Object-Oriented Programming

GUI Programming
Part 2

United International College

Outline

- Event handling.
- Action listeners.
- Mouse listeners / adapters.
- Mouse motion listeners / adapters.
- Keyboard listeners / adapters.
- Action listeners with timers.
- Other listeners.

Event handling

Many different kinds of events can happen when you are running a Swing program:

- Button clicks.
- Mouse clicks.
- Mouse movements.
- Keyboard presses.
- Timer ticks.
- Etc.

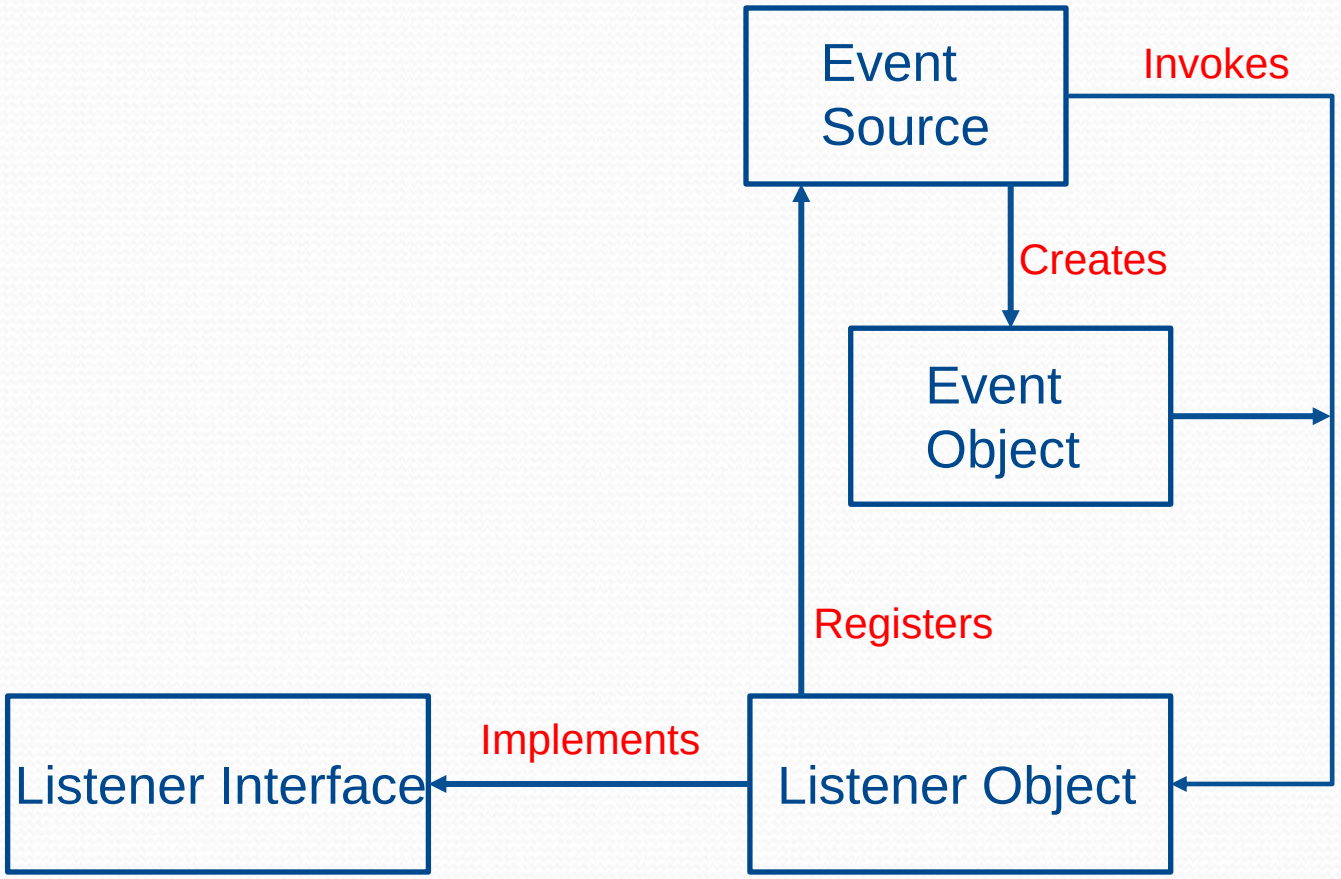
Event handling

- For every event that occurs, Swing automatically:
  - Creates an event object representing the event.
  - Calls an event handler method (if you defined one) that takes the event object as argument and processes it (handles it).
- Every specific kind of event (described by a class) has a corresponding specific kind of event handler (described by an interface).
- For example: when you click on a **`JButton`**, Swing automatically generates an object of class **`ActionEvent`**, which is automatically given by Swing to an event handler object (if you defined one) from a class that implements the **`ActionListener`** interface.

Swing Event Delegation Model

- Event Source
  - An object that generates an event
  - A source must register listeners for the listeners to receive notifications about a specific type of events
- Event Listener
  - An object that is notified when an event occurs
  - Must be registered with a source
  - It must implement specific interface methods to receive and process the notifications
- Event
  - An object that describes the change in the state of the source
  - Generated as a result of user interaction with GUI components.

Swing Event Delegation Model



Event handling

The only thing your code needs to do is:

1. Create a class that implements the correct interface for the type of events that you want to handle.
   - For example: create a class that implements the `ActionListener` interface to handle button click events.
2. Create an object from that class.
3. Tell Swing to use your object as an event handler for some specific kind of event.
   - For example: use the `addActionListener` method of a button to tell Swing that your object will handle the click events of that button.

The event handler (your object) then becomes a listener for this kind of event.

Event Classes and Listener Interfaces

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener
KeyEvent	KeyListener
ItemEvent	ItemListener
TextEvent	TextListener
FoucusEvent	FocusListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener

All event classes and listener interfaces are in **`java.awt.event`** package.

Listeners Supported by Swing Component

- You can tell what kinds of events a component can fire by looking at the kinds of event listeners you can register on it.
- JButton class has the following three listener registration methods:
  - addActionListener
  - addChangeListener
  - addItemListener

  Thus, JButton supports the aforementioned three listeners in addition to the listener methods inherits from JComponent.
- Refer to https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html#all for the detailed information.

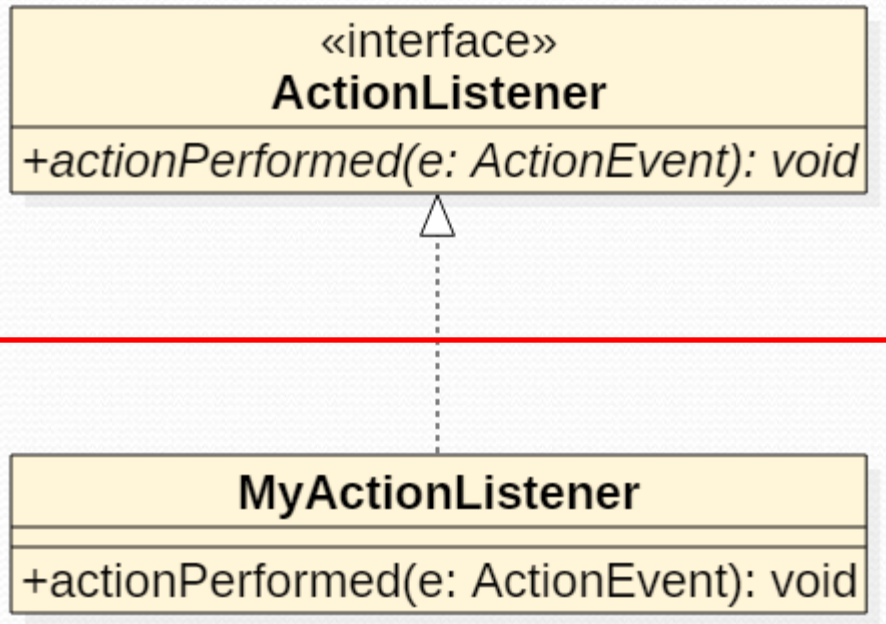
Action Listeners

Action listeners are used to process buttons clicks, menu item selection, and text field input (when the user presses the Enter key on the keyboard).

1. Create a class that implements the correct interface for the type of events that you want to handle:

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class MyActionListener implements ActionListener {
// This actionPerformed method is called every time the user
// clicks on the button;e is the event object created by Swing.
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println ("Button clicked!"+e.getActionCommand());
    }
}
```

Action listeners



Swing

You

Action listeners

2. Create an object from that class.
3. Tell Swing to use your object as an event handler for some specific kind of event.

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout()); // Layout manager.
        JButton b = new JButton("hello"); // Button.
        this.add(b);
        // Create a listener and use it to handle button clicks.
        b.addActionListener(new MyActionListener());
        this.setVisible(true);
    }
}
```

Action listeners

The code must run on the event dispatch thread (as usual):

```
public class Test {
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable()
{
            @Override
            public void run() {
                new MyFrame();
            }
        });
    }
}
```

Then, every time the user clicks on the button, Swing automatically creates an **ActionEvent** object, which is automatically given as argument to the **actionPerformed** method of the button's listener, which then prints a message on the screen.

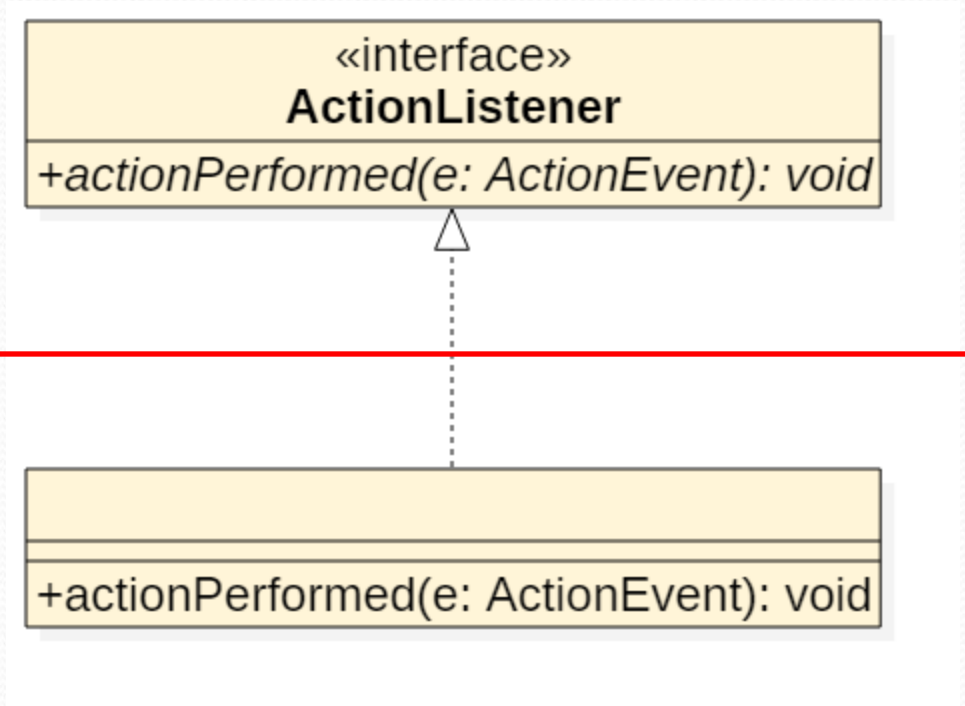
Action listeners

- Note: we use an anonymous class that implements the **`Runnable`** interface in the **`Test`** class.
- Similarly, we can also use an anonymous class that implements the **`ActionListener`** interface in the **`MyFrame`** class!
- Then we do not need to have a separate **`MyActionListener`** class anymore.
- That way, all the button-related code is in the same place.
- (The **`Test`** class remains the same.)

Action listeners

```
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setLayout(new FlowLayout()); // Layout manager.
        JButton b = new JButton("hello"); // Button.
        this.add(b);
        b.addActionListener(new ActionListener() { // Anonymous class{}
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("Button clicked!"+e.getActionCommand());
            }
        });
        this.setVisible(true);
    }
}
```

Action listeners



Swing

You

Mouse listeners

Mouse listeners are used to process mouse clicks , mouse presses, mouse releases and in-out mouse movements, in frames or in panels.

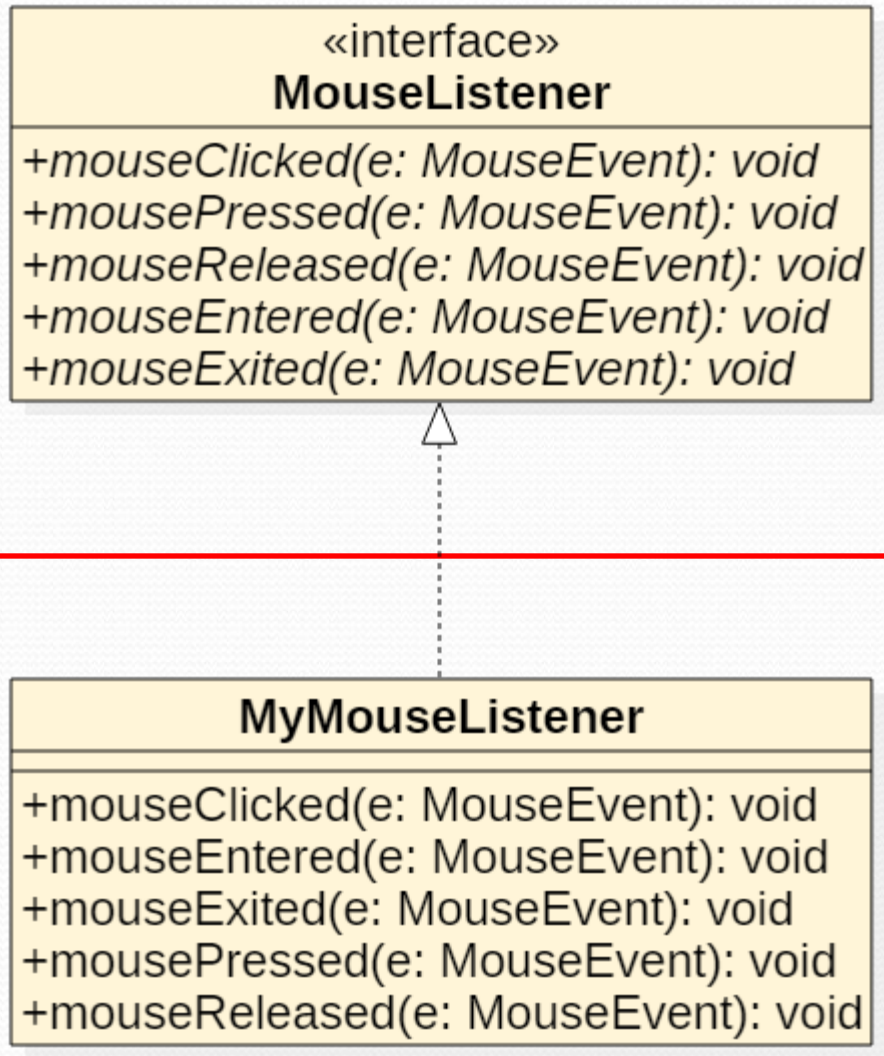
1. Create a class that implements the correct interface for the type of events that you want to handle:

```
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
public class MyMouseListener implements
MouseListener {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse clicked");
    }
    @Override
    public void mouseEntered(MouseEvent e) {
        System.out.println("Mouse entered");
    }
    ...
```

Mouse listeners

```
    ...
    @Override
    public void mouseExited(MouseEvent e) {
        System.out.println("Mouse exited");
    }
    @Override
    public void mousePressed(MouseEvent e) {
        System.out.println("Mouse pressed");
    }
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("Mouse released");
    }
}
```

Mouse listeners



Mouse listeners

2. Create an object from that class.
3. Tell Swing to use your object as an event handler for some specific kind of event.

```
import javax.swing.JFrame;
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Create a listener object and use it to handle mouse
        // events for the frame (also works for panels).
        this.addMouseListener(new MyMouseListener());
        this.setVisible(true);
    }
}
public class Test {
    ... same as before ...
}
```

Mouse adapters

- In practice it is annoying to have to implement all five methods of the `MouseListener` interface if we are not interested in all five different kinds of events.
- So for convenience Swing provides a `MouseAdapter` class which already implements the interface with empty methods.
- Then you just have to create a subclass of `MouseAdapter` and only override the methods you are interested in!
- Such an empty class that implement an interface is called an adapter.

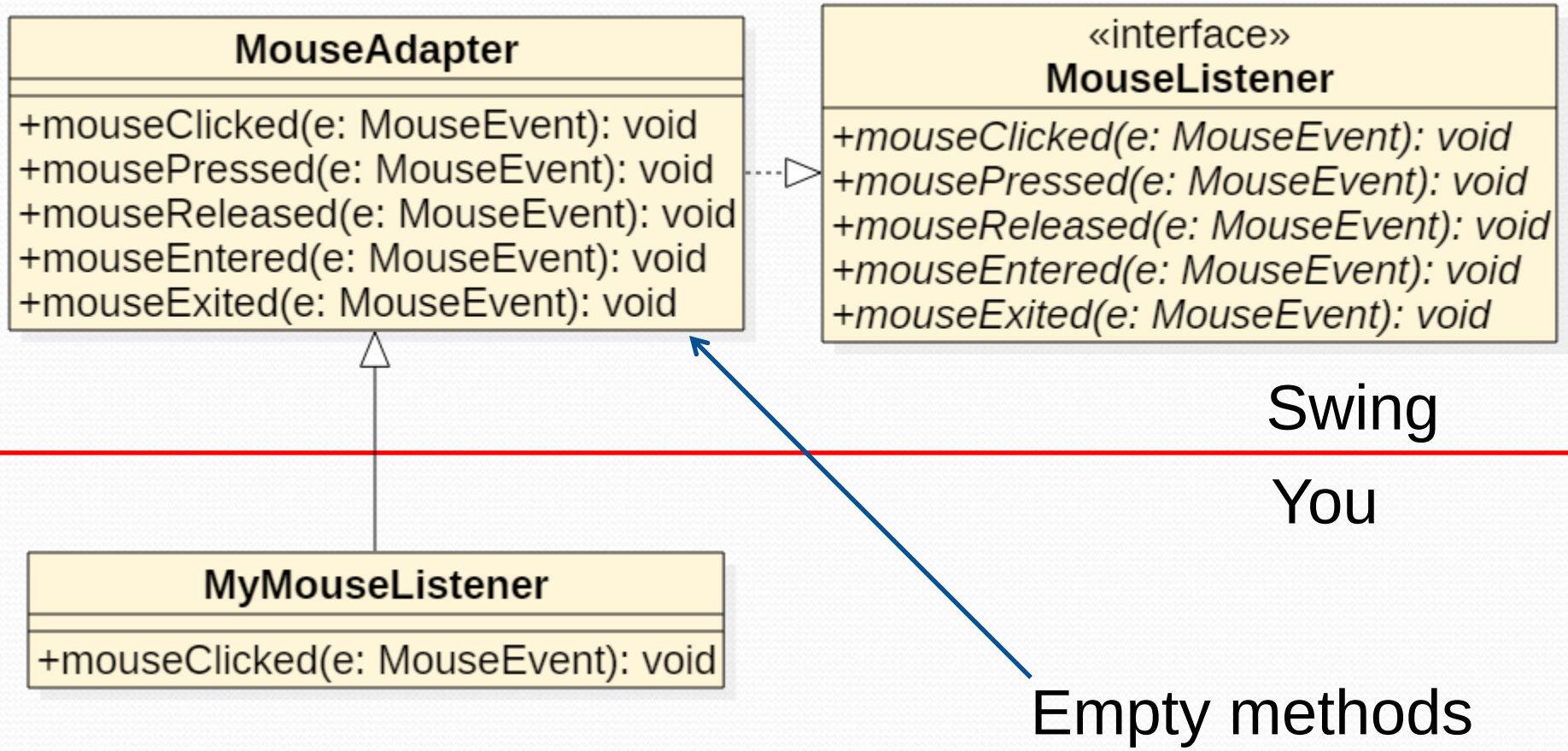
Mouse adapters

```
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
public class MyMouseListener extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse clicked");
    }
}

public class MyFrame extends JFrame {
    ... same as before ...
}

public class Test {
    ... same as before ...
}
```

Mouse adapters

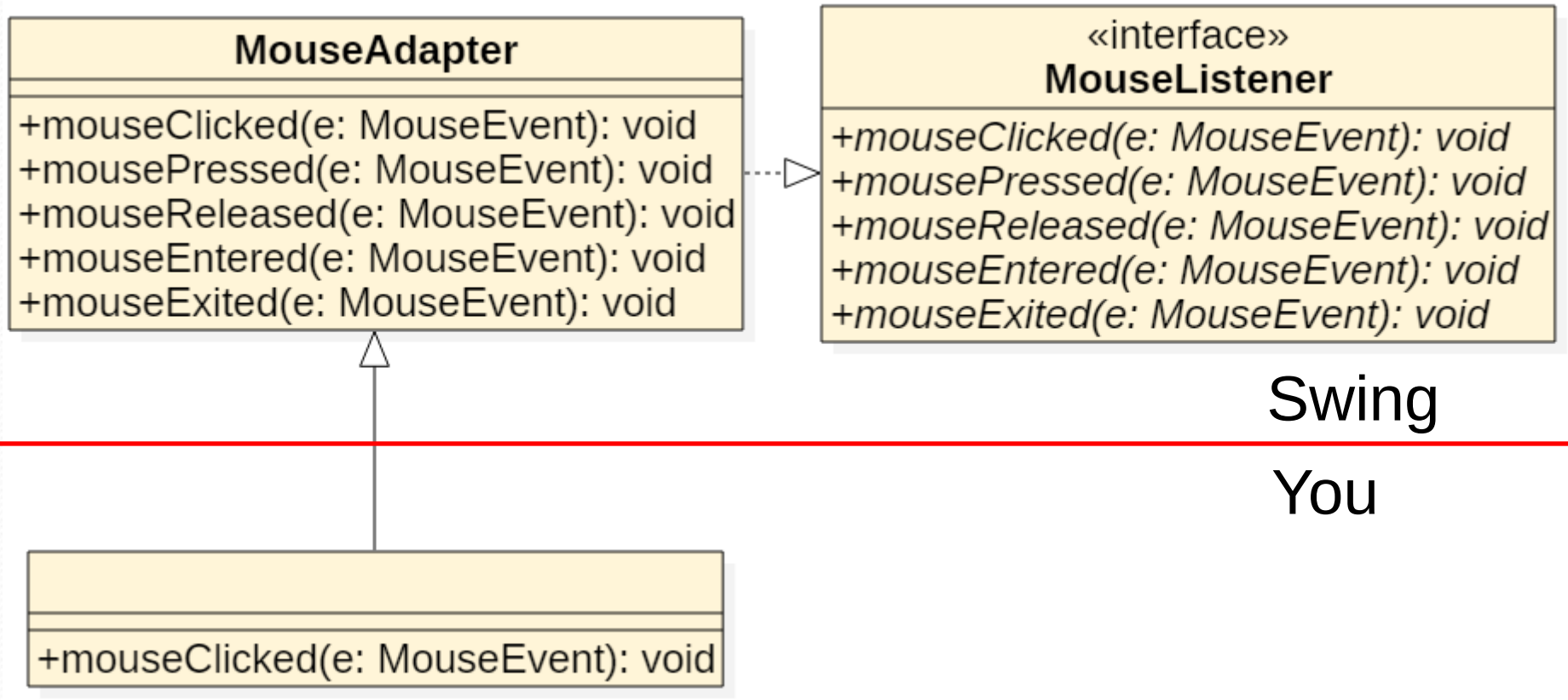


Mouse adapters

Then again we can replace the **MyMouseListener** class with an anonymous class inside **MyFrame** (the **Test** class again remains the same):

```
import ...
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Create a listener object and use it to handle mouse
        // events for the frame (also works for panels).
        this.addMouseListener(new MouseAdapter() { // Anonymous class.
            @Override
            public void mouseClicked(MouseEvent e) {
                System.out.println("Mouse clicked");
            }
        });
        this.setVisible(true);
    }
}
```

Mouse adapters



Mouse motion listeners

Mouse motion listeners are used to process events when moving or dragging (moving with the mouse button pressed) the mouse, in frames or in panels.

As usual, you can either:

1. Create a separate class that implements the **`MouseMotionListener`** interface.
2. Or create a separate class the extends the **`MouseMotionAdapter`** class.
3. Or create an anonymous class that implements the **`MouseMotionListener`** interface.
4. Or create an anonymous class the extends the **`MouseMotionAdapter`** class.

Mouse motion listeners

```
import ...
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Create a listener object and use it to handle mouse
        // move / drag events for the frame (also works for panels).
        this.addMouseMotionListener(new MouseMotionListener() {
            @Override
          public void mouseMoved(MouseEvent e) {
            System.out.println("mouse move: "+e.getX()+" "+e.getY());
            }
            @Override
          public void mouseDragged(MouseEvent e) {
            System.out.println("mouse drag: "+e.getX()+" "+e.getY());
            }
        });
        this.setVisible(true);
    }
}
```

Mouse motion listeners

```
import ...
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Create a listener object and use it to handle mouse
        // move events for the frame (also works for panels).
        this.addMouseMotionListener(new MouseMotionAdapter() {
            @Override
          public void mouseMoved(MouseEvent e) {
            System.out.println("mouse move: "+e.getX()+" "+e.getY());
            }
        });
        this.setVisible(true);
    }
}
```

Keyboard listeners

Keyboard listeners are used to process events when using the keyboard, in frames or in panels.

- Note that in Swing most text input is usually done using **`JTextField`** components, not by reading keyboard events one by one using a listener.
- So keyboard listeners are only useful for processing very simple keyboard events.
  - Example: pausing a game when the user presses the letter **`p`** in the keyboard.
- As usual, Swing provides both a **`KeyListener`** interface and a **`KeyAdapter`** class, and you can use a separate class or an anonymous one.

Keyboard listeners

```
import ...
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addKeyListener(new KeyListener() {
            @Override
            public void keyPressed(KeyEvent e) {
                System.out.println("key pressed: " + e.getKeyChar());
            }
            @Override
            public void keyReleased(KeyEvent e) {
                System.out.println("key released: " + e.getKeyChar());
            }
            @Override
            public void keyTyped(KeyEvent e) {
                System.out.println("key typed: " + e.getKeyChar());
            }
        });
        this.setVisible(true);
    }
}
```

Keyboard adapters

```
import ...
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.addKeyListener(new KeyAdapter() {
            @Override
            public void keyTyped(KeyEvent e) {
                System.out.println("key typed: " + e.getKeyChar());
            }
        });
        this.setVisible(true);
    }
}
```

Action listeners with timers

In addition to being used for handling button click events, action listeners can also be used to process time events.

This is useful for creating animations or processing data at regular intervals.

- Create an object from the Swing `Timer` class (not the `java.util.Timer` class, which is different!)
- When creating the timer object, you must specify:
  - How often (in milliseconds) the timer will tick.
  - An action listener that will be automatically called by Swing every time the timer ticks.
- Then call the `start()` method of the timer to make the timer start ticking. Do not forget this!

Action listeners with timers

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.Timer;
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Timer t = new Timer(1000, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                System.out.println("tick");
            }
        });
        // Do not forget to start the timer!
        t.start();
        this.setVisible(true);
    }
}
```

Other listeners

- There are many more kinds of listener interfaces available in Swing:
  - `ComponentListener`
  - `FocusListener`
  - `MouseWheelListener`
  - Etc: Listeners Supported by Swing Components (Oracle web site)
- The listener interfaces with many methods usually have a corresponding adapter class, for convenience.
- A single component (such as a panel) can have many different listeners at the same time to handle different kinds of events.

Example

- Here is an example that shows how to use a timer to create some basic animation.
  - We use a timer to update the angle of a red arc every few milliseconds.
  - We must call Swing's **`repaint`** method every time we change what we want to draw.
    - Swing then automatically calls the **`paintComponent`** method of our panel to redraw it.
    - If we do not call **`repaint`** then Swing does not know that it needs to redraw the panel and nothing changes on the screen!
- We also use a mouse click listener to change the position of the arc every time the user clicks.

Example

```
import ...
public class MyPanel extends JPanel {
    private int clockRate = 20; // Ticks per second.
    private int x = 100; // Start x position of center.
    private int y = 100; // Start y position of center.
    private int radius = 50;
    private int arc = 0;                    // Arc angle.
    private int arcpt = 360 / clockRate; // Arc angle change per tick.
    public MyPanel() {
        this.addMouseListener(new MouseAdapter() {
            // If the user clicks somewhere with the mouse:
            @Override
            public void mouseClicked(MouseEvent e) {
                // then we move the position of the center to the
                // position of the mouse click:
                x = e.getX();
                y = e.getY();
                // and immediately ask Swing to redraw the whole panel
                // (without waiting for the next clock tick) using the
                // paintComponent method below.
                repaint();
            }
        });
```

Example

```
        Timer t = new Timer(1000 / clockRate, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
            // The minus operation is because we grow the arc in
            // a clockwise direction (+ means counter-clockwise).
              arc = (arc - arcpt) % 360;
            // Ask Swing to redraw the panel after every clock tick.
                repaint();
            }
        });
        t.start(); // Do not forget to start the timer!
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        // (x - radius, y - radius) is the position of the upper-left
        // corner of the rectangle containing the arc if (x, y) is the
        // position of the center of the arc.
        // 2 * radius is the width and height of the same rectangle.
        g.fillArc(x - radius, y - radius, 2*radius, 2*radius, 90, arc);
    }
}
```

Example

```
import javax.swing.JFrame;
public class MyFrame extends JFrame {
    public MyFrame() {
        this.setTitle("MyFrame Title");
        this.setSize(400, 300);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // No layout manager otherwise the panel will become invisible!
        this.add(new MyPanel());
        this.setVisible(true);
    }
}

public class Test { // Always the same.
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new MyFrame();
            }
        });
    }
}
```

Summary

- Event handling.
- Action listeners.
- Mouse listeners / adapters.
- Mouse motion listeners / adapters.
- Keyboard listeners / adapters.
- Action listeners with timers.
- Other listeners.