2023/2024(1)
EF234302 Object Oriented Programming

Lecture #9b

GUI Programming

Misbakhul Munir IRFAN SUBAKTI
司馬伊凡
Мисбакхул Мунир Ирфан Субакти

Inner classes

- Java supports *inner classes*, i.e., classes defined inside other classes
- Inner classes are often used to defined listeners for events

MyActionEvent3.java

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4  public class MyActionEvent3 extends JPanel {
5      JButton darken = new JButton("Darken"),
6              lighten = new JButton("Lighten");
7      int red = 255, green = 255, blue = 255;
8      private class DarkenListener implements ActionListener {
9          public void actionPerformed(ActionEvent e) {
10             int delta = 0;
11             if (e.getSource() == lighten) {
12                 delta = 10;
13             } else if (e.getSource() == darken) {
14                 delta = -10;
15             }
16             red = checkBoundary(red + delta);
17             green = checkBoundary(green + delta);
18             blue = checkBoundary(blue + delta);
19             repaint();
20         }
21     }
```

Inner classes (continued)

- The inner class can access the instance variables of the outer class (e.g., `darken`, `lighten`, `red`, `green`, `blue`) as well as the methods (e.g., `repaint()`);
- An instance of the inner class can be supplied as the listener for events

```
22    public MyActionEvent3() {
23        add(darken);
24        darken.addActionListener(new DarkenListener());
25        add(lighten);
26        lighten.addActionListener(new DarkenListener());
27    }
```

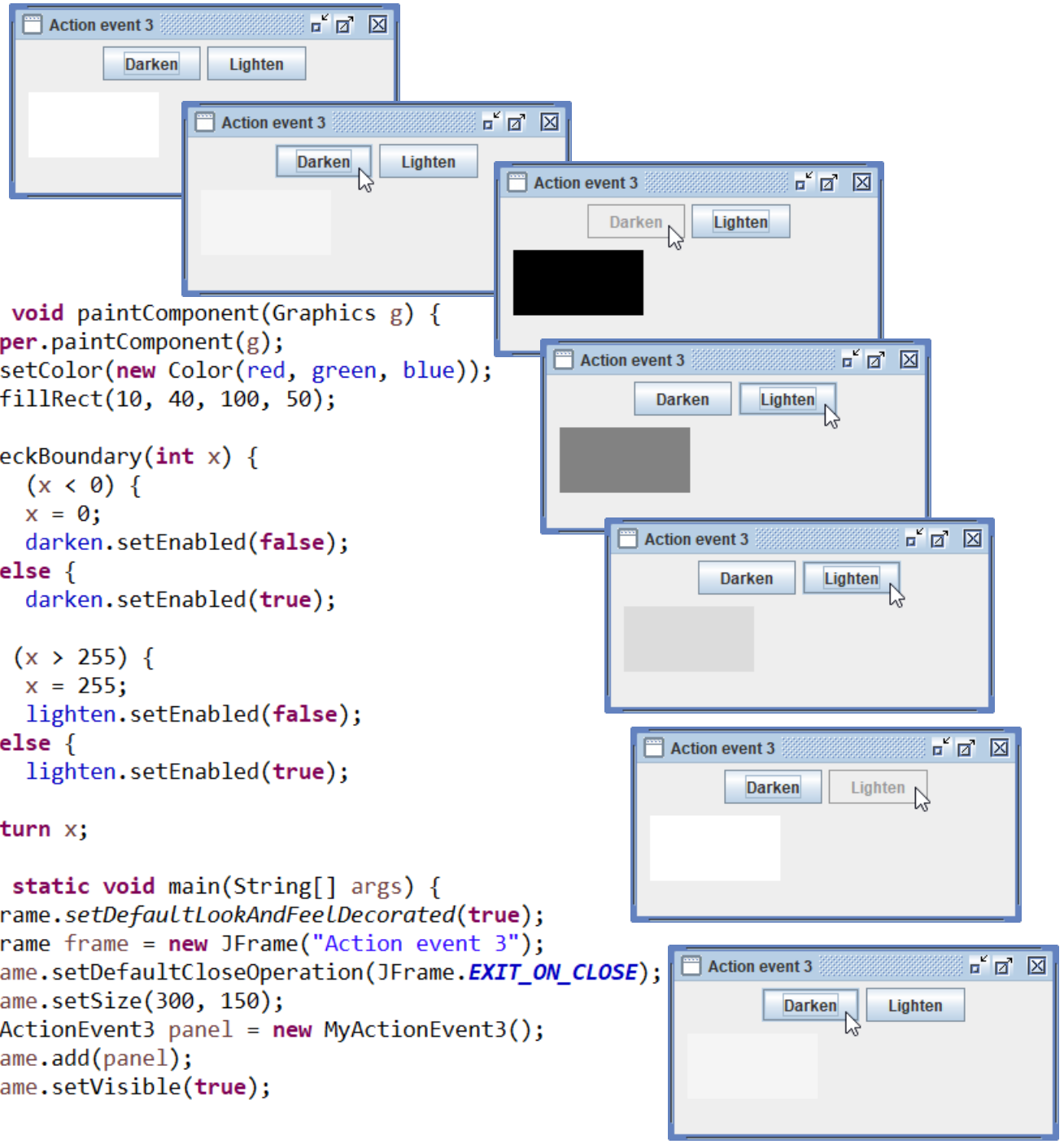
Why use inner classes?

- We can handle each event separately in a self-contained method, e.g.,
  - `DarkenListener.actionPerformed`
  - `LightenListener.actionPerformed`
- Leads to better structure when a class has to implement listeners for a large number of events

Inner classes: Example

MyActionEvent3.java

```
1  import java.awt.*;
2  import javax.swing.*;
3  import java.awt.event.*;
4  public class MyActionEvent3 extends JPanel {
5      JButton darken = new JButton("Darken"),
6              lighten = new JButton("Lighten");
7      int red = 255, green = 255, blue = 255;
8      private class DarkenListener implements ActionListener {
9          public void actionPerformed(ActionEvent e) {
10             int delta = 0;
11             if (e.getSource() == lighten) {
12                 delta = 10;
13             } else if (e.getSource() == darken) {
14                 delta = -10;
15             }
16             red = checkBoundary(red + delta);
17             green = checkBoundary(green + delta);
18             blue = checkBoundary(blue + delta);
19             repaint();
20         }
21     }
22     public MyActionEvent3() {
23         add(darken);
24         darken.addActionListener(new DarkenListener());
25         add(lighten);
26         lighten.addActionListener(new DarkenListener());
27     }
28     public void paintComponent(Graphics g) {
29         super.paintComponent(g);
30         g.setColor(new Color(red, green, blue));
31         g.fillRect(10, 40, 100, 50);
32     }
33     int checkBoundary(int x) {
34         if  (x < 0) {
35             x = 0;
36             darken.setEnabled(false);
37         } else {
38             darken.setEnabled(true);
39         }
40         if (x > 255) {
41             x = 255;
42             lighten.setEnabled(false);
43         } else {
44             lighten.setEnabled(true);
45         }
46         return x;
47     }
48     public static void main(String[] args) {
49         JFrame.setDefaultLookAndFeelDecorated(true);
50         JFrame frame = new JFrame("Action event 3");
51         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
52         frame.setSize(300, 150);
53         MyActionEvent3 panel = new MyActionEvent3();
54         frame.add(panel);
55         frame.setVisible(true);
56     }
57 }
```



Keyword: `this`

- Any object can refer to ***itself*** using the keyword `this`

```
this.x
this.setX(10.0)
button.addActionListener(this)
```

- Note that a **superclass** can call a **subclass** method (due to *overriding*)

Animal.java

```java
package animal;
public class Animal {
    // Information hiding = encapsulation
    // This class property set to be private -> cannot be accessed
    // directly. Only can be accessed by getter(s) & setter(s)
    private String name = "No name"; // Default name: No name
    private int age = 0; // Default age: 0
    private boolean vaccinated = false; // Default: false
    private double price = 10.0; // Default: 10.0

    // Overloading: a class has more than one method of
    // the same name & their parameter(s) are different

    // Constructor: special case of function, where its name
    // is the same with the class' name
    Animal() { // Overloading
        System.out.println("Animal: Default constructor");
    }
    Animal(String name) { // Overloading
        this.name = name;
        System.out.println("Constructor with name");
    }
    Animal(boolean vaccinated) { // Overloading
        this.vaccinated = vaccinated;
        System.out.println("Constructor with vaccinated status");
    }
    Animal(String name, int age) { // Overloading
        this.name = name;
        this.age = age;
        System.out.println("Constructor with name & age");
    }
```

Cat.java

```java
package animal;
public class Cat extends Animal {
    // Overriding: same method name & same parameter(s)
    // By using overriding, Run Time Polymorphism can be achieved

    @Override
    public void sound() { // Polymorphism
        System.out.println("Cat says: meow");
    }

    // An example of Cat's method/function
    public void catMethod() {
        System.out.println("Cat: catMethod");
    }
}
```

AnimalTest.java

```java
        System.out.print("Showing that a superclass can  call ");
        System.out.println("a subclass method");
        System.out.println("-----------------------------------------------------");
        System.out.println("a1 is defined by superclass: Animal, instantiated by superclass: Animal");
        Animal a1 = new Animal();
        System.out.println("b1 is defined by subclass: Cat, instantiated by subclass: Cat");
        Cat b1 = new Cat();
        System.out.println("b1 calls the Cat's method: catMethod()");
        b1.catMethod();
//      System.out.println("Try: a1 calls the subclass' method: catMethod() by casting to Cat");
//      ((Cat) a1).catMethod(); // Exception will happen
        System.out.println("c1 is defined by superclass: Animal, instantiated by subclass: Cat");
        Animal c1 = new Cat();
//      c1.catMethod(); // The subclass' method: catMethod() is undefined for the type Animal
        ((Cat) c1).catMethod(); // Need to cast, so that an instance of superclass can call
                                // the subclass' method
    }
}
```

Keyword: `super`

- Keyword `super` refers to the superclass instance ***implicit*** inside the current object

```
28      public void paintComponent(Graphics g) {
29          super.paintComponent(g);
30          g.setColor(new Color(red, green, blue));
31          g.fillRect(10, 40, 100, 50);
32      }
```

Subclass `constructor`

- A constructor of a subclass always has an implicit call to the superclass' constructor at the beginning

```
3  public class MyRedSquare extends JPanel {
4      public MyRedSquare() {
5          // super(); // Implicitly
6          setPreferredSize(new Dimension(100, 100));
7          setBackground(Color.white);
8      }
```

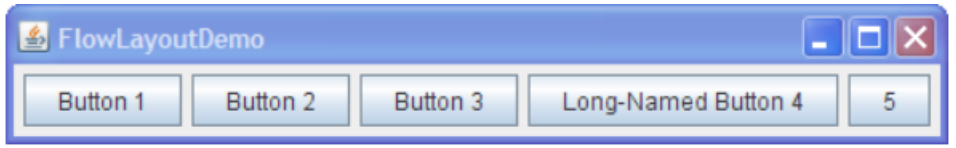
- However, if there are arguments to the superclass constructor, need to call it explicitly

Layout

- Java GUI library is designed to be flexible
- Create interfaces for multiple platforms, screen sizes, window sizes, font sizes, international languages, etc.
- This requires a Layout Manager for each GUI panel
- Comes into picture whenever the panel needs to be laid out
- See Java tutorial for quick intro

  https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

FlowLayout



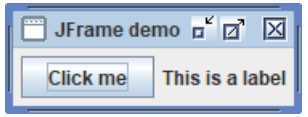
- The default layout manager for a panel
- Places components in top-to-bottom, left-to-right order, like a word processor
- Components are kept at their preferred sizes as far as possible
  - No stretching or shrinking
- Often components are centered horizontally

MyPanel.java

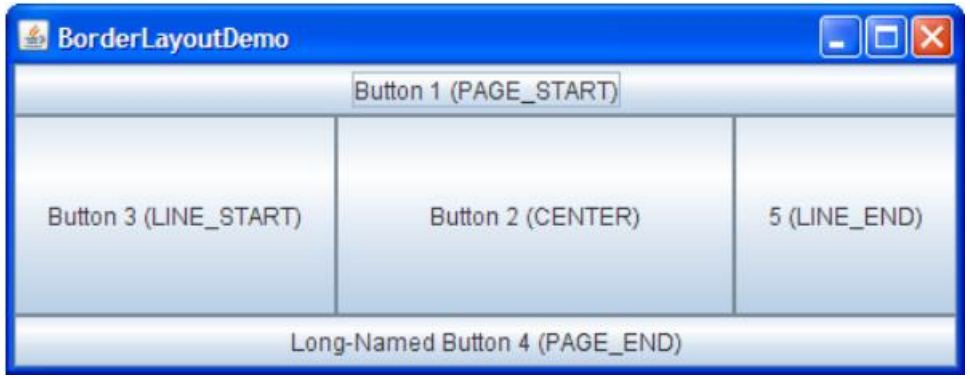
```
1  import javax.swing.*;
2  public class MyPanel extends JPanel {
3      int count = 0; // The state
4      JButton button = new JButton("Click me");
5      JLabel label = new JLabel("This is a label");
6      public MyPanel() {
7          add(button);
8          add(label);
9      }
10 }
```

MyPanelTest.java

```
1  import javax.swing.*;
2  public class MyPanelTest {
3      public static void main(String[] args) {
4          JFrame.setDefaultLookAndFeelDecorated(true);
5          JFrame frame = new JFrame("JFrame demo");
6          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
7          MyPanel panel = new MyPanel();
8          frame.setContentPane(panel);
9          frame.pack();
10         frame.setVisible(true);
11     }
12 }
```



BorderLayout



- Default layout for JFrames. For other containers, set it:

```
container.setLayout(new BorderLayout());
```

- A BorderLayout container contains exactly 5 components: `PAGE_START`, `PAGE_END`, `LINE_START`, `LINE_END`, and `CENTER`

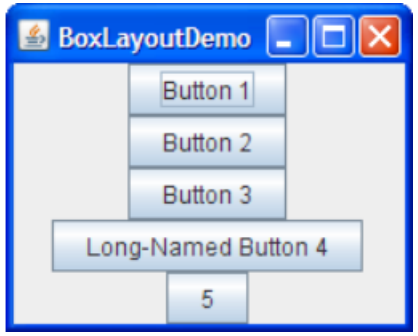
```
container.add(component, BorderLayout.CENTER);
```

- Of course, any of these components can be a panel that groups together smaller components
- Those 5 components stretch to fill all available space

MyContentPane2.java

```
1 import java.awt.*; // BorderLayout, etc.
2 import javax.swing.*; // JFrame, JLabel, etc.
3 public class MyContentPane2 {
4     public static void main(String[] args) {
5         JFrame.setDefaultLookAndFeelDecorated(true);
6         JFrame frame = new JFrame("JFrame demo");
7         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8         JPanel pane = new JPanel(new BorderLayout());
9         JLabel labelOnePiece = new JLabel("One piece world");
10        labelOnePiece.setIcon(new ImageIcon("res/one_piece_world.gif"));
11        pane.add(labelOnePiece, BorderLayout.PAGE_START);
12        pane.add(new JLabel("Hello world!"));
13        pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
14        frame.setContentPane(pane);
15        frame.pack();
16        frame.setVisible(true);
17    }
18 }
```

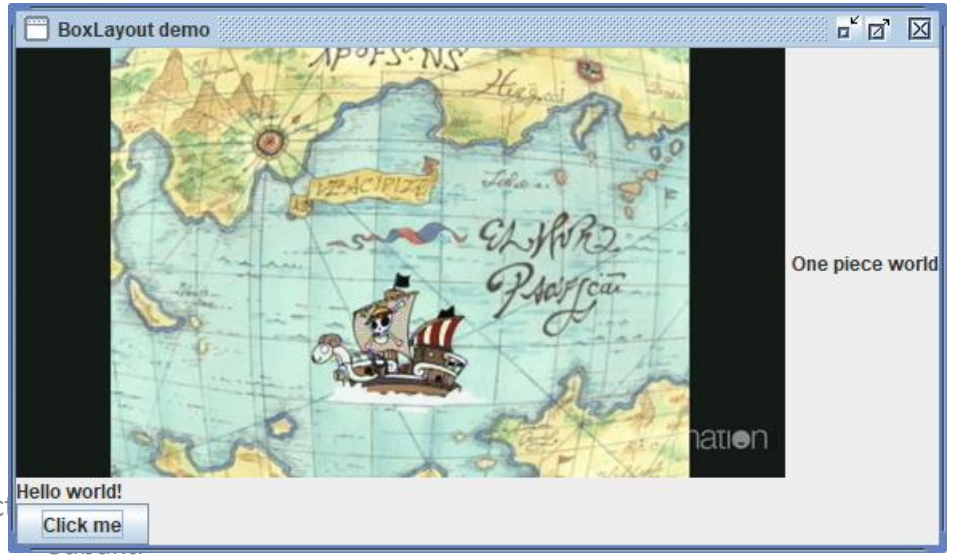
BoxLayout



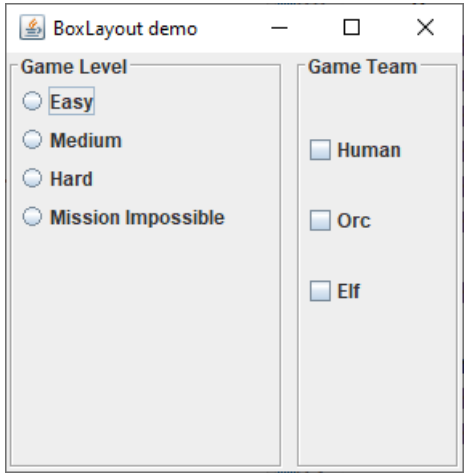
- A box is an array of components laid out horizontally or vertically
- Use it via the Box container: `Box b = Box.createVerticalBox();`
- Components in a box can be *aligned*: left, right or center (center by default)
  - Similarly, top, bottom or center for horizontal boxes

MyBoxLayout.java

```
1 import java.awt.*; // BoxLayout, etc.
2 import javax.swing.*; // JFrame, JLabel, etc.
3 public class MyBoxLayout {
4     public static void main(String[] args) {
5         JFrame.setDefaultLookAndFeelDecorated(true);
6         JFrame frame = new JFrame("BoxLayout demo");
7         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8         JPanel pane = new JPanel();
9         pane.setLayout(new BoxLayout(pane, BoxLayout.Y_AXIS));
10        JLabel labelOnePiece = new JLabel("One piece world");
11        labelOnePiece.setIcon(new ImageIcon("res/one_piece_world.gif"));
12        pane.add(labelOnePiece, BorderLayout.PAGE_START);
13        pane.add(new JLabel("Hello world!"));
14        pane.add(new JButton("Click me"), BorderLayout.PAGE_END);
15        frame.setContentPane(pane);
16        frame.pack();
17        frame.setVisible(true);
18    }
19 }
```



BoxLayout (continued)

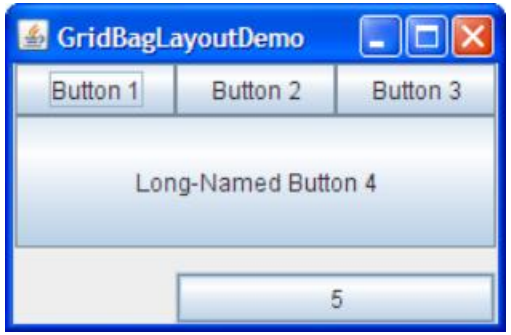


MyBoxLayout2.java

```java
import java.awt.*;
import javax.swing.*;
import javax.swing.border.EtchedBorder;
import javax.swing.border.TitledBorder;
public class MyBoxLayout2 {
  public static void main(String[] args) {
    JFrame frame = new JFrame("BoxLayout demo");
    frame.setBounds(200, 200, 300, 300);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    Box left = Box.createVerticalBox();
    ButtonGroup radioGroup = new ButtonGroup();
    JRadioButton rbutton;
    radioGroup.add(rbutton = new JRadioButton("Easy"));
    left.add(rbutton);
    radioGroup.add(rbutton = new JRadioButton("Medium"));
    left.add(rbutton);
    radioGroup.add(rbutton = new JRadioButton("Hard"));
    left.add(rbutton);
    radioGroup.add(rbutton = new JRadioButton("Mission Impossible"));
    left.add(rbutton);

    left.add(Box.createGlue());

    JPanel leftPanel = new JPanel(new BorderLayout());
    leftPanel.setBorder(new TitledBorder(new EtchedBorder(), "Game Level"));
    leftPanel.add(left, BorderLayout.CENTER);

    Box right = Box.createVerticalBox();
    right.add(Box.createVerticalStrut(30));
    right.add(new JCheckBox("Human"));
    right.add(Box.createVerticalStrut(20));
    right.add(new JCheckBox("Orc"));
    right.add(Box.createVerticalStrut(20));
    right.add(new JCheckBox("Elf"));

    right.add(Box.createGlue());

    JPanel rightPanel = new JPanel(new BorderLayout());
    rightPanel.setBorder(new TitledBorder(new EtchedBorder(), "Game Team"));
    rightPanel.add(right, BorderLayout.CENTER);

    Box top = Box.createHorizontalBox();
    top.add(leftPanel);
    top.add(Box.createHorizontalStrut(5));
    top.add(rightPanel);

    Container content = frame.getContentPane();
    content.setLayout(new BorderLayout());
    content.add(top, BorderLayout.CENTER);

    BoxLayout box = new BoxLayout(content, BoxLayout.Y_AXIS);

    content.setLayout(box);
    content.add(top);

    frame.setVisible(true);
  }
}
```

GridBagLayout

- The most flexible layout manager in Swing
- Imagine a panel arranged as a grid of invisible “cells”
- A component can occupy any rectangular area of cells
- Within its rectangular area, a component can either stretch or not, aligned or not, padded or not → lots of options



GridBagConstraints

- To specify all these options, Swing provides another class called `GridBagConstraints`

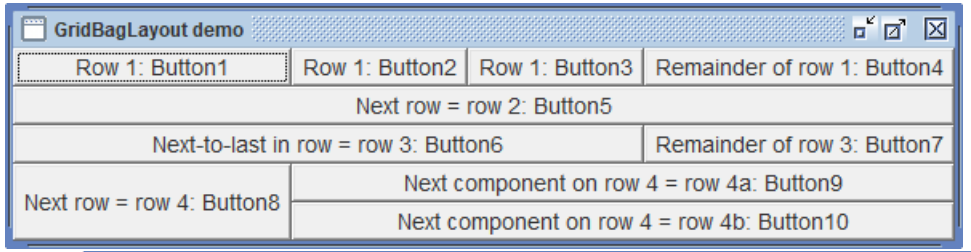
```
panel.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
c.gridx = 0; c.gridy = 0; // Position
c.gridwidth = 1; c.gridheight = 3; // Size
c.weightx = 100; c.weighty = 100; // Stretch
JList style = new JList();
panel.add(style, c);
```

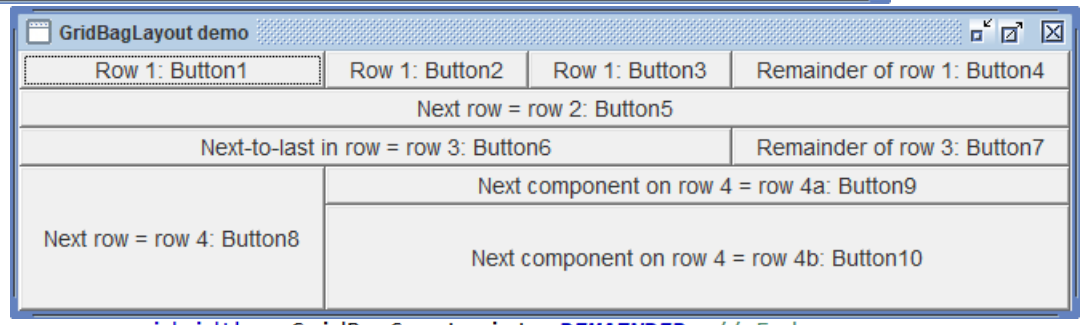
GridBagLayout: Programming

- Don’t use GridBagLayout class directly
- Define subclasses of JPanel with appropriate layout built-in. Example:

```
public class GridBagPanel extends JPanel {
   GridBagConstraints gbc = new GridBagConstraints();
   public GridBagPanel() {
      setLayout(new GridBagLayout());
      ... // Set default constraints
   }
   public add(Component c, int x1, int y1, int x2, int y2) {
   ...
   }
```

GridBagLayout: Example





MyGridBag.java

```java
import java.awt.*;
import javax.swing.*;
public class MyGridBag extends JPanel {
    protected void makeButton(String name, GridBagLayout gridbag, GridBagConstraints c) {
        Button button = new Button(name);
        gridbag.setConstraints(button, c);
        add(button);
    }
    public void init() {
        GridBagLayout gridbag = new GridBagLayout();
        GridBagConstraints c = new GridBagConstraints();
        setFont(new Font("SansSerif", Font.PLAIN, 14));
        setLayout(gridbag);
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        // If the space within a panel is greater than the preferredDimension of the
        // components contained within, the weightx and weighty is used to distribute the
        // extra space to the individual components.
        // E.g., use values from 0.0 to 1.0, or any value (think of this a percentage).
        // weightx is horizontal spacing, weighty is vertical spacing
        // The most common scenario is that the side panes stay a fixed size
        // (weightx/weighty = 0.0) and the center pane takes up the remaining space
        // (weightx/weighty = 1.0).
        // The extra space is allocated per the percentage of the individual components
        // weight divided by the total weight of all components in the row or column
        // E.g., if two buttons have a weightx of 0.2 and 0.8 one of the button will get
        // 20% and the other one 80% of the space
        makeButton("Row 1: Button1", gridbag, c);
        makeButton("Row 1: Button2", gridbag, c);
        makeButton("Row 1: Button3", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; // End row
        makeButton("Remainder of row 1: Button4", gridbag, c);
        c.weightx = 0.0; // Reset to the default
        makeButton("Next row = row 2: Button5", gridbag, c); // Another row
        c.gridwidth = GridBagConstraints.RELATIVE; // Next-to-last in row
        makeButton("Next-to-last in row = row 3: Button6", gridbag, c);
        c.gridwidth = GridBagConstraints.REMAINDER; // End row
        makeButton("Remainder of row 3: Button7", gridbag, c);
        c.gridwidth = 1; // Reset to the default
        c.gridheight = 2;
        c.weighty = 1.0;
        makeButton("Next row = row 4: Button8", gridbag, c);
        c.weighty = 0.0; // Reset to the default
        c.gridwidth = GridBagConstraints.REMAINDER; // End row
        c.gridheight = 1; // Reset to the default
        makeButton("Next component on row 4 = row 4a: Button9", gridbag, c);
        makeButton("Next component on row 4 = row 4b: Button10", gridbag, c);
    }
    public static void main(String args[]) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        JFrame frame = new JFrame("GridBagLayout demo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        MyGridBag gridBag = new MyGridBag();
        gridBag.init();
        frame.add("Center", gridBag); // Center, North, East, South, West
        frame.pack();
        frame.setVisible(true);
    }
```