Лабораторная работа №

Создание графических интерфейсов средствами Swing

Практически любое приложение содержит можно разделить на две части. Первая – это программный код, выполняющий некоторые действия (ради выполнения этих действий, собственно говоря, и разрабатывается приложение). Вторая часть – это интерфейс, который определяет порядок взаимодействия пользователя с программой. качественно написанный программный продукт обязан иметь интерфейс, удовлетворяющим требованиям и привычкам пользователя.

В том случае, если программа разрабатывается с использованием языка программирования Java, то для создания качественных графических интерфейсов используется Swing.

Swing представляет собой набор классов, применяемых для создания графических пользовательских интерфейсов. Swing представляет набор визуальных компонентов, таких как кнопки, поля редактирования, таблиц и т.д., достаточный для создания современных интерфейсов.

В ранних версия Java Swing отсутствовал, использовала оконная подсистема AWT (Abstract Window Toolkit). В AWT был определен базовый набор окон и управляющих элементов, позволяющий создавать графические интерфейсы. Одним из ограничений AWT была платформенно-ориентированная поддержка визуальных компонентов. В результате внешний вид интерфейсных элементов определялся не средствами Java, а используемой платформой (в Linux и Windows, например, одни и те-же компоненты выглядели различно). Элементы, созданные средствами AWT, назывались ***тяжеловесными***.

Набор классов Swing был создан для того, чтобы преодолеть ограничения, связанные с использованием AWT. Swing полностью написан на Java и не зависит от средств той платформы, на которой выполняется программа и называются ***легковесными***.

Несмотря на то, что Swing снимает ряд ограничений, присущих AWT, он не заменяет AWT. Более того, в основу Swing положены некоторые основные решения, принятые в AWT.

Архитектура MVC

Визуальный компонент сочетает в себе следующие характеристики:

- способ отображения на экране;

- реакция на действия пользователя;
- информация, связанная с данным компонентом.

Этими тремя характеристиками компонент обладает независимо от того, какая архитектура использовалась при его создании. Тем не менее, одна из архитектур стала чрезвычайно популярной. Это архитектура ***«модель – представление – контроллер»*** («Model – View – Controller» - MVC). Преимущество данной архитектуры состоит в том, что каждая ее составляющая в точности соответствует одной из характеристик компонента.

Модель задает состояние объекта. Например, для флажка модель содержит признак того, установлен или сброшен флажок.

Представление определяет, как компонент будет отображаться на экране, в том числе как будет представлено текущее состояние модели.

Контроллер обеспечивает реакцию компонента на действия пользователя. Например, когда пользователь щелкает на флажке опций, реакцией контроллера является изменение модели (установка сброшенного флажка или сброс установленного).

Разделяя компонент на модель, представление и контроллер можно добиться того, что реализация одной части не будет оказывать никакого влияния на две остальные. Например, разные представления могут отображать один и тот же компонент разными способами: при этом модель и контроллер остаются неизменными.

В Swing используется модифицированный вариант MVC, в котором представление и контроллер объединены в один элемент, называемый ***представлением пользовательского интерфейса***. Подход, используемый в Swing известен как архитектура ***«модель – представитель»*** или архитектура с ***выделенной моделью***.

Поддержка архитектуры «модель – представитель» обусловливает наличие в компонентах Swing двух объектов. Первый из них представляет модель, второй соответствует представителю пользовательского интерфейса.

Компоненты и контейнеры

В состав графического интерфейса, созданного средствами Swing, входят элементы двух типов: ***компоненты*** и ***контейнеры***.

Компоненты – это независимые элементы, например, кнопки или линейные регуляторы.

Контейнеры – могут содержать в себе несколько компонентов и представляют собой специальный тип компонентов. В Swing определены два типа контейнеров – контейнеры верхнего уровня (тяжеловесные, то есть

зависимые оп платформы) и легковесные контейнеры, реализованные Java. Легковесные контейнеры могут включаться в другие контейнеры, поэтому они часто используются для объединения в группы связанных друг с другом компонентов.

Чтобы компонент отобразился на экране, его надо поместить в контейнер. Следовательно, в составе графического пользовательского интерфейса должен присутствовать хотя бы один контейнер. Поскольку контейнеры являются компонентами (хотя и специфическими), один контейнер может находиться в составе другого. Это позволяет формировать иерархию контейнеров, на вершине которой должен находиться ***контейнер верхнего уровня***.

Панели контейнеров верхнего уровня

К контейнерам верхнего уровня относятся контейнеры JFrame, JApplet, JWindow и JDialog.

В очень часто используемом контейнере верхнего уровня JFrame реализован следующий набор панелей (рисунок 1).

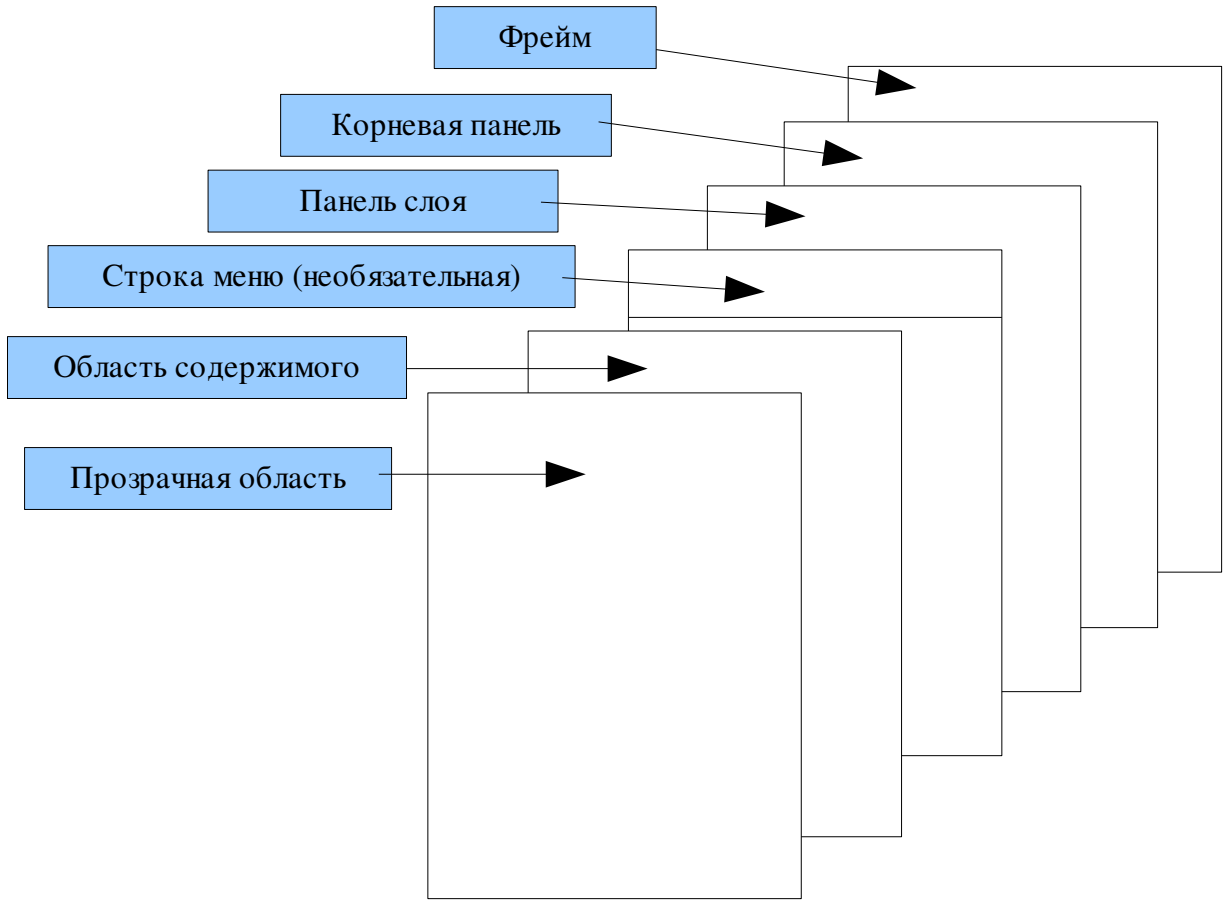

Рисунок 1. Панели контейнеров верхнего уровня

На вершине иерархии панелей находится корневая панель. Это легковесный контейнер, цель которого – управление другими панелями и, при необходимости, полосой меню. Корневая панель включает в себя прозрачную («стеклянную») панель (glass pane), панель содержимого (content pane) и панель слоя (layered pane).

«Стеклянная» панель – это панель верхнего уровня, которая расположена поверх остальных панелей. Она позволяет управлять событиями мыши, относящимися ко всему контейнеру, а не к содержащимся в нем компонентам. Она обеспечивает рисование компонентов. В большинстве случаев необходимость непосредственного взаимодействия со «стеклянной» панелью не возникает.

Панель слоя поддерживает «третье измерение» для компонентов, то есть определяет правила перекрытия одних компонентов другими. В составе панели слоя находится панель содержимого и может находиться строка меню.

Несмотря на то, что «стеклянная» панель и панель слоя являются неотъемлемыми частями контейнера верхнего уровня и выполняют важные функции, их действия большей частью скрыты от пользователей и разработчиков. Приложение в основном взаимодействует с панелью содержимого, поскольку именно в нее включаются визуальные компоненты. Чтобы добавить компонент (например, кнопку) к контейнеру верхнего уровня, его надо добавить в панель содержимого.

Создание фреймов

Окно верхнего уровня, то есть окно не содержащиеся внутри другого окна, в Java называется ***фреймом*** (frame). В библиотеке AWT для этого окна предусмотрен класс Frame. В библиотеке Swing аналогом этого класса является JFrame. Класс JFrame расширяет класс Frame.

На рисунке 2 приведен пример программы, предназначенной для создания простого фрейма, и фрейм, созданный этой программой. Ниже приводится текст программы, создавшей простой фрейм, не содержащий никаких компонентов.

```
     //Основные классы Swing содержатся в пакете javax.swing
import javax.swing.*;

public class FrameDemo {

     FrameDemo(){
          // Создание контейнера верхнего уровня JFrame
          JFrame jfrm = new JFrame("Простой фрейм");
```

```
            // Установка начальных размеров фрейма
            jfrm.setSize(275, 100);
            // Завершение программы при закрытии пользователем
            // окна приложения
            jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            // Отображение фрейма
            jfrm.setVisible(true);
      }
      public static void main(String[] args) {
            // Создание фрейма в потоке обработки событий
            SwingUtilities.invokeLater(new Runnable(){
                  public void run(){
                        new FrameDemo();
                  }
      });
      }
}
```

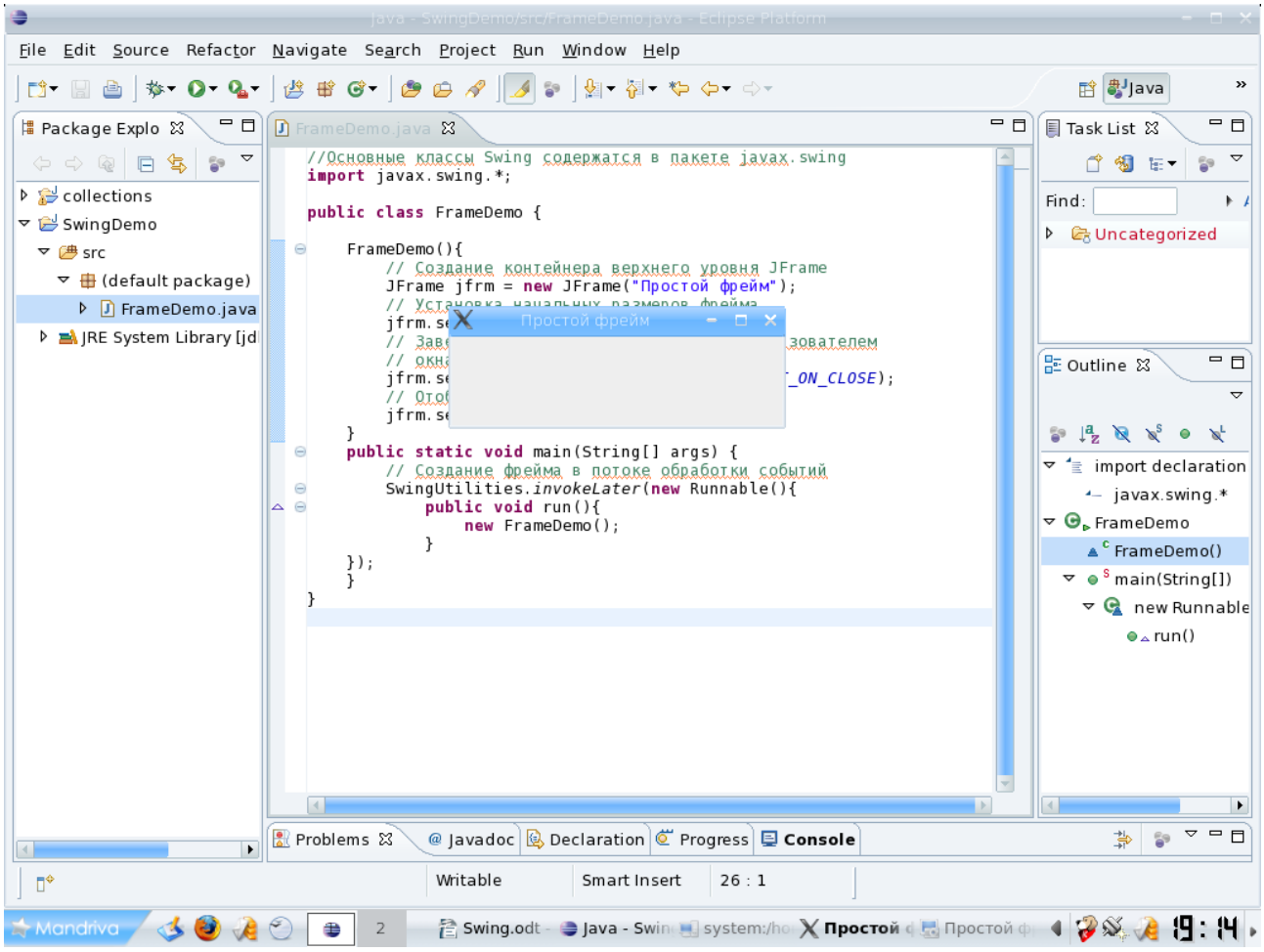

Рисунок 2. Простой фрейм на фоне программы, его создавшей

Класс JFrame

При создании приложений (не аплетов) на Java в качестве контейнера верхнего уровня наиболее часто используется класс JFrame. Иерархия наследования для класса JFrame (и класса Jpanel, который обсуждается далее)

приведена на рисунке 3.

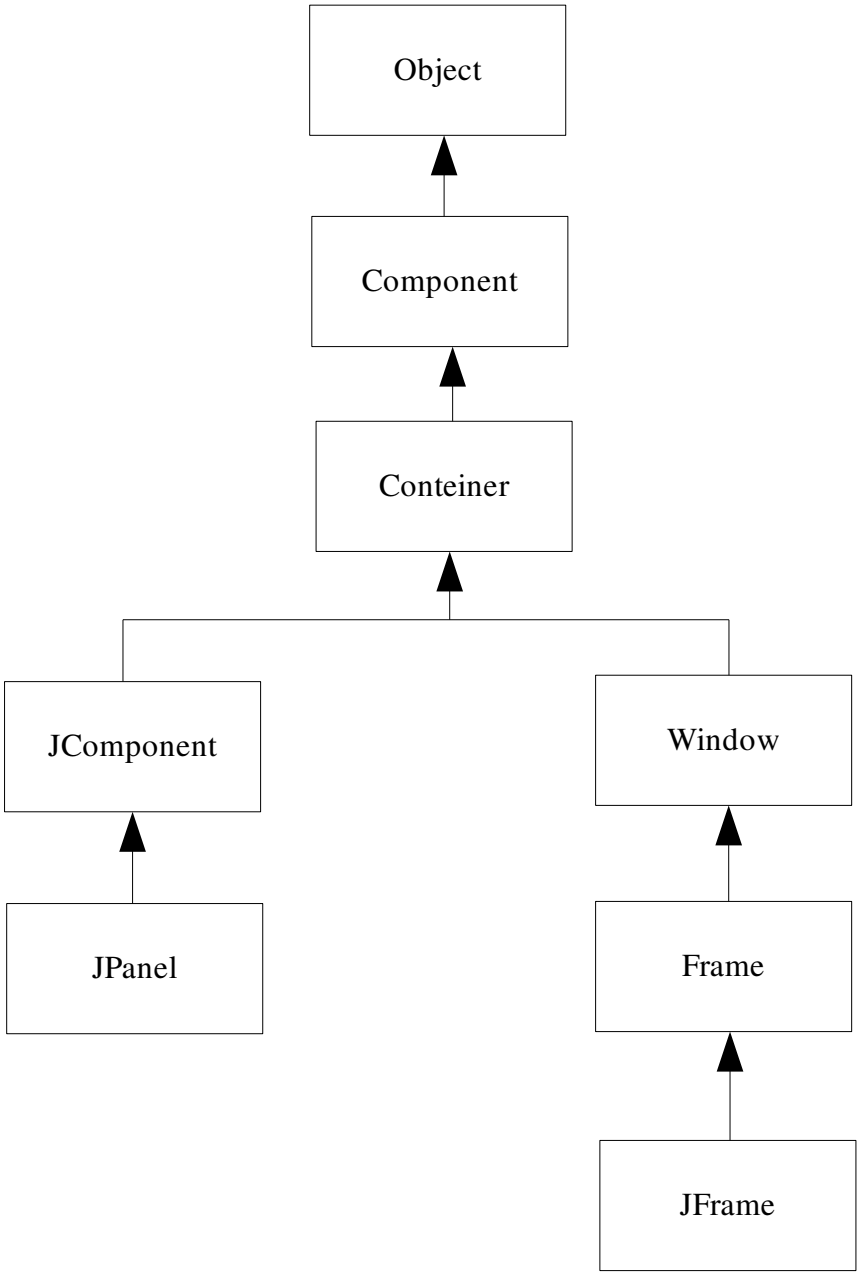

Рисунок 3. Иерархия наследования для классов JFrame и JPanel

В классе java.awt.Component определены следующие важные методы.

boolean ***isVisible()*** - проверяет, является ли данный компонент видимым. Компоненты являются видимыми изначально, за исключением таких элементов верхнего уровня, как JFrame.

void ***setVisible(boolean b)*** - отображает или скрывает компонент в зависимости от параметра b (true или false).

boolean ***isShowing()*** - проверяет, можно ли отобразить компонент на экране. Чтобы компонент мог быть отображенным, он должен быть видимым и находиться внутри контейнера, отображение которого также разрешено.

boolean ***isEnable()*** - проверяет, доступен ли компонент. Доступный компонент может получать информацию с клавиатуры. Компоненты являются доступными изначально.

void ***setEnable(boolean b)*** – разрешает или запрещает доступ к объекту.

Point ***getLocation()*** - восстанавливает местоположение левого верхнего угла компонента по отношению в левому верхнему углу содержащего его контейнера. Возвращаемый объект p класса Point инкапсулирует координаты x и y в полях p.x и p.y.

Point ***getLocationOnScreen()*** - возвращает местоположение левого верхнего угла компонента в экранной системе координат.

void ***setBound(int x, int y, int width, int height)*** – перемещает компонент и изменяет его размеры. Расположение верхнего левого угла задают параметры x и y, новый размер – параметры width и height.

void ***setLocation(int x, int y)***,

void ***setLocation(Point p)*** - оба метода перемещают компонент в новую точку. Если компонент не является компонентом верхнего уровня, координаты x и y отсчитываются относительно контейнера, в противном случае (например, для JFrame) используется экранная система координат.

Dimension ***setSize(Dimension d)*** – задает новые размеры компонента.

В классе java.awt.Window определены следующие методы.

void ***toFront()*** - выводит окно на экран поверх других окон.

void ***toBack()*** - помещает окно на экран позади всех других окон.

В классе java.awt.Frame определены следующие методы.

void ***setResizable(boolean b)*** – определяет, может ли пользователь изменять размеры фрейма.

void ***setTitle(String s)*** – отображает в заголовке фрейма строку s.

void ***setIconImage(Image image)*** – определяет изображение, которое будет использоваться в качестве пиктограммы для фрейма.

void ***setUndecorated(boolean b)*** – если параметр b имеет значение true, то строка заголовка, кнопки изменения размера и другие подобные элементы

фрейма не отображаются.

boolean ***isUndecorated()*** - возвращает true, если строка заголовка, кнопки изменения размера и другие подобные элементы не отображаются.

int ***getExtendedState()*** **-** определяет состояние окна**.** Возвращаемые значения:

- Frame.NORMAL
- Frame.ICONIFIED
- Frame.MAXIMIZED_HORIZ
- Frame.MAXIMIZED_VERT
- Frame.MAXIMIZED_BOTH

void ***setExtendedState(int state)*** - устанавливает окно в определенное состояние. Возможные значения переменной state совпадают со значениями, возвращаемые ***getExtendedState().***

При работе с контейнерами верхнего уровня существенный интерес представляют следующие класса java.awt.Toolkit.

static ***Toolkit.getDefaultToolkit()*** - возвращает объект класса Toolkit, предусмотренный по умолчанию.

Dimension ***getScreenSize()*** - определяет размеры экрана.

Image ***getImage(String filename)*** – загружает изображение из файла filename.

Покажем, как можно использовать Toolkit, например, для центрирования начального расположения фрейма. Поставим следующую задачу: разработать программу, которая выводит пустой фрейм, занимающий четверть экрана и отображающийся в его центре.

Ниже приводится программа для решения этой задачи.

```
import java.awt.Dimension;
import java.awt.Toolkit;
import javax.swing.JFrame;
import javax.swing.SwingUtilities;


public class CentrFrame {

          CentrFrame(){
               // Определение размеров окна, в котором
               // будет отображаться фрейм (дисплея)
```

```
                //
                // Получаем объект класса Toolkit
                Toolkit tk = Toolkit.getDefaultToolkit();
                // Определяем размерность дисплея
                Dimension screenSize = tk.getScreenSize();
                // Определяем высоту и ширину дисплея в пикселах
                int height = screenSize.height;
                int width = screenSize.width;
                // Создаём контейнер верхнего уровня JFrame
                JFrame jf = new JFrame("Фрейм в центре окна");
                // Устанавливаем размеры фрейма
                jf.setSize(width/2, height/2);
                // Устанавливаем положение фрейма на дисплее
                jf.setLocation(width/4, height/4);
                // Завершение программы при закрытии пользователем
                // окна приложения
                jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                // Отображаем фрейм
                jf.setVisible(true);
        }
        public static void main(String[] args) {
                // Создание фрейма в потоке обработки событий
                SwingUtilities.invokeLater(new Runnable(){
                        public void run(){
                                new CentrFrame();
                        }
        });
        }
}
```

Панели

В приложениях Swing редко встречаются компоненты (кнопки, метки и т.д.), непосредственно включаемые в панель верхнего уровня (хотя это возможно). Вместо этого они помещаются в одну или несколько панелей, которые включаются в панель содержимого. Чаще остальных используется контейнер JPanel, достаточно распространены JScrollPane, JTabbedPane, JSplitPane, которые носят общее название «панели». Панели позволяют лучше организовать графический пользовательский интерфейс, предоставляя возможность управления группами элементов.

Класс JPanel

JPanel – это контейнер общего назначения, предназначенный для размещения других компонентов. Поскольку JPanel является потомком класса JComponen (рисунок 3), он является компонентом. Таким образом, JPanel – это легковесный контейнер, способный включать в себя другие компоненты, в том

числе и JPanel.

Класс JPanel имеет следующие конструкторы.

JPanel() - создается панель по умолчанию, с которой связывается диспетчер компоновки FlowLayout (о диспетчерах компоновки речь пойдет ниже) и используется двойная буферизация – механизм позволяющий избегать мерцаний при перерисовки.

JPanel(LayoutManager lm) – создает панель и связывает с ней диспетчер компоновки ***lm***. Панель поддерживает двойную буферизацию.

JPanel(boolean doubleBuf) - создается панель, с которой связывается диспетчер компоновки FlowLayout. Если значение ***doubleBuf*** равно true, панель поддерживает двойную буферизацию, в противном случае двойная буферизация отключается.

JPanel(LayoutManager lm, boolean doubleBuf) - создает панель и связывает с ней диспетчер компоновки ***lm.*** Если значение ***doubleBuf*** равно true, панель поддерживает двойную буферизацию, в противном случае двойная буферизация отключается.

Для того, чтобы добавить некоторый компонент (например, кнопку или метку) к панели JPanel достаточно воспользоваться методом ***add()*** и передать в него ссылку на включаемый в панель компонент.

add(Component comp) – добавляет компонент в панель. Добавляемый элемент добавляется в конец списка компонентов панели.

add(Component comp, int index) – добавляет компонент в панель. В списке компонентов добавляемый компонент располагается в позиции ***index***. Если ***index=-1***, то компонент добавляется в конец списка.

Класс JScrollPane

Панель JScrollPane автоматически осуществляет прокрутку для отображения содержащегося в ней компонента. Поскольку легковесные контейнеры являются компонентами, JScrollPane может прокручивать, например, содержимое JPanel.

Структура контейнера JScrollPane приведена на рисунке 4.

Наибольшие размеры имеет часть JScrollPane, называемая областью просмотра (viewport). Это окно, в котором выводится компонент, предназначенный для прокрутки. По умолчанию JScrollPane по мере необходимости динамически отображает или удаляет полосу прокрутки. Например, если высота компонента больше чем область просмотра, отображается вертикальная полоса прокрутки. Если компонент полностью

помещается в области просмотра, то полосы прокрутки удаляются. Можно добавить заголовок строки или столбца, описывающие отображаемые данные. В четырех углах можно разместить компоненты, которые будут реализовывать дополнительные функции.

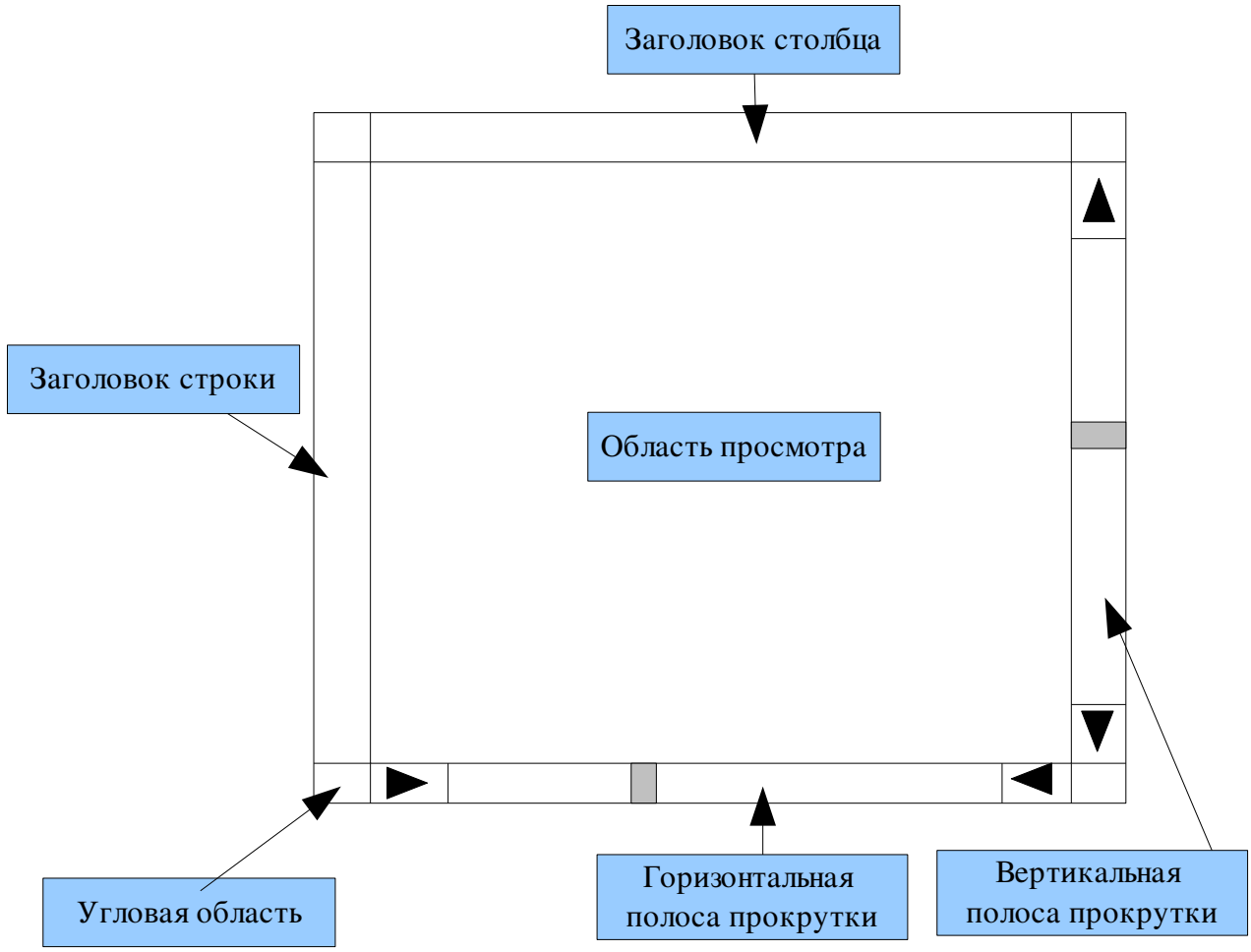

Рисунок 4. Структура контейнера JScrollPane

В класс JScrollPane определены следующие конструкторы.

JScrollPane() - создает объект JScrollPane, для которого область просмотра не определена.

JscrollPane(Component com) - создает объект JScrollPane, который автоматически прокручивает компонент, задаваемый посредством параметра ***com***.

JscrollPane(int vertSBP, int horizSBP) - создает объект JScrollPane, для которого область просмотра не определена. Политика использования полос прокрутки задается с помощью параметров ***vertSBP*** (для вертикальной полосы прокрутки) и ***horizSBP*** (для горизонтальной полосы прокрутки).

JScrollPane(Component com, int vertSBP, int horizSBP) - создает объект

JScrollPane, который автоматически прокручивает компонент, задаваемый посредством параметра ***com***. Политика использования полос прокрутки задается с помощью параметров ***vertSBP*** и ***horizSBP***.

Политики полос прокрутки определяются с помощью следующих констант, описанных в интерфейсе ScrollPaneConstants, которые описаны в таблице 1.

Таблица 1.Константы для описания политик полос прокрутки

Константа	Политика
HORIZONTAL_SCROLLBAR_AS_NEEDED	Горизонтальная полоса прокрутки отображается только при необходимости. Эта политика принимается по умолчанию.
HORIZONTAL_SCROLLBAR_NEVER	Горизонтальная полоса прокрутки никогда не отображается.
HORIZONTAL_SCROLLBAR_ALWAYS	Горизонтальная полоса прокрутки отображается всегда.
VERTICAL_SCROLLBAR_AS_NEEDED	Вертикальная полоса прокрутки отображается только при необходимости. Эта политика принимается по умолчанию.
VERTICAL_SCROLLBAR_NEVER	Вертикальная полоса прокрутки никогда не отображается.
VERTICAL_SCROLLBAR_ALWAYS	Вертикальная полоса прокрутки отображается всегда.

Класс JTabbedPane

Контейнер JTabbedPane управляет компонентами, размещая их на вкладках. Выбирая определенную вкладку, пользователь переводит связанные с ней компоненты на передний план.

В классе JTabbedPane определены следующие конструкторы.

JTabbedPane() - создает панель с вкладками, в которой ярлыки вкладок располагаются вдоль верхней границы окна.

JTabbedPane(int where) - создает панель с вкладками, в которой размещение ярлыков определяется параметром where. Этот параметр может принимать следующие значения: ***JtabbedPane.TOP*** (ярлыки вдоль верхней границы)***,*** ***JtabbedPane.BOTTOM*** (ярлыки вдоль нижней границы)***,***

JtabbedPane.LEFT (ярлыки вдоль левой границы)***, JtabbedPane.RIGHT*** (ярлыки вдоль правой границы)**.**

JTabbedPane(int where, int sclrorWrap) - создает панель с вкладками, в которой размещение ярлыков определяется параметром where. Этот параметр может принимать следующие значения: ***JTabbedPane.TOP*** (ярлыки вдоль верхней границы)***, JTabbedPane.BOTTOM*** (ярлыки вдоль нижней границы)***, JTabbedPane.LEFT*** (ярлыки вдоль левой границы)***, JTabbedPane.RIGHT*** (ярлыки вдоль правой границы)**.** Если число ярлыков становится большим и они не помещаются в одну строку, их организация задается параметром ***sclrorWrap.*** Этот параметр может принимать значения ***JtabbedPane.WRAP_TAB_LAYOUT или JtabbedPane.SCROLL_TAB_LAYOUT.***

Последний конструктор позволяет определить положение вкладок и политику, управляющую размещением ярлыков в том случае, когда они не помещаются в одну строку. Возможны два варианта. Ярлыки могут располагаться в несколько строк (такая политика принимается по умолчанию). В этом случае в качестве последнего параметра передается ***JtabbedPane.WRAP_TAB_LAYOUT.*** Согласно другой политике, задаваемой с помощью ***JtabbedPane.SCROLL_TAB_LAYOUT,*** все ярлыки размещаются в одну строку и пользователю предоставляются средства для их прокрутки.

Использование панели с вкладками не составляет труда. В первую очередь надо создать экземпляр ***JTabbedPane***, а затем заполнить панель компонентами. При добавлении каждого компонента указывается имя вкладки. Если необходимо разместить на вкладке группу компонентов, желательно сначала организовать ее с помощью JPanel.

Для добавления вкладок можно использовать метод addTab(). Существуют три варианта данного метода. Самый простой из них следующий:

void ***addTab(String name, Component comp)***.

В результате выполнения этого метода создается вкладка с именем, определенным посредством параметра ***name***, и содержащая компонент, указанный с помощью параметра ***comp***.

Менеджеры компоновки

Расположение компонентов в составе контейнера определяется с помощью ***менеджера компоновки***. В Java определено несколько таких менеджеров. Большинство из них входят в пакет AWT, но Swing также предоставляет несколько дополнительных менеджеров компоновки. В таблице 2 представлены несколько популярных менеджеров компоновки.

Таблица 2.Менеджеры компоновки

Наименование менеджера	**Действие менеджера компоновки**
FlowLayout	Располагает элементы в строке слева направо; следующая строка размещается под предыдущей
BorderLayout	Помещает компоненты в пяти областях, расположенных по центру и по краям контейнера. По умолчанию такой диспетчер компоновки связан с панелью содержимого
GridLayout	Располагает компоненты в виде таблицы
GridBagLayout	Располагает компоненты в виде таблицы с ячейками различных размеров
BoxLayout	Располагает компоненты по вертикали и горизонтали
SpringLayout	Использует при размещении компонентов специальные ограничения

FlowLayout – это менеджер ***поточной компоновки***. Компоненты размещаются от левого верхнего угла окна, слева направо, сверху вниз. Выше, ниже, слева и справа, а также между каждыми компонентами оставляется небольшое пустое пространство. Этот менеджер компоновки используется по умолчанию.

Конструкторы FlowLayout:

FlowLayout() - создает размещение по умолчанию, которое выравнивает компоненты по центру и сотавляет 5 пикселов пробела между компонентами;

FlowLayout(int how) – позволяет определить, как выравнивается каждая строка. Доаустимы следующие значения параметра ***how***: FlowLayout.LEFT (выравнивание влево), FlowLayout.CENTER (выравнивание по центру), FlowLayout.RIGHT (выравнивание вправо).

FlowLayout(int how, int horz,int vert) – позволяет задать в параметрах ***horz*** и ***vert*** задавать горизонтальный и вертикальный пробел, оставляемый между компонентами (величина пробела указывается в пикселах).

BorderLayout реализует граничный стиль компоновки. Он имеет четыре узких компонента фиксированной ширины по краям окна и один в иде большой области в центре. Четыре краевых компонента называются Север (North), Юг (South), Восток (East) и Запад (West). Средняя область называется Центр (Center).

В BorderLayout определены следующие конструкторы:

BorderLayout() - определяет граничное размещение по умолчанию;

BorderLayout(int horz, int vert) – позволяет указывать в параметрах ***horz*** и ***vert*** горизонтальные и вертикальные пробелы между компонентами.

BorderLayout определяет следующие константы, которые используются при определении области размещения: BorderLayout.CENTER, BorderLayout.NORTH, BorderLayout.SOUTH, BorderLayout.EAST, BorderLayout.WEST.

При добавлении компонента эти константы используются со следующей формой метода add():

void add(Component comp, Object region).

Здесь ***comp*** – компонент, который будет добавлен, а ***region*** определяет область размещения компонента.