# Использование элементов управления и диспетчеров компоновки

Элементами управления называют компоненты, которые дают пользователю возможность по-разному взаимодействовать с прикладной программой. К числу наиболее распространенных элементов управления относится экранная кнопка.

Диспетчер компоновки автоматически размещает компоненты в контейнере. Поэтому внешний вид окна зависит как от состава элементов управления, так и от диспетчера компоновки, с помощью которого они размещаются в окне.

Библиотека Swing построена на основе библиотеки AWT. Именно поэтому библиотека AWT до сих пор является важной составной частью Java. Кроме того, в библиотеке Swing применяется тот же самый механизм обработки событий, что и в библиотеке AWT.

## Легковесные компоненты Swing

За редким исключением, компоненты библиотеки Swing являются легковесными. Это означает, что они написаны исключительно на Java и не преобразуются в равноправные компоненты для конкретной платформы. Следовательно, легковесные компоненты являются более эффективными и гибкими. Более того, легковесные компоненты не преобразуются в равноправные платформенно-ориентированные компоненты, поэтому внешний вид каждого компонента определяется библиотекой Swing, а не базовой операционной системой. Это означает, что каждый компонент будет действовать одинаково на всех платформах.

## Подключаемый стиль оформления

В библиотеке Swing поддерживается подключаемый стиль оформления. Каждый компонент библиотеки Swing воспроизводится кодом, написанным на Java, а не платформенно-ориентированными равноправными компонентами, поэтому стиль оформления компонента находится под управлением библиотеки Swing. Это означает, что стиль оформления можно отделить от логики компонента, что и делается в библиотеке Swing. Это дает следующее преимущество: изменить способ воспроизведения компонента, не затрагивая остальные его свойства. Иными словами, новый стиль оформления любого компонента можно «подключить» без побочных эффектов в коде, использующем данный компонент. Более того, можно определить наборы стилей оформления, чтобы представить разные стили оформления GUI. Чтобы воспользоваться определенным стилем оформления, достаточно «подключить» его. Как только это будет сделано, все компоненты автоматически будут воспроизводиться в подключенном стиле оформления.

У подключаемых стилей оформления имеется ряд важных преимуществ. Например, можно определить такой стиль оформления, который будет одинаковым для всех платформ. С другой стороны, можно определить стиль оформления, характерный для отдельной платформы. Например, если заранее известно, что прикладная программа будет эксплуатироваться только в среде Windows, для ее GUI можно определить стиль оформления, характерный для Windows. Кроме того, можно разработать специальный стиль оформления. И, наконец, стиль оформления можно динамически изменять во время работы прикладной программы.

В языке Java предоставляются стили оформления metal, Motif и Nimbus, которые доступны всем пользователям библиотеки Swing. Стиль оформления metal (металлический) называется также стилем оформления Java. Он не зависит от платформы и доступен во всех исполняющих средах Java. Кроме того, этот стиль оформления выбирается по умолчанию. В среде Windows доступен также стиль оформления Windows.

## Связь с архитектурой MVC

В общем, визуальный компонент определяется тремя отдельными составляющими.

Внешний вид компонента при воспроизведении на экране.

Взаимодействие с пользователем.

Сведения о состоянии компонента.

Независимо от того, какая именно архитектура используется для реализации компонента, она должна неявно включать в себя эти три его составляющие. В течение многих лет свою исключительную эффективность доказала архитектура MVC — «модель – представление – контроллер».

Успех архитектуры MVC объясняется тем, что каждая часть этой архитектуры соответствует отдельной составляющей компонента. Согласно терминологии MVC, модель соответствует сведениям о состоянии компонента. Так, для флажка модель содержит поле, которое показывает, установлен ли флажок. Представление определяет порядок отображения компонента на экране, включая любые составляющие представления,

зависящие от текущего состояния модели. Контроллер определяет порядок реагирования компонента на действия пользователя. Так, если пользователь щелкает на флажке, контроллер реагирует на это действие, изменяя модель, чтобы отразить выбор пользователя (установку или сброс флажка). В итоге представление обновляется. Разделяя компонент на модель, представление и контроллер, можно изменять конкретную реализацию любой из этих составляющих архитектуры MVC, не затрагивая остальные. Например, различные реализации представлений могут воспроизводить один и тот же компонент разными способами, но это не будет оказывать никакого влияния на модель или контроллер.

Хотя архитектура MVC и положенные в ее основу принципы выглядят вполне благоразумно, высокая степень разделения представления и контроллера не дает никаких преимуществ компонентам библиотеки Swing. Поэтому в библиотеке Swing применяется видоизмененный вариант архитектуры MVC, где представление и контроллер объединены в один логический объект, называемый представителем пользовательского интерфейса. В связи с этим подход, применяемый в библиотеке Swing, называется архитектурой «модель-представитель» или архитектурой «разделяемая модель». Таким образом, в архитектуре компонентов библиотеки Swing не используется классическая реализация архитектуры MVC, несмотря на то, что первая основывается на последней.

Подключаемый стиль оформления стал возможным в библиотеке Swing благодаря архитектуре «модель-представитель» Представление и контроллер отделены от модели, поэтому стиль оформления можно изменять, не оказывая влияния на то, как компонент применяется в программе. И наоборот, модель можно настроить, не оказывая влияния на то, как компонент отображается на экране или реагирует на действия пользователя.

Для поддержания архитектуры «модель-представитель» большинство компонентов библиотеки Swing содержит два объекта. Один из них представляет модель, а другой — представитель пользовательского интерфейса. Модели определяются в интерфейсах. Например, модель кнопки определяется в интерфейсе ButtonModel. А представители пользовательского интерфейса являются классами, наследующими от класса ComponentUI. Например, представителем пользовательского интерфейса кнопки является класс ButtonUI. Как правило, прикладные программы не взаимодействуют непосредственно с представителями пользовательского интерфейса.

## Компоненты и контейнеры

GUI, создаваемый средствами Swing, состоит из двух основных элементов: компонентов и контейнеров. Но это, по существу, концептуальное разделение, поскольку все контейнеры также являются компонентами. Отличие этих двух элементов заключается в их назначении: компонент является независимым визуальным элементом управления вроде кнопки или ползунка, а контейнер содержит группу компонентов. Таким образом, контейнер является особым типом компонента и предназначен для хранения других компонентов. Более того, компонент должен находиться в контейнере, чтобы его можно было отобразить. Так, во всех GUI, создаваемых средствами Swing, имеется как минимум один контейнер. А поскольку контейнеры являются компонентами, то один контейнер может содержать другие контейнеры. Благодаря этому в библиотеке Swing можно определить иерархию вложенности, на вершине которой должен находиться контейнер верхнего уровня. А теперь рассмотрим подробнее компоненты и контейнеры.

## Компоненты

В общем, компоненты библиотеки Swing происходят от класса JComponent. (Исключением из этого правила являются четыре контейнера верхнего уровня, о которых речь пойдет ниже). В классе JComponent предоставляются функциональные возможности, общие для всех компонентов. Так, в классе JComponent поддерживается подключаемый стиль оформления. Класс JComponent наследует классы Container и Component из библиотеки AWT. Следовательно, компонент библиотеки Swing построен на основе компонента библиотеки AWT и совместим с ним. Все компоненты Swing представлены классами, определенными в пакете javax.swing. Ниже перечислены некоторые классы компонентов Swing.

JButton, JCheckBox, JColorChooser, JComboBox, JComponent, JFrame, JLabel, JMenu, JMenuBar, JPanel, JRadioButton, JScrollBar, JTextArea, JTextField, JWindow и д. р.

Обратите внимание на то, что все классы компонентов начинаются с буквы J. Например, класс для создания метки называется JLabel, класс для создания кнопки — JButton, а класс для создания ползунка — JScrollBar.

## Контейнеры

В библиотеке Swing определены два типа контейнеров. К первому типу относятся контейнеры верхнего уровня, представленные классами JFrame, JApplet, JWindow и JDialog. Классы этих контейнеров не наследуют от класса JComponent, но они наследуют от классов Component и Container из библиотеки AWT. В отличие от остальных компонентов Swing, которые являются легковесными, компоненты верхнего уровня являются тяжеловесными. Поэтому в библиотеке Swing контейнеры являются особым случаем компонентов.

Судя по названию, контейнер верхнего уровня должен находиться на вершине иерархии контейнеров. Контейнер верхнего уровня не содержится ни в одном из других контейнеров. Более того, каждая иерархия вложенности должна начинаться с контейнера верхнего уровня. Таким контейнером в прикладных программах чаще всего является класс JFrame.

Ко второму типу контейнеров, поддерживаемых в библиотеке Swing, относятся легковесные контейнеры. Они наследуют от класса JComponent. Примером легковесного контейнера служит класс JPanel, который представляет контейнер общего назначения. Легковесные контейнеры нередко применяются для организации и управления группами связанных вместе компонентов, поскольку легковесный контейнер может находиться в другом контейнере. Следовательно, легковесные контейнеры вроде класса JPanel можно применять для создания подгрупп связанных вместе элементов управления, содержащихся во внешнем контейнере.

## Панели контейнеров верхнего уровня

Каждый контейнер верхнего уровня определяет ряд панелей. На вершине иерархии панелей находится корневая панель в виде экземпляра класса JRootPane. Класс JRootPane представляет легковесный контейнер, предназначенный для управления остальными панелями. Он также помогает управлять дополнительной, хотя и не обязательной строкой меню. Панели, составляющие корневую панель, называются прозрачной панелью, панелью содержимого и многослойной панелью соответственно.

Прозрачная панель является панелью верхнего уровня. Она находится над всеми панелями и покрывает их полностью. По умолчанию эта панель представлена прозрачным экземпляром класса JPanel. Прозрачная панель позволяет управлять событиями от мыши, оказывающими влияние на весь контейнер в целом, а не на отдельный элемент управления, или, например, рисовать поверх любого другого компонента. Как правило, обращаться к прозрачной панели непосредственно не требуется, но если она все же понадобится, то ее нетрудно обнаружить там, где она обычно находится.

Многослойная панель представлена экземпляром класса JLayeredPane. Она позволяет задать определенную глубину размещения компонентов. Глубина определяет степень перекрытия компонентов. (В связи с этим многослойные панели позволяют задавать упорядоченность компонентов по координате Z, хотя это требуется не всегда). На многослойной панели находится панель содержимого и дополнительно, хотя и не обязательно, — строка меню. Несмотря на то, что прозрачная и многослойная панели являются неотъемлемыми частями контейнера верхнего уровня и служат для разных целей, большая часть их возможностей скрыта от пользователей. Прикладная программа чаще всего будет обращаться к панели содержимого, поскольку именно на ней обычно располагаются визуальные компоненты. Иными словами, когда компонент (например, кнопка) вводится в контейнер верхнего уровня, он оказывается на панели содержимого. По умолчанию панель содержимого представлена непрозрачным экземпляром класса JPanel.

## Пакеты библиотеки Swing

Библиотека Swing — это довольно крупная подсистема, в которой задействовано большое количество пакетов. Самым главным среди них является пакет javax.swing. Его следует импортировать в любую прикладную программу, пользующуюся библиотекой Swing. В этом пакете содержатся классы, реализующие базовые компоненты Swing, в том числе кнопки, метки и флажки.

## Простое Swing приложение

Несмотря на всю краткость рассматриваемого здесь примера программы, он наглядно демонстрирует один из способов разработки Swing-приложения, а также основные средства библиотеки Swing. В данном примере используются два компонента Swing: JFrame и JLabel. Класс JFrame представляет контейнер верхнего уровня, который обычно применяется в Swing-приложениях, а класс JLabel компонент Swing, создающий метку для отображения информации. Метка является самым простым компонентом Swing, поскольку это пассивный компонент. Это означает, что метка не реагирует на действия пользователя. Она служит лишь для отображения выводимых данных. В данном примере контейнер типа JFrame служит для хранения метки в виде экземпляра класса JLabel. Метка отображает краткое текстовое сообщение.

```
//пример простого Swing-приложения
import javax.swing.*;

class SwingDemo {
      SwingDemo() {
            //создать новый контейнер типа JFrame
            JFrame jfrm = new JFrame("A Simple Swing Application");
            //задать исходные размеры фрейма
            jfrm.setSize(275,100);
            //завершить работу, если пользователь
            //закрывает приложение
```

```
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //создать метку с текстом сообщения
            JLabel jlab = new JLabel("Swing means powerful GUIs.");
            //ввести метку на панели содержимого
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      public static void main(String args[]) {
            //создать фрейм
            new SwingDemo();
      }
}
```

Пример приложения SwingDemo демонстрирует ряд основных понятий библиотеки Swing, поэтому рассмотрим этот пример построчно. Данное Swing-приложение начинается с импорта пакета javax.swing. Как упоминалось ранее, этот пакет содержит компоненты и модели, определяемые в библиотеке Swing. Так, в пакете javax.swing определяются классы, реализующие метки, кнопки, текстовые элементы управления и меню. Поэтому этот пакет обычно включается во все программы, пользующиеся библиотекой Swing. Затем объявляются класс SwingDemo и его конструктор, в котором выполняется большинство действий данной программы. Сначала создается экземпляр класса JFrame, как показано ниже.

```
JFrame jfrm = new JFrame("A Simple Swing Application");
```

В методе setSize(), наследуемом классом JFrame от класса Component из библиотеки AWT, задаются размеры окна в пикселях. Ниже приведена общая форма этого метода. В данном примере задается ширина окна 275 пикселей, а высота — 100 пикселей.

```
void setSize(int ширина, int высота)
```

Когда закрывается окно верхнего уровня (например, после того, как пользователь щелкнет на кнопке закрытия), по умолчанию окно удаляется с экрана, но работа приложения не прекращается. И хотя такое стандартное поведение иногда оказывается полезным, для большинства приложений оно не подходит. Чаще всего при закрытии окна верхнего уровня требуется завершить работу всего приложения в целом. Это можно сделать двумя способами. Самый простой из них состоит в том, чтобы вызвать метод setDefaultCloseOperation(),что и делается в данном приложении следующим образом:

```
jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

В результате вызова этого метода приложение полностью завершает свою работу при закрытии окна. Общая форма метода setDefaultCloseOperation() выглядит следующим образом:

```
void setDefaultCloseOperation(int что)
```

Значение константы, передаваемое в качестве параметра что, определяет, что именно происходит при закрытии окна. Помимо константы JFrame.EXIT_ON_CLOSE, имеются также следующие константы:

```
DISPOSE_ON_CLOSE
HIDE_ON_CLOSE
DO_NOTHING_ON_CLOSE
```

Имена этих констант отражают выполняемые действия. Эти константы объявляются в интерфейсе WindowConstants, который определяется в пакете javax.swing и реализуется в классе JFrame. В следующей строке кода создается компонент типа JLabel из библиотеки Swing:

```
JLabel jlab = new JLabel("Swing means powerful GUIs.");
```

Класс JLabel определяет метку как самый простой в употреблении компонент, поскольку он не принимает вводимые пользователем данные, а только отображает информацию в виде текста, значка или того и другого. В данном примере создается метка, которая содержит только текст, передаваемый конструктору класса JLabel. В следующей строке кода метка вводится на панели содержимого фрейма:

```
jfrm.add(jlab);
```

Как пояснялось ранее, у всех контейнеров верхнего уровня имеется панель содержимого, на которой размещаются отдельные компоненты. Следовательно, чтобы ввести компонент во фрейм, его нужно ввести на

панели содержимого фрейма. Для этого достаточно вызвать метод add() по ссылке на экземпляр класса JFrame (в данном случае jfrm). Ниже приведена общая форма метода add(). Метод add() наследуется классом JFrame от класса Container из библиотеки AWT.

```
Component add(Component компонент)
```

По умолчанию на панели содержимого, связанной с компонентом типа JFrame, применяется граничная компоновка. В приведенной выше форме метода add() метка вводится по центру панели содержимого. Другие формы метода add() позволяют задать одну из граничных областей. Когда компонент вводится по центру, его размеры автоматически подгоняются таким образом, чтобы он разместился в центре.

Последний оператор в конструкторе класса SwingDemo требуется для того, чтобы сделать окно видимым, как показано ниже.

```
jfrm.setVisible(true);
```

Метод setVisible() наследуется от класса Component из библиотеки AWT. Если его аргумент принимает логическое значение true, то окно отобразится, а иначе оно будет скрыто. По умолчанию контейнер типа JFrame невидим, поэтому придется вызвать метод setVisible(true), чтобы показать его. В методе main() создается объект типа SwingDemo, чтобы отобразить окно и метку.

```
new SwingDemo();
```

## Обработка событий

В предыдущем примере была продемонстрирована основная форма Swing-приложения, но ей недостает одной важной части: обработки событий. Компонент типа JLabel не принимает данные, вводимые пользователем, и не генерирует события, и поэтому обработка событий в данном примере не требовалась. Но остальные компоненты библиотеки Swing реагируют на вводимые пользователем данные, а, следовательно, требуется каким-то образом обработать события, наступающие в результате подобных взаимодействий. Событие происходит, когда, например, срабатывает таймер. Так или иначе, обработка событий занимает большую часть любого Swing-приложения. Механизм обработки событий, применяемый в библиотеке Swing, ничем не отличается от аналогичного механизма из библиотеки AWT, называемого моделью делегирования событий. Как правило, в библиотеке Swing используются те же самые события, что и в библиотеке AWT, и эти события определены в пакете java.awt.event. А события, характерные только для библиотеки Swing, определены в пакете javax.swing.event. Несмотря на то, что события обрабатываются в библиотеке Swing таким же образом, как и в библиотеке AWT, этот процесс лучше всего рассмотреть на простом примере. В приведенной ниже программе обрабатывается событие, генерируемое после щелчка на экранной кнопке, определяемой соответствующим компонентом Swing.

```
//обработка события в Swing-приложении
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class EventDemo {
      JLabel jlab;

      EventDemo() {
            //создать новый контейнер типа JFrame
            JFrame jfrm = new JFrame("An Event Example");
            //определить диспетчер поточной
            //компоновки типа FlowLayout
            jfrm.setLayout(new FlowLayout());
            //установить исходные размеры фрейма
            jfrm.setSize(220, 90);
            //завершить работу приложения, если
            //пользователь закрывает его окно
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            //создать две кнопки Jbutton
            JButton jbtnAlpha = new JButton("Alpha");
            JButton jbtnBeta = new JButton("Beta");
            //ввести приемник действий от кнопки Alpha
            jbtnAlpha.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        jlab.setText("Alpha was pressed.");
                  }
            });
```

```
            //ввести приемник действий от кнопки Beta
            jbtnBeta.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ae) {
                        jlab.setText("Beta was pressed.");
                  }
            });
            //ввести кнопки на панели содержимого
            jfrm.add(jbtnAlpha);
            jfrm.add(jbtnBeta);
            //создать текстовую метку
            jlab = new JLabel("Press a button.");
            //ввести метку на панели содержимого
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      public static void main(String args[]) {
            //создать фрейм
            new EventDemo();
      }
}
```

Прежде всего обратите внимание на то, что в данном примере программы теперь импортируются пакеты java.awt и java.awt.event. Пакет java.awt требуется потому, что в нем содержится класс FlowLayout, поддерживающий стандартный диспетчер поточной компоновки, который применяется для размещения компонентов во фрейме. (Описание диспетчеров компоновки см. ниже). А пакет java.awt.event требуется потому, что в нем определяются интерфейс ActionListener и класс ActionEvent. Сначала в конструкторе класса EventDemo создается контейнер jfrm типа JFrame, а затем устанавливается диспетчер компоновки типа FlowLayout для панели содержимого контейнера jfrm. По умолчанию для размещения компонентов на панели содержимого применяется диспетчер компоновки типа BorderLayout, но для данного примера больше подходит диспетчер компоновки типа FlowLayout.

После определения размеров и стандартной операции, выполняемой при закрытии окна, в конструкторе класса EventDemo создаются две экранные кнопки, как показано ниже.

```
JButton jbtnAlpha = new JButton("Alpha");
JButton jbtnBeta = new JButton("Beta");
```

Первая кнопка будет содержать надпись «Alpha», а вторая — надпись «Beta». Экранные кнопки из библиотеки Swing являются экземплярами класса JButton. В классе JButton предоставляется несколько конструкторов. Здесь используется приведенный ниже конструктор, где параметр сообщение обозначает символьную строку, которая будет отображаться в виде надписи на экранной кнопке.

```
JButton(String сообщение)
```

После щелчка на экранной кнопке наступает событие типа ActionEvent. Поэтому в классе JButton предоставляется метод addActionListener(), который служит для ввода приемника подобных событий. (В классе JButton предоставляется также метод removeActionListener() для удаления приемника событий, но в рассматриваемом здесь примере программы он не применяется). В интерфейсе ActionListener определяется единственный метод actionPerformed(). Ниже приводится его общая форма.

```
void actionPerformed(ActionEvent событие_действия)
```

Этот метод вызывается после щелчка на экранной кнопке. Иными словами, это обработчик, который вызывается, когда наступает событие нажатия экранной кнопки. Далее вводится приемник событий от двух экранных кнопок, в данном случае используются анонимные внутренние классы, чтобы предоставить обработчики событий от двух экранных кнопок.

Всякий раз, когда нажимается экранная кнопка, символьная строка, отображаемая на месте метки jlab, изменяется в зависимости от того, какая кнопка была нажата. Затем кнопки вводятся на панели содержимого следующим образом:

```
jfrm.add(jbtnAlpha);
jfrm.add(jbtnBeta);
```

И, наконец, на панели содержимого вводится метка jlab, и окно приложения становится видимым.

Если после запуска данного Swing-приложения щелкнуть на любой из двух экранных кнопок, то на месте метки отобразится сообщение, извещающее, какая именно кнопка была нажата.

## Кнопки из библиотеки Swing

В библиотеке Swing определены четыре класса экранных кнопок: JButton, JToggleButton, JCheckBox и JRadioButton. Все они являются производными от класса AbstractButton, расширяющего класс JComponent. Таким образом, у экранных кнопок имеются общие характеристики. Класс AbstractButton содержит немало методов, позволяющих управлять поведением экранных кнопок. С их помощью можно, например, определить различные значки, которые будут отображаться на месте экранной кнопки, когда она отключена, нажата или выбрана. Другой значок можно использовать для динамической подстановки, чтобы он отображался при наведении указателя мыши на экранную кнопку. Ниже приведены общие формы методов, с помощью которых можно задавать эти значки.

```
void setDisabledIcon(Icon di)
void setPressedIcon(Icon pi)
void setSelectedIcon(Icon si)
void setRolloverIcon(Icon ri)
```

Параметры di, pi, si и ri определяют значки, используемые для обозначения различных состояний экранной кнопки. Текст, связанный с экранной кнопкой, можно прочитать и записать с помощью приведенных ниже методов, где параметр строка обозначает текст надписи на экранной кнопке.

```
String getText()
void setText(String строка)
```

Модель, применяемая во всех экранных кнопках, определяется в интерфейсе ButtonModel. Экранная кнопка генерирует событие действия, когда ее нажимает пользователь. Возможны и другие события.

## Класс JButton

В классе JButton определяются функциональные возможности экранной кнопки. Простая форма конструктора этого класса была представлена ранее. Компонент типа JButton позволяет связать с экранной кнопкой значок, символьную строку или же и то и другое. Ниже приведены три конструктора класса JButton, где параметры значок и строка обозначают соответственно значок и строку, используемые для представления экранной кнопки.

```
JButton(Icon значок)
JButton(String строка)
JButton(String строка, Icon значок)
```

Когда пользователь нажимает экранную кнопку, наступает событие типа ActionEvent. Используя объект типа ActionEvent, передаваемый методу actionPerformed() зарегистрированного приемника действий типа ActionListener, можно получить символьную строку с командой действия, связанной с данной кнопкой. По умолчанию эта символьная строка отображается в пределах экранной кнопки. Но команду действия можно также задать, вызвав метод setActionCommand() для экранной кнопки. А получить команду действия можно, вызвав метод getActionCommand() для объекта события. Этот метод объявляется следующим образом:

```
String getActionCommand()
```

Команда действия обозначает экранную кнопку. Так, если в одном приложении используются две или больше экранных кнопок, команда действия позволяет легко определить, какая именно кнопка была нажата.

## КлассJToggleButton

Полезной разновидностью экранной кнопки является переключатель. Эта кнопка похожа на обычную экранную кнопку, но действует иначе, поскольку может находиться в двух состояниях: нажатом и отпущенном. После щелчка на переключателе он остается нажатым, а не отпускается, как обычная экранная кнопка. Если после этого щелкнуть на переключателе еще раз, он отпускается. Таким образом, всякий раз, когда пользователь щелкает кнопкой мыши на переключателе, он переходит в одно из двух возможных состояний.

Переключатели определяются объектами класса JToggleButton, производного от класса AbstractButton. Помимо создания стандартных переключателей, класс JToggleButton служит суперклассом для двух других компонентов Swing, которые также представляют элементы управления, имеющие два состояния. Это классы JCheckBox и JRadioButton. Таким образом, класс JToggleButton определяет базовые функции всех компонентов, имеющих два состояния.

В классе JToggleButton определяется несколько конструкторов. Ниже приведен один из конструкторов.

```
JToggleButton(String строка)
```

Этот конструктор создает переключатель с текстом надписи, задаваемым в качестве параметра строка. Стандартным является отпущенное состояние переключателя. Остальные конструкторы данного класса позволяют создавать переключатели с изображением или текстом надписи и изображением. В классе JToggleButton применяется модель, определяемая во вложенном классе JToggleButton.ToggleButtonModel. Как правило, обращаться непосредственно к этой модели не требуется, чтобы воспользоваться стандартным переключателем. Как и компонент типа JButton, компонент типа JToggleButton генерирует событие действия всякий раз, когда пользователь щелкает на переключателе. Но, в отличие от класса JButton, компонент типа JToggleButton генерирует также событие от выбираемого элемента. Это событие используется теми компонентами, которые действуют по принципу выбора. Если переключатель типа JToggleButton нажат, он считается выбранным. Если же пользователь отпускает нажатый переключатель, выбор отменяется. Для обработки событий в выбираемых элементах следует реализовать интерфейс ItemListener. Всякий раз, когда генерируется событие от выбираемого элемента, оно передается методу itemStateChanged(), определяемому в интерфейсе ItemListener. Из метода itemStateChanged() может быть вызван метод getItem() для объекта типа ItemEvent, чтобы получить ссылку на экземпляр класса JToggleButton, сгенерировавший данное событие. Ниже приведена общая форма метода getItem().

```
Object getItem()
```

Этот метод возвращает ссылку на экранную кнопку. Эту ссылку следует привести к типу JToggleButton. Чтобы определить состояние переключателя, проще всего вызвать метод isSelected(), наследуемый из класса AbstractButton, для экранной кнопки, сгенерировавшей событие. Ниже приведена общая форма метода isSelected(). Этот метод возвращает логическое значение true, если экранная кнопка выбрана, а иначе — логическое значение false.

```
boolean isSelected()
```

В приведенном ниже примере прикладной программы демонстрируется применение переключателя. Обратите внимание на приемник событий, который просто вызывает метод isSelected(), чтобы определить состояние кнопки.

```
//продемонстрировать применение компонента типа JToggleButton
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JToggleButtonDemo {
      public JToggleButtonDemo() {
            //установить фрейм средствами класса JFrame
            JFrame jfrm = new JFrame("JToggleButtonDemo");
            jfrm.setLayout(new FlowLayout());
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jfrm.setSize(200, 100);
            //создать метку
            final JLabel jlab = new JLabel("Button is off.");
            //создать переключатель
            final JToggleButton jtbn = new JToggleButton("On/Off");
            //ввести приемник событий от переключателя
            jtbn.addItemListener(new ItemListener() {
                  public void itemStateChanged(ItemEvent ie) {
                        if(jtbn.isSelected())
                              jlab.setText("Button is on.");
                        else
                              jlab.setText("Button is off.");
                  }
            });
            //ввести переключатель и метку на панель содержимого
            jfrm.add(jtbn);
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      public static void main(String[] args) {
            // создать фрейм
            new JToggleButtonDemo();
      }
```

```
}
```

## Флажки

Класс JCheckBox определяет функции флажка. Его суперклассом служит класс JToggleButton, поддерживающий кнопки с двумя состояниями, как пояснялось выше. В классе JCheckBox определяется ряд конструкторов. Один из них выглядит следующим образом:

```
JCheckBox(String строка)
```

Этот конструктор создает флажок с текстом метки, определяемым в качестве параметра строка. Остальные конструкторы позволяют определить исходное состояние выбора флажка и указать значок. Если пользователь устанавливает или сбрасывает флажок, генерируется событие типа ItemEvent. Чтобы получить ссылку на компонент типа JCheckBox, сгенерировавший событие, следует вызвать метод getItem() для объекта типа ItemEvent, который передается в качестве события от выбранного элемента методу itemStateChanged(), определяемому в интерфейсе ItemListener. Определить выбранное состояние флажка проще всего, вызвав метод isSelected() для экземпляра класса JCheckBox. В приведенном ниже примере прикладной программы демонстрируется применение флажков. В окне данной программы отображаются четыре флажка и метка. Когда пользователь щелкает кнопкой мыши на флажке, наступает событие типа ItemEvent. Из метода itemStateChanged() вызывается метод getItem() для получения ссылки на объект типа JCheckBox, сгенерировавший событие от выбранного элемента. Далее вызывается метод isSelected() с целью определить установленное или сброшенное состояние флажка. А метод getText() вызывается с целью получить текст, выводимый на месте метки данного флажка.

```
//продемонстрировать применение компонента типа JCheckbox
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JCheckBoxDemo implements ItemListener {
      JLabel jlab;

      public JCheckBoxDemo() {
            //установить фрейм средствами класса JFrame
            JFrame jfrm = new JFrame("JCheckBoxDemo");
            jfrm.setLayout(new FlowLayout());
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jfrm.setSize(250, 100);
            //ввести флажки на панели содержимого
            JCheckBox cb = new JCheckBox("C");
            cb.addItemListener(this);
            jfrm.add(cb);
            cb = new JCheckBox("C++");
            cb.addItemListener(this);
            jfrm.add(cb);
            cb = new JCheckBox("Java");
            cb.addItemListener(this);
            jfrm.add(cb);
            cb = new JCheckBox("Perl");
            cb.addItemListener(this);
            jfrm.add(cb);
            //создать метку и ввести ее на панели содержимого
            jlab = new JLabel("Select languages");
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      //обработать события от выбираемых элементов,
      //наступающие при установке и сбросе флажков
      public void itemStateChanged(ItemEvent ie) {
            JCheckBox cb = (JCheckBox)ie.getItem();
            if(cb.isSelected())
                  jlab.setText(cb.getText() + " is selected");
            else
                  jlab.setText(cb.getText() + " is cleared");
      }
```

```
	public static void main(String[] args) {
		//создать фрейм
		new JCheckBoxDemo();
	}
}
```

## Кнопки-переключатели

Кнопки-переключатели образуют группу взаимоисключающих экранных кнопок, из которых можно выбрать только одну. Они поддерживаются в классе JRadioButton, расширяющем класс JToggleButton. В классе JRadioButton предоставляется несколько конструкторов. Ниже приведен один из них.

```
JRadioButton(String строка)
```

Здесь параметр строка обозначает метку кнопки-переключателя. Остальные конструкторы позволяют определить исходное состояние кнопки-переключателя и указать для нее значок.

Кнопки-переключатели следует объединить в группу, где можно выбрать только одну из них. Так, если пользователь выбирает какую-нибудь кнопку-переключатель из группы, то кнопка-переключатель, выбранная ранее в этой группе, автоматически выключается. Для создания группы кнопок-переключателей служит класс ButtonGroup. С этой целью вызывается его конструктор по умолчанию. После этого в группу можно ввести отдельные кнопки-переключатели с помощью приведенного ниже метода, где параметр ab обозначает ссылку на кнопку-переключатель, которую требуется ввести в группу.

```
void add(AbstractButton ab)
```

Компонент типа JRadioBut ton генерирует события действия, события от выбираемых элементов и события изменения всякий раз, когда выбирается другая кнопка-переключатель в группе. Зачастую обрабатывается событие действия, а это, как правило, означает необходимость реализовать интерфейс ActionListener, в котором определяется единственный метод actionPerformed(). В этом методе можно несколькими способами выяснить, какая именно кнопка-переключатель была выбрана. Во-первых, можно проверить ее команду действия, вызвав метод getActionCommand(). По умолчанию команда действия аналогична метке кнопки, но, вызвав метод setActionCommand() для кнопки-переключателя, можно задать какую-нибудь другую команду действия. Во-вторых, можно вызвать метод getSource() для объекта типа ActionEvent и проверить ссылку по отношению к кнопкам-переключателям. И, наконец, для каждой кнопки можно вызвать свой обработчик событий действия, реализуемый в виде анонимного класса. Не следует, однако, забывать, что всякий раз, когда наступает событие действия, оно означает, что выбранная кнопка-переключатель была изменена и что была выбрана одна и только одна кнопка-переключатель.

В приведенном ниже примере прикладной программы демонстрируется применение кнопок-переключателей. В данной программе создаются и объединяются в группу три кнопки-переключателя. Как пояснялось ранее, это требуется для того, чтобы они действовали, взаимно исключая друг друга. При выборе кнопки-переключателя наступает событие действия, которое обрабатывается методом actionPerformed(). В этом обработчике событий метод getActionCommand() получает текст, связанный с кнопкой-переключателем, чтобы отобразить его на месте метки.

```
//продемонстрировать применение компонента
//типа JRadioButton
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JRadioButtonDemo implements ActionListener {
	JLabel jlab;

	public JRadioButtonDemo() {
		//установить фрейм средствами класса JFrame
		JFrame jfrm = new JFrame("JRadioButtonDemo");
		jfrm.setLayout(new FlowLayout());
		jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		jfrm.setSize(250, 100);
		//создать кнопки-переключатели и ввести
		//их на панели содержимого
		JRadioButton b1 = new JRadioButton("A");
		b1.addActionListener(this);
		jfrm.add(b1);
		JRadioButton b2 = new JRadioButton("B");
		b2.addActionListener(this);
		jfrm.add(b2);
```

```
            JRadioButton b3 = new JRadioButton("C");
            b3.addActionListener(this);
            jfrm.add(b3);
            //определить группу кнопок
            ButtonGroup bg = new ButtonGroup();
            bg.add(b1);
            bg.add(b2);
            bg.add(b3);
            //создать метку и ввести ее на панели содержимого
            jlab = new JLabel("Select One");
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      //обработать событие выбора кнопки-переключателя
      public void actionPerformed(ActionEvent ae) {
            jlab.setText("You selected " + ae.getActionCommand());
      }
      public static void main(String[] args) {
            //создать фрейм
            new JRadioButtonDemo();
      }
}
```

## Класс JScrollPane

Класс JScrollPane представляет легковесный контейнер, автоматически выполняющий прокрутку другого компонента. Прокручиваться может как отдельный компонент (например, таблица), так и группа компонентов, содержащихся в другом легковесном контейнере, например JPanel. Но в любом случае прокручиваемый компонент дополняется горизонтальной и/или вертикальной полосой прокрутки, если он больше области просмотра, что позволяет прокручивать компонент в пределах панели.

Класс JScrollPane автоматизирует процесс прокрутки, избавляя от необходимости управлять отдельными полосами прокрутки. Просматриваемая область панели с полосами прокрутки называется окном просмотра. Это окно, в котором отображается прокручиваемый компонент. Таким образом, в окне просмотра будет показана видимая часть прокручиваемого компонента. Полосы прокрутки служат для прокручивания компонента в окне просмотра. По умолчанию класс JScrollPane динамически добавляет или удаляет полосу прокрутки по мере надобности. Так, если компонент оказывается больше по высоте, чем окно просмотра, то оно дополняется вертикальной полосой прокрутки. А если компонент полностью размещается в окне просмотра, то полосы прокрутки исключаются.

В классе JScrollPane определяется несколько конструкторов. Ниже приведен один из них.

```
JScrollPane(Component компонент)
```

Прокручиваемый компонент указывается в качестве параметра компонент. Полосы прокрутки автоматически отображаются, если содержимое панели превышает размеры окна просмотра. Чтобы воспользоваться панелью с полосами прокрутки, достаточно выполнить следующие действия.

Создать прокручиваемый компонент.

Создать экземпляр класса JScrollPane, передав ему прокручиваемый компонент как объект.

Ввести панель с полосами прокрутки на панели содержимого.

В приведенном ниже примере прикладной программы демонстрируется применение панели с полосами прокрутки. Сначала в данной программе создается панель в виде объекта типа JPanel, а затем на этой панели вводится 400 кнопок, размещаемых в 20 столбцах. Далее эта панель вводится на панели с полосами прокрутки, а последняя — на панели содержимого. Эта панель оказывается больше окна просмотра, поэтому она автоматически дополняется вертикальной и горизонтальной полосами прокрутки. Полосы прокрутки служат для прокручивания кнопок в окне просмотра.

```
//продемонстрировать применение компонента
//типа JScrollPane
import java.awt.*;
import javax.swing.*;
```

```
public class JScrollPaneDemo {
      public JScrollPaneDemo() {
            //установить фрейм средствами класса JFrame
            //использовать выбираемый по умолчанию диспетчер
            //граничной компоновки типа BorderLayout
            JFrame jfrm = new JFrame("JScrollPaneDemo");
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jfrm.setSize(400, 400);
            //создать панель и ввести на ней 400 экранных кнопок
            JPanel jp = new JPanel();
            jp.setLayout(new GridLayout(20, 20));
            int b = 0;
            for(int i = 0; i < 20; i++)
                  for(int j = 0; j < 20; j++)
                  {
                        jp.add(new JButton("Button " + b));
                        ++b;
                  }
            //создать панель с полосами прокрутки
            JScrollPane jsp = new JScrollPane(jp);
            //ввести панель с полосами прокрутки на панели
            //содержимого. По умолчанию выполняется граничная
            //компоновка, и поэтому вводимая панель с полосами
            //прокрутки располагается по центру
            jfrm.add(jsp, BorderLayout.CENTER);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      public static void main(String[] args) {
            //создать фрейм
            new JScrollPaneDemo();
      }
}
```

## Класс JList

Базовым для составления списков в Swing служит класс JList. В этом классе поддерживается выбор одного или нескольких элементов из списка. Зачастую список состоит из символьных строк, но ничто не мешает составить список из любых объектов, которые только можно отобразить. Раньше элементы списка типа JList были представлены ссылками на класс Object. Но, начиная с версии JDK 7, класс JList стал обобщенным, и теперь он объявляется приведенным ниже образом, где параметр E обозначает тип элементов в списке.

```
class JList<E>
```

В классе JList предоставляется несколько конструкторов. Ниже приведен один из наиболее употребительных конструкторов данного класса. Этот конструктор создает список типа JList, содержащий элементы в массиве, обозначаемом в качестве параметра элементы.

```
JList(E[] элементы)
```

Класс JList основывается на двух моделях. Первая модель определяется в интерфейсе ListModel и устанавливает порядок доступа к данным в списке. Вторая модель определяется в интерфейсе ListSelectionModel, где объявляются методы, позволяющие выявить выбранный из списка элемент (или ряд элементов).

Несмотря на то, что компонент типа JList вполне способен действовать самостоятельно, он обычно размещается на панели типа JScrollPane. Благодаря этому длинные списки становятся автоматически прокручиваемыми. Этим упрощается не только построение GUI, но и изменение количества записей в списке, не требуя изменять размеры компонента типа JList.

Компонент типа JList генерирует событие типа ListSelectionEvent, когда пользователь выбирает элемент или изменяет выбор элемента в списке. Это событие наступает и в том случае, если пользователь отменяет выбор элемента. Оно обрабатывается приемником событий, реализующим интерфейс ListSelectionListener, в котором определяется единственный метод valueChanged():

```
void valueChanged(ListSelectionEvent событие_списка)
```

Здесь параметр событие_списка обозначает ссылку на событие, наступающее в списке. И хотя в классе ListSelectionEvent предоставляются свои методы для выявления событий, наступающих при выборе элементов из списка, как правило, для этой цели достаточно обратиться непосредственно к объекту типа JList.

Класс ListSelectionEvent и интерфейс ListSelectionListener определены в пакете javax.swing.event. По умолчанию компонент типа JList позволяет выбирать несколько элементов из списка, но это поведение можно изменить, вызвав метод setSelectionModel(), определяемый в классе JList. Ниже приведена его общая форма.

```
void setSelectionMode(int режим)
```

Здесь параметр режим обозначает заданный режим выбора. Этот параметр должен принимать значение одной из следующих констант, определенных в интерфейсе ListSelectionModel:

```
SINGLE_SELECTION
SINGLE_INTERVAL_SELECTION
MULTIPLE INTERVAL SELECTION
```

По умолчанию выбирается значение последней константы, позволяющее выбирать несколько интервалов элементов из списка. Если же задан режим выбора в одном интервале (константа SINGLE_INTERVAL_SELECTION), то выбрать можно только один ряд элементов из списка. А если задан режим выбора одного элемента (константа SINGLE_SELECTION), то выбрать можно только один элемент из списка. Разумеется, один элемент можно выбрать и в двух других режимах, но эти режимы позволяют также выбирать несколько элементов из списка.

Вызвав метод getSelectedIndex(), можно получить индекс первого выбранного элемента, который оказывается также индексом единственного выбранного элемента в режиме SINGLE_SELECTION. Ниже приведена общая форма метода getSelectedIndex().

```
int getSelectedIndex()
```

Индексация элементов списка начинается с нуля. Так, если выбран первый элемент, метод getSelectedIndex() возвращает нулевое значение. А если не выбрано ни одного элемента, то возвращается значение -1. Вместо того чтобы получать индекс выбранного элемента, можно получить значение, связанное с выбранным элементом, вызвав метод getSelectedValue(). Ниже приведена общая форма этого метода.

```
E getSelectedValue()
```

Этот метод возвращает ссылку на первое выбранное значение. Если не выбрано ни одного значения, то возвращается пустое значение null.

В приведенном ниже примере прикладной программы демонстрируется применение простого компонента типа JList, содержащего список городов. Всякий раз, когда из этого списка выбирается город, наступает событие типа ListSelectionEvent, обрабатываемое методом valueChanged(), определяемым в интерфейсе ListSelectionListener. Этот метод получает индекс выбранного элемента и отображает имя выбранного города на месте метки.

```
//продемонстрировать применение компонента типа JList
import javax. swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;

public class JListDemo {
      //создать массив из названий городов
      String Cities [] = {"New York", "Chicago", "Houston",
                          "Denver", "Los Angeles", "Seattle",
                          "London", "Paris", "New Delhi",
                          "Hong Kong", "Tokyo", "Sydney"};

      public JListDemo() {
            //установить фрейм средствами класса JFrame
            JFrame jfrm = new JFrame("JListDemo");
            jfrm.setLayout(new FlowLayout());
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            jfrm.setSize(200, 200);
            //создать список на основе компонента типа Jlist
            final JList<String> jlst = new JList<String>(Cities);
            //задать режим выбора единственного элемента из списка
            jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
            //ввести список на панели с полосами прокрутки
            JScrollPane jscrlp = new JScrollPane(jlst);
            //задать предпочтительные размеры панели
            //с полосами прокрутки
            jscrlp.setPreferredSize(new Dimension(120, 90));
            //создать метку для отображения выбранного города
            final JLabel jlab = new JLabel("Choose a City");
            //ввести приемник событий выбора элементов из списка
            jlst.addListSelectionListener(new ListSelectionListener() {
                  public void valueChanged(ListSelectionEvent le) {
                        //получить индекс измененного элемента
                        int idx = jlst.getSelectedIndex();
                        //отобразить сделанный выбор, если элемент
                        //был выбран из списка
                        if(idx != -1)
                              jlab.setText( "Current selection: " + Cities[idx]);
                        else
                              //в противном случае еще раз предложить
                              //выбрать город из списка
                              jlab.setText("Choose a City");
                  }
            });
            //ввести список и метку на панели содержимого
            jfrm.add(jscrlp);
            jfrm.add(jlab);
            //отобразить фрейм
            jfrm.setVisible(true);
      }

      public static void main(String[] args) {
            //создать фрейм
            new JListDemo();
      }
}
```

## Диспетчеры компоновки

Диспетчер компоновки автоматически размещает элементы управления в окне по определенному алгоритму. Элементы управления, создаваемые средствами Java, можно также размещать вручную, но делать это не стоит по следующим причинам. Во-первых, размещать вручную большое количество компонентов — довольно утомительное занятие. И, во-вторых, в тот момент, когда требуется разместить какой-нибудь элемент управления, его ширина и высота могут быть неизвестны, поскольку еще не готовы компоненты платформенно-ориентированных инструментальных средств.

У каждого объекта класса Container имеется свой диспетчер компоновки, который является экземпляром любого класса, реализующего интерфейс LayoutManager. Диспетчер компоновки устанавливается с помощью метода setLayout(). Если же метод setLayout() не вызывается, то используется диспетчер компоновки, выбираемый по умолчанию. Всякий раз, когда изменяются размеры контейнера, в том числе и в первый раз, диспетчер компоновки применяется для размещения каждого компонента в контейнере. Метод setLayout() имеет следующую общую форму:

```
void setLayout(LayoutManager объект_компоновки)
```

Здесь параметр объект_компоновки обозначает ссылку на требуемый диспетчер компоновки. Если требуется отменить действие диспетчера компоновки и разместить компоненты вручную, в качестве параметра объект_компоновки следует передать пустое значение null. В таком случае форму и расположение каждого компонента придется определить вручную с помощью метода setBounds() из класса Component.

Но, как правило, компоненты GUI размещаются с помощью диспетчера компоновки. Каждый диспетчер компоновки отслеживает список компонентов, хранящихся под своими именами. Диспетчер компоновки получает уведомление всякий раз, когда компонент вводится в контейнер. А когда требуется изменить размеры контейнера, диспетчер компоновки вызывает для этой цели свои методы minimumLayoutSize() и preferredLayoutSize(). Каждый компонент, которым манипулирует диспетчер компоновки, содержит методы getPreferredSize() и getMinimumSize(). Они возвращают предпочтительные и минимальные размеры для отображения каждого компонента. Диспетчер компоновки будет учитывать эти размеры, если это вообще возможно, не нарушая правила компоновки. Эти методы можно переопределить в создаваемых подклассах элементов управления, а иначе предоставляются значения по умолчанию.

В языке Java имеется ряд предопределенных классов диспетчеров компоновки, часть из которых описывается далее. Среди них можно выбрать такой диспетчер компоновки, который лучше всего подходит для конкретной прикладной программы.

## Класс FlowLayout

Класс FlowLayout реализует диспетчер поточной компоновки, выбираемый по умолчанию (см. класс JCheckBoxDemo выше). Диспетчер компоновки типа FlowLayout реализует простой стиль компоновки, который напоминает порядок следования слов в редакторе текста. Направление компоновки определяется свойством ориентации компонента в контейнере, которое по умолчанию задает направление слева направо и сверху вниз. Следовательно, по умолчанию компоненты размещаются построчно, начиная с левого верхнего угла. Но в любом случае компонент переносится на следующую строку, если он не умещается в текущей строке. Между компонентами остаются небольшие промежутки сверху, снизу, справа и слева. Ниже приведены конструкторы класса FlowLayout.

```
FlowLayout()
FlowLayout(int способ)
FlowLayout(int способ, int горизонтально, int вертикально)
```

В первой форме конструктора выполняется компоновка по умолчанию, т.е. компоненты размещаются по центру, а между ними остается промежуток пять пикселей.

Во второй форме конструктора можно определить способ расположения каждой строки. Ниже представлены допустимые значения параметра способ.

```
FlowLayout.LEFT
FlowLayout.CENTER
FlowLayout.RIGHT
FlowLayout.LEADING
FlowLayout.TRAILING
```

Эти значения определяют выравнивание по левому краю, по центру, по правому краю, переднему и заднему краю соответственно.

В третьей форме конструктора в качестве параметров горизонтально и вертикально можно определить промежутки между компонентами по горизонтали и по вертикали.

## Класс BorderLayout

Этот класс реализует общий стиль граничной компоновки для окон переднего плана. Он имеет четыре узких компонента фиксированной ширины по краям и одну крупную область в центре. Четыре стороны именуются по сторонам света: север, юг, запад и восток, а область посредине называется центром. Ниже приведены конструкторы, определяемые в классе BorderLayout.

```
BorderLayout()
BorderLayout(int горизонтально, int вертикально)
```

В первой форме конструктора выполняется граничная компоновка по умолчанию. А во второй форме в качестве параметров горизонтально и вертикально устанавливается горизонтальный и вертикальный промежутки между компонентами. В классе BorderLayout определяются следующие константы для указания областей граничной компоновки:

```
BorderLayout.CENTER
BorderLayout.SOUTH
BorderLayout.EAST
BorderLayout.WEST
BorderLayout.NORTH
```

Эти константы обычно используются при вводе компонентов в компоновку с помощью следующей формы метода add() из класса Container:

```
void add(Component ссылка_на_компонент, Object область)
```

Здесь параметр ссылка_на_компонент обозначает ссылку на вводимый компонент, а область — место для ввода компонента в компоновку.

## Класс GridLayout

При использовании класса GridLayout компоненты размещаются табличным способом в двухмерной сетке (см. класс JScrollPaneDemo выше). Реализуя класс GridLayout, следует определить количество строк и столбцов. Ниже приведены конструкторы, предоставляемые в классе GridLayout.

```
GridLayout()
GridLayout(int количество_строк, int количество_столбцов)
GridLayout(int количество_строк, int количество_столбцов, int горизонтально, int
вертикально)
```

В первой форме конструктора выполняется сеточная компоновка с одним столбцом, а во второй форме конструктора — сеточная компоновка с заданным количеством строк и столбцов. Третья форма позволяет определить в качестве параметров горизонтально и вертикально промежутки между компонентами по горизонтали и по вертикали. Любой из параметров количество_строк или количество_столбцов может принимать нулевое значение. Так, если нулевое значение принимает параметр количество_строк, то ограничение на ширину столбцов не налагается. А если нулевое значение принимает параметр количество_столбцов, то ограничение не налагается на длину строк.

## Класс CardLayout

Особое место среди классов диспетчеров компоновки принадлежит классу CardLayout, поскольку он позволяет хранить разные компоновки. Каждую компоновку можно представить в виде отдельной карты из колоды. Карты можно перетасовывать как угодно, чтобы в любой момент наверху колоды находилась какая-нибудь карта. Карточная компоновка может оказаться удобной для пользовательских интерфейсов с необязательными компонентами, которые можно динамически включать и отключать в зависимости от вводимых пользователем данных. Имеется возможность подготовить разные виды компоновки и скрыть их до того момента, когда они потребуются. В классе CardLayout предоставляются следующие конструкторы:

```
CardLayout()
CardLayout(int горизонтально, int вертикально)
```

В первой форме выполняется карточная компоновка по умолчанию. А во второй форме в качестве параметров горизонтально и вертикально можно указать промежутки между компонентами по горизонтали и по вертикали.

Карточная компоновка требует немного больших затрат труда, чем другие виды компоновки. Карты обычно хранятся в объекте типа Panel. Для этой панели следует выбрать диспетчер компоновки типа CardLayout. Карты, составляющие колоду, как правило, также являются объектами типа Panel. Следовательно, придется сначала создать панель для колоды карт, а также отдельную панель для каждой карты из этой колоды. Затем следует ввести на соответствующей панели компоненты, формирующие каждую карту. После этого панели отдельных карт нужно ввести на главной панели с диспетчером компоновки типа CardLayout. И, наконец, главную панель следует ввести в окно. Как только это будет сделано, нужно предоставить пользователю возможность выбирать каким-нибудь способом карты из колоды. Когда карта вводится на панели, ей обычно присваивается имя. Для этой цели чаще всего употребляется приведенная ниже форма метода add(), где имя обозначает конкретное имя карты, панель которой определяется параметром ссылка_на_панель.

```
void add(Component ссылка_на_панель, Object имя)
```

Как только колода карт будет сформирована, отдельные карты в ней активизируются с помощью одного из следующих методов, определяемых в классе CardLayout:

```
void first(Container панель)
void last(Container панель)
void next(Container панель)
void previous(Container панель)
void show(Container панель, String имя_карты)
```

Здесь параметр панель обозначает ссылку на контейнер (обычно панель), где хранятся карты, а имя_карты — конкретное имя карты. В результате вызова метода first() отображается первая карта в колоде. Для отображения последней карты в колоде следует вызвать метод last(), для отображения следующей карты — метод next(), а для отображения предыдущей карты — метод previous(). Методы next() и previous() автоматически перебирают колоду карт снизу вверх или сверху вниз соответственно. А метод show() отображает карту по заданному имени_карты.

## Класс GridBagLayout

Рассмотренные выше виды компоновки вполне пригодны для применения во многих прикладных программах, тем не менее в некоторых из них требуется более точное управление расположением компонентов в окне. Для этой

цели подходит сеточно-контейнерная компоновка, реализуемая в классе GridBagLayout. Удобство такой компоновки состоит в том, что она позволяет задавать относительное расположение компонентов, указывая его в ячейках сетки. Но самое главное, что каждый компонент может иметь свои размеры, а каждая строка — свое количество столбцов. Именно поэтому данная разновидность компоновки называется сеточно-контейнерной и представляет собой совокупность мелких соединенных вместе сеток. Местонахождение и размеры каждого компонента в сеточно-контейнерной компоновке определяются рядом связанных с ним ограничений, которые содержатся в объекте класса GridBagConstraints. В частности, ограничения налагаются на высоту и ширину ячейки, расположение компонента, его выравнивание и точку привязки в самой ячейке. Общая процедура сеточно-контейнерной компоновки выполняется следующим образом. Сначала создается новый объект типа GridBagLayout в качестве текущего диспетчера компоновки. Затем налагаются ограничения на каждый компонент, вводимый в сеточный контейнер. После этого компоненты вводятся в диспетчер компоновки.

В классе GridBagLayout определяется следующий единственный конструктор:

```
GridBagLayout()
```

Кроме того, в этом классе определяется ряд методов, многие из которых являются защищенными и не предназначены для общего употребления. Но среди них имеется один метод, который все же необходимо использовать. Это метод setConstraints(), общая форма которого приведена ниже.

```
void setConstraints(Component компонент, GridBagConstraints ограничения)
```

Здесь параметр компонент обозначает тот компонент, на который налагаются указанные ограничения. Этот метод описывает ограничения, налагаемые на каждый компонент в сеточном контейнере. Залогом успешного применения класса GridBagLayout является тщательная установка ограничений, которые хранятся в объекте класса GridBagConstraints.

В классе GridBagConstraints определяется несколько полей, которые можно устанавливать для управления размерами компонентов, их размещением и промежутками между ними.