HÁSKÓLI ÍSLANDS
Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
HBV202G: Software Design and Construction · Spring 2025
Andri Valur Guðjohnsen · Dr. Helmut Neukirchen

Assignment 7 · Due Tuesday 5.3.2025 / Friday 8.3.2025

Formalities:

- Submit your source code solution as a team via Gradescope as upload from GitHub.

Objectives: Learn using the AI-assisted coding tools with your IDE.

1. Install an AI-assisted coding tool for your IDE, e.g.:
   - US-based Codeium `https://www.codeium.com/` → Get Extension
2. Not graded, just to get familiar with AI-assisted coding by trying to re-do some parts of Assignment 5 with AI assistance:
   (a) Clone `https://github.com/helmutneukirchen/HBV202GAssignment5`
   (No need to fork/import: you will not submit this, but just use it to play around.)
   (b) Let the AI create the `isEmpty()` method for you that checks whether the stack is empty and let the AI as well create test cases for testing `isEmpty()` on an empty and a non-empty stack (i.e. Question 9 of Assignment 5), also run the tests.
   (c) You will notice that each AI-generated test case method creates a new stack that is stored in a local variable (because it tries to mimic the existing test case that does exactly the same).
   Now, lets try whether the AI adapts to your style if you would have used a *test fixture* (Ch. 5, slide 5-17) right from the start:
      i. Undo the AI-generated test cases, either by using your IDE's undo functionality or by reverting to the last committed version:
      IntelliJ: *Git*→*Uncommitted Changes*→*Rollback...*.
      Visual Studio Code: in left icon bar: *Source Control* icon→click on the three dots next your Assignment 5 project→*Changes*→*Discard All Changes*
      ii. Extract in the remaining test case `testNewStackIsNotFull()` the local variable containing the stack into a field and use a set up method annotated with `@Before` to initialise that field before executing each test case (either do it manually, as in Question 5 of Assignment 5, or use an IDE-provided automated refactoring, as in Question 6 of Assignment 6, or let AI do this, e.g. with Codeium: you can try the *Codium: Refactor* that is shown above the test class or if that does not work well, use the Chat where Codeium forwards your code to ChatGPT – but this needs to have telemetry enabled in your IDE as Code is sent to external services and in your user settings on `https://codeium.com/settings`, the item *Disable code snippet telemetry* must not be enabled.
      iii. With this changed context, try now how the above two test cases look like if you let the AI generate them.
   (d) Let the AI solve Question 5 of Assignment 5, i.e. create a test that checks that a full stack is reported as full. Run the generated test. If the generated test is not correct, provide in your comment to the AI more context (e.g. refer to the capacity) until the generated test is correct.

3. Now (graded), create a "Stone, Scissors, Paper" game with the help of an AI coding assistant:

   (a) Create a new Maven-based Java project in your IDE using version 1.4 of `org.apache.maven.archetypes:maven-archetype-quickstart` (see Assignment 4, steps 1. to 4, i.e. update the Java version to a recent JDK and also add a `maven.compiler.release` version and remove the `FIXME`, so that `mvn compile` and `mvn exec:java` can be used to run you code). Have a reasonable `.gitignore` file. Commit and push this initial project now to a new GitHub repository.

       Sometimes, your IDE is confused by your JDK version changes and it helps to delete now your project in your filesystem and to clone it again from GitHub.

   (b) Think about how to make the AI generate the game for you (typically, it is enough to use your IDE to create a new class with the name of the game and let then the AI make suggestions for methods inside that class: Adjust your IDE keyboard shortcuts, so that you can ask for the next/previous suggestion, see slides 7-18/19).

       The game needs to have one human player who makes choices via a textual UI (either entering "scissors" or "1" for scissors, etc.) but illegal input needs to be rejected) and who plays against a computer player that makes random choices. The game has to display what the computer player and the human player did choose and has to evaluate who won and to print out this result. After each round, the user is asked whether they want to continue or quit.

       - Most of the time, the AI creates references to classes and calls to methods that do not exist, yet. You can use your IDE's non-AI suggestions to create new classes and methods and then use AI to create the bodies of methods.
       - Typically, you have to force the AI with an own code line in order to make the AI to continue into the direction that you have in mind, e.g.:
         - to enforce using `Scanner(System.in)` for the textual user interface (note that the needed `import java.util.Scanner;` is typically not generated),
         - to introduce an `enum` for the values stone, scissors, paper,
         - to introduce an `enum` for the result of, evaluating the game, i.e. tie, computer won, you/the human player won.
       - Sometimes, the AI creates code that maybe looks correct to someone how cannot programme, but the generated code is complete bogus: then, create more context (either start coding on your own, or add more comments).

         Sometimes, the AI creates almost correct code, but you need later to do some manual changes to make it work.

   (c) You need to end up with some working "Stone, Scissors, Paper" game of a human player playing via a textual user interface against a computer player that makes random choices. The game has to display what the computer player and the human player did choose and has to evaluate who won and print out this result.

       Note: There is no need to add JUnit tests! (In principle, you should get used to have unit tests for your code. But typically, the AI-generated code is not easy to test: often, the game logic is mixed together with the textual user interface into one method – a testable design would have the game logic in a separate method and then, you could write unit tests for that method. If you really want to use AI-assisted coding in future, then you need to learn how to make the AI create code that is easy to test.)

4. Check that `mvn compile` and `mvn exec:java` work and take care that your have committed and pushed all your changes to your remote repository and submit your project to Gradescope. Remember to select your other team member in one of the first submission steps. The autograder needs to be able to compile and execute your submission.