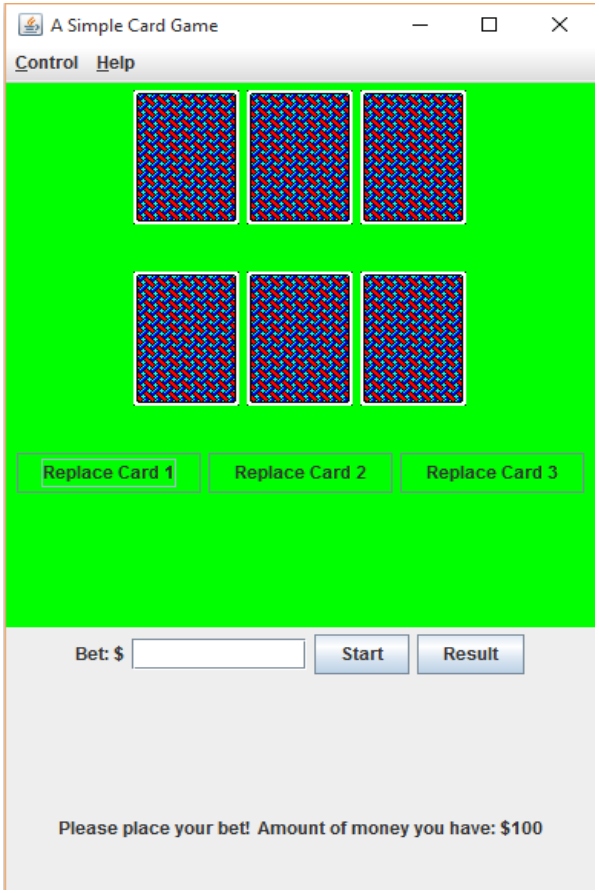
Chapter 9.

GUI and Event-Handling

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong

It All Starts with a Window

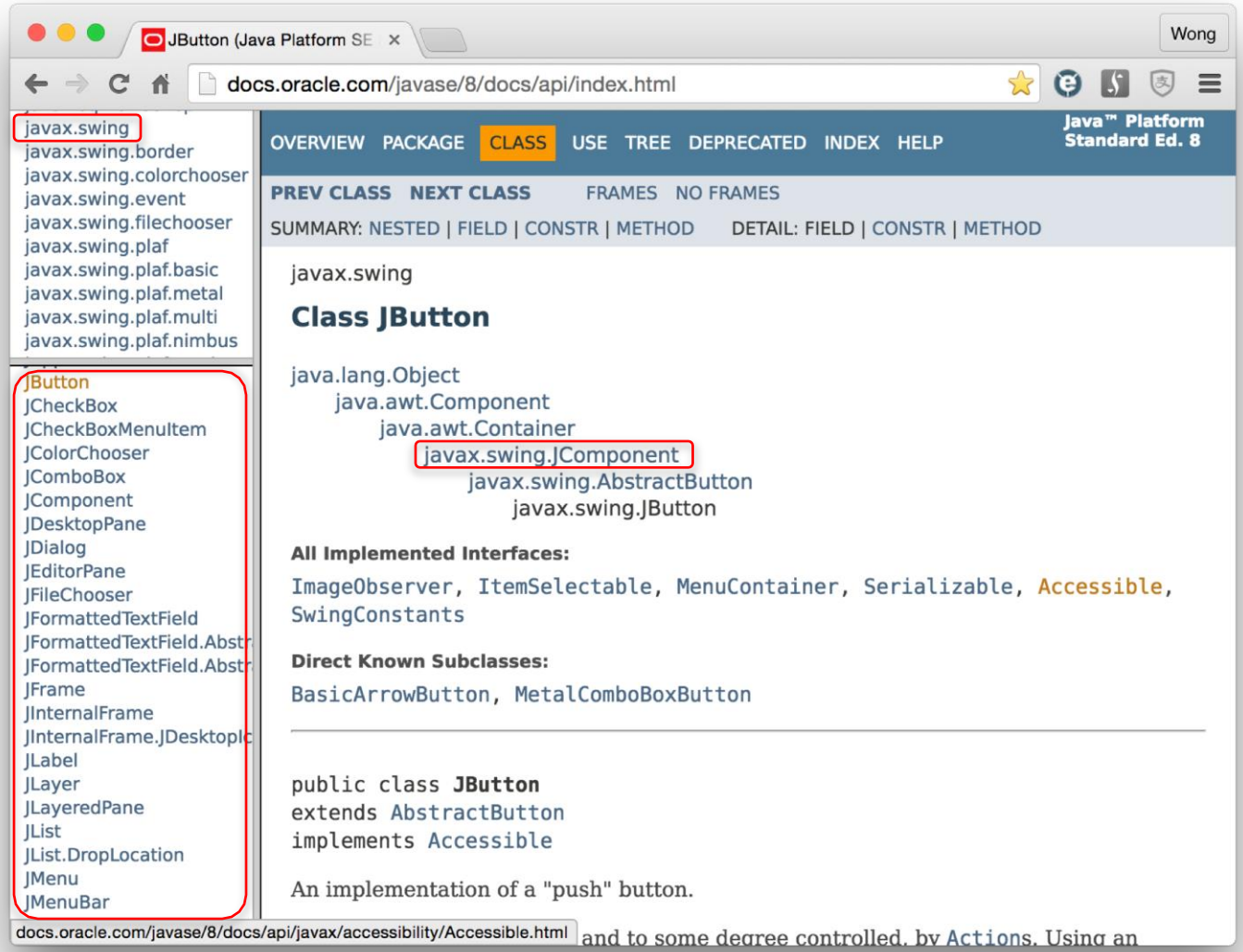
- A JFrame is an object that represents a window on the screen
- It is where you put all the interface things ('widgets') like buttons, checkboxes, text fields, and so on
- It can have a menu bar with menu items, and have all the little windowing icons for minimizing, maximizing and closing the window
- Once you have a JFrame, you can put widgets in it by adding them to the JFrame



Swing Components

- A widget is technically a Swing component
- You can find tons of Swing components from the javax.swing package
- The most common ones include JButton, JRadioButton, JCheckBox, JLabel, JTextArea, JTextField, JList, JScrollPane, JSlider, JMenuBar, JMenu, JMenuItem, etc.
- Almost all Swing components extend from javax.swing.JComponent

Swing Components



Swing Components

- In Swing, virtually all components are capable of holding other components
- Most of the time, however, you will add user interactive components (e.g., buttons and lists) into background components (e.g., frames and panels)
- With the exception of JFrame, the distinction between interactive components and background components is artificial (e.g., a JPanel can also be interactive, and can handle events like mouse clicks and keystrokes)

Making a GUI is Easy

- 4 simple steps in making a GUI
  1. Create a frame

```
JFrame frame = new JFrame();
```

  2. Create a widget (e.g., button, text field, etc.)

```
JButton button = new JButton("click me");
```

  3. Add the widget to the frame

```
frame.add(button);
```

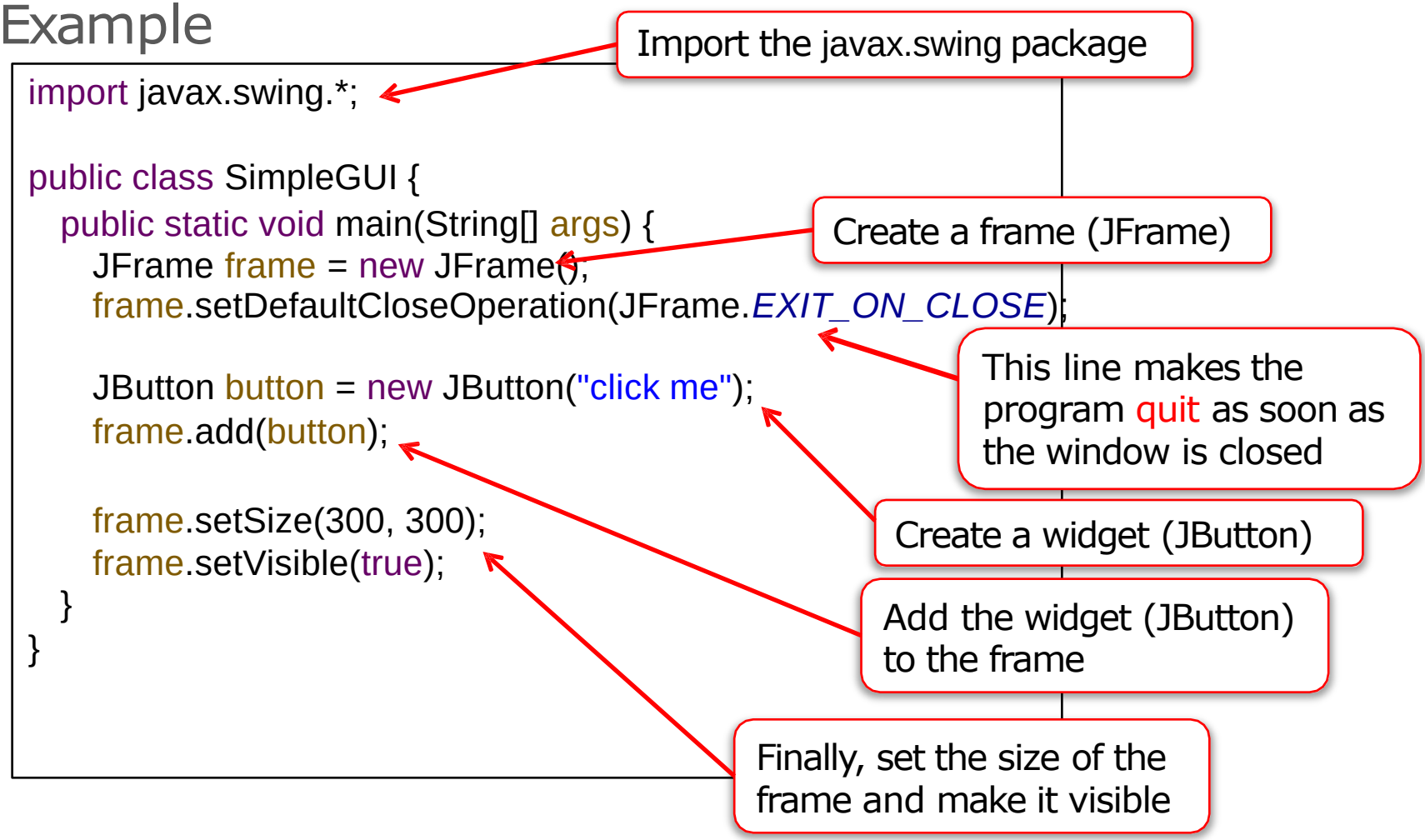
  4. Display it (give it a size and make it visible)

```
frame.setSize(300, 300);
frame.setVisible(true);
```

My First GUI

Example

```
import javax.swing.*;

public class SimpleGUI {
   public static void main(String[] args) {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JButton button = new JButton("click me");
      frame.add(button);

      frame.setSize(300, 300);
      frame.setVisible(true);
   }
}
```



My First GUI

- Sample output

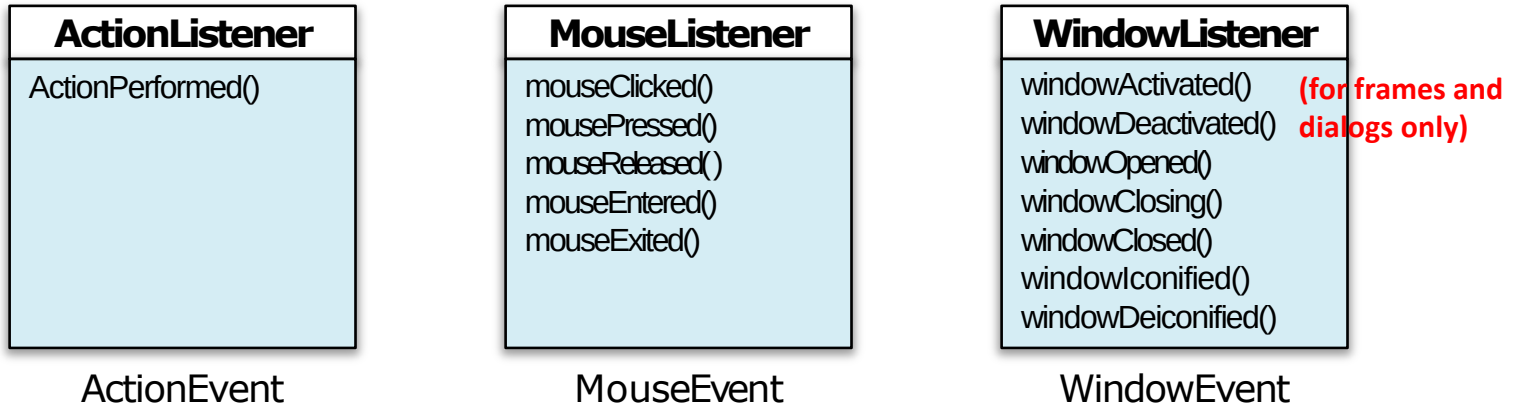
The button fills all the available space in the frame. Nothing happens when the button is being clicked.

Event-Handling

- In Java, the processing of getting and handling a user action is called event-handling
- There are many different event types, most of which involve GUI user actions
- The Swing GUI components are event sources (i.e., objects that can turn user actions into events)
- In Java, an event is represented as an object of some event class (e.g., ActionEvent, MouseEvent, WindowEvent, KeyEvent, etc., check the java.awt.event package for more event classes)
- An event source (e.g., a button) creates an event object when the user does something that matters (e.g., clicking the button)

Event-Handling

- To handle an event, implement a listener interface
- Every event type has a matching listener interface (e.g., implement the ActionListener to handle ActionEvent, the MouseListener to handle MouseEvent, and the WindowListener to handle WindowEvent)
- Some interfaces have more than one method because the event itself comes in different flavors
- Example



Event-Handling

- A class that implements a listener interface is known as a listener
- Before a listener can receive events from an event source, it must first register itself with the event source
- This can be done by calling a registration method on the event source and providing a reference to the listener as an argument
- The registration methods always take the form of add<EventType>Listener() (e.g., addActionListener(), addMouseListener(), addWindowListener())
- The listener must provide implementations for the event-handling methods (aka event handlers) from the listener interface

Event-Handling

Example

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

public class SimpleGUI2 implements ActionListener {
   JButton button;

   public static void main(String[] args) {
      SimpleGUI2 gui = new SimpleGUI2();
      gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Import the java.awt.event package

Implement the ActionListener interface

Event-Handling

Example

```
        button = new JButton("click me");
        button.addActionListener(this);

        frame.add(button);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {
        button.setText("I've been clicked!");
    }
}
```

Register this listener with the button

Implement the actionPerformed() method from the ActionListener interface. This is the actual event-handling method

Event-Handling

- Sample output

Event-Handling

- Summary
  - A listener
    - Implements a listener interface
    - Registers itself with an event source
    - Provides implementations for the event handlers
  - An event source
    - Accepts registrations from listeners
    - Gets user actions and creates event objects
    - Calls the event handlers of the listeners
  - An event object
    - Carries information of the event to the listener (e.g., the screen coo*rdina*tes of the mouse in a MouseEvent)

Making a Drawing Panel

- To put your own graphics on the screen
  1. Make a subclass of JPanel
  2. Override the paintComponent() method and put all your graphics code inside this method
  3. Create an instance of your drawing panel and add it to the frame (just like adding a button or any other widgets)
- Whenever the frame holding your drawing panel is displayed or refreshed, your paintComponent() method will be called and your graphics will be drawn on the screen
- Note that you never call this method yourself! (The argument to this method is a Graphics object which is the actual drawing canvas that gets slapped onto the real display, and you cannot get this by yourself!)

Making a Drawing Panel

- Example

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

public class MyDrawPanel extends JPanel {
   public void paintComponent(Graphics g) {
      g.setColor(Color.ORANGE);
      g.fillRect(20, 50, 100, 70);
   }
}
```

Make a subclass of JPanel

Override the paintComponent() method

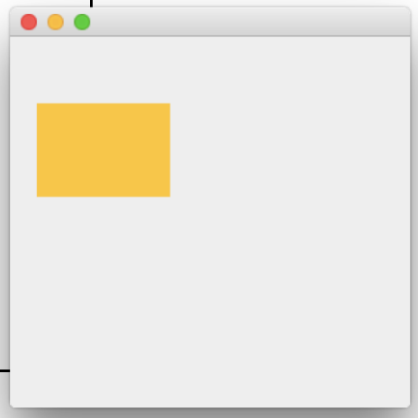
Set the color to orange

Draw a filled rectangle at (20, 50) with a dimension of 100 (width) by 70 (height)

Making a Drawing Panel

Example

```
import javax.swing.*;

public class MyDrawPanelTestDrive {
   public static void main(String[] args) {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      MyDrawPanel drawPanel = new MyDrawPanel();
      frame.add(drawPanel);

      frame.setSize(300, 300);
      frame.setVisible(true);
   }
}
```



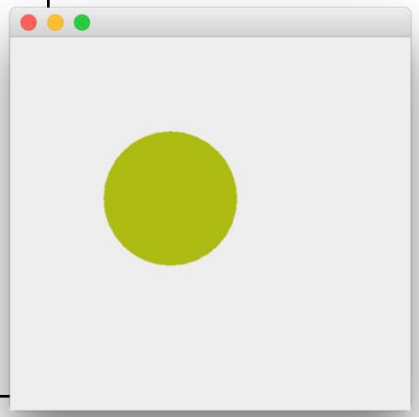
Making a Drawing Panel

- Example: Display a JPEG

```
public void paintComponent(Graphics g) {
   Image image = new ImageIcon("batman.jpg").getImage();
   g.drawImage(image, 3, 4, this);
}
```

- Example: Paint a randomly-colored circle

```
public void paintComponent(Graphics g) {
   int red = (int) (Math.random() * 256);
   int green = (int) (Math.random() * 256);
   int blue = (int) (Math.random() * 256);
   g.setColor(new Color(red, green, blue));
   g.fillOval(70, 70, 100, 100);
}
```



Graphics2D Object

- The argument to the paintComponent() is actually an instance of the Graphics2D class (a subclass of Graphics)
- A Graphics2D object can do more than a Graphics object

Graphics
drawImage()
drawLine()
drawPolygon()
drawRect()
drawOval()
fillRect()
fillOval()
setColor()
…

Graphics2D
rotate()
scale()
shear()
transform()
setPaint()
setRenderingHints()
hit()
fill()
…

- Cast a Graphics reference to a Graphics2D object to a Graphics2D reference when you need to use methods from the Graphics2D class

```
Graphics2D g2D = (Graphics2D) g;
```

Graphics2D Object

Example

```
public void paintComponent(Graphics g) {
   Graphics2D g2D = (Graphics2D) g;
   GradientPaint paint = new GradientPaint(70, 70, Color.BLUE,
                                           150, 150, Color.ORANGE);
   g2D.setPaint(paint);  g2D.fillOval(70, 70, 100, 100);
}
```

GradientPaint

```
public GradientPaint(float x1,
                     float y1,
                     Color color1,
                     float x2,
                     float y2,
                     Color color2)
```

Constructs a simple acyclic `GradientPaint` object.

Parameters:

`x1` - x coordinate of the first specified `Point` in user space

`y1` - y coordinate of the first specified `Point` in user space

`color1` - `Color` at the first specified `Point`

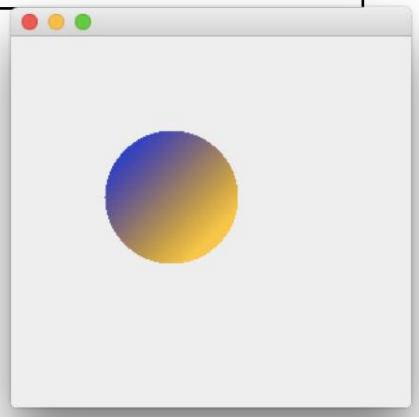
`x2` - x coordinate of the second specified `Point` in user space

`y2` - y coordinate of the second specified `Point` in user space

`color2` - `Color` at the second specified `Point`

Throws:

`NullPointerException` - if either one of colors is null

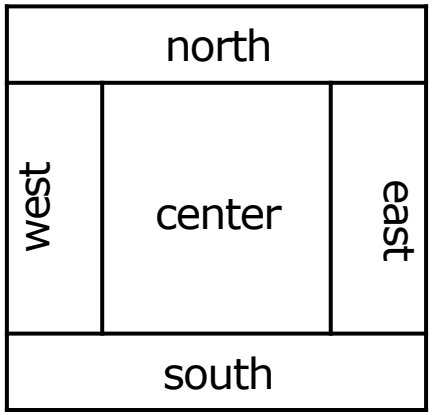


Example: A Color-Changing Circle

- Summary
  - Build a frame with 2 widgets, namely a drawing panel and a button
  - Create a listener and register it with the button
  - When the user clicks the button, the button creates an event object and calls the listener's event handler
  - The event handler calls `repaint()` on the frame, causing the system to call `paintComponent()` on the drawing panel
  - Each time `paintComponent()` runs, it draws a circle filled with a random color on the screen

Example: A Color-Changing Circle

- By default, a frame has 5 regions, namely east, south, west, north and center
- Only 1 widget can be added to each region of a frame, but a widget itself might be a panel which can hold other widgets
- Specify a region when adding a widget to a frame by calling the 2-argument add() method, e.g.,




```
frame.add(widget, BorderLayout.EAST);
frame.add(widget, BorderLayout.SOUTH);
frame.add(widget, BorderLayout.WEST);
frame.add(widget, BorderLayout.NORTH);
frame.add(widget, BorderLayout.CENTER);
```

When calling the single-argument add(), the widget will be added to the center region

Example: A Color-Changing Circle

Example

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

class MyDrawPanel2 extends JPanel {
   public void paintComponent(Graphics g) {
      int red = (int) (Math.random() * 256);
      int green = (int) (Math.random() * 256);
      int blue = (int) (Math.random() * 256);
      g.setColor(new Color(red, green, blue));
      g.fillOval(70, 70, 100, 100);
   }
}
```

Example: A Color-Changing Circle

Example

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

public class SimpleGUI3 implements ActionListener {
   JFrame frame;

   public static void main(String[] args) {
      SimpleGUI3 gui = new SimpleGUI3();
      gui.go();
   }

   public void go() {
      frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

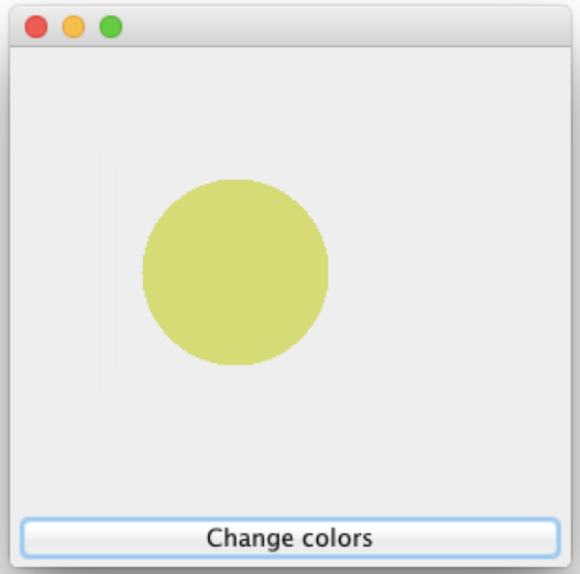
Example: A Color-Changing Circle

Example

```
        JButton button = new JButton("Change colors");
        button.addActionListener(this);

        MyDrawPanel2 drawPanel = new MyDrawPanel2();

        frame.add(button, BorderLayout.SOUTH);
        frame.add(drawPanel, BorderLayout.CENTER);
        frame.setSize(300, 300);
        frame.setVisible(true);
    }

    public void actionPerformed(ActionEvent event) {
        frame.repaint();
    }
}
```

Example: A Color-Changing Circle

- Sample output



Example: A GUI with 2 Buttons

- Summary
  - Continue with the previous example
  - Add 1 button to the west region of the frame
  - Add 1 label to the east region of the frame
  - When the user clicks the button on the west, the label on the east will be changed

How to handle action events for 2 different buttons when each button needs to do something different?

Example: A GUI with 2 Buttons

- Option 1: Implement 2 actionPerformed() methods

```
class MyGUI implements ActionListener {
   // lots of code here...

   public void actionPerformed(ActionEvent event) {
      frame.repaint();
   }

   public void actionPerformed(ActionEvent event) {
      label.setText("That hurts!");
   }
}
```

This won't compile!

It is not possible to implement the same method twice in a Java class. Even if you could, how would the event source know which of the 2 methods to call?

Example: A GUI with 2 Buttons

- Option 2: Check the event object to determine the event source and hence what to do

```
class MyGUI implements ActionListener {
   // lots of code here...

   public void go() {
      colorButton = new JButton();
      labelButton = new JButton();
      colorButton.addActionListener(this);
      labelButton.addActionListener(this);
      // more GUI code here...
   }

   public void actionPerformed(ActionEvent event) {
      if (event.getSource() == colorButton) { frame.repaint(); }
      else { label.setText("That hurts!"); }
   }
}
```

Register the same listener with both buttons

This does work. However, having 1 event handler doing many different things is not very OO

Example: A GUI with 2 Buttons

- Option 3: Create 2 separate ActionListener classes

```
class MyGUI {
   JFrame frame;
   JLabel label;
   // more GUI code here...
}
```

```
class ColorButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent event) {
      frame.repaint();
   }
}
```

These classes do not have access to the instance variables frame and label of MyGUI

```
class LabelButtonListener implements ActionListener {
   public void actionPerformed(ActionEvent event) {
      label.setText("That hurts!");
   }
}
```

This can be fixed by giving each listener a reference to the main GUI class and providing getters for the GUI widgets, but this gets messier and more complicated

Inner Class

- An inner class is defined inside the curly braces of another class (the outer class)
- An inner class can use all the methods and instance variables of the outer class, including the private ones, just as if they were declared within the inner class (and vice-versa)
- An inner class object must be tied to an outer class objects on the heap that creates it
- The inner class object can access only the methods and instance variables of the outer class object that it is tied to, but not any other outer class objects
- Code in an outer class can instantiate its own inner classes in exactly the same way it instantiates another class

Inner Class

- Example

```
class MyOuterClass {
   private int x;
   MyInnerClass inner = new MyInnerClass();

   public void doStuff() {
      inner.go();
   }

   class MyInnerClass {
      private void go() {
         x = 42;
      }
   } // close inner class
} // close outer class
```

Create an instance of the inner class

Use 'x' as if it were declared within the inner class

Example: A GUI with 2 Buttons

Example

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

public class TwoButtons {
   JFrame frame;
   JLabel label;

   public static void main(String[] args) {
      TwoButtons gui = new TwoButtons();
      gui.go();
   }

   public void go() {
      frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Example: A GUI with 2 Buttons

- Example

```
    JButton labelButton = new JButton("Change Label");
    labelButton.addActionListener(new LabelListener());
    JButton colorButton = new JButton("Change Circle");
    colorButton.addActionListener(new ColorListener());

    label = new JLabel("I'm a label");
    MyDrawPanel2 drawPanel = new MyDrawPanel2();

    frame.add(colorButton, BorderLayout.SOUTH);
    frame.add(drawPanel, BorderLayout.CENTER);
    frame.add(labelButton, BorderLayout.WEST);
    frame.add(label, BorderLayout.EAST);

    frame.setSize(500, 300);
    frame.setVisible(true);
}
```

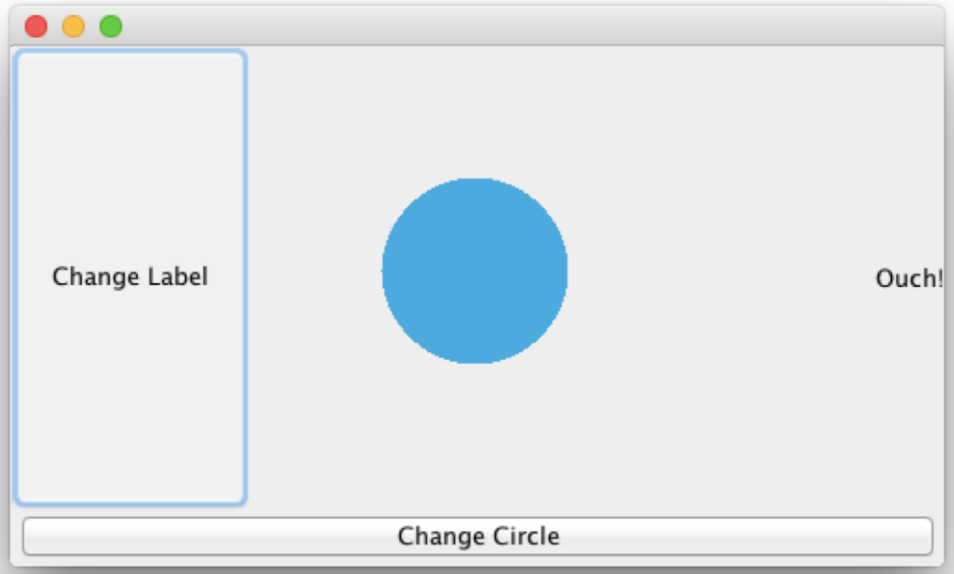
Example: A GUI with 2 Buttons

- Example

```
    class LabelListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            label.setText("Ouch!");
        }
    }   // close inner class

    class ColorListener implements ActionListener {
        public void actionPerformed(ActionEvent event) {
            frame.repaint();
        }
    }   // close inner class
}
```

Example: A GUI with 2 Buttons

- Sample output



Example: A Simple Animation

Example

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

public class SimpleAnimation {
   int x = 70;
   int y = 70;

   public static void main(String[] args) {
      SimpleAnimation gui = new SimpleAnimation();
      gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Example: A Simple Animation

Example

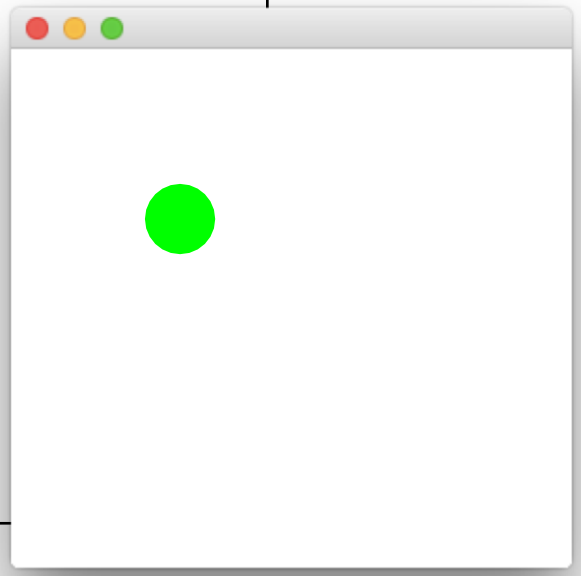
```
    MyAnimPanel animPanel = new MyAnimPanel();

    frame.add(animPanel);
    frame.setSize(300, 300);
    frame.setVisible(true);

    for (int i = 0; i < 130; i++) {
       x++;
       y++;
       animPanel.repaint();

       try {
          Thread.sleep(50);
       } catch (Exception ex) { }
    }
  } // close go() method
```

The sleep() method may throw an exception (i.e., fail at runtime), and must be called within a try/catch block

Example: A Simple Animation

Example

```
   class MyAnimPanel extends JPanel {
      public void paintComponent(Graphics g) {
         g.clearRect(0, 0, this.getWidth(), this.getHeight());
         g.setColor(Color.GREEN);
         g.fillOval(x, y, 40, 40);
      }
   } // close inner class
} // close outer class
```



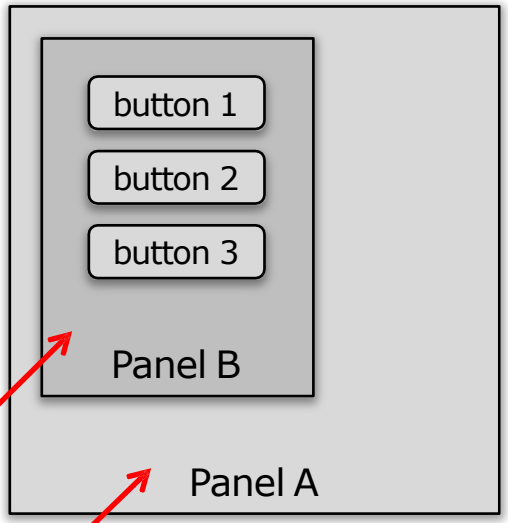
Layout Managers

- A layout manager is a Java object associated with a particular component
- The layout manager controls the size and placement of the components contained within the component the layout manager is associated with
- Each background component can have its own layout manager
- There are different kinds of layout managers, each has its own polices to follow when building a layout
- For example, one layout manager might insist that all components in a panel must be the same size and arranged in a grid, while another layout manager might let each component choose its own size but stack them vertically

Layout Managers

Example

```
JPanel panelA = new JPanel();
JPanel panelB = new JPanel();
panelB.add(new JButton("button 1");
panelB.add(new JButton("button 2");
panelB.add(new JButton("button 3");
panelA.add(panelB);
```



The layout manager of Panel B controls the size and placement of the 3 buttons

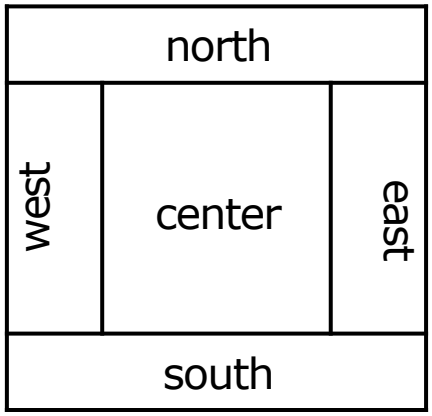
The layout manager of Panel A controls the size and placement of Panel B, but has nothing to say about the 3 buttons. The hierarchy of control is 1 level only!

Layout Managers

- Commonly used layout managers
  - BorderLayout
  - FlowLayout
  - BoxLayout
  - GridBagLayout

BorderLayout

- Divides a background component into 5 regions
- Only 1 component can be added to each region
- Components usually do not get to have their preferred size
- Default layout manager for a frame



BorderLayout

- Example: Adding a button to the east region

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

public class BorderLayoutEx {
   public static void main(String[] args) {
      BorderLayoutEx gui = new BorderLayoutEx();
      gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JButton button = new JButton("click me");
      frame.add(button, BorderLayout.EAST);
      frame.setSize(200, 200);
      frame.setVisible(true);
   }
}
```

Adding the button to the east region

BorderLayout

- Example: Adding more characters to the button

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JButton button = new JButton("click like you mean it");
   frame.add(button, BorderLayout.EAST);
   frame.setSize(200, 200);
   frame.setVisible(true);
}
```

Changing only the text on the button

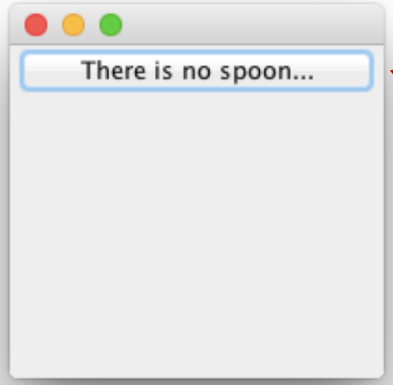
A component in the east/west region of a border layout gets its preferred width, but not height (it will be as tall as the frame)

BorderLayout

- Example: Adding a button to the north region

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JButton button = new JButton("There is no spoon...");
   frame.add(button, BorderLayout.NORTH);
   frame.setSize(200, 200);
   frame.setVisible(true);
}
```

Adding the button to the north region



A component in the north/south region of a border layout gets its preferred height, but not width (it will be as wide as the frame)

BorderLayout

- Example: Making the button taller

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JButton button = new JButton("Click this!");
   Font bigFont = new Font("serif", Font.BOLD, 28);
   button.setFont(bigFont);
   frame.add(button, BorderLayout.NORTH);
   frame.setSize(200, 200);
   frame.setVisible(true);
}
```

A bigger font will force the frame to allocate more space for the button's height

The width stays the same, but now the button is taller. The north region is stretched to accommodate the new preferred height of the button

BorderLayout

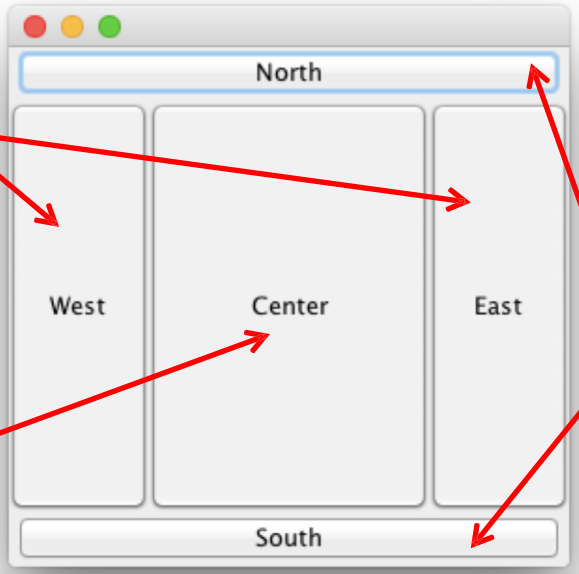
- Example: Adding a button to each of the 5 regions

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JButton east = new JButton("East");
   JButton west = new JButton("West");
   JButton north = new JButton("North");
   JButton south = new JButton("South");
   JButton  center =  new JButton("Center");

   frame.add(east, BorderLayout.EAST);
   frame.add(west, BorderLayout.WEST);
   frame.add(north, BorderLayout.NORTH);
   frame.add(south, BorderLayout.SOUTH);
   frame.add(center, BorderLayout.CENTER);

   frame.setSize(300, 300);
   frame.setVisible(true);
}
```

BorderLayout

—Sample output

Components in the east and west get their preferred width



Components in the north and south get their preferred height

The component in the center gets whatever space is left over based on the frame dimension

When you put something in the north or south, it goes all the way across the frame, so the things in the east and west won't be as tall as they would be if the north and south regions were empty

FlowLayout

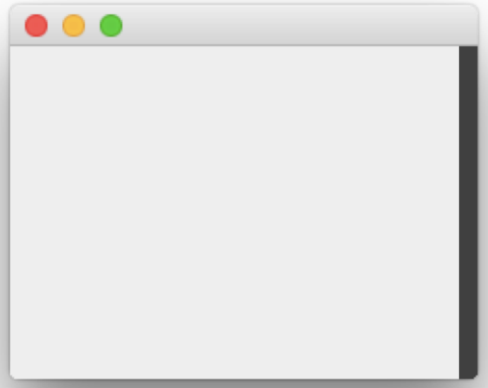
- Acts kind of like a word processor, except with components instead of words
- Each component gets to have its own size
- Components are laid out left to right in the order that they are added
- When a component does not fit horizontally, it drops to the next "line" in the layout
- Default layout manager for a panel

FlowLayout

- Example: Adding a panel to the east region

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

public class FlowLayoutEx {
   public static void main(String[] args) {
      FlowLayoutEx gui = new FlowLayoutEx();
      gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setBackground(Color.DARK_GRAY);
      frame.add(panel, BorderLayout.EAST);
      frame.setSize(250, 200);
      frame.setVisible(true);
   }
}
```

The default layout manager of a panel is FlowLayout. When a panel is added to a frame, its size and placement is still controlled by the layout manager of the frame (i.e., BorderLayout)

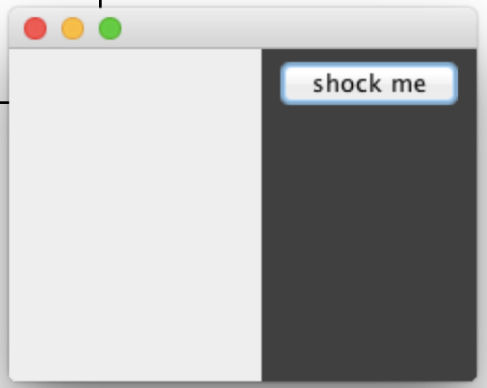


FlowLayout

- Example: Adding a button to the panel

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JPanel panel = new JPanel();
   panel.setBackground(Color.DARK_GRAY);
   JButton button = new JButton("shock me");
   panel.add(button);
   frame.add(panel, BorderLayout.EAST);
   frame.setSize(250, 200);
   frame.setVisible(true);
}
```

Adding the button to the panel



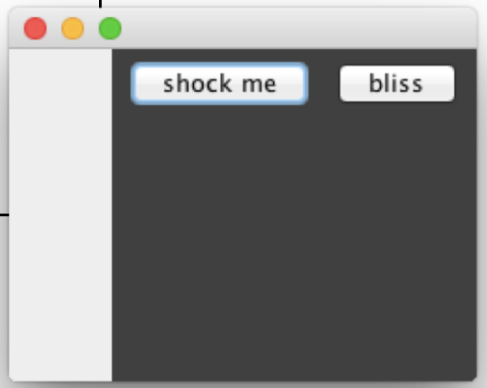
The panel expands and the button gets its preferred size in both dimensions because panel uses FlowLayout and the button is part of the panel (not the frame)

FlowLayout

- Example: Adding 2 buttons to the panel

```
public void go() {
   JFrame frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   JPanel panel = new JPanel();
   panel.setBackground(Color.DARK_GRAY);
   JButton button = new JButton("shock me");
   JButton button2 = new JButton("bliss");
   panel.add(button);
   panel.add(button2);
   frame.add(panel, BorderLayout.EAST);
   frame.setSize(250, 200);
   frame.setVisible(true);
}
```

Adding 2 buttons to the panel



The panel expands to fit both buttons side by side. Notice that the 'bliss' button is smaller than the 'shock me' button. That's how FlowLayout works

BoxLayout

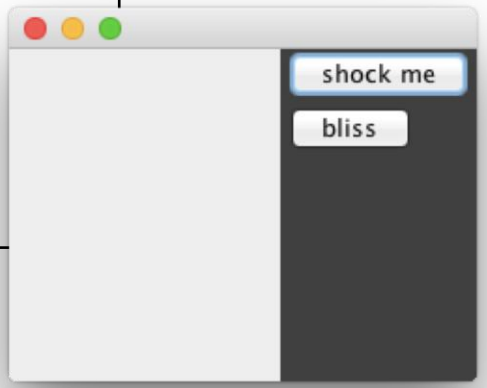
- Like a FlowLayout in that each component gets to have its own size, and the components are placed in the order in which they are added
- Can stack the components either vertically or horizontally

BoxLayout

Example: Stacking the buttons vertically

```
public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setBackground(Color.DARK_GRAY);
      panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
      JButton button = new JButton("shock me");
      JButton button2 = new JButton("bliss");
      panel.add(button);
      panel.add(button2);
      frame.add(panel, BorderLayout.EAST);
      frame.setSize(250, 200);
      frame.setVisible(true);
}
```

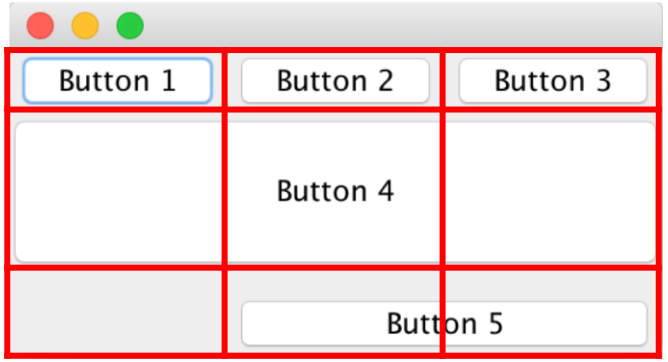
Change the layout manager to a new instance of BoxLayout



Notice how the panel is narrower again, because it does not need to fit both buttons horizontally

GridBagLayout

- One of the most flexible and complex layout managers
- Components are placed in a grid of rows and columns
- Rows can be of different heights
- Columns can be of different widths
- Components can span multiple rows or columns
- The size and placement of a component in the grid is specified by a GridBagConstraints object



GridBagConstraints

```
public GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight,
                          double weightx, double weighty, int anchor, int fill,
                          Insets insets, int ipadx, int ipady)
```

- Parameters
  - gridx – specifies the cell containing the leading edge of the component's display area
  - gridy – specifies the cell at the top of the component's display area
  - gridwidth – specifies the number of cells in a row for the component's display area
  - gridheight – specifies the number of cells in a column for the component's display area
  - weightx – specifies how to distribute extra horizontal space
  - weighty – specifies how to distribute extra vertical space

For further details, please refer to
https://docs.oracle.com/javase/8/docs/api/java/awt/GridBagConstraints.html

GridBagConstraints

```
public GridBagConstraints(int gridx, int gridy, int gridwidth, int gridheight,
                          double weightx, double weighty, int anchor, int fill,
                          Insets insets, int ipadx, int ipady)
```

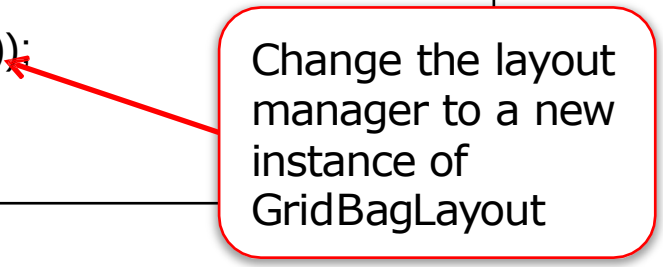
- Parameters
  - anchor – specifies how to place the component when it is smaller than its display area
  - fill – specifies how to resize the component when it is smaller than its display area
  - insets – specifies the external padding of the component
  - ipadx, ipady – specifies the internal padding of the component

For further details, please refer to
https://docs.oracle.com/javase/8/docs/api/java/awt/GridBagConstraints.html

GridBagLayout

- Example:

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

public class GridBagLayoutEx {
   public static void main(String[] args) {
      GridBagLayoutEx gui = new GridBagLayoutEx();
      gui.go();
   }

   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JPanel panel = new JPanel();
      panel.setLayout(new GridBagLayout());
```



GridBagLayout

- Example:

```
GridBagConstraints c = new GridBagConstraints();
c.gridx = 0;
c.gridy = 0;
c.gridwidth = 1; // default value
c.gridheight = 1; // default value
c.weightx = 0.0; // default value
c.weighty = 0.0; // default value
c.anchor = GridBagConstraints.CENTER; // default value
c.fill = GridBagConstraints.HORIZONTAL;
c.insets = new Insets(0, 0, 0, 0); // default value
c.ipadx = 0; // default value
c.ipady =  0; // default value

JButton button = new JButton("Button 1");
c.weightx = 0.5;
panel.add(button, c);
```

GridBagLayout

- Example:

```
button = new JButton("Button 2");  c.gridx = 1;
// 2nd column
panel.add(button, c);

button = new JButton("Button 3");  c.gridx = 2;
// 3rd column
panel.add(button, c);

button = new JButton("Button 4");  c.gridx = 0;
// 1st column
c.gridy = 1; // 2nd row
c.gridwidth = 3; // spans 3 columns
c.weightx = 0.0;
c.ipady = 40; // makes the button tall
panel.add(button, c);
```

GridBagLayout

- Example:

```
        button = new JButton("Button 5");  c.gridx =
        1; // 2nd column
        c.gridy = 2; // 3rd row
        c.gridwidth = 2; // spans 2 columns
        c.weighty = 1.0; // takes up extra vertical space
        c.anchor = GridBagConstraints.SOUTH;
        c.insets = new Insets(10, 0, 0, 0); // top padding
        c.ipady = 0;
        panel.add(button, c);

        frame.add(panel);
        frame.pack();
        frame.setVisible(true);
    }
}
```

So complicated…
Don't worry, we are with you!

If you encounter any problems in understanding the materials in the lectures, **please feel free to contact me or my TAs**.
We are always with you!

We wish you enjoy learning Java in this class.

Chapter 9.

End

2020-2021
COMP2396 Object-Oriented Programming and Java
Dr. T.W. Chim (E-mail: twchim@cs.hku.hk)
Department of Computer Science, The University of Hong Kong