

BBM104 Introduction to Programming Laboratory II

ASSIGNMENT II

Due Date: 01/04/2022 (23:59)

Subject: Inheritance and Polymorphism

1. INTRODUCTION

In this experiment, you are expected to implement non graphical Java version of the classic Monopoly game with the given rule. Main focus point of this experiment is to get you familiar with polymorphism and inheritance.

Inheritance is one of the core concepts of object-oriented programming (OOP) languages. It is a mechanism where you can derive a class from another class for a hierarchy of classes that share a set of attributes and methods.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in object oriented design occurs when a parent class reference is used to refer to a child class object.

A Java abstract class is a class which cannot be instantiated, meaning you cannot create new instances of an abstract class. The purpose of an abstract class is to function as a base for the subclasses. Implementing the program without using the benefits of polymorphism and abstract classes will be penalized.

2. THE PROBLEM

In this experiment you are expected to implement the monopoly game. The game consists of a sequential set of 40 squares containing 28 properties, 6 action squares (3 "**chance**", 3 "**community chest**"), 2 tax squares, "**go**", "**jail**", "**free parking**", and "**go to jail**". Every time a player lands on a square, she/he takes a different action if this square is one of the properties. For the other squares, the player takes a different action depend on the type of the square. For example, if she/he lands on a "**tax square**", she/he must pay a certain amount of money to the banker, but if she/he lands on an "**action square**" she/he must draw a card from the appropriate pile and she/he must follow the instructions on the card. In this game, properties are classified into **Land**, **Company** and **Rail Roads**. Users are classified into **Player** and **Banker** and they have name and money attributes.

Your task is to design the game with the given rules by using polymorphism and inheritance. You must use these concepts while designing your program.

You will be given two JSON formatted input files:

a) Properties file

b) Cards file

JSON is a format for storing data in the files for users (see Appendix A). You are supposed to use JSON. simple library, which is a simple Java library for reading and writing JSON formatted files (with full compliance with JSON specification (RFC4627)).

a) Properties File

The properties are taken from a property file. **Land, Railroads, Company** are numerated as 1,2,3 respectively. This file includes information for all the properties in the game. The following information is provided for each property: property id (the position of this property on board), name of the property, the cost of the property.

A sample property file is given as follows (Figure 1):

```
{
  "1": [
    {"id": "2", "name": "Old Kent Road", "cost": "600"},
    {"id": "4", "name": "Whitechapel Road", "cost": "600"},
    {"id": "7", "name": "The Angel Islington", "cost": "1000"},
    {"id": "9", "name": "Euston Road", "cost": "1000"},
    {"id": "10", "name": "Pentonville Road", "cost": "1200"},
    {"id": "12", "name": "Pall Mall", "cost": "1400"},
    {"id": "14", "name": "Whitehall", "cost": "1400"},
    {"id": "15", "name": "Northumberland Avenue", "cost": "1600"},
    {"id": "17", "name": "Bow Street", "cost": "1800"},
    {"id": "19", "name": "Marlborough Street", "cost": "1800"},
    {"id": "20", "name": "Vine Street", "cost": "2000"},
    {"id": "22", "name": "The Strand", "cost": "2200"},
    {"id": "24", "name": "Fleet Street", "cost": "2200"},
    {"id": "25", "name": "Trafalgar Square", "cost": "2400"},
    {"id": "27", "name": "Leicester Square", "cost": "2600"},
    {"id": "28", "name": "Coventry Street", "cost": "2600"},
    {"id": "30", "name": "Piccadilly", "cost": "2800"},
    {"id": "32", "name": "Regent Street", "cost": "3000"},
    {"id": "33", "name": "Oxford Street", "cost": "3000"},
    {"id": "35", "name": "Bond Street", "cost": "3200"},
    {"id": "38", "name": "Park Lane", "cost": "3500"},
    {"id": "40", "name": "Mayfair", "cost": "4000"}
  ],
  "2": [
    {"id": "6", "name": "Kings Cross Station", "cost": "2000"},
    {"id": "16", "name": "Marylebone Station", "cost": "2000"},
    {"id": "26", "name": "Fenchurch St Station", "cost": "2000"},
    {"id": "36", "name": "Liverpool Street Station", "cost": "2000"}
  ],
  "3": [
    {"id": "13", "name": "Electric Company", "cost": "1500"},
    {"id": "29", "name": "Water Works", "cost": "1500"}
  ]
}
```

Figure 1: Property File

b) Cards File

This file involves all the information about the cards in the game, which are divided into **Chance List** and **Community Chest List**. A sample cards file is given below:

```
{
  "chanceList":[
    {"item":"Advance to Go (Collect $200)"},
    {"item":"Advance to Leicester Square"},
    {"item":"Go back 3 spaces"},
    {"item":"Pay poor tax of $15"},
    {"item":"Your building loan matures - collect $150"},
    {"item":"You have won a crossword competition - collect $100 "}
  ],
  "communityChestList":[
    {"item":"Advance to Go (Collect $200)"},
    {"item":"Bank error in your favor - collect $75"},
    {"item":"Doctor's fees - Pay $50"},
    {"item":"It is your birthday Collect $10 from each player"},
    {"item":"Grand Opera Night - collect $50 from every player for opening night seats"},
    {"item":"Income Tax refund - collect $20"},
    {"item":"Life Insurance Matures - collect $100"},
    {"item":"Pay Hospital Fees of $100"},
    {"item":"Pay School Fees of $50"},
    {"item":"You inherit $100"},
    {"item":"From sale of stock you get $50"}
  ]
}
```

Figure 2: Cards File

The game flows as follows: You will read the properties file to have a list of properties in the game. You will read the cards file to have a list of cards to draw in any action square (chance vs community chest). Your players will take the actions written in the commands file and you will write the results of the commands in an output file. In the commands, there will be also a 'show' command. You will write the players' properties, money and banker's money in an output file whenever you come across a show command. At the end, the result of the game will be written in the same output file as if a show command is met.

2.1 General Rules of the Game

- There are only two players and a banker in the game. They have attributes like name and money. Players have got 15000 TL and banker has got 100000 TL at the beginning of the game.
- Player moves to new square by rolling a dice. When she/he lands on a "property" square if any player does not own the property, player can buy this property if she/he can afford to buy the property.
- If any player owns the property where the current player landed on, the current player must pay for the rent of the property.
- If the property is a land, the rent is calculated based on the cost of the land (for 0-2000 rent is 40%, for 2001-3000 30%, for 3001-4000 35% of the cost). If the property is a company, the rent amount is 4xdice. If the property is a railroad, the rent of the property is 25*(the number of the railroads other players has) TL.

- If the player lands on an "action" square, she/he draws a card and plays depending on the instructions on the card.
- Cards are selected sequentially from the given card list.
- If the player lands on a "jail" square, she/he must wait for 3 times without playing her/his move.
- If the player lands on a "tax square", she/he must pay for 100 TL to the banker.
- If the player lands on a "free parking" square, she/he waits for a tour without playing.
- If the player lands on a "go to jail" square, she/he must go to the jail square.
- If she passes by a "go" square, she/he can't take money for passing a go square.
- If the player lands on or passes by a "go" square, she/he takes 200 TL from the banker.
- The game will be finished when all the commands in the command file are played or any player goes bankrupt.
- The game board is circle, the "go" square (the 1st square) follows the 40th square on the board (see Appendix B).

2.2. Commands

This file includes the commands in the game. Each command consists of a player id of the player to be moved, which is followed by an integer that is the number on the rolled dice. Id and the number are delimited by semicolons:

```
[Player name]; [dice]
[Player name]; [dice]
[Player name]; [dice]
show()
.....
```

An example command file is given below:

```

Player 1;9
Player 2;5
show()
Player 1;4
Player 2;4
Player 1;4
Player 2;6
Player 1;10
Player 2;7
Player 1;5
Player 2;3

```

After reading the input files, your program will start reading the commands file line by line and execute the moves of the players. The program will exit when it reaches the end of the commands file or any player's money is zero or under zero.

2.3.Output

Once all the lines in the command file are performed, your program will create and output the current status of the players on the board and players' money as shown in the figure given below:

```

[Player]tab[Dice]tab[New Position]tab[Player1 Money]tab[Player2 Money]tab[Processing]
Player 1      9      10      13800  15000  Player 1 bought Pentonville Road
Player 2      5      6      13800  13000  Player 2 bought King Cross Station
-----

Player 1      13800  have: Pentonville Road
Player 2      13000  have: King Cross Station
Banker  103200
Winner  Player 1
-----

Player 1      4      14      12400  13000  Player 1 bought Whitehall
Player 2      4      10      12880  12520  Player 2 paid rent for Pentonville Road
Player 1      4      1      13080  12520  Player 1 draw Community Chest -advance
to go
Player 2      6      16      13080  10520  Player 2 bought Marylebone Station
Player 1      10     11      13080  10520  Player 1 went to jail
Player 2      7      1      13080  10720  Player 2 advance to go (collect 200)
Player 1      5      11      13080  10720  Player 1 in jail (count=1)
Player 2      3      4      13080  10120  Player 2 bought Whitechapel Road
-----

Player 1      13080  have: Pentonville Road, Whitehall
Player 2      10120  have: King Cross Station, Marylebone Station, Whitechapel Road
Banker  106800
Winner  Player 1
-----

```

3. Grading and Evaluation

- Your work will be graded over a maximum of 100 points.
- Your total score will be partial according to the grading policy stated below.

Reading the file contents and compile	10p
Correct output	70p
Code design, clean and readable code, algorithmic perspective, comments	20p

Execution and Test

The input files (command.txt) should be given as an argument. Upload your java files to your server account (dev.cs.hacettepe.edu.tr)

- Upload your java files to your server account (dev.cs.hacettepe.edu.tr)
- Compile your code (javac *.java, or javac Main.java)
- Run your program (java Main command.txt)
- Control your output file (monitoring.txt) and format.

Note: You should upload src file to dev with json-simple.jar. And you should specify where the jar file and main class by using classpath(-cp) command.

Usage example:

```
> javac -cp "../json-simple.jar" Main .java
```

```
> java -cp "../json-simple.jar" Main command.txt
```

Submit Format

File hierarchy must be zipped before submitted (Not .rar, only .zip files are supported by the system)

<student id>.zip

- src

- Main.java (required)

- *.java)

Late Policy

You may use up to three extension days for the assignment. But each extension day will bring about additional 10% degradation for evaluation of the assignment.

Notes and Constraints

You should obey the constraints described below. Otherwise your experiment will not be evaluated as well as you expected.

- The assignment must be original, individual work. Downloaded or modified source codes will be considered as cheating. Also the students who share their works will be punished in the same way.
- You can ask your questions through course's piazza group and you are supposed to be aware of everything discussed in the piazza group. General discussion of the problem is allowed, but DO NOT SHARE answers, algorithms, source codes.
- Don't forget to write comments of your codes when necessary.
- The names of classes', attributes' and methods' should obey to Java naming convention.
- Save all work until the assignment is graded.
- Compile your code on DEV server before submitting your work to make sure it compiles without any problems on our server.
- You are responsible for a correct model design. Your design should be accurate.
- Do not miss the deadline. Submission will be end at 01/04/2022 23:59:59.
- You will use online submission system to submit your experiments. <https://submit.cs.hacettepe.edu.tr/> No other submission method (email or etc.) will be accepted. Do not submit any file via e-mail related with this assignment.

Appendix A.

JSON (JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is the most common data format used for asynchronous browser/server communication, largely replacing XML which is used by AJAX. There are several APIs for writing or reading JSON formatted objects. [1]

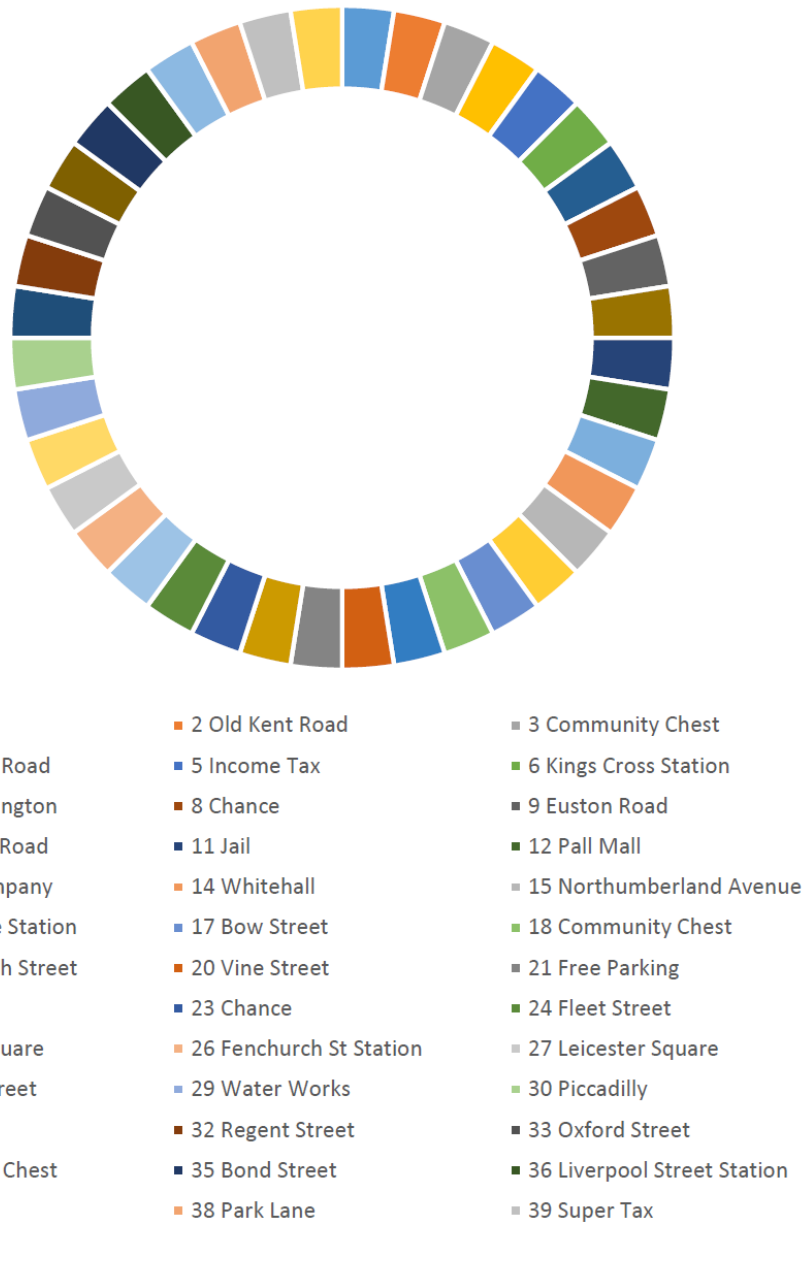
JSON Example

```
{"employees":[
```

```
{ "firstName": "John", "lastName": "Doe"},  
  { "firstName": "Anna", "lastName": "Smith"},  
  { "firstName": "Peter", "lastName": "Jones"}  
}]
```

Appendix B.

The order of squares is given to help you.



References

- [1] <https://en.wikipedia.org/wiki/JSON>
- [2] <http://tutorials.jenkov.com/java/abstract-classes.html>
- [3] <http://www.mkyong.com/java/json-simple-example-read-and-write-json/>