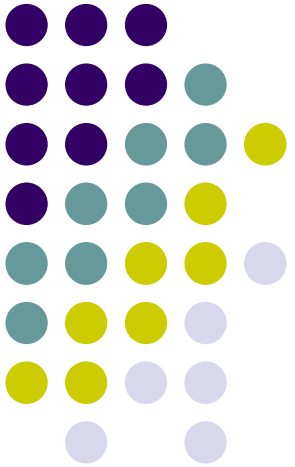
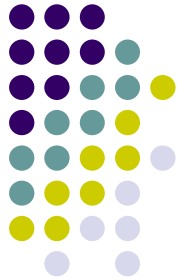
# GUI Event Handling

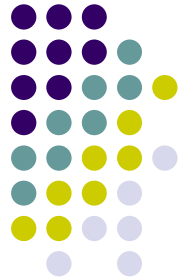


# Nội dung



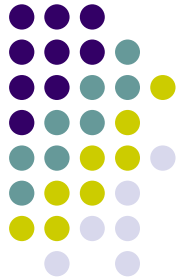
- Mô hình sự kiện
- Những lớp sự kiện
- Những Event Listener
  - Phương thức ActionListener
  - Phương thức MouseListener
  - Phương thức MouseMotionListener
  - Phương thức WindowListener
  - Những hướng dẫn cho việc tạo những ứng dụng xử lý sự kiện.

# Mô hình sự kiện



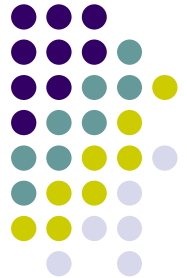
- Mô hình sự kiện
  - Mô hình để xử lý những tương tác của người dùng với thành phần giao diện.
  - Miêu tả bằng cách nào chương trình có thể trả lời những tương tác người dùng.
- 3 thành phần quan trọng
  - Event source
  - Event Listener/Handler
  - Event Object

# Mô hình sự kiện



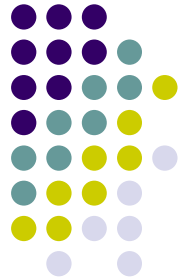
- Event source
  - Thành phần giao diện tạo ra sự kiện.
  - Ví dụ: button, mouse, keyboard
- Event Listener/Handler
  - Nhận những sự kiện và xử lý tương tác của người dùng
  - Ví dụ:
    - Hiển thị những thông tin cho người dùng.
    - Tác vụ tính toán …

# Mô hình sự kiện



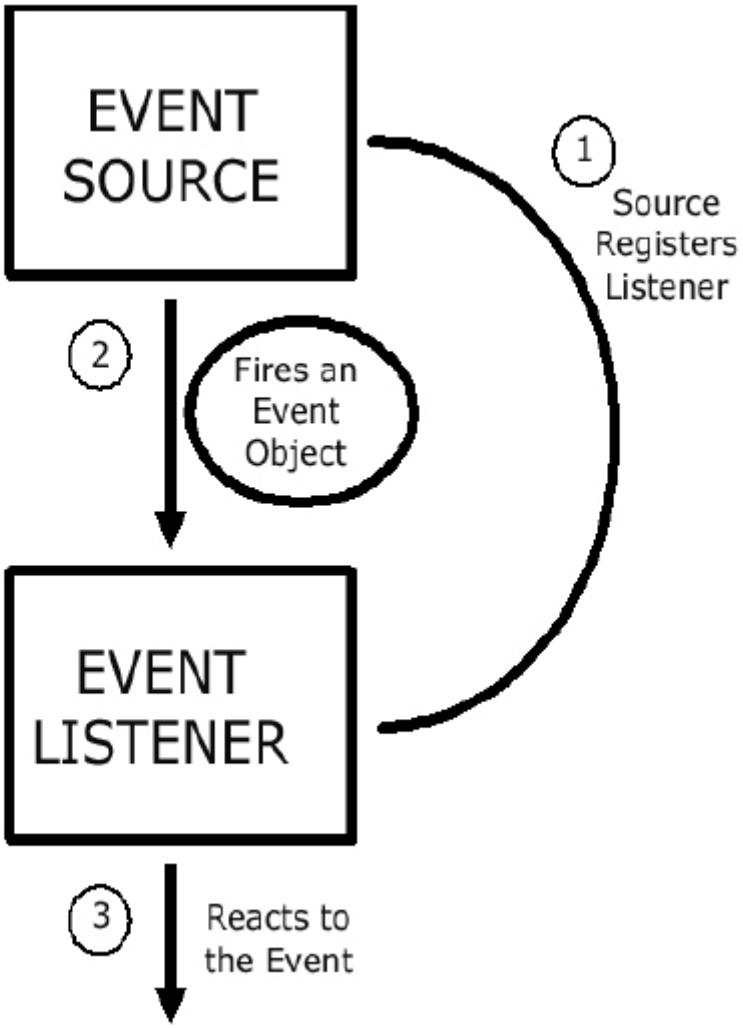
- Event Object
  - Tạo ra khi sự kiện xảy ra.
  - Chứa tất cả những thông tin về sự kiện mà nó xảy ra:
    - Loại sự kiện xảy ra.
    - Nguồn của sự kiện

# Mô hình sự kiện

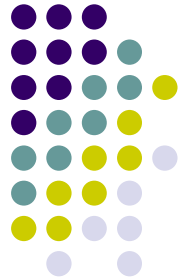


- Một listener được đăng ký với một nguồn
- Listener được đăng ký, và nó chờ cho tới khi sự kiện xảy ra.
- Khi sự kiện xảy ra
  - Một đối tượng sự kiện được tạo ra
  - Đối tượng sự kiện được kích hoạt bằng nguồn đã đăng ký listener
- Mỗi lần listener nhận đối tượng sự kiện từ nguồn.
  - Giải mã những thông điệp
  - Xử lý sự kiện mà nó xuất hiện

# Mô hình sự kiện



# Đăng ký Listener



- Nguồn sự kiện đăng ký một listener

```
void add<Type>Listener(<Type>Listener
listenerObj)
```

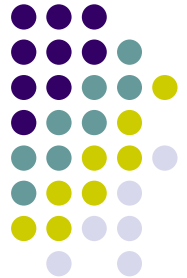
Trong đó:

  - <Type> tùy thuộc vào loại nguồn sự kiện: Key, Mouse, Focus, Component, Action và những cái khác.
  - Một nguồn sự kiện có thể đăng ký vài Listener.

- Listener huỷ bỏ đăng ký.

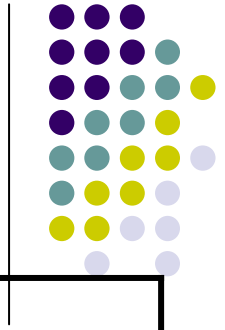
```
void remove<Type>Listener(<Type>Listener
listenerObj)
```

# Những class sự kiện



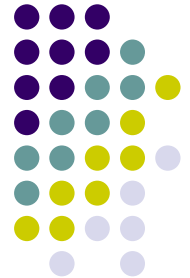
- Một đối tượng sự kiện có class sự kiện như và kiểu dữ liệu tham chiếu của nó.
- Lớp *EventObject*
  - Trong gói *java.util*
- Lớp *AWTEvent*
  - Một lớp con trực tiếp của *EventObject*
  - Định nghĩa trong gói *java.awt*
  - Gốc của tất cả các sự kiện dựa trên AWT
  - Lớp con có tên quy ước **<Type>Event**

# Những lớp sự kiện



| 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 trìu tượng cho tất cả các lớp sự kiện input |
| ActionEvent | Exends 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 của sổ close, open, active, deactive, iconified, deiconified hoặc khi focus được chuyển vào trong hoặc ra ngoài cửa sổ. |

# Những sự kiện Listener



- Những lớp mà nó thể hiện những <Type>Listener interface

| Event class | Miêu tả |
|---|---|
| ActionListener | Nhận những sự kiện action |
| MouseListener | Nhận những sự kiện mouse |
| MouseMotionListener | Nhận những sự kiện di chuyển chuột, bao gồm kéo lê và di chuyển chuột |
| WindowListener | Nhận những sự kiện liên quan đến cửa sổ |

# Phương thức của ActionListener

- Chỉ có 1 phương thức.

```
public void actionPerformed(ActionEvent e)
```

Chứa xử lý cho sự kiện ActionEvent khi nó xảy ra

# Những phương thức của MouseListener

- `public void mouseClicked(MouseEvent e)`

Chứa những xử lý cho sự kiện chuột click.

- `public void mouseEntered(MouseEvent e)`

Chứa những xử lý cho sự kiện chuột di chuyển vào component

- `public void mouseExited(MouseEvent e)`

Chứa những xử lý cho sự kiện chuột ra khỏi component

- `public void mousePressed(MouseEvent e)`

Chứa những xử lý cho sự kiện nhấn chuột

- `public void mouseReleased(MouseEvent e)`

Chứa những xử lý cho sự kiện release chuột

# Những phương thức của MouseMotionListener

- `public void mouseDragged(MouseEvent e)`

  Chứa những xử lý khi phím chuột nhấn vào component sau đó kéo lê chuột.

- `public void mouseMoved(MouseEvent e)`

  Chứa những xử lý khi con trỏ chuột di chuyển vào trong component, không nhấn phím chuột.

# Những phương thức của WindowListener

- `public void windowOpened(WindowEvent e)`

Chứa những xử lý cho sự kiện cửa sổ được mở .

- `public void windowClosing(WindowEvent e)`

Chứa những xử lý cho sự kiện cửa sổ chuẩn bị đóng

- `public void windowClosed(WindowEvent e)`

Chứa những xử lý cho sự kiện cửa sổ sau khi đóng

- `public void windowActivated(WindowEvent e)`

Chứa những xử lý cho sự kiện cửa sổ kích hoạt

- `public void windowDeactivated(WindowEvent e)`

Chứa những xử lý cho sự kiện ẩn cửa sổ

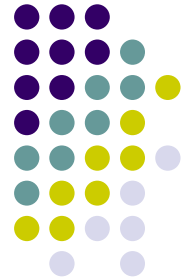
- `public void windowIconified(WindowEvent e)`

Chứa những xử lý cho sự kiện làm thu nhỏ cửa sổ

- `public void windowDeiconified(WindowEvent e)`

Chứa những xử lý cho sự kiện làm phóng to cửa sổ

# Tạo các ứng dụng giao diện với xử lý sự kiện

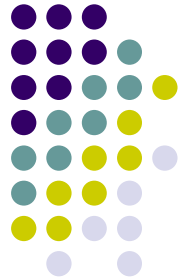


1. Tạo GUI class
   - Miêu tả và hiển thị hình ảnh của ứng dụng giao diện của bạn
2. Tạo một class thực thi listener inteface phù hợp
   - Có thể tham chiếu tới cùng class trong bước 1
3. Trong việc thực thi class
   - Override tất cả các phương thức của listener phù hợp
   - Miêu tả trong mỗi phương thức mà bạn muốn sự kiện được xử lý như thế nào
   - Có thể đưa ra những thực thi rỗng cho những phương thức mà bạn không cần
4. Đăng ký đối tượng listener với nguồn
   - Đối tượng là thể hiện của lới listener trong bước 2
   - Sử dụng phương thức add<Type>Listener

# Ví dụ Mouse Event

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class MouseEventsDemo extends JFrame implements
                        MouseListener, MouseMotionListener {
    JTextField tf;
    public MouseEventsDemo(String title){
            super(title);
            tf = new JTextField(60);
            addMouseListener(this);
    }
    public void launchFrame(Container c) {
            /* Add components to the frame */
            c.setLayout(new BorderLayout());
            c.add(tf, BorderLayout.SOUTH);
            setSize(300,300);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setVisible(true);
     }
    public void mouseClicked(MouseEvent me) {
            String msg = "Mouse clicked.";
            tf.setText(msg);
    }
```

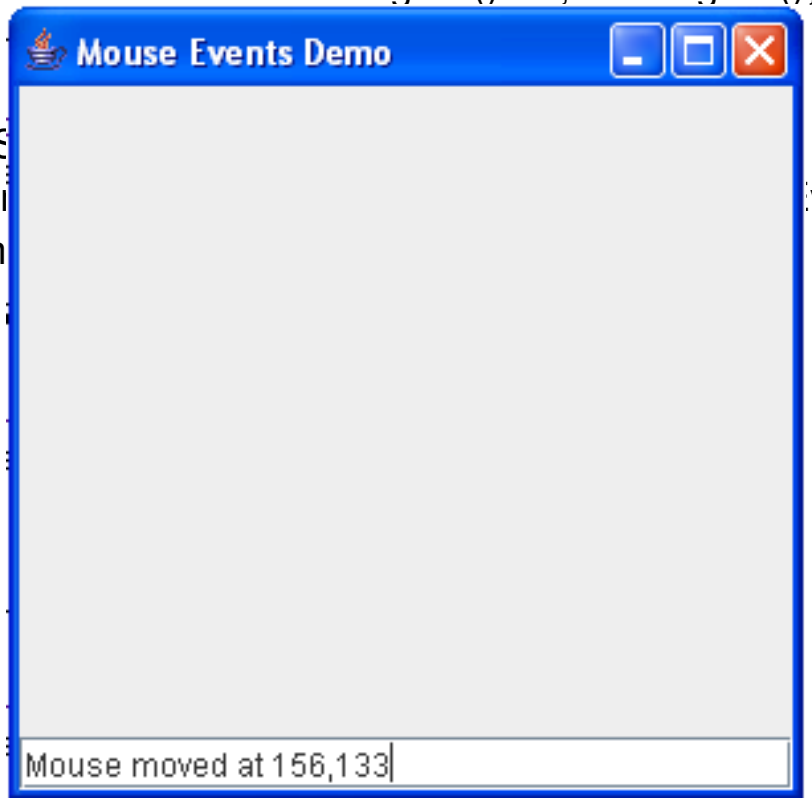
# Ví dụ Mouse Event



```
public void mouseEntered(MouseEvent me) {
      String msg = "Mouse entered component.";
      tf.setText(msg);
}
public void mouseExited(MouseEvent me) {
       String msg = "Mouse exited component.";
       tf.setText(msg);
}
public void mousePressed(MouseEvent me) {
      String msg = "Mouse pressed.";
      tf.setText(msg);
}
public void mouseReleased(MouseEvent me) {
      String msg = "Mouse released.";
       tf.setText(msg);
}
public void mouseDragged(MouseEvent me) {
      String msg = "Mouse dragged at " + me.getX() + "," + me.getY();
      tf.setText(msg);
}
```

# Ví dụ Mouse Event

```
    public void mouseMoved(MouseEvent me) {
            String msg = "Mouse moved at " + me.getX() + "," + me.getY();
            tf.setText(msg);
    }
    public static void main(S
            MouseEventsDe                                          vents Demo");
            med.launchFram
    }
}
```

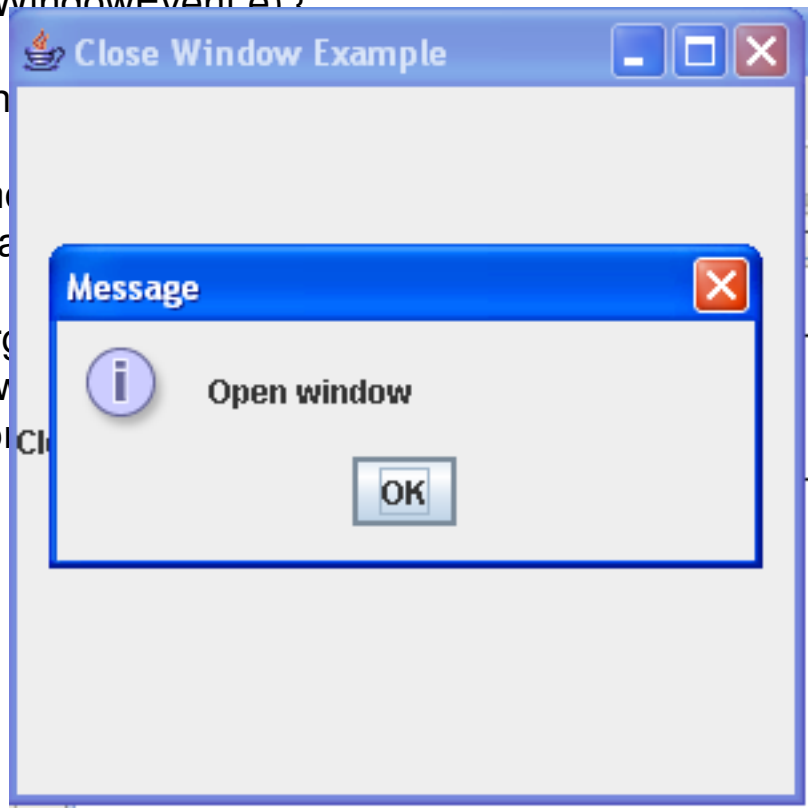


# Ví dụ Close Window

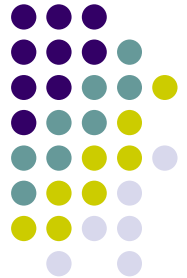
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CloseFrame extends JFrame implements WindowListener {
    JLabel label;
    public CloseFrame(String title) {
        super(title);
        label = new JLabel("Close the frame.");
        this.addWindowListener(this);
    }
    void launchFrame(Container c) {
        c.add(label);
        setSize(300,300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setVisible(true);
    }
    public void windowActivated(WindowEvent e) {
    }
    public void windowClosed(WindowEvent e) {
    }
    public void windowClosing(WindowEvent e) {
        JOptionPane.showMessageDialog(this, "Close window");
    }
```

# Ví dụ Close Window

```
    public void windowDeactivated(WindowEvent e) {
    }
    public void windowDeiconified(W
    }
    public void windowIconified(Win
    }
    public void windowOpened(Win
            JOptionPane.showMessa
    }
    public static void main(String ar
            CloseFrame cf =      new
            cf.launchFrame(cf.getCo
    }
}
```

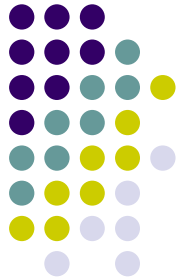


# Những class Adapter



- Tại sao phải sử dụng những class Adapter?
  - Thực thi tất cả các phương thức của interface sẽ mất nhiều thời gian.
  - Chỉ cần quan tâm chỉ thực thi một vài phương thức
- Những class Adapter
  - Xây dựng bằng Java
  - Thực thi tất cả phương thức của listener
  - Những thực thi của các phương thức là rỗng.

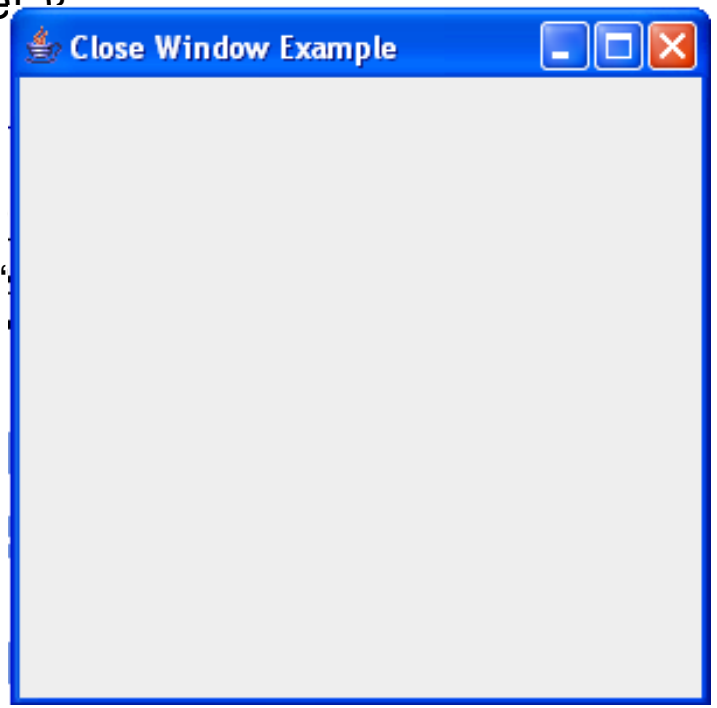
# Ví dụ



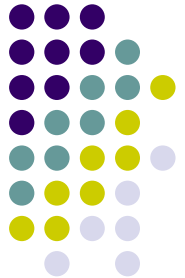
```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CloseFrameAdapter extends JFrame{
    CFListener w = new CFListener(this);
    public CloseFrameAdapter(String title) {
        super(title);
        this.addWindowListener(w);
    }
    void launchFrame(Container c) {
        setSize(300,300);
        setVisible(true);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        CloseFrameAdapter cf = new CloseFrameAdapter("Close Window Example");
        cf.launchFrame(cf.getContentPane());
    }
}
```

# Ví dụ

```
class CFListener extends WindowAdapter {
    CloseFrameAdapter ref;
    public CFListener( CloseFrameAdapter ref ){
        this.ref = ref;
    }
    public void windowClosing(WindowEvent
        JOptionPane.showMessageDialog(this, '

    }
}
```



# Inner Classes

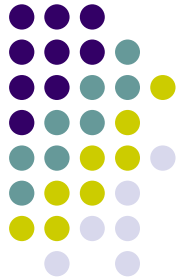


- Class được khai báo trong class khác
- Tại sao phải sử dụng inner class
  - Giúp đơn giản chương trình
  - Đặc biệt là trong xử lý sự kiện.

# Ví dụ

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

class CloseFrameInnerClass extends JFrame{
	public CloseFrameInnerClass(String title) {
		super(title);
		this.addWindowListener(new CFListener());
	 }
	void launchFrame() {
		setSize(300,300);
		setVisible(true);
	}
	class CFListener extends WindowAdapter {
		public void windowClosing(WindowEvent e) {
			System.out.println("window closing");
			dispose();
			System.exit(1);
		}
	}

	public static void main(String args[]) {
		CloseFrameInnerClass cf = new CloseFrameInnerClass("Close Window Example");
		cf.launchFrame();
	}
}
```

# Giấu tên những class inner

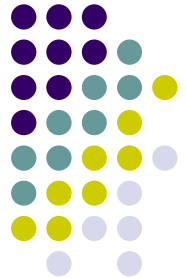


- Không đặt tên những class inner
- Tại sao lại giấu tên những class inner?
  - Làm cho code của bạn đơn giản hơn.
  - Đặc biệt trong xử lý sự kiện.

# Ví dụ

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

class CloseFrameAnon extends JFrame{
	public CloseFrameAnon(String title) {
		super(title);
		this.addWindowListener(new WindowAdapter() {
			public void windowClosing(WindowEvent e) {
				System.out.println("window closing");
				dispose();
				System.exit(1);
			}
		});
	 }
	void launchFrame() {
		setSize(300,300);
		setVisible(true);
	}
	public static void main(String args[]) {
		CloseFrameAnon cf = new CloseFrameAnon("Close Window Example");
		cf.launchFrame();
	}
}
```

# Tóm tắt

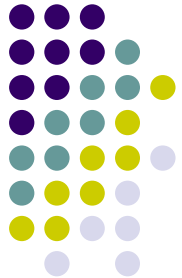


- Mô hình sự kiện
  - Đăng ký listener.

    ```
    void add<Type>Listener(<Type>Listener listenerObj)
    ```

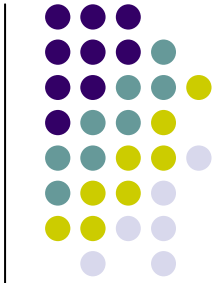
  - Listener chờ khi sự kiện xảy ra.
  - Khi sự kiện xảy ra.
    - Đối tượng sự kiện được tạo ra.
    - Object được kích hoạt bằng nguồn được đăng ký listener.
  - Khi listener nhận đối tượng sự kiện:
    - Giải mã thông điệp.
    - Xử lý sự kiện.

# Tóm tắt



- Những thành phần của mô hình sự kiện.
  - Nguồn sự kiện
  - Xử lý sự kiện
  - Đối tượng sự kiện
- Những class sự kiện
  - Lớp *EventObject*
  - Lớp *AWTEvent*
    - Gốc của tất cả các sự kiện AWT
    - Những lớp con đặt tên theo cách:
      <Type>Event
- Những Listener Event
  - *ActionListener* Method
  - *MouseListener* Methods
  - *MouseMotionListener* Methods
  - *WindowListener* Methods

# Tóm tắt



Tạo một ứng dụng giao diện với sử lý sự kiện

1. Tạo GUI class
   - Miêu tả và hiển thị hình ảnh của ứng dụng giao diện của bạn
2. Tạo một class thực thi listener inteface phù hợp
   - Có thể tham chiếu tới cùng class trong bước 1
3. Trong việc thực thi class
   - Override tất cả các phương thức của listener phù hợp
   - Miêu tả trong mỗi phương thức mà bạn muốn sự kiện được xử lý như thế nào
   - Có thể đưa ra những thực thi rỗng cho những phương thức mà bạn không cần
4. Đăng ký đối tượng listener với nguồn
   - Đối tượng là thể hiện của lới listener trong bước 2
   - Sử dụng phương thức add<Type>Listener