Software Construction HS 2021

Instructor: Prof. Dr. Alberto Bacchelli — Assignment 1

Teaching Assistants: Enrico Fregnan, Robin Baettig, Jan Bauer, Deborah Jakobi, Matthias Mylaeus, Joel Ruettiman, Lukas Zehnder — Week 02

To correctly complete this assignment you **must**:

- Carry out the assignment with your team only (unless otherwise stated). You are allowed to discuss solutions with other teams, but each team should come up its own personal solution. A strict plagiarism policy is going to be applied to all the artifacts submitted for evaluation.
- Provide solutions to the exercises. Each solution will consist of source code and its changes and/or explanations (*e.g.*, of decisions taken):
  - The explanations must be written in a PDF file with the name: *Group[id on OLAT]-a[AssignmentNumber].pdf*[1]
  - Source code and its changes, as well as explanations must be pushed to the master branch of your GitHub repository by **Oct 19, 2021 @ 16:00**[2] and the assistant that follows your group has to be tagged in the commit they need to consider.

Exercise 1 - A Bird's-eye View of Pacman (12 pts)

Consider the following Java project: https://github.com/SERG-Delft/jpacman-framework. It is a Pacman-inspired game, developed for teaching testing purposes, which has a nice design to explore and understand.

Use what you learned in Lecture 2 to understand this software system's structure and dynamic behavior. Once you have a clearer understanding of the design of this project, do the following:

1. Draw the structure of the project's packages and classes. You have to: (i) decide the level of abstraction you want to use in this depiction, (ii) use natural language to explain your decision, and (iii) describe what you understood from this depiction of the system (**6 pts**).
2. Draw a call graph, starting from what you deem the most prominent entry point. You have to: (i) decide how many levels you want to have in the call graph, (ii) use natural language to explain your decision, and (iii) explain what you have understood from this call graph about the dynamic behavior of the system. Hint: this project may have several entry points; those contained in an example or test folder/class are hardly the most prominent ones (**6 pts**).

[1] *e.g.*, a correct name would be: *Group1-a1.pdf*.

[2] Solutions sent within the first 24 hours after the deadline will be given 50% of the points they would normally get. Solutions sent after 24 hours from the deadline will not be graded.

Exercise 2 - A Checkers Game - Design (30 pts)

This exercise asks you to *design* a working Checkers game (please see Appendix A, at the end of this assignment, to find the rules to be used for the game). The game's requirements are the following:

Checker's Game Requirements

In this game, two users play Checkers using the same computer and input their moves at each round via the terminal. When the game starts, the program outputs—on the terminal—the board with the pieces in their initial position. The board should be displayed as shown in the following example:

```
    a     b     c     d     e     f     g     h
  +-----------------------------------------------+
8 |  [   ] [R_P] [   ] [R_P] [   ] [R_P] [   ] [R_P] | 8
7 | [R_P] [   ] [R_P] [   ] [R_P] [   ] [R_P] [   ] | 7
6 |  [   ] [R_P] [   ] [R_P] [   ] [R_P] [   ] [R_P] | 6
5 |  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] | 5
4 |  [   ] [   ] [   ] [   ] [   ] [   ] [   ] [   ] | 4
3 | [W_P] [   ] [W_P] [   ] [W_P] [   ] [W_P] [   ] | 3
2 |  [   ] [W_P] [   ] [W_P] [   ] [W_P] [   ] [W_P] | 2
1 | [W_P] [   ] [W_P] [   ] [W_P] [   ] [W_P] [   ] | 1
  +-----------------------------------------------+
    a     b     c     d     e     f     g     h
```

The first character ('`R`' or '`W`') represents the piece's color (Red or White) and the second one the piece's type (i.e., '`P`' = Pawn, '`K`' = King[3]). Each row of the board is indexed by a number (starting from the bottom row with '`1`') and each column is indexed by a character (starting from the left with '`a`').

The game then asks the first player to enter their move on the terminal. Users declare their moves as in the following: `[current piece position]X[future piece position]`; for example by typing `[a3]X[b4]` they want to move a piece from position `a3` into position `b4`. The validity of the move must be checked; if not valid the user has to enter another move until they enter a valid one. After the player inserted a valid move, the program prints an updated version of the board based on the new positions of the pieces.

Afterwards, the program asks the next user to move, unless the game is finished, in which case it prints out who the winner is.

Use what you learned in Lecture 3 to complete the points below:

1. Following the Responsibility Driven Design, start from the game's requirements and rules and derive classes, responsibilities, and collaborations (use CRC cards). Describe each step you make and store the final cards in your answers (**8 pts**).
2. Following the Responsibility Driven Design, describe the *main* classes you designed to be your project in terms of responsibilities and collaborations (**6 pts**).
3. Why do you consider the other classes as less important? Following the Responsibility Driven Design, reflect if some of those non-main classes have similar/little responsibility and could be changed, merged, or removed (**6 pts**).
4. Draw the class diagram of the aforementioned main elements of your game (do not forget to use elements such as parametrized classes or association constrains, if necessary) (**5 pts**).
5. Draw the sequence diagram to describe how the main elements of your game interact (consider asynchrony and constraints, if necessary) (**5 pts**).

[3] there are no Kings at the beginning of the game.

Exercise 3 - A Checkers Game - Implementation (48 pts)

This exercise asks you to implement a working Checkers game. Use what you learned in the first Lectures to complete the following:

1. Implement in Java the game design that you created in the previous exercise (**48 pts**[4]).

[4]To obtain full points, it must be possible to play the entire game on the terminal.

Appendix A: Checkers' Rules

The following rules have been adapted from the Standard Laws of Checkers by RIT:

a. The checkerboard is an 8x8 grid of light and dark squares in the famous "checkerboard" pattern. Each player has a dark square on the far left and a light square on his far right. The double-corner sometimes mentioned is the distinctive pair of dark squares in the near right corner.

b. The checkers to be used shall be round and red and white in color. The pieces shall be placed on the dark squares. The starting position is with each player having twelve pieces, on the twelve dark squares closest to the player's edge of the board.

c. The red player moves first.

d. A player must move each turn. If the player cannot move, the player loses the game.

e. In each turn, a player can make a simple move, a single jump, or a multiple jump move.

   i *Simple move:* Single pieces can move one adjacent square diagonally forward away from the player. A piece can only move to a vacant dark square.

   ii *Single jump move:* A player captures an opponent's piece by jumping over it, diagonally, to an adjacent vacant dark square. The opponent's captured piece is removed from the board. The player can never jump over, even without capturing, one of the player's own pieces. A player cannot jump the same piece twice.

   iii *Multiple jump move:* Within one turn, a player can make a multiple jump move with the same piece by jumping from vacant dark square to vacant dark square. The player must capture one of the opponent's pieces with each jump. The player can capture several pieces with a move of several jumps.

f. If a jump move is possible, the player must make that jump move. A multiple jump move must be completed. The player cannot stop part way through a multiple jump. If the player has a choice of jumps, the player can choose among them, regardless of whether some of them are multiple, or not.

g. When a single piece reaches the row of the board furthest from the player, i.e the king-row, by reason of a simple move, or as the completion of a jump, it becomes a king. This ends the player's turn. The opponent crowns the piece by placing a second piece on top of it.

h. A king follows the same move rules as a single piece except that a king can move and jump diagonally forward away from the player or diagonally backward toward the player. Within one multiple jump move, the jumps can be any combination of forward or backward jumps. At any point, if multiple jumps are available to a king, the player can choose among them.

i. A player who loses all of their pieces to captures loses the game.