

Lập trình hướng đối tượng

Biên soạn: Nguyễn Thị Tuyết Hải

Email: tuyethai@ptithcm.edu.vn

Chapter 12: User Interface Components with Swing

12.1 Swing and the Model-View-Controller Design Pattern

12.2 Layout Management

12.3 Text Input

12.4 Choice Components

12.5 Menus

12.6 Sophisticated Layout Management

12.7 Dialog Boxes

12.1 Swing and the Model-View-Controller Design Pattern

12.1.1 Design Patterns: a method for presenting a solution in a structured way.

1. The pattern first describes a context, a situation that gives rise to a design problem.
2. The problem is explained, usually as a set of conflicting forces.
3. Finally, the solution shows a configuration that balances these forces.

In the “model-view-controller” pattern,

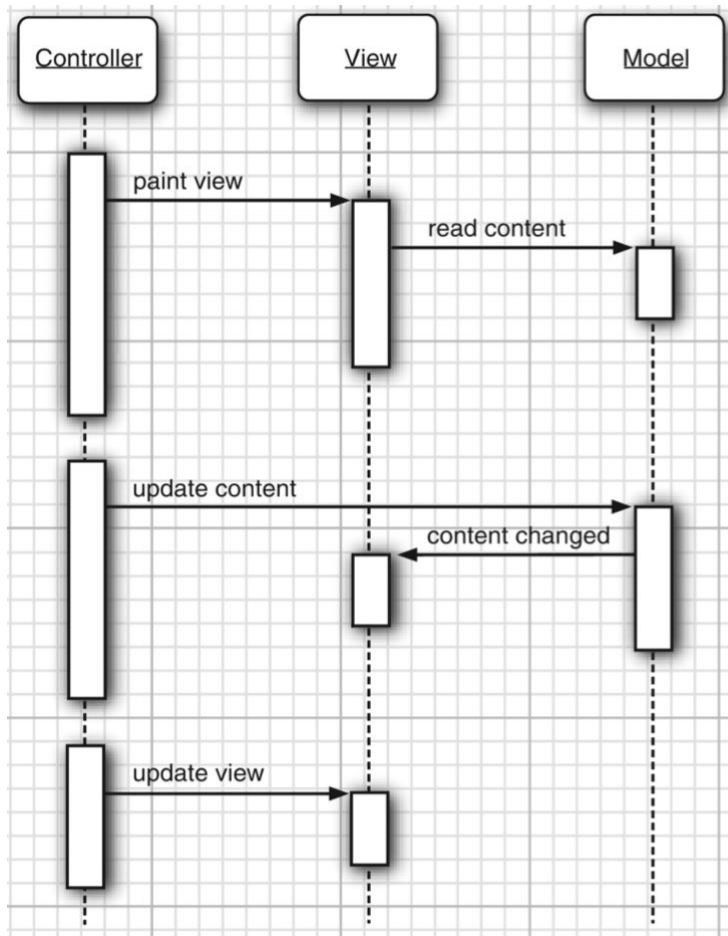
- context: a user interface system that presents information and receives user input.
- several forces: multiple visual representations of the same data that need to be updated together, the interaction mechanisms may change to support voice commands.
- solution: distribute responsibilities into three separate interacting components: the model, the view, and the controller.

12.1 Swing and the Model-View-Controller Design Pattern

12.1.2 The Model-View-Controller Pattern

Implement three separate classes:

- The model, which stores the content
- The view, which displays the content
- The controller, which handles user input



12.1 Swing and the Model-View-Controller Design Pattern

12.1.3 A Model-View-Controller Analysis of Swing Buttons

- Model: `DefaultButtonModel`
- View: `BasicButtonUI`
- Controller: `ButtonUIListener`

12.2 Introduction to Layout Management

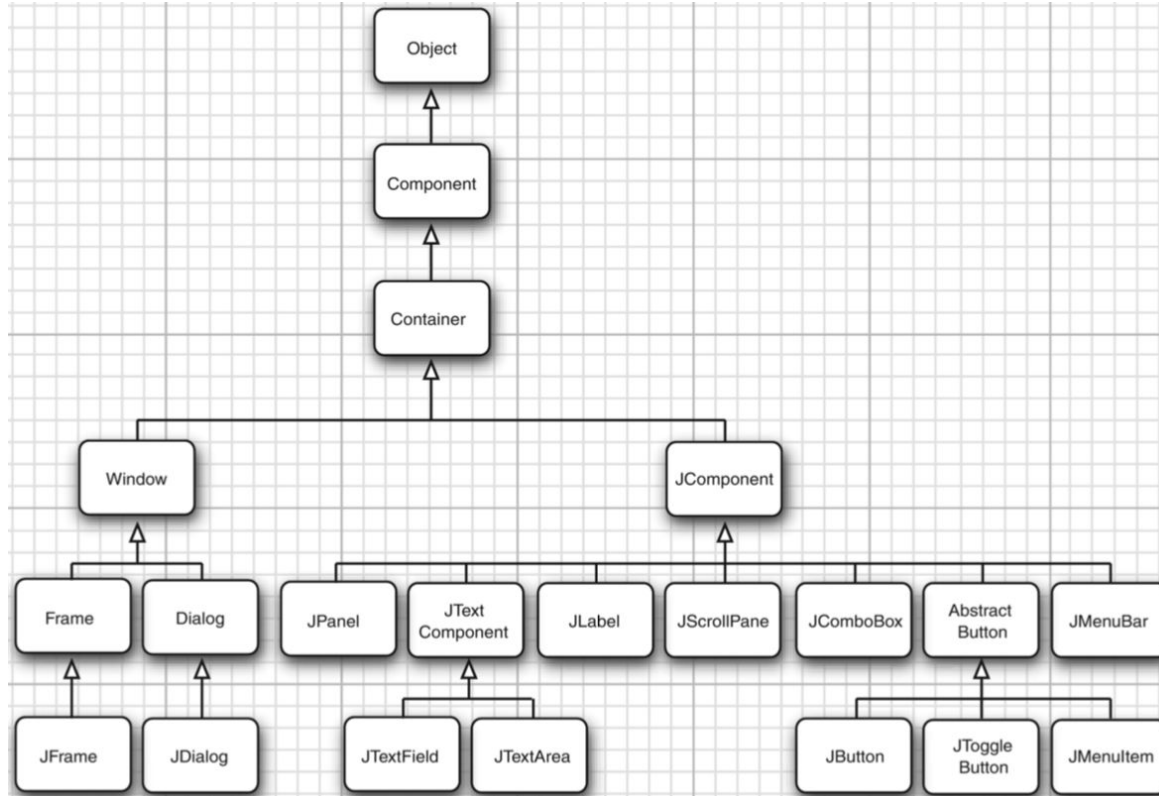
- Components (button, text field, ...) are placed inside containers (panels)
- A layout manager determines the positions and sizes of components in a container
- Each container has a default layout manager, but you can always set your own



A panel with three buttons -> A panel with six buttons managed by a flow layout -> Changing the panel size rearranges the buttons automatically.

12.2 Introduction to Layout Management

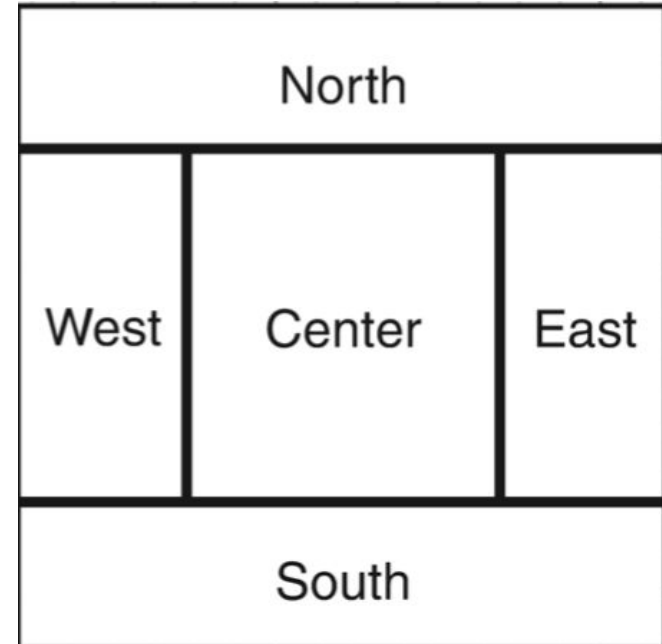
Inheritance hierarchy for the Component class



12.2 Introduction to Layout Management

12.2.1 Border Layout

- The border layout manager is the default layout manager of the content pane of every JFrame
- You can choose to place the component in the center, north, south, east, or west of the content pane



12.2 Introduction to Layout Management

12.2.1 Border Layout

- The border layout grows all components to fill the available space.
- The flow layout leaves each component at its preferred size.

```
frame.add(yellowButton, BorderLayout.SOUTH);
```

```
JPanel panel = new JPanel();  
panel.add(yellowButton);  
panel.add(blueButton);  
panel.add(redButton);  
frame.add(panel, BorderLayout.SOUTH);
```

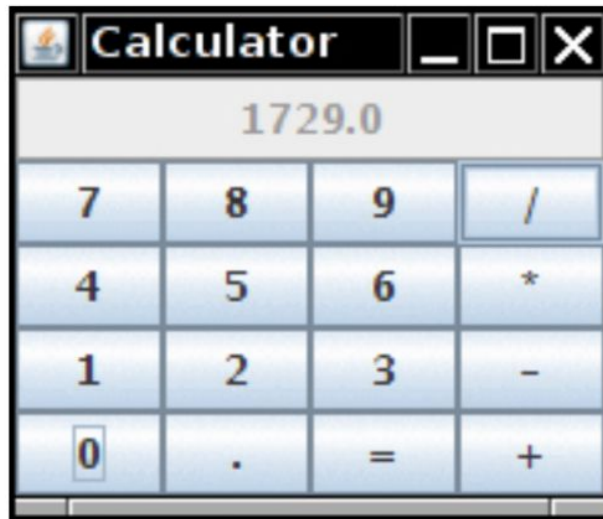


12.2 Introduction to Layout Management

12.2.2 Grid Layout

- The grid layout arranges all components in rows and columns like a spreadsheet.
- All components are given the same size.

```
panel.setLayout(new GridLayout(4, 4));  
panel.add(new JButton("1"));  
panel.add(new JButton("2"));  
...
```



12.3 Text Input

12.3.1 Text Fields

- accepts one line of text
- add a text field to a window: add it to a panel or other container.

```
JPanel panel = new JPanel();  
JTextField textField = new JTextField("Default input", 20);  
panel.add(textField);
```

```
textField.setText("Hello!"); // change the text  
String text = textField.getText().trim(); // get the text
```

12.3 Text Input

12.3.2 Labels and Labeling Components

Labels are components that hold text.

- have no decorations (for example, no boundaries).
- do not react to user input.
- often use to identify components
 - construct a JLabel component with the correct text.
 - place it close enough to the component you want to identify so that the user can see that the label identifies the correct component.

```
JLabel label = new JLabel("User name: ", SwingConstants.RIGHT);
```

or

```
JLabel label = new JLabel("User name: ", JLabel.RIGHT);
```

SwingConstants: specify alignment

12.3 Text Input

12.3.3 Password Fields

- accepts one line of text without showing the contents
- each typed character is represented by an echo character, typically an asterisk (*)

```
JPasswordField passwordField = new JPasswordField();  
panel.add(passwordField);  
frame.add(panel, BorderLayout.NORTH);
```

12.3 Text Input

12.3.4 Text Areas

- accepts many lines of text

```
JTextArea textArea = new JTextArea(8, 40) // 8 lines of 40 columns each  
panel.add(textArea);  
frame.add(panel, BorderLayout.NORTH);
```

12.3.5 Scroll Panes

In Swing, a text area does not have scrollbars.

If want scrollbars, place the text area inside a scroll pane.

```
textArea = new JTextArea(8, 40);  
JScrollPane scrollPane = new JScrollPane(textArea);
```

12.3 Text Input

```
public class TextComponentFrame extends JFrame {  
    public static final int TEXTAREA_ROWS = 8;  
    public static final int TEXTAREA_COLUMNS = 20;
```

```
    public TextComponentFrame() {  
        JTextField textField = new JTextField();  
        JPasswordField passwordField = new JPasswordField();
```

```
        JPanel northPanel = new JPanel();  
        northPanel.setLayout(new GridLayout(2, 2));  
        northPanel.add(new JLabel("User name: ", SwingConstants.RIGHT));  
        northPanel.add(textField);  
        northPanel.add(new JLabel("Password: ", SwingConstants.RIGHT));  
        northPanel.add(passwordField);  
        add(northPanel, BorderLayout.NORTH);
```

```
        JTextArea textArea = new JTextArea(TEXTAREA_ROWS, TEXTAREA_COLUMNS);  
        JScrollPane scrollPane = new JScrollPane(textArea);  
        add(scrollPane, BorderLayout.CENTER);
```

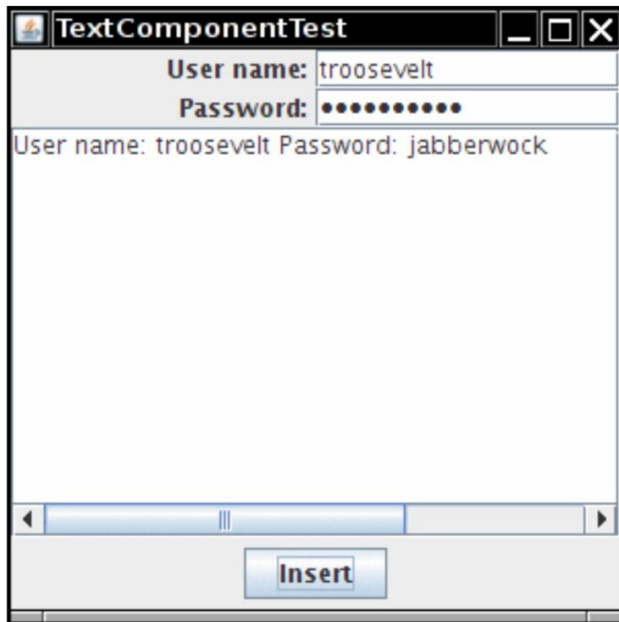
```
// add button to append text into the text area
```

```
JPanel southPanel = new JPanel();  
JButton insertButton = new JButton("Insert");  
southPanel.add(insertButton);  
insertButton.addActionListener(event -> textArea.append(  
    "User name: " + textField.getText() + " Password: "  
    + new String(passwordField.getPassword()) + "\n"));
```

```
add(southPanel, BorderLayout.SOUTH);
```

```
pack();
```

```
    }  
}
```



12.4 Choice Components

12.4.1 Checkboxes

If you want to collect just a “yes” or “no” input, use a checkbox component. Checkboxes automatically come with labels that identify them.



```
public class CheckBoxFrame extends JFrame {
    private JLabel label;
    private JCheckBox bold;
    private JCheckBox italic;
    private static final int FONTSIZE = 24;
    public CheckBoxFrame() {
        // add the sample text label
        label = new JLabel("The quick brown fox jumps over the lazy dog.");
        label.setFont(new Font("Serif", Font.BOLD, FONTSIZE));
        add(label, BorderLayout.CENTER);

        // this listener sets the font attribute of
        // the label to the check box state
        ActionListener listener = event -> {
            int mode = 0;
            if (bold.isSelected())
                mode += Font.BOLD;
            if (italic.isSelected())
                mode += Font.ITALIC;
            label.setFont(new Font("Serif", mode, FONTSIZE));
        };

        // add the check boxes
        JPanel buttonPanel = new JPanel();

        bold = new JCheckBox("Bold");
        bold.addActionListener(listener);
        bold.setSelected(true);
        buttonPanel.add(bold);

        italic = new JCheckBox("Italic");
        italic.addActionListener(listener);
        buttonPanel.add(italic);

        add(buttonPanel, BorderLayout.SOUTH);
        pack();
    }
}
```


12.4 Choice Components

12.4.2 Radio Buttons

- allow the user to check only one of several boxes.
- when another box is checked, the previous box is automatically unchecked

ButtonGroup group = new ButtonGroup(); // controls only the behavior of the buttons

JRadioButton smallButton = new JRadioButton("Small", false);

group.add(smallButton);

JRadioButton mediumButton = new JRadioButton("Medium", true);

group.add(mediumButton);

12.4 Choice Components

```
public class RadioButtonFrame extends JFrame {
    private JPanel buttonPanel; private ButtonGroup group;
    private JLabel label; private static final int DEFAULT_SIZE = 36;
    public RadioButtonFrame() {
        // add the sample text label
        label = new JLabel("The quick brown fox jumps over the lazy dog.");
        label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));
        add(label, BorderLayout.CENTER);

        // add the radio buttons
        buttonPanel = new JPanel();
        group = new ButtonGroup();

        addRadioButton("Small", 8); addRadioButton("Medium", 12); addRadioButton("Large", 18); addRadioButton("Extra large", 36);

        add(buttonPanel, BorderLayout.SOUTH);
        pack();
    }
    public void addRadioButton(String name, int size) {
        boolean selected = size == DEFAULT_SIZE;
        JRadioButton button = new JRadioButton(name, selected);
        group.add(button);
        buttonPanel.add(button);

        // this listener sets the label font size
        ActionListener listener = event -> label.setFont(new Font("Serif", Font.PLAIN, size));
        button.addActionListener(listener);
    }
}
```



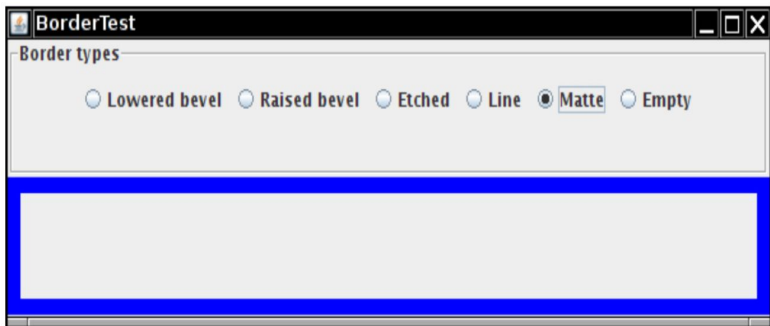
12.4 Choice Components

12.4.3 Borders

Place a border around a panel and fill that panel with other user interface elements, such as radio buttons

1. create a border: *BorderFactory*
2. add a title to the border by passing it to *BorderFactory.createTitledBorder*.
3. add the resulting border to the component by calling the *setBorder* method of the *JComponent* class.

12.4 Choice Components



```
public class BorderFrame extends JFrame {  
    private JPanel demoPanel;  
    private JPanel buttonPanel;  
    private ButtonGroup group;  
    public BorderFrame() {
```

```
        demoPanel = new JPanel();  
        buttonPanel = new JPanel();  
        group = new ButtonGroup();  
  
        addRadioButton("Lowered bevel", BorderFactory.createLoweredBevelBorder());  
        addRadioButton("Raised bevel", BorderFactory.createRaisedBevelBorder());  
        addRadioButton("Etched", BorderFactory.createEtchedBorder());  
        addRadioButton("Line", BorderFactory.createLineBorder(Color.BLUE));  
        addRadioButton("Matte", BorderFactory.createMatteBorder(10, 10, 10, 10, Color.BLUE));  
        addRadioButton("Empty", BorderFactory.createEmptyBorder());
```

```
        Border etched = BorderFactory.createEtchedBorder();  
        Border titled = BorderFactory.createTitledBorder(etched, "Border types");  
        buttonPanel.setBorder(titled);
```

```
        setLayout(new GridLayout(2, 1));  
        add(buttonPanel);  
        add(demoPanel);  
        pack();
```

```
    }
```

```
    public void addRadioButton(String buttonName, Border b) {  
        JRadioButton button = new JRadioButton(buttonName);  
        button.addActionListener(event -> demoPanel.setBorder(b));  
        group.add(button);  
        buttonPanel.add(button);
```

```
    }
```

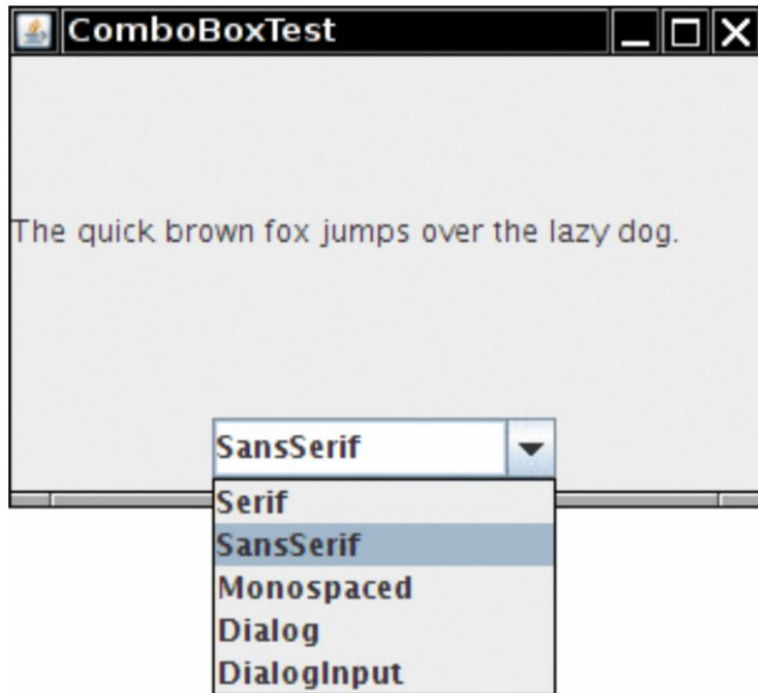
```
}
```

12.4 Choice Components

12.4.4 Combo Boxes

- radio buttons take up too much screen space
- combo boxes: the user can then select item from a list of choices drops down

12.4 Choice Components



```
public class ComboBoxFrame extends JFrame {  
    private JComboBox<String> faceCombo;  
    private JLabel label;  
    private static final int DEFAULT_SIZE = 24;
```

```
    public ComboBoxFrame() {  
        // add the sample text label
```

```
        label = new JLabel("The quick brown fox jumps over the lazy dog.");  
        label.setFont(new Font("Serif", Font.PLAIN, DEFAULT_SIZE));  
        add(label, BorderLayout.CENTER);
```

```
        // make a combo box and add face names
```

```
        faceCombo = new JComboBox<>();  
        faceCombo.addItem("Serif");  
        faceCombo.addItem("SansSerif");  
        faceCombo.addItem("Monospaced");  
        faceCombo.addItem("Dialog");  
        faceCombo.addItem("DialogInput");
```

```
        // the combo box listener changes the label font to the selected face name
```

```
        faceCombo.addActionListener(event -> label  
            .setFont(new Font  
                (faceCombo.getItemAt(faceCombo.getSelectedIndex()),  
                Font.PLAIN, DEFAULT_SIZE)));
```

```
        // add combo box to a panel at the frame's southern border
```

```
        JPanel comboPanel = new JPanel();  
        comboPanel.add(faceCombo);  
        add(comboPanel, BorderLayout.SOUTH);  
        pack();
```

```
    }  
}
```

12.4 Choice Components

12.4.5 Sliders

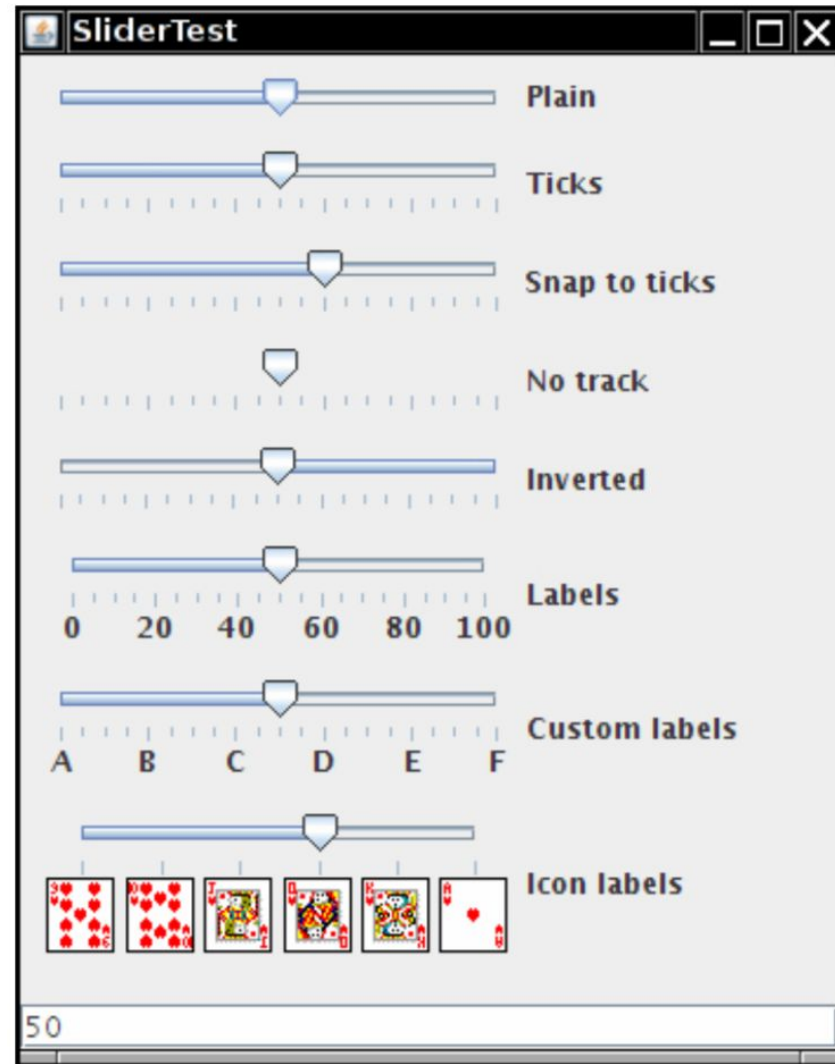
- Combo boxes let users choose from a discrete set of values.
- Sliders offer a choice from a continuum of values - e.g., any number between 1 and 100.
- Construct slider

```
JSlider slider = new JSlider(min, max, initialValue);
```

- retrieve the slider value:
ChangeListener listener = event -> {
JSlider slider = (JSlider) event.getSource();
int value = slider.getValue();

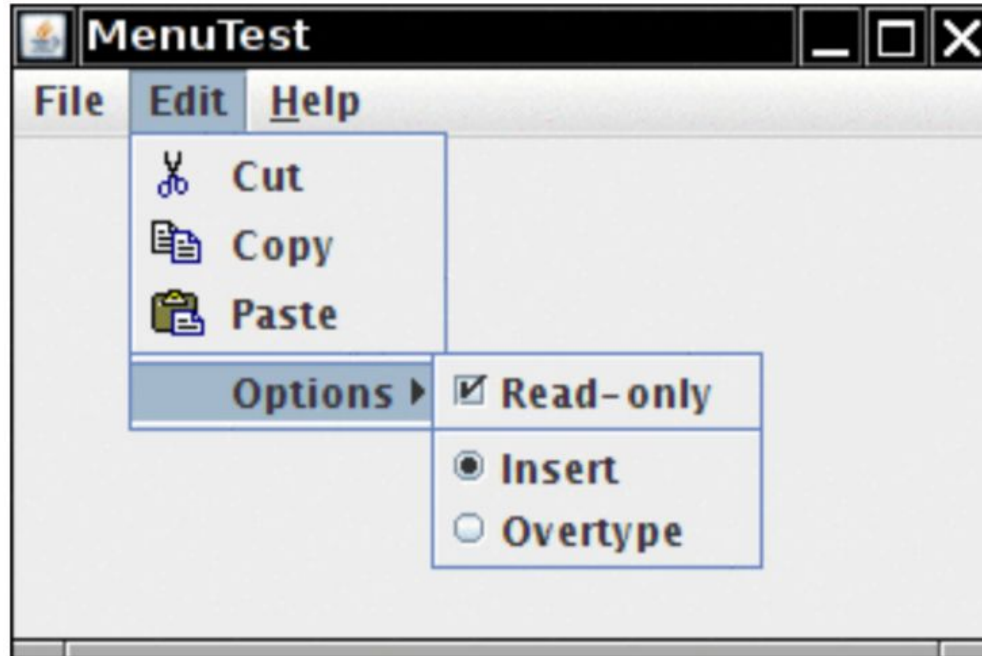
```
...  
};
```

Code: v1ch12.slider



12.5 Menus

A menu bar at the top of a window contains the names of the pull-down menus. Clicking on a name opens the menu containing menu items and submenus.



12.5 Menus

12.5.1 Menu Building

- create a menu bar:

```
JMenuBar menuBar = new JMenuBar();
```

- add it at the top of the frame

```
frame.setJMenuBar(menuBar);
```

- for each menu, you create a menu object:

```
JMenu editMenu = new JMenu("Edit");
```

- add the top-level menus to the menu bar:

```
menuBar.add(editMenu);
```

- add menu items, separators, and submenus to the menu object:

```
JMenuItem pasteItem = new JMenuItem("Paste");
```

```
editMenu.add(pasteItem);
```

```
editMenu.addSeparator();
```

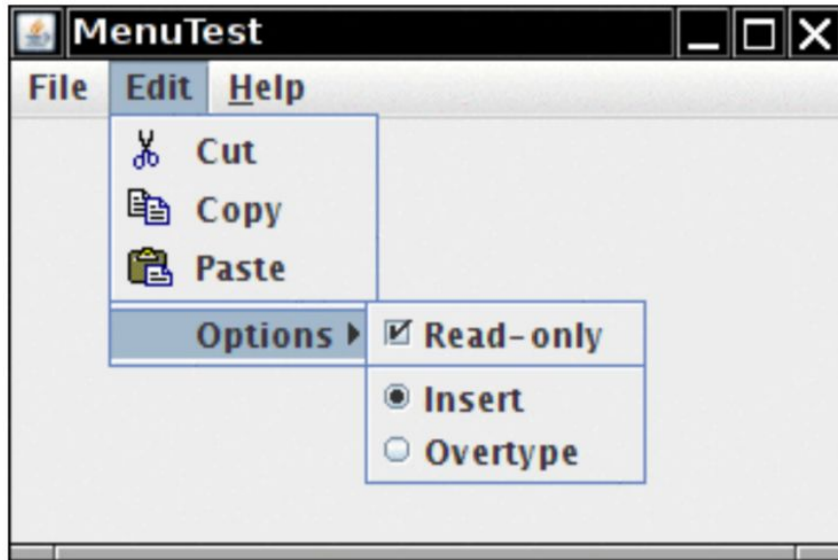
```
JMenu optionsMenu = . . . ; // a submenu
```

```
editMenu.add(optionsMenu);
```

- install an action listener for each menu item:

```
ActionListener listener = . . . ;
```

```
pasteItem.addActionListener(listener);
```



12.5 Menus

12.5.2 Icons in Menu Items

- specify the icon with the `JMenuItem(String, Icon)` or `JMenuItem(Icon)` constructor, `setIcon`
`JMenuItem cutItem = new JMenuItem("Cut", new ImageIcon("cut.gif"));`
- By default, the menu item text is placed to the right of the icon;
change it by `setHorizontalTextPosition`
`cutItem.setHorizontalTextPosition(SwingConstants.LEFT);`
- you can set the icon in the `AbstractAction` constructor:

```
cutAction = new AbstractAction("Cut", new ImageIcon("cut.gif")) {  
    public void actionPerformed(ActionEvent event) {  
        ...  
    }  
};
```

12.5 Menus

12.5.3 Checkbox and Radio Button Menu Items

Checkbox and radio button menu items display a checkbox or radio button next to the name

- Create a checkbox menu item

```
JCheckBoxMenuItem readonlyItem = new  
JCheckBoxMenuItem("Read-only");  
optionsMenu.add(readonlyItem);
```

- Create radio button menu items, add them to a button group

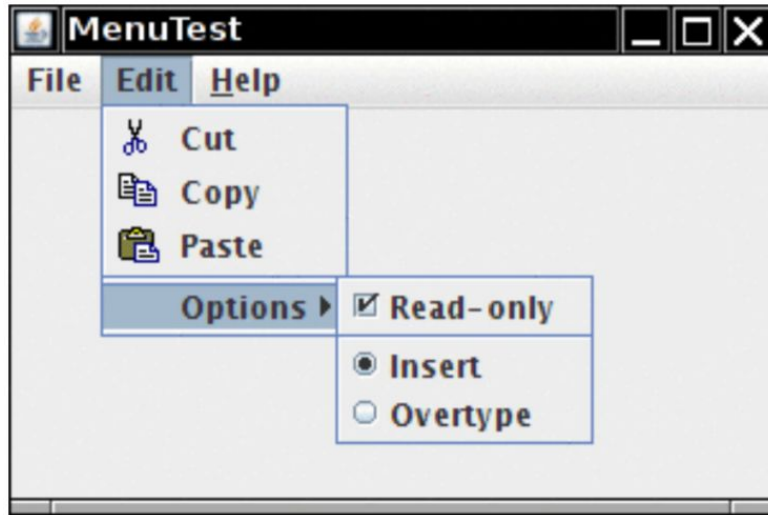
```
ButtonGroup group = new ButtonGroup();
```

```
JRadioButtonMenuItem insertItem = new  
JRadioButtonMenuItem("Insert");  
insertItem.setSelected(true);
```

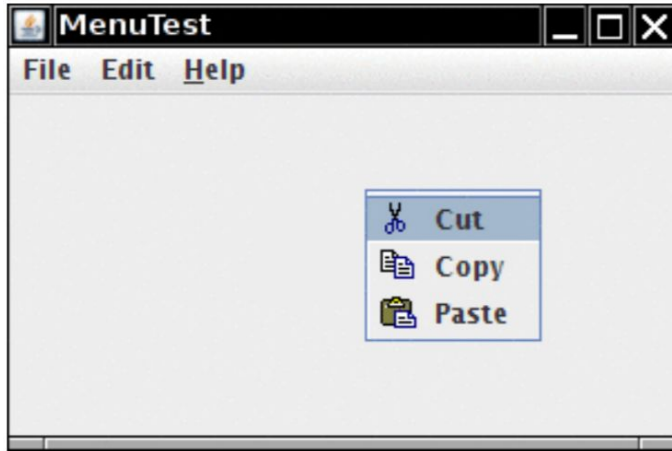
```
JRadioButtonMenuItem overtypeItem = new  
JRadioButtonMenuItem("Overtyping");
```

```
group.add(insertItem);  
group.add(overtypingItem);
```

```
optionsMenu.add(insertItem);  
optionsMenu.add(overtypingItem);
```



12.5 Menus



12.5.4 Pop-Up Menus: a menu that is not attached to a menu bar but floats somewhere

- create a pop-up menu

```
JPopupMenu popup = new JPopupMenu();
```

- add your menu items

```
JMenuItem item = new JMenuItem("Cut");  
item.addActionListener(listener);  
popup.add(item);
```

- Pop up a menu when the user clicks on a component
`component.setComponentPopupMenu(popup);`

12.5 Menus



12.5.5 Keyboard Mnemonics and Accelerators:

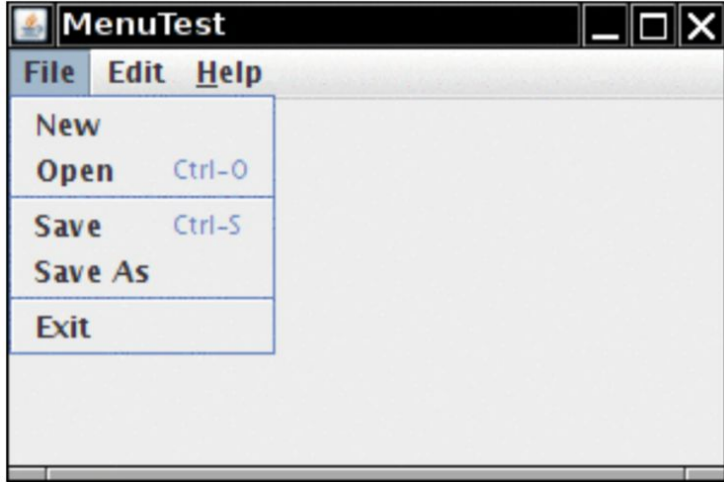
- Keyboard mnemonics: select a submenu or menu item from the currently open menu
 - specifying a mnemonic letter in the menu item constructor, or setMnemonic

```
JMenuItem aboutItem = new JMenuItem("About", 'A');
```

- If you have an Action object, you can add the mnemonic as the value of the Action.MNEMONIC_KEY key:

```
cutAction.putValue(Action.MNEMONIC_KEY, new  
Integer('A'));
```

12.5 Menus

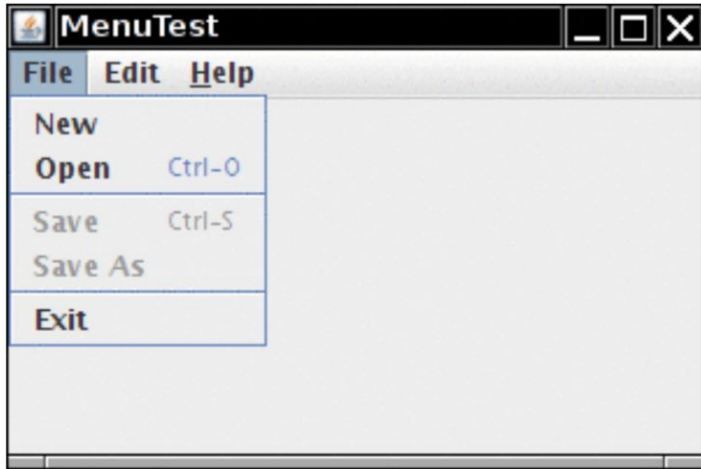


12.5.5 Keyboard Mnemonics and Accelerators

- Accelerators are keyboard shortcuts that let you select menu items without ever opening a menu
- Attaches the accelerator Ctrl+O to the `openItem` menu item

```
openItem.setAccelerator(KeyStroke.getKeyStroke("ctrl O"));
```

12.5 Menus



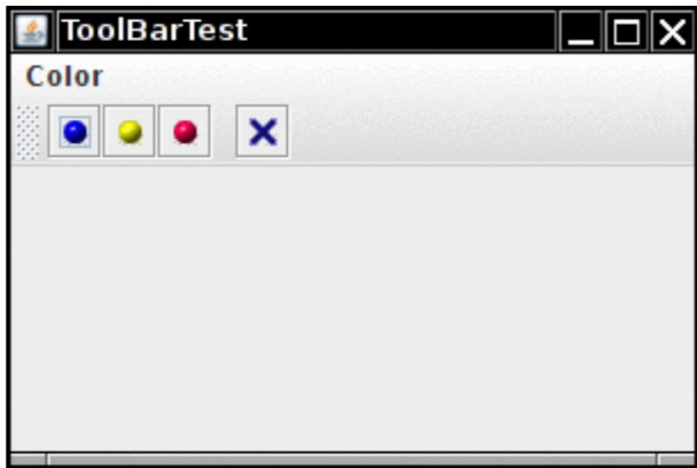
12.5.6 Enabling and Disabling Menu Items

- enable or disable a menu item, use the `setEnabled` method

```
public void menuSelected(MenuEvent event) {  
    saveAction.setEnabled(!readonlyItem.isSelected());  
    saveAsAction.setEnabled(!readonlyItem.isSelected());  
}
```

Full codes: [v1ch12.menu](#)

12.5 Menus



12.5.7 Toolbars

- A toolbar is a button bar that gives quick access to the most commonly used commands in a program, you can move toolbars elsewhere
- create a toolbar, add components or an action object to it:

```
JToolBar bar = new JToolBar();
```

```
bar.add(blueButton);
```

```
// or add action
```

```
// The icon of the action is displayed in the toolbar.
```

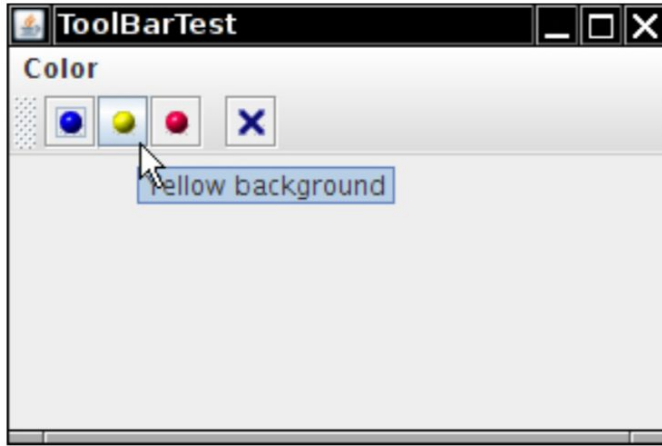
```
bar.add(blueAction);
```

- separate groups of buttons with a separator

```
bar.addSeparator();
```
- add the toolbar to the frame

```
add(bar, BorderLayout.NORTH);
```


12.5 Menus

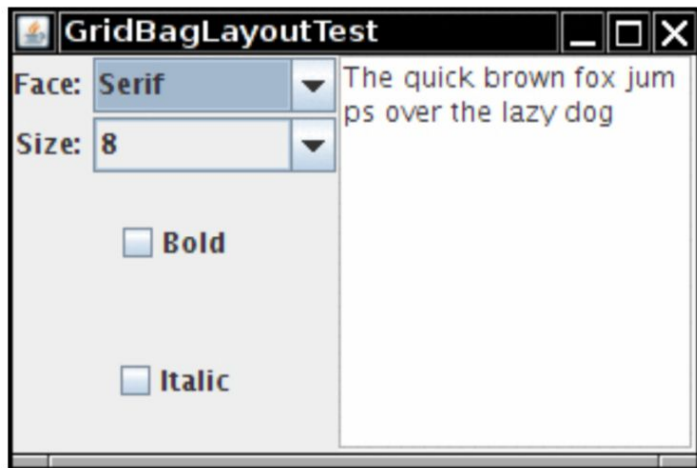


12.5.8 Tooltips

- A tooltip is activated when the cursor rests for a moment over a button.
- The tooltip text is displayed inside a colored rectangle.
- add tooltips to any JComponent simply by calling the `setToolTipText` method
`exitButton.setToolTipText("Exit");`
- if using Action objects, associate the tooltip with the `SHORT_DESCRIPTION` key:
`exitAction.putValue(Action.SHORT_DESCRIPTION, "Exit");`

Full codes: [v1ch12.toolbar](#)

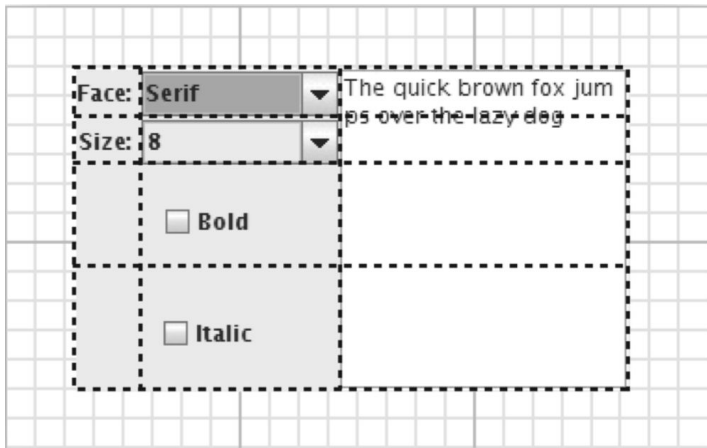
12.6 Sophisticated Layout Management



12.6.1 The Grid Bag Layout

- The grid bag layout is the mother of all layout managers.
- In a grid bag layout, the rows and columns can have variable sizes.
- You can join adjacent cells to make room for larger components.

12.6 Sophisticated Layout Management



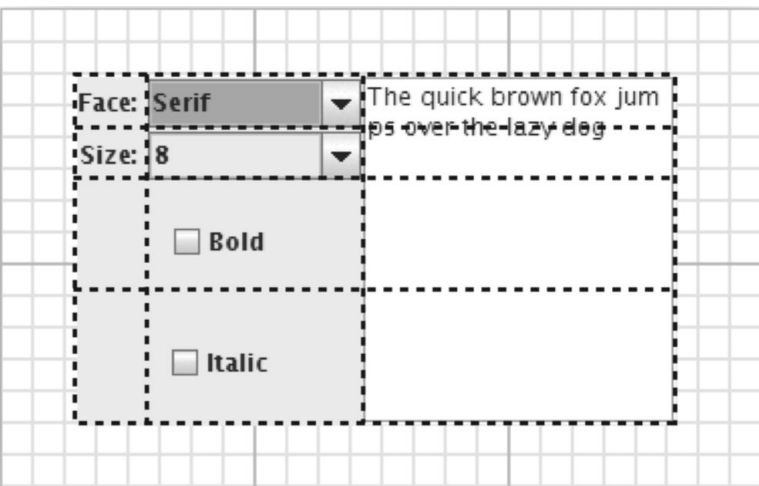
- Create an object of type GridBagLayout, the layout manager will try to guess it from the information you give it later.

`GridBagLayout layout = new GridBagLayout();`

- Set this GridBagLayout object to be the layout manager for the component.

`panel.setLayout(layout);`

12.6 Sophisticated Layout Management



- For each component, create an object of type `GridBagConstraints`.
 - weights are used to determine how to distribute space among columns (`weightx`) and among rows (`weighty`)
 - `gridx` and `gridy` values specify the column and row positions of the upper left corner of the component to be added.
 - `gridwidth` and `gridheight` values determine how many columns and rows the component occupies.

```
GridBagConstraints constraints = new GridBagConstraints();  
constraints.weightx = 100;  
constraints.weighty = 100;
```

```
constraints.gridx = 0;  
constraints.gridy = 2;  
constraints.gridwidth = 2; // 2 columns  
constraints.gridheight = 1; // 1 row
```

12.6 Sophisticated Layout Management

12.6.1.3 The fill and anchor Parameters

- If you don't want a component to stretch out and fill the entire area, set the fill constraint.
 - `GridBagConstraints.NONE`,
 - `GridBagConstraints.HORIZONTAL`,
 - `GridBagConstraints.VERTICAL`,
 - `GridBagConstraints.BOTH`.
- specify where in the area you want it by setting the anchor field
 - `GridBagConstraints.CENTER` (the default),
 - `GridBagConstraints.NORTH`,
 - `GridBagConstraints.NORTHEAST`,
 - `GridBagConstraints.EAST`

12.6 Sophisticated Layout Management



- Create an object of type `GridBagLayout`, the layout manager will try to guess it from the information you give it later.
`GridBagLayout layout = new GridBagLayout();`
- Set this `GridBagLayout` object to be the layout manager for the component.
`panel.setLayout(layout);`
- For each component, create an object of type `GridBagConstraints`.
Set field values of the `GridBagConstraints` object to specify how the components are laid out within the grid bag.
`GridBagConstraints constraints = new GridBagConstraints();`
`constraints.weightx = 100; constraints.weighty = 100;`
`constraints.gridx = 0; constraints.gridy = 2;`
`constraints.gridwidth = 2; constraints.gridheight = 1;`
- add each component with its constraints by using the call
`add(component, constraints);`
`panel.add(component, constraints);`

12.6 Sophisticated Layout Management

The following recipe makes gridbag layouts relatively trouble free:

1. Sketch out the component layout on a piece of paper.
2. Find a grid such that the small components are each contained in a cell and the larger components span multiple cells.
3. Label the rows and columns of your grid with 0, 1, 2, 3, . . . You can now read off the gridx, gridy, gridwidth, and gridheight values.
4. For each component, ask yourself whether it needs to fill its cell horizontally or vertically. If not, how do you want it aligned? This tells you the fill and anchor parameters.
5. Set all weights to 100. If you want a particular row or column to always stay at its default size, set the weightx or weighty to 0 in all components that belong to that row or column.
6. Write the code. Carefully double-check your settings for the GridBagConstraints.
7. Compile, run, and enjoy.

12.6.3 Using No Layout Manager

Drop a component at a fixed location: not a great idea for platform-independent applications, but there is nothing wrong with using it for a quick prototype.

1. Set the layout manager to null.
2. Add the component you want to the container.
3. Specify the position and size that you want:

```
frame.setLayout(null);  
JButton ok = new JButton("OK");  
frame.add(ok);  
ok.setBounds(10, 10, 30, 15);
```

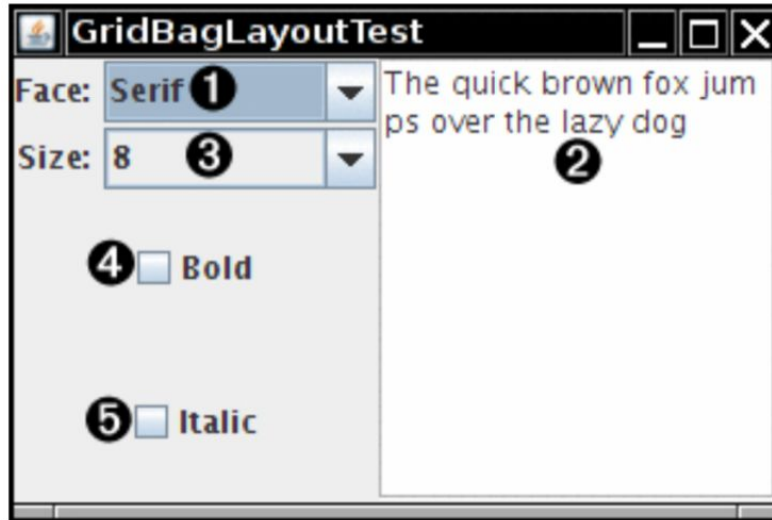

12.6.4 Custom Layout Managers

- design your own `LayoutManager` class that manages components in a special way.
- must implement the `LayoutManager` interface => override the following five methods:
 - `void addLayoutComponent(String s, Component c);`
 - `void removeLayoutComponent(Component c);`
 - `Dimension preferredLayoutSize(Container parent);`
 - `Dimension minimumLayoutSize(Container parent);`
 - `void layoutContainer(Container parent);`

Full codes: `v1ch12.circleLayout`

12.6.5 Traversal Order

- When a window is first displayed, the first component in the traversal order has the keyboard focus.
- Each time the user presses the Tab key, the next component gains focus.
- The traversal order is straightforward: first, left to right, and then, top to bottom.
- Remove a component from the focus traversal: `component.setFocusable(false);`



12.7 Dialog Boxes

- Dialog boxes give information to, or get information from, the user.
- Modal dialog boxes: users cannot interact with the remaining windows of the application; use it when you need information from the user before you can proceed with execution
- Modeless dialog boxes: the user can enter information in both the dialog box and the remainder of the application

12.7 Dialog Boxes

12.7.1 Option Dialogs

The JOptionPane has four static methods to show these simple dialogs:

- `showMessageDialog`: Show a message and wait for the user to click OK
- `showConfirmDialog`: Show a message and get a confirmation (like OK/Cancel)
- `showOptionDialog`: Show a message and get a user option from a set of options
- `showInputDialog`: Show a message and get one line of user input

12.7 Dialog Boxes

12.7.1 Option Dialogs

1. Choose the dialog type (message, confirmation, option, or input).
2. Choose the icon (error, information, warning, question, none, or custom).
3. Choose the message (string, icon, custom component, or a stack of them).
4. Choose the option type
 - a. For a confirmation dialog: (default, Yes/No, Yes/No/Cancel, or OK/Cancel).
 - b. For an option dialog: (strings, icons, or custom components) and the default option.
 - c. For an input dialog: choose between a text field and a combo box.
5. Locate the appropriate method to call in the JOptionPane API.

Full codes: `v1ch12.optionDialog`

12.7 Dialog Boxes



12.7.2 Creating Dialogs

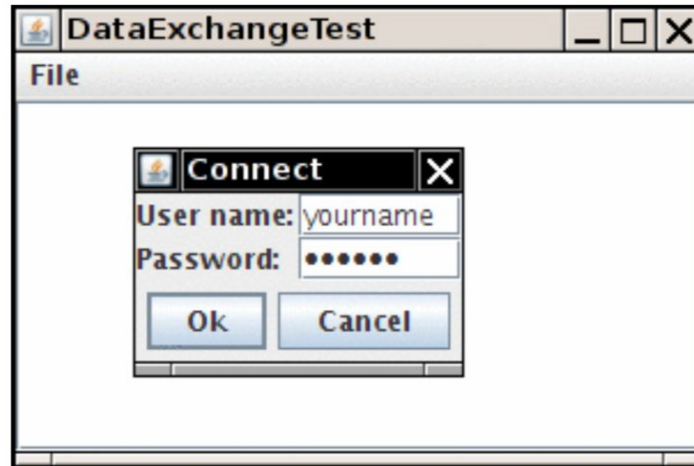
1. In the constructor of your dialog box, call the constructor of the superclass `JDialog`.
2. Add the user interface components of the dialog box.
3. Add the event handlers.
4. Set the size for the dialog box.

```
public class AboutDialog extends JDialog {  
    public AboutDialog(JFrame owner) {  
        super(owner, "About DialogTest", true);  
  
        // add label  
        add(new JLabel("Core Java\n By Cay Horstmann"), BorderLayout.CENTER);  
  
        // OK button closes the dialog  
        JButton ok = new JButton("OK");  
        ok.addActionListener(event -> setVisible(false));  
  
        // add OK button to southern border  
        JPanel panel = new JPanel();  
        panel.add(ok);  
        add(panel, BorderLayout.SOUTH);  
  
        pack();  
    }  
}
```

12.7 Dialog Boxes

12.7.3 Data Exchange

Full codes: v1ch12.dataExchange



```

public class PasswordChooser extends JPanel {
    private JTextField username;
    private JPasswordField password;
    private JButton okButton;
    private boolean ok;
    private JDialog dialog;

    public PasswordChooser() {
        setLayout(new BorderLayout());

        // construct a panel with user name and password fields
        JPanel panel = new JPanel();
        panel.setLayout(new GridLayout(2, 2));
        panel.add(new JLabel("User name:"));
        panel.add(username = new JTextField(""));
        panel.add(new JLabel("Password:"));
        panel.add(password = new JPasswordField(""));
        add(panel, BorderLayout.CENTER);

        // create Ok and Cancel buttons that terminate the dialog
        okButton = new JButton("Ok");
        okButton.addActionListener(event -> {
            ok = true;
            dialog.setVisible(false);
        });

        JButton cancelButton = new JButton("Cancel");
        cancelButton.addActionListener(event -> dialog.setVisible(false));

        // add buttons to southern border
        JPanel buttonPanel = new JPanel();
        buttonPanel.add(okButton);
        buttonPanel.add(cancelButton);
        add(buttonPanel, BorderLayout.SOUTH);
    }
}

```

```

public void setUser(User u) {
    username.setText(u.getName());
}

public User getUser() {
    return new User(username.getText(), password.getPassword());
}

public boolean showDialog(Component parent, String title) {
    ok = false;

    // locate the owner frame
    Frame owner = null;
    if (parent instanceof Frame)
        owner = (Frame) parent;
    else
        owner = (Frame) SwingUtilities.
            getAncestorOfClass(Frame.class, parent);

    // if first time, or if owner has changed, make new dialog
    if (dialog == null || dialog.getOwner() != owner) {
        dialog = new JDialog(owner, true);
        dialog.add(this);
        dialog.getRootPane().setDefaultButton(okButton);
        dialog.pack();
    }

    // set title and show dialog
    dialog.setTitle(title);
    dialog.setVisible(true);
    return ok;
}

```


12.7 Dialog Boxes

```
public class DataExchangeFrame extends JFrame {
    public static final int TEXT_ROWS = 20;
    public static final int TEXT_COLUMNS = 40;
    private PasswordChooser dialog = null;
    private JTextArea textArea;

    public DataExchangeFrame() {
        // construct a File menu
        JMenuBar mbar = new JMenuBar();
        setJMenuBar(mbar);
        JMenu fileMenu = new JMenu("File");
        mbar.add(fileMenu);

        // add Connect and Exit menu items
        JMenuItem connectItem = new JMenuItem("Connect");
        connectItem.addActionListener(new ConnectAction());
        fileMenu.add(connectItem);

        // The Exit item exits the program
        JMenuItem exitItem = new JMenuItem("Exit");
        exitItem.addActionListener(event -> System.exit(0));
        fileMenu.add(exitItem);

        textArea = new JTextArea(TEXT_ROWS, TEXT_COLUMNS);
        add(new JScrollPane(textArea), BorderLayout.CENTER);
        pack();
    }
}
```

```
private class ConnectAction implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        if (dialog == null)
            dialog = new PasswordChooser();

        // set default values
        dialog.setUser(new User("yourname", null));

        // pop up dialog
        if (dialog.showDialog(DataExchangeFrame.this,
                               "Connect")) {
            // if accepted, retrieve user input
            User u = dialog.getUser();
            textArea.append("user name = " + u.getName()
                           + ", password = "
                           + (new String(u.getPassword())) + "\n");
        }
    }
}
```

12.7 Dialog Boxes

12.7.4 File Dialogs

To put up a file dialog box and recover what the user chooses from the box:

1. Make a JFileChooser object, possible to reuse a file chooser dialog with multiple frames.
`JFileChooser chooser = new JFileChooser();`
2. Set the directory by calling the setCurrentDirectory method.
`chooser.setCurrentDirectory(new File("."));`
3. If you have a default file name that you expect the user to choose, supply it with the setSelectedFile method:
`chooser.setSelectedFile(new File(filename));`
4. To enable the user to select multiple files in the dialog, call the setMultiSelectionEnabled method.
`chooser.setMultiSelectionEnabled(true);`
5. If you want to restrict the display of files in the dialog to those of a particular type, you need to set a file filter.
`chooser.setFileFilter(new FileNameExtensionFilter("Image files", "gif", "jpg"));`

12.7 Dialog Boxes

12.7.4 File Dialogs

6. By default, a user can select only files with a file chooser. If you want the user to select directories **setFileSelectionMode** with `JFileChooser.FILES_ONLY`, `JFileChooser.DIRECTORIES_ONLY`, or `JFileChooser.FILES_AND_DIRECTORIES`
7. Show the dialog box by calling the `showOpenDialog` or `showSaveDialog` method
These calls return only when the user has approved, canceled, or dismissed the file dialog.
8. Get the selected file or files with the `getSelectedFile()` or `getSelectedFiles()` method.
`String filename = chooser.getSelectedFile().getPath();`

```

public class ImageViewerFrame extends JFrame {
    private static final int DEFAULT_WIDTH = 300;
    private static final int DEFAULT_HEIGHT = 400;
    private JLabel label;
    private JFileChooser chooser;

    public ImageViewerFrame() {
        setSize(DEFAULT_WIDTH, DEFAULT_HEIGHT);

        // set up menu bar
        JMenuBar menuBar = new JMenuBar();
        setJMenuBar(menuBar);

        JMenu menu = new JMenu("File");
        menuBar.add(menu);

        JMenuItem openItem = new JMenuItem("Open");
        menu.add(openItem);
        openItem.addActionListener(event -> {
            chooser.setCurrentDirectory(new File("."));

            // show file chooser dialog
            int result = chooser.showOpenDialog(ImageViewerFrame.this);

            // if image file accepted, set it as icon of the label
            if (result == JFileChooser.APPROVE_OPTION) {
                String name = chooser.getSelectedFile().getPath();
                label.setIcon(new ImageIcon(name));
                pack();
            }
        });
    }
}

```

```

JMenuItem exitItem = new JMenuItem("Exit");
menu.add(exitItem);
exitItem.addActionListener(event -> System.exit(0));

// use a label to display the images
label = new JLabel();
add(label);

// set up file chooser
chooser = new JFileChooser();

// accept all image files ending with .jpg, .jpeg, .gif
FileFilter filter = new FileNameExtensionFilter
    ("Image files", "jpg", "jpeg", "gif");
chooser.setFileFilter(filter);

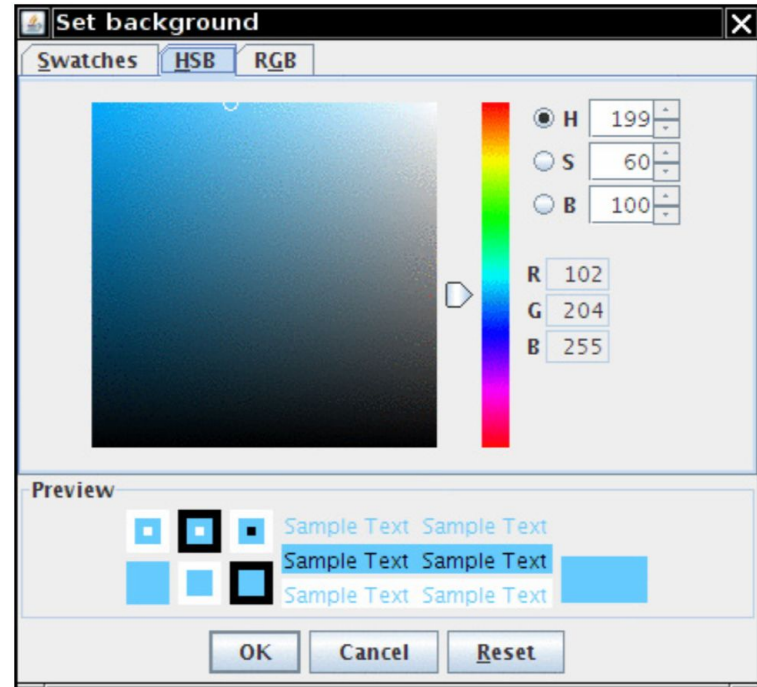
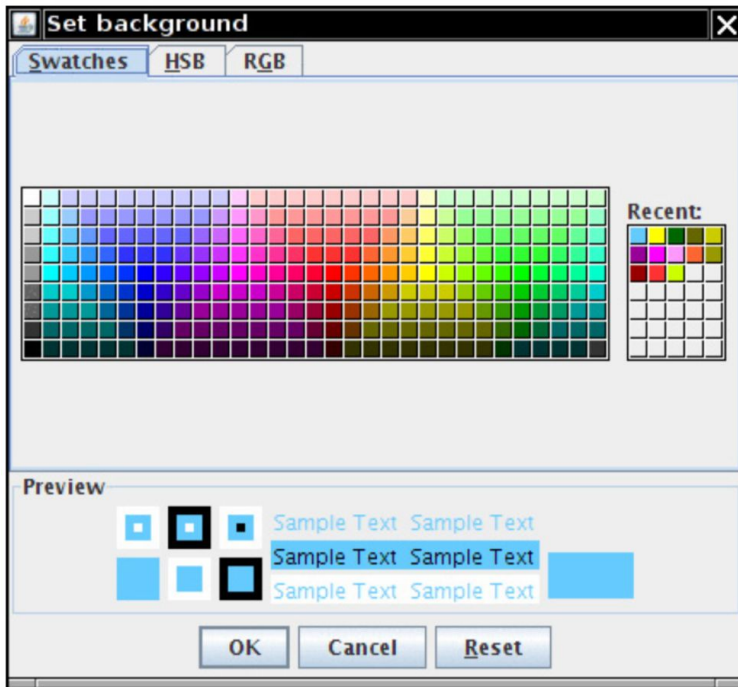
chooser.setAccessory
    (new ImagePreviewer(chooser));
}
}

```

12.7 Dialog Boxes

12.7.5 Color Choosers

- JColorChooser lets users pick a color value



12.7 Dialog Boxes

12.7.5 Color Choosers

- show a modal dialog with a color chooser:

```
Color selectedColor = JColorChooser.showDialog(parent,title, initialColor);
```

- make a modeless dialog that sets the background color when the user clicks the OK button:

```
chooser = new JColorChooser();  
dialog = JColorChooser.createDialog(  
    parent,  
    "Background Color",  
    false, /* not modal */  
    chooser,  
    event -> setBackground(chooser.getColor()),  
    null /* no Cancel button listener */  
);
```

Full codes: [v1ch12.colorChooser](#)