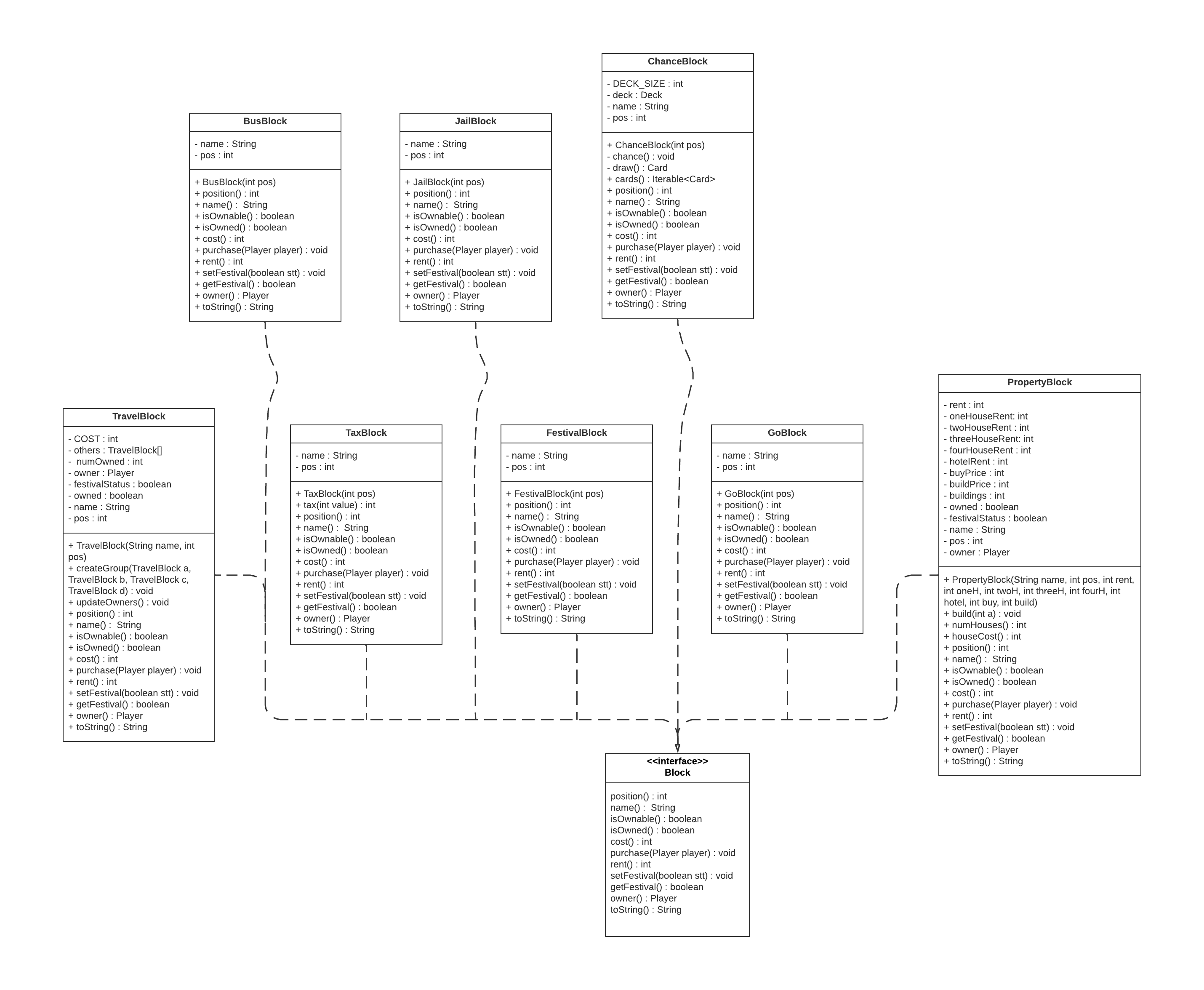
**MONOPOLY OOP REPORT**

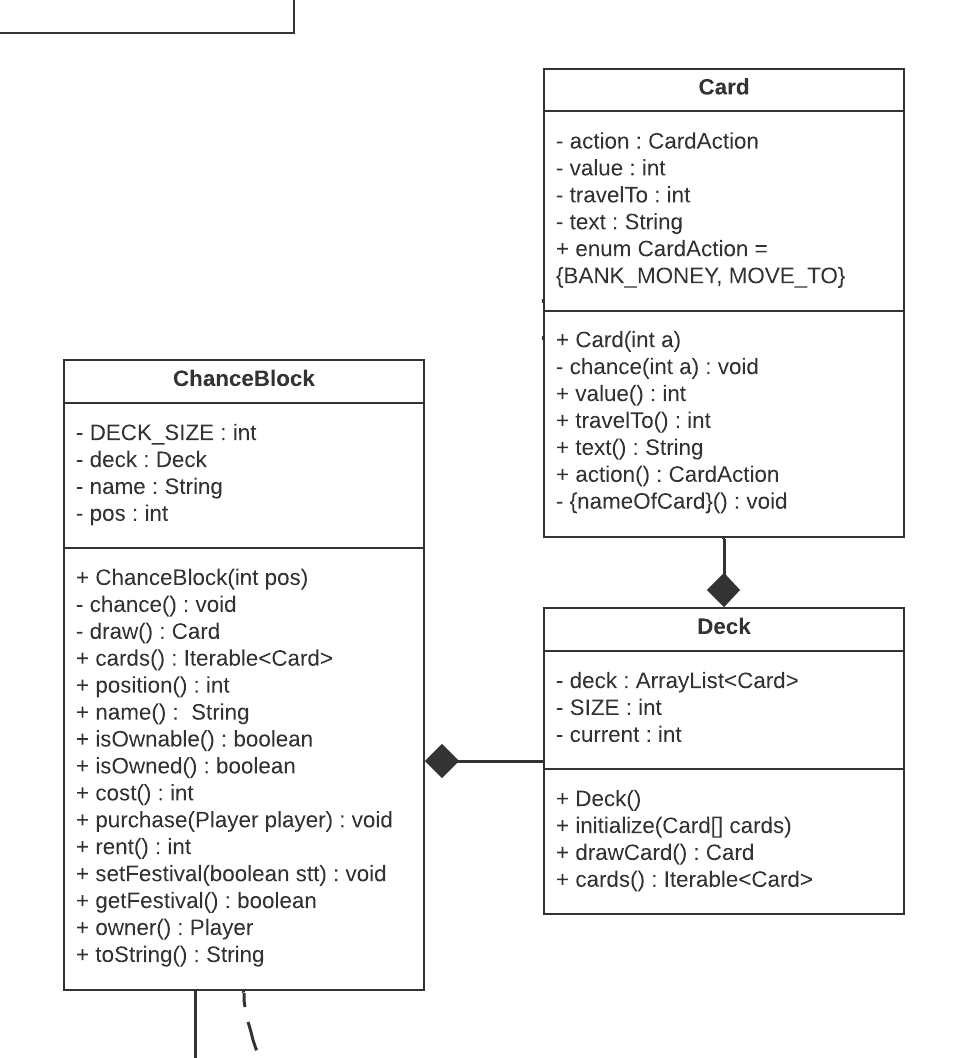
1. **Introduction**

Monopoly is a game, in which the player’ s goal is to remain financially solvent while forcing opponents into bankruptcy by using and developing pieces of property. Each player will roll two dices by turn and go following the total number they have on dices. This report bellow will describe how our game operates.

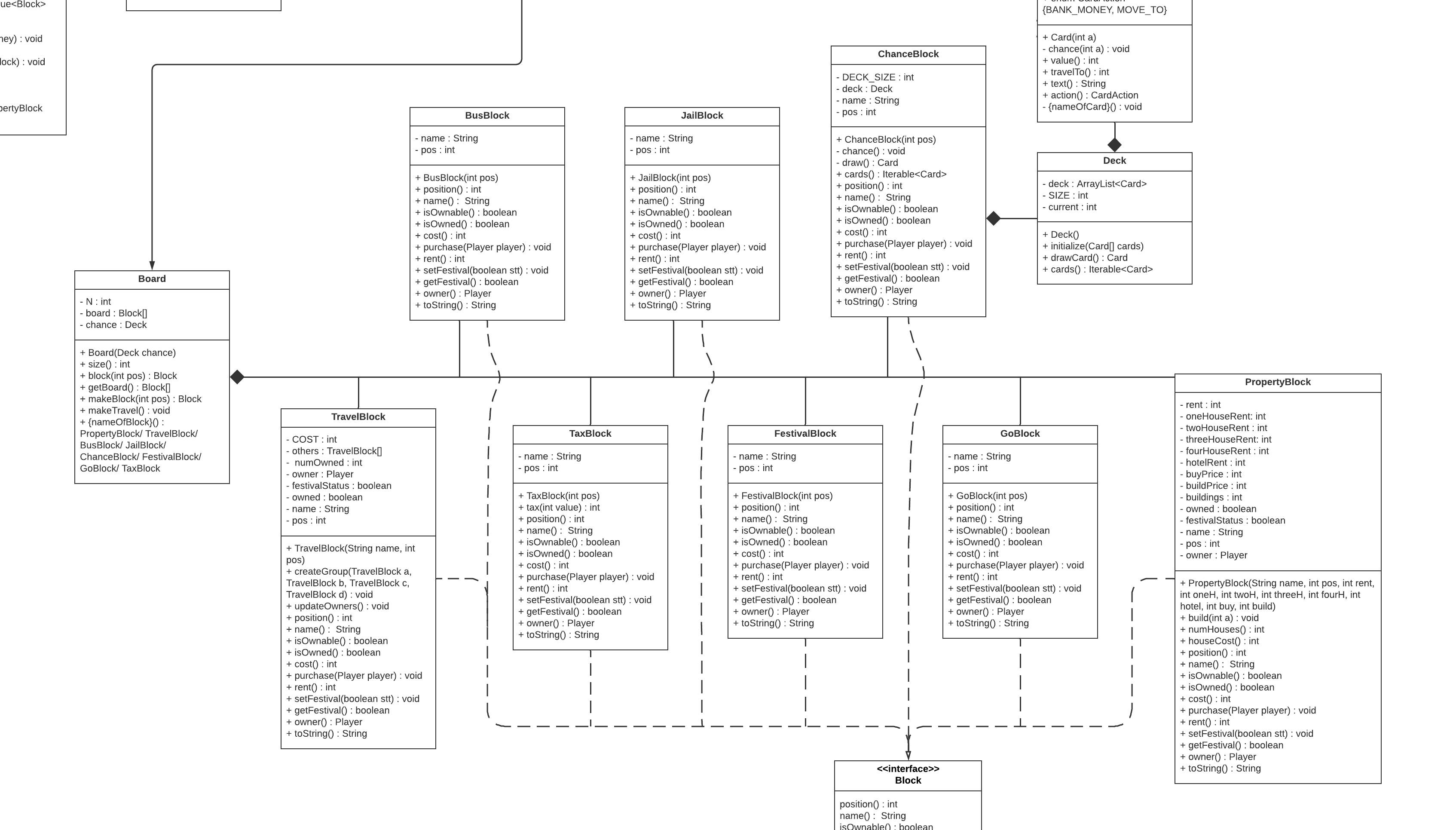
1. **Java Core game**

Firstly, monopoly’s game board is a 32-block board. Each block has unique attributes as well as different methods. In this list of blocks, some of them which are Go, Jail, Festival, Bus, Chance, and Tax are special blocks. However, we realize that all of them have some same methods such that name(), position(), cost(), etc. So, interface class Block is created as a template for classes TravelBlock, TaxBlock, FestivalBlock, GoBlock, PropertyBlock, BusBlock, JailBlock, and ChanceBlock to implement. Then, some of necessary attributes and methods in each block also will also be declared for features to work. 

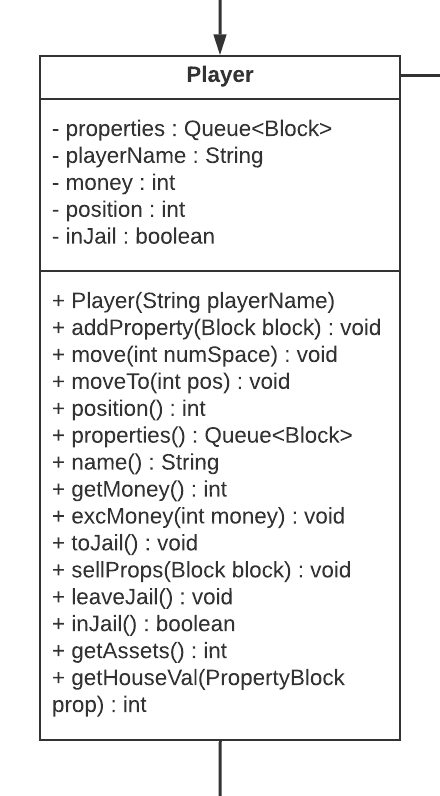
In ChanceBlock, now it is only an “empty” block, we will add a deck of cards for it. Class Deck will be the “container” of class Card. Meanwhile, class Card contains many different chance cards which bring many surprise things to lucky players. When players go into the chance block, they can have ability to draw a card in this deck. In ChanceBlock, its constructor will create an object of deck and call the method chance(). chance() will create an array of object Card and pass them to this deck through the method deck.initialize(cards). Then, when player go into, they will use the method draw(), it will return a card took from method deck.drawCard().



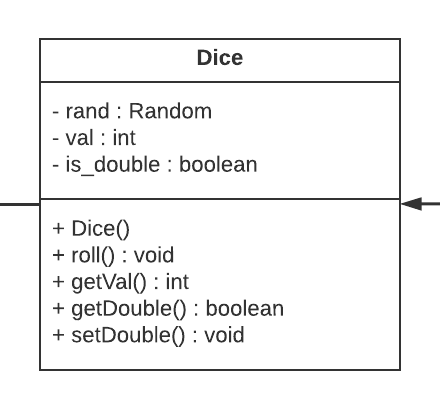
We want a “big” class to control all of “small” blocks. Therefore, we aggregate all them to Board class. Board class is used to create all 32 objects of blocks for game board through method makeBlock(int pos). In game, TravelBlock is a property in which players cannot build houses, they only try to own them as much as possible because rent price will increase by formula: 200 \* (number of TravelBlock you own). So, makeTravel() will group all other travel blocks to each group for each object of TravelBlock. We can count how many travel blocks which is owned by player thanks to this method.



In a game, players are the thing which cannot be lost. Player in monopoly needs some attributes like name, money, some assets, position, as well as some methods such as move(), moveTo(), excMoney(), addProperty(), etc. A full Player class's features which is initialized by us show bellow.

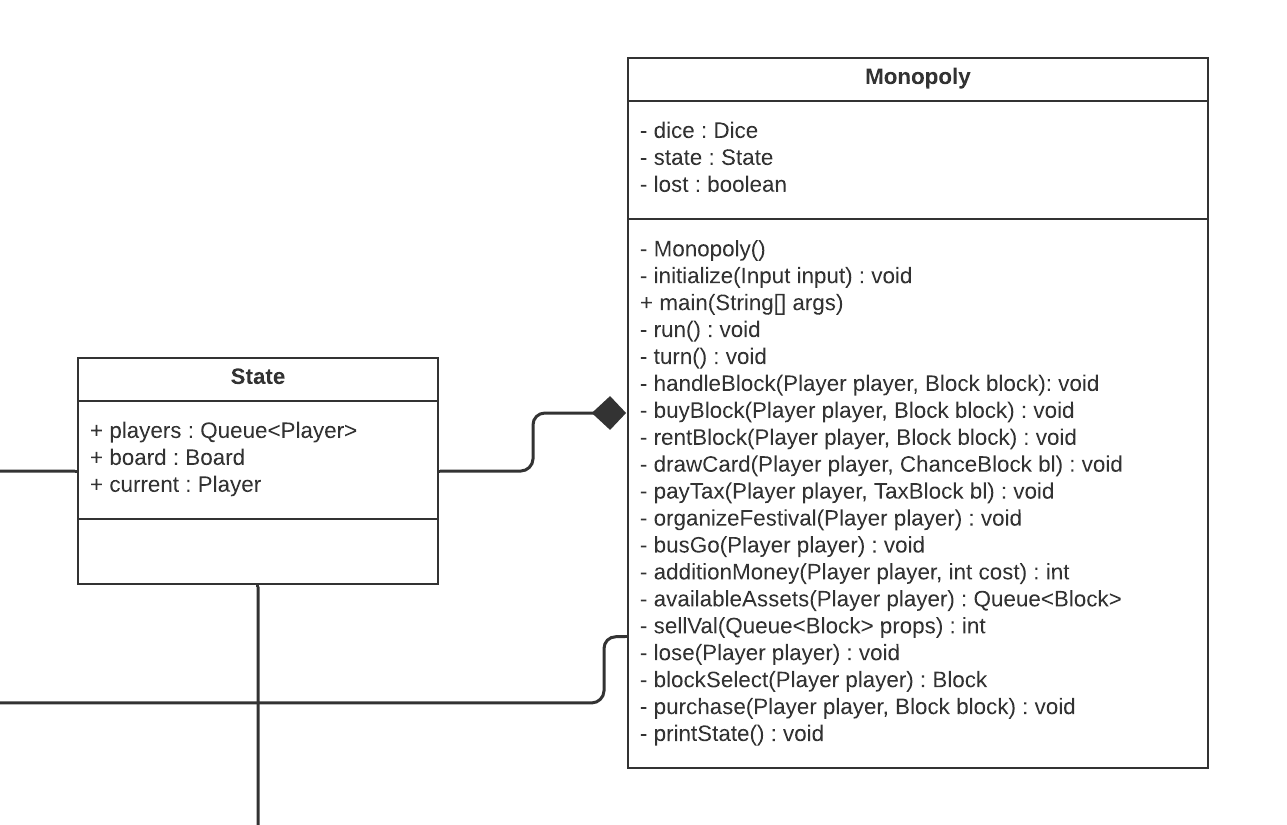


Next, it is apparent that player will roll two dice in their turn. So, class Dice will play a main role for this feature. The private attribute rand will be the Random type and its object will be created in Dice’s constructor. Two remain private attributes: val - saving total of dices’ value, is\_double – checking whether two values are duplicated or not. Roll() method will assign value to these variables.



Finally, main class for our game is Monopoly class. Class State which is an essential class contain list of players in game, a game’s board, and player playing at that time. The process of our game will run in order: Monopoly() -> initialize() (input player, random the order of players’ turn) -> main() -> -> run() (loop until have winner) -> turn() (One player plays each turn) -> printState() -> have winner - END

**In turn():** dice() -> handleBlock() (player will go and operate the feature in destination) -> buyBlock() / rentBlock() / payTax() / toJail() / organizeFestival() / busGo() (we also add some additional abilities in each circumstance which can occurs – additionMoney(), purchase(), sellVal(), etc)



For convenience, we also have class Input to handle all the input in game. For instance, inputBool() for player input their decision (yes, no) to buy house, dice, etc, inputInt() for player input the number of houses they want to build, and inputPlayer for player input their name in Monopoly.initialize().

