Graphical User Interfaces 2

- In this section:
  - The Java event handling framework
  - Introduction to more widgets: sliders and labels
  - Drawing graphics
- More will appear through lab worksheets

Key components of Java event handling

- If a program is to be interactive with GUI buttons (`JButtons`):
- The program must "implement `ActionListener`", eg:

```
public class CarPark extends JFrame
                implements ActionListener
```

- We must declare one global variable for each button's details, eg:

```
private JButton doIt;
```

- In `createGUI` we must create the button, add it to the display, and register the program as listening for clicks on it, e.g.

```
doIt = new JButton("Click me");
window.add(doIt);
doIt.addActionListener(this);
```

The constructor has one parameter: the button's label

- The program *must* contain the appropriate event handling method:

```
public void actionPerformed(ActionEvent event)
{
    ...
}
```

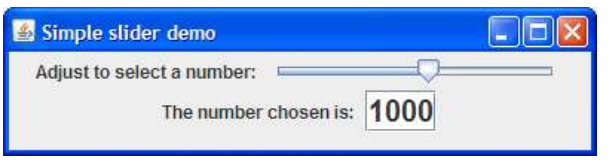
- The parameter of event handling methods brings information about the specific event into the method:
  - For `actionPerformed` the `event` (of type `ActionEvent`) contains the identity of the pressed GUI button
  - We can extract it using: `event.getSource()`
  - And we can use it like this:

```
if (event.getSource() == button variable name)
    action
```

- All interactive widgets follow the same scheme
  - Advanced Java also uses the scheme for interaction between parts of the same program

Introducing `JSlider`s – a simple input device

- Java Swing `JSlider` widgets allow the user to select a number in a set range by dragging a slider with the mouse
- The `SimpleSlider` application:



Demo

- The slider range is -200 to 2000 (in this example)
- The currently selected value is always displayed in a text field
- There is *one event* to consider: slider adjustment
  - The event handler is `stateChanged`
  - The action is to fetch the new setting and display in the text field

The `SimpleSlider` application

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
-----------------------------------------------
public class SimpleSlider extends Jframe
                    implements ChangeListener {
-----------------------------------------------
private JSlider slider;
private JTextField displayField;
-------------------------------------------------
public static void main (String[] args) {
     SimpleSlider frame = new SimpleSlider();
     frame.set ...;
     frame.createGUI();
     frame.setVisible(true);
}
```



```
private void createGUI() {
     setDefaultCloseOperation(EXIT_ON_CLOSE);
     Container window = getContentPane();
     window.setLayout(new FlowLayout() );
     ------------------------------------------------
     JLabel sliderLabel =
          new JLabel("Adjust to select a number: ");
     window.add(sliderLabel);
     ------------------------------------------------
     slider = new JSlider(JSlider.HORIZONTAL,
                               -200, 2000, 1000);
     window.add(slider);
     slider.addChangeListener(this);
     ------------------------------------------------
     JLabel displayLabel =
          new JLabel("The number chosen is: ");
     window.add(displayLabel);
     ------------------------------------------------
     displayField = new JTextField("1000");
     window.add(displayField);
}
```

- Finally the event handling method for ChangeListener:

```
public void stateChanged(ChangeEvent e) {
    int selectedNumber = slider.getValue();
    displayField.setText(
        Integer.toString(selectedNumber));
}
```

- Note:
  - Here, the `ChangeEvent` information is not used
  - `slider.getValue()` fetches the slider's new setting
  - `stateChanged` is called *frequently!*

`JLabel`s and `JSlider`s

- There are two `JLabel`s
  - A `JLabel` is a Swing GUI widget for displaying text
  - The JVM makes sure that the text is redrawn whenever necessary
  - No other interesting properties: no input, no events
  - `JLabel` is another Swing library class
  - The `JLabel` "constructor" has *one* parameter: the text to be displayed – there are other options
  - We create instances and add to the display in the usual way:

```
JLabel sliderLabel =
      new JLabel("Adjust to select a number: ")
window.add(sliderLabel);
```

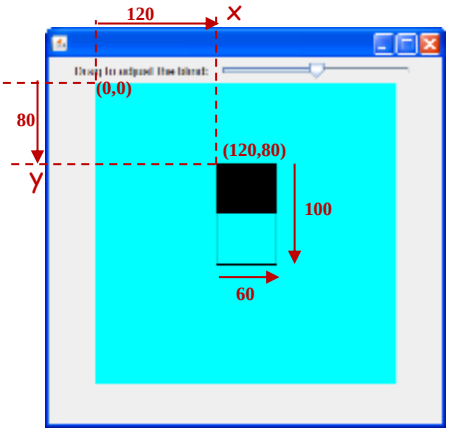
- Similarly, `JSlider` is a Swing GUI widget
  - Highly customizable – see API
  - Causes a `ChangeEvent` when adjusted
  - `JSlider` is a library class describing the functions of a slider
  - Instantiated and added to window in the usual way
- The `JSlider` constructor has 4 parameters
  - `JSlider.HORIZONTAL` (or `VERTICAL`)
  - Minimum, maximum and initial settings (-200, 2000, 1000 here)
- Need to register the program as wanting to be told about `slider`'s adjustment events (`ChangeEvent`s):

```
slider.addChangeListener(this);
```

- A `JSlider` can be interrogated to find out its setting:

```
int selectedNumber = slider.getValue();
```


A simple graphics application, with a slider



Demo

Coordinates are (x,y)

In terms of *pixels*

The *top left* panel corner is pixel coordinate (0,0)

- Note:
  - The blind is adjusted when the slider is dragged
  - The graphics drawing uses a `JPanel`

The Java source code Slide 1/4

```
import ...

public class WindowBlind extends JFrame
                      implements ChangeListener
{
    private JSlider slider;
    private JPanel panel;
    private int blindHeight = 50;

    public static void main (String[] args)
    {
        WindowBlind frame = new WindowBlind();
        frame.setSize(400,400);
        frame.setLocation(200,200);
        frame.createGUI();
        frame.setVisible(true);
    }
```


Slide 2/4

```
    private void createGUI()
    {
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        Container window = getContentPane();
        window.setLayout(new FlowLayout() );

        JLabel sliderLabel = new JLabel("Drag …");
        window.add(sliderLabel);

        slider = new JSlider(JSlider.HORIZONTAL,
                                        0, 100, 50);
        window.add(slider);
        slider.addChangeListener(this);


        continued...
```

Slide 3/4

How to "refresh" the panel

```
panel = new JPanel() {
   public void paintComponent(Graphics g)
   {
      super.paintComponent(g);
      paintScreen(g);
   }
};
panel.setPreferredSize(
                     new Dimension(300, 200));
panel.setBackground(Color.cyan);
window.add(panel);

} // end of creatGUI
```

This is more complex – use it as a "recipe"!

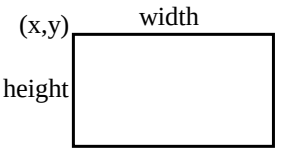
Slide 4/4

```
public void stateChanged(ChangeEvent e)
{
   blindHeight = slider.getValue();
   repaint();  // Forces a screen refresh
}

// Called from paintComponent in the JPanel
private void paintScreen(Graphics g)
{
   g.setColor(Color.black);
   g.drawRect(120, 80, 60, 100);
   g.fillRect(120, 80, 60, blindHeight);
}
```

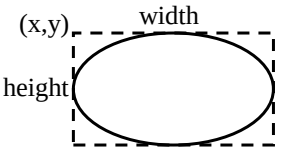
`Graphics` drawing methods (see API)


```
g.drawLine(x1,y1,x2,y2);
```




```
g.drawRect(x,y,width,height);
```
(also `fillRect`)

squares?




```
g.drawOval(x,y,width,height);
```
(also `fillOval`)

circles?

- All of these draw or fill using the "current colour"
  - Initially black

Changing the drawing colour

- There are colour names like:

  `Color.black` `Color.blue` `Color.red`

  - Note the upper/lower case and the spelling!
  - The colours can also be in upper case: `Color.RED`
- The *background colour* of the drawing panel can be changed with a call of the library method:

  `panel.setBackground(` colour name `);`
- During drawing, the current *drawing/filling* colour can be changed by:

  `g.setColor(` colour name `);`
  - In a *sequence* of drawing method calls, items drawn *after* the `setColor` have the new colour
  - `setColor` is rather like "change pen"

End of section