Project Report - Monopoly

COMP2011 OBJECT ORIENTED PROGRAMMING

Group 08

Shi Yuanjing | Jiao Ying | Zheng Yufei

2016/12/1

Content

1. Introduction

This document describes the design and implementation of the Monopoly game of group 8. The project is part of the course COMP2021 Object-Oriented Programming at PolyU. The following sections introduce the Monopoly game before describing the requirements that were implemented and the design decisions taken. The last section describes the available commands in the game.

2. The Monopoly Game

Monopoly is a board game. Players toss die to decide the numbers of squares they can go in each turn. All the players moving around buying house square to collect tax from their opponents, in order to drive opponents into bankruptcy and control the entire economy. Squares on the game board may also cause changes on players' properties. Players need to pay money when they land on tax square, house squares owned by others or they want to get out of jail. Passing through go square can gain 1500 salary each time. And the change square can make players either gain or lose money. The game ends either if there is only one player left or after 100 rounds. The winner is the player with the most money after the end of the game. Ties (multiple winners) are possible.

2.1 Requirements Describe

Req-1. The game supports a command line user interface.

<u>class Monopoly, “main()”:</u>

- Ask player to choose 1 to see the board map, 2 to read the instructions, 3 to start game;
- Ask player to input the total player they want (between 2 and 6), and choose the quantity and position of the players to be set as AI player(s).

class Monopoly, “startGame()”:

For non-AI players, the player will be asked to choose 1 for continue the game, 2 for reading report, 3 for auto and 4 for retire from the game every step.

class HouseSquare, “squareMove()”:

The player will be asked whether they want to buy the unoccupied house square when they land on it.(1 for yes, 2 for no.)

class JailSquare, “getOutJail()”：

Players in jail will be asked to choose whether they want to pay the fine or stay in jail. (1. for pay the fine, 2 for still stay in jail.)

class “ConsoleIO()”:

This class is aim to inherit feature from class “IOService” and achieve all interactive feature, like get user input and display corresponding user output.

Req-2. The game supports both human players and computer players.

class Monopoly, main():

Ask player to choose 1 to see the board map, 2 to read the instructions, 3 to start game;

Ask player to input the total player they want (between 2 and 6), and choose the quantity and position of the players to be set as AI player(s).

class HouseSquare, squareMove():

The human player will be asked whether they want to buy the unoccupied house square when they land on it.(1 for yes, 2 for no.) AI player(s) will choose buy or not buy randomly.

class JailSquare, getOutJail()：

Human players in jail will be asked to choose whether they want to pay the fine or stay in jail. (1 for pay the fine, 2 for still stay in jail.) AI player(s) will choose pay the fine or stay in jail randomly.

<u>class Player():</u>

Contains a Boolean field isAI to distinguish human players and AI players.

Req-3. At each step of a human player, the game asks for a command from the player. Supported commands must include: continue, report, auto, and retire.

<u>class Monopoly, startGame():</u>

For non-AI players, the player will be asked to choose 1 for continue the game, 2 for reading report, 3 for auto and 4 for retire from the game every step.

Req-4. Upon receiving the command continue, the game lets the player take his/her turn.

<u>class Monopoly, startGame():</u>

If the player chooses 1 for continue, he will still be a human player, which means he can make choose of buying squares, getting out of jail and will be still asked for a command every step.

Req-5. Upon receiving the command report, the game prints out the information of each square on the board and each player's location on the board.

<u>class Monopoly, startGame():</u>

Let players to choose 2 to read the report. Call the getReport() function in class Board.

class Board, getReport():

Print out all the information in array square. Call the getCurrentPositon() function in class Player to print the current position of players.

class Player, getCurrentPosition():

Return the current position of player.

Req-6. Upon receiving the command auto, the game uses a computer player to takes over the player’s place. A computer player always continues until the end of the game and makes decisions in Req-8 randomly.

class Monopoly, startGame():

Let players to choose 3 to auto play. Call the getCurrentPlayer() function in class Board.

class Board, getCurrentPlayer():

Return current player. Call setAI() function in class Player.

class Player, setAI():

Chance the isAI field in class Player to be true, which means all the command asked will be decided randomly.

Req-7. Upon receiving the command retire, the game makes the player retire (i.e., the player leaves the game with all his/her property becoming unowned).

class Monopoly, startGame():

Let players to choose 4 to retire. Call the getCurrentPlayer() function in class Board.

class Board, getCurrentPlayer():

Return current player. Call the setBreakOut() function in class Player.

class Player, setBreakOut():

Set the private Boolean field to be true and all the property becoming unowned.

Req-8. When a human player can decide whether to buy a property or to pay the fine to get out of Jail, the game asks for the player’s input.

class HouseSquare, squareMove():

The player will be asked whether they want to buy the unoccupied house square when they land on it.(1 for yes, 2 for no.)

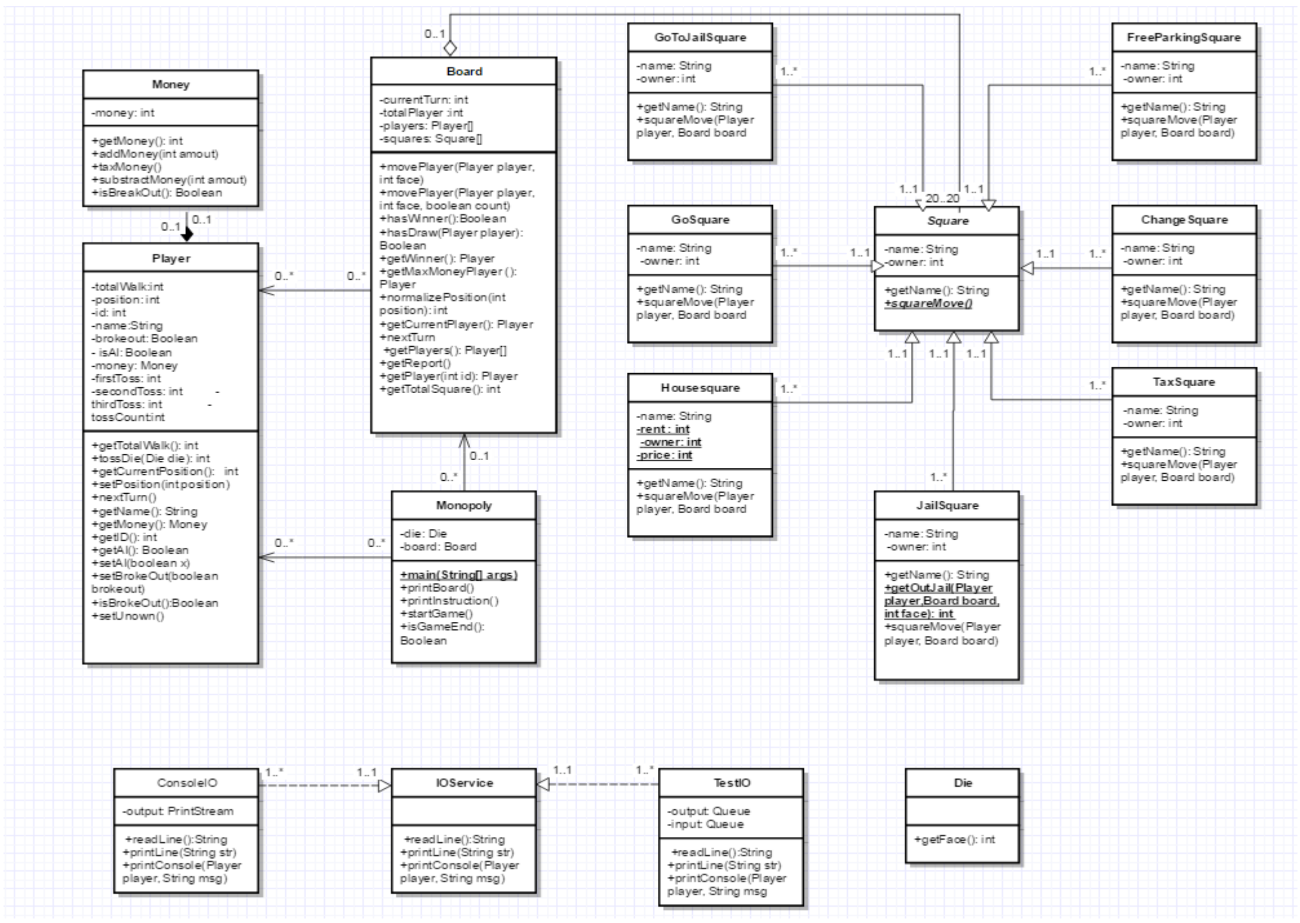
class JailSquare, getOutJail()：

Players in jail will be asked to choose whether they want to pay the fine or stay in jail. (1 for pay the fine, 2 for still stay in jail.)

class ConsoleIO:

Used to print the command and get the different kinds of inputs from players.

2.2 Design



Our project contains 16 classes: Monopoly, Board, Player, Money, Square(abstract), GoToJailSquare, GoSquare, HouseSquare, JailSquare, TaxSquare, ChanceSquare, FreeParkingSquare, ConsoleIO, StubIO and one interface IOService.

Monopoly class which contains main() function works as an entry of this game.

Board and Player class are important classes. The board class contains current turn number, all information of squares in Square[] squares and lots of functions to control players. The movePlayer() and nomalizePosition() functions move players and set the new positions for players. The end of game can also be determined by Board class using hasWinner(), hasDraw(), getWinner() and getMaxMoneyPlayer() functions. getReport() funtion helps generate the report of current condition of this game when human choose to read report. Player class contains detail information of players like total walk, id, position, name, money, isAI etc. Get functions in Player class return the information of player. setAI() and setBreakOut()

are also necessary for changing the state of player. And tossDie() is used to get the result of face. Money class is closely related to Player class which can be regarded as a wallet of players. All functions changing players' properties are included in this class.

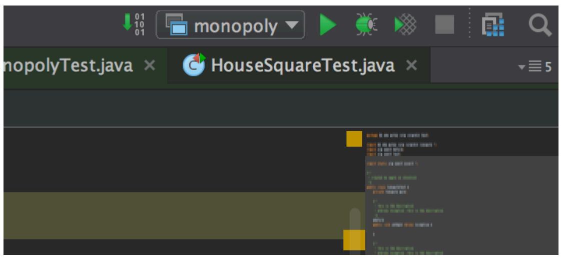
The design of squares classes uses inherit. The super class is abstract class Square. And all other class inherit constructor and getName() method from it. All the subclasses need to override the squareMove() method. The HouseSquare and JailSquare have different implements. Fields rent, price and owner is added to HouseSquare since those squares can be bought and owned by players and get rent when others land on it. A new method called getOutOfJail() is added to let human players to decide whether to get out of jail or not.

The design of interface IOService and two subclasses named ConsoleIO and TestIO uses polymorphic. The ConsoleIO and TestIO classes overriding abstract methods in interface IOService. ConsoleIO calss is used to pay the game normally: get users' input from keyboard and use the PrintStream field to print results on the screen. As for the TestIO class is created for test use. The program can read input from "input" field in this class and the keyboard input is no longer needed.

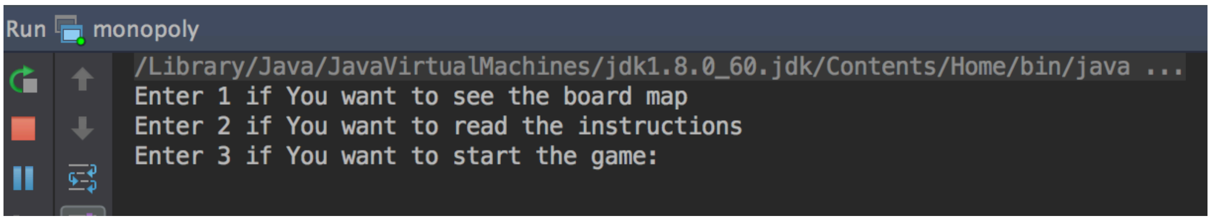
Die class is used to generate random number as the result of toss die. PauseTest class is used to get different inputs and print commands. PrintBoard class is used to print the board.

3. Quick Start Guide

(1) Edit the configuration of this project to "monopoly" and run the program.

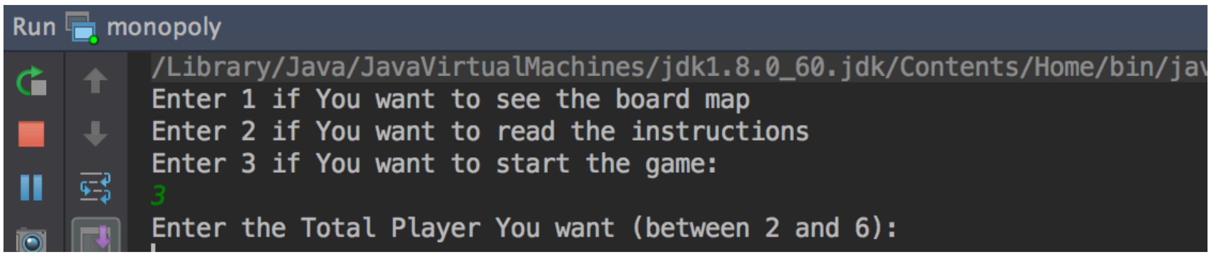


(2) Choose 1 to see the board map, 2 to read the instructions, 3 to start game.



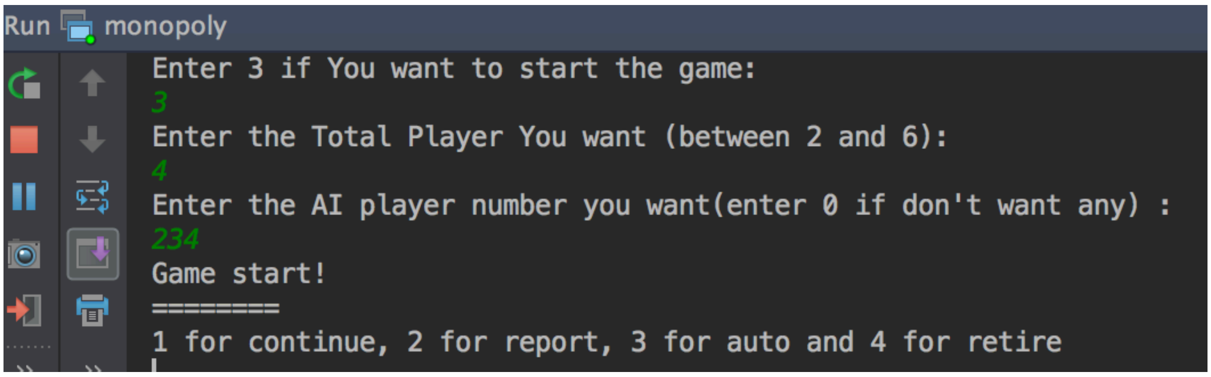
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/java ...
Enter 1 if You want to see the board map
Enter 2 if You want to read the instructions
Enter 3 if You want to start the game:
```

(3) Input the total player they want (between 2 and 6), and choose the quantity(s) and position(s) of the players to be set as AI player(s), if no AI players is needed, simply press 0.



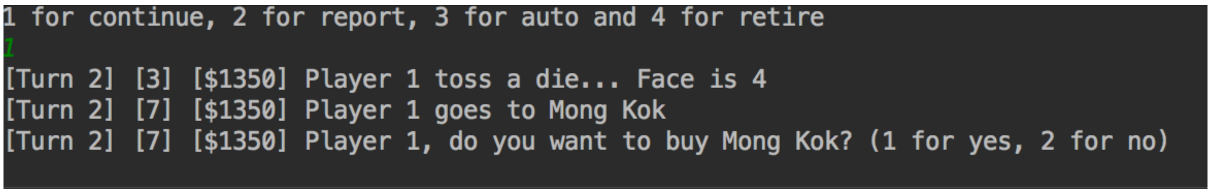
```
/Library/Java/JavaVirtualMachines/jdk1.8.0_60.jdk/Contents/Home/bin/jav
Enter 1 if You want to see the board map
Enter 2 if You want to read the instructions
Enter 3 if You want to start the game:
3
Enter the Total Player You want (between 2 and 6):
```

(4) The player will be asked to choose 1 for continue the game, 2 for reading report, 3 for auto and 4 for retire from the game every step. In our screenshot, we entered 4 as total number of players and player 2, 3, 4 would be AIs.



```
Enter 3 if You want to start the game:
3
Enter the Total Player You want (between 2 and 6):
4
Enter the AI player number you want(enter 0 if don't want any) :
234
Game start!
========
1 for continue, 2 for report, 3 for auto and 4 for retire
```

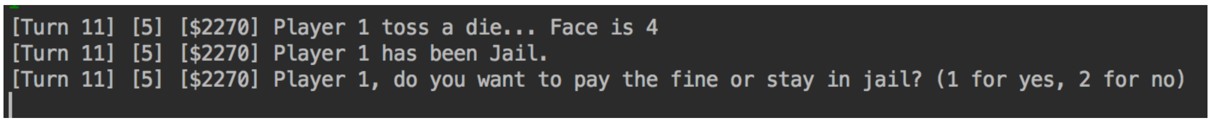
(5) The player will be asked whether they want to buy the unoccupied house square when they land on it. (1 for yes, 2 for no.)



```
1 for continue, 2 for report, 3 for auto and 4 for retire
1
[Turn 2] [3] [$1350] Player 1 toss a die... Face is 4
[Turn 2] [7] [$1350] Player 1 goes to Mong Kok
[Turn 2] [7] [$1350] Player 1, do you want to buy Mong Kok? (1 for yes, 2 for no)
```

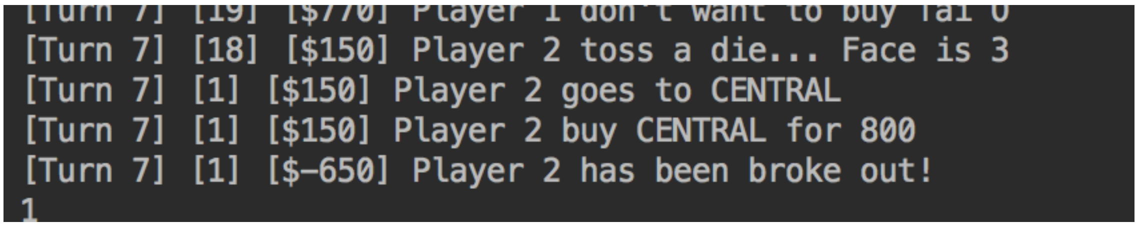
(6) Human players in jail will be asked to choose whether they want to pay the fine or stay in jail.

(1 for pay the fine, 2 for still stay in jail.) AI player(s) will choose pay the fine or stay in jail randomly.

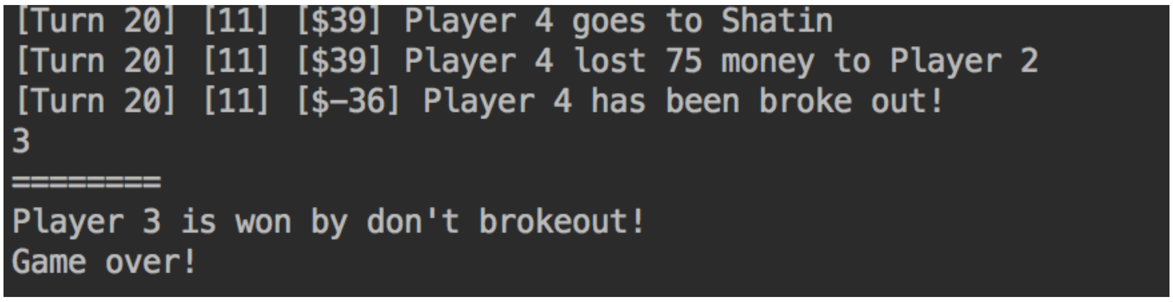

```
[Turn 11] [5] [$2270] Player 1 toss a die... Face is 4
[Turn 11] [5] [$2270] Player 1 has been Jail.
[Turn 11] [5] [$2270] Player 1, do you want to pay the fine or stay in jail? (1 for yes, 2 for no)
```

(7) If one player break out, there would be a message on the screen and other players would go on

playing.

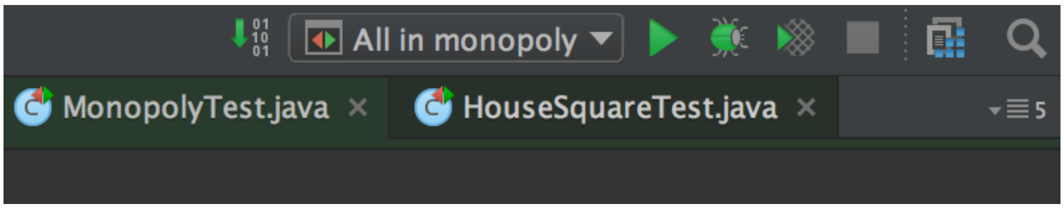

```
[Turn 7] [18] [$150] Player 2 toss a die... Face is 3
[Turn 7] [1] [$150] Player 2 goes to CENTRAL
[Turn 7] [1] [$150] Player 2 buy CENTRAL for 800
[Turn 7] [1] [$-650] Player 2 has been broke out!
1
```

(8) The winner information with be shown before game ends.

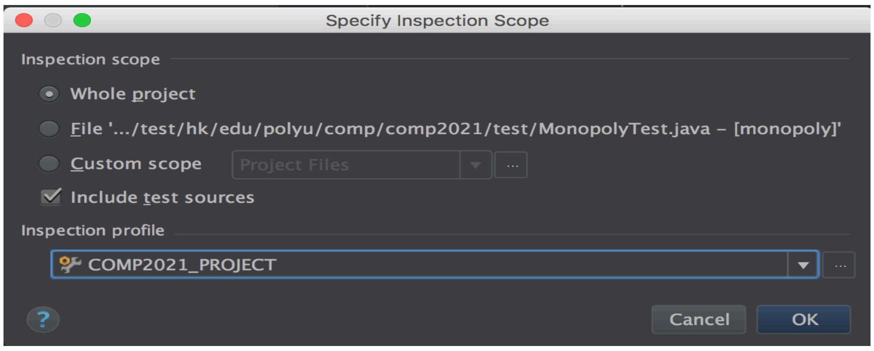

```
[Turn 20] [11] [$39] Player 4 goes to Shatin
[Turn 20] [11] [$39] Player 4 lost 75 money to Player 2
[Turn 20] [11] [$-36] Player 4 has been broke out!
3
========
Player 3 is won by don't brokeout!
Game over!
```

4. Test Usage

1. Simply edit the configuration into



2. Run “Inspect Code” in “Analyze” and the choose the Inspection File-“COMP2021-PROJECT”


Specify Inspection Scope

Inspection scope

Whole project

File '.../test/hk/edu/polyu/comp/comp2021/test/MonopolyTest.java – [monopoly]'

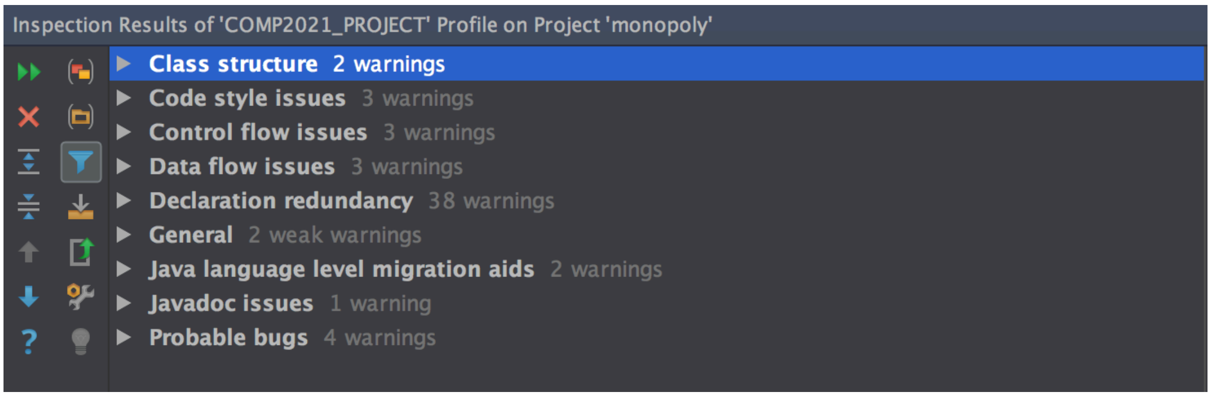
Custom scope Project Files

Include test sources

Inspection profile

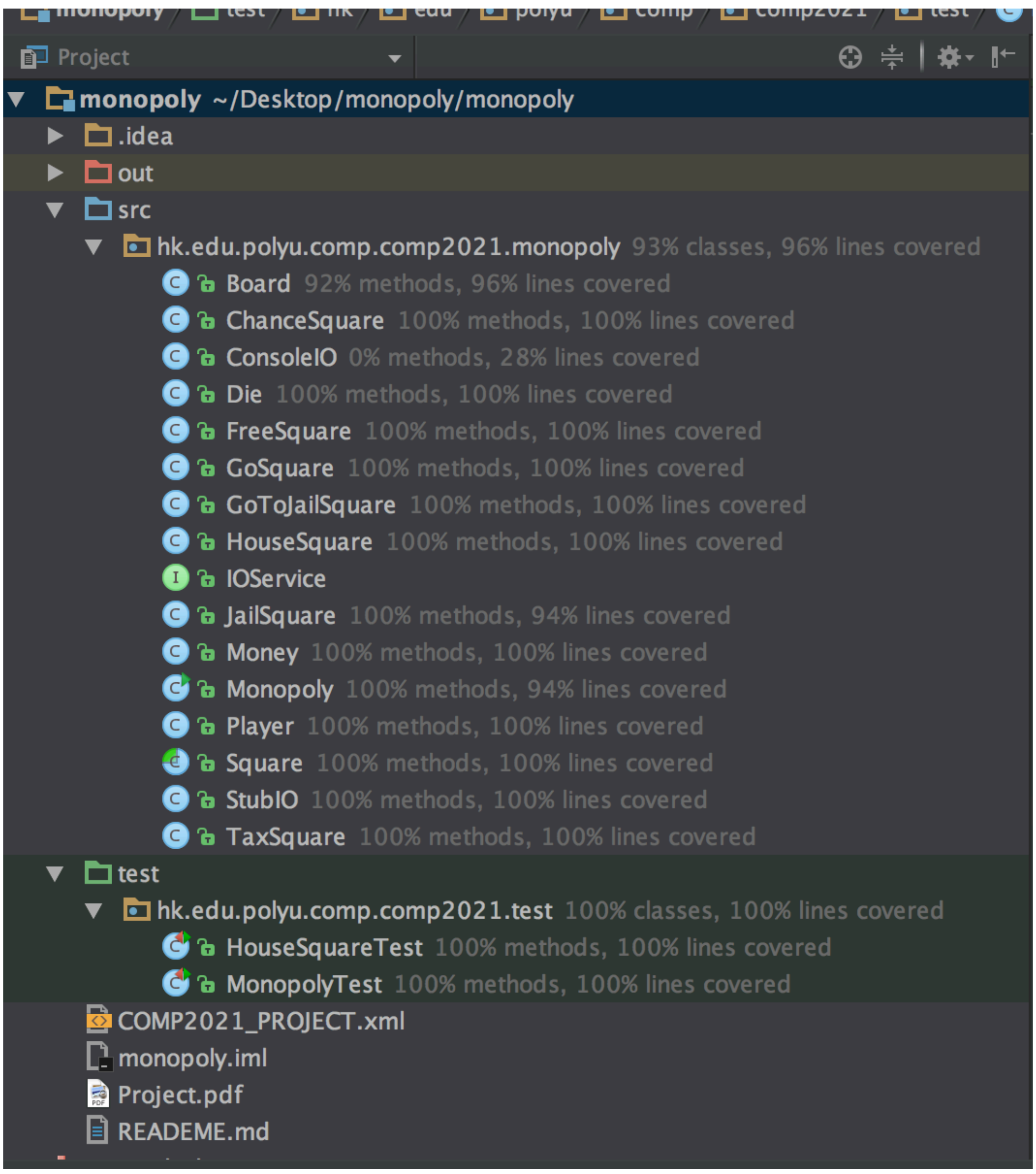
COMP2021_PROJECT

3. The following message would be displayed (no errors!):



```
Inspection Results of 'COMP2021_PROJECT' Profile on Project 'monopoly'
▶ Class structure  2 warnings
▶ Code style issues  3 warnings
▶ Control flow issues  3 warnings
▶ Data flow issues  3 warnings
▶ Declaration redundancy  38 warnings
▶ General  2 weak warnings
▶ Java language level migration aids  2 warnings
▶ Javadoc issues  1 warning
▶ Probable bugs  4 warnings
```

4. Run "All in monopoly" test class with coverage and the following message would be displayed:



```
Project
▼ monopoly ~/Desktop/monopoly/monopoly
  ▶ .idea
  ▶ out
  ▼ src
    ▼ hk.edu.polyu.comp.comp2021.monopoly 93% classes, 96% lines covered
        Board 92% methods, 96% lines covered
        ChanceSquare 100% methods, 100% lines covered
        ConsoleIO 0% methods, 28% lines covered
        Die 100% methods, 100% lines covered
        FreeSquare 100% methods, 100% lines covered
        GoSquare 100% methods, 100% lines covered
        GoToJailSquare 100% methods, 100% lines covered
        HouseSquare 100% methods, 100% lines covered
        IOService
        JailSquare 100% methods, 94% lines covered
        Money 100% methods, 100% lines covered
        Monopoly 100% methods, 94% lines covered
        Player 100% methods, 100% lines covered
        Square 100% methods, 100% lines covered
        StubIO 100% methods, 100% lines covered
        TaxSquare 100% methods, 100% lines covered
  ▼ test
    ▼ hk.edu.polyu.comp.comp2021.test 100% classes, 100% lines covered
        HouseSquareTest 100% methods, 100% lines covered
        MonopolyTest 100% methods, 100% lines covered
    COMP2021_PROJECT.xml
    monopoly.iml
    Project.pdf
    READEME.md
```

Hopefully, the total coverage of our test class would be 96% and some method in ConsoleIO() was not tested since we use StubIO() in our test case. And we would attach all test data in index with order.

5. Index

1. StubIO constructor, in which the test data is presented:

```
public StubIO(){
    input.add("1");
    input.add("2");
    input.add("3");
    input.add("6");
    input.add("1234");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("4");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("2");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("1");
    input.add("2");
    input.add("3");
    input.add("1");
    input.add("2");
    input.add("1");
    input.add("1");
    input.add("4");


}
```