(Color.blue);  
        Panel p3 = new Panel();      p3.setBackground(Color.cyan);  
        Panel p4 = new Panel();      p4.setBackground(Color.darkGray);  
        Panel p5 = new Panel();      p5.setBackground(Color.gray);  
        Panel p6 = new Panel();      p6.setBackground(Color.green);  
        Panel p7 = new Panel();      p7.setBackground(Color.lightGray);  
        Panel p8 = new Panel();      p8.setBackground(Color.magenta);  
        Panel p9 = new Panel();      p9.setBackground(Color.orange);  
        Panel p10 = new Panel();     p10.setBackground(Color.pink);  
        Panel p11 = new Panel();     p11.setBackground(Color.red);  
        Panel p12 = new Panel();     p12.setBackground(Color.white);  
        Panel p13 = new Panel();     p13.setBackground(Color.yellow);  
        Panel p14 = new Panel();     p14.setBackground(new Color(50,100,100));  
    }  
}
```





# Color

---

```
f.add(new Label("Color.black"));
f.add(new Label("Color.blue"));
f.add(new Label("Color.cyan"));
f.add(new Label("Color.darkGray"));
f.add(new Label("Color.gray"));
f.add(new Label("Color.green"));
f.add(new Label("Color.lightGray"));
f.add(new Label("Color.magenta"));
f.add(new Label("Color.orange"));
f.add(new Label("Color.pink"));
f.add(new Label("Color.red"));
f.add(new Label("Color.white"));
f.add(new Label("Color.yellow"));
f.add(new Label("Color(50, 100, 100)"));
f.setSize(250, 300);
f.setVisible(true);
}
}
```

```
f.add(p1);
f.add(p2);
f.add(p3);
f.add(p4);
f.add(p5);
f.add(p6);
f.add(p7);
f.add(p8);
f.add(p9);
f.add(p10);
f.add(p11);
f.add(p12);
f.add(p13);
f.add(p14);
```

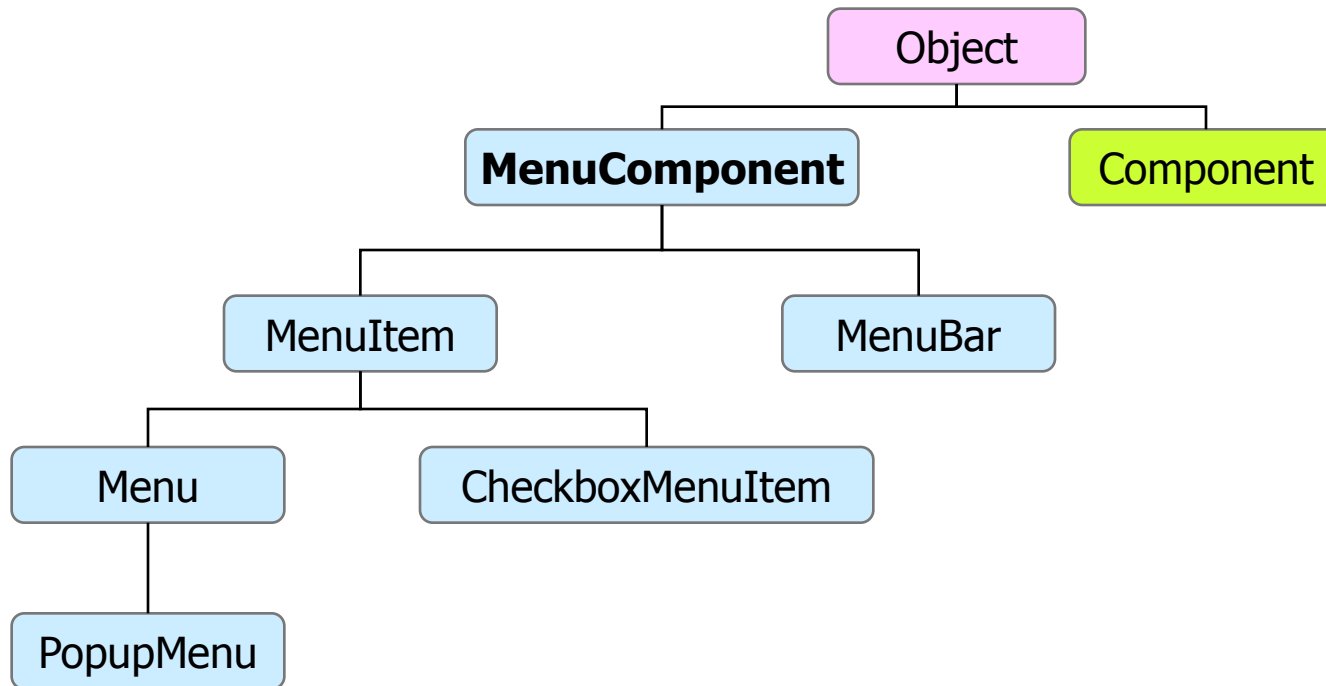


# 메뉴 만들기

---

# 메뉴 만들기

- 메뉴를 구성하는 컴포넌트
  - Frame에 메뉴를 구성하기 위해서는 MenuBar, Menu, MenuItem과 같은 컴포넌트들을 사용
- 메뉴관련 컴포넌트의 상속계층도





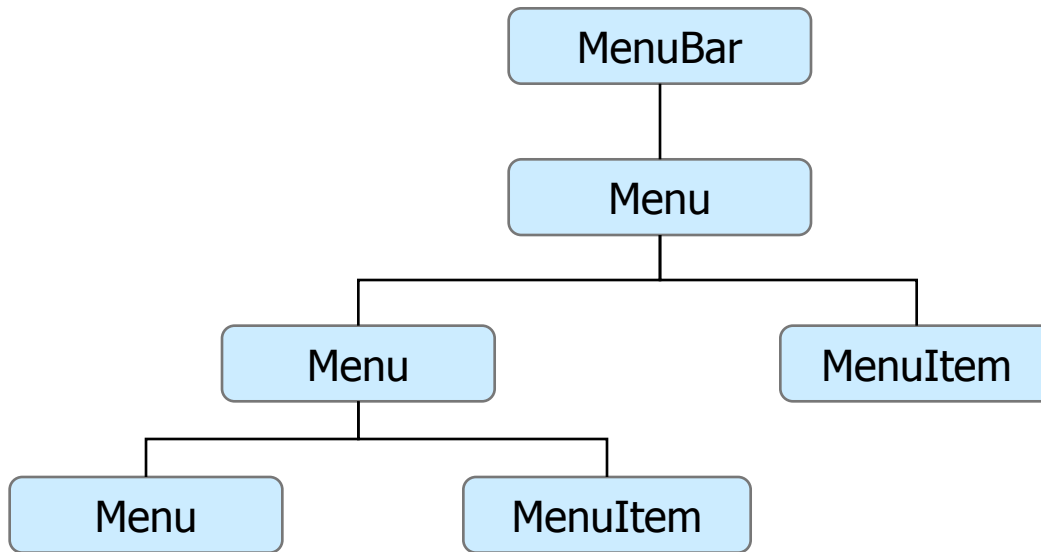
# 메뉴 만들기

---

- 폴더에 폴더를 담듯이 Menu에 다시 Menu를 담아서 계층형으로 메뉴를 구성할 수 있음
- 항상 화면에 나타나는 최상위 메뉴는 MenuBar에 담고, MenuBar는 다시 Frame에 추가함으로써 메뉴 작성을 마치게 됨
  - Menu에 MenuItem들을 추가
  - MenuBar에 Menu를 추가
  - Frame에 MenuBar를 포함시킨다
- CheckboxMenuItem – 메뉴를 클릭할 때마다 메뉴 앞에 체크표시가 설정되거나 해제됨
- 생성자
  - CheckboxMenuItem(String name, boolean status)
    - status : 값이 true이면 체크된 상태로 생성됨
  - CheckboxMenuItem(String name) : status는 기본적으로 false
- 메서드 : setHelpMenu, addSeparator

# 메뉴 만들기

## ■ 메뉴 컴포넌트간의 포함관계

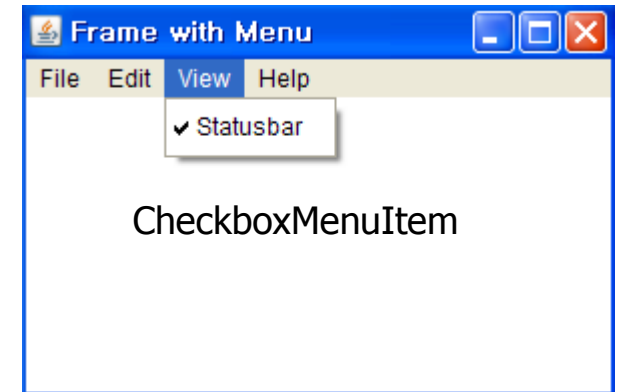
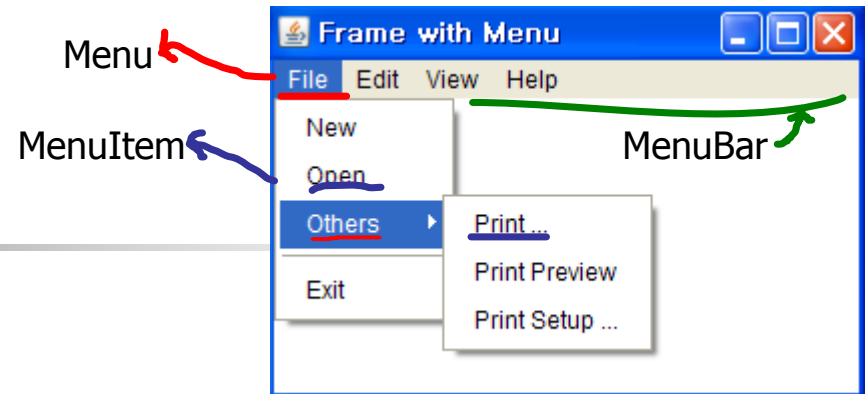


- MenuBar에는 Menu만 포함될 수 있다.(MenuItem은 포함시킬 수 없다)
- Menu에는 Menu와 MenuItem이 포함될 수 있다
- 계층형 menu를 만들고자 할 때는 Menu에 Menu를 포함시키면 된다.

# 메뉴 만들기

```
import java.awt.*;
```

```
class MenuTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("Frame with Menu");  
        f.setSize(300, 200);  
  
        MenuBar mb = new MenuBar();  
        Menu mFile = new Menu("File");  
  
        MenuItem miNew = new MenuItem("New");  
        MenuItem miOpen = new MenuItem("Open");  
        Menu mOthers = new Menu("Others"); // MenuItem이 아니라 Menu임에 주의  
        MenuItem miExit = new MenuItem("Exit");  
  
        mFile.add(miNew); // Menu에 MenuItem들을 추가한다.  
        mFile.add(miOpen);  
        mFile.add(mOthers); // Menu에 Menu를 추가한다.  
        mFile.addSeparator(); // 메뉴 분리선을 넣는다.  
        mFile.add(miExit);  
    }  
}
```





# 메뉴 만들기

---

```
MenuItem miPrint = new MenuItem("Print ...");
MenuItem miPreview = new MenuItem("Print Preview");
MenuItem miSetup = new MenuItem("Print Setup ...");
mOthers.add(miPrint);
mOthers.add(miPreview);
mOthers.add(miSetup);

Menu mEdit = new Menu("Edit");
Menu mView = new Menu("View");
Menu mHelp = new Menu("Help");
CheckboxMenuItem miStatusbar = new CheckboxMenuItem("Statusbar");
mView.add(miStatusbar);

mb.add(mFile); // MenuBar에 Menu를 추가한다.
mb.add(mEdit);
mb.add(mView);
mb.setHelpMenu(mHelp); // mHelp를 HelpMenu로 지정한다.

f.setMenuBar(mb); // Frame에 MenuBar를 포함시킨다.
f.setVisible(true);

}
```





# PopupMenu

---

- PopupMenu – Frame에 붙어있는 고정적인 MenuBar와는 달리, Frame 내의 어디서나 마우스의 오른쪽 버튼을 누르면 나타나는, 위치가 고정되어 있지 않은 메뉴

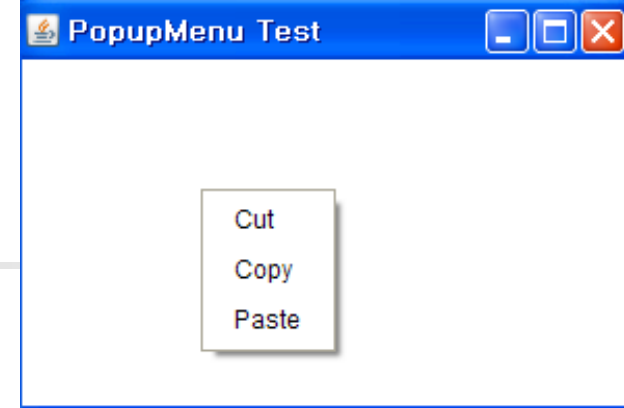
# PopupMenu

```
import java.awt.*;
import java.awt.event.*;          // 이벤트를 처리를 위해서 추가
class PopupMenuTest {
    public static void main(String args[]) {
        final Frame f = new Frame("PopupMenu Test");
        f.setSize(300, 200);

        final PopupMenu pMenu = new PopupMenu("Edit");
        MenuItem miCut = new MenuItem("Cut");
        MenuItem miCopy = new MenuItem("Copy");
        MenuItem miPaste = new MenuItem("Paste");
        pMenu.add(miCut);           // PopupMenu에 MenuItem들을 추가한다.
        pMenu.add(miCopy);
        pMenu.add(miPaste);

        f.add(pMenu);              // PopupMenu를 Frame에 추가한다.
        f.addMouseListener( new MouseAdapter() {
            public void mousePressed(MouseEvent me) {
                // 오른쪽 마우스버튼을 누르는 경우에만 PopupMenu를 화면에 보여준다.
                if(me.getModifiers() == me.BUTTON3_MASK)
                    pMenu.show(f, me.getX(), me.getY());
            }
        });
        f.setVisible(true);
    }
}
```

public void show(Component origin, int x, int y)





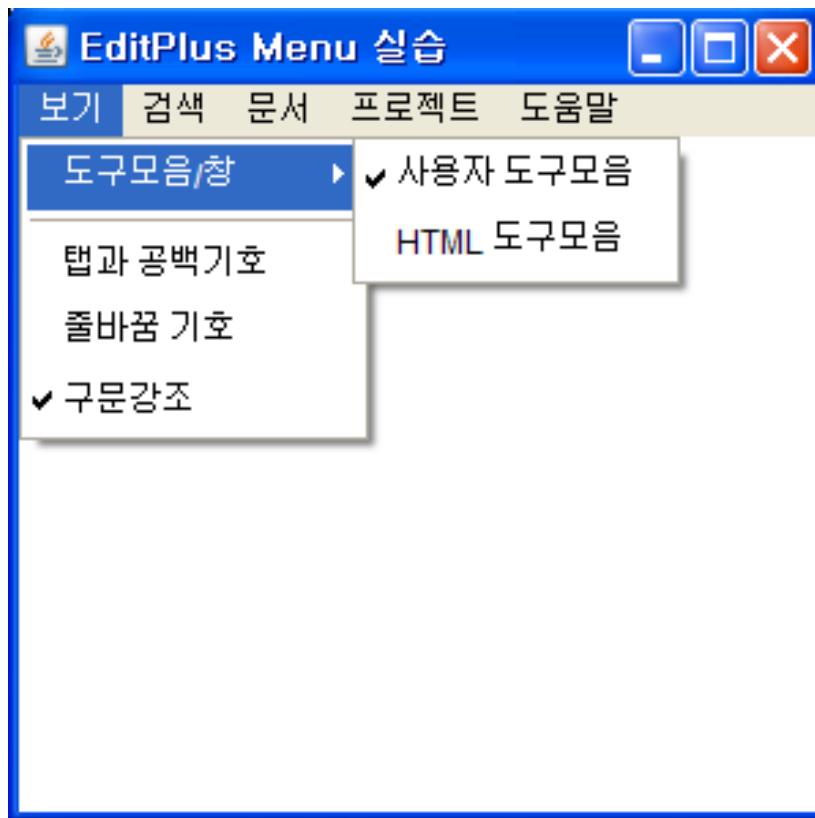
# PopupMenu

---

- 익명 클래스를 main 메서드 내에 정의했기 때문에, 익명 클래스에서 Frame 타입 참조변수 f 와 PopupMenu 타입 참조변수 pMenu를 참조하기 위해서 final 키워드 사용
  - 메서드 내에 정의된 클래스(지역 클래스)에서는 같은 메서드 내에 선언된 지역변수들 중 final 키워드가 붙은 것들만 참조가 가능하므로

# 실습

## ■ 메뉴 만들기





# 레이아웃 매니저

---



# 레이아웃 매니저

---

- 레이아웃 매니저
  - 컨테이너에 포함된 컴포넌트들의 배치(Layout)를 자동적으로 관리해주는 일을 함
  - 컨테이너에 새로운 컴포넌트가 추가되거나 컨테이너의 크기가 변경되었을 경우, 컨테이너에 포함된 컴포넌트들의 재배치를 레이아웃 매니저가 자동적으로 처리해줌
  - 컨테이너에는 단 하나의 레이아웃 매니저만을 설정할 수 있음
  - 모든 컨테이너는 따로 설정하지 않아도 기본적으로 레이아웃 매니저가 지정되어 있음
- 레이아웃 매니저의 종류
  - BorderLayout
  - FlowLayout
  - GridLayout
  - CardLayout
  - GridbagLayout



# 레이아웃 매니저

---

- 컨테이너별 기본 레이아웃 매니저
  - FlowLayout – Panel, Applet
  - BorderLayout – Window, Dialog, Frame



# BorderLayout

- BorderLayout – 컨테이너를 North, South, East, West, Center 모두 5개의 영역으로 나누고, **각 영역에 하나의 컴포넌트만을 배치**할 수 있도록 함
- 한 영역에 여러 개의 컴포넌트를 배치하면, 마지막에 추가한 컴포넌트만 보이게 됨
- 한 영역에 하나 이상의 컴포넌트를 넣기 위해 Panel을 이용해야 함
- 생성자
  - BorderLayout(int hgap, int vgap) : 각 영역 사이에 간격이 있는 BorderLayout을 생성함
    - hgap – 각 영역의 사이에 간격을 줌(좌우), vgap(위아래)
  - BorderLayout() : 영역 사이에 간격이 없는 BorderLayout 생성
- 메서드
  - add(String name, Component c)
  - add(Component c, String name)
    - name : "North", "South", "East", "West", "Center"중의 하나
    - 또는 BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST, BorderLayout.CENTER



# BorderLayout

```
import java.awt.*;
```

```
class BorderLayoutTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("BorderLayoutTest");  
        f.setSize(200, 200);
```

```
// Frame은 기본적으로 BorderLayout로 설정되어있으므로 따로 설정하지 않아도 됨
```

```
        f.setLayout(new BorderLayout());  
        Button north = new Button("North");  
        Button south = new Button("South");  
        Button east = new Button("East");  
        Button west = new Button("West");  
        Button center = new Button("Center");
```

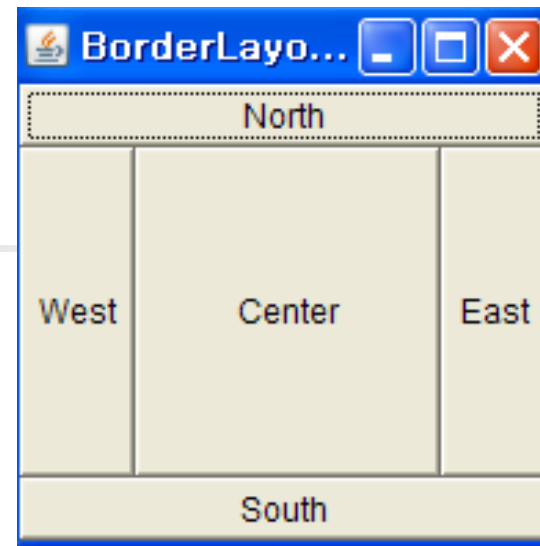
```
// Frame의 5개의 각 영역에 Button을 하나씩 추가한다.
```

```
        f.add(north, "North");    // f.add("North",north);와 같이 쓸 수도 있다.  
        f.add(south, "South");    // South의 대소문자 정확히  
        f.add(east, "East");        // East대신, BorderLayout.EAST 사용가능  
        f.add(west, "West");  
        f.add(center, "Center");
```

```
        f.setVisible(true);
```

```
    }
```

```
}
```



- **Center**의 컴포넌트가 없으면 그 자리는 비어있게 됨
- 다른 자리가 채워져 있지 않으면 **Center**의 컴포넌트가 그 자리를 메우게 됨

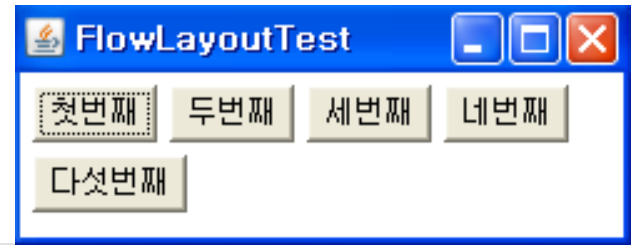


# FlowLayout

---

- FlowLayout – 컴포넌트들이 추가되는 순서에 따라 왼쪽에서 오른쪽으로 이어져나가며, 공간이 부족하면 아랫줄에 추가됨
  - 컴포넌트를 프레임상에 원래의 크기대로 차례차례 배치
- 컴포넌트의 정렬 – 왼쪽 정렬, 오른쪽 정렬, 가운데 정렬
- 생성자
  - FlowLayout(int align, int hgap, int vgap)
    - align : 컴포넌트들의 정렬방법
      - FlowLayout.LEFT, FlowLayout.CENTER, FlowLayout.RIGHT
    - hgap – 각 컴포넌트간의 사이에 간격을 줌(좌우), vgap(위아래)
  - FlowLayout(int align)
    - hgap, vgap 이 5 픽셀인 FlowLayout을 생성
  - FlowLayout()
    - 가운데 정렬이면서 hgap, vgap 이 5 픽셀인 FlowLayout을 생성

# FlowLayout



```
import java.awt.*;
```

```
class FlowLayoutTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("FlowLayoutTest");  
        f.setSize(250, 100);  
        f.setLayout(new FlowLayout(FlowLayout.LEFT)); // 왼쪽정렬의 FlowLayout생성  
  
        f.add(new Button("첫 번째"));  
        f.add(new Button("두 번째"));  
        f.add(new Button("세 번째"));  
        f.add(new Button("네 번째"));  
        f.add(new Button("다섯 번째"));  
  
        f.setVisible(true);  
    }  
}
```



# GridLayout

---

- GridLayout – 컨테이너를 테이블처럼 행과 열로 나누어 컴포넌트를 배치
- 여기에 추가되는 컴포넌트들은 모두 같은 크기로 나누어지며, 컨테이너의 크기를 변경하면 각 영역이 모두 같은 비율로 커지거나 작아지는 성질을 갖고 있음
- 생성자
  - GridLayout(int row, int col, int hgap, int vgap)
    - 영역들 간의 사이에 간격이 있는 GridLayout 을 생성
    - row – 컨테이너를 몇 개의 행으로 나눌것인지
    - col – 컨테이너를 몇 개의 열로 나눌것인지
  - GridLayout(int row, int col)
    - 영역들 간의 사이에 간격이 없는 GridLayout 을 생성

# GridLayout



```
import java.awt.*;
```

```
class GridLayoutTest {  
    public static void main(String args[]) {  
        Frame f = new Frame("GridLayoutTest");  
        f.setSize(150, 150);  
        f.setLayout(new GridLayout(3, 2)); // 3행 2열의 테이블을 만든다.  
  
        f.add(new Button("1"));           // 추가되는 순서대로 Button에 번호를 붙였다.  
        f.add(new Button("2"));  
        f.add(new Button("3"));  
        f.add(new Button("4"));  
        f.add(new Button("5"));  
        f.add(new Button("6"));  
  
        f.setVisible(true);  
    }  
}
```



# GridbagLayout

---

- GridbagLayout – GridLayout 과 같이 컨테이너를 열과 행으로 나누어 컴포넌트들을 배치할 수 있음
  - 프레임의 좌측 상단을 기준으로 폭과 크기를 정하여 컴포넌트를 배치
  - 화면 정중앙에 배치됨
  - 배치되는 위치를 조절할 수 있음
- 각 영역은 서로 다른 크기로 지정될 수 있으며, 인접한 열 또는 행으로의 확장이 가능함
- Html 문서의 table의 레이아웃을 설정하는 것과 비슷



# CardLayout

- CardLayout – 여러 화면을 슬라이드처럼 바꿔가며 보여줄 수 있음
  - 프레임에 카드를 얹어 놓은 듯이 여러 개의 Container를 상속받은 Panel 등과 같은 객체를 포개 놓은 듯한 배치
- 여러 개의 컨테이너를 CardLayout에 추가한 다음, 순서대로 또는 임의의 컨테이너를 선택해서 보여줄 수 있음
- 설치 프로그램과 같이 단계별로 다른 화면으로 이동하는 경우에 사용하면 유용
- 생성자
  - CardLayout(int hgap, int vgap)
    - hgap : 컨테이너와 CardLayout 사이에 간격을 줌(수평)
  - CardLayout() : 컨테이너와 간격이 없는 CardLayout 생성
- 메서드
  - add(Container parent, String name)
    - name : 주어진 이름으로, parent : 지정된 컨테이너에 추가
  - show, first, last, previous, next

# CardLayout

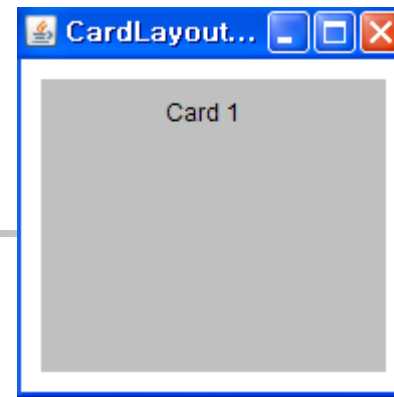
```
import java.awt.*;  
import java.awt.event.*;
```

```
class CardLayoutTest {  
    public static void main(String args[]) {  
        final Frame f = new Frame("CardLayoutTest");  
        final CardLayout card = new CardLayout(10, 10);  
        f.setLayout(card);
```

```
        Panel card1= new Panel();  
        card1.setBackground(Color.lightGray);  
        card1.add(new Label("Card 1"));  
        Panel card2= new Panel();  
        card2.add(new Label("Card 2"));  
        card2.setBackground(Color.orange);  
        Panel card3= new Panel();  
        card3.add(new Label("Card 3"));  
        card3.setBackground(Color.cyan);
```

```
        f.add(card1, "1");  
        f.add(card2, "2");  
        f.add(card3, "3");
```

// Frame에 card1을 "1"이라고 이름 붙여 추가한다.







# CardLayout

```
class Handler extends MouseAdapter {  
    public void mouseClicked(MouseEvent e) {  
        // 마우스 오른쪽 버튼을 눌렀을때  
        if(e.getModifiers() == e.BUTTON3_MASK) {  
            card.previous(f); // CardLayout의 이전 Panel을 보여준  
다.  
        } else {  
            card.next(f); // CardLayout의 다음 Panel을  
보여준다.  
        }  
    }  
} // class Handler
```

```
card1.addMouseListener(new Handler());  
card2.addMouseListener(new Handler());  
card3.addMouseListener(new Handler());
```

```
f.setSize(200, 200);  
f.setLocation(200, 200);  
f.setVisible(true);
```

```
card.show(f,"1"); // Frame에 추가된 Component중 이름이 "1"인 것을 보여준다.
```

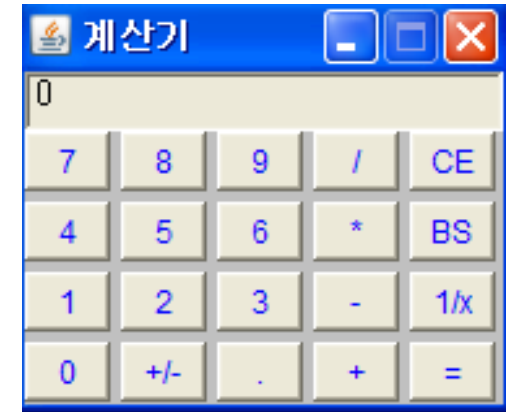
```
    }//main  
} //class
```

- Frame은 North, Center 두 영역으로만 되어 있는 BorderLayout 으로 설정
- Frame의 Center 영역에 4행 5열의 GridLayout으로 구성된 Panel 추가

## 예제2

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

```
class Calc {  
    public static void main(String args[]) {  
        Frame f = new Frame("계산기");  
        TextField tf = new TextField("0");  
        tf.setEditable(false);  
        f.setSize(190, 160);  
        f.setLocation(300,300);  
  
        f.add("North", tf);  
        Panel numPanel = new Panel();  
        Button[] numButtons = null;  
        numPanel.setLayout(new GridLayout(4, 5, 4, 4));  
        numPanel.setBackground(Color.lightGray);  
        f.add("Center", numPanel);  
  
        String numStr[] = { "7", "8", "9", "/", "CE",  
                            "4", "5", "6", "*", "BS",  
                            "1", "2", "3", "-", "1/x",  
                            "0", "+/-", ".", "+", "=" };  
        numButtons = new Button[numStr.length];
```





## 예제2

---

```
for(int i=0;i<numStr.length;i++) {  
    numButtons[i] = new Button(numStr[i]);  
    numButtons[i].setForeground(Color.blue);  
    numPanel.add(numButtons[i]);  
}
```

```
f.setResizable(false);  
f.setVisible(true);
```

```
}  
{
```



# 예제

- Container 클래스의 메서드
  - public Insets **getInsets()**
    - Insets 생성자 : Insets(int top, int left, int bottom, int right)
  - 컨테이너의 바깥여백을 주기 위해서는 Container 클래스의 **getInsets()** 메소드를 오버라이딩

```
public class MyFrame extends Frame
{
    public Insets getInsets(){
        //Insets(int top, int left, int bottom, int right)
        Insets i=new Insets(10,20,30,40);
        return i;
    }
}
```

```
class MyPanel extends Panel
{
    public Insets getInsets(){
        return new Insets(10,10,10,10);
    }
}
```

# 예제

```
import java.applet.*;
import java.awt.*;

class InsetTest extends Frame
{
    Button bt1, bt2;
    TextArea ta;

    public InsetTest(){
        super("실습");
        //전체 레이아웃 설정
        this.setLayout(new BorderLayout());

        MyPanel p1=new MyPanel();
        p1.setBackground(Color.cyan);

        //패널의 레이아웃 설정
        p1.setLayout(new GridLayout(1,2,10,10));

        bt1=new Button("버튼1");
        bt2=new Button("버튼2");

        p1.add(bt1);
        p1.add(bt2);
    }
}
```



# 예제

```
ta = new TextArea(5, 30);  
add(ta,"Center");  
add(p1,"South");
```

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

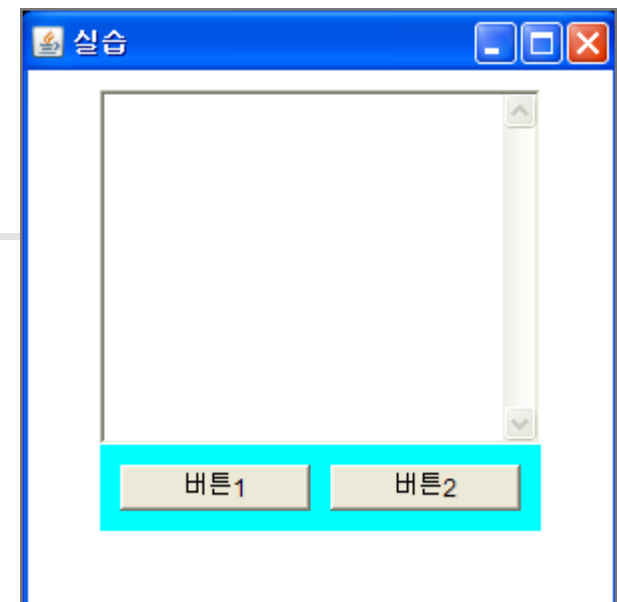
```
}
```

```
public Insets getInsets(){  
    //Insets(int top, int left, int bottom, int right)  
    return new Insets(40,40,40,40);  
} //getInsets()
```

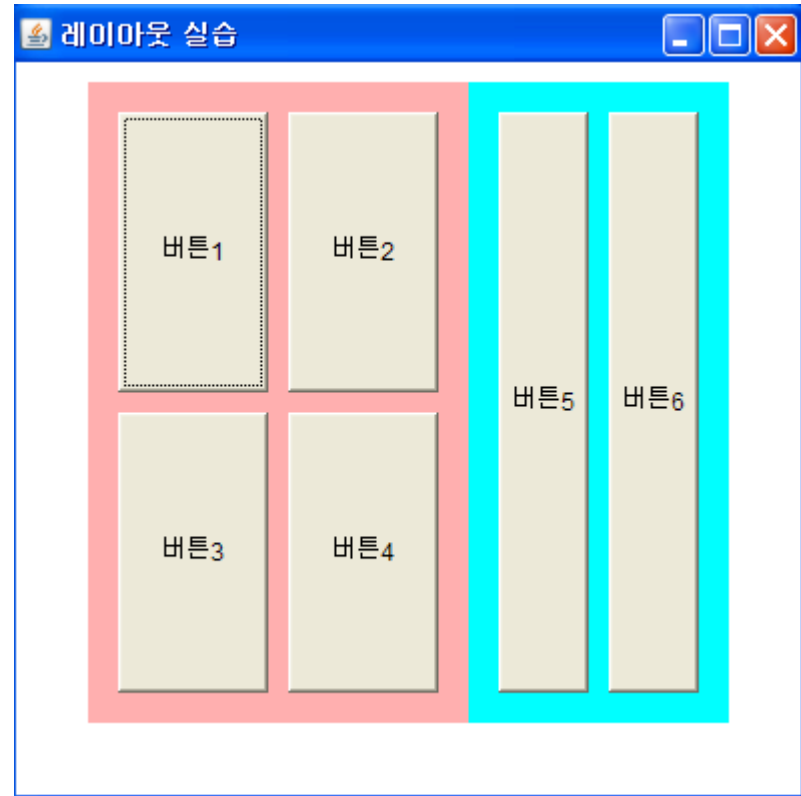
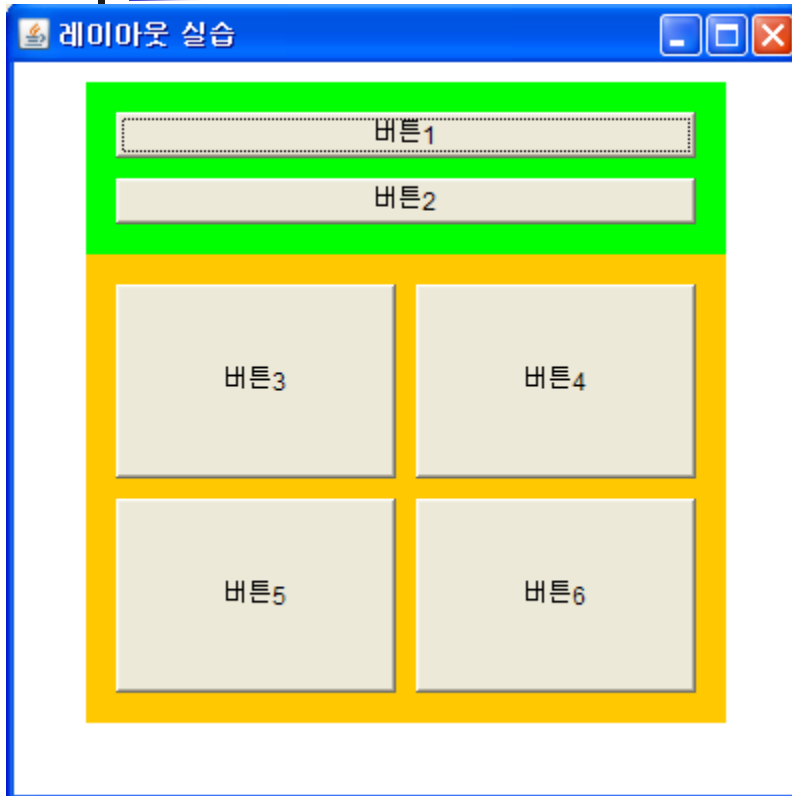
```
public static void main(String[] args){  
    InsetTest f = new InsetTest();  
} //main
```

```
} //
```

```
class MyPanel extends Panel  
{  
    public Insets getInsets(){  
        return new Insets(10,10,10,10);  
    }  
} //
```



# 실습



# 실습

레이아웃 실습

아이디

비밀번호

확인 취소

레이아웃 실습

이름

주민번호

확인 취소

게시판

글목록

글보기

글쓰기 새로고침 수정 삭제 종료