HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

HBV202G: Software Design and Construction · Spring 2025
Andri Valur Guðjohnsen · Dr. Helmut Neukirchen

Assignment 4 · Due Tuesday 18.2.2025 / Friday 21.2.2025

Formalities:

- Submit your source code solution as upload from GitHub as a team via Gradescope. Remember to select your other team member in one of the first submission steps.

Objectives: Learn using Maven.
And do this in the Maven assignment.

1. In your IDE[1], create a new Java project using version 1.4 of the Maven archetype `maven-archetype-quickstart` from group `org.apache.maven.archetypes` at Maven Central. If you cannot select version 1.4, then you are not using Maven Central, but some internal or outdated catalog: check what your IDE uses (e.g. in IntelliJ, when creating a new project via a Maven archetype, "Internal" might be selected as "Catalog").

   If your IDE supports already at that step to create a local Git repository, enable this.

   Use as *groupId* your HÍ email address as reverse domain name and add `hbv202g.ass4`, e.g. `abc12@hi.is` would use as package name: `is.hi.abc12.hbv202g.ass4`. As *artifactId*, use `ass4`.

   Investigate the created project structure: it should be different to the normal structure used by your IDE and rather follow the Maven project layout, e.g. the Java source code is in directory `src/main/java`.[2]

2. If your IDE did not create it automatically for you, create a useful `.gitignore` file (in particular, exclude the `target` directory, see Slide 4-21) and share this newly generated project to GitHub in order to collaborate within your team, i.e.:

   - If your IDE has support for GitHub (not just for Git in general), your IDE should support to publish or share your local project to GitHub where a new repository will then get created. In GitHub, add your teammate as collaborator (→*Settings*).

3. Via your IDE, make Maven execute a compile: it will probably report a warning or a failure (see slides 4-26 and 4-27): Therefore, in the properties section of the `pom.xml`, update the compiler `source` and `target` version and add a `release` version to be used. In addition, check for any `FIXME` in the POM and fix that (i.e. delete or update entry), so that your submission does not contain anything to be fixed.
   Use a `mvn` command to check for available updates for your dependencies and then update them.

4. Add the `exec-maven-plugin` from *groupId* `org.codehaus.mojo` in version `3.5.0` as build plugin to your POM (do not add it inside `pluginManagement`, but in parallel to it) and configure there as `mainClass` the fully qualified name of the class containing your `main` method (see Slide 4-34).

[1]On Slide 4-42, check the web pages with the Maven tutorial for your IDE to see how you create in your IDE a project from an archetype.
[2]In this assignment, we ignore the dummy tests that the archetype created in directory `src/test/java`.

Check that you can now execute your `main` method both using the normal run functionality of your IDE (i.e. without Maven[3]) and using `mvn exec:java`.

5. We now want to experience, how easy it is to add an external dependency, namely to add a library from the Apache Commons project that calculates prime numbers. To find out, what you need to add to your POM in order to make use of that library, see:
   `https://commons.apache.org/proper/commons-numbers/commons-numbers-primes/dependency-info.html`
   Add this to your POM and check that `mvn compile` still works.

6. Now, make use of that prime number library, i.e.
   - import in your main class the class `Primes` that contains that library. Use the fully qualified package name and the class name in your `import` statement (you find this information in the web page linked below).
   - instead of printing out `Hello World!`, let the code print out the next prime number that comes after the number 123456789.

   For the name of the class and package, as well as the available methods[4], see:
   `https://commons.apache.org/proper/commons-numbers/commons-numbers-primes/apidocs/org/apache/commons/numbers/primes/Primes.html`
   Check that your changed code compiles and runs correctly.

7. Now, we want to create a jar and run it from command line:

   Create a file `createjar.cmd` in the root of your project directory that calls `mvn package` (see Slide 4-29) and run that file in a command line terminal.[5] Check that a jar file has been created in directory `target`.

   Create another file `runjar.cmd` to run your jar file – however, if you try to use the `java -cp` command as shown on Slide 4-29 (adjust the classpath according to the name of your jar in directory `target` and the classname according to your package name), you will notice that class `org.apache.commons.numbers.primes.Primes` is not found by the JVM: The reason is that the generated jar does only contain your own code, but not your dependencies.

   To solve this problem, make Maven copy the `commons-numbers-primes` jar file into sub-directory `dependency` in your project's `target` directory by adding to your `createjar.cmd` file a line that copies the dependencies (using the `dependency:copy-dependencies` goal from the `install` phase, see Slide 4-41:).
   Run your `createjar.cmd` file and check that this jar has been copied into your project.

   Add the `commons-numbers-primes` jar file (including the directory path leading to it, i.e. `dependency` inside `target`) to the classpath parameter `-cp` in your `runjar.cmd` file.
   - Hint: In order to have multiple entries in a classpath, see also Chapter 3, Slide 3-27 – MS Windows users using a POSIX shell, e.g. in their IDE's terminal window: see also Slide 3-27.

   Check that `runjar.cmd` works now.

8. Take care that your newly added `*.cmd` files have been pushed together with all other changes to your remote repository and submit your GitHub project to Gradescope.

9. If your GitHub repository is not private: on the GitHub project web page, change it to private: *Settings* → scroll down to *Danger Zone* → *Change visibility*.

[3]Some IntelliJ users get here an error if Maven compiled with a newer Java version setting than IntelliJ assumes: try everywhere in the *Project Structure...* the settings to set the SDK version to the same Java version that is set in the POM.

[4]Note that these are *static* methods, i.e. you call them on the *class*, not on an *object*.

[5]If the `mvn` command is not found on the command line, double check that you have Maven installed (e.g., IntelliJ has a built-in Maven that you cannot use from the command line).