Create a Maven project

Project Description Object Model

One of the things that Maven standardized in the first place was the project description. Before Maven, each IDE had its own project file, which stored information about the project and its assembly (and often in binary form).

Maven has come up with an XML-based, universal, open standard that describes what a project is, how it should be built, and what dependencies it has using various tags. The project description is contained in a single file, usually named **pom.xml** .

An example pom.xml file :

```
<?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>example.com</groupId>
   <artifactId>example</artifactId>
   <version>1.0-SNAPSHOT</version>

   <dependencies>
       <dependency>
           <groupId>commons-io </groupId>
           <artifactId>commons-io</artifactId>
        <version>2.6</version>
        </dependency>
   </dependencies>

</project>
```

This example has three things written:

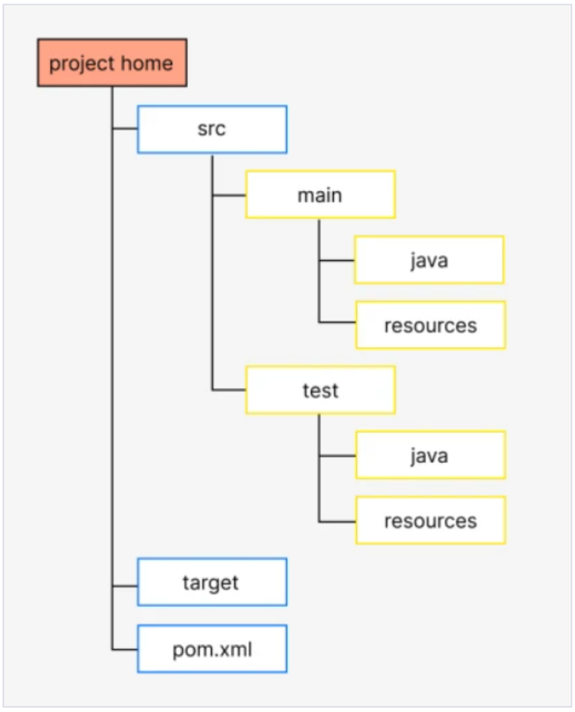
- Information about the version of the maven project standard is blue.
- Information about the project itself is in red.
- Information about used libraries is green.

Let's take a closer look at the pom file device.

Maven project structure

And immediately the question is: did you pay attention to the strangeness in the last example? It does not contain information about the project code itself! There is no word on where java files, resources, properties files, html, build scripts and the like are stored.

And the answer is simple - Maven standardized the design of the project. There are several options for organizing code within a project, and the most common is:



The structure is a little unusual after the standard IDEA projects, but for that it is universal. 90% of the projects you will encounter in your life will have this folder structure .

If you create a Maven project (using IDEA or using the console), then it will take the specified form. Let's see how it all works here.

The src folder , you guessed it, contains the source code for the project. It has two subfolders: **main** and **test** .

The /src/main/java folder is the root for all Java classes in the project. If you have a com.codegym.Cat class, then it will be in the /src/main/java /com/codegym /Cat.java folder . If there are text or binary resources, they should be stored in the /src/main/resources folder .

The structure of the **/src/test** folder is similar to the structure of the **/src/main** folder , but it contains the tests and their resources. Maven itself knows how to run the necessary tests when building a project, but we'll talk about this in a separate lecture.

There is also a /target folder in the project , where Maven will save the project after it is built. Since large projects often have non-trivial build scripts, nothing is stored in this folder.

The second purpose of the **/target** folder is to cache intermediate build results. When building a large project, Maven can only rebuild that part of it that has changed, thus speeding up the build time by several times.

Well, as a cherry on the cake - at the very root of the project is the pom.xml file. It contains all the necessary information about the project, which we will discuss below.

Device pom.xml

To start with, the pom file is xml, so it contains the standard headers and namespaces information. This is all about the purely XML standard, so we won't talk about it in detail. It means this:

```
<?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">


        …

</project>
```

Also, usually the first line inside the <project> tag is a description of the version of the pom-file standard. Almost always it is 4.0. This, too, is of no interest to us.

The first lines we are interested in look like this:

```
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.sample.app</groupId>
  <artifactId>new-app</artifactId>
  <version>1.0-SNAPSHOT</version>
```

In order not to understand once again what we describe (program, project, module, library, etc.) in the Maven standard, this is all called the word **artifact** . What you can’t refuse the creators of Maven is the love of standardization.

The three tags you see mean:

- **groupId** - the package to which the application belongs, with the addition of the domain name;
- **artifactId** – unique string key (project id);
- **version - version of the project.**

These three parameters are enough to unambiguously describe any artifact .

Further, after the description of the project, there is usually a list of artifacts (libraries) that the project uses. It looks something like this:

```
<dependencies>

    <dependency>
        <groupId>commons-io</groupId>
        <artifactId>commons-io</artifactId>
        <version>2.6</version>
    </dependency>

</dependencies>
```

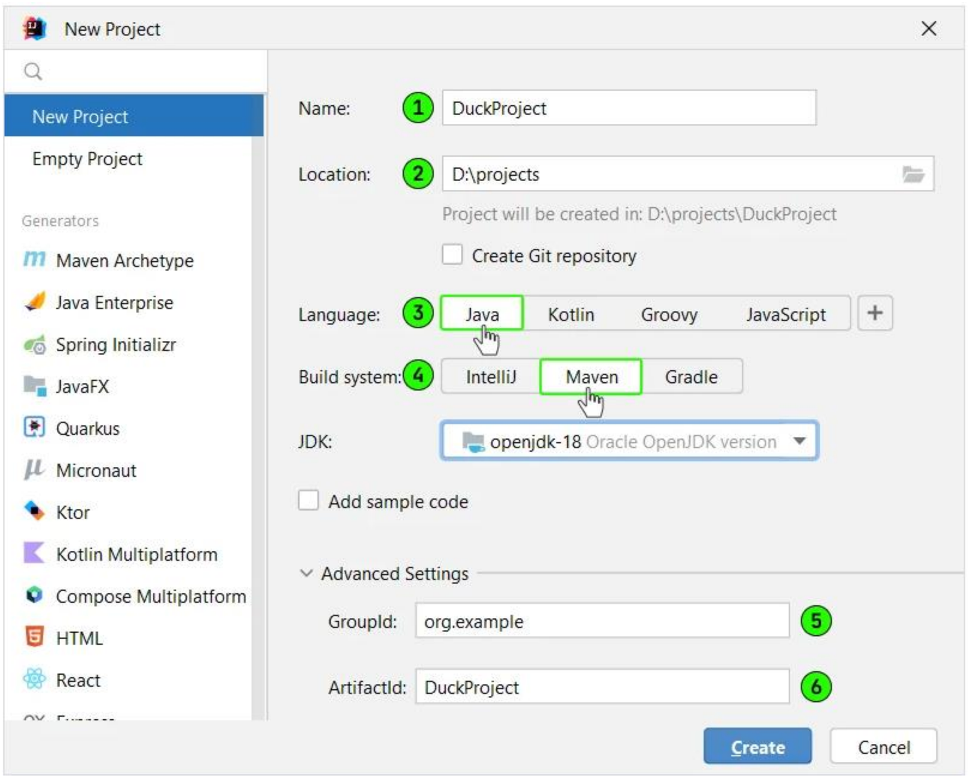
How IDEA works with Maven

Intellij IDEA is great at working with Maven. She knows how to open such projects, create them herself, run various build scripts, and perfectly understands the included libraries.

It even has its own built-in Maven for some time, but you still need to be able to install and configure it yourself, so this feature of IDEA was not mentioned earlier. In theory, IDEA can have a conflict between two Mavens, so it's good for you to know that there are two.

How to create a new Maven project in IDEA:

Click the menu Files > New Project. Select the menu item **New Project** on the left .

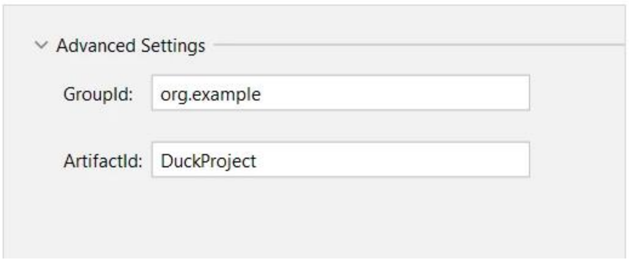


Name: DuckProject
Location: D:\projects
Project will be created in: D:\projects\DuckProject
Create Git repository
Language: [✓ Java] Kotlin Groovy JavaScript
Build system: IntelliJ [✓ Maven] Gradle
JDK: openjdk-18 Oracle OpenJDK version
Add sample code
Advanced Settings
GroupId: org.example
ArtifactId: DuckProject

Let's clarify some points:

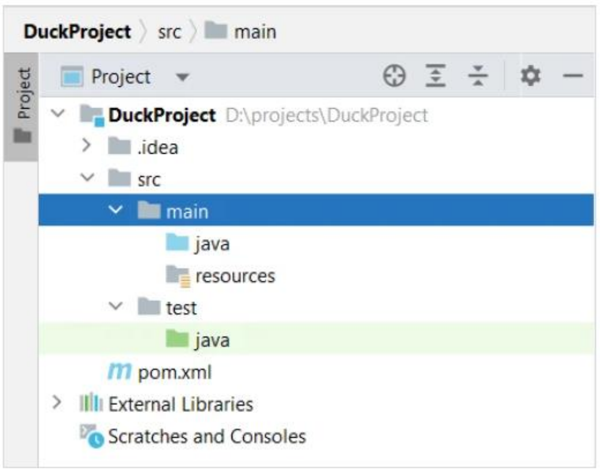
1. Project name;
2. Folder for the project;
3. The project language is Java;
4. The project type is Maven.

In the Advanced Settings section at the bottom, IDEA will prompt you to specify the goupID, artifactID, and version of our new project. This data can always be easily changed later. Choose from the suggested ones or enter your own:



Advanced Settings
GroupId: org.example
ArtifactId: DuckProject

Next, standardly create a project in the required location. As a result, we see the structure:



Classes and packages must be created in the java folder, we have already talked about this. And I think you can easily handle it. We're ready to move on, but let's go back a little, to one important issue that we "skipped" a bit.