

#1. Инфраструктура Java, сборка и запуск

[Jump to bottom](#)

Sidikov Marsel edited this page on Mar 10, 2018 · 11 revisions

Инфраструктура Java и набор необходимых компонентов

Работа с командной строкой (терминалом)

- `cd имя_папки` - переход в необходимую папку (указывается полный, либо относительный путь)
- `ls` либо `dir` - показывает содержимое папки, в которой "находится" терминал
- `mkdir имя_папки` - создает папку
- `rm имя_файла` - удаляет заданный файл
- `cd ../` - "переводит" терминал на один уровень вверх

Установка компонентов Java

- JDK (Java Development Kit) - набор для разработки на Java, включает необходимые утилиты для компиляции и сборки java-программ (например, `javac`).
- JRE (Java Runtime Environment) - пакет, содержащий необходимые компоненты для исполнения java-программ. Входит в состав JDK, но может распространяться и отдельно.
- JVM (Java Virtual Machine) - программа для интерпретации java-программ (class-файлов) в машинный код. Входит в состав JRE.

Установить JDK для вашей архитектуры можно [тут](#)

Maven

Скачиваем по [ссылке](#), выбираем `apache-maven-3.5.2-bin.zip`, и распаковываем архив в папке Program Files.

Переменные окружения

Для полноценного функционирования java-приложений необходимо указать системе пути к каждому из установленных компонентов.

В каждой ОС переменные окружения настраиваются по-разному.

Системная переменная Path

- Системная переменная Path содержит информацию о программах, к которым можно получить доступ во время работы в командной строке.

Настройка переменной Path

В данной переменной следует указать путь к программам Java и Maven (к папкам bin). Путь к папкам bin следует указать через точку с запятой в конце значения переменной, например так:

```
%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;%SYSTEMROOT%\System32\WindowsPc  
es\Java\jdk1.8.0_102\bin;C:\Program Files\Maven\apache-maven-3.5.2\bin;
```

Системные переменные Maven и Java

Необходимо добавить/проверить наличие следующих системных переменных:

- JRE_HOME - C:\Program Files\Java\jre1.8.0_131
- JAVA_HOME - C:\Program Files\Java\jdk1.8.0_102

Проверка работоспособности установленных компонентов

В командной строке необходимо набрать следующие команды

- java -version

Ответ на команду:

```
java version "1.8.0_131"  
Java(TM) SE Runtime Environment (build 1.8.0_131-b11)  
Java HotSpot(TM) 64-Bit Server VM (build 25.131-b11, mixed mode)
```

- `javac -version`

Ответ на команду

```
javac 1.8.0_102
```

- `mvn -version`

Ответ на команду:

```
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5; 2015-11-10T19:41:47+03:00)
Maven home: C:\Program Files\JetBrains\IntelliJ IDEA 2017.2.2\plugins\maven\lib\maven3
Java version: 1.8.0_102, vendor: Oracle Corporation
Java home: C:\Program Files\Java\jdk1.8.0_102\jre
Default locale: ru_RU, platform encoding: Cp1251
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

Если maven не работает, пробуем посмотреть [тут](#).

Компиляция, интерпретация и сборка

javac

- `javac` (java-компилятор) - программа, выполняющая преобразование исходного кода на Java (файлов с расширением `.java`) в код виртуальной машины (байт-код, файлы с расширением `.class`).

Пример использования в командной строке:

```
javac -d target Program.java
```

Вызов программы `javac` с компиляцией файла `Program.java`. Значение, указанное после параметра `-d` говорит компилятору, куда следует поместить скомпилированный `class`-файл.

java

- `java` - программа, выполняющая запуск JVM и скомпилированного байт-кода.

Пример использования в командной строке:

```
java Program
```

Запускает на исполнение программу, чей байт-код находится в файле Program.class. Виртуальная машина выполняет интерпретацию байт-кода в машинный код.

jar

- jar - программа, выполняющая сборку скомпилированных .class-файлов (и не только) в один архив, который может использоваться как библиотека с определенным функционалом, так и полноценная запускаемая программа.

Пусть проект имеет следующую структуру

- src
 - ru
 - ivmmit
 - Box.java
 - CostCalculator.java
- target

Для того, чтобы скомпилировать файлы в папку target перейдем в папку target (предполагается, что терминал запущен в корне проекта):

```
cd target
```

Поскольку класс Box CostCalculator будет использоваться в CostCalculator, скомпилируем в первую очередь класс Box:

```
javac ../src/ru/ivmmit/Box.java -d .
```

Символ . говорит компилятору поместить скомпилированный класс в текущую (target) директорию. Далее скомпилируем класс CostCalculator (аналогично).

Теперь выполним сборку полученных классов в один архив:

```
jar cvf cost-calc.jar .
```

Содержимое созданного архива можно просмотреть, используя любой из известных архиваторов.

Для того, чтобы использовать классы полученного jar-архива в других программах, необходимо как при компиляции, так и при запуске указывать параметр `-classpath` (путь к class-файлам) для компилятора `javac` и интерпретатора `java`:

```
javac -classpath ".;cost-calc.jar" Program.java
```

В параметре `classpath` указан путь к jar-файлу, а также путь к текущей директории (символ - точка). Запуск `Program.class` файла:

```
java -classpath ".;cost-calc.jar" Program
```

Сборка с помощью Maven

[Подробнее о Maven](#)

- Maven позволяет автоматизировать процесс сборки проектов и скачивания зависимостей (сторонних jar-библиотек) и помещения их в папку `.m2`, которая находится в корне пользовательской папки (`C:/Users/имя_пользователя`)

Maven требует определенной структуры создаваемого проекта:

- `src` - файлы исходного кода
 - `main` - файлы основного кода проекта
 - `java` - исходные файлы на java
 - `resource` - дополнительные файлы проекта (изображения, звуки, файлы настроек и т.д.)
 - `test` - файлы тест-кода проекта
- `target` - скомпилированные файлы и собранные jar-архивы
- `pom.xml` - файл конфигурации maven-проекта

`pom.xml` содержит описание характеристик проекта (`groupId` - идентификатор организации, `artifactId` - идентификатор проекта, `version` - версия), а также параметры сборки проекта.

В самом начале `pom.xml` указывается схема. [Подробнее об xml-схемах](#). Далее, в блоке `dependencies` можно перечислить сторонние зависимости. Внутри тега `build` указываются параметры сборки.

- Компиляция maven-проекта

```
mvn compile
```

Данная команда, выполненная в корне проекта скомпилирует все java-файлы и поместит class-файлы в папку target.

- Сборка проекта в jar-архив:

```
mvn package
```

Результирующий jar-архив будет размещен в папке **target**

- Очистка папки target

```
mvn clean
```

В случае, когда был собран "запускаемый" jar-файл (см. пример ColoredTerminal), его можно запустить с использованием команды:

```
java -jar colored-0.1-jar-with-dependencies.jar
```

Следует обратить внимание, что в MANIFEST.MF файле "запускаемых" jar-архивов указан класс, который имеет точку входа в приложение (Main-метод).

▼ Pages 9

[Home](#)[# Разбор залитых приложений](#)[#0. Темы для самостоятельного изучения](#)[#1. Инфраструктура Java, сборка и запуск](#)[#2. Сервлеты](#)[#3. JSP, куки и сессии](#)[#4. Базы данных](#)[#5. Базы данных в Java](#)[Домашние задания](#)

Clone this wiki locally

`https://github.com/MarselSidikov/JAVA_FIX.wiki.git`

