Documentation for first Assignment

Name : Soe Thet Naing

Neptun : x0flxy

Assignment Name: Capitaly Game

Assignment No: 3

Task:

Simulate a simplified Capitaly game. There are some players with different strategies, and a

cyclical board with several fields. Players can move around the board, by moving forward with

the amount they rolled with a dice. A field can be a property, service, or lucky field.

A property can be bought for 1000, and stepping on it the next time the player can build a house

on it for 4000. If a player steps on a property field which is owned by somebody else, the player should pay to the owner 500, if there is no house on the field, or 2000, if there is a house on it. Stepping on a service field, the player should pay to the bank (the amount of money is a parameter of the field). Stepping on a lucky field, the player gets some money (the amount is defined as a parameter of the field). There are three different kinds of strategies exist. Initially, every player has 10000.

Greedy player: If he steps on an unowned property, or his own property without a house, he starts buying it, if he has enough money for it.
Careful player: he buys in a round only for at most half the amount of his money.
Tactical player: he skips each second chance when he could buy.

If a player has to pay, but he runs out of money because of this, he loses. In this case, his properties are lost, and become free to buy.

Read the parameters of the game from a text file. This file defines the number of fields, and then

defines them. We know about all fields: the type. If a field is a service or lucky field, the cost of it

is also defined. After the these parameters, the file tells the number of the players, and then enumerates the players with their names and strategies.

In order to prepare the program for testing, make it possible to the program to read the roll dices

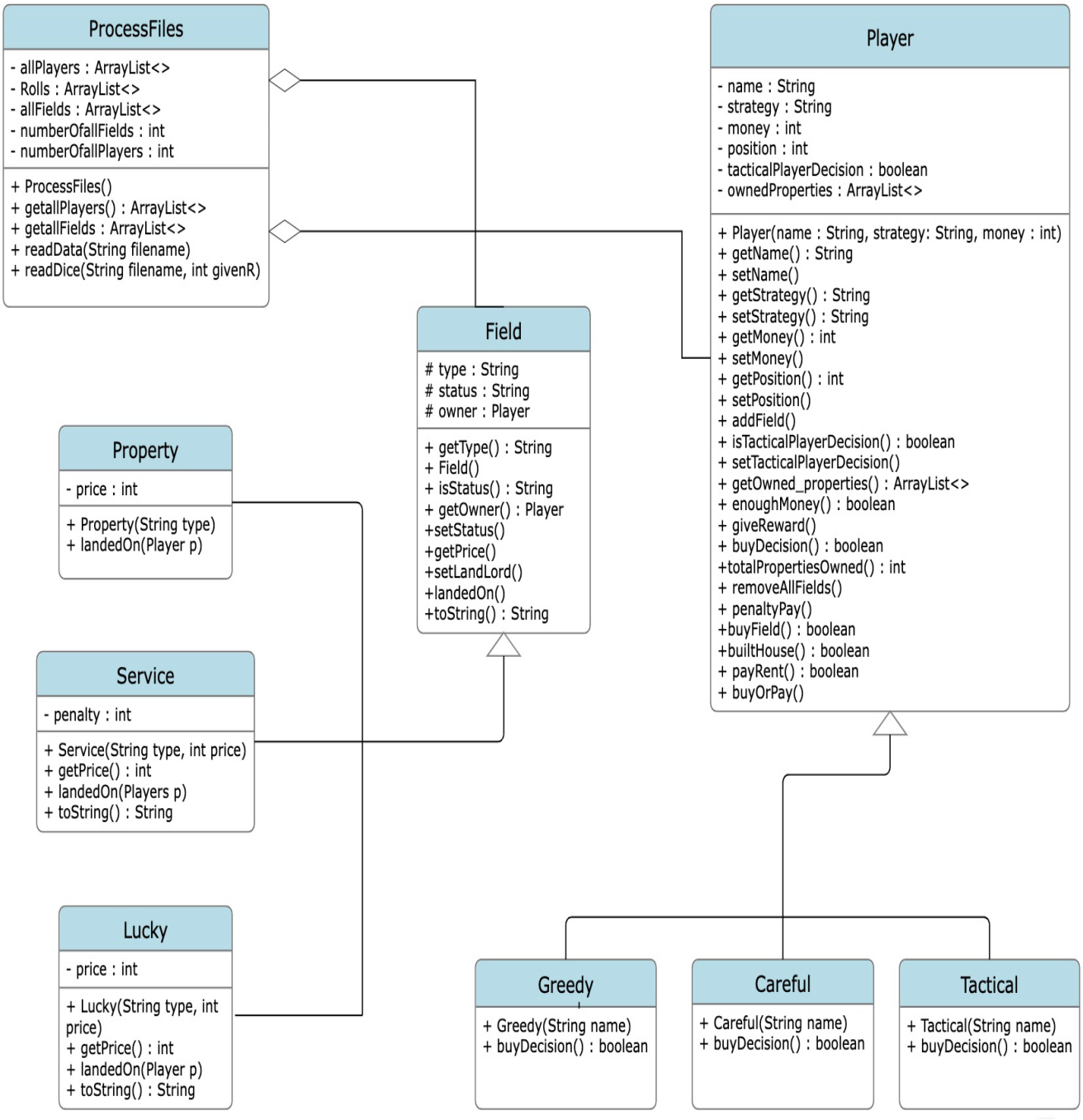
from the file.

Print out what we know about each player after a given number of rounds (balance, owned properties).

Brief description of the Solution:

We declared two separate classes for Field and Player. Field has three sub classes namely Property, Service and Lucky. They differ in the implementation of some of the methods of Field class. Similarly, class Player has three sub-Classes i.e., Greedy, Careful and Tactical, based on their strategies. Owing to their different strategies, they have different implementation of some of the functions of the Player class. Both the Field and Player classes are joined to the ProcessFiles class by an association. ProcessFiles class has been made to read the input from a text files and then simulate the game on the basis of dice rolls in the file. Players change their positions, buy and build properties and lose or earn money during the simulation process. We read number of rounds from console and then print out the number of properties and money owned by players after the given round.

Class Diagram:



Brief description of the methods:

Methods From ProcessFiles Class:

readData (): A void function which takes the filename as a parameter and reads the data of players and fields from that text file and stores them in the respective

ReadDice(): A void function which takes the filename as a parameter and reads the data of dice rolls from that text file and stores them in the respective ArrayLists. After that it simulates the game on the basis of the dice Rolls.

Methods From Player Class:

addAsset(): Takes a Field f and adds it to the assets of the Player
removeAllAssets(): Removes all Fields from the assets of a Player and sets their status to free.

EnoughMoney(): Takes a Field and tells whether the Player's money is greater or equal to the field's price.

buyOrPay(): Takes a property and adds it the assets of the Player and if its already there builds a house on it. If it belongs to some other Player, then stepping player has to pay the penalty.

penaltyPay (): Takes a Service and deducts its cost from the money of the Player. If the player doesn't have enough money, then he/she loses all the assets.

getReward(): Takes a Lucky and adds its cost to the Player's money.
BuyField() : Player can buy the field if it is free and he has enough money for it.

BuiltHouse(): Build a house if player stepped on his owned field and he has enough money for it.

PayRent(): If player steps on a field owned by someone else then he will pay the rent accordingly.

Method From Greedy Class:
buyDecision (): Overrides the buyDecision method of the Player class and returns the

EnoughMoney() : method of the super class

Method From Careful Class:
buyingDecision (): Takes a Field f and returns true if Player's money is greater or equal to

double of the f's price.

Method From Tactical Class:
buyDecision (): Takes a Field f and returns true if Player has enough money to buy the field

and its tactics attribute is true.

Method From Field Class:
landedOn(): It's a virtual method which takes a Player p and does nothing

Methods From sub-Classes of Field:

landedOn(): (Lucky)Overrides the landedOn method of its super class by taking a Player p as parameter and calling the getReward method of the Player class.

landedOn(): (Service)Overrides the landedOn method of its super class by taking a Player p as parameter and calling the Penalty Pay method of the Player class.

landedOn() : (Property) Takes a Player p and calls the buyOrPay method of p.

Testing:

Test and Dice:

Here the test file has 20 fields and 5 players. The dice file has 100 rolls of dice in it. Both files are in good format with no errors so the data is readed successfully and the program run very smooth for given rounds.

Test1 and Dice1:

Here the dice file has 30 rolls of dice in it but the test1 file is empty. So the program will throw the Empty file exception.

Test2 and Dice2:

Here the test file contains more fields than given number at the start of file which will result in input mis match. The dice 2 file is also empty so it also results in error which we will catch trough exceptions.

Test3 and Dice3:

Here the dice file contains some rolls which are negative. So this will cause an error and the test file also contains the number of players value in negative or zero.

Test4:

The file test4 contains 10 fields which are in proper format but the number of players are greater than the original players names and strategies followed by it. So, it will cause in no such element exception which we will handle in the main.