Hello Maven

TestNG, Eclipse, IntelliJ IDEA

Óbuda University, Java Enterprise Edition
John von Neumann Faculty of Informatics
Lab 2

Dávid Bedők
2017.09.19.
v0.1

Java project structure

How to organize our source codes?

▷ **javac**

▷ **IDE** (Eclipse, IntelliJ IDEA, ...)

▷ **build tool**

Java Build tools:

▷ batch files / bash scripts

▷ Apache ANT (+ Apache IVY)

▷ Apache Maven

▷ Gradle

What kinds of things should be considered?

▷ There will be **unit tests**?
- separate `source folder` for the unit tests (**test** and **main**)

▷ There will be **resources**?
- separate `source folder` for the resources (**resources**)

▷ There will be **non-java source codes** too?
- separate `source folders` for the disparate sources (**java**, **scala**, **groovy**, etc.)

Java philosophy

▷ transparency, clarity

▷ **classpath** (cp) usage

▷ usage of JAR, WAR, EAR, SAR*, APK** packagings

SAR*: JBoss specific
APK**: android specific

Java project structure

javac

We can do almost 'everything' with that (there are no rules)! We add all the source folders to the javac program via CLI.

Directory structure

```
bin/
src1/ → source folder
src2/ → source folder
```

The Application class uses an instance of ImperialToMetricCalculator class. Both classes are compiled into the same package, so application doesn't import the calculator class (at runtime both classes will be at the same place).

\hellotest

```
> javac -d ./bin ./src1/hu/qwaevisz/demo/Application.java
    ./src2/hu/qwaevisz/demo/ImperialToMetricCalculator.java
> java -cp ./bin hu.qwaevisz.demo.Application
```

Java project structure

Eclipse IDE

Configuration of the Eclipse's default 'Java project':

Directory structure

```
bin/
src/ → source folder
```

Eclipse's 'Java project' with unit tests:

Directory structure

```
bin/
src/
    main/ → source folder
    test/ → source folder
```

In most of the IDEs these rules can be configure completely (Eclipse: `Project properties` | `Java Build Path` | `Source tab`).

Maven's default directory configuration:

Directory structure

```
src/
   main/
      java/ → source folder
      resources/ → source folder
   test/
      java/ → source folder
      resources/ → source folder
```

Of course you are able to change these settings in Maven. But if we use this default configuration we will have a very simple, clean and small* build script to start the work (*: not as small as the same build script in *Gradle*).

Note: The `resources` directories are part of the classpath as any other source directories. We can reach its content during runtime.

JAR - Java ARchive

ZIP format which keeps (Java) `byte codes` (`*.class`), configuration files (e.g.: `*.properties`, `*.xml`, etc.) and a special metafile which contains key-value pairs (**MANIFEST.MF**).

Directory structure

```
META-INF/
   MANIFEST.MF
hu/
   qwaevisz/
      demo/
         HelloWorld.class
         Lorem.class
log4j.xml
```

It's structure is predefined, there is an option to store source files (e.g.: `*.java`, `*.groovy`, etc.) at the same place where the `byte codes` are located.

```
Manifest-Version: 1.0
Created-By: 1.7.0_67 (Oracle Corporation)
```

MANIFEST.MF

Executable JAR file

The Main-Class key has to be part of the MANIFEST.MF file, and the value of this key is the full qualified name of the entry point of the application.

```
Manifest-Version: 1.0
Created-By: 1.7.0_67 (Oracle Corporation)
Main-Class: hu.qwaevisz.demo.Application
```

MANIFEST.MF

```
> cd bin
> jar cvfe calculator.jar hu.qwaevisz.demo.Application
    hu/qwaevisz/demo/Application.class
    hu/qwaevisz/demo/ImperialToMetricCalculator.class
> cd ..
> java -jar bin/calculator.jar
```

create new archive
verbose
specify archive **f**ile name (2)
sepcify **e**ntry point (main class) (3)

Eclipse OXYGEN

IDE, 2017 June

Eclipse IDE for Java EE Developers

Download: https://www.eclipse.org/downloads/

Version: **4.7.0**

Install: unzip or installer

Integrated plugins:

- ▷ Gradle
- ▷ Maven
- ▷ Git
- ▷ EclEmma Java Code Coverage
- ▷ ...

In case of Hungarian keyboard layout (and usage) you have to turn off some shortcut keys (e.g. "{" (Ctrl + B): `Preferences | General | Keys | Skip all brakepoints` (Ctrl + Alt + B) → Unbind

Basic usage of Eclipse IDE: http://users.nik.uni-obuda.hu/bedok.david/jse.html

Additional plugins (`Help / Eclipse Marketplace`):

- ▷ **TestNG** (filter: `testng`)
  - http://beust.com/eclipse

Eclipse configuration

Code Style Formatter

`Window` | `Preferences` (type: `formatter`)

- ▷ `Java` | `Code Style` | `Formatter`
  - New... / Import...: **uni-obuda-java-formatter**
    - Initialize: Eclipse [build-in]
    - Indentation | Indent | Statement within ‘switch’ body
    - Line Wrapping | General | Maximum line width: 160
    - Line Wrapping | Enum declaration
      - Policy: Wrap all elements, every element on a new line
      - Constants policy: Wrap all elements, every element on a new line + Force split
    - Comments | Line width | Maximum: 120

```
\eclipse\uni-obuda-java-formatter.xml
```

Eclipse configuration
Save Actions

`Window | Preferences` (type: `save actions`)

▷ `Java | Editor | Save Actions`

- Perform the selected actions on save
  - **Format source code** (all lines)
  - Organize imports
  - Additional actions - Configure
    * Code Organaizing: Remove trailing whitespaces
    * Code Style: Use blocks in if/while/for/do statements
    * Member Accesses: Use ‘this’ qualifier for field accesses: Always
    * Member Accesses: Use ‘this’ qualifier for method accesses: Always
    * Unnecessary Code: Remove unused imports

IntelliJ IDEA
IDE

Download: https://www.jetbrains.com/idea/

- ▷ **Commercial product**
- ▷ The community version doesn't support the JavaEE, but without this feature it is usable for professional work as well (event JavaEE projects).
- ▷ Sometimes it is faster than Eclipse
- ▷ Different shortcut keys, hard to get used to
- ▷ Integrated Maven/Gradle/Git plugins

Hello World

src | main | java | hu | qwaevisz | hello | Application.java

```java
package hu.qwaevisz.hello;

public class Application {

  public static void main(final String[] args) {
    System.out.println("Hello World");
  }

  public int add(final int a, final int b) {
    return a + b;
  }

}
```

Application.java

[gradle|maven]\helloworld

TestNG

3rd party library

- http://testng.org/
- GitHub: https://github.com/cbeust/testng
- Version: **6.11**
- Artifactory URL:
  - `'org.testng:testng:6.11'`
  - group/groupId: `org.testng`
  - name/artifactId: `testng`
  - version: `6.11`

Unit Test with TestNG

src | test | java | hu | qwaevisz | hello | ApplicationTest.java

```java
package hu.qwaevisz.hello;

import org.testng.Assert;
import org.testng.annotations.Test;

public class ApplicationTest {

  @Test
  public void addNumbers() {
    Application app = new Application();
    Assert.assertEquals(app.add(2, 3), 5);
  }

}
```

ApplicationTest.java

Apache Maven

Build tool

- https://maven.apache.org/
- Download : https://maven.apache.org/download.cgi
- Version : **3.5.0**
- Apache Maven is a software project management and comprehension tool.
- supports monorepo and multi-repo as well
- POM : Project Object Model
- Install : unzip

Environment Variables :

- **MAVEN_HOME** →`c:\apps\apache-maven-3.3.9`
- **Path** módosítása → `%Path%;%MAVEN_HOME%\bin`

Maven
Major properties

Maven's primary goal is to allow a developer to comprehend the complete state of a development effort in the shortest period of time.

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Providing guidelines for best practices development
- Allowing transparent migration to new features

Maven is - at its heart - a **plugin execution framework**, all work is done by plugins.

Maven

```
> mvn --version
Apache Maven 3.3.9 (bb52d8502b132ec0a5a3f4c09453c07478323dc5;
    2015-11-10T17:41:47+01:00)
Maven home: c:\apps\apache-maven-3.3.9\bin\..
Java version: 1.8.0_102, vendor: Oracle Corporation
Java home: c:\apps\java\jdk1.8.0_102\jre
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 7", version: "6.1", arch: "amd64", family: "dos"
```

Maven

Phases

validate	validate the project is correct and all necessary information is available
compile	compile the source code of the project
test	test the compiled source code using a suitable **unit testing** framework
package	take the compiled code and package it in its distributable format, such as a `jar`
verify	run any checks on results of **integration tests** to ensure quality criteria are met
install	install the package into the local repository, for use as a dependency in other projects locally
deploy	done in the build environment, copies the final package to the remote repository for sharing with other developers and projects
clean	cleans up *artifacts* created by prior builds
site	generates site documentation for this project

Phases are actually mapped to underlying **goals** (e.g.: for example, package executes `jar:jar` if the project type is a `jar`, and `war:war` if the project type is a `jar`).

Maven

Archetype

With the help of **archetypes** we could generate **blueprint** projects which we should do it anyway (if we follow best practices).
Without these blueprints, we have to type a lot even for a simple 'hello world' project.

But: we will ignore the archetypes entirely in the future...

Create HelloWorld with the help of archetypes

maven\helloworld

```
> mvn archetype:generate -DgroupId=hu.qwaevisz.hello
   -DartifactId=hellomaven
   -DarchetypeArtifactId=maven-archetype-quickstart
   -DinteractiveMode=false
```

- **archetype** → Maven plugin
- **generate** → goal (belongs to the plugin)

The project structure and the `pom.xml` are created.

Hello Maven!

```
<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/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>hu.qwaevisz.hello</groupId>
  <artifactId>hellomaven</artifactId>
  <packaging>jar</packaging>
  <version>1.0</version>
  <name>Hello Maven</name>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncodin
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.testng</groupId>
      <artifactId>testng</artifactId>
      <version>6.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

<packaging>jar</packaging> $\rightarrow$ The output of the project will be a JAR file.

Maven

Compile and run

```
> mvn clean package
```

Output: target/hellomaven−1.0.jar

```
> java -cp target/hellomaven-1.0.jar
   hu.qwaevisz.hello.Application
```

Maven Eclipse integration

- Eclipse Eclipse **m2e plugin** recognizes the configuration files of maven and create/modify the eclipse project configuration files according to that.
- There is an **Eclipse plugin** for Maven which produces the Eclipse configuration files, but this plugin has been deprecated already.

Eclipse Maven project

`File` | `Import...` | `Maven` | `Existing Maven Project`

- ▷ Project root directory: `\helloworld`

Where do I execute Maven's goals?

The main goal of the `m2e plugin` is to geneate and handle the Maven's project structure. Run and manage the Maven's tasks from Eclipse is an other thing.. (execute Maven tasks from command line is a very comfortable and IDE independent (!) way of working process).