Initial Draft

Assignment 1 Task no. 4

By Ryan Supawarapong

for Programming Theory 2023 Autumn Semester

Monopoly Game (CLI)

Common requirements:

Use a Collection to store the objects of classes derived from the same super class.

Use foreach to process the elements of a Collection.

Validate the data what you get from the user; throw Exception for invalid data, and handle the thrown Exceptions.

The documentation should contain:

* the description of the exercise,
* the class diagram,
* the short description of each methods,
* and the testing (white box / black box)

Task:

Simulate a simplified Capitaly(Monopoly) game. There are some players with different strategies, and acyclical board with several fields. Players can move around the board, by moving forward withthe 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).

**Simplified version of the description**

Simulate a simplified Capitaly (Monopoly) game. The game involves players with different strategies navigating a cyclical board with various types of fields. Players can move around the board based on the roll of a dice. The fields can be of three types: Property, Service, or Lucky. The game has specific rules for buying properties, building houses, and paying rent or service fees. Players can employ different strategies: Greedy, Careful, and Tactical. The game ends when a player runs out of money, and the remaining player is declared the winner.

This is the UML diagram of the program.

A diagram of a computer program

Description automatically generated with medium confidence

**Method Description**

MonopolyJava Class

* main(String[] args): Initializes the game board and players, then starts the game.
* createBoard(Scanner scanner): Initializes the game board.
* createPlayers(Scanner scanner): Creates a list of players.
* playGame(List<Field> board, List<Player> players): Main game loop.
* leaderBoard(List<Player> players): Displays the current status of players.

Field Class

* getValue(): Gets the value associated with this field.
* setValue(int value): Sets the value associated with this field.

Property Class

* getValue(): Gets the rental value of this property.
* totalCost(): Gets the total cost of this property.
* buyPrice(): Gets the purchase price of this property.
* isForSale(): Checks if this property is available for purchase.
* setOwner(Player player): Sets the owner of this property.
* getOwner(): Gets the owner of this property.
* removeOwner(): Removes the current owner of this property.
* addHouse(Player player): Adds a house to this property.

PlayerStrategy Interface

* execute(Player player, Field field): Executes the player strategy based on the field they are on.

GreedyStrategy Class

* execute(Player player, Field field): Executes the greedy strategy for a given player on a given field.

TacticalStrategy Class

* execute(Player player, Field field): Executes the tactical strategy for a given player on a given field.

CarefulStrategy Class

* execute(Player player, Field field): Executes the careful strategy for a given player on a given field.

**Black-Box Testing:**

* testCreateBoard: Tests if the board is created correctly based on input.
* testCreatePlayers: Tests if players are created correctly based on input.
* testPlayGame: Tests the main game loop to see if it behaves as expected.

**White-Box Testing:**

* testCannotBuyOwnedProperty: Tests a specific exception case where a player tries to buy an already owned property.
* testPlayerLosesAfterNoMoney: Tests if a player is removed from the game when they run out of money.