Home Exam – Software Engineering – D7032E – Reexam 1

Teacher: Josef Hallberg, josef.hallberg@ltu.se, A3305

Instructions

- The home reexam is an individual examination.
- **The home reexam has no deadline**. Take as much time as you need.
    - The teacher makes no promises of quick assessment of your reexam.
- **Book at time for an oral exam-session** with Josef Hallberg when you are done.
    - Availability of oral exam-sessions depends on the teacher's schedule and workload.
- The submission should consist of a compressed file containing a pdf file with answers to the written questions and any diagrams/pictures you wish to include, and your code. Place all files in a folder named as your username before compressing it. You may find it difficult to attach compressed files directly in e-mails, in which case you can email a link to a file-sharing site where your compressed file is stored.
- Use subject "**D7032E: Home Exam**" so your hand-in won't get lost (I will send a reply by email within a few days if I've received it).
- It should contain ***original*** work. You are not allowed to cheat, in any interpretation of the word. You are allowed to discuss questions with classmates, but you must provide your own answers. Additionally, you are allowed to use AI tools and use the material produced by these tools, however, ***what you hand in should represent your knowledge and understanding of the subject*.** That means, if AI tools are generating answers or code for you, it is your responsibility to study the material provided by an AI and **make sure you understand the generated material, understand design decisions that were made by the AI model, and are able to justify and motivate design decisions.**
- Your external references for your work should be referenced in the hand in text. All external references should be complete and included in a separate section at the end of the hand in.
- The language can be Swedish or English.
- The examiner reserves the right to refuse to grade a hand-in that does not use a correct/readable language. Remember to spellcheck your document before you submit it.
- Write in running text (i.e. not just bullets) – but be short, concrete and to the point!
- Use a 12 point text size in a readable font.
- It's fine to draw pictures by hand and scanning them, or take a photo of a drawing and include the picture; however make sure that the quality is good enough for the picture to be clear.
- Judgment will be based on the following questions:
    - Is the answer complete? (Does it actually answer the question?)
    - Is the answer technically correct? (Is the answer feasible?)
    - Are selections and decisions motivated? (Is the answer based on facts?)
    - Are references correctly included where needed and correctly? (Not just loose facts?)
- **To have part B assessed and graded you need to motivate your design in a one-to-one oral exam session. The purpose of the oral exam session is to assess whether the material you have handed in represent your understanding of the subject. This is to make sure you do not hand in AI generated material, or material produced by other people, that you do not yourself understand. The examiner reserves the right refuse assessing part B in cases where the material you have handed in does not represent your understanding of the subject.**

Total points: 25	
Grade	Required points
5	22p
4	18p
3	13p
U (Fail)	0-12p

Good luck! /Josef

Scenario: Apples to apples

(http://www.com-www.com/applestoapples/)
(https://www.youtube.com/watch?v=4fgCEMFSF_Q)
Apples to apples is a party game in which players must try and associate an adjective to one of the nouns printed on the cards held in the player's hand. A group of players compete to come up with the most amusing association and one player each round is awarded the point for the best answer. (If you want to turn this game into "Cards against humanity" for your own amusement it is fine, the rules are similar and the card texts for "Cards against humanity" are available online)

Rules:

Setting up the game

1. Read all the green apples (adjectives) from a file and add to the green apples deck.
2. Read all the red apples (nouns) from a file and add to the red apples deck.
3. Shuffle both the green apples and red apples decks.
4. Deal seven red apples to each player, including the judge.
5. Randomise which player starts being the judge.

Playing the game

6. A green apple is drawn from the pile and shown to everyone
7. All players (except the judge) choose one red apple from their hand (based on the green apple) and plays it.
8. The printed order of the played red apples should be randomised before shown to everyone.
9. All players (except the judge) must play their red apples before the results are shown.
10. The judge selects a favourite red apple. The player who submitted the favourite red apple is rewarded the green apple as a point (rule 14).
11. All the submitted red apples are discarded
12. All players are given new red apples until they have 7 red apples
13. The next player in the list becomes the judge. Repeat from step 6 until someone wins the game.

Winning the game

14. Keep score by keeping the green apples you've won.
15. Here's how to tell when the game is over:
    - For 4 players, 8 green apples win.
    - For 5 players, 7 green apples win.
    - For 6 players, 6 green apples win.
    - For 7 players, 5 green apples win.
    - For 8+ players, 4 green apples win.

Future modifications: (you do not need to implement these but create your design for modifiability and extensibility)
The game currently has 4 primary phases:

A. Draw a green apple
B. Players submit a red apple
C. The judge selects a winner
D. Replenish players' hands with red apples

The developer is thinking of altering the game mechanics by adding or replacing phases. For example, the developer is considering replacing phase C by letting players all the players vote on their favourite red apple (but they can't vote for their own) and have the majority vote elect the winner. The developer is also considering letting the judge replace cards in their hand (consequently adding a phase before the current phase A). Additionally, the developer is also considering adding additional functionalities to the red apples deck. The developer is considering adding "Wild red apples" which are empty red apples cards which will let the player fill in their own answer, and "Apples and Pears" cards that can be played together with another red apple card and will cause the green apple to be replaced before the vote for the favourite answer is made (but after players have submitted their answers).

In addition to the developer's ideas for future modifications you may also consider the following extensions:

- Apple's eye view: http://www.com-www.com/applestoapples/applestoapples-rules-variations-14.html
- Bad harvest: http://www.com-www.com/applestoapples/applestoapples-rules-variations-20.html
- Two-for-one apples: http://www.com-www.com/applestoapples/applestoapples-rules-variations-06.html
- And other changes that may come in the future that somehow changes what can be done in a phase, that adds a phase, or that replaces an existing phase.

Quality attributes: Design with priority on *Modifiability*, *Extensibility* (Future modifications), and *Testability*.

Questions

Part A

1. Unit testing (2p, max 1 page)
Which requirement(s) (rules and requirements 1 – 15 on previous page) is/are currently not being fulfilled by the code (refer to the requirement number in your answer)? For each of the requirements that are not fulfilled answer:

- If it is possible to test the requirement using JUnit without modifying the existing code, write what the JUnit assert (or appropriate code for testing it) for it would be.
- If it is not possible to test the requirement using JUnit without modifying the existing code, motivate why it is not.

2. Software Architecture design and refactoring (9p, max 2 pages excluding diagrams)
Consider the requirements (rules and requirements 1 – 15 and the quality attributes with future modifications on the previous page in mind) and the existing implementation. Update / redesign the software architecture design for the application. The documentation should be sufficient for a developer to understand how to develop the code, know where functionalities are to be located, understand interfaces, understand relations between classes and modules, and understand communication flow. Use good software architecture design and coding best practices according to SOLID principles and Booch's metrics. Also reflect on and motivate:

- how you are addressing the quality attribute requirements (in the context of the potential future modifications as listed on the previous page). What design choices did you make specifically to meet these quality attribute requirements?
- the use of design-patterns, if any, in your design. What purpose do these serve and how do they improve your design?

Note: The oral exam-session will cover primarily your software architecture design. For this session you will be expected to answer questions about your design and motivate design choices. You are allowed to refer to diagrams and code provided in your hand-in. It may therefore be more important to produce supporting diagrams than lengthy texts since motivations for design choices can be given verbally. You should be able to answer questions about your design and about design choices without having to look at the written text (but looking at diagrams is permitted).

Part B

3. Quality Attributes, Design Patterns, Documentation, Re-engineering, Testing (14p)
Refactor the code so that it matches the design in question 2 (you may want to iterate question 4 once you have completed your refactoring to make sure the design documentation is up to date). The refactored code should adhere to the requirement (rules and requirements 1 – 17 on previous page). Things that are likely to change in the future, divided into quality attributes, are:

- Extensibility: House rules, such as those described in the "Future modifications" may be introduced in the future. Other types of phases and functionalities may also be considered in the future.
- Modifiability: The way network functionality and bot-functionality is currently handled may be changed in the future. Network features in the future may be designed to make the server-client solution more flexible and robust, as well as easier to understand and work with.
- Testability: In the future when changes are made to both implementation, game rules, and phases of the game, it is important to have established a test suite and perhaps even coding guidelines to make sure that future changes can be properly tested. You only need to make tests for requirements 1-15.

Please help the newbie developer by re-engineering the code and create better code, which is easier to understand. There is no documentation other than the comments made inside the code and the requirements specified in this Home Exam on page 2.

The code and red/green apples files are available at: http://staff.www.ltu.se/~qwazi/d7032e2018/

The source code is provided in the Apples2Apples.java filee (compile with: javac Apples2Apples.java and run with: java Apples2Apples [Optional: #onlineClients || ipAddress_to_server]). Running without parameters creates 3 bots together with a server player, when adding a number as a parameter the server will expect that many instances to connect to the server before starting, and specifying an IP address the software will act in online client mode and connect to the specified IP. Note that the current software uses ExecutorService which does not exist in earlier versions of Java. Additional files with red and green apples texts are available in greenApples.txt and redApples.txt and these files are expected to be in the same directory as Apples2Apples.java in the current design of the software (but you may change this if you wish for your own design).

The Server does not need to host a Player or Bot and does not need to launch functionality for these. It is ok to distribute such functionalities to other classes or even to the online Clients.

Add unit-tests, which verifies that the game runs correctly (it should result in a pass or fail output), where appropriate. It is enough to create unit-tests for requirements (rules and requirements 1 – 15 on page 2). The syntax for running the unit-test should also be clearly documented. Note that the implementation of the unit-tests should not interfere with the rest of the design.

Examination criteria - Your code will be judged on the following criteria:

- Whether it is easy for a developer to customise the game mechanics and modes of the game.
- How easy it is for a developer to understand your code by giving your files/code a quick glance.
- To what extent the coding best-practices have been utilised.
- Whether you have paid attention to the quality metrics when you re-engineered the code.
- Whether you have used appropriate design-patterns in your design.
- Whether you have used appropriate variable/function names.
- To what extent you have managed to clean up the messy code.
- Whether program uses appropriate error handling and error reporting.
- Whether the code is appropriately documented.
- Whether you have created the appropriate unit-tests.

If you are unfamiliar with Java you may re-engineer the code in another structured programming language. However, instructions need to be provided on how to compile and run the code on either a Windows or MacOS machine (including where to find the appropriate compiler if not provided by the OS by default).

It is not essential that the visual output when you run the program looks exactly the same. It is therefore ok to change how things are being printed etc.

Evaluation criteria for Reexam 1

Question 2: Update / redesign the software architecture design for the application. Use good software architecture design and coding best practices according to SOLID principles and Booch's metrics. Also reflect on and motivate:

- how you are addressing the quality attribute requirements (in the context of the potential future modifications as listed on the previous page). What design choices did you make specifically to meet these quality attribute requirements?
- the use of design-patterns, if any, in your design. What purpose do these serve and how do they improve your design?

Quality attributes: Modifiability, Extensibility, Testability

Note about evaluation for question 2: During the 15-minute oral exam it will be difficult to assess to what extent the student has been able to satisfy the quality attribute and therefore the assessment of this will be done in Question 3. Instead, the assessment of Question 2 will focus on the student's ability to correctly reason about and justify design choices that have been made to satisfy the quality attributes with arguments supported by best practices, SOLID principles, and Booch's metrics.

Evaluation criteria:
Total 9 points

1. Identifying shortcomings in the original design based on best-practices, SOLID principles, and Booch's metrics (0-1p)
2. Reasoning and justifying design choices in the refactored/redesigned code based on best-practices, SOLID principles, and Booch's metrics (0-3p)
3. Reasoning about how the updated design satisfies the quality attributes Modifiability, Extensibility, and Testability (0-3p)
4. Motivating the choice of design pattern(s) or choice not to use design pattern(s) that might have been applicable (note: the exam does not require you to use design-patterns if you can motivate why it would not improve the design, points will be awarded for based on your understanding of why and when design patterns should or should not be used) (0-2p)

Question 3: Refactor the code so that it matches the design in question 2. The refactored code should adhere to the requirements and should address the Extensibility, Modifiability, and Testability quality attributes.

Note about question 3: Assessment of Extensibility overlaps slightly with assessment of Modifiability and assessment of Testability due to the nature of the Extensibility quality attribute. Therefore, for the evaluation criteria the assessment 1) below will be focused specifically on to which extent new extensions (new game modes) can be integrated with the game without affecting existing (compiled) implementation. This includes being able to unit-test and integration test the extension prior to launching it (unit-tests may update when adding an extension). For additional information please refer to the documentation (available in Canvas→Files→Slides from 2023)

Evaluation criteria:
Total 14 points

The extent to which the code

1. can be extended (alternate rules and phases in future modifications) (0-2p)
2. can be modified (network functionality, bot functionality, and parts of future modifications) (0-3p)
3. is designed for testability (0-1p)
4. is unit-tested (requirements 1-12) (0-2p)
5. follows best practices (structure, standards, naming, etc.) (0-1p)
6. correctly implements the functionality of the game (0-2p)
7. handles and reports errors (0-1p)
8. is appropriately documented (0-1p)
9. is true to the design in question 2 (0-1p)