PROJECT DESCRIPTION

What is the POM?

POM stands for "Project Object Model". It is an XML representation of a Maven project held in a file named pom.xml.

Root element and title.

```
<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>
```

The root <project> element, a schema that makes editing and validation easier, and a POM version.

The project tag contains basic and required information about the project:

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

In Maven, each project is identified by a groupId artifactId pair. To avoid name conflicts, groupId is the name of an organization or department and usually the same rules apply as when naming packages in Java - they record the domain name of the organization or project site. artifactId is the name of the project. Inside the version tag, as you might guess, the version of the project is stored. The triple groupId, artifactId, version (hereinafter - GAV) can uniquely identify the jar file of the application or library. If the state of the code for the project is not fixed, then "-SNAPSHOT" is added to the version name at the end, which means that the version is under development and the resulting jar file may change. <packaging> ... </packaging> determines what type of file will be generated as a result of the build. Possible options are pom, jar, war, ear

Let's take a better look at the example of the project powermock-core groupId - org.powermock, artifactId - powermock-core, version - 1.4.6

It also adds information that is not used by the maven itself, but is needed for the programmer to understand what this project is about:

- <name> powermock-core </name> human project name
- <description> PowerMock core functionality. </description> Project Description
- <url> http://www.powermock.org </url> project site.

Dependencies

Dependencies - the next very important part of pom.xml - it contains a list of all libraries (dependencies) that are used in the project. Each library is identified as well as the project itself - by the triple groupId, artifactId, version (GAV). The dependency declaration is wrapped in the <dependencies> ... </dependencies> tag.

```
<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>
```

As you may have noticed, in addition to GAV, the <scope> tag may be present when describing a dependency. Scope specifies what the library is used for. This example says that the GAV junit: junit: 4.4 library is only needed to run tests.

The <build>

The <build> tag is optional as there are default values. This section contains information on the assembly itself: where are the source files, where are the resources, what plugins are used. For instance:

```
<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>
```

Let's take a closer look at this example.

- <sourceDirectory> defines where maven will get its source files from. By default, this is src / main / java, but you can define where it suits you. There can be only one directory (without using special plugins)
- <resources> and the nested <resource> tags define one or more directories where resource files are stored. Assets are simply copied during build, unlike source files. The default directory is src / main / resources
- <outputDirectory> defines in which directory the compiler will save the compilation results - * .class files. The default is target / classes
- <finalName> - the name of the resulting jar (war, ear ..) file with the corresponding extension type, which is created in the package phase. The default is artifactId-version.

Maven plugins allow you to specify additional actions to be performed during build. For example, in the above example, a plugin has been added that automatically checks the code for "bad" code and potential errors.

for more information on Maven, visit their official documentation located at http://maven.apache.org/pom.html