A Project **Report** (including **Synopsis)** on

Basic Chess game

Submitted to Manipal University, Jaipur
Towards the end-semester mini project submission of

Object Oriented Programming in Java

In Computer Science and Engineering

2019-2023

By

Shubh Gupta

(199301305)

Under the guidance of

Dr. Ankit Shrivastava

Department of Computer Science and Engineering

School of Computing and Information Technology

Manipal University Jaipur

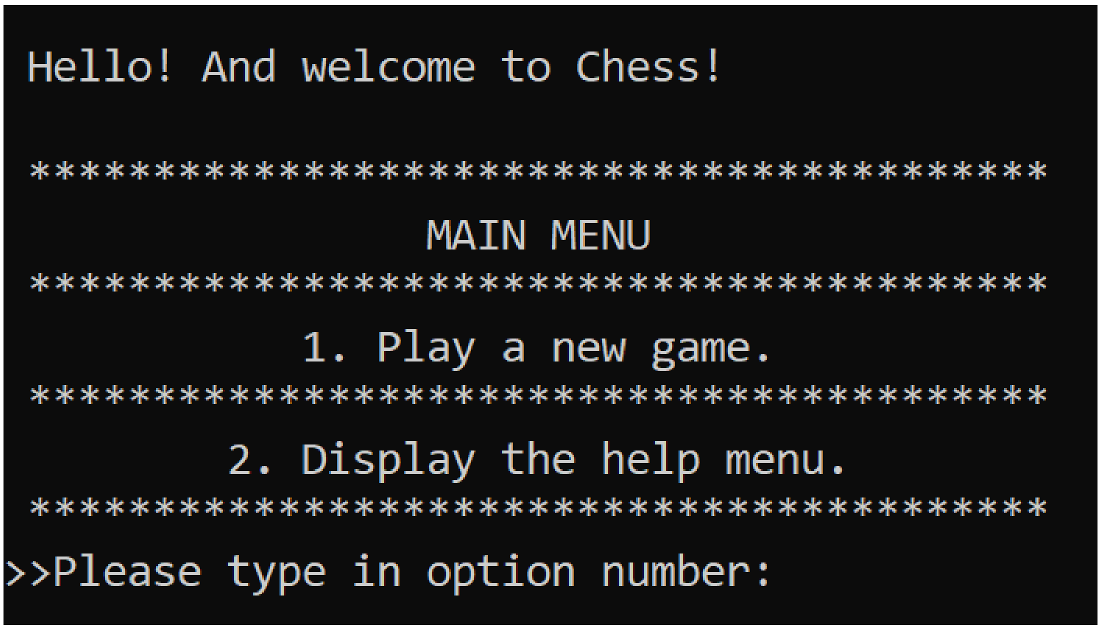
Jaipur, Rajasthan

INTRODUCTION

For my project, I chose to design a Java application that allows two users to play chess. It is built using native java and vanilla java with only inbuilt libraries, which can be further extended to have a chess engine where a player can play against a CPU player(It has not been implemented yet).

The chess game is designed by using multiple classes that work together to form the game.

While, the main function of playing chess is present and the application is open to future improvements. Potential improvements include a GUI, finishing the saving/loading system, interfacing with the cloud to play against someone remotely using Hadoop or something similar, etc.



```
 Hello! And welcome to Chess!

 *****************************************
                 MAIN MENU
 *****************************************
            1. Play a new game.
 *****************************************
         2. Display the help menu.
 *****************************************
>>Please type in option number:
```

PROJECT OBJECTIVES

The major objective is to get more familiarized with the concept of object-oriented programming and learning the application of Polymorphism, Abstraction, Encapsulation, Multi-threading, Exception Handling and File Handling.

TECHNOLOGIES USED

- Visual Studio Code
- Java
- Launcher4j
- JDK and JRE
- Windows PowerShell

METHODOLOGY

The chess game is designed by using multiple classes that work together to form the game. Following is a description of each class as well as some of the more important class methods.

As the classes are described, the logic used to design the chess game as a whole will become

apparent.

Class ChessMain :

This is the chess game's top module, which allows the user to actually play the game.

Primarily this class is programmed in the main method. This is because my game is based

in the command line and so all of the user's inputs have to go through the terminal. This

class has nothing too special in it except for compiling everything together to play the game!

Class Piece

Class Piece is used to define each of the 32 pieces used in a chess game. Each Piece is con-

structed in one of two ways: empty, or with a colour, type, and row and column coordinate

specified as given parameters. The different piece types are programmed into Class Piece as

constants and are as follows:

• King = 1

• Queen = 2
• Rook = 3
• Knight = 4
• Bishop = 5
• Pawn = 6

The methods in Piece are used to access its data and other classes utilize these methods to
view and change the data in a certain Piece. One method unique to Piece is getPieceName(),
which returns the piece's shortened board name for display on the text based game in the
terminal. This function is used by the Board class.

Class Board

Class Board holds the information for the chess board in each game. The board is designed
to be completely dynamic, being updated each time a player makes a move. The board itself
is an 8x8 Piece array, where blank spaces are empty Piece objects. Board also contains an
ArrayList object lled with the 32 pieces with which each game of chess starts. I chose to
use ArrayList objects throughout my program for a few reasons. First, they allow me to
create type-safe containers so that the wrong object will never be placed into the wrong list.
Next, because I cannot predict the order that pieces will be captured from game to game,

ArrayList's ability to access its elements randomly is really handy. Additionally, as I will
describe later, ArrayList can hold another ArrayList as its object which was extremely useful
when keeping track of the location of each piece on the board.
In Board, the Piece ArrayList holding all of the game's pieces is _lled using the createPieces()
class. Pieces are simply added each game by using ArrayList's method \add(element)". This
method either creates a classic chess board if the user chooses to play a game of classic chess.
If the NEW NAME game was chosen, the user is prompted through the command line to
choose the location of their queen, rooks, knights, and bishops.
Another method of note in Board is the populateBoard() method. This is used to update
the board array each time the Piece object ArrayList is changed.
The method currentGameState() is the graphical user interface for my text based version of
chess. Each time it is called, it outputs to the terminal a display of the current game state.
This method makes use of Piece's method getPieceName() while building the game board
each turn.
To update the game board, the method updateGameBoard() is called. This method _rst

calls \clearBoard()" which as the name suggests, clears the board and then uses populate-
Board() to _ll the board array.
The remaining methods in Board are used to access its members as well as \pawnPromo-
tion(color)" that promotes any pawn of the given color that made it to the opposite end
line.

Class Move
Class Move was the most meticulous class out of them all to write. Chess has so many factors
to consider when moving pieces across the board: is the path clear? If not, what type of
piece is blocking the path, your own or your enemy? If it is an enemy, can you capture it
next turn?
These are the type of questions that I had to ask myself while I was coding the Move class.
Move is constructed with a single parameter of the current game board. This is because
Move needs an updated board to determine if the proposed move is valid.
The primary method in Move is \boolean move(currentSpace, destinationSpace, player)"
and I will use its description to describe many of the other notable methods in Move. Both

currentSpace and destinationSpace are Strings that are input by the user from the command
line. The strings are converted from input form: a letter followed by a number (example -
\d3") into an integer row value and column value. This conversion is done using the meth-
ods \inputToRow(String)" and \inputToCol(String)". Once the current space's coordinates
and the destination space's coordinates are known, these values are checked to make sure
that they are legal values. If they are, the piece at the current space coordinates is found
using the method _ndPiece(row,col)". Then an array list of an array list of integers (Ar-
rayList<ArrayList<Integer>>) is formed with a list of all of the found piece's available moves.
This is done using the method \legalPieceMoves(piece, boolean)" which takes input param-
eters Piece and a boolean (true if king is in checkmate, false otherwise) and returns all of
the input piece's allowed moves. It does this by _lling the _rst ArrayList in the returned
ArrayList with allowed row values and the second ArrayList is _lled with the column values
that correspond to each row. The method knows the possible values by utilizing methods
like: \possKingMoves", \possQueenMoves", etc. that when called return row, column Ar-
rayLists with that speci_c piece type's allowed moves. Once the to-be-moved piece's legal

moves have been determined, the list of moves is iterated though until the destination coor-
dinates match up with a legal move space's coordinates. If they match then the piece can
be moved to the target space. Additionally, if they match and an opponent is on the space,
the opponent is captured. However if the destination coordinates do not match any of the
piece's allowed move coordinates, the move is a failure and the user is prompted to input a
valid move.

Class Help

This primary function of this class is to help the user, as the name states. When Help's
method \display()" is called, a few options are shown to the user: see application information,
see basic chess rules, see chess scramble rules, castle piece, and quit game.
This class, along with FileSystem, are the two classes that did not get completely get _nished.
Ideally, the player could use Help to access some of chess's special commands like: castle,
surrender, quit game, save game, load game. However currently it is being used for just basic
information output to the user.

CONCLUSION

From this “ChessGame” I learned to apply the concepts of Object - Oriented Programming and learned to use the basics and advanced topics in Java. It was a wonderful experience.