Home Exam – Software Engineering – D7032E – 2021

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

Instructions

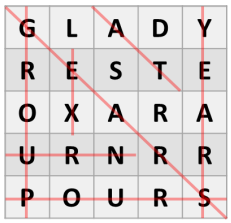
- The home exam is an individual examination.
- The home exam is to be handed in by **Friday October 29, at 07.00**, please upload your answers in Canvas (in the Assignment 6 hand-in area) as a compressed file (preferably one of following: rar, tar, zip), containing a pdf file with answers to the written 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: Variety WordSquares (WordSquares Variants)

WordSquares is a simple strategy game where players take turns picking the letter which all players must place on their personal gameboard. The idea is to place letters such that they form as many words as possible in the horizontal, vertical, and forward diagonal (upper left to lower right) directions. Scoring in traditional WordSquares is based on the length and number of words formed. Alternate game-modes include using Scrabbleboard tiles (double and triple letter-scores, double and triple word-scores) as well as using scrabble letters (letters are worth different points - https://en.wikipedia.org/wiki/Scrabble_letter_distributions). Variety WordSquares is a digital version containing the a few different variants of the game.



Example:
URN
POURS
GROUPS
EX
YEARS
GEARS
AT

Wosq is not a very experienced programmer and has not done a very good job of designing the Variety WordSquares code, even though it largely works (but with plenty of bugs) for the game variants that currently exists. Your role is to help Wosq improve upon the design by following the remaining instructions in this test. You should also structure the design in such a way that new WordSquares variants can be added in the future (such as the game-modes in requirements 13 and 14).

Rules and requirements:

1. Game participants consists of at least one player and any number of bots.
2. The game-board is a grid with at least one row and at least one column.
3. Random participant starts picking the letter to place on the board, then participants take turns picking.
4. All participants place the picked letter somewhere on their personal game-board.
5. When the game-board is fully populated with letters the score is calculated for each player.
6. Words are identified horizontally, vertically, and forward diagonally.
7. A scrabble dictionary is used for the selected language to identify words.
8. For standard WordSquares scoring the following rules apply:
    a. Words consisting of fewer than 3 letters = 0 points
    b. Words consisting of 3 letters = 1 points
    c. Words with more than 3 letters: (word-length-3) * 2 points
    d. A word can only be counted once when calculating the score.
    e. If a word is used more than once only the highest scoring instance is counted.
9. For scrabble WordSquares scoring the following rules apply:
    a. Letters are assigned a value according to the Scrabble rules for the used language. https://en.wikipedia.org/wiki/Scrabble_letter_distributions
    b. The Double Letter tile doubles the value of the letter placed there
    c. The Triple Letter tile triples the value of the letter placed there
    d. The Double Word tile doubles the value of all words formed using the tile (after letter-multipliers are applied).
    e. The Triple Word tile triples the value of all words formed using the tile (after letter-multipliers are applied).
    f. Multiple word-multipliers used in a word are multiplied together.
    g. Tiles with no modifier are regular tiles and have a multiplier of 1 (e.g. Scrabble WordSquares on regular board)
10. The player with the highest point total is announced as the winner at the end of each match.
11. Scrabble boards where specified special tiles are either pre-placed or randomised can be loaded from the menu.
12. Supported languages have their own letter-points configurations.

Future modifications to the game

13. New game-mode: Boggle Word Squares
    a. Instead of placing letters on a personal board players take turns picking and placing the letter on a shared board.
    b. When all letters are placed, players now compete by finding words in the grid themselves within a time-limit.
    c. The time-limit for the finding the word can be configured in the menu but is set to 60 seconds by default.
    d. Bots can use the algorithm (`checkWords`) which finds all possible words (no need to program a special AI).
    e. This game-mode can use both regular and scrabble-boards, and the score is calculated according to rule 8 & 9.
14. New game-mode: Don't make a word
    a. Players take turns picking letters, but the next player in line places the letter on a shared board.
    b. Players must not form a word longer than two letters in any of the directions specified in rule 6.
    c. A player loses when they form a word longer than two letters while placing their assigned letter.
    d. The game ends when only one player remains (the winner) or when there is no more room for letters on the board (remaining players are then shared winners).
15. Add a way for players to define and load new scrabble-boards with either pre-placed or randomised scrabble-tiles.
16. Add support for additional languages (in addition to English), including dictionary and letter-points configurations.
17. Add support for showing the list of point-scoring words and associated points at the end of the game for each player which can be enabled from the settings menu.

Additional requirements (in addition to rules 1-17)

18. 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).
19. 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

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

Which requirement(s) (rules and requirements 1 – 12 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 18-19 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 – 19 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 18 – 19 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 – 19 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.
- 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 Wosq 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 an English scrabble dictionary are available at: https://staff.www.ltu.se/~qwazi/d7032e2021/

The source code is provided in one file, `VarietyWordSquares.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 VarietyWordSquares` for client – run with: `java VarietyWordSquares client`). The server must be started, and one of the game-modes need to be launched from the menu, before any of the clients are started. The server waits for the online clients to connect before starting the game, and the number of players can be set from the menu.

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 – 12 and any of the additional future modifications that are implemented from requirements 13 – 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 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.

If your console does not print colours, you can find consoles that do here: https://superuser.com/questions/413073/windows-console-with-ansi-colors-handling