

JAV

Spring 2014

Lecture 7

GUI programming (1)
Basic issues

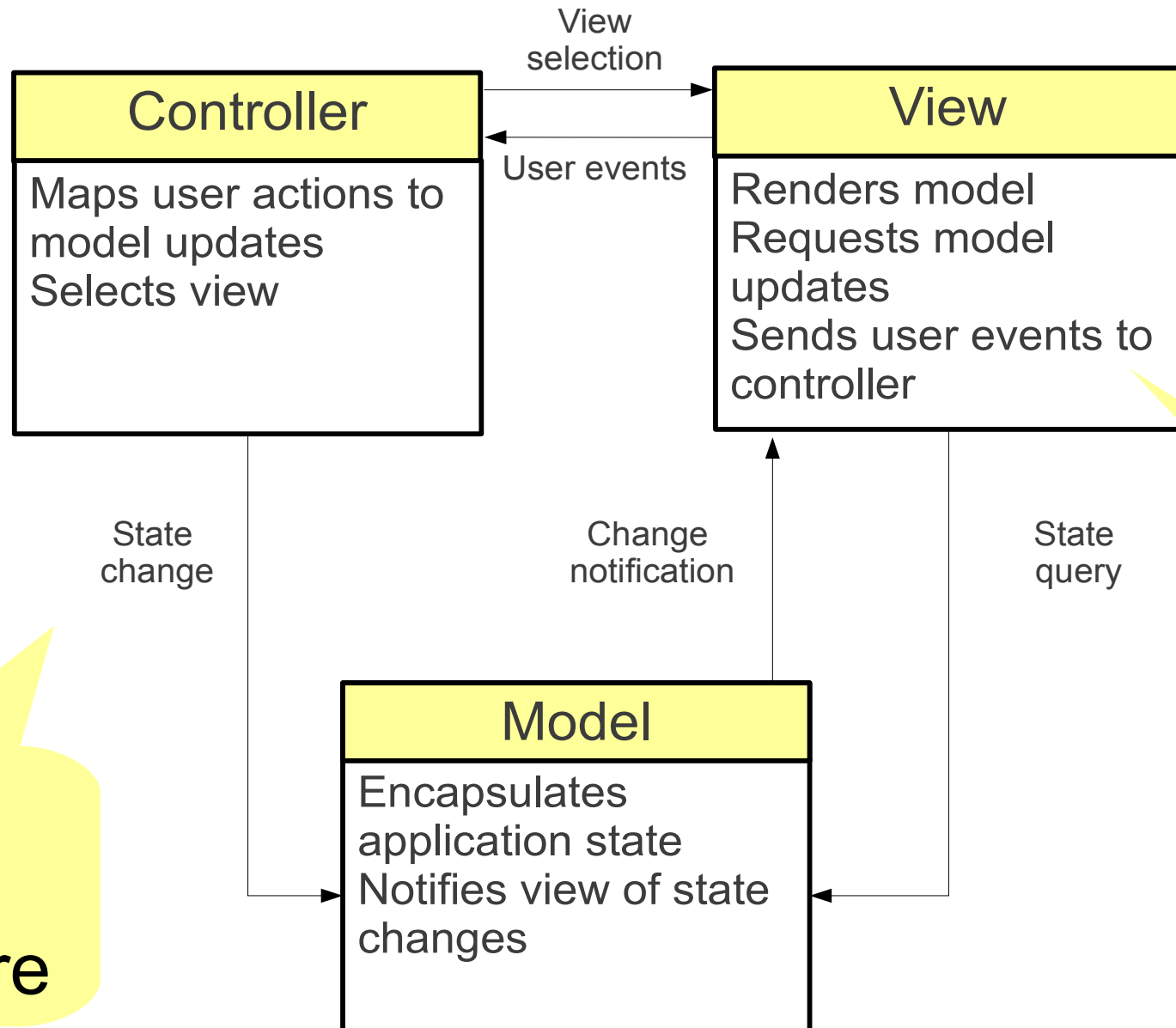
Lecture outline

- GUI & GUI application
- Event driven programming
- Java's support for GUI programming:
 - Swing, AWT
- Basic GUI application:
 - tasks and components

GUI & GUI application

- A **graphical user interface (GUI)** consists of display components of an application
- Display components present information/data graphically
- A **GUI application** is application + GUI:
 - must capture and handle user interactions on GUI

A general application architecture



A.k.a
MVC
architecture

GUI

Our GUI application scope

- 80% GUI:
 - display components & how to create them
 - 3 levels: basic, intermediate, advanced
- 15% user interaction:
 - event-driven programming
- 5% design: intermediate design based on MVC

Java's support for GUI programming (1)

- Two APIs: AWT & Swing
- **Abstract Window Toolkit (AWT):**
 - (older) introduced since JDK 1.0
 - decouple display component specifications from their implementations
 - *heavy-weight*: uses a native peer class for each display component
 - the peer classes are fixed for given platform
 - has a modular interaction model (JDK 1.1)

Java's support for GUI programming (2)

- **Swing:**
 - introduced since JDK 1.2
 - improved from AWT
 - *light-weight*: Java's implementation of the display (no need for peer classes)
 - look & feel of the display components can be customised for a given platform
 - has enhanced display components
 - extends AWT 1.1's interaction model

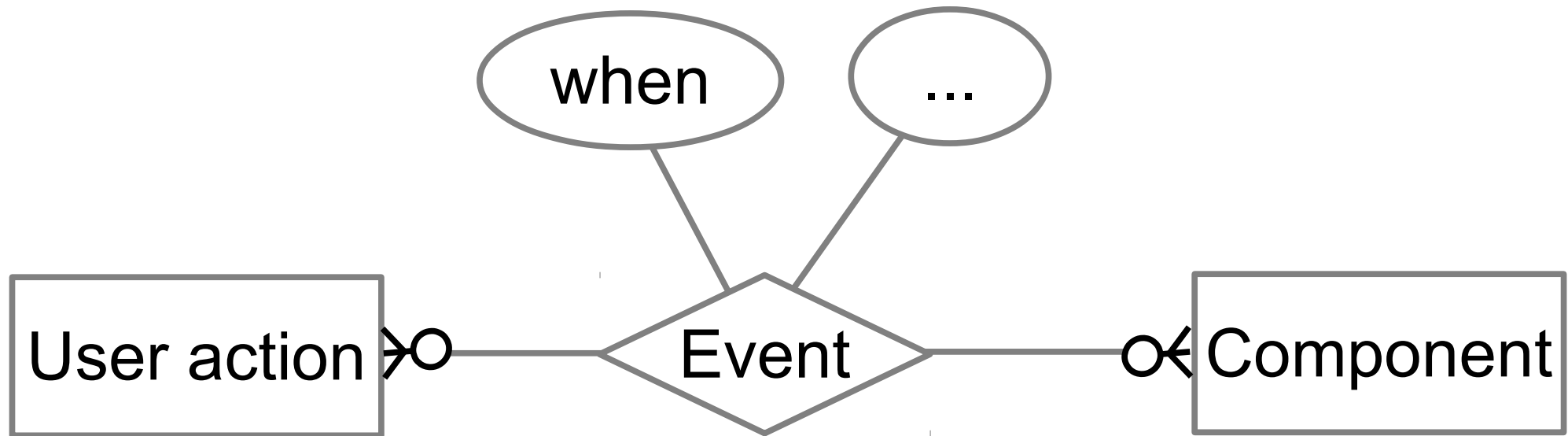
Event-driven programming

- User interaction model
- Event-driven programming (EDP)
- Java's support for EDP

What is EDP?

- Programming technique to *capture* and *handle* events caused by interactions with external entities
- EDP is *not* just for GUI applications
- EDP scope for GUI application:
 - **user** interactions on **GUI** components

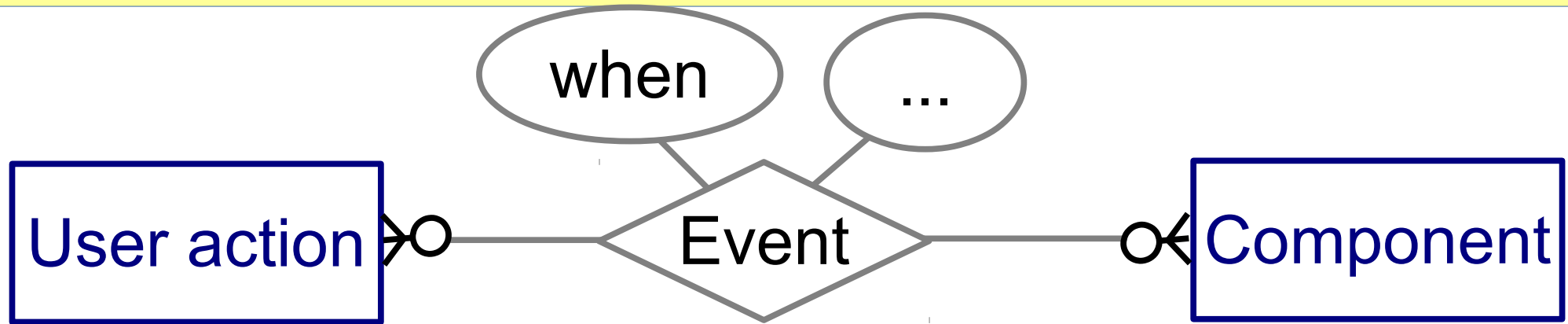
User interaction model



User interaction

- A user interaction consists of a sequence of actions
- *User action* is performed via an input device (mouse or keyboard)
- User action on a display component results in an *event*

User action & component



mouse:

click
double-click
move

...

keyboard:

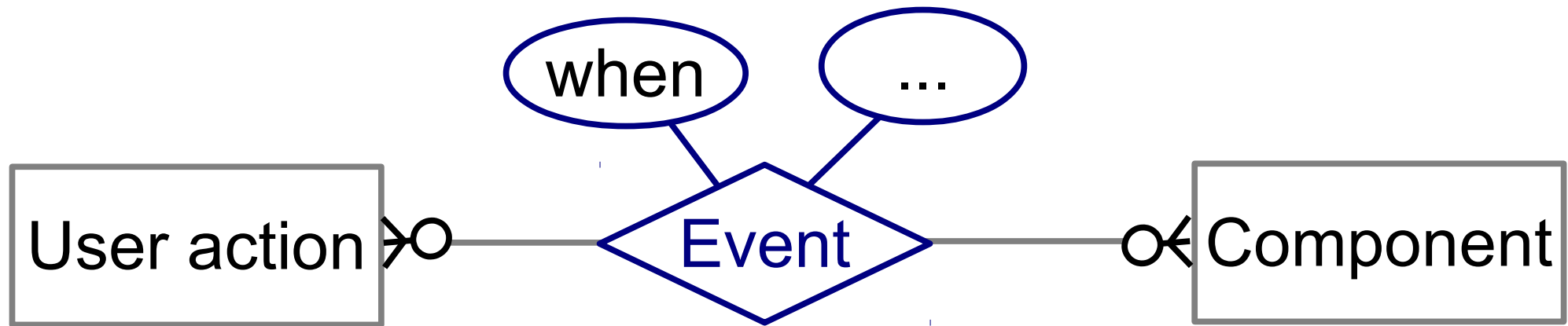
key press
key release

...

window
button
label
text field

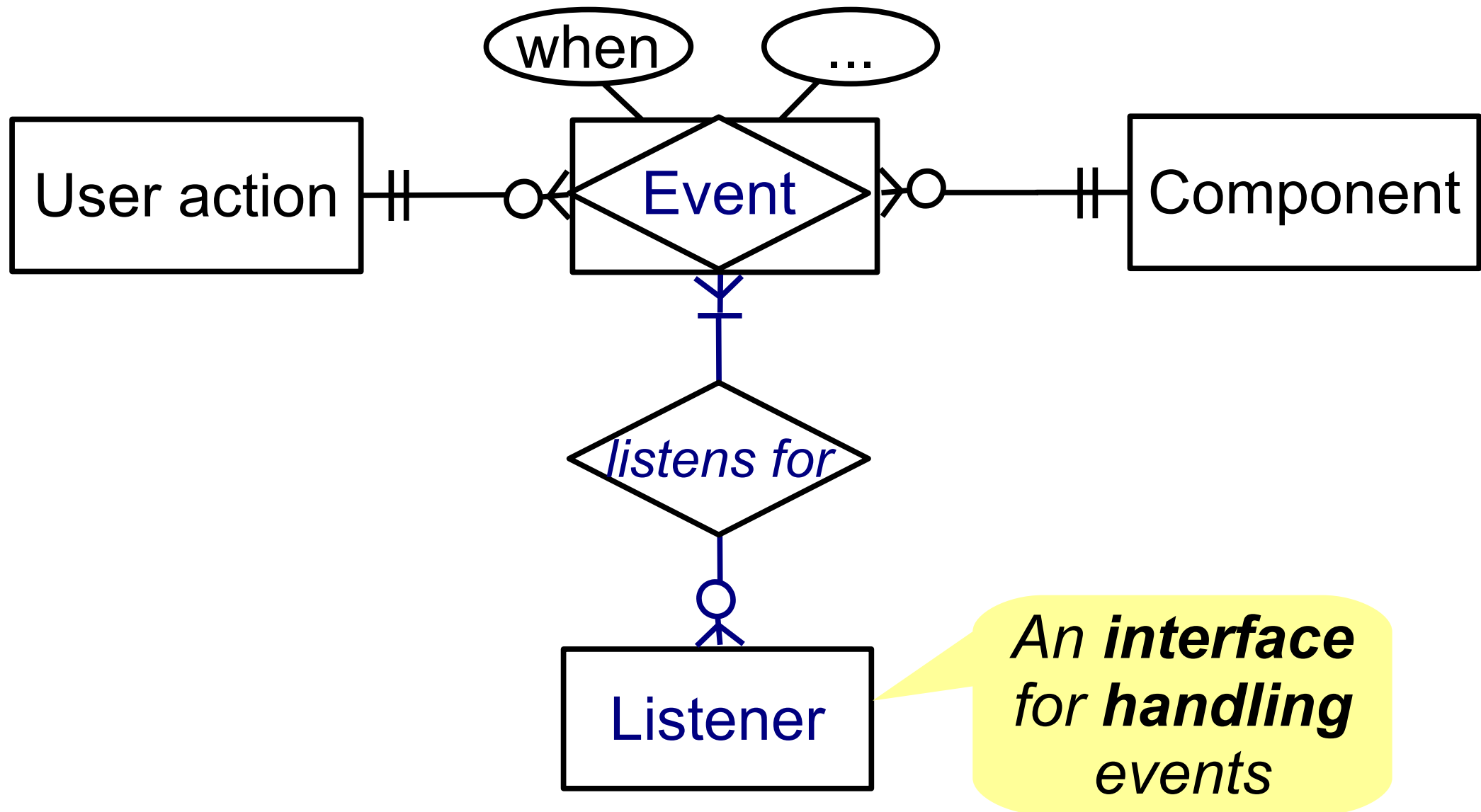
...

What is an event?

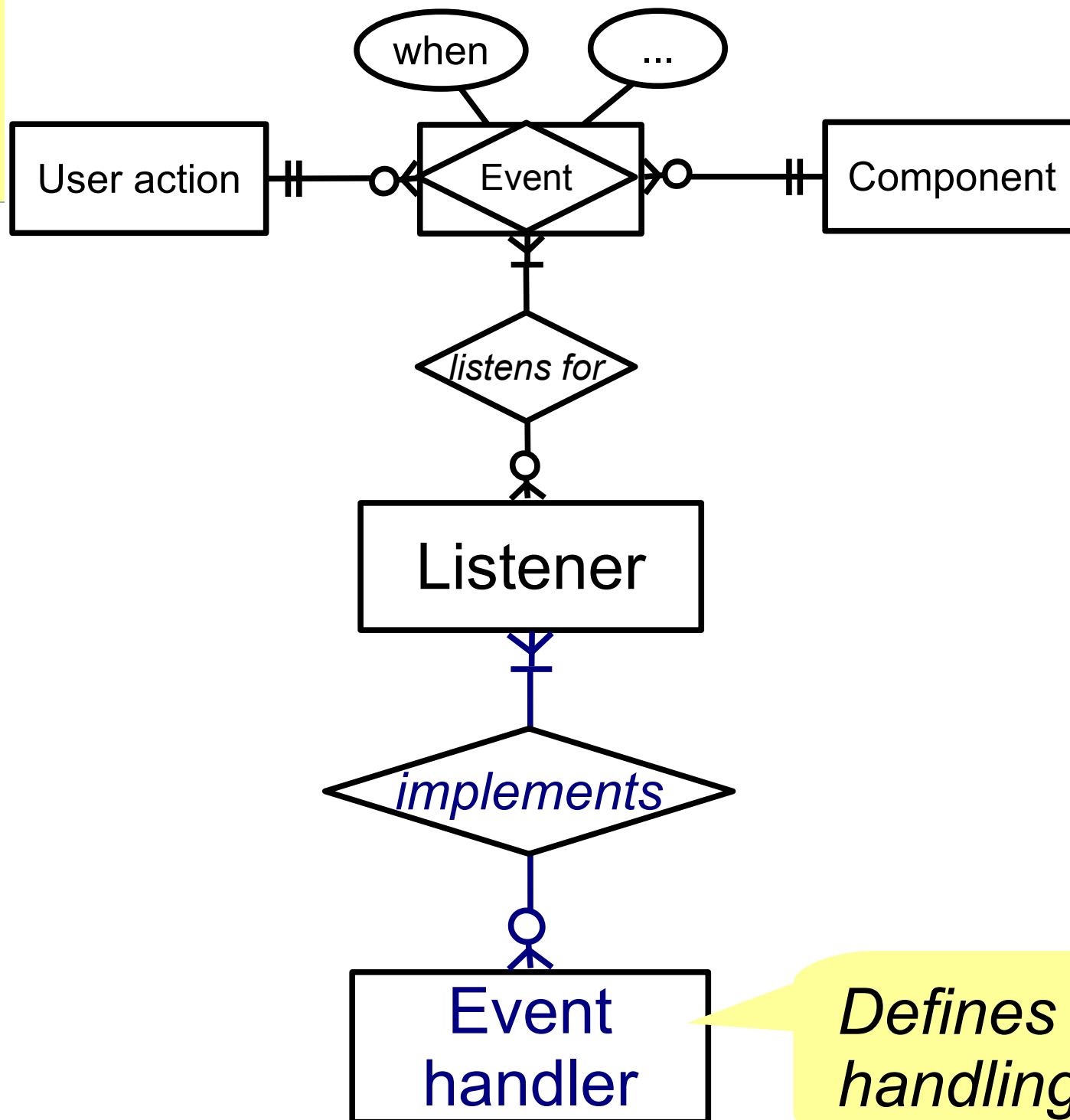


*an occurrence
of a user action
on a display
component*

EDP model (1)

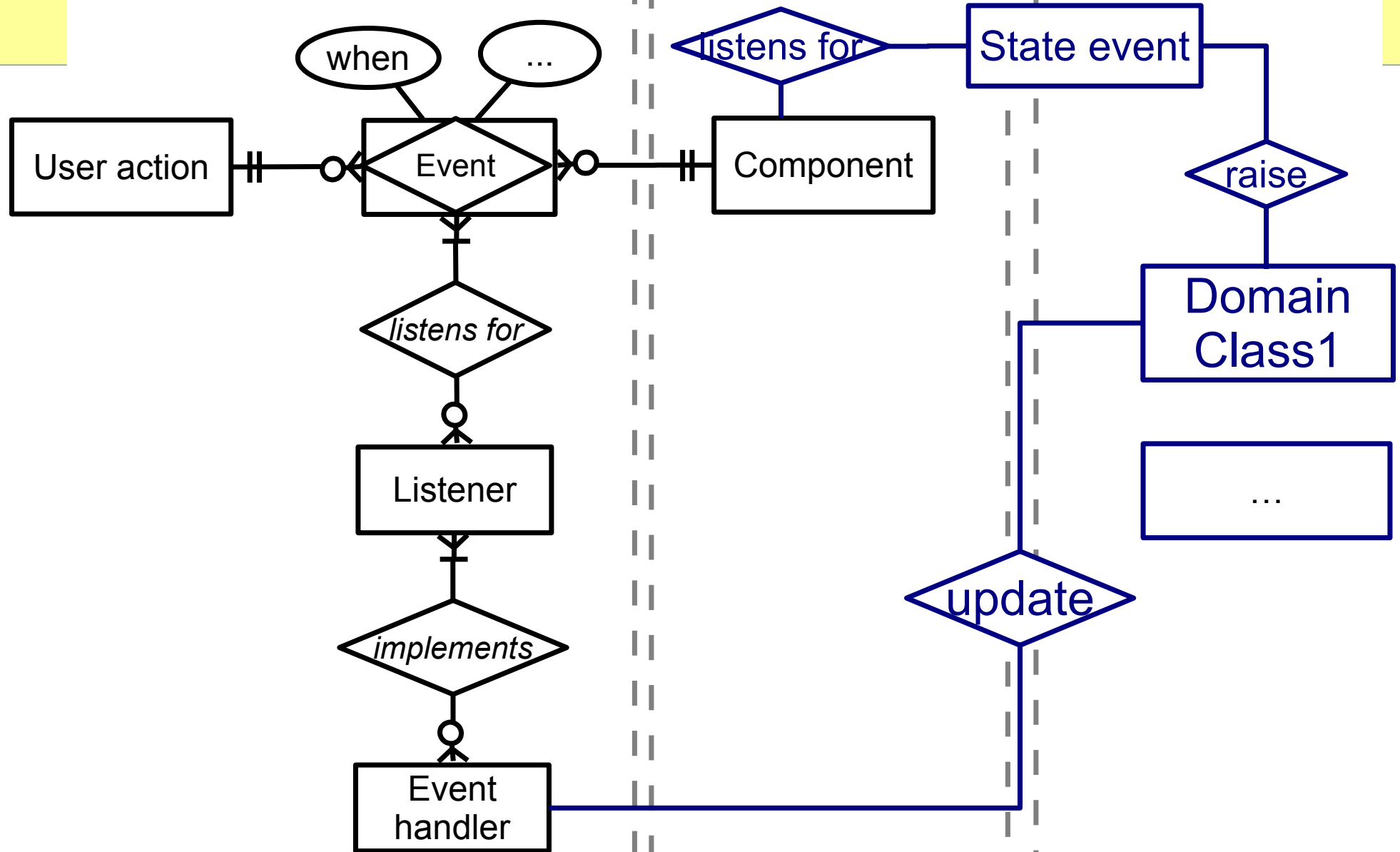


(2)

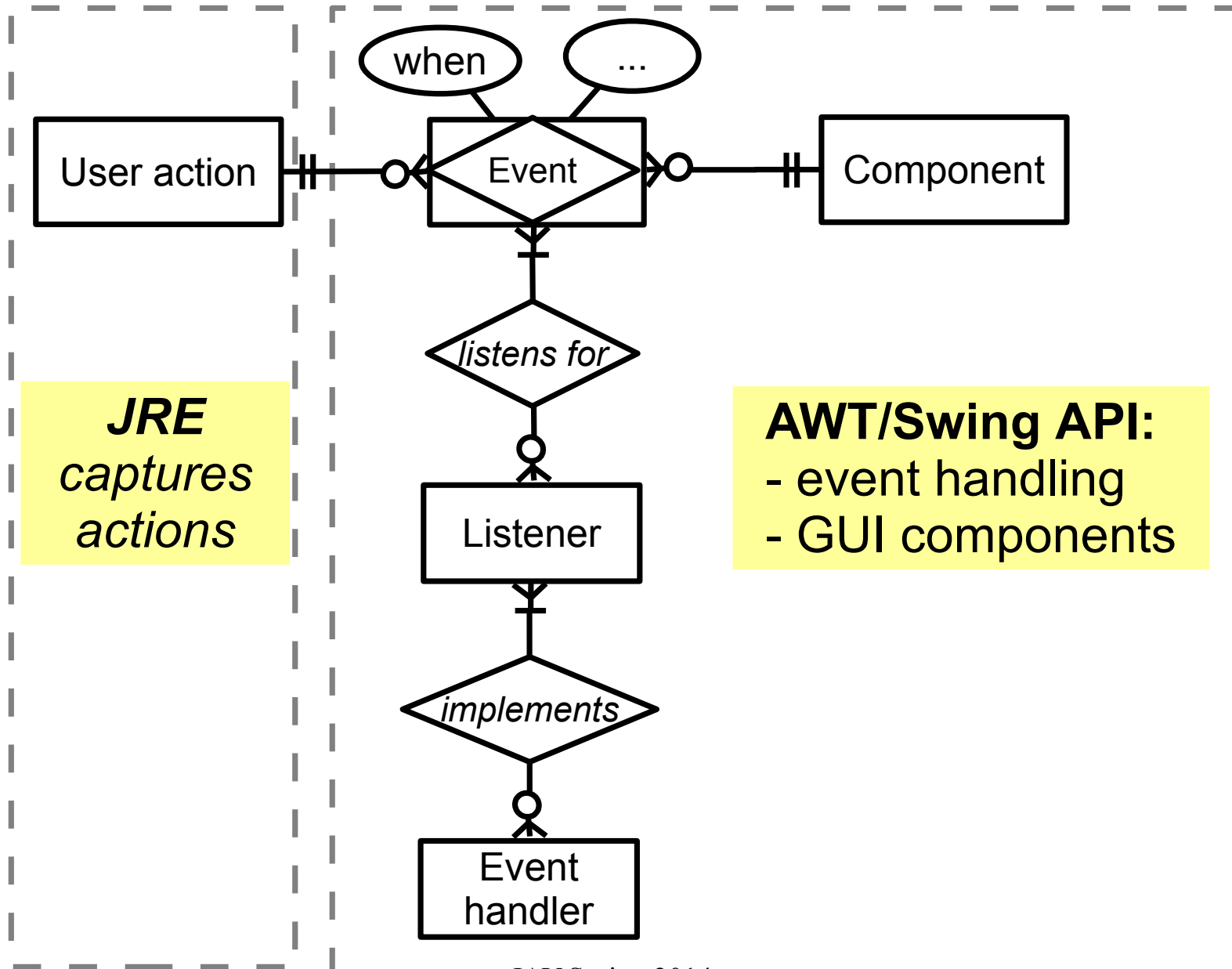


Defines event handling logic

An event-driven GUI architecture



Java's support for EDP



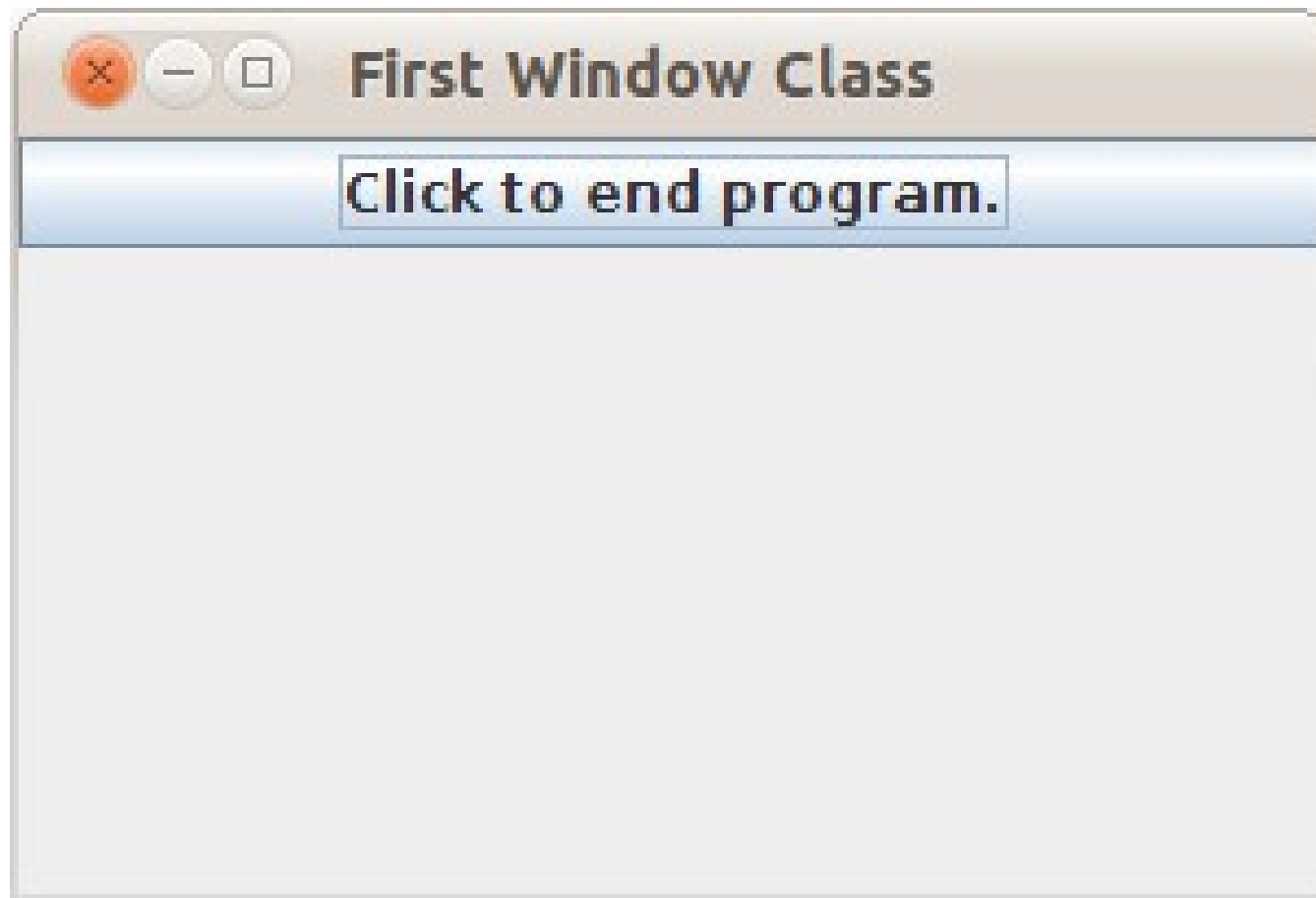
Basic GUI application

- Tasks
- Components:
 - Window
 - Button
 - Color

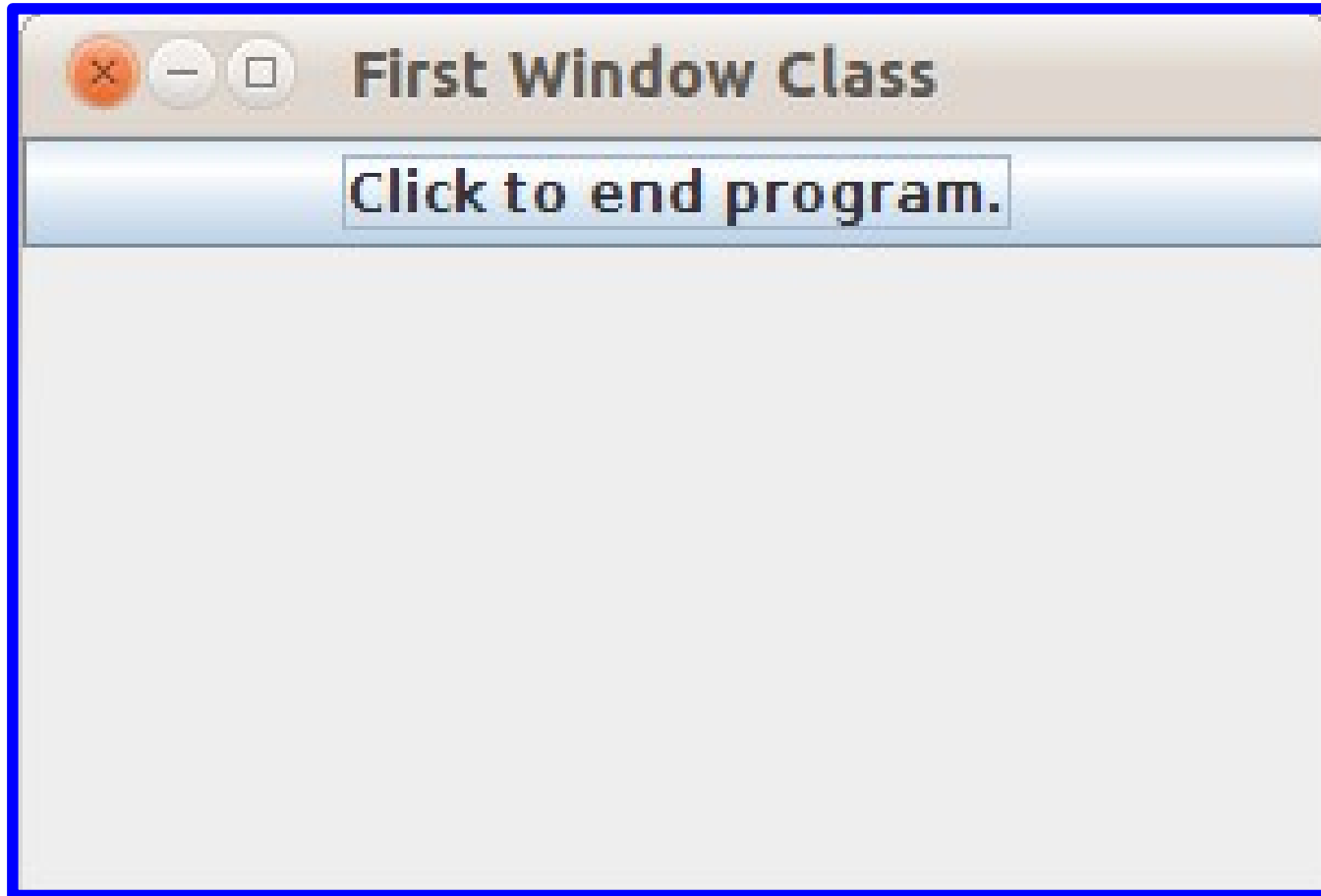
Tasks

- Create a **window** object
- **Set up** the window
- Handle relevant window's **events**
- For each display **component**:
 - create
 - set up
 - handle the relevant events
 - add to the window
- **Display** the window

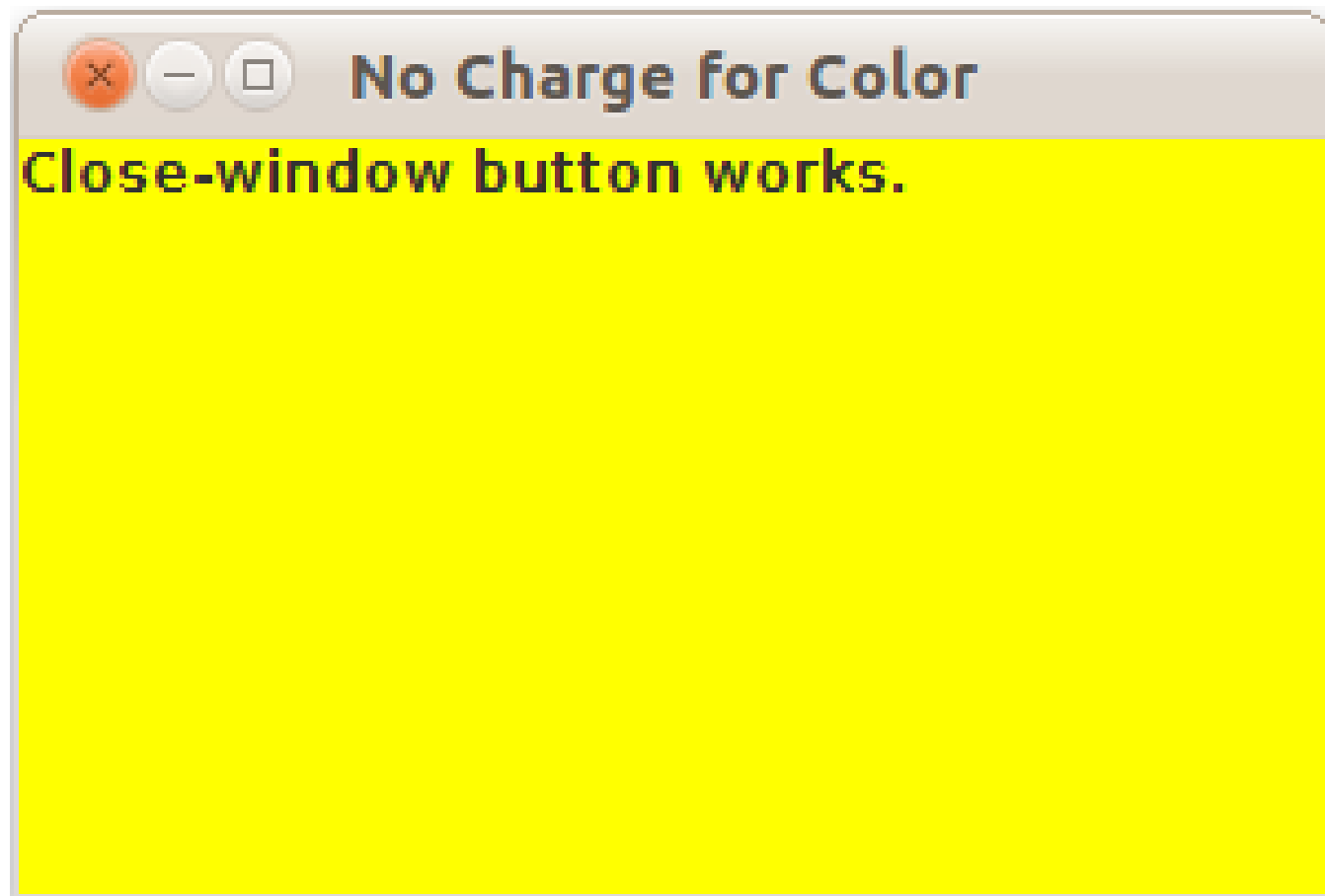
Example



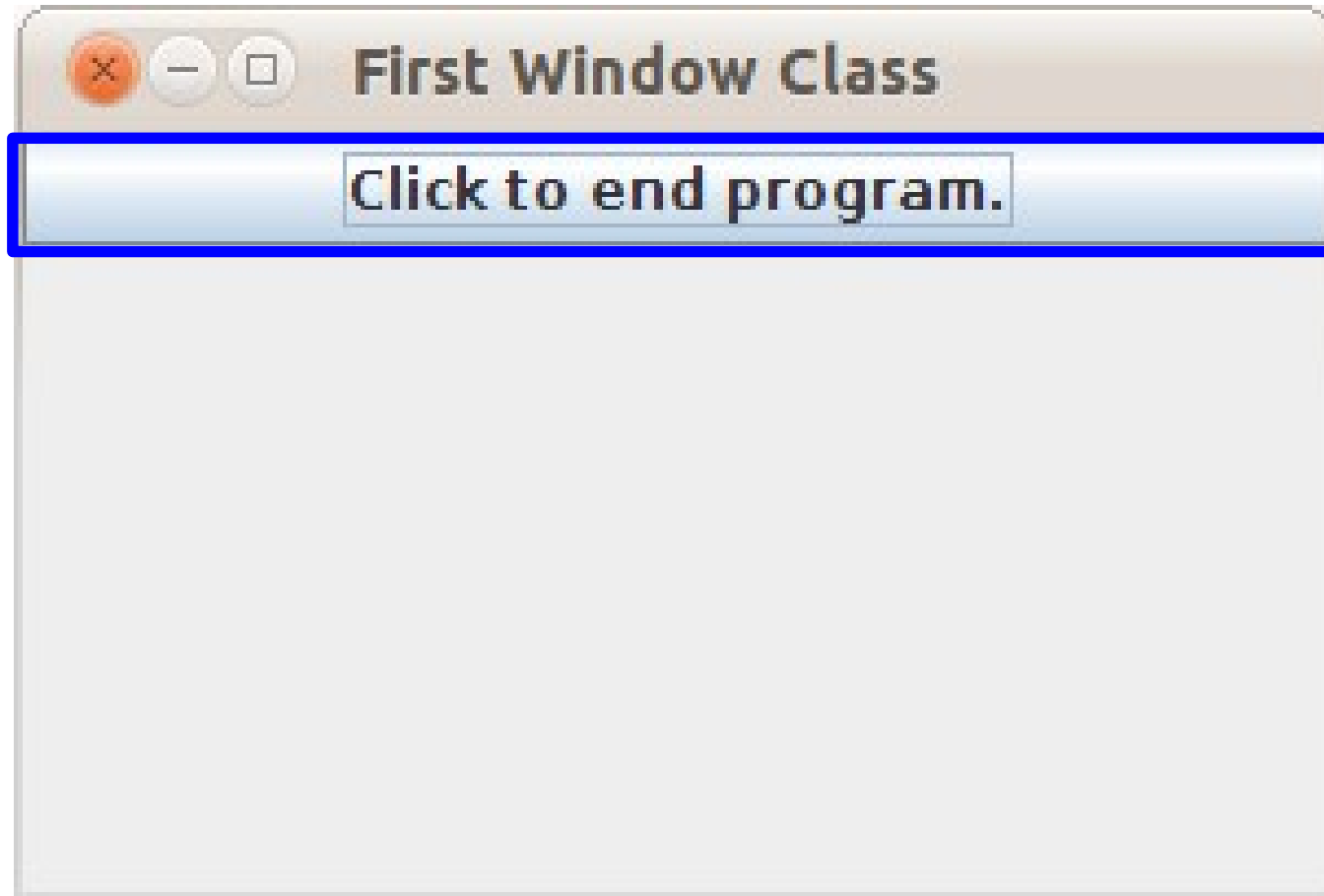
Window



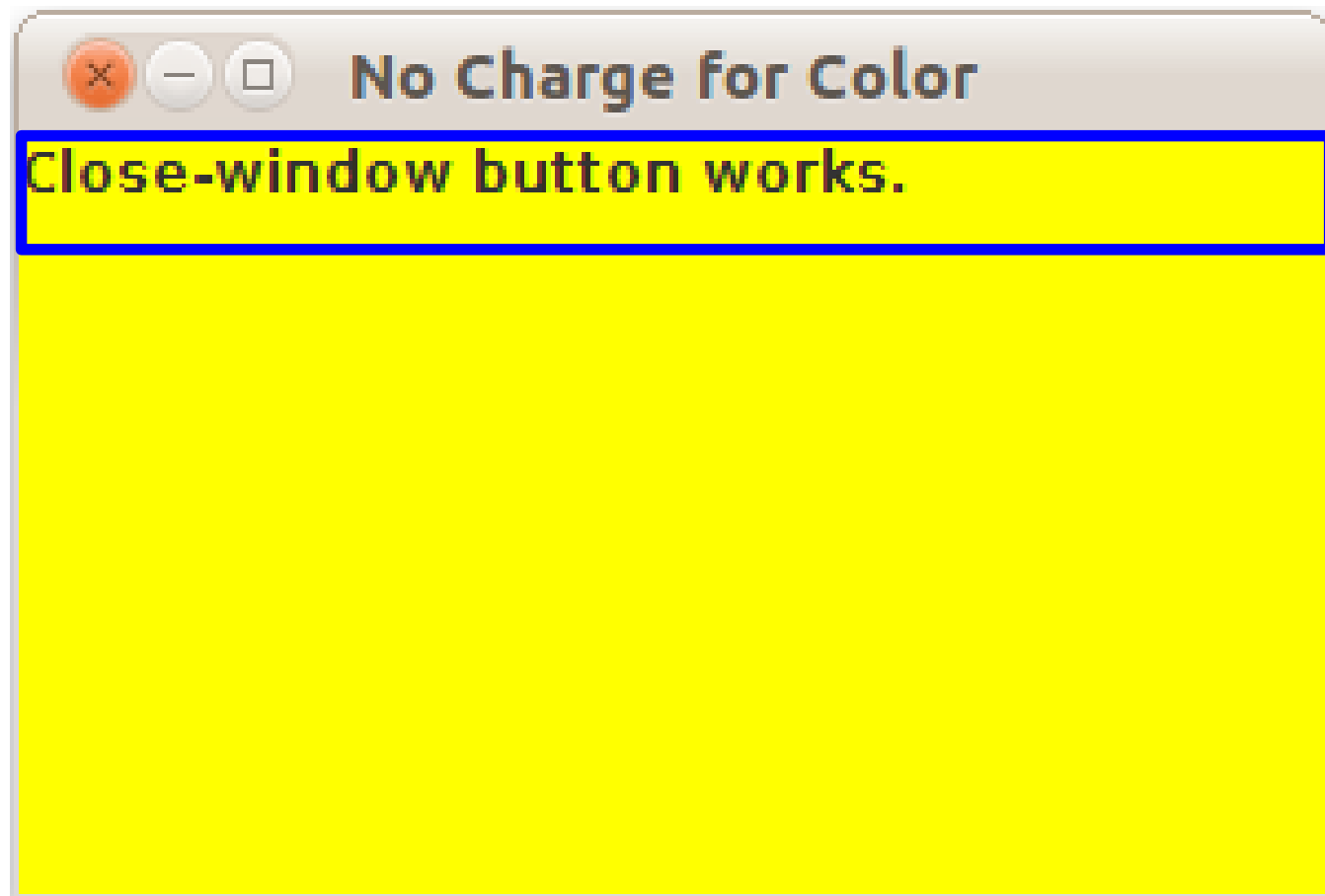
Coloured window



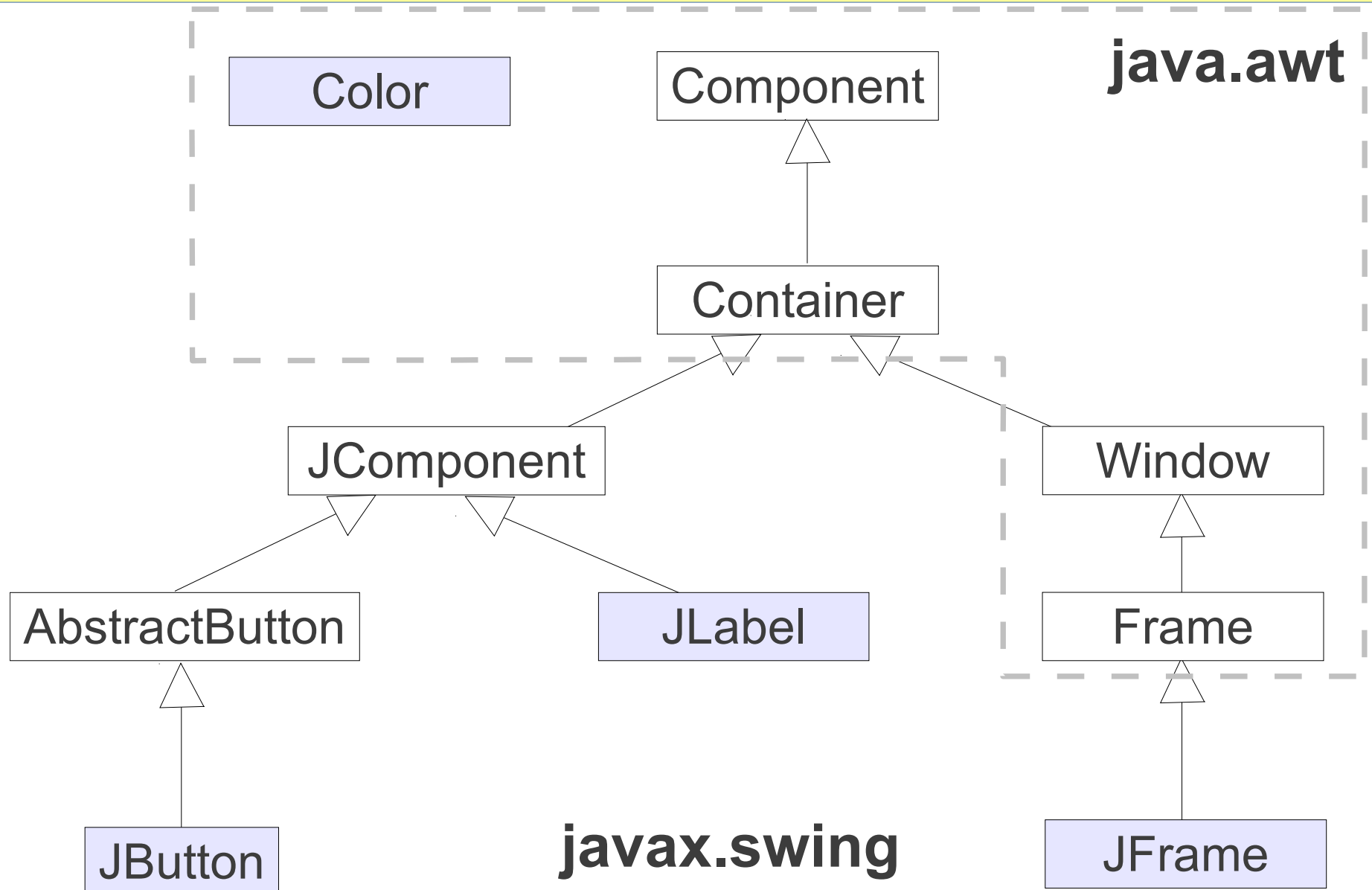
Button



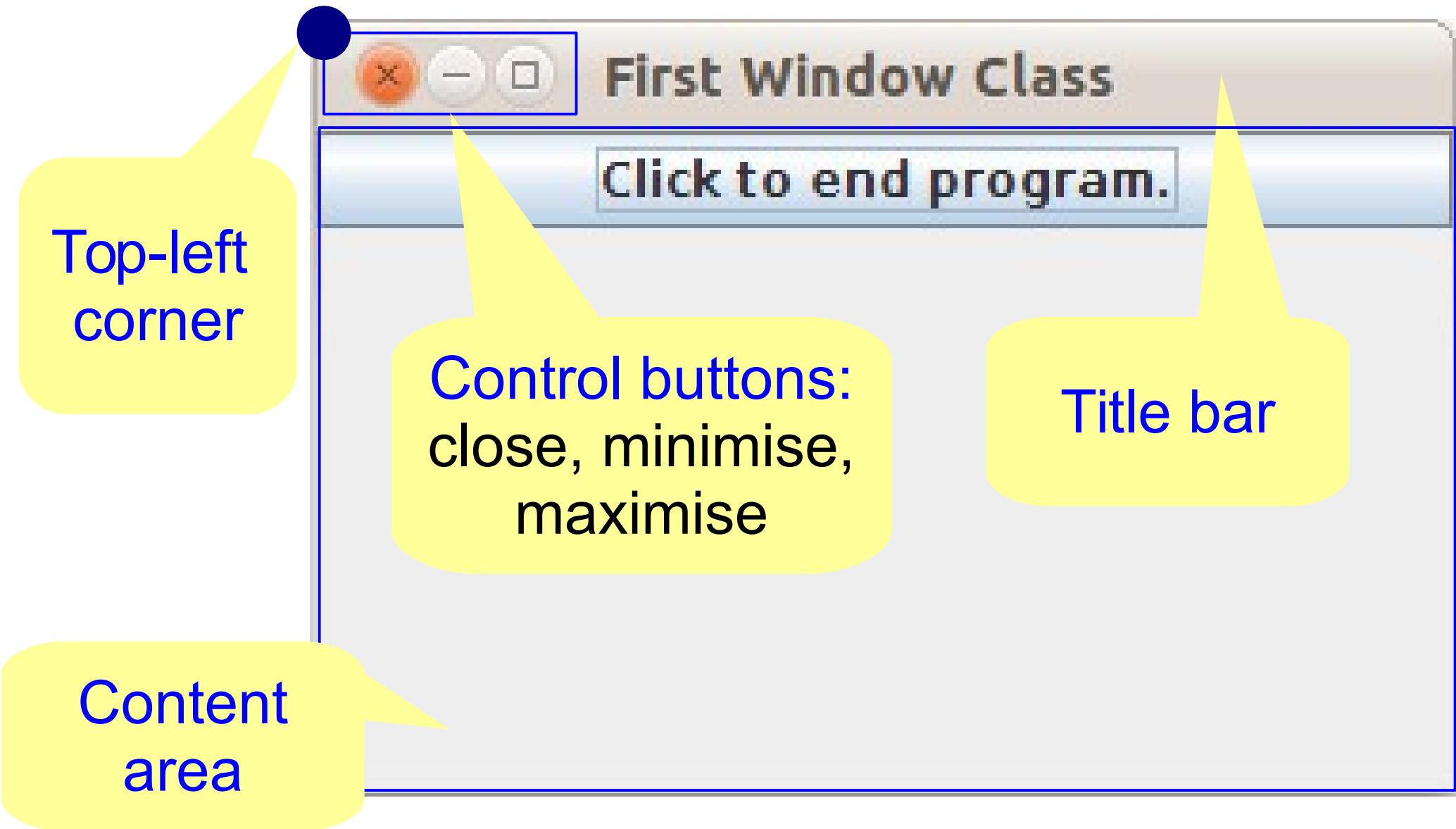
Label



Component class hierarchy



Window basic features



Window class

JFrame
+ ^s EXIT_ON_CLOSE
+ JFrame() + JFrame(String t) + setDefaultCloseOperation(int opt) + setLayout(LayoutManager lm) + setJMenuBar(JMenuBar bar)

Inherited methods

`java.awt.Component`

- + `setSize(int, int)`
- + `getSize()`
- + `setBackground(Color)`
- + `setForeground(Color)`
- + `setLocation(int, int)`
- + `setVisible(boolean)`
- + `isVisible(): boolean`

`java.awt.Window`

- + `dispose()`

`java.awt.Frame`

- + `setTitle(String t)`
- + `getTitle(): String`

`java.awt.Container`

- + `add(Component)`

Inherited fields

<i>WindowConstants</i>	
+ ^s	DISPOSE_ON_CLOSE
+ ^s	DO_NOTHING_ON_CLOSE
+ ^s	HIDE_ON_CLOSE

Create a window object

- **Directly** using the JFrame class:

```
JFrame w = new JFrame();
```

```
JFrame w1 = new JFrame("First Window Class");
```

- **Indirectly** via a sub-class of JFrame:

```
public class FirstWindow extends JFrame  
{...}
```

```
JFrame w = new FirstWindow();
```

Set up the window

- Set up window properties:
 - title
 - size: width and height
 - location: (x,y) coordinates of top-left corner
 - colour: fore/background
- Use the corresponding mutators:

```
w.setTitle("First Window Class");  
w.setSize(300,200);  
w.setLocation(100,100);  
w.setBackground(Color.YELLOW);
```

Set up colour

- A colour is an object of `java.awt.Color`
- Colour components (red, green, blue) can be specified as input
- `Color` has pre-defined colours, defined as public constants, e.g:

BLACK	RED	ORANGE	...
BLUE	PINK	GREEN	
CYAN	YELLOW	GRAY	

Handle window events (1)

- API package: `java.awt.event`
- Window events are objects of `WindowEvent`
- Basic event listener is `WindowListener` interface:
 - defines methods for each window event (opened, closing, closed, activated, etc.)
 - methods take a `WindowEvent` object as argument
- Helper class `WindowAdapter` implements `WindowListener` to ease coding

Handle window events (2)

- Three ways to implement event handlers:
 - implement `WindowListener`
 - must declare all methods
 - extend `WindowAdapter`
 - only declare methods that need to be overridden
 - only applicable if class is *not* in multiple inheritance
 - invoke a helper method:
 - for closing event only
 - no code, just invoke

Example: using listener interface

```
public class MyWindowListener implements
                                   WindowListener {
    public void windowOpened(WindowEvent e) {}
    public void windowClosing(WindowEvent e) {
        // handling code
        System.exit(0);
    }
    public void windowClosed(WindowEvent e) {}
    public void windowIconified(WindowEvent e) {}
    public void windowDeiconified(WindowEvent e) {}
    public void windowActivated(WindowEvent e) {}
    public void windowDeactivated(WindowEvent e) {}
}
```

Example: using adapter

```
public class MyWindowListener extends  
WindowAdapter {  
    public void windowClosing(WindowEvent e) {  
        // handling code  
        System.exit(0);  
    }  
}
```

Example: using helper method

```
JFrame w = new JFrame("my window");  
w.setCloseDefaultOperation(  
    JFrame.EXIT_ON_CLOSE)  
// w.setCloseDefaultOperation(  
//     JFrame.DISPOSE_ON_CLOSE)  
// w.setCloseDefaultOperation(  
//     JFrame.HIDE_ON_CLOSE)  
// w.setCloseDefaultOperation(  
//     JFrame.DO_NOTHING_ON_CLOSE)
```

Create a component

- Import the component class
- Create an object using a suitable constructor
- Example: button & label

```
JButton endButton = new JButton("Click to  
end program");
```

```
JLabel aLabel = new JLabel("Close-window  
button works");
```

Setting up a component

- Set up the component's properties:
 - size
 - text: label or default text (for text fields)
 - icon (for button-type components)
 - fore/background colours
 - etc.

- Use the corresponding setters:

```
endButton.setText("Click to end program");  
aLabel.setText("Hello world");
```

Handle component events

- Event types are based on the component hierarchy:
 - (super) component: mouse, key, focus, etc.
 - container: component addition/removal
 - component-specific: **action**, change, item
- Action events are objects of `ActionEvent`
- Action event listener: `ActionListener`
 - event handling methods:
`actionPerformed(ActionEvent e)`

Handle button's action event (1)

- Corresponds to the button's pressed action:
 - caused by a mouse click or pressing Enter key
- `ActionEvent`
 - source: button object
 - action command: the button's action name (can be different from the text)

Handle button's action event (2)

- Basic steps:
 - Create an event handler from `ActionListener`
 - using normal or anonymous class
 - Create an event handler object
 - Register the handler to listen for the button's action event

Create an event handler (1)

- Using a normal class:

```
public class EndingListener
    implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
}
...
ActionListener l = new EndingListener();
```

Create an event handler (2)

- Using an *anonymous* class:

```
ActionListener l = new ActionListener() {  
    public void actionPerformed(ActionEvent e) {  
        System.exit(0);  
    }  
};
```

Register the event handler

- Invoke the `addActionListener` method on the button
- More than one listeners of the same event can be registered to the same component
- All listeners of the event are notified when it occurs
- Example:

```
endButton.addActionListener(l);
```

Add component to window

- Non-menu components are added using method `Container.add`:
 - (optional) an index position can be specified
- Example: add a button to a window

// to a default position

```
w.add(endButton);
```

// to a specified index position

```
w.add(endButton, BorderLayout.NORTH);
```

Display the window

- Windows are not displayed on the screen by default
- To display/hide, invoke method `setVisible` on the window object:
 - to display: invoke with argument `true`
 - to hide: invoke with the argument `false`

Basic GUI applications

- Basic
 - `gui.ch17.DemoWindow`
 - `gui.ch17.DemoColoredWindow`
- More about window events:
 - `gui.ch17.WindowEventDemo`

Summary

- GUI application consists of controller, view and model components
- User actions are handled via an interaction model
- Event driven programming for GUI applications
- Java provides AWT and Swing API for GUI programming
- A basic GUI application contains window, button, and label

References

Savitch W., Absolute Java, 4th, Addison-Wesley, 2009

- Chapter 17