SUBMISSION :- ARBAB ALI

ASSIGNMENT_01 - QNO3

ASSIGNMENT Description:

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

INSTRUCTIONS TO USE :-

To use this game, you need to create two INPUT text files and place it in the project folder. Data in the text files should be like respectively :
File 1: **data.txt**

First line of contains integer input **n** and next **n** lines of input should be in this format

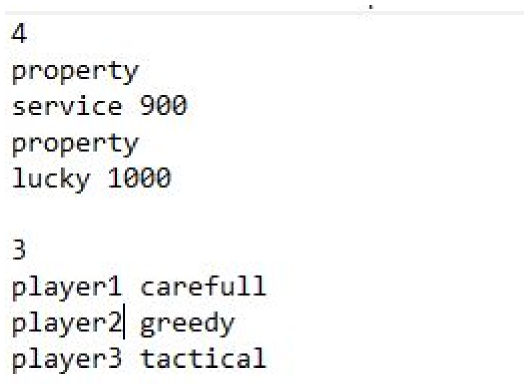
FieldName PriceParameter (in case of lucky /service field only)
FieldName (in case of Property field)
...
Field Types can only be chosen from the following three

- Service
- Lucky
- Property

After **n** lines of fields
The next line after **n** lines of fields is Integer input **m ,** and next **m** lines of input shall be data of player in following format
m
playerName   playerStrategy
.
.
..m
Type of player can only be one the following three,

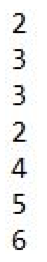
- Greedy
- Carefull
- Tactical

EXAMPLE OF INPUT FILE 1 :



```
4
property
service 900
property
lucky 1000

3
player1 carefull
player2 greedy
player3 tactical
```

File 2: **dice.txt**
This file contains the Integer input of rolls dice
n can be (1,2,3,4,5,6)
Example of file2:



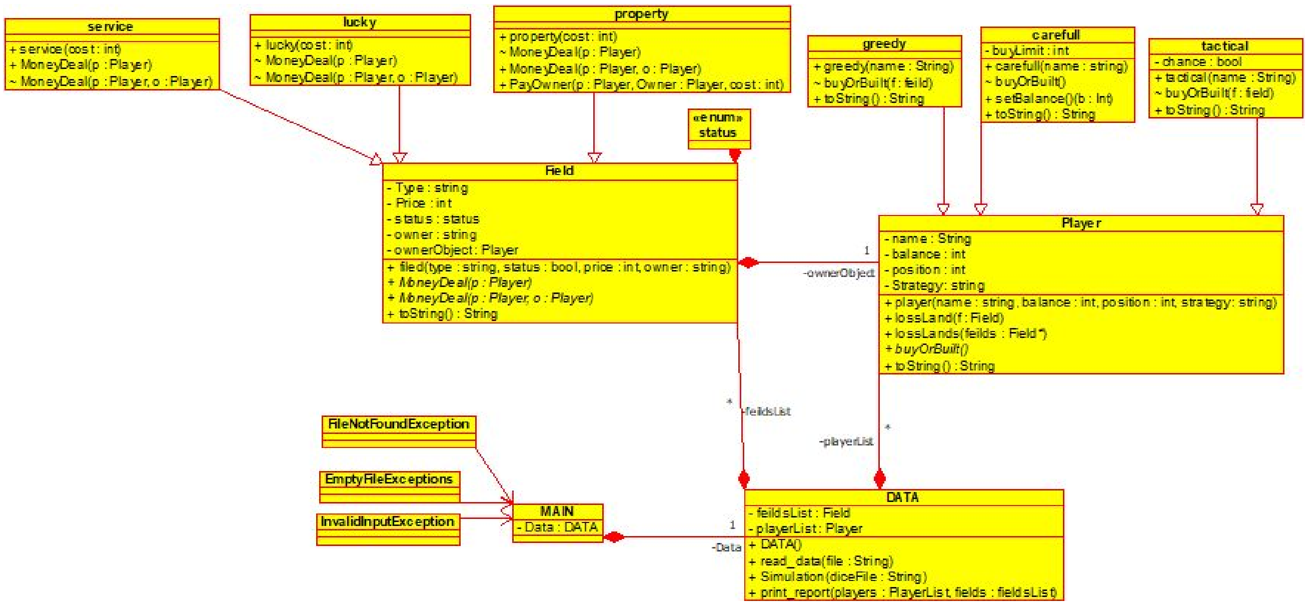
```
2
3
3
2
4
5
6
```

IT IS ASSUMED THAT DATA GIVEN IN INPUT FILES MUST BE VALID TYPES

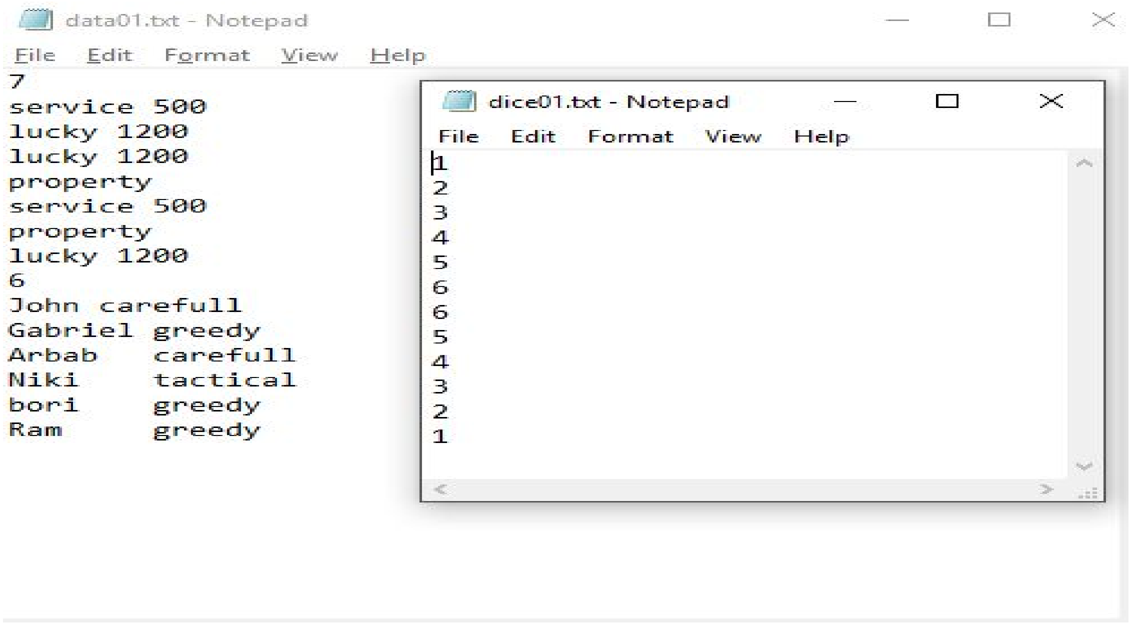
CLASS DIAGRAM :

There are two main abstract class , **Field** class that is extended by service ,lucky and property class and field class has two abstract methods ***MoneyDeal(player p)*** and ***MoneyDeal (player p , Player o)*** which are implemented in child class according to their conditions (*special-note :* the compile-time polymorphism is used for MoneyDeal function because of different behaviour of child class *property* than lucky and service), Property class have one auxiliary private method ***PayOwner(player P ,player Owner,int cost)*** .
Player class that is extended by greedy,careful and tactical class, player class has one abstract method ***buyorBuilt()*** ,which is implemented in child class according to their conditions
Class DATA is used to to read and save players data in Collections java.util.ArrayList privately with method ***read_data(file:string)***
And simulates the behaviour of fields and Players instances by reading the ***dice.txt*** in method ***simulation(diceFile:string)***

EXCEPTIONS are handled in Main Class and Instance of DATA is initialized in Main class as well



SAMPLE OUTPUT AND INPUT:
1)



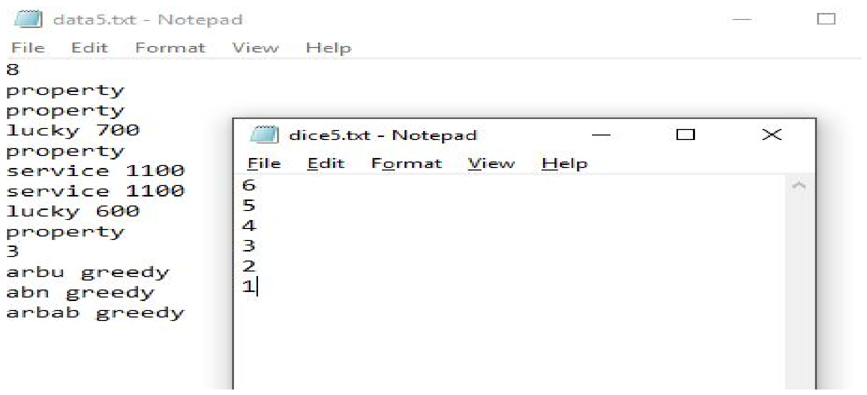
```
7
service 500
lucky 1200
lucky 1200
property
service 500
property
lucky 1200
6
John carefull
Gabriel greedy
Arbab    carefull
Niki     tactical
bori     greedy
Ram      greedy
```

```
1
2
3
4
5
6
6
5
4
3
2
1
```

output

```
roundCount: 12

BOARD OF GAME
Field: service Status: owned Owner-> bank Price-> 500
Field: Lucky Status: owned Owner-> ForEveryone Price-> 1200
Field: Lucky Status: owned Owner-> ForEveryone Price-> 1200
Field: property Status: owned Owner-> Niki Price-> 4000
Field: service Status: owned Owner-> bank Price-> 500
Field: property Status: owned Owner-> Ram Price-> 4000
Field: Lucky Status: owned Owner-> ForEveryone Price-> 1200

PLAYER IN THE GAME:-
Name: John Strategy:carefull balance :10700 BuyLimit: 5350 Position: 7 Owned Properties :0
Name: Gabriel Strategy:greedy balance :12400 Position: 7 Owned Properties :0
Name: Arbab Strategy:carefull balance :12400 BuyLimit: 6200 Position: 7 Owned Properties :0
Name: Niki Strategy:tactical balance :10200 isItSecondChance: false Position: 7 Owned Properties :1
Name: bori Strategy:greedy balance :10700 Position: 7 Owned Properties :0
Name: Ram Strategy:greedy balance :10200 Position: 7 Owned Properties :1
```

2)



```
8
property
property
lucky 700
property
service 1100
service 1100
lucky 600
property
3
arbu greedy
abn greedy
arbab greedy
```

```
6
5
4
3
2
1
```

output

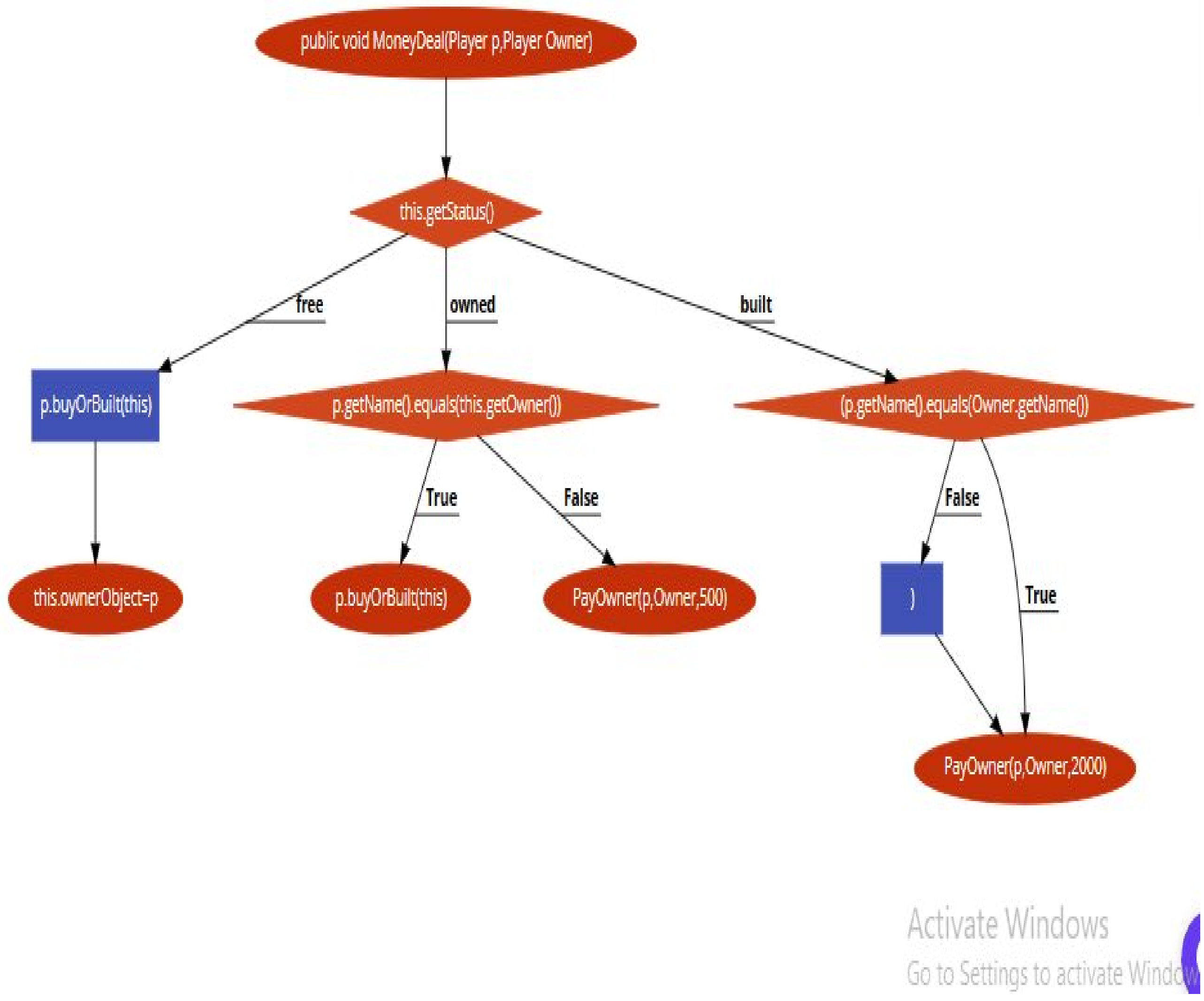
```
roundCount: 6

BOARD OF GAME
Field: property Status: owned Owner-> arbu Price-> 4000
Field: property Status: free Owner-> none Price-> 1000
Field: Lucky Status: owned Owner-> ForEveryone Price-> 700
Field: property Status: owned Owner-> arbab Price-> 4000
Field: service Status: owned Owner-> bank Price-> 1100
Field: service Status: owned Owner-> bank Price-> 1100
Field: Lucky Status: owned Owner-> ForEveryone Price-> 600
Field: property Status: free Owner-> none Price-> 1000

PLAYER IN THE GAME:-
Name: arbu Strategy:greedy balance :7900 Position: 1 Owned Properties :1
Name: abn Strategy:greedy balance :9500 Position: 7 Owned Properties :0
Name: arbab Strategy:greedy balance :7900 Position: 5 Owned Properties :1
________________________________________________
```

IMPLEMENTATION OF IMPORTANT METHODS IN FLOWCHARTS:

PROPERTY CLASS :

MoneyDeal(Player p ,Player owner)

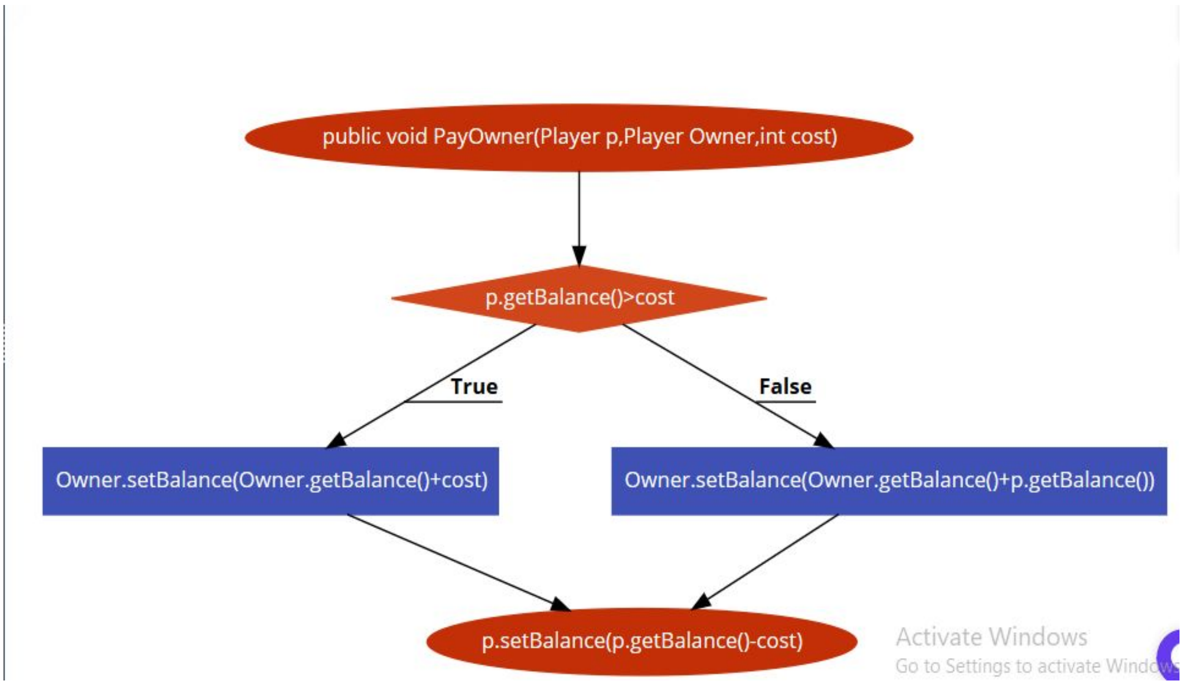
[1]*Checks the status of field if free then calls the buyOrBuilt method otherwise looks if he is owner if not then pays the price to owner otherwise builds the house or nothing*



1

PayOwner(Player p ,Player o ,int cost)

In case of status not free and owner is not the player p then this method is called which checks the balance of player and pays the cost according to balance in account and subtracts from players accounts

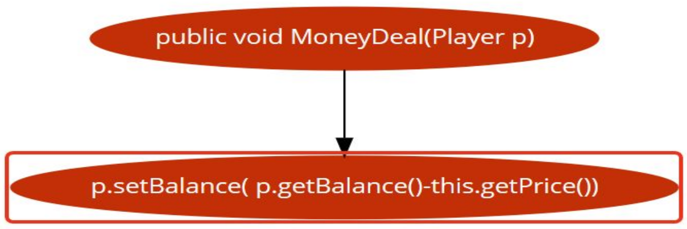


public void PayOwner(Player p,Player Owner,int cost)
p.getBalance()>cost
True
False
Owner.setBalance(Owner.getBalance()+cost)
Owner.setBalance(Owner.getBalance()+p.getBalance())
p.setBalance(p.getBalance()-cost)
Activate Windows
Go to Settings to activate Windows

SERVICE CLASS

MoneyDeal(Player p):

In case of player landing on service land it subtracts the money(parameter of service) from players balance

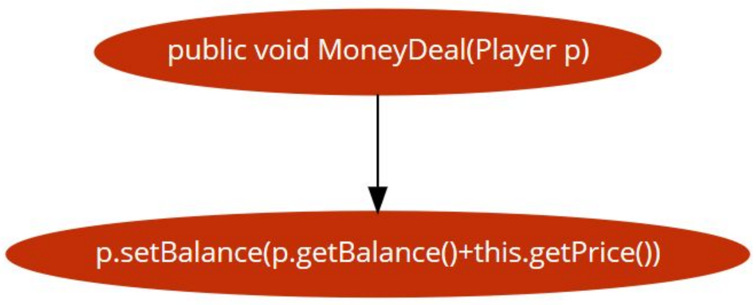


public void MoneyDeal(Player p)
p.setBalance(p.getBalance()-this.getPrice())

LUCKY CLASS:

<u>MoneyDeal(Player p):</u>

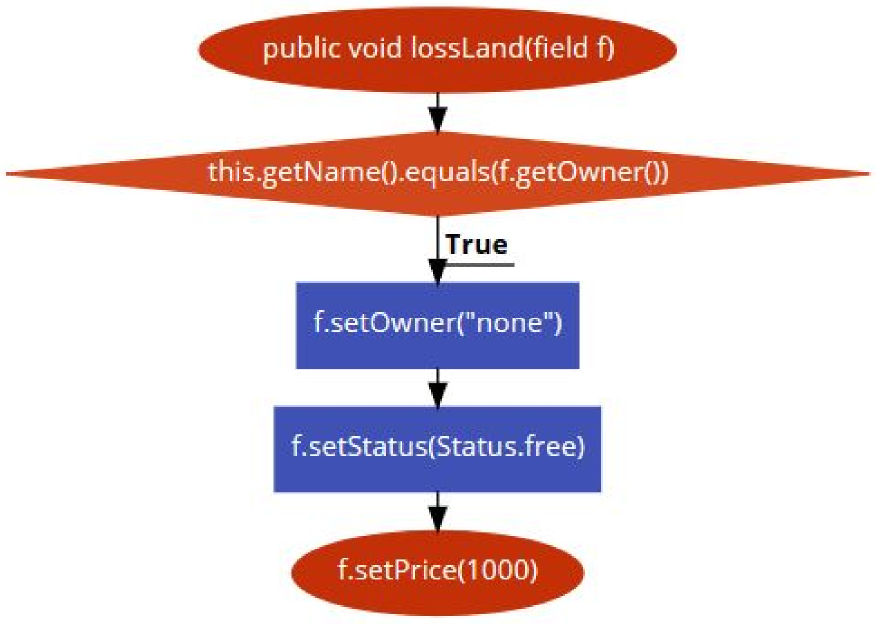
In case of player landing on lucky land it adds the money(parameter of lucky) in players balance


public void MoneyDeal(Player p)
p.setBalance(p.getBalance()+this.getPrice())

PLAYER CLASS :

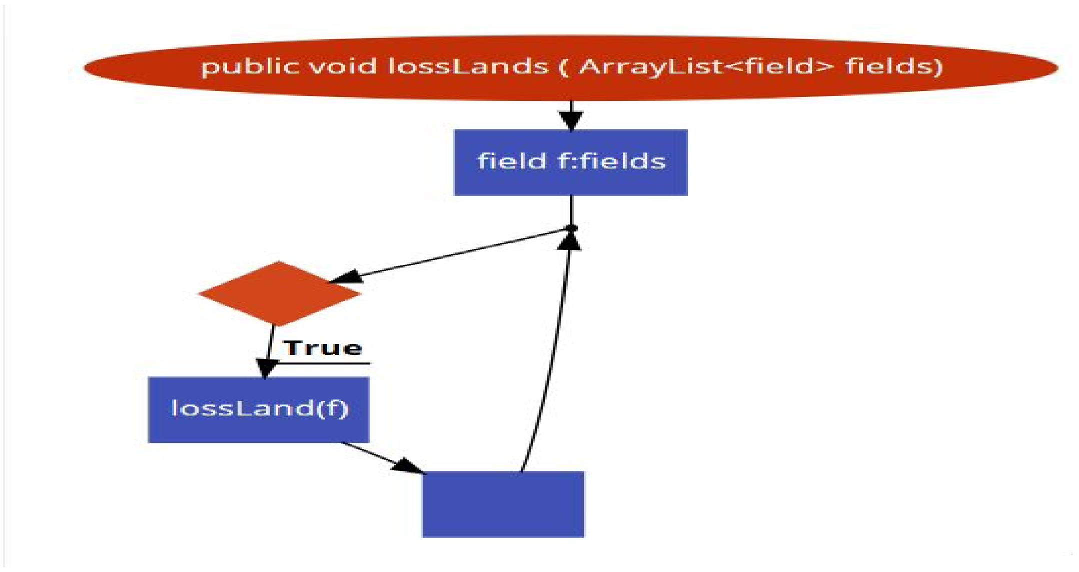
LossLand(field f):

This method takes field parameter and checks if player is owner of land if yes then resets the field as free


public void lossLand(field f)
this.getName().equals(f.getOwner())
True
f.setOwner("none")
f.setStatus(Status.free)
f.setPrice(1000)

LossLands(ArrayList<field> fields):

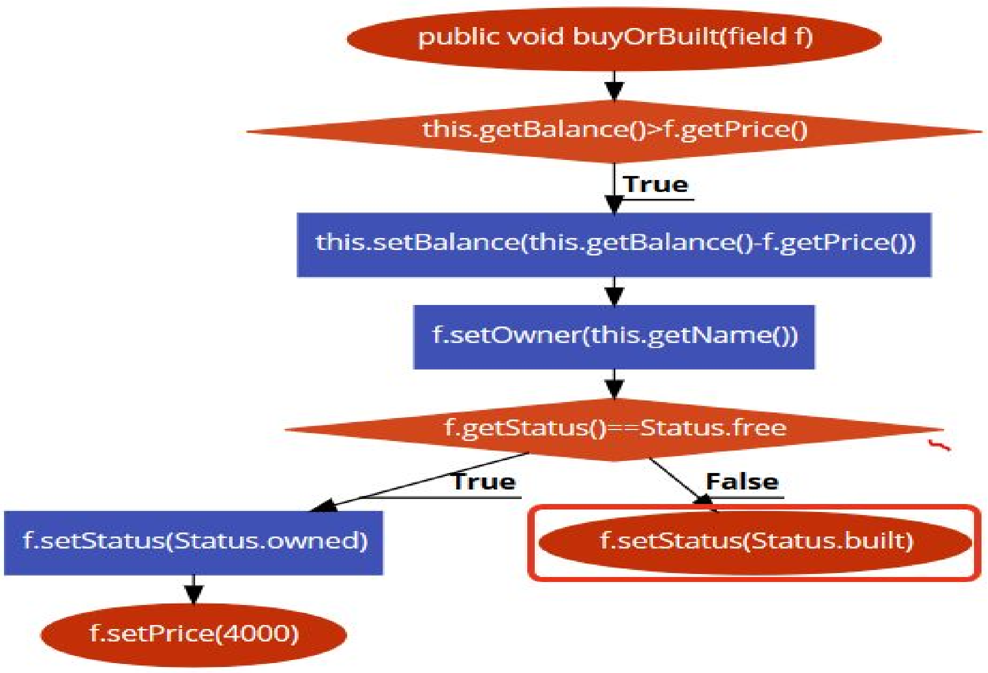
This method takes arrayList of fields and checks on each fields if player is owner and if yes then calls the LossLand function to reset the fields attributes



GREEDY CLASS:

buyOrBuilt(field f) :

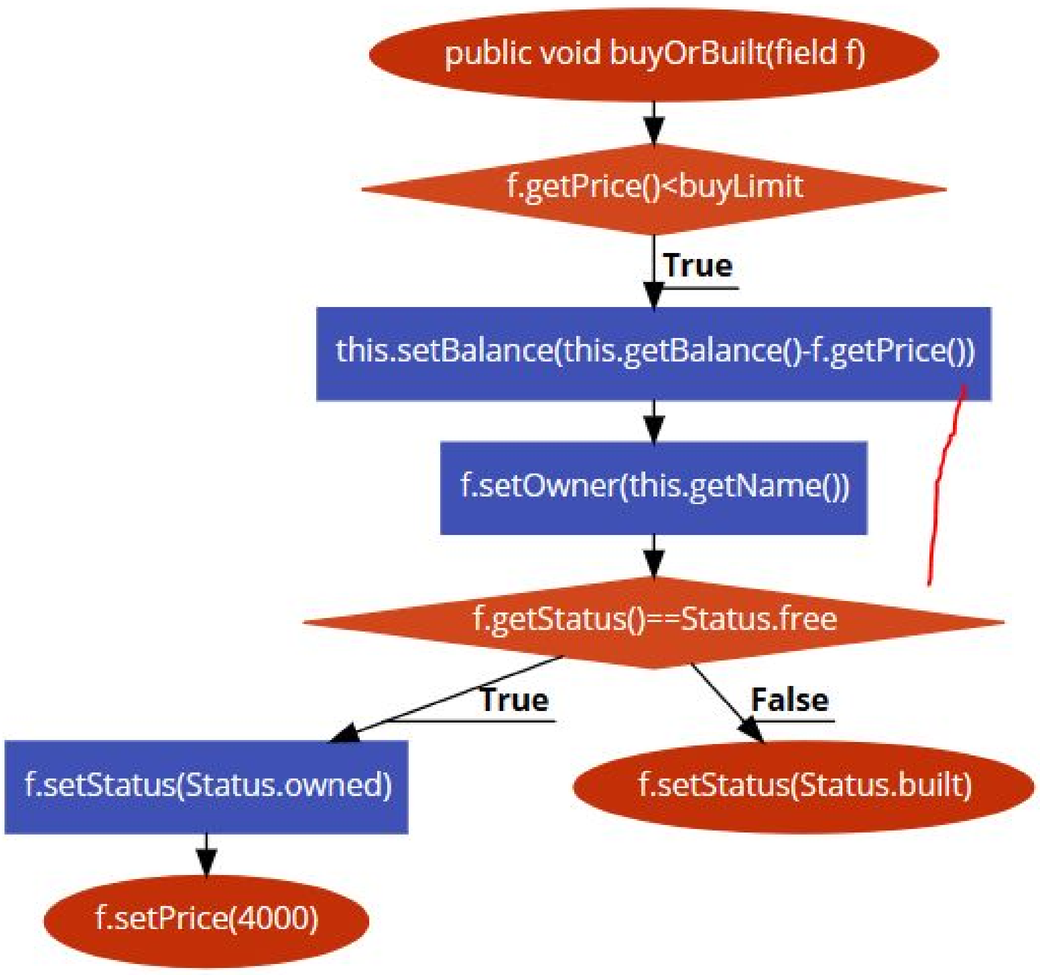
This method is called when the status of land is free or owned by a player , it checks the balance amount if more than price of field then subtracts amount from account also if land is free then it becomes owned now and price becomes 4000 or if its not free but owned then its status becomes built.


public void buyOrBuilt(field f)
this.getBalance()>f.getPrice()
True
this.setBalance(this.getBalance()-f.getPrice())
f.setOwner(this.getName())
f.getStatus()==Status.free
True
False
f.setStatus(Status.owned)
f.setStatus(Status.built)
f.setPrice(4000)

CAREFUL CLASS

buyOrBuilt(field f) :

This method is called when the status of land is free or owned by a player ,in case of careful player it checks the buyLimit amount if more than price of field then subtracts amount from balance also if land is free then it becomes owned now and price becomes 4000 or if its not free but owned then its status becomes built.

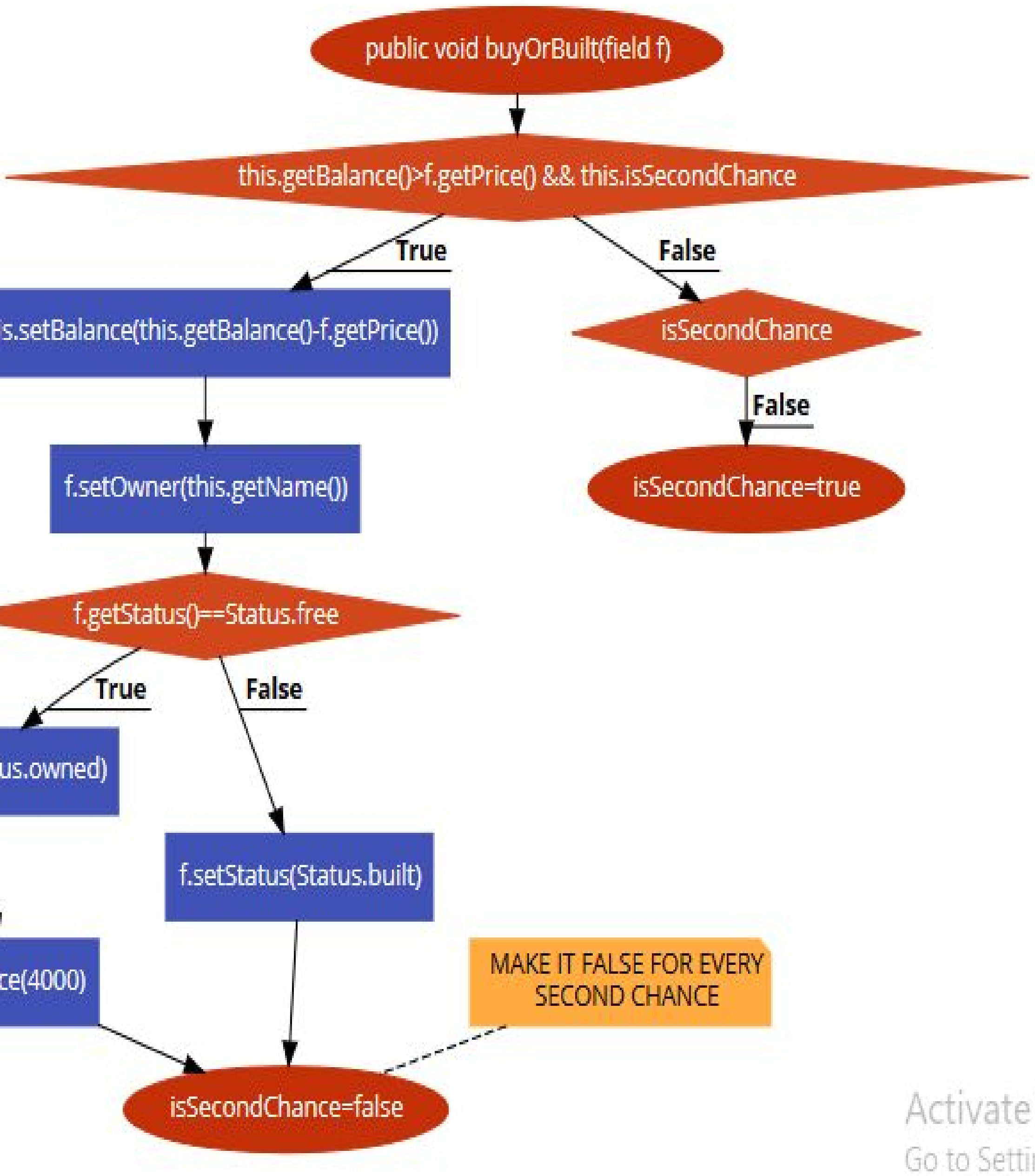


public void buyOrBuilt(field f)
f.getPrice()<buyLimit
True
this.setBalance(this.getBalance()-f.getPrice())
f.setOwner(this.getName())
f.getStatus()==Status.free
True
False
f.setStatus(Status.owned)
f.setStatus(Status.built)
f.setPrice(4000)

TACTICAL CLASS :-

buyOrBuilt(field f) :

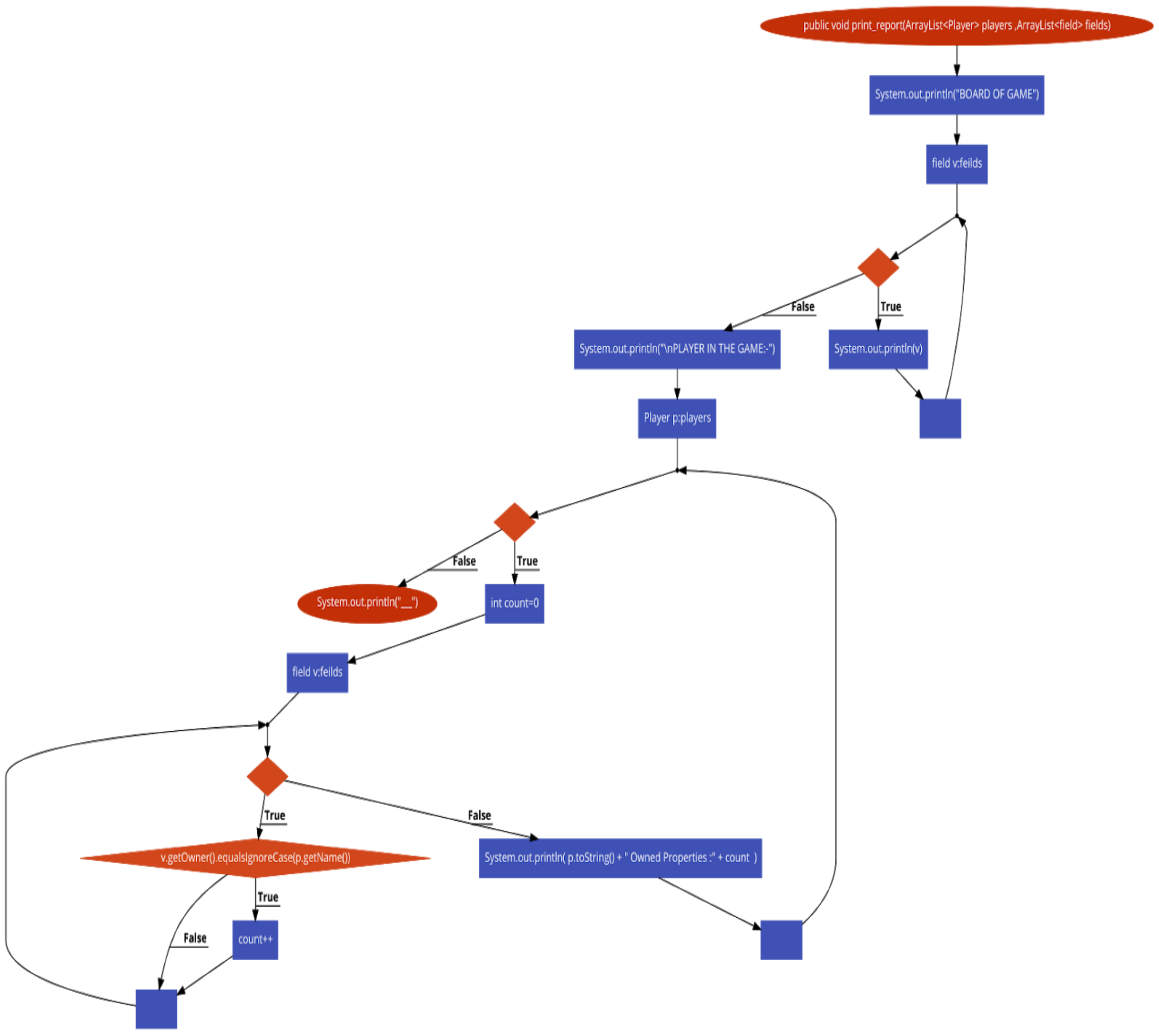
This method is called when the status of land is free or owned by a player ,in case of tactical player it checks the balance amount if more than price of field and also if its second chance of player to buy then subtracts amount from balance also if land is free then it becomes owned now and price becomes 4000 or if its not free but owned then its status becomes built.if it wasn't second chance then it makes it true for next chance


public void buyOrBuilt(field f)
this.getBalance()>f.getPrice() && this.isSecondChance
True
False
this.setBalance(this.getBalance()-f.getPrice())
isSecondChance
False
f.setOwner(this.getName())
isSecondChance=true
f.getStatus()==Status.free
True
False
f.setStatus(Status.owned)
f.setStatus(Status.built)
f.setPrice(4000)
MAKE IT FALSE FOR EVERY SECOND CHANCE
isSecondChance=false

DATA CLASS:

print_report(ArrayList<field> fields,ArrayList<player> players)

This method prints the attributes of fields and player and also counts the number of properties owned by each player


public void print_report(ArrayList<Player> players ,ArrayList<field> fields)
System.out.println("BOARD OF GAME")
field v:feilds
False
True
System.out.println("\nPLAYER IN THE GAME:-")
System.out.println(v)
Player p:players
False
True
System.out.println("__")
int count=0
field v:feilds
True
False
v.getOwner().equalsIgnoreCase(p.getName())
System.out.println(p.toString() + " Owned Properties :" + count)
True
False
count++

Next two methods read_data and simulations are not uploaded as flow-chart due to their size but coded as follow

read_data :-

This method reads data and saves it in arraylists

```
public void read_data(String filename) throws FileNotFoundException, InvalidInputException
       {
       Scanner sc = new Scanner(new BufferedReader(new FileReader(filename)));
       while(sc.hasNext()){

       int numProperty=sc.nextInt();
       for(int i=0;i<numProperty;i++)
       {
               field  newfield=null;

       String a=sc.next() ;
       switch(a){
       case "property":
               newfield= new property(1000);
               break;
       case "lucky":
               newfield= new lucky(sc.nextInt());
               break;
       case "service":
               newfield= new service(sc.nextInt());
               break;
       default:
               throw new InvalidInputException();
       }
       feilds.add(newfield);

       }

       int numPlayer=sc.nextInt();
       for(int i=0;i<numPlayer;i++)
       {
       Player newPlayer ;
       while(sc.hasNext())
       {
               String name=sc.next();
               switch(sc.next())
               {
               case "carefull":
               newPlayer=new carefull(name);
               break;
               case "greedy":
               newPlayer=new greedy(name);
               break;
               case "tactical":
               newPlayer=new tactical(name);
               break;
               default:
               throw new InvalidInputException();
               }
       players.add(newPlayer);
       ///SAVING THE OWNERS ADDRESSES
       }       }}}
```

simulation(dicefile:string):

This method reads data from diceFile and simulates fields and players behaviours on input in diceFile .

```
/**
 * Reads the Data for dice from file  and simulates the game behavior
        * @param diceFile
 * @throws FileNotFoundException
 * @throws InvalidInputException
 */
public void simulation(String diceFile) throws FileNotFoundException, InvalidInputException
{
 Scanner dice = new Scanner(new BufferedReader(new FileReader(diceFile)));

 print_report(players,feilds);
        System.out.println("\nSIMULATION....\n");

        //ArrayList<Player>lossers=new ArrayList<>();
        int roundCount=0;
        Iterator<Player> p_Itr=players.iterator();
        Player p;
        while( dice.hasNextInt()){
        //  int random_int = (int)(Math.random() * (6 - 1 + 1) + 1);
        if(!p_Itr.hasNext())
        {
        p_Itr=players.iterator();
        }

        p=p_Itr.next();

        int i =dice.nextInt();
        System.out.println("\ndice:"+i);

        p.setPosition(p.getPosition()+i);
        if(p.getPosition()>feilds.size())
        {
        p.setPosition(1);
        }
        field v=feilds.get(p.getPosition()-1);

        System.out.println("Playing NOW: "+p.getName());
        System.out.println("LANDED ON Field OF: "+v.getType() +" "+v.getOwner());
```

```
                if(v.getType().equals("property"))
                        v.MoneyDeal(p,v.ownerObject);
                else
                v.MoneyDeal(p);

        if(p.getBalance()<0)
        {
                p.lossLands(feilds);
                // lossers.add(p);
                p_Itr.remove();
        }



        roundCount+=1;
        System.out.println("\nroundCount: " +roundCount +"\n");
        print_report(players,feilds);
        }

        }
```

CASES:

Test cases are designed with *Junit* test tool,there are total 13 tests cases and 7 tests files ,I have checked the implementation of Players methods and its child class methods as well as field class childs methods

For each class i have tested the methods created according to the project description And also the constructors of each child class .

TestCases can be found in assignment_01/src/test

EXCEPTION HANDLING:

- FileNotFoundException
- InvalidInputException
- EmptyFileException