Home Exam – Software Engineering – D7032E – 2019

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 **Friday November 1, at 17.00**, 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 copy information from the Internet and other sources directly. It also means that it is not allowed to cheat, in any interpretation of the word. You are allowed to discuss questions with class-mates but you must provide your own answers.
- 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 text should be language wise correct and the examiner reserves the right to refuse to correct 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?)

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

Good luck! /Josef

Scenario: King of Tokyo: Power Up! (https://www.youtube.com/watch?v=HqdOaAzPtek)
King of Tokyo is a boardgame in which you play a monster intent on destroying Tokyo while whacking the other monsters having the same ambition. Power Up is an expansion to the base game in which you are also able to evolve new abilities as a monster. Mr. NoPlan has decided to turn the beloved boardgame into a computer game but hasn't exactly done a good job of designing the code, even though it largely works (but with plenty of bugs). Your role is to help Mr. NoPlan improve upon the design by following the remaining instructions in this thesis.

Rules:

Setting up the game

1. Each player is assigned a monster.
2. Set Victory Points to 0
3. Set Life to 10
4. Shuffle the store cards (contained in the deck) (todo: add support for more store cards)
   - Start with 3 cards face up (available for purchase).
5. Shuffle the evolution cards for the respective monsters (todo: add support for more evolution cards)
6. Randomise which monster starts the game.

Playing the game

7. If your monster is inside of Tokyo – Gain 1 star
8. Roll your 6 dice
9. Select which of your 6 dice to reroll
10. Reroll the selected dice
11. Repeat step 9 and 10 once
12. Sum up the dice and assign stars, health, or damage
    - Tripple 1's = 1 Star Each additional 1 equals +1 star
    - Tripple 2's = 2 Stars Each additional 2 equals +1 star
    - Tripple 3's = 3 Stars Each additional 3 equals +1 star
    - Each energy = 1 energy
    - Each heart
      i. Inside Tokyo – no extra health
      ii. Outside Tokyo - +1 health (up to your max life, normally 10 unless altered by a card)
    - Tripple hearts = Draw an Evolution Card (todo: add support for more evolution cards).
    - Each claw
      i. Inside Tokyo – 1 damage dealt to each monster outside of Tokyo
      ii. Outside Tokyo
        1. Tokyo Unoccupied = Move into Tokyo and Gain 1 star
        2. Tokyo Occupied
           a. 1 damage dealt to the monster inside Tokyo
           b. Monsters damaged may choose to leave Tokyo
           c. If there is an open spot in Tokyo – Move into Tokyo and Gain 1 star
13. Buying Cards (As long as long as you have the Energy, you can take any of the following actions)
    - Purchase a card = Pay energy equal to the card cost (replace purchased cards with new from the deck)
    - Reset store – pay 2 energy
14. A store card can be of either type "Keep" or "Discard". "Discard" cards take effect immediately when purchased, and "Keep" cards may either be played when the owner desires or provides an active power/ability.
15. End of turn.

Winning the game

16. First monster to get 20 stars win the game
17. The sole surviving monster wins the game (other monsters at 0 or less health)
18. A monster that reaches 0 or less health is out of the game

Future modifications to the game

The game currently only supports a few store cards and a few card mechanisms. Additionally, only one evolution card is currently supported for each of the monsters. Furthermore, only three monsters are currently in play. Mr. NoPlan would like the game to support more store and evolution cards. Mr. NoPlan reckons that adding more monsters is not very difficult once the software design supports different kinds of card mechanisms. Find the King of Tokyo cards at the links below:

The complete list of store cards is available at: http://staff.www.ltu.se/~qwazi/d7032e2019/StoreCardList.html
The complete list of evolution cards is available at: http://staff.www.ltu.se/~qwazi/d7032e2019/EvolutionCardList.html

Additional requirements (in addition to rules 1-18)

19. It should be easy to modify and extend your software (*modifiability* and *extensibility* quality attributes). It is to support future modifications as the ones proposed in the "future modifications of the game" section (you don't need to implement these unless you want to (but do add some cards), just structure the architecture so it is easy to do in the future).
20. It should be easy to test, and to some extent debug, your software (*testability* quality attribute). This is to be able to test the game rules as well as different phases of the game.

Questions (page 3/4)

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

Which requirement(s) (rules and requirements 1 – 18 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. Quality attributes, requirements and testability (1p, max 1 paragraph)

Why are requirements 19-20 on the previous page poorly written (hint: see the title of this question)? Motivate your answer and what consequences these poorly written requirements will have on development.

3. Software Architecture and code review (3p, max 1 page)

Reflect on and explain why the current code and design is bad from:

- an Extensibility quality attribute standpoint
- a Modifiability quality attribute standpoint
- a Testability quality attribute standpoint

Use terminologies from quality attributes: coupling, cohesion, sufficiency, completeness, primitiveness - https://atomicobject.com/resources/oo-programming/oo-quality).

4. Software Architecture design and refactoring (6p, max 2 pages excluding diagrams)

Consider the requirements (rules and requirements 1 – 20 on 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 (keeping in mind the quality attributes: coupling, cohesion, sufficiency, completeness, primitiveness - https://atomicobject.com/resources/oo-programming/oo-quality). Also reflect on and motivate:

- how you are addressing the quality attribute requirements (in requirements 19 – 20 on 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?

5. Quality Attributes, Design Patterns, Documentation, Re-engineering, Testing (13p)

Refactor the code so that it matches the design in question 4 (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 – 20 on previous page). Things that are likely to change in the future, divided into quality attributes, are:

- Extensibility: Additional cards, such as those described in the "Future modifications to the game" may be introduced in the future. Other types of cards and additional expansions 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. Additionally, currently the game only supports 3 players, and the number of players may need to be more flexible in the future.
- Testability: In the future when changes are made to both implementation, game rules, and cards 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.

Please help Mr. NoPlan 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 card lists (store/evolution cards) are available at: http://staff.www.ltu.se/~qwazi/d7032e2019/

The source code is provided in two files, `KingTokyoPowerUpServer.java` which contains the server and the code for one player, as well as all the game-states and game logic. (Run with: `java KingTokyoPowerUpServer`). This must be started before any of the clients are started. The client is located in `KingTokyoPowerUpClient.java` and can be run either as a person controlled online client (but current code only connects to localhost) or as a bot. (Run with: `java KingTokyoPowerUpClient [Optional: bot]`). The server currently assumes two online players (either person controlled or bot controlled) and waits for them 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 – 18 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 phases 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 object-oriented 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.