Unit 6

Swing: Layouts

Project of Software Analysis and Design

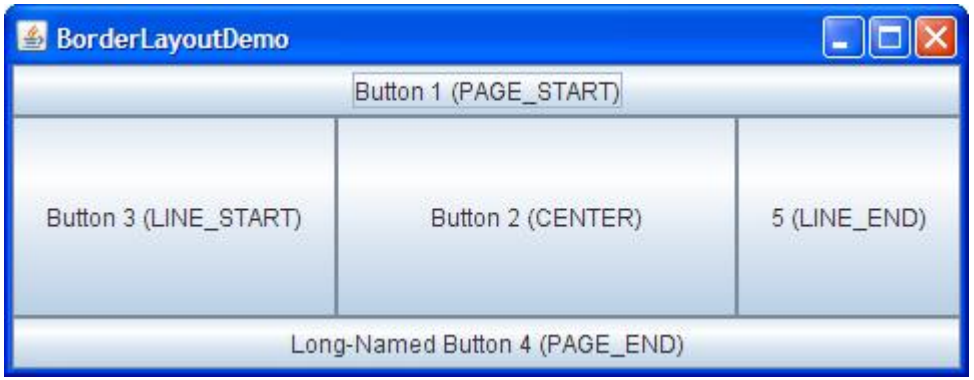
Universidad Autónoma de Madrid

Layout

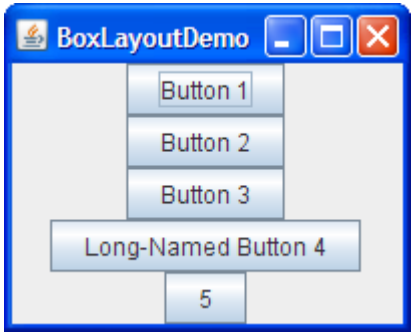
- Strategies used to render components in a container.
- Each strategy implements a layout manager, which is assigned ot the container.
- The manager is in charge of locating the components of the container according to the selected startegy.
- It is possible to nest components with different layouts.

Layouts: Examples

BorderLayout: The components can be located at North, South, East or Center

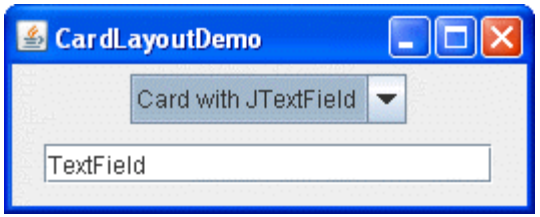


BoxLayout: The components are located along a unique row or column.

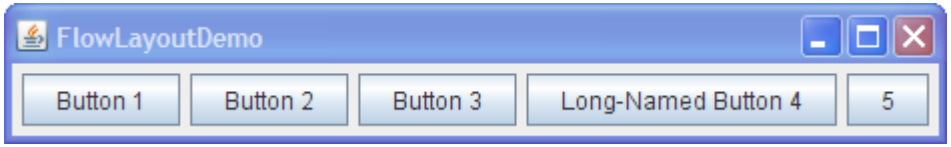


Layouts: Examples

CardLayout: Allows showing different componets at different instants. This can also be achieved by means of tabs (JTabbedPane).

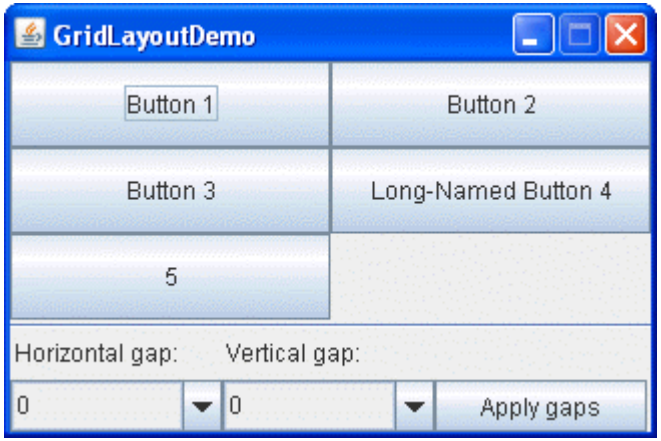


FlowLayout: Components are located along a row and, in case they do not fit, a second row starts. This is the default layout for JPanel.

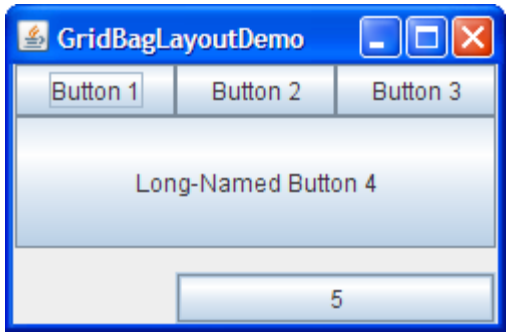


Layouts: Examples

GridLayout: Componets are located in a grid.

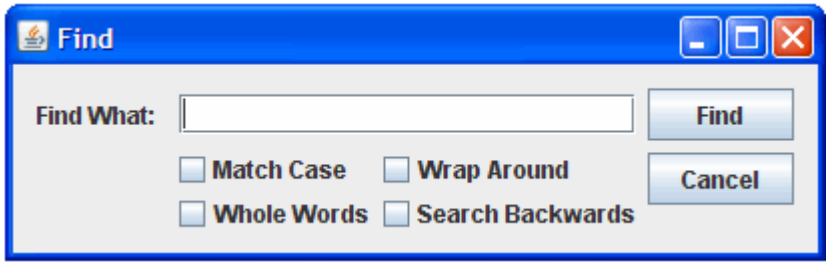


GridBagLayout: Components are located in a grid, each cell can have a different size and components can span more tan one cell

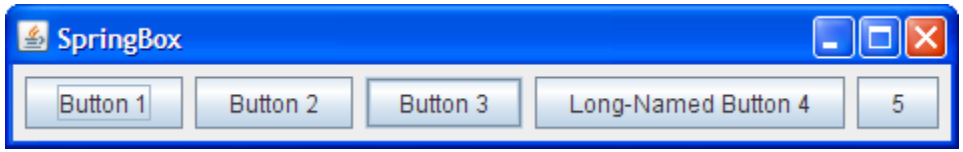


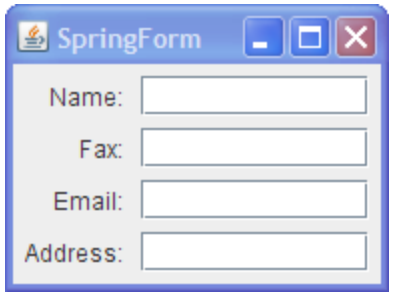
Layouts: Examples

GroupLayout: A more sophisticated **l**ayout that requires the specification of the horizontal and vertical location of each component



SpringLayout: Allows the definition of restrictions for the distance between the borders of components





Steps

1. Set the Layout manager.
2. Add components to the container.
3. Set mínimum, máximum and preferred sizes for componets (optional).
   - Many layout managers do not use these specifications, but they are used by BoxLayout, SpringLayout and GroupLayout.
4. Specify space to be left between components (optional).
5. Set container orientation (optional).

```
public JPanel build() {
   // Elements to be located in the container
   JButton buttons[]={ new JButton("Table 0"),
                       new JButton("Table 1"),
                       new JButton("Table 2"),
                       new JButton("Table 3"),
                       new JButton("Table 4")
   };

   // Container where components are located
   JPanel p = new JPanel();
   // Set grid layout
   p.setLayout(new GridLayout(0,2));
   // Add buttons to container
   for (JButton j : buttons) {
     p.add(j); // No parameters: buttons are
   }           // added following the natural
               // order
   // Change the default order used
   p.applyComponentOrientation(
    ComponentOrientation.RIGHT_TO_LEFT);

   return p;
}
```

Steps

1. Set the Layout manager.
2. Add components to the container.
3. Set mínimum, máximum and preferred sizes for componets (optional).
   - Many layout managers do not use these specifications, but they are used by BoxLayout, SpringLayout and GroupLayout.
4. Specify space to be left between components (optional).
5. Set container orientation (option-al).

```
public JPanel build() {
   // Elements to be located in the container
   JButton buttons[]={ new JButton("Table 0"),
                       new JButton("Table 1"),
                       new JButton("Table 2"),
                       new JButton("Table 3"),
                       new JButton("Table 4")
   };

   // Container where components are located
   JPanel p = new JPanel();
   // Set grid layout
   p.setLayout(new GridLayout(0,2));
   // Add buttons to container
   for (JButton j : buttons) {
     p.add(j); // No parameters: buttons are
   }           // added following the natural
               // order
   // Change the default order used
   p.applyComponentOrientation(
    ComponentOrientation.RIGHT_TO_LEFT);

   return p;
}
```

Steps

1. Set the Layout manager.
2. Add components to the container.
3. Set mínimum, máximum and preferred sizes for componets (optional).
   - Many layout managers do not use these specifications, but they are used by BoxLayout, SpringLayout and GroupLayout.
4. Specify space to be left between components (optional).
5. Set container orientation (option-al).

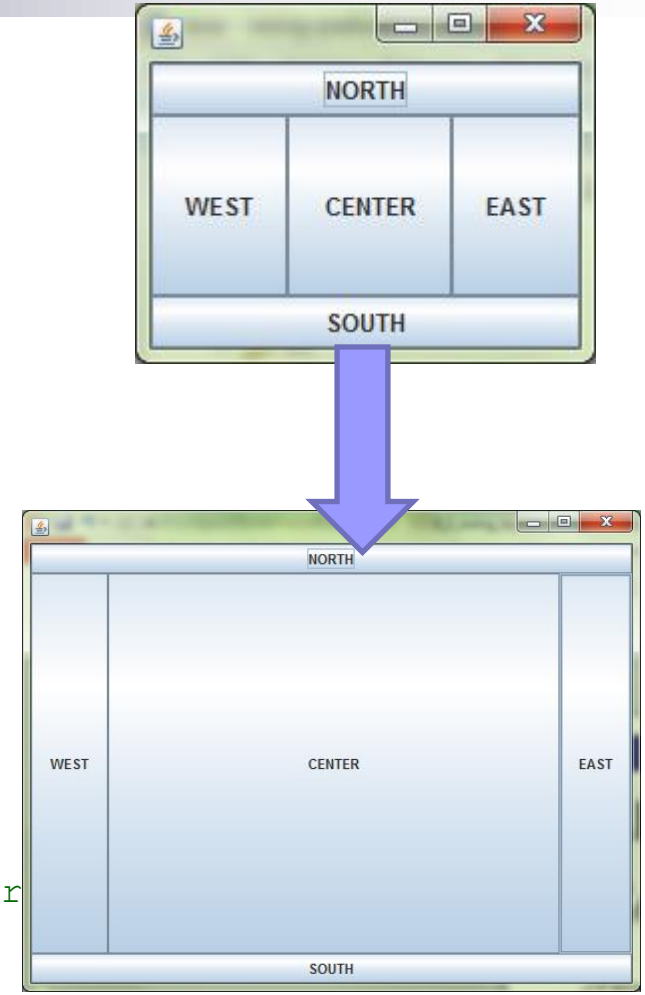
```
public JPanel build() {
   // Elements to be located in the container
   JButton buttons[]={ new JButton("Table 0"),
                       new JButton("Table 1"),
                       new JButton("Table 2"),
                       new JButton("Table 3"),
                       new JButton("Table 4")
   };

   // Container where components are located
   JPanel p = new JPanel();
   // Set grid layout
   p.setLayout(new GridLayout(0,2));
   // Add buttons to container
   for (JButton j : buttons) {
     p.add(j); // No parameters: buttons are
   }           // added following the natural
               // order
   // Change the default order used
   p.applyComponentOrientation(
    ComponentOrientation.RIGHT_TO_LEFT);

   return p;
}
```

Steps

1. Set the Layout manager.
2. Add components to the container.
3. Set mínimum, máximum and preferred sizes for componets (optional).
   - Many layout managers do not use these specifications, but they are used by BoxLayout, SpringLayout and GroupLayout.
4. Specify space to be left between components (optional).
5. Set container orientation (optional).

```
public JPanel build() {
   // Elements to be located in the container
   JButton buttons[]={ new JButton("Table 0"),
                       new JButton("Table 1"),
                       new JButton("Table 2"),
                       new JButton("Table 3"),
                       new JButton("Table 4")
   };

   // Container where components are located
   JPanel p = new JPanel();
   // Set grid layout
   p.setLayout(new GridLayout(0,2));
   // Add buttons to container
   for (JButton j : buttons) {
     p.add(j); // No parameters: buttons are
   }           // added following the natural
               // order
   // Change the default order used
   p.applyComponentOrientation(
    ComponentOrientation.RIGHT_TO_LEFT);

   return p;
}
```

Another Example: Border Layout

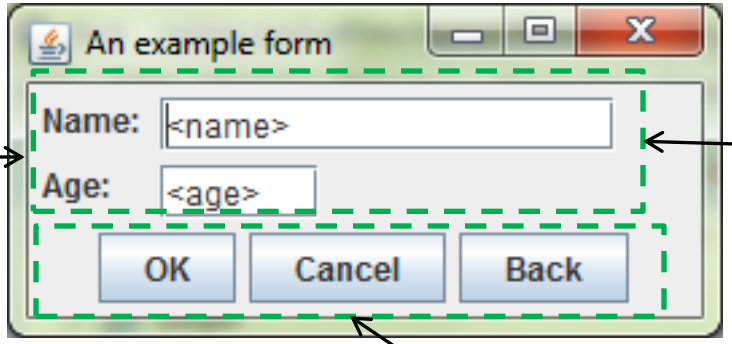



```
JPanel jp = new JPanel();
// Add border layout
jp.setLayout(new BorderLayout());

// Create componets to be located in the panel
JButton button1    = new JButton("NORTH");
JButton button2    = new JButton("SOUTH");
JButton button3    = new JButton("EAST");
JButton button4    = new JButton("WEST");
JButton button5    = new JButton("CENTER");

// Add components
jp.add(button1,  BorderLayout.NORTH);  // 2nd ar
jp.add(button2,  BorderLayout.SOUTH);
jp.add(button3,  BorderLayout.EAST);
jp.add(button4,  BorderLayout.WEST);
jp.add(button5,  BorderLayout.CENTER);
```

All components, in particular the one in the middle, will be expanded so they span all the available space in the container. Nested components with different layouts can be used.

Nested Components

JFrame: BorderLayout



JPanel with SpringLayout, located at BorderLayout.CENTER of JFrame.

JPanel with FlowLayout, Located at BorderLayout.SOUTH of JFrame.

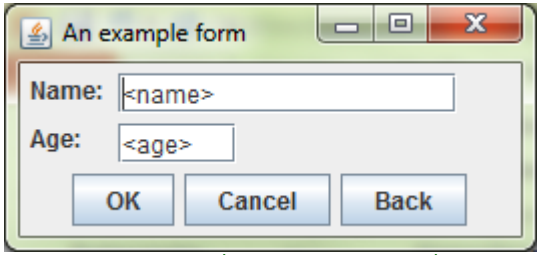
If we add no component at North, East or West, the panel reserves no room for them.

Nested Components (1/2)

```
class WindowForm extends JFrame {
  private JPanel buttonPanel = new JPanel();
  private Form form          = new Form (); // Hereda de JPanel
  private JButton ok         = new JButton("OK");
  private JButton cancel     = new JButton("Cancel");
  private JButton back       = new JButton("Back");

  public WindowForm() {
    super("An example form");
    Container cp = this.getContentPane();      // Get the Frame container
    cp.setLayout(new BorderLayout());          // Add BorderLayout to it

    // The button panel(JPanel) by default uses FlowLayout, so we do nothing
    buttonPanel.add(ok);    // Components rendered with flow layout are shown ...
    buttonPanel.add(cancel);          // by default from left to write …
    buttonPanel.add(back);  // using new rows if there is not enough horizontal space

    cp.add(buttonPanel, BorderLayout.SOUTH);   // Button panel located at South
    cp.add(form, BorderLayout.CENTER);         // Form located at center

    this.pack();            // Important: subcomponents are located according to …
                            // layout using their prefered sizes.
    this.setVisible(true);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```

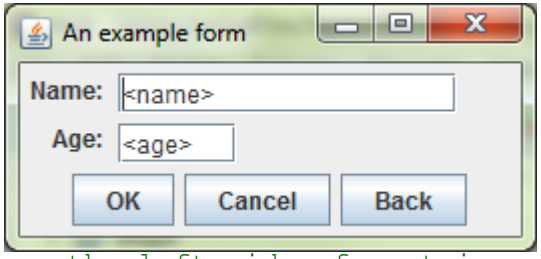
Nested Components (2/2)

```
class Form extends JPanel {
  private JLabel label, label2;
  private JTextField field, field2;

  public Form () {
    SpringLayout layout = new SpringLayout();          // Layout based on restrictions ...
    this.setLayout(layout);                            // Very flexible but low level.

    // Components to be located ...
    label  = new JLabel("Name: ");
    field  = new JTextField("<name>", 15);
    label2 = new JLabel("Age: ");
    field2 = new JTextField("<age>", 5);

    // The left side of label will be located 5 pixels away from the left side of container
    layout.putConstraint(SpringLayout.WEST, label, 5, SpringLayout.WEST, this);
    // The upper side of label will be located 5 pixels away from the upper part of container
    layout.putConstraint(SpringLayout.NORTH, label, 5, SpringLayout.NORTH, this);

    // The left side of field will be located 5 pixels away from the right side of label
    layout.putConstraint(SpringLayout.WEST, field, 5, SpringLayout.EAST, label);
    // The upper side of the field will be located 5 pixels away from the upper side of container
    layout.putConstraint(SpringLayout.NORTH, field, 5, SpringLayout.NORTH, this);

    // The left side of label2 will be aligned with the left border of label
    layout.putConstraint(SpringLayout.WEST, label2, 0, SpringLayout.WEST, label);
    // The upper part of label2 will be located 5 pixels away from the lower border of label
    layout.putConstraint(SpringLayout.NORTH, label2, 8, SpringLayout.SOUTH, label);

    // The left side of field2 will be aligned with the left side of field
    layout.putConstraint(SpringLayout.WEST, field2, 0, SpringLayout.WEST, field);
    // El upper side of field2 will be located 5 pixels away from field.
    layout.putConstraint(SpringLayout.NORTH, field2, 5, SpringLayout.SOUTH, field);

    this.add(label); this.add(field); this.add(label2); this.add(field2);
    this.setPreferredSize(new Dimension(250,50));      // important: preferred size for this panel
    this.setVisible(true);  }
}
```

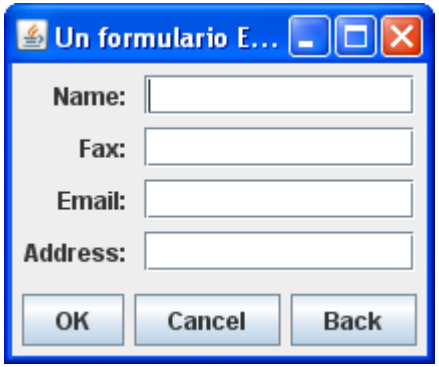


Nested Components (2/2)

```
class Form extends JPanel {
  private JLabel label, label2;
  private JTextField field, field2;

  public Form () {
    SpringLayout layout = new SpringLayout();          // Layout based on restrictions ...
    this.setLayout(layout);                            // Very flexible but low level.

    // Components to be located ...
    label  = new JLabel("Name: ");
    field  = new JTextField("<name>", 15);
    label2 = new JLabel("Age: ");
    field2 = new JTextField("<age>", 5);

    // The left side of label will be located 5 pixels away from the left side of container
    layout.putConstraint(SpringLayout.WEST, label, 5, SpringLayout.WEST, this);
    // The upper side of label will be located 5 pixels away from the upper part of container
    layout.putConstraint(SpringLayout.NORTH, label, 5, SpringLayout.NORTH, this);

    // The left side of field will be located 5 pixels away from the right side of label
    layout.putConstraint(SpringLayout.WEST, field, 5, SpringLayout.EAST, label);
    // The upper side of the field will be located 5 pixels away from the upper side of container
    layout.putConstraint(SpringLayout.NORTH, field, 5, SpringLayout.NORTH, this);

    // The right side of label2 will be aligned with the right side of label
    layout.putConstraint(SpringLayout.EAST, label2, 0, SpringLayout.EAST, label);
    // The upper part of label2 will be located 5 pixels away from the lower border of label
    layout.putConstraint(SpringLayout.NORTH, label2, 8, SpringLayout.SOUTH, label);

    // The left side of field2 will be aligned with the left side of field
    layout.putConstraint(SpringLayout.WEST, field2, 0, SpringLayout.WEST, field);
    // El upper side of field2 will be located 5 pixels away from field.
    layout.putConstraint(SpringLayout.NORTH, field2, 5, SpringLayout.SOUTH, field);

    this.add(label); this.add(field); this.add(label2); this.add(field2);
    this.setPreferredSize(new Dimension(250,50));      // important: preferred size for this panel
    this.setVisible(true);  }
}
```



Components alignment




```
public Form () {
  SpringLayout layout = new SpringLayout();
  this.setLayout(layout);

  // Labels to be included in the form
  String[] labels = {"Name: ", "Fax: ", "Email: ", "Address: "};
  int numPairs = labels.length;

  // Create label components and edition fields
  for (int i = 0; i < numPairs; i++) {
    JLabel l = new JLabel(labels[i], JLabel.TRAILING);   // 2nd parám= horizontal alignment
    this.add(l);                                         // Add without any restriction
    JTextField textField = new JTextField(10);           // 10=field size, unit: columns
    l.setLabelFor(textField);                            // Associate label to field
    this.add(textField);                                 // Add without any restriction
  }

  // This method is useful when trying to locate the components
  SpringUtilities.makeCompactGrid(this, // Container where the components are to be located
                                  numPairs, 2, // muber of rows and columns
                                  6, 6,        // initX, initY
                                  6, 6);       // horizontal and vertical separations
  this.setVisible(true);
}
```

API for SpringUtilities class:
http://download.oracle.com/javase/tutorial/uiswing/examples/layout/SpringGridProject/src/layout/SpringUtilities.java

How to switch to a different panel in the same window

- Use either tabs or *CardLayout*.

- Components managed by a CardLayout are like a stack of cards.
  - Only the component that is on top of the stack is visible.

- It is posible to switch the visible component:
  - Show the first or last component in the stack.
  - Show the component that follows or precedes to the current one.
  - Show a specific component selected by its id.

CardLayout

Operation. Step 1.

```
// Cards panel declaration
JPanel cards;
final static String BUTTONPANEL = "Card with JButtons";
final static String TEXTPANEL   = “Card with JTextField";

// Create and initialize each card
JPanel card1 = new JPanel();
...
JPanel card2 = new JPanel();
...

// Create the panel that contains the cards
cards = new JPanel(new CardLayout());
cards.add(card1, BUTTONPANEL);
cards.add(card2, TEXTPANEL);
```

CardLayout

Operation. Step 1.

```
// Cards panel declaration
JPanel cards;
final static String BUTTONPANEL = "Card with JButtons";
final static String TEXTPANEL   = "Card with JTextField";

// Create and initialize each card
JPanel card1 = new JPanel();
...
JPanel card2 = new JPanel();
...

// Create the panel that contains the cards
cards = new JPanel(new CardLayout());
cards.add(card1, BUTTONPANEL);
cards.add(card2, TEXTPANEL);
```

Panel for cards

String that identifies the cards

CardLayout

Operation. Step 1.

```
// Cards panel declaration
JPanel cards;
final static String BUTTONPANEL = "Card with JButtons";
final static String TEXTPANEL   = "Card with JTextField";

// Create and initialize each card
JPanel card1 = new JPanel();
...
JPanel card2 = new JPanel();
...

// Create the panel that contains the cards
cards = new JPanel(new CardLayout());
cards.add(card1, BUTTONPANEL);
cards.add(card2, TEXTPANEL);
```

A panel for each card

Add cards to container panel, indicating their ids

CardLayout

Operation. Step 2.

```
JPanel comboBoxPane = new JPanel(); //FlowLayout is used by default
String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL };

JComboBox<String> cb = new JComboBox<String>(comboBoxItems);   // Controls which card is shown

cb.setEditable(false);                          // Combo box used just for selection
cb.addItemListener(this);                       // itemStateChanged to be called at selection
comboBoxPane.add(cb);                           // Add combo box to father panel
...
pane.add(comboBoxPane, BorderLayout.NORTH);     // Show the combo box
pane.add(cards, BorderLayout.CENTER);           // Show cards panel
...
```

```
// This method is necessary for the ItemListener interface,
// it makes it possible to select the panel that will be shown
public void itemStateChanged(ItemEvent evt) {
   CardLayout cl = (CardLayout)(cards.getLayout());  // Get cards layout
   cl.show(cards, (String)evt.getItem());   // Show the card that corresponds to the
                                             // id chosen in the combo box.
}
```

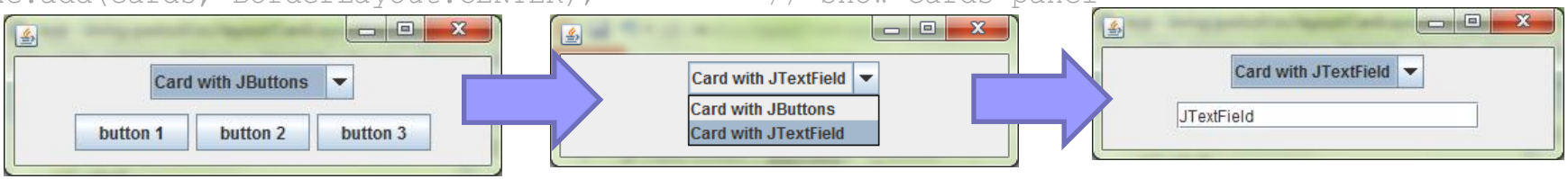
CardLayout

Operation. Step 2.

A ComboBox will control which panel is shown:

- The combo box is created, which shows the cards' ids.
- itemStateChanged shows a new card each time a selection is made in the combo box

```
JPanel comboBoxPane = new JPanel(); //FlowLayout
String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL

JComboBox cb = new JComboBox(comboBoxItems);    //

cb.setEditable(false);                          // Combo box used just for selection
cb.addItemListener(this);                       // itemStateChanged to be called at selection
comboBoxPane.add(cb);                           // Add combo box to father panel
...
pane.add(comboBoxPane, BorderLayout.NORTH);     // Show the combo box
pane.add(cards, BorderLayout.CENTER);           // Show cards panel
...
```




```
// This method is necessary for the ItemListener interface,
// it makes it possible to select the panel that will be shown
public void itemStateChanged(ItemEvent evt) {
   CardLayout cl = (CardLayout)(cards.getLayout());  // Get cards layout
   cl.show(cards, (String)evt.getItem());   // Show the card that corresponds to the
                                            // id chosen in the combo box.
}
```

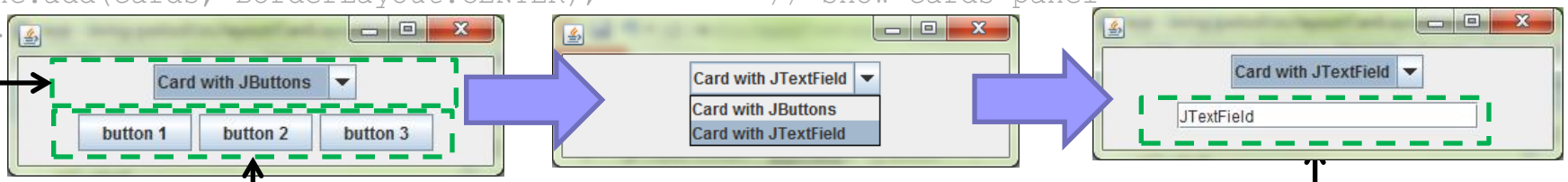
CardLayout

Operation. Step 2.

A ComboBox will control which panel is shown:

- The combo box is created, which shows the cards ids.
- itemStateChanged shows a new card each time a selection is made in the combo box

```
JPanel comboBoxPane = new JPanel(); //FlowLayout
String comboBoxItems[] = { BUTTONPANEL, TEXTPANEL

JComboBox cb = new JComboBox(comboBoxItems);    //

cb.setEditable(false);                          // Combo box used just for selection
cb.addItemListener(this);                       // itemStateChanged to be called at selection
comboBoxPane.add(cb);                           // Add combo box to father panel
...
pane.add(comboBoxPane, BorderLayout.NORTH);     // Show the combo box
pane.add(cards, BorderLayout.CENTER);           // Show cards panel
...
```




```
// This method is necessary for the ItemListener interface,
// it makes it possible to select the panel that will be shown
public void itemStateChanged(ItemEvent evt) {
    CardLayout cl = (CardLayout)(cards.getLayout());
    String ge = (String)evt.getItem();
    // Show the card that the c
}
```

comboBoxPane

Cards panel.
The card with id BUTTONPANEL is shown

Cards panel.
The card with id TEXTPANEL is shown

References

- Swing Tutorial:
  http://docs.oracle.com/javase/tutorial/uiswing/

- Swing JavaDoc API:
  http://download.oracle.com/javase/6/docs/api/javax/swing/package-summary.html

- Tutorial for layouts:
  http://download.oracle.com/javase/tutorial/uiswing/layout/index.html