A. Kompetensi

Setelah mengikuti praktikum ini, mahasiswa diharapkan mampu menerapkan Jpanel dan BoxLayout dalam pembuatan program.

B. Dasar Teori

JPanel

JPanel is a generic lightweight container: a generic Swing container object that can contain other Swing and AWT components. Components added to a container are tracked in a list. The order of the list will define the components' front-to-back stacking order within the container. If no index is specified when adding a component to a container, it will be added to the end of the list (and hence to the bottom of the stacking order).

Common Public Constructors:

JPanel ()

Constructs a new blank JPanel.

Common Public Methods:

- ✓ *void paint (Graphics g)*
  Overrides the paint method of JPanel.
- ✓ *void repaint ()*
  Repaints this component, and causes a call to the paint method.
- ✓ *void setLayout (LayoutManager Manager)*
  Sets the layout manager for the panel.
- ✓ *Component add (Component Item)*
  Appends the specified component to the end of this container.
- ✓ *void add (Component Item, Object Constraints)*
  Appends the specified component with the specified constraints.

BoxLayout Class

A layout manager that allows GUI components to be arranged left-to-right or top-to-bottom in a container. Class `Box` declares a container with `BoxLayout` as its default layout manager and provides `static` methods to create a `Box` with a horizontal or vertical `BoxLayout`.

- ➢ Constructor
  - ✓ *public BoxLayout (Container target, int axis)*
    The only constructor for this layout manager; it takes as input the container to manage and how the components should be laid out: left to right (`X_AXIS`) or top to bottom (`Y_AXIS`).
  - ✓ *Box(int)*

Creates a `Box` -- a lightweight container that uses a `BoxLayout` with the specified alignment (`BoxLayout.X_AXIS` or `BoxLayout.Y_AXIS`). Note that a `Box` is *not* a `JComponent` -- it's implemented as a subclass of `Container`. This makes it as lightweight as possible, but it lacks `JComponent` features such as borders. If you want a simple `JComponent` as a container, use `JPanel`.

- Constants

The `BoxLayout` class contains the two constants listed in Table 1.

Table 1. BoxLayout constants

Constant	Type	Description
X_AXIS	int	Used with the constructor to create a manager that lays out components along a horizontal axis. This constant can also be used with the constructor for the `Box` class.
Y_AXIS	int	Used with the constructor to create a manager that lays out components along a vertical axis. This constant can also be used with the constructor for the `Box` class.

- Method

Method	Purpose
`createHorizontalBox()`	Creates a `Box` that lays out its components from left to right.
`createVerticalBox()`	Creates a `Box` that lays out its components from top to bottom.
`createRigidArea(Dimension)`	Create a rigid lightweight component.

`createHorizontalGlue()`
`createVerticalGlue()`
`createGlue()`
	Create a glue lightweight component. Horizontal glue and vertical glue can be very useful.

`createHorizontalStrut()` `createVerticalStrut()`	Create a "strut" lightweight component. We recommend using rigid areas instead of struts.
`Box.Filler(Dimension, Dimension, Dimension)`	Creates a lightweight component with the specified minimum, preferred, and maximum sizes (with the arguments specified in that order). See the custom `Box.Filler` discussion, earlier in this section, for details.

The `Box` class defines a nested class, `Box.Filler` that provides invisible components. The `Box` class provides convenience methods to help you create

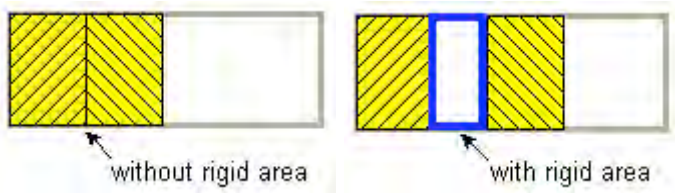
common kinds of filler. The following table gives details about creating invisible components with `Box` and `Box.Filler`.

Type		Size Constraints	How to Create
rigid area			`Box.createRigidArea(size)`
glue	horizontal		`Box.createHorizontalGlue()`
	vertical		`Box.createVerticalGlue()`
custom `Box.Filler`		*(as specified)*	`new Box.Filler(minSize, prefSize, maxSize)`

Here's how you generally use each type of filler:

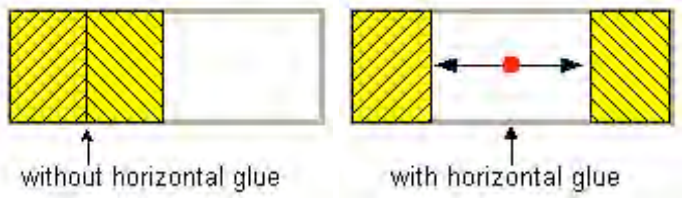
- **Rigid area**
  Use this when you want a fixed-size space between two components. For example, to put 5 pixels between two components in a left-to-right box, you can use this code:

```
container.add(firstComponent);
container.add(Box.createRigidArea(new Dimension(5,0)));
container.add(secondComponent);
```



- **Glue**
  Use this to specify where excess space in a layout should go. Think of it as semi-wet glue -- stretchy and expandable, yet taking up no space unless you pull apart the components that it's sticking to. For example, by putting horizontal glue between two components in a left-to-right box, you make any extra space go between those components, instead of to the right of all the components. Here's an example of making the space in a left-to-right box go between two components, instead of to the right of the components:

```
container.add(firstComponent);
container.add(Box.createHorizontalGlue());
container.add(secondComponent);
```



- **Custom `Box.Filler`**
  Use this to specify a component with whatever minimum, preferred, and maximum sizes you want. For example, to create some filler in a left-to-right layout that puts at least 5 pixels between two components and ensures that the container has a minimum height of 100 pixels, you could use this code:

```
container.add(firstComponent);
Dimension minSize = new Dimension(5, 100);
Dimension prefSize = new Dimension(5, 100);
Dimension maxSize = new Dimension(Short.MAX_VALUE, 100);
container.add(new Box.Filler(minSize, prefSize, maxSize));
container.add(secondComponent);
```

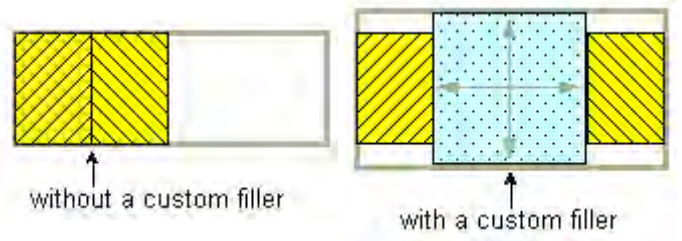
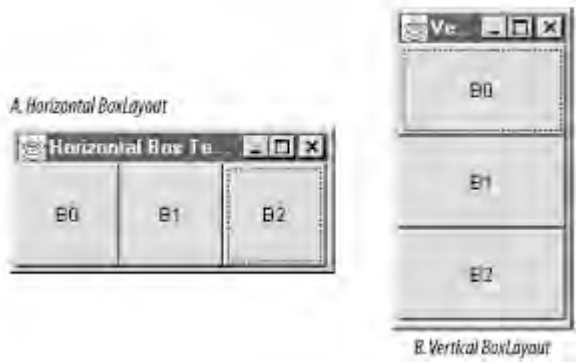


Figure 1 shows an example of a horizontal panel (A) and a vertical panel (B). To prove this `BoxLayout` is just a layout manager, we'll use regular AWT panels and buttons.



Here's a look at the code to generate the horizontal `Box`. (You need to change the axis you use in the constructor to get the vertical `Box`. That example is online as *VBox.java*.)

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

public class HBox extends JFrame {
  public HBox() {
    super("Horizontal Box Test Frame");
    setSize(200, 100);
    Panel box = new Panel( );

        // Use BoxLayout.Y_AXIS if you want a vertical Box.
    box.setLayout(new BoxLayout(box, BoxLayout.X_AXIS));
    setContentPane(box);
    for (int i = 0; i < 3; i++) {
      Button b = new Button("B" + i);
      box.add(b);
    }
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
  }

  public static void main(String args[]) {
    HBox bt = new HBox( );
  }
}
```

Demonstrating BoxLayout.

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

public class BoxLayoutFrame extends JFrame {
   // set up GUI
  public BoxLayoutFrame() {
    super( "Demonstrating BoxLayout" );
        // create Box containers with BoxLayout
    Box horizontal1 = Box.createHorizontalBox();
    Box vertical1 = Box.createVerticalBox();
    Box horizontal2 = Box.createHorizontalBox();
    Box vertical2 = Box.createVerticalBox();
    final int SIZE = 3;   // number of buttons on each Box

        // add buttons to Box horizontal1
    for (int count = 0; count < SIZE; count++)
      horizontal1.add( new JButton("Button " + count));
        // create strut and add buttons to Box vertical1
    for (int count = 0; count < SIZE; count++)
     {
```

```
      vertical1.add( Box.createVerticalStrut(25));
      vertical1.add( new JButton("Button " + count ));
     }
      // create horizontal glue and add buttons to Box horizontal2
    for (int count = 0; count < SIZE; count++)
     {
      horizontal2.add( Box.createHorizontalGlue() );
      horizontal2.add( new JButton( "Button " + count ));
     }
       // create rigid area and add buttons to Box vertical2
    for (int count = 0; count < SIZE; count++)
     {
      vertical2.add( Box.createRigidArea(new Dimension(12, 8)));
      vertical2.add( new JButton("Button " + count));
     }
       // create vertical glue and add buttons to panel
    JPanel panel = new JPanel();
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
    for (int count = 0; count < SIZE; count++)
     {
       panel.add(Box.createGlue());
       panel.add(new JButton("Button " + count));
     }
       // create a JTabbedPane
    JTabbedPane tabs = new JTabbedPane( JTabbedPane.TOP,
        JTabbedPane.SCROLL_TAB_LAYOUT );

        // place each container on tabbed pane
    tabs.addTab( "Horizontal Box", horizontal1 );
    tabs.addTab( "Vertical Box with Struts", vertical1 );
    tabs.addTab( "Horizontal Box with Glue", horizontal2 );
    tabs.addTab( "Vertical Box with Rigid Areas", vertical2 );
    tabs.addTab( "Vertical Box with Glue", panel );

    add( tabs ); // place tabbed pane on frame
  } // end BoxLayoutFrame constructor

 public static void main(String args[])  {
     BoxLayoutFrame boxLayoutFrame = new BoxLayoutFrame();
     boxLayoutFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     boxLayoutFrame.setSize(400, 220); // set frame size
     boxLayoutFrame.setVisible(true); // display frame
  } // end main
 } // end
```

C. Alat/ Bahan

1. Lab. Sheet Pemrograman Java 10
2. PC / Laptop with OS installed
3. JDK 1.5 or latest
4. J-Creator or text editor

D. Langkah Kerja

1. Baca dan pahami dasar teori di atas.
2. Lakukan kompilasi dan eksekusi terhadap contoh-contoh source code atau program yang ada di dasar teori dan LAMPIRAN.
3. Kerjakan tugas individu di bawah.

E. Tugas Individu

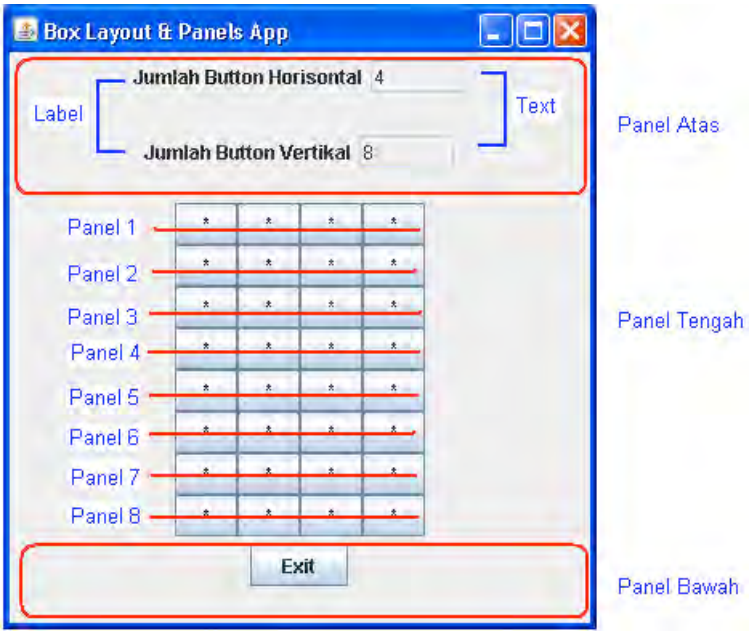
Dengan menggunakan JPanel dan Box Layout, buatlah sebuah program yang menampilkan kumpulan JButton dengan tampilan sebagai berikut

- Input Box *(Joptionpane)* yang meminta masukan untuk menentukan jumlah Button dalam arah vertikal.

- Setelah di Enter (OK), kemudian muncul Input Box *(Joptionpane)* yang meminta masukan untuk menentukan jumlah Button dalam arah horisontal.

- Kemudian muncul Frame yang menampilkan label, text, button dengan jumlah sesuai dengan input di atas.
  - Contoh untuk jumlah button 8x5

- Contoh untuk jumlah button 4x8



- Contoh untuk jumlah button 0x0

- Tombol Exit untuk keluar (menutup frame)
- Tinggi dan lebar Frame otomatis mengikuti jumlah Button yang dimasukkan

F. Lampiran

- **CH 62** (Introduction to Computer Science using Java, Java 5.0 version, January 2006, Bradley Kjell, Central Connecticut State University http://chortle.ccsu.edu/CS151/cs151java.html)

- **Java™ Swing, 2nd Edition,** Brian Cole, Robert Eckstein, James Elliott, Marc Loy, David Wood