Home Exam – Software Engineering – D7032E – 2024

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

Instructions

- The home exam is an individual examination.
- The home exam is to be handed in by **Wednesday October 23, at 09.00**, please upload your answers in Canvas (in the Assignment 6 hand-in area) as a compressed file (preferably one of following: rar, tar, zip), containing a pdf file with answers to the written questions and any diagrams/pictures you may wish to include, and your code. Place all files in a folder named as your username before compressing it (it's easier for me to keep the hand-ins apart when I decompress them). If for some reason you can not use Canvas (should only be one or two people) you can email the Home Exam to me. Note that you can NOT email a zip file since the mail filters remove these, so use another compression format. 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).
- Every page should have your full **name** (such that a singular page can be matched to you) and each page should be numbered.
- 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.**

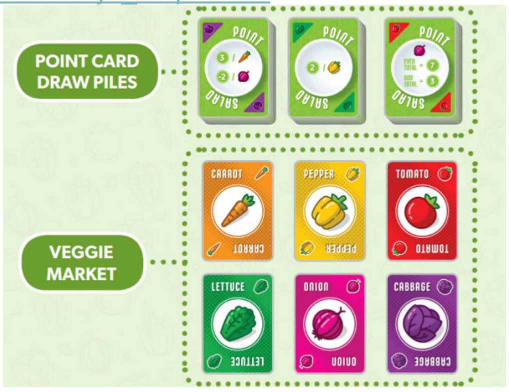
Total points: 25	
Grade	Required points
5	22p
4	18p
3	13p
U (Fail)	0-12p

Good luck! /Josef

Scenario: Point Salad
Point Salad is a card-drafting game for 2-6 players. Players take turns building a salad of veggies and collecting point cards in order to score the most points for the ingredients in their salad. The game consists of 108 double-sided veggie/point cards. The designers of Point Salad has also released the game Point City, which is a similar card-drafting game and that is the premise for the task in this Home Exam. Complete original rules (for both Point Salad and Point City) are available as separate PDFs in Canvas.

- Short overview of Point Salad: https://www.youtube.com/watch?v=yi3wODxl8RM
- Longer playthrough of Point Salad: https://www.youtube.com/watch?v=jB_56quRmWI

SillySalad is a newbie developer who has decided to turn Point Salad into an online computer game and has future plans of expanding the game to support additional variations of the game, such as the Point City game. Unfortunately SillySalad isn't very good at software design and there are several bugs. Your role is to help SillySalad improve upon the design by following the remaining instructions in this test. You should also structure the design in such a way that new variations of the game, such as the Point City game, can be added in the future (but you don't need to implement these changes, just make sure the design support future extension and modifications according to best practices).



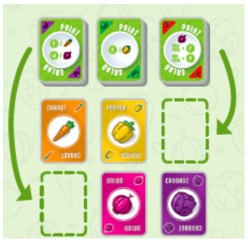
The Point Salad Market

Rules:

1. There can be between 2 and 6 players.
2. The deck consists of 108 cards (as specified in the PointSaladManifest.json file published in Canvas). A card has two sides, one with the criteria consisting of scoring rules, the other with one of six different vegetables (Pepper, Lettuce, Carrot, Cabbage, Onion, Tomato). There are 18 of each vegetable.
3. Form the deck so that
   a. 2 Players: Use 6 random vegetable of each (36 cards in total)
   b. 3 Players: Use 9 random vegetable of each (54 cards in total)
   c. 4 Players: Use 12 random vegetable of each (72 cards in total)
   d. 5 Players: Use 15 random vegetable of each (90 cards in total)
   e. 6 Players: Use the entire deck

   Do not reveal which cards were removed

4. Shuffle the cards and create three roughly equal draw piles with roughly equal draw piles with point card sides visible.
5. Flip over two cards from each draw pile to form the vegetable market.
6. Randomly choose a start player
7. On a player's turn the player may draft one or more cards and add to the player's hand. Either:
   a. One point card from the top of any of the draw piles, or
   b. Two veggie cards from those available in the veggie market.
8. The player may opt to turn a point card to its vegetable side (but not the other way around).
9. Show the hand to the other players.
10. Replace the market with cards from the top of the corresponding point card draw pile
11. If a draw pile runs out of cards, then draw cards from the bottom of the draw pile with the most cards instead.
12. Continue step 7-12 for the next player until there are no more cards in the Point Salad Market
13. Calculate the score for each player according to the point cards in hand.
14. Announce the winner with the highest score



Future modifications: (you do not need to implement these but create your design for modifiability and extensibility)

15. Support the Point City variation of the game. It has many similarities with Point Salad, but some additional game mechanics (rules PDF in canvas and a short video: https://www.youtube.com/watch?v=E0-Ut66g1HY)

Quality attributes:

16. Design with priority on Modifiability, Extensibility (requirement 15), and Testability.

Questions

Part A

1. Unit testing (2p, max 1 page)

Which requirement(s) (rules and requirements 1 – 14 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 – 14 and the quality attributes with requirement 16 in mind on the previous page) 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 requirements 16 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 recommended).

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 2 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 – 14 on previous page). Things that are likely to change in the future, divided into quality attributes, are:

- Extensibility: Additional game-modes, such as those described in the "Future modifications to the game" may be introduced in the future.
- Modifiability: The way network 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. In addition, the potential game extension (requirement 15) adds new types card types, tokens, and scoring mechanisms, which will change how some operations are handled.
- Testability: In the future when changes are made to both implementation, game rules, and game modes 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 1-14.

Please help SillySalad 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 official game rules for the core game as well as additional game extensions are available in Canvas

The source code is provided in one file, PointSalad.java which contains the server, the client, and the code for one player, as well as all the game-states and game logic. There is also a jar-file for json processing (json.jar) and a manifest of all of the 108 cards in the game (PointSaladManifest.json). If all files are in the same folder:

- You can compile the game with "`javac –cp .:json.jar PointSalad.java`"
- You can run the server with "`java –cp .:json.jar PointSalad`"
  (will then let you set number of players and bots in the beginning of the game)
- You can also run the server with arguments for number of players and bots
  "`java –cp .:json.jar PointSalad [#Players] [#Bots]`"
- You can connect an "online" client player: "`java –cp .:json.jar PointSalad 127.0.0.1`"

The server must be started before any of the clients are started. The server waits for the online clients to connect before starting the game.

In the re-engineering of the code the server does not need to host a player and does not need to launch functionality for this. It is ok to distribute such functionalities to other classes or even to the online Clients. The essential part is that the general functionality remains the same.

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 – 14 and any of the additional future modifications that are implemented from requirement 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 (*Complete Rubrics in the next page*):

- 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 Rubrics

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 requirement 16 on page 2). 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 variant) 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 (new game modes) (0-2p)
2. can be modified (network functionality, new cards, scoring, optional rule) (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)