**Documentation**

**Assignment 1 Subject 4**



**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 kinds of strategies exist. Initially, every player has 10000.

1. 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.
2. Careful player: he buys in a round only for at most half the amount of his money.
3. 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 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).**

**Methods descriptions**

**The methods in Program**

1. public Program(String filename): create a program with parameter filename (String);.
2. public void run(): run the program to print the winner data to console
3. public String progToString(): get the winner data to string

**The methods in Capitaly**

1. Capitaly(ArrayList<Field> circuit, ArrayList<Player> players, ArrayList<Integer> rolls): create a capitaly game by parameter circuit, players, and rolls
2. public ArrayList<Field> getCircuit(): return \_circuit.
3. public ArrayList<Player> getPlayers(): return \_players.
4. public ArrayList<Integer> getRolls(): return \_rolls.
5. public Boolean isLastWinner(Player p): check wheter a player p is the last winner in the fields, in case that every player in the circuit before p is already lost in the same round, p will automatically win.
6. public void cleanUp(): clear the \_circuit, \_players, and \_rolls array list in case we will end the game.
7. public void doEvent(): looping rounds for the players to roll dice and move to the fields. Until only one player is left. The person will be the winner, his/her data will be printed out: type, name, balance, owned properties.
8. public String getWinnerString(): as same as doEvent(), but the output is string, not just to command line.

**The methods in Player**

1. public String getName(): get name of the player.
2. public Player(String name): create player.
3. public ArrayList<Field> getFields(): get fields the player owned.
4. public int getPos(): get position on the board of the player.
5. public int getBalance(): get balance the player owns.
6. public int getInGame(): get whether the player is lost or not.
7. public abstract String getType(): get the type of the player (implement later).
8. public abstract void buy(Property f) : player buys property (implement later).
9. public void buy(Field f): use as a transitional method.
10. public boolean setMoney(int amount): set the amount of the balance.
11. public void addField(Field field): add field.
12. public void removeField(): remove fields.
13. public void move(int amount): move the player.
14. public void pay(Player player, int amount): play another player as the given amount.
15. public void print(): print the data of the player.
16. public String printString(): print the data of the player as a string.
17. public void doEvent(Field f): given a field a player will reacted differently
    1. “Property”: the player buys it.
    2. “Service”: the player pays for it, if balance <= 0, the player is out of game.
    3. “Lucky”: the player gets money.

**The methods in CarefulPlayer**

1. public CarefulPlayer(String name): create CarefulPlayer
2. public String getType(): return “Careful Player”
3. public void setMoneyForTheRound(): set money for the round as half of the balance
4. public void buy(Property f): buy property if and only if he has 2 times the money

**The methods in GreedyPlayer**

1. public GreedyPlayer(String name): create GreedyPlayer
2. public String getType(): return “Greedy Player”
3. public void buy(Property f): buy property, he buys land if he can, and he buys house if he can (has enough balance)

**The methods in TacticalPlayer**

1. public TacticalPlayer(String name): create TacticalPlayer
2. public String getType(): return “Tactical Player”
3. public void buy(Property f): if it is the second round, then he will not buy the property

**The methods in Field**

1. Field(String name, int p, int value, boolean buyable): create field, parameters name, position, value of the field, buyablility of the field.
2. public int getPos(): get postion of the field.
3. public String getName(): get the name of the field.
4. public boolean getBuyable(): get the buyability of the field.
5. public int getValue(): get the value of the field.
6. public abstract String getType(): get type of the field (implemented later).
7. public abstract void print(): print the data of the field (implemented later).
8. public abstract Player getOwner(): get the owner of the field.
9. public abstract void removeOwner(): remove the owner of the field.
10. public void setBuyable(boolean b): set the buyability of the field (for property implementation)
11. public void removeField(): remove fields.
12. public abstract boolean isHouseBought(): transitional method for property

**The methods in Property**

1. public Property(String name, int pos, boolean buyable): create Property
2. public String getType(): return “Property”
3. public boolean isHouseBought(): return \_houseBought
4. public int getHousePrice(): return \_housePrice
5. public Player getOwner(): return \_owner
6. public void setHouseBought(boolean b): set \_houseBought as b
7. public void setOwner(Player p): set \_owner to p
8. public void removeOwner(): remove \_owner
9. public void print(): print data of the property

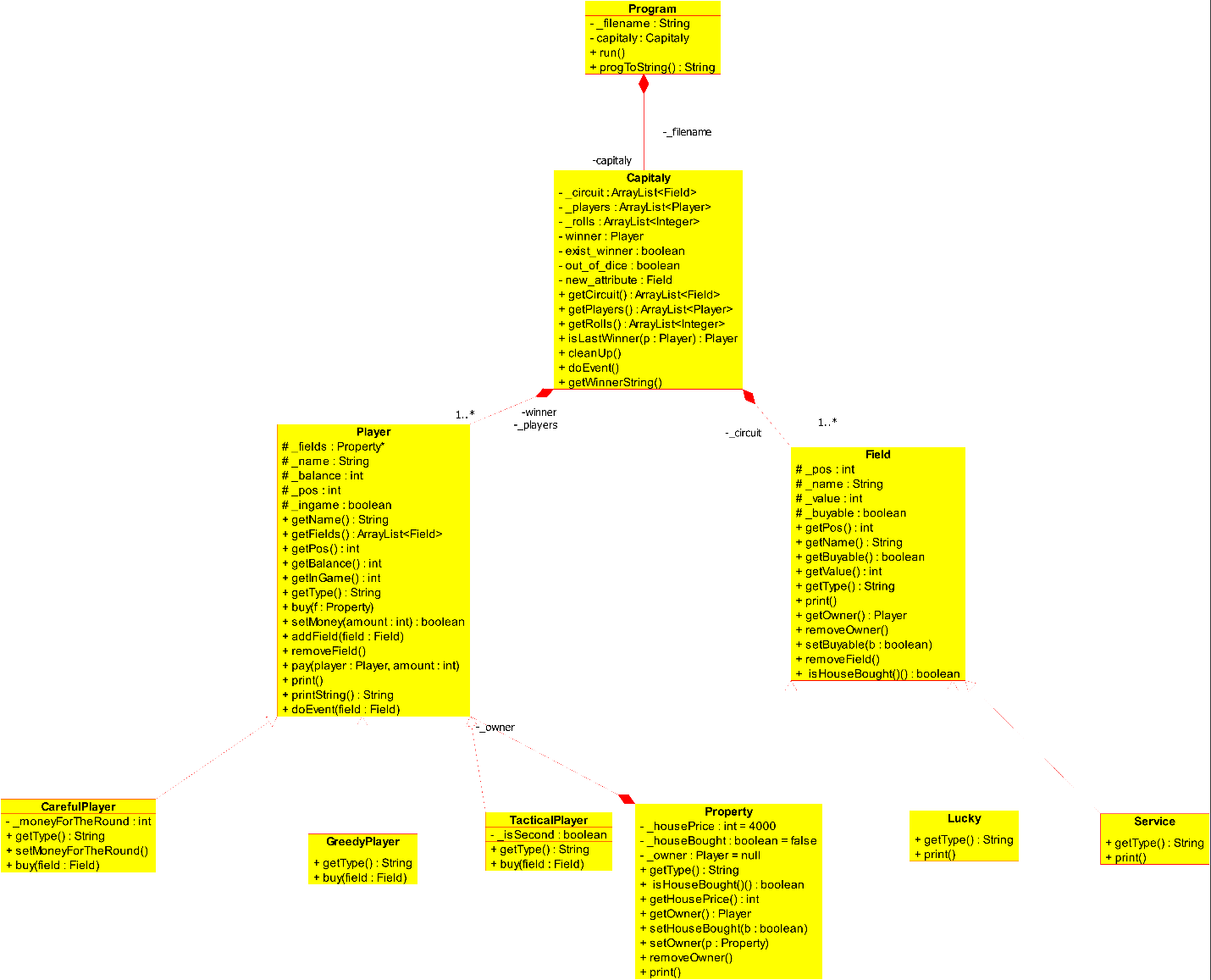
**The methods in Service**

1. public Service(String name, int pos, int fee, boolean buyable): create Service
2. public String getType(): return “Service”
3. public Player getOwner(): return null
4. public void print(): print data of Service
5. public boolean isHouseBought(): return false
6. public void removeOwner(): null

**The methods in Lucky**

1. public Lucky(String name, int pos, int value, boolean buyable): create Lucky
2. public String getType(): return “Lucky”
3. public Player getOwner():null
4. public void print(): print data of Lucky
5. public boolean isHouseBought(): null
6. public void removeOwner(): null

**UML Diagram**

****

**Testing files**

1. sample/sample1\_no\_field.txt
2. sample/sample2\_no\_player.txt
3. sample/sample3\_big.txt
4. sample/sample4\_big2.txt
5. sample/sample5\_only\_tactical.txt
6. sample/sample6\_only\_property.txt
7. sample/sample7\_one\_player.txt
8. sample/sample8\_not\_enough\_dice.txt

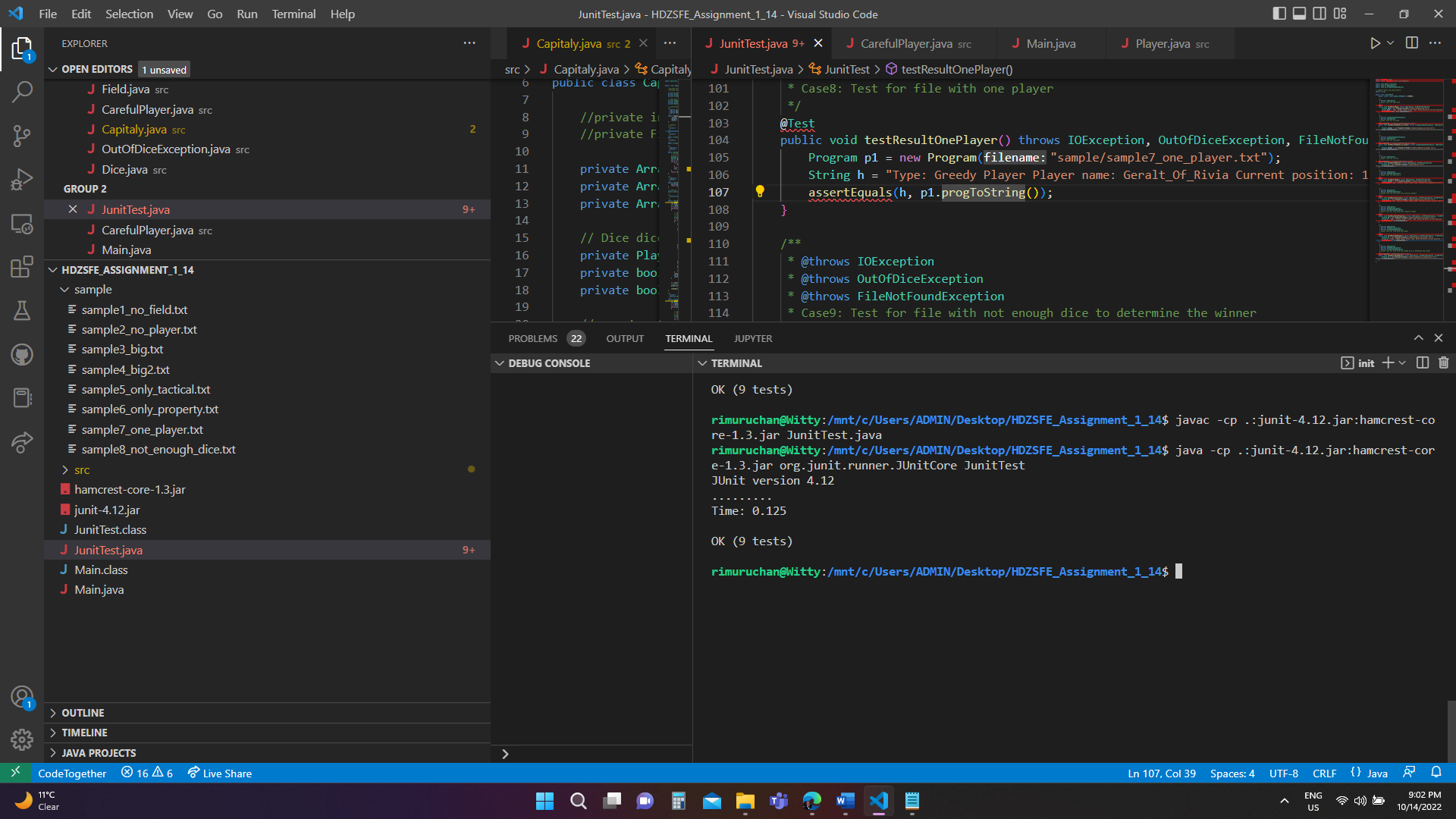
**Testing**

Using JUit4.12 and Hamcrest-core-1.3 to test the correctness of the program.

**Compiling and executing the JUnit testing on Linux:**

javac -cp .:junit-4.12.jar:hamcrest-core-1.3.jar JunitTest.java

java -cp .:junit-4.12.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore JunitTest



1. Case1: Test for big field
2. Case2: Test for no field
3. Case3: Test for no player
4. Case4: Test for no file
5. Case5: Test for big file 2
6. Case6: Test for file with only tactical players
7. Case7: Test for file with only property fields
8. Case8: Test for file with one player
9. Case9: Test for file with not enough dice to determine the winner