CSY2030
Systems Design & Development

Graphical User Interfaces 1

Introduction

- Up till now you have written programs that communicate with the end user through a text-based interface
  - Using *System.out* for output
  - Using Keyboard for input.

- Java provides two sets of facilities for developing GUIs:
  - The Abstract Window Toolkit (**AWT**): package **java.awt**
  - **Swing**: package **javax.swing**

Understanding events

Text-based interface

- Predetermined sequence of events
- The program pauses execution when it expects input from the user and continues on the same set path after it receives the input

Graphical interface

- No set sequence of events
- The user can do any number of things at any time (type in a text box, resize the window, press a button)
- These responses are called *events*.
- We say the program is *event-driven*.

A first **Swing** application

```
import javax.swing.*;
public class FirstGUI
 {
        public static void main(String[] args)
        {
                JFrame f = new JFrame();
                f.setVisible(true);
        }
 }
```

Displays the window and enters the event loop

Class for drawing a window on the screen

When you run this program, a tiny window appears:

The close button does not work

Shutting down the application properly

```
import javax.swing.*;
public class FirstGUI  {
    public static void main(String[] args)
    {
            JFrame f = new JFrame( );
            f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            f.setVisible(true);
   }
}
```

Need to add a single statement to program the close button

- Now the close button will work:

Giving application a title

```
import javax.swing.*;
public class FirstGUI  {
        public static void main(String[] args)
        {
                JFrame f = new JFrame( “My first GUI”);
                f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                f.setVisible(true);
        }
}
```

To give the application a title, give the constructor a string

- The GUI will now look like the following:

Components and containers

- A **component** is any GUI element, such as a window, button or label.
- A **container** is a type of component that has the purpose of containing other components.
- Types of containers:
  - **Top-level containers**: Every Swing program contains at least one top-level container (e.g. **JFrame, JDialog** or **JApplet**). Top-level containers cannot be added to other containers.
  - **Intermediate containers**: used to group components so that they can be handled as a single component (e.g **JPanel, JTabbe**dPane).
  - **Atomic components (basic controls)**: cannot contain other components (e.g **JButton, JTextField**).

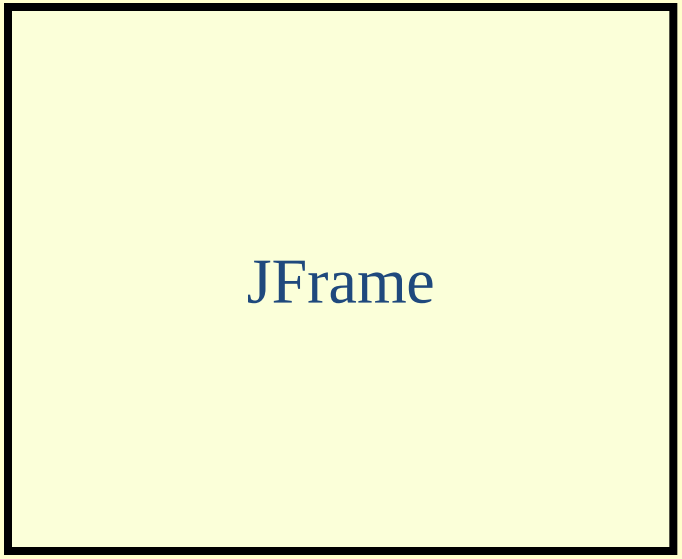
Examples of Atomic Components

Often called *widgets*:

- Label – used to put a label next to another component
- Button – used to make the program "do something"
- Checkbox component – used for yes/no, true/false response from user
- Choice component – drop-down list
- TextField – used to type single line of text
- … etc

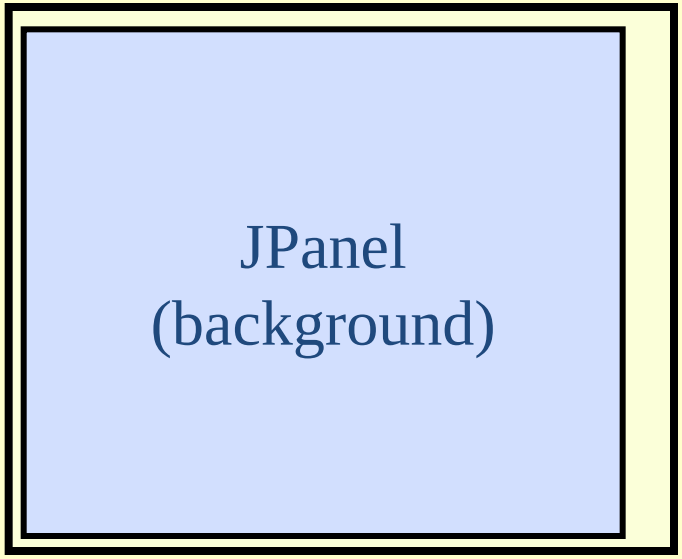
Top-Level Containers

- Top-Level Container can hold other components
  - Can hold intermediate containers
  - Can hold atomic components



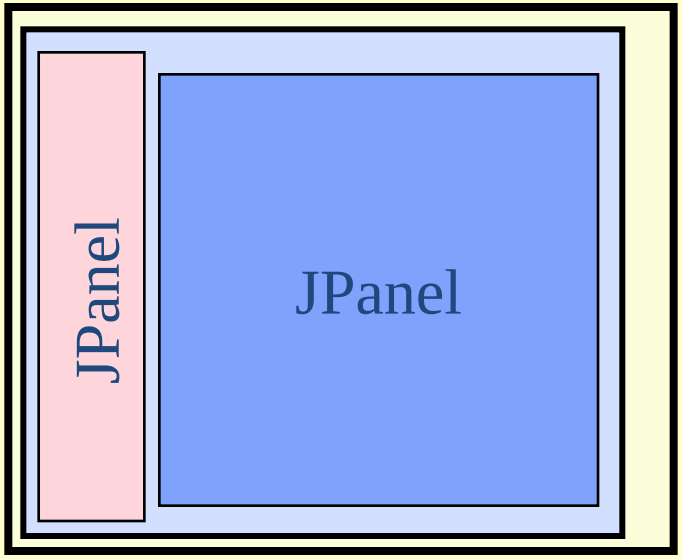
Panels

- Panels can be added to JFrames
  - Done using add() method
- Each panel can have its own layout



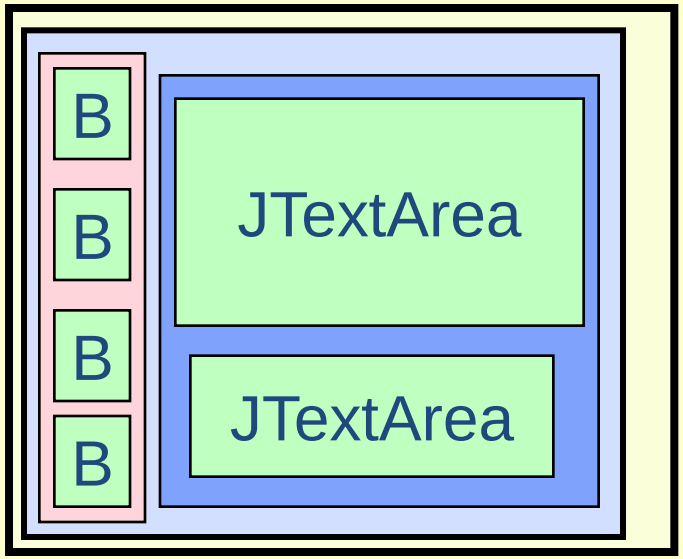
Panels cont.

- Panels can contain other panels
  - This is also done using the add() method
- Each panel can have its own layout



Panels cont.

- Panels can also contain atomic components like buttons and text fields



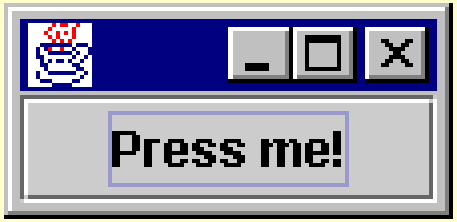
Adding a button to the application

```
import javax.swing.*;
public class FirstGUI
{
    public static void main(String[] args)
    {
        JFrame f = new JFrame( );
        JButton button = new JButton("Press me!");
        f.getContentPane().add(button);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```

Create a button labelled "Press me!"

add the button to the frame

The pack method sizes the frame so that all its contents are at or above their preferred size



Organising the code in a better way

- As we start adding more components, the main method will become too large and messy.
- A better way:
  - Create a class that extends JFrame
  - Put all components into the class (as data members)
  - Do the rest in the constructor

```
import javax.swing.*;
public class SimpleFrame extends JFrame
{
        private JButton button = new JButton("Press me!");
        public SimpleFrame()
        {
                getContentPane().add(button);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                pack();
        }
 }
```

Creating a SimpleFrame object

```
public class FirstGUI   {
        public static void main(String[] args)  {
                SimpleFrame s = new SimpleFrame( );
                s.setVisible(true);
        }
}
```

- SimpleFrame extends JFrame, therefore s is also a JFrame object (and so we can call the setVisible method).
- In the SimpleFrame class:
  - SimpleFrame defines a specialisation of JFrame by adding an additional component.
  - To call methods of JFrame (such as getContentPane or pack), we no longer need an object handle, since these methods are now inherited from JFrame).

Modified Code

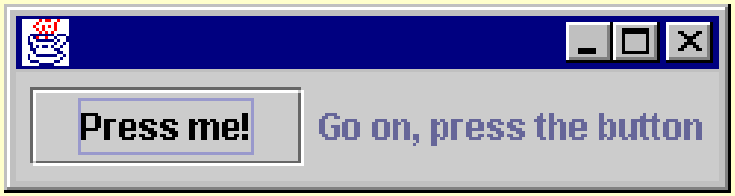
```
import javax.swing.*;
public class SimpleFrame extends JFrame
{
        private JButton button = new JButton("Press me!");
        private JLabel label = new JLabel("Go on, press the button");
        private JPanel background = new JPanel();

        public SimpleFrame()
        {
                background.add(button);  // add button to background
                background.add(label);   // add label to background

                getContentPane().add(background); // add background to frame
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                pack();
        }
 }
```

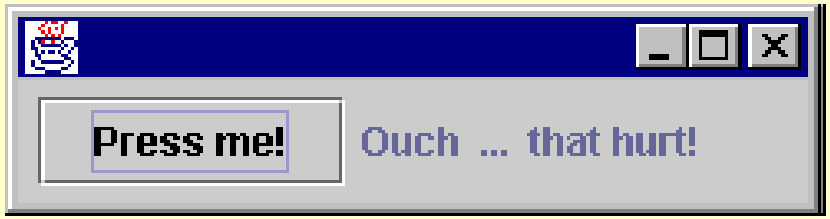
JLabel used for placing plain text on a GUI

JPanel is an intermediate container



Getting the button to do something

- Currently, if the user clicks on our button, nothing happens.
- We would like to change the program, so that the label changes when the button is clicked:



- The code that responds to that event of the user clicking the mouse on our button is called the *listener* for that button.
- We would therefore like to program the listener of the button to have the code:

```
label.setText(" Ouch ... that hurt! ");
```

```
import javax.swing.*;
import java.awt.event.*;  <-------------------------------- Code related to event handling
public class SimpleFrame extends JFrame
{
        private JButton button = new JButton("Press me!");
        private JLabel label = new JLabel("Go on, press the button");
        private JPanel background = new JPanel();
        public SimpleFrame()
        {
                button.addActionListener(
                        new ActionListener() {
                                public void actionPerformed(ActionEvent e)
                                {
                                        // code to be executed when button is pushed
                                        label.setText("Ouch  ...  that hurt! ");
                                }
                        });
                background.add(button);
                background.add(label);
                getContentPane().add(background);
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                pack();
        }
 }
```

Event Handling

- Every time the user types a character or pushes a mouse button, an event occurs.
  - Any object can be notified of the event.
  - All it has to do is implement the appropriate interface and be registered as an *event listener* on the appropriate *event source*.
- Act that results in the event Listener type:
  - User clicks a button, presses Return while typing in a text field, or chooses a menu item - **ActionListener**
  - User closes a frame (main window) - **WindowListener**
  - User presses a mouse button while the cursor is over a component -**MouseListener**
  - User moves the mouse over a component - **MouseMotionListener**
  - Component becomes visible - **ComponentListener**
  - Component gets the keyboard focus - **FocusListener**
  - Table or list selection changes - **ListSelectionListener**

Arranging Components

- **Layout managers** are used to control the size and position of components in containers.
- The Java platform provides a number of layout managers, including **BorderLayout, FlowLayout** and **GridLayout**.
- To use layout mangers, you have to import **java.awt.*.**
- To use a particular layout manager, you use the **setLayout** method.

```
import javax.swing.*;
import java.awt.*;
public class TestFlowLayout extends JFrame

      private JButton button1 = new JButton("One");
      private JButton button2 = new JButton("Two");
      private JButton button3 = new JButton("Three");
      private JButton button4 = new JButton("Four");
      private JPanel background = new JPanel();

     public TestFlowLayout()
     {
            background.setLayout(new FlowLayout());
            background.add(button1);
            background.add(button2);
            background.add(button3);
            background.add(button4);
            getContentPane().add(background);
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            pack();
    }
}
```

FlowLayout Manager:
Buttons are positioned from left to right as they are added. If you resize the window, the buttons are not resized

```
import javax.swing.*;
import java.awt.*;
public class TestBorderLayout extends JFrame
{
    private JButton buttonN = new JButton("North");
    private JButton buttonS = new JButton("South");
    private JButton buttonE = new JButton("East");
    private JButton buttonW = new JButton("West");
    private JButton buttonC = new JButton("Center");
    private JPanel background = new JPanel();

    public TestBorderLayout()
    {
        background.setLayout(newBorderLayout());
        background.add(buttonN, BorderLayout.NORTH);
        background.add(buttonS, BorderLayout.SOUTH);
        background.add(buttonE, BorderLayout.EAST);
        background.add(buttonW, BorderLayout.WEST);
        background.add(buttonC, BorderLayout.CENTER);
        getContentPane().add(background);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
   }
}
```

BorderLayout manager:
When we add components,
we specify a particular position.
Not suitable for buttons,
but is useful for
positioning panels of components.