Java Programming 2 Lecture #14

Mary Ellen Foster *(presented by Mireilla Bikanga Ada)*
MaryEllen.Foster@glasgow.ac.uk
6 November 2019

Outline

GUI programming in Java

Introduction to Swing

Overview

Components

Events

Extended example

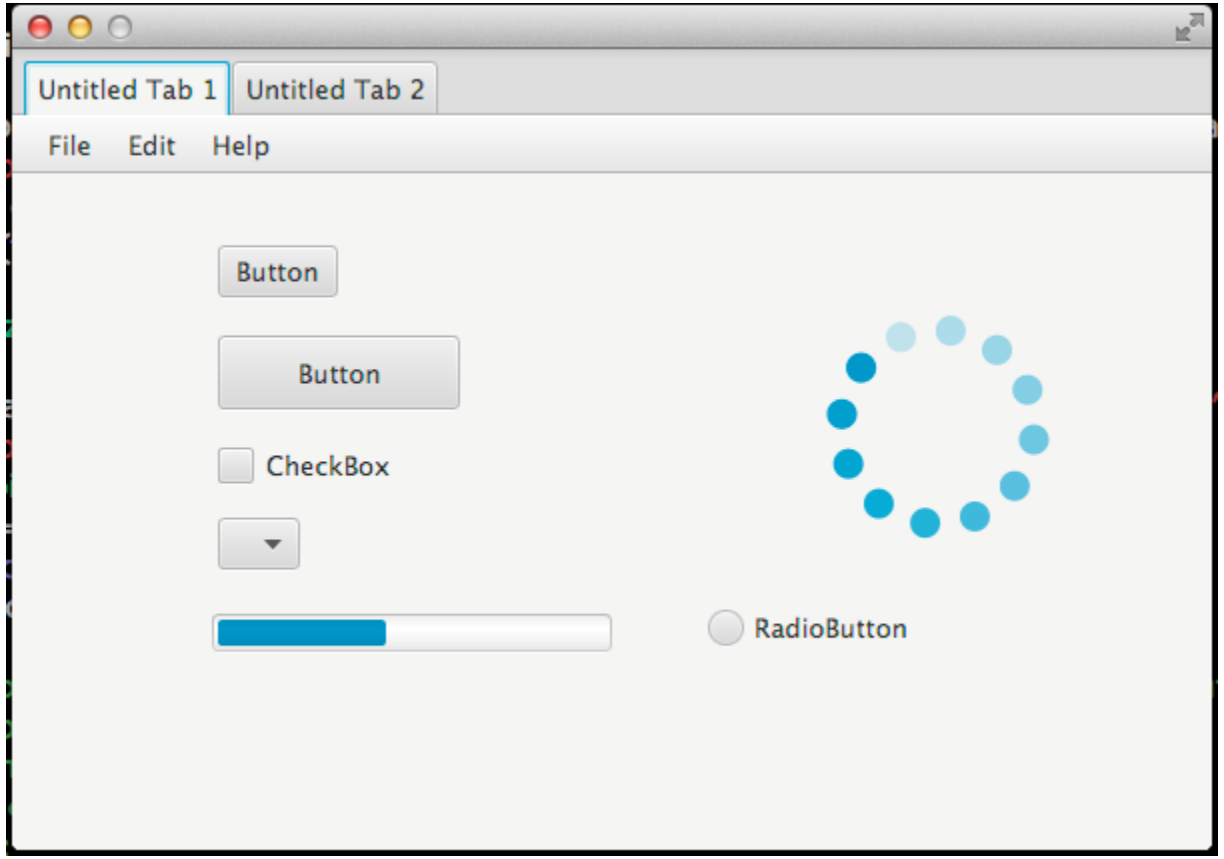


Image from https://stackoverflow.com/q/23862779

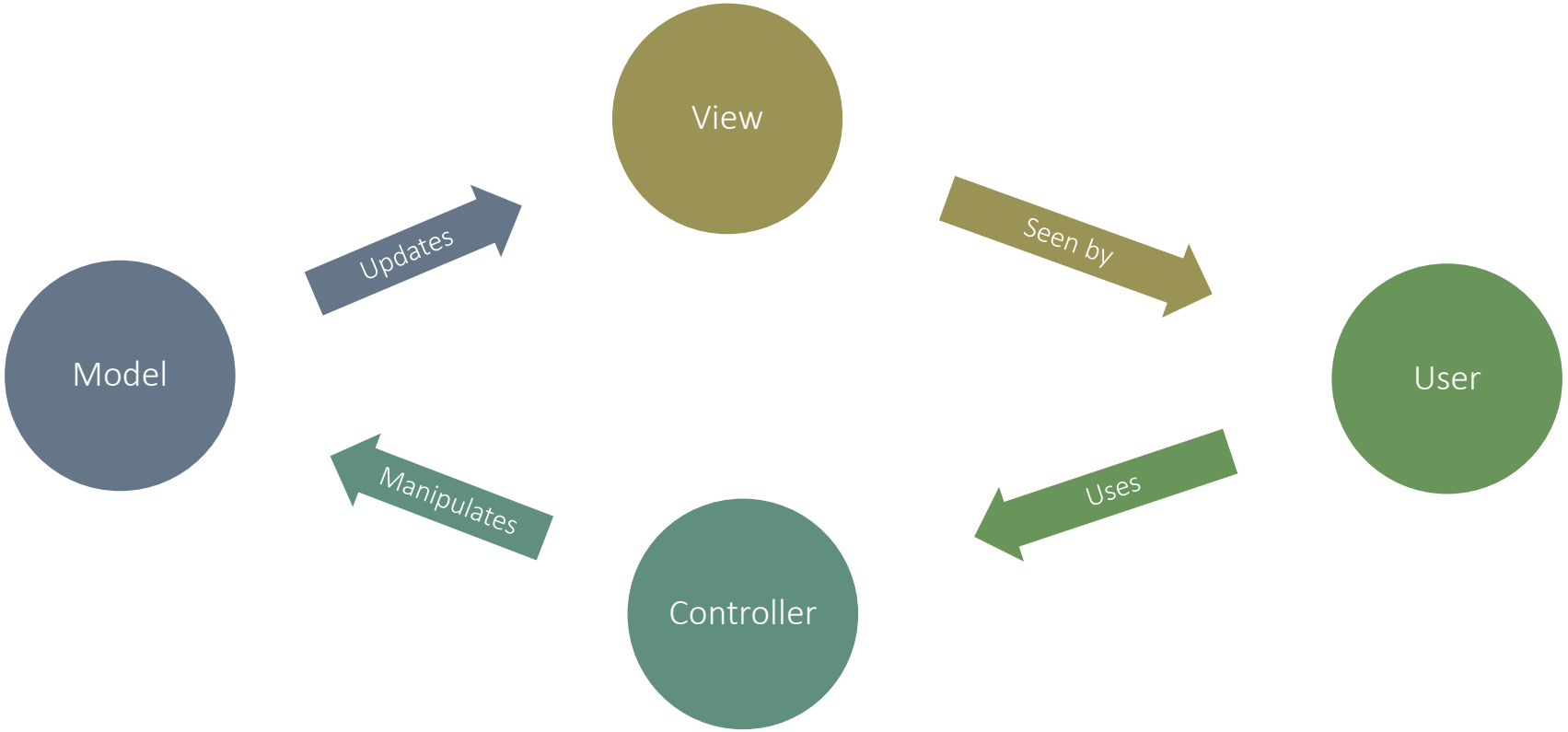
Java GUI toolkits: AWT, Swing, JavaFX

AWT ("Abstract Windowing Toolkit") – since the very beginning (January 1996)
- First Java GUI toolkit: set of "heavyweight" classes using native GUI widgets
- Still included in Java but not widely used

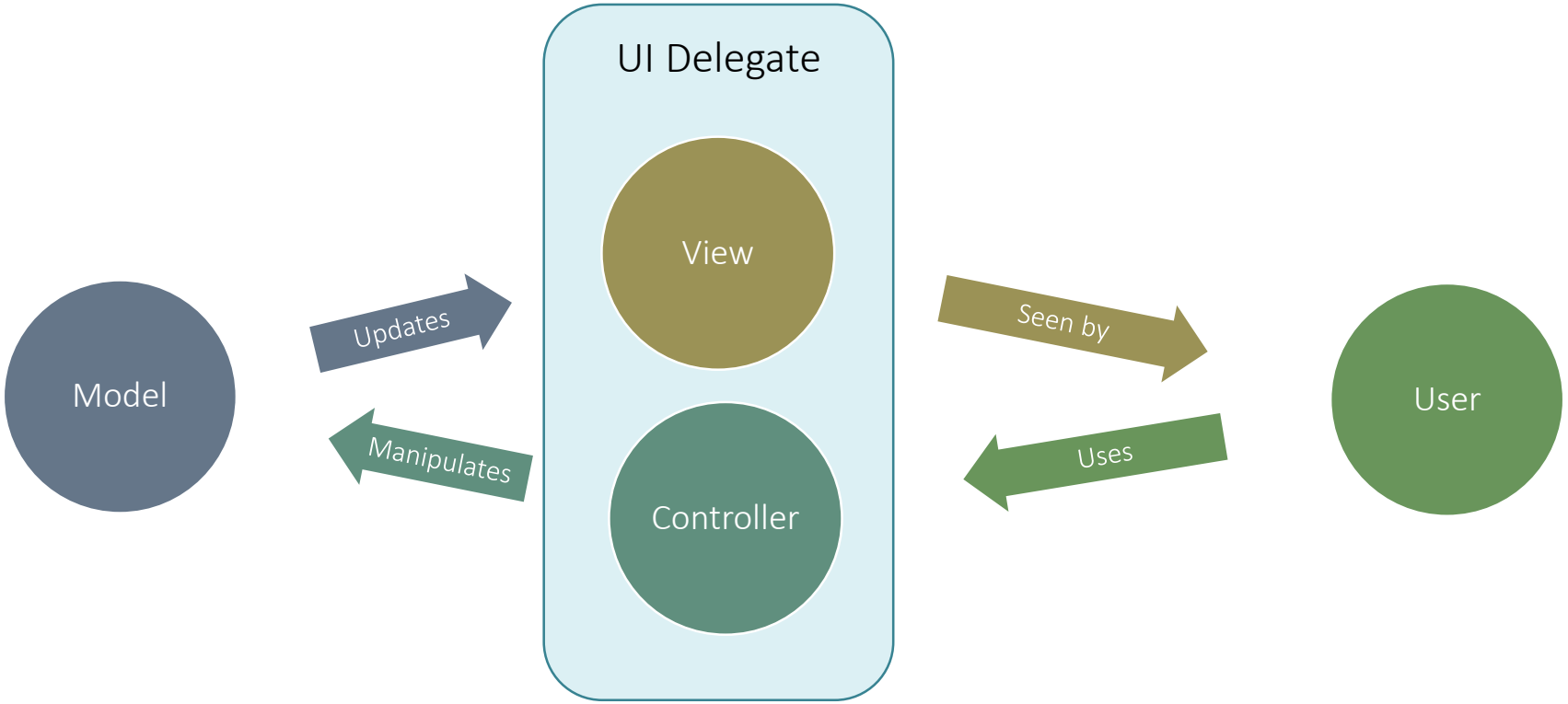
Swing – since Java 2 (December 1998)
- Cross-platform "lightweight" GUI components written entirely in Java
- More powerful and flexible components than AWT
- Still included in Java, widely used

JavaFX
- Introduced in 2007, bundled with Java as of Java 8 (March 2014)
- Un-bundled again as of Java 11; now separate open-source project at https://openjfx.io/
- Somewhat widely used – seen as more flexible than Swing, but never got traction, and now HTML5 appears to be taking over

Classic Model-View-Controller (MVC)



“Modified MVC” in Swing



More details at http://www.oracle.com/technetwork/articles/javase/index-142890.html

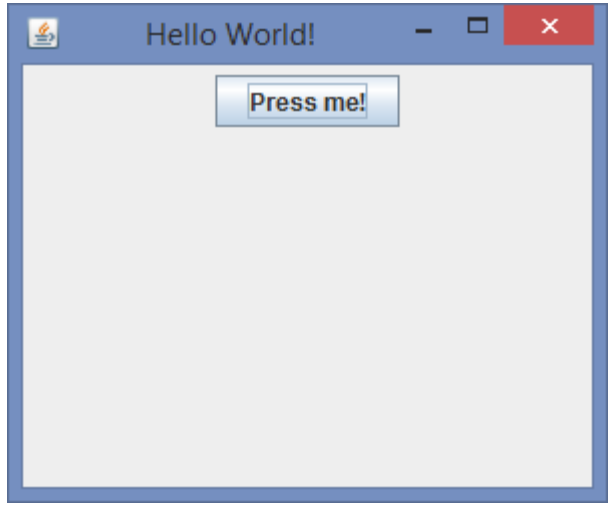
First Swing program

```
public class HelloWorld {
    private static void createAndShowGUI() {
        JFrame frame = new JFrame("Hello World!");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setLayout(new FlowLayout());
        frame.setSize(300, 250);

        JButton button = new JButton("Press me!");
        frame.getContentPane().add(button);

        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```



`javax.swing.JFrame`

Represents a top level window

Relevant operations

- `setSize(width, height)` – sets default dimensions
- `setLocationByPlatform(boolean)` – if true, lets the OS decide where to place the window
- `setVisible(boolean)` – shows/hides the window
- `setDefaultCloseOperation()` – what should the program do when the window closes
  - *EXIT_ON_CLOSE – entire program exits when window closes*

Adding components

`JFrame` is a **container** – it can hold other components inside it

To access the main container in a `JFrame`, use `getContentPane()`

(Just adding directly to the `JFrame` will often also work, but this is the official method)

Then we added a **`JButton`** – a button that can be pressed

`JButton` constructor sets the button label

Not discussed today: layout managers
https://docs.oracle.com/javase/tutorial/uiswing/layout/using.html

Other useful Swing components

JLabel: a textual label

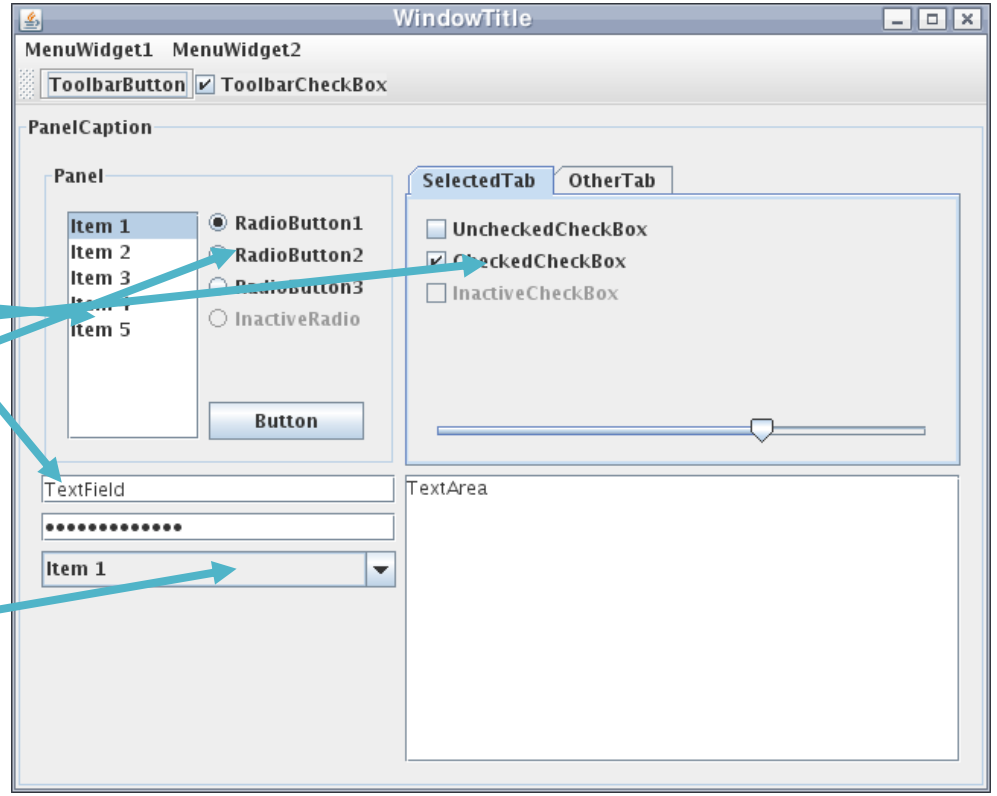
JTextField: a field for entering text

JList: a list box

JCheckbox: a box that can be checked or not

JRadioButton: one of a set of option buttons

JComboBox: a drop-down list



JList in Modified MVC

JList is the UI delegate (i.e., it is the view and the controller)

It has an associated **ListModel** which provides the model (usually you can use a **DefaultListModel**)

ListModel is almost the same as **java.util.List** – methods include **addElement()**, **remove(), insertElementAt(), getSize()**

When the ListModel is changed, the information displayed in the Jlist is changed as well

Similar link between JTable and TableModel

More on Swing Components

Parent class of all components (except top-level windows like `JFrame`, `JDialog`, `JWindow`): `javax.swing.JComponent`

All have a `setEnabled(boolean)` method

- When true: component is active and can be used
- When false: component is "greyed out"

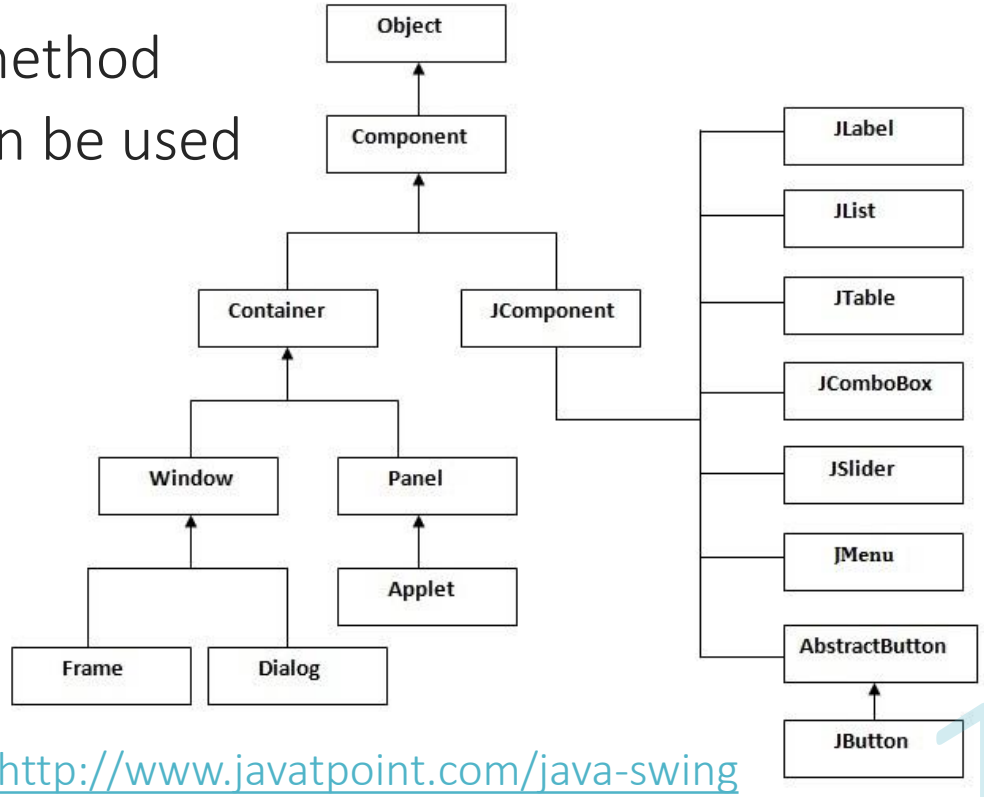


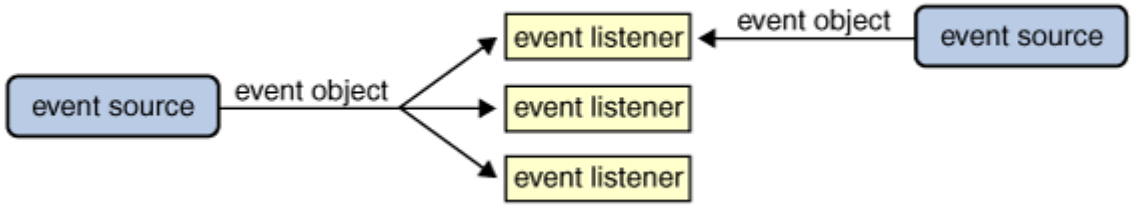
Image from http://www.javatpoint.com/java-swing

Events

Events in Swing

An event is **fired** every time something happens in the program

- JButton is pressed
- Window is shown/hidden/opened/closed
- User selects an item in a JComboBox or JList
- User types into a JTextField
- …

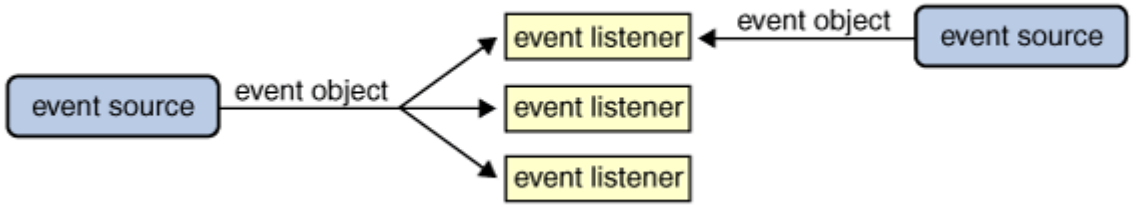


Event listeners

If you want to deal with an event, you need to

- Implement a **listener** for that type of event
- **Register** it with the event source

Then every time an event of that type is fired, your event listener will be called



Handling button press events

```
JButton button = new JButton("Press me!");
frame.getContentPane().add(button);
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Hello world");
    }
});
```

Anonymous inner class

What's going on?

Button fires an `ActionEvent` when it is pressed

You need an `ActionListener` to process that event

- One method: `actionPerformed`

We have provided an `ActionListener` and registered it with the button through `addActionListener`

So:

- Every time the button is pressed …
- Our `ActionListener.actionPerformed()` method is called!
- (Give it a try in the lab/at home)

Some useful Swing listeners

Listener	Methods
ActionListener	actionPerformed(ActionEvent)
ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) …
MouseInputListener	mouseClicked(MouseEvent) mouseEntered(MouseEvent) …
ListSelectionListener	valueChanged(ListSelectionEvent)
WindowListener	windowOpened(WindowEvent) windowClosed(WindowEvent) …

All options at https://docs.oracle.com/javase/tutorial/uiswing/events/api.html

General strategy

1. Instantiate the appropriate GUI components (buttons, combo boxes, etc)
2. Register **callbacks** for their interactive behaviour
   i.e., register for all events that you might need to catch
3. Add each component to the top-level window (e.g., JFrame)
4. Make the window visible

Example – employee manager GUI

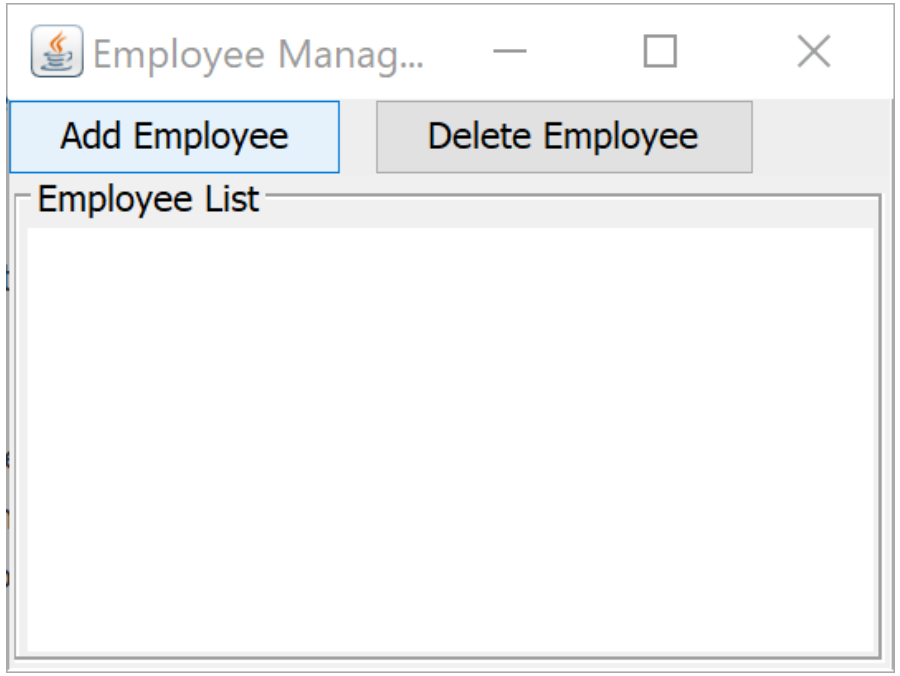
```
public abstract class Employee {
    // Common fields
    private int employeeNumber;
    private String familyName;
    private String givenName;

    // “Hourly” or “Salaried”
    protected String
employeeType;
}
```

```
public class HourlyEmployee extends Employee {
    private double hourlyRate;
    private int contractedHours;
}

public class SalariedEmployee extends Employee
{
    private double annualSalary;
}
```

EmployeeFrame




```
public class EmployeeFrame
    extends JFrame implements ActionListener {

        // The buttons to display
        private JButton addButton;
        private JButton deleteButton;

        // The underlying list of employees, and the GUI object to display them
        private DefaultListModel<Employee> listModel;
        private JList<Employee> employeeList;
```

Initialising list box to hold Employees

```
// Inside EmployeeFrame constructor …

// Initialise an empty list model, a JList to display it, and a scroll
// pane to contain the list so that it is scrollable
listModel = new DefaultListModel<>();
employeeList = new JList<>(listModel);
JScrollPane employeeScroll = new JScrollPane(employeeList);
employeeScroll.setBorder(new TitledBorder("Employee List"));
```

Adding an action listener to buttons

```
// Inside EmployeeFrame constructor …

// Initialise all buttons and add the current class as an action
// listener to all
addButton = new JButton("Add Employee");
addButton.addActionListener(this);
deleteButton = new JButton("Delete Employee");
deleteButton.addActionListener(this);
```

EmployeeFrame.actionPerformed

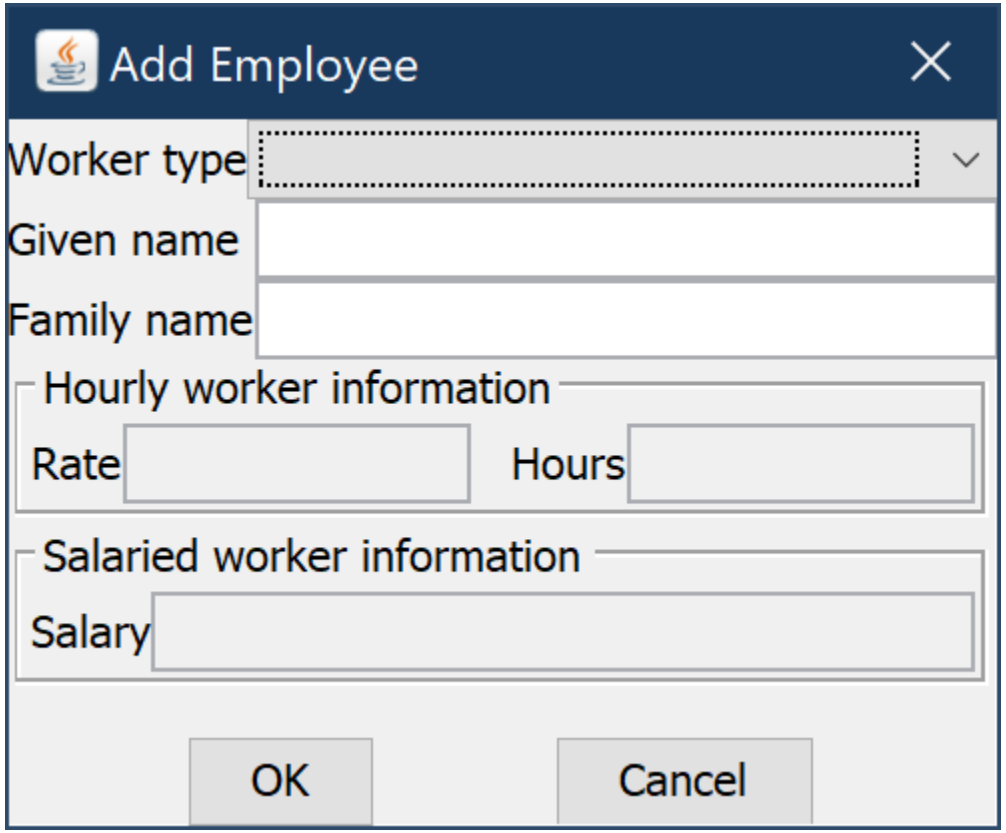
```
// Called when one of the buttons is clicked
public void actionPerformed(ActionEvent event) {
        // Determine which button was pushed
        Object source = event.getSource();
        if (source == deleteButton) {
                // If an employee is selected in the list, remove it from the model
                Employee selection = employeeList.getSelectedValue();
                if (selection != null) {
                        listModel.removeElement(selection);
                }
        } else if (source == addButton) {
                // Create and display a new dialogue to add the employee
                new AddEmployeeDialog(this).setVisible(true);
        }
}
```

AddEmployeeDialog

Pops up when **Add employee** is clicked

Allows employee details to be specified, and updates ListModel when employee is created

Code not shown here but will be included in posted code on Moodle



Coming up …

Thursday: Lab 7 distributed (equals()/hashCode(), Swing programming)

You will not need to create GUI elements – you will need to write event handlers

Friday: **NO TUTORIAL**

Next week:

Monday/Wednesday: lectures on Threads and concurrent programming