# CHESS GAME DESIGN ASSIGNMENT

APRIL 18, 2022

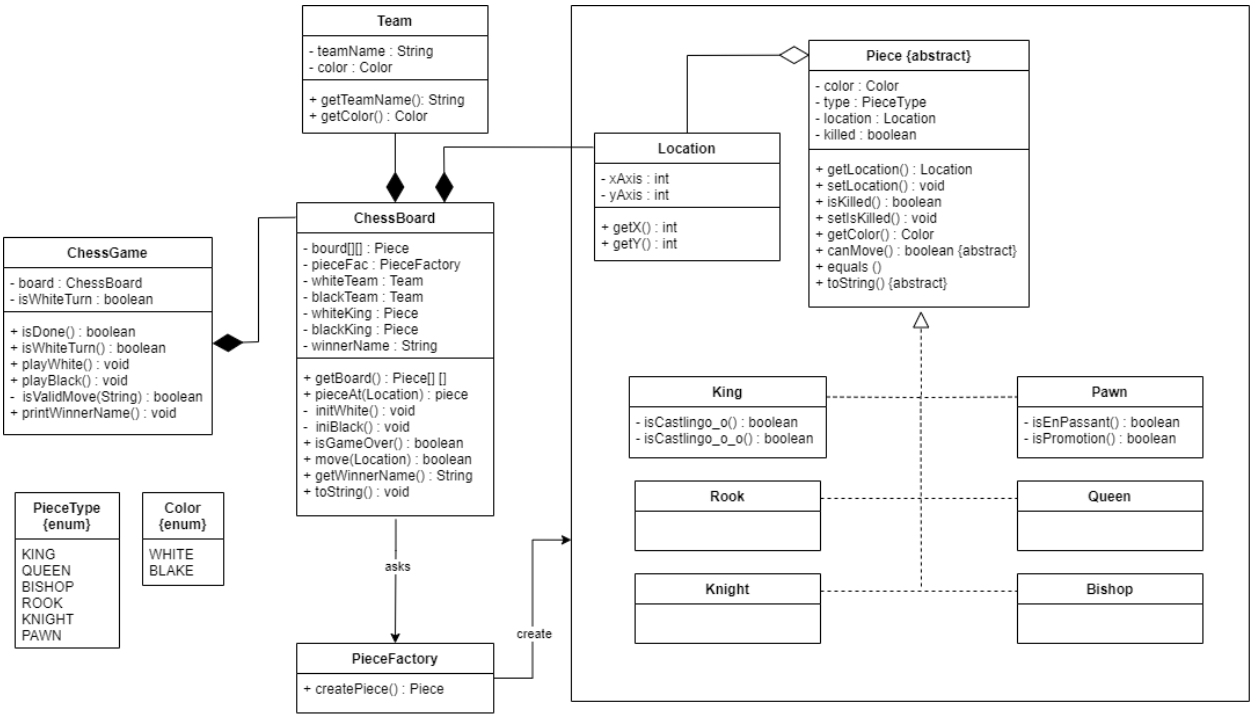
ABDALRHMAN ALI ALJAMMAL
ATYPON

# Contents

## Abstract

In this assignment we supposed to design a chess game, show the implementation of all classes, and describe each of them, and in this report, I will show the design, implementation, and I will defend my code against clean code, effective java, SOLID principles.

## Design (UML diagram)



## Implementation

Each chess game has a lot of elements like chess board, two players or teams, and pieces.

Chess Game class: in this class we have two elements a Board and a Boolean variable called isWhiteTurn, also the class have a lot of methods to check player turn, the game state, player move and print the winner's name when the game is done.

Chess Board class: in this class we have a two-dimensional array represents the chess board, two players or teams (black team and white team), PieceFactory to create pieces by its type and a lot of methods like:

pieceAt(Location): which return the Piece located at the given Location.

initWhite,Black(): which create pieces and locate them in their initial locations.

Move(currentLocation, newLocation): to move the piece located at currentLoc to the newLoc if the piece can move to the newLoc.

toString(): which return a board include all pieces as a string.

a method to check if the game is over when any king is killed and a method to return the winner team name.

Piece Factory Class: I decide to use the factory method design pattern in my implementation to create pieces in the board class, and the reason of that to hide creation logic.

Team class: in this class we have a team name and a team color which represents the color of the pieces in this team.

Piece class: to implement this class I decide to make it as an abstract class and super class for all pieces (king, queen, etc..), and the reason for that is to make the code expandable by ability to add new piece in the future.

This class have a color attribute which represent the color and the team of the piece, type which represent the Piece Type (King, Queen, etc..), and a Location attribute represent where the piece located in the board, and a Boolean variable called is Killed, also the class have a getters and setters for these attributes and have two abstract methods:

canMove(Board, currentLoc, newLoc): which check if the piece can move to the new location and check if there is a special move like en-passant, promotion or castling. For simplify the code the method will always return true.

toString(): which return the Unicode for each piece in string to print it in the board.

Location class: a class to represent the x and the y axis of the piece in the board as a location.

## Defend against clean code principals

My code follows the most of clean code principles like :

- It has no long comments
- It avoids code duplication as possible (DRY)
- Has clear and meaningful naming for variables, methods, and classes
- Using factory method instead of using constructor to create pieces objects
- Avoids returning null
- Has short methods and each method has a single responsibly
- Has no long classes
- It avoids data clumps by using Location class
- Methods has 0-2 parameters
- Avoids primitive obsession
- Follows just-in-time design rule and avoids over-engineering

In other hand it violates some principles like:

- Has a data classes like Location and Team class because there is no functionality can add to it
- Board class is a little bit long class, and it has some of coupling

## Defend against Effective java items

There are some items that my code follows from Effective java book:

- **Consider static factory methods instead of constructors**: use factory method to create and initialize Pieces in the board
- **Always override toString()**
- **Minimize the accessibility of classes and members**: make variables and methods private to encapsulate them
- **In public classes, use accessor methods, not public fields**
- **Use Enums instead of int constants:** use two Enums (Color and Piece Type)
- **Design method signatures carefully:** use clear and meaningful names for the methods and have a short list of parameters
- **Minimize the scope of local variables:** declare variables where its first use
- **Avoid strings where other types are more appropriate:** create objects like Location instead of String to move a Pieces in the board

## Defend against SOLID principals

1. Single Responsibility Principle: I implement my code to follow this principle by make each method and class to do or be responsible to only one thing like chess game class is responsible to check the game state, switch players turn and print the winner name, chess board class is responsible to initialize teams and move pieces, and each method in these classes has only one thing to do.

2. Open Closed Principle: some of my code follows “closed” like Chess Board and Chess Game methods which is fully implemented and tested, in other hand I have some of “open” methods like Piece class methods which can be extendable and can implement a new functionality in it.

3. Liskuv Substitution Principle: my classes follow this principle because all subclasses have the same behavior of the super classes without any changes like the King, Queen and all piece classes shared the same behavior of the Piece class by implementing the canMove() method.

4. Interface Segregation Principle: the piece class follows this principle because its not force any of the subclasses to implement unnecessary methods instead of that its implement them as non-abstract methods.

5. Dependency Inversion Principle: I think my code violate this principle; it has some of coupling in the Chess Board class because it depends on concretion not on abstraction.