HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild

HBV202G: Software Design and Construction · Spring 2025

Andri Valur Guðjohnsen · Dr. Helmut Neukirchen

Assignment 3 · Due Tuesday 11.2.2025 / Friday 14.2.2025

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 `javac`*,* `java`*,* `jar` *on the commandline, learn using Java packages.* In the following, create command line commands to compile and run a Java project provided in GitHub. That Java project contains empty script files `compile.cmd`, `run.cmd`, `createjar.cmd`, and `runjar.cmd` where you need to add the respective calls to `javac`, `java`, and `jar`. For the latter, you find also a `mymanifest.mf` where you will need to add lines.

1. Go to the web page: `https://github.com/helmutneukirchen/HBV202GAssignment3` Instead of forking, you should find in the upper right part a green button *Use this template*: click on it, then → *Create a new repository.* It does not matter whether you select to *include all branches* – there anyway is only the default branch *main.* Select *Private.* → *Create repository* to create your own copy of the repository template. If you have a team mate, you can later use *Settings* to add a *Collaborator.*

2. *Warm-up:* Create from your newly imported GitHub project a project in your IDE (as your IDE might expect another project layout, you might not be able to compile or run the Java files from your IDE or the IDE might even complain – ignore this: we are anyway using the command line and use the IDE only as editor and Git client).

   Update the files `compile.cmd` and `run.cmd` as needed to compile (i.e. add a `javac` command) and run the Java project (i.e. a `java` command). Use *relative pathnames* (assume your current working directory is the directory that contains these `cmd` files).

   Check that `compile.cmd` and `run.cmd` work by calling them from the command line[1] and commit and push your change to your repository.

3. Change the project structure by moving the source code into a new directory `src` and the to be generated byte code into a new directory `bin`.
   - Either your IDE allows to create new directories (**but do not let your IDE create a new package with name `src`**)
   - or (e.g. if your IDE support only to create a package, but not a simple directory) do this simply outside your IDE (if the IDE does not notice these changes, you may need to do some IDE reload/refresh command to see the changes there). `git mv` can be used to move your existing files under version control to a new location.

   Create in the `bin` directory a `.gitignore` file with the following contents:

```
# Ignore everything in this directory (prevent committing generated files)
*
# Except this file
!.gitignore
```

[1] On the MS Windows command line terminal, you should be able to run these by simply typing in the filename. Those who use a POSIX shell, i.e. Mac OS or Linux, but also on Windows, VS Code and IntelliJ have the *zsh* POSIX shell in the IDE terminal: use `source`, e.g.: `source compile.cmd`

Adjust your files `compile.cmd` and `run.cmd` so that they use the new directory structure: check that they still compile and run the code when you call them from the command line.

Your IDE might be confused by your directory structure and, e.g., think that `src` is a package and therefore complain that your classes are not in package `src` and suggest to a line `package src` – ignore this!

Put the new project structure under version control. Use `git status` to check for any changes that might still need to be added to git: commit and push your changes.

If your IDE is confused by your directory structure, you can now delete now your project from the file system and clone it again from GitHub: the IDE should now understand the new project structure.

4. Introduce Java packages: While class `Main` should stay in the default package (so that it can still be used without any package name prefix), the other classes shall be moved as follows: Class `UserInterface` belongs into a frontend package and the two classes `Language` and `Greeter` into a backend package. For the package names, use your HÍ email address as reverse domain name and append `hbv202g.ass3.backend` and `hbv202g.ass3.frontend`, e.g. `abc12@hi.is` would use as package name: `is.hi.abc12.hbv202g.ass3.frontend`.

   Add a suitable `package` statement at the start of each class and add any needed `import` statement. Move the source code files accordingly into the a new directory structure in `src` that matches the package structure. (No need to create directories in the `bin` directory: the compiler will create them anyway during compilation.) Put the new directory structure under version control. Adjust the file `compile.cmd` as needed.

   Once compiling and running from the command line works: Try `git status` to see any changes that need to be added to git. Commit and push your changes.

5. Use the `jar` command to create a jar file named `myjar.jar` that contains the byte code files to run your program; this includes adding the needed entries to file `mymanifest.mf`.

   Note that you need to `cd` into the `bin` directory before you create the jar file. While you are in the `bin` directory, the manifest file is then in the parent directory of the `bin` directory. Also the jar file itself shall be created created in the parent directory of `bin`. In both cases, you need to reference in the parameter of the jar command the parent directory using `..` (i.e. two dots). After having created the jar file, you can change back into the parent directory using `cd ..`

   Update the `createjar.cmd` to contain the commands needed to change into the `bin` directory, create that jar file, and change back into the parent directory as described above. Also update `runjar.cmd` to contain a simple `java -jar` command to run the jar file that is in the same directory as the file `runjar.cmd` itself. (Double check that running the jar file still works after having emptied your `bin` directory.)

   Once everything works: commit and push your changes.

   Submit the current version of the repository containing your solution to Gradescope *Assignment 3* by selecting *GitHub* as *Submission Method*.

   Note that Gradecsope copies the repository files as they were when you submitted then. Any later updates to your GitHub repository are not automatically transferred to Gradescope.

6. 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*.