

Maven

Системы сборки проектов

Мавен - это инструмент для сборки Java проекта: компиляции, создания jar, создания дистрибутива программы, генерации документации. Простые проекты можно собрать в командной строке. Если собирать большие проекты с командной строки, то команда для сборки будет очень длинной, поэтому её иногда записывают в bat/sh скрипт. Но такие скрипты зависят от платформы. Для того чтобы избавиться от этой зависимости и упростить написание скрипта используют инструменты для сборки проекта.

Для платформы Java существуют два основных инструмента для сборки: [Ant](#) и [Maven](#).

Основные преимущества Maven

- **Независимость от OS.** Сборка проекта происходит в любой операционной системе. Файл проекта один и тот же.
- **Управление зависимостями.** Редко какие проекты пишутся без использования сторонних библиотек(зависимостей). Эти сторонние библиотеки зачастую тоже в свою очередь используют библиотеки разных версий. Мавен позволяет управлять такими сложными зависимостями. Что позволяет разрешать конфликты версий и в случае необходимости легко переходить на новые версии библиотек.
- **Возможна сборка из командной строки.** Такое часто необходимо для автоматической сборки проекта на сервере ([Continuous Integration](#)).
- **Хорошая интеграция со средами разработки.** Основные среды разработки на java легко открывают проекты которые собираются с помощью maven. При этом зачастую проект настраивать не нужно - он сразу готов к дальнейшей разработке.
Как следствие - если с проектом работают в разных средах разработки, то maven удобный способ хранения настроек. Настроечный файл среды разработки и для сборки один и тот же - меньше дублирования данных и соответственно ошибок.

Установка

- Зайдите на официальный сайт мавен в раздел [загрузка](#) и скачайте последнюю стабильную версию bin.zip
- Распакуйте архив в инсталляционную директорию. Например в C:\Program Files\maven\ в Windows или /opt/maven в Linux
- Установите переменную окружения **M2_HOME**:

В **Windows** кликните правой кнопкой мыши на "мой компьютер" ->свойства->дополнительные параметры->переменные среды->системные переменные и там добавьте "M2_HOME" и "C:\Program Files\maven\" .

В **Linux** можно добавить строку "export M2_HOME=/opt/maven"в файл /etc/profile .

- Установите переменную окружения PATH В **Windows** в переменной PATH добавьте к списку директорий строку %M2_HOME%\bin". В **Linux** можно добавить строку "export PATH=\$PATH:\$M2_HOME/bin"в файл /etc/profile .
- Проверьте корректность установки, набрав в командной строке

mvn -version

```
Командная строка
Microsoft Windows [Version 10.0.19044.2486]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\anast>mvn -version
"mvn" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.

C:\Users\anast>_
```

```
Командная строка
Microsoft Windows [Version 10.0.19044.2486]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\Users\anast>mvn -v
Apache Maven 3.9.0 (9b58d2bad23a66be161c4664ef21ce219c2c8584)
Maven home: C:\Program Files\maven
Java version: 17.0.6, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk-17
Default locale: ru_RU, platform encoding: Cp1251
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\anast>
```

Если результат примерно такой

```
dima@myhost ~ $ mvn -version
Apache Maven 3.0 (r1004208; 2010-10-04 15:50:56+0400)
Java version: 1.6.0_22
Java home: /opt/sun-jdk-1.6.0.22/jre
Default locale: ru_RU, platform encoding: UTF-8
OS name: "linux" version: "2.6.34-gentoo-r12" arch: "amd64" Family: "unix"
```

то поздравляю, вы успешно установили Maven.

*Во многих дистрибутивах Linux, maven устанавливается автоматически, с помощью менеджера пакетов.

Если что-то не работает

- Проверьте, установлен ли у вас JDK.

Для этого наберите в консоли «java -version» ответ должен быть примерно таким:

```
java version "1.6.0_22"  
OpenJDK Runtime Environment (IcedTea6 1.10.5) (ArchLinux-6.b22_1.10.5-1-x86_64)  
OpenJDK 64-Bit Server VM (build 19.0-b09, mixed mode)
```

- Проверьте установлена ли переменная окружения JAVA_HOME
Если у вас Windows наберите в консоли:

```
echo %JAVA_HOME%
```

Если у вас Linux наберите в консоли:

```
echo $JAVA_HOME
```

команда должна вывести путь к JDK.

Простой пример

Давайте создадим новый проект. Для этого выполним команду:

```
mvn archetype:generate
```

2. Далее мы увидим большой список доступных вариантов шаблонов создания проекта, наподобие представленного ниже.

```
1 594: remote -> org.zkoss:zk-archetype-component (The ZK Component archetype)
2 595: remote -> org.zkoss:zk-archetype-webapp (The ZK wepapp archetype)
3 596: remote -> ru.circumflex:circumflex-archetype (-)
4 597: remote -> se.vgregion.javg.maven.archetypes:javg-minimal-archetype (-)
5 598: remote -> sk.seges.sesam:sesam-annotation-archetype (-)
6
7 Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains
8
```

Maven предложит ввести номер желаемого варианта создания проекта, по умолчанию предлагается создать проект по образцу **maven-archetype-quickstart**, в приведенном примере он под номером 193. Таким образом если мы ничего не вводим и просто нажимаем ввод, то запускается визард создания стандартного *Maven* проекта.

3. Далее визард предлагает выбрать версию образца создания проекта, по умолчанию предлагается последняя стабильная версия.

```
1 Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
2 1: 1.0-alpha-1
3 2: 1.0-alpha-2
4 3: 1.0-alpha-3
5 4: 1.0-alpha-4
6 5: 1.0
7 6: 1.1
8 Choose a number: 6:
```

4. Далее визард предлагает ввести координаты создаваемого проекта, такие как *groupId*, *artifactId*, *version* и *package*.

```
1 Define value for property 'groupId': : ru.javacore
2 Define value for property 'artifactId': : myproject
3 Define value for property 'version': : 1.0-SNAPSHOT: :
4 Define value for property 'package': : ru.javacore: :
```

groupId - идентификатор группы. Например, организация, ответственная за производство артефакта.

artifactId - имя данного артефакта. Например, имя проекта.

version - номер версии артефакта.

5. После указания всех идентификационных данных проекта остается только подтвердить достоверность введенных данных

```
1 Y: : y
```


Командная строка

```
artifactId: myproject
version: 1.0-SNAPSHOT
package: ru.javacore
Y: : y
[INFO] -----
[INFO] Using following parameters for creating project from Archetype: maven-archetype-quicks
art:1.4
[INFO] -----
[INFO] Parameter: groupId, Value: ru.javacore
[INFO] Parameter: artifactId, Value: myproject
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Parameter: package, Value: ru.javacore
[INFO] Parameter: packageInPathFormat, Value: ru/javacore
[INFO] Parameter: package, Value: ru.javacore
[INFO] Parameter: groupId, Value: ru.javacore
[INFO] Parameter: artifactId, Value: myproject
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] Project created from Archetype in dir: C:\Users\anast\myproject
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 04:45 min
[INFO] Finished at: 2023-02-22T23:01:43+03:00
[INFO] -----
C:\Users\anast>
```

Чтобы скомпилировать, нужно перейти в директорию проекта testproject1 и набрать в консоли mvn compile. Если в консоль выведется

```
...  
[INFO] BUILD SUCCESS  
.....
```

то компиляция прошла успешно и в созданной директории target/classes будут class файлы с нашей программой.

Если вы наберёте mvn package, в директории target будет создан jar файл testproject1-1.0-SNAPSHOT.jar

Давайте запустим скомпилированную нами программу

```
[dima@myhost testproject1]$ java -cp ./target/classes ru.apache_maven.App  
Hello World!
```

Поздравляю, программа работает!

cmd Командная строка

```
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.0.1/maven-shared-utils-3.0.1.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/maven-archiver/3.1.1/maven-archiver-3.1.1.jar (24 kB at 227 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5/xz-1.5.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-io/2.7.1/plexus-io-2.7.1.jar (86 kB at 298 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/3.0.1/maven-shared-utils-3.0.1.jar (154 kB at 225 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/tukaani/xz/1.5/xz-1.5.jar (100 kB at 100 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-archiver/3.4/plexus-archiver-3.4.jar (187 kB at 181 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-compress/1.11/commons-compress-1.11.jar (426 kB at 263 kB/s)
[INFO] Building jar: C:\Users\anast\myproject\target\myproject-1.0-SNAPSHOT.jar
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 13.859 s
[INFO] Finished at: 2023-02-22T23:07:31+03:00
[INFO] -----

C:\Users\anast\myproject>java -cp ./target/classes ru.javacore.App
Hello World!

C:\Users\anast\myproject>
```


Интеграция

Хочу обратить внимание, что полученный проект можно сразу открывать средой разработки:

- для среды IntelliJ Idea проект открывается сразу ("File/OpenProject")
- для среды NetBeans проект открывается сразу ("Файл/Создать проект/Проект maven с существующим POM")
- для eclipse нужно установить соответствующий плагин.

Задание 1:

1. Сгенерировать свой первый проект, используя maven.
2. Фразу “Hello, world” заменить на “Hello, ИМЯ”.
3. Скомпилировать, запустить.

```
Windows PowerShell
Downloading from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.15/plexus-utils-3.0.15.jar
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/classworlds/classworlds/1.1-alpha-2/classworlds-1.1-alpha-2.jar (38 kB at 87 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/commons-codec/commons-codec/1.6/commons-codec-1.6.jar (233 kB at 412 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/junit/junit/3.8.1/junit-3.8.1.jar (121 kB at 204 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-shared-utils/0.4/maven-shared-utils-0.4.jar (155 kB at 236 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/codehaus/plexus/plexus-utils/3.0.15/plexus-utils-3.0.15.jar (239 kB at 267 kB/s)
[INFO] Installing C:\Users\anast\myproject\target\myproject-1.0-SNAPSHOT.jar to C:\Users\anast\.m2\repository\ru\javacore\myproject\1.0-SNAPSHOT\myproject-1.0-SNAPSHOT.jar
[INFO] Installing C:\Users\anast\myproject\pom.xml to C:\Users\anast\.m2\repository\ru\javacore\myproject\1.0-SNAPSHOT\myproject-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.948 s
[INFO] Finished at: 2023-02-22T23:17:20+03:00
[INFO] -----
PS C:\Users\anast\myproject> java -cp ./target/classes/ ru.javacore.App
Hello World!
PS C:\Users\anast\myproject> mvn compile
[INFO] Scanning for projects...
[INFO] -----< ru.javacore:myproject >-----
[INFO] Building myproject 1.0-SNAPSHOT
[INFO] from pom.xml
[INFO] -----[ jar ]-----
[INFO] --- resources:3.0.2:resources (default-resources) @ myproject ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\anast\myproject\src\main\resources
[INFO] --- compiler:3.8.0:compile (default-compile) @ myproject ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\anast\myproject\target\classes
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 2.396 s
[INFO] Finished at: 2023-02-22T23:24:33+03:00
[INFO] -----
PS C:\Users\anast\myproject> java -cp ./target/classes/ ru.javacore.App
Всем привет!
PS C:\Users\anast\myproject>
```

Что такое pom.xml

pom.xml - это основной файл, который описывает проект. Вообще могут быть дополнительные файлы, но они играют второстепенную роль.

Давайте разберём из чего состоит файл pom.xml

Корневой элемент и заголовок.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
                      http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  .....
</project>
```

Корневой элемент `<project>`, схема, которая облегчает редактирование и проверку, и версия POM.

Корневой элемент `<project>`, схема, которая облегчает редактирование и проверку, и версия POM.

Внутри тэга `project` содержится основная и обязательная информация о проекте:

```
<!-- The Basics -->  
<groupId>...</groupId>  
<artifactId>...</artifactId>  
<version>...</version>
```

В Maven каждый проект идентифицируется парой `groupId artifactId`. Во избежание конфликта имён, `groupId` - наименование организации или подразделения и обычно действуют такие же правила как и при именовании пакетов в Java - записывают доменное имя организации или сайта проекта. `artifactId` - название проекта. Внутри тэга `version`, как можно догадаться хранится версия проекта. Тройкой `groupId, artifactId, version` (далее - GAV) можно однозначно идентифицировать `jar` файл приложения или библиотеки. Если состояние кода для проекта не зафиксировано, то в конце к имени версии добавляется "-SNAPSHOT" что обозначает, что версия в разработке и результирующий `jar` файл может меняться. `<packaging>...</packaging>` определяет какого типа файл будет создаваться как результат сборки. Возможные варианты `pom, jar, war, ear`

Зависимости

Зависимости - следующая очень важная часть pom.xml - тут хранится список всех библиотек (зависимостей) которые используются в проекте. Каждая библиотека идентифицируется также как и сам проект - тройкой groupId, artifactId, version (GAV). Объявление зависимостей заключено в тэг <dependencies>...</dependencies>.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.4</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.powermock</groupId>
    <artifactId>powermock-reflect</artifactId>
    <version>${version}</version>
  </dependency>
  <dependency>
    <groupId>org.javassist</groupId>
    <artifactId>javassist</artifactId>
    <version>3.13.0-GA</version>
    <scope>compile</scope>
  </dependency>
</dependencies>
```

Как вы могли заметить, кроме GAV при описании зависимости может присутствовать тэг <scope>. Scope задаёт, для чего библиотека используется. В данном примере говорится, что библиотека с GAV junit:junit:4.4 нужна только для выполнения тестов.

Пример добавления в проект новой зависимости:

1. Перейти на сайт <https://mvnrepository.com/>
2. Найти там нужную библиотеку (в нашем случае она лежит здесь:
<https://mvnrepository.com/artifact/org.apache.commons/commons-lang3/3.0>)
3. Скопировать зависимость:

License	Apache 2.0
Categories	Core Utilities
HomePage	http://commons.apache.org/lang/
Date	(Jul 19, 2011)
Files	pom (17 KB) jar (298 KB) View All
Repositories	Central Adobe Apache Public Apache Releases Redhat GA
Used By	9,904 artifacts

Note: There is a new version for this artifact

New Version	3.7
-------------	-----

Maven Gradle SBT Ivy Grape Leiningen Buildr

```
<!-- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3 -->
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.0</version>
</dependency>
```

4. Доба

5. Выполнить mvn clean install

6. Теперь в коде проекта можно использовать всю функциональность этой библиотеки

Тэг <build>

Тэг <build> не обязательный, т. к. существуют значения по умолчанию. Этот раздел содержит информацию по самой сборке: где находятся исходные файлы, где ресурсы, какие плагины используются. Например:

```
<build>
<outputDirectory>target2</outputDirectory>
  <finalName>ROOT</finalName>
<sourceDirectory>src/java</sourceDirectory>
  <resources>
    <resource>
      <directory>${basedir}/src/java</directory>
      <includes>
        <include>**/*.properties</include>
      </includes>
    </resource>
  </resources>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-pmd-plugin</artifactId>
      <version>2.4</version>
    </plugin>
  </plugins>
</build>
```

- `<sourceDirectory>`
определяет, откуда maven будет брать файлы исходного кода. По умолчанию это `src/main/java`, но вы можете определить, где это вам удобно. Директория может быть только одна (без использования специальных плагинов)
- `<resources>`
и вложенные в неё тэги `<resource>` определяют, одну или несколько директорий, где хранятся файлы ресурсов. Ресурсы в отличие от файлов исходного кода при сборке просто копируются. Директория по умолчанию `src/main/resources`
- `<outputDirectory>`
определяет, в какую директорию компилятор будет сохранять результаты компиляции - *.class файлы. Значение по умолчанию - `target/classes`
- `<finalName>`
- имя результирующего jar (war, ear..) файла с соответствующим типу расширением, который создаётся на фазе `package`. Значение по умолчанию — `artifactId-version`.

Репозитории - это место где хранятся артефакты: jar файлы, pom -файлы, javadoc, исходники.
Существуют:

- Локальный репозиторий по умолчанию он расположен в <home директория>/.m2/repository - персональный для каждого пользователя.
- центральный репозиторий который расположен в <http://repo1.maven.org/maven2/> и доступен на чтение для всех пользователей в интернете.
- Внутренний "Корпоративный" репозиторий- дополнительный репозиторий, один на несколько пользователей.

Локальный репозиторий

Локальный репозиторий по умолчанию расположен в `<home директория>/.m2/repository`. Здесь лежат артефакты которые были скачаны из центрального репозитория либо добавлены другим способом. Например если вы наберёте команду

```
mvn install
```

в текущем проекте, то соберётся jar (или war, pom в зависимости от содержимого тэга `packaging`) который установится в локальный репозиторий. Найти его можно в `<home директория>/.m2/repository/<groupIdPath>/<artifactId>/<version>/<artifactId>-<version>.jar` где `groupIdPath` получается заменой всех точек на слеш. Например для проекта

```
<groupId>ru.apache-maven</groupId>  
<artifactId>site</artifactId>  
<version>1.0-SNAPSHOT</version>
```

jar файл будет лежать по пути: `<home директория>/.m2/repository/ru/apache-maven/site/1.0-SNAPSHOT/site-1.0-SNAPSHOT.jar`

Центральный репозиторий

Чтобы самому каждый раз не создавать репозиторий, сообщество для Вас поддерживает центральный репозиторий. Если для сборки вашего проекта не хватает зависимостей, то они по умолчанию автоматически скачиваются с <http://repo1.maven.org/maven2>. В этом репозитории лежат практически все опенсорсные фреймворки и библиотеки.

Самому в центральный репозиторий **положить** нельзя. Т.к. этот репозиторий используют все, то перед тем как туда попадают артефакты они проверяются, тем более что если артефакт однажды попал в репозиторий, то по правилам изменить его нельзя.

Для поиска нужной библиотеки очень удобно пользоваться сайтами <http://mavenrepository.com/> и <http://findjar.com/>

Корпоративный репозиторий

Если вы хотите создать свой репозиторий, содержимое которого вы можете полностью контролировать (как локальный), и сделать так, чтобы он был доступен для нескольких человек, вам будет полезен корпоративный репозиторий. Доступ к артефактам можно ограничивать настройками безопасности сервера так, что код ваших проектов не будет доступен извне.

Чтобы добавить репозиторий в список, откуда будут скачиваться зависимости, нужно добавить секцию `repositories` в `pom.xml`, например:

```
<project>
  ...
  <repositories>
    <repository>
      <id>my-company-repo</id>
      <url>http://my-company-site.ru/repo</url>
    </repository>
  </repositories>
  ...
</project>
```

Существуют несколько реализаций серверов - репозитория Maven. Наиболее известные это [artifactory](#), [continuum](#), [nexus](#).

Основные фазы сборки проекта

1. compile
Компилирование проекта
2. test
Тестирование с помощью JUnit тестов
3. package
Создание .jar файла или war, ear в зависимости от типа проекта
4. integration-test
Запуск интеграционных тестов
5. install
Копирование .jar (war , ear) в локальный репозиторий
6. deploy
публикация файла в удалённый репозиторий

К примеру нам нужно создать jar проекта. Чтобы его создать набираем:

```
mvn package
```

Но перед созданием jar-файла будут выполняться все предыдущие фазы compile и test , а фазы integration-test, install, deploy не выполнятся. Если набрать

```
mvn deploy
```

то выполнятся все приведённые выше фазы.

С полным списком фаз и их описанием (на английском языке) можно ознакомиться здесь:

http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference

Требования к сборке сильно зависят от проекта. Плагины - это способ расширить функциональность maven в больших диапазонах.

Использование плагина

В простейшем случае запустить плагин просто, например:

```
mvn org.apache.maven.plugins:maven-checkstyle-plugin:check
```

В данном примере вызывается плагин с

- groupId "org.apache.maven.plugins"
- artifactId "maven-checkstyle-plugin"
- последней версией
- и целью (goal) "check"

Цель - это действие, которое плагин может выполнить. Целей может быть несколько.

плагины с groupId "org.apache.maven.plugins" можно запустить в более краткой форме:

```
mvn maven-checkstyle-plugin:check
```

или даже так:

```
mvn checkstyle:check
```

Объявление плагина в pom.xml

Объявление плагина похоже на объявление зависимости. Также, как и зависимости плагины идентифицируются с помощью GAV(groupId,artifactId,version). Например:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.6</version>
</plugin>
```

Объявление плагина в pom.xml позволяет зафиксировать версию плагина, задать ему необходимые параметры, привязать к фазам.

Привязка к фазам сборки проекта

После того как плагин объявлен, его можно настроить так, чтобы он автоматически запускался в нужный момент. Это делается с помощью привязки плагина к **фазе** сборки проекта:

```
<plugin>
  <groupId>org.apache.maven.plugins</groupId>
  <artifactId>maven-checkstyle-plugin</artifactId>
  <version>2.6</version>
  <executions>
    <execution>
      <phase>package</phase>
      <goals>
        <goal>check</goal>
      </goals>
    </execution>
  </executions>
</plugin>
```

в данном примере плагин запустится в фазе проекта package

Общий алгоритм:

1. `mvn archetype:generate`
2. Вводим все необходимые данные или нажимаем Enter

Результат: сгенерировался простейший проект

Для сборки проекта:

1. Переходим в папку сгенерированного проекта
2. `mvn clean install`

Результат: подтянулись прописанные в `pom.xml` зависимости

Для запуска сгенерированного проекта:

1. Переходим в папку сгенерированного проекта
2. `mvn compile`
3. `java -cp ./target/classes ru.javacore.App` (путь к `.class` файлу)

Задание 2

Создать класс Tester со следующими полями:

-name

-surname

-experienceInYears

-englishLevel

-salary

1. Перегрузить в этом классе 3 конструктора, каждый из которых будет последовательно вызывать другой используя this(...);
2. Написать 3 любых перегруженных метода в этом классе
3. Написать статический метод и показать пример его корректного вызова в другом классе