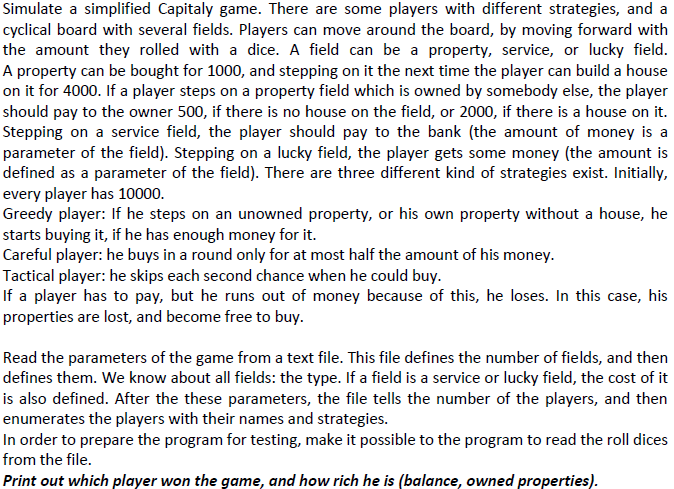
**Documentation for first Assignment**

**Name:** Rizwan Hussain **Assignment Name:** Capitaly Game

**Neptun:** ULLBQG **Assignment No:** 4

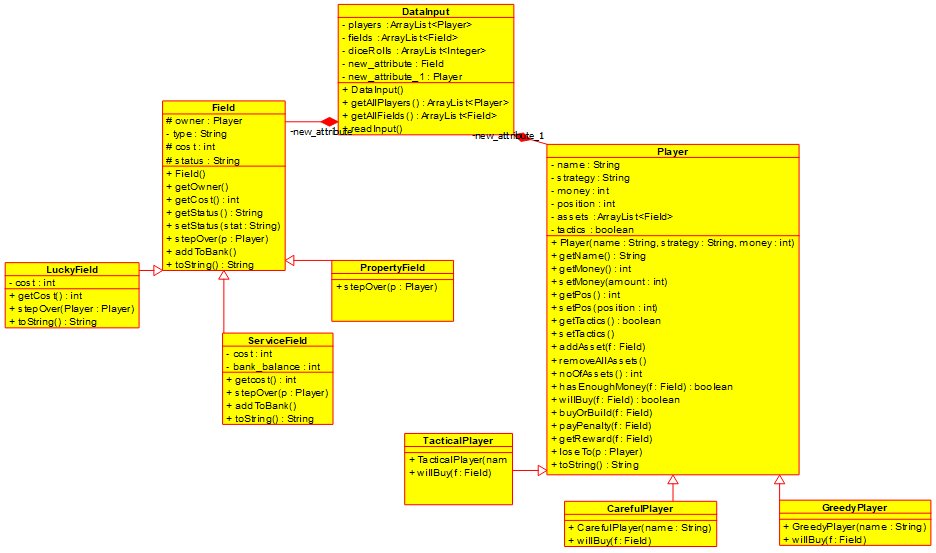
**Task:**

****

**Brief description of the Solution:**

We declared two separate classes for Field and Player. Field has three sub classes namely PropertyField, ServiceField and LuckyField. They differ in the implementation of some of the methods of Field class. Similarly, class Player has three sub Classes i.e. GreedyPlayer, CarefulPlayer and TacticalPlayer, 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 DataInput class by an association. DataInput class has been made to read the input from a text file 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. At the end the player with most number of properties is declared as the winner and if two players have same number of properties, then the one having more balance is declared as the winner.

**Class Diagram:**

****

**Brief description of the methods:**

**DataInput Class:**

**Constructors:**

DataInput(): The constructor which initializes the players, fields and diceRolls as empty ArrayLists.

**Getters:**

getAllPlayers(): Getter function which returns the players, The ArrayList of Player.

getAllFields(): Getter function which returns the fields, the ArrayList of Field.

**Other Methods:**

readInput(): A void function which takes the filename as a parameter and reads the data of players, fields and diceRolls from that text file and stores them in the respective ArrayLists. After that it simulates the game on the basis of the diceRolls.

**Player Class:**

**Constructors:**

Player(): A parameter-less constructor for the Player class.

Player(): Another constructor which takes player’s name, strategy and money as parameter and initializes the respective fields.

**Getters:**

getName(): Returns the name

getMoney(): Returns the money

getTactics(): A virtual function which is always true

getPos(): Returns the position of the player

noOfAssets(): Returns the number of Properties

**Setters:**

setMoney(): Takes an integer amount and adds/subtracts to/from the money

setTactics(): Reverse the value of the tactics, Boolean value

setPos(): Takes an int position and sets the player’s position accordingly

**Other Methods:**

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.

hasEnoughMoney(): Takes a Field f and tells whether the Player’s money is greater or equal to the f’s cost.

willBuy(): A virtual function which always returns true

buyOrBuild(): Takes a PropertyField 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.

payPenalty(): Takes a ServiceField 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 LuckyField and adds its cost to the Player’s money.

loseTo(): Takes another Player winner and gives all the money of the Player to the winner and calls the removeAllAssets() method.

toString(): Prints out a Player

**GreedyPlayer Class:**

**Constructor:**

GreedyPlayer(): Takes a String name and calls the Player’s constructor with name, String “GP” as strategy and an int 10000 as money.

**Other Methods:**

willBuy(): Overrides the willBuy method of the Player class and returns the hasEnoughMoney method of the super class

**CarefulPlayer Class:**

**Constructor:**

CarefulPlayer(): Takes a String name and calls the Player’s constructor with name, String “CP” as strategy and an int 10000 as money.

**Other Methods:**

willBuy(): Takes a Field f and returns true if Player’s money is greater or equal to double of the f’s cost.

**TacticalPlayer Class:**

**Constructor:**

TacticalPlayer(): Takes a String name and calls the Player’s constructor with name, String “TP” as strategy and an int 10000 as money. And initializes the tactics as false.

**Other Methods:**

willBuy(): Takes a Field f and returns true if Player has enough money to buy the field and its tactics attribute is true.

**Field Class:**

**Constructor:**

Field (): Takes a string type and initializes its owner field to null and its type attribute to type and its status to “free”.

**Getters:**

getOwner(): returns the owner of the Field

getCost(): returns the cost of the Field.

**Setters:**

setOwner(): Takes a Player own and sets it as the owner

setStatus(): Takes a String stat and makes status equal to stat.

**Other Methods:**

stepover(): It’s a virtual method which takes a Player p and does nothing

addToBank(): It’s a virtual method which takes no parameters and does nothing.

toString(): Prints out the attributes of the Field.

**LuckyField:**

**Constructors:**

LuckyField(): Takes a String type and int cost and calls the super’s constructor with type and initializes the cost attribute wit cost.

**Getters:**

getCost(): Overrides the getCost method of the super class and returns the cost.

**Other Methods:**

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

toString()**:** Overrides the toString() method of the super class by calling it and adding the cost attribute as well.

**ServiceField:**

**Constructors:**

ServiceField(): Takes a String type and int cost and calls the super’s constructor with type and initializes the cost attribute wit cost.

**Getters:**

getCost(): Overrides the getCost method of the super class and returns the cost.

**Other Methods:**

addToBank(): Overrides the addToBank method of the super class and adds the cost value to the bank\_balance attribute on call.

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

toString()**:** Overrides the toString() method of the super class by calling it and adding the cost attribute as well.

**PropertyField:**

**Constructor:**

PropertyField(): Takes a String type and calls the super constructor with type.

**Other Methods:**

stepOver(): Takes a Player p and calls the payOrBuild method of p.

**Testing:**

**Inp1(PropertyField):** There is one PropertyField and two Players. In first round, One of the players buys a property and the second steps onto it. In second round, first player builds a house and second again steps onto it.

**Inp2(LuckyField)**: Two players step over LuckyField and get rewards everytime they step onto it.

**Inp3(ServiceField)**: Two players step over ServiceField and pay penalty everytime they step onto it.

**Inp4(TacticalPlayer)**: There is one TacticalPlayer who has money but doesn’t buy the Poperty in first round and buys it second time.

**Inp5(CarefulPlayer):** There is one CarefulPlayer who doesn’t build a house because he doesn’t have the double amount of the required money.

**Inp6(GreedyPlayer)**: There is one GreedyPlayer who buys or builds a Property whenever he steps onto it and has enough money.

**Inp7:** There are all types of Players and fields and there is a winner at the end of 4th round.