Prog. Tech 1st Assignment - Documentation

ALI AFZAL **1. assignment/4th task** 6th October 2022

BLVHI9
blvhi9@inf.elte.hu
Group 1

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 kind 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 which player won the game, and how rich he is (balance, owned properties).

Plan

To describe the fields, 4 classes are introduced: base class *Fields* to describe the general properties and 3 children for the concrete fields: *Property*, *Service*, and *LuckyField*. Regardless the type of the field, they have several common properties, like the name (*name*) and the cost (*cost*), the getters and setters of the same, and it can be examined what happens when an player variable step on it. The *Property* class has some additional properties like if it is owned (*owned*), the name of the owner (*owner*), if there is a house built on the property (*house*) and the getters and setters of the same. The general class *Fields* is going to be abstract as we do not wish to instantiate such class.

To describe the strategy of the players, 4 classes are introduced: base class *Players* to describe the general properties and 3 children for the concrete strategy: *Greedy*, *Careful*, and *Tactical.* Regardless the type of the strategy, they have several common properties, like the name (*name*) and the money (*money*), the list of owned Properties(*owns*), current position on the board(*currentPos*), if the player is still alive in the game (*alive*), the getters and setters of the same, a method to delete all owned properties (*deleteAllProperty()*), a method to check if a player can spent money on the field (*canSpent(cost)*) and a method to check if the player owns a particular property (*searchProperty(property)*).

Every concrete strategic player has a methods (*stepOnField()*) that will determine what happens if a player step on a particular field. Operations *deleteAllProperty(), canSpent(cost) and searchProperty(property)* may be implemented in the base class already, but *stepOnField()* just on the level of the concrete classes as its effect depends on the type of the field. Therefore, the general class *Players* is going to be abstract, as method *stepOnField()* is abstract and we do not wish to instantiate such class.

Description of *void stepOnField(Field a)* method :-

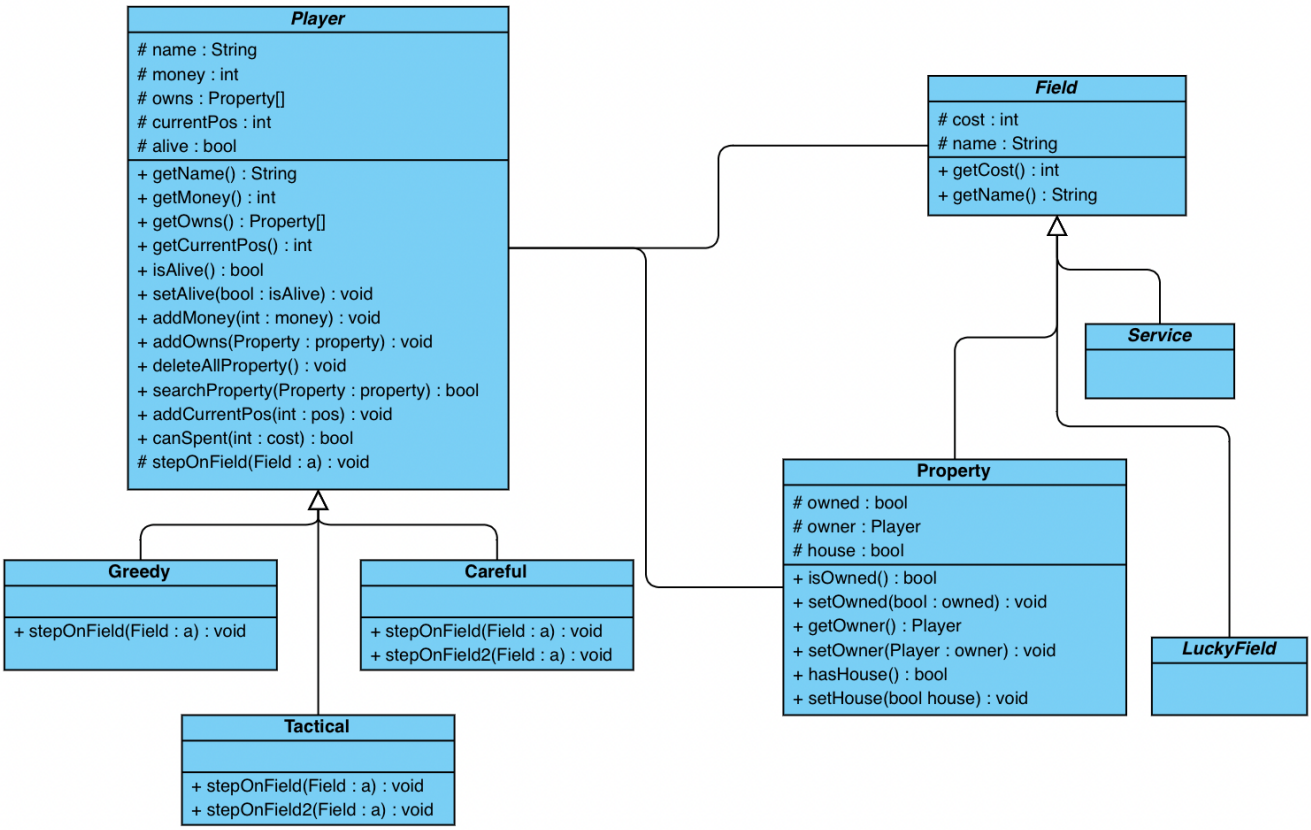
Task:- Demonstrate what will happen if the player steps on a particular type of *Field*.

Input:- variable ***a*** of type ***Field***.

Activity:- First we will check the type of *Field* (*Service*, *LuckyField* or *Property*), then we do the following:

- Property – First we will check if the *Player* has already owned the *Property*, if yes then we will check if he has enough money to spent in order to built a house on it and if yes then we simply deduct the balance and built a house on the *Property (set house to true).* And if the property is owned by someone else then we deduct the balance (if he has enough balance or he loses the game and his all properties are released) based on if there is a house built on the property. Lastly if the property is not owned, the *Player* can buy the property depending upon his strategy (if enough balance).
- Service – If the *Player* has enough balance to pay the bank, we deduct the amount from his balance else he loses the game and his all properties are released and free to buy.
- LuckyField – We add the particular amount to the balance of the *Player*.

UML Class Diagram



Testing

1) test.txt & test1.txt-

Sample Input – 20 *Fields* and 5 *Players*
Sample Output = 1 Player wins the game after many rounds

2) test2.txt-

Sample Input – 1 *Service* field and 2 *Players*
Sample Output = The Player with the second chance wins

3) test3.txt-

Sample Input – 1 *Property* field and 2 *Players*
Sample Output = The Player with the second chance wins and his balance should be exactly 15000 (as the first time he steps on a *Property*, he buys it for 1000 and then later he builds a house on it for 4000 and the other player should pay him every time he steps on the *Property*)

4) test4.txt-

Sample Input – 1 *LuckyField* field and 2 *Players*
Sample Output = The game will never end