Оглавление

[1 Библиотека swing 2](#_Toc530954667)

[2 Иерархическая схема класса Java Swing 2](#_Toc530954668)

[3 Окно JFrame 2](#_Toc530954669)

[4 Класс Container (контейнер) 4](#_Toc530954670)

[5 Класс JPanel (панель) 5](#_Toc530954671)

[6 Менеджеры расположения Layout 5](#_Toc530954672)

[7 Элементы управления Swing 6](#_Toc530954673)

[8 Структура приложения с графическим пользовательским интерфейсом 6](#_Toc530954674)

[9 Основные визуальные компоненты Swing 7](#_Toc530954675)

[9.1 Класс JComponent 7](#_Toc530954676)

[9.2 Метка JLabel 8](#_Toc530954677)

[10 Обычная кнопка JButton 9](#_Toc530954678)

[11 Радиокнопки 10](#_Toc530954679)

[12 Кнопки JCheckBoxMenuItem и JRadioButtonMenuItem 11](#_Toc530954680)

[12.1 Текстовое поле JTextField 13](#_Toc530954681)

[12.2 Поле для ввода пароля JPasswordField 13](#_Toc530954682)

[12.3 Область для ввода текста JTextArea 14](#_Toc530954683)

[12.4 Панель прокрутки JScrollPane 14](#_Toc530954684)

[12.5 Инструментальная панель JToolBar 15](#_Toc530954685)

[12.6 Выпадающий список JComboBox 16](#_Toc530954686)

[12.7 Ползунок JSlider 17](#_Toc530954687)

[12.8 Список JList 18](#_Toc530954688)

[12.9 JTable 18](#_Toc530954689)

[12.10 Модель столбцов таблицы TableColumnModel 20](#_Toc530954690)

[12.11 Модель выделения SelectionModel 22](#_Toc530954691)

[12.12 Заголовок таблицы JTableHeader 23](#_Toc530954692)

[13 Диалоговые окна FileDialog ... 24](#_Toc530954693)

[13.1 Диалоговое окно FileDialog 24](#_Toc530954694)

[14 Окно сообщений MessageBox 27](#_Toc530954695)

[**Исключения диалоговых окон** 28](#_Toc530954696)

[**Диалоговое окно JFileChooser** 28](#_Toc530954697)

[**Конструкторы JFileChooser** 29](#_Toc530954698)

[**Основные метод JFileChooser** 30](#_Toc530954699)

[**Режимы работы JFileChooser** 31](#_Toc530954700)

[**Возвращаемые компонентом JFileChooser значения** 31](#_Toc530954701)

[**Пример использования JFileChooser** 31](#_Toc530954702)

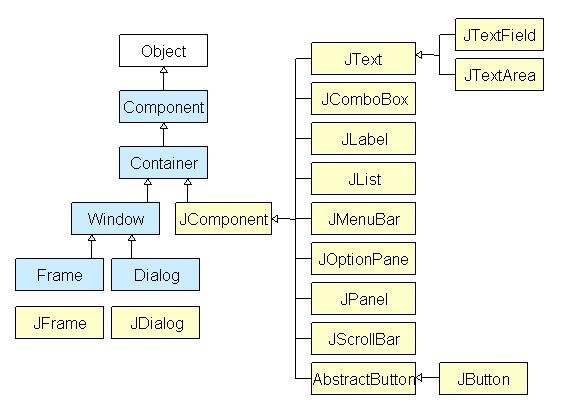
# Библиотека swing

Для работы с большинством классов библиотеки Swing используется пакет java.swing.\*

Java Swing – это легкий инструментарий графического интерфейса пользователя (GUI), который включает в себя богатый набор виджетов. Он включает в себя пакет, позволяющий создавать компоненты графического интерфейса для ваших приложений Java, и он не зависит от платформы.

Библиотека Swing построена поверх набора инструментов Java Abstract Widget Toolkit (AWT), более раннего инструментария GUI, зависящего от платформы. Вы можете использовать компоненты Java GUI, такие как кнопка, текстовое поле и т. д. из библиотеки, и не нужно создавать компоненты с нуля.

# Иерархическая схема класса Java Swing



# Окно JFrame

Каждая GUI-программа запускается в окне и по ходу работы может открывать несколько дополнительных окон.

Класс JFrame библиотекеи Swing определяет окно с **рамкой** и **строкой заголовка** (с кнопками «Свернуть», «Во весь экран» и «Закрыть»). Оно может изменять размеры и перемещаться по экрану.

Методы класса **JFrame**

|  |  |
| --- | --- |
| Конструкторы | |
| JFrame() | создает пустое окно. |
| JFrame(String title) | создает пустое окно с заголовком title. |
| Другие методы | |
| setSize(int width, int height) | устанавливает размеры окна. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка. |
| setDefaultCloseOperation(int operation)  operation: EXIT\_ON\_CLOSE – закрыть приложение при закрытии окна | позволяет указать действие, которое необходимо выполнить, когда пользователь закрывает окно нажатием на крестик.  Пример применения параметра EXIT\_ON\_CLOSE  forma1.setDefaultCloseOperation(EXIT\_ON\_CLOSE) – закрывает приложение |
| setVisible(boolean visible) | отобразить окно на экране, вызывается данный метод с параметром true. Если вызвать его с параметром false, окно снова станет невидимым. |

Примечание 1. Окно должно быть подготовлено к отображению, поэтому:

* оформить его, для этого надо выполнить настройку его свойств, например, таких как: фон, шрифты, привязку к положению на экране;
* создать множество элементов управления, настроить их внешний вид, разместить в нужных местах окна.

Примечание 2. Так как в приложении может быть много окон, то выполнять их настройку в методе main не очень хорошо. Лучше создавать для каждого окна свой класс и в нем производить настройку и определять методы. Это Что соответствует свойству инкапсуляции.

Пример . Программа и класс формы

//класс программы инициализирует объект

package GlobalProgram;

import javax.swing.\*;

public class GlobalProgram {

public static void main (String [] args) {

JFrame myForm = new ();

myForm.setVisible(true);

}}

//класс формы, программирование вручную

import javax.swing.\*;

import java.awt.\*;

import java.awt.Color;

import java.awt.GridLayout;

import javax.swing.border.\*;

public class Forma1 extends JFrame {

public Forma1()

{

super("Окно первой формы");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

JPanel p = new JPanel();

add(p); //установили на форме панель

setSize(400, 400);//пиксели

}}

# Класс Container (контейнер)

Классы контейнеров – это классы, на которых могут быть другие компоненты. Поэтому для создания графического интерфейса нам нужен хотя бы один объект-контейнер. Существует 3 типа контейнеров.

1. *Панель*: это чистый контейнер и не является окном. Единственная цель группы – организовать компоненты для окна.
2. *Рамка:* это полностью функционирующее окно со своим названием и значками.
3. *Диалог:* его можно рассматривать как всплывающее окно, которое появляется, когда сообщение должно отображаться. Это не полностью функционирующее окно, это рамка.

Все перечисленные контейнеры являются потомками класса Container и наследуют от него ряд полезных методов:

|  |  |
| --- | --- |
| *void add(Component component)* | добавляет в контейнер элемент component; |
| *void remove(Component component)* | удаляет из контейнера элемент component; |
| *void removeAll()* | удаляет все элементы контейнера; |
| *int getComponentCount()* | возвращает число элементов контейнера |

И другие методы. Этот класс имеет методы схожие с методами контейнерных классов Java. В частности, он имеет метод getComponentAt(int x, int y), возвращающий компонент, в который попадает точка с заданными координатами (координаты отсчитываются от левого верхнего угла компонента) и ряд других

# Класс JPanel (панель)

Панель JPanel — это элемент управления - контейнер, представляющий собой прямоугольное пространство, на котором можно размещать другие элементы. Содержимое панели занимает все пространство окна. Обратиться к этой панели можно методом getContentPanel() класса JFrame.

|  |  |
| --- | --- |
| getContentPanel() | Обратиться к содержимому панели формы |
| add(Component) | Добавление элементов управления на панели и панели в форме |

# Менеджеры расположения Layout

Определяют, каким образом на форме будут располагаться компоненты. Независимо от платформы, виртуальной машины, разрешения и размеров экрана менеджер расположения гарантирует, что компоненты будут иметь предпочтительный или близкий к нему размер и располагаться в том порядке, который был указан программистом при создании программы.

* Полярное расположение [BorderLayout](http://java-online.ru/swing-layout.xhtml" \l "borderlayout)
* Последовательное расположение [FlowLayout](http://java-online.ru/swing-layout.xhtml" \l "flowlayout)
* Табличное расположение [GridLayout](http://java-online.ru/swing-layout.xhtml" \l "gridlayout)
* Менеджер расположения [GridBagLayout](http://java-online.ru/swing-layout.xhtml" \l "gridbaglayout)
* Менеджер расположения [CardLayout](http://java-online.ru/swing-layout.xhtml" \l "cardlayout)
* Менеджер расположения [BoxLayout](http://java-online.ru/swing-layout.xhtml" \l "boxlayout)
* Менеджер расположения [GroupLayout](http://java-online.ru/swing-layout.xhtml" \l "grouplayout)
* Менеджер расположения [SpringLayout](http://java-online.ru/swing-layout.xhtml" \l "springlayout)
* Пример [диалогового окна авторизации](http://java-online.ru/swing-layout.xhtml#logindialog)
* Пример собственного [менеджера расположения](http://java-online.ru/swing-layout.xhtml#verticallayout)

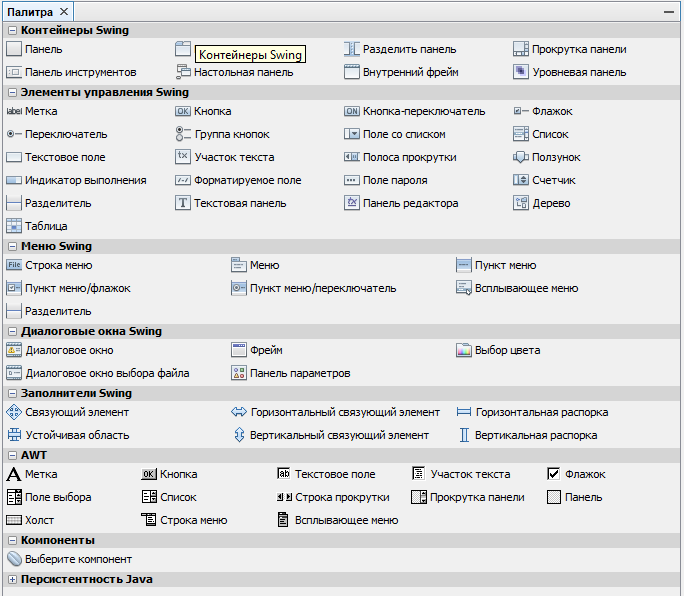
В Swing менеджер расположения играет еще большую роль, чем обычно. Он позволяет не только сгладить различия между операционными системами, но к тому же дает возможность с легкостью менять внешний вид формы, не заботясь о том, как при этом изменяются размеры и расположение компонентов.

Поддержка менеджеров расположения встроена в базовый класс контейнеров java.awt.Container. Все компоненты библиотеки Swing унаследованы от базового класса JComponent, который, в свою очередь, унаследован от класса Container. Таким образом, для любого компонента Swing можно установить менеджер расположения или узнать, какой менеджер им используется в данный момент. Для этого предназначены методы setLayout() и getLayout().

Изменять расположение желательно только в контейнерах, которые предназначены для размещения в них компонентов пользовательского интерфейса, то есть в панелях

setBounds(0, 0, 800, 600) (JPanel) и окнах (унаследованных от класса Window). В стандартной библиотеке Java существует несколько готовых менеджеров

# Элементы управления Swing



# Структура приложения с графическим пользовательским интерфейсом

1. Структура приложения

Класс программы

package GlobalProgram;

import javax.swing.\*;

public class GlobalProgram {

public static void main (String [] args) {

JFrame myForm = new Forma1();

myForm.setVisible(true);

}

}

1. Класс формы. Причем для каждой формы отдельный класс

package GlobalProgram;

import javax.swing.\*;

import java.awt.\*;

public class Forma1 extends JFrame {

Forma1()

{

super("Окно первой формы");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(400, 400);//пиксели

pack();//когда элементы на форме размещены, то автоматически //устанавливается размер окна

}}

# Основные визуальные компоненты Swing

Каждый визуальный компонент имеет три типа размера: *минимально допустимый, максимально допустимый и предпочтительный.* Узнать, чему равны эти размеры для данного компонента можно с помощью соответствующих методов:

Dimension getMinimumSize(),

Dimension getPreferredSize(),

Dimension getMaximumSize().

Методы возвращают результат типа Dimension. Они запрограммированы в соответствующем классе. Например, у кнопки минимальный размер — нулевой, максимальный размер не ограничен, а предпочтительный зависит от надписи на кнопке (вычисляется как размер текста надписи плюс размеры полей).

## Класс JComponent

Все визуальные компоненты библиотеки Swing унаследованы от класса *JComponent*. Сам этот класс является абстрактными и непосредственно не используется, но все визуальные компоненты наследуют его методы.

Основные методы

|  |  |
| --- | --- |
| void setEnabled(boolean enabled) | Управление доступностью компонент; параметр true – элемент доступен; false – недоступен |
| boolean isEnabled() | Возвращает **true**, если компонент активен и **false** в противном случае. |
| Void setVisible(boolean visible) | Отобразить компонент управления (например открыть форму). Многие компоненты этого не требуют, так как они установлены на форме, например, кнопка, поле теста, метки и т.д. |
| boolean isVisible() | Возвращает **false**, если компонент невидим и true в противном случае. |
| void setBackground(Color color) | изменить цвет заднего фона компонента (если компонент не прозрачен) |
| Color getBackground()и boolean isOpaque() | возвращают текущий цвет заднего фона и непрозрачность компонента. |
| setOpaque(boolean opaque) | устанавливается прозрачность компонента с параметром **true**. |

## Метка JLabel

Это обычный текст, который выводится в заданном месте окна и используется для вывода вспомогательной текстовой информации: подписи к другим элементам, инструкции и предупреждения для пользователя. В Swing метка позволяет достичь более интересных эффектов. Во-первых, помимо текста можно использовать значок. Во-вторых, с ее помощью можно выводить отформатированный текст.

|  |  |
| --- | --- |
| Конструкторы |  |
| JLabel(String text) | создает метку с надписью text |
| JLabel(Icon image) | создает метку со значком image |
| JLabel(String text, Icon image, int align) | создает метку с надписью text и значком image. Третий параметр задает выравнивание текста вместе со значком. В качестве него может быть использована одна из констант, описанных в интерфейсе SwingConstants: LEFT, RIGHT, CENTER. |
| В библиотеке Swing метка (и не только она) может быть настроена для отображения отформатированного текста в формате HTML. Для этого необходимо, чтобы строка, устанавливаемая в качестве надписи метки, начиналась с тега <html>. После этого можно использовать в строке любые теги языка HTML и они будут преобразовываться в соответствующие атрибуты форматирования. | |
| Основные методы |  |
| String getText() | возвращает текущий текст надписи метки |
| void setText(String text) | задает новый текст надписи |
| void setIcon(Icon image) | устанавливает новый значок. Значок - это объект класса ImageIcon |
| int getVerticalAlignment()  int getHorizontalAlignment(), | Возвращают правило выравнивания метки |
| void setVerticalAlignment(int align),  void setHorizontalAlignment(int align) | Установить новое выравнивание (по горизонтали и вертикали) |
| Для выравнивания по горизонтали служат константы LEFT\_ALIGNMENT (по левому краю), RIGHT\_ALIGNMENT (по правому краю) и CENTER\_ALIGNMENT (по центру). Для выравнивания по вертикали — BOTTOM\_ALIGNMENT (по нижнему краю), TOP\_ALIGNMENT (по верхнему краю) и CENTER\_ALIGNMENT (по центру). | |
|  |  |

# Обычная кнопка JButton

Класс JButton библиотеки Swing для создания обычных кнопок предлагает несколько различных конструкторов:

* *JButton()*
* *JButton(String)*
* *JButton(Icon)*
* *JButton(String, Icon)*

# Радиокнопки

Радиокнопки - их еще иногда называют кнопками с зависимой фиксацией - применяются исключительно в группах. В один момент времени включенной может быть только одна кнопка: как только пользователь включает другую кнопку, та кнопка, которая была включена до этого, выключается. Библиотека Swing для создания радиокнопок содержит конструкторы класса JRadioButton:

JRadioButton()

JRadioButton(Icon)

JRadioButton(Icon, boolean)

JRadioButton(String)

JRadioButton(String, boolean)

JRadioButton(String, Icon)

JRadioButton(String, Icon, boolean)

Параметры типа String задают надпись на кнопке, а параметры типа Icon определяют рисунок, который будет отображен вместо стандартного, рисуемого по умолчанию. Начальное состояние (включено/выключено) задается параметром конструктора, который имеет тип boolean. При значении true кнопка включается, а при false - выключается.

Методы класса JRadioButton практически совпадают с методами кнопок класса JCheckBox. Однако использовать радиокнопки несколько сложнее, поскольку они должны быть объединены в группы. А перед этим группа должна быть создана. Библиотека Swing содержит специальный класс ButtonGroup, от которого нам требуется всего три метода:

* конструктор ButtonGroup();
* add(AbstractButton) - метод добавления кнопки в группу;
* remove(AbstractButton) - метод удаления кнопки из группы.

Создав экземпляр класса ButtonGroup, надо последовательно вызывать метод add для каждой из радиокнопок, сгруппировать их.

# Кнопки JCheckBoxMenuItem и JRadioButtonMenuItem

Библиотека Swing предлагает воспользоваться новыми возможностями. Так, теперь вы сможете добавить к пунктам меню отмечаемые и радиокнопки. Это очень удобно, особенно когда вы создаете пункты меню для настройки опций приложения.

Меню с отмечаемыми кнопками можно сделать, воспользовавшись классом JCheckBoxMenuItem, для которого определены следующие конструкторы:

JCheckBoxMenuItem()

JCheckBoxMenuItem(Icon)

JCheckBoxMenuItem(String)

JCheckBoxMenuItem(String, Icon)

JCheckBoxMenuItem(String, boolean)

JCheckBoxMenuItem(String, Icon, boolean)

Как и в случае с другими кнопками, описанными ранее, аргументы типа String задают надпись для пункта меню, Icon - отображаемый в меню рисунок, boolean - включение/выключение кнопки меню.

Меню с радиокнопками создается конструкторами на базе класса JCheckBoxMenuItem:

JRadioButtonMenuItem()

JRadioButtonMenuItem(Icon)

JRadioButtonMenuItem(String)

JRadioButtonMenuItem(String, Icon)

|  |  |
| --- | --- |
| **Таблица 1. Основные методы кнопочных классов** | |
| Метод | Назначение |
| void setText(String) | Установить надпись на кнопке |
| String getText() | Считать надпись на кнопке |
| void setIcon(Icon) | Установить пиктограмму, отображаемую в нормальном состоянии |
| Icon getIcon() | Считать пиктограмму, отображаемую в нормальном состоянии |
| void setDisabledIcon(Icon) | Установить пиктограмму, отображаемую, когда кнопка отключена |
| Icon getDisabledIcon() | Считать пиктограмму, отображаемую, когда кнопка отключена |
| void setPressedIcon(Icon) | Установить пиктограмму для нажатой кнопки |
| Icon getPressedIcon() | Считать пиктограмму для нажатой кнопки |
| void setRolloverIcon(Icon) | Установить пиктограмму, отображаемую в тот момент, когда указатель мыши находится над кнопкой |
| Icon getRolloverIcon() | Считать пиктограмму, отображаемую в тот момент, когда указатель мыши находится над кнопкой |
| void setHorizontalAlignment(int) | Установить горизонтальное выравнивание кнопки (константами LEFT, CENTER и RIGHT) |
| void setVerticalAlignment(int) | Установить вертикальное выравнивание кнопки (константами TOP, CENTER и BOTTOM) |
| int getHorizontalAlignment() | Считать горизонтальное выравнивание кнопки |
| int getVerticalAlignment() | Считать вертикальное выравнивание кнопки |
| void setHorizontalTextPosition(int) | Установить горизонтальное выравнивание текста относительно отображаемой на кнопке пиктограммы (константы LEFT, CENTER и RIGHT) |
| void setVerticalTextPosition(int) | Установить вертикальное выравнивание текста относительно отображаемой на кнопке пиктограммы (константы TOP, CENTER и BOTTOM) |
| int getHorizontalTextPosition() | Считать горизонтальное выравнивание текста относительно отображаемой на кнопке пиктограммы |
| int getVerticalTextPosition() | Считать вертикальное выравнивание текста относительно отображаемой на кнопке пиктограммы |
| void setMargin(Insets) | Установить зазор между краями кнопки и ее содержимым (в пикселах) |
| Insets getMargin() | Считать зазор между краями кнопки и ее содержимым (в пикселах) |
| void setFocusPainted(boolean) | Установить состояние флага, определяющего необходимость отображения графической рамки, если кнопка получила фокус ввода (значение true включает отображение) |
| boolean isFocusPainted() | Считать состояние флага, определяющего необходимость отображения графической рамки в том случае, если кнопка получила фокус ввода |
| void setBorderPainted(boolean) | Установить состояние флага, определяющего необходимость отрисовки краев кнопки (значение true включает отрисовку) |
| boolean isBorderPainted() | Считать состояние флага, определяющего необходимость отрисовки краев кнопки |
| void setActionCommand(String) | Установить имя команды, выполняемой при нажатии на кнопку |
| String getActionCommand(void) | Считать имя команды, выполняемой при нажатии на кнопку |
| void setMnemonic(char) | Установить клавишную комбинацию для нажатия кнопки |
| char getMnemonic() | Считать клавишную комбинацию для нажатия кнопки |
| void doClick() | Программно эмулировать нажатие кнопки |

## Текстовое поле JTextField

Для создания текстового поля чаще всего используются конструкторы:

* **JTextField(int columns)** — создает пустое текстовое поле, ширина которого достаточна для размещения columns символов. При этом пользователь может вводить в текстовое поле строку какой угодно длины: она просто будет прокручиваться.
* **JTextField(String text)** — создает текстовое поле с начальным текстом text.
* **JTextField(String text, int columns)** — устанавливает и ширину и начальный текст.

**Методы**

|  |  |
| --- | --- |
| **get/setText(String text)** | Занести текст в поле и считать. |
| **getSelectedText()** | Позволяет получить выделенную часть текста |
| **replaceSelection(String content)** | Заменить выделенный текст другим |
| **get/setCaretPosition()** | Возвращает позицию курсора (каретки) в текстовом поле и позволяет задать ее программно. |
| **setCaretColor(Color color)** | Изменить цвет курсора |
| **setHorizontalAlignment(int align)** | Выравнивание текстового поля в качестве параметра передается одна из констант выравнивания, определенных в этом же классе **JTextField: LEFT, CENTER, RIGHT**. |

## Поле для ввода пароля JPasswordField

JPasswordField является прямым потомком JTextField, поэтому для него справедливо все сказанное выше.

|  |  |
| --- | --- |
| set/getEchoChar(char echo) | Получить или заменить символы которые скрываются |
| getPassword() | Считать поля возвращающий массив символов char[]. |

## Область для ввода текста JTextArea

JTextArea также является потомком JTextField и наследует все его методы.

|  |  |
| --- | --- |
| [http://wiki.it-wiki.org.ua/lib/exe/fetch.php/9_text.gif](http://wiki.it-wiki.org.ua/lib/exe/detail.php/9_text.gif?id=java_swing) | Пример текстовой области |
| setWrapStyleWord(boolean wrapStyle) | если TRUE то слова переносятся целиком на новую строку. |
| append(String text) | добавляет строку text в конец уже имеющегося текста |
| insert(String text, int position) | вставляет текст в указанную позицию position |
| setEditable(Bool b) | Задаем можно писать символы или нет |

## Панель прокрутки JScrollPane

**JScrollPane** — панель прокрутки. Чаще всего она просто «надевается» на требуемый объект посредством собственного конструктора, принимающего этот объект в качестве параметра. Например, чтобы текстовая область textArea из предыдущего примера обрела полосы прокрутки, необходимо заменить команду

getContentPane().add(textArea);

на команду

getContentPane().add(new [JScrollPane](http://www.google.com/search?hl=en&q=allinurl%3Adocs.oracle.com+javase+docs+api+jscrollpane)(textArea));

|  |  |
| --- | --- |
| **setHorizontalScrollBarPolicy(int policy)** | позволяет задать стратегию работы с горизонтальной полосой прокрутки. Возможные значения представлены константами **HORIZONTAL\_SCROLLBAR\_ALWAYS (отображать всегда), HORIZONTAL\_SCROLLBAR\_AS\_NEEDED (отображать при необходимости) и HORIZONTAL\_SCROLLBAR\_NEVER (не отображать никогда)**. Данные константы определены в интерфейсе ScrollPaneConstants. |
| **setVerticalScrollBarPolicy(int policy)** | позволяет задать стратегию работы с вертикальной полосой прокрутки посредством констант **VERTICAL\_SCROLLBAR\_ALWAYS, VERTICAL\_SCROLLBAR\_AS\_NEEDED и VERTICAL\_SCROLLBAR\_NEVER** |

## Инструментальная панель JToolBar

В Swing для инструментальных панелей разработан визуальный компонент JToolBar, в котором заложена просто потрясающая функциональность.

|  |  |
| --- | --- |
| **JToolBar(String title)** | создает горизонтальную панель с заданным заголовком |
| **JToolBar(String title, int orientation)** | Для создания вертикальной панели где параметр orientation задается константой **VERTICAL** из интерфейса SwingConstants. |
| **is/setFloatable(boolean floatable)** | разрешает либо запрещает (по умолчанию разрешает) пользователю откреплять панель от места ее начального расположения. |
| **add(Component component)** | добавляет на инструментальную панель новый элемент управления |
| **addSeparator()** | добавляет разделитель |

## Выпадающий список JComboBox

Выпадающий список — весьма распространенный элемент управления. Он содержит множество вариантов, из которых пользователь может выбрать один и только один, либо (если выпадающий список это позволяет) ввести свой собственный.

|  |  |
| --- | --- |
| **JComboBox(Object[] elements)**  **JComboBox(Vector elements)** | конструкторы |
| **addItem(Object item)** | добавить новый элемент в конец списка. |
| **insertItemAt(Object item, int index)** | уточнить позицию, в которую требуется вставить элемент |
| **getItemAt(int index)** | обратиться к произвольному элементу |
| **removeAllItems()** | Удаление из JComboBox всех элементов |
| **removeItem(Object item)** | Удалить конкретный элемент |
| **getSelectedIndex()** | получить индекс выбранного пользователем элемента |
| **getSelectedItem()** | возвращает сам выбранный объект |
| **setSelectedIndex(int index) или setSelectedItem(Object item)** | Сделать конкретный элемент выбранным |
| **setEditable(boolean editable)** | пользователь может ввести свой вариант, который не присутствует в списке, должен быть вызван метод с параметром true. |

## Ползунок JSlider

Ползунок позволяет пользователю выбрать некоторое число из диапазона доступных значений, наглядно представив этот диапазон.

|  |  |
| --- | --- |
| **JSlider(int orientation, int min, int max, int value)** | Конструктор первый параметр ориентация ползунка **(HORIZONTAL или VERTICAL)**. |
| **setOrientation(int)** | Изменить позицию |
| **setMinimum(int min)** | Изменить минимальное значения |
| **setMaximum(int max)** | * - Изменить максимальное значения. |
| **get/setValue(int value)** | - Получить и установить текущее значения |
| **setMajorTickSpacing(int spacing)** | задать расстояние, через которое будут выводиться большие деления. |
| **setMinorTickSpacing(int spacing)** | Расстояние, через которые будут выводиться маленькие деления |
| **setPaintTicks(boolean paint)** | Включает или отключает прорисовку этих делений |
| **setSnapToTicks(boolean snap)** | Включает или отключает «прилипание» ползунка к делениям |
| **setPaintLabels(boolean paint)** | включает или отключает прорисовку меток под большими делениями |
| [http://wiki.it-wiki.org.ua/lib/exe/fetch.php/9_slider.gif](http://wiki.it-wiki.org.ua/lib/exe/detail.php/9_slider.gif?id=java_swing) | Пример элемента |

## Список JList

Список содержит группу элементов, аналогично выпадающему списку JComboBox, но обладает двумя отличительными особенностями. Во-первых, на экране видны одновременно несколько элементов списка. Во-вторых, пользователь может выбрать в списке не один элемент, а несколько (если установлен соответствующий режим выделения).

Создать список можно с помощью конструктора, работающего на основе массива Object[] или вектора Vector (**аналогично JComboBox).**

|  |  |
| --- | --- |
| **setVisibleRowCount(int count)** | * устанавливает количество видимых элементов списка. |
| **setSelectionMode(int mode)** | указывает на ListSelectionModel:  **SINGLE\_SELECTION**— может быть выделен только один элемент,  **SINGLE\_INTERVAL\_SELECTION**— может быть выделено несколько элементов, но составляющих непрерывный интервал,  **MULTIPLE\_INTERVAL\_SELECTION** — может быть выделено произвольное количество смежных и несмежных элементов. |
| **set/getSelectedValue()** | Выделенный элемент списка (если он один |
| **set/getSelectedValues()** | Возвращает все выделенные элементы списка в виде массива Object[]. |
| **set/getSelectedIndex() и set/getSelectedIndices()** | -возвращают индексы элементов списка |

## JTable

Свойства определения внешнего вида таблицы JTable

|  |  |
| --- | --- |
| **Свойства** | **Описание** |
| selectionBackground, selectionForeground | Управление выделением цвета прорисовки фона выделенной ячейки и текста выделенной ячейки. Вместе со стандартными свойствами background и foreground, отвечающими за фон и цвет компонента Swing, позволяют настроить цветовую гамму таблицы |
| rowHeight | Определение высоты всех строк таблицы. С помощью перегруженного метода set можно отдельно задать высоту некоторой строки таблицы. Данный метод используется в примере. |
| intercellSpacing | Управление расстоянием между ячейками как по оси X, так и по оси Y. В примере расстояние задается в виде объекта Dimension. |
| showVerticalLines, showHorizontalLines, | Управление прорисовкой вертикальных и горизонтальных линий сетки таблицы. |
| showGrid | Управление прорисовкой (показать или скрыть) сетки таблицы. |
| gridColor | Определение цвета сетки таблицы JTa |

Основные методы модели данных TableModel

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| int getRowCount() | Метод возвращает количество строк в таблице. |
| int getColumnCount() | Метод возвращает количество столбцов в таблице. |
| Object getValueAt(строка, столбец) | Чтение данных ячейки таблицы. Данные могут иметь определенный тип. Метод возвращает ссылку на базовый тип Object. |
| setColumnName(столбец) | Метод определения имени столбца, которое будет отображаться в заголовке таблицы JTableHeader. Заголовок таблицы появляется при размещении таблицы в панели прокрутки. |
| isCellEditable(строка, столбец) | Метод определения возможности редактирования ячейки таблицы. |
| setValueAt(значениe, строка, столбец) | Метод используется для определения значения ячейки таблицы. Реализуйте этот метод, если в таблице есть редактируемые ячейки, иначе их значение невозможно будет поменять. |
| getColumnClass(столбец) | Метод определения типа данных, хранимых в столбце. Тип задается в виде объекта Class. На основе типа данных определяется, как следует отображать и редактировать эти данные. Таблица JTable стандартно поддерживает несколько типов данных для столбцов. |

## Модель столбцов таблицы TableColumnModel

Модель **TableColumnModel** позволяет настраивать столбцы таблицы *JTable* : устанавливать размеры столбцов, менять их местами, настраивать объекты для отображения в столбце и для редактирования ячеек столбца, управлять заголовком столбца и т.д.

Таблица *JTable* отображает в интерфейсе порядок столбцов согласно параметрам, определенным в **TableColumnModel**. Каждый столбец таблицы представлен объектом *TableColumn*, c помощью которого указываются :

* размеры столбца - *столбец TableColumn, как обычный компонент, имеет предпочтительный, максимальный и минимальный размеры;*
* индекс столбца - *в модели данных TableModel по этому индексу таблица определяет, какой набор данных отображает столбец;*
* объекты - *для отображения и редактирования данных, хранящихся в столбце.*

С помощью **TableColumn** можно настроить и другие параметры, в том числе и модель выделения столбцов.

В **JTable** для хранения информации о столбцах по умолчанию используется стандартная модель*DefaultTableColumnModel*. В подавляющем большинстве случаев ее возможностей хватает. Модель столбцов таблицы хранит список *TableColumn* и при смене какого-либо свойства столбца или его расположения оповещает об этом событии слушателей *TableColumnModelListener*. В следующем примере демонстрируется обработка различных событий модели столбцов с использованием слушателя *TableColumnModelListener*.

Свойства модели TableColumnModel

|  |  |
| --- | --- |
| **Методы и свойства** | **Описание** |
| addColumn(), removeColumn() | Методы динамического управления количеством хранимых в модели TableColumnModel столбцов. Первый метод добавляет к модели новый столбец, второй удаляет из модели заданный столбец. |
| addColumnModelListener(), removeColumnModelListener() | Методы подключения и отключения слушателей TableColumnModelListener. |
| getColumnCount() | Функция получения количества столбцов модели. |
| getColumn(int idx) | Функция получения столбца по его порядковому номеру. |
| Enumeration getColumns() | Функция получения всех содержащихся в модели столбцов. |
| moveColumn(int columnIndex, int newIndex) | Метод изменения позиции столбца. |
| columnMargin (методы get/set) | Свойство управления расстоянием между столбцами таблицы (в пикселах). |

Свойства столбца TableColumn

|  |  |
| --- | --- |
| **Свойства** | **Описание** |
| modelIndex | Индекс столбца в модели данных TableModel. Данные столбца отображает объект TableColumn. |
| preferredWidth, minWidth, maxWidth | Предпочтительный, минимальный и максимальный размеры столбца таблицы. |
| cellRenderer, cellEditor, headerRenderer | Объекты для отображения и редактирования ячеек столбца, а также объект, отображающий заголовок столбца. |
| headerValue, identifier | Первое свойство хранит отображаемое в заголовке столбца значение, а второе позволяет связать со столбцом уникальный идентификатор. |

## Модель выделения SelectionModel

Ячейки таблицы можно выделять разными способами : строками, столбцами, интервалами, единичными ячейками и т.д. Эту задачу решают модели выделения **SelectionModel**, которые хранят выделенные строки и столбцы таблицы. Следует отметить, что и для столбцов, и для строк таблицы имеются собственные модели выделения, представленные интерфейсом *ListSelectionModel*. Таким образом таблица имеет две модели выделения. Первая модель хранится непосредственно в **JTable** и отвечает за выделение строк таблицы. Вторая модель отвечает за выделение столбцов таблицы и хранится в модели столбцов таблицы*TableColumnModel*.

Модель выделения списка использует один из трех режимов выделения :

* **SINGLE\_SELECTION** - режим выделения одиночных элементов;
* **SINGLE\_INTERVAL\_SELECTION** - режим выделения непрерывными интервалами;
* **MULTIPLE\_INTERVAL\_SELECTION** - режим выделения произвольного набора строк или столбцов.

Таким образом, для строк и для столбцов имеется по три режима выделения. То есть, таблица предоставляет девять комбинаций. По умолчанию в таблице и для строк, и для столбцов используется стандартная реализация интерфейса *ListSelectionModel* — класс *DefaultListSelectionModel*.

Основные методы и свойства SelectionModel

|  |  |
| --- | --- |
| **Методы или свойства** | **Описание** |
| selectionModel | Свойство хранит модель выделения в JTable. По умолчанию используется стандартная реализация DefaultListSelectionModel. |
| rowSelectionAllowed, columnSelectionAllowed | Первое свойство - включение и отключение выделения строк. Второе свойство - включает или отключает выделение столбцов (имеется и в модели выделения столбцов и в классе JTable). По умолчанию выделение по столбцам отключено. |
| selectionMode | Свойство моделей выделения таблицы JTable. |
| selectAll() | Выделение всех ячеек таблицы. |
| clearSelection() | Снятие выделения со всех ячеек. |
| cellSelectionAllowed | Метод разрешения или запрещения выделять отдельные ячейки таблицы. |
| rowSelectionInterval, columnSelectionInterval (add, set, remove) | Методы выделения строк или столбцов. Методы «add» присоединяют к уже имеющемуся выделению новые строки и столбцы, методы «set» заменяют имеющееся выделение новым. Методы «remove» удаляют из выделения указанные строки или столбцы. |

Методы получения информации о выделениях модели SelectionModel

|  |  |
| --- | --- |
| **Методы** | **Описание** |
| isSelectedIndex() | Функция проверки выделения элемента в некоторой позиции. |
| isRowSelected(), isColumnSelected() | Методы определения, выделена ли строка или столбец или нет. |
| isCellSelected() | Метод определения, выделена ли ячейка или нет. |
| getSelectedRows(), getSelectedColumns() | Методы получения массивов индексов выделенных строк и столбцов. |

## Заголовок таблицы JTableHeader

Заголовок таблицы **JTableHeader** позволяет отображать названия столбцов, менять столбцы местами, изменять размеры столбцов. Таблица **JTable** автоматически добавляет заголовок таблицы в панель прокрутки, если таблица размещается в панели прокрутки. Если таблица будет размещаться в любом другом контейнере, то для заголовка таблицы места не найдется.

Данные **JTableHeader** получает из модели столбцов таблицы *TableColumnModel*, которая передается в конструктор класса JTableHeader или присоединяется к заголовку позже. Фактически заголовок таблицы играет роль дополнительной модели столбцов. Никаких данных в самом заголовке не хранится — все находится в модели TableColumnModel.

Возможности стандартного **JTableHeader** ограничены. Можно разрешить или запретить перетаскивание столбцов, изменение их размеров, а также определить внешний интерфейс.

# Диалоговые окна FileDialog ...

Класс **Dialog** бибилиотеки SWT является базовым классом следующих диалоговых окон :

* FileDialog - окно выбора и сохранения файла;
* [DirectoryDialog](http://java-online.ru/swt-filedialog.xhtml#directorydialog) - окно выбора и создания директории;
* [PrintDialog](http://java-online.ru/swt-filedialog.xhtml#printdialog) - окно выбора принтера, установки различных параметров печати и отправки задания на печать.
* [FontDialog](http://java-online.ru/swt-filedialog.xhtml#fontdialog) - окно выбора шрифта системы;
* ColorDialog - окно выбора цвета;
* [MessageBox](http://java-online.ru/swt-filedialog.xhtml#messagebox) - окно вывода сообщений/предупреждений;

## Диалоговое окно FileDialog

#### Конструкторы FileDialog

FileDialog(Shell parent)

FileDialog(Shell parent, int style)

Конструктор создания диалогового окна работы с файлами *FileDialog* в качестве параметра принимает родительское окно *parent*. Второму конструктору дополнительно необходимо передать стиль *style*, который может принимать одно из значений [SWT.SAVE, SWT.OPEN, SWT.MULTI] и определяет интерфейс и поведение диалогового окна **FileDialog**.

#### Основные методы FileDialog

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| String getFileName() | Функция получения выбранного файла относительно пути фильтра. Если файл не выбран, то функция вернет пустую строку |
| String[] getFileNames() | Функция получения массива файлов относительно пути фильтра. Если файлы не выбраны, то функция вернет пустой массив |
| String[] getFilterExtensions() | Функция получения расширений файлов, которые FileDialog будет использовать для фильтрации |
| int getFilterIndex() | Функция получения индекса массива описаний фильтра, который был выбран пользователем. Отсчет от 0. Функция вернет -1, если фильтр не был использован |
| String getFilterPath() | Функция получения пути фильтра директории, который использовал FileDialog. Если фильтр не был использован, то функция вернет пустую строку |
| String open() | Функция открытия диалогового окна FileDialog и размещение его над всеми остальными окнами |
| void setFileName(String string) | Определение наименования файла (может быть null), который FileDialog выделит при открытии окна |
| void setFilterExtensions(String[] extensions) | Определение расширений файлов (может быть null), которые FileDialog будет использовать для фильтрации файлов |
| void setFilterIndex(int index) | Определение расширения файлов согласно индексу массива фильтров, который FileDialog будет использовать при инициализации окна |
| void setFilterNames(String[] names) | Определение наименований фильтров согласно списку расширений, которые использует FileDialog |
| void setFilterPath(String string) | Определение пути директории (может быть null), которые FileDialog будет использовать для представления файловой системы |

#### Пример использования FileDialog с фильтрацией

В следующем коде формируется диалоговое окно выбора файла с использованием фильтра. Для подключения фильтра вызываются методы *setFilterNames* и *setFilterExtensions*. Описание расширений файлов для фильтра представлено в переменноЙ *FILTERS*.

// Описание файловых фильтров

private static final String[][] FILTERS = {{"Файлы Word (\*.docx)" , "\*.docx"},

{"Файлы Excel (\*.xlsx)", "\*.xlsx"},

{"Файлы Adobe (\*.pdf)" , "\*.pdf" },

{"Все файлы (\*.\*)" , "\*.\*" }};

private void setFilters(FileDialog dialog)

{

String[] names = new String[FILTERS.length];

String[] exts = new String[FILTERS.length];

for (int i = 0; i < FILTERS.length; i++) {

names[i] = FILTERS[i][0];

exts [i] = FILTERS[i][1];

}

// Определение фильтра диалога FileDialog

dialog.setFilterNames(names);

dialog.setFilterExtensions(exts);

}

. . .

// Диалоговое окно чтения файла

FileDialog dlg = new FileDialog(shell, SWT.OPEN);

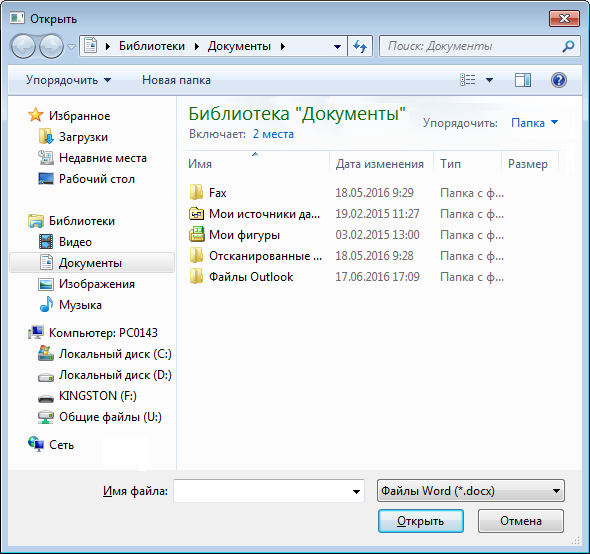
setFilters(dlg);

String fname = dlg.open();

if (fname != null)

System.out.println ("" + fname);

Интерфейс диалогового окна выбора файла представлен на следующем скриншоте. Выпадающий список с фильтрами размещается в правом нижнем углу над кнопками управления.



В следующем примере для сохранения нового файла конструктору **FileDialog** передается опция *SWT.SAVE*.

// Диалоговое окно сохранения файла

FileDialog dlg = new FileDialog(shell, SWT.SAVE);

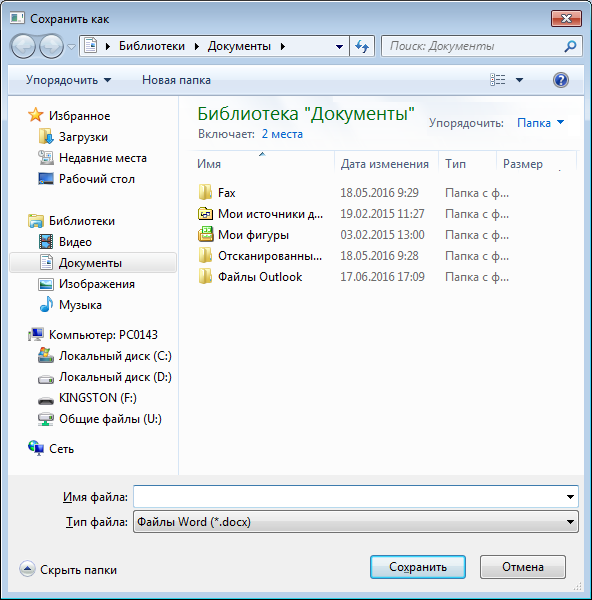
setFilters(dlg);

String fname = dlg.open();

if (fname != null)

System.out.println ("" + fname);

Интерфейс диалогового окна сохранения файла представлен на следующем скриншоте. Выпадающий список с фильтрами размещается в нижней части под компонентом определения наименования файла.



# Окно сообщений MessageBox

**MessageBox** позволяет вывести информацию в диалоговом окне сообщений. *Класс MessageBox нельзя наследовать*.

**Конструкторы MessageBox**

MessageBox(Shell parent)

MessageBox(Shell parent, int style)

**Стили MessageBox**

Стиль *style* позволяет определить внешний вид окна сообщений с представлением определенного изображения и кнопок управления :

* изображение в окне - ICON\_ERROR, ICON\_INFORMATION, ICON\_QUESTION, ICON\_WARNING, ICON\_WORKING
* кнопки - OK, OK | CANCEL
* кнопки - YES | NO, YES | NO | CANCEL
* кнопки - RETRY | CANCEL
* кнопки - ABORT | RETRY | IGNORE

**Основные методы MessageBox**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| String getMessage() | Функция получения текста сообщения (строка может быть пустой); |
| int open() | Открыть окно сообщений MessageBox поверх остальных окон; |
| void setMessage(String string) | Метод определения текста сообщения. |

**Пример открытия окна сообщения MessageBox**

int style = SWT.APPLICATION\_MODAL | SWT.OK | SWT.CANCEL;

MessageBox messageBox = new MessageBox (shell, style);

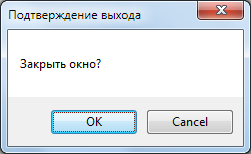
messageBox.setText ("Подтверждение выхода");

messageBox.setMessage ("Закрыть окно?");

int rc = return messageBox.open();

System.out.println ("return code : " + rc);

Интерфейс примера окна сообщений **MessageBox** представлен на следующем скриншоте.



**Исключения диалоговых окон**

Диалоговые окна могут вызвать следующие исключения :

* IllegalArgumentException : ERROR\_NULL\_ARGUMENT если parent не определен (null);
* SWTException
  + ERROR\_THREAD\_INVALID\_ACCESS
  + ERROR\_INVALID\_SUBCLASS

**Диалоговое окно JFileChooser**

Swing включает мощное средство для работы с файлами — компонент *JFileChooser* —, представляющий контейнер, в котором расположены несколько компонентов, списков и кнопок, «управляющих» выбором файлов. **JFileChooser** можно добавить в любое место пользовательского интерфейса, поскольку это весьма гибкий компонент, позволяющий тонко настраивать внешний вид. При необходимости можно полностью изменить стандартное расположение входящих в *JFileChooser* компонентов и добавить дополнительные элементы, такие как панели предварительного просмотра файлов.

Все стандартные диалоговые окна Swing имеют собственные UI-представители, отвечающие за интерфейс окна в используемом приложении. Это особенно важно для внешних видов окон, имитирующих известные платформы, пользователи которых не должны ощущать значительной разницы при переходе от «родных» приложений к Java-приложения. UIManager позволяет выполнять настройку и локализацию интерфейса диалогового окна **JFileChooser**.

Начиная с выпуска JDK 1.3 библиотека Swing предлагает легко настраиваемый инструмент **JFileChooser** для выбора файлов и при необходимости каталогов. Особенности различных файловых систем скрыты в подклассах абстрактного класса *FileSystemView*, который представляет внешний вид файловой структуры согласно используемой операционной системе.

*JFileChooser* — это обычный компонент, унаследованный от класса JComponent, так что можно включить его в любое место интерфейса. Настроить и вывести на экран несложное диалоговое окно для открытия файла или сохранения в нем данных совсем легко.

**Конструкторы JFileChooser**

// Создание JFileChooser с указанием директории пользователя по умолчанию

JFileChooser()

// Создание JFileChooser с указанием currentDirectory директории

JFileChooser(File currentDirectory)

// Создание JFileChooser с указанием currentDirectory директории и

// файловой системы

JFileChooser(File currentDirectory, FileSystemView fsv)

// Создание JFileChooser с определенной файловой системы

JFileChooser(FileSystemView fsv)

// Создание JFileChooser с указанием currentDirectoryPath пути

JFileChooser(String currentDirectoryPath)

// Создание JFileChooser с указанием currentDirectoryPath пути и

// файловой системы

JFileChooser(String currentDirectoryPath, FileSystemView fsv)

**Основные метод JFileChooser**

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| File getCurrentDirectory() | Функция чтения текущей директории |
| String getDialogTitle() | Функция чтения заголовка окна |
| int getDialogType() | Функция чтения типа диалогового окна |
| FileFilter getFileFilter() | Функция чтения текущего фильтра |
| File getSelectedFile() | Функция чтения выделенного файла |
| File[] getSelectedFiles() | Функция получения списка выделенных файлов, если установлен флаг выделения нескольких файлов MULTI\_SELECTION\_ENABLED\_CHANGED\_PROPERTY |
| void setCurrentDirectory(File dir) | Метод определения текущей директории |
| void setDialogTitle(String dialogTitle) | Метод определения заголовка диалогового окна |
| void setDialogType(int dialogType) | Метод определения типа диалогового окна |
| void setFileFilter(FileFilter filter) | Метод установки файлового фильтра |
| void setFileSelectionMode(int mode) | Метод определения выделяемых объектов - файлы, директории или файлы с директориями |
| void setMultiSelectionEnabled(boolean b) | Метод определения возможности выделения нескольких файлов |
| void setSelectedFile(File file) | Метод выделения файла |
| void setSelectedFiles(File[] selectedFiles) | Метод выделения списка файлов, если установлен флаг выделения нескольких файлов MULTI\_SELECTION\_ENABLED\_CHANGED\_PROPERTY |
| int showDialog(Component parent, String approveButtonText) | Функция открытия окна выбора файла с настроенным наименованием кнопки |
| int showOpenDialog(Component parent) | Функция открытия диалогового окна «Открыть файл» |
| int showSaveDialog(Component parent) | Функция открытия диалогового окна «Сохранить файл» |

**Режимы работы JFileChooser**

Перед открытием диалогового окна для выбора файлов или директории необходимо определить режим работы **JFileChooser**. Компонент *JFileChooser* может работать в одном из трех режимов, который сохраняется в свойстве *fileSelectionMode* :

* FILES\_ONLY - доступны только файлы, независимо от того, сохраняется файл или открывается. По умолчанию *JFileChooser* работает именно в этом режиме.
* FILES\_AND\_DIRECTORIES - доступны каталоги и файлы. Этот режим следует использовать только в том случае, когда необходимо поменять общие свойства файловой системы (файлы не отличаются от каталогов).
* DIRECTORIES\_ONLY - доступны только каталоги.

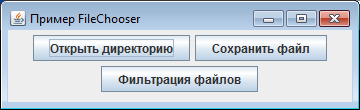
Для определения режима используется метод *setFileSelectionMode(mode)*.

**Возвращаемые компонентом JFileChooser значения**

* APPROVE\_OPTION - выбор файла в диалоговом окне прошел успешно; выбранный файл можно получить методом getFile();
* CANCEL\_OPTION - выбор файла отменен нажатием на кнопке *Cancel*;
* ERROR\_OPTION - при выборе файла произошла ошибка, или было закрыто диалоговое окно выбора файла.

**Пример использования JFileChooser**

В примере *FileChooserTest.java* используется диалоговое окно выбора файла и директории с использованием компонента **JFileChooser**. Интерфейс окна включает 3 кнопки, по нажатию на которые открывается соответствующее диалоговое окно. Интерфейс главного окна представлен на следующем скриншоте.



import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class Example

{

public static void run()

{

//Создадим окно и установим заголовок

final JFrame window = new JFrame("Caption");

//Подключаем иконку из корня папки проекта

ImageIcon img = new ImageIcon("java.png");

window.setIconImage(img.getImage());

//Событие "закрыть" при нажатии по крестику окна

window.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//Текстовое поле

JTextField textField = new JTextField();

textField.setBackground(Color.WHITE);

textField.setColumns(14);

//Создадим панель

JPanel panel = new JPanel();

//Создадим кнопки

JButton minButton = new JButton("Свернуть");

JButton maxButton = new JButton("Растянуть");

JButton normalButton = new JButton("Оригинал");

JButton exitButton = new JButton("Выход");

JButton helloButton = new JButton("Hello");

//Событие для кнопки "Свернуть"

minButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

//Действие

window.setState(JFrame.ICONIFIED);

}

});

//Событие для кнопки "Растянуть"

maxButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

//Действие

window.setExtendedState(JFrame.MAXIMIZED\_BOTH);

}

});

//Событие для кнопки "Оригинал"

normalButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

//Действие

window.setExtendedState(JFrame.NORMAL);

}

});

//Событие для кнопки "Выход"

exitButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

//Действие

window.setVisible(false);

System.exit(0);

}

});

//Событие для кнопки "Hello"

helloButton.addActionListener(new ActionListener()

{

public void actionPerformed(ActionEvent e)

{

//Действие

textField.setText("Hello World!");

}

});

//Добавим кнопки и поля на панель

panel.add(minButton);

panel.add(maxButton);

panel.add(normalButton);

panel.add(exitButton);

panel.add(textField);

panel.add(helloButton);

//Добавим панель в окно

window.getContentPane().add(panel);

window.pack();

//Разместим программу по центру

window.setLocationRelativeTo(null);

window.setVisible(true);

}

//Запускаем

public static void main(String[] args)

{

run();

}

//Больше документации https://docs.oracle.com/javase/tutorial/uiswing/components/componentlist.html

}