



Java course

- [Начало Java](#)
- [Проект «Отдел кадров»](#)
- [Курсы](#)
- [Статьи](#)
- [Контакты/Вопросы](#)

- [Введение](#)
- [Установка JDK](#)
- [Основные шаги](#)
- [Данные](#)
- [Порядок операций](#)
- [IDE NetBeans](#)
- [ООП](#)
- [Инкапсуляция](#)
- [Наследование](#)
- [Пакеты](#)
- [Переопределение и перегрузка](#)
- [Полиморфизм](#)
- [Статические свойства и методы](#)
- [Отношения между классами](#)
- [Визуализация работа](#)
- [Пример — очередь объектов](#)
- [Массивы — знакомство](#)
- [Многомерные массивы](#)
- [Абстрактные классы](#)
- [Интерфейсы](#)

Пакеты

Пришло время упомянуть о пакетах. Вполне возможно, что вы уже задумались над следующим вопросом: «Если классов будет много, то их все в одной директории держать? И как потом с ними разбираться?». Вы совершенно правы — в большом приложении классов создается тысячи и десятки тысяч. Конечно же необходим механизм каталогизации. И такой механизм создан. Причем достаточно простой и очевидный — директории. Мы уже привыкли, что на диске наши файлы лежат в разных директориях, которые мы сами организовываем в удобном порядке. В Java сделано тоже самое — физически класс кладется в определенную директорию. Существует даже некоторые правила именования этих директорий. Например для коммерчески проектов директория должна начинаться сначала с префикса «com» а за ним следует название компании — например «mysompany». Далее следует название проекта. Потом уже идет более точное разделение по какому-либо признаку — чаще всего функциональному, но никто не мешает их разбивать даже в алфавитном порядке (хотя это вряд ли хорошая идея). Т.е. когда компания IBM создает проект education, то полный путь до классов вполне может выглядеть вот так: com/ibm/education. Для проектов OpenSource часто начинается все с org. Существуют и другие варианты — например для государственных органов РФ в Санкт-Петербурге начало может выглядеть как ru/spb. Если вы не работали в Unix, то вас возможно смутит, что я пишу разделитель директорий не как в Windows — там используется обратный слэш — «\». Я опять немного забегу вперед — как мы уже говорили, Java является кросс-платформенной системой, т.е. не зависит от операционной системы. Но сама операционная система все-таки влияет на программы на Java. Одним из таких проявлений является разделитель директорий. Открою небольшую тайну — удобнее использовать прямой слэш — «/» — даже

- [Расширенное описание классов](#)
- [Исключения](#)
- [Решения на основе классов](#)
- [Список контактов — начало](#)
- [Коллекции — базовые принципы](#)
- [Коллекции — продолжение](#)
- [Список контактов — GUI приложение](#)
- [Что такое JAR-файлы](#)
- [Многопоточность — первые шаги](#)
- [Многопоточность и синхронизация](#)
- [Работаем с XML](#)
- [Reflection — основы](#)
- [Установка СУБД PostgreSQL](#)
- [Базы данных на Java — первые шаги](#)
- [Возможности JDBC — второй этап](#)
- [JDBC — групповые операции](#)
- [Список контактов — работаем с БД](#)
- [Переезжаем на Maven](#)
- [Потоки ввода-вывода](#)
- [Сетевое взаимодействие](#)
- [С чего начинается Web](#)

в Windows это работает. Когда мы будем знакомиться с системой ввода-вывода, мы узнаем еще более официальный способ.

В итоге все выглядит достаточно понятно, но есть один момент. Вы уже обратили внимание, что NetBeans по умолчанию помещает все наши классы в папку src внутри директории с проектом. Но тогда может возникнуть еще один вопрос: «Мы же не указали вообще никаких пакетов и файл лежит в директории C:\JavaLesson\Robot\src. Значит ли это, что пакет по умолчанию вот так и называется?».

Конечно же нет. На самом деле в проекте на Java всегда существует некоторая корневая директория относительно которой и строится дерево пакетов. В проекте в NetBeans этой корневой директорией является src. Но если мы не используем NetBeans, то какая директория является корневой? Определение корневой директории определяется специальной директивой package в начале файла с классом. Например если я хочу поместить класс Robot в пакет edu.robot и моя корневая директория называется C:\JavaLesson\Robot\src, то я должен сделать три действия:

1. Создать в директории src директорию edu и уже внутри нее robot
2. Поместить файл Robot.java в директорию edu/robot
3. Поместить в файл Robot.java директиву package edu.robot

Полный текст нашего робота (я опустил пока все его методы) будет выглядеть вот так:

```

1 // Название пакета идет в самом начале файла
2 package edu.robot;
3
4 public class Robot {
5     // Код, который мы еще напишем
6 }
```

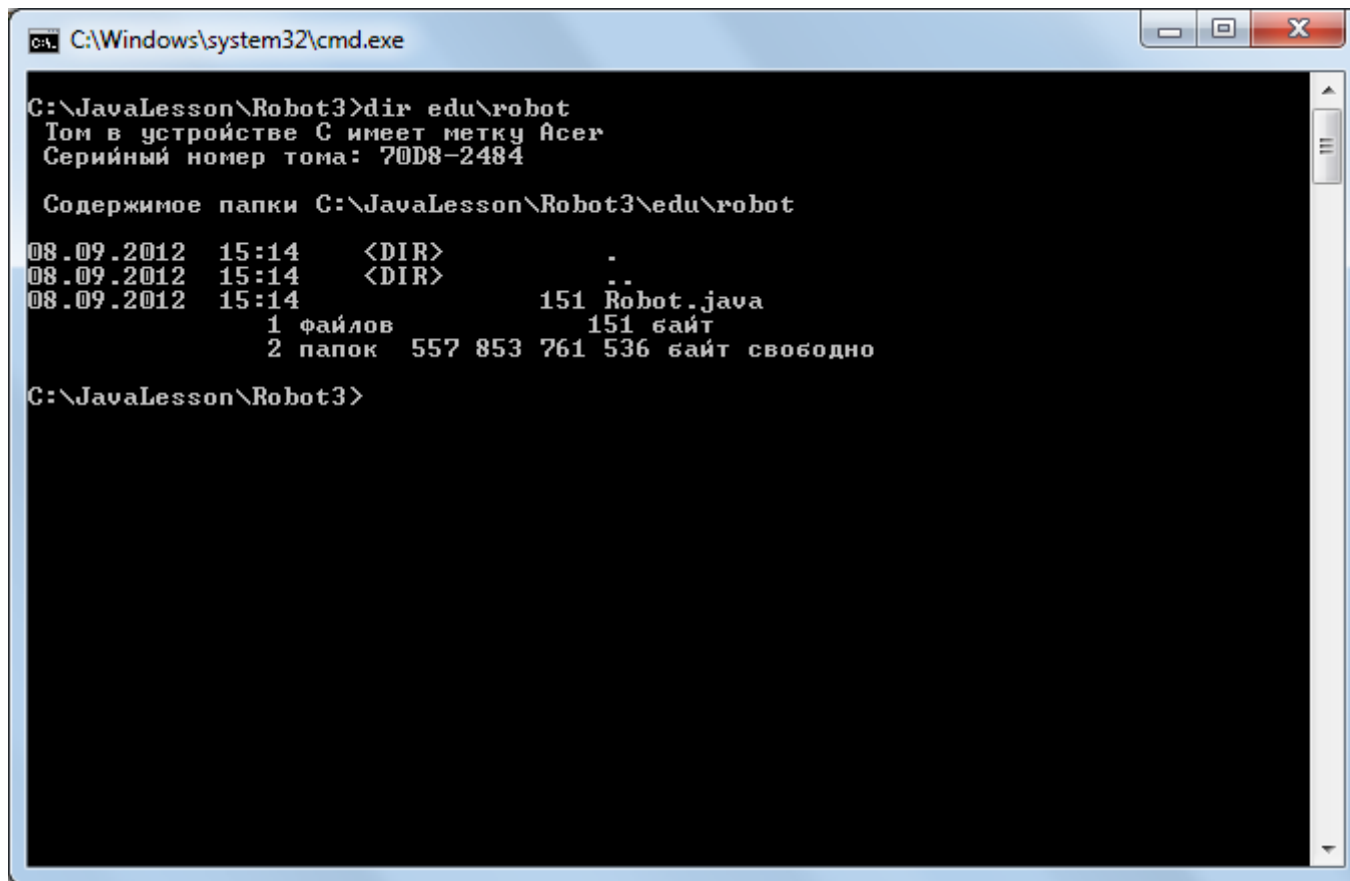
Обратите внимание, что название пакета идет в самом начале файла и разделителем для директорий является не слэш, а точка — чуть дальше мы опять ее увидим.

Теперь компилятор будет знать, что класс объявляет себя находящимся в определенной директории и будет за этим следить. Давайте снова перенесемся в командную строку (надеюсь вы ее еще не забыли) и посмотрим как компилировать и запускать программу, когда класс находится не в корневой директории, а уже в пакете (в поддиректориях). Давайте запустим командную строку (кто не помнит — идем в главу [Основные шаги](#)).

Я создал директорию Robot3 в нашей папке C:\javaLesson и в ней директории edu/robot. В самой «нижней» директории robot я создал файл Robot.java вот такого вида (чтобы его можно было запустить и посмотреть результат):

```
1 package edu.robot;  
2  
3 public class Robot {  
4  
5     public static void main(String[] args) {  
6         System.out.println("Hello from Robot");  
7     }  
8 }
```

Выполним команду `dir edu\robot` и у нас должно получиться то, что вы видите на рисунке.



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The current directory is "C:\JavaLesson\Robot3". The user has entered the command `dir edu\robot`. The output shows system information for drive C: and then the contents of the directory "C:\JavaLesson\Robot3\edu\robot".

```
C:\Windows\system32\cmd.exe  
  
C:\JavaLesson\Robot3>dir edu\robot  
Том в устройстве C имеет метку Acer  
Серийный номер тома: 70D8-2484  
  
Содержимое папки C:\JavaLesson\Robot3\edu\robot  
08.09.2012  15:14    <DIR>          .  
08.09.2012  15:14    <DIR>          ..  
08.09.2012  15:14                151 Robot.java  
                1 файл          151 байт  
                2 папок    557 853 761 536 байт свободно  
  
C:\JavaLesson\Robot3>
```

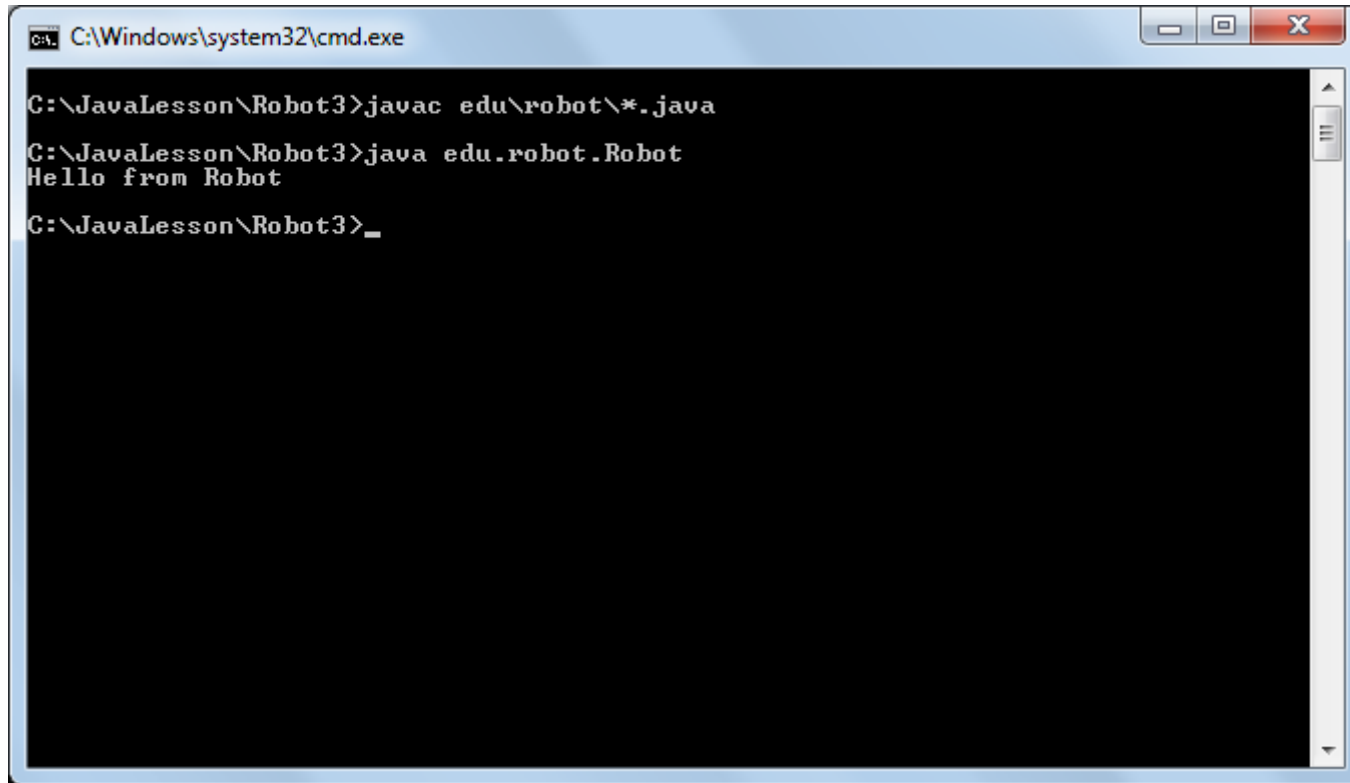
Теперь нам надо скомпилировать наш файл. Для этого надо выполнить команду

```
javac edu\robot\*.java
```

Если у вас все сделано правильно, то компилятор вам ничего не скажет а в директории edu/robot появиться файл Robot.class. Теперь нам надо его выполнить вот такой командой

```
java edu.robot.Robot
```

Результат наших действий можно видеть на рисунке.

A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. The text shows the following sequence of commands and output:

```
C:\JavaLesson\Robot3>javac edu\robot\*.java
C:\JavaLesson\Robot3>java edu.robot.Robot
Hello from Robot
C:\JavaLesson\Robot3>_
```

Здесь важно отметить следующие моменты:

- Нашей текущей директорией является «корневая» директория наших пакетов
- Вызов класса происходит по так называемому «полному имени класса», которое включает полное название пакета и само имя

Полное имя класса — весьма важный момент. Разделение классов по пакетам служит не только для удобства, но решает еще одну важную задачу — уникальность имен классов. Наверняка в большом проекте будет участвовать много людей и каждый будет писать свои классы. И наверняка имена этих классов нередко будут одинаковые. Если опять забежать немного вперед, то вы скорее всего будете подключать внешние библиотеки (мы узнаем, как это делается) и в них будут классы, которые будут называться так же как ваши. Единственным спасением различать их — поместить в разные пакеты. Классы с названиями User, Session, Controller, Handler встречаются достаточно часто и разделение на пакеты спасает нас от кошмара искать

новое подходящее имя для переменной. Кстати хорошее название очень важно при разработке — писать комментарии программисты не любят и хорошее название может заменить много строк комментариев.

А теперь самое время использовать наши знания о классах, как строительных блоках, и новые знания о пакетах. Напишем простое графическое приложение, которое создаст нам обычное окно. Наш класс `SimpleFrame` создаст экземпляр уже готового класса `JFrame` и будет устанавливать его свойства. Представьте себе, что вы получили в распоряжение конструктор с деталями, у которых можно менять размеры, выставлять цвет, надписи и т.д. попробуйте отнестись к классу `JFrame` именно так. Наш класс `SimpleFrame` мы поместили в пакет (что является чуть ли не обязательным — помещать классы в корневую директорию считается очень плохой практикой. Так что первые наши программы были с этой точки зрения ужасающими — не делайте так больше. Хотя иногда для проведения каких-либо простых экспериментов я создаю такие классы). В коде нашего класса мы воспользовались директивой `import` для подключения класса `JFrame`. мы поговорим о ней несколько позже, но сначала посмотрим код. Полный пример проекта можно взять тут — [SimpleFrame](#)

```
1 package edu.javacourse;
2
3 // Мы ИМПОРТИРОВАЛИ класс из пакета
4 import javax.swing.JFrame;
5
6 public class SimpleFrame {
7
8     public static void main(String[] args) {
9         // Создали экземпляр класса - объект
10        JFrame sf = new JFrame();
11        // Установим заголовок
12        sf.setTitle("First window");
13        // Установим свойство - завершить приложение при закрытии окна
14        sf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15        // Выставим координаты и размеры
16        // Верхний левый угол - первые два числа 100 и 100
17        // Ширина и высота - вторые два числа 400 и 200
18        sf.setBounds(100, 100, 400, 200);
19        // Отобразим окно, сделав его видимым
20        sf.setVisible(true);
21        // Теперь у нас на экране появилось окно, которым
22        // можно управлять - перетаскивать, менять размеры.
23        // Разве не красиво ?
24    }
25 }
```

Как видно из кода, мы создали экземпляр объекта класса `JFrame`, а потом использовали его методы для того, чтобы наш объект сделал определенные действия. Методы, которые мы используем УЖЕ существуют в классе `JFrame` и нам остается их только использовать.

Не появилось ощущения, что мы по сути «управляем» нашим объектом, который умеет делать многое ? Когда вы научитесь использовать объекты, программирование превратится в увлекательные конструкторские задачи — как построить систему из готовых объектов и какие объекты (классы) с какими свойствами надо еще добавить. Мы обязательно вернемся к таким задачам — я очень люблю использовать графические программы для демонстрации — они наглядно показывают идею. Предвосхищая вопрос «а где можно посмотреть какие есть классы и методы у классов» могу предложить совершенно не оригинальный способ — читайте документацию. Если ее нет — читайте исходные коды. Если и их нет — я иногда для понимания использую декомпиляцию. Если же классы обработаны специальными средствами, чтобы их нельзя было декомпилировать — может тогда не надо использовать такие классы совсем ?

Но т.к. сейчас мы рассматриваем пакеты, давайте обратим наше внимание на директиву `import`.

Т.к. мы сами не создавали код класса `JFrame` разумно предположить, что он находится где-то вне нашего проекта. Это так и есть — этот класс входит в стандартную библиотеку Java и может использоваться всеми программами. Но если это так, то все-таки зачем нам импортировать этот класс ? Если он есть — ну и здорово. Но мы также говорили, что у классов бывают одинаковые имена (имена файлов). А вот полные имена (с учетом пакетов) у них всегда должны быть разные. Иногда по ошибке программистов может случиться ситуация, в которой два класса будут иметь полностью одинаковые имена. В этом случае возникнет ошибка при загрузке такого класса и программа просто не заработает. Хотя если пойти еще глубже, то и это случай можно сделать работоспособным (больше не буду туманить вам голову — может позже).

Но как вы заметили мы пишем простые имена — в нашей программе мы используем простое имя `JFrame`. Так вот директива `import` и указывает, из какого пакета берется класс `JFrame`, чтобы в остальном коде не надо было использовать полное имя. Хотя мы можем это сделать сами.

```
1 package edu.javacourse;
2
3 public class SimpleFrame {
4
5     public static void main(String[] args) {
6         // Создали экземпляр класса - объект
7         javax.swing.JFrame sf = new javax.swing.JFrame();
8         // Установим заголовок
9         sf.setTitle("First window");
10        // Установим свойство - завершить приложение при закрытии окна
11        sf.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
12        // Выставим координаты и размеры
13        // Верхний левый угол - первые два числа 100 и 100
14        // Ширина и высота - вторы два числа 400 и 200
15        sf.setBounds(100, 100, 400, 200);
16        // Отобразим окно, сделав его видимым
17        sf.setVisible(true);
18        // Теперь у нас на экране появилось окно, которым
19        // можно управлять - перетаскивать, менять размеры.
20        // Разве не красиво ?
21    }
22 }
```

Как видим, в этом варианте мы не делаем импорт, но вынуждены каждый раз при обращении в классу JFrame использовать его полное имя с указанием всех пакетов.

Возможно вы уже догадались, но тем не менее давайте укажем это явно — если в коде класса А вам потребовалась ссылка на класс В, то директива `import` для класса В потребуется тогда, когда класс В находится в другом пакете, нежели класс А. Если же оба класса находятся в одном пакете, то директива `import` не потребуется.

Вы можете задаться вопросом: «А зачем может потребоваться полное имя в коде?». Такое возможно в случае когда в одном классе надо обращаться к двум классам с одинаковыми короткими именами, но разными полными именами. Например, существует два очень используемых класса `java.util.Date` и `java.sql.Date`. и бывают случаи, когда оба класса нужны внутри одного класса. Тогда надо применить именно такой прием — один класс может быть определен через `import`, а второй — по полному пути во всем коде.

На этом мы пока закончим рассмотрение пакетов, хотя возможно будем их вспоминать в связи с новыми вопросами. Пока же запомните одно хорошее правило — не создавайте классы вне пакетов.

И теперь нас ждет следующая статья: [Переопределение и перегрузка](#)

11 comments to *Пакеты*



Апрель 1, 2015 at 22:03

куц says:

Спасибо за интересный учебник.!!!Браво!!

[Reply](#)



Август 9, 2015 at 10:08

Стас says:

Спасибо за замечательный учебник.

У меня возник вопрос. В строке `sf.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);`

вместо `JFrame` IDE вбивает `WindowConstants`. Что лучше употреблять если пишешь кроссплатформенное приложение.

[Reply](#)



o

Август 9, 2015 at 17:33

admin says:

Там достаточно сложная история — `WindowConstants.EXIT_ON_CLOSE` и `JFrame.EXIT_ON_CLOSE` имеют одну и ту же величину. так что выбирать самому. В принципе `JFrame` имплементирует интерфейс `WindowConstants` и посему тут «масло масляное» — но уж так исторически сложилось. Видимо IDE находит такую же константу в интерфейсе-предке и «пытается быть умной».

[Reply](#)



•

Сентябрь 7, 2016 at 11:56

Роман says:

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

А почему без этой команды окно всё равно закрывается?

[Reply](#)



o

Сентябрь 7, 2016 at 13:03

admin says:

Эта команда позволяет при закрытии окна завершить приложение. В противном случае окно закрывается, но приложение продолжает работать.

[Reply](#)



•

Ноябрь 6, 2016 at 22:19

Ploskiy says:

Простите, у меня возник вопрос.(Пишу в среде IDEA)

Вроде как всё понятно, но по логике если мы создаем какой либо проект(небольшую программку) для того, что бы её скомпилировать ей нужна точка отсчета(аналогия Index.php или RunMe.bat)? Разве всё в дереве иерархии не должно выглядеть так ?

Main.java

package1\Robot.java

package2\Navi.java

Параллельно читаю несколько учебников, но хочу признать, что у вас получилось очень здорово без лишних заморочек.

Небольшая просьба, если это представляется возможным, сделать управление величиной шрифта, для более комфортного чтения.

[Reply.](#)



o

Ноябрь 8, 2016 at 08:53

admin says:

не должна. Любой класс ожет запускаться, если у него есть метод main.

[Reply.](#)



•

Февраль 23, 2017 at 21:29

Юрий says:

А где можно посмотреть, что можно импортировать?

Нашел вот здесь целую гору каких-то неизвестных плагинов :

C:\Program Files (x86)\Java\jdk1.7.0_80\lib\missioncontrol\plugins

Как узнать для чего все это хозяйство?

[Reply.](#)



o

Февраль 24, 2017 at 00:02

admin says:

Не совсем корректный вопрос — сначала надо понять, какие именно классы нужны и потом уже подключать нужные JAR-файлы. Об этом узнают из документации, на форумах, из статей.
По поводу указанной директории — не использовал, не могу сказать.

[Reply](#)



Ноябрь 7, 2017 at 14:28

Zina says:

Помогите найти ошибку.
Хотела запустить програмку и посмотреть что за окно в результате получится.
Скачала проект, скомпилировала через командную строку, а вот запустить не получилось:

```
D:\lessons\SimpleFrame\SimpleFrame> javac src\edu\javacourse\*.java
```

```
D:\lessons\SimpleFrame\SimpleFrame> java src.edu.java
course.SimpleFrame
Error: Could not find or load main class src.edu.javacourse.SimpleFrame
Caused by: java.lang.NoClassDefFoundError: edu/javacourse/SimpleFrame (wrong name: src/edu/javacourse/SimpleFrame)
```

[Reply](#)



Ноябрь 7, 2017 at 23:10

admin says:

Пакет не имеет src — надо зайти внутрь этого каталога. Почитайте внимательно про пакеты и поймите, что это такое.

[Reply](#)

Leave a reply

Comment

You may use these HTML tags and attributes: <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <i> <q cite=""> <s> <strike> <pre class="" title="" data-url="">

Имя *

E-mail *

Сайт

2 - 1 = 

Add comment

Copyright © 2018 [Java Course](#)

Designed by [Blog templates](#), thanks to: [Free WordPress themes for photographers](#), [LizardThemes.com](#) and [Free WordPress real estate themes](#)

