Lab1- Введение в язык программирования Java

Важные Организационные моменты

Освоение компетенций, предусмотренных образовательной программой, оценивается по 100-балльной системе:

- 40 баллов - текущий учебный контроль (ТКУ)
- 60 баллов - зачет или экзамен.

Баллы за текущий контроль выставляются в соответствии с БРС дисциплины, разработанной ведущим лектором в соответствии с РПД. Баллы за ТКУ делятся на две части:

- 20 баллов за промежуточную аттестацию - выставляются в электронные ведомости в середине семестра (начало ноября, начало апреля, конкретные даты объявляются рассылкой по Департаменту);
- 20 баллов за работу в семестре выставляются в электронные ведомости вместе с баллами за зачет или экзамен.

Обучающийся, получивший в промежуточную аттестацию:

- от 7 до 20 баллов, считается аттестованным,
- от 0 до 6 баллов - не аттестованным.

Порядок перевода 100-балльной оценки в пятибалльную

100-балльная система	5-балльная система
86-100	отлично
70-85	хорошо
50-69	удовлетворительно
менее 50	неудовлетворительно
50-100	Зачтено
менее 50	не зачтено

Студентам, набравшим более 35 баллов в семестре по решению преподавателя, может быть выставлена оценка за зачет автоматом. При выставлении зачета автоматом баллы, набранные в семестре студентом, умножаются на коэффициент 1,5 для получения балла за зачет (с округлением при необходимости). Пример: студент набрал в семестре 37 баллов. При выставлении автоматом ему ставится оценка за зачет 37 * 1,5 = 55,5 ~= 56 баллов. Итоговая оценка таким образом получается 37 + 56 = 93 балла.

Критерии балльной оценки:

- Посещение занятий (20%). Допускается пропуск двух занятий в случае крайней необходимости. После этого вы начнете терять баллы.
- Работа и участие в уроках (20 %). В конце каждого урока я проверяю, что было сделано, а что нет. У вас есть шанс сделать то, что вы не успелы дома и показать мне потом.
- Короткие и простые тесты в конце каждого модуля (20%).
- Домашнее задание (40 %). Вы несете ответственность за то, что пишете! Не пишите того, чего не понимаете. Не копируйте работу своих коллег.

В случае каких-либо изменений я дам вам знать!!!

Основные цели семинара

- Знакомство со следующими понятиями
    - Java development kit JDK
    - Java Runtime Environment
    - Maven
- Структура проекта
- Установка Java JDK и Intellij Idea
- Написать программу Hello World на Java

Комплект разработчика (Java development kit JDK)

Существует четыре платформы или комплекты разработчика Java:

- Java SE (аббревиатура от «Java Standard Edition»): Java SE API предоставляет основные функции и возможности языка программирования Java.
- Java EE (аббревиатура от «Java Enterprise Edition»): Платформа Java EE построена на основе платформы Java SE. Платформа Java EE предоставляет API и среду выполнения для разработки и запуска крупномасштабных, многоуровневых, масштабируемых, надежных и безопасных сетевых приложений.
- Java ME (аббревиатура от «Java Micro Edition»): Платформа Java ME предоставляет API и небольшую виртуальную машину для запуска приложений языка программирования Java на небольших и менее мощных устройствах, таких как мобильные телефоны. Java ME API — это подмножество Java SE API вместе со специальными библиотеками классов, полезными для разработки приложений для устройств. Приложения Java ME часто являются клиентами служб платформы Java EE.
- JavaFX - это платформа для создания RIA (Rich Internet Applications) с использованием легковесного API пользовательского интерфейса. Приложения JavaFX используют графику с аппаратным ускорением в дополнение к движкам мультимедиа. Существует также высокоуровневый API для подключения к сетевым источникам данных. Приложения JavaFX могут быть клиентами служб платформы Java EE.

Все четыре поддерживаются компанией Oracle и не являюстя программным обеспечением с открытым исходным кодом. Они подлежат лицензированию Oracle. Есть еще один вариант-OpenJDK (Open Java Development Kit) , являющийся бесплатной реализацией платформы Java Standard Edition (Java SE) с открытым исходным кодом.

Среда выполнения Java JRE (Java Runtime Environment)

среда выполнения — это часть программного обеспечения, предназначенная для запуска другого программного обеспечения.

Java JRE – это среда выполнения для Java, которая работает поверх операционной системы, предоставляя дополнительные ресурсы, специфичные для Java. JRE использует для работы три основных компонента:

- Загрузчик классов (Java ClassLoader) -динамически загружает все необходимые файлы классов в виртуальную машину Java (JVM) по требованию.
- Верификатор байт-кода - проверяет формат и точность кода Java перед его загрузкой в JVM. Например, если код нарушает целостность системы или права доступа, JRE не загрузит файл класса.
- Интерпретатор - после успешной загрузки байт-кода интерпретатор Java создает экземпляр JVM, который запускает программу Java.

Различие между JDK и JRE

Различие заключается в том, что JDK представляет собой пакет инструментов для разработки программного обеспечения, тогда как JRE представляет собой пакет инструментов для запуска Java-кода. JRE может

использоваться, как отдельный компонент для простого запуска Java-программ, либо быть частью JDK. JDK требуется JRE, потому что запуск программ является неотъемлемой частью их разработки.

Интегрированная среда разработки (IDE)

Существуют множество сложных сред разработки среди доступных на рынке:

1- Eclipse
2- Netbeans
3- Intellij Idea
4- BlueJ
5- JDeveloper
6- DrJava
7- jGRASP
8- MyEclipse
9- Xcode
10- Codenvy

Установка Java JDK и Intellij Idea

Это интегрированная среда разработки программного обеспечения для многих языков программирования, в частности Java, JavaScript, Python, разработанная компанией JetBrains.

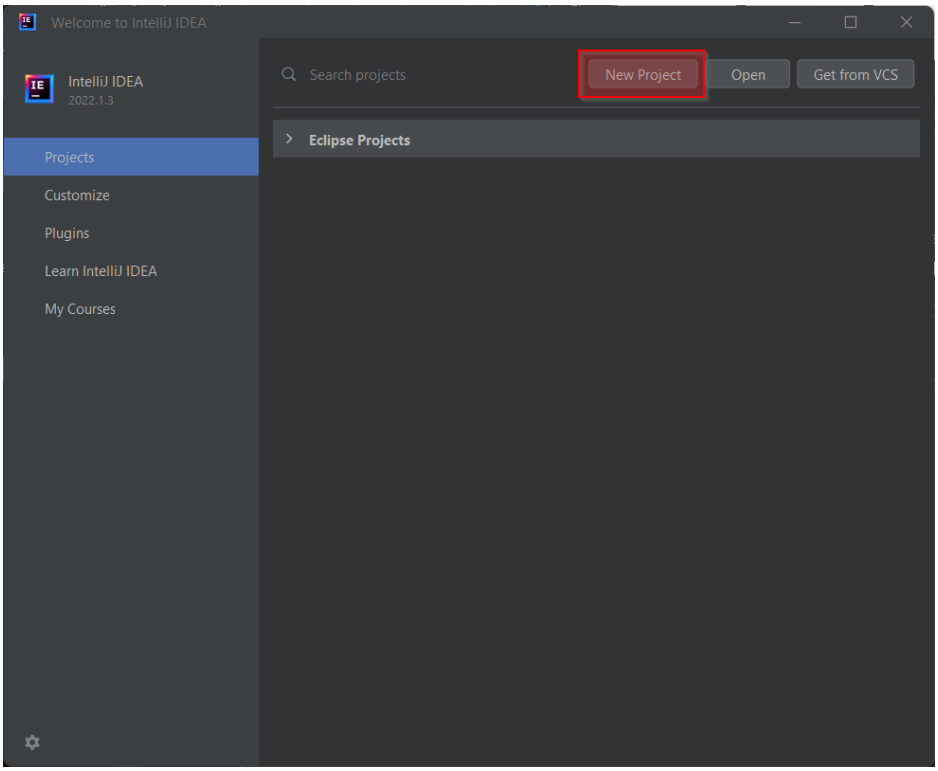
Его можно скачать по следующей ссылке: https://www.jetbrains.com/idea/download

Java SE Development Kit можно скачать по ссылке: https://www.oracle.com/java/technologies/downloads/#jdk18-windows

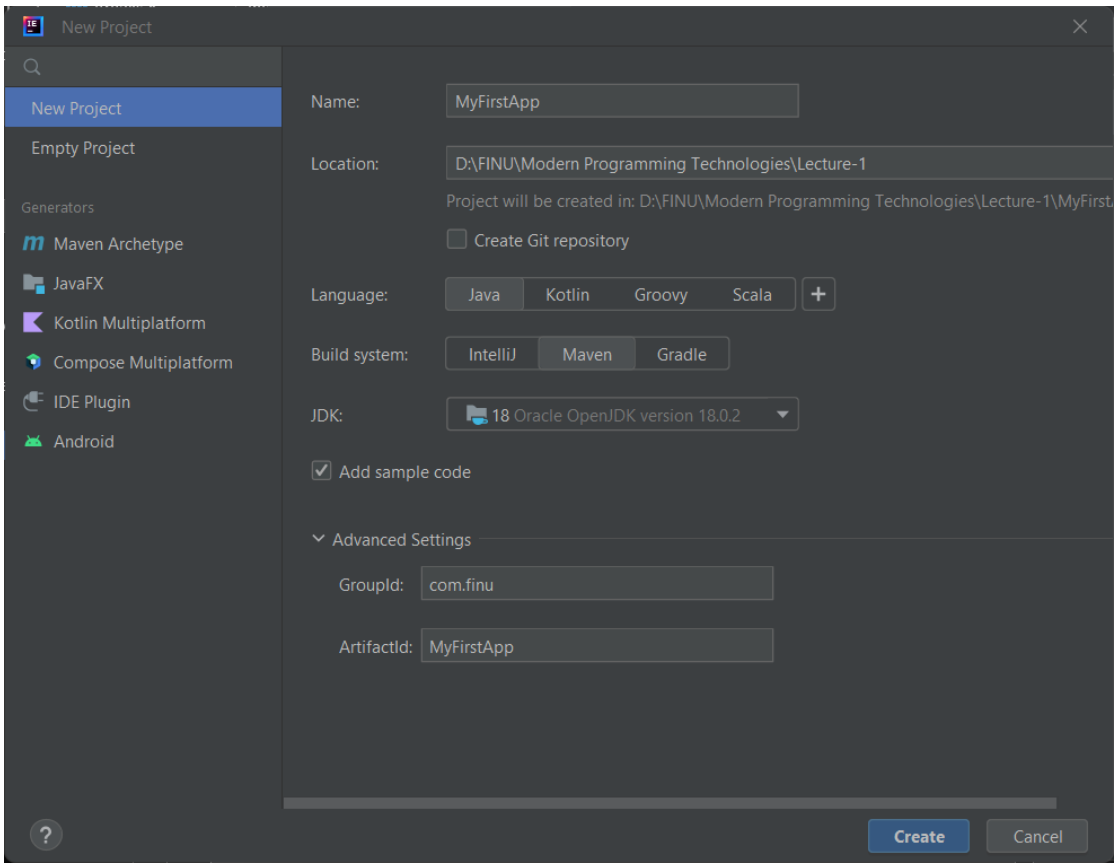
Первое Java-приложение

1-Запустить IntelliJ IDEA.

2-Нажать на кнопку (New Project).



3- Установить все настройки согласно картинке ниже.



2-Нажать на кнопку (Create).

3- Изменить название класса с (Main) на (MyFirstApp).

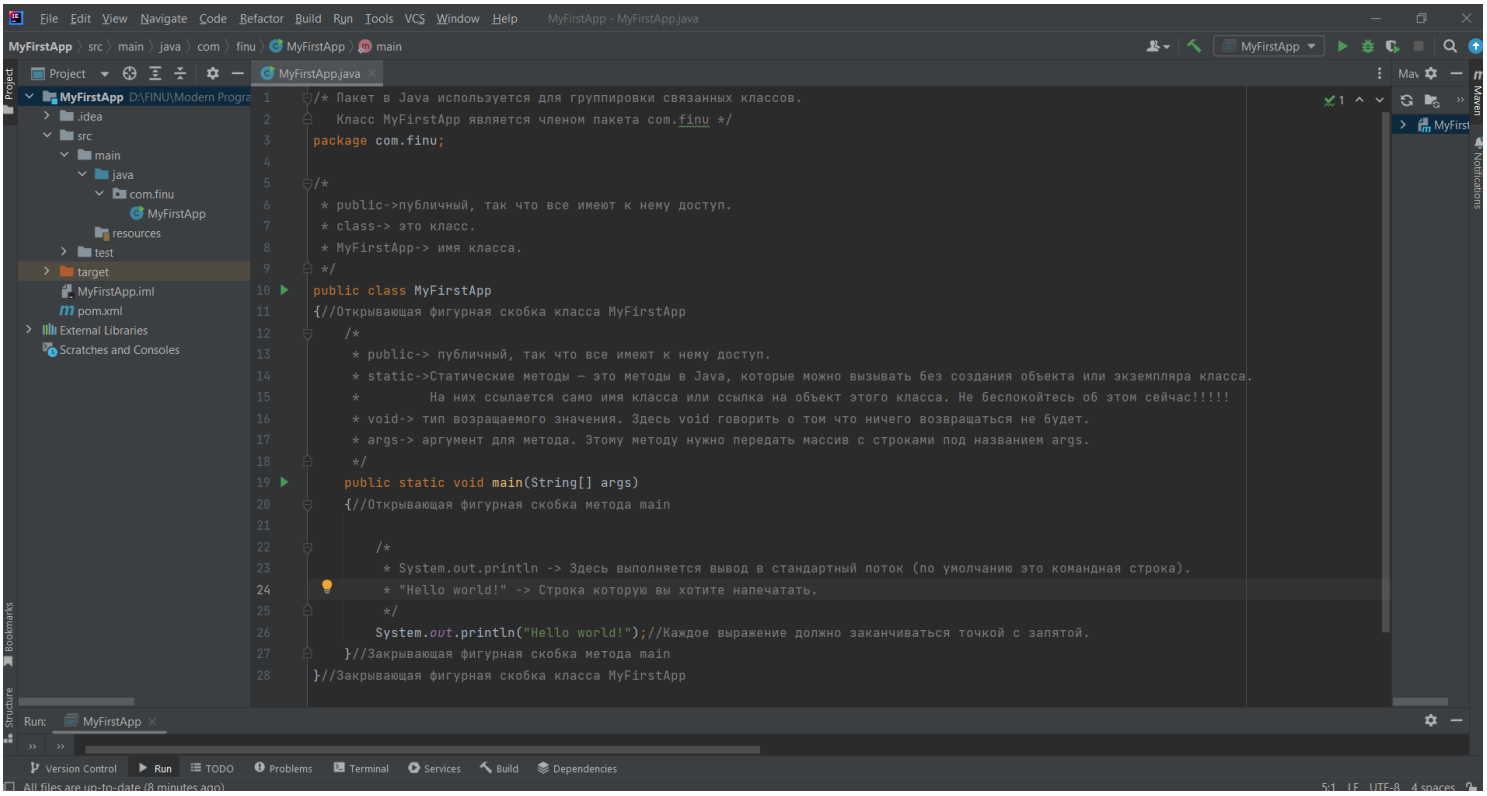
4- Изменить название файла с (Main.java) на (MyFirstApp.java).

5- Скопировать следующий код в файл (MyFirstApp.java).

```
/* Пакет в Java используется для группировки связанных классов.
  Класс MyFirstApp является членом пакета com.finu */
package com.finu;

/*
 * public->публичный, так что все имеют к нему доступ.
 * class-> это класс.
 * MyFirstApp-> имя класса.
 */
public class MyFirstApp
{//Открывающая фигурная скобка класса MyFirstApp
   /*
    * public-> публичный, так что все имеют к нему доступ.
    * static->Статические методы — это методы в Java, которые можно вызывать без создания объекта или экземпляра класса.
    *       На них ссылается само имя класса или ссылка на объект этого класса. Не беспокойтесь об этом сейчас!!!!!
    * void-> тип возращаемого значения. Здесь void говорить о том что ничего возвращаться не будет.
    * args-> аргумент для метода. Этому методу нужно передать массив с строками под названием args.
```

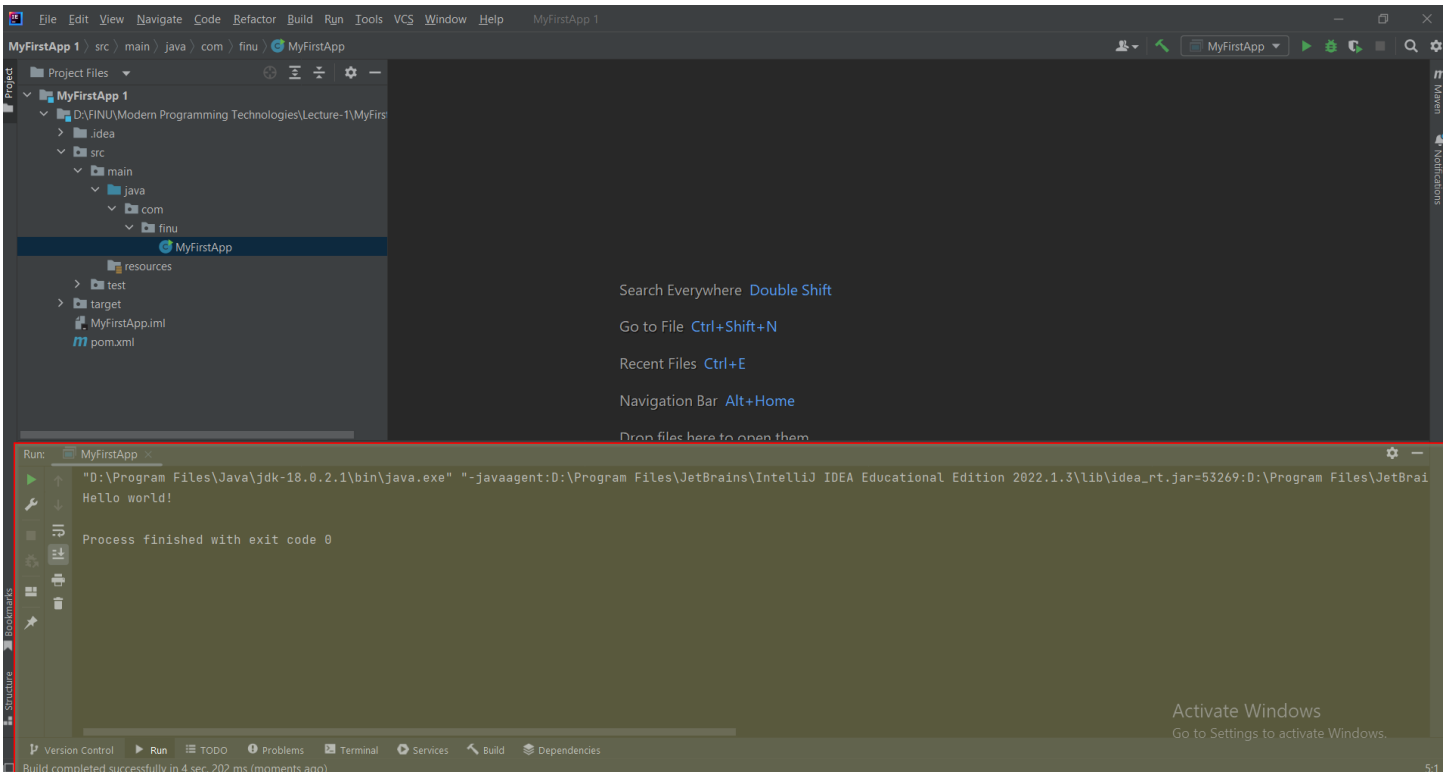
```
    */
   public static void main(String[] args)
   {//Открывающая фигурная скобка метода main

      /*
       * System.out.println -> Здесь выполняется вывод в стандартный поток (по умолчанию это командная строка).
       * "Hello world!" -> Строка которую вы хотите напечатать.
       */
      System.out.println("Hello world!");//Каждое выражение должно заканчиваться точкой с запятой.
   }//Закрывающая фигурная скобка метода main
}//Закрывающая фигурная скобка класса MyFirstApp
```

Окно приложения должно выглядеть, как на картинке ниже.



6- щелкнуть файл (MyFirstApp.java) правой кнопкой мыши и выбрать (Run 'MyFirstApp.main()'). В случае каких-либо ошибок попробовать Build->Rebuild Project.

7- Проверить результаты выполнения в окне (Run) в нижней части экрана.



Система сборки проектов Maven

Система сборки – это программное обеспечение, обеспечивающее автоматизацию сборки проекта. Инструменты сборки позволяют решать самые разные задачи автоматизации сборки, в том числе:

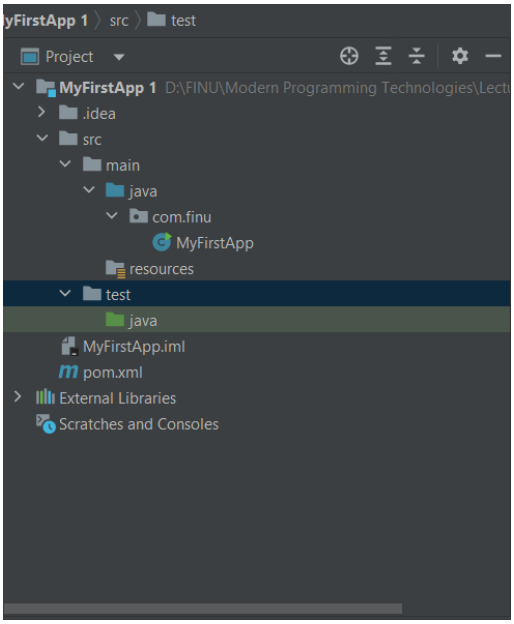
- Компиляция: компиляция исходного кода в машинный код
- Управление зависимостями: выявление и исправление необходимых сторонних интеграций
- Автоматические тесты: выполнение тестов и сообщение об ошибке
- Пакетирование приложения для развертывания: подготавливает исходный код для развертывания на серверах.

Maven-это инструмент сборки и менеджер проектов на основе XML. Maven — это проект Apache с открытым исходным кодом. Структура и содержание проекта Maven указывается в специальном xml-файле, который называется Project Object Model (POM), который является базовым модулем всей системы.

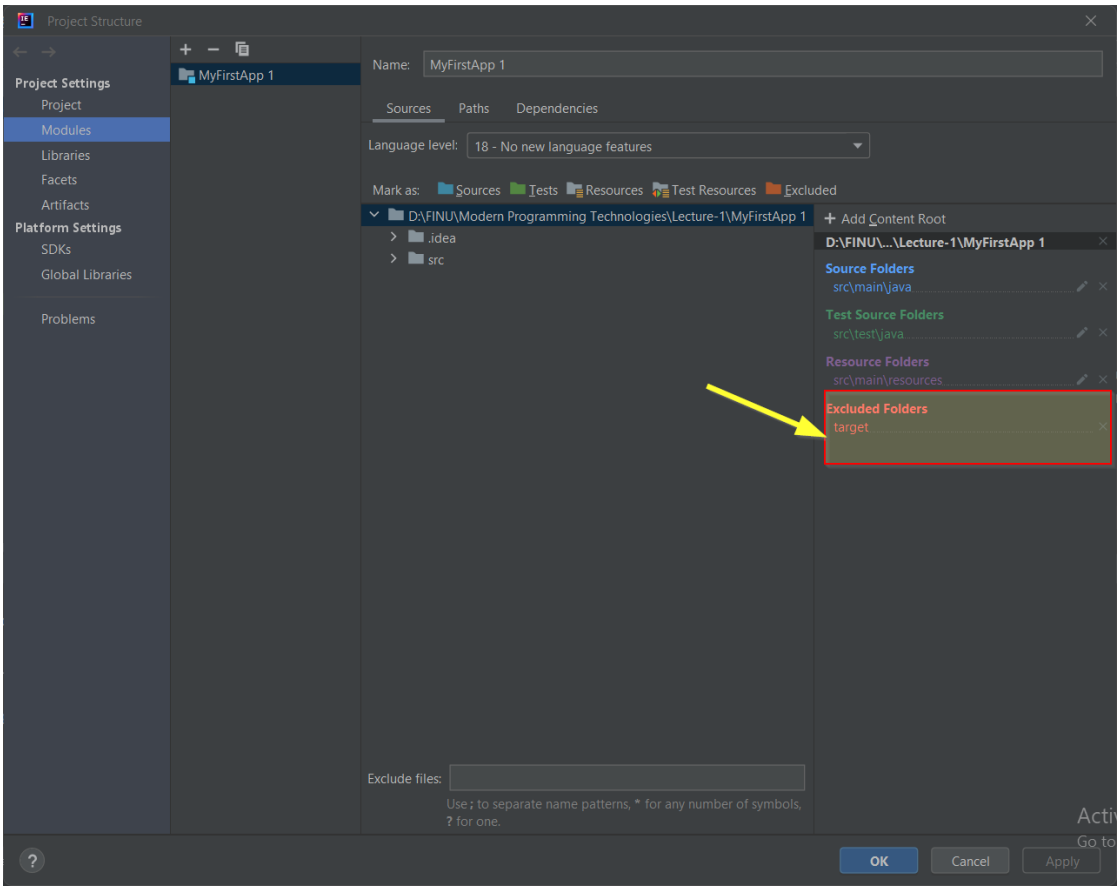
Стандартная структура проекта созданного на основе Maven:

Исходный код приложений или библиотек	src/main/java
Ресурсы для приложений или библиотек	src/main/resources
Тесты	src/test/java
Ресурсы для тестов	src/test/resources
Дистрибутив JAR	target
Скомпилированный байт-код	target/classes
Исходный код веб-приложений	src/main/webapp
Веб-сайт	src/site

Вернемся к нашему первому проекту в окне IntelliJ IDEA. Слева мы можем видеть текущую структуру проекта в.



Чтобы показать папку «target», нажимаем на название проекта правой кнопкой и выбираем «Open Module Settings» или просто нажимаем «F4». Появится окно, в котором нажмем «x» рядом с «target» в разделе «Excluded Folders». Наконец, нажимаем ОК.



Файлы POM

Открываем файл «pom.xml». POM (Project Object Model) является базовым модулем Maven. Это специальный XML-файл, который всегда хранится в базовой директории проекта и называется pom.xml

pom.xml (MyFirstApp)

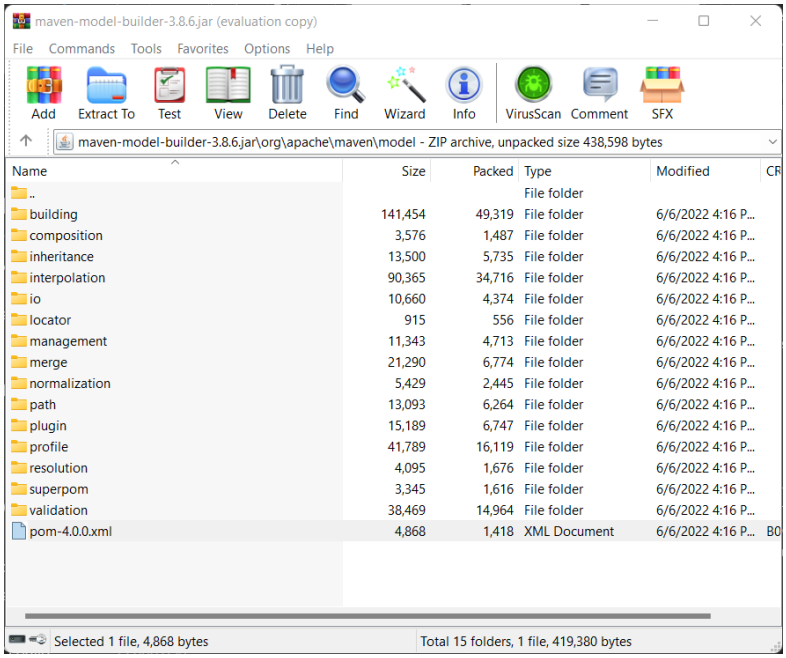
```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.finu</groupId>
    <artifactId>MyFirstApp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>18</maven.compiler.source>
        <maven.compiler.target>18</maven.compiler.target>
    </properties>

</project>
```

Корневой элемент в файле- это <project>. Каждому проекту иметь три обязательных элемента:

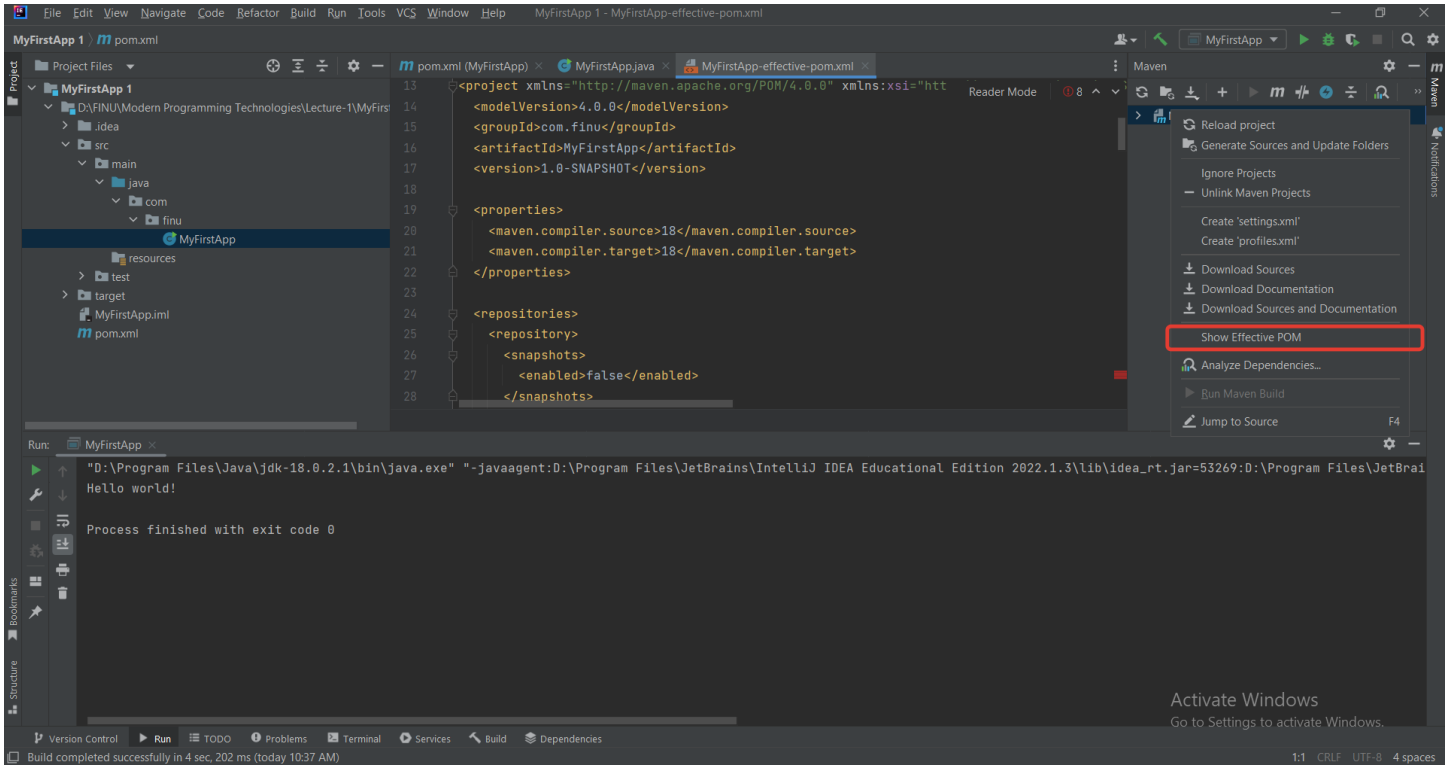
- groupId — уникальное базовое имя компании или группы, создавшей проект.
- ArtiftId — уникальное название проекта.
- version – версия проекта

В репозитории проект выглядит следующим образом: groupId:artifactId:version.

Все POM-файлы являются наследниками Super POM который содержит значения, унаследованные по умолчанию. В зависимости от версии Maven его можно найти в файле maven-x.y.z-uber.jar или maven-model-builder-xy.z.jar в %MAVEN_HOME%\lib. Если вы посмотрить в этот файл JAR, можно найдете файл с именем pom-4.0.0.xml в «org/apache/maven/model» . MAVEN_HOME — системная переменная среды, указывающая папку установки Maven. Можно просто перейти к этой папке, если мы уже ее знаем, затем открываем папку lib и используем программу, такую как WinRAR, чтобы открыть необходимый файл jar.



Существует также один тип файлов POM, который называется Effective POM. Effective POM сочетает в себе все настройки по умолчанию из Super POM и конфигурацию, определенную в POM нашего проекта. В IntelliJ IDEA этот файл можно показать, щелкнув правой кнопкой мыши по имени проекта в окне Maven и выбрав «Show Effective POM»



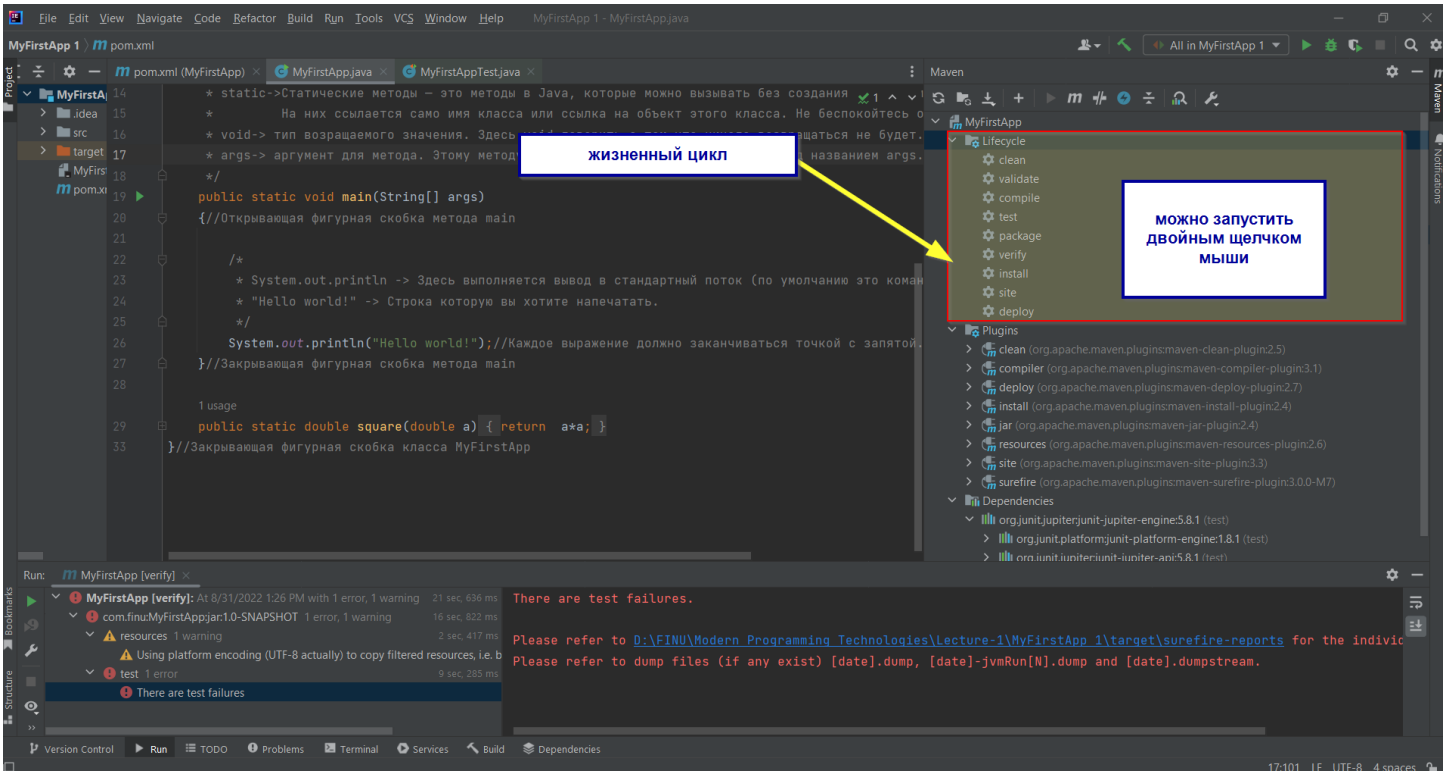
Основные понятия Maven

Жизненный цикл- Maven имеет три встроенных жизненных цикла сборки:

- default(По умолчанию)- жизненный цикл будет обрабатывать развертывание проекта. Он в основном включает следующие этапы или фази строительства
  - validate - подтвердить правильность проекта и наличие всей необходимой информации
  - compile - скомпилировать исходный код проекта
  - test — протестируйте скомпилированный исходный код, используя подходящую среду модульного тестирования. Эти тесты не должны требовать упаковки или развертывания кода.
  - package — возьмите скомпилированный код и упакуйте его в распространяемом формате, таком как JAR.
  - verify — запустить проверку результатов интеграционных тестов, чтобы убедиться, что критерии качества соблюдены.
  - install - установить пакет в локальный репозиторий для использования в качестве зависимости в других проектах локально
  - deploy — скопировать окончательный пакет в удаленный репозиторий для совместного использования с другими разработчиками и проектами.

  Эти фазы жизненного цикла (плюс другие фазы жизненного цикла, не показанные здесь) выполняются последовательно.

- clean(Очистка) Жизненный цикл отвечает за очистку проекта.
- site - Документация проекта генерации жизненного цикла (java-документация).



Цель (goal)- представляет собой конкретную задачу (точнее, чем этап или фаза сборки), которая способствует созданию проекта и управлению им. Он может быть привязан к нулю или нескольким фазам сборки. Цель, не привязанная к какой-либо фазе сборки, может быть выполнена вне жизненного цикла сборки путем прямого вызова. При использовании командной строки (чего мы не делаем в этом уроке) мы можем использовать следующий синтаксис:

```
mvn [имя-плагина]:[имя-цели]
```

Репозиторий (Repository) — это каталог (директория, папка), в котором хранятся артефакты, которыми Maven может воспользоваться (файлы jar, плагины…). Они могут быть:

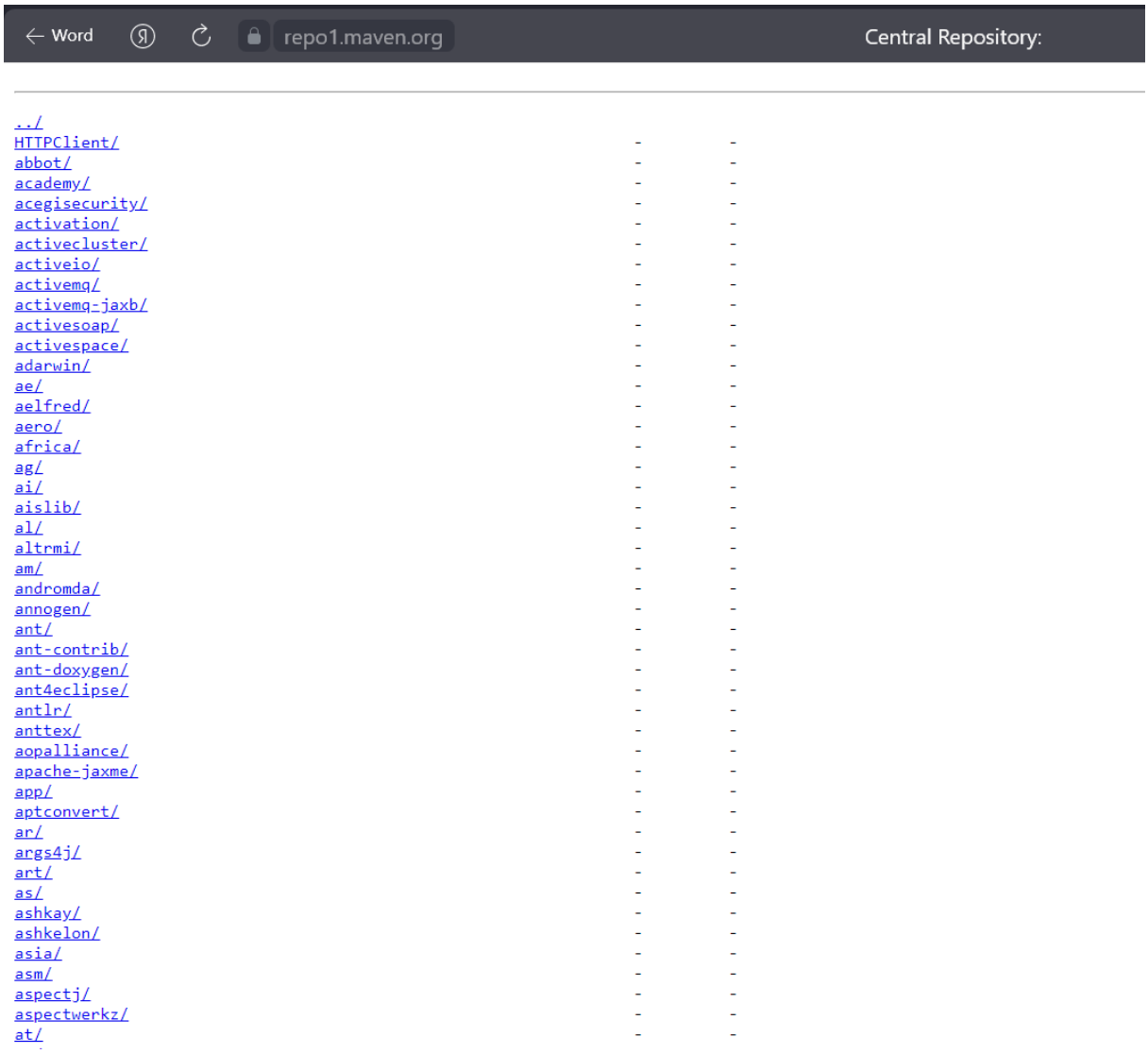
- Локальные (local): это директория, которая хранится на нашем компьютере. сборку проекта с помощью Maven, то все зависимости (их JAR-файлы) автоматически загружаются в локальный репозиторий. По умолчанию, локальный репозиторий создаётся Maven в директории %USER_HOME%.

Windows	C:\Users\<User_Name>\.m2
Linux	/home/<User_Name>/.m2
Mac	/Users/<user_name>/.m2

  Для того, чтобы изменить директорию нам необходимо указать её в файле %M2_HOME%\conf\ settings.xml следущим образом:

```
    <localRepository>/Path/To/Your/Local/Repo</localRepository>
```

- Центральные (central): это репозитоприий, который обеспечивается сообществом Maven (https://repo1.maven.org/maven2/).



- Удалённые (remote): Это репозиторий, который определяется самим разработчиком. Для того, чтобы настроить удалённый репозиторий, необходимо внести некоторые изменения в файл pom.xml.

```
<repositories>
 <repository>
     <id>finu.repo1</id>
     <url>http://url.to.repository/maven2/repo1</url>
 </repository>
 <repository>
     <id> finu.repo1</id>
     <url> http://url.to.repository/maven2/repo1</url>
 </repository>
</repositories>
```

Порядок поиска зависимостей Maven:

- Поиск зависимостей в локальном репозитории Если зависимости не обнаружены, происходит переход к шагу 2.
- Поиск зависимостей в центральном репозитории. Если они не обнаружены и удалённый репозиторий определён, то происходит переход к шагу 4.

- Если удалённый репозиторий не определён, то процесс сборки прекращается и выводится сообщение об ошибке.
- Поиск зависимостей на удалённом репозитории, если они найдены, то происходит их загрузка в локальный репозиторий, если нет – выводится сообщение об ошибке.

Плагины (plugins) -это способ расширить функциональность Maven. В Maven каждая задача, по сути своей, выполняется с помощью плагинов.Существует два типа плагинов в Maven:

- Плагины сборки: Выполняются в процессе сборки и должны быть конфигурированны внутри блока <build></build> файла pom.xml
- Плагины отчётов: Выполняются в процессе генерирования сайта и должны быть конфигурированны внутри блока <reporting></reporting> файла pom.xml.

Примеры плагинов в таблице ниже

surefire	Запускает тесты JUnit. Создаёт отчёты о тестировании.
javadoc	Собирает JAR файл текущего проекта.
compiler	Компилирует исходные Java файлы.
jar	Собирает JAR файл текущего проекта.
war	Собирает WAR файл текущего проекта.

Зависимости (dependencies) - в большинстве случаев проекту нужны дополнительные библиотеки, которым иногда необходимо включить в сборку. Поэтому в maven-проекте необходимо использовать зависимость dependency, устанавливаемые в файле pom.xml, где для каждого используемого в проекте артефакта необходимо указать :

- groupId - идентификатор производителя объекта.
- artifactId - идентификатор объекта. Обычно это имя создаваемого модуля или приложения.
- version - версия описываемого объекта. Для незавершенных проектов принято добавлять суффикс SNAPSHOT.

Классификатор classifier используется в тех случаях, когда деление артефакта по версиям является недостаточным. К примеру, определенная библиотека (артефакт) может быть использована только с определенной JDK (VM), либо разработана под windows или linux.

```
<dependencies>
  <dependency>
    <groupId>net.sf.json-lib</groupId>
    <artifactId>json-lib</artifactId>
    <version>2.4</version>
    <classifier>jdk15</classifier>
  </dependency>
<dependencies>
```

Область действия зависимости (scope):

Test -Мы используем эту область, чтобы указать, что зависимость не требуется во время выполнения приложения, а используется только в целях тестирования. Стандартный вариант использования этой области — добавление тестовой библиотеки, такой как JUnit, в наше приложение.

Compile - Это область по умолчанию, если не указана другая область. Зависимость, помеченная как compile будет доступна как для компиляции основного приложения и его тестов, так и на стадиях запуска основного приложения или тестов.

Provided - область действия зависимости provided аналогична compile, за исключением того, что артефакт используется на этапе компиляции и тестирования, а в сборку не включается. Предполагается, что среда исполнения (JDK или WEB-контейнер) предоставят данный артефакт во время выполнения программы. Наглядным примером подобных артефактов являются такие библиотеки, как hibernate или jsf, которые необходимы на этапе разработки приложения.

Runtime - область действия зависимости runtime не нужна для компиляции проекта и используется только на стадии выполнения приложения.

System - область действия зависимости system аналогична provided за исключением того, что содержащий зависимость артефакт указывается явно в виде абсолютного пути к файлу, определенному в теге systemPath. Обычно к таким артефактам относятся собственные наработки, и искать их в центральном репозитории, куда Вы его не размещали, не имеет смысла.

Import - It's only available for the dependency type pom. import indicates that this dependency should be replaced with all effective dependencies declared in its POM.

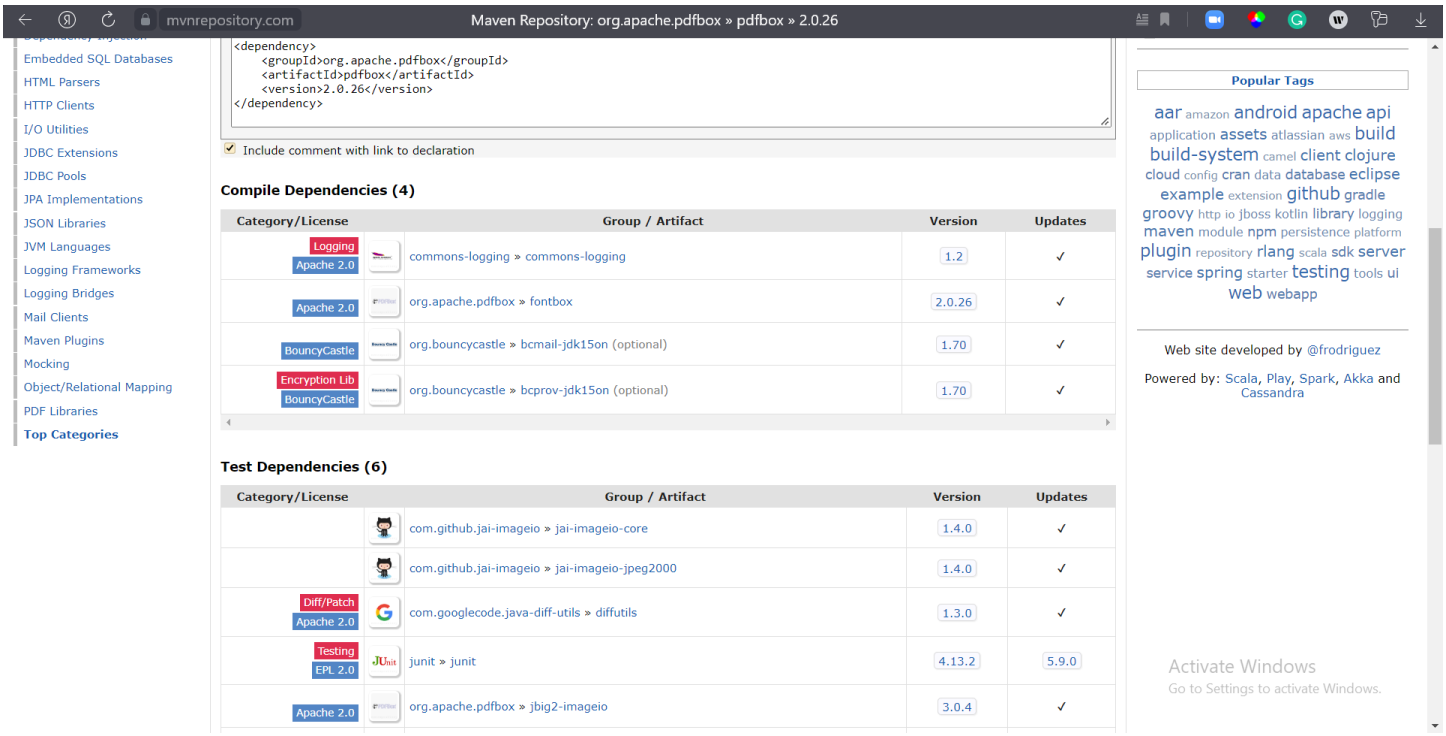
Транзитивные зависимости - позволяют избегать необходимости изучения и определения библиотек, которые требуются для самой зависимости. Maven включает их автоматически.

В следующей табличке, позаимствованной с сайта maven, представлен набор правил переноса области scope. К примеру, если scope артефакта «B» compile, а он, в свою очередь, подключает библиотеку «C» как provided, то наш проект «A» будет зависеть от «C» так как указано в ячейке находящейся на пересечении строки «compile» и столбца «provided».

	Compile	Provided	Runtime	Test
Compile	Compile	-	Runtime	-
Provided	Provided	Provided	Provided	-
Runtime	Runtime	-	Runtime	-
Test	Test	Test	Test	-

Давайте воспользуемся этой ссылкой, чтобы перейти в центральный репозиторий и посмотреть, какие зависимости нужны для библиотеки Apache PDFBox:

https://mvnrepository.com/artifact/org.apache.pdfbox/pdfbox/2.0.26



Профиль сборки (Profiles) – это множество настроек, которые могут быть использованы для установки или перезаписи стандартных значений сборки Maven. Используя профиль сборки Maven, мы можем настраивать сборку для различных окружений, таких как Разработка или Продакшн.

В Maven существует три основных типа профилей сборки:

- Per Project - Определяется в POM файле, pom.xml
- Per User - Определяется в настройках Maven – xml файл (%USER_HOME%/.m2/settings.xml)
- Global - Определяется в глобальных настройках – xml файл (%M2_HOME%/conf/settings.xml)

Профиль сборки Maven может быть активирован различными способами:

- Использованием команды в консоли
- С помощью настроек Maven
- С помощью переменных окружения
- Настройках ОС
- Существующими, отсутствующими файлами.