Home Exam – Software Engineering – D7032E – 2022

Teacher: Josef Hallberg, josef.hallberg@ltu.se, A2403

Instructions

- The home exam is an individual examination.
- The home exam is to be handed in by **Friday October 28, 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 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: Exploding Kittens

Exploding kittens is a simple card-game for 2 to 4 players (https://www.explodingkittens.com/pages/rules-kittens). Players start with a hand of cards, and take turns drawing and playing cards with different effects. If you draw an exploding kitten and are unable to defuse it then you explode and lose. If you don't explode, you win. DevCat has decided to turn this card-game into an online computer game, and has implemented the base version of the game but has big future plans to implement additional features, including the expansion packs.

DevCat is not a very experienced programmer and has not done a very good job of designing the Exploding Kittens code, even though it largely works (but with plenty of bugs) for the base version of the game. Your role is to help DevCat improve upon the design by following the remaining instructions in this test. You should also structure the design in such a way that new Exploding Kittens expansions can be added in the future (such as the expansions in requirement 14) and to support additions like menu and bot (requirements 15 and 16).

Rules and requirements:

1. There can be between 2 and 5 players
2. Each player should be given 1 Defuse card
3. The deck should consist of the following cards before the first hands are dealt:
    a. 2 Defuse cards for 2-4 players, 1 Defuse card for 5 players.
    b. 4 Attack cards
    c. 4 Favor cards
    d. 5 Nope cards
    e. 4 Shuffle cards
    f. 4 Skip cards
    g. 5 SeeTheFuture cards
    h. 4 HairyPotatoCat cards
    i. 4 Cattermelon cards
    j. 4 RainbowRalphingCat cards
    k. 4 TacoCat cards
    l. 4 OverweightBikiniCat cards
4. Shuffle the deck before the first hands are dealt.
5. Deal 7 cards from the deck to each player.
6. Insert enough ExplodingKitten cards into the deck so there is 1 fewer than the number of players
7. Shuffle the deck before the game starts
8. Random player goes first
9. On a player's turn the player can either Pass or Play one or several cards
    a. Pass will end the turn and the player will draw a card (hoping it is not an ExplodingKitten)
    b. A player can play as many or as few cards as desired upon one's turn
10. If an ExplodingKitten card is drawn the Player has to play a Defuse card if one is on hand or explode.
    a. A defused ExplodingKitten card is inserted back into the deck at the desired location by the surviving Player
    b. An exploded Player discards the cards on hand, including the ExplodingKitten card, and may not take additional turns.
11. Playing a card
    a. Playing an Attack card will end the turn without drawing a card
        i. The next Player takes 2 turns in a row unless that Player plays another Attack card, in which case the consecutive player has to take an additional 2 turns (e.g. 4 turns, then 6, etc.)
    b. Playing a Favor card forces another player to give you 1 card of their choice to you.
    c. Playing a Shuffle card will shuffle the deck.
    d. Playing a Skip card will end your turn without drawing a card (only 1 tun in case of taking multiple turns)
    e. Playing a SeeTheFuture card will allow the Player to view the top 3 cards in the deck.
    f. Playing two cards of any kind will allow the Player to steal random card from another player
    g. Playing three cards of any kind will allow the Player to name a card to receive from another player (if available)
12. A Nope card can be played within 5 seconds of any card(s) defined in rule 11 and 12 to stop their action.
    a. A Nope card played on top of another Nope alternates its meaning (between Nope and Yup)
13. When all but one players have exploded the remaining player is announced as the winner to all players.

Future modifications to the game

14. Support expansions like Imploding Kittens, Streaking Kittens, and Barking Kittens (but don't include physical gimmicks like the cone of shame or tower of power crown).
15. Add support for a menu from which to select which expansions or other game variants to play with.
16. Support the addition of bots that can join the game instead of players.

Additional requirements (in addition to rules 1-16)

17. It should be easy to modify and extend your software (*modifiability* and *extensibility* quality attributes). It is to support future modifications such as the ones proposed in the "future modifications to the game" section (you don't need to implement these unless you want to, just structure the architecture so it is easy to do in the future. Though implementing some of the future modifications may help you see whether your updated structure is good or not).
18. 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-play.

Questions (page 3/4)

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

Which requirement(s) (rules and requirements 1 – 13 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 17-18 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 – 18 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 (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 17 – 18 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?

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 – 18 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 expansions (requirement 14) adds new types of card interactions for both new and existing cards, 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.

Please help DevCat 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 expansions are available in Canvas

The source code is provided in one file, `ExplodingKittens.java` which contains the server, the client, and the code for one player, as well as all the game-states and game logic. (For server - run with: `java ExplodingKittens [#Players] [#Bots]` *`(example: java ExplodingKittens 2 0)`* for client – run with: `java ExplodingKittens [IP-address of server]` *`(example: java ExplodingKittens 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 – 13 and any of the additional future modifications that are implemented from requirements 14 – 16 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.

Q4 point distribution (though points may be awarded/deducted in one place but not the other if overlap in the assessment exists)

- *Coding best practices (metrics) (1p)*
- *Classification of functionality / classes (comprehensibility) (2p)*
- *Quality attributes (reasoning) (2p)*
- *Design patterns (reasoning - if applicable) (1p)*

Q5 point distribution (though points may be awarded/deducted in one place but not the other if overlap in the assessment exists)

- *Customising game mechanics and phases of the game (3p)*
- *Develop and understand code based on quick glance (1p)*
- *Coding best practices (structure, standards, naming, etc.) (1p)*
- *Quality attributes / metrics (implementation) (2p)*
- *Design patterns (implementation - if applicable) (1p)*
- *Error handling / error reporting (1p)*
- *Documentation (1p)*
- *Unit testing (2p)*
- *True to the design documents (1p)*