Atypon
Java and DevOps training
console-based Chess game Assignment
Due Aug 27

Instructors :
Fahed Jubair
Motasim Aldiab

Done by :
Abdelrahman Ajawi

Problem Statement:

The problem is to design and implement console-based Chess game that achieves solid design pattern and clean code principles .

main classes :

ChessGame : starting playing and instantiate game members(players , controller,board,pieces).

Player : Player class represents one of the participants playing the game , have ability to move pieces.

Controller : Represents the referee who judges the player's ability to move pieces and promotion pawn .

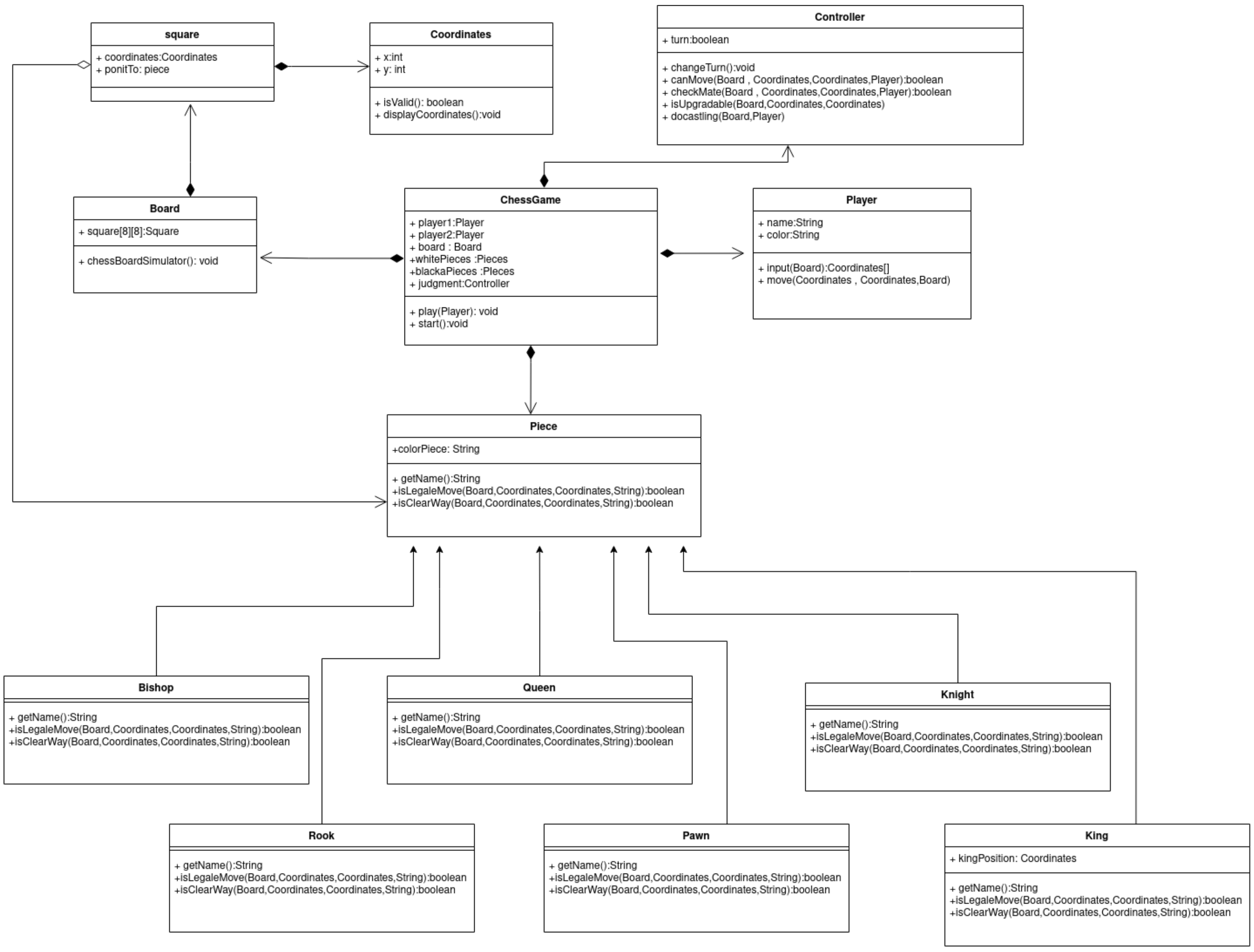
Board: Board is an 8×8 set of squares containing all active chess pieces**.**

Square : Square: contains its position on the board and indicates what is above it.

Piece: The basic building block of the system, every piece will be placed on a square. Piece class is an abstract class. The extended classes (Pawn, King, Queen, Rook, Knight, Bishop) implements the abstracted operations.

Rules: Cheetsheet of movement rule in chess game.

High Level Design :


square
+ coordinates:Coordinates
+ ponitTo: piece
Coordinates
+ x:int
+ y: int
+ isValid(): boolean
+ displayCoordinates():void
Controller
+ turn:boolean
+ changeTurn():void
+ canMove(Board , Coordinates,Coordinates,Player):boolean
+ checkMate(Board , Coordinates,Coordinates,Player):boolean
+ isUpgradable(Board,Coordinates,Coordinates)
+ docastling(Board,Player)
Board
+ square[8][8]:Square
+ chessBoardSimulator(): void
ChessGame
+ player1:Player
+ player2:Player
+ board : Board
+whitePieces :Pieces
+blackaPieces :Pieces
+ judgment:Controller
+ play(Player): void
+ start():void
Player
+ name:String
+ color:String
+ input(Board):Coordinates[]
+ move(Coordinates , Coordinates,Board)
Piece
+colorPiece: String
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
Bishop
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
Queen
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
Knight
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
Rook
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
Pawn
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean
King
+ kingPosition: Coordinates
+ getName():String
+isLegaleMove(Board,Coordinates,Coordinates,String):boolean
+isClearWay(Board,Coordinates,Coordinates,String):boolean

- ChessGame have player1 , player2 ,board , whitePiece , blackPiece and judgment , it has composition relationship with all members because what is the benefit for being a live after the game in this situation .

- Create Piece class encapsulate common attributes and behaviors from king , Queen , Bishop, knight , Rook,Pawn that reduce redundancy in the code.

- Board have 64 square that help Controller to make decision easily , and reduce complexity in code when play move pieces.

- Square have aggregation relationship with piece that makes it easy to move the piece to and from it .

- Coordinates class associated with all other class .

Implementation Rules of movement :

- king :
- diagonal : check slop equal one or minus one
- vertical :
  check the difference between destination-x and source-x equal -1,1 and
  check the difference between destination-y and source-y equal 0
- horizontal :
  check the difference between destination-y and source-y equal 0 and
  check the difference between destination-x and source-x equal 1,-1

- Queen :
- diagonal : check slop equal one or minus one
- vertical :
  check the difference between destination-x and source-x equal 0
- horizontal :
  check the difference between destination-y and source-y equal 0

- Bishop :
- diagonal : check slop equal one or minus one

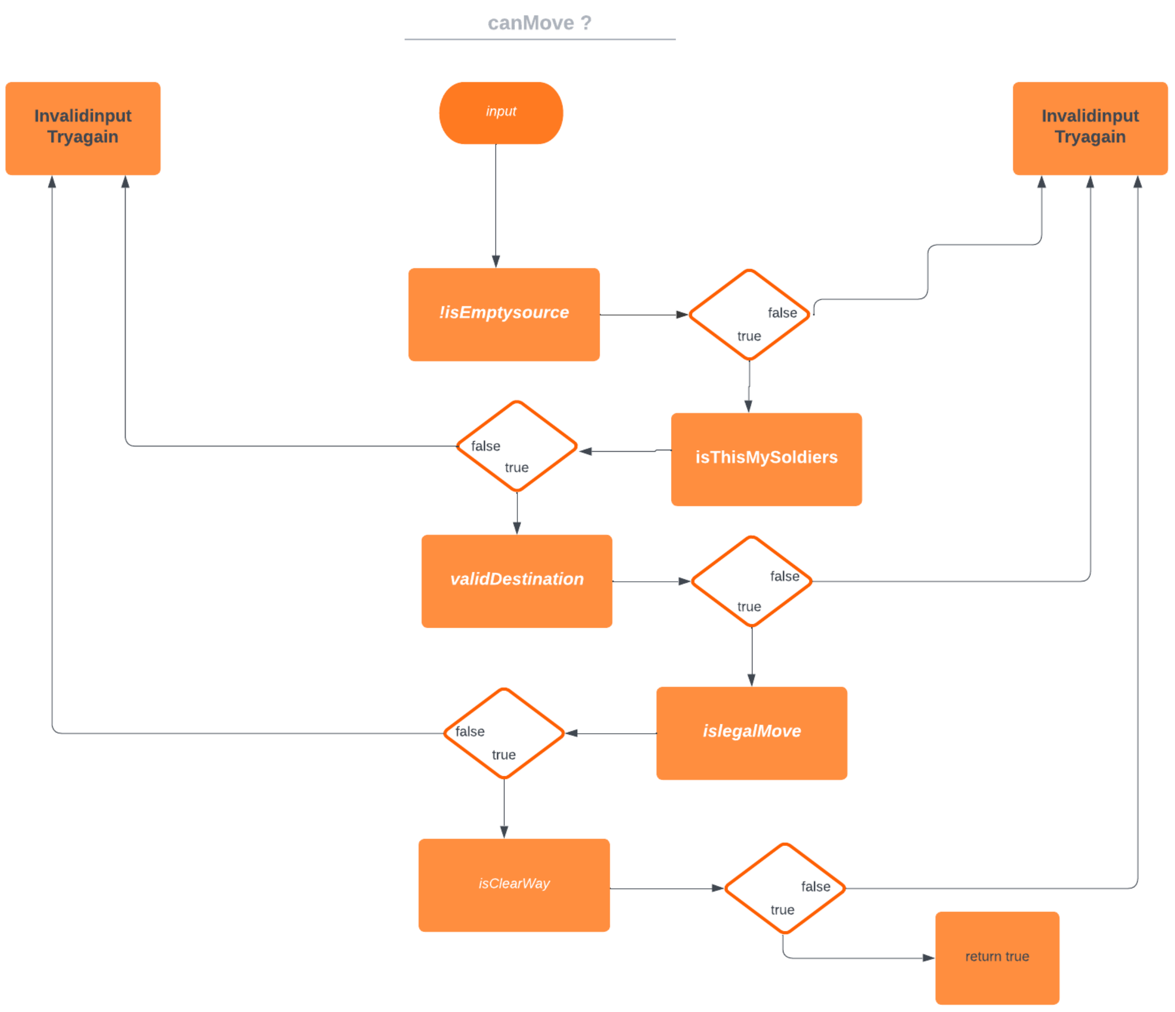
- knight :
difference between destination-y and source-y =2,-2 and
difference destination-x and source-x =1,-1
or
difference between destination-x and source-x =2,-2 and
destination-y and source-y =1,-1

- Rook :
vertical : check the difference between destination-x and source-x
equal 0
horizontal : check the difference between destination-y and source-y
equal 0

- Pawn :
diagonal :
forward up one square diagonal if there an enemy in this place
vertical :
not move yet it can move two square or one square
difference between destination-y and source-y equal 2
difference between destination-x and source-x equal 0
otherwise :
difference between destination-y and source-y equal 1
difference between destination-x and source-x equal 0

process for movement :

input

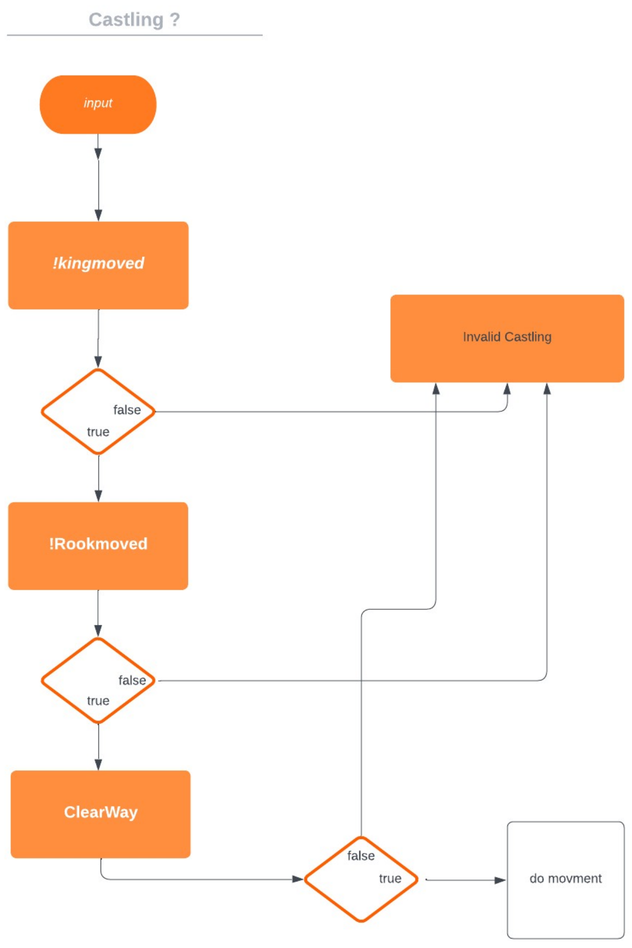


if true do movement .

Then check for possibility to kill the king by check isilegalMove --- > ClearWay

special movement :

- castling :



-Promotion :

if pawn move to last square in his vertical direction then you have to upgrade it

same way I checking validate to promotion function in my code :

```
if(destinationCoordinate.getY() equal 8 ) for white pawn
 if( destinationCoordinate.getY()=1 ) for black pawn
```

SOLID and clean code principles:

- as possible I do my code to achieves Single responsibility principles by by make methods and classes do one job related to his name that lead to higher cohesion and Lower coupling: .

- make piece super type and king , Queen , Bishop, knight , Rook, Pawn sub type   ensures that inheritance is not misused ;

- My code is clean as possible away from having duplicate code , long parameter ,large method ,large classes ,presence of data clumps (instead having class Coordinates in my code) ,rely on using primitive types and too much comments