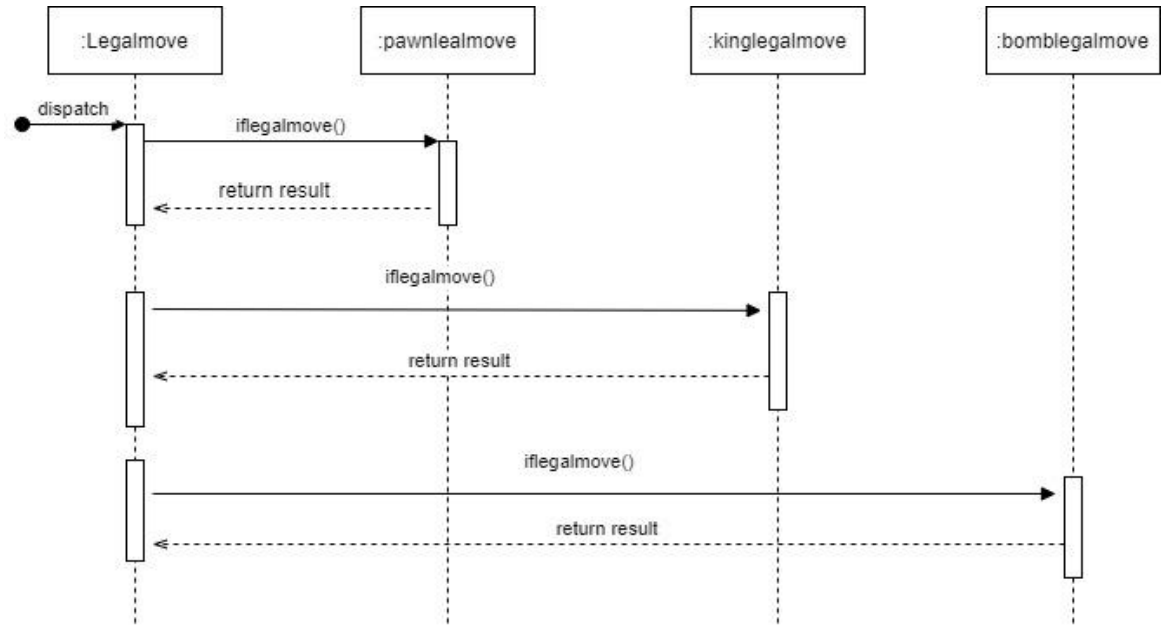
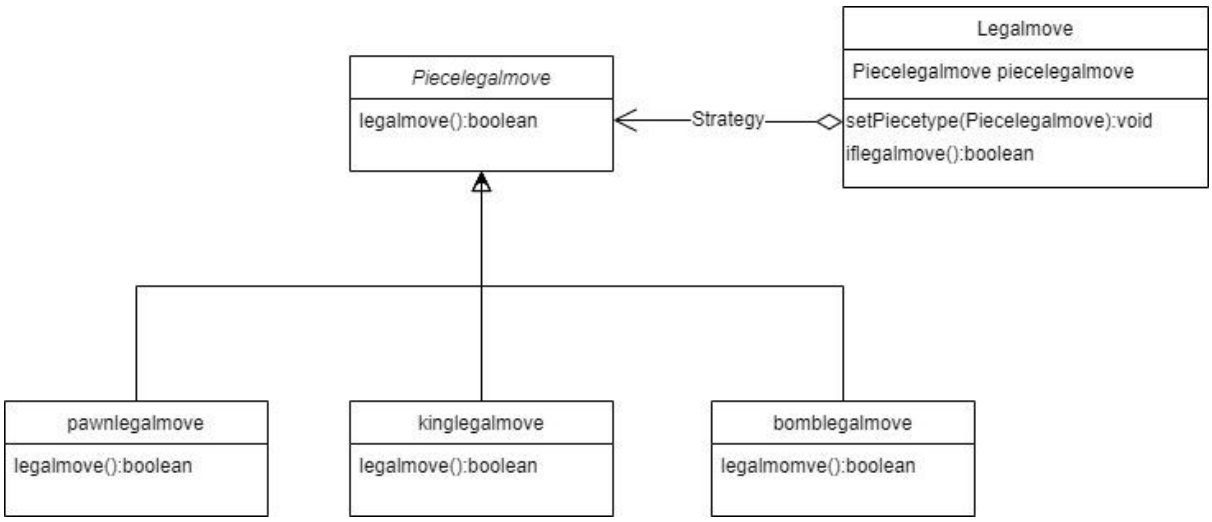
Assignment 3, Group 47

Exercise 1

Strategy pattern:
We implement strategy pattern to judge whether the move of different types of pieces are legal. Different types of pieces(Pawns, Kings and Bombs) have different rules to restrict their movement, in strategy pattern, we create objects represents various strategies to fulfil requirements.
We create an interface Piecelegalmove, three new classes implementing the same interface: pawnlegalmove, kinglegalmove, bomblegalmove, and a context class Legalmove.

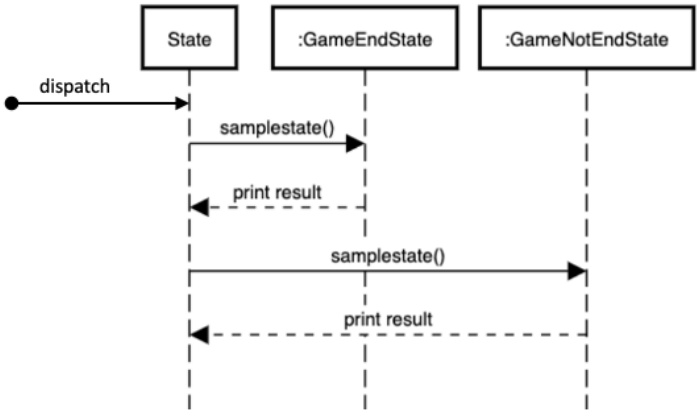


Sequence diagram of strategy pattern

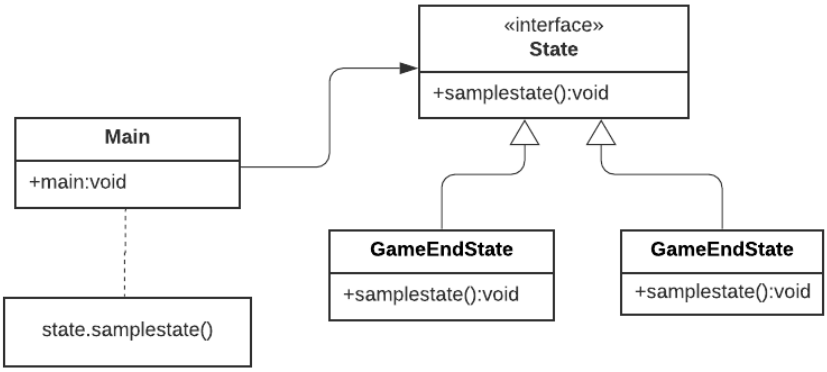


Class diagram of strategy pattern

State Pattern:
We originally wanted to implement the State Pattern for the Pieces, making at first 4 states, RedPawnState, RedKingState, WhitePawnState, WhiteKingState, then when adding the Bombpiece two additional States RedBombState and WhiteBombState, however we ran into a problem we didn't know how to resolve, because when we instatiated a board in the States we always got the error 'java: array required, but com.assignment.board found' even though the board is an array of strings. Since the adding of the Bomb Piece was feasible by mostly adding more if/else Statements to previous methods, we decided the State pattern didn't really make sense with our code when applied to the pieces. So we decided to use the State pattern to replace the method ifend() in the class board with a State ifend, because it seemed more doable and the Exercise requires the implementation of two design patterns.
We also ran into some issues there, with the error 'Exception in thread "main" java.lang.NullPointerException: Cannot invoke "com.assignment.State.ifend()" because "this.state" is null', but at least it only was one error and Player one could enter one move before it appeared. So even though the code was worse than before implementing that method, it kind of worked better than when applied to the pieces.
We then decided to make two States with the method samplestate() instead: GameEndState and GameNotEndState, that print wether the game is over. They are both concrete States to the interface 'State'. The method samplestate() either prints an empty space, so nothing visible if the Game is ongoing, or prints a statement that the game has ended in the GameEndState.

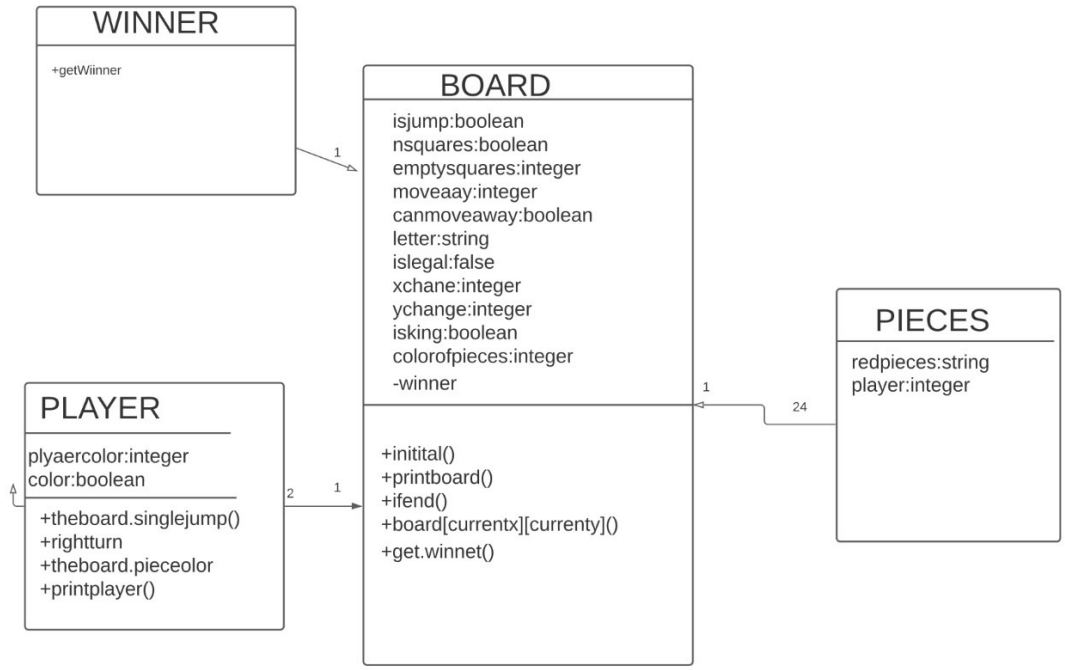


Sequence diagram of State pattern



Class diagram of State pattern

Exercise 2



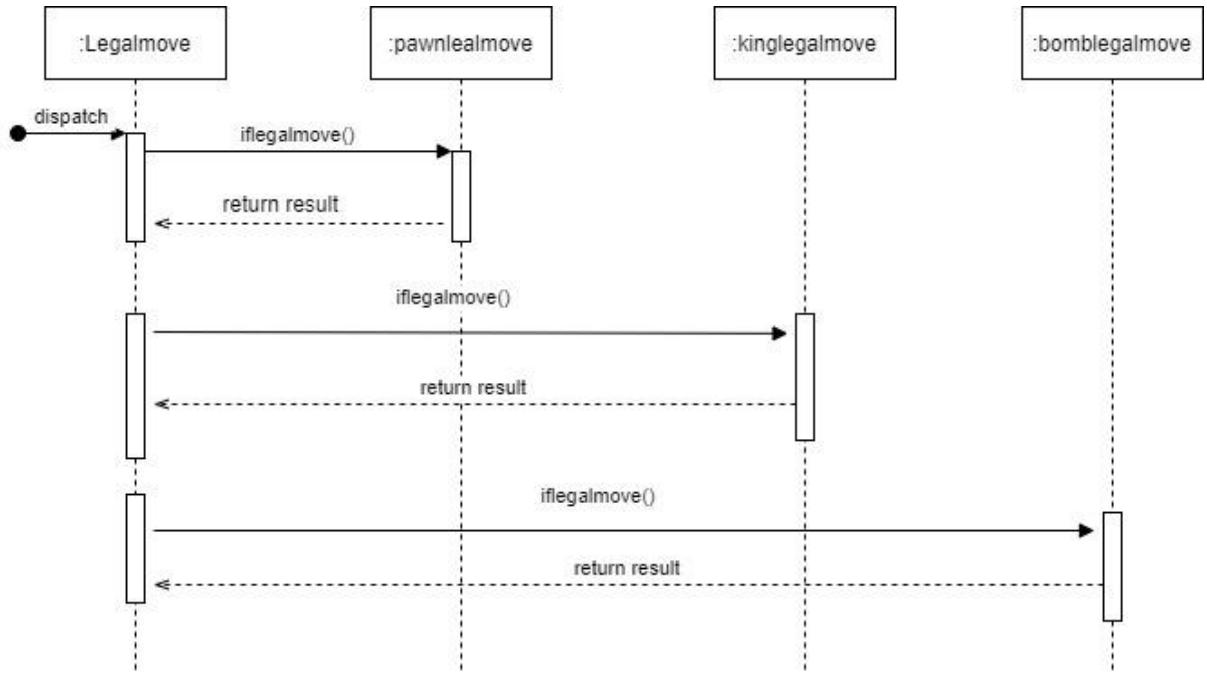
Class diagram

1.We consider these classes are important in our checkers game:

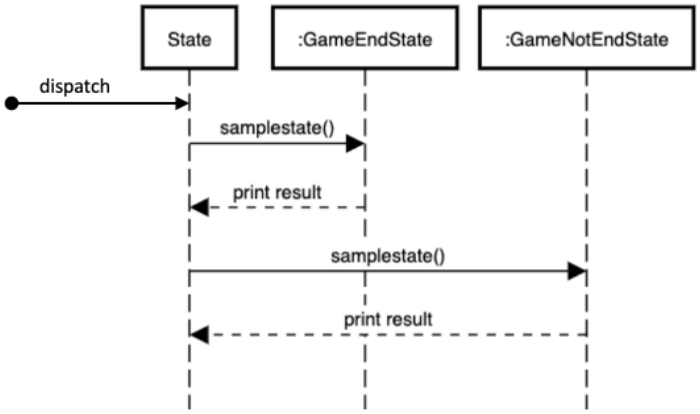
Main;
board;
pieces;
The player;
Legalmove;
pawnlegalmove;
kinglegalmove;

bomblegalmove;
Statecontext;
GameEndState;

The class Main is important, it is always important in java because without it the program does not work. We use the main class to start a new round of game and run the game, to realize the input of the user. It is highly related to other classes of the program.
The classes board, pieces, The player are important, because they correspond to the actual real life objects in a checkers game, that are essential for the game to work. Those classes, especially the board, define most of the behavior of the interaction between the objects.
As we showed in the exercise 1, we implemented the strategy pattern to determine if a movement is valid. This step plays an important role in the game because everytime the user moves a pieces, the game should check whether the movement matches the rule of the checker game. And we create four classes for this design: Legalmove, pawnlegalmove, kinglegalmove and bomb legalmove.



The classe GameEndState is important because it determines wether the game is over. GameEndState also prints out that the game is over and by that informs the user, as is seen in our sequence diagram.

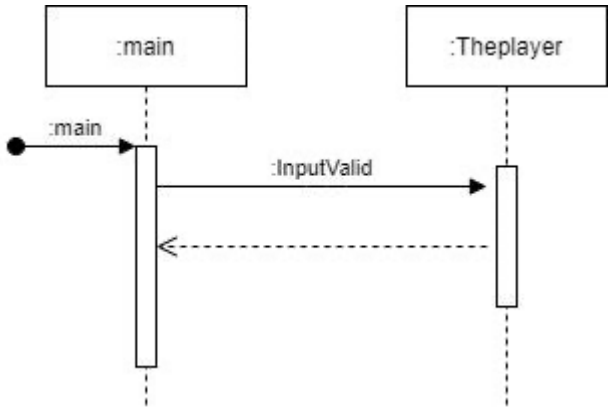


2. Due to some unexpected reasons we can not complete this part this time.

Exercise 3

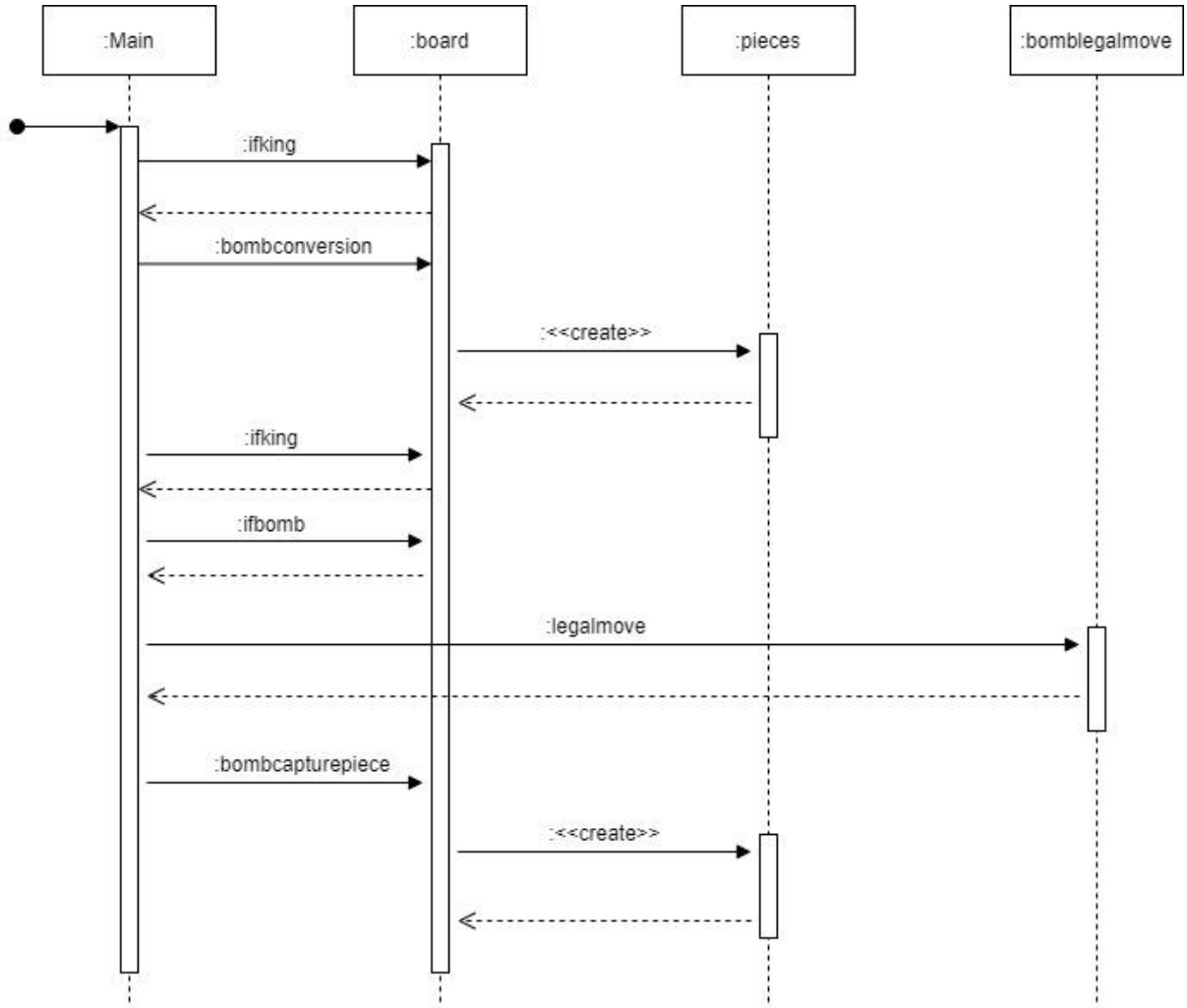
We improved our checkers game in this parts:

1.In the previous program, if the user enter the input in the wrong format, the program would crash. We add a method InputValid in class Theplayer to determine if inputs matches the correct format, and we applied it in class main which makes sure if the user enters the input in the wrong format, the user can enter the input again.

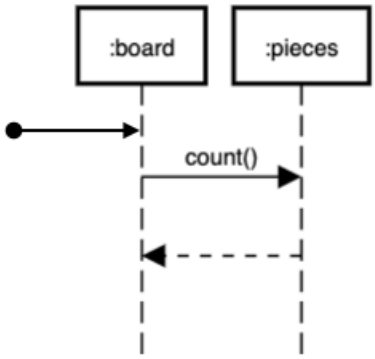


2.We add a new type of pieces called Bomb, When a king piece reaches the row of the board closest to the player, by reason of a simple move, or as the completion of

a jump, it becomes a Bomb. Bomb pieces follow the same movement rules with King pieces, and everytime a Bomb piece jumps, other pieces inside the 3x3 squares with it as the center will be captured.



3. We implemented a function count() to count the number of pieces each player has and called that method in the show() method of the board to display it. We previously counted the pieces only for the ifend() method, but we figured a separate function makes more sense, and also called that for inside the ifend() method.


:board
:pieces
count()