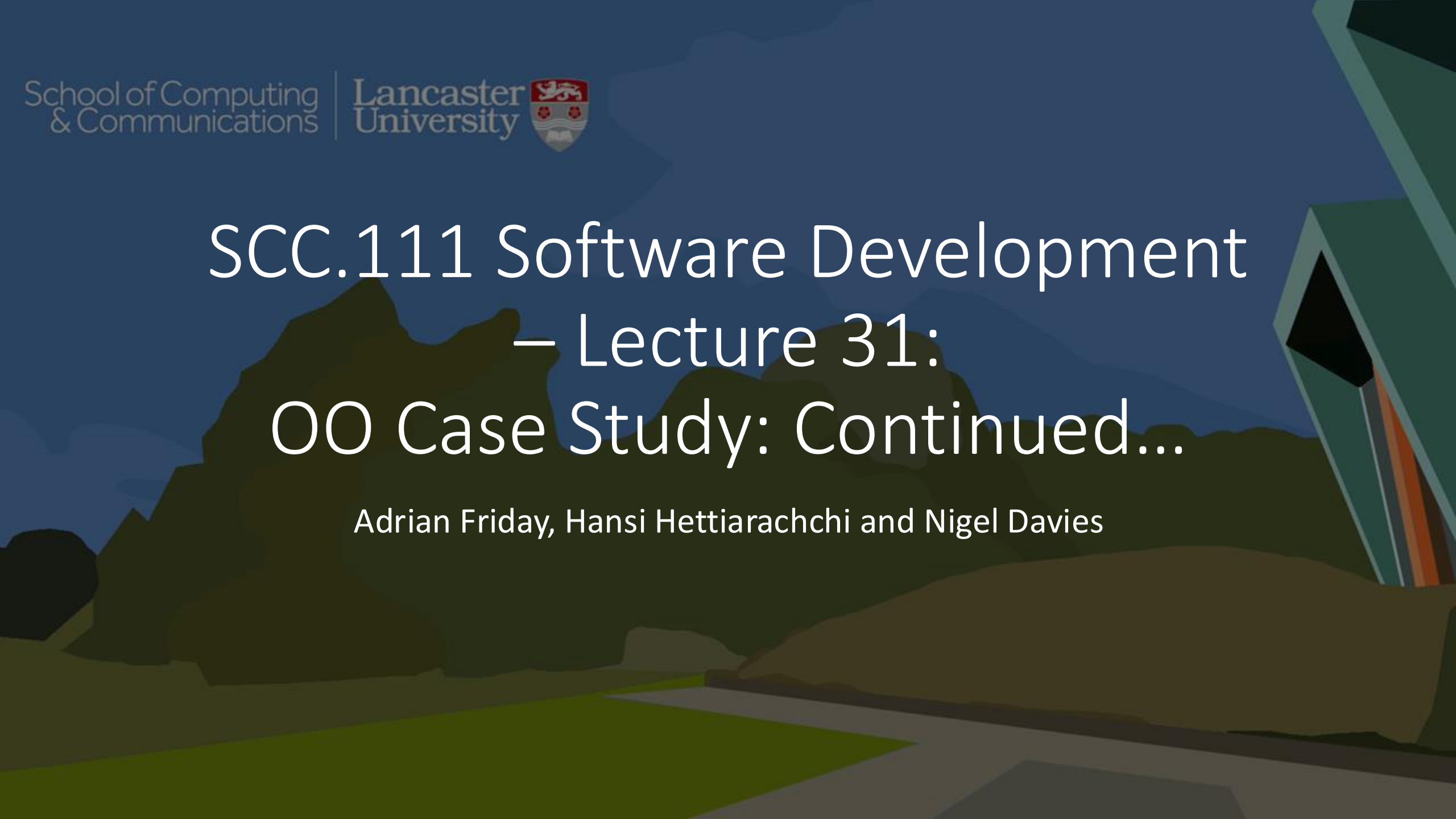


SCC.111 Software Development – Lecture 31: OO Case Study: Continued…

Adrian Friday, Hansi Hettiarachchi and Nigel Davies

Introduction

Lancaster University

- Last lecture, we looked at:
  - How OO principles are used to enable graphical user interfaces in Java
  - A real examples of **composition** in action
  - The static and final keywords
  - Why Lucy from the Lego movie is awesome

- Today we're going to continue this case study…
  - Component organization
  - Asynchronous programming
  - See our first example of a **design pattern**
  - **By the end of this lecture, you should be able to create your own fully functional GUI in Java**

Creating non-trivial user interfaces

Lancaster University

Most GUIs require many components to be useful

- We have seen how we can easily add many components to a JPanel…
- But how do we defined where they are displayed? Statically defined locations?
- All swing component allow you to define specific locations if you really want to.

BUT Java explicitly tries to prevent you from specifying where your components go.

- Swing provides a set of Layout Managers class that dynamically layout GUIs for you.
- These help to keep your application flexible and platform independent.
- Layout Managers are part of a package called AWT, so…

```
import java.awt.*;
```

FlowLayout

Lancaster University

FlowLayout is the simplest layout manager

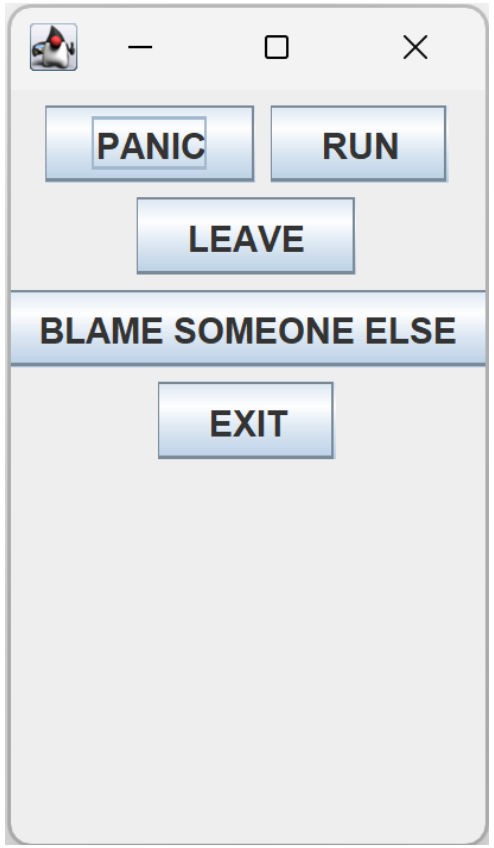
- Arrange components best to fit the size of the JPanel
- Purely in a left to right, top to bottom order
- Useful for simple GUIs, lists etc.

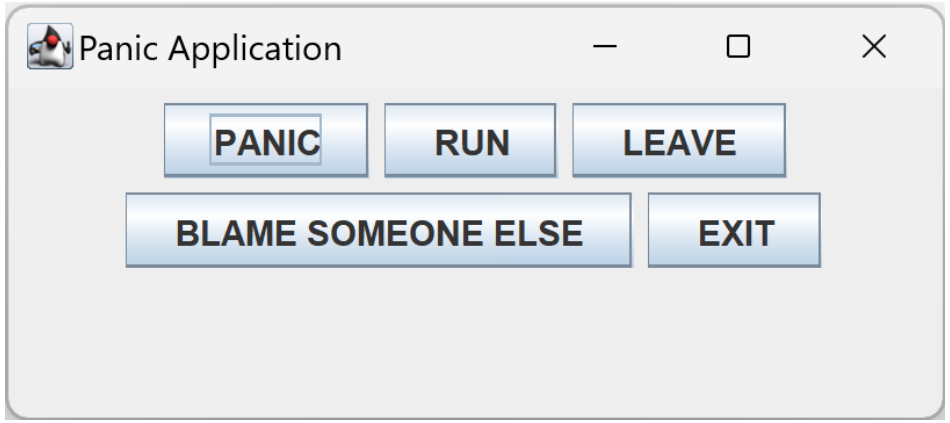
- Implemented in the FlowLayout class, instantiate one if you need one….
- The **setLayout()** method provided by JPanel allows you to define the layout manager you want to use in that JPanel

```
FlowLayout layout = new FlowLayout();     // Create Layout manager
panel.setLayout(layout);                  // Assign to Panel
```

FlowLayout: examples

Lancaster University





FlowLayout: benefits and drawbacks

Lancaster University

Benefits and drawbacks:

- Highly reactive: layout reacts dynamically to changes in size of the panel
- Very simple to use
- Lack of control over where components are placed

Some very limited control over alignment possible

- By default, FlowLayout will centre your components in panel
- But you can specify left, right or centre alignment in the constructor
- Note the American spelling of CENTER

```
FlowLayout layout = new FlowLayout(FlowLayout.LEFT);
FlowLayout layout = new FlowLayout(FlowLayout.CENTER);
FlowLayout layout = new FlowLayout(FlowLayout.RIGHT);
```

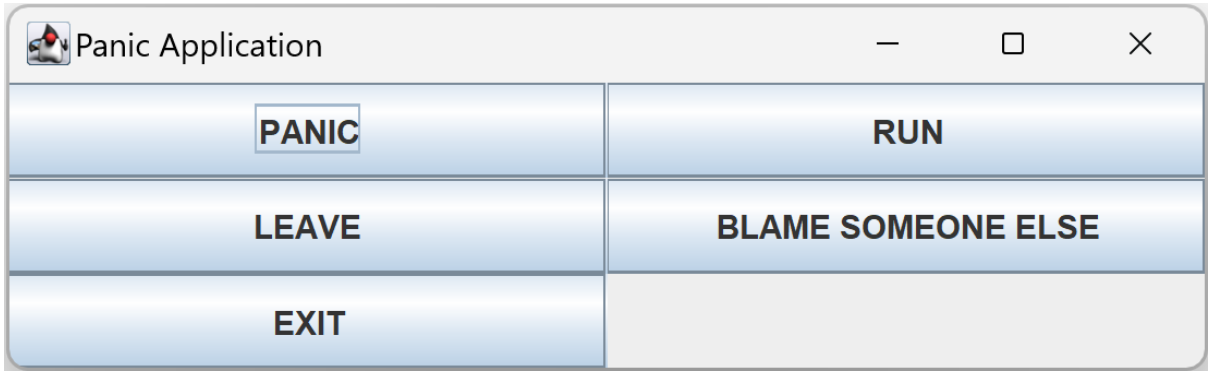
GridLayout

Lancaster University

Fixed size components in a matrix

- The GridLayout fits components into a **n x m** grid structure.
- Still arranged left to right, top to bottom, but on grid boundaries, and you can specify the shape of the grid…

```
GridLayout layout = new GridLayout(int rows, int columns);
panel.setLayout(layout);
```



GridLayout: benefits and drawbacks

Lancaster University

GridLayout managers are excellent for repeating sets of components

- Which is remarkably common…
- e.g. a mixing desk, numerical keyboard, powerpoint…

Far too clinical to layout everything this way though…

- We still don’t really have any control over which component goes where…

BorderLayout

Lancaster University

BorderLayout provides relative positioning of up to five components

- BorderLayout divides the JPanel into five areas
  - North, South, East, West and Center
  - Compass points take priority over space

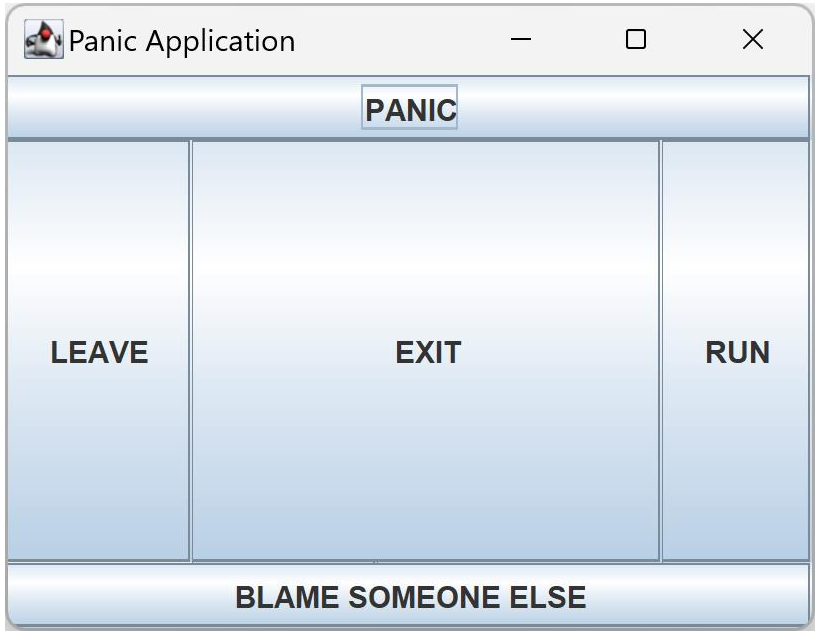
Specify which area to put component inside the add method

  - Only one component permitted per location

```
BorderLayout layout = new BorderLayout();
panel.setLayout(layout);
panel.add("North", buttonPanic);
```

BorderLayout: examples

```
BorderLayout layout = new BorderLayout();
panel.setLayout(layout);
panel.add("North", buttonPanic);
panel.add("East", buttonRun);
panel.add("West", buttonLeave);
panel.add("South", buttonBlame);
panel.add("Center", buttonExit);
```



Setting your own layout

Lancaster University

You can specify positions manually if you wish

- Specify a **null** Layout manager
- Use **setLocation(int x, int y)** and **setSize(int x, int y)** to manually place components
- All Swing components respond to these methods

```
panel.setLayout(null);  // Disable layout management
```

```
public void setLocation(int x, int y);
public void setSize(int x, int y);
```

- We can also use ratios

```
JButton button = new JButton("Press");
int width = (int)(frame.getWidth() * 0.5);
int height = (int)(frame.getHeight() * 0.2);
button.setSize(width, height);
```

There is no one-size-fits-all solution

Lancaster University

- Manual positioning provides total control, but no flexibility
  - Difficult to handle conditions of window resizing
  - Painful to specify locations manually (IDEs help here though)

- Layout managers provide flexibility at the cost of control
  - Need to give up control of GUI design – often not too tempting!

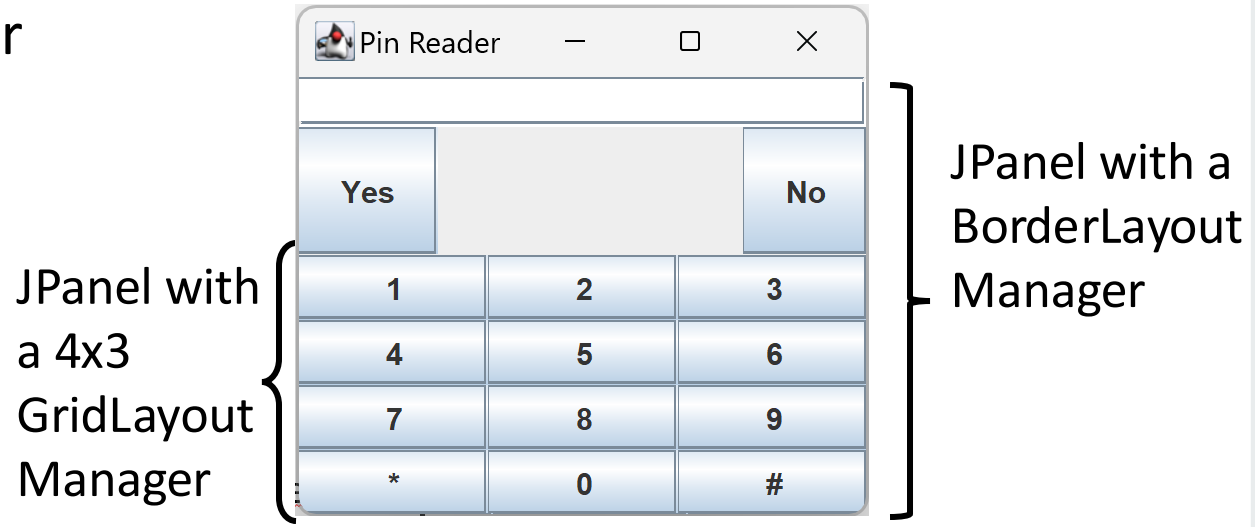
But we can draw on the modularity of OO to help

Using multiple panels…

Panels can be placed inside other panels…

- Each JPanel has its own LayoutManager
- This can give us much more flexibility with the layout, while maintaining the ability to handle window resizes

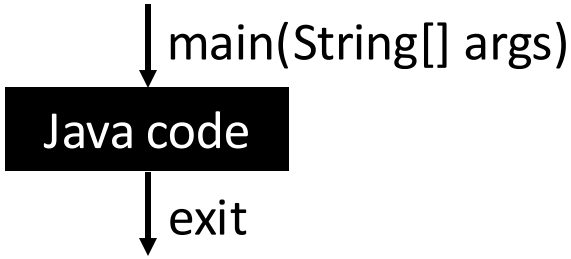
- For example, a panel with a GridLayout inside a panel with a BorderLayout:



Sequential programming

Lancaster University

- So far, all our Java programs have looked a bit like this:



- Which means all **interactive** programs would have to look like this:

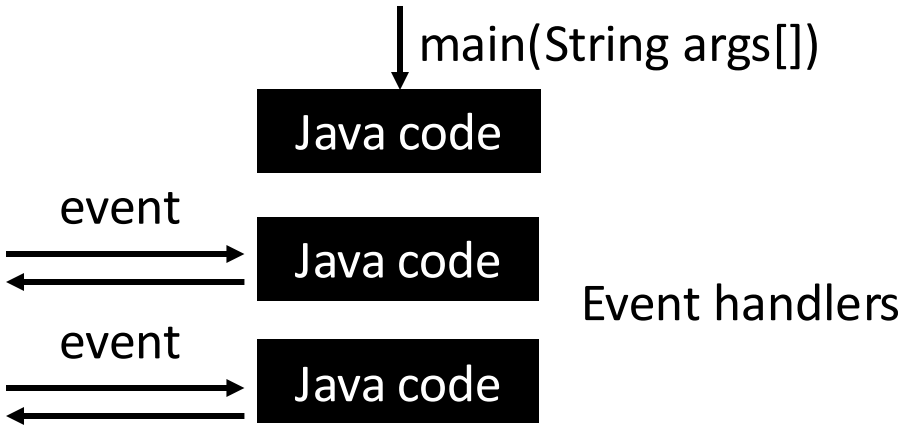
```
while (true) {
    // Do some work
}
```

- This is **not good** for modularity or efficiency…

Event-based programming

Lancaster University

- **Also known as asynchronous programming**
- When we use event-based programming, your code does NOT have a simple start to end flow of execution… Instead:
- You application registers interest in certain events (e.g. button presses)
- The environment (in this case Java) informs the application when they occur



Listener interfaces

Lancaster University

Swing lets you register "Listeners" on GUI components:

- You register an interest in the events you wish to receive
- You provide an object with a well-known method
- That method is then invoked when that event occurs

For example, the following interfaces notify you when:

- **ActionListener**: when buttons are clicked
- **ChangeListener**: when sliders are moved
- **KeyListener**: when a button is pressed on the keyboard
- **MouseListener**: when a mouse button is pressed/moved
- **WindowListener**: when windows are resized/closed/minimized

Implementing an ActionListener

Lancaster University

All Listeners follow the same steps:

- Import the **java.awt.event** package
- Declare you want to receive an event by adding **implements ActionListener** to your class definition
- Use the **addActionListener()** method to register your interest in the relevant buttons
- Write a method called **actionPerformed()** in your class, with the parameters shown below…

```
public void actionPerformed(ActionEvent e) {
    // code you want to execute when the button is clicked
}
```

Implementing an ActionListener: example

Lancaster University

```
import java.awt.event.*;
import javax.swing.*;

public class HelloWorld implements ActionListener {

    private JButton someButton;

    public HelloWorld() {
        someButton = new JButton("Click me!");
        someButton.addActionListener(this);
    }

    public void actionPerformed(ActionEvent e) {
        // code you want to execute when the button is clicked
    }
}
```

Supporting multiple buttons…

Lancaster University

One ActionListener can be register with multiple buttons

- The **ActionEvent** object can differentiate the source for us
- Responds to the **getSource()** method
- Returns the object that generated the event (the JButton)

- Combined with a conditional this can be used to trap all your events in one place:

```
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == okButton)
        //…

    if (e.getSource() == cancelButton)
        //…
}
```

Summary

Lancaster University

- Today we learned:
  - How to use Java classes to layout a scalable GUI
  - Examples of asynchronous programming principles
  - That the observer design pattern exists.
  - More examples of OO programming in action.
  - Remember the Java API documentation contains all the details you need to know for other GUI components and listeners:

https://docs.oracle.com/en/java/javase/23/docs/api/index.html