1.

package messenger\_client**;**

Пакеты, которые мы создали сами

Javascript library for sending a message through Facebook Messenger Send API.

Пакет является специальным библиотечным модулем, который содержит группу классов, объединённых в одном пространстве имён. Например, существует системная библиотека **android.widget**, в состав которой входит класс **Button**. Чтобы использовать класс в программе, можно привести его полное имя **android.widget.Button**. Но длинные имена классов не слишком удобно использовать в коде, поэтому можно использовать ключевое слово **import**.

<http://developer.alexanderklimov.ru/android/java/package.php>

2.

### Импорт пакетов и классов

Если нам надо использовать классы из других пакетов, то нам надо подключить эти пакеты и классы. Исключение составляют классы из пакета **java.lang** (например, String), которые подключаются в программу автоматически.

Например, знакомый по прошлым темам класс **Scanner** находится в пакете **java.util**, поэтому мы можем получить к нему доступ следующим способом:

java.util.Scanner in = new java.util.Scanner(System.in);

То есть мы указываем полный путь к файлу в пакете при создании его объекта. Однако такое нагромождение имен пакетов не всегда удобно, и в качестве альтернативы мы можем импортировать пакеты и классы в проект с помощью директивы **import**, которая указывается после директивы package:

package study;

import java.util.Scanner; // импорт класса Scanner

public class Program {

public static void main(String[] args) {

Scanner in = new Scanner(System.in);

}

}

Директива import указывается в самом начале кода, после чего идет имя подключаемого класса (в данном случае класса Scanner).

В примере выше мы подключили только один класс, однако пакет java.util содержит еще множество классов. И чтобы не подключать по отдельности каждый класс, мы можем сразу подключить весь пакет:

import java.util.\*; // импорт всех классов из пакета java.util

Теперь мы можем использовать любой класс из пакета java.util.

Возможна ситуация, когда мы используем два класса с одним и тем же названием из двух разных пакетов, например, класс Date имеется и в пакете *java.util*, и в пакете *java.sql*. И если нам надо одновременно использовать два этих класса, то необходимо указывать полный путь к этим классам в пакете:

java.util.Date utilDate = new java.util.Date();

java.sql.Date sqlDate = new java.sql.Date();

1. import java.awt.EventQueue**;**
2. import javax.swing.JFrame**;**
3. import javax.swing.JPanel**;**
4. import javax.swing.border.EmptyBorder**;**
5. import javax.swing.JLabel**;**
6. import javax.swing.JTextField**;**
7. import javax.swing.JButton**;**
8. import java.awt.GridBagLayout**;**
9. import java.awt.GridBagConstraints**;**
10. import java.awt.Insets**;**
11. import java.awt.event.ActionEvent**;**
12. import java.awt.event.ActionListener**;**

Импорт разных шаблонных библиотек

1. Class **java**.**awt**.**EventQueue**. **EventQueue** is a platform-independent class that queues events, both from the underlying peer classes and from trusted application classes. There is only one **EventQueue** for the system. Remove an event from the queue and return it.

1) Класс java.awt.EventQueue. EventQueue - это независимый от платформы класс, который ставит в очередь события как из базовых одноранговых классов, так и из классов доверенных приложений. Для системы существует только один EventQueue. Удалить событие из очереди и вернуть его.

1. В Java есть три библиотеки визуальных компонентов для создания графического интерфейса пользователя. Самая ранняя из них называется AWT. Считается, что при ее проектировании был допущен ряд недочетов, вследствие которых с ней довольно сложно работать. Библиотека Swing разработана на базе AWT и заменяет большинство ее компонентов своими, спроектированными более тщательно и удобно. Третья, самая новая библиотека, называется SWT.

Каждая библиотека предоставляет набор классов для работы с кнопками, списками, окнами, меню и т.д., но эти классы спроектированы по-разному: они имеют различный набор методов с разными параметрами, поэтому «перевести» программу с одной библиотеки на другую (например, с целью увеличения быстродействия) не так-то просто. Это почти как перейти с одного языка программирования на другой: все языки умеют делать одно и то же, но у каждого из них свой синтаксис, своя программная структура и свои многочисленные хитрости.

По этой причине вместо того, чтобы делать обзор всех трех библиотек, мы постараемся получше разобраться в одной из них — библиотеке Swing. Полноценный графический интерфейс может быть разработан с ее помощью.

# **Использование JPanel контейнера панели**

<https://javaswing.wordpress.com/2009/07/19/using-jpanel/>

1. [**EmptyBorder**](https://docs.oracle.com/javase/7/docs/api/javax/swing/border/EmptyBorder.html#EmptyBorder(java.awt.Insets))([**Insets**](https://docs.oracle.com/javase/7/docs/api/java/awt/Insets.html) borderInsets)

Creates an empty border with the specified insets.

# **Работаем с JLabel**

JLabel это пожалуй самый простой компонент, который есть в Java Swing. С помощью JLabel можно показать текст с иконкой. Если нужен какой-то компонент, чтобы отобразить какое-то сообщение пользователю или сделать для поля ввода текстовую метку, или показать иконку — используем JLabel. Текст, который показывает JLabel выделять нельзя, только смотреть.

<https://javaswing.wordpress.com/2009/08/11/work_with_jlabel/>

1. Пакет javax.swing.text библиотеки Swing содержит компоненты для работы с текстом. Мощные текстовые компоненты позволяют реализовать средства ввода и редактирования текста любой сложности, начиная от однострочного текстового поля и заканчивая многофункциональным текстовым редактором с разнообразными возможностями.

Текстовое поле **JTextField** является самым простым компонентом и наиболее часто встречающимся в пользовательских интерфейсах. Как правило, поле является однострочным и служит для ввода текста. В библиотеке Swing имеется два текстовых поля. Первое, представленное классом *JTextField*, позволяет вводить однострочный текст. Второе поле, реализованное классом **JPasswordField** и унаследованное от поля **JTextField**, дает возможность организовать ввод «закрытой» информации (чаще всего паролей), которая не должна напрямую отображаться на экране.

Оба текстовых поля **JTextField, JPasswordField** просты. Работа с ними чаще всего сводится к заданию количества отображаемых в поле символов и начального текста, если таковой требуется. После чего остается только поместить поле в контейнер и в нужный момент получить из него набранный пользователем текст.

<http://java-online.ru/swing-jtextfield.xhtml>

1. Каждое приложение, которое имеет графический интерфейс пользователя не может обходиться без кнопок. В Java Swing кнопка представлена классом JButton. У кнопки имеются различные методы для ее конфигурирования — установка надписи на JButton, установка иконки, выравнивание текста, установка размеров и так далее. Кроме всего прочего разработчику необходимо навесить на JButton слушателя, который будет выполняться как только пользователь нажмет на кнопку.

<https://javaswing.wordpress.com/2009/07/26/jbutton_pressing/>

## Режим GridBagLayout - заполнение формы

Режим размещения компонентов GridBagLayout удобен для создания диалоговых панелей, содержащих такие компоненты, как текстовые поля редактирования, переключатели, кнопки и так далее.

Выбирая соответствующим образом параметры размещения отдельных компонент путем заполнения соответствующих полей класса GridBagConstraints, можно создавать панели, напоминающие по своему внешнему виду и поведению стандрантые диалоговые панели Windows или других операционных систем с графическим интерфейсом.

При этом можно добиться, чтобы размеры компонент и их взаимное расположение не изменялись при изменении размеров окна контейнера. Это невозможно при использовании других режимов размещения, таких как FlowLayout или GridLayout.

<http://www.frolov-lib.ru/programming/javasamples/vol5/vol5_6/index.html>

1. Class **java**.**awt**.**GridBagConstraints**. The **GridBagConstraints** class specifies constraints for components that are laid out using the GridBagLayout class. This field is used when the component is smaller than its display area. Put the component in the center of its display area.
2. Class **java**.**awt**.**Insets**. The **insets** of a container. This class is used to layout containers. Constructs and initializes a new **Inset** with the specified top, left, bottom, and right **insets**.
3. Class **java**.**awt**.**event**.**ActionEvent**. ... The **event** is passed to every every ActionListener object that registered to receive such **events** using the component's addActionListener method. The object that implements the ActionListener interface gets this **ActionEvent** when the **event** occurs.
4. Interface **ActionListener**. ... The class that is interested in processing an action**event** implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action **event** occurs, that object's actionPerformed method is invoked.

3.

public static void main(String[] args) { //основная функция для реализации программы  
 EventQueue.*invokeLater*(new Runnable(){

<http://qaru.site/questions/219102/javaawteventqueueinvokelater-explained>

Полная обработка Swing выполняется в потоке с именем **EDT (Event Dispatching Thread)**. Поэтому вы должны блокировать GUI, если вы будете вычислять некоторые длительные вычисления в этом потоке.

Здесь вы можете обработать свои вычисления в другом потоке, поэтому ваш графический интерфейс остается отзывчивым. В конце вы хотите обновить свой графический интерфейс, который **должен**быть выполнен в EDT.

Теперь EventQueue.invokeLater вступает в игру. Он отправляет событие (ваш Runnable) в конце списка событий Swings и обрабатывается после обработки всех предыдущих графических событий.

Также возможно использование EventQueue.invokeAndWait. Разница в том, что ваш поток вычислений блокируется до тех пор, пока ваш графический интерфейс не будет обновлен. Таким образом, очевидно, что это не должно использоваться из EDT.

4.

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

# setDefaultCloseOperation()

The setDefaultCloseOperation() method is used to specify one of several options for the close button. Use one of the following constants to specify your choice:

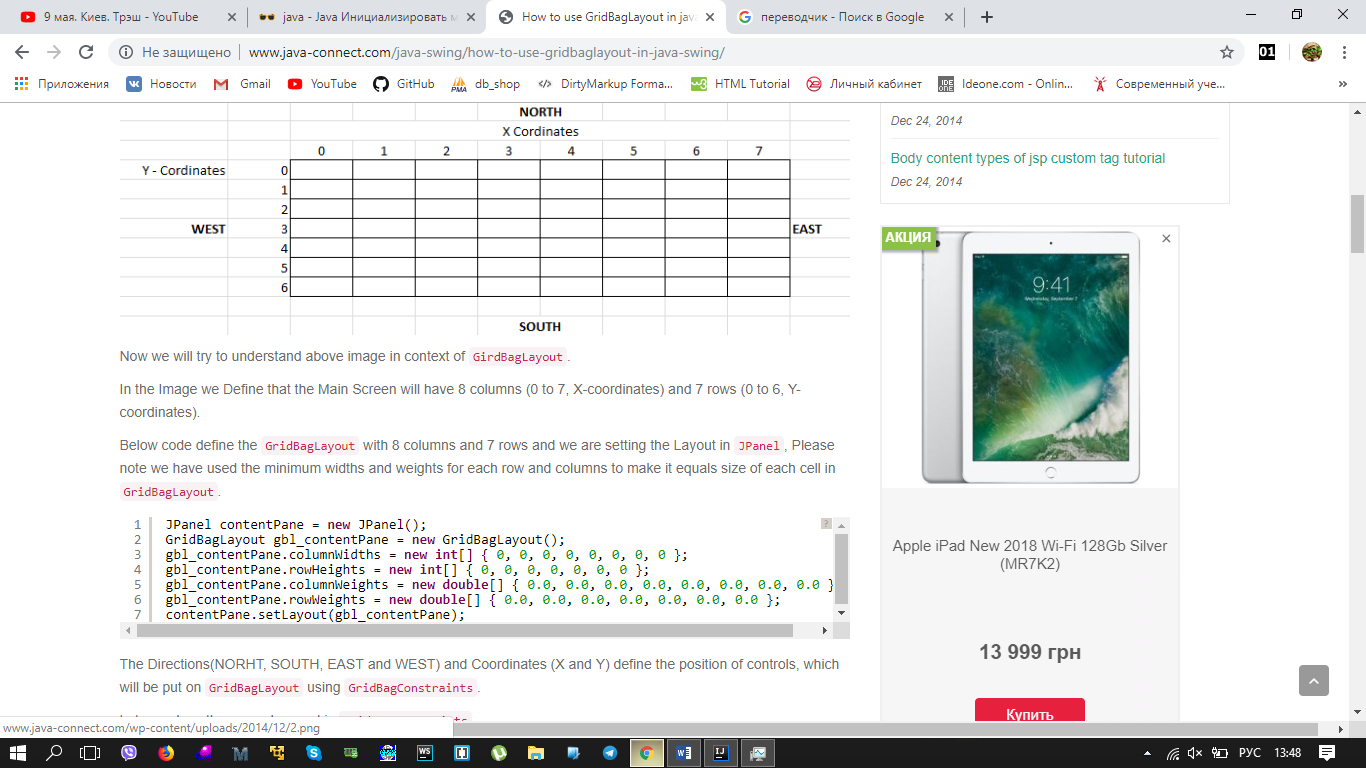
* JFrame.EXIT\_ON\_CLOSE — Exit the application.
* JFrame.HIDE\_ON\_CLOSE — Hide the frame, but keep the application running.
* JFrame.DISPOSE\_ON\_CLOSE — Dispose of the frame object, but keep the application running.
* JFrame.DO\_NOTHING\_ON\_CLOSE — Ignore the click.

If you forget to call setDefaultCloseOperation() you will get JFrame.HIDE\_ON\_CLOSE by default. This can be frustrating, because it looks like you have "killed" the program, but it keeps on running, and you see no frame.

5.

<http://www.java-connect.com/java-swing/how-to-use-gridbaglayout-in-java-swing/>

GridBagLayout gbl\_contentPane = new GridBagLayout()**;** // creating the new object for диалоговая панель, содержащая текстовые поля редактирования, переключатели, кнопки и т.д.  
//задаем массив в конструкторе, инициализируемый 0  
gbl\_contentPane.columnWidths = new int[]{**0, 0, 0, 0**}**;** // ширина колонок диалоговой панели  
gbl\_contentPane.rowHeights = new int[]{**0, 0, 0, 0, 0, 0, 0, 0, 0**}**;**gbl\_contentPane.columnWeights = new double[]{**0.0, 0.0, 1.0,** Double.*MIN\_VALUE*}**;**gbl\_contentPane.rowWeights = new double[]{**0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,** Double.*MIN\_VALUE*}**;**contentPane.setLayout(gbl\_contentPane)**;**



Теперь попробуем разобраться с изображением выше в контексте GirdBagLayout.

В изображении мы определяем, что на главном экране будет 8 столбцов (от 0 до 7, координаты X) и 7 строк (от 0 до 6, координаты Y).

Ниже код определяет GridBagLayout с 8 столбцами и 7 строками, и мы устанавливаем макет в JPanel. Обратите внимание, что мы использовали минимальные значения ширины и веса для каждой строки и столбцов, чтобы сделать его равным размеру каждой ячейки в GridBagLayout.

Направления (NORHT, SOUTH, EAST и WEST) и Координаты (X и Y) определяют положение элементов управления, которые будут помещены в GridBagLayout с использованием GridBagConstraints.

6.

<http://www.frolov-lib.ru/programming/javasamples/vol5/vol5_5/index.html>

Java anchor – a constraint (ограничение) that is used to position that element at a particular (определенное) location when the component is smaller than its display area. anchor can have values CENTER, NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, and NORTHWEST.

gbc\_lblNickname.insets = new Insets(**0, 0, 5, 0**)**;**

A Java Insets object is a representation of the borders of a container. It specifies the space that a container must leave at each of its edges. The space can be a border, a blank space, or a title.

Java gridx и gridy – задают, соответственно, номер столбца и номер строки для ячейки, в которую будет помещен компонент. Верхней левой ячейке соответствуют нулевые значения.

#7

An [ActionListener](http://java.sun.com/javase/6/docs/api/java/awt/event/ActionListener.html) - это механизм обратного вызова. Всякий раз, когда элемент управления добавляется к огню ActionEvent, вызывается метод public void actionPerformed(ActionEvent e).

Я не понимаю, где вызывается actionPerformed. Я вижу, что он определен внутри класса, но не существует места, где этот метод вызывается.

Это вызвано внутренними механизмами компонента пользовательского интерфейса.

@Override

**@Override** ключевое слово, которое позволяет в дочернем классе заново создать реализацию метода родительского класса.

На самом деле аннотация @Override указывает, что далее мы **собираемся** переопределять метод базового класса.  
При этом, если в базовом классе не окажется метода с аналогичной сигнатурой, то мы получим предупреждение компилятора о том, что хотя мы и собирались что-то переопределить, по факту этого не произошло.

На этом действие аннотации заканчивается.

Таким образом, аннотация никак не влияет на сам факт переопределения метода - при совпадении сигнатур с методом базового класса он и так будет переопределен, независимо от наличия, либо отсутствия этой аннотации. Аннотация служит лишь **для контроля** успешности действия при сборке проекта.

В случае отсутствия данной аннотации и не совпадении сигнатур (в результате ошибки) с методом, который мы собирались переопределять - случится страшное - очень трудноуловимая ошибка, когда вы думаете, что должно бы переопределяться, а по факту имеете отдельный независимый метод, который скорее всего вообще не выполняется. При этом компилятор считает, что с его точки зрения все в порядке - хотите собственный метод в классе-наследнике - нет проблем. И молчит.