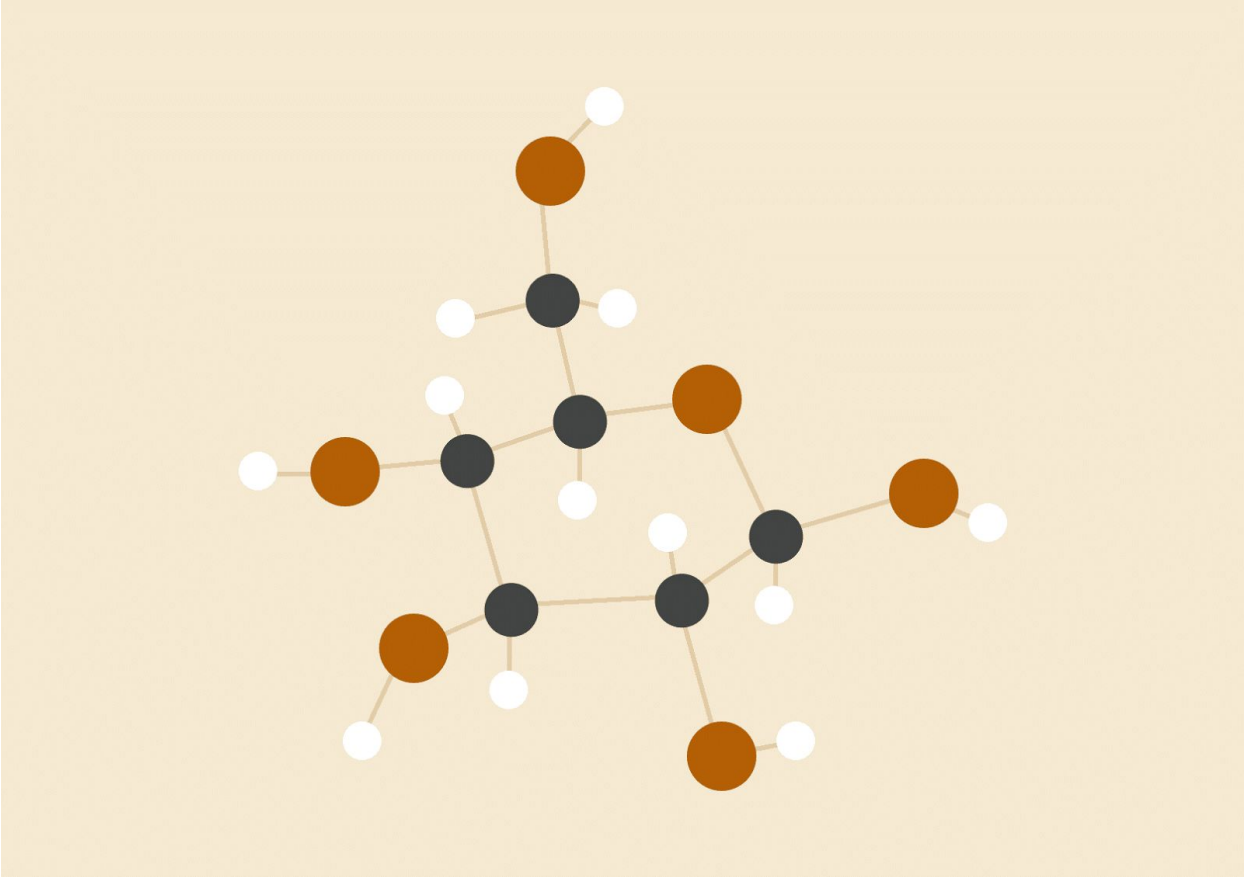
FINAL REPORT

Final report of 3rd assignment.



Metehan YILDIRIM

21427507

Subject : Polymorphism and Abstract Classes

INTRODUCTION

In this assignment we are expected to implement a made-up game of Monopoly which there are two players and one banker. We are expected to use inheritance, abstraction and polymorphism extensively. The game goes on until the commands end or one player goes bankrupt. We need to keep track of their situations such as money, properties etc. Properties consist of land, company and railroads. There are also chance and chest squares which when landed on you take a card and do what it says there. Then there are 2 jail squares. If you land on them you are banned from playing 3 rounds although you will throw your dice anyway. There are also tax squares which if you land on them you pay 100$. There is go square which if you pass it you take 200$. Winner is decided by the total amounts of money.

PROBLEM

First of all we need to represent the boards in an array. But all elements in the array have different properties. We need to find a way to represent these different properties in the array. The elements do what it's supposed to do without we knowing what it is. And we need some class to represent our players and our banker. We need to define some actions on these properties which affects our players and banker.

SOLUTION

To represent objects with different properties in an array we need to use the concept of polymorphism. To achieve this we should have a square class which is the mother of everything. Under this we should have a class for all the properties such as land, company and railroads. I named this class PropertySQ. Other than that we should have a class for pulling chest and chance cards. Mother of these two is named ActionSQ. For taxes we need a class named TaxSQ. For jails we need a class named Jail. Now we need to define players I did this with a class named Player. This class consists of our player and our bankers.

ALGORITHM

First let me talk about the data structures in my program. First of all I have a mother class named Square for holding all of my squares. Children of this class are; ActionSQ, PropertySQ, Jail, TaxSQ, FreePark. Under ActionSQ I have 2 classes named ChestSQ, ChanceSQ. Under PropertySQ there are 3 classes namely; Land, RailRoad and Company. Let me go over these

- Square : Mother class. Has abstract function “landedOn(Player)”.
  - ActionSQ : This class is for our chance and chest squares. Has abstract function namely “takeACard(Player)” which is inside “landedOn(Player)” when called.
    - ChestSQ : This is for pulling our chest cards. Implemented function “takeAcard(Player)”. takeACard(Player) : Pulls a card from the singleton object ChestList. And do what that object returns. And returns the action as a string for the output.
    - ChanceSQ : This is for pulling our chance cards. Implemented function “takeAcard(Player)”. takeACard(Player): Pulls a card from the singleton object ChanceList. And do what that object returns. And returns the action as a string for the output.
  - PropertySQ : This class is for our properties namely; Land, Company and Railroad. Has abstract function “landedOn(Player)” which is from Square. And this class has a variable named price for the buying of the property.
    - Land: This class is for our lands. A variable named rent is added to this class. Which is calculated from the price in this class. Implemented function “landedOn(Player)”. landedOn(Player) : This function first controls if property is owned by the other player if it is money is taken from the first player and given to the second player. If it’s player’s own property nothing is done. If it is not owned player buys it. This returns the result as a string for the output.
    - Company: This class is for our companies. BEWARE: “landedOn(Player)” is not implemented in this class because this class also needs to pass the dice to the function so I implemented “landedOnCompany(Player,dice)”.
    - RailRoad: This class is for our railroads. Implemented

"landedOn(Player)". landedOn(Player): This function checks how many other railroads are owned and calculates a price according to it.

- Jail: This class is for representing the jail. When landed on this a variable named jailcount is triggered. When jailcount = 0 - Not Jail , jailcount != 0 - Jail.
- TaxSQ : When landed on this tax is paid.
- FreePark : Nothing is done.

This is the main structure. I have a class named Player for players. And classes such as JSONParse, Commands, SquareList, ChanceList, ChestList.

- Player: This is for our players and our banker. This class has 2 constructors one for players and the other for our banker. Has 3 functions namely; "rollTheDice(dice)", "numOfRailRoads()", "buyProperty(PropertySQ ps)".
  - rollTheDice(dice) : This function is for the main action which consists of rolling the dice. It first controls the jail situation. After that controls if "GO" is passed. Then it determines the position it arrived. After that bankruptcy is controlled. That position is sent as index to the SquareList and an object is pulled. And the landedOn() function is called on said object.
  - numOfRailRoads() : This returns the number of railroads the player owns for when any of the players is landed on a Railroad.
  - buyProperty(PropertySQ ps) : This is for buying a property. After this is called an object from the SquareList added to the player's owned list and money is exchanged between the banker and the player.
- JSONParse: This class is for parsing our JSON files. It has a constructor that takes two JSONObjects properties and list. And 2 functions parse these jsons in the constructor and objects are added to ChanceList, ChestList and SquareList.
- Commands : This class is for parsing the commands.
- SquareList: This class is a singleton. It just stores and array with type Square. And consists the whole board inside it. Any object is pulled whenever needed.
- ChanceList: This is a singleton as well which stores our cards as a Queue. When a card is pulled it is put back on the queue and returned as a string.
- ChestList: This is a singleton as well. This stores our chest cards as a Queue. When a card is pulled it is put back on the queue and returned as a string.

REFERENCES

http://www.tutorialspoint.com/java/java_polymorphism.htm

http://www.tutorialspoint.com/java/java_abstraction.htm

http://www.tutorialspoint.com/java/java_using_singleton.htm