Designing a chess Game

board → 8×8

→ each player will have 16 peices. → a king, queen, two rooks, two knights, two bishop & eight pawn.

→ If no way king can move then its checkmate

→ Each peice has its own movement.

→ Draw if not enough peices on board, same position Repeated thrice then its draw.

Expectation from the interviewee

- How many chess peices are their in the game?
- what are the different chess peices, & what are their respective moves?
- which chess peice is the weakest in chess?
- which peice is strongest in chess?
- which player takes the first turn?
- what are the rules of the game?
- what is a checkmate?
- How does a stalemate happen
- can a player forfeit / resign from the game?

Requirement Collection

R1: The purpose of this system is to enable multiple in a game of chess via an online platform.

R2: The game will be played according to the official rules of a international chess game.

R3: Each player is randomly assigned the color - either black or white

R4: At the start of the game, each player will have eight pawns, two rooks, two bishop, 2 knights, one queen & one king on the board

R5: The player with the white pieces will make the first move

R6: It is not possible for a player to retract or undo their move once it has been made.

R7: The system will keep a record of all moves made by both players.

R8: The game may end in a checkmate, forfeiture, stalemate (a draw), or resignation.

Piece	Rules
King	• It can move one step in any direction. • It cannot move to a box that might cause a check.
Queen	• It can move horizontally, vertically, or diagonally unless it is blocked by a piece of the opponent. • It cannot jump over the opponent's pieces.
Pawn	• It can move one box forward. • It is allowed to move two boxes forward if it is the first move by the player. • It can move one box diagonally to kill the opponent's pawn.
Bishop	• It can move only diagonally in any direction unless it is blocked by a piece of the opponent. • It cannot jump over the opponent's piece.
Rook	• It can move only horizontally or vertically unless it is blocked by a piece of the opponent. • It cannot jump over the opponent's piece.
Knight	• It can only move in an L-shape position by jumping two boxes horizontally or one box vertically. • It can jump over other pieces.

Rules for pieces

Situation	Rules
Checkmate	• This is when a player's king is in check (can be captured by the opponent's pieces) and there is no possible escape for the piece. • This situation decides the winner of the game.
Stalemate	• This is when a situation in which the player's king is not in check and no other move is possible. • It draws the match.
Forfeiture	• If a player does not show up for the game, then the player is considered to have forfeited.
Resignation	• If a player is at a position in the game where they understand that the stronger opponent will win in case of any move and decides to quit, then they have resigned from the game.
Castling	• A player moves their king two boxes towards the rook on the same row. • The rook is moved to the box the king passed over, which is next to the new position of the king. • The king and rook should be at their original positions and should not have been moved before. • No other piece should be between the king and the rook.

Rules for situation

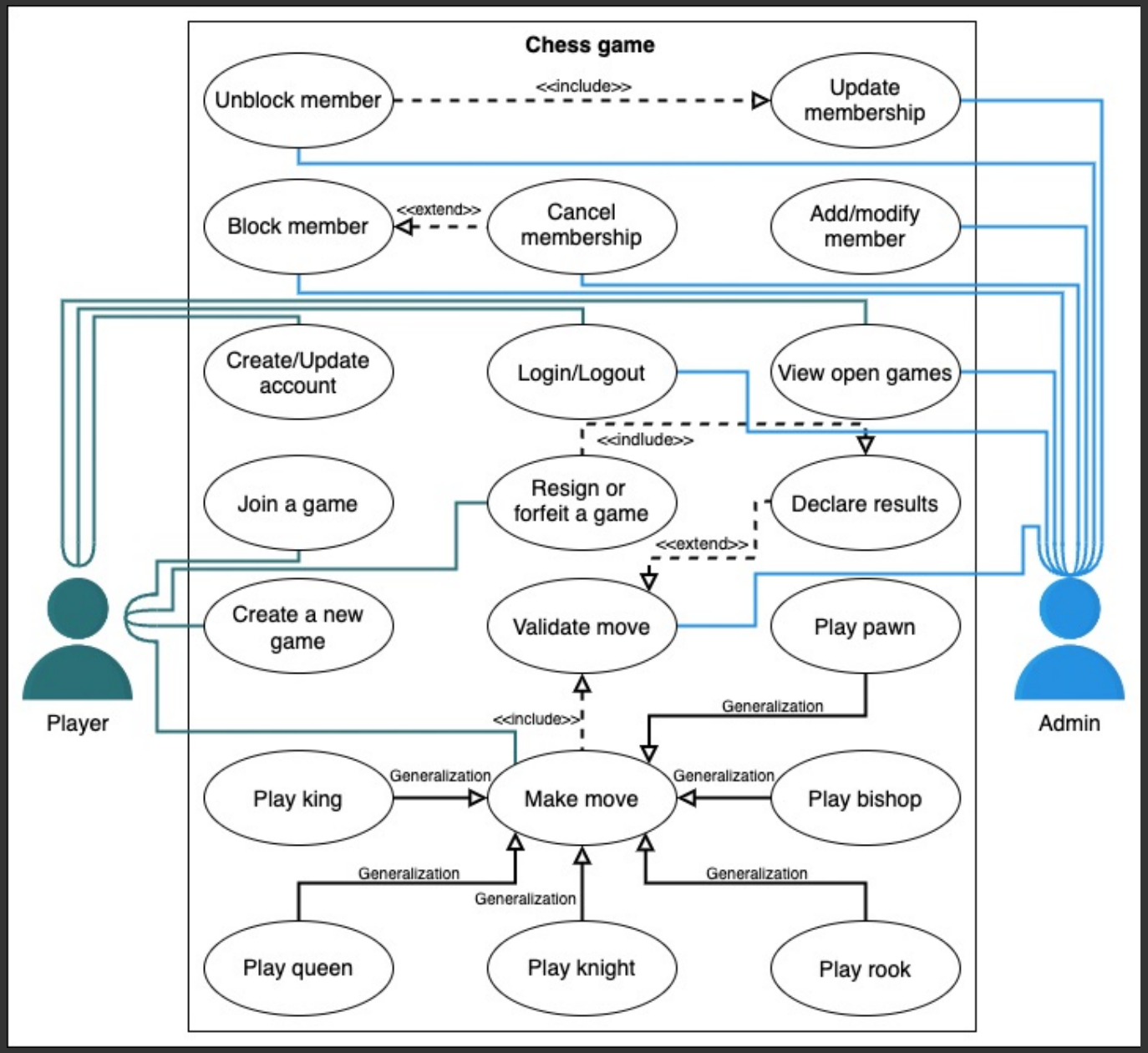
System → "Chess Game"

Primary Actor : Player : Primary actor for playing game.

Secondary Actor : Admin : add, remove or update a player account & membership. view open games & validate player move.

Player	Admin
Create/Update account	Block/unblock member
Join a game	Cancel/Update membership
Create a new game	Add/modify member
Make move	Login/Logout
Resign or forfeit a game	Validate Moves
View open games	View open games
Login/Logout	Declare results

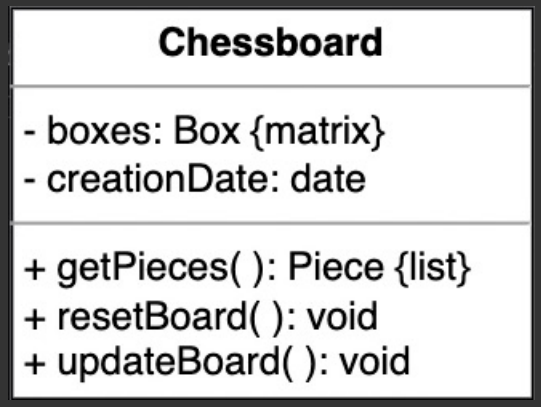
Use case Diagram



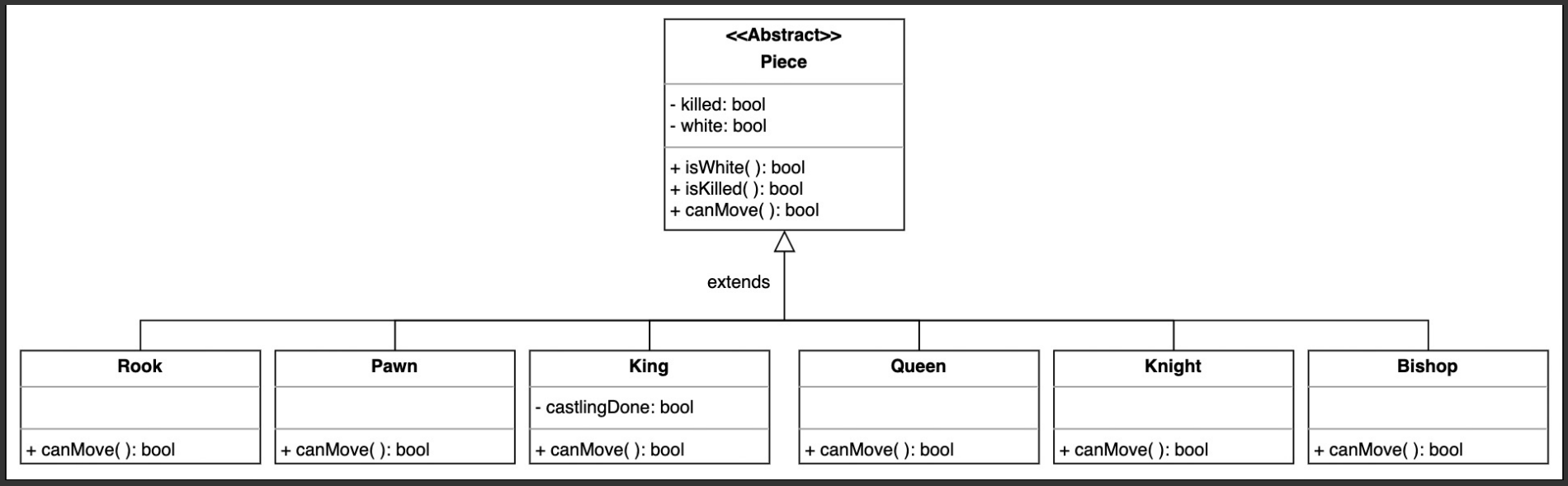
Class Diagram for Chess Game

1. Box

2. Chessboard



3. Piece



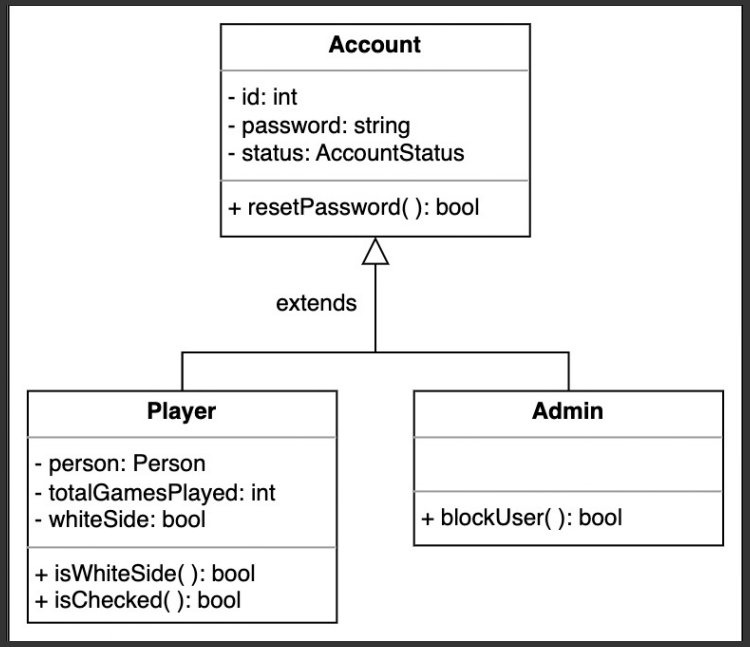
4. Move

Move

- startBox: Box
- endBox: Box
- pieceKilled: Piece
- pieceMoved: Piece
- player: Player
- castlingMove: bool

+ isCastlingMove( ): bool

5. Account



6. Chess Move Controller

ChessMoveController

+ validateMove( ): bool

7. Chess Game Move

ChessGameView

+ playMove( ): void

8. Chess Game

ChessGame

- players: Player {list}
- board: Chessboard
- currentTurn: Player
- status: GameStatus
- movesPlayed: Move {list}

+ isOver( ): bool
+ playerMove( ): bool
+ makeMove( ): bool

9. Enumeration & Custom Data types

<<enumeration>>
GameStatus

Active
BlackWin
WhiteWin
Forfeit
Stalemate
Resignation

<<enumeration>>
AccountStatus

Active
Closed
Canceled
Blacklisted
None

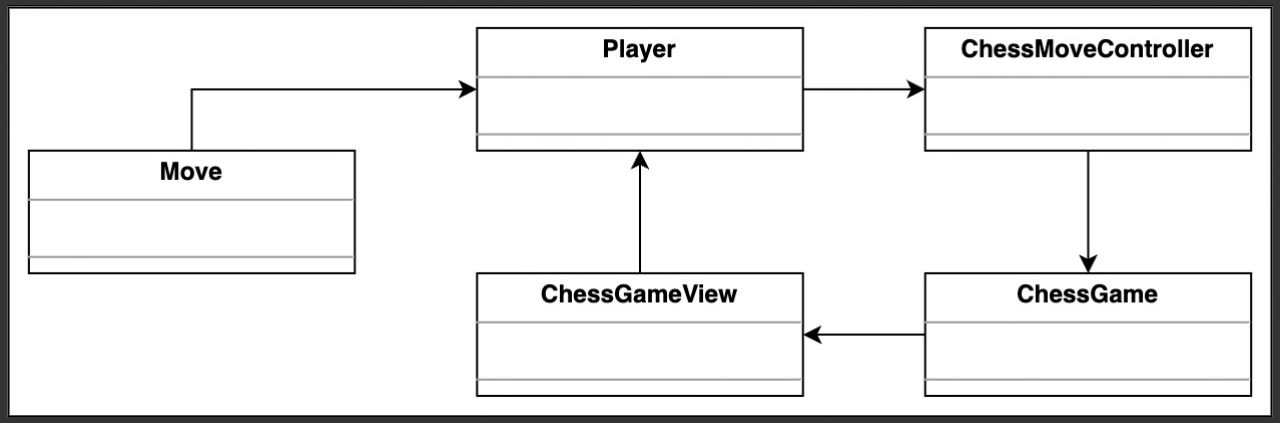
10. Person

Person

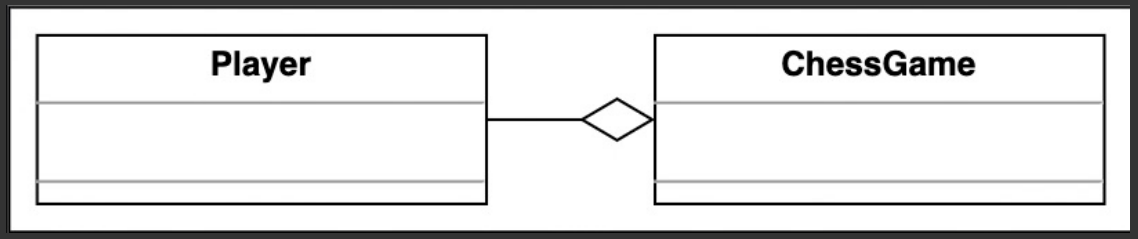
- name : string
- streetAddress : string
- city : string
- state : string
- zipcode : int
- country : string

Relationship between classes

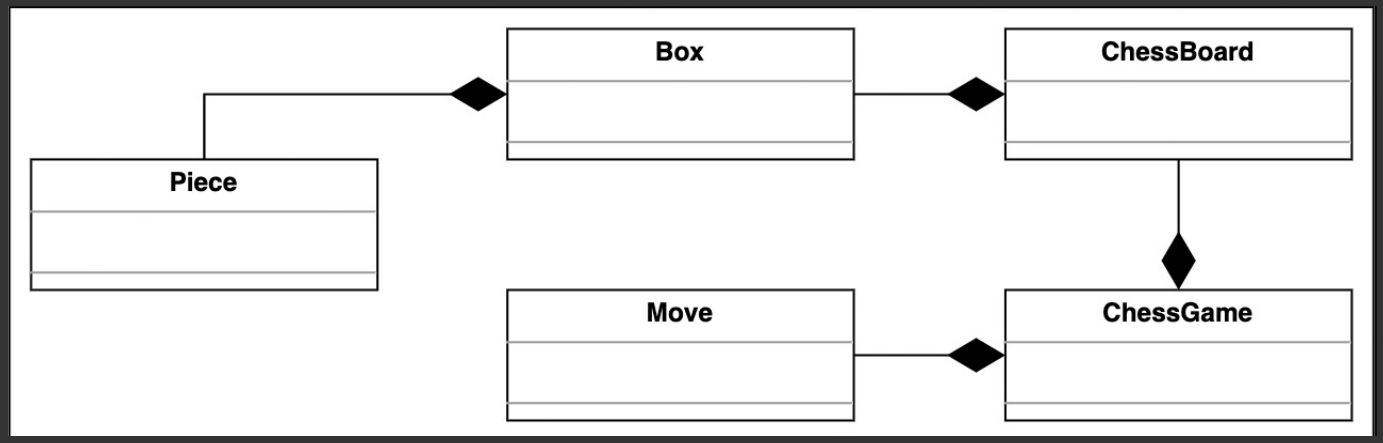
1. Association



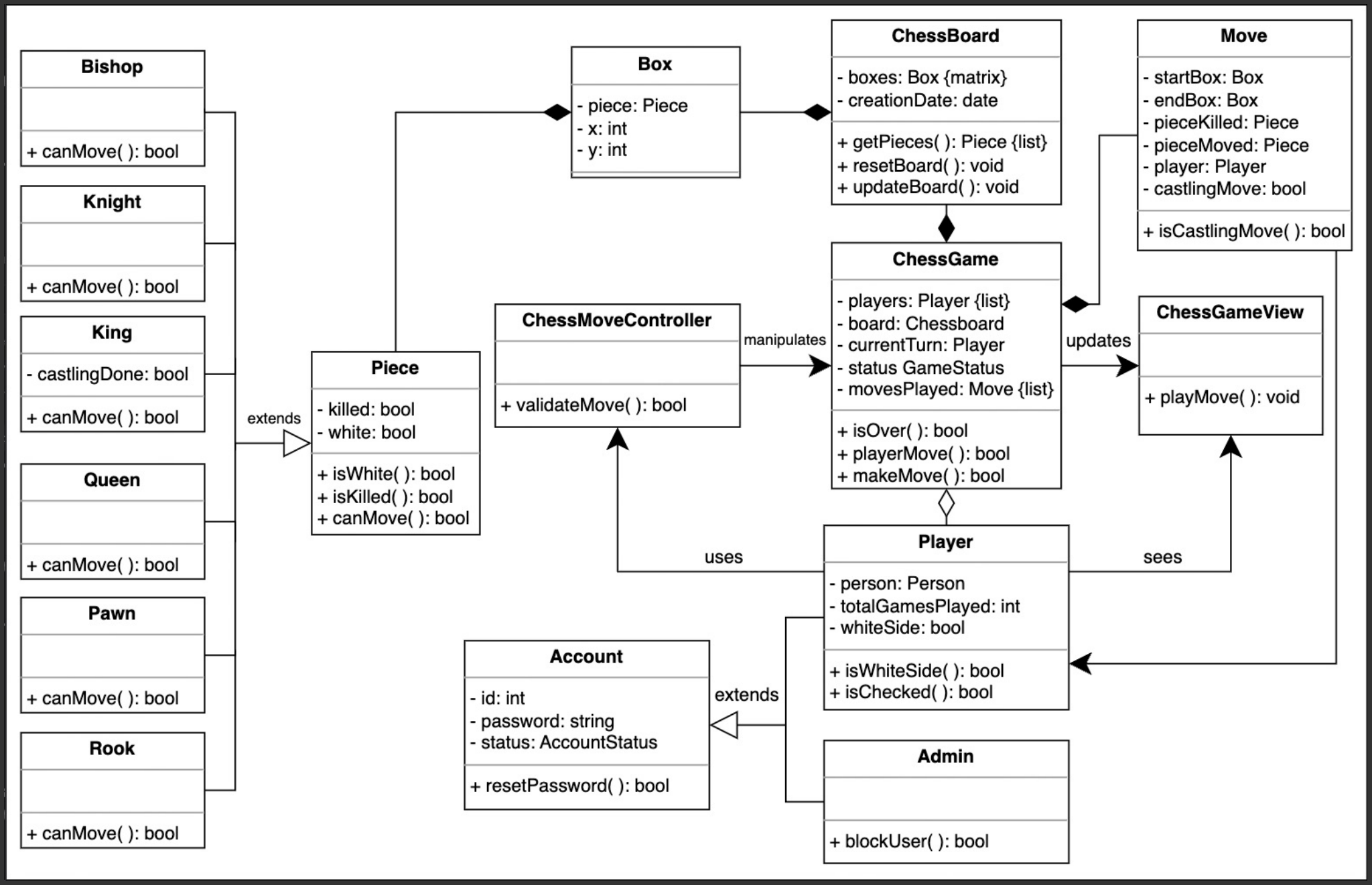
2. Aggregation



3. composition

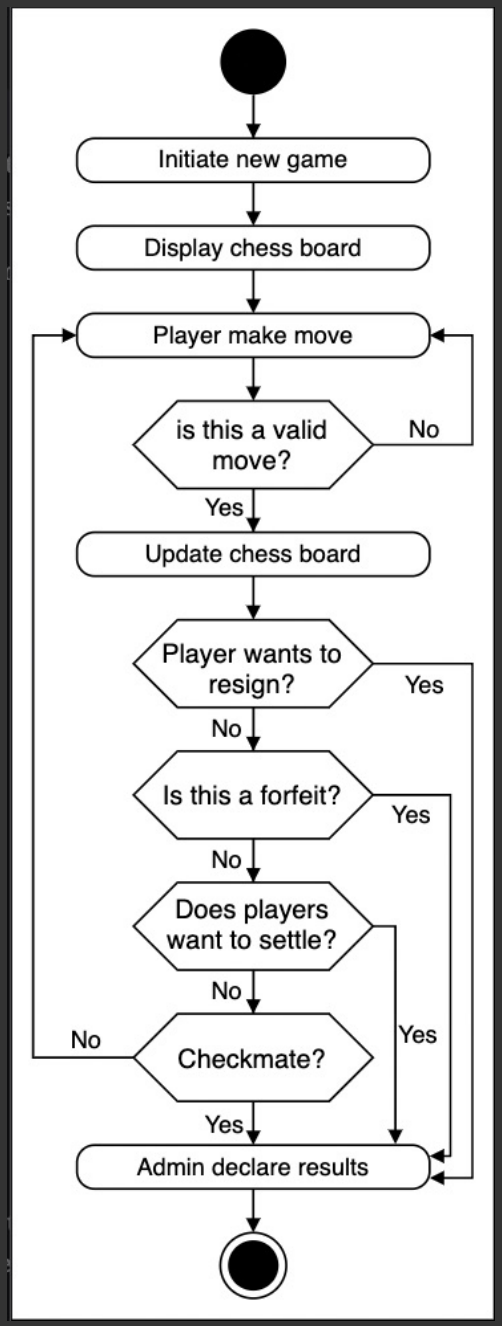


Class Diagram of Chess Game



→ Iterator design pattern can be used, all to use player piece without knowing underlying logic

Activity Diagram



Code for Chess Game

1. Enumeration & Custom Datatypes

```
// Enumerations
enum GameStatus {
  Active,
  BlackWin,
  WhiteWin,
  Forfeit,
  Stalemate,
  Resignation
}

enum AccountStatus {
  ACTIVE,
  CLOSED,
  CANCELED,
  BLACKLISTED,
  NONE
}

// Custom Person data type class
public class Person {
  private String name;
  private String streetAddress;
  private String city;
  private String state;
  private int zipCode;
  private String country;
}
```

2. Box & Chessboard

```
public class Box {
  private Piece piece;
  private int x;
  private int y;
}

public class Chessboard {
  private Box[][] boxes;
  private Date creationDate;

  public List<Piece> getPieces()
  public void resetBoard()
  public void updateBoard()
}
```

3. Peice

```
public abstract class Piece {
  private boolean killed = false;
  private boolean white = false;

  public boolean isWhite();
  public boolean isKilled();
  public abstract boolean canMove(Chessboard board, Box start, Box end);
}

public class King extends Piece {
  private boolean castlingDone = false;

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}

public class Queen extends Piece {

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}

public class Knight extends Piece {

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}

public class Bishop extends Piece {

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}

public class Rook extends Piece {

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}
```

```
public class Pawn extends Piece {

  @Override
  public boolean canMove(Board board, Box start, Box end) {
    // definition
  }
}
```

4. Move

```
public class Move {
  private Box start;
  private Box end;
  private Piece pieceKilled;
  private Piece pieceMoved;
  private Player player;
  private boolean castlingMove = false;

  public boolean isCastlingMove();
}
```

5. Account, player & Admin

```
public class Account {
  private int id;
  private String password;
  private AccountStatus status;

  public boolean resetPassword();
}

public class Player extends Account {
  private Person person;
  private boolean whiteSide = false;
  private int totalGamesPlayed;

  public boolean isWhiteSide();
  public boolean isChecked();
}

public Admin extends Account {
  public boolean blockUser();
}
```

6. Chess Move Controller

```
public class ChessMoveController {
  public boolean validateMove();
}

public class ChessGameView {
  public void playMove();
}
```

7. Chess Game

```
public class ChessGame {
  private Player[] players;
  private Chessboard board;
  private Player currentTurn;
  private GameStatus status;
  private List<Move> movesPlayed;

  public boolean isOver();
  public boolean playerMove(Player player, int startX, int startY, int
endX, int endY) {
    /* 1. start box
       2. end box
       3. move
       4. call makeMove() method
    */
  }
  private boolean makeMove(Move move, Player player) {
    /* 1. Validation of source piece
       2. Check whether or not the color ofthe piece is white
       3. Check if it is a valid move or not
       4. Check whether it is a castling move or not
       5. Store the move
    */
  }
}
```