COMP3358: Tutorial 4
Java GUI

GUI packages

- java.awt.*
  - Native GUI components
- java.awt.event.*
  - Events handling
- javax.swing.*
  - High level GUI built on AWT

Simple GUI program

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


public class Demo {
    public static void main(String[] args) {
        JFrame frame = new JFrame("Demo");
        frame.add(new JButton("This is a JButton"));
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

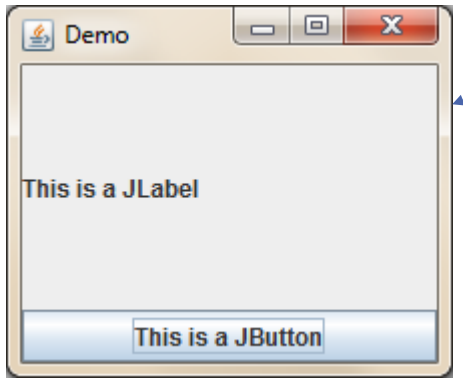
Terminate program when frame closed

Calculate size of frame

Show frame

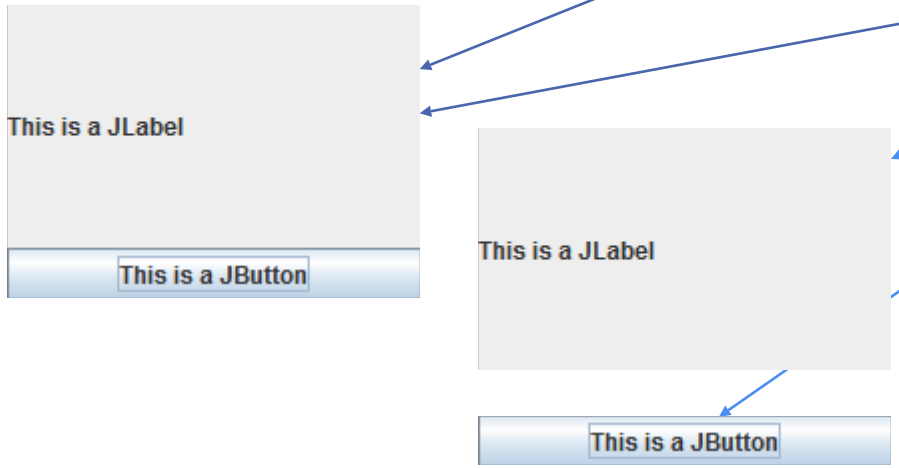
Container and component

Top-level containter (JFrame)



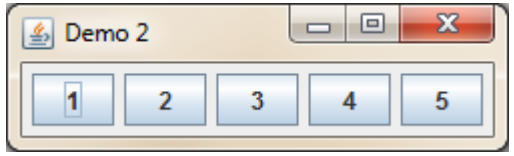
Container (JPanel)
Can be used to hold one or more components

Component (JLabel, JButton)
Container can also be a component

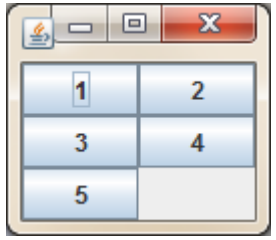


Layout

- **JFrame** use **BorderLayout** by default
- Layout can be set using `setLayout()` method
  - http://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html




```
JPanel panel = new JPanel();
panel.setLayout(new FlowLayout());
panel.add(new JButton("1"));
panel.add(new JButton("2"));
panel.add(new JButton("3"));
panel.add(new JButton("4"));
panel.add(new JButton("5"));
```




```
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(0,2));
panel.add(new JButton("1"));
panel.add(new JButton("2"));
panel.add(new JButton("3"));
panel.add(new JButton("4"));
panel.add(new JButton("5"));
```

Sizing

- Two way to control the sizing of GUI components
  - Absolute sizing using `setSize()`
    - Need to set the size of every component
    - Not flexible, resizing may not be possible
    - Some component/container will ignore `setSize()`
  - Optimal sizing using `setPreferredSize()` - more preferred
    - Allow Java to layout/resize component automatically

```
JFrame frame = new JFrame("Demo 3");
frame.setLayout(new GridLayout(0,2));
for(int i=1; i<=5; ++i) {
    JButton btn = new JButton(""+i);
    btn.setPreferredSize(new Dimension(70,70));
    frame.add(btn);
}
frame.pack();
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setVisible(true);
```

Preferred size of each button is 70x70

Resize frame to fit with the preferred button size

Events

- Events are handled by:
  - Register a listener to a event
  - Implement a listener to handle the event
    - http://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

```
JButton btn = new JButton("Click me");
btn.addActionListener(new BtnListener());
```

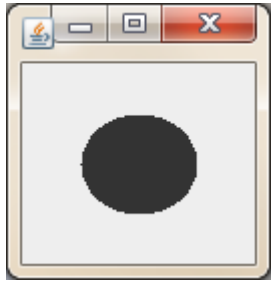
```
class BtnListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Hello!");
        }
}
```

- Alternatively, we could use anonymous object (not recommended)

```
JButton btn = new JButton("Click me");
btn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "Hello!");
        }
});
```

Custom painting

- We can extend existing GUI component and customize how it is painted in Java by:
  - Create a class extending the component
  - Overriding implementation of `paintComponent(Graphic g)`



```
class MyPanel extends JPanel {
        public MyPanel() {
                this.setPreferredSize(new Dimension(100,100));
        }
        public void paintComponent(Graphics g) {
                super.paintComponent(g);
                Dimension size = getSize();
                g.fillOval(size.width/4, size.height/4,
                                size.width/2, size.height/2);
        }

}
```

This allow the component to paint the background properly
Alternative you can paint the background yourself by drawing a big rectangle

Thread and GUI

- Ideally, a Java SWING program consists of three types of threads:
  - The initial **main thread**
    - Responsible to initiate the SWING application
  - **Event-dispatching thread**
    - Responsible to handle event
    - It should be the only thread that manipulate SWING components
  - **Worker threads**
    - Any other thread the runs in the background
    - Mostly manipulating data in the application

Demo: DemoThreadGUI.java

```
SwingUtilities.invokeLater(new Runnable() {
        public void run() {
                generateGUI();
        }
});
```

`generateGUI()` will manipulate GUI components, so ideally it should be done in the event-dispatching thread.
`SwingUtilities.invokeLater()` will queue the execution for you.

```
public void mouseClicked(MouseEvent event) {
        x = event.getX();
        y = event.getY();
        repaint();
}
```

`mouseClicked()` will be executed in the event dispatching thread.
Action here must be finished quickly or otherwise the thread will be occupied and GUI becomes irresponsive.

Animation

- Suppose the last demo is modified to create an animation…

```
public void mouseClicked(MouseEvent event) {
        int targetX = event.getX();
        int targetY = event.getY();
        for(int i=0;i<10;++i) {
                x = (x+targetX)/2;
                y = (y+targetY)/2;
                repaint();
                try {
                        Thread.sleep(100);
                } catch (InterruptedException e) { }
        }
}
```

Move towards target 10 times

Do once every 0.1s

Does it work? Why?

Worker

- A thread is needed in the previous example

```
public void mouseClicked(MouseEvent event) {
        new Animation(this, event).start();
}
```

Start a thread in background to release the event dispatching thread

```
class Animation extends Thread {
        /* … */
        public void run() {
                int targetX = event.getX();
                int targetY = event.getY();

                for(int i=0;i<10;++i) {
                        panel.x = (panel.x+targetX)/2;
                        panel.y = (panel.y+targetY)/2;
                        panel.repaint();
                        try {
                                Thread.sleep(100);
                        } catch (InterruptedException e) {
                        }
                }
        }
}
```

Call repaint to update GUI in event dispatching thread

For more intensive tasks, we can use SwingWorker instead

Exercise

- Modify **`DemoThreadGUI.java`** and complete the animation as shown in the last two slides
- **Please do the programming assignments on your virtual machine. Submit the code you have modified and a document to Moodle. The doc should contain the highlight of the code you modified (part I) and the screen shots of the GUI (part II). We show an example of part II in gif (note that gif is not mandatory, screenshots are acceptable):**