

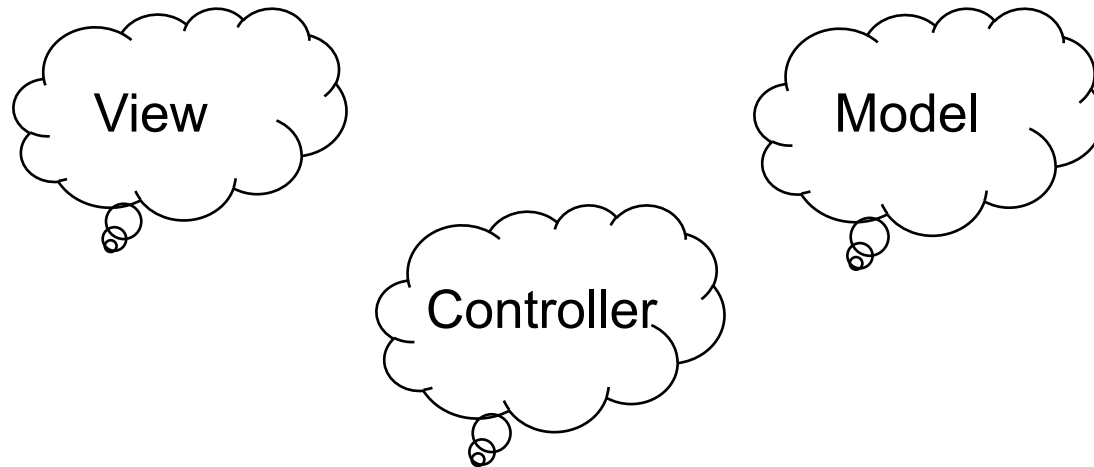
Week 3

Layout managers,
The Stopwatch artefact (part 1)

Modularity

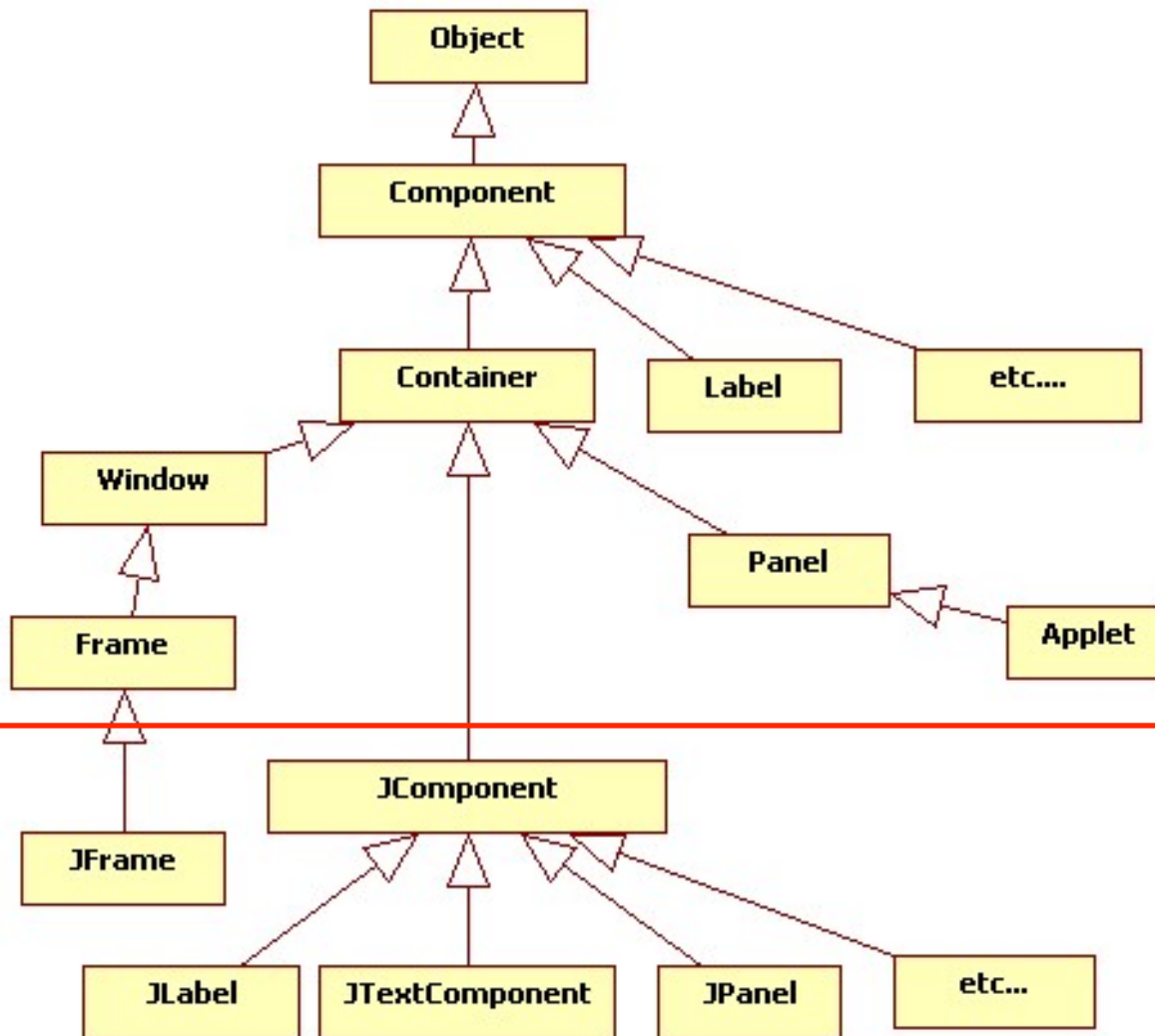
- See *Software Engineering, a Programming approach*, 4th Edition, by Douglas Bell , Chapter 6
- Complex software benefits from a good architecture and division into well defined modules each with a clear purpose – using for example object-oriented programming. This helps in:
 - **Understanding the overall structure and functioning of the system** – separation of concerns reflected in separate software components or component groups/layers (for example see next slide)
 - **Debugging** – to find place where bug occurred
 - **Testing** – easier to test individual components in isolation
 - **Maintenance** – fix bugs and introduce enhancements in one component without affecting other components
 - **Independent concurrent development within a team**
 - **Damage control** – an error in one part of the program doesn't spread to other parts
 - **Software re-use** – for example for inheritance or aggregation in OO design and programming

Model View Controller Architecture



- Artefacts which have a graphical user interface should be constructed in three distinct layers.
 - A Model layer containing only the application functionality.
 - a View layer containing only the visible part of the interface.
 - a Controller layer providing the behaviour of the artefact.
- Of these the Controller layer is likely to be the most complex.

AWT and Swing Hierarchy



AWT

Swing

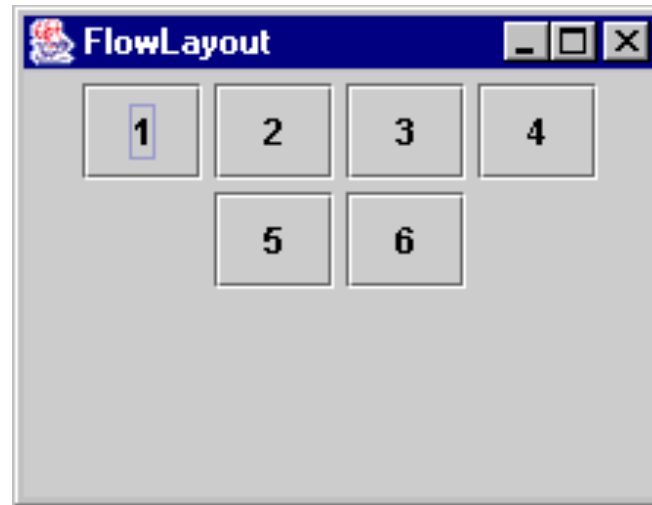
The Role of a layout manager

- A layout manager object is attached to any Swing Container.
- It acts as a “helper” to the Container
- Just before the display of the Container and the Components it graphically contains, the layout manager calculates the size of the Container and how the contents are going to be arranged.
- Each of the contents can itself be a Container with its own different layout manager

Some layout managers

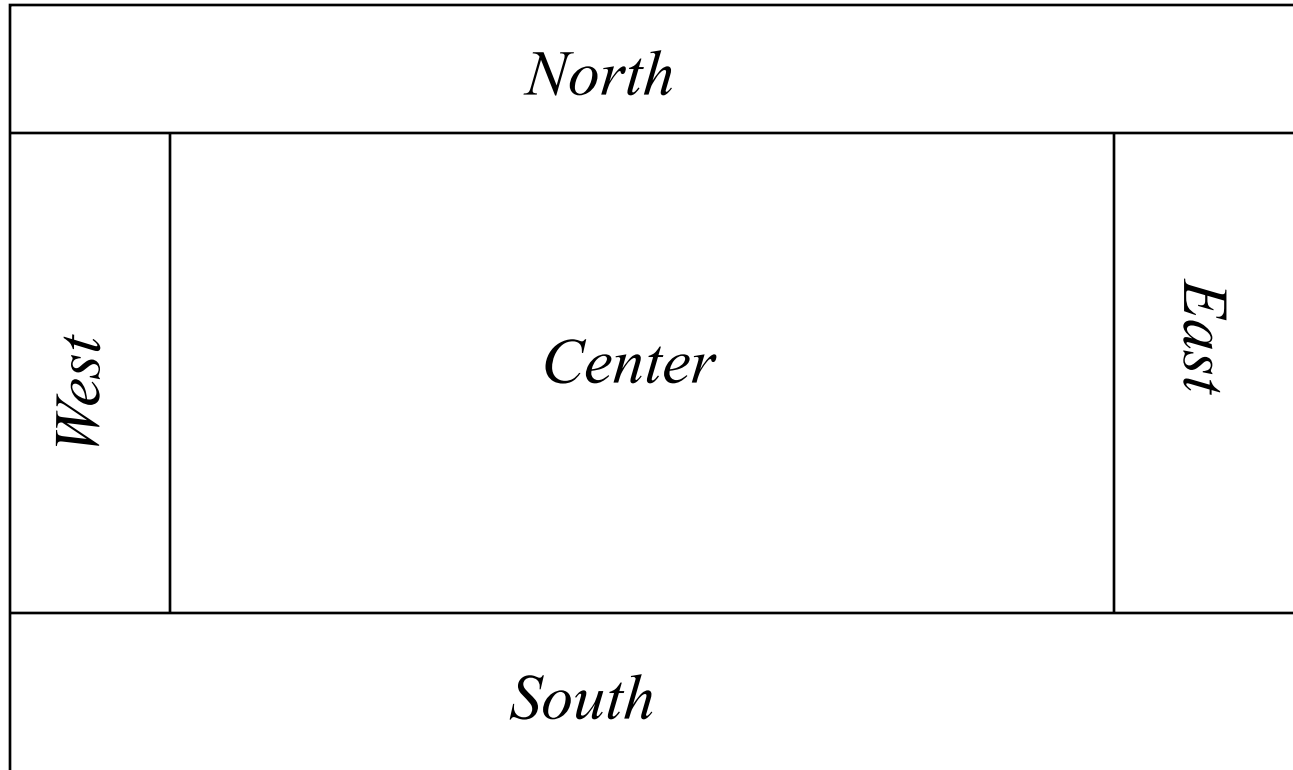
- FlowLayout
- BorderLayout
- GridLayout
- CardLayout covered in a later lecture
- GroupLayout
 - The layout manager used for auto-generation of code using drag/drop in NetBeans
- BoxLayout (not covered here)
- GridBagLayout (not covered here)

FlowLayout



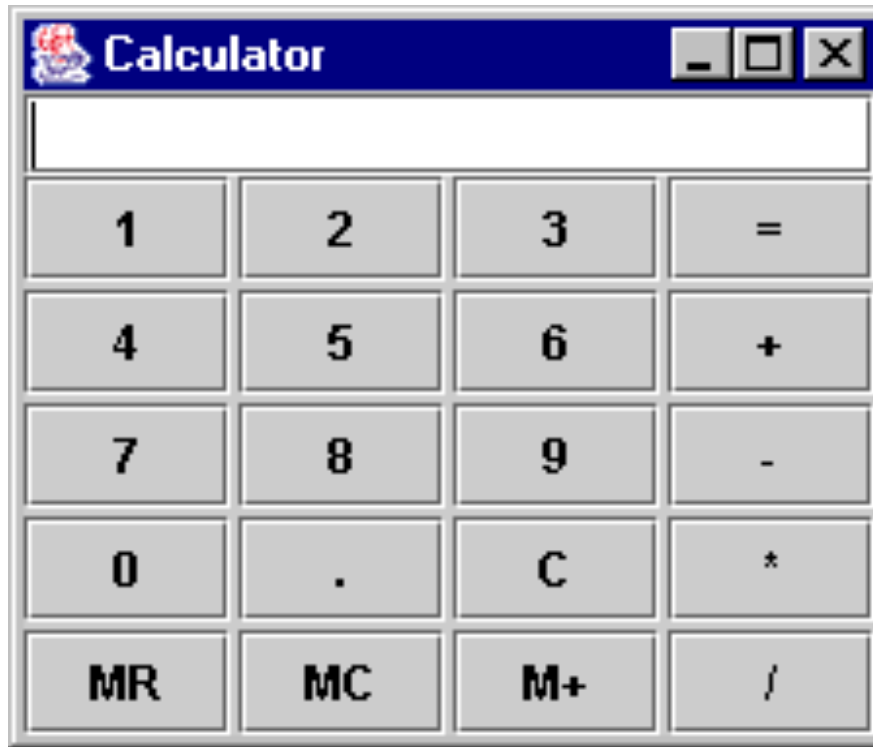
- Components are centered in Container
- FlowLayout is default manager for JPanel

BorderLayout



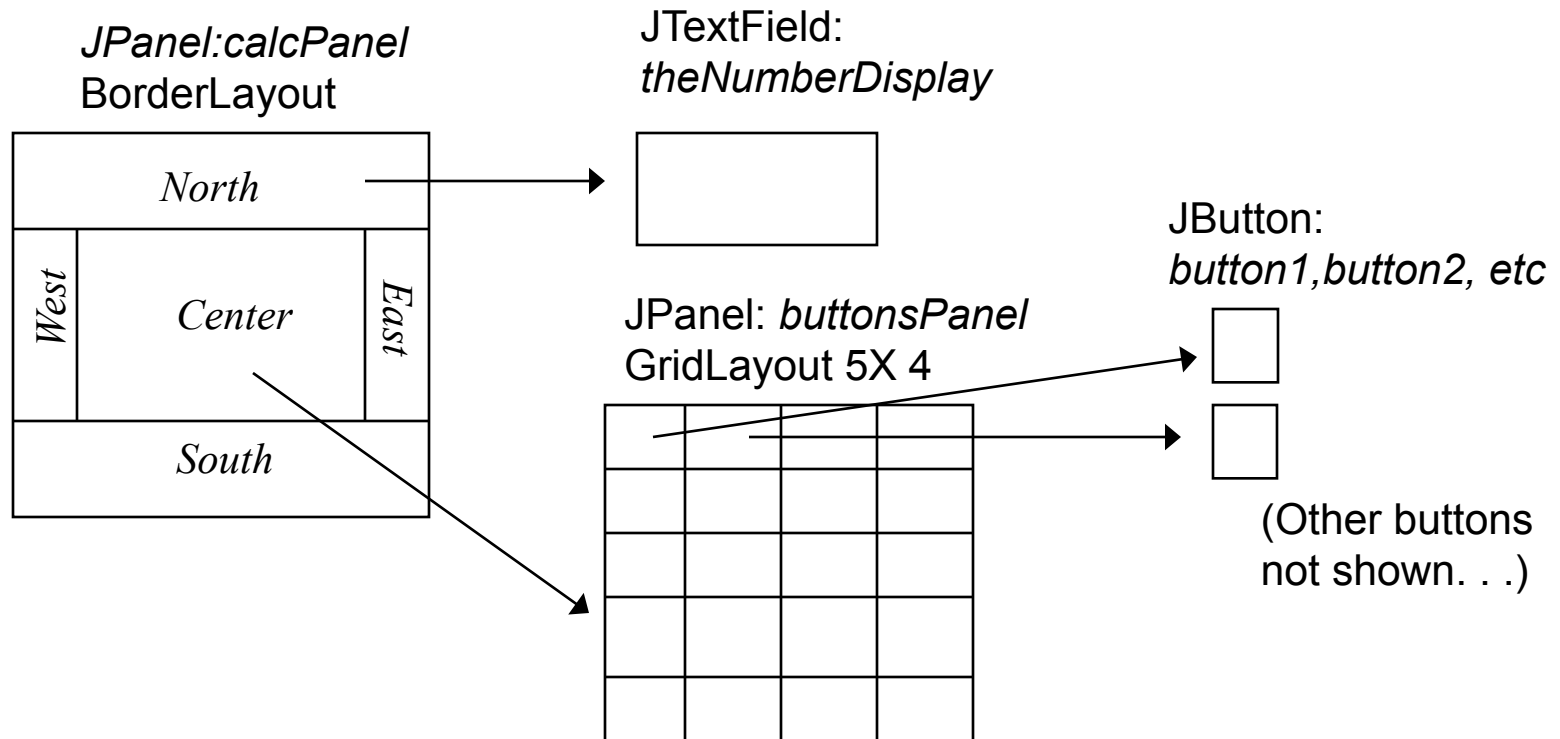
- A Container that has a BorderLayout manager can have at most 5 children with constraints as shown (e.g. the WEST and EAST components must be the same height but can be of different widths).

An example: CalculatorUI



This JPanel is laid out using GridLayout. Each compartment has the same size

CalculatorUI - layout management diagram



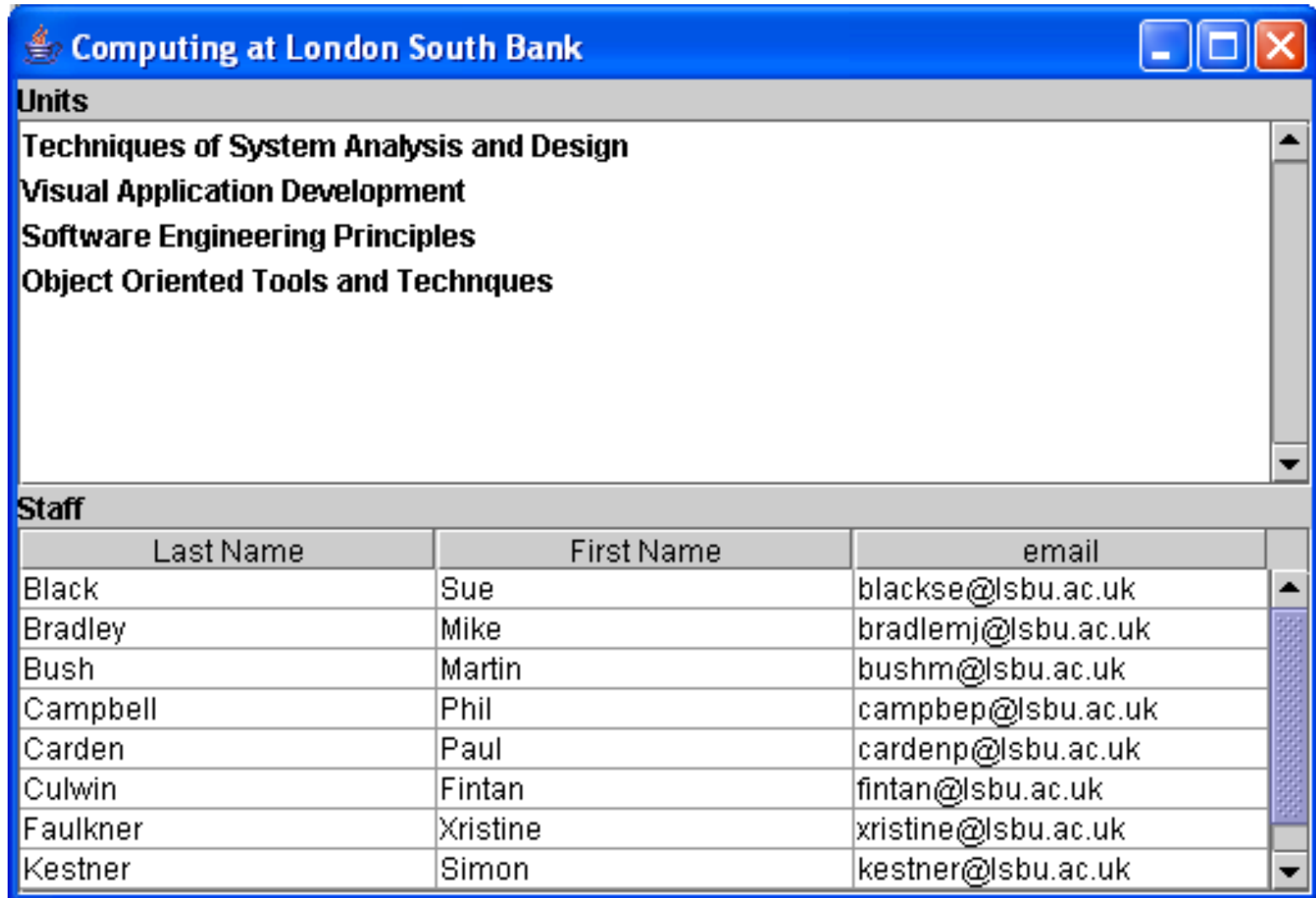
CalculatorUI - layout management (cont'd)

- The JPanel *calcPanel* requires a BorderLayout layout manager with only its NORTH and CENTER locations occupied.
- The NORTH location contains the JTextField *theNumberDisplay*
- The CENTER location contains an intermediate JPanel, *buttonsPanel*, using a GridLayout layout manager and the JButton instances are added to it.

CalculatorUI - code fragment

```
. . .  
button1 = new JButton("1");  
button2 = new JButton("2");  
  
.   
buttonEquals = new JButton("=");  
  
.   
JPanel buttonsPanel = new JPanel();  
buttonsPanel.setLayout(new GridLayout(5,4,2,2));  
buttonsPanel.add(button1);           // top row  
buttonsPanel.add(button2);  
  
.   
buttonsPanel.add(buttonEquals); // end of top row  
  
.   
etc  
  
.
```

ComputingUI



Computing at London South Bank

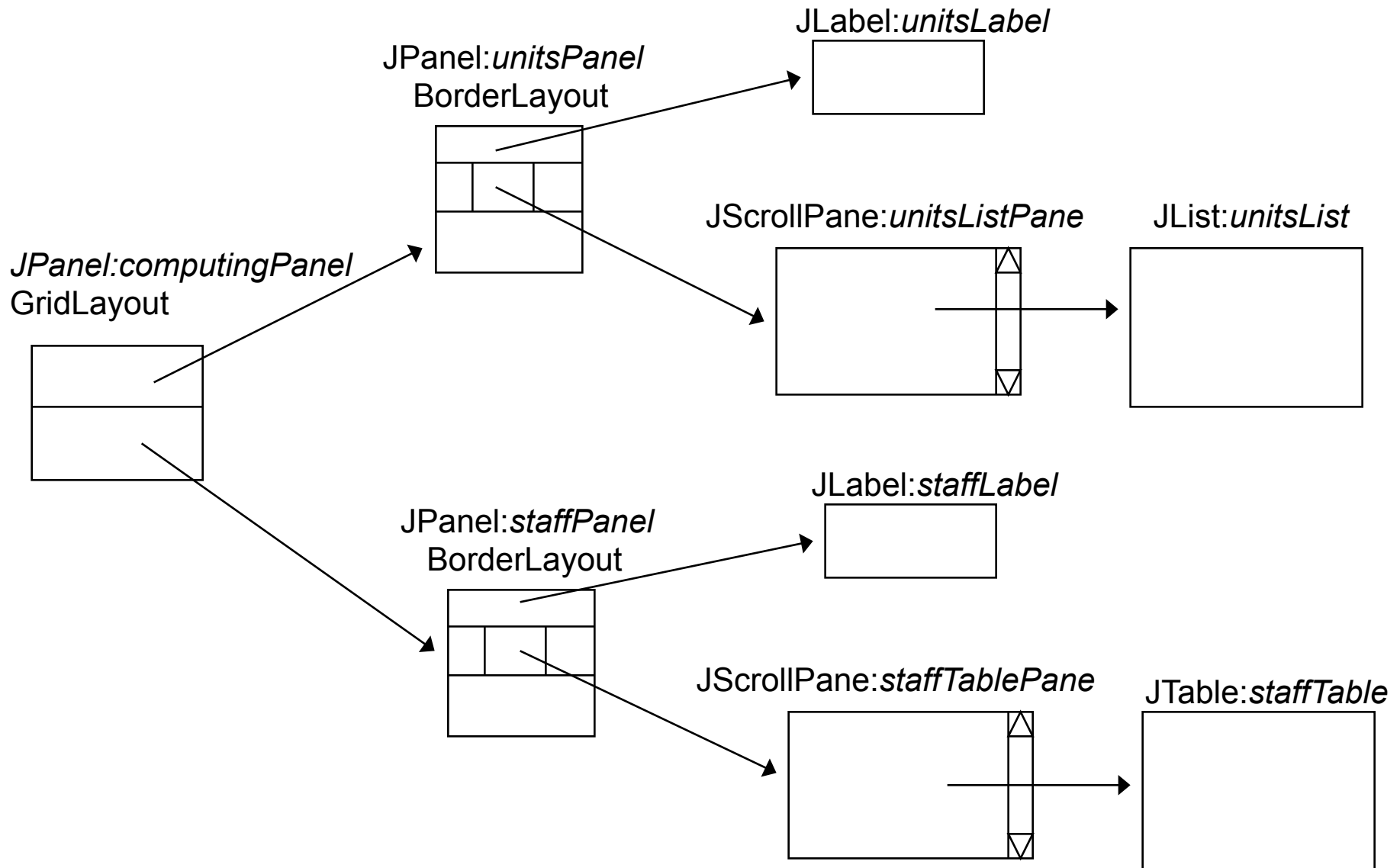
Units

- Techniques of System Analysis and Design
- Visual Application Development
- Software Engineering Principles
- Object Oriented Tools and Techniques

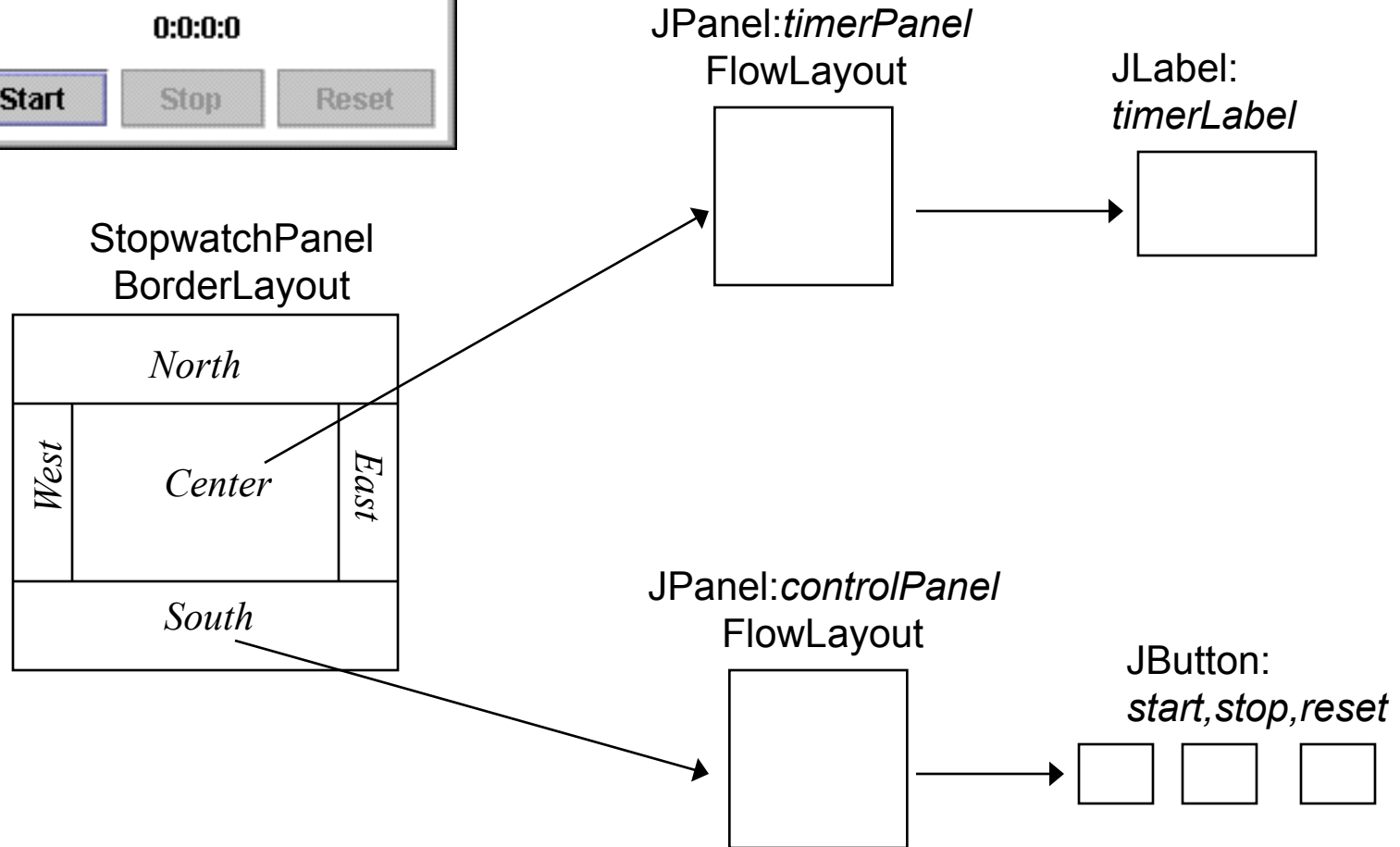
Staff

Last Name	First Name	email
Black	Sue	blackse@lsbu.ac.uk
Bradley	Mike	bradlemj@lsbu.ac.uk
Bush	Martin	bushm@lsbu.ac.uk
Campbell	Phil	campbep@lsbu.ac.uk
Carden	Paul	cardenp@lsbu.ac.uk
Culwin	Fintan	fintan@lsbu.ac.uk
Faulkner	Xristine	xristine@lsbu.ac.uk
Kestner	Simon	kestner@lsbu.ac.uk

ComputingUI Layout Management Diagram



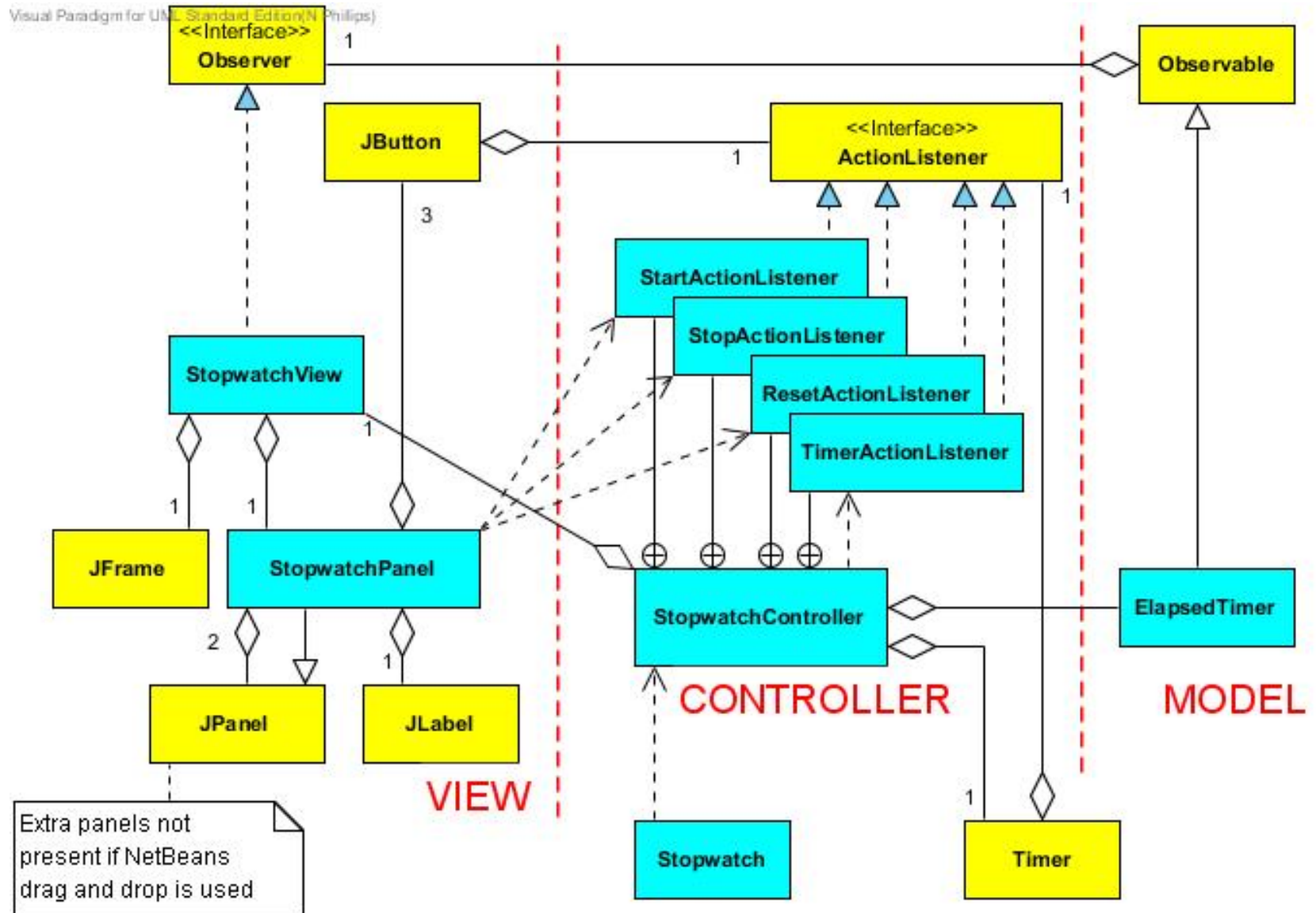
StopwatchPanel - layout management



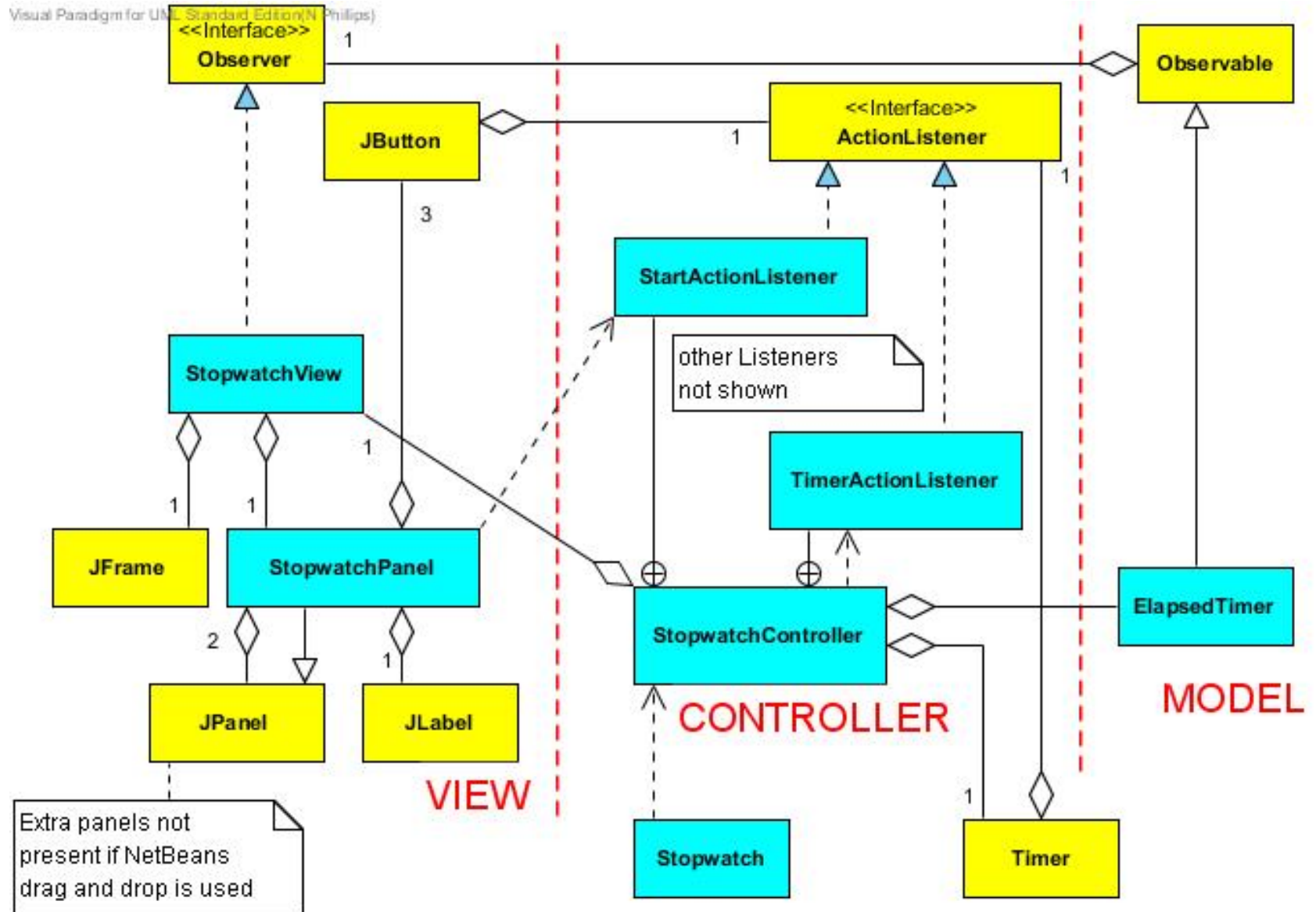
StopwatchController - layout management (cont'd)

- The `StopwatchPanel` requires a `BorderLayout` manager with only its `CENTER` and `SOUTH` locations occupied.
- The `SOUTH` location contains an intermediate `JPanel`, *controlPanel*, with a default `FlowLayout` and the three `JButton` instances added to it.
- The `CENTER` location also contains an intermediate `JPanel`, *timerPanel*, with a default `FlowLayout` and *timerLabel* mounted upon it.

The Stopwatch class diagram



The Stopwatch class diagram simplified



Stopwatch class diagram - notes

- Note that in the previous slides, the + connections in the UML diagrams indicate the containment of *inner classes*
- Note also the difference between:
 - the “has a” aggregation relationship indicating that the *StopwatchPanel* class aggregates its component classes
 - the graphical containment relationship shown in the layout management diagram indicating the top level panel graphically containing other panels which in turn graphically contain the low level components.
- In this week’s lecture we will be covering the Stopwatch Panel

StopwatchPanel - declarations

```
package stopwatch;
import java.awt.BorderLayout;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
public class StopwatchPanel extends JPanel{
    . . .
    private JLabel timerLabel;
    private JPanel timerPanel;
    private JPanel controlPanel;

    private JButton startButton;
    private JButton stopButton;
    private JButton resetButton;
    . . .
}
```

- This part of the implementation contains the required package declaration and imports, the declaration of the class as a subclass of **JPanel** and the declaration (but not the construction) of instance variables for the four visible components, and the two intermediate **JPanels**.

StopwatchPanel - constructor

```
. . .  
public StopwatchPanel(StopwatchController controller ) {  
    initComponents();  
    startButton.addActionListener(controller.new StartActionListener());  
    stopButton.addActionListener(controller.new StopActionListener());  
    resetButton.addActionListener(controller.new ResetActionListener());  
}
```

- **Code in the constructor is part of the preparation phase**
- The constructor takes an object reference to a **StopwatchController** as parameter. It first of all calls the private method *initComponents()* which will create the visible components and arrange them in a hierarchy of panels (see next slides).
- It then creates an instance of the different listener classes corresponding to each **JButton**. These listener classes are *inner classes* of the class **StopwatchController** (see subsequent lecture). It then registers each of the listeners to its corresponding **JButton**.
- Each **JButton** sees its listener via the **ActionListener** interface

StopwatchPanel – *initComponents()* -1

```
. . .
private void initComponents() {
    this.setLayout(new BorderLayout());
    timerLabel = new JLabel("dummy");//will be immediately overwritten
    timerPanel = new JPanel();
    timerPanel.add( timerLabel);
    startButton = new JButton(    "Start");
    stopButton = new JButton(    "Stop");
    resetButton = new JButton(    "Reset");
    . . .
}
. . .
```

- Code in this method is part of the preparation phase.
- A BorderLayout object is constructed and associated with StopwatchPanel via the *setLayout()* method. The effect of this is that subcomponents of StopwatchPanel will be laid out according the BorderLayout geometry
- A new instance *timerLabel* of the class JLabel is constructed. This is then added to a new instance *timerPanel* of the class JPanel, which is laid out using the default FlowLayout layout manager – centralizing the label.
- New instances of JButton for Start, Stop, and Reset are constructed

StopwatchPanel – *initComponents()* -2

```
. . .  
private void initComponents() {  
    . . .  
    controlPanel    = new JPanel();  
    controlPanel.add( startButton);  
    controlPanel.add( stopButton);  
    controlPanel.add( resetButton);  
    this.add( timerPanel, BorderLayout.CENTER);  
    this.add( controlPanel, BorderLayout.SOUTH);  
}//end of initComponents  
. . .
```

- A new instance *controlPanel* of the class *JPanel*, with the default *FlowLayout* is constructed.
- Each of the *JButtons* is then added in turn to this subpanel
- The *timerPanel* and *controlPanel* are then added as subpanels to the *StopwatchPanel*. The first is placed in the *CENTER* compartment and the second is placed in the *SOUTH* compartment as given by the static constants of *BorderLayout*.

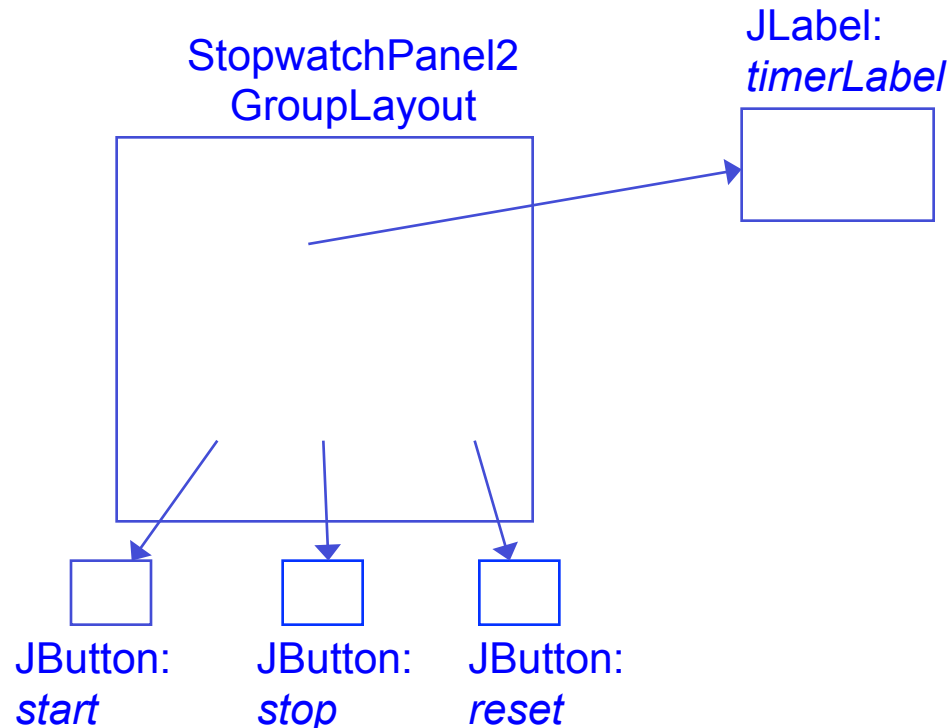
StopwatchPanel – *setButtons(...)* and *setTime(...)*

```
public void setButtons(boolean startEnabled, boolean stopEnabled,  
                       boolean resetEnabled) {  
    startButton.setEnabled(startEnabled);  
    stopButton.setEnabled(stopEnabled);  
    resetButton.setEnabled(resetEnabled);  
}
```

```
public void setTime(String time) {  
    timerLabel.setText(time);  
}
```

- **These methods have code that is part of the operation phase**
- *setButtons(. . .)* sets the buttons to be enabled or disabled according to the values of the parameters provided.
- *setTime(. . .)* sets the text of *timerLabel*, using the *setText(...)* method of *JLabel*, according to the value of the *time* parameter provided.

StopwatchPanel2 Layout management



- Building a JPanel using Drag and Drop in NetBeans automatically generates a GroupLayout associated with the JPanel
- You don't need to understand the details of the generated code
- You can only edit the components and the positioning of the components interactively
- Typically you may want to use a combination of Drag and Drop of individual JPanels *and* programmatic layout configuration in your artefact

StopwatchPanel2 initComponents(), instance variable declarations - 1

- Only initComponents() and component/panel field declarations are different to StopwatchPanel

```
/**
 * This method is called from within the constructor to initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is always
 * regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {
    startButton = new javax.swing.JButton();
    stopButton = new javax.swing.JButton();
    resetButton = new javax.swing.JButton();
    timerLabel = new javax.swing.JLabel();
    startButton.setText("Start");
    stopButton.setText("Stop");
    resetButton.setText("Reset");
    timerLabel.setText(" ");
    timerLabel.setMinimumSize(new java.awt.Dimension(100, 16));
    org.jdesktop.layout.GroupLayout layout = new org.jdesktop.layout.GroupLayout(this);
    this.setLayout(layout);
}
```

StopwatchPanel2 initComponents(), instance variable declarations - 2

```
layout.setHorizontalGroup(  
    layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)  
        .add(layout.createSequentialGroup()  
            .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)  
                .add(layout.createSequentialGroup()  
                    .add(38, 38, 38)  
                    .add(startButton)  
                    .add(18, 18, 18)  
                    .add(stopButton)  
                    .add(18, 18, 18)  
                    .add(resetButton))  
                .add(layout.createSequentialGroup()  
                    .add(143, 143, 143)  
                    .add(timerLabel, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,  
                        org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,  
                        org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)))  
            .addContainerGap(70, Short.MAX_VALUE))  
);
```

StopwatchPanel2 initComponents(), instance variable declarations - 3

```
layout.setVerticalGroup(  
    layout.createParallelGroup(org.jdesktop.layout.GroupLayout.LEADING)  
        .add(org.jdesktop.layout.GroupLayout.TRAILING, layout.createSequentialGroup()  
            .add(14, 14, 14)  
            .add(timerLabel, org.jdesktop.layout.GroupLayout.PREFERRED_SIZE,  
                org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,  
                org.jdesktop.layout.GroupLayout.PREFERRED_SIZE)  
            .addPreferredGap(org.jdesktop.layout.LayoutStyle.RELATED,  
                org.jdesktop.layout.GroupLayout.DEFAULT_SIZE,  
                Short.MAX_VALUE)  
            .add(layout.createParallelGroup(org.jdesktop.layout.GroupLayout.BASELINE)  
                .add(startButton)  
                .add(stopButton)  
                .add(resetButton))  
            .add(24, 24, 24))  
);  
} // </editor-fold>  
// Variables declaration - do not modify  
private javax.swing.JButton resetButton;  
private javax.swing.JButton startButton;  
private javax.swing.JButton stopButton;  
private javax.swing.JLabel timerLabel;  
// End of variables declaration
```