Layout Manager

Lesson 3

Learning Objectives

In this lesson you will:

- Learn the basic GUI components
- Understand the event handling delegation model
- Learn the types of event
- Learn some Swing event classes

Layout Manager

- Classes that arrange components (like buttons, labels, text fields, etc.) in a container (like JPanel, JFrame, or JDialog)

- Automatically handle the positioning and resizing of components, making it easier to create flexible and platform-independent GUIs

Layout Manager (cont)

- Defined in the AWT

- FlowLayout
- BorderLayout
- GridLayout
- CardLayout
- GridBagLayout

- Defined in Swing

- BoxLayout
- Overlay Layout
- GroupLayout
- SpringLayout

BoxLayout

BoxLayout

- Arranges components in a single row or column
- Usage: when stacking components vertically or horizontally
- Key Methods: **Box.createHorizontalBox()**, **Box.create.VerticalBox()**

BoxLayout Methods

Method	Description	Example
`setAxis(int axis)`	Sets the axis for layout management. The axis can be either `BoxLayout.X_AXIS` (horizontal) or `BoxLayout.Y_AXIS` (vertical).	`boxLayout.setAxis(BoxLayout.Y_AXIS);` *This sets the layout to arrange components vertically.*
`getAxis()`	Returns the current axis of the layout (horizontal or vertical).	`int axis = boxLayout.getAxis();` *Returns the current axis (either X or Y axis).*
`addLayoutComponent(Component comp, Object constraints)`	Adds a component to the layout with the specified constraints.	`boxLayout.addLayoutComponent(button, "button1");` *Adds a button to the layout with a constraint of "button1".*

BoxLayout Methods (cont)

`invalidateLayout(Container target)`	Invalidates the layout of the container, forcing it to recalculate the layout of its components.	`boxLayout.invalidateLayout(container);` *Forces the layout to reflow the components within the container.*
`preferredLayoutSize(Container target)`	Returns the preferred size of the container, considering all its components and their preferred sizes.	`Dimension preferredSize = boxLayout.preferredLayoutSize(container);` *Returns the preferred size of the container.*
`minimumLayoutSize(Container target)`	Returns the minimum size of the container based on its components.	`Dimension minSize = boxLayout.minimumLayoutSize(container);` *Returns the minimum size of the container.*
`maximumLayoutSize(Container target)`	Returns the maximum size of the container based on its components.	`Dimension maxSize = boxLayout.maximumLayoutSize(container);` *Returns the maximum size of the container.*

BoxLayout Methods (cont)

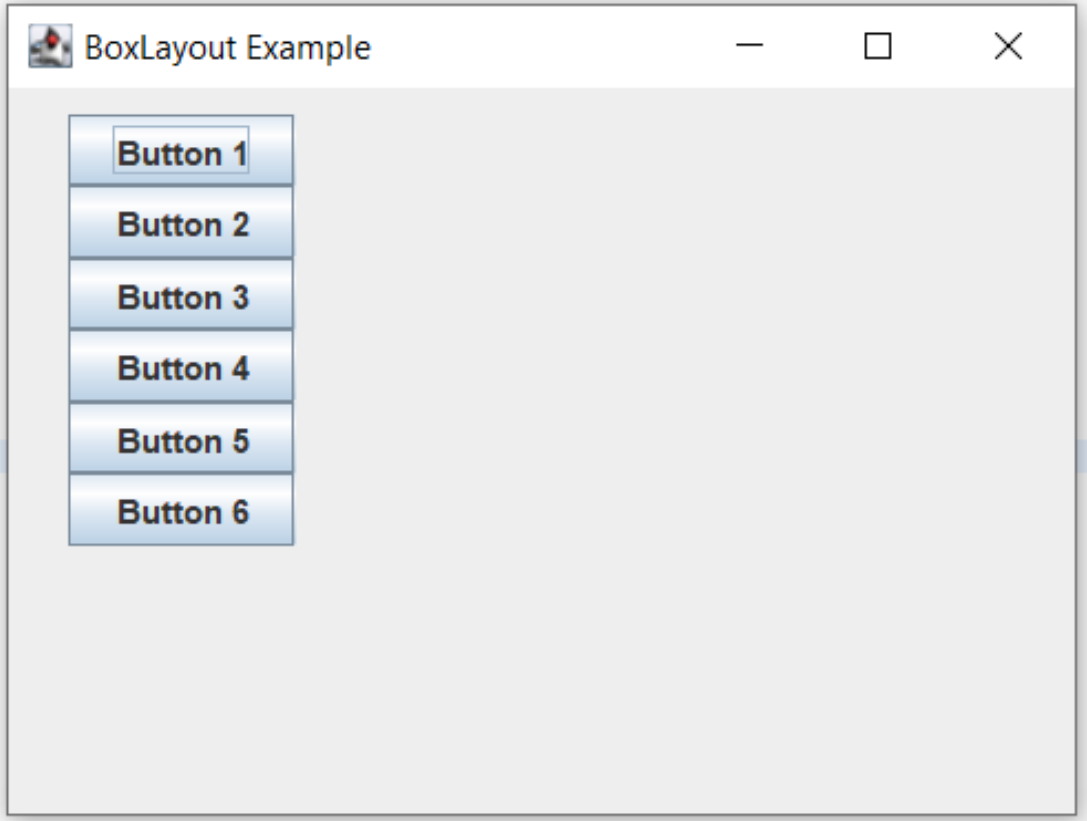
Summary of Key Methods in `BoxLayout`

Method	Description
`Box.createRigidArea(Dimension d)`	Adds **fixed** spacing
`Box.createVerticalGlue()`	Expands **vertically** to push components apart
`Box.createHorizontalGlue()`	Expands **horizontally** to push components apart
`Box.createVerticalStrut(int height)`	Adds **fixed** vertical space
`Box.createHorizontalStrut(int width)`	Adds **fixed** horizontal space

Vertical BoxLayout Sample1

BoxLayoutWithbutttons.java

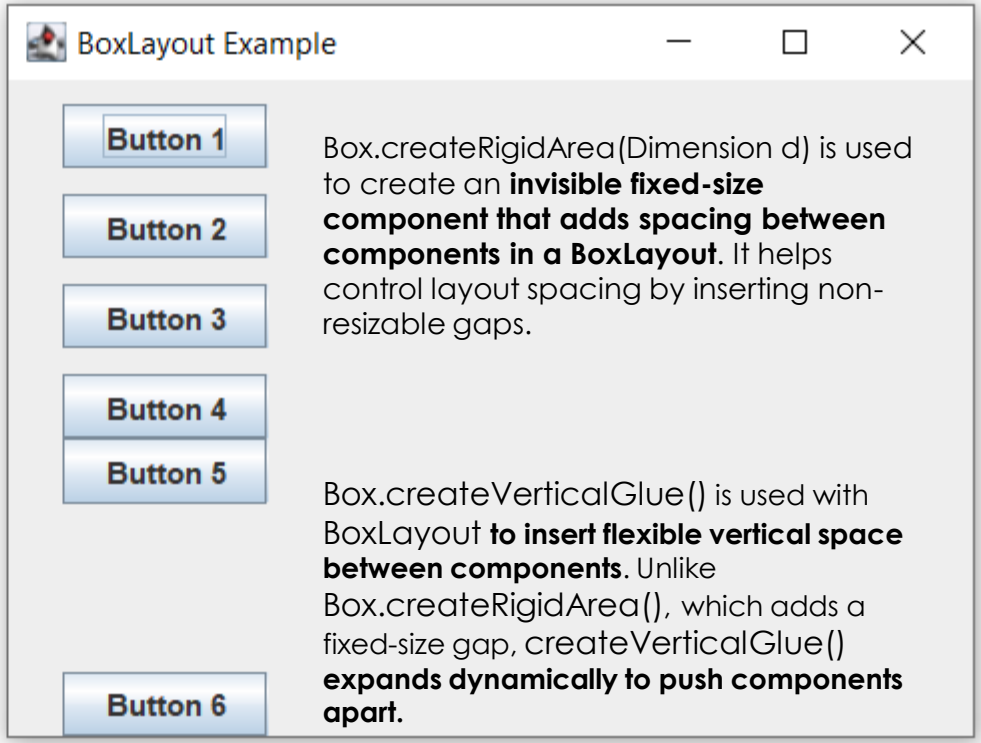
```java
package boxLayout;
import javax.swing.*;
import java.awt.*;
import javax.swing.JFrame;

public class BoxLayoutWithbutttons {

    public static void main(String[] args) {
        JFrame frame = new JFrame("BoxLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,300);

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));

        panel.add(new JButton("Button 1"));
        panel.add(new JButton("Button 2"));
        panel.add(new JButton("Button 3"));
        panel.add(new JButton("Button 4"));
        panel.add(new JButton("Button 5"));
        panel.add(new JButton("Button 6"));
        frame.add(panel);
        frame.setVisible(true);

    }

}
```



Vertical BoxLayout Sample 2

BoxLayoutWithbutttons.java

```java
package boxLayout;
import javax.swing.*;

public class BoxLayoutWithbutttons {

    public static void main(String[] args) {
        JFrame frame = new JFrame("BoxLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,300);

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));

        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));

        panel.add(new JButton("Button 1"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 2"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 3"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 4"));
        panel.add(new JButton("Button 5"));
        panel.add(Box.createVerticalGlue());
        panel.add(new JButton("Button 6"));
        frame.add(panel);
        frame.setVisible(true);

    }

}
```



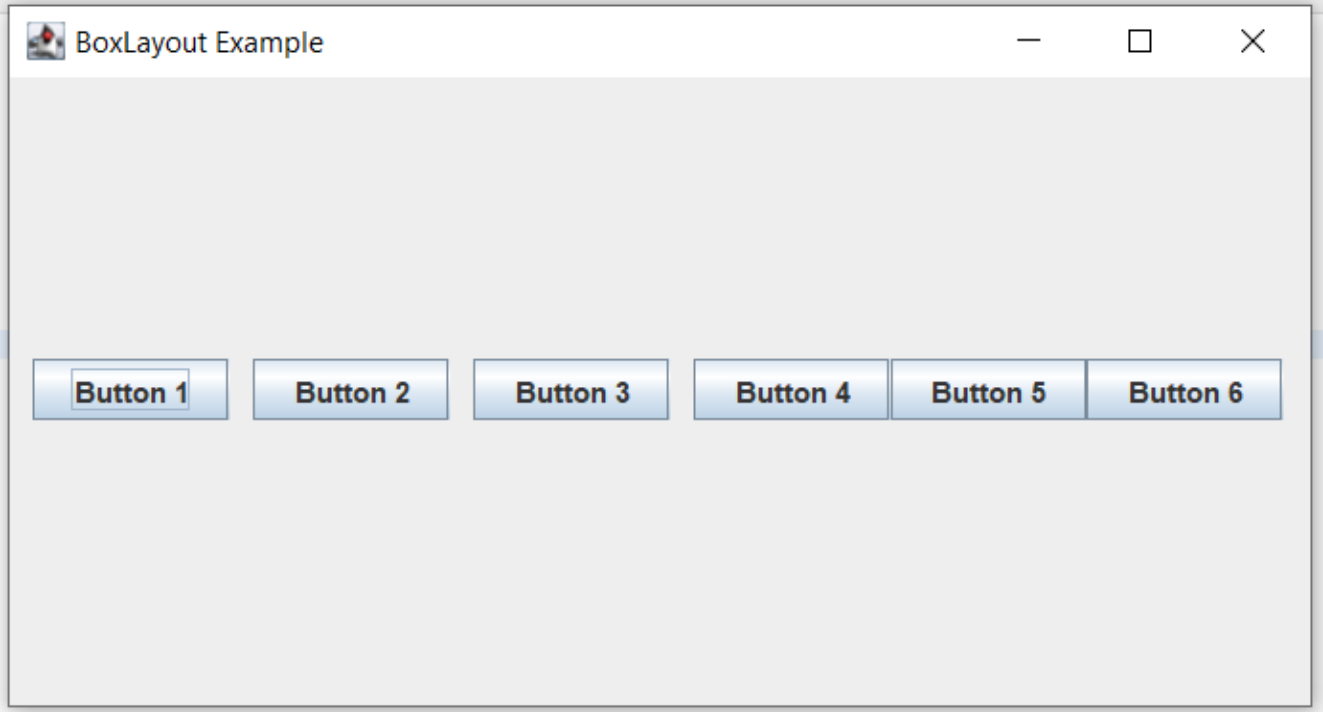
Box.createRigidArea(Dimension d) is used to create an **invisible fixed-size component that adds spacing between components in a BoxLayout**. It helps control layout spacing by inserting non-resizable gaps.

Box.createVerticalGlue() is used with BoxLayout **to insert flexible vertical space between components**. Unlike Box.createRigidArea(), which adds a fixed-size gap, createVerticalGlue() **expands dynamically to push components apart.**

Horizontal BoxLayout Sample 1

BoxLayoutWithbutttons.java

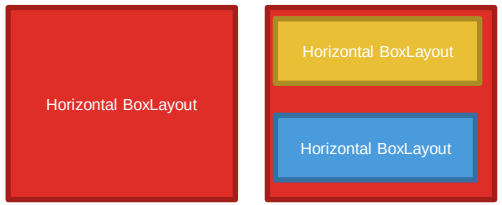
```java
package boxLayout;
import javax.swing.*;

public class BoxLayoutWithbutttons {

    public static void main(String[] args) {
        JFrame frame = new JFrame("BoxLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,300);

        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.X_AXIS));

        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));

        panel.add(new JButton("Button 1"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 2"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 3"));
        // Space between components Left, top
        panel.add(Box.createRigidArea(new Dimension(10, 10)));
        panel.add(new JButton("Button 4"));
        panel.add(new JButton("Button 5"));
        panel.add(Box.createVerticalGlue());
        panel.add(new JButton("Button 6"));
        frame.add(panel);
        frame.setVisible(true);

    }

}
```

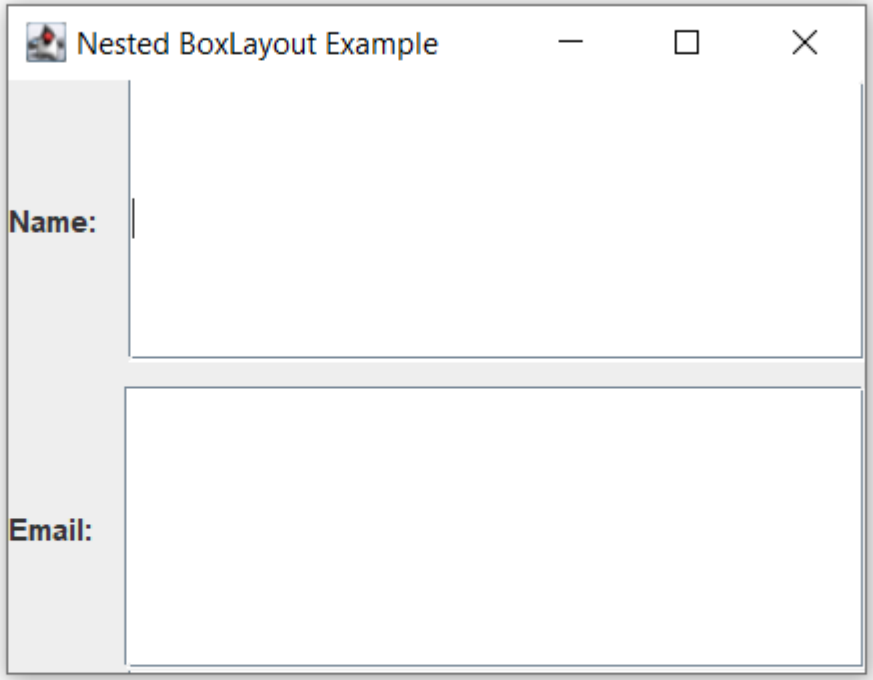


Nested BoxLayout

*NestedBoxLayout.java

```java
1  package boxLayout;
2  import javax.swing.*;
3  import java.awt.*;
4  public class NestedBoxLayout {
5
6      public static void main(String[] args) {
7          JFrame frame = new JFrame("Nested BoxLayout Example");
8          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
9          frame.setLayout(new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS));
10
11         JPanel row1 = new JPanel();
12         row1.setLayout(new BoxLayout(row1, BoxLayout.X_AXIS));
13         row1.add(new JLabel("Name: "));
14         row1.add(Box.createHorizontalStrut(10)); // Fixed spacing
15         row1.add(new JTextField(10));
16
17         JPanel row2 = new JPanel();
18         row2.setLayout(new BoxLayout(row2, BoxLayout.X_AXIS));
19         row2.add(new JLabel("Email: "));
20         row2.add(Box.createHorizontalStrut(10)); // Fixed spacing
21         row2.add(new JTextField(10));
22
23         frame.add(row1);
24         frame.add(Box.createRigidArea(new Dimension(0, 10))); // Fixed vertical space
25         frame.add(row2);
26
27         frame.pack();
28         frame.setLocationRelativeTo(null);
29         frame.setVisible(true);
30
31
32     }
33
34 }
```

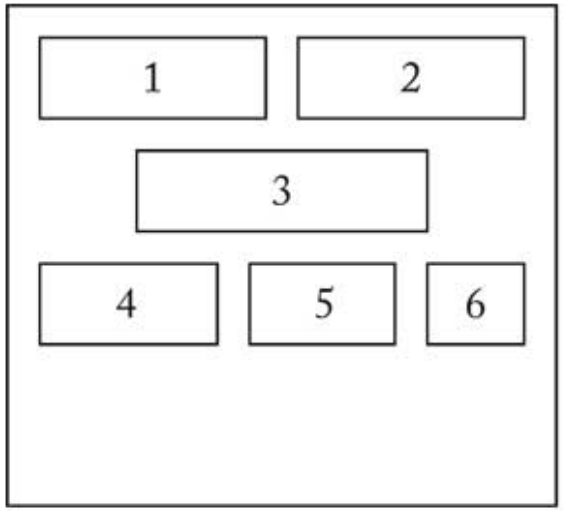




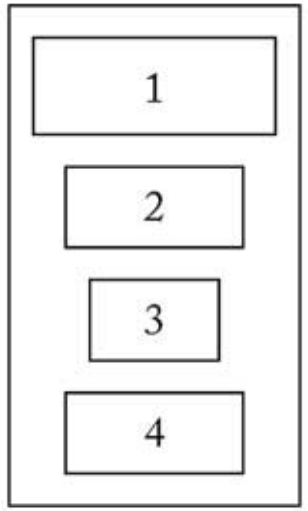
Difference Between `createHorizontalStrut()` **and** `createHorizontalGlue()`

Method	Behavior
`Box.createHorizontalStrut(int width)`	Adds **fixed** horizontal spacing
`Box.createHorizontalGlue()`	Adds **flexible** horizontal spacing that expands

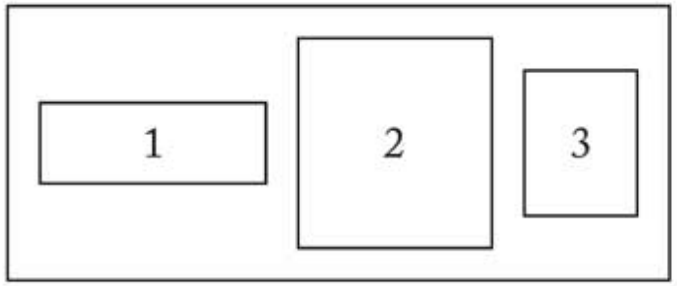
Layout Managers



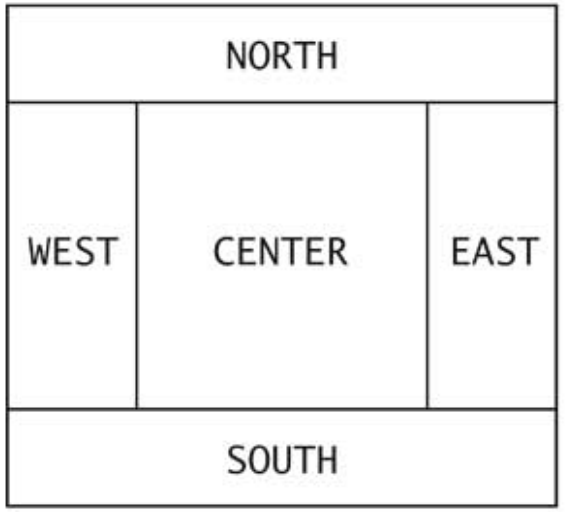
FlowLayout



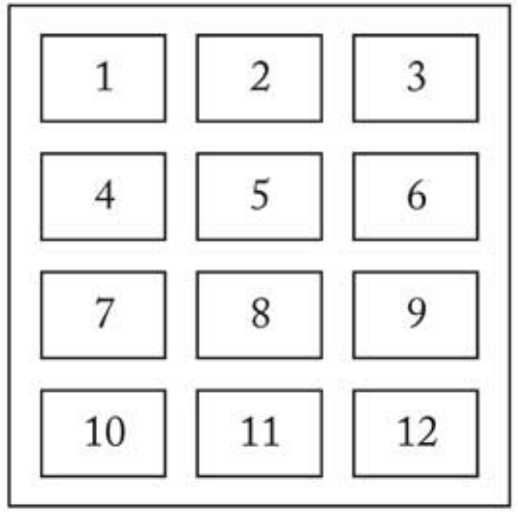
BoxLayout (vertical)



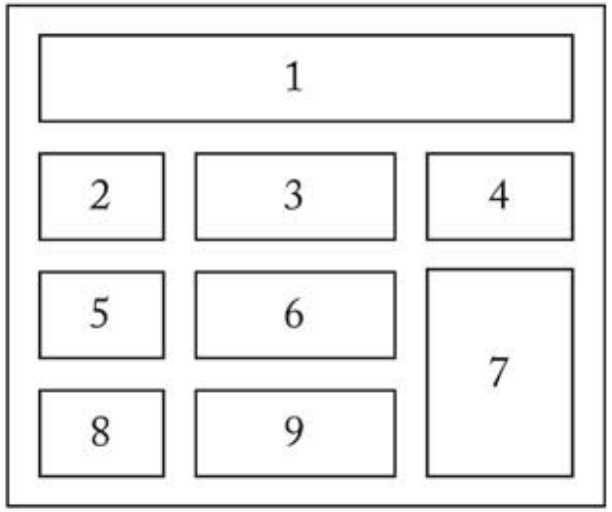
BoxLayout (horizontal)



BorderLayout



GridLayout



GridBagLayout

CardLayout

CardLayout

- Allows to manage multiple components (or "cards") in a container, where only one card is visible at a time.
- Usage: Wizards, tabbed panels, or forms with multiple steps
- Key Methods: **show**, **next**, **previous**

CardLayout Methods

Method	Description	Example
`addLayoutComponent(Component comp, Object constraints)`	Adds a component to the layout with a specific identifier (card name).	`cardLayout.addLayoutComponent(card1, "Card1");` *Adds a card component with the identifier "Card1".*
`first(Container parent)`	Displays the first card in the container.	`cardLayout.first(container);` *Displays the first card in the container.*
`last(Container parent)`	Displays the last card in the container.	`cardLayout.last(container);` *Displays the last card in the container.*
`next(Container parent)`	Displays the next card in the container. If the current card is the last one, it wraps around to the first.	`cardLayout.next(container);` *Displays the next card. If at the last, it wraps to the first card.*

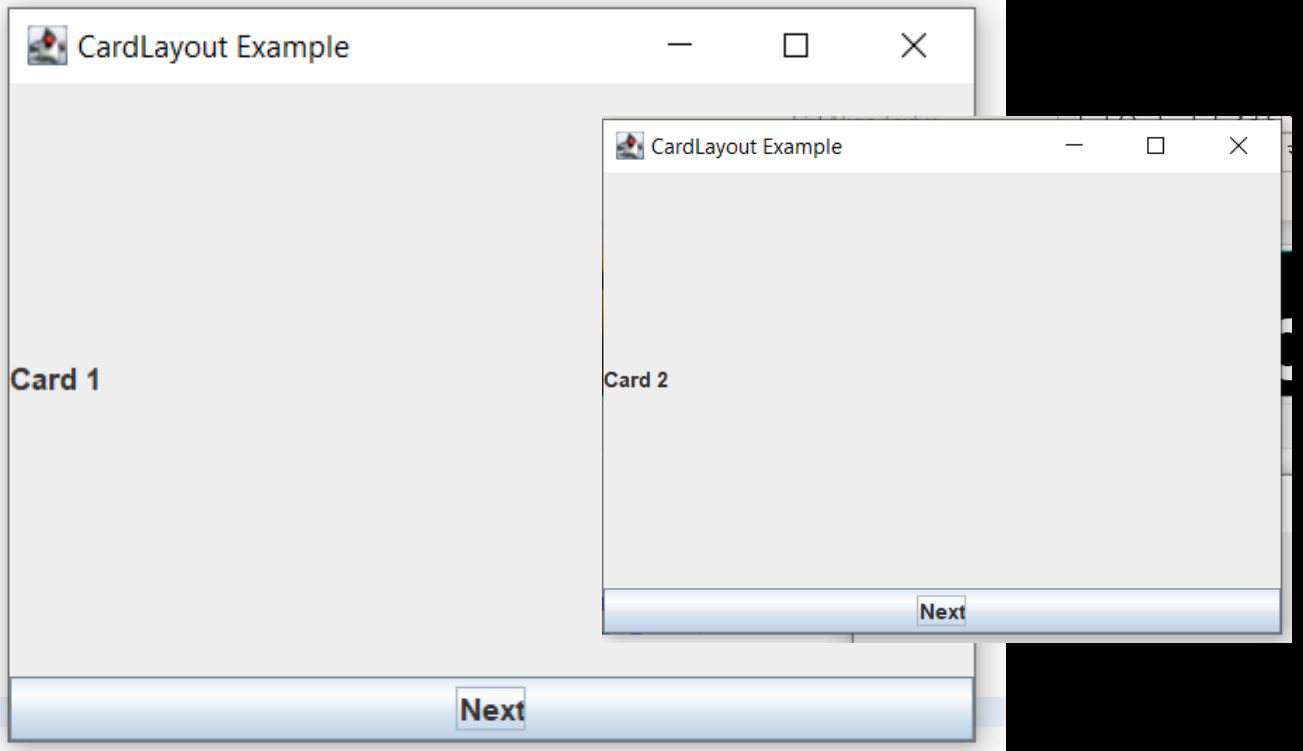
CardLayout Methods (cont)

`previous(Container parent)`	Displays the previous card in the container. If the current card is the first one, it wraps around to the last.	`cardLayout.previous(container);` *Displays the previous card. If at the first, it wraps to the last card.*
`show(Container parent, String name)`	Displays the card with the specified name.	`cardLayout.show(container, "Card2");` *Displays the card named "Card2".*
`getHgap()`	Returns the horizontal gap between cards.	`int hgap = cardLayout.getHgap();` *Returns the horizontal gap between the cards.*
`getVgap()`	Returns the vertical gap between cards.	`int vgap = cardLayout.getVgap();` *Returns the vertical gap between the cards.*
`setHgap(int hgap)`	Sets the horizontal gap between cards.	`cardLayout.setHgap(10);` *Sets the horizontal gap between cards to 10 pixels.*
`setVgap(int vgap)`	Sets the vertical gap between cards.	`cardLayout.setVgap(5);` *Sets the vertical gap between cards to 5 pixels.*

Card Layout Sample1

CardLayoutExample.java

```java
package cardLayout;
import javax.swing.*;

public class CardLayoutExample {

    public static void main(String[] args) {
        JFrame frame = new JFrame("CardLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400,300);

        CardLayout cardlayout = new CardLayout();
        JPanel panel = new JPanel(cardlayout);

        //create panel 1 - first to execute
        panel.add(new JLabel("Card 1"), "Card1");
        //create panel 2 - will display when user clicks on Next button
        panel.add(new JLabel("Card 2"), "Card2");

        JButton nextButton = new JButton("Next");
        //display the next panel when user clicks on Next button
        nextButton.addActionListener(e -> cardlayout.next(panel));

        //position panel in the center of the frame
        frame.add(panel, BorderLayout.CENTER);
        //position nextButton in the south region of the frame
        frame.add(nextButton, BorderLayout.SOUTH);

        frame.setVisible(true);

    }

}
```

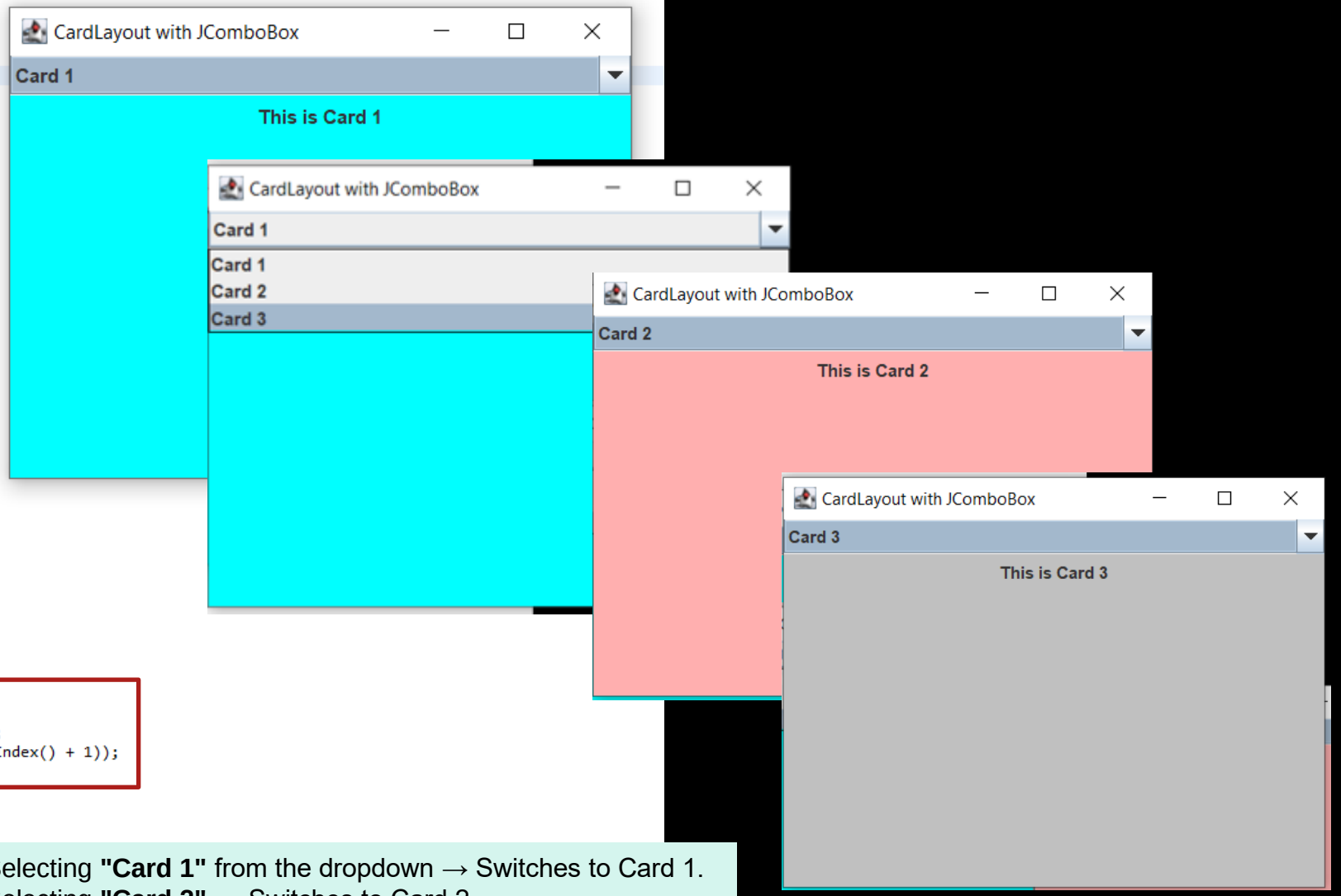


Card 1 is visible, Card 2 is invisible.
When user clicks on Next button, Card 1 is invisible
And Card 2 is visible.

Card Layout with ComboBox

*CardLayoutWithComboBox.java

```java
package cardLayout;
import javax.swing.*;
import java.awt.*;
public class CardLayoutWithComboBox {
    public static void main(String[] args) {
        JFrame frame = new JFrame("CardLayout with JComboBox");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setLayout(new BorderLayout());

        // Create CardLayout and Panel
        CardLayout cardLayout = new CardLayout();
        JPanel cardPanel = new JPanel(cardLayout);

        // Create Cards (Panels with different content)
        JPanel card1 = new JPanel();
        card1.setBackground(Color.CYAN);
        card1.add(new JLabel("This is Card 1"));

        JPanel card2 = new JPanel();
        card2.setBackground(Color.PINK);
        card2.add(new JLabel("This is Card 2"));

        JPanel card3 = new JPanel();
        card3.setBackground(Color.LIGHT_GRAY);
        card3.add(new JLabel("This is Card 3"));

        // Add cards to the card panel with unique identifiers
        cardPanel.add(card1, "Card1");
        cardPanel.add(card2, "Card2");
        cardPanel.add(card3, "Card3");

        // Create JComboBox for selecting cards
        String[] cardNames = {"Card 1", "Card 2", "Card 3"};
        JComboBox<String> comboBox = new JComboBox<>(cardNames);

        // Add ActionListener to switch cards
        comboBox.addActionListener(e -> {
            String selectedCard = (String) comboBox.getSelectedItem();
            cardLayout.show(cardPanel, "Card" + (comboBox.getSelectedIndex() + 1));
        });

        // Add components to the frame
        frame.add(comboBox, BorderLayout.NORTH);
        frame.add(cardPanel, BorderLayout.CENTER);

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



- Selecting **"Card 1"** from the dropdown → Switches to Card 1.
- Selecting **"Card 2"** → Switches to Card 2.
- Selecting **"Card 3"** → Switches to Card 3.

GroupLayout

GroupLayout

- Arranges components in a hierarchical group, allowing for precise alignment
- Usage: Designed for use with GUI builders like NetBeans
- Key Methods: **setHorizontalGroup**, **setVerticalGroup**

GroupLayout Methods

Method	Description	Example
`setAutoCreateGaps(boolean autoCreateGaps)`	Sets whether the layout manager should automatically create gaps between components.	`groupLayout.setAutoCreateGaps(true);` *Automatically creates gaps between components.*
`setAutoCreateContainerGaps(boolean autoCreateContainerGaps)`	Sets whether the layout manager should automatically create gaps between the container's edges and components.	`groupLayout.setAutoCreateContainerGaps(true);` *Automatically creates gaps around the container's edges.*

GroupLayout Methods(cont)

`addComponent(Component component)`	Adds a component to the layout.	`groupLayout.addComponent(button1);` *Adds a button to the layout.*

`createParallelGroup(int alignment)`	Creates a parallel group for components, aligned according to the specified alignment.	`GroupLayout.ParallelGroup hGroup =`
		`groupLayout.createParallelGroup(GroupLayout.Alignment.CENTER);`
		Creates a center-aligned horizontal parallel group.

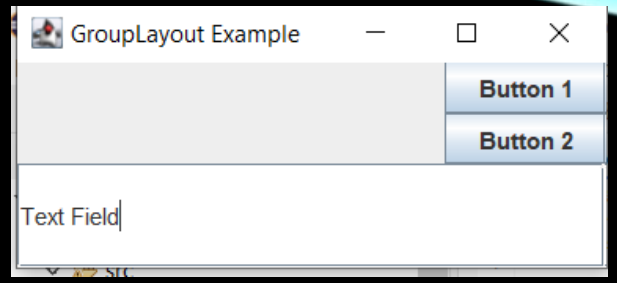
`createSequentialGroup()`	Creates a sequential group for components to be arranged one after another.	`GroupLayout.SequentialGroup vGroup =`
		`groupLayout.createSequentialGroup();`
		Creates a vertical sequential group.

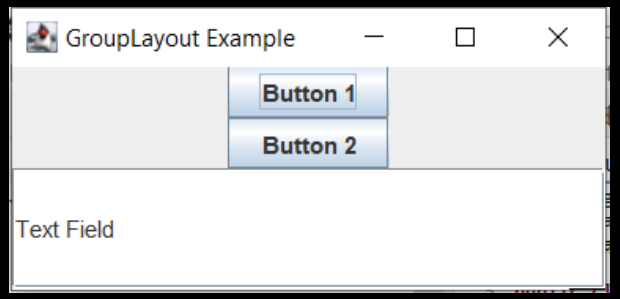
GroupLayout Methods (cont)

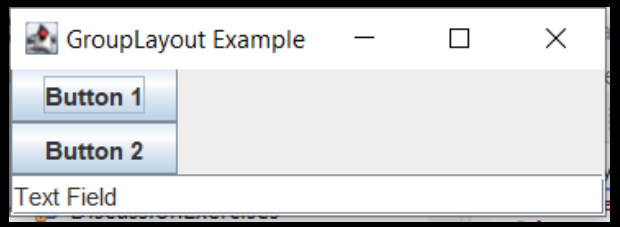
`addComponent(Component component, int minSize, int prefSize, int maxSize)`	Adds a component to a group with specific minimum, preferred, and maximum sizes.	`groupLayout.addComponent(button1, 50, 100, 150);` *Adds a button with specified size constraints.*
`linkSize(Component... components)`	Links the sizes of multiple components so that they are resized equally.	`groupLayout.linkSize(button1, button2);` *Links the sizes of button1 and button2 so they resize equally.*
`setLayoutAlignment(GroupLayout.Alignment alignment)`	Sets the alignment fo[illegible] layout.	`groupLayout.setLayoutAlignment(GroupLayout.Alignment.CENTER);` *Centers the components within the container.*

GroupLayoutExample1.java

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

public class GroupLayoutExample1 {

    public static void main(String[] args) {
        JFrame frame = new JFrame("GroupLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create components
        JButton button1 = new JButton("Button 1");
        JButton button2 = new JButton("Button 2");
        JTextField textField = new JTextField("Text Field");

        // Create GroupLayout for the frame's content pane
        GroupLayout groupLayout = new GroupLayout(frame.getContentPane());
        frame.setLayout(groupLayout);

        // Create a parallel group for horizontal alignment
        //GroupLayout.ParallelGroup hGroup = groupLayout.createParallelGroup(GroupLayout.Alignment.TRAILING);
        //GroupLayout.ParallelGroup hGroup = groupLayout.createParallelGroup(GroupLayout.Alignment.CENTER);
        GroupLayout.ParallelGroup hGroup = groupLayout.createParallelGroup(GroupLayout.Alignment.LEADING);
        hGroup.addComponent(button1)
                .addComponent(button2)
                .addComponent(textField);

        // Create a sequential group for vertical alignment
        GroupLayout.SequentialGroup vGroup = groupLayout.createSequentialGroup();
        vGroup.addComponent(button1)
                .addComponent(button2)
                .addComponent(textField);

        // Set the horizontal and vertical groups
        groupLayout.setHorizontalGroup(hGroup);
        groupLayout.setVerticalGroup(vGroup);

        // Display the frame
        frame.pack();
        frame.setVisible(true);

    }

}
```







`createParallelGroup(GroupLayout.Alignment.LEADING)` creates a parallel group where components will be aligned vertically (leading alignment).

`createSequentialGroup()` creates a sequential group where components will be stacked one below the other vertically.

`groupLayout.setHorizontalGroup(hGroup)` and `groupLayout.setVerticalGroup(vGroup)` set the layout groups for horizontal and vertical alignments, respectively.

SpringLayout

GroupLayout

- A flexible layout that positions components relative to each other using “springs” to define distances.
- Usage: When you need fine-grained control over component positioning

SpringLayout Methods

Method	Description	Example
`putConstraint(String key, Component comp, int x, int y)`	Sets a constraint for the specified component with an offset.	`springLayout.putConstraint(SpringLayout.WEST, button, 10, SpringLayout.WEST, panel);` *Aligns the button's west edge 10 pixels from the panel's west edge.*
`getConstraint(String key)`	Retrieves the constraint for the specified component (typically for debugging).	`Spring spring = springLayout.getConstraint(SpringLayout.WEST, button);` *Retrieves the west constraint for the button.*
`putConstraint(String key, Component comp, Spring spring)`	Sets a constraint using a Spring object, which represents a flexible size constraint.	`springLayout.putConstraint(SpringLayout.WEST, button, Spring.constant(10));` *Aligns the button's west edge with a constant spring of 10 pixels.*

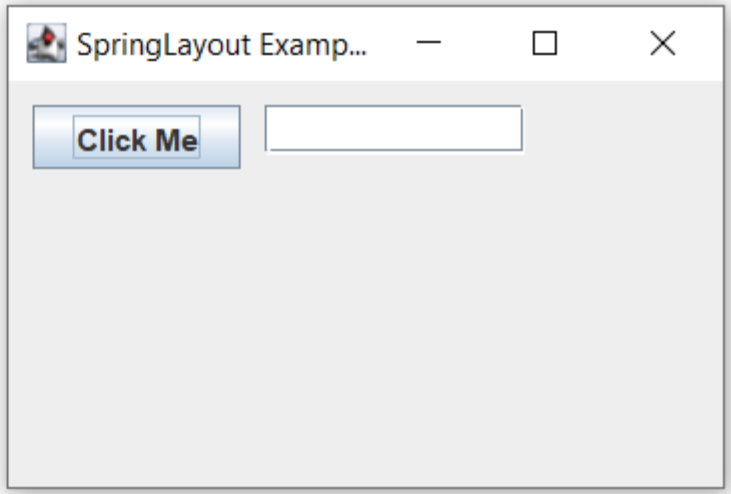
SpringLayout Methods (cont)

`getLayoutAlignmentX()`	Gets the alignment of the container's horizontal axis (used for the alignment of the components).	`float alignment = springLayout.getLayoutAlignmentX(panel);` *Gets the alignment of the container's horizontal axis.*
`getLayoutAlignmentY()`	Gets the alignment of the container's vertical axis.	`float alignment = springLayout.getLayoutAlignmentY(panel);` *Gets the alignment of the container's vertical axis.*
`getConstraints(Component comp)`	Retrieves the constraints for a given component.	`Constraint constraint = springLayout.getConstraints(button);` *Gets the constraints of the button.*
`addLayoutComponent(String name, Component comp)`	Adds a component to the layout with a name for referencing its constraints.	`springLayout.addLayoutComponent("button", button);` *Adds the button component to the layout with a name "button".*

SpringLayout Methods

springLayoutBasic.java

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

public class springLayoutBasic {

    public static void main(String[] args) {
        JFrame frame = new JFrame("SpringLayout Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Create panel and layout
        JPanel panel = new JPanel();
        SpringLayout springLayout = new SpringLayout();
        panel.setLayout(springLayout);

        // Create components
        JButton button = new JButton("Click Me");
        JTextField textField = new JTextField(10);

        // Add components to the panel
        panel.add(button);
        panel.add(textField);

        // Set constraints for the components
        springLayout.putConstraint(SpringLayout.WEST, button, 10, SpringLayout.WEST, panel);
        springLayout.putConstraint(SpringLayout.NORTH, button, 10, SpringLayout.NORTH, panel);
        springLayout.putConstraint(SpringLayout.WEST, textField, 10, SpringLayout.EAST, button);
        springLayout.putConstraint(SpringLayout.NORTH, textField, 0, SpringLayout.NORTH, button);

        // Add the panel to the frame
        frame.add(panel);

        // Display the frame
        frame.setSize(300, 200);
        frame.setVisible(true);

    }

}
```



OverLayout

OverlayLayout

- Allows multiple components to be stacked on top of each other.
- All components added to the container managed by OverlayLayout are rendered on the same position, creating an overlay effect where the last component added is drawn on top of the previous ones.

OverlayoutLayout Methods

Method	Description	Example
`setAlignmentX(float alignment)`	Sets the horizontal alignment of components within the layout.	`overlayLayout.setAlignmentX(0.5f);` *Aligns the components horizontally at the center of the container.*
`setAlignmentY(float alignment)`	Sets the vertical alignment of components within the layout.	`overlayLayout.setAlignmentY(0.5f);` *Aligns the components vertically at the center of the container.*
`addLayoutComponent(String name, Component comp)`	Adds a component to the layout with a specified name.	`overlayLayout.addLayoutComponent("label", label);` *Adds a label to the overlay layout with the name "label".*

OverlayoutLayout Methods

`preferredLayoutSize(Container target)`	Returns the preferred size of the container, which is typically the size of the largest component in the layout.	`Dimension preferredSize = overlayLayout.preferredLayoutSize(container);` *Returns the preferred size of the container.*
`minimumLayoutSize(Container target)`	Returns the minimum size of the container based on the smallest component in the layout.	`Dimension minSize = overlayLayout.minimumLayoutSize(container);` *Returns the minimum size of the container.*