

## **Часть 1:**

### **Типовая структура Java приложения.**

### **Использование системы управления ЖЦ проекта Apache Maven.**

#### **Цель**

Познакомиться с принципами разработки Java-приложений с использованием автоматизированной системы управления жизненным циклом проекта Apache Maven. Получить практические навыки решения типовых задач при сборке Java проекта.

#### **Задание**

1. Создать структуру каталогов, необходимую для организации типового проекта на Java.
2. Разработать приложение, решающее выбранную задачу. Приложение должно содержать не менее 2-х классов, а также вспомогательные ресурсы.
3. Организовать сборку приложения с использованием фреймворка Apache Maven.
4. Сгенерировать основу второго приложения с помощью Apache Maven Archetype. Реализовать вспомогательные классы, которые будут использоваться основным приложением.
5. Объединить оба приложения в один многомодульный Maven-проект.
6. Определить второй проект как зависимость в конфигурационном файле основного проекта.
7. Организовать модульное тестирование приложения (мин. 1 класс, 2 теста)

#### **Дополнительное задание (advanced)**

1. Определить параметры сборки проекта.
2. Подключить стороннюю библиотеку в проект, определив Maven-зависимость.
3. Настроить несколько профилей сборки проекта (мин. 2 профиля). Определить различные параметры сборки проекта в зависимости от профиля.
4. Настроить проверку стиля и качества исходного кода с помощью соответствующего плагина Apache Maven.
5. Настроить вызов maven-команд для сборки проекта из IDE.

## Инструментарий

1. Java Development Kit 8.
2. Apache Maven 3.5.
3. Текстовый редактор с подсветкой синтаксиса Java. Например, Sublime 3.
4. Клиент Git 2.18.
5. Доступ к сети Интернет

## Вопросы для допуска

1. Какие основные каталоги выделяют в типовой структуре Java проекта? Объясните их назначение.
2. Что такое Apache Maven? Объясните понятие «maven-артефакт».
3. Как выглядит полноквалифицированное имя артефакта Maven?
4. Объясните понятие «classpath проекта».
5. Какую задачу будет решать разрабатываемое приложение?

## Теоретические сведения

Теоретические сведения приведены в приложении к настоящим методическим указаниям (файл JavaTech\_Maven.pdf)

Кроме того, обратите внимание на следующие электронные ресурсы:

1. Официальный сайт проекта Apache Maven – <https://maven.apache.org/>
2. Список книг и статей об Apache Maven – <https://maven.apache.org/articles.html>
3. Детальное описание типовой структуры проекта – <https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html>
4. Страница Apache Maven на Wikipedia.org – [https://ru.wikipedia.org/wiki/Apache\\_Maven](https://ru.wikipedia.org/wiki/Apache_Maven)
5. Неофициальное русскоязычное руководство по Apache Maven – <http://www.apache-maven.ru/>
6. <https://www.baeldung.com/maven>
7. <http://cs.lmu.edu/~ray/notes/largejavaapps/>

## Подготовка рабочего окружения

1. Создать рабочую директорию, т.н. workspace. Настоятельно рекомендуем, чтобы полный путь до создаваемого каталога не содержал пробелов и состоял только из латинских символов. В нашем случае это

```
C:\Users\wirbel\devel\uni
```

```
# development workspace
```

2. Структура подкаталогов предлагается следующей:

```
<workspace>
  apps          # каталог приложений и утилит разработки
  dist          # дистрибутивы приложений и утилит
  projects      # исходный код проектов
```

В каталоге apps расположены переносимые (portable) версии приложений, таких как Sublime, Git Client. Apache Maven так же допускает возможность запуска из произвольного каталога без непосредственной установки. В случае штатной установки в систему всех необходимых утилит, катало можно не использовать.

3. Убедиться, что JDK 8 установлен корректно. Загрузить дистрибутив можно по ссылке:

<http://www.oracle.com/technetwork/pt/java/javase/downloads/jdk8-downloads-2133151.html>

Для этого необходимо в консоли выполнить команду

```
java -version
```

Результат должен быть примерно следующим:

```
java version "1.8.0_181"
Java(TM) SE Runtime Environment (build 1.8.0_181-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.181-b13, mixed mode)
```

Если нет, проверьте, что JDK действительно установлен. По умолчанию, путь до домашней (установочной) директории JDK имеет вид:

```
C:\Program Files\Java\jdk1.8.0_181
```

Также убедитесь, что путь к бинарным файлам прописан в переменной окружения Path:

```
C:\Program Files\Java\jdk1.8.0_181\bin\
```

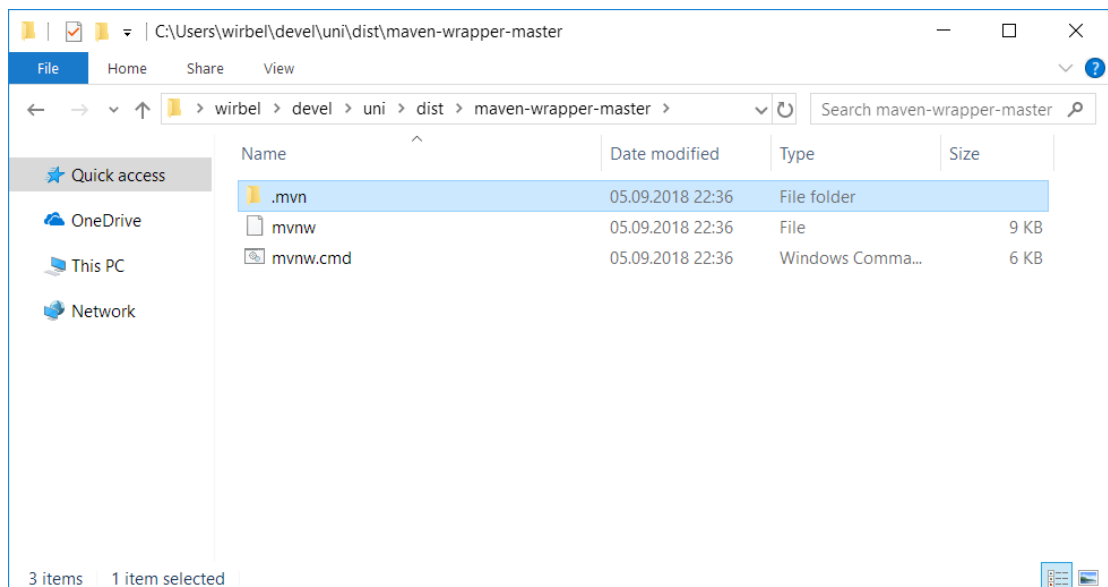
4. Кроме того, для корректной работы Apache Maven необходимо установить переменную окружения

```
JAVA_HOME=C:\Program Files\Java\jdk1.8.0_181\
```

Убедитесь, что переменная определена корректно, возможно, потребуется повторный вход в систему или перезагрузка компьютера:

```
C:\Users\wirbel\devel\uni>echo %JAVA_HOME%
C:\Program Files\Java\jdk1.8.0_181\
```

5. Скачать и установить текстовый редактор с подсветкой синтаксиса Java. Интегрированные среды разработки (IDE) в данной лабораторной работе не используются. В качестве примера будем использовать текстовый редактор Sublime Text 3. Портативная версия доступна по адресу: <https://www.sublimetext.com/3>
6. Скачать и установить Apache Maven 3.5.4.  
Сделать это можно по инструкции, размещенной на официальном сайте. Или с использованием утилиты maven-wrapper. Maven Wrapper позволяет «встроить» дистрибутив Apache Maven непосредственно в исходный код проекта, обеспечив тем самым полную переносимость проекта и его независимость от установленного программного обеспечения. Скрипт-обертка (mvnw) при первом обращении автоматически скачает нужный дистрибутив Apache Maven и выполнит инициализацию. Далее этот скрипт необходимо использовать ровно так же, как и исполняемый файл Apache Maven (mvn). Все команды будут транслироваться в вызовы установленного дистрибутива Apache Maven. Apache Maven Wrapper можно загрузить в виде архива из репозитория разработчика на github.com: <https://github.com/takari/maven-wrapper>. Для этого нажмите «Clone or download» и загрузите архив с проектом. Из архива нам понадобятся каталог .mvn и два скрипта: mvnw.cmd для Windows и его Linux-версия mvnw.



## Порядок выполнения работы

1. Создайте структуру каталогов для проекта lab01, согласно описанию в разделе «Теоретические сведения»:

```
cd <workspace>\projects\  
md lab01\myapp  
cd lab01\myapp  
md src\main\java src\main\resources src\test\java src\test\resources  
target
```

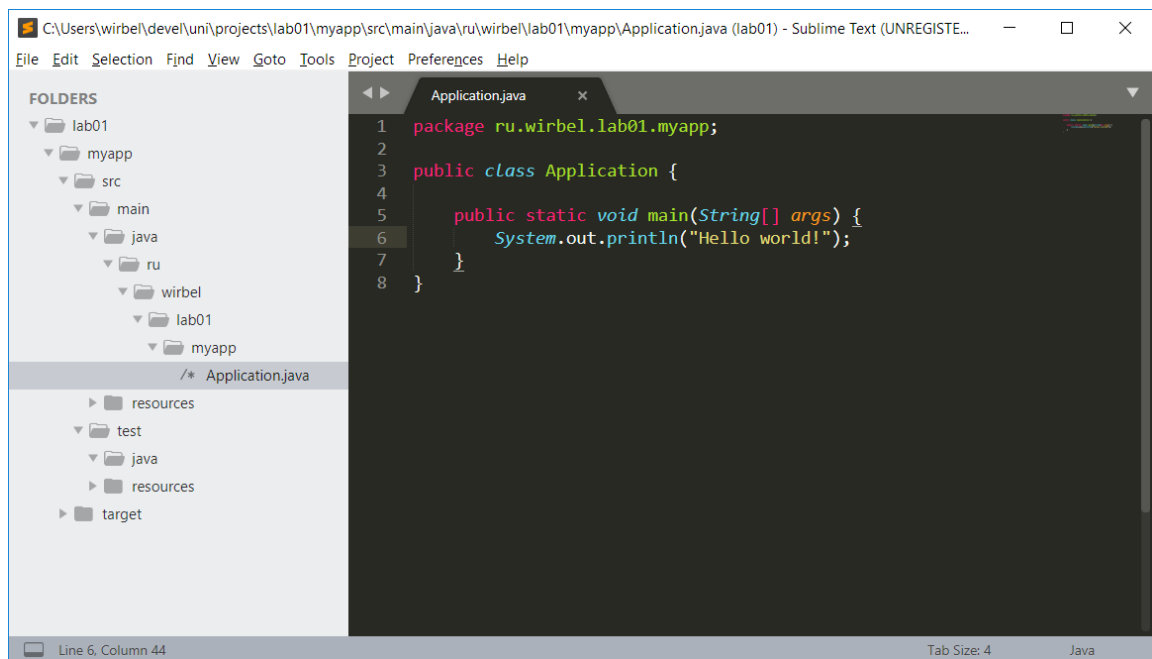
2. После этого запустите текстовый редактор Sublime Text и откройте каталог <workspace>\projects\lab01. Для этого в меню File выберите опцию Open folder... На панели в левой части окна будет доступно дерево созданных каталогов.
3. Создайте основной класс Java приложения Application.java. Для этого предварительно создайте иерархию каталогов, соответствующую пакету класса Application. Полное имя пакета имеет вид:

```
<domain>.<company|user>.<project>.<module>.<package>
```

В нашем случае это будет выглядеть так:

```
cd <workspace>\projects\lab01\myapp\src\main\java  
md ru\wirbel\lab01\myapp
```

Содержимое класса Application представлено ниже:

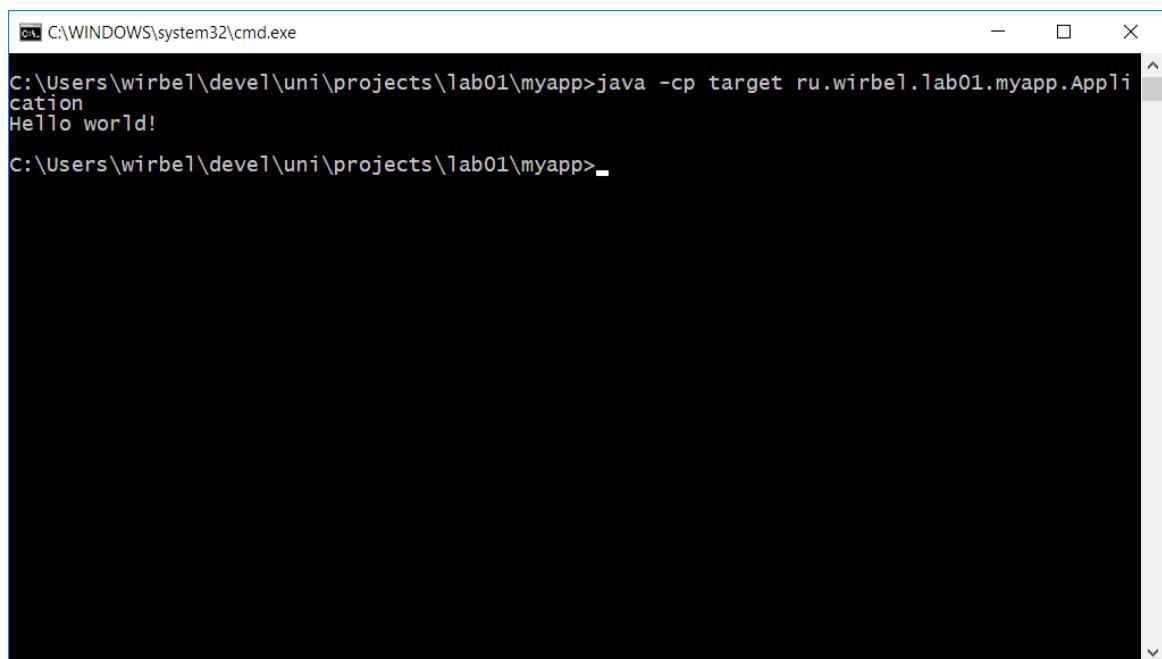


Убедитесь, что этот и все последующие файлы сохранены в кодировке UTF-8. Изменить кодировку можно с помощью меню File опции Save with encoding...

4. Далее необходимо выполнить компиляцию Java-классов приложения. Скомпилированные классы должны размещаться в соответствующем подкаталоге директории target. Для этого выполните javac со следующими параметрами:

```
C:\Users\wirbel\devel\uni\projects\lab01\myapp>javac -d target ^  
-sourcepath src\main\java ^  
-encoding UTF-8 ^  
src\main\java\ru\wirbel\lab01\myapp\Application.java
```

5. Теперь можно выполнить скомпилированный класс и убедиться, что результат соответствует ожидаемому.



```
C:\WINDOWS\system32\cmd.exe  
C:\Users\wirbel\devel\uni\projects\lab01\myapp>java -cp target ru.wirbel.lab01.myapp.Appli  
cation  
Hello world!  
C:\Users\wirbel\devel\uni\projects\lab01\myapp>
```

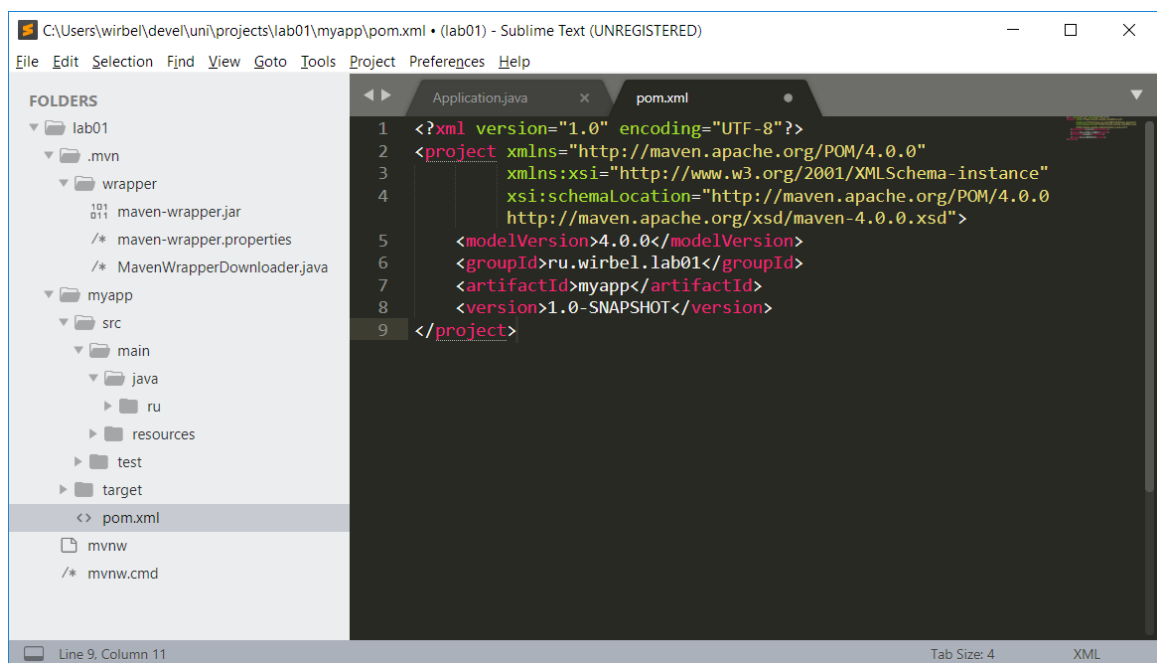
Таким образом выглядит сборка приложения вручную с использованием базовых средств платформы Java SE.

6. Теперь добавим поддержку Apache Maven в наш проект. Для этого достаточно создать конфигурационный файл `pom.xml` в корневой директории проекта, и определить в нём базовые параметры приложения.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>

  <groupId> </groupId>          <!-- Идентификатор группы -->
  <artifactId> </artifactId>    <!-- Идентификатор артефакта -->
  <version> </version>          <!-- Версия артефакта -->
  <name> </name>                <!-- Название модуля -->

</project>
```



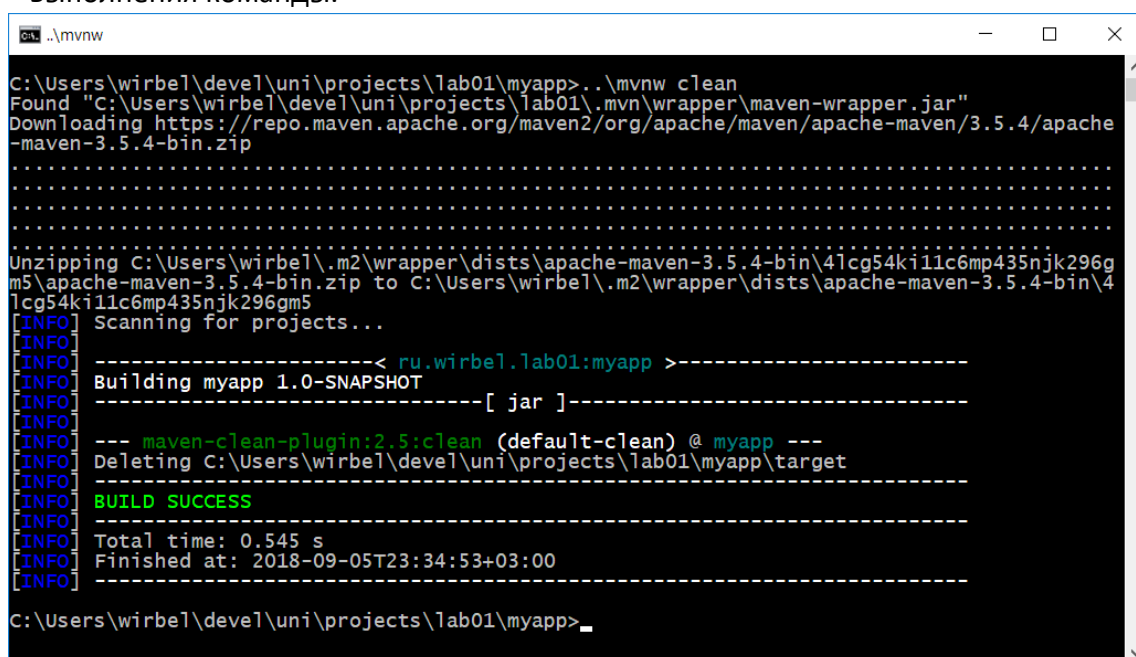
7. Если вы выполнили штатную установку Apache Maven и не планируете использовать Maven Wrapper, переходите к следующему пункту. Для установки встроенного Maven в проект необходимо скопировать каталог `.mvn` и скрипты `mvnw.cmd` и `mvnw` в директорию

```
<workspace>\projects\lab01
```

8. Для первоначальной инициализации Maven-проекта достаточно выполнить любую команду, например, очистку проекта. Обратите внимание, что команды, относящиеся к работе с проектом `myapp`, необходимо выполнять из корневой директории этого проекта, тогда как утилита `mvnw` расположена каталогом выше.

```
cd workspace\projects\lab01\myapp
..\mvnw clean
```

При первом запуске скрипт выполняет загрузку дистрибутива Apache Maven и транслирует команду `clean`. В свою очередь Maven выполняет инициализацию проекта и при необходимости загружает зависимости, которые требуются для выполнения команды.



```
C:\Users\wirbel\devel\uni\projects\lab01\myapp>..\mvnw clean
Found "C:\Users\wirbel\devel\uni\projects\lab01\..\.mvn\wrapper\maven-wrapper.jar"
Downloading https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.5.4/apache-maven-3.5.4-bin.zip
.....
Unzipping C:\Users\wirbel\.m2\wrapper\dists\apache-maven-3.5.4-bin\41cg54ki11c6mp435nj296gm5\apache-maven-3.5.4-bin.zip to C:\Users\wirbel\.m2\wrapper\dists\apache-maven-3.5.4-bin\41cg54ki11c6mp435nj296gm5
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ru.wirbel.lab01:myapp >-----
[INFO] Building myapp 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ myapp ---
[INFO] Deleting C:\Users\wirbel\devel\uni\projects\lab01\myapp\target
[INFO] BUILD SUCCESS
[INFO]
[INFO] -----
[INFO] Total time: 0.545 s
[INFO] Finished at: 2018-09-05T23:34:53+03:00
[INFO] -----
C:\Users\wirbel\devel\uni\projects\lab01\myapp>
```

Убедитесь, что в результате выполнения команды `mvn clean` была удалена директория `target` со всем содержимым.

9. Выполните компиляцию и сборку проекта в виде `jar`-архива. Для этого выполните команду `package`.

```
..\mvnw package
```

Убедитесь, что в каталоге `target` были созданы скомпилированные классы Java, а также JAR-архив с приложением. Обратите внимание на название архива.



10. Запустите собранный проект, убедитесь, что результат работы приложения по-прежнему соответствует ожидаемому.

```
C:\Users\wirbel\devel\uni\projects\lab01\myapp>java -cp target\myapp-1.0-SNAPSHOT.jar ru.wirbel.lab01.myapp.Application
```

Обратите внимание, что все зависимости вашего приложения должны быть указаны в classpath при запуске.

11. Сконфигурируем проект таким образом, чтобы запуск приложения, как и сборка, осуществлялся силами Maven. Фреймворк автоматически на основании зависимостей проекта сформирует описание classpath и соответствующим образом выполнит запуск.

Запуск приложения можно автоматизировать с помощью специального Maven-плагина `org.codehaus.mojo:exec-maven-plugin`.

Определим привязку плагина к фазам сборки проекта, для этого внесём изменения в `pom.xml`:

```
<project>
  ...
  <build>
    <plugins>
      <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>exec-maven-plugin</artifactId>
        <version>1.6.0</version>
        <configuration>
          <mainClass>ru.wirbel.lab01.myapp.Application</mainClass>
        </configuration>
      </plugin>
    </plugins>
  </build>
  ...
</project>
```

В атрибуте `mainClass` указывается полноквалифицированное имя класса приложения, содержащего метод `main(String[] args)`.

12. Выполните запуск проекта с помощью команды `exec:java`. Обратите внимание, что проект должен быть предварительно скомпилирован. Команды Maven, относящиеся к разным циклам сборки приложения, можно выполнять вместе. При этом они будут выполнены последовательно в порядке, указанном при вызове команды. Выполним компиляцию и запуск проекта.

```
C:\Users\wirbel\devel\uni\projects\lab01\myapp>..\mvnw compile exec:java
Found "C:\Users\wirbel\devel\uni\projects\lab01\.mvn\wrapper\maven-wrapper.jar"
[INFO] Scanning for projects...
[INFO]
[INFO] -----< ru.wirbel.lab01:myapp >-----
[INFO] Building myapp 1.0-SNAPSHOT
[INFO] -----[ jar ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ myapp ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 0 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ myapp ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\wirbel\devel\uni\projects\lab01\myapp\target\classes
[INFO]
[INFO] --- exec-maven-plugin:1.6.0:java (default-cli) @ myapp ---
Привет, Мир!
[INFO] BUILD SUCCESS
[INFO]
[INFO] Total time: 3.665 s
[INFO] Finished at: 2018-09-06T06:07:56+03:00
[INFO]
C:\Users\wirbel\devel\uni\projects\lab01\myapp>
```

13. Если вы выводите в консоль Windows символы кириллицы, то вместо желаемого результата сначала получите нечитаемый текст. Это происходит из-за того, что кодировка консоли Windows (в большинстве случаев Cp866) отличается от кодировки русского текста в исходных файлах проекта (UTF-8). Кроме того, Java при записи в стандартный поток вывода определяет системную кодировку и выполняет необходимое преобразование. Для Windows системной кодировкой является Cp1251.

Исправить ситуацию можно в два шага, например, так. Во-первых, отключим автоматическое преобразование кодировки в системную при записи в консоль. Вместо этого всегда будем писать в кодировке UTF-8. В принципе, этого достаточно для обеспечения переносимости приложения. Linux-системы по умолчанию используют как раз UTF-8. В любом случае, оставим себе возможность изменить кодировку с помощью определения системной переменной.

Для этого добавим в класс Application служебный метод:

```
public static final String DEFAULT_CONSOLE_ENCODING = "UTF-8";
public static final String CONSOLE_ENCODING_PROPERTY = "consoleEncoding";

private static void setConsoleEncoding() {
    // чтение системной переменной
    String consoleEncoding = System.getProperty(CONSOLE_ENCODING_PROPERTY,
                                                DEFAULT_CONSOLE_ENCODING);

    try {
        // установить кодировку стандартной консоли вывода
        System.setOut(new PrintStream(
            System.out,
            true,
            consoleEncoding)
        );

    } catch (UnsupportedEncodingException ex) {
        System.err.println(
            "Unsupported encoding set for console: " + consoleEncoding
        );
    }
}
```

Теперь достаточно вызвать этот метод до первого вызова записи в консоль `System.out.println(“”)`.

14. Теперь мы пишем в консоль строго в кодировке UTF-8, если другая не задана системной переменной. Осталось «научить» консоль Windows распознавать UTF-8. Для текущего сеанса работы кодировку консоли можно переключить в UTF-8 с помощью команды:

```
chcp 65001
```

```
# change code page
```

15. Создайте в проекте туарр дополнительно несколько классов, которые будут использоваться для решения выбранной задачи. В нашем случае, это будет класс `GreetingsBuilder`, в задачи которого будет входить формирование приветствия в зависимости от языка и стиля. Для начала будем поддерживать два языка: русский и английский – и два стиля: официальный и дружеский. Требование по мультиязычности реализуем с помощью средств интернационализации (i18n), заложенных в Java SE. Тексты сообщений на разных языках будем хранить в ресурсах приложения в файлах типа `properties`, и загружать сообщение по ключу из соответствующего файла в зависимости от выбранного языка. Для решения вышей задачи также определитесь, какие ресурсы приложения вы будете использовать. Это также могут быть коллекции сообщений (`message bundle`) или что-то другое. Итак, мы добавили классы `Greetings` и `GreetingsBuilder`. Полный текст этих классов вы можете найти в исходном коде проекта в приложении к этим методическим указаниям. Также были созданы два файла-словаря:

```
src/main/resources/messages_ru.properties
src/main/resources/messages_en.properties
```

```
# русский
# английский
```

16. Обратите внимание на плагин `org.codehaus.mojo:native2ascii-maven-plugin`, который подключен в конфигурационном файле `pom.xml` и выполняет преобразование содержимого словарей (\*.properties) в последовательность кодов `ascii`. Это необходимо для корректной обработки словарей и связано с особенностями реализации класса `PropertyResourceBundle`, который выполняет загрузку сообщений из словаря в кодировке `ISO-8859-1`.
17. Как правило, с развитием и ростом приложения появляется необходимость дополнительно структурировать исходный код и вынести его часть в отдельное приложение или библиотеку. Тем самым может решаться задача повторного использования этого кода, выделения его в независимый проект или даже организации разработки этого нового проекта отдельной командой. Создадим новый проект `mylib` и перенесем классы `Greetings`, `GreetingsBuilder`, а также файлы словарей в эту библиотеку. Для этого выполним генерацию шаблона проекта с помощью подсистемы `Apache Maven Archetype`. Система архетипов представляет собой коллекцию шаблонов проектов, обладающих определенной функциональностью. Выполним создание проекта на основе архетипа `maven-archetype-quickstart` в интерактивном режиме. Система последовательно запросит реквизиты нового артефакта (`groupId`, `artifactId`, `version`, `package`). Определим следующие значения (см. скриншот ниже):

```
groupId = ru.wirbel.lab01
artifactId = mylib
version = 1.0-SNAPSHOT
package = ru.wirbel.lab01.mylib
```

```
C:\Users\wirbel\devel\uni\projects\lab01>mvnw archetype:generate ^
-DarchetypeArtifactId=maven-archetype-quickstart
```

Убедитесь, что в текущей директории был создан каталог `mylib`, внутри которого сгенерирован `pom.xml` и необходимые каталоги/файлы проекта.

```
mvnw
C:\Users\wirbel\devel\uni\projects\lab01>mvnw archetype:generate -DarchetypeArtifactId=maven-archetype-quickstart
Found "C:\Users\wirbel\devel\uni\projects\lab01\mvn\wrapper\maven-wrapper.jar"
[INFO] Scanning for projects...
[INFO] -----< org.apache.maven:standalone-pom >-----
[INFO] Building Maven Stub Project (No POM) 1
[INFO] -----[ pom ]-----
[INFO] >>> maven-archetype-plugin:3.0.1:generate (default-cli) > generate-sources @ standalone-pom >>>
[INFO] <<< maven-archetype-plugin:3.0.1:generate (default-cli) < generate-sources @ standalone-pom <<<
[INFO] --- maven-archetype-plugin:3.0.1:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
Define value for property 'groupId': ru.wirbel.lab01
Define value for property 'artifactId': mylib
Define value for property 'version': 1.0-SNAPSHOT: :
Define value for property 'package': ru.wirbel.lab01: : ru.wirbel.lab01.mylib
Confirm properties configuration:
groupId: ru.wirbel.lab01
artifactId: mylib
version: 1.0-SNAPSHOT
package: ru.wirbel.lab01.mylib
Y: :
[INFO] Using following parameters for creating project from Old (1.x) Archetype: maven-archetype-quickstart:1.0
[INFO] -----
[INFO] Parameter: basedir, Value: C:\Users\wirbel\devel\uni\projects\lab01
[INFO] Parameter: package, Value: ru.wirbel.lab01.mylib
[INFO] Parameter: groupId, Value: ru.wirbel.lab01
[INFO] Parameter: artifactId, Value: mylib
[INFO] Parameter: packageName, Value: ru.wirbel.lab01.mylib
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: C:\Users\wirbel\devel\uni\projects\lab01\mylib
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 35.651 s
[INFO] Finished at: 2018-09-06T01:55:22+03:00
[INFO] -----
C:\Users\wirbel\devel\uni\projects\lab01>_
```

18. Перенесем классы Greetings, GreetingsBuilder из проекта myapp в проект mylib. Пакет при перемещении не меняется, классы будут перенесены в тот же подкаталог

```
src\main\java\ru\wirbel\lab01\mylib\
```

19. Аналогичным образом перенесем файлы словарей message\_\*.properties в каталог

```
src\main\resources
```

20. Перенести объявление плагина org.codehaus.mojo:native2ascii-maven-plugin из <workspace>\projects\lab01\myapp\pom.xml в файл <workspace>\projects\lab01\mylib\pom.xml.

21. Убедимся, что проект mylib компилируется, с помощью maven-команды compile.
22. Убедимся, что проект myapp НЕ компилируется. Объясните причину ошибки.
23. В конфигурационном файле myapp/pom.xml определим зависимость от библиотеки mylib.

```
<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">

    ...

    <dependencies>
        <dependency>
            <groupId>ru.wirbel.lab01</groupId>
            <artifactId>mylib</artifactId>
            <version>${project.version}</version>
            <scope>compile</scope>
        </dependency>
    </dependencies>

    ...
</project>
```

24. Вернитесь к пункту 22. Объясните причину ошибки компиляции.
25. Выполните сборку и установку артефакта mylib в локальный репозиторий Maven, и далее компиляцию и запуск myapp.

```
<workspace>\projects\lab01\mylib>..\mvnw clean install
<workspace>\projects\lab01\myapp>..\mvnw compile exec:java
```

Команда install выполняет установку артефакта в локальный репозиторий в домашней директории пользователя. В нашем случае это:

```
C:\Users\wirbel\.m2\repository
```

Найдите там установленный артефакт нашего проекта.

26. Модель проекта Apache Maven предусматривает возможность наследования. Т.е., определив зависимости, фазы сборки, переменные в родительском проекте, можно таким образом сформировать базовую конфигурацию для всех дочерних проектов. Кроме того, управляя жизненным циклом родительского проекта, например, выполняя компиляцию и сборку артефакта, Maven рекурсивно выполнит компиляцию и сборку всех дочерних проектов, самостоятельно определив порядок запуска в соответствии с графом зависимостей подпроектов друг от друга.
- Определим корневой проект lab01, состоящий из двух модулей myapp и mylib.

Для этого необходимо создать pom.xml в директории

```
<workspace>\projects\lab01\
```

следующего содержания

```
<?xml version="1.0" encoding="UTF-8"?>
<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>

  <groupId>ru.wirbel.lab01</groupId>
  <artifactId>lab01</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>

  <modules>
    <module>myapp</module>
    <module>mylib</module>
  </modules>

</project>
```

27. Определим в корневом проекте базовые настройки, задающие целевую версию Java SE и кодировку исходных файлов проекта:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
  ...

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.target>1.8</maven.compiler.target>
    <maven.compiler.source>1.8</maven.compiler.source>
    <java.version>1.8</java.version>
  </properties>

  ...
</project>
```

28. Очистите и соберите корневой проект, убедитесь, что сформированы артефакты в каталогах target дочерних проектов.

```
C:\Users\wirbel\devel\uni\projects\lab01>mvnw clean install
```

29. Убедитесь, что проект myapp по-прежнему выполняется.

```
C:\Users\wirbel\devel\uni\projects\lab01\myapp>..\mvnw exec:java
```

30. Далее необходимо написать тест для одного или нескольких классов вашего приложения. В нашем примере будем писать тест для класса GreetingsBuilder. Обратите внимание на тест AppTest, который был сгенерирован при создании проекта.

Тест использует фреймворк для модульного тестирования JUnit версии 3. Начиная с JUnit 4 поддерживаются аннотации. Изменим версию JUnit в объявлении зависимости в mylib/pom.xml. Обратите внимание на scope (область применения) зависимости -- test.

```
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
```

31. Напишите тест, используя аннотации JUnit. Описание аннотаций доступно в комментариях исходного кода. Модульный тест к классу MyClass – это, как правило, Java-класс с суффиксом Test (MyClassTest), располагающийся в директории src/test/java проекта в каталоге, соответствующем пакету исходного класса (в том же пакете). В нашем случае это

```
src/test/java/ru/wirbel/lab01/mylib/GreetingsBuilderTest.java
```

32. Тестирование maven-проекта запускается автоматически при сборке, т.к. является неотъемлемой частью жизненного цикла проекта. Тестирование можно запустить как самостоятельную фазу. Запустим тестирование всего проекта

```
C:\Users\wirbel\devel\uni\projects\lab01>mvnw test
```

Обратите внимание на последовательность выполнения проаннотированных методов теста (см. лог запуска).

33. Второй тест напишите самостоятельно. Ниже приведен пример запуска с проваленным тестом. Обратите внимание, что сборка проекта myapp даже не начиналась, т.к. была обнаружена ошибка в проекте mylib, артефакт которого необходим в качестве зависимости.



```
mvnw

Results :

Failed tests:   testGreetingsBuilder1(ru.wirbel.lab01.mylib.GreetingsBuilderTest): English
formal greetings check expected:<Hello,[ ]World!> but was:<Hello,[ ]World!>

Tests run: 2, Failures: 1, Errors: 0, Skipped: 0

[INFO] -----
[INFO] Reactor Summary:
[INFO] lab01 1.0-SNAPSHOT ..... SUCCESS [ 0.017 s]
[INFO] mylib ..... FAILURE [ 3.283 s]
[INFO] myapp 1.0-SNAPSHOT ..... SKIPPED
[INFO] -----
[INFO] BUILD FAILURE
[INFO] -----
[INFO] Total time: 3.478 s
[INFO] Finished at: 2018-09-06T02:36:02+03:00
[INFO] -----
[ERROR] Failed to execute goal org.apache.maven.plugins:maven-surefire-plugin:2.12.4:test (
default-test) on project mylib: There are test failures.
[ERROR] Please refer to C:\Users\wirbel\devel\uni\projects\lab01\mylib\target\surefire-repo
[ERROR] rts for the individual test results.
[ERROR] -> [Help 1]
[ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
[ERROR] Re-run Maven using the -X switch to enable full debug logging.
[ERROR] For more information about the errors and possible solutions, please read the follo
[ERROR] wing articles:
[ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoFailureException
[ERROR] After correcting the problems, you can resume the build with the command
[ERROR] mvn <goals> -rf :mylib

C:\Users\wirbel\devel\uni\projects\lab01>
```

34. Переходите к самостоятельному выполнению дополнительных заданий или ко второй части лабораторной работы – работа с системой контроля версий исходного кода проекта Git.

## Контрольные вопросы

1. Назовите основные фазы жизненного цикла проекта, предусмотренные методологией Apache Maven
2. Что отличает maven-проект от обычного java-проекта? Какой минимальный набор конфигураций необходимо выполнить.
3. Каким образом определить, что модуль A зависит от модуля B?
4. Каким образом определить, что проект зависит от сторонней библиотеки?
5. Какие области видимости (scopes) предусмотрены при определении зависимости? Чем они отличаются?
6. И т.д.