RV COLLEGE OF ENGINEERING®
BENGALURU – 560059
(Autonomous Institution Affiliated to VTU, Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


# "Chess game with JavaFx"

MINI-PROJECT REPORT
OBJECT ORIENTED PROGRAMMING USING JAVA (18CS45)
IV SEMESTER

2020-21

Submitted by

Abdur Rahaman – 1RV19CS001
Harihar s pawar – 1RV19CS054

Under the Guidance of

**Dr.Poonam Ghuli**
**Department of CSE, RVCE,**
**Bengaluru - 560059**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING


CERTIFICATE

Certified that the Mini-project work titled "Chess game with javafx" has been carried out by Abdur Rahaman(1RV19CS001) and Harihar S Pawar(1RV19CS054), bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the Assessment of Course: OBJECT ORIENTED PROGRAMMING USING JAVA (18SC45) – Open-Ended Experiments during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report.

**Faculty Incharge**
Department of CSE,
RVCE., Bengaluru –59

**Head of Department**
Department of CSE,
RVCE, Bengaluru–59

RV COLLEGE OF ENGINEERING®, BENGALURU - 560059
(Autonomous Institution Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, Abdur Rahaman(1RV19CS001) and Harihar S Pawar(1RV19CS054) the students of 4th Semester B.E., Department of Computer Science and Engineering, RV College of Engineering, Bengaluru hereby declare that the Mini-Project titled “Chess game with JavaFx” has been carried out by us and submitted in partial fulfillment for the Assessment of Course: OBJECT ORIENTED PROGRAMMING USING JAVA (16CS44) - Open-Ended Experiment during the year 2020-2021.

Place: Bengaluru Abdur Rahaman Signature

Date: Harihar S Pawar Signature

# Abstract:

We build a 2 player chess game.In this project rules of the chess game are implemented. There are 2 players black and white. The game can be played in 2 modes human vs human and human vs computer. The result of the match is declared at the ending and the same is stored in a text file. There is no implementation of any advanced algorithm  to calculate the next move since the primary objective of this project is to demonstrate OOP and JavaFX concepts hence little consideration is given to implementation of any advanced algorithm.

# 1. Introduction

## 1.1. Traditional vs Object Oriented Approach

Traditional approach:

- Traditional approach is process oriented or procedure oriented.
- Uses common processes likes: analysis, design, implementation, and testing.
- It is primarily based on creating and calling procedures or functions
- Follows a top down approach
- Examples of languages that are based on traditional approach: C, Pascal, etc

Object oriented approach

- This approach is based on the concepts of objects.
- Uses UML notations likes: use case, class diagram, communication diagram, development diagram and sequence diagram.
- It is primarily based on creating and defining classes.
- Need to more time than Traditional approach and leads that to more cost.
- Follows a bottom up approach.
- Examples of languages that are based on object oriented approach: Java, python, c++, etc.

### 1.1.1. OOA, OOD & OOP and their Relationship

OOA is concerned with developing an object model that captures the requirement examines the requirements of a problem through the classes and objects found in the vocabulary of the problem domain

OOD is concerned with translating the OOA model into a specific that can be implemented by software leads to Object Oriented decomposition and uses logical and physical notations to represent the static and dynamic aspects of the system

OOP is concerned with realising the OOD model using an OO programming language such as Java or C++ uses objects not algorithms: each object is an instance of a class which is related to another via inheritance relationships

## 1.2. Overview of Java Programming language

### 1.2.1. Object Oriented –

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

- Platform Independent − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform-independent bytecode. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

- Simple − Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.
- Secure − With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.
- Portable − Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. The compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.
- Robust − Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.
- Multithreaded − With Java multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.
- Interpreted − Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.
- High Performance − With the use of Just-In-Time compilers, Java enables high performance.
- Distributed − Java is designed for the distributed environment of the internet.
- Dynamic − Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry an extensive amount of run-time information that can be used to verify and resolve accesses to objects at run-time.

### 1.2.1. Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods and fields) of another. With the use of inheritance the information is made manageable in a hierarchical order.

The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

### 1.2.2. Interfaces & Packages

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

A Package can be defined as a grouping of related types (classes, interfaces, enumerations and annotations ) providing access protection and namespace management

### 1.2.3. Exception Handling

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, therefore, these exceptions are to be handled.

Exception handling ensures that the flow of the program doesn't break when an exception occurs. For example, if a program has bunch of statements and an exception occurs mid way after executing certain statements then the statements after the exception will not execute and the program will terminate abruptly.

By handling we make sure that all the statements execute and the flow of program doesn't break.

### 1.2.4. Multithreaded Programming

Java is a multi-threaded programming language which means we can develop multi-threaded program using Java. A multi-threaded program contains two or more parts that can run concurrently and each part can handle a different task at the same time making optimal use of the available resources specially when your computer has multiple CPUs.

By definition, multitasking is when multiple processes share common processing resources such as a CPU. Multi-threading extends the idea of multitasking into applications where you can subdivide specific operations within a single application into individual threads. Each of the threads can run in parallel. The OS divides processing time not only among different applications, but also among each thread within an application.

Multi-threading enables you to write in a way where multiple activities can proceed concurrently in the same program.

### 1.2.5. Lambda Expressions

Lambda expression is a new and important feature of Java which was included in Java SE 8. It provides a clear and concise way to represent one method interface using an expression. It is very useful in collection library. It helps to iterate, filter and extract data from collection.

The Lambda expression is used to provide the implementation of an interface which has functional interface. It saves a lot of code. In case of lambda expression, we don't need to define the method again for providing the implementation. Here, we just write the implementation code. Java lambda expression is treated as a function, so compiler does not create .class file.

### 1.2.6. Regular Experssions

A regular expression (regex) defines a search pattern for strings. The search pattern can be anything from a simple character, a fixed string or a complex expression containing special characters describing the pattern.
A regex can be used to search, edit and manipulate text, this process is called: The regular expression is applied to the text/string.

### 1.2.7. Strings

Strings, which are widely used in Java programming, are a sequence of characters. In Java programming language, strings are treated as objects.The Java platform provides the String class to create and manipulate strings.

### 1.2.8. Collection Framework

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects. Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

### 1.2.9. JavaFX framework

JavaFX is an open source Java-based framework for developing rich client applications. It is comparable to other frameworks on the market such as Adobe Flex and Microsoft Silverlight. JavaFX is also seen as the successor of Swing in the arena of graphical user interface (GUI) development technology in Java platform. The JavaFX library is available as a public Java application programming interface (API).

## 1.3. Proposed system

### 1.3.1 Objectives:

A 2 player chess game is built using JavaFx and OOP concepts. The objective of the current project is to demonstrate various object oriented concepts and javafx framework that we have learned in the course 18CS44.

### 1.3.2. Methodology:

A standalone application is built that runs a chess game on a JavaFX framework. The whole program is divided into 2 packages Move and Game based on the primary tasks that each package performs.

The *game* package: This handles the necessary data and functions related to players
It has 3 primary classes:

*Main* class: To begin the application
*Player* class: Encapsulates a human player
*Comp* class: Encapsulates a computer player

The *Move* package: This handles the necessary data and functions related to board and rules of the game.
It has 3 primary classes:

*Board* class: Encapsulates a board

*Tile* class: This class directly extends stackpane node from the JavaFx framework. This encapsulates a tile in the board that has background color(white or black) and an image node that represents a chess piece(eg. Pawn or bishop).

*move* class: This class encapsulates the rules of the game(eg. Checking of a valid move) and operations that are performed on the board(eg. Removing a piece).

### 1.3.3 Scope:

- Almost all the OOP concepts are demonstrated that were taught in the course.
- Legal moves of a chess game are implemented and a basic chess board is created using JavaFx framework.
- Does not employ any sophisticated or advanced AI algorithm to determine the next move of the computer.
- Stores the result of a match in a text file.
- Can be further improvised by adding an AI algorithm(eg. minmax) and by including more detailed rules of the game.

# 2. Requirement specification

## 2.1 Hardware requirements:

No complex hardware required except a computer supporting java runtime environment.

## 2.2 Software requirements:

- Java JDK(latest version is recommended)
- JavaFx SDK(latest version is recommended)
- IntelliJ IDE

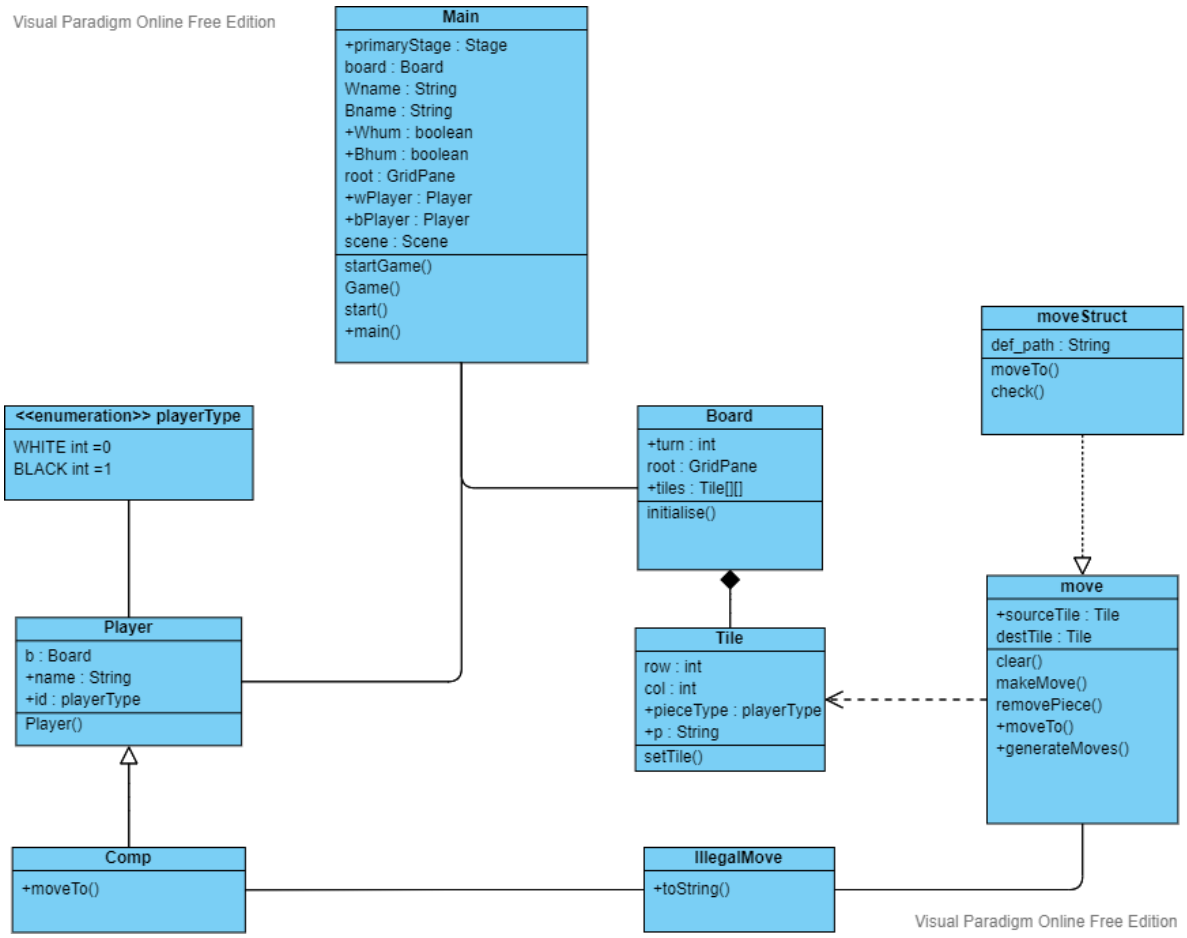
# 3. System design and implementation

## 3.1 Class diagram:

Class diagram is a static diagram. It represents the static view of an application. Class diagram is not only used for visualizing, describing, and documenting different aspects of a system but also for constructing executable code of the software application.

Class diagram describes the attributes and operations of a class and also the constraints imposed on the system. The class diagrams are widely used in the modeling of objectoriented systems because they are the only UML diagrams, which can be mapped directly with object-oriented languages.

Class diagram shows a collection of classes, interfaces, associations, collaborations, and constraints. It is also known as a structural diagram




## 3.2 Modular description/pseudo code:

There are 2 packages: *game* and *Move*

*game* package has 3 primary classes that defines its functionality:

*Main* class: The *Main* class extends *Application* class from the JavaFx framework and has the main function that launches the application

This class primarily performs 2 functions:

- Creates a form to collect details of player: Uses a VBox as a container and loads the name textfield and the radiobuttons to choose the kind of player i.e either a computer or human for both black and white player. The scene contains 2 textfields for name, 2 sets of radiobutton and a submit button.
- Creates a concluding form
- Contains a label that declares the winner
- Contains 2 buttons: *exit* to exit the application and *analyze* to analyze the previous results that is stored in a text file

- Other functions of *Main* class is to create the board object and player objects to start the game.

*Player* class: The *Player* class contains the details of the human player

- 2 important fields *name* and *id*
- *name* is a string object that contains the name of the player
- *id* is a *playerType* object(which is an enum) that refers to the type of the player i.e either black or white

*Comp* class:

- The *Comp* class contains the details of the computer player.
- This class inherits the *Player* class.
- This class does one additional function that is it chooses a random piece of appropriate type and makes a random move through it.

*Move* has 3 primary classes that defines its functionality

*Board* class: This class creates and initializes the tiles and keeps track of the current turn between the players. The first turn is always made by the white player.

*Tile* class: *Tile* class extends Stackpane of the JavaFx. Precisely 64 tiles are created which forms a matrix of 8 rows and 8 columns. The background color of the tiles is alternated between black and white which forms a 8X8 chess board. *Tiles* also contain image node which refers to the image of a piece. *Tile* listens to action events and responds appropriately.

*move* class: This class contains all the rules and constraints of a game. This class is involved in performing the actual move by removing and adding the image nodes of the tiles according to the defined rules of the games. If an illegal move is made then the board is not updated and an illegal move exception is thrown.

The scene graph:
3 scenarios:

1. The first form, that collects the player's details, has:
   - A stage called *primaryStage*.
   - A scene called *scene*.
   - A root node called *root* of type Grid pane.
   - A VBox called *contatiner*.
   - 2 Text fields and 2 sets of radio buttons.
   - A submit button

   Text fields and buttons are added to the container, container is added to the root node, root node is included in the scene and it is displayed.

2. The chessboard has:
   - An array of *Tiles*(which extends stackpane) arranged in the form of a chess board
   - Each tile has a background color either white or black and an image node representing a piece if it is occupied

- The array is added to the root node which is then added to a scene and the scene is set on a stage and the stage is displayed.

3. The concluding scene:
   - Has 2 buttons *exit* and *analyze*
   - Has a label that declares the winner
   - The label and the buttons both are added to a VBox
   - VBox is added to the root node
   - The root node is added to the scene and scene is set on the stage and stage is displayed.

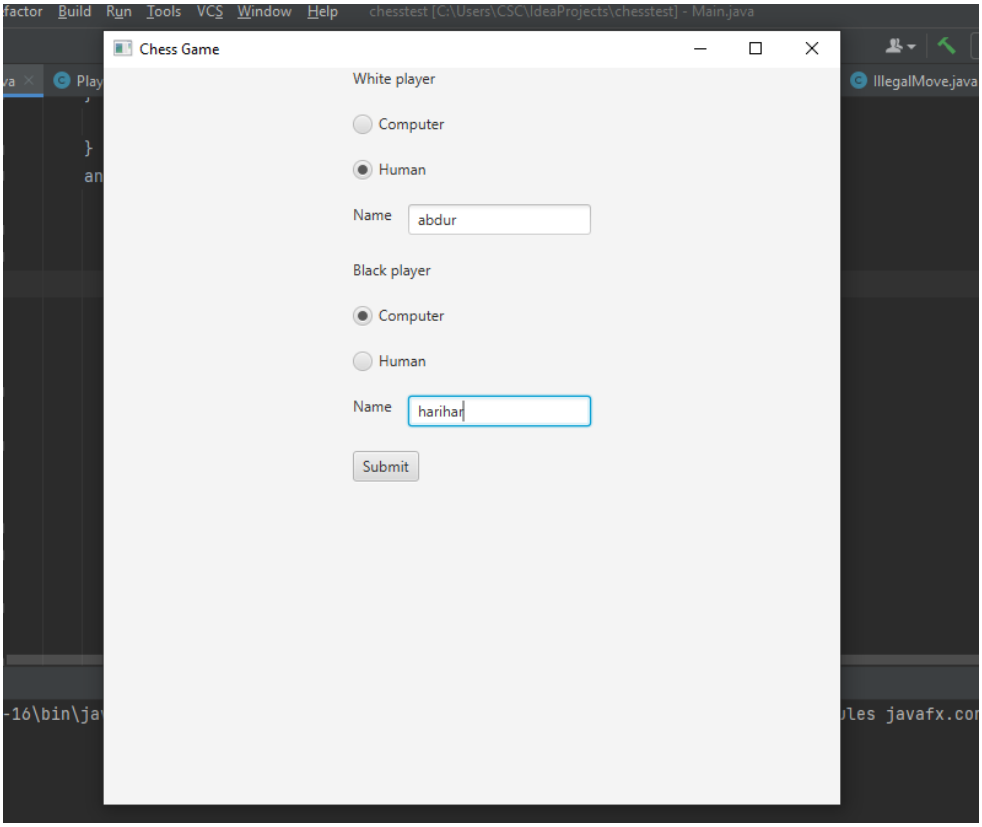
OOP concepts used:

- Inheritance: *Comp* class inherits *Player* class.
- Interface: *move* class implements the interface *movestruct*
- Lambda Expression: A functional interface legal with a single abstract method called *islegal*() is used to create a lambda expression to check the validity of a move. Lambda expression is implemented in the *move* class. It is defined at around line number 55 and called at around line number 81 of *move* class.
- Multithreading: 2 threads are used in calculating the next move. Multithreading is implemented in the *Comp* class.
- Exception handling: An exception class called *IllegalMove* is created to throw an exception if any illegal move is made by the user.
- Collection framework: Arraylists and its functionalities are extensively used in the *generateMoves*() function of the move class which stores all the possible moves in the form of a boolean 2X2 matrix implemented using ArrayLists.
- RegularExpression: Regex is used in the *exit*() method of *Main* class to analyze the results of the winner.

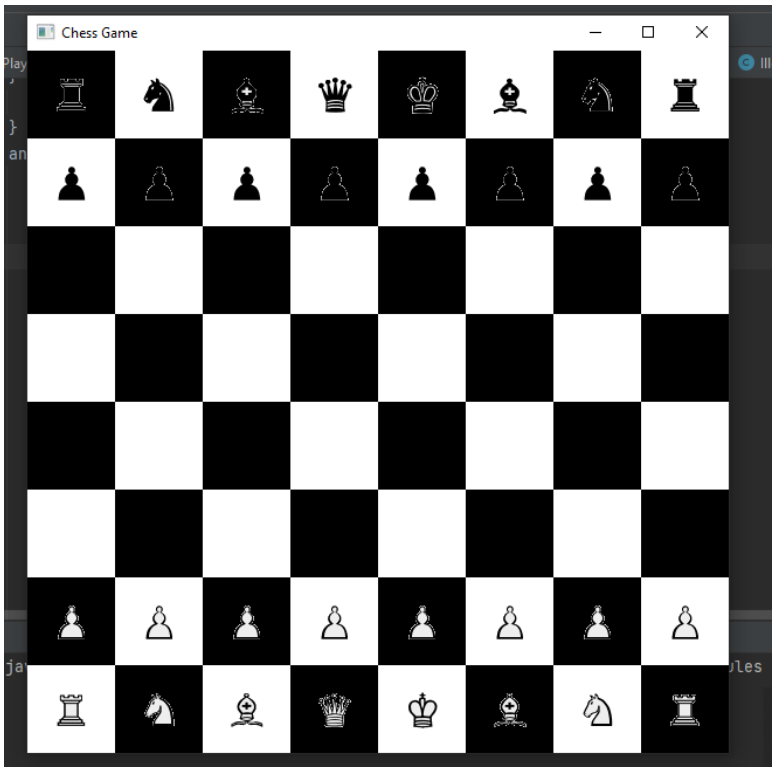
# 4. Results and snapshots:

- Initially a form is displayed that asks for the user details of the player
- The players can't be both of computer type and an error message is displayed
- The text fields can't be blank
- After entering appropriate details the chess board loads
- Illegal moves are not allowed.
- If a player kills the king of the other side then the board is cleared and a final scene is created.
- In the final scene the winner is declared
- There are 2 buttons analyze and exit
- When the analyze button is pressed the frequency of the matches won by the player is displayed and stored in a text file
- The exit button closes the game

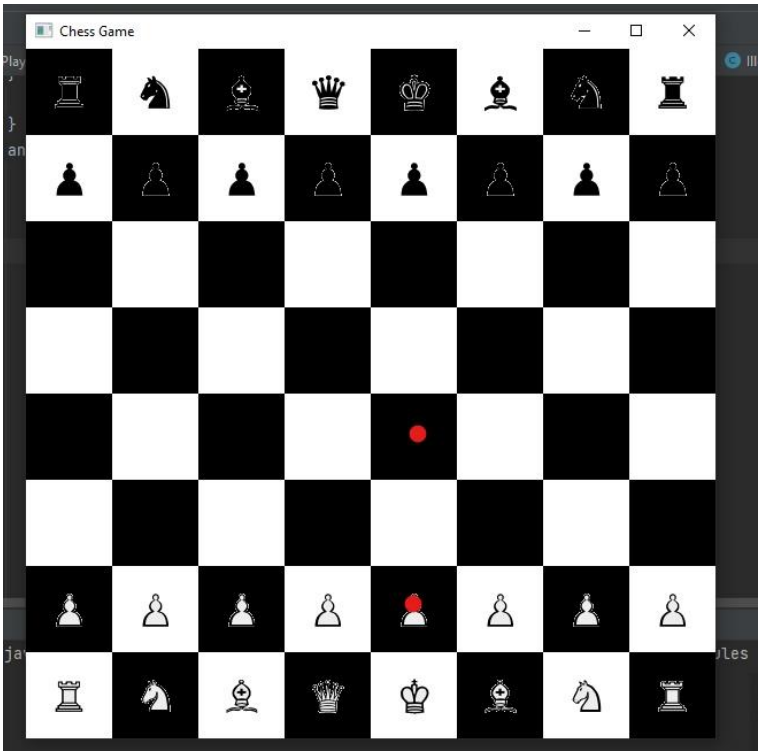
a. Starting the game:



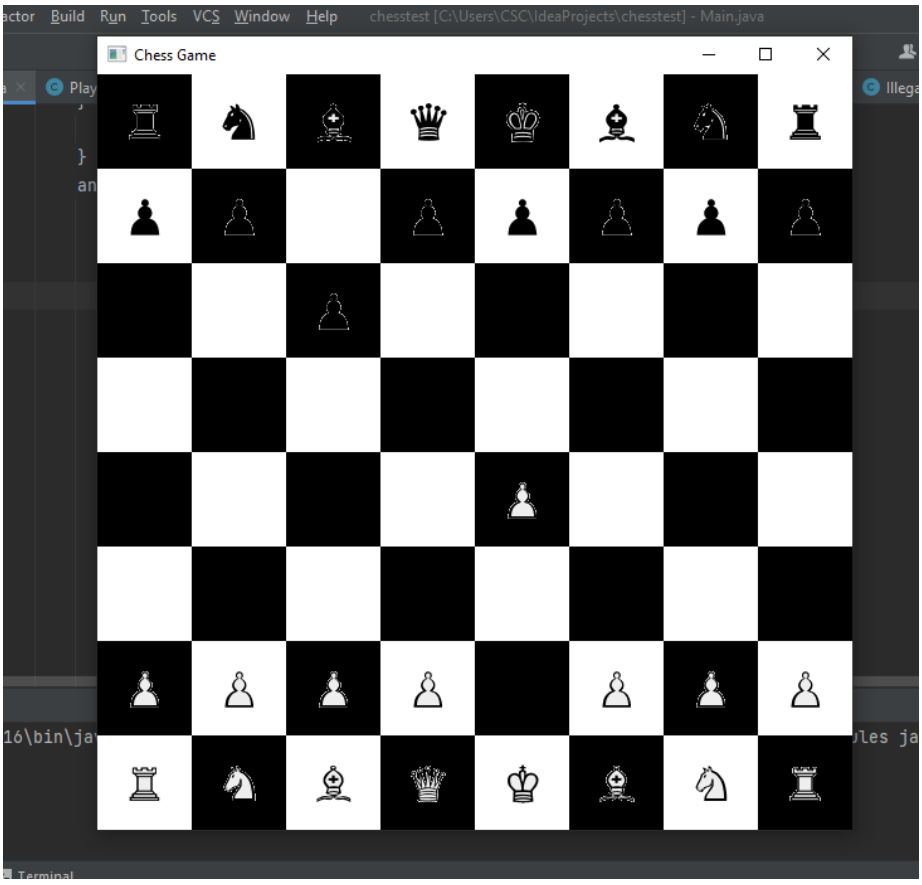

b. After submission the chess board loads



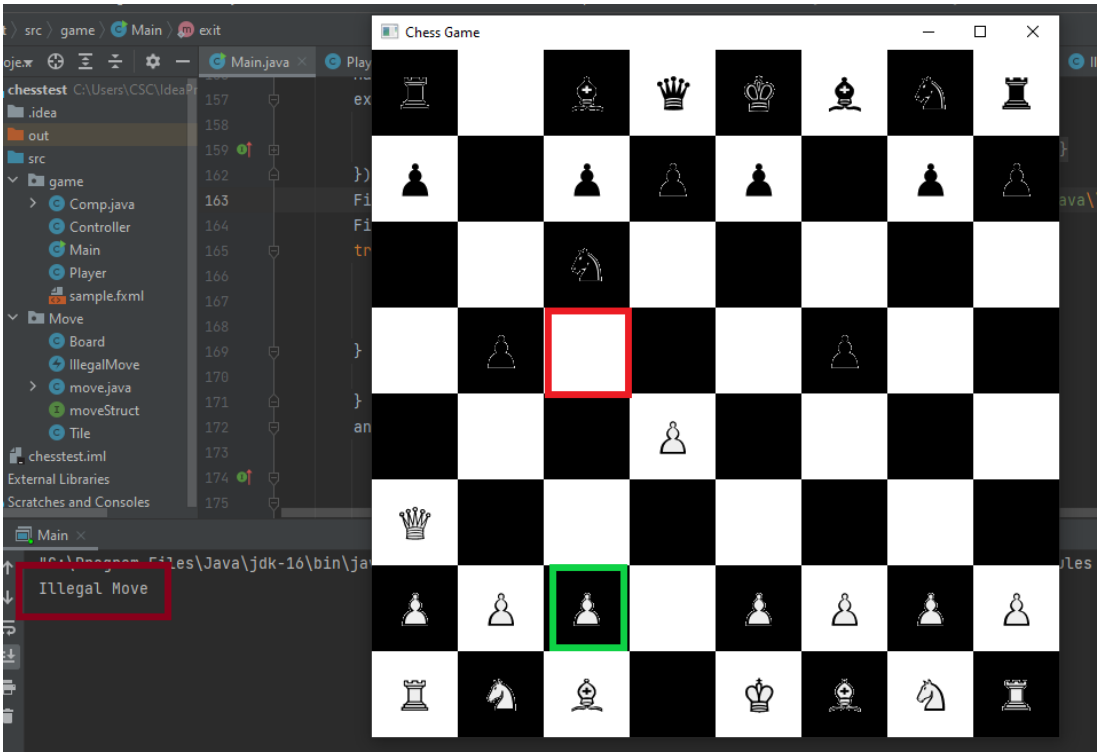
c. First move



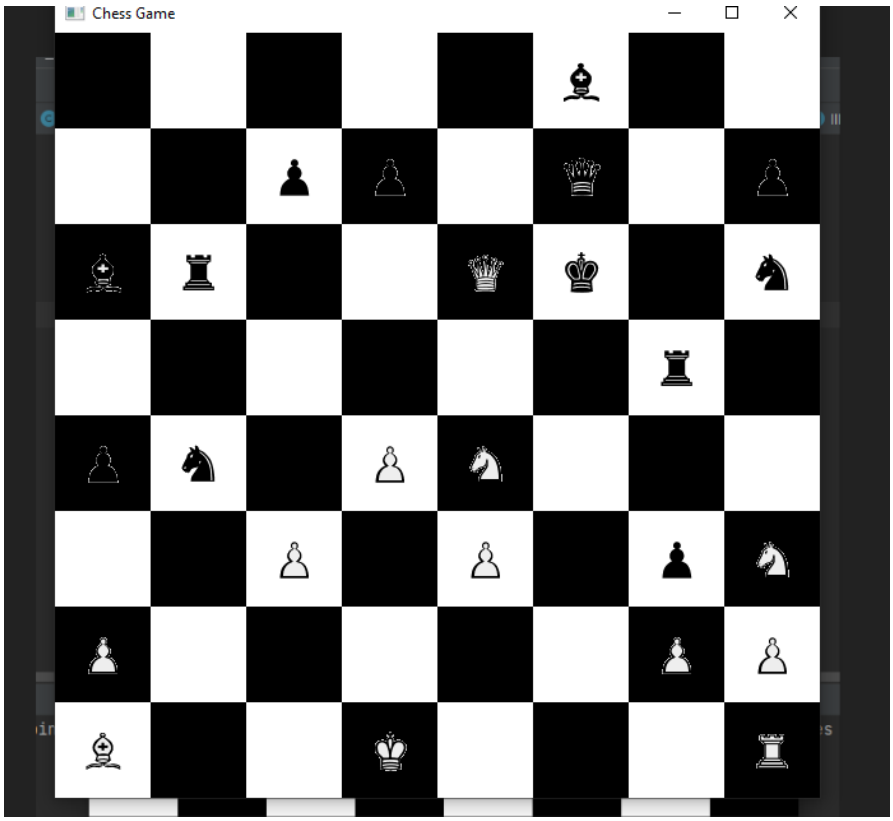

d. The black player makes the move automatically



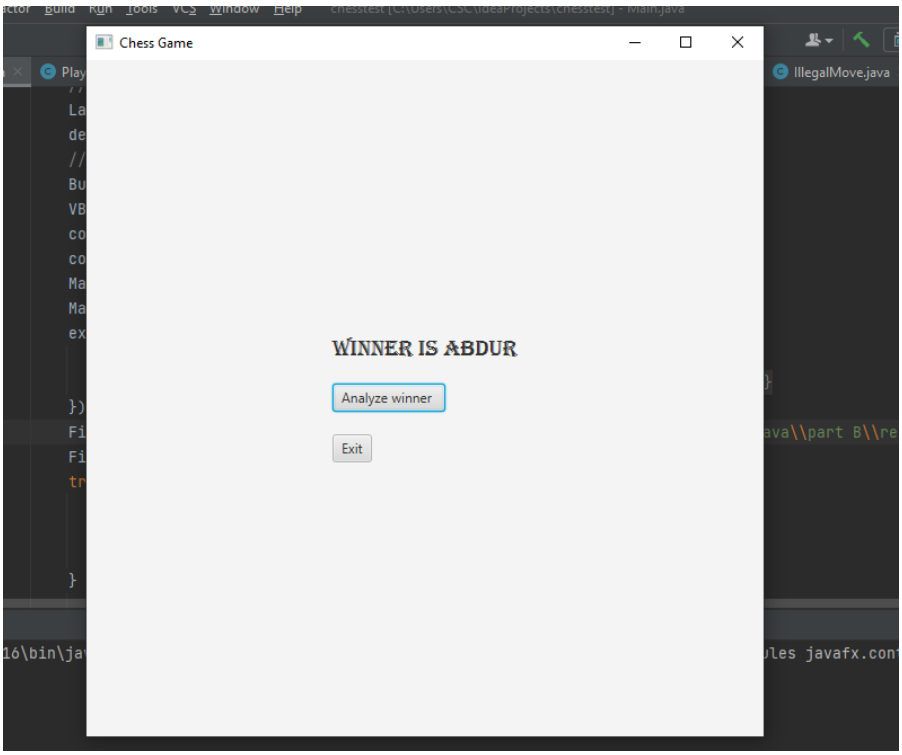
e. Illegal move is made



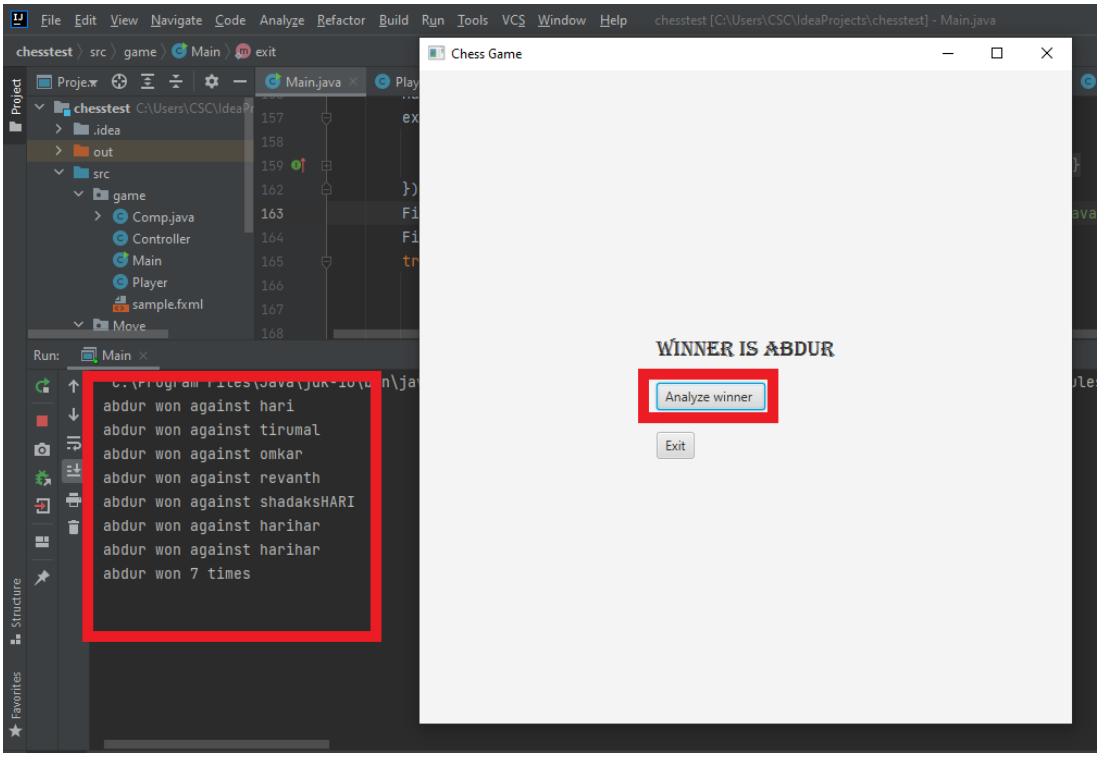

f. Middle of the game



g. After killing the king




h. Analyze winner is pressed



i. The result file is updated with the result of the recent game


j. The game closes after exit button is pressed

# 4. Conclusion:

- The power of OOP concepts along with JavaFx is demonstrated in this project.
- Object oriented paradigm has a wide variety of concepts, some of which are demonstrated, from inheritance to exception handling to lambda expressions.
- JavaFx provides the programmer with a wide range of choices including panes, buttons and labels.
- The oop concepts used in this project are : Inheritance, interface, multithreading, lambda expressions, exception handling, collection framework and regular expressions.
- The JavaFx features utilized are: Stages, scenes, labels, radio buttons, text fields, buttons, grid pane, stack pane, image view, event handlers, Vbox and Hbox.
- Although the scope of the current project is very limited it can easily expanded as per the requirements.

# 5. References/resources:


- Class notes
- Java the complete reference by Herbert Schildt


Appendix A: Source code drive link
https://drive.google.com/drive/folders/1WJxES_upTktHFQFGgtKP5XucRgskpTNX?usp=sharing