Home Exam – Software Engineering – D7032E – 2017

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 October 27, 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

A developer has attempted to create an Uno game in Java:
https://en.wikipedia.org/wiki/Uno_(card_game) (the rules stated here do not fully match the rules used in the implementation the developer has made, so only use the wiki for clarification and context).

The rules the developer has decided to use are:

1. Each player is given 7 cards at the start
2. The starting card is drawn from the deck
3. If a starting card is a special card (Skip, Reverse, Draw Two, Wild, or Wild Draw Four) then that card is to be mixed into the deck and a new card is to be drawn.
4. A player (or bot) may play a card that either matches the value of the previously played card (0, 1, … 9, +2, Skip, Reverse, Draw Two, Wild, or Wild Draw Four), that matches the color of the previously played card (red, yellow, green, blue), or a wild card (Wild or Wild Draw Four). Note that there is no restrictions for Wild cards in this implementation like the original rules state.
5. A player (or bot) may choose to play multiple cards of the same value at one time, but the first card that is played must be a valid play in accordance to rule 4.
6. If multiples of a special card that may affect the next player is played (Skip, Reverse, Draw Two, Wild Draw Four) then the effects should stack. That is, 2xSkip means skipping the next two players, 2xReverse means continuing in the same direction as currently, 2xDraw Two results in the next player having to draw 4 cards, and 2xWild Draw Four results in the next player having to draw 8 cards.
7. If the player (or bot) is unable to play a card the player must draw a card from the deck. This continues until the player (or bot) is able to play a card.
8. When it is the turn of the player (or bot) the player (or bot) must play a valid card or follow rule 6.
9. If the entire deck is used during play, the top discard is set aside and the rest of the pile is shuffled to create a new deck. Play then proceeds normally.
10. It is illegal to trade cards of any sort with another player.
11. "uno" must be called when the player (or bot) plays the second to last card. (In the code this is done by playing one or multiple cards, separated by comma, and then adding "uno" to the end, also separated by comma: 5, 1, uno)
12. A player (or bot) must have called "uno" on the previous round and have exactly one valid card to play on the turn of the player (or bot) to win.
13. If a player (or bot) has forgotten to call "uno" when the second to last card was played the player (or bot) is penalized by having to draw one card from the deck at the start of their next turn (and may consequently not win that round).
14. The player (or bot) who first gets rid of all cards in hand (0 cards left) wins. Note that this is different from the 500 points that the original rules state.

Additionally, the developer has decided upon the following requirements:

15. It should be easy to add new types of Uno cards in the future (Extensibility quality attribute)
16. It should be easy to modify the effect of current cards and the behavior of the bots (Modifiability quality attribute)
17. It should be easy for other developers to understand the code and the structure of the code so that they can develop their own clients and bot programs to compete with each other and add new functionalities (Comprehensibility quality attribute).

Questions

1. Unit testing (3p, max 1 page)
Which requirements (rules and requirements 1 – 14 on previous page) are currently not being fulfilled by the code (in your answer refer to the requirement number)? 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 15-17 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 (2p, 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

Use terminologies from quality attributes: coupling, cohesion, sufficiency, completeness, primitiveness - http://atomicobject.com/pages/OO+Quality).

4. Software Architecture design and refactoring (6p, max 2 pages excluding diagrams)
Consider the requirements (rules and requirements 1 – 17 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 - http://atomicobject.com/pages/OO+Quality). Also reflect on and motivate:

- how you are addressing the quality attribute requirements (in requirements 15 – 17 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 – 17 on previous page). Thigs that are likely to change in the future, divided into quality attributes, are:

- Extensibility: House rules, such as those described on the wiki-page at: https://en.wikipedia.org/wiki/Uno_(card_game) may be introduced in the future. Other types of cards 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. The bot-functionality may be improved and it should be relatively easy for third-party developers to create their own bot-programs to compete with other developers with (battle of the bots).
- Comprehensibility: In the future the developer wishes to have an active community of third-party developers, and wants it to be possible to provide good documentation and good APIs to simplify their development work.

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 source code is provided in the `Server.java, Client.java, and Card.java` files (compile with: `javac Server.java` and `javac Client.java` respectively. Run with: `java Server [[Optional:] #OnlineClients]` and `java Client` respectively).

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, preferably color coded), where appropriate. It is enough to create unit-tests for requirements (rules and requirements 1 – 17 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 customize the game mechanics and functionality of cards.
- 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 utilized.
- 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. The current visual output uses Unicode for colorization of text, which is supported in the Mac OS terminal and other terminals, but not in the Windows Command prompt (you may download and use Cygwin if you wish to have this functionality). It is therefore ok to change how things are being printed etc.