THE UNIVERSITY OF AUCKLAND

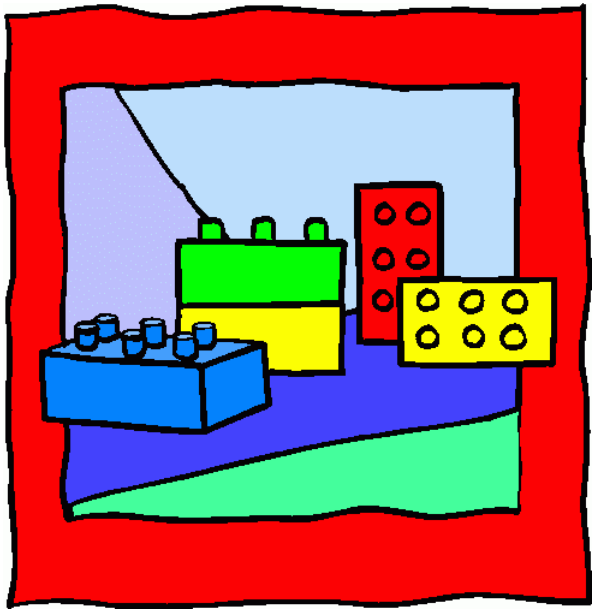
COMPSCI 230

Software Design and Construction

User Interface Layout

2013-05-06

Design Principles

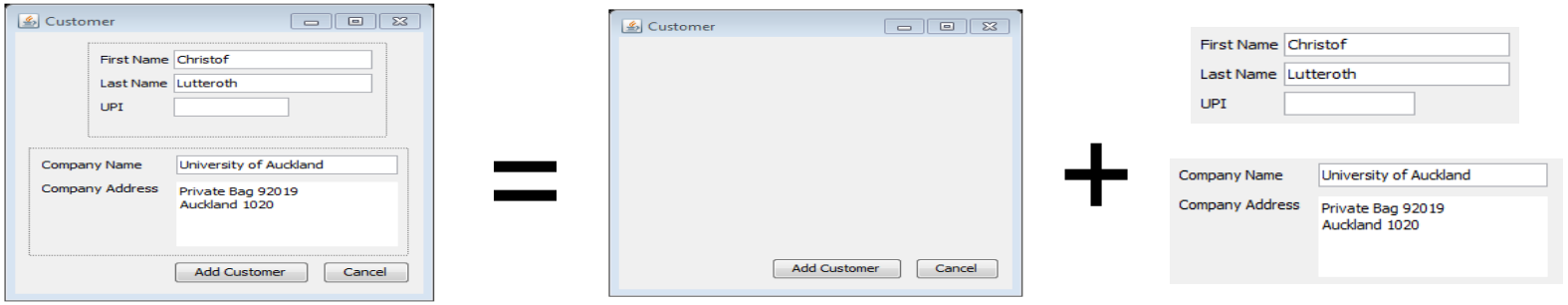


LEGO is not a toy.
It's a way of life.
(Mike Smith)

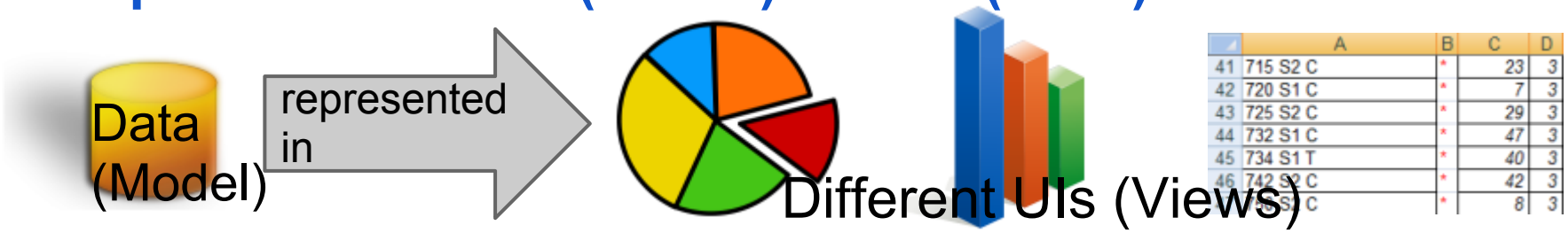
*

Recap: Separation of Concerns

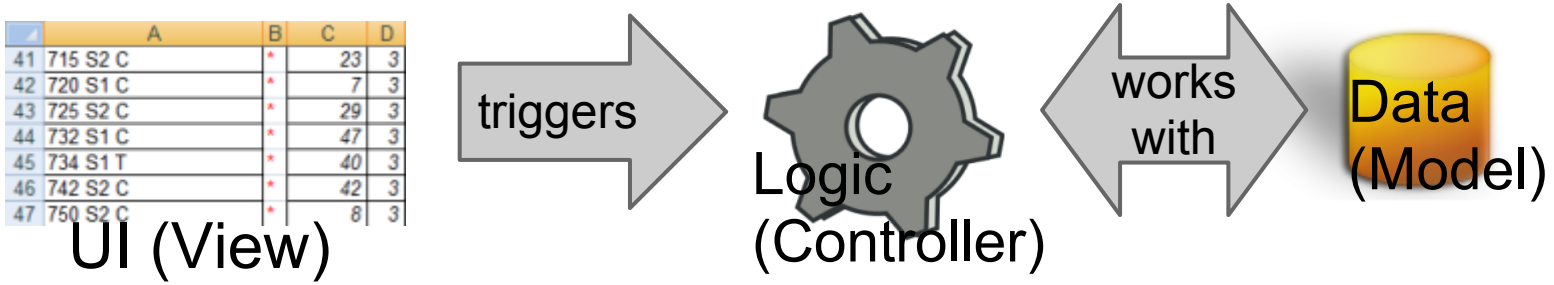
Hierarchical UI Decomposition



Separation of Data (Model) and UI (View)



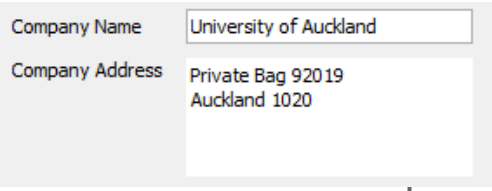
Separation of Logic (Controller) and UI (View)



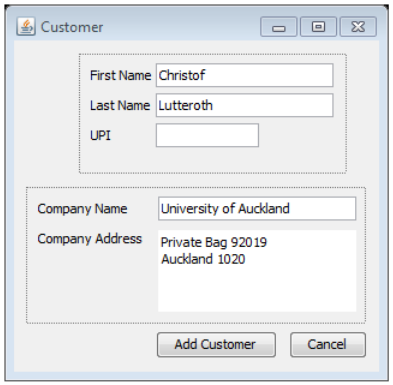
Recap: Hierarchical Decomposition

```
public class PersonDataForm extends JPanel {
    private JLabel firstNameLabel;
    private JTextField firstNameField;
    private JLabel lastNameLabel; ...

    public PersonDataForm() {
        firstNameLabel = new JLabel("First Name"); ...
    }
}
```


```
public class CompanyDataForm extends JPanel {
    private JLabel companyNameLabel; ...
}
```




```
public class CustomerForm extends JFrame {
    PersonDataForm personDataForm;
    CompanyDataForm companyDataForm;
    JButton addButton; ...
}
```



Recap: List Model Example

```
listModel = new DefaultListModel();
listModel.addElement("Alan Sommerer");
list = new JList(listModel);
...
hireButton.addActionListener(new ActionListener(){
  void actionPerformed(ActionEvent e) {
    listModel.addElement(nameField.getText());
  }});

fireButton.addActionListener(new ActionListener(){
  void actionPerformed(ActionEvent e) {
    int index = list.getSelectedIndex();
    listModel.remove(index);
  }});
...
```

Full source code at:
http://docs.oracle.com/javase/tutorial/uiswing/components/list.html

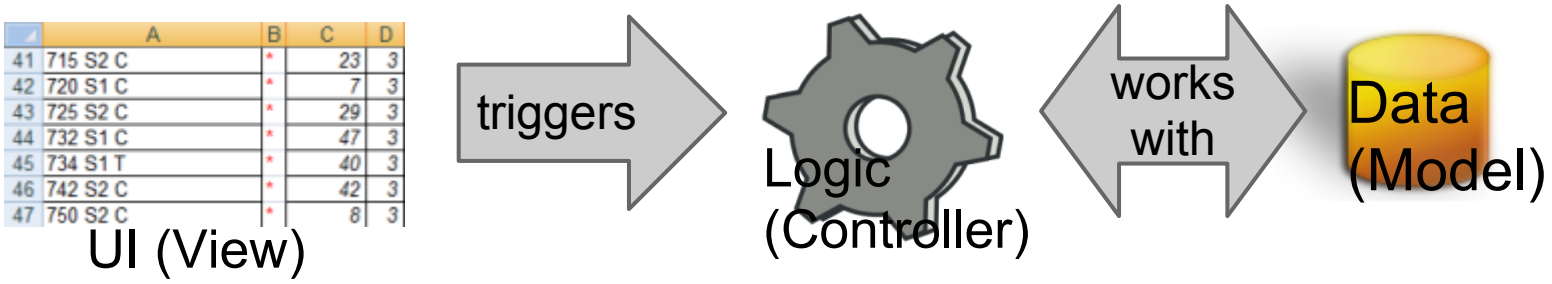
SEPARATION OF UI and Logic

Use different classes for View and Logic:

- **User Interface (View)**: the presentation of the data on the screen, i.e. the widgets that paint the data (classes & methods)
- **Logic (Controller)**: the operations that the program performs, e.g. decisions, calculations, data processing/filtering, etc.

The logic of an application is implemented in **your own classes**

- Methods for the different operations triggered through the UI that read data from the model and work with it
- Should have a well-defined interface to view
- Main advantage: easier development & maintenance through separation of concerns



Separation of Logic Example

```
…
hireButton.addActionListener(new ActionListener(){
  void actionPerformed(ActionEvent e) {
    String name = nameField.getText();
    logic.hire(name);
  }});

fireButton.addActionListener(new ActionListener(){
  void actionPerformed(ActionEvent e) {
    String name = (String) list.getSelectedValue();
    logic.fire(name);
  }});
…
```

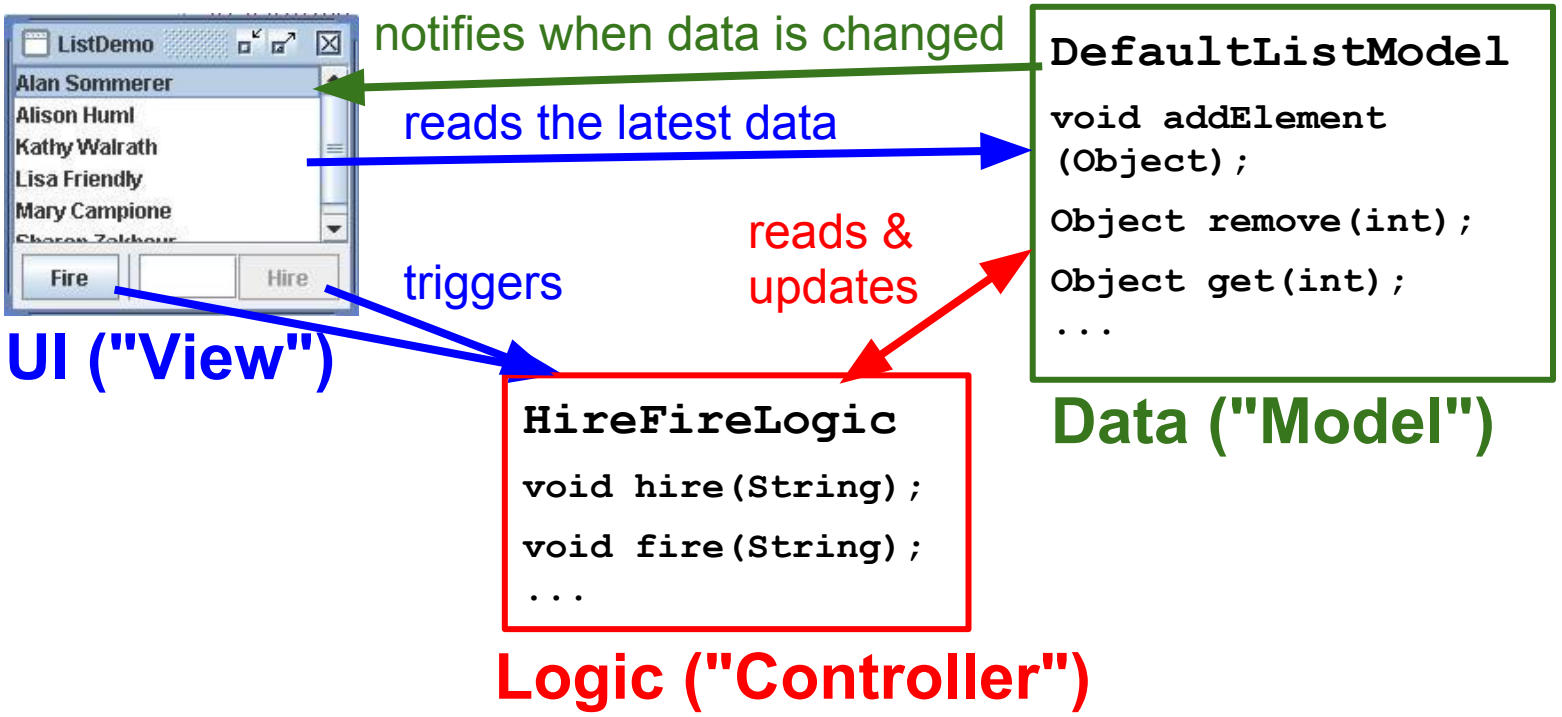
Logic class defines methods for hiring and firing e.g. **`hire()`**

- Validate input: check if the name is correct
- Check data constraints: make sure there is a vacancy
- Update model: add new employee

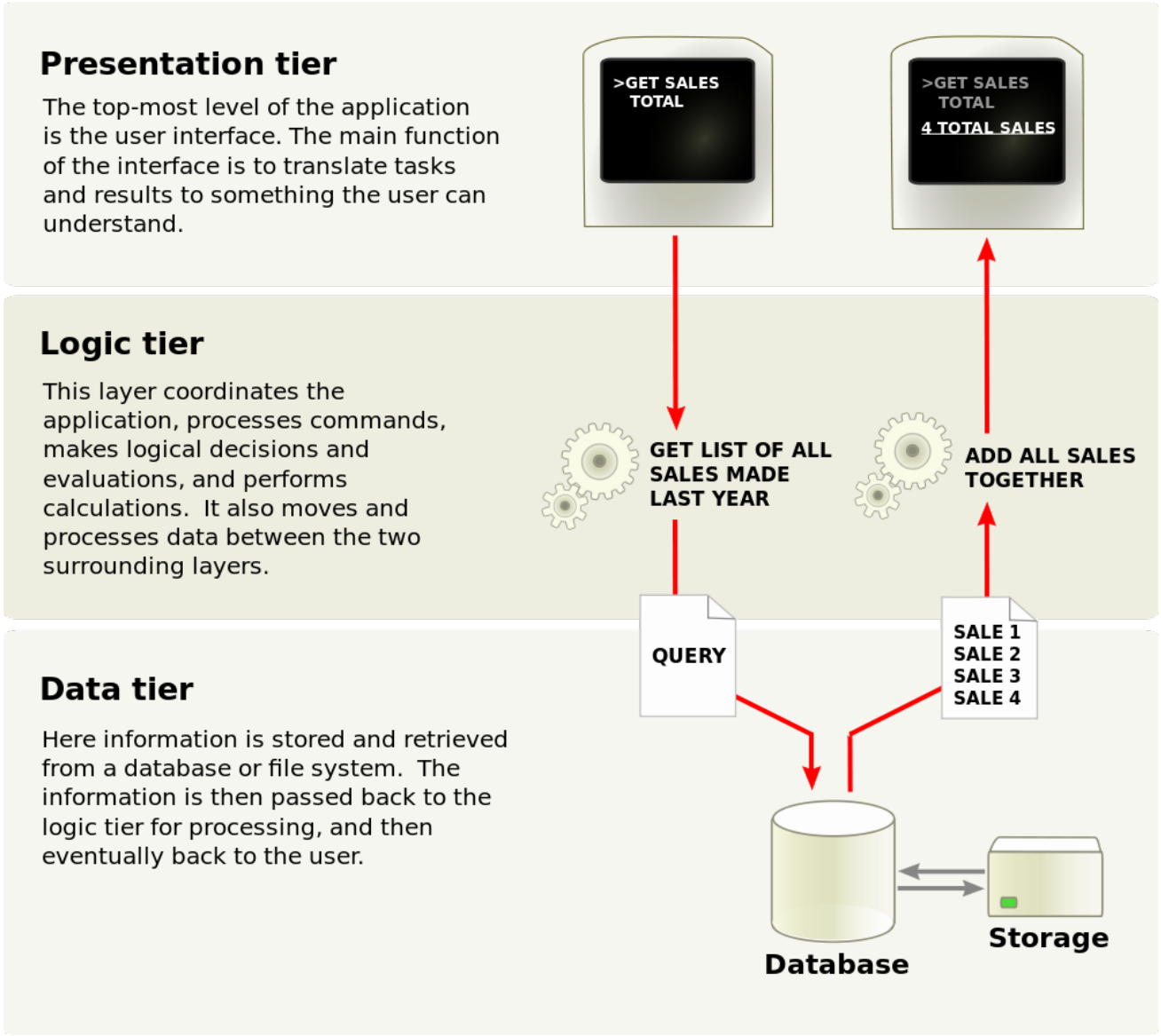
Model-View-Controller (MVC)

Separate UI ("View"), Logic ("Controller") and Data ("Model")

- UI shows the data (reads it from the model objects)
- When UI is used, logic is triggered (through event listeners)
- Logic reads and updates the data to implement functionality
- Data notifies the UI when data is updated
- Note: there are various slightly different definitions of MVC



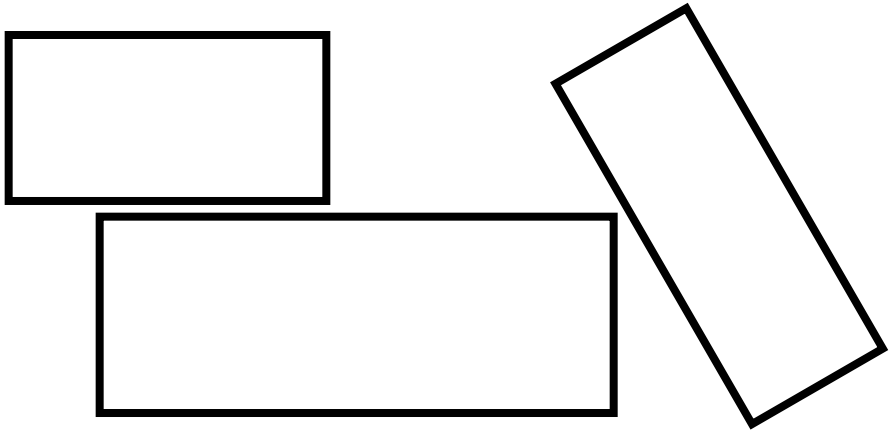
Recap: Three-Tier Architecture

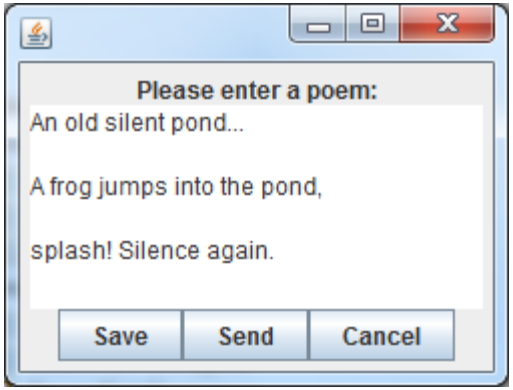


Separation of concerns

- Tiers can be developed & maintained fairly independently
- Important for system evolution
- Similar to other designs such as model-view-controller (MVC)

Layout Managers



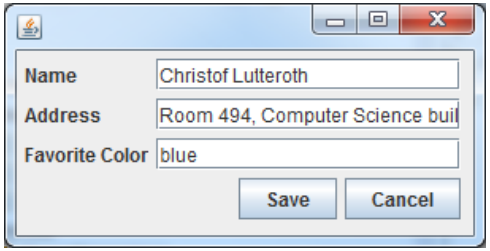


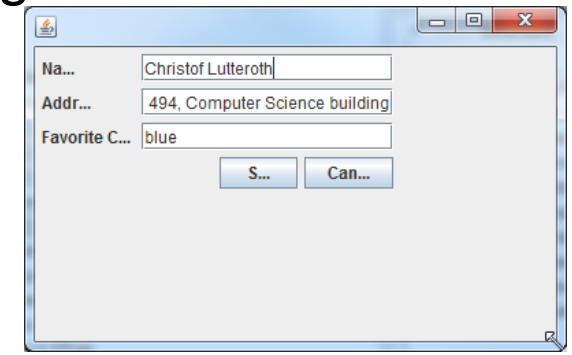
Layout Managers

Layout managers are components of a widget toolkit

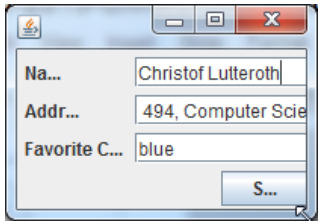
- Can be given to a container widget
- Get a layout specification as input
- Recalculate the positions and sizes of the child widgets after each container resizing
- Support for resizing, different screens, possibly even different devices

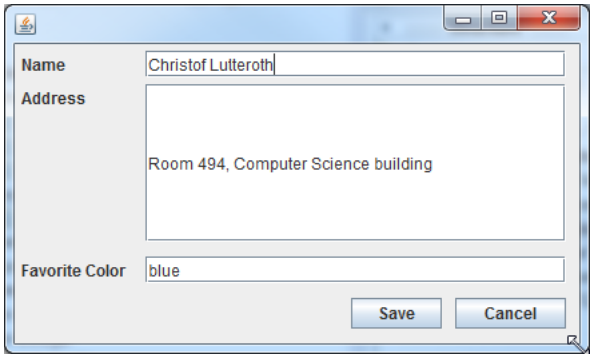
Original UI



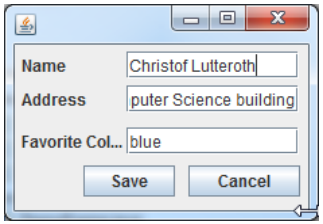


Resizing without Layout Manager





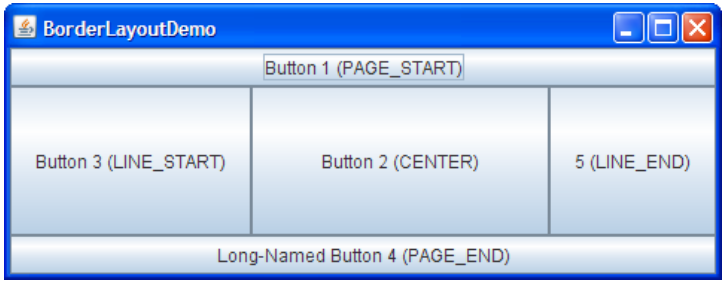
Resizing with Layout Manager

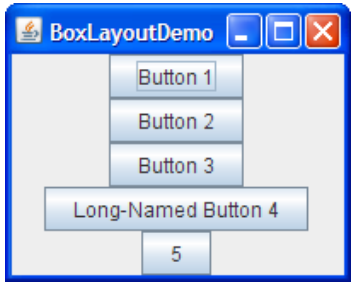


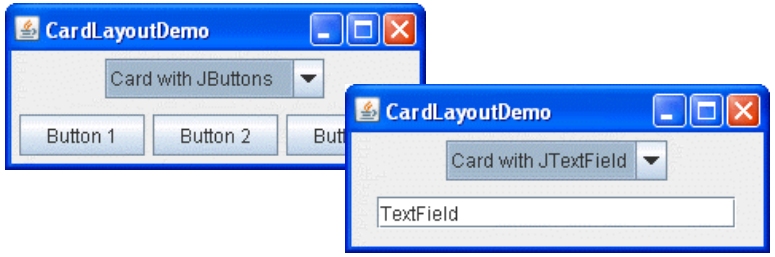
Swing Layout Managers

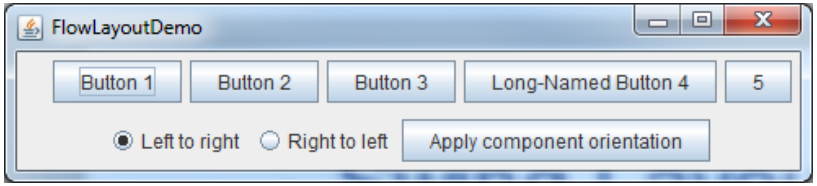
Layout managers implement the **`LayoutManager`** interface

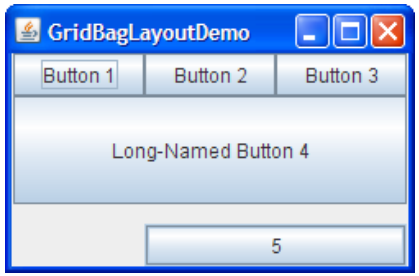
- Has a method **`void layoutContainer(Container parent)`** that does all the layout work
- Is called by the container whenever it is changed / resized











Using Layout Managers

1. Select the layout manager by setting the **`layout`** property (default: **`BorderLayout`** or **`FlowLayout`**), e.g.

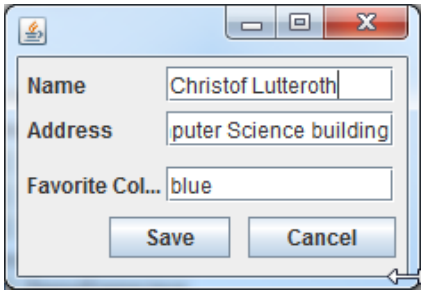
```
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

2. Specify layout with extra parameters when adding widgets to your GUI, e.g.

```
pane.add(button1, BorderLayout.PAGE_START);
```

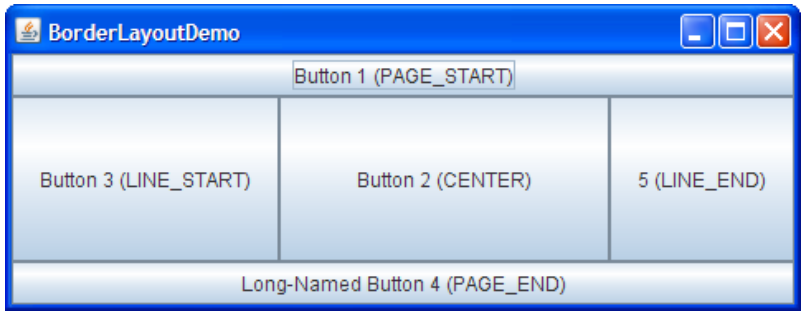
3. Provide additional size information for widgets ("layout hints") using the **`minimumSize`**, **`preferredSize`**, **`maximumSize`** properties e.g.

```
button1.setMaximumSize( new Dimension(
 Integer.MAX_VALUE, Integer.MAX_VALUE));
```



BorderLayout

- Places components in up to five areas: top, bottom, left, right, and center
- All extra space is placed in the center area
- Fairly easy to use, but very limited



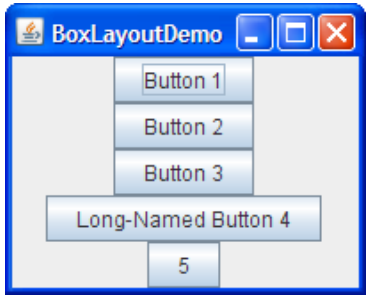

```
...
Container pane =
    getContentPane();
JButton button = new JButton
(
    "Button 1 (PAGE_START)");
pane.add(button,
    BorderLayout.PAGE_START);
```

```
button = new JButton(
    "Button 2 (CENTER)");
button.setPreferredSize(
    new Dimension(200, 100));
pane.add(button,
    BorderLayout.CENTER);
button = new JButton(
    "Button 3 (LINE_START)");
pane.add(button,
    BorderLayout.LINE_START);
button = new JButton(
    "Long-Named Button 4 ...");
pane.add(button,
    BorderLayout.PAGE_END);
...
```

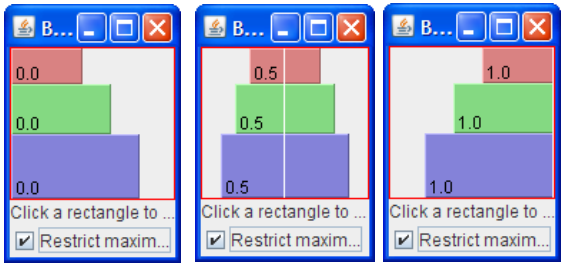
http://docs.oracle.com/javase/tutorial/uiswing/layout/border.html

BoxLayout

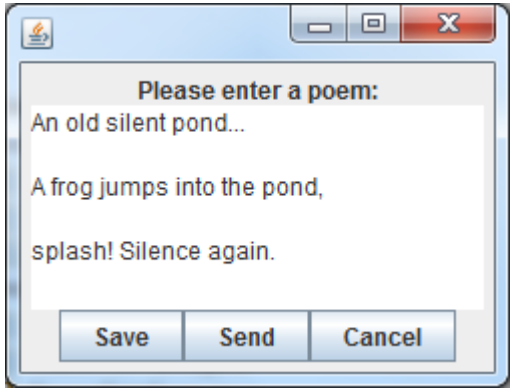
- Puts components either in a single row or in a single column



- Also possible to set alignment of each widget, e.g. **`Component.LEFT_ALIGNMENT`**, **`Component.CENTER_ALIGNMENT`**, **`Component.RIGHT_ALIGNMENT`**

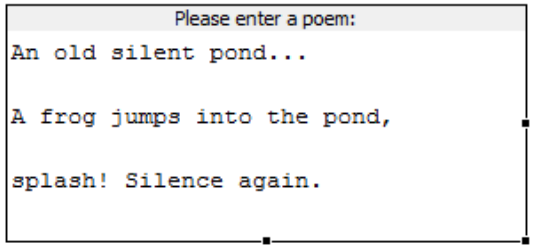


- Used often when decomposing UI into panels with single rows/columns of widgets



http://docs.oracle.com/javase/tutorial/uiswing/layout/box.html

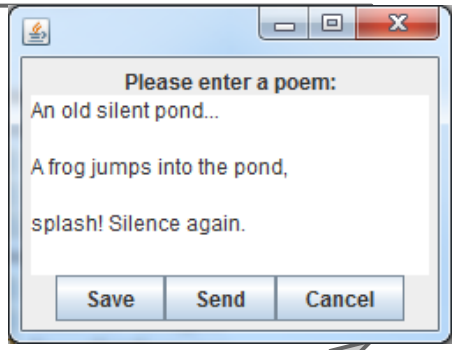
BoxLayout Example




```
public class PoemPanel extends JPanel {
    public PoemPanel() {
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        JLabel label = new JLabel("Please enter ...");
        label.setAlignmentX(Component.CENTER_ALIGNMENT);
        label.setHorizontalAlignment(SwingConstants.LEFT);
        add(label);
        add(new JTextArea());
}   }
```

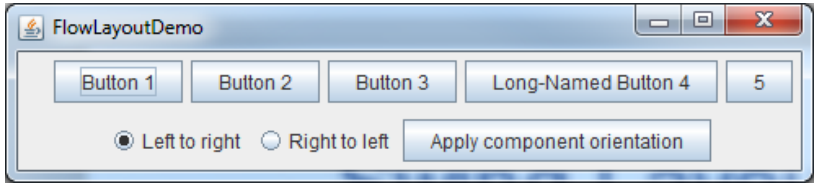
```
public class ButtonPanel extends JPanel {
    public ButtonPanel() {
        setLayout(new BoxLayout(this, BoxLayout.X_AXIS));
        ...
```

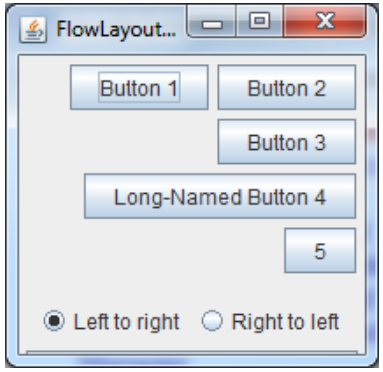

```
public class BoxLayoutExample extends JFrame {
    public BoxLayoutExample() {
        contentPane.setLayout(new BoxLayout(
            contentPane, BoxLayout.Y_AXIS));
        contentPane.add(new PoemPanel());
        contentPane.add(new ButtonPanel()); ...
```



FlowLayout

- Arranges widgets over several lines from left to right, with top to bottom line breaks, just like text
- Can also adjust flow direction (e.g. first top to bottom)
- Default for **`JPanel`**



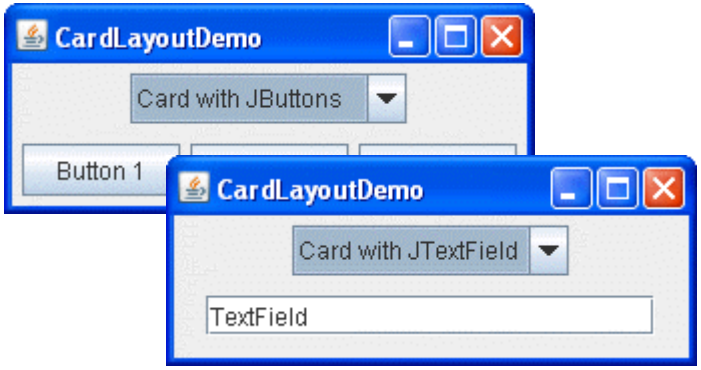


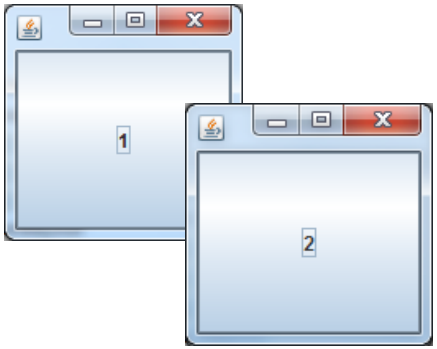

```
...
setLayout(new FlowLayout());
setComponentOrientation(
    ComponentOrientation.
        LEFT_TO_RIGHT);

add(new JButton("Button 1"));
add(new JButton("Button 2"));
add(new JButton("Button 3"));
add(new JButton(
    "Long-Named Button 4"));
add(new JButton("5"));
...
```

http://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html

CardLayout

- Layout that contains different widgets at different times
- Used for switching between different UI screens in the same window / panel
- Like changing the card on the top of a stack of cards



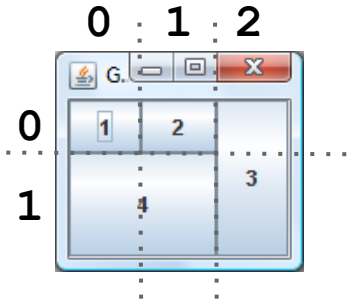


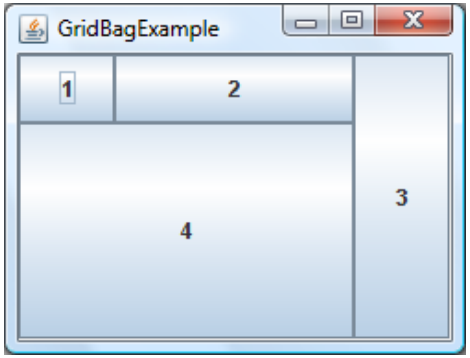

```
...
CardLayout cardLayout =
    new CardLayout();
setLayout(cardLayout);

add(new JButton("1"), "Screen
1");
add(new JButton("2"), "Screen
2");
        cardLayout.show
(getContentPane(),
    "Screen 2");
...
```

http://docs.oracle.com/javase/tutorial/uiswing/layout/flow.html

GridBagLayout

- Widgets arranged in table (aka. a grid) with rows, columns and cells
- Position and size properties of each widget are specified in a `GridBagConstraints` object
  - `gridx` for start column index
  - `gridy` for start row index
  - `gridwidth` for column span (i.e. width in columns)
  - `gridheight` for row span (i.e. height in rows)
  - `weightx` and `weighty` for relative size
  - `ipadx`, `ipady`, `insets` for margins around widgets
  - `fill` to set whether to fill extra space with the widget
- One of the most popular, also in other toolkits (e.g. HTML tables)





http://docs.oracle.com/javase/tutorial/uiswing/layout/gridbag.html

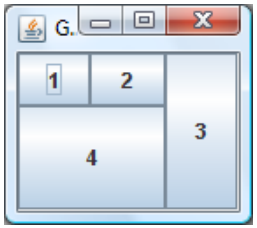
GridBagLayout Example

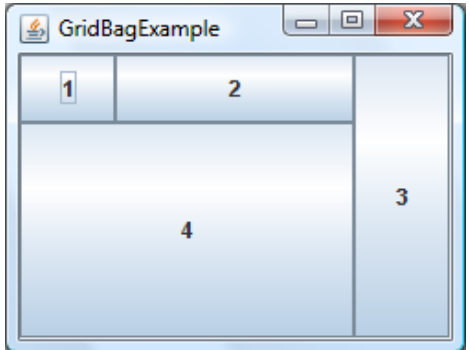
```
panel.setLayout(
   new GridBagLayout());
GridBagConstraints c =
   new GridBagConstraints();

JButton b1 = new JButton
   ("1");
c.gridx = 0; c.gridy = 0;
c.weightx = 0.1;
c.weighty = 0.1;
c.fill =
   GridBagConstraints.BOTH;
panel.add(b1, c);
```

```
JButton b2 = new JButton
   ("2");
c.gridx = 1; c.gridy = 0;
c.weightx = 0.8;
panel.add(b2, c);
```

```
JButton b3 = new JButton
   ("3");
c.gridx = 2; c.gridy = 0;
c.gridheight = 2;
c.weightx = 0.1;
panel.add(b3, c);
```

```
JButton b4 = new JButton
   ("4");
c.gridx = 0; c.gridy = 1;
c.gridheight = 1;
c.gridwidth = 2;
c.weighty = 0.8;
panel.add(b4, c);
```





Constraint-Based Layout

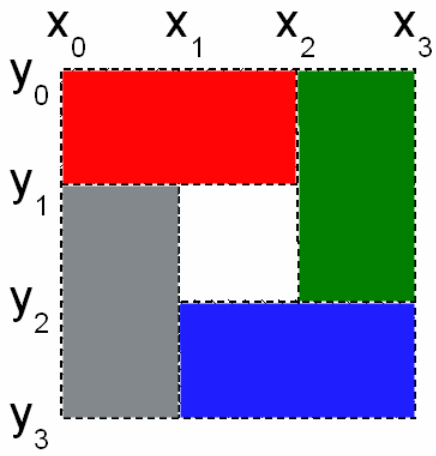
- Modern way of specifying layout, e.g. Apple AutoLayout
- Grid lines are variables with coordinates (*tabs*)
- Place controls by choosing left, top, right and bottom tab:

$$a =_{def} (x_1, y_1, x_2, y_2, layer, content)$$

- Positions and sizes are specified with linear constraints, e.g. $x_2 - x_1 = 2(x_4 - x_3)$
- Layout is calculated with a constraint solver

Example:

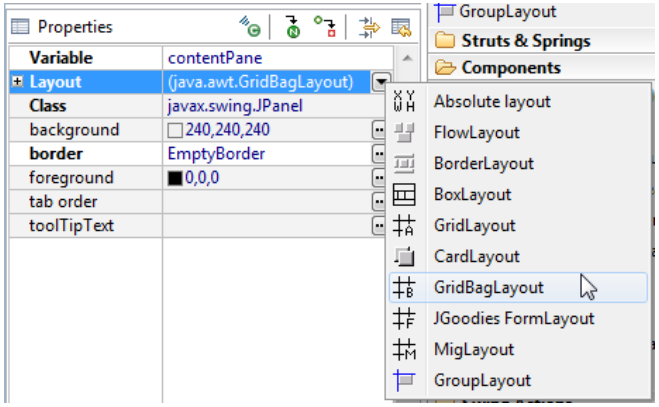
$$\begin{aligned} A = \{&(x_0, y_0, x_2, y_1, 0, red), (x_2, y_0, x_3, y_2, 0, green), \\ &(x_1, y_2, x_3, y_3, 0, blue), (x_0, y_1, x_1, y_3, 0, grey), \\ &(x_1, y_1, x_2, y_2, 0, empty)\} \end{aligned}$$



Layout Managers and WindowBuilder

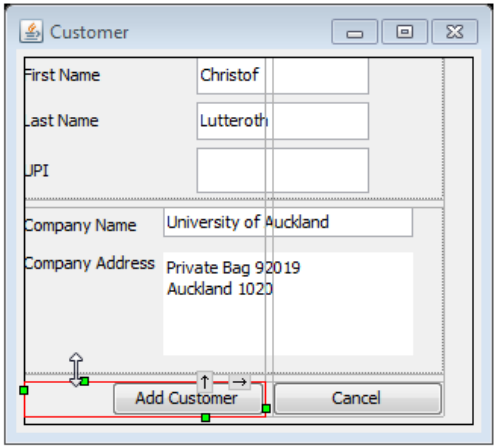
WindowBuilder supports the common Swing layout managers

- Absolute layout = no layout manager
  i.e. only fixed positions and sizes



Layouts can be edited visually, to a degree

- Drag & drop
- Manual coding necessary sometimes



Support for converting layouts, e.g.

- Start with absolute layout (easy)
- Then just switch to GridBagLayout to make UI somewhat resizable

THE UNIVERSITY OF AUCKLAND

Summary

- **Model-View-Controller** is a common design for separation of concerns in apps
- **Layout managers** make sure UIs resize properly
  - **`BorderLayout`** for the simplest layouts
  - Nested **`BoxLayout`**s for decomposing layout into rows and columns
  - **`GridBagLayout`** for sophisticated, tabular layouts

Assignment 3 Extension:
due on Monday 13/5 at 4pm

Quiz

1. According to the lecture, what is the "controller" in MVC?
2. What does a layout manager do?
3. How does GridBagLayout work? Explain how positioning of widgets works, and how widget sizes are generally specified.

Poweruser tools
Windows reboot handle