

# NGÔN NGỮ LẬP TRÌNH JAVA

---

# Nội dung

## Cơ bản về ngôn ngữ lập trình Java

Lập trình hướng  
đối tượng

Biến, từ khoá,  
kiểu dữ liệu

Biểu thức, các  
cấu trúc điều  
khiển

Dữ liệu kiểu  
mảng

## Các khía cạnh nâng cao của lập trình hướng đối tượng

Thiết kế lớp

Thiết kế lớp  
nâng cao

Xử lý ngoại lệ

Generics

Java Collection  
Framework

## Xây dựng ứng dụng đồ họa Java

LT đồ họa với  
AWT

LT đồ họa với  
SWING

# GUI PROGRAMMING WITH AWT&SWING

---

# Nội dung

1. Giới thiệu về AWT & SWING
2. Lập trình GUI với AWT
3. Lập trình GUI với SWING

# 1. Giới thiệu về AWT-SWING

- ❑ Vấn đề : lập trình với giao diện đồ hoạ
  - ❑ Tự viết các lớp đồ hoạ:
    - ❑ Mất thời gian & công sức : vài tháng tới vài năm
  - ❑ Sử dụng thư viện sẵn có trong Java.
    - ❑ được phát triển bởi các chuyên gia,
    - ❑ rất phức tạp và sử dụng nhiều khái niệm cao cấp
    - ❑ Nhưng: dễ dàng tái sử dụng theo tài liệu API, các ví dụ và các mẫu lập trình
- ❑ Tồn tại 2 bộ Java API cho lập trình đồ hoạ
  - ❑ Abstract Windowing Toolkit (AWT):
    - ❑ Xuất hiện từ bản JDK 1.0
    - ❑ Đã lỗi thời, được thay thế bởi SWING
  - ❑ SWING:
    - ❑ Thư viện mới hơn, khắc phục các điểm yếu của AWT
    - ❑ Là một thành phần trong JFC (Java Foundation Classes)

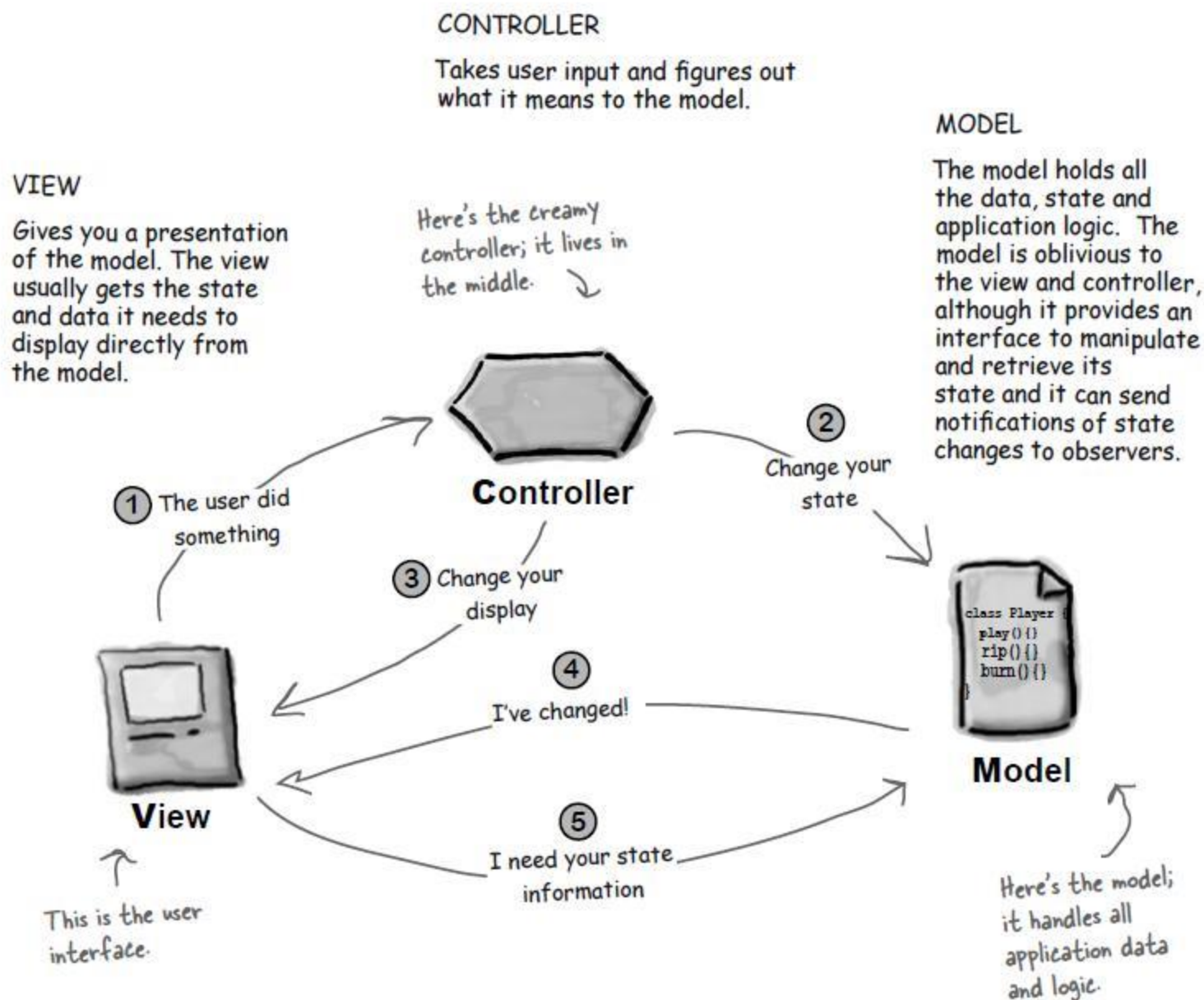
# 1. Giới thiệu về AWT-SWING

## ❑ SWING

- ❑ Được xây dựng theo kiến trúc Model-View-Controller (MVC)
  - ❑ Model: biểu diễn dữ liệu của component
  - ❑ View: là thể hiện dưới dạng đồ họa của dữ liệu
  - ❑ Controller: lấy input từ user trên view và thực hiện sự thay đổi trên dữ liệu của component

### **Ưu điểm của kiến trúc MVC:**

- ❑ Phân tách model/view : 1 model có nhiều view (cách biểu diễn khác nhau)
- ❑ Phân tách view/controller: 1 controller có thể gắn với nhiều view => thay đổi cách 1 view phản ứng với input của user mà ko cần thay đổi giao diện



# 1. Giới thiệu về AWT-SWING

## ❑ Đặc điểm của SWING

- ❑ Thuần ngôn ngữ Java => 100% portable
- ❑ Các components được thiết kế đơn giản, ko sử dụng nhiều tài nguyên
- ❑ sử dụng các lớp điều khiển sự kiện của AWT (java.awt.event)
- ❑ sử dụng trình quản lý giao diện (layout manager) của AWT
- ❑ Hiểu được cách lập trình trên AWT => việc chuyển sang SWING là đơn giản



## 2. Lập trình GUI với AWT

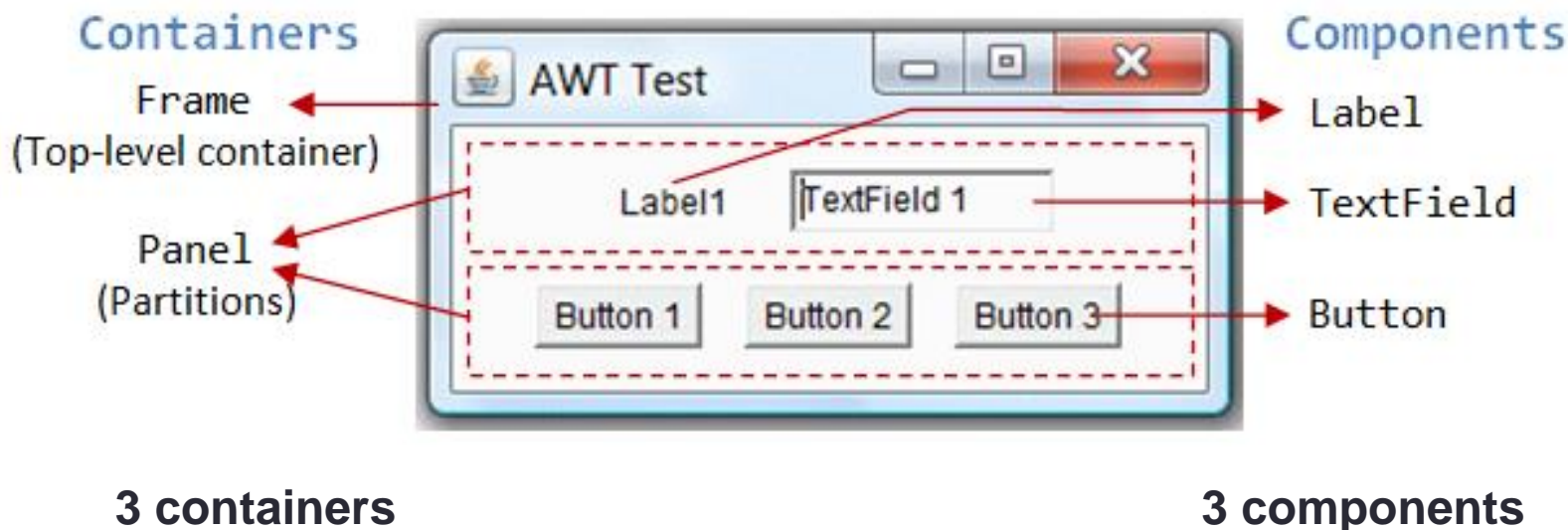
### 2.1. AWT packages

- ❑ AWT bao gồm 12 packages
- ❑ 2 gói được sử dụng thường xuyên: `java.awt` và `java.awt.event`
- ❑ Gói `java.awt`:
  - ❑ Các lớp GUI Component : Button, TextField, Label, ...
  - ❑ Các lớp GUI Container : Frame, Panel, Dialog, ScrollPane, ...
  - ❑ Các lớp layout: FlowLayout, BorderLayout, GridLayout
  - ❑ Các lớp khác: Graphics, Color và Font
- ❑ Gói `java.awt.event`:
  - ❑ Các lớp sự kiện : ActionEvent, MouseEvent, KeyEvent, ...
  - ❑ Event Listener Interfaces : ActionListener, MouseListener, KeyListener,
  - ❑ Event Listener Adapter classes: MouseAdapter, KeyAdapter, ...

## 2. Lập trình GUI với AWT

### 2.2. Containers & Components

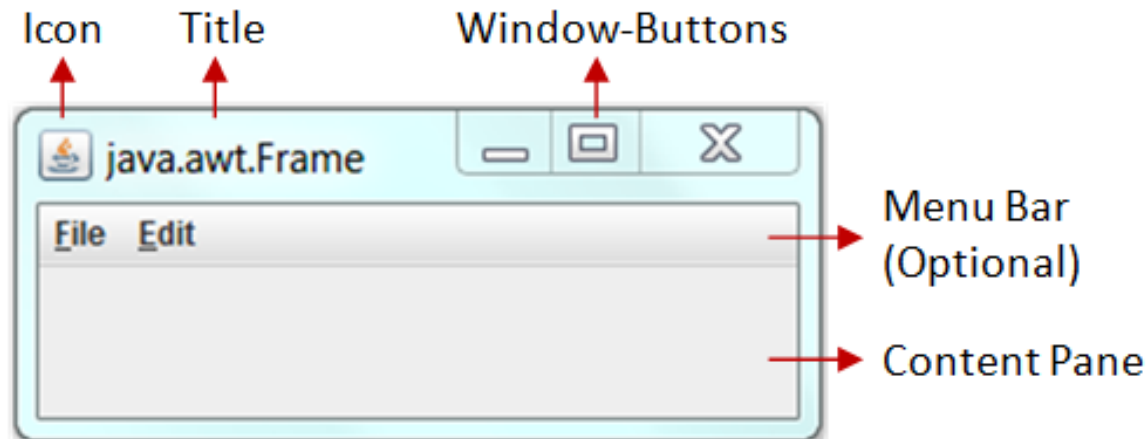
- ❑ Có 2 loại thành phần: container & component
  - ❑ Container là các thùng chứa, để giữ các component trong một layout cụ thể.
    - ❑ 1 container có thể chứa các container khác
  - ❑ Component là các đơn vị thực thể giao diện, vd: Button, Label, ...
    - ❑ phải được đặt trong 1 container



## 2. Lập trình GUI với AWT

### 2.3. Các lớp Container

- ❑ Mỗi một ứng dụng GUI có một toplevel container
- ❑ 3 Toplevel Container hay dùng: Frame, Dialog, & Applet
- ❑ **Frame**: cung cấp cửa sổ chính cho ứng dụng, bao gồm
  - ❑ Title bar: icon, title, minimize, maximize/restore-down, close
  - ❑ Menu bar: File, Edit, ...
  - ❑ Content pane: hiển thị nội dung



## 2. Lập trình GUI với AWT

### 2.3. Các lớp Container

- ❑ Để bắt đầu viết một ứng dụng GUI
  - ❑ Khai báo một lớp kế thừa Frame

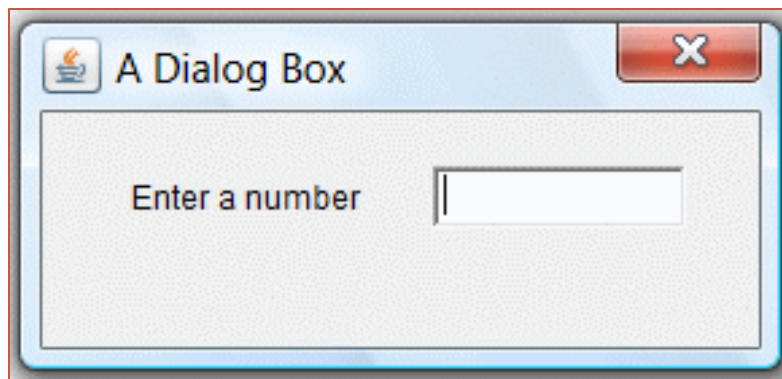
```
import java.awt.Frame; // Using Frame class in package java.awt

public class MyGUIProgram extends Frame {
    // Constructor to setup the GUI components
    public MyGUIProgram() { ..... }
    .....
    .....
    // The entry main() method
    public static void main(String[] args) {
        // Invoke the constructor (to setup the GUI) by allocating an instance
        MyGUIProgram m = new MyGUIProgram();
    }
}
```

## 2. Lập trình GUI với AWT

### 2.3. Các lớp Container

- ❑ 3 Toplevel Container hay dùng: Frame, Dialog, & Applet
- ❑ **Dialog**: là hộp thoại để tương tác với người sử dụng, gồm
  - ❑ Title bar : thanh tiêu đề
  - ❑ Content display area: khu vực hiển thị nội dung



## 2. Lập trình GUI với AWT

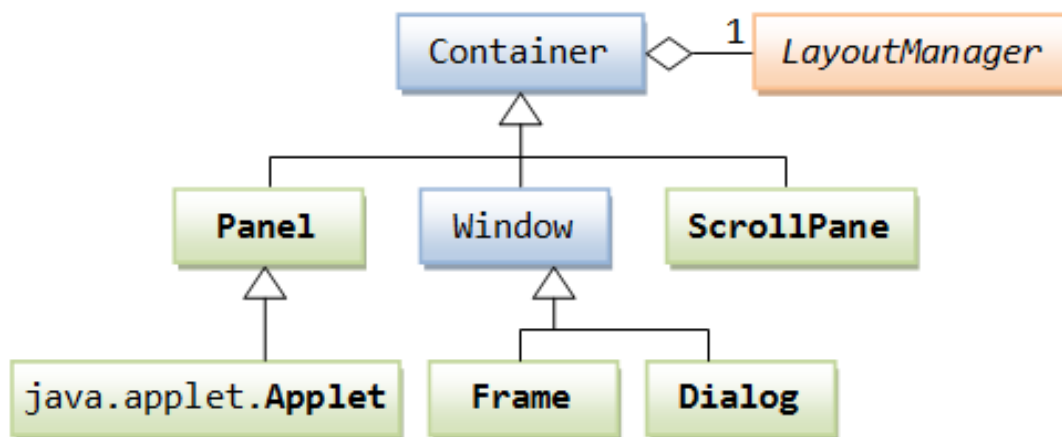
### 2.3. Các lớp Container

- ❑ 3 Toplevel Container hay dùng: Frame, Dialog, & Applet
  - ❑ **Applet**: là chương trình Java được tải xuống và thực thi trên trình duyệt web
    - ❑ Không cần cài đặt
    - ❑ Viết một lần, chạy mọi nơi
    - ❑ Sự phong phú của giao diện đồ họa
  - ❑ Ví dụ:
    - ❑ AnywareOffice của VistaSource

## 2. Lập trình GUI với AWT

### 2.3. Các lớp Container

- ❑ Secondary Container: Panel & ScrollPane
  - ❑ Được đặt trong container mức đỉnh hoặc container mức 2
  - ❑ Panel:
    - ❑ Là một hình chữ nhật,
    - ❑ Dùng để bố trí các GUI components khác
  - ❑ ScrollPane:
    - ❑ Cung cấp tính năng cuộn lên / xuống cho một component đơn.

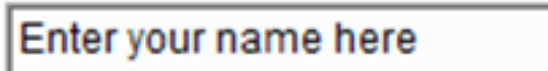


Cây phân cấp các lớp Container

## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

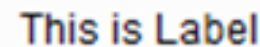
❑ AWT cung cấp rất nhiều GUI components có sẵn

A rectangular text input field with a thin border and the text "Enter your name here" inside.

TextField

A rectangular button with a 3D effect and the text "Click Me!" inside.

Button

A rectangular label with a thin border and the text "This is Label" inside.

Label

A choice component showing a dropdown menu. The current selection is "Red". The menu is open, showing options: "Red", "Green" (highlighted), and "Blue".

Choice

Three checkboxes with labels "one", "two", and "three". The "one" checkbox is checked.

CheckBox

Three radio buttons with labels "Alpha", "Beta", and "Charlie". The "Alpha" radio button is selected.

CheckBoxGroup

A list component showing a vertical list of planet names: "Mercury", "Venus", "Earth" (highlighted), "Mars", "Jupiter", "Saturn", "Uranus", and "Neptune".

List

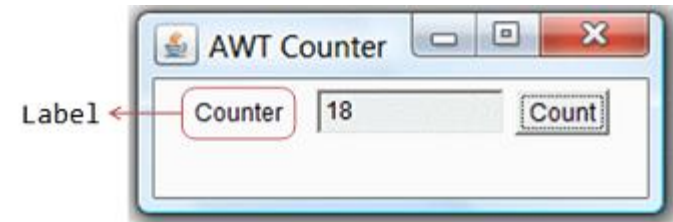


## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

#### 2.4.1. Label

- ❑ Cung cấp nhãn mô tả
- ❑ Các hàm khởi tạo
  - ❑ `public Label(String strLabel, int alignment);`
    - ❑ // Construct a Label with the given text String, of the text alignment
  - ❑ `public Label(String strLabel);`
    - ❑ // Construct a Label with the given text String
  - ❑ `public Label();` // Construct an initially empty Label
- ❑ Hằng số
  - ❑ `public static final LEFT;` // Label.LEFT
  - ❑ `public static final RIGHT;` // Label.RIGHT
  - ❑ `public static final CENTER;` // Label.CENTER
- ❑ Phương thức
  - ❑ `public String getText();`
  - ❑ `public void setText(String strLabel);`
  - ❑ `public int getAlignment();`
  - ❑ `public void setAlignment(int alignment);`



## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

- ❑ 3 bước để khởi tạo một component và đặt vào một container
  1. Khai báo component với một định danh
  2. Tạo component = gọi hàm khởi tạo với toán tử new
  3. Xác định container (Frame hoặc Panel) để đặt component

- ❑ Ví dụ:

```
Label lblInput;
```

```
lblInput = new Label("Enter ID");
```

```
this.add(lblInput); // "this" is typically a subclass of Frame or Panel
```

```
lblInput.setText("Enter password");
```

```
lblInput.getText();
```

## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

#### **Đối tượng không có định danh (anonymous instance)**

- ☐ 1 đối tượng Label sau khi được tạo ra thì thường sẽ không thay đổi nội dung
- ☐ Có thể sử dụng anonymous instance
- ☐ Không thể tham chiếu đến đối tượng sau khi đã khai báo

☐ Ví dụ:

```
add(new Label("Enter Name: ", Label.RIGHT));
```

Tương tự

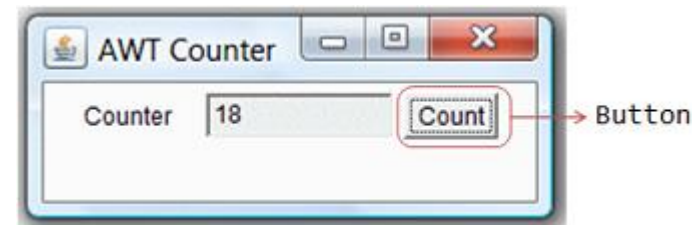
```
Label lblXxx = new Label("Enter Name: ", Label.RIGHT);  
add(lblXxx);
```

## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

#### 2.4.2. Button

- ❑ Kích hoạt một hoạt động nào đó khi nút được ấn
- ❑ Các hàm khởi tạo
  - ❑ `public Button(String buttonLabel);`
    - ❑ `// Construct a Button with the given label`
  - ❑ `public Button();`
    - ❑ `// Construct a Button with empty label`
- ❑ Phương thức
  - ❑ `public String getLabel();`
  - ❑ `public void setLabel(String buttonLabel);`
  - ❑ `public void setEnable(boolean enable);`
- ❑ Sự kiện:
  - ❑ Click vào button sẽ sinh ra sự kiện `ActionEvent` và kích hoạt một đoạn mã phản ứng



## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

#### 2.4.2. Button

❑ Kích hoạt một hoạt động nào đó khi nút được ấn

❑ Ví dụ :

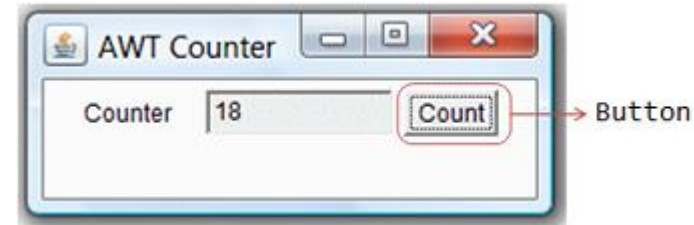
```
Button btnColor = new Button("Red");  
add(btnColor);
```

...

```
btnColor.setLabel("green");  
btnColor.getLabel();
```

...

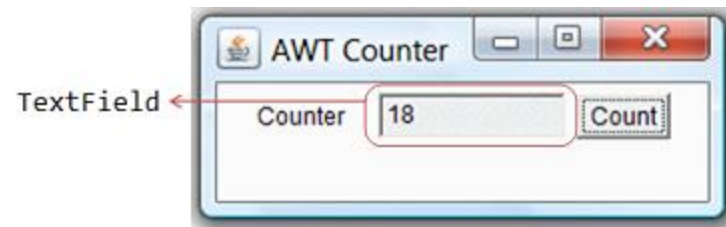
```
// Create an anonymous Button. It CANNOT be referenced later  
add(Button("Blue"));
```



## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

#### 2.4.3. TextField



- ❑ Là một hộp nhập cho phép người dùng nhập vào một đoạn văn bản
- ❑ Các hàm khởi tạo
  - ❑ `public TextField(String strInitialText, int columns);`
    - ❑ // Construct a TextField instance with the given initial text string with the number of columns.
  - ❑ `public TextField(String strInitialText);`
    - ❑ // Construct a TextField instance with the given initial text string.
- ❑ Phương thức
  - ❑ `public String getText();`
  - ❑ `public void setText(String strText);`
  - ❑ `public void setEditable(boolean editable);`
- ❑ Sự kiện:
  - ❑ Ấn phím ENTER trên textfield sẽ sinh ra sự kiện `ActionEvent` và kích hoạt một đoạn mã phản ứng

## 2. Lập trình GUI với AWT

### 2.4. Các lớp Component

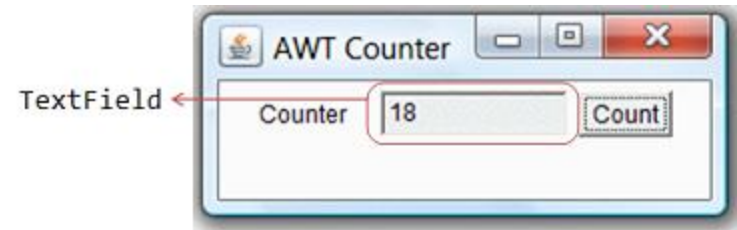
#### 2.4.3. TextField

❑ Là một hộp nhập cho phép người dùng nhập vào một đoạn văn bản

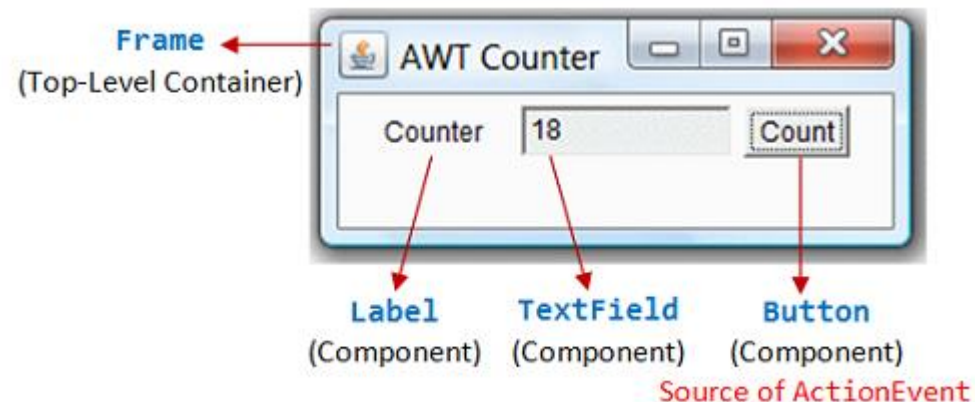
❑ Ví dụ:

```
TextField tfInput = new TextField(30)
add(tfInput); //"this" Container adds the TextField
TextField tfResult = new TextField();
tfResult.setEditable(false) ; //Set to read-only
add(tfResult);
```

```
...
// Read an int from TextField "tfInput"
int number = Integer.parseInt(tfInput.getText());
// square it
number *= number;
// display on «tfResult »
tfResult.setText(number + "");
```



## Ví dụ 1: AWTCounter



- ❑ Tạo ra một bộ đếm đơn giản
- ❑ Toplevel container là Frame, chứa 3 components
  - ❑ 1 Label « Counter »
  - ❑ 1 TextField non-editable
    - ❑ Hiện thị bộ đếm hiện tại
    - ❑ Giá trị ban đầu = 0
  - ❑ 1 Button « Count »
    - ❑ Mỗi lần bấm chuột vào button « Count », giá trị bộ đếm Counter tăng lên 1 đơn vị



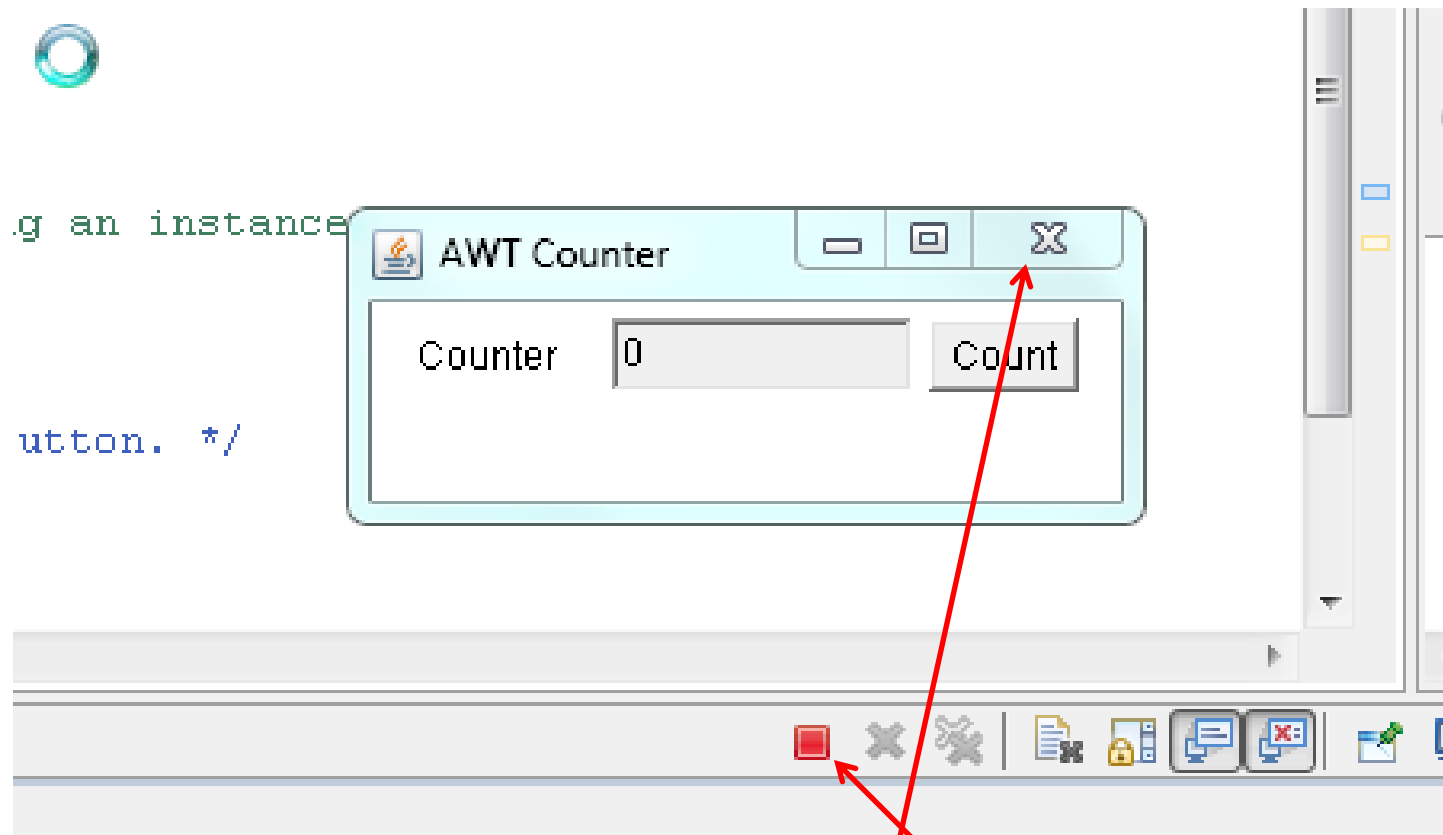
## Ví dụ 1: AWTCounter

```
1  import java.awt.*;           // using AWT containers and components
2  import java.awt.event.*;     // using AWT events and listener interfaces
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTCounter extends Frame implements ActionListener {
6      private Label lblCount;   // declare component Label
7      private TextField tfCount; // declare component TextField
8      private Button btnCount;  // declare component Button
9      private int count = 0;    // counter's value
10
11     /** Constructor to setup GUI components */
12     public AWTCounter () {
13         setLayout(new FlowLayout());
14         // "this" Frame sets its layout to FlowLayout, which arranges the components
15         // from left-to-right, and flow to next row from top-to-bottom.
16
17         lblCount = new Label("Counter"); // construct Label
18         add(lblCount);                  // "this" Frame adds Label
19
20         tfCount = new TextField("0", 10); // construct TextField
21         tfCount.setEditable(false);       // set to read-only
22         add(tfCount);                    // "this" Frame adds tfCount
23
24         btnCount = new Button("Count"); // construct Button
25         add(btnCount);                  // "this" Frame adds Button
26
```

## Ví dụ 1: AWTCounter

```
27         btnCount.addActionListener(this); // for event-handling
28
29         setTitle("AWT Counter"); // "this" Frame sets title
30         setSize(250, 100);      // "this" Frame sets initial window size
31         setVisible(true);       // "this" Frame shows
32     }
33
34     /** The entry main() method */
35     public static void main(String[] args) {
36         // Invoke the constructor to setup the GUI, by allocating an instance
37         AWTCounter app = new AWTCounter();
38     }
39
40     /** ActionEvent handler - Called back when user clicks the button. */
41     @Override
42     public void actionPerformed(ActionEvent evt) {
43         ++count; // increase the counter value
44         // Display the counter value on the TextField tfCount
45         tfCount.setText(count + ""); // convert int to String
46     }
47 }
```

## Ví dụ 1: AWTCounter



Để đóng chương trình, nhấn vào nút **stop** trên Eclipse's Application Console

Lý do: chưa viết mã cho nút **Close** trên Frame.

## Ví dụ 1: AWTCounter

### Khảo sát các đối tượng GUI với toString()

```
System.out.println(this);
System.out.println(lblCount);
System.out.println(tfCount);
System.out.println(btnCount);
```

```
setVisible(true);           // "this" Frame shows
```

```
System.out.println(this);
System.out.println(lblCount);
System.out.println(tfCount);
System.out.println(btnCount);
```

```
AWTCounter[frame0,0,0,250x100,invalid,hidden,layout=java.awt.Fl
java.awt.Label[label0,0,0,0x0,invalid,align=left,text=Counter]
java.awt.TextField[textfield0,0,0,0x0,invalid,text=0,selection=
java.awt.Button[button0,0,0,0x0,invalid,label=Count]
AWTCounter[frame0,0,0,250x100,layout=java.awt.FlowLayout,title=
java.awt.Label[label0,20,35,58x23,align=left,text=Counter]
java.awt.TextField[textfield0,83,35,94x23,text=0,selection=0-0]
java.awt.Button[button0,182,35,47x23,label=Count]
```

## 2. Lập trình GUI với AWT

### 2.5. AWT Event-handling

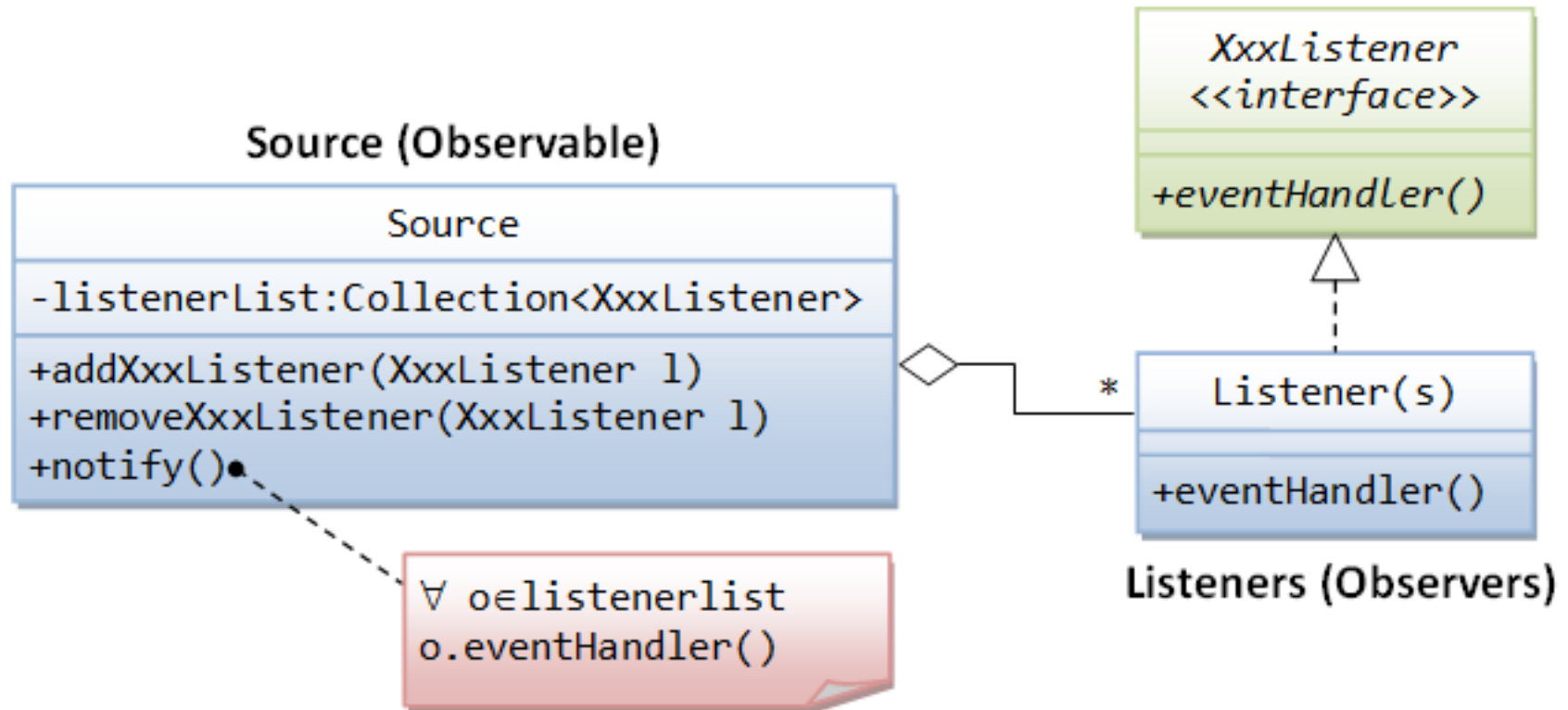
- ❑ Là một ngôn ngữ lập trình trực quan, Java hỗ trợ lập trình hướng sự kiện (event-driven programming)
- ❑ Lập trình hướng sự kiện:
  - ❑ Luồng thực thi của ứng dụng được xác định bởi các sự kiện xảy ra
  - ❑ Đoạn mã xử lý sự kiện được thực thi khi một sự kiện được tạo ra để phản ứng lại thao tác của người sử dụng (nhấn phím, click chuột)

## 2. Lập trình GUI với AWT

### 2.5. AWT Event-handling

- ❑ Mô hình hướng sự kiện: source, listener và event
  - ❑ Source object:
    - ❑ thành phần giao diện tạo ra sự kiện,
    - ❑ thành phần này tương tác với người dùng: Button, TextField
  - ❑ Event object:
    - ❑ Được tạo ra khi một sự kiện xảy ra tại source object.
    - ❑ Chứa tất cả thông tin về sự kiện: loại sự kiện, nguồn sự kiện
  - ❑ Listener object:
    - ❑ Nhận sự kiện và xử lý tương tác của người dùng
    - ❑ Đối tượng sự kiện sẽ được chuyển tới tất cả các đối tượng lắng nghe đã đăng ký (registered listener object),
    - ❑ Và đoạn mã xử lý sự kiện của listener sẽ được gọi để phản ứng lại.

# Observer Pattern



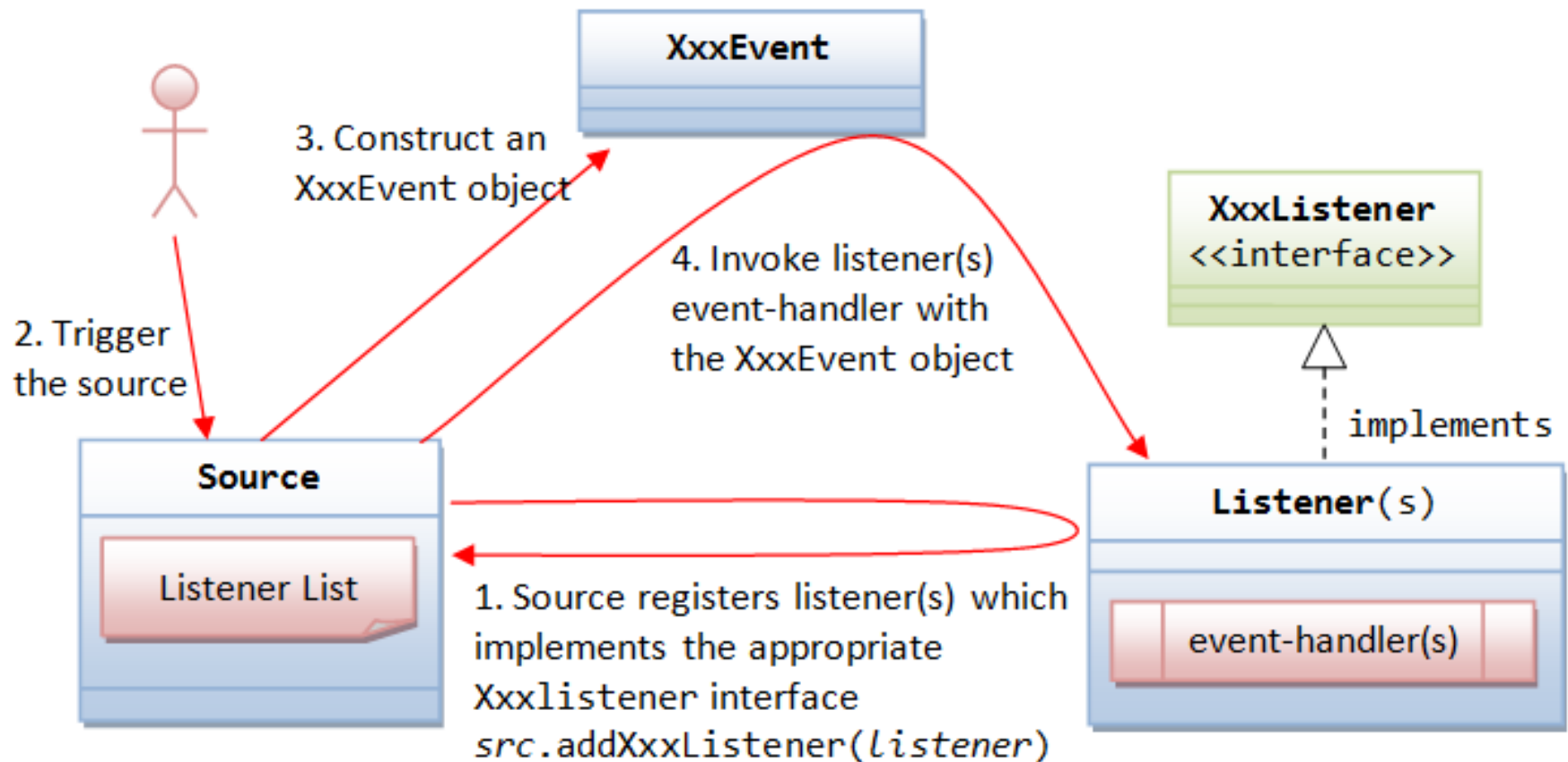
❑ Để thể hiện sự quan tâm tới một event của đối tượng source, đối tượng listener phải được đăng ký bởi đối tượng source.

❑ Publish-subscribe or Observer pattern

❑ Bất cứ khi nào trạng thái của đ/t source thay đổi => thông báo đến tất cả các listener đã đăng ký với source.

## 2. Lập trình GUI với AWT

### 2.5. AWT Event-handling

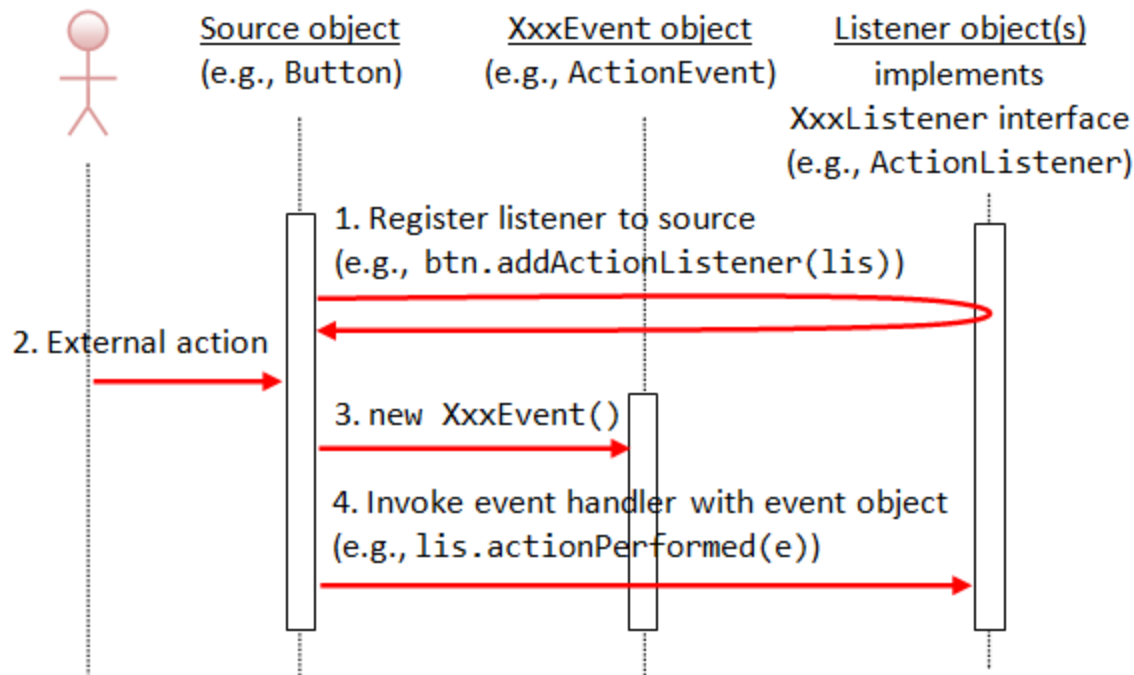




## 2. Lập trình GUI với AWT

### 2.5. AWT Event-handling

#### Quay trở lại ví dụ AWT-Counter



ActionEvent listener phải cài đặt ActionListener interface, với một phương thức :

```
interface ActionListener {
```

```
    // Called back upon button clicked, enter key pressed
    public void actionPerformed(ActionEvent e);
```

```
}
```

## 2. Lập trình GUI với AWT

### 2.6. Lớp sự kiện (Event Classes)

Event Classes	Miêu tả
ComponentEvent	Extends AWTEvent. Được thể hiện khi một component di chuyển, thay đổi kích thước, hiển thị hoặc ẩn đi.
InputEvent	Extends ComponentEvent. Lớp sự kiện gốc trừu tượng cho tất cả các lớp sự kiện input
ActionEvent	Extends AWTEvent. Được thể hiện khi một button được nhấn, một list item được double-click, hoặc một menu item được chọn
ItemEvent	Extends AWTEvent. Được thể hiện khi một item được chọn hoặc bỏ chọn chẳng hạn như trong List hoặc checkbox
KeyEvent	Extends InputEvent. Được thể hiện khi một key được press, release, type
MouseEvent	Extends InputEvent. Được thể hiện khi một nút nhấn chuột press, release or click(press+release), hoặc con trỏ chuột di chuyển vào hoặc ra khỏi vùng của một component hiển thị.
TextEvent	Extends AWTEvent. Được thể hiện khi giá trị của một text field hoặc text area thay đổi.
WindowEvent	Extends ComponentEvent. Được thể hiện khi một cửa sổ close, open, active, deactivate, iconified, deiconified hoặc khi focus được chuyển vào trong hoặc ra ngoài cửa sổ.

## 2. Lập trình GUI với AWT

### 2.6. Lớp xử lý sự kiện (Listener Classes)

Listener Class	Mô tả
ActionListener	Nhận những sự kiện action
MouseListener	Nhận những sự kiện mouse
MouseMotionListener	Nhận những sự kiện di chuyển của chuột
WindowListener	Nhận những sự kiện liên quan đến cửa sổ
KeyListener	Nhận những sự kiện liên quan đến bàn phím
...	...

## 2. Lập trình GUI với AWT

### 2.7. WindowEvent & WindowListener Interface



- ❑ Một WindowEvent được tạo ra khi một cửa sổ chịu tác động sau: opened/closed, activated/deactivated, iconified/deiconified
- ❑ 1 WindowEvent listener phải cài đặt WindowListener interface, với 7 phương thức trừu tượng sau:
  - ❑ public void **windowClosing**(WindowEvent e)
  - ❑ public void windowOpened(WindowEvent e)
  - ❑ public void windowClosed(WindowEvent e)
  - ❑ public void windowActivated(WindowEvent e)
  - ❑ public void windowDeactivated(WindowEvent e)
  - ❑ public void windowIconified(WindowEvent e)
  - ❑ public void windowDeiconified(WindowEvent e)

## Ví dụ 2: WindowEventDemo

```

1  import java.awt.*;           // using AWT containers and components
2  import java.awt.event.*;     // using AWT events and listener interfaces
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class WindowEventDemo extends Frame
6  {
7      implements ActionListener, WindowListener {
8          // This class acts as listener for ActionEvent and WindowEvent
9          // Java support only single inheritance, where a class can extend
10         // one superclass, but can implement multiple interfaces.
11
12         private TextField tfCount;
13         private int count = 0; // Counter's value
14
15         /** Constructor to setup the GUI */
16         public WindowEventDemo () {
17             setLayout(new FlowLayout()); // "this" Frame sets to FlowLayout
18
19             add(new Label("Counter")); // "this" Frame adds an anonymous Label
20
21             tfCount = new TextField("0", 10); // allocate TextField
22             tfCount.setEditable(false); // read-only
23             add(tfCount); // "this" Frame adds tfCount
24
25             Button btnCount = new Button("Count"); // declare and allocate a Button
26             add(btnCount); // "this" Frame adds btnCount
27
28             btnCount.addActionListener(this);
29             // btnCount fires ActionEvent to its registered ActionEvent listener
30             // btnCount adds "this" object as an ActionEvent listener
31             addWindowListener(this);
32             // "this" Frame fires WindowEvent its registered WindowEvent listener
33             // "this" Frame adds "this" object as a WindowEvent listener
34
35             setTitle("WindowEvent Demo"); // "this" Frame sets title
36             setSize(250, 100); // "this" Frame sets initial size
37             setVisible(true); // "this" Frame shows
38         }
39     }

```

## Ví dụ 2: AWTCounterWithCloseButton

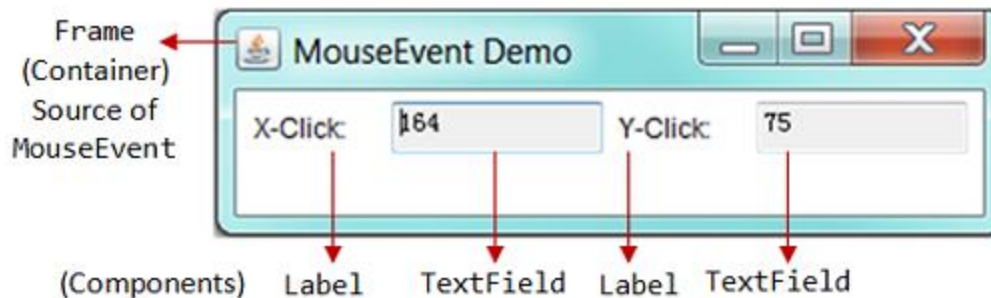
```

39  /** The entry main() method */
40  public static void main(String[] args) {
41      new WindowEventDemo(); // Let the construct do the job
42  }
43
44  /** ActionEvent handler */
45  @Override
46  public void actionPerformed(ActionEvent evt) {
47      ++count;
48      tfCount.setText(count + "");
49  }
50
51  /** WindowEvent handlers */
52  // Called back upon clicking close-window button
53  @Override
54  public void windowClosing(WindowEvent e) {
55      System.exit(0); // terminate the program
56  }
57
58  // Not Used, but need to provide an empty body for compilation
59  @Override
60  public void windowOpened(WindowEvent e) { }
61  @Override
62  public void windowClosed(WindowEvent e) { }
63  @Override
64  public void windowIconified(WindowEvent e) { }
65  @Override
66  public void windowDeiconified(WindowEvent e) { }
67  @Override
68  public void windowActivated(WindowEvent e) { }
69  @Override
70  public void windowDeactivated(WindowEvent e) { }
71  }

```

## 2. Lập trình GUI với AWT

### 2.8. MouseEvent & MouseListener Interface



❑ Một MouseEvent được tạo ra khi người dùng nhấn, thả hoặc click(nhấn-thả) một phím chuột (phím trái hoặc phải) tại đối tượng source, hoặc di chuyển con trỏ chuột vào hoặc ra đối tượng source.

❑ 1 MouseEvent listener phải cài đặt MouseListener interface, với 5 phương thức trừu tượng sau:

- ❑ `public void mouseClicked(MouseEvent e)`
- ❑ `public void mousePressed(MouseEvent e)`
- ❑ `public void mouseReleased(MouseEvent e)`
- ❑ `public void mouseEntered(MouseEvent e)`
- ❑ `public void mouseExited(MouseEvent e)`

## Ví dụ 3: MouseEventDemo

```
1  import java.awt.*;
2  import java.awt.event.MouseEvent;
3  import java.awt.event.MouseListener;
4
5  public class MouseEventDemo extends Frame implements MouseListener {
6
7      // Private variables
8      private TextField tfMouseX; // to display mouse-click-x
9      private TextField tfMouseY; // to display mouse-click-y
10
11     // Constructor - Setup the UI
12     public MouseEventDemo() {
13         setLayout(new FlowLayout()); // "this" frame sets layout
14
15         // Label
16         add(new Label("X-Click: ")); // "this" frame adds component
17
18         // TextField
19         tfMouseX = new TextField(10); // 10 columns
20         tfMouseX.setEditable(false); // read-only
21         add(tfMouseX); // "this" frame adds component
22
23         // Label
24         add(new Label("Y-Click: ")); // "this" frame adds component
25
26         // TextField
27         tfMouseY = new TextField(10);
28         tfMouseY.setEditable(false); // read-only
29         add(tfMouseY); // "this" frame adds component
```



## Ví dụ 3: MouseEventDemo

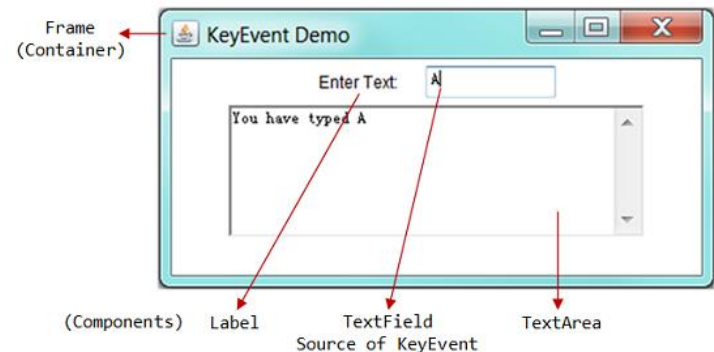
```

31     this.addMouseListener(this);
32         // "this" frame fires the MouseEvent
33         // "this" frame adds "this" object as MouseEvent listener
34
35     setTitle("MouseEvent Demo"); // "this" Frame sets title
36     setSize(350, 100);          // "this" Frame sets initial size
37     setVisible(true);           // "this" Frame shows
38 }
39
40 public static void main(String[] args) {
41     new MouseEventDemo(); // Let the constructor do the job
42 }
43
44 // MouseEvent handlers
45 @Override
46 public void mouseClicked(MouseEvent e) {
47     tfMouseX.setText(e.getX() + "");
48     tfMouseY.setText(e.getY() + "");
49 }
50
51 @Override
52 public void mousePressed(MouseEvent e) { }
53
54 @Override
55 public void mouseReleased(MouseEvent e) { }
56
57 @Override
58 public void mouseEntered(MouseEvent e) { }
59
60 @Override
61 public void mouseExited(MouseEvent e) { }
62 }

```

## 2. Lập trình GUI với AWT

### 2.9. KeyEvent & KeyListener



- ❑ 1 KeyEvent được tạo ra khi người dùng nhấn, thả hoặc gõ (nhấn-thả) một phím tại đối tượng source.
- ❑ 1 KeyEvent listener phải cài đặt KeyListener interface, với 3 phương thức trừu tượng sau:
  - ❑ public void keyTyped(KeyEvent e)
  - ❑ public void keyPressed(KeyEvent e)
  - ❑ public void keyReleased(KeyEvent e)

## Ví dụ 4: KeyEventDemo

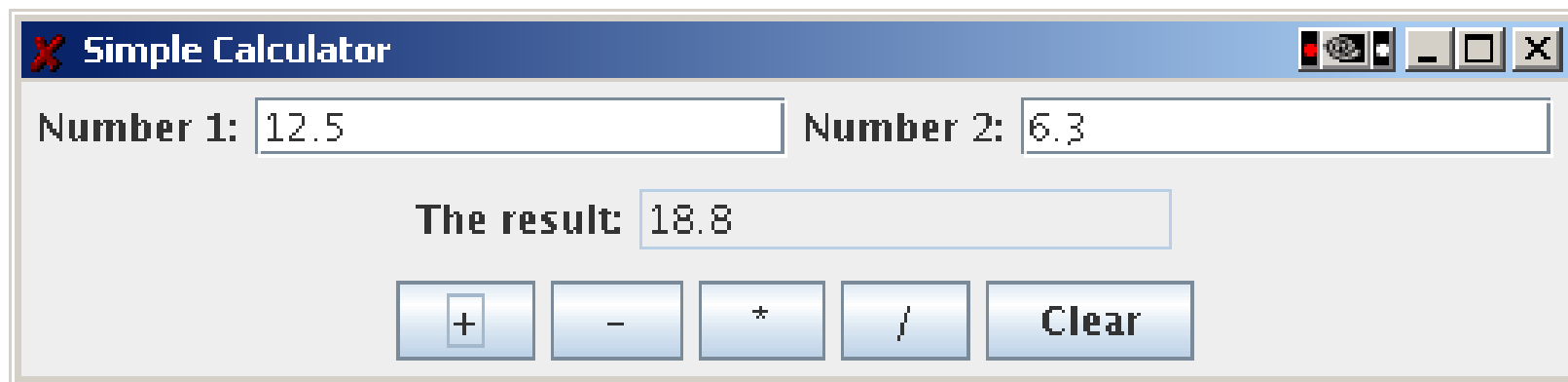
```
1  import java.awt.*;
2  import java.awt.event.KeyEvent;
3  import java.awt.event.KeyListener;
4
5  // An AWT GUI program inherits the top-level container java.awt.Frame
6  public class KeyEventDemo extends Frame implements KeyListener {
7      // This class acts as KeyEvent Listener
8
9      private TextField tfInput; // single-line TextField to receive tfInput key
10     private TextArea taDisplay; // multi-line TextArea to taDisplay result
11
12     /** Constructor to setup the GUI */
13     public KeyEventDemo() {
14         setLayout(new FlowLayout()); // "this" frame sets to FlowLayout
15
16         add(new Label("Enter Text: "));
17         tfInput = new TextField(10);
18         add(tfInput);
19         taDisplay = new TextArea(5, 40); // 5 rows, 40 columns
20         add(taDisplay);
21
22         tfInput.addKeyListener(this);
23         // tfInput TextField fires KeyEvent to its registered KeyListener
24         // It adds "this" object as a KeyEvent listener
```

## Ví dụ 4: KeyEventDemo

```
26         setTitle("KeyEvent Demo"); // "this" Frame sets title
27         setSize(400, 200);         // "this" Frame sets initial size
28         setVisible(true);           // "this" Frame shows
29     }
30
31     /** The entry main() method */
32     public static void main(String[] args) {
33         new KeyEventDemo(); // Let the constructor do the job
34     }
35
36     /** KeyEvent handlers */
37     // Called back when a key has been typed (pressed and released)
38     @Override
39     public void keyTyped(KeyEvent e) {
40         taDisplay.append("You have typed " + e.getKeyChar() + "\n");
41     }
42
43     // Not Used, but need to provide an empty body for compilation
44     @Override
45     public void keyPressed(KeyEvent e) { }
46     @Override
47     public void keyReleased(KeyEvent e) { }
48 }
```

# Bài tập tự làm

- Tạo ứng dụng máy tính đơn giản Calculator.java.
- Máy tính có khả năng tính toán 4 phép toán cơ bản là +, -, \*, / với số thực.
- Người dùng nhập số thứ nhất và số thứ hai vào 2 trường văn bản, phép tính được thực hiện khi người dùng nhấn phép tính tương ứng.
- Nút “clear” dùng để xóa cả 3 trường văn bản
- Cần xử lí trường hợp 2 đầu vào không phù hợp (không chuyển được thành số hoặc rỗng).



## 2. Lập trình GUI với AWT

### 2.10 Inner (nested) classes

Một inner class là một lớp được định nghĩa bên trong một lớp khác.

```
public class MyOuterClass {    // outer class defined here
    .....
    private class MyNestedClass1 { ... } // an nested class defined inside the outer class
    public static class MyNestedClass2 { ... } // an "static" nested class defined inside the outer class
    .....
}
```

Đặc điểm của 1 inner class:

- Là 1 lớp bình thường với hàm khởi tạo, biến, phương thức
- Là một thành phần của lớp bao bên ngoài
- Có thể truy cập đến các thành phần (dữ liệu, phương thức) riêng của lớp
- Có các thuộc tính truy cập: public, private, protected, default
- Có thể được khai báo : static, final, abstract
- Không có nghĩa là một lớp con của lớp bao ngoài

## 2. Lập trình GUI với AWT

### 2.10 Inner (Nested) classes

Khuyến nghị về việc sử dụng nested class:

- Để kiểm soát quyền truy xuất giữa inner/outer class
- Làm cho chương trình rõ ràng và dễ hiểu hơn:
  - Đặt một định nghĩa lớp gần với vị trí nó sẽ được sử dụng
- Để phục vụ việc quản lý không gian tên

## 2. Lập trình GUI với AWT

### 2.10 Inner (Nested) classes

4 kiểu inner class:

1. static nested
2. non-static inner class
3. local inner class (định nghĩa bên trong một phương thức)
4. anonymous local inner class (định nghĩa bên trong một phương thức)



## Static vs Non-static Inner classes

- Static inner class có thể được sử dụng mà không cần tạo ra một thể hiện của outer class.
- Có thể được tham chiếu
  - `OuterClassName.InnerClassName`
- Non-Static inner class được sử dụng với một thể hiện của outer class.
- Có thể được tham chiếu
  - `OuterClassInstanceName.InnerClassInstanceName`

## Ví dụ : non-static inner class

```
1  public class MyOuterClassWithInnerClass {
2      // Private member variable of the outer class
3      private String msgOuter = "Hello from outer class";
4
5      // Define an inner class as a member of the outer class
6      // This is merely an definition.
7      // Not instantiation takes place when an instance of outer class is constructed
8      public class MyInnerClass {
9          // Private variable of the inner class
10         private String msgInner;
11         // Constructor of the inner class
12         public MyInnerClass(String msgInner) {
13             this.msgInner = msgInner;
14             System.out.println("Constructing an inner class instance: " + msgOuter);
15             // can access private member variable of outer class
16         }
17         // A method of inner class
18         public void printMessage() {
19             System.out.println(msgInner);
20         }
21     }
22
23     // Declare and construct an instance of the inner class, inside the outer class
24     MyInnerClass anInner = new MyInnerClass("Hi from inner class");
25 }
```

# Ví dụ : non-static inner class

```
1 public class TestInnerClass {
2     public static void main(String[] args) {
3         // Construct an instance of outer class, which create anInner
4         MyOuterClassWithInnerClass anOuter = new MyOuterClassWithInnerClass();
5         // Invoke inner class's method from this outer class instance
6         anOuter.anInner.printMessage();
7
8         // Explicitly construct another instance of inner class
9         MyOuterClassWithInnerClass.MyInnerClass inner2
10            = anOuter.new MyInnerClass("Inner class 2");
11        inner2.printMessage();
12
13        // Explicitly construct an instance of inner class, under another instance of outer class
14        MyOuterClassWithInnerClass.MyInnerClass inner3
15            = new MyOuterClassWithInnerClass().new MyInnerClass("Inner class 3");
16        inner3.printMessage();
17    }
18 }
```

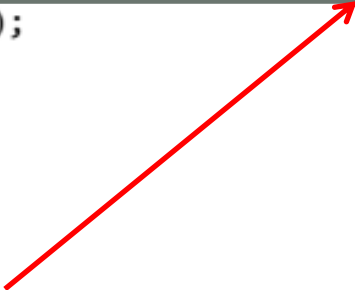
Phải tạo ra một thể hiện của outer class để khởi tạo một non-static inner class

## Ví dụ : static inner class

```
1  public class MyOuterClassWithStaticNestedClass {
2      // Private "static" member variable of the outer class
3      private static String msgOuter = "Hello from outer class";
4
5      // Define a "static" nested class as a member of the outer class
6      // It can access private "static" variable of the outer class
7      public static class MyStaticNestedClass {
8          // Private variable of inner class
9          private String msgInner;
10         // Constructor of inner class
11         public MyStaticNestedClass(String msgInner) {
12             this.msgInner = msgInner;
13             System.out.println(msgOuter); // access private member of the outer class
14         }
15         // A method of inner class
16         public void printMessage() {
17             System.out.println(msgInner);
18         }
19     }
20 }
```

## Ví dụ : static inner class

```
1 public class TestStaticNestedClass {  
2     public static void main(String[] args) {  
3         // Construct an instance of static nested class  
4         // A "static" nested class, like other "static" members, can be accessed via  
5         // the Classname.membername  
6         MyOuterClassWithStaticNestedClass.MyStaticNestedClass aNestedInner =  
7         new MyOuterClassWithStaticNestedClass.MyStaticNestedClass("Hi from inner class");  
8         aNestedInner.printMessage();  
9     }  
10 }
```



Có thể khởi tạo một static inner class mà không cần tạo ra một thể hiện của outer class

## Local Inner Class

- Local inner class được định nghĩa bên trong một phương thức.
- Local inner class ko tồn tại cho đến khi phương thức được gọi, và sẽ bị xoá khi phương thức đó kết thúc

## Đặc điểm của Local Inner Class

- Ko thể khai báo quyền truy xuất như private, public
- Ko thể khai báo static
- Có thể truy cập đến mọi biến và phương thức của outer class.
- Có thể truy cập tới biến nội bộ của phương thức nếu biến đó được khai báo là final
  - Để tránh các hiệu ứng không mong muốn

## Ví dụ : Local Inner Class

```
1  public class MyOuterClassWithLocalInnerClass {
2      // Private member variable of the outer class
3      private String msgOuter = "Hello from outer class";
4
5      // A member method of the outer class
6      public void doSomething() {
7
8          // A local variable of the method
9          final String msgMethod = "Hello from method";
10
11         // Define a local inner class inside the method
12         class MyInnerClass {
13             // Private variable of the inner class
14             private String msgInner;
15             // Constructor of the inner class
16             public MyInnerClass(String msgInner) {
17                 this.msgInner = msgInner;
18                 System.out.println("Constructing an inner class instance: " + msgOuter);
19                 // can access private member variable of outer class
20                 System.out.println("Accessing final variable of the method: " + msgMethod);
21                 // can access final variable of the method
22             }
23             // A method of inner class
24             public void printMessage() {
25                 System.out.println(msgInner);
26             }
27         }
```

## Ví dụ : Local Inner Class

```
29         // Create an instance of inner class and invoke its method
30         MyInnerClass anInner = new MyInnerClass("Hi, from inner class");
31         anInner.printMessage();
32     }
33
34     // Test main() method
35     public static void main(String[] args) {
36         // Create an instance of the outer class and invoke the method.
37         new MyOuterClassWithLocalInnerClass().doSomething();
38     }
```



## Ứng dụng: Local Inner Class as Event Listener

Quay lại ví dụ AWTCounter:

- Thay vì sử dụng con trỏ « this » như là ActionListener cho nút button
- Định nghĩa lớp mới BtnCountListener như là một inner class
- Tạo ra một thể hiện của BtnCountListener như là ActionListener
- Bỏ « implements ActionListener » từ định nghĩa lớp

## Ứng dụng: Local Inner Class as Event Listener

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTCounterNamedInnerClass extends Frame {
6      // This class is NOT a ActionListener, hence, it does not implement ActionListener
7
8      // The event-handler actionPerformed() needs to access these private variables
9      private TextField tfCount;
10     private int count = 0;
11
12     /** Constructor to setup the GUI */
13     public AWTCounterNamedInnerClass () {
14         setLayout(new FlowLayout()); // "this" Frame sets to FlowLayout
15         add(new Label("Counter"));   // anonymous instance of Label
16         tfCount = new TextField("0", 10);
17         tfCount.setEditable(false);  // read-only
18         add(tfCount);                // "this" Frame adds tfCount
19
20         Button btnCount = new Button("Count");
21         add(btnCount);               // "this" Frame adds btnCount
22
23         // Construct an anonymous instance of BtnCountListener (a named inner class).
24         // btnCount adds this instance as a ActionListener.
25         btnCount.addActionListener(new BtnCountListener());
26
27         setTitle("AWT Counter");
28         setSize(250, 100);
29         setVisible(true);
30     }
```

## Ứng dụng: Local Inner Class as Event Listener

```
37  /**
38   * BtnCountListener is a "named inner class" used as ActionListener.
39   * This inner class can access private variables of the outer class.
40   */
41  private class BtnCountListener implements ActionListener {
42      @Override
43      public void actionPerformed(ActionEvent e) {
44          ++count;
45          tfCount.setText(count + "");
46      }
47  }
48 }
```

## Anonymous Local Inner Class

- là local inner class
- không có khai báo tường minh tên lớp
- Hoặc extends một lớp cha đã tồn tại hoặc implements một interface
- Được khai báo và thể hiện hoá trong một câu lệnh với toán tử new

## Ví dụ Anonymous Local Inner Class

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTCounterAnonymousInnerClass extends Frame {
6      // This class is NOT a ActionListener, hence, it does not implement ActionListener
7
8      // The event-handler actionPerformed() needs to access these private variables
9      private TextField tfCount;
10     private int count = 0;
11
12     /** Constructor to setup the GUI */
13     public AWTCounterAnonymousInnerClass () {
14         setLayout(new FlowLayout()); // "this" Frame sets to FlowLayout
15         add(new Label("Counter"));   // an anonymous instance of Label
16         tfCount = new TextField("0", 10);
17         tfCount.setEditable(false);   // read-only
18         add(tfCount);                 // "this" Frame adds tfCount
19
20         Button btnCount = new Button("Count");
21         add(btnCount);                // "this" Frame adds btnCount
```

## Ví dụ Anonymous Local Inner Class

```
23      // Construct an anonymous instance of an anonymous class.
24      // btnCount adds this instance as a ActionListener.
25      btnCount.addActionListener(new ActionListener() {
26          @Override
27          public void actionPerformed(ActionEvent e) {
28              ++count;
29              tfCount.setText(count + "");
30          }
31      });
32
33      setTitle("AWT Counter");
34      setSize(250, 100);
35      setVisible(true);
36  }
37
38  /** The entry main method */
39  public static void main(String[] args) {
40      new AWTCounterAnonymousInnerClass(); // Let the constructor do the job
41  }
42 }
```

## Ví dụ Anonymous Local Inner Class

Hai đoạn mã sau là tương đương:

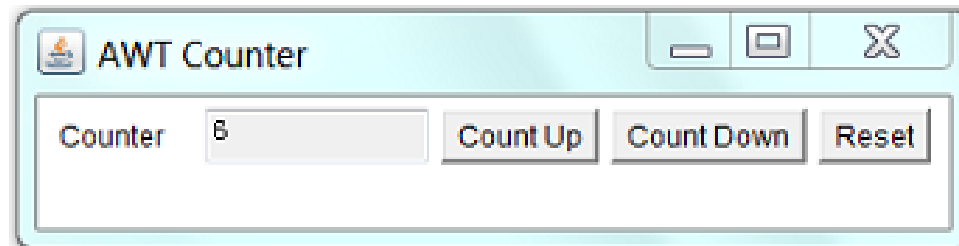
```
btnCount.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        ++count;
        tfCount.setText(count + "");
    }
});
```

```
private class N implements ActionListener { //
    @Override
    public void actionPerformed(ActionEvent e) {
        ++count;
        tfCount.setText(count + "");
    }
}
btnCount.addActionListener(new N());
```

## Ứng dụng: Tạo Anonymous Local Inner Class cho mỗi source

Một ActionListener cho  
cả 3 buttons.

- getSource()



```
private class BtnListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Need to determine which button has fired the event.
        Button source = (Button)e.getSource();
        // Get a reference of the source that has fired the event.
        // getSource() returns a java.lang.Object. Downcast back to Button.
        if (source == btnCountUp) {
            ++count;
        } else if (source == btnCountDown) {
            --count;
        } else {
            count = 0;
        }
        tfCount.setText(count + "");
    }
}
```



## Ứng dụng: Tạo Anonymous Local Inner Class cho mỗi source

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class AWTCounter3Buttons extends Frame {
6      private TextField tfCount;
7      private int count = 0;
8
9      /** Constructor to setup the GUI */
10     public AWTCounter3Buttons () {
11         setLayout(new FlowLayout());
12         add(new Label("Counter")); // an anonymous instance of Label
13         tfCount = new TextField("0", 10);
14         tfCount.setEditable(false); // read-only
15         add(tfCount); // "this" Frame adds tfCount
16
17         Button btnCountUp = new Button("Count Up");
18         add(btnCountUp);
19         // Construct an anonymous instance of an anonymous inner class.
20         // The source Button adds this instance as ActionEvent listener
21         btnCountUp.addActionListener(new ActionListener() {
22             @Override
23             public void actionPerformed(ActionEvent e) {
24                 ++count;
25                 tfCount.setText(count + "");
26             }
27         });
```

## Ứng dụng: Tạo Anonymous Local Inner Class cho mỗi source

```

29     Button btnCountDown = new Button("Count Down");
30     add(btnCountDown);
31     btnCountDown.addActionListener(new ActionListener() {
32         @Override
33         public void actionPerformed(ActionEvent e) {
34             count--;
35             tfCount.setText(count + "");
36         }
37     });
38
39     Button btnReset = new Button("Reset");
40     add(btnReset);
41     btnReset.addActionListener(new ActionListener() {
42         @Override
43         public void actionPerformed(ActionEvent e) {
44             count = 0;
45             tfCount.setText("0");
46         }
47     });
48
49     setTitle("AWT Counter");
50     setSize(400, 100);
51     setVisible(true);
52 }
53
54 /** The entry main method */
55 public static void main(String[] args) {
56     new AWTCounter3Buttons(); // Let the constructor do the job
57 }
58 }

```

## 2. Lập trình GUI với AWT

### 2.11 Adapter classes

- Tại sao phải sử dụng adapter class:
  - Cài đặt tất cả các phương thức của interface là nhàm chán và mất thời gian
  - Chỉ cần quan tâm và cài đặt một số phương thức
- Đặc điểm của adapter class:
  - Được viết bằng Java
  - Cài đặt của tất cả các phương thức của listener
  - Tất cả các cài đặt của các phương thức đều là rỗng
- Adapter classes:
  - MouseAdapter, MouseMotionAdapter
  - KeyAdapter, FocusAdapter
  - WindowAdapter,...
- Chú ý: ActionListener không tồn tại !!!

## Ví dụ : WindowAdapter classes

```
1  import java.awt.*;
2  import java.awt.event.*;
3
4  // An AWT GUI program inherits the top-level container java.awt.Frame
5  public class WindowEventDemoAdapter extends Frame {
6      private TextField tfCount;
7      private int count = 0;
8
9      /** Constructor to setup the GUI */
10     public WindowEventDemoAdapter () {
11         setLayout(new FlowLayout());
12         add(new Label("Counter"));
13         tfCount = new TextField("0", 10);
14         tfCount.setEditable(false);
15         add(tfCount);
16
17         Button btnCount = new Button("Count");
18         add(btnCount);
19         btnCount.addActionListener(new ActionListener() {
20             @Override
21             public void actionPerformed(ActionEvent evt) {
22                 ++count;
23                 tfCount.setText(count + "");
24             }
25         });
```

## Ví dụ : Adapter classes

```
27      // Allocate an anonymous instance of an anonymous inner class
28      // that extends WindowAdapter.
29      // "this" Frame adds the instance as WindowEvent listener.
30      addWindowListener(new WindowAdapter() {
31          @Override
32          public void windowClosing(WindowEvent e) {
33              System.exit(0); // Terminate the program
34          }
35      });

36
37      setTitle("WindowEvent Demo");
38      setSize(250, 100);
39      setVisible(true);
40  }

41
42  /** The entry main method */
43  public static void main(String[] args) {
44      new WindowEventDemoAdapter(); // Let the constructor do the job
45  }
46 }
```

## 2. Lập trình GUI với AWT

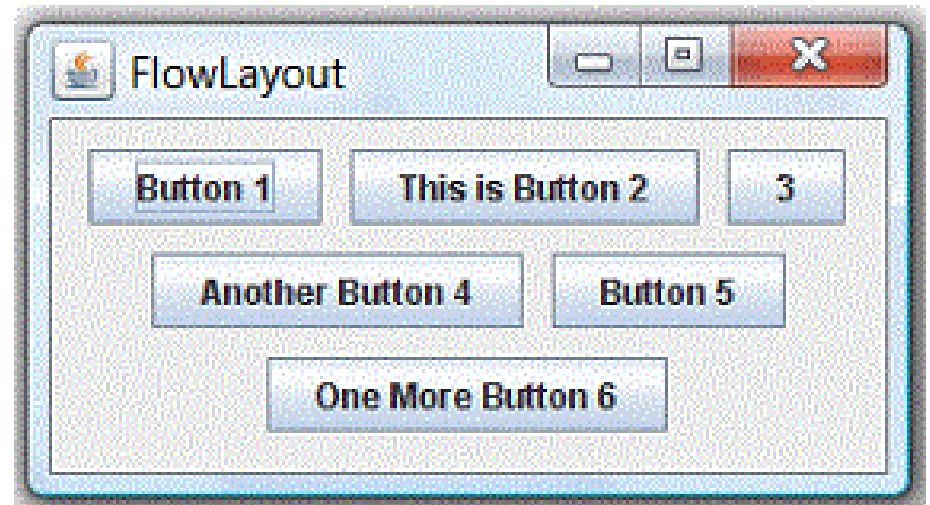
### 2.12 Layout

- 1 container có một layout để sắp xếp các components
- AWT cung cấp các lớp layout sau:
  - FlowLayout
  - GridLayout,
  - BorderLayout,
  - GridBagLayout
  - BoxLayout,
  - CardLayout

# FlowLayout

`aContainer.add(aComponent)`

- Từ trái sang phải
- Tự xuống dòng mới



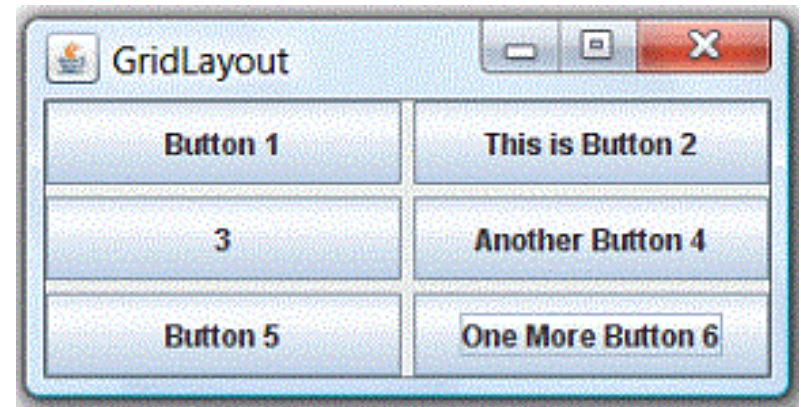
## Constructors

```
public FlowLayout();  
public FlowLayout(int align);  
public FlowLayout(int align, int hgap, int vgap);  
    // align: FlowLayout.LEFT (or LEADING), FlowLayout.RIGHT (or TRAILING), or FlowLayout.CENTER  
    // hgap, vgap: horizontal/vertical gap between the components  
    // By default: hgap=5, vgap=5, align=CENTER
```

# GridLayout

`aContainer.add(aComponent)`

- `aComponent` được sắp xếp vào một lưới
- Từ trái sang phải, trên xuống dưới



## Constructors

```
public GridLayout(int rows, int columns);  
public GridLayout(int rows, int columns, int hgap, int vgap);  
    // By default: rows=1, cols=0, hgap=0, vgap=0
```

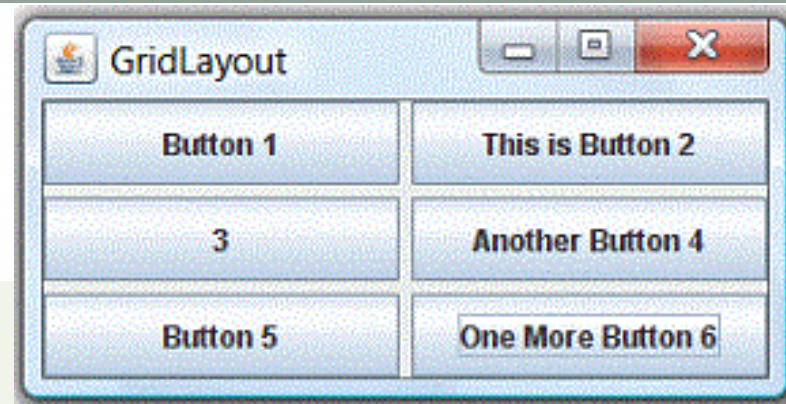


# GridLayout

```
/** Constructor to setup GUI components */
public AWTGridLayoutDemo () {
    setLayout(new GridLayout(3, 2, 3, 3));
    // "this" Frame sets layout to 3x2 GridLayout, horizontal and vertical gaps of 3 pixels

    // The components are added from left-to-right, top-to-bottom
    btn1 = new Button("Button 1");
    add(btn1);
    btn2 = new Button("This is Button 2");
    add(btn2);
    btn3 = new Button("3");
    add(btn3);
    btn4 = new Button("Another Button 4");
    add(btn4);
    btn5 = new Button("Button 5");
    add(btn5);
    btn6 = new Button("One More Button 6");
    add(btn6);

    setTitle("GridLayout Demo"); // "this" Frame sets title
    setSize(280, 150);         // "this" Frame sets initial size
    setVisible(true);           // "this" Frame shows
}
```





### 3. Lập trình GUI với SWING