Step 6 – Larger task

It's time to do a bigger task. This week I am not introducing any new topics, but we are spending time writing code based on topics we have already been through. I have chosen to create two alternative tasks.

Task 1 is going to be an option for those of you who think the topic has been demanding so far. It is based on tasks that we have already worked on. There are therefore solution proposals that can be used as support along the way. It is also a great advantage if you collaborate with others and seek help from the supervisors if you are stuck.

Task 2 is something completely different. It is intended to be an alternative for those of you who are doing well and who need a new challenge.

For those of you who think task 1 will be too big a challenge, I recommend going back to previous weeks' tasks and doing as many tasks as you can. Use this step to try to catch up as much as you can.

Here is the questionnaire for step 6 where you can tell how it went.

In Canvas you will find some files you need for this week. Get the zip in Canvas and unpack.

Task 1 – Book Registry

Our well-known book register. Those who have done the extra tasks will have already done much of what is required for this task. You choose whether you want to start from scratch or start from an existing starting point (your own code or a solution proposal). This is how the program can work:

At the beginning of the program, you should read data about books from a file (bok.txt - located in Canvas). Create book objects and place them in a book register.

Your program will then display a menu with the following options:

- Overview of all books.
- Add a book.
- Modify a book.
- Find a book based on genre.
- Find a book based on author.
- Find a book based on ISBN.
- Remove a book.
- Exit.

When the program ends, the information about the books (which may have changed) must be written to a file.

Some tips:

- Do not try to make everything at once. When creating a program, it will be an advantage to start with something simple and build on, little by little. Check that what you make works along the way. A sensible first step could be, for example, to create a program that manages to display a menu. A next step may be to try to read the books when the program starts.
- This is not a work requirement, so you can easily opt out of some functionality. Make as much as you can with the time you have. You can also choose to implement something other than what the task asks for. The most important thing is that you spend time writing code.
- Some may have made chapters in the books in an extra assignment before. Bring it if you want. Remember that you must then enter information about chapters in bok.txt.
- Have you written to file, but there is no content there? Did you remember to close fileWriter?
- Not sure how to create an enum based on text? You will need that when you are going to read from a file and turn the genre into an enum. Maybe this can help? You will also find an example of this in a previous solution proposal as well. If you do not get it, then you can always choose to let the genre be String, and not enum.

Task 2 – Fill in the blanks

To quote Per: "En klassiker på barnebursdager er adjektivhistorien". A story has placeholders in it where you place random adjectives. If you choose adjectives somewhat creatively, then the story tends to be quite funny (or odd, your mileage may vary)

Think, when you finish this task you have a useful tool for various occasions!

Here is a suggested program (but feel free to make your own adjustments):

When the program starts, read the story (story.txt) and the list of adjective suggestions (adjective.txt). The user is presented with the following menu:

1) Create adjective history automatically.
2) Create adjective history with your own adjectives.
3) Exit.

In (1), the already existing adjective suggestions shall be used to create the story. But which adjectives are used, and the order of them, should be arbitrary. Completed history is written to file.

In (2), the user shall be given the opportunity to enter their own adjectives. You must know in advance how many different adjectives you need (and the user must know how many adjectives remain after each adjective). When all the adjectives are in place, the story should be written to file. The adjectives must also come here in random order.

It can be an advantage to use unique file names when the story is saved to file…

Possible extensions:

- Offer more stories for different occasions. These may also have specific input requirements, such as names of birthday children, names of participants in the company etc. (See example here)

- Offer several different sets of adjective suggestions for the chosen story.
- Let the user select some adjectives. If the user does not want to create more himself, adjective suggestions (read from file) are used for the remaining ones.
- Extra difficult: Create different types of placeholders that match different types of inflections of the adjectives. That way you can insert a correct inflection of the adjectives in the story. It also means that you must have the correct bends available for insertion. That was a real nut, I think!

Some tips:

- - The adjectives must be placed in a random order (that's part of the point). There are several ways to solve it. One such way is the class Random. Another is to use a static method in Collections.
- - You will probably also need some methods you find in the String class (find placeholders, replace parts of text, etc.)