

Chương 5:

Abstract Window Toolkit Swing

Objectives

- Explain Abstract Window Toolkit (AWT)
- Explain various Containers and Components
 - Frame
 - Panel
 - Label
 - TextFields and TextAreas
 - Checkboxes and RadioButtons
 - Button
 - Choice
- Identify events generated by components
- Create a standalone AWT application

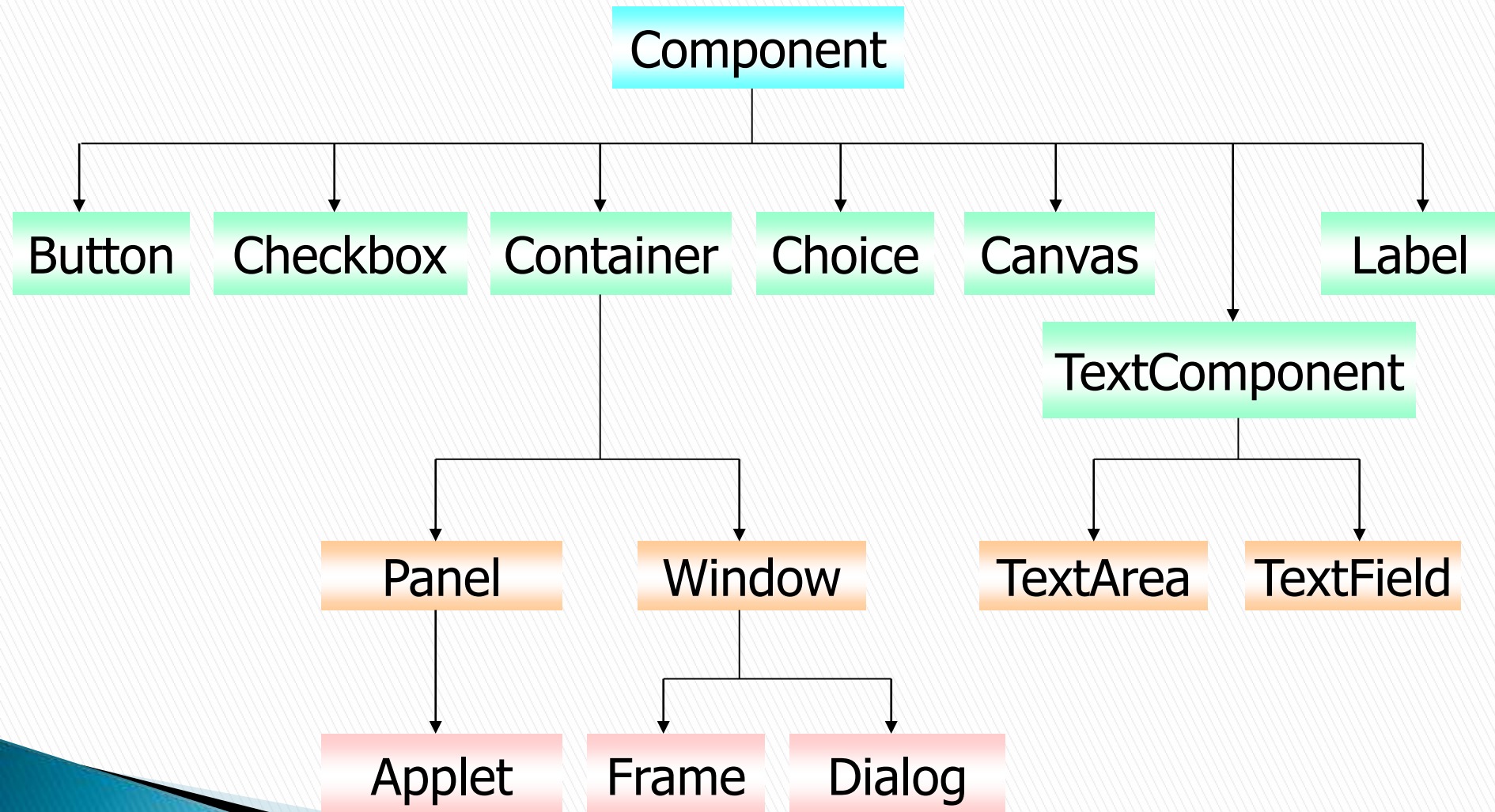
Abstract Window Toolkit - AWT

- ▶ **Graphical User Interface** (GUI) được dùng để nhận dữ liệu nhập theo một cách thức thân thiện với người dùng
- ▶ **Abstract Window Toolkit** (AWT) một tập các lớp trong Java cho phép tạo GUI và nhận dữ liệu nhập của người dùng qua chuột và bàn phím
- ▶ AWT cung cấp các item cho phép tạo giao diện hấp dẫn và hiệu quả

Types of GUI Programs

- ▶ Có 2 kiểu chương trình GUI trong Java:
 - Stand-alone Applications: chạy độc lập
 - Applets: chạy trên web page
 - ▶ GUI dùng các đối tượng (components) để tạo ra tương tác với người dùng
 - ▶ Để dùng Components:
 - import **awt** (Abstract Windows Toolkits) or
 - import **swing** component libraries: là sự mở rộng và cải tiến của AWT
- ```
Import java.awt.*;
Import javax.swing.*;
```

# Hierarchy of Component classes



# Component

## ▶ Component:

- Bất cứ gì có thể được đặt vào giao diện và có thể hiển thị hoặc thay đổi kích thước
- Một số ví dụ: Textfield, Label, Checkbox, Textarea, Button...

The image shows a Java Swing window with a form. The form is divided into several sections, each with a different background color. The sections are: a purple section for 'Name', a light blue section for 'Favourite sports', a light green section for 'Gender', a light pink section for 'Comments', and a red section for buttons. The 'Name' section contains a 'Label' and a 'Text field'. The 'Favourite sports' section contains three 'Checkbox' items: 'Cricket', 'Badminton', and 'Golf'. The 'Gender' section contains two 'Radio button' items: 'Male' and 'Female'. The 'Comments' section contains a 'Text Area'. The red section contains two 'Button' items: 'Submit' and 'Reset'. Arrows point from the labels on the right to the corresponding components in the form.

Label — **Name :** — Text field

**Favourite sports :** ☐ Cricket }  
☐ Badminton } — Checkbox  
☐ Golf }

**Gender :** ☐ Male }  
☐ Female } — Radio button

**Comments :** — Text Area

Submit Reset — Button

# Component class

- ▶ Component class
  - Cung cấp các phương thức cho nó và
  - cho các lớp con của nó:
- ▶ Phương thức:
  - Dimension getSize()
  - Void setSize(int w, int h)
  - Void setSize(Dimension p)
  - Point getLocation()
  - Void setLocation(int x, int y)
  - Void setLocation(Point p)

◦ ...

# Container

- ▶ **Container** là một vùng chứa các thành phần
- ▶ Lớp **Container** trong gói **java.awt** dẫn xuất ra hai **container** được sử dụng phổ biến nhất - **Frame** và **Panel**
  - **Frame** là một cửa sổ riêng biệt và có đường viền
  - **Panel** là một vùng không có đường viền, chứa trong một cửa sổ
- ▶ Cung cấp hàm **add()** được nạp chồng để bổ sung một thành phần vào lớp cho trước



# Frame

- ▶ Một cửa sổ độc lập, được sử dụng tạo ra ngưỡng windows cho các giao diện ứng dụng GUI
- ▶ Là lớp con của Window, có thanh tiêu đề, thanh thực đơn, đường viền và các góc có thể thay đổi kích thước
- ▶ Được tạo ra bằng cách dùng các constructor
  - **Frame()** :
    - Tạo 1 Frame ẩn không có tiêu đề
  - **Frame(String Title)**
    - Tạo 1 Frame ẩn có tiêu đề Title
- ▶ Để giúp `Frame` hiển thị, sử dụng phương thức **`setVisible()`**

# Frame

Để xây dựng một giao diện ứng dụng cần:

- ▶ Tạo 1 frame:
  - Ex, `Frame TestFrame = new Frame ("My Test Frame")`
- ▶ Dùng `add()` để thêm các thành phần GUI khác
  - Ex, `TestFrame.add(new Button("OK")) //optional`
- ▶ Đặt kích thước cho frame:
  - Ex, `TestFrame.setSize(300,400);// 300x400`
- ▶ Đóng gói frame
  - `TestFrame.pack() //optional`
- ▶ Cho hiện thị frame
  - `TestFrame.setVisible(true);`

# Ex: FrameDemo

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
class FrameDemo extends Frame {
public FrameDemo(String title)
{
 super(title);
}

public static void main (String args[])
{
 FrameDemo ObjFr = new FrameDemo(" My Test Frame!!!");
 ObjFr.setSize(300,200);
 ObjFr.add(new Button("OK"));
 ObjFr.setVisible(true);
}
}
```

```
addWindowListener(new
WindowAdapter() {
 public void
 windowClosing(WindowEvent we)
 {
 setVisible(false);
 System.exit(0);
 }
});
```

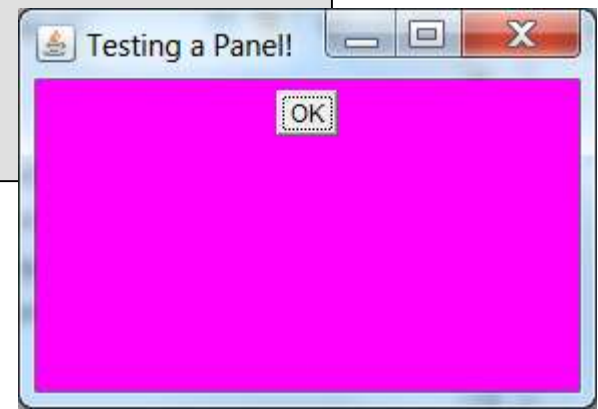


# Panel

- ▶ Được dùng để nhóm các thành phần lại với nhau
- ▶ Cách đơn giản nhất tạo ra panel là dùng constructor `Panel()`
- ▶ Vì panel không thể được nhìn thấy trực tiếp, nó phải được thêm vào một Frame

# Ex: PanelDemo

```
import java.awt.*;
class PanelDemo extends Panel {
public static void main(String args[]) {
 PanelDemo ObjPanel = new PanelDemo();
 Frame ObjFr = new Frame("Testing a Panel!");
 ObjFr.add(ObjPanel);
 ObjFr.setSize(300,200);
 ObjFr.setVisible(true);
}
public PanelDemo()
{
 setBackground (Color.magenta);
 add(new Button("OK"));
}
}
```



# Tạo các component của GUI

- ▶ Để sử dụng được các thành phần giao diện đồ họa, cần thực hiện:

- Tạo một thành phần GUI bởi 1 constructor thích hợp:

Ex: `Button btnOK = new Button("OK")`

- Bổ sung thành phần vừa tạo vào thành phần chứa container (ex, Frame):

Ex: `add(btnOK); //this.add(btnOK)`

- Nhận và xử lý sự kiện khi chúng xuất hiện:

Implement `ActionListener`, ...

Viết code.

# Label

- ▶ Chỉ ra mục đích của một item
- ▶ Người dùng không chỉnh sửa được
- ▶ Được tạo ra bằng cách dùng các constructor
  - **Label()**  
Tạo một nhãn không có nội dung
  - **Label(String labeltext)**  
Tạo một nhãn với nội dung là labeltext

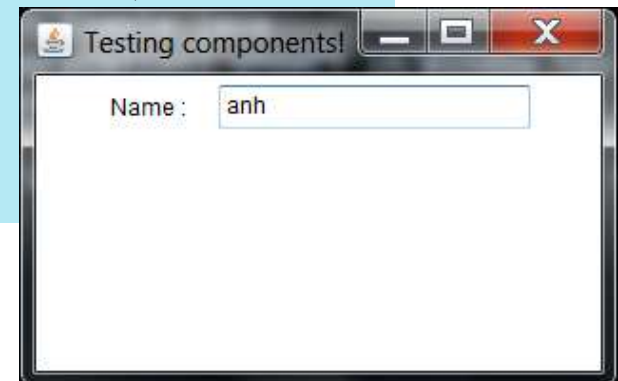
# TextField

- ▶ Dùng để nhập văn bản
- ▶ Chỉ nhập một dòng
- ▶ Được tạo ra bằng cách dùng các constructor
  - **TextField()**  
Tạo một textfield mới
  - **TextField(int columns)**  
Tạo một textfield mới có columns cột (layout)
  - **TextField(String s)**  
Tạo một textfield mới có nội dung s
  - **TextField(String s, int columns)**  
Tạo một textfield mới có nội dung s và columns cột



# Example: Label TextFieldDemo

```
import java.awt.*;
public class Label TextFieldDemo extends Frame {
 TextField txtName = new TextField(20);
 Label lblName = new Label("Name :");
 public Label_TextFieldDemo (String title)
 {
 super(title);
 setLayout(new FlowLayout());
 setSize(300,200);
 add(lblName);
 add(txtName);
 setVisible(true);
 }
 public static void main(String args[])
 {
 new Label_TextFieldDemo ("Testing components!");
 }
}
```



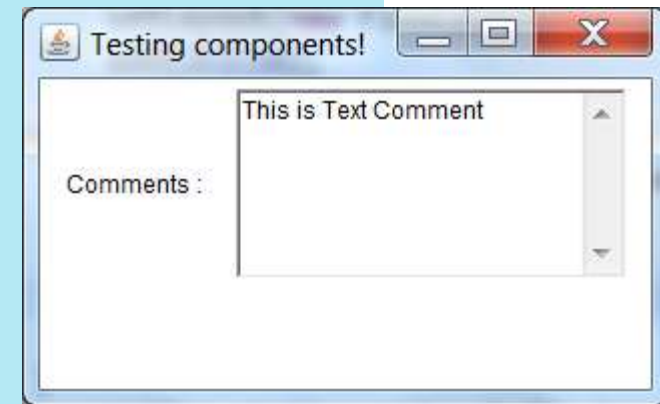
# TextArea

- ▶ Được dùng khi văn bản có thể nằm trên nhiều dòng
- ▶ TextArea được tạo ra bằng cách dùng các constructor
  - **TextArea()**  
Tạo một textarea mới
  - **TextArea(int rows, int cols)**  
Tạo một textarea mới với rows dòng và cols cột
  - **TextArea(String text)**  
Tạo một textarea mới với nội dung text
  - **TextArea(String text, int rows, int cols)**  
Tạo một textarea mới với nội dung text, rows dòng và cols cột

# Example: Text Area

```
import java.awt.*;
import java.awt.event.*;
class TextComments extends Frame {
 TextArea txtComment = new TextArea(5,25);
 Label lblCom = new Label("Comments :");
 public TextComments(String title) {
 super(title);
 setLayout(new FlowLayout());
 add(lblCom);
 add(txtComment);
 addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent we) {
 setVisible(false);
 System.exit(0);
 }
 });
 }

 public static void main(String args[]) {
 TextComments ObjComment = new TextComments("Testing components!");
 ObjComment.setSize(200,200);
 ObjComment.show();
 }
}
```



# Buttons

- ▶ Cách tốt nhất để ghi nhận các hoạt động của người dùng
- ▶ Được tạo ra bằng cách dùng các constructor
  - **Button()**  
Tạo một button mới
  - **Button(String text)**  
Tạo một button mới với nội dung text

# Example Button

```
public class ButtonDemo extends Frame{
 Button btnRed = new Button("Red!");
 Button btnBlue = new Button("Blue!");
 Button btnGreen = new Button("Green!");
 public ButtonDemo(String title) {
 super(title);
 setLayout(new FlowLayout());
 add(btnRed);
 add(btnBlue);
 add(btnGreen);
 setSize(300,200);
 setVisible(true);
 addWindowListener(new WindowAdapter() {
 public void windowClosing(WindowEvent we) {
 setVisible(false);
 System.exit(0);
 }
 });
 }
}
```

```
public static void main(String
args[]) {
 new ButtonDemo("The three buttons");
}
```



# Checkbox

- ▶ Dùng để nhận dữ liệu nhiều lựa chọn, người dùng có thể chọn hoặc bỏ chọn bằng cách click vào chúng
- ▶ Được tạo ra bằng cách dùng các constructor
  - **Checkbox()**  
Tạo một checkbox rỗng
  - **Checkbox(String text)**  
Tạo một checkbox với nội dung text
  - **Checkbox(String text, boolean on)**  
Tạo một checkbox với nội dung text và thiết lập trạng thái của checkbox là true hay false thông qua biến on

# Example: Checkbox

```
class CheckboxDemo extends Frame {
 Checkbox cboxRead = new Checkbox("Reading",false);
 Checkbox cboxMus = new Checkbox("Music",false);
 Checkbox cboxPaint = new Checkbox("Painting",false);
 Checkbox cboxMovie = new Checkbox("Movies",false);
 Checkbox cboxDance = new Checkbox("Dancing",false);
 Label lblQts = new Label("Enter the number of times you have done this activity");
 public CheckboxDemo(String str) {
 super(str);
 add(lblQts);
 add(cboxRead);
 add(cboxMus);
 add(cboxPaint);
 add(cboxMovie);
 add(cboxDance);
 setVisible(true);
 }
 public void windowClosing(WindowEvent we) {
 setVisible(false);
 System.exit(0);
 }
}

public static void main(String args[]) {
 new CheckboxDemo ("A basket full of checkboxes!");
}
```

Reading

Music

☒ Painting

☐ Movies

☐ Dancing

# Radiobuttons

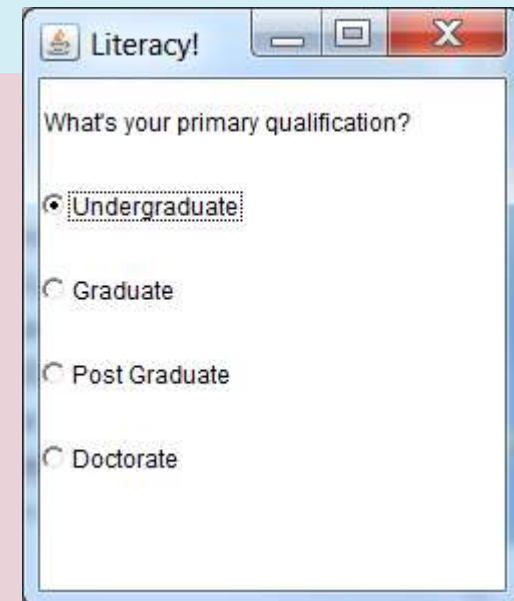
- ▶ Được dùng làm tùy chọn để xác định các chọn lựa
- ▶ Chỉ một thành phần trong group được chọn
- ▶ Trước tiên tạo ra một đối tượng CheckboxGroup
  - `CheckboxGroup cg = new CheckboxGroup();`
- ▶ Sau đó tạo ra từng radio button
  - `Checkbox male = Checkbox("male", cg, true);`
  - `Checkbox female =  
Checkbox("female", cg, false);`



# Example:RadioButtonDemo

```
class RadioButtonDemo extends Frame {
CheckboxGroup cg = new CheckboxGroup();
Checkbox radUnder = new Checkbox("Undergraduate",cg,false);
Checkbox radGra = new Checkbox("Graduate",cg,false);
Checkbox radPost = new Checkbox("Post Graduate",cg,false);
Checkbox radDoc = new Checkbox("Doctorate",cg,false);
Label lblQts = new Label("What's your primary qualification? ");
public RadioButtonDemo(String str) {
 super(str);
 setSize(250,300);
 setLayout(new GridLayout(6,1));
 add(lblQts);
 add(radUnder);
 add(radGra);
 add(radPost);
 add(radDoc);
 addWindowListener(new WindowAdapter(){
 public void windowClosing(WindowEvent we) {
 setVisible(false);
 System.exit(0);
 }
 });
 setVisible(true);
}
```

```
public static void main(String args[]){
 RadioButtonDemo Obj = new RadioButtonDemo
 ("Literacy!");
}
```



# Lists

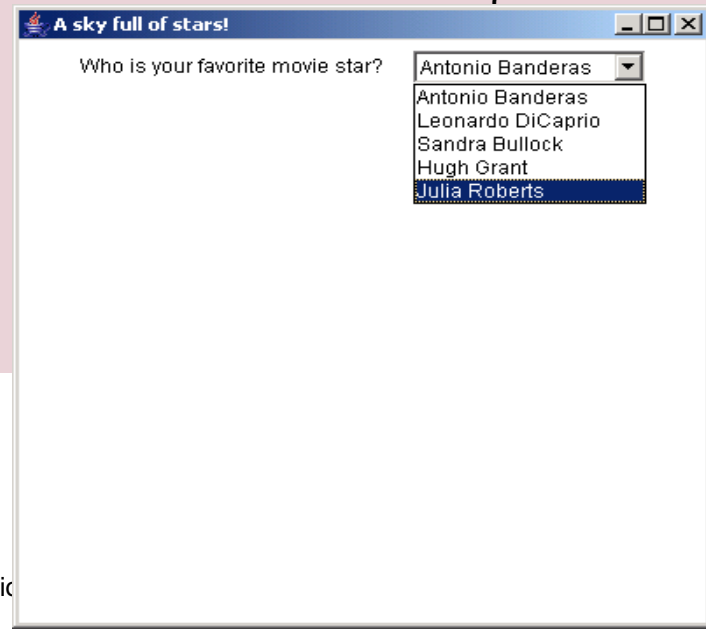
- ▶ Hiển thị một danh sách chọn lựa cho người dùng
- ▶ Lớp Choice: tạo ra danh sách nhiều item, ex
  - `Choice Names = new Choice();`
- ▶ Thêm item: dùng phương thức `addItem()`
  - `Names.addItem("Antonio");`
  - `Names.addItem("Leonardo");`
- ▶ Chọn 1 item: dùng `getSelectedItem()`
  - `Names.getSelectedItem() == "Antonio"`

# Example: List

```
import java.awt.*;
import java.awt.event.*;
class Stars extends Frame
{
 Choice moviestars = new Choice();
 Label lblQts = new Label("Who is your favorite movie star?");
 public Stars(String str)
 {
 super(str);
 setLayout(new FlowLayout());
 moviestars.addItem("Antonio Banderas");
 moviestars.addItem("Leonardo DiCaprio");
 moviestars.addItem("Sandra Bullock");
 moviestars.addItem("Hugh Grant");
 moviestars.addItem("Julia Roberts");
 add(lblQts);
 add(moviestars);
 }
}
```

```
addWindowList
{
 public v
{

}
});
}
public static v
{
```



# Menu

- ▶ Xây dựng hệ thống thực đơn cho chương trình
- ▶ Dùng các lớp MenuBar, Menu, MenuItem
- ▶ Abstract class MenuComponent is based class:
  - MenuBar class: define menu bar
  - MenuItem class: define Item in menu bar
  - Menu class: set up pull-down menu in a Item menu
  - PopUpMenu class: display for pop-up menu
  - CheckboxMenuItem class: contain items menu checked

# Tạo menu

Việc tạo lập một thanh thực đơn cho một *frame* được thực hiện như sau

1. Tạo ra một thanh thực đơn,

```
MenuBar thanhThDon = new MenuBar();
```

2. Tạo ra một thực đơn,

```
Menu thucDon = new Menu("Cac loai banh");
```

3. Tạo ra các mục trong thực đơn và đưa vào thực đơn,

```
MenuItem muc = new MenuItem("Banh day");
thucDon.add(muc); // Đưa muc vào thucDon
```

4. Đưa các thực đơn vào thanh thực đơn,

```
 thanhThDon.add(thucDon); // Đưa thucDon vào thanhThDon
```

5. Tạo ra một *frame* và đưa thanh thực đơn vào *frame* đó.

```
Frame frame = new Frame("Cac mon an");
frame.add(thanhThDon); // Đưa thanhThDon vào frame
```

# ex

```
import java.awt.*;
import java.awt.event.*;
class Calculator
{
 public static void main(String[] args)
 {
 // Tao Frame ung dung
 Frame fr = new Frame();
 fr.setLayout(new BorderLayout());
 // Tao cac menu bar
 MenuBar menu = new MenuBar();

 Menu menuEdit = new Menu("Edit");
 MenuItem copyItem = new MenuItem("Copy Ctrl+C");
 MenuItem pasteItem = new MenuItem("Paste Ctrl+V");
 menuEdit.add(copyItem);
 menuEdit.add(pasteItem);

 Menu menuHelp = new Menu("Help");
 MenuItem hTopicItem = new MenuItem("Help Topics");
 MenuItem hAboutItem = new MenuItem("About Calculator");
 menuHelp.add(hTopicItem);
 menuHelp.addSeparator();
 menuHelp.add(hAboutItem);

 menu.add(menuEdit);
 menu.add(menuHelp);
 fr.setMenuBar(menu);
 fr.setBounds(100, 100, 300, 200);
 fr.setTitle("Calculator");
 fr.setVisible(true);
 }
}
```



# Xử lý sự kiện trong GUI

- ▶ Mô hình xử lý sự kiện
- ▶ Các bước xử lý sự kiện
- ▶ Các lớp sự kiện
  - `ActionEvent`
  - `AdjustmentEvent`
  - `FocusEvent`
  - `ItemEvent`
  - `WindowEvent`
  - `MouseEvent`
  - `KeyEvent`

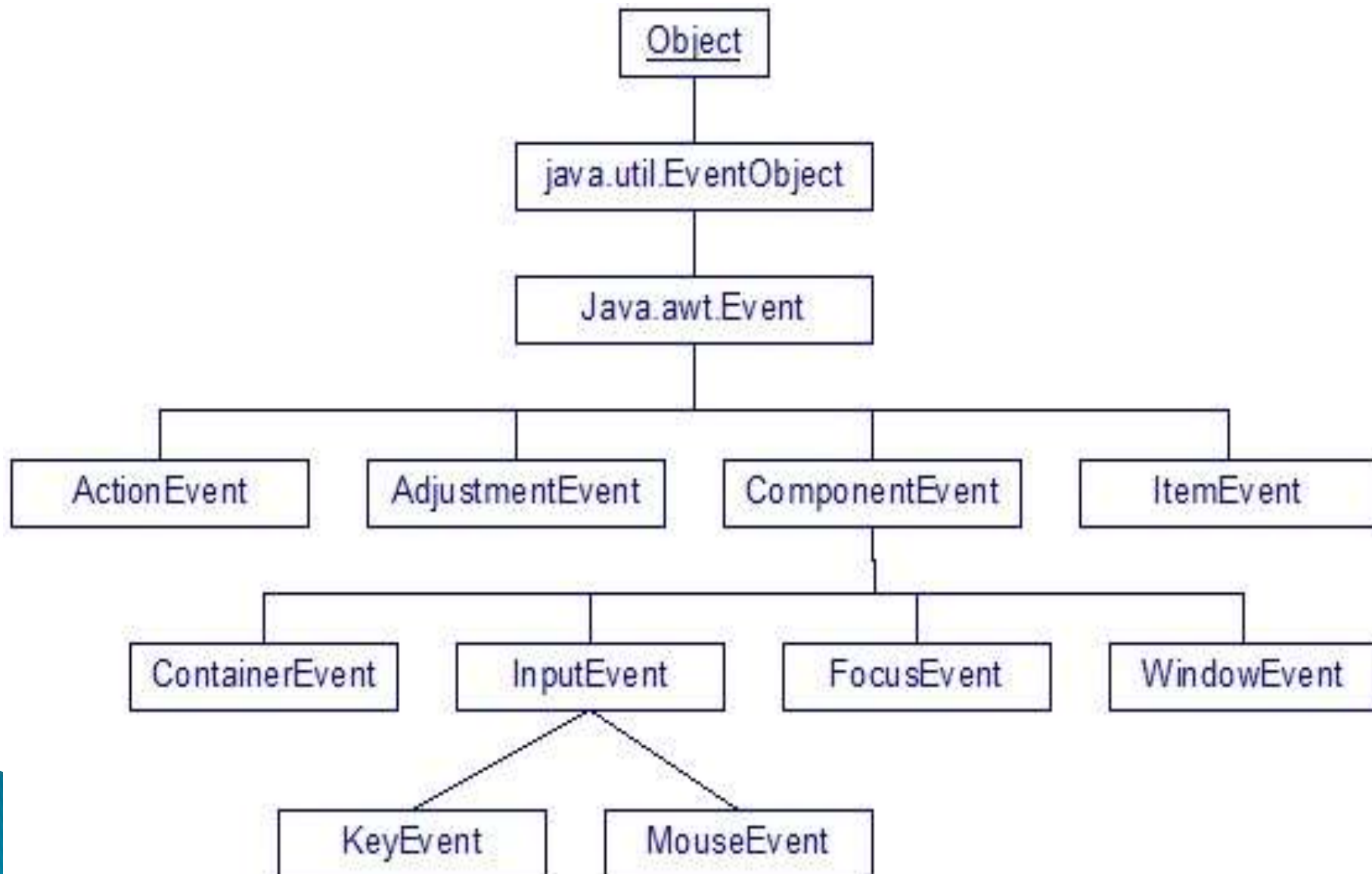


# Xử lý các sự kiện

- ▶ Các ứng dụng với GUI thường được hướng dẫn bởi các events.
- ▶ Trong Java, các events được thể hiện bằng các đối tượng.
- ▶ Hai nhóm:
  - Lớp mô tả về ngữ nghĩa của các events
  - Lớp events ở mức thấp



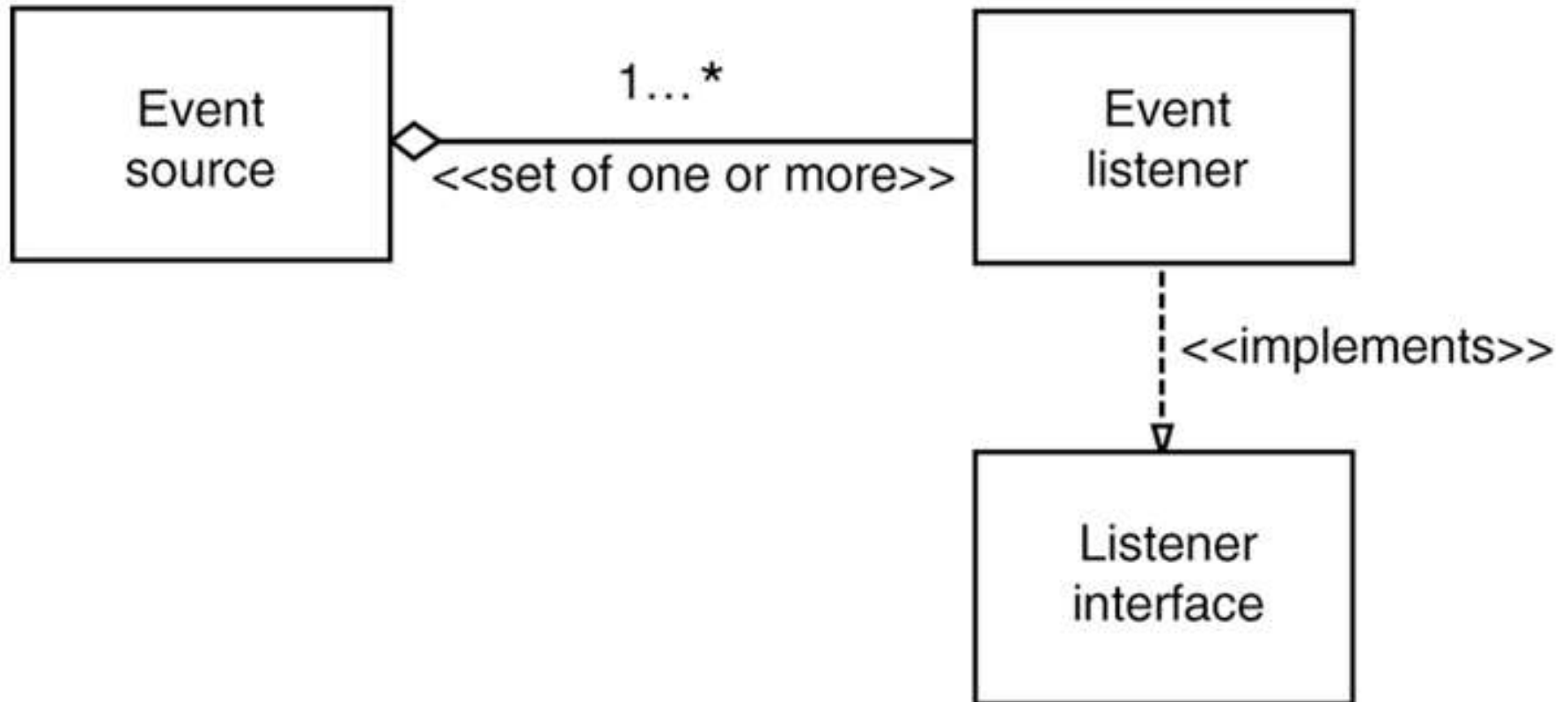
# Các lớp sự kiện trong gói Event



# Mô hình xử lý sự kiện

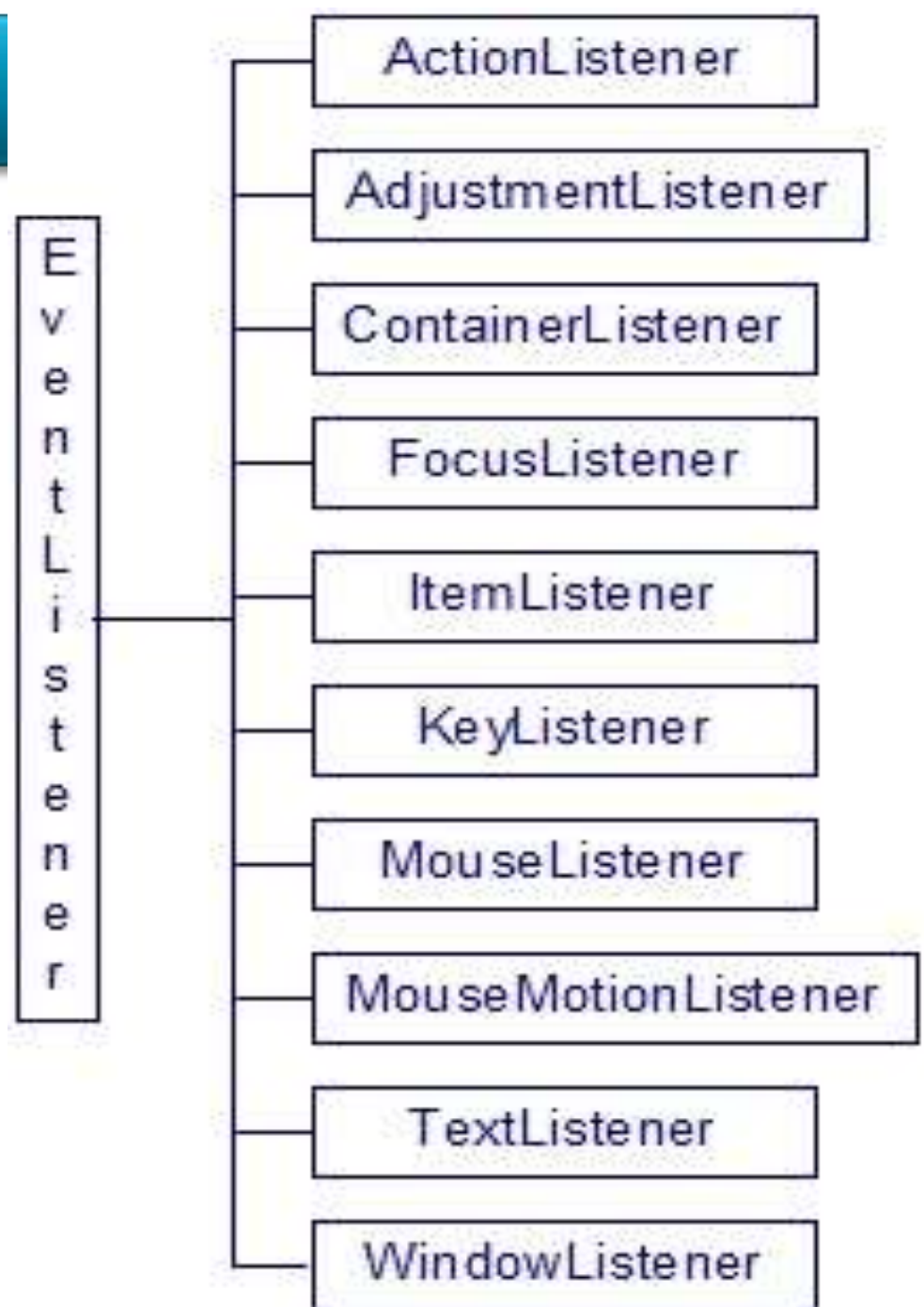
- ▶ Ba thành phần chính của mô hình
  - **Event source**: nguồn phát sinh sự kiện, là thành phần của giao diện mà người dùng tác động
    - Ex: click, move, drag....
  - **Event object**: đối tượng lưu thông tin về sự kiện
    - lưu trong một đối tượng sự kiện thuộc lớp con của lớp AWT. Event (gói java.awt.event)
  - **Event listener**: bộ lắng nghe sự kiện, nhận sự kiện và xử lý
    - Là cách chương trình có thể xử lý các sự kiện trên các thành phần của giao diện

# Mô hình xử lý sự kiện



# Event Listener

## ► Interface



# Các lớp sự kiện -> event source

| Lớp sự kiện     | Mô tả                                                      |
|-----------------|------------------------------------------------------------|
| ActionEvent     | Nhấn một nút, chọn menu, chọn radiobutton                  |
| AdjustmentEvent | Di chuyển thanh cuộn                                       |
| FocusEvent      | Đặt, chuyển focus                                          |
| ItemEvent       | Chọn 1 item trong list, chọn các checkbox                  |
| WindowEvent     | Đóng, mở, di chuyển cửa sổ                                 |
| MouseEvent      | Di chuyển, click, kéo thả chuột                            |
| KeyEvent        | Nhấn, thả bàn phím                                         |
| ComponentEvent  | thay đổi size, di chuyển, bị ẩn hay làm cho hoạt động được |
| TextEvent       | giá trị trong textfield /textarea thay đổi                 |

# Các bước xử lý sự kiện

## ► Những việc cần làm:

- Nhúng vào gói event

`import java.awt.event.*;`

- Cài đặt giao diện thích hợp với sự kiện muốn xử lý

ex, class abc extends Frame **implements ActionListener**

- Chỉ ra thành phần của giao diện muốn xử lý sự kiện trên đó

ex, button1.**addActionListener**(this)

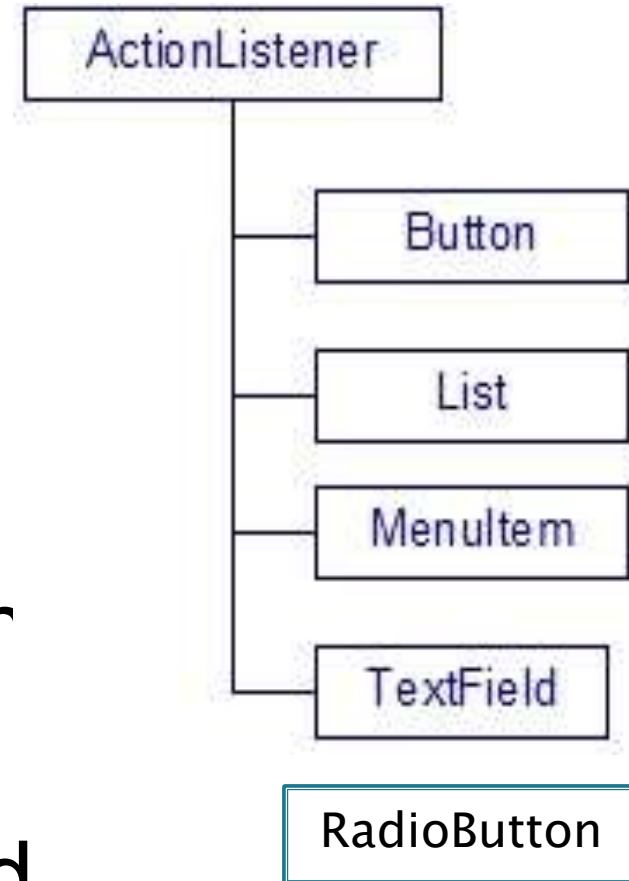
- Viết hàm xử lý sự kiện tương ứng

ex, public void **actionPerformed**(ActionEvent x)

```
{
}
```

# Lớp sự kiện `ActionEvent`

- ▶ Lớp: `ActionEvent`
- ▶ Giao diện: `ActionListener`
- ▶ Lắng nghe: `addActionListener`
- ▶ Hàm xử lý: `actionPerformed`





# Ví dụ: button

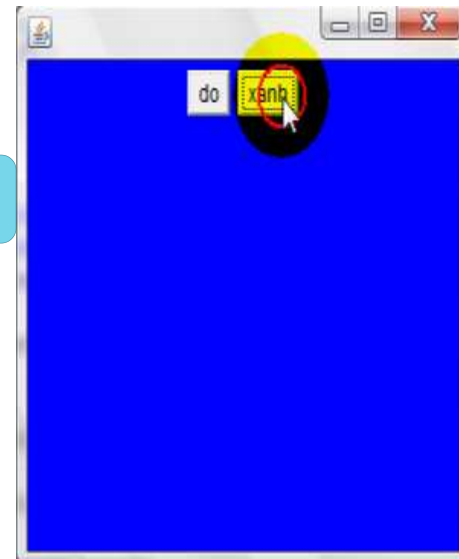
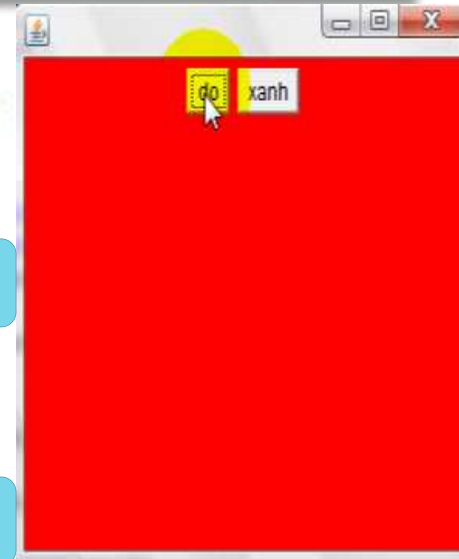
```
import java.awt.event.*;
public class bsk1 extends Frame implements ActionListener
{
 Button b1=new Button("do");
 Button b2=new Button("xanh");
 public bsk1()
 {
 this.add(b1);
 this.add(b2);
 b1.addActionListener(this);
 b2.addActionListener(this);
 this.setLayout(new FlowLayout());
 }
 public void actionPerformed(ActionEvent e)
 {
 if(e.getSource() == b1)
 this.setBackground(Color.red);
 else
 this.setBackground(Color.blue);
 }
 public static void main(String arg[])
 {
 bsk1 a=new bsk1();
 a.setSize(300,300);
 a.setVisible(true);
 }
}
```

Listener

Interface

Class

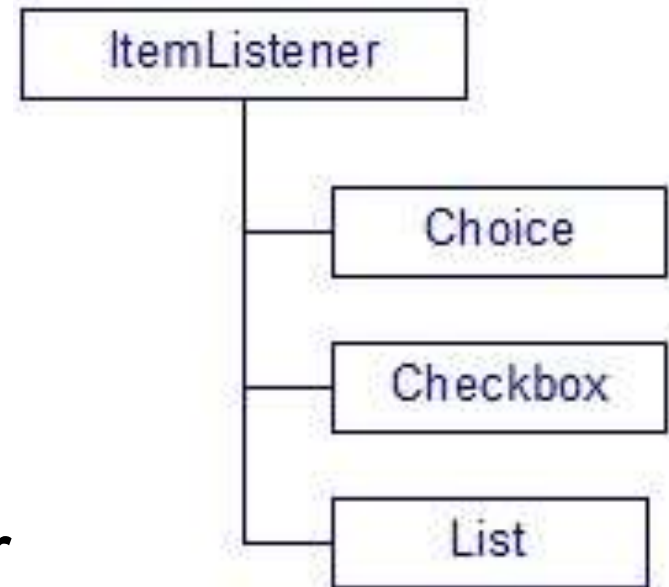
Action





# Lớp sự kiện ItemEvent

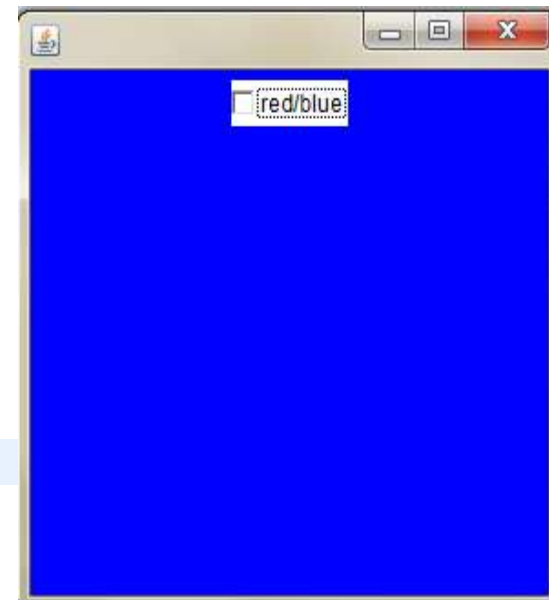
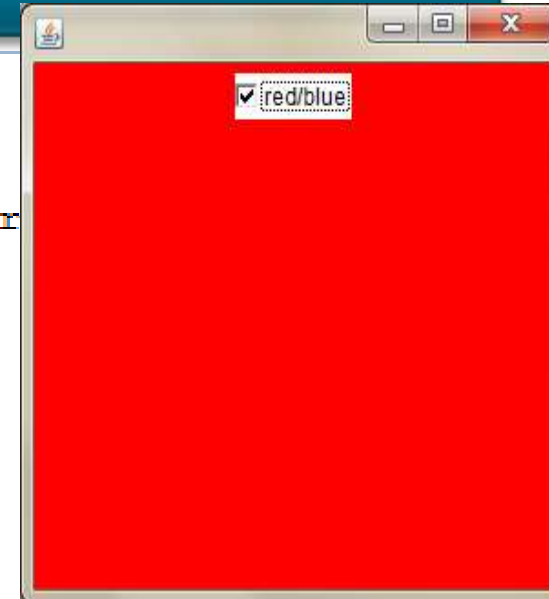
- ▶ Lớp: ItemEvent
- ▶ Giao diện: ItemListener
- ▶ Lắng nghe: addItemListener
- ▶ Hàm xử lý: itemStateChanged



# Ví dụ: checkbox

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

public class bsk1 extends Frame implements ItemListener
{
 Checkbox cb = new Checkbox("red/blue");
 public bsk1()
 {
 add(cb);
 cb.addItemListener(this);
 this.setLayout(new FlowLayout());
 }
 public void itemStateChanged(ItemEvent e)
 {
 if (e.getStateChange() == ItemEvent.SELECTED)
 this.setBackground(Color.red);
 else
 this.setBackground(Color.blue);
 }
 public static void main(String arg[])
 {
 bsk1 a = new bsk1();
 a.setSize(300,300);
 a.setVisible(true);
 }
}
```



# Lớp sự kiện MouseEvent

- ▶ Lớp: MouseEvent
- ▶ Giao diện: MouseListener
- ▶ Lắng nghe: addMouseListener
- ▶ Hàm xử lý: mouseClicked, mousePressed, mouseEntered, mouseExited, mouseReleased

# MouseEvent & MouseListener

**MouseListener** is used as a listener for mouse events, an object must implement this **MouseListener** interface.

The **MouseListener** interface specifies five different instance methods:

```
public void mousePressed(MouseEvent evt);
public void mouseReleased(MouseEvent evt);
public void mouseClicked(MouseEvent evt);
public void mouseEntered(MouseEvent evt);
public void mouseExited(MouseEvent evt);
```

# Ví dụ

```
public class bsk1 extends Frame implements MouseListener
{
 TextField tf2 = new TextField(10);
 Label lb2 = new Label("Toa do click chuot:");
 public bsk1()
 {
 add(lb2);
 add(tf2);
 this.addMouseListener(this);
 this.setLayout(new FlowLayout());
 }
 public void mouseClicked(MouseEvent e)
 {
 int x = e.getX();
 int y = e.getY();
 tf2.setText(String.valueOf(x)+":"+String.valueOf(y));
 }
 public void mousePressed(MouseEvent e)
 {
 }
 public void mouseEntered(MouseEvent e)
 {
 }
 public void mouseExited(MouseEvent e)
 {
 }
 public void mouseReleased(MouseEvent e)
 {
 }
 public static void main(String arg[])
 {
 bsk1 a = new bsk1();
 a.setSize(300,300);
 a.setVisible(true);
 }
}
```



# MouseMotionListener

**MouseMotionListener** is used as a listener for mouse motion events, an object must implement this MouseMotionListener interface to handle motion event

The interface specifies two different instance methods:

```
public void mouseDragged(MouseEvent
evt);
```

```
public void mouseMoved(MouseEvent evt);
```

# Example : Handle Mause

```
/* <applet code = Mousey width = 400 height = 400>
</applet>*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
public class Mousey extends Applet implements
MouseListener, MouseMotionListener
{
 int x1, y1, x2, y2;
 public void init()
 {
 setLayout(new FlowLayout());
 setBounds(100,100,300,300);
 addMouseListener(this);
 addMouseMotionListener(this);
 this.setVisible(true);
 }
 public void mouseClicked(MouseEvent e)
 {
 }
 public void mousePressed(MouseEvent e)
 {
 x1 = e.getX();
 y1 = e.getY();
 }
}
```

```
public void mouseMoved(MouseEvent e)
{
}

public void mouseReleased(MouseEvent e)
{
 x2 = e.getX();
 y2 = e.getY();
 repaint();
}

public void mouseEntered(MouseEvent e)
{
}
public void mouseDragged(MouseEvent e)
{
}
public void mouseExited(MouseEvent e)
{
}

public void paint(Graphics g)
{
 g.drawRect(x1, y1, x2-x1, y2-y1);
 x2 = 0;
 y2 = 0;
}
}
```

# Example : rect\_mouse

```
/*
<applet code=Painting width=400 height=400>
</applet>
*/

import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class Painting extends Applet implements ActionListener
MouseListener
{
 Button bdraw = new Button("Draw Rectangle");
 int count = 0,x1,x2,x3,x4;
 public void init()
 {
 BorderLayout border = new BorderLayout();
 setLayout(border);
 add(bdraw, BorderLayout.PAGE_END);
 bdraw.addActionListener(this);
 addMouseListener(this);
 this.setVisible(true);
 }
 public void mouseClicked(MouseEvent e)
 {}
}
```

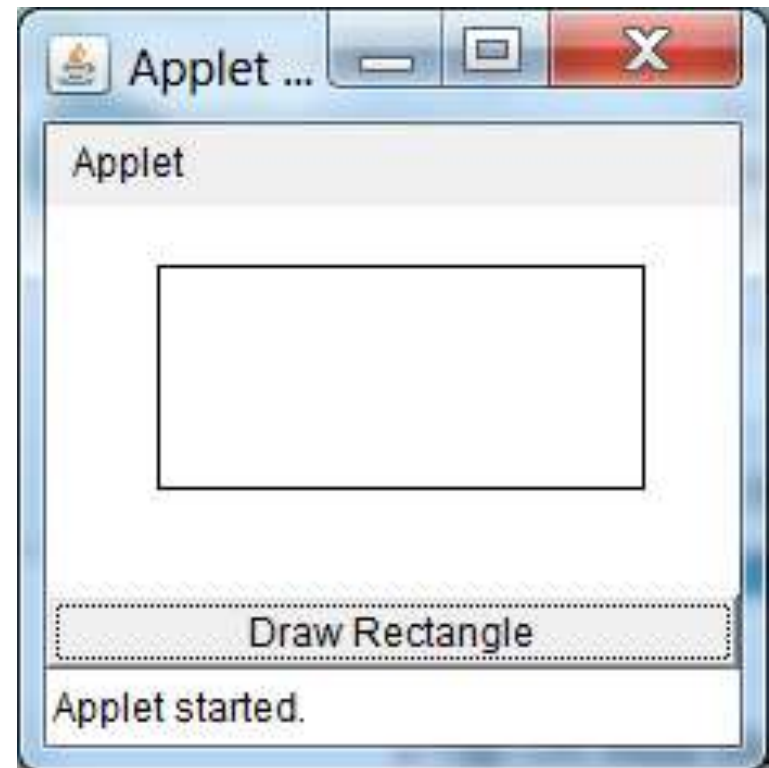
```
public void mousePressed(MouseEvent e)
{
 x1 = e.getX();
 x2 = e.getY();
}
public void mouseMove(MouseEvent e)
{
 x3 = e.getX();
 x4 = e.getY();
 repaint();
}
public void mouseReleased(MouseEvent e)
{
 x3 = e.getX();
 x4 = e.getY();
 repaint();
}
public void mouseEntered(MouseEvent e)
{
}
public void mouseExited(MouseEvent e)
{
}
```



# Example Contd...

```
public void actionPerformed(ActionEvent e)
{
 String str = e.getActionCommand();
 if("Draw Rectangle".equals(str))
 {
 count = 1;
 repaint();
 }
}

public void paint(Graphics g)
{
 if(count == 1)
 {
 g.drawRect(x1,x2,(x3-x1),(x4-
x2));
 x3 = x4 = 0;
 }
}
}
```



# Event handling with Components

- ▶ Events are generated whenever a user clicks a mouse, presses or releases a key.
- ▶ Event Delegation Model is used to handle events.
- ▶ This model allows the program to register handlers called 'listeners' with objects whose events need to be captured.
- ▶ Handlers are automatically called when an event takes place.
- ▶ Action that has to be done following an event is performed by the respective listener.

# Các lớp bố trí và sắp xếp các tp GUI

- ▶ **FlowLayout**
  - Xếp các tp theo hàng/cột từ trái/trên
  - Mặc định với Panel và Applet
- ▶ **GridLayout**
  - Xếp trong ô lưới: 1 tp 1 ô
- ▶ **BorderLayout**
  - Xếp các thành phần (<5) theo các vị trí: N-S-W-E
  - Mặc định với Window, Frame, Dialog
- ▶ **GridBagLayout**
  - Xếp các tp vào ô lưới: 1 tp chiếm nhiều ô

# FlowLayout

- Constructors for the FlowLayout are :  
`FlowLayout mylayout = new FlowLayout();`  
`FlowLayout exLayout = new  
FlowLayout(FlowLayout.LEFT) ;`  
Ex: `setLayout(new FlowLayout());`

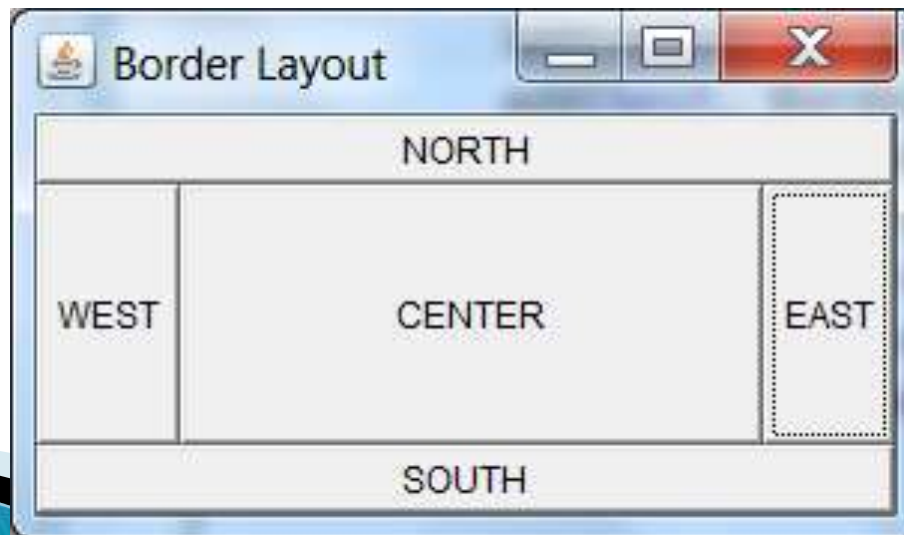


# BorderLayout Manager

- Các giá trị hằng cho phép định vị các component trong BorderLayout
  - **PAGE\_START**: đỉnh container
  - **LINE\_END**: bên phải container
  - **PAGE\_END**: đáy container
  - **LINE\_START**: bên trái container
  - **LINE\_CENTER**: giữa container

# Ví dụ: BorderLayout

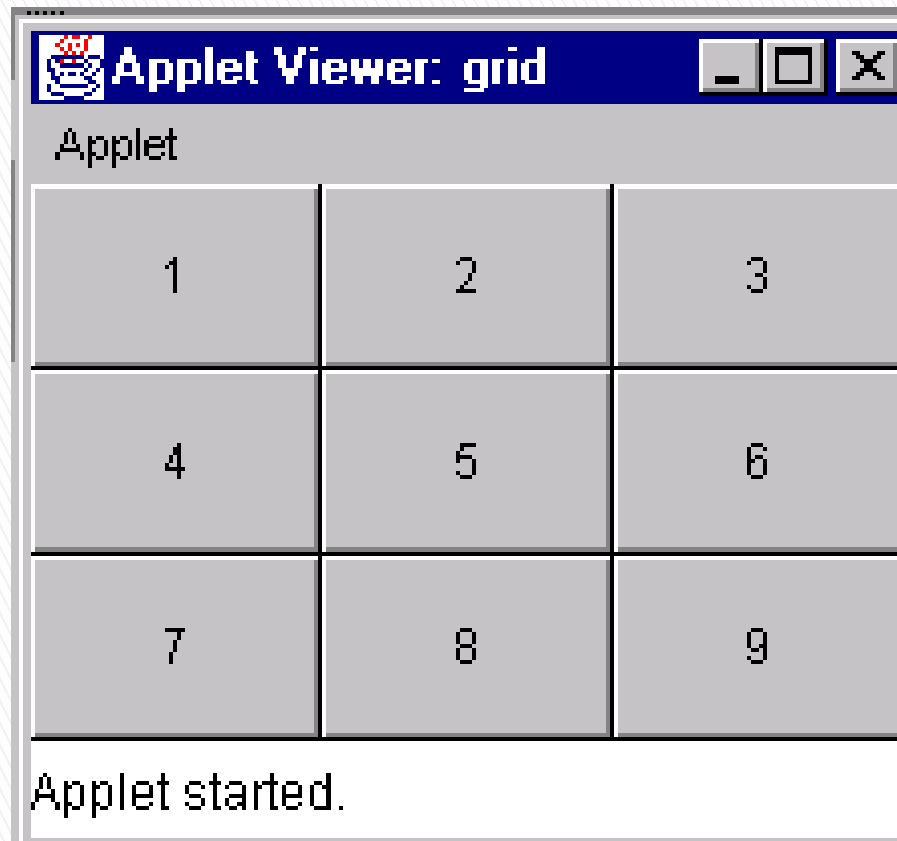
```
setLayout(new BorderLayout());
add(best, BorderLayout.LINE_END);
add(bwest, BorderLayout.LINE_START);
add(bnorth, BorderLayout.PAGE_START);
add(bsouth, BorderLayout.PAGE_END);
add(bcentre, BorderLayout.CENTER);
```



# GridLayout Manager

- ▶ Chia container thành lưới hình chữ nhật
- ▶ Các component được sắp xếp theo hàng và cột
- ▶ Dùng khi tất cả các component có cùng kích thước
- ▶ Constructor
  - `GridLayout g1= new GridLayout (4, 3);`

# GridLayout





# GridBagLayout Manager

- ▶ Các component không cần có cùng kích thước
- ▶ Các component được sắp theo hàng và cột
- ▶ Thứ tự đặt các component không phải từ trên xuống hay từ trái qua
- ▶ Một container được thiết lập GridBagLayout :  

```
GridBagLayout gb = new GridBagLayout();
ContainerName.setLayout(gb);
```

# GridBagLayout Manager

- ▶ Để dùng layout này, thông tin phải được cung cấp là kích thước và layout của mỗi thành phần
- ▶ Lớp `GridBagConstraints` giữ mọi thông tin được yêu cầu bởi lớp `GridBagLayout` đến vị trí và kích thước của mỗi component

# GridBagLayout Manager

- ▶ Danh sách biến đối tượng của lớp `GridBagConstraints`
  - `gridx`, `gridy`: vị trí ô đưa component vào (cột, hàng)
  - `gridwidth`, `gridheight`: kích thước của component, chứa mấy ô theo chiều ngang (width) và dọc (height)
  - `Fill`: xác định làm thế nào một component lấp đầy ô nếu ô lớn hơn component
    - `GridBagConstraints.NONE` giữ nguyên size đối tượng.
    - `GridBagConstraints.VERTICAL` giãn chiều cao cho khít
    - `GridBagConstraints.HORIZONTAL` giãn chiều ngang của đối tượng cho khít
    - `GridBagConstraints.BOTH` giãn cả 2 chiều của đối tượng cho khít

# GridBagLayout Manager

- ▶ Danh sách biến đối tượng của lớp `GridBagConstraints`
  - `ipadx`, `ipady`: tăng kích thước hai bên trái phải (hoặc trên dưới) component thêm `ipadx` (`ipady`) pixel khi container thay đổi kích thước
  - `Insets`: xác định khoảng trống giữa các component trên đỉnh, đáy, trái và phải
  - `Anchor`: neo component vào vị trí nào đó so với ô đặt nó: `NORTH`, `NORTHEAST`, `NORTHWEST`, `WEST`, `EAST`, `SOUTH`, `SOUTHEAST`, `SOUTHWEST`
  - `weightx`, `weighty`: khoảng cách lớn ra tương đối giữa các đối tượng

```
import java.awt.*;
import java.applet.Applet;
public class MyGridBag extends Applet
{
 TextArea ObjTa;
 TextField ObjTf;
 Button butta, buttf;
 CheckboxGroup cbg;
 Checkbox cbbold,cbitalic,cbplain,cbboth;
 GridBagLayout gb;
 GridBagConstraints gbc;
 public void init()
 {
 gb = new GridBagLayout();
 setLayout(gb);
 gbc = new GridBagConstraints();
 ObjTa = new TextArea("Textarea ",5,10);
 ObjTf = new TextField("enter your name");
 butta = new Button("TextArea");
 buttf = new Button("TextField");
```

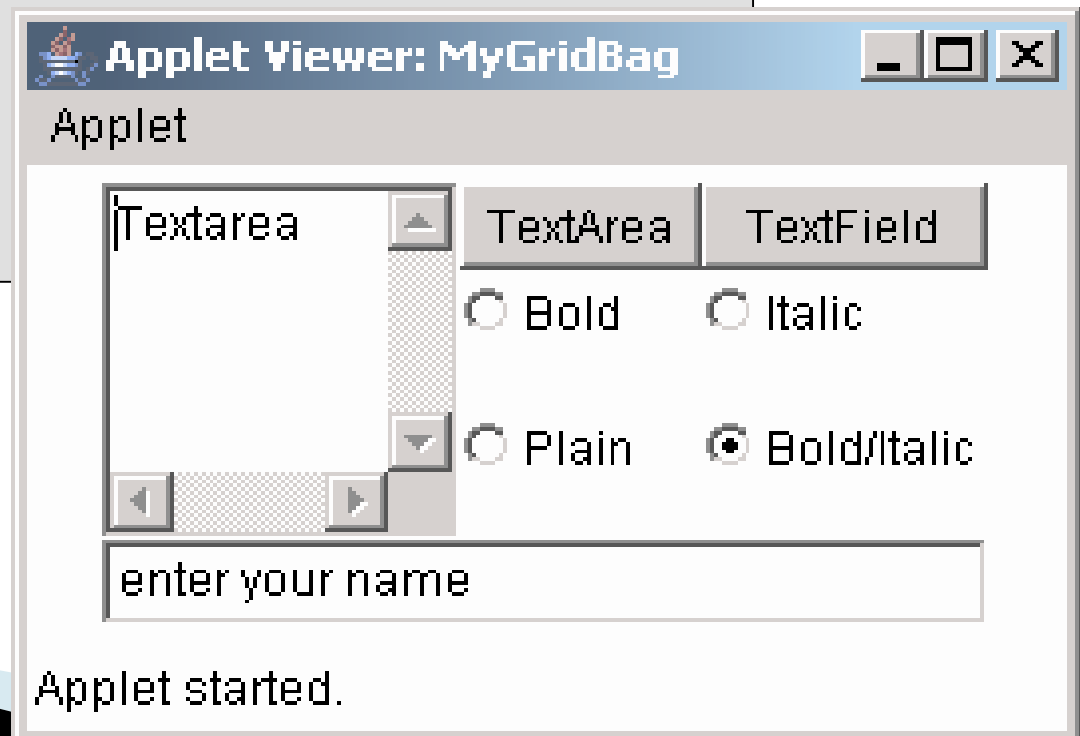
```
 cbg = new CheckboxGroup();
 cbbold = new Checkbox("Bold",cbg,false);
 cbitalic = new Checkbox("Italic",cbg,false);
 cbplain = new Checkbox("Plain",cbg,false);
 cbboth = new Checkbox("Bold/Italic",cbg,true);
 gbc.fill = GridBagConstraints.BOTH;
 addComponent(ObjTa,0,0,4,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 addComponent(butta,0,1,1,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 addComponent(buttf,0,2,1,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 addComponent(cbbold,2,1,1,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 addComponent(cbitalic,2,2,1,1);
 gbc.fill = GridBagConstraints.HORIZONTAL;
 addComponent(cbplain,3,1,1,1);
```

# Ex, cont

```
gbc.fill = GridBagConstraints.HORIZONTAL;
addComponent(cbboth,3,2,1,1);
gbc.fill = GridBagConstraints.HORIZONTAL;
addComponent(ObjTf,4,0,1,3);
}
public void addComponent(Component comp, int row, int col, int nrow, int ncol)
{
 gbc.gridx = col;
 gbc.gridy = row;

 gbc.gridwidth = ncol;
 gbc.gridheight = nrow;

 gb.setConstraints(comp,gbc);
 add(comp);
}
}
```



# Using Swing



# Introduction

- ▶ Swing is preferable over AWT as more lightweight and is newer.
- ▶ Swing provides some classes support for GUI



# Swing components

## Top-level Container

- JFrame (Form Application)
- JApplet (Applet)
- JDialog

## Intermediate-level Container

- JPanel
- JTabbedPane
- JSplit

## Atomic components

- JLabel, JTextField, JTextArea, JButton, JComboBox, JList, JCheckBox, JSlider



# Simple application

```
import javax.swing.*;
public class FrameApp
extends JFrame {
 FrameApp() {
 super("My Frame");
 setDefaultCloseOperation(
 JFrame.EXIT_ON_CLOSE);
 setVisible(true);
 }
 public static void
 main(String[] args) {
 new FrameApp();
 }
}
```

```
import javax.swing.*;
public class FrameApp {
 public static void
 main(String[] args) {
 JFrame w = new JFrame();
 w. setDefaultCloseOperation(
 JFrame.EXIT_ON_CLOSE);
 w.setVisible(true);
 }
}
```

# Setting up GUI

- ▶ To create a Java program that uses a GUI, we must:
  - define and set up the necessary components,
  - create listener objects and establish the relationship between the listeners and the components which generate the events of interest, and
  - define what happens as a result of the various user interactions that could occur.

# Setting up GUI

- ▶ Create a container and assign a layout manager to it,
- ▶ Create components and add them to the container,
- ▶ Use the container as the content pane of a window or applet.

# Detail

- ▶ JFrame
- ▶ JPanel
- ▶ JSlider
- ▶ JTabbedPane
- ▶ JOptionPane (Dialog)
- ▶ JSplitPane

# Container

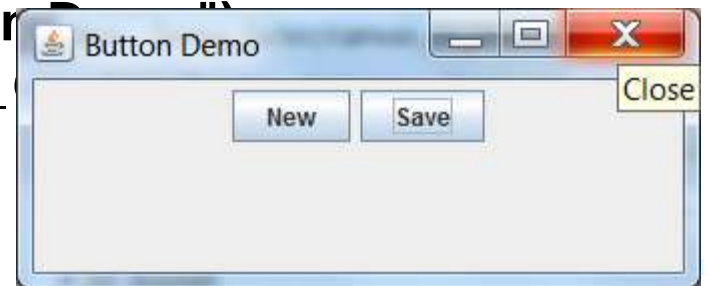
- ▶ A basic **JFrame** has a blank **content pane**;
- ▶ One can either:
  - Add things to that pane or
  - Replace the basic content pane entirely with another container.
- ▶ **JPanel** is one of fundamental classes in Swing. The basic JPanel is just a blank rectangle.

There are two ways to make a useful JPanel:

1. The first is to **add other components** to the panel;
  2. The second is to **draw something** on the panel.
- ▶ **JTabbedPane**

# Ex: JFrame as a container

```
import java.awt.FlowLayout;
import javax.swing.*;
public class content_ex extends JFrame {
 content_ex(String title){
 super(title);
 JButton a = new JButton("New");
 JButton b = new JButton("Save");
 setLayout(new FlowLayout());
 setSize(200,300);
 this.add(a);
 this.add(b);
 setVisible(true);
 }
 public static void main(String[] args) {
 content_ex w = new content_ex("Button Demo");
 w.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```



# Jpanel as a container

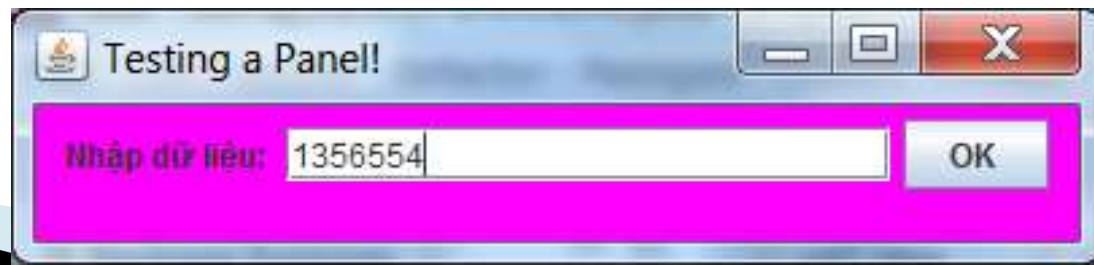
- ▶ Another way of using a JPanel is as a container to hold other components.
- ▶ Java has many classes that define GUI components. Before these components can appear on the screen, they must be added to a container.



# Ex: Jpanel as a container

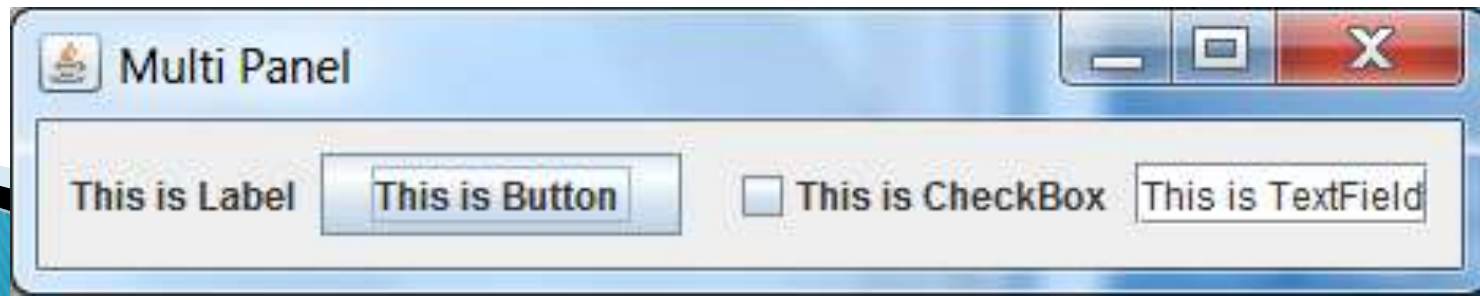
```
import java.awt.*;
import javax.swing.*;
class PanelDemo extends JPanel {
public static void main(String args[]){
 PanelDemo ObjPanel = new PanelDemo();
 JFrame ObjFr = new JFrame("Testing a Panel!");
 ObjFr.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 ObjFr.add(ObjPanel);
 ObjFr.setSize(400,400);
 ObjFr.setVisible(true);
 }

 public PanelDemo() {
 setBackground (Color.magenta);
 add(new JLabel("Nhập dữ liệu: "));
 add(new JTextField(20));
 add(new JButton("OK "));
 }
}
```



# Ex: MultiPanel

```
public class TitleDemo {
 public static void main (String[] args) {
 JFrame frame = new JFrame ("Multi Panel");
 frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 JPanel panel = new JPanel();
 JPanel p1 = new JPanel();
 p1.add (new JLabel ("This is Label "));
 p1.add (new JButton ("This is Button "));
 panel.add (p1);
 JPanel p2 = new JPanel();
 p2.add (new JCheckBox ("This is CheckBox"));
 p2.add (new JTextField ("This is TextField"));
 panel.add (p2);
 frame.setContentPane(panel); //frame.getContentPane().add (panel);
 frame.pack();
 frame.show();
 }
}
```



# Tabbed pane

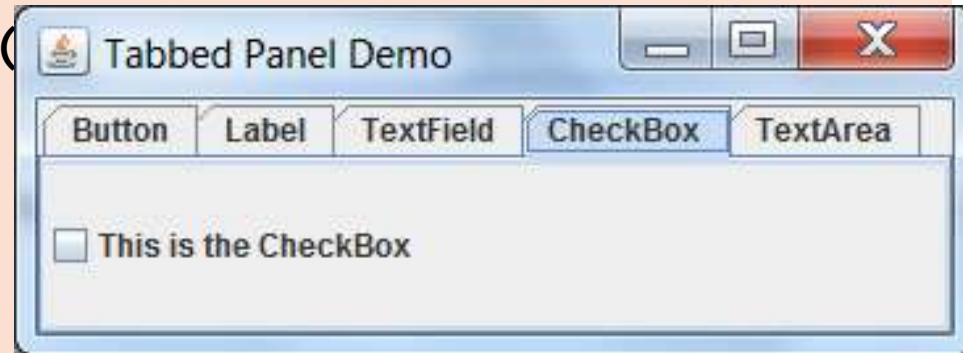
- ▶ Tabbed pane is a container that allows the user to select (by clicking on a tab) which of several panes are currently visible.
- ▶ A tabbed pane is defined by the **JTabbedPane** class.
- ▶ The **addTab** method creates a tab, specifying the name that appears on the tab and the component to be displayed on that pane when it achieves focus by being “brought to the front” and made visible to the user.

# Ex: TabbedPaneDemo

```
import javax.swing.*;

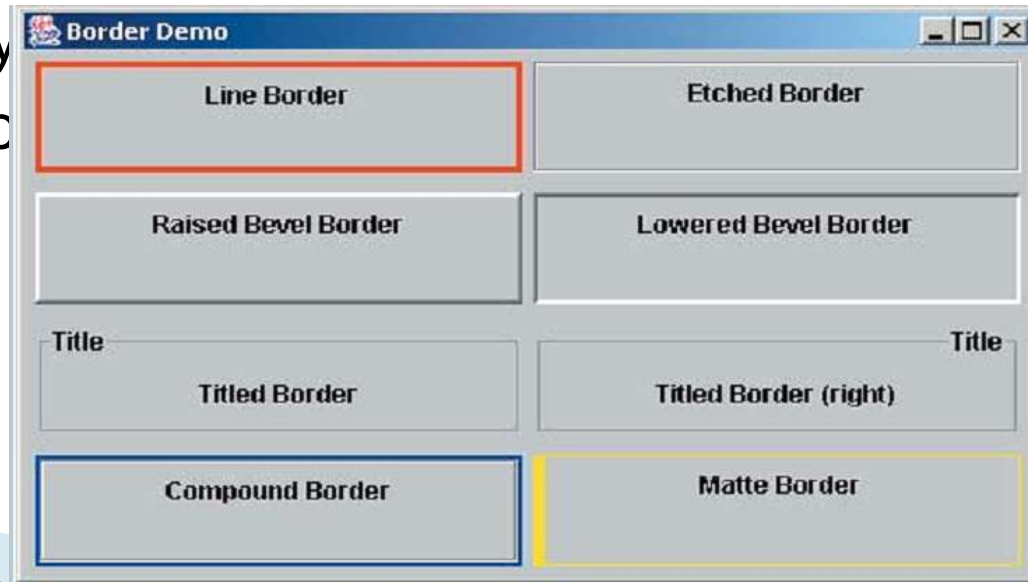
public class TabbedPaneDemo extends JTabbedPane {
 public static void main (String[] args) {
 JFrame frame = new JFrame ("Tabbed Panel Demo");
 frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 frame.setSize(500,100);
 TabbedPaneDemo tp = new TabbedPaneDemo();
 frame.getContentPane().add(tp);
 frame.setVisible(true);
 }

 TabbedPaneDemo(){
 addTab("Button", new JButton("This is the Button"));
 addTab ("Label", new JLabel("This is the Label"));
 addTab ("TextField", new JTextField("This is the TextField"));
 addTab ("CheckBox", new JCheckBox("This is the CheckBox"));
 addTab ("TextArea", new JTextArea("This is the TextArea"));
 }
}
```



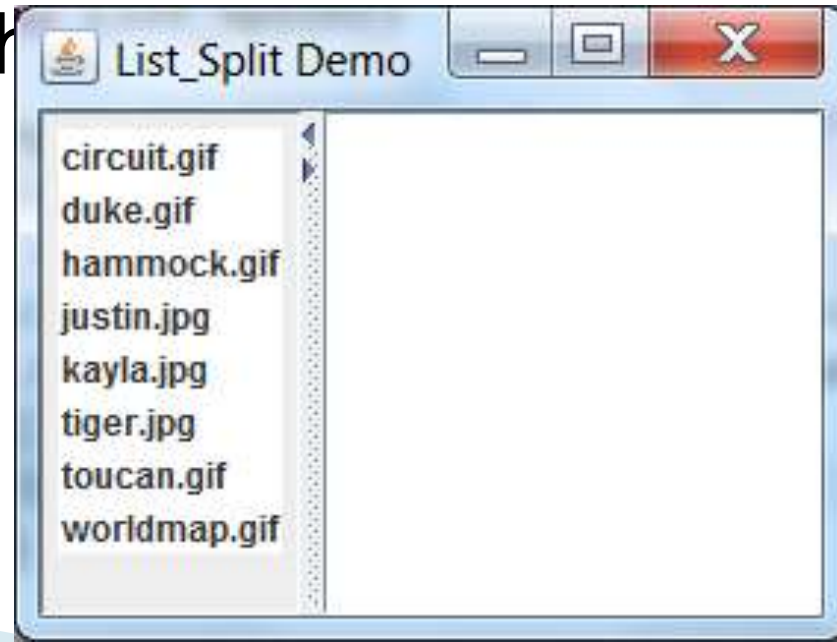
# Border

- ▶ Java also provides the ability to put a border around any Swing component.
- ▶ The `BorderFactory` class is useful for creating borders for components.
- ▶ Set border for a component: `setBorder()`
- ❖ EX:
  - ▶ `Border b1 = BorderFactory.createLineBorder (Color.blue, 2);`
  - ▶ `Border b2 = BorderFactory`
  - ▶ `p7.setBorder (BorderFacto`
  - ▶ `b2));`



# Split panes

- ▶ A split pane is a container that displays two components separated by a moveable divider bar.
- ▶ Depending on how the split pane is set up, the two components are displayed either side by side or one on top of the other.
- ▶ In Java, we create a split pane using the **JSplitPane** class.



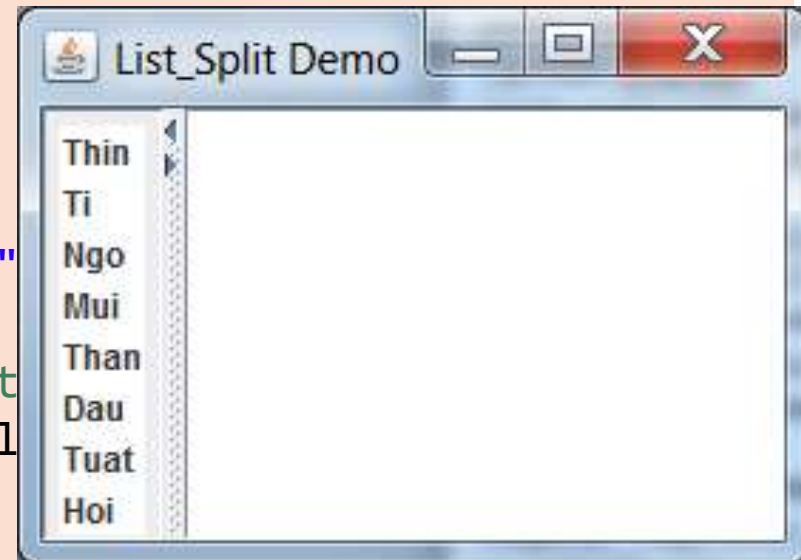


# Lists

- ▶ The **Swing JList class** represents a list of items from which the user can choose.
- ▶ In general, all of the options in a list are visible. When the user selects an item using the mouse, it is highlighted.
- ▶ `getSelectedValue`: returns the item selected.
- ▶ Event Handling: The `ListSelectionListener` interface contains one method called `valueChanged`.

# Ex: List\_Split

```
public class List_Split extends JPanel {
 private JList list;
 public static void main (String[] args) {
 JFrame frame = new JFrame ("List_Split Demo");
 frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 JPanel Pa = new JPanel(); //tp panel phai
 imagePanel.setBackground (Color.white);
 List_Split Li = new List_Split (); //tp List ben trai
 JSplitPane sp = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
 Li, Pa); //2 tp Panel va List
 sp.setOneTouchExpandable (true);
 frame.getContentPane().add (sp);
 frame.pack(); frame.show();
 }
 public List_Split() {
 String[] fileNames = {"Thin", "Ti", "Ngo", "
list = new JList (fileNames);
 //list.addListSelectionListener (new List
 list.setSelectionMode (ListSelectionMode
 add (list);
 }
}
```





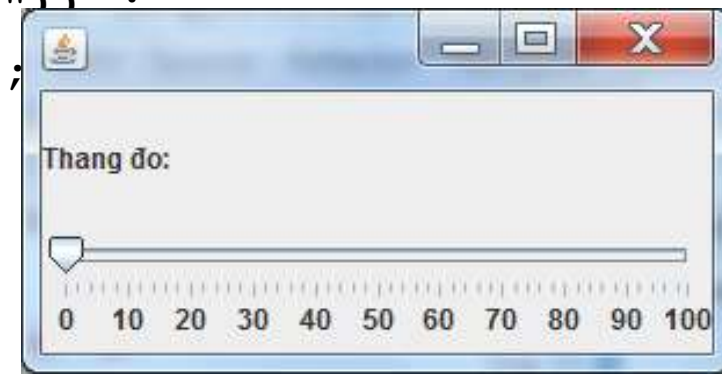
# Sliders (Scrollbar)

- ▶ A slider is a component that allows the user to specify a numeric value within a bounded range.
- ▶ A slider can be presented either vertically or horizontally and can have optional tick marks and labels indicating the range of values.
- ▶ The JSlider class has several methods
  - setMajorTickSpacing: Major tick marks can be set at specific intervals
  - setMinorTickSpacing: minor tick marks.
  - setPaintTicks: Neither is displayed, however, unless the with a parameter of true, is invoked as well.
  - setPaintLabels: Labels indicating the value of the major tick marks are displayed

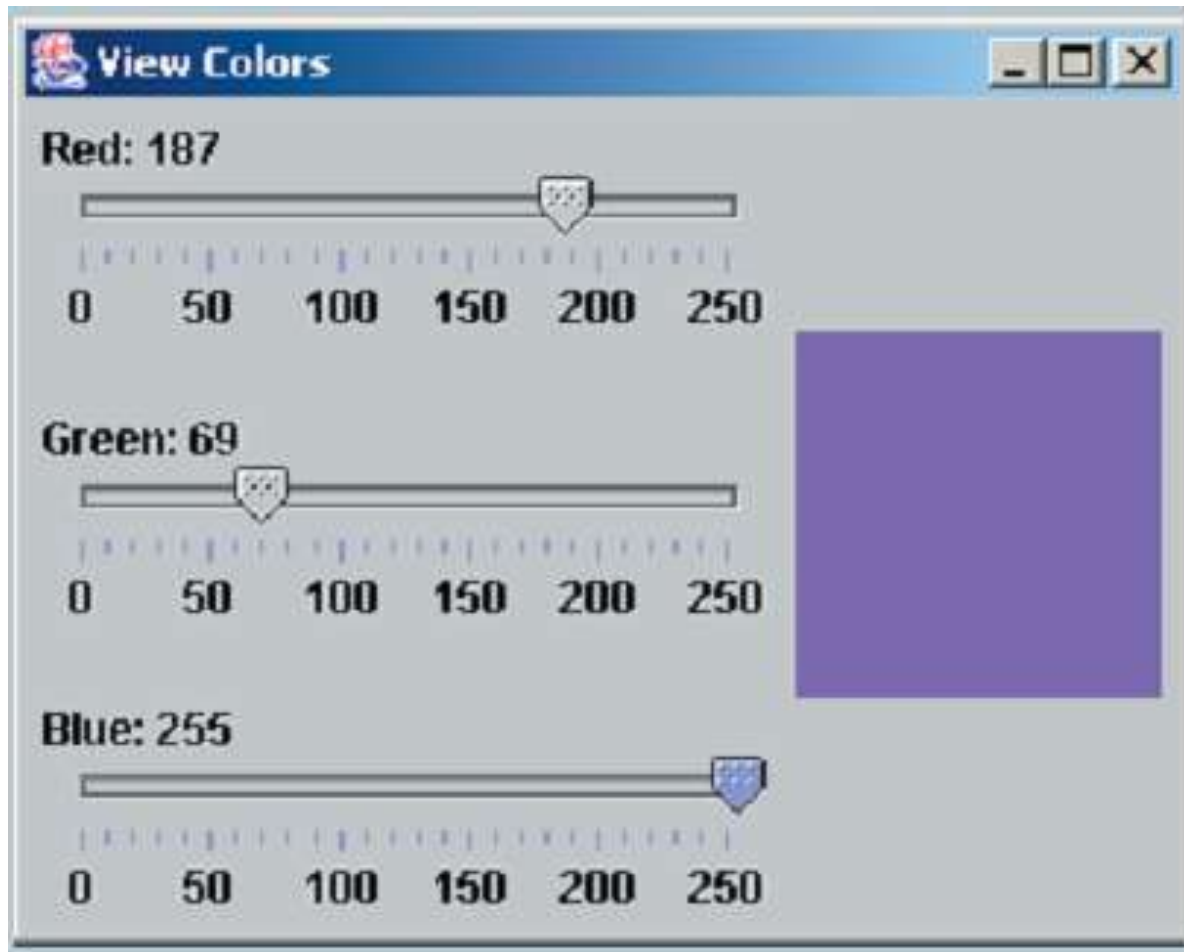
# Slider, ex

```
import java.awt.*;
import javax.swing.*;
class SliderDemo extends JFrame {
 public static void main(String[] args) {
 JFrame window = new JFrame();
 window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 window.setSize(300,200); window.setVisible(true);
 JSlider vdBar; //
 vdBar = new JSlider(JSlider.HORIZONTAL, 0, 100, 0);
 vdBar.setMajorTickSpacing(10);vdBar.setMinorTickSpacing(2);
 vdBar.setPaintTicks(true); vdBar.setPaintLabels(true);
 window.setLayout(new GridLayout(2,1));
 window.add(new JLabel("Thang đo:"));
 JTextField tx=new JTextField(10);
 window.add(tx);
 window.add(vdBar);

 }
 //end main
}
```



# Slider, exercise (home)



# Dialog box

- ▶ A dialog box is a graphical window that pops up on top of any currently active window so that the user can interact with it.
- ▶ A dialog box can serve a variety of purposes, :
  - conveying some information, (truyền đạt)
  - confirming an action, or
  - Allowing the user to enter some information.
- ▶ Class JOptionPane : create and use of basic dialog boxes.
- ▶ The basic formats for a JOptionPane dialog box:
  - A message dialog simply displays an output string.
  - A confirm dialog presents the user with a simple Y/N question.
  - An input dialog presents a prompt and a single input text field into which the user can enter one string of data.

# Dialog box - Methods

- ▶ **static String showInputDialog (Object msg)**

Displays a dialog box containing the specified message and an input text field. The contents of the text field are returned.

- ▶ **static int showConfirmDialog (Component parent, Object msg)**

Displays a dialog box containing the specified message and Yes/No button options. If the parent component is null, the box is centered on the screen.

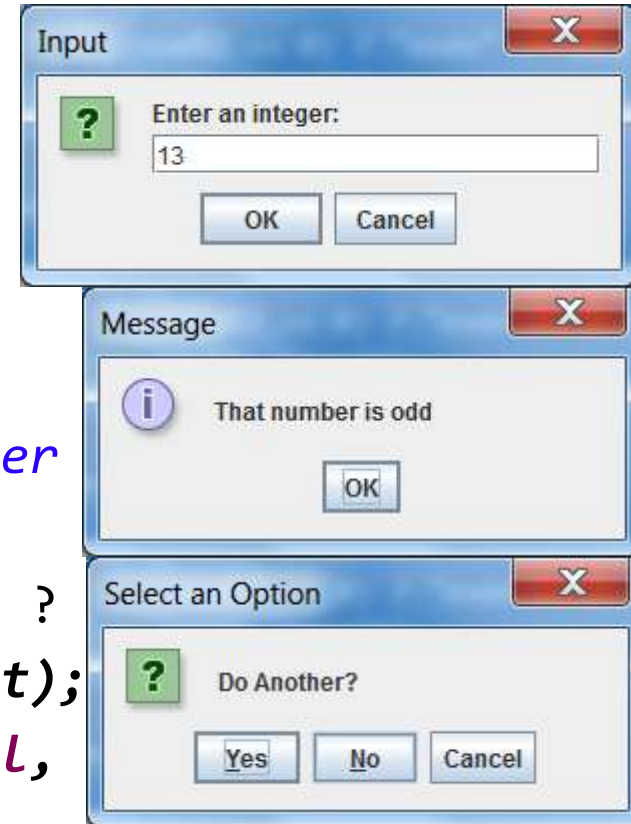
- ▶ **static int showMessageDialog (Component parent, Object msg)**

Displays a dialog box containing the specified message. If the parent component is null, the box is centered on the screen.

# Dialog Box, ex

```
import javax.swing.JOptionPane;

public class EvenOdd {
 public static void main (String[] args)
 { String numStr, result;
 int num, again;
 do {
 numStr = JOptionPane.showInputDialog ("Enter
 num = Integer.parseInt(numStr);
 result = "That number is " + ((num%2 == 0) ?
 JOptionPane.showMessageDialog (null, result);
 again = JOptionPane.showConfirmDialog (null,
 }
 while (again == JOptionPane.YES_OPTION);
 }
}
```



# Layout

- ▶ The way components are added to a container,
- ▶ In Java the technique for laying out components is to use a layout manager.
- ▶ A layout manager is an object to arrange the components in a container;



# Layout

| Layout Manager | Description                                                                                |
|----------------|--------------------------------------------------------------------------------------------|
| Border Layout  | Organizes components into five areas (North, South, East, West and Center).                |
| Box Layout     | Organizes components into a single row or column.                                          |
| Card Layout    | Organizes components into one area such that only one is visible at any time.              |
| Flow Layout    | Organizes components from left to right, starting new rows as necessary.                   |
| Grid Layout    | Organizes components into a grid of rows and columns.                                      |
| GridBag Layout | Organizes components into a grid of cells, allowing components to span more than one cell. |



# FlowLayout, ex

```
setLayout (new FlowLayout());
setBackground (Color.magenta);
```



Resize the Width

# BoxLayout

- ▶ BoxLayout cho phép add các control theo dòng hoặc cột, tại mỗi vị trí add nó chỉ chấp nhận 1 control,
- ▶ Muốn xuất hiện nhiều control tại một vị trí thì nên add vị trí đó là 1 JPanel rồi sau đó add các control khác vào JPanel này.
- ▶ `BoxLayout.X_AXIS`: Add các control theo dòng.
- ▶ `BoxLayout.Y_AXIS` : Add các control theo cột
- ▶ BoxLayout sẽ không tự động xuống dòng khi hết chỗ chứa, tức là các control sẽ bị che khuất nếu như thiếu không gian chứa nó.

# Box Layout

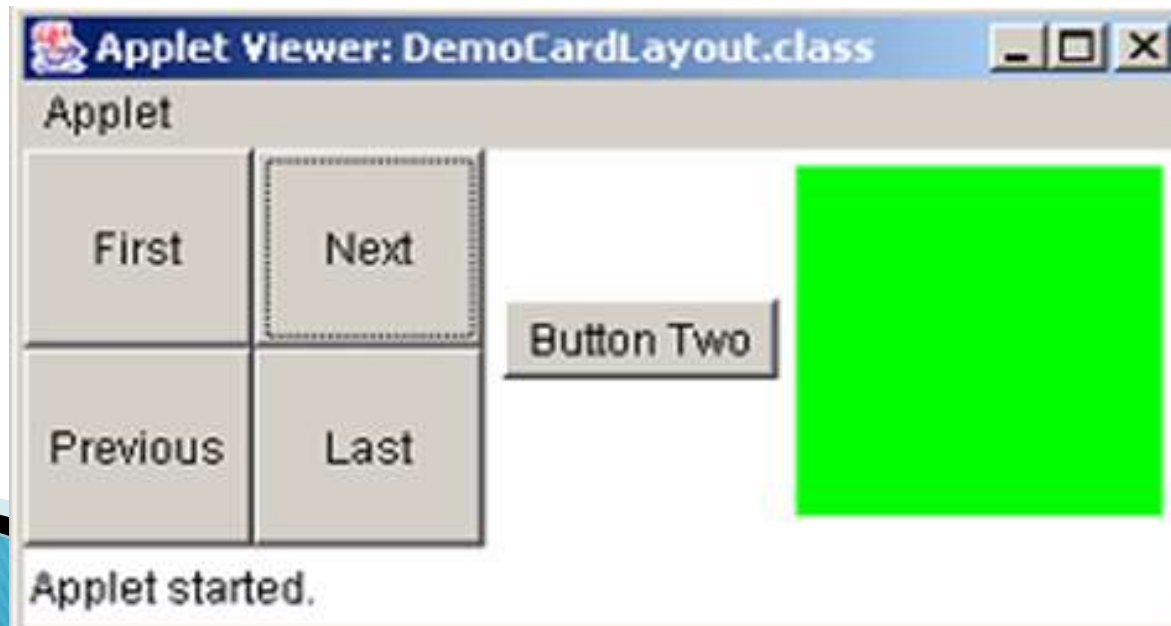
```
JPanel pnBox=new JPanel();
pnBox.setLayout(new BorderLayout(pnBox, BorderLayout.X_AXIS));
JButton btn1=new JButton("BoxLayout");
btn1.setForeground(Color.RED);
Font font=new Font("Arial",Font.BOLD / Font.ITALIC,25);
btn1.setFont(font);pnBox.add(btn1);
JButton btn2=new JButton("X_AXIS");
btn2.setForeground(Color.BLUE);
btn2.setFont(font);pnBox.add(btn2);
JButton btn3=new JButton("Y_AXIS");
btn3.setForeground(Color.ORANGE);
btn3.setFont(font);pnBox.add(btn3);
```

```
Container con=getContentPane()
con.add(pnBox);
```



# CardLayout

- ▶ Sắp xếp các thành phần giống như các lá bài. Tại mỗi thời điểm chỉ lá bài đầu tiên được hiển thị.
- ▶ Mỗi lá bài thường là một Panel và trên đó có thể dùng bất kỳ một bố cục nào



# Border layout

- ▶ A border layout has five areas to which components can be added: North, South, East, West, and Center. The areas have a particular positional relationship to each





# A Layout Example

The entire interface is governed by a border layout with a panel in each area.

North: two labels

East: two labels, a slider, and a combo box with vertical spacing in a box layout

Survey

Demographic Survey

Please complete this form. Press the Submit button when complete.

First Name:

Last Name:

Middle Initial:

Gender

☐ Male

☐ Female

Hobbies

☐ Reading

☐ Hiking

☐ Woodworking

☐ Origami

☐ Classic Sports

☐ Xtreme Sports

Other:

Age:

Salary Range:

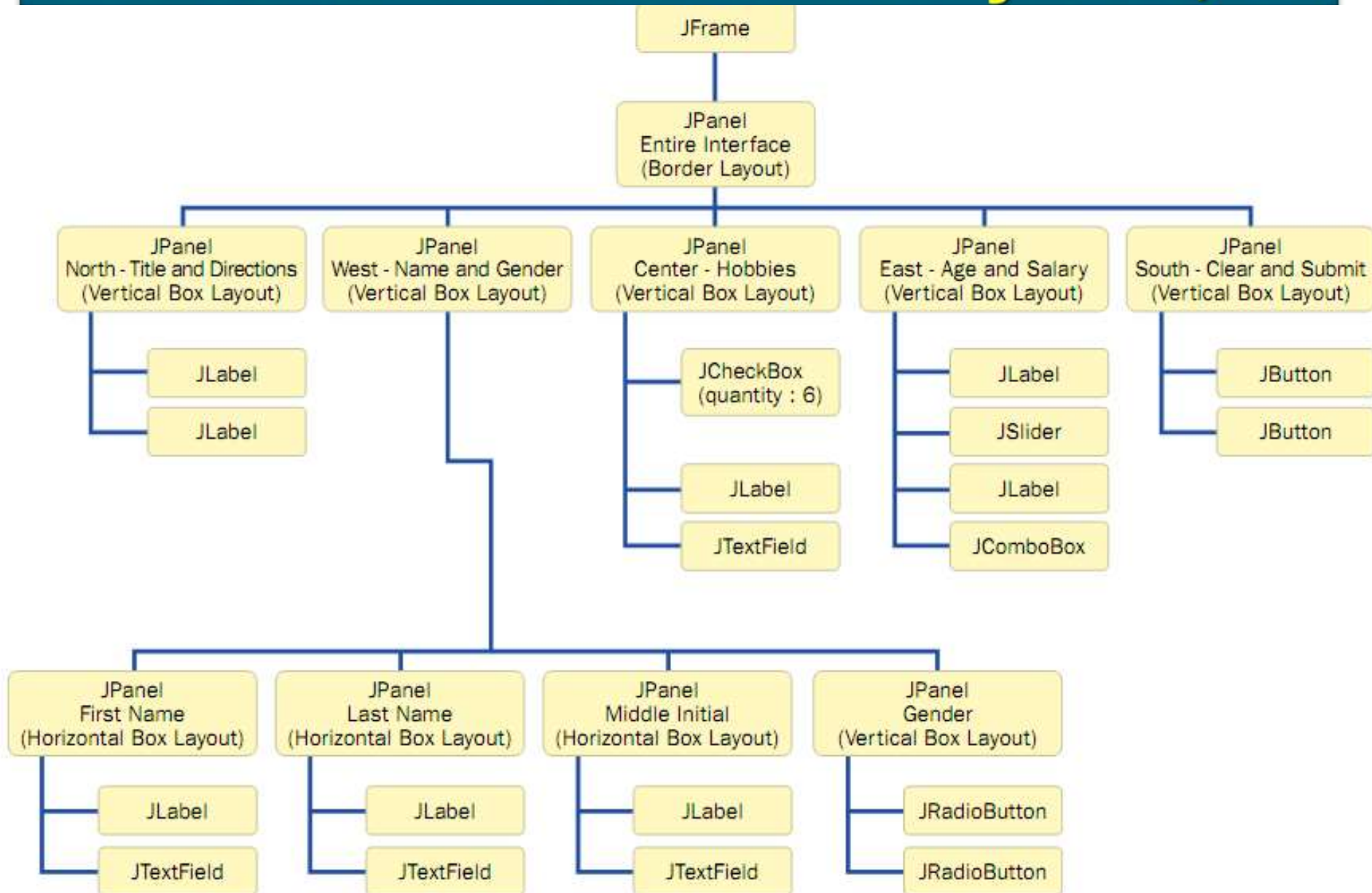
Clear Submit

West: four smaller panels in a vertical box layout. One panel for each label / text field combination and another for the gender radio buttons

Center: several check boxes, a label, and a text field

South: two buttons preceded by horizontal glue in a box layout

# The containment hierarchy tree, ex



# Events and Listeners

- ▶ GUIs are largely event-driven;
- ▶ The program waits for events that are generated by the user's actions
- ▶ When an event occurs, the program responds by executing an event-handling method.
- ▶ In order to program the behavior of a GUI, an event-handling methods must be implemented to respond to the events.



# Events and Listeners

- ▶ A listener is an object that includes one or more event-handling methods.
- ▶ The listener, has the responsibility of responding to the event.
- ▶ The event itself is actually represented by a third object, which carries information about the type of event, when it occurred, and so on.

# Ex: PanelDemo - ActionListener

```
class PanelDemo extends JPanel implements ActionListener {
 public static void main(String args[]){
 PanelDemo ObjPanel = new PanelDemo();
 JFrame ObjFr = new JFrame("Testing Event Handling!");
 ObjFr.add(ObjPanel);
 ObjFr.setSize(400,400);ObjFr.setVisible(true);
 }
 public PanelDemo() {
 JButton bt1 =new JButton("Input");
 JButton bt2 =new JButton("Exit");
 add(bt1); bt1.addActionListener(this);
 add(bt2); bt2.addActionListener(this);
 }
 public void actionPerformed(ActionEvent evt)
 if (evt.getActionCommand().equals("Input"))
 JOptionPane.showInputDialog ("Enter a string: ");
 else if (evt.getActionCommand().equals("Exit"))
 System.exit(0);
 }
}
```

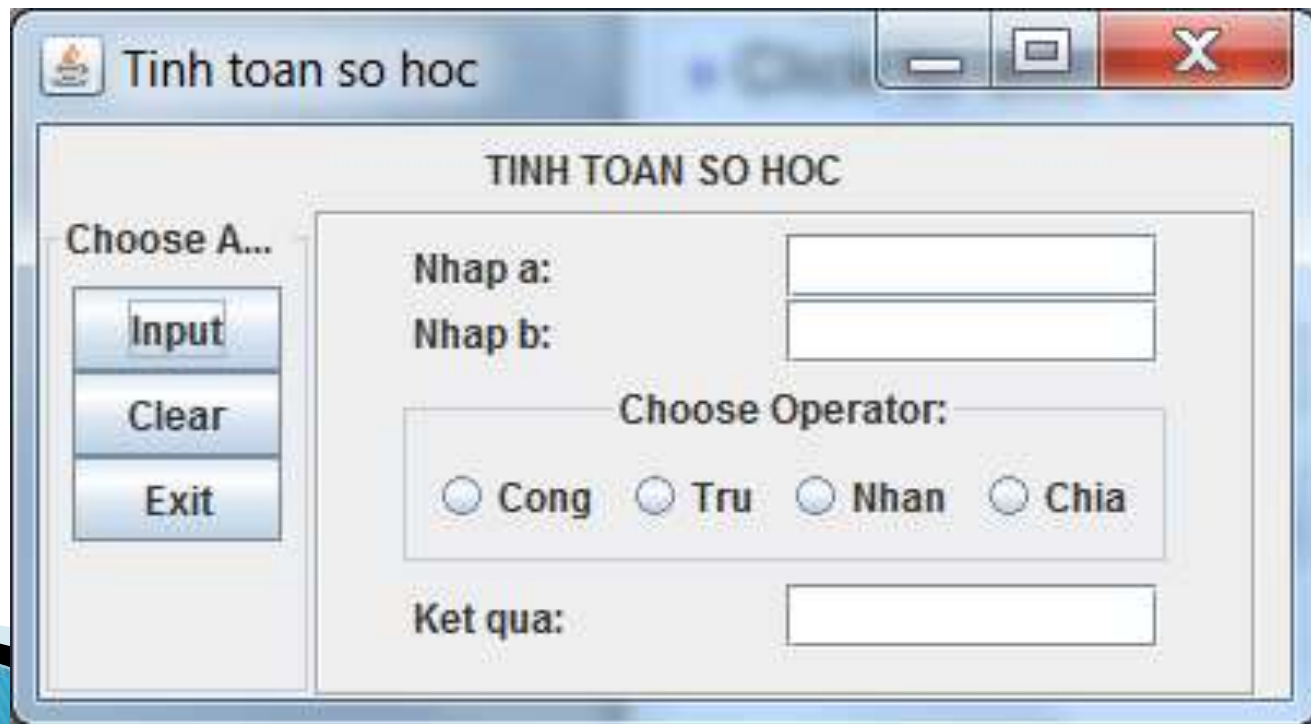


# Xây dựng lớp sự kiện

```
public class check_ex extends JFrame {
 JLabel saying; JCheckBox bold, italic;
 public static void main (String[] args) {
 check_ex gui = new check_ex();
 gui.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
 gui.pack(); gui.show()
 }
 private class StyleListener implements
 ItemListener {
 public void itemStateChanged (ItemEvent e)
 {
 int style = Font.PLAIN;
 if (bold.isSelected())
 style = Font.BOLD;
 if (italic.isSelected())
 style += Font.ITALIC;
 saying.setFont (new Font ("Helvetica",
 style, 30)); }
 }
 public check_ex() {
 saying = new JLabel ();
 saying.setFont (new Font ("Helvetica",
 bold = new JCheckBox ();
 italic = new JCheckBox ();
 StyleListener listener = new StyleListener ();
 bold.addItemListener listener;
 italic.addItemListener listener;
 add (saying); add (bold); add (italic);
 setLayout(new FlowLayout());
 }
}
```

# Event RadioButton

- ▶ Ex: Tinh.java
- ▶ Interface: ActionListener



# Canvas – khung vẽ

- ▶ Một thành phần tổng quát để vẽ và thiết kế các thành phần giao diện đồ họa mới.
- ▶ Canvas là một vùng chuyên để vẽ đồ họa, nó không bị che bởi các thành phần giao diện khác.
- ▶ Canvas có thể xử lý các sự kiện giống Applet.
- ▶ Để sử dụng Canvas, cần tạo một lớp khác dẫn xuất từ Canvas và cài đặt nạp chồng phương thức paint().
- ▶ Nên gọi setSize cho khung vẽ. Toạ độ vẽ là (0,0) tính trong khung vẽ.

# Canvas

```
import java.awt.*;
public class MyCanvas extends Canvas {
 public MyCanvas()
 { //... }
 public void paint(Graphics g)
 { /* We override the method here. The graphics
 * code comes here within the method body. */
 }
}
```

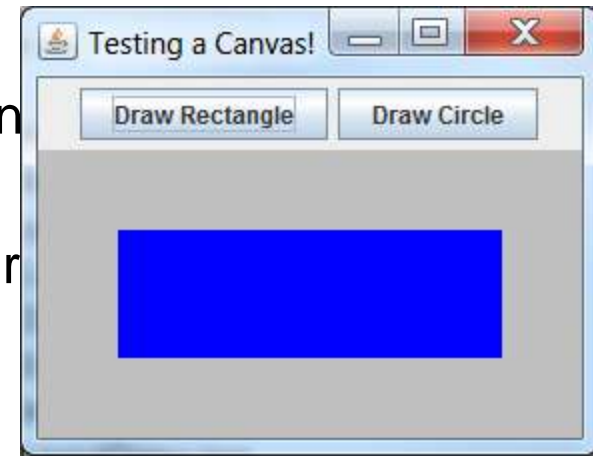
# Ex, build class vdcanvas

```
// import
public class vdcanvas extends Canvas {
 private int shape;
 public void paint(Graphics g) {
 Dimension size = getSize();
 g.setColor(Color.BLUE);
 if (shape == 1)
 g.fillRect(40, 40, size.width-80, size.height-80);
 else if (shape == 2)
 g.fillOval(40, 40, size.width-80, size.height-80);
 }
 public void draw(int shape) {
 this.shape = shape;
 repaint();
 }
}
```



# Ex, using vdcanvas class

```
class canvas_ex extends JPanel implements ActionListener
{ static vdcanvas vd1 = new vdcanvas();
 canvas_ex(){ //khai tao cac tp button }
 public static void main(String[] args) {
 JFrame f = new JFrame("Testing a Canvas!");
 canvas_ex w = new canvas_ex();
 vd1.setBackground(Color.lightGray); f.setLayout(new BorderLayout());
 f.add(w, BorderLayout.NORTH);
 f.add(vd1, BorderLayout.CENTER);
 f.pack(); f.show();
 }
 public void actionPerformed(ActionEvent event) {
 if (event.getActionCommand().equals("Draw Rectangle"))
 vd1.draw(1);
 else if (event.getActionCommand().equals("Draw Circle"))
 vd1.draw(2); }
}
```





# Setting up Drawing Panel on Frame

## PaintDemo

```
public class demoTwo {
 public static class displayPanel extends JPanel {
 public displayPanel() {
 g.setColor(Color.RED);
 g.drawOval(50, 50, 150, 150);
 }
 }

 public static void main(String[] args) {
 JFrame win = new JFrame("Demo two");
 win.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 win.setSize(400, 300);
 win.setVisible(true);
 displayPanel aPanel = new displayPanel();
 win.setContentPane(aPanel);
 }
```

# Animation. Timer class

- ▶ `Timer (int delay, ActionListener listener)`

Constructor: Creates a timer that generates an action event at regular intervals, specified by the delay. The event will be handled by the specified listener.

- ▶ `void addActionListener (ActionListener listener)`

Adds an action listener to the timer.

- ▶ `boolean isRunning ()`

Returns true if the timer is running.

- ▶ `void setDelay (int delay)`

Sets the delay of the timer.

- ▶ `void start ()`

Starts the timer, causing it to generate action events.

- ▶ `void stop ()`

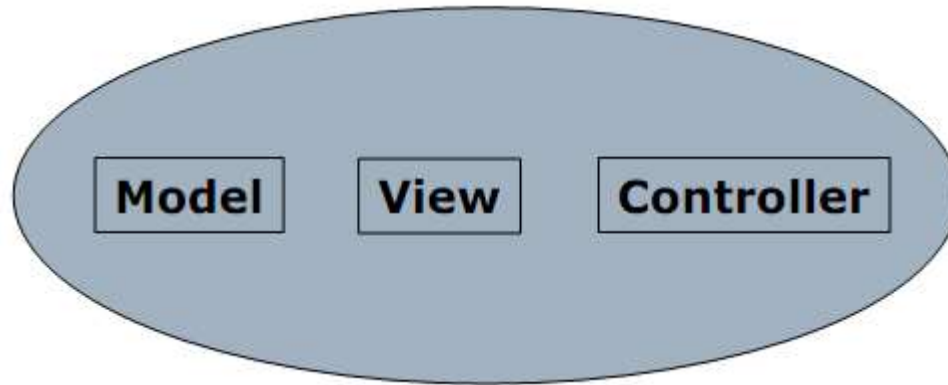
Stops the timer, causing it to stop generating action events.

# Thiết kế chương trình

- ▶ Các thành phần của chương trình
    - Dữ liệu của bài toán cần xử lý (Model)
    - Hiển thị dữ liệu của bài toán thông qua giao diện (View)
    - Điều khiển tương tác với người dùng (Controller)
  - ▶ Ví dụ: Chương trình điều khiển quả bóng
    - Model: Dữ liệu về quả bóng gồm toạ độ tâm (x,y) và bán kính bóng
    - View: Giao diện hiển thị dữ liệu quả bóng gồm có hình quả bóng và 2 nút điều khiển
    - Controller: Điều khiển di chuyển quả bóng
- Khi ấn nút điều khiển thì quả bóng di chuyển


# PP thiết kế chương trình Big Blob

- ▶ Tất cả thành phần Model, View, Controller đặt trong một lớp duy nhất




```
public class TestBall
{
 public static void main(String[] args)
 {
 MyBallFrame myFrame = new MyBallFrame("Ball
 Frame");
 myFrame.setSize(400, 300);
 myFrame.setVisible(true);

 ...
 }
}
```



```
// No chua ca model, view va controller
class MyBallFrame extends Frame implements ActionListener
{
 private int x, y, radius; // du lieu ve qua bong (model)
 private Button moveLeft, moveRight; // thanh phan GUI (view)
 ...
 moveLeft.addActionListener(this);
 moveRight.addActionListener(this);
 ...
 // xu ly su kien (controller)
 public void actionPerformed(ActionEvent event)
 ...
}
```



# Example

- ▶ Xử lý với multi checkbox
- ▶ MultiCheck.java
- ▶ Interface: ItemListener
- ▶ Class: ItemEvent
- ▶ Method: ItemStateChange(ItemEvent e)
- ▶ Attribute: ItemEvent.SELECTED()





# EXERCISE

## ► Mô tả

- Input: Nhập ngẫu nhiên a,b
- Clear: xóa trống a,b
- Exit: Thoát
- Chọn: radiobutton Cong, Tru, Nhan, Chia để tính

## ► Yêu cầu:

- Bố trí GUI hợp lý với multi panel
- Xử lý sự kiện: button, radiobutton

## ► Tinh.java

