Chess Game Design

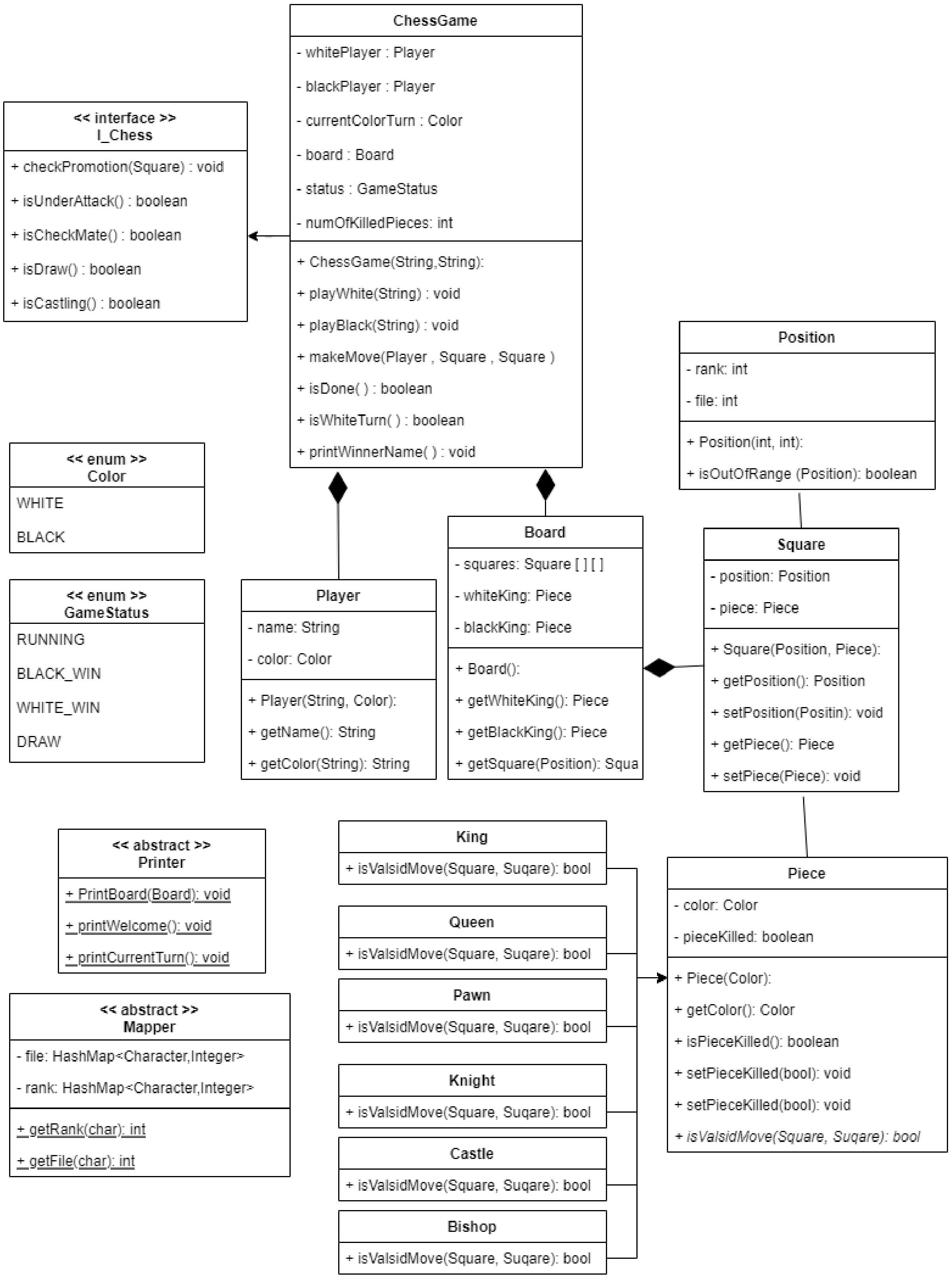
Dr. Fahed Jubair

Mohammad Amjad AlQuraan

Main Class:

- Create new ChessGame instance in the main method
- While loop until the game ends
- Ask for move from the players using the method:
  read_move_from_console()
- Use the RegEx to check the validation of user input:
  Pattern pattern = Pattern.compile("move [a-h][1-8] [a-h][1-8]", Pattern.CASE_INSENSITIVE);

Object Oriented Design:


ChessGame
- whitePlayer : Player
- blackPlayer : Player
- currentColorTurn : Color
- board : Board
- status : GameStatus
- numOfKilledPieces: int
+ ChessGame(String,String):
+ playWhite(String) : void
+ playBlack(String) : void
+ makeMove(Player , Square , Square)
+ isDone( ) : boolean
+ isWhiteTurn( ) : boolean
+ printWinnerName( ) : void
<< interface >>
I_Chess
+ checkPromotion(Square) : void
+ isUnderAttack() : boolean
+ isCheckMate() : boolean
+ isDraw() : boolean
+ isCastling() : boolean
Position
- rank: int
- file: int
+ Position(int, int):
+ isOutOfRange (Position): boolean
<< enum >>
Color
WHITE
BLACK
<< enum >>
GameStatus
RUNNING
BLACK_WIN
WHITE_WIN
DRAW
Player
- name: String
- color: Color
+ Player(String, Color):
+ getName(): String
+ getColor(String): String
Board
- squares: Square [ ] [ ]
- whiteKing: Piece
- blackKing: Piece
+ Board():
+ getWhiteKing(): Piece
+ getBlackKing(): Piece
+ getSquare(Position): Squa
Square
- position: Position
- piece: Piece
+ Square(Position, Piece):
+ getPosition(): Position
+ setPosition(Positin): void
+ getPiece(): Piece
+ setPiece(Piece): void
<< abstract >>
Printer
+ PrintBoard(Board): void
+ printWelcome(): void
+ printCurrentTurn(): void
<< abstract >>
Mapper
- file: HashMap<Character,Integer>
- rank: HashMap<Character,Integer>
+ getRank(char): int
+ getFile(char): int
King
+ isValsidMove(Square, Suqare): bool
Queen
+ isValsidMove(Square, Suqare): bool
Pawn
+ isValsidMove(Square, Suqare): bool
Knight
+ isValsidMove(Square, Suqare): bool
Castle
+ isValsidMove(Square, Suqare): bool
Bishop
+ isValsidMove(Square, Suqare): bool
Piece
- color: Color
- pieceKilled: boolean
+ Piece(Color):
+ getColor(): Color
+ isPieceKilled(): boolean
+ setPieceKilled(bool): void
+ setPieceKilled(bool): void
+ isValsidMove(Square, Suqare): bool

Piece (Abstract):

- Define the common piece attributes for all chess pieces
  - Color
  - Killed statues
- Setters and getters for the attributes
- Abstract class isValidMove() to define the validation of each piece type inherits the Piece class

Pawn, King, Queen, Rook, Knight, Bishop:

- Classes extends the Piece class and implement the abstract method
- Scince each class has different rules of movement the will implement the isValidMove() to check if the move is valid or not

Position:

- The class responsible for holding board coordination
- Class attributes:
  - rank
  - file
- Class methods:
  - IsOutOfRange(Position p)
    A static method check if position out of range

Square:

- The class represent a square from the 68 squares on the board and may holds pieces on it
- Class attributes:
  - Piece
  - Position
- Class methods:
  - Getters and setters

Board:

- The class represent the chess board with 64 squares and 32 pieces on it
- The board instantiated when new ChessGame is instantiated
- Class attributes:
    - 2D array of squares [8][8]
    - White King & Black King:
      to track their status
- Class methods:
    - Constructer:
        - When a new board created it initialize the 64 squares with all pieces king, queen ...etc. and the empty squares have Null
    - Getter for a square at specific position
    - Getters for the kings

Player:

- The class represent and hold the information of the chess player
- The player objects instantiated when new ChessGame is instantiated.
- Class attributes:
    - Name
    - Color
- Class methods:
    - Getters for the attributes

Chess Game:

- The class represent the flow of the game and keep track of the game status
- Class attributes:
    - White Player
    - Black Player
    - currnetColorTurn
    - board
    - status
    - numOfKilledPieces
- Class methods:
    - Constructer
        - Takes two strings as a parameter to create two players
          first name is the white and the last is the black player
        - Create new board
        - Set the status of the game to running
        - Set the current color turn to white
        - Print welcome and print the board to the players
        - Print the current turn player to let him make a move
    - PlayWhite(String move): take the move as a string extract the move information and call the makeMove() function to make the move
    - PlayBlack(String move): take the move as a string extract the move information and call the makeMove() function to make the move
    - makeMove() takes the player, source square and destination square
        - check move validation
        - make the move
        - check game condition after the move and call the printer to print the board
    - isDone() is game still running
    - isWhiteTurn() check turn
    - checkPromotion() if pawn reaches the last rank
    - isDraw() check draw condition
    - isCastling() check for castling move

Mapper (abstract):

- The class responsible of mapping character position of file and rank to integer to get the correct square from the 2D array
- Attributes:
    - HashMap(character, integer) file
    - HashMap(character, integer) rank
- Static Methods:
    - getRank(char c)
    - getFile(char c)

Printer(abstract):

- The class responsible of printing methods
- Static methods:
    - printBoard()
    - printWelcome()
    - printCurrentTurn()

SOLID Principles:

- I tried to break and analyze all entities of the problem Pieces, board, square, Player ...etc. so each entity has single responsibility of itself, same for the methods inside entities
- Each piece type king, queen …etc. has a full is-a relationship with the abstract class and each one of them is open for extend and has its own abstract implementation of the abstract method
- Subtypes of the abstract class Piece will not break the Piece functionality
- Square class does not depend on specific concrete class of pieces subtypes it depends on the abstract piece class

Clean Code Principles:

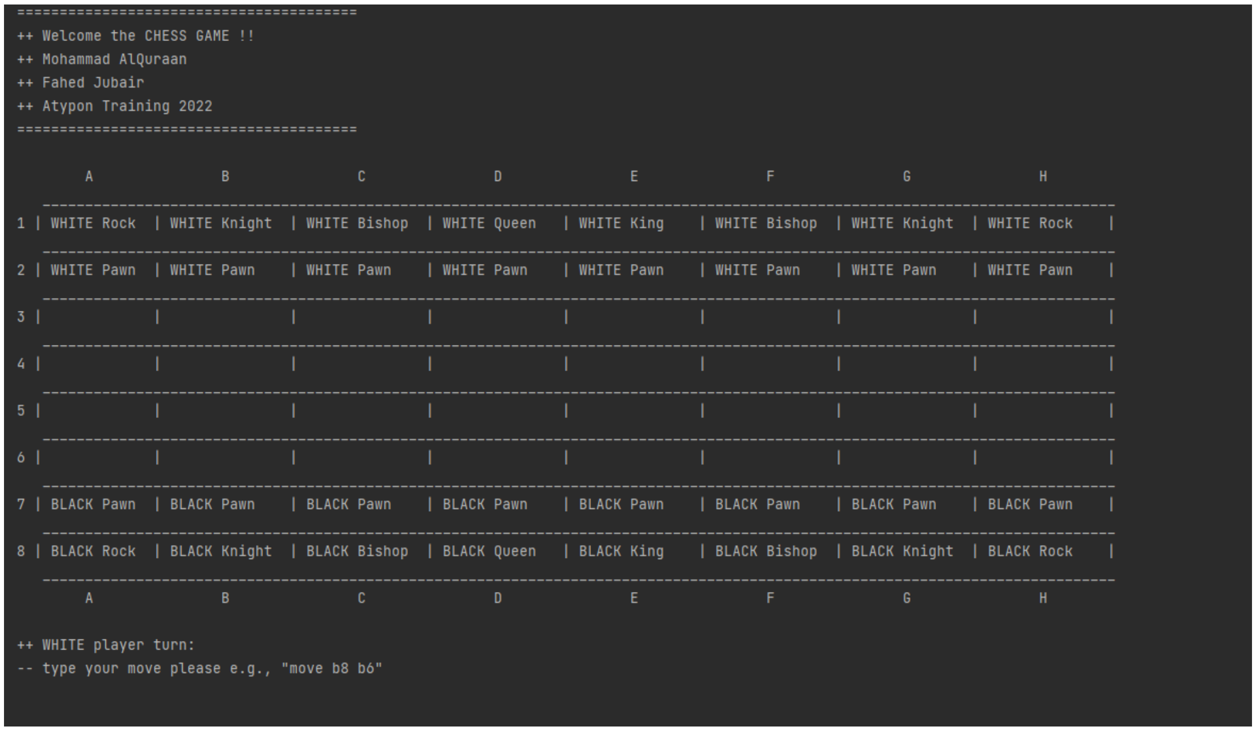
- I tried to choose descriptive and unambiguous names as much as I can for attributes and methods, and I used camel Case
- Almost all methods short and do one thing and have few arguments no more than 3 arguments
- Boolean prefix with is: isDone() isDraw() isValidMove() … etc.
- Loose coupling because I tried to stick with single responsibility as much as I can
- No useless comments I tried to let the code simple and speak for itself

Effective Java:

- Override toString() in the pieces classes
- Minimize the accessibility of classes and members
- In public classes, I used accessor methods (getters) and mutators (setters), not public fields
- Favor composition over inheritance the game has player and board, and the board has a square
- Avoid Strings where other types are more appropriate, so I convert the move source and destination given by the player to integers and access them by 2d array
- It tried to Minimize the scope of local variables as much as I can

Visualizing the flow of the game:

- At the beginning of the game and after each move the board is printed to the players



```
========================================
++ Welcome the CHESS GAME !!
++ Mohammad AlQuraan
++ Fahed Jubair
++ Atypon Training 2022
========================================

       A              B              C              D              E              F              G              H

   -------------------------------------------------------------------------------------------------------------------------------
1 | WHITE Rock    | WHITE Knight  | WHITE Bishop  | WHITE Queen   | WHITE King    | WHITE Bishop  | WHITE Knight  | WHITE Rock    |
   -------------------------------------------------------------------------------------------------------------------------------
2 | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    | WHITE Pawn    |
   -------------------------------------------------------------------------------------------------------------------------------
3 |               |               |               |               |               |               |               |               |
   -------------------------------------------------------------------------------------------------------------------------------
4 |               |               |               |               |               |               |               |               |
   -------------------------------------------------------------------------------------------------------------------------------
5 |               |               |               |               |               |               |               |               |
   -------------------------------------------------------------------------------------------------------------------------------
6 |               |               |               |               |               |               |               |               |
   -------------------------------------------------------------------------------------------------------------------------------
7 | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    | BLACK Pawn    |
   -------------------------------------------------------------------------------------------------------------------------------
8 | BLACK Rock    | BLACK Knight  | BLACK Bishop  | BLACK Queen   | BLACK King    | BLACK Bishop  | BLACK Knight  | BLACK Rock    |
   -------------------------------------------------------------------------------------------------------------------------------
       A              B              C              D              E              F              G              H

++ WHITE player turn:
-- type your move please e.g., "move b8 b6"
```