JAR-файлы — что это такое ?

JAR расшифровывается как Java ARchive — архив Java. Если не особо вдаваться в подробности (что в общем-то часто и происходит в реальности), то JAR-файл представляет собой обычный ZIP-файл с некоторыми дополнениями. Основная задача этого архива — хранить файлы с классами. Т.е. пакеты/каталоги (мы о них говорили в разделе Пакеты), внутри которых находятся class-файлы архивируются и JVM может их использовать уже в более удобном (компактном) виде.
Выгода очевидна — в реальных проектах вы вряд ли будете ограничиваться использованием библиотек классов только из JDK — скорее всего вы будете подключать сотни (если не тысячи) классов, которые были созданы для решения определенных задач. Если все это огромное количество лежало бы на диске в виде class-файлов, то несложно понять, насколько это неудобно и громоздко. Разработчики Java предложили вполне элегантное решение — вы можете подключать архивные файлы, в которых запакованы тысячи скомпилированных классов. Что конечно же удобнее, чем таскать за собой в каждый проект все директории со всеми файлами.
Для примера можете заглянуть в каталог JAVA_HOME/jre/lib — там находится очень важный архив — **rt.jar**. Эта библиотека содержит практически все классы, который мы рассматривали — String, JFrame, JButton, Date, Object. Открыть его вы можете с помощью любого архиватора.
Просматривая содержимое JAR-файла, вы рано или поздно обратите внимание на каталог **META-INF**. В нем содержится файл **MANIFEST.MF**. Я настоятельно рекомендую вам почитать информацию о нем в документе Working with Manifest Files: The Basics. Файл позволяет расширить функциональность — кроме обычного набора классов, JAR-файл может выполнять функции электронной подписи, поддержки

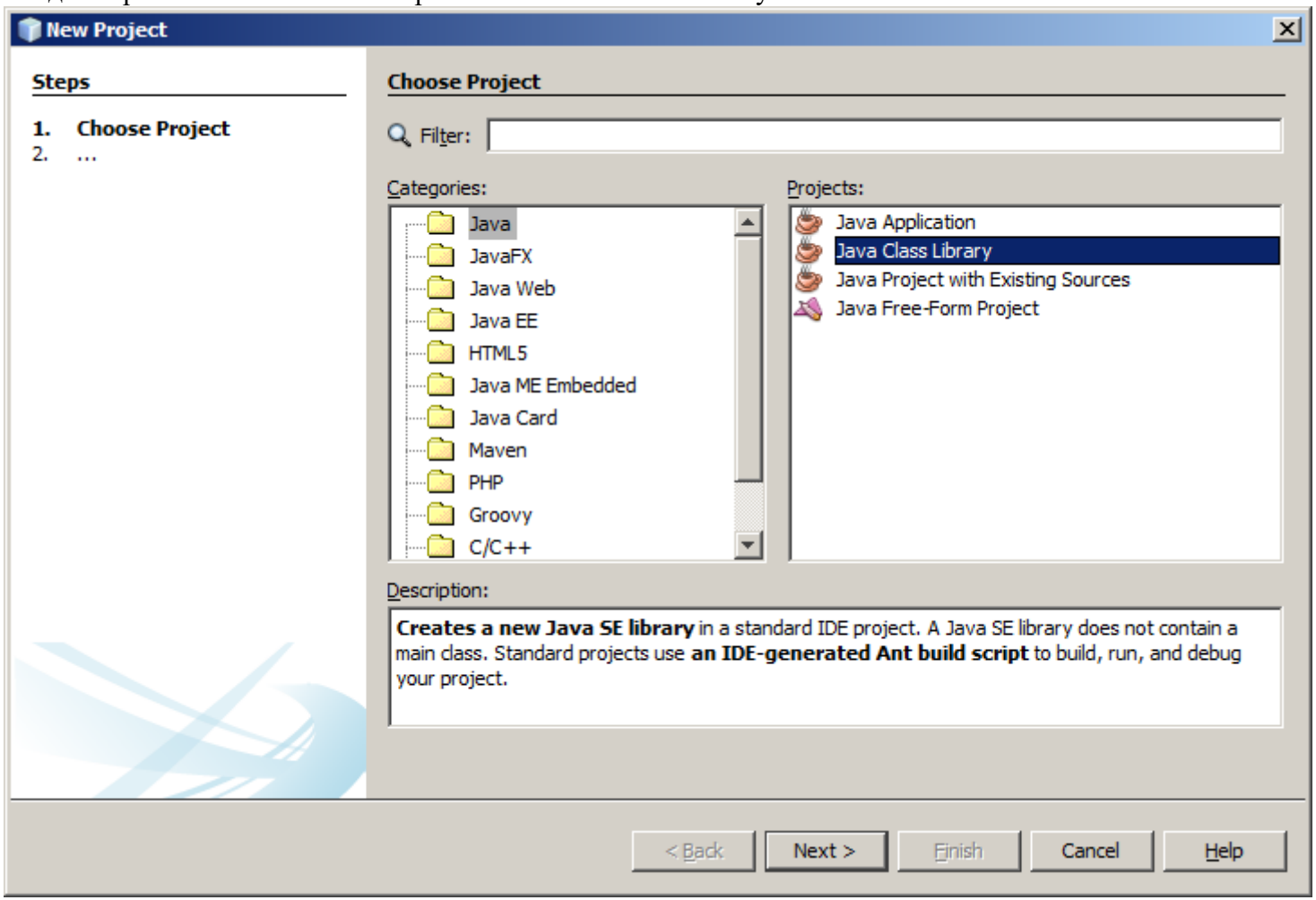
версионности, подключения библиотек к самому архиву, определение класса для запуска (который содержит метод **main**). В данной статье я не ставлю себе цель разборать эти возможности, так что отложим на будущее. Займемся базовыми вопросами использования JAR-файлов.

Создание JAR

Использование IDE

Практически все IDE умеют создавать JAR. Мы сделаем это на примере NetBeans, другие IDE вы можете рассмотреть самостоятельно. Что является не всегда интуитивно понятной задачей. В среде NetBeans создать JAR очень несложно. Для этого надо сделать следующие шаги:

1. Создать проект в NetBeans. Тип проекта — «New Class Library»



2. В этом проекте вы создаете нужные вам классы и потом выполняете команду «Build». Это либо кнопка F11, либо пункт меню «Run — Build Project», либо кликаете правой кнопкой на проекте в панели Project и выбираете пункт «Build» (или «Clean and Build»).

В каталоге с проектом можно увидеть каталог **dist** в котором и будет находиться JAR-файл. Как я уже упоминал, в других типах IDE этот процесс может выглядеть иначе. Например, в Eclipse это делается через экспорт, а в IDEA — через создание артефактов. В данной цикле статей я не планирую пока останавливаться на этом подробно, хотя если получится — значит прямо здесь и добавлю.

Использование утилиты jar

Также JAR-файл может быть создан с использованием утилиты jar, которая находится в каталоге JAVA_HOME/bin. Достаточно подробно эта утилита разобрана здесь: Creating a JAR File.
В самом просто варианте вам надо создать классы в какой-либо директории, а потом вызвать эту утилиту с такой командой

```
jar cf <имя JAR-файла> <список входных файлов>
```

Например, если у меня есть класс **edu/javacourse/jar/SayHello.class**, то команда создания архивного файла с именем **say.jar** выглядела бы так:

```
jar cf say.jar edu/javacourse/jar/SayHello.class
```

Опции «cf» означают: «c» — создать, «f» — файл. Ну а дальше я думаю очевидно — имя файла архива и список файлов. Причем в списке можно указывать много файлов через пробел, а также можно использовать маску — знаки «*» и «?». Если вы не знакомы с такими знаками — наберите в поисковике «маски файлов использование» и наслаждайтесь новыми знаниями.
Чтобы посмотреть содержимое файла вы можете (как я уже говорил выше) использовать практически любой архиватор или тут же команду **jar.exe** вот в таком виде

```
jar tf say.jar
```

В обоих случаях (IDE или утилита **jar**) в архиве будет создаваться файл **MANIFEST.MF**. Как я уже писал — есть смысл почитать о дополнительных свойствах JAR-файлов. Т.к. в документации все очень неплохо написано, не буду заниматься переводом. Так что перейдем к следующему пункту нашего путешествия по JAR-файлам.

Подключение готовых JAR-файлов

Командная строка

Итак, я сказал, что основная функция JAR — содержать набор классов в виде class-файлов. Теперь нам надо понять, как подключать JAR-файлы к запускаемому классу, чтобы запускаемый класс мог использовать классы из JAR-файла.
Как вы уже хорошо знаете, запуск программы на Java включает запуск JVM (для Windows это файл java.exe, для Unix — java) и передача ей из командной строки имени класса, который вы собираетесь запускать. Например, для запуска класса **First** в пакете **edu.javacourse.test** надо в каталоге, внутри которого находится каталог **edu/javacourse/test** набрать такую команду:

```
java edu.javacourse.test.First
```

Как видите, в качестве аргумента мы передали полное имя класса, который собираемся запускать. JVM кроме имени класса принимает достаточно много параметров, которые позволяют конфигурировать те или иные свойства JVM. В этой статье мы разберем очень важный элемент — **CLASSPATH** — который как раз и используется для подключения JAR-файлов.
Сейчас мы проделаем полный цикл создания JAR-файла и его подключения без использования IDE — я очень трепетно отношусь к умению программиста пользоваться командной строкой. Во-первых, это показывает, что вы действительно понимаете суть, во-вторых — этот навык ну просто очень полезен. Например, в том же Unix/Linux очень большое количество работы гораздо удобнее делать в командной строке, да и некоторые продукты содержат утилиты, запускаемые из командной строки. Так что займемся.
Создадим каталог **JavaLesson** (в принципе мы его уже создавали в разделе Основные шаги. В этом каталоге создадим стуктуру каталогов **JarLib/edu/javacourse/jar**. С помощью текстового редактора создадим файл **SayHello.java**

```
package edu.javacourse.jar;

public class SayHello
{
    public void sayHello() {
        System.out.println("HELLO");
    }
}
```

Структура наших каталогов должна выглядеть вот так:

```
C\:
  JavaLesson
    JarLib
      edu
        javacourse
          jar
            SayHello.java
```

Теперь время за командной строкой. Запускаем команду «cmd» (если не помните — смотрим раздел Основные шаги). Переходим в каталог **JavaLesson/JarLib** и в нем сначала компилируем наш файл **SayHello.java** вот такой командой

```
javac edu/javacourse/jar/SayHello.java
```

После успешной компиляции создаем JAR-файл с помощью команды

```
jar cf say.jar edu/javacourse/jar/SayHello.class
```

Если у вас все получилось, то в каталоге **JavaLesson/JarLib** должен появится файл **say.jar**.
Теперь напишем небольшой класс, который будет использовать наш класс **SayHello**. **ВАЖНО !!!** Для «чистоты эксперимента» создадим этот класс в каталоге **JavaLesson**. Предлагаю создать класс без использования пакетов (хотя это не должно быть правилом — в реальных проектах создавать классы без пакетов не надо). Вот наш файл **UseHello.java**, который будет находится в каталоге **JavaLesson**

```
import edu.javacourse.jar.SayHello;

public class UseHello
{
    public static void main(String[] args) {
        SayHello sh = new SayHello();
        sh.sayHello();
    }
}
```

Обратите внимание на первую строчку — в ней мы импортируем класс **SayHello**. В методе **main** мы создаем объект и вызываем его метод. Структура наших каталогов должна выглядеть вот так:

```
C\:
  JavaLesson
    JarLib
      edu
        javacourse
          jar
            SayHello.java
    UseHello.java
```

Теперь нам надо скомпилировать наш замечательный класс. Пробуем команду **javac**. ВНИМАНИЕ !!! Мы находимся в каталоге **JavaLesson**.

```
javac UseHello.java
```

И мы получаем ошибку:

```
C:\JavaLesson>javac UseHello.java
UseHello.java:1: error: package edu.javacourse.jar does not exist
import edu.javacourse.jar.SayHello;
                         ^
UseHello.java:6: error: cannot find symbol
        SayHello sh = new SayHello();
        ^
  symbol:   class SayHello
  location: class UseHello
UseHello.java:6: error: cannot find symbol
        SayHello sh = new SayHello();
                          ^
  symbol:   class SayHello
  location: class UseHello
3 errors
```

Сообщение в данном случае достаточно информативное — «UseHello.java:1: error: package edu.javacourse.jar does not exist» и «UseHello.java:6: error: cannot find symbol». Мы же не подключали наш класс **SayHello** — вот компилятор и не знает, откуда его брать и вообще откуда такой пакет/класс — **edu.javacourse.jar.SayHello**. Ну что же — перефразируя классику — «Ваше слово, товарищ CLASSPATH».
Очевидно, что нам надо компилятору подсказать, что он должен использовать не только те JAR-файлы, которые у него уже есть — тот же **rt.jar** — но и наш JAR-файл. Делается это с помощью специальной опции при запуске компилятора.

```
javac -cp JarLib/say.jar UseHello.java
```

В этой команде мы добавили строку **-cp JarLib/say.jar**. Часть **-cp** говорит, что за ней должен идти список JAR-файлов, которые содержат нужные классы. Список может включать несколько файлов, разделенных для Windows знаком «;», для Unix/Linux — «:». Например, если мне надо указать два файла — say1.jar и say2.jar — находящихся в каталоге **C:/Anton/Libraries**, то команда (для Windows) выглядела бы так:

```
javac -cp C:/Anton/Libraries/say1.jar;C:/Anton/Libraries/say2.jar UseHello.java
```

Кстати, «-cp» — сокращенно от **classpath** — раньше надо было писать именно так, да и сейчас тоже можно.

```
javac -classpath C:/Anton/Libraries/say1.jar;C:/Anton/Libraries/say2.jar UseHello.java
```

Также важно отметить, что в CLASSPATH можно включать не только JAR-файлы — туда можно включать путь до каталога с файлами .class (опять же — их может быть указано несколько штук). Если бы мы не создавали JAR-файл **say.jar**, а использовали скомпилированный файл **SayHello.class** из каталога **JarLib**, то команда выглядела бы вот так:

```
1 java -cp JarLib UseHello.java
```

Вы можете спросить — почему мы указали путь только до **JarLib**, а не полностью каталог с файлом **SayHello.class**. Дело в том, что т.к. наш файл содержится в пакете, то нам надо указать путь именно до пакета, а он находится в каталоге **JarLib**. Можно указать полный (а не относительный) путь — вот так:

```
1 javac -cp C:/JavaLesson/JarLib UseHello.java
```

Ну что же, скомпилировать нам удалось, самое время запускать наше замечательный класс. Скорее всего, вы уже догадываетесь, что при запуске нам тоже надо указать наш файл **say.jar** и выглядеть команда должна как-то так. ВНИМАНИЕ !!! Мы находимся в каталоге **JavaLesson**.

```
1 java -cp JarLib/say.jar UseHello
```

Но при запуске этой программы нас постигает неудача. Вот такое сообщение выдает JVM:

```
1 Error: Could not find or load main class UseHello
```

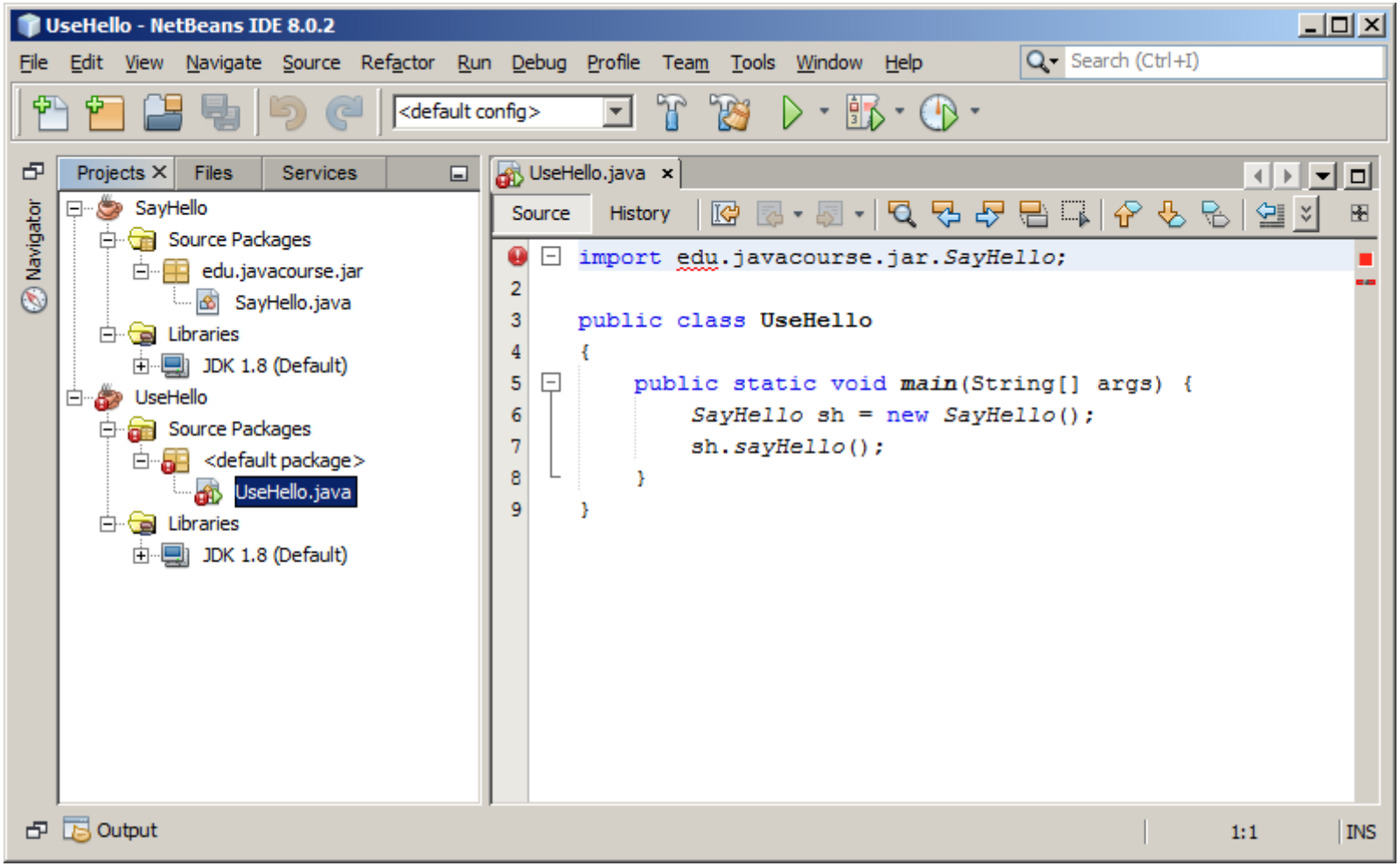
JVM не может найти наш класс. Как это, вот же он, прямо в этой директории ? Хитрость в том, что когда вы указываете CLASSPATH в командной строке, то JVM берет классы ТОЛЬКО из этих файлов/каталогов. А наш файл с классом **UseHello** находится в ТЕКУЩЕМ каталоге, который НЕ УКАЗАН. Вот такая вот эпидерсия. Нам надо указать JVM, чтобы она брала файлы и из текущего каталога тоже. Делается это так — в CLASSPATH надо указать символ «.». Это и будет текущий каталог. Наща команда должна выглядеть вот так:

```
1 java -cp .;JarLib/say.jar UseHello
```

Ну что же — теперь все должно получится и мы увидим надпись HELLO.
Фокус с CLASSPATH может сыграть с вами достаточно злую шутку. Дело в том, что JVM использует переменную среды, которая так и называется «CLASSPATH». Для Windows она устанавливается точно так же как переменные «PATH» и «JAVA_HOME» — мы делали это в разделе Установка JDK. В этой переменной указывается набор JAR-файлов и каталогов, которые JVM будет использовать в случае, если вы НЕ указываете CLASSPATH при запуске своей программы.
Некоторые программы при своей установке создают (или обновляют) переменную среды «CLASSPATH» и могут даже вам ничего не сказать. И вот вы из командной строки запускаете свой класс, а JVM его НЕ ВИДИТ. Потому? что она dpzkf переменную «CLASSPATH» и в ней нет символа «.». В этом случае происходит то, что мы уже видели — файлы из текущего каталога не загружаются. Так что будьте внимательны.

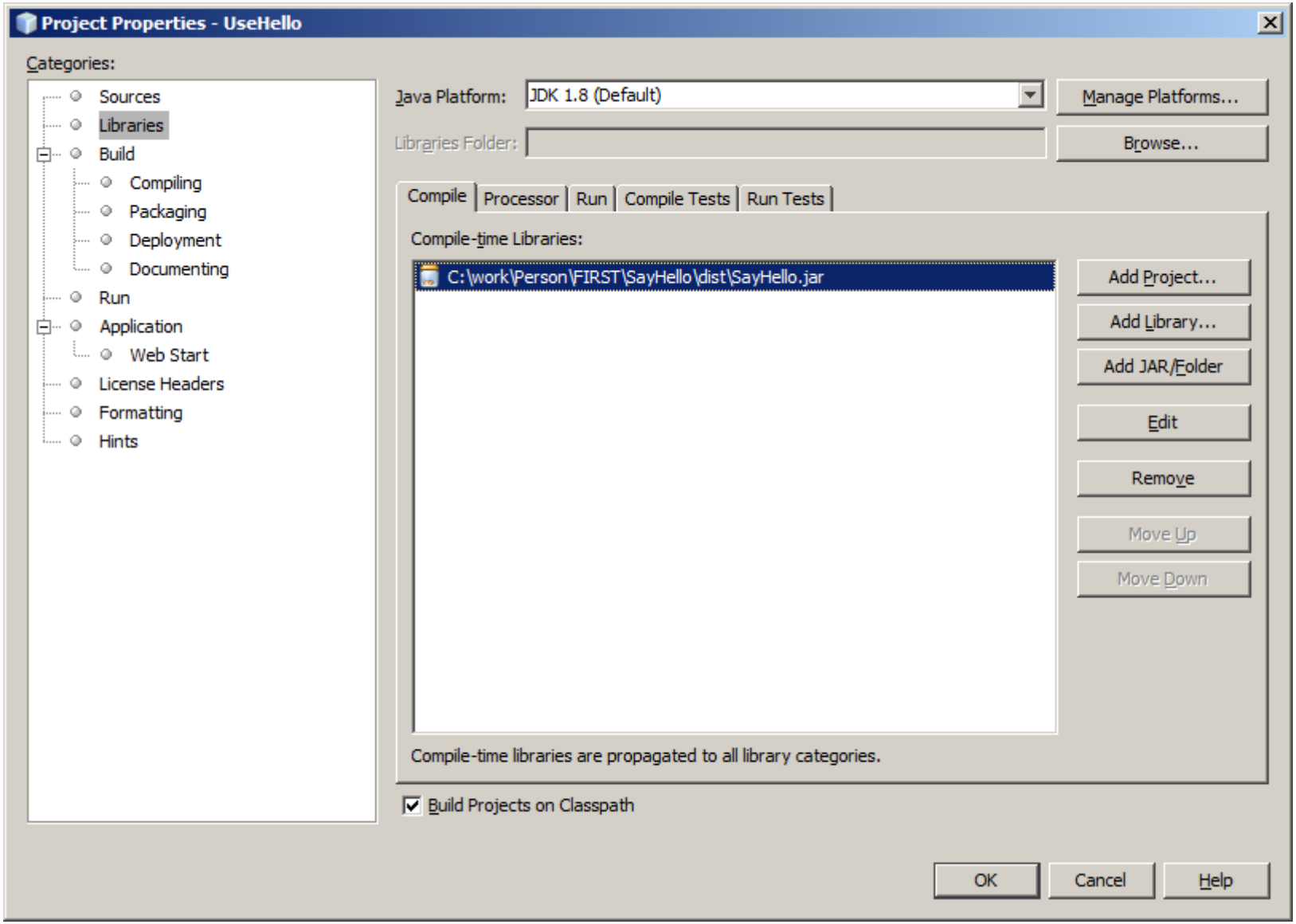
Подключение JAR-библиотек в IDE NetBeans

Т.к. в данном курсе я использую для демонстрации NetBeans, то наверно будет неправильно не показать, как подлкючать JAR в этой среде. Как я уже упоминал, если появится время на другие IDE, я буду писать их в этой же статье. Но не думаю, что это будет в ближащее время.
Если вы посмотрите служебное окно «Projects», которое обычно находится слева, то вы в нем вы можете увидеть структуру вашего проекта, которая содержит раздел «Libraries». Я создал два проекта **SayHello** и **UseHello**



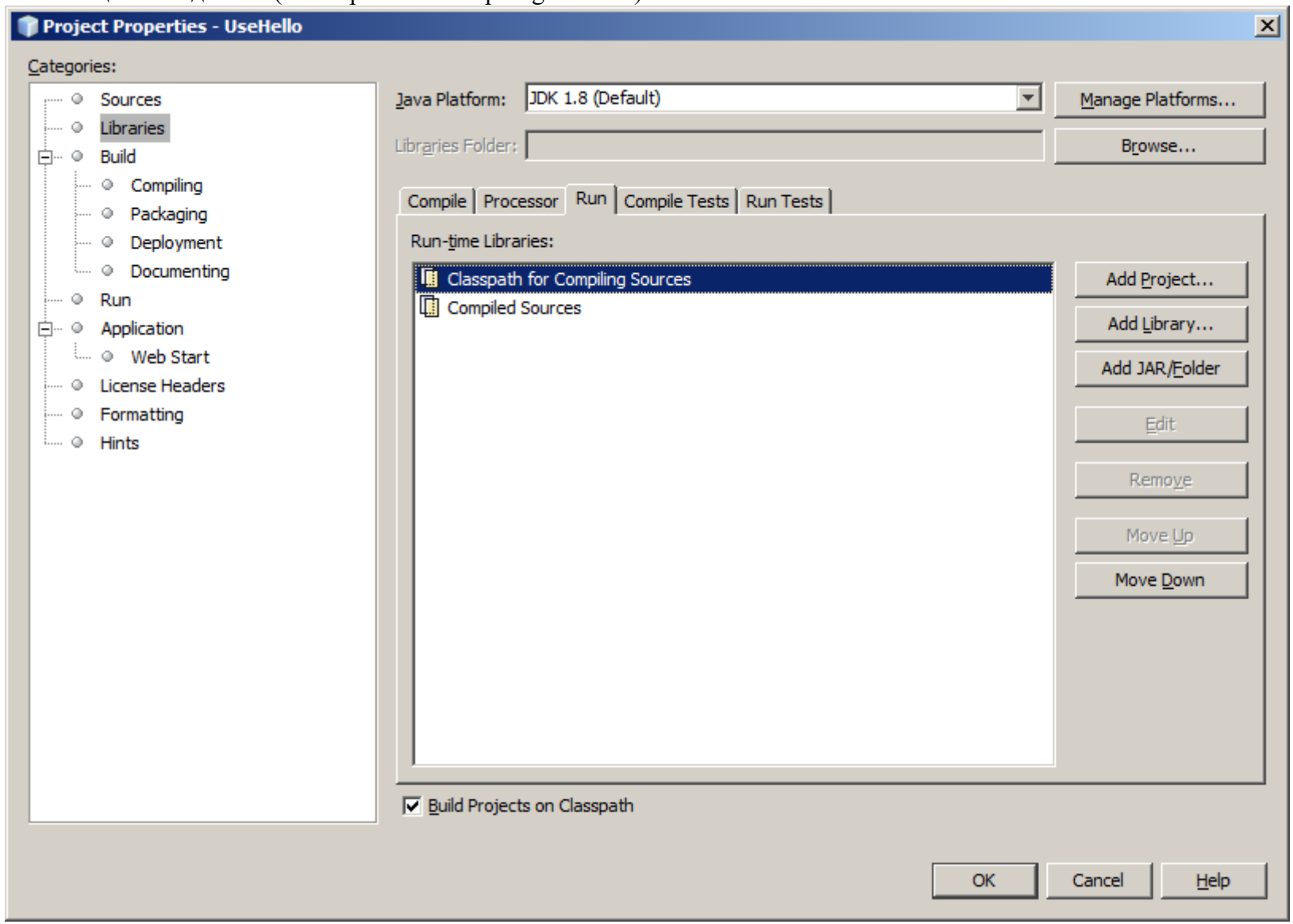
В открытом файле **UseHello.java** в самой первой строке видно, что компилятор выдает в ней ошибку — как мы уже знаем, она говорит об отсутствии нужного класса. В принципе IDE позволяют подключать не только готовые JAR-файлы, но и проекты, но мы не будем использовать эту функцию (в данном случае специально). Чтобы подключить библиотеку из проекта **SayHello** нам для начала надо собрать этот проект через команду «Build». Результат можем увидеть в каталоге **dist**. В нем мы можем увидеть файл **SayHello.jar**. Предвидя вопрос — а можно поменять имя файла — для того, чтобы файл создавался с другим именем, надо исправлять конфигурационный файл **nbproject/project.properties**. Найти в нем опцию с именем **dist.jar** И поменять имя файла. В общем не очень удобно. Так что я обычно этого не делаю.
Теперь нам надо подключить готовый JAR-файл к нашему второму проекту **UseHello**. Сделать это можно несколькими способами.

1. Щелкнуть правой кнопкой мышки на пункте «Libraries» в структуре проекта и в нем выбрать пункт «Add JAR/Folder». После выбора файла его можно будет видеть под веткой «Libraries»
2. Целкнуть правой кнопкой мышки на проекте **UseHello** в окне «Projects» и в выпадающем меню выбрать пункт (обычно самый нижний) «Properties». В открывшемся диалоговом окне выбрать слева пункт «Libraries» и в закладке «Compile» использовать кнопку «Add JAR/Folder».



Также предлагаю заглянуть на закладку «Run». В нем можно увидеть, что при запуске будет подключаться те библиотеки, которые подключаются при

компиляции исходников («Classpath for Compiling Source»).



На этом можно закруглиться, хотя конечно же процесс познания бесконечен и вам наверняка встретится еще много интересной информации по использованию и созданию jar-файлов.

Исходный код для проектов под NetBeans можно скачать здесь. Библиотеку в них надо подключить самостоятельно. Так что можете потренироваться. Удачи.

И теперь нас ждет следующая статья: Многопоточность — первые шаги.

2 comments to *Что такое JAR-файлы*

- 
  Июнь 24, 2018 at 15:35
  Cmac says:

  Очень познавательная статья, спасибо! До неё не понимал, для чего вообще нужна cp и jar-файлы. P.s. у вас описка на этой строчке. «java -cp JarLib UseHello.java» д.б. javac

  Reply

- 
  Июнь 24, 2018 at 15:41
  Stas says:

  Прошу прощения, все верно

  Reply

Leave a reply

Comment

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

Имя *

E-mail *

Сайт

два + = 5

Add comment