EECS 3311 Section Z
Software Design
Lab 2 (100 points), Version 1

Instructor: Song Wang
Release Date: Feb 1st 2022

Due: Feb 18th 2022, 11:59pm on eclass

All your lab submissions must be compilable on the department machines. It is then crucial that should you choose to work on your own machine, you are responsible for testing your project before submitting it for grading. This lab is intended to help you get familiar with the basic OOP design principles.

Check the Amendments section of this document regularly for changes, fixes, and clarifications.

Ask questions on the course forum on the eClass site.

1 Policies

- Your (submitted or un-submitted) solution to this lab (which is not revealed to the public) remains the property of the EECS department. Do not distribute or share your code in any public media (e.g., a non-private Github repository) in any way, shape, or form before you get the permission from your instructors.

- You are required to **work on your own for this lab**. No group partners are allowed.
- When you submit your solution, you claim that it is solely your work. Therefore, it is considered as an violation of academic integrity if you copy or share any parts of your code or documentation.
- When assessing your submission, the instructor and TA may examine your doc/code, and suspicious submissions will be reported to the department/faculty if necessary. We do not tolerate academic dishonesty, so please obey this policy strictly.

- You are entirely responsible for making your submission in time.

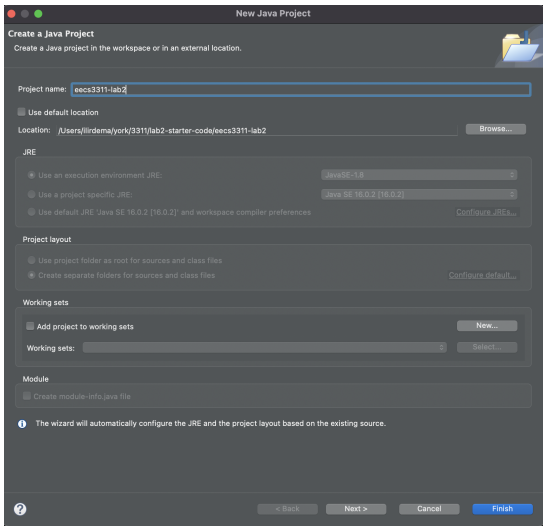
- You may submit multiple times prior to the deadline: only the last submission before the deadline will be graded.
- Practice submitting your project early even before it is in its final form.
- No excuses will be accepted for failing to submit shortly before the deadline.
- Back up your work periodically, so as to minimize the damage should any sort of computer failures occur. You can use a **private** Github repository for your labs/projects.
- The deadline is strict with no excuses.
- Emailing your solutions to the instruction or TAs will not be acceptable.

Amendments

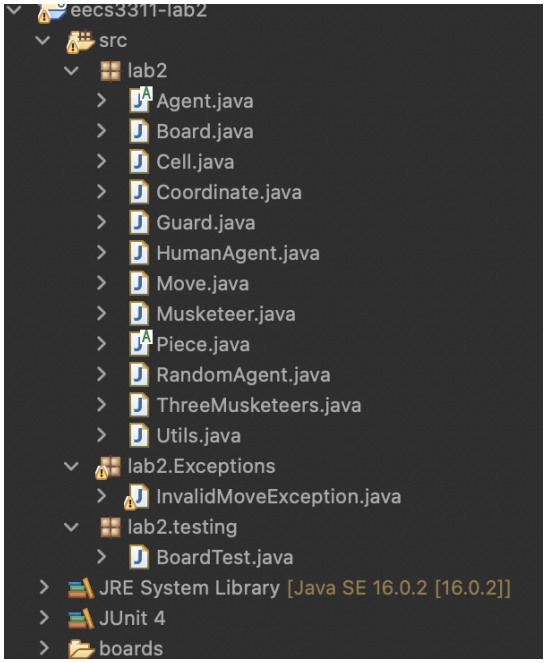
- so far so good

2 Getting Started

- In this lab, we use the latest Eclipse (2021-12 build):`https://www.eclipse.org/downloads/packages/release/2021-12/r/eclipse-ide-java-developers`
- Go to the course eClass page for your section. Under Lab 2, download the file **EECS3311_Lab2.zip** which contains the starter project for this lab.
- Unzip the file, and you should see a directory named **eecs3311-lab2**. It's a Eclipse project.
- You can import this project into your Eclipse as an General Java project.



- The start project should have the following structure and **on default does compile.**



Description

This lab's major learning goal is applying object oriented design in Java and understand the applicable design patterns. In this lab, you will be:

- Designing basic classes
- Ensuring classes are small and do one thing (Single Responsibility Principle)
- Creating class abstractions (Dependency Inversion)
- Using inheritance, composition, and polymorphism
- Identify design patterns

We will be encoding the Three Musketeers Board game. Feel free to play against an AI to gain some intuition about this game.

THE GAME

This game is played on a 5x5 square board with this starting position:

- Starting Position:

```
  | A B C D E
--+----------
1 | O O O O X
2 | O O O O O
3 | O O X O O
4 | O O O O O
5 | X O O O O
```

RULES

- TURNS - The Musketeers always go first. Each player makes one move before passing to their opponent.
- MUSKETEER - On the Musketeers' turn, a musketeer can move to an adjacent orthogonal (neighboring, non-diagonal) cell occupied by a guard, capturing it and occupying that cell.
- GUARD - On the Guards' turn, a guard can move to an adjacent orthogonal empty cell.
- GOALS -
  - The Musketeers win if they cannot capture any guards AND if they are **not** on the same row or column.
  - The guards win if all the musketeers are on the same row or column at any point in the game.

An Example

The Guards' turn:

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |       O
5 | O   O X
```

– Guard moves $D4 \rightarrow E4$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 | O   O X
```

– Musketeers' only move is to capture the guard on $C5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 | O   X
```

– Guard moves $A5 \rightarrow B5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 |   O X
```

– Musketeers can only capture the guard on $B5$

```
  | A B C D E
--+----------
1 |   X
2 |       O
3 |   X   O O
4 |         O
5 |   X
```

– Guards win because 3 musketeers are all on the B column

TASKS

You will be given starter code to implement classes and methods.

Your implementation will include a collection of features, including:

- implement Human VS Human play
- implement Human VS Computer (with Random strategy)

(There is a glossary on some of these terms at the very end of this handout if you are interested in what some of the words mean.)

In your implementation you are allowed/required to:

- may add more private methods (no removal of the provided methods though!)
- may add additional classes
- use static and instance variables/methods as appropriate, with appropriate protections
- appropriately document your code
- make use of a stack to undo moves (or really any basic data structure as you see fit)
- use inheritance, composition, and polymorphism where possible

There should be no need to make any new files of your own.

DEMOS

To help you understand how the game runs, we also provide two demos with two different modes, i.e., Human vs Human and Human vs Computer.

You can find the traces here:

- Demo of Human vs Human
- Demo of Human vs Computer

TODO

ThreeMusketeers.java

- **move(Agent agent): (*Line 134*)**

  Gets a valid move from the given agent (Human/Random), adds a copy of the move using the Move copy constructor to the moves stack for undoing later, then does the move on the board.

- **undoMove(): (*Line 140*)**

  Removes a move from the top of the moves stack and undoes the move on the board.

Board.java

- **getCell(Coordinate coordinate): (*Line 66*)**

  Gets the cell on the board at the given coordinate.

- **getMusketeerCells(): (*Line 81*)**

  Gets all the musketeer cells on the board.

- **getGuardCells(): (*Line 89*)**

  Gets all the guard cells on the board.

- **move(Move move): (*Line 97*)**

  Executes the given valid move on the board.

- **undoMove(Move move): (*Line 105*)**

  Undo the given move on the board. The given move is a copy of the original move. So the cells in the copy are not the same as the cells on the board and the copy has the pieces that were originally in the fromCell and toCell.

- **isValidMove(Move move): (*Line 116*)**

  Checks if the given move is valid according to the rules of the game. (1) the toCell is next to the fromCell. (2) the fromCell piece can move onto the toCell piece.

- **getPossibleCells(): (*Line 125*)**

  Gets all the cells that have pieces that can be moved this turn. Needs to check if the cell has at least one possible destination.

- **getPossibleDestinations(Cell fromCell): (*Line 133*)**

  Gets all the cells that the piece in the given fromCell can move to.

- **getPossibleMoves(): (*Line 141*)**

  Gets all the possible moves that can happen this turn. This function can use the getPossibleCells and getPossibleDestinations functions to help get the list of possible moves.

- **isGameOver(): (*Line 149*)**

  Checks if the game is over and if it is, sets the winner attribute in the Board class to the winner Piece Type.

Guard.java

- **canMoveOnto(Cell cell): (*Line 15*)**

  Returns true if the Guard can move onto the given cell according to the rules of the game, false otherwise.

Musketeer.java

- **canMoveOnto(Cell cell): (*Line 15*)**

  Returns true if the Musketeer can move onto the given cell according to the rules of the game, false otherwise.

HumanAgent.java

- **getMove(): (*Line 15*)**

  Asks the human for a move to be done. This function needs to validate the human input and make sure the move is valid for the piece type that is moving.

RandomAgent.java

- **getMove(): (*Line 14*)**

  Gets a valid random move that can be done on the board.

Testing

Unit tests can be written in the `src/lab2.testing` folder to test each class and the functions you implement. Unit tests created for this lab will not be marked, but will be useful for you to ensure your code works for various cases.

`BoardTest.java` is an example test file, given to help you create more tests for `Board.java`.

Getting Started

Starter Code

Starter code can be downloaded from eclass.

Playing the Game

The first thing you should do is to play the game, in order to get a better understanding of how it works and what you will be creating in this l. Here is a link where you can play a completed version of the game against an AI.

Implementation

Once you have loaded the starter code project into the your workspace, you can find the files in the appropriate source directory.

To gain a better understanding of how the game is pieced together, try applying a "top-down" approach to reading the files (this means start from an overview of how the game works and then look at the implementation details of the smaller components). Start by opening `ThreeMusketeers.java`. Running this file will launch the game itself in the console of your editor.

Once you have done so, delve into the components of the game, such as the board the game is played on (`Board.java`), then the game pieces that fill the board (`Guard.java` and `Musketeer.java`), and the `Agent`s.

GLOSSARY

- Strategy: In game theory, a player's strategy is an algorithm of how they decide to play (Wikipedia, 2021). An example of this could be to copy whatever the opponent did on their turn in chess. This would be a strategy, but perhaps not the best strategy to win the game of chess.
- Random (Strategy): Moving randomly, "without a strategy", per se.
- Heuristic: A gauge that does not necessarily lead to a victory, but could provide some insight towards victory. An example could be to mentally keep an amount of moves you need to check your opponent's king in chess.

3 Submission

To get ready to submit:

- Close Eclipse
- Zip your lab2 project with name 'EECS3311-Lab2.zip'.

By the due date, submit it on eClass.