Assignment 4: Consumable Tracker Web Service (25% of course total) Marking Guide

Must be done individually.

Total = [50] marks. Use the formula Quiz_completed?(Quiz_mark+Program_mark):(0.5*Program_mark).

Quiz_mark

[5] Quiz: Online quiz.

Program_mark has three parts [32]+[8]+[5]

0 on Program_mark if cannot compile, or client and server are not in separate IntelliJ projects (communicate this to the student).

[32] Consumables Tracker Functionality

<u>Client Side [13]:</u>

- [2] Client does not generate nor use any json files
- [2] Able to start in a GUI, must start from a class with only the main method and calling only the SwingUtilities.invokeLater(Runnable) the start the GUI (otherwise -2)
- [1] Able to list all items correctly, must be in a JScrollPane (otherwise -1)
- [2+2] Able to add items correctly, both expired and not (the list all items view will be updated)
    - Must be achieved via an object of a user-defined class extending JDialog (otherwise -4)
    - Weight/Volume must change according to the selection made at the type field, which is a JComboBox (otherwise -1)
    - Only notes can be empty, if anything else is empty need to prompt the user to fix (if no checking -1)
    - LGoodDatePicker must be used for expiry date selection (otherwise -1)
    - User must be able to cancel this action (otherwise -1)
- [1] Able to remove items correctly (the list all items view will be updated)
- [1] Able to list expired items correctly
    - Must be in a JScrollPane (otherwise -1)
- [1] Able to list items that are not expired correctly
    - Must be in a JScrollPane (otherwise -1)
- [1] Able to list items expiring in 7 days
    - Must be in a JScrollPane (otherwise -1)

<u>Server Side [19]:</u>

- [2] Able to save and load added tasks for subsequent running of the program automatically
    - No marks for this point if methods other than JSON is used
    - When the client side exits the list must be saved as a json file (otherwise -2)
        - As triggered by GET /exit
- [2] Able to respond to GET /ping correctly (client receives a “System is up!” string)

- [2] Able to respond to GET /listAll correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
- [2] Able to respond to POST /addItem correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
  - It is possible to have variations on the URL and the data part, check the curlCommands.txt in the doc directory
- [2] Able to respond to POST /removeItem correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
  - It is possible to have variations on the URL and the data part, check the curlCommands.txt in the doc directory
- [2] Able to respond to GET /listExpired correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
- [2] Able to respond to GET /listNonExpired correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
- [2] Able to respond to GET /listExpiringIn7Days correctly (client receives the updated items list as JSON array object, which is automatically create via the Spring framework)
- [3] File describing how to communicate with the server directly via curl commands exists
  - (3-clear; 2-ok, 1-confusing)

[8] GUI Quality (while students can design their own GUIs, the GUIs have to be usable)

- [3] User Manual clearly states how all operations can be used (3-clear, 2-ok, 1-confusing)
- [3] Information shown in a reasonable way (e.g., good component alignment, component placement makes sense, information easy to find)
- [2] Application does not hang/crash during testing (2-smooth, 1-sometimes)

[5] Code Quality and Style

- Reasonable object-oriented structure (follow the required package structure)
  - must at least 4 non-trivial classes placed in required packages.
- Very minor violations of style guide have no penalty
  - (e.g., having "int myCount=0;" (spacing wrong) once)
- Lose a few marks for consistent problems
  - (like always getting the spacing wrong).
- Larger penalties possible for horrific code (0 on Program_mark)

Some specifics to check with suggested deductions (to Program_mark)

- -3 Poor/missing JavaDoc comment on a class (not needed on fields)
- -5 Incorrect indentation, brackets, or spacing (use IDE's reformat if needed).
- -5 Poor intention revealing class, method/variable names.
  - (e.g., making everything public or static, undecipherable names)

Forward to instructor if material is suspiciously similar to another submission or code posted online.