Class Diagram for the Chess Game

Learn to create a class diagram for the chess game using the bottom-up approach.

We'll cover the following

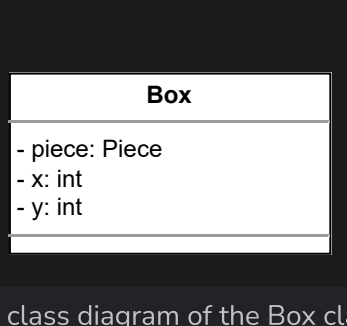
We'll create the class diagram for the chess game. In the class diagram, we will first design the classes and then identify the relationship between classes according to the requirements for the chess game problem.

Components of Chess

As mentioned earlier, we'll follow the bottom-up approach to designing a class diagram for the chess game.

Box

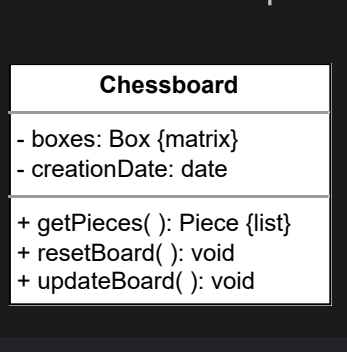
A `Box` is a specific position/block on the 8x8 chessboard which is defined by row x and column y, respectively.



The class diagram of the Box class

Chessboard

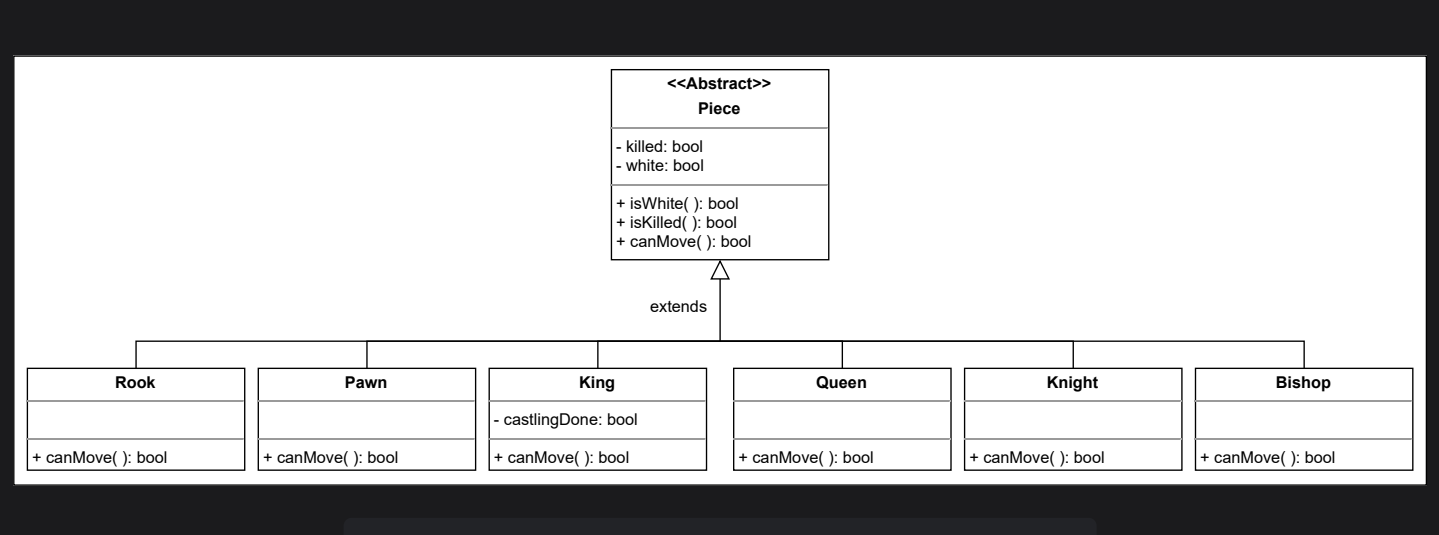
A `Chessboard` is the 8x8 board that stores all the current game's active pieces. It is identified by the date of its creation and can be updated or reset.



The class diagram of the Chessboard class

Piece

A chess `Piece` can only be black or white in color. It might be alive or killed depending on the moves made by the opposition. There can be six types of pieces (`Rook`, `Pawn`, `King`, `Queen`, `Knight`, and `Bishop` derived from `Piece`) that have their respective moves based on the rules of the game which decides whether they are eligible to move or not.



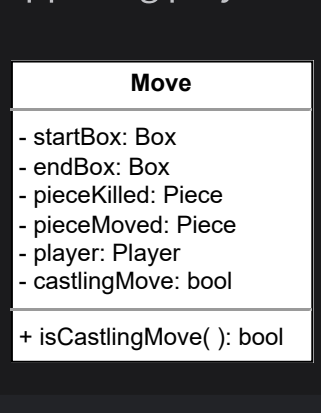
The class diagram of the Piece and its derived classes

R4: Chess Game

R4: At the start of the game, each player will have eight pawns, two rooks, two bishops, two knights, one queen, and one king on the board.

Move

A `Move` is the displacement of a `Piece` from one `Box` to another on the chessboard. It may or may not kill a piece of the opposing player.



The class diagram of Move class

R5 and R6: Chess Game

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

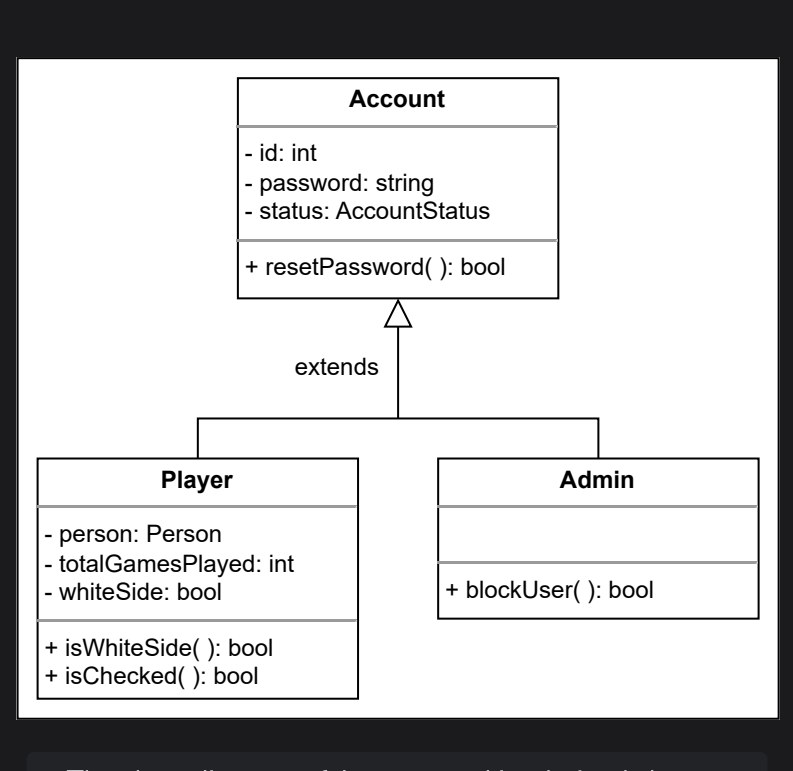
R6: It is not possible for a player to retract or undo their move once it has been made.

Account, Player, and Admin

The `Account` class is a parent class that has two types: `Player` and `Admin`. These classes are derived from the `Account` class. This class stores the user ID, password, and account status.

`Player`: This derived class represents the players of the game and all records of the games played. It also keeps track of whether or not the player's chosen color is white.

`Admin`: This derived class decides whether or not a user account is blocked.



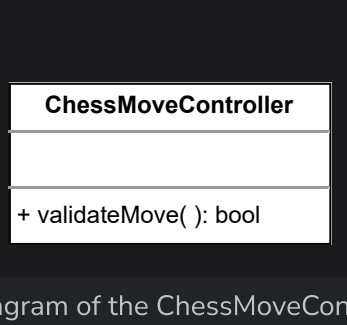
The class diagram of Account and its derived classes

R1: Chess Game

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

Chess move controller

The `ChessMoveController` class validates the moves made by a player and responds accordingly.



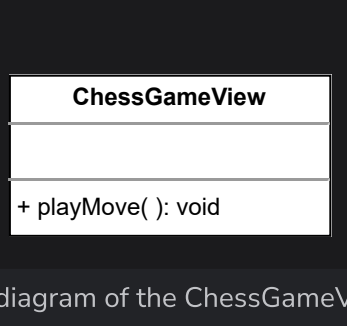
The class diagram of the ChessMoveController class

R2: Chess Game

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

Chess game view

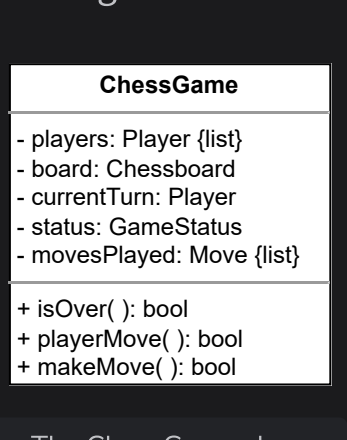
The `ChessGameView` class represents the game view. The `ChessGame` class updates the `ChessGameView` class.



The class diagram of the ChessGameView class

Chess game

The `ChessGame` represents the gameplay of chess. It keeps track of the moves played by both the players, the turns, and the game status.

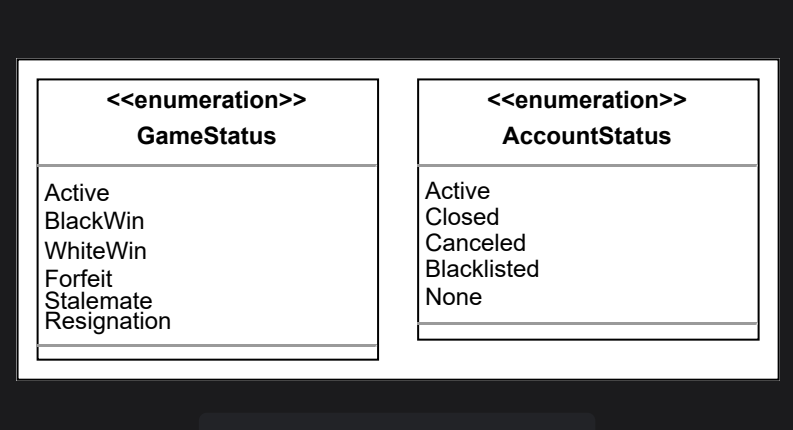


The ChessGame class

Enumerations and custom data types

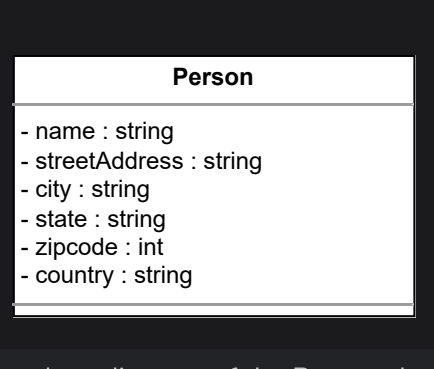
The enumerations and custom data types required in the chess game design problem are listed below:

- `GameStatus`: This enumeration keeps track of the active status of the player and the game, i.e., who wins and whether or not the game is a draw.
- `AccountStatus`: We need to create an enumeration to keep track of the account status, whether it is active, canceled, closed, blocked, or none.



Enums in the chess game

- `Person`: This is used to store information related to a person like a name, street address, country, etc.



The class diagram of the Person class

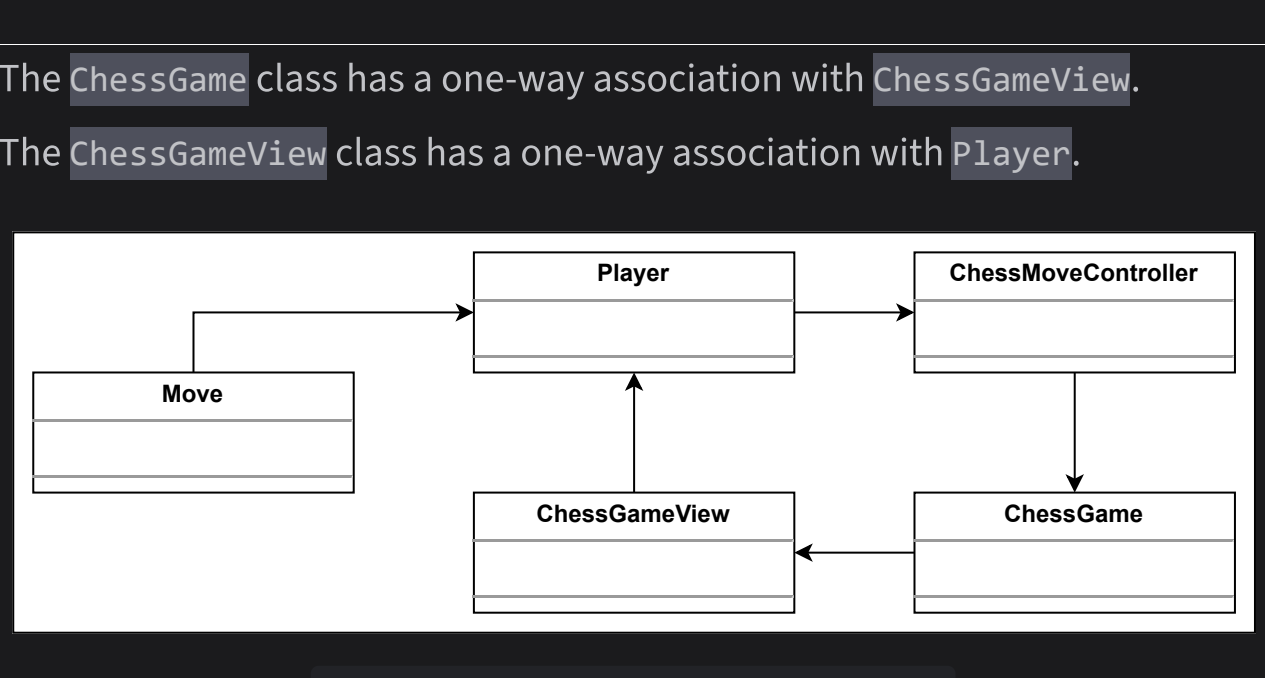
Relationship between the classes

Now, we'll discuss the relationships between the classes we have defined above for our chess game design.

Association

The class diagram has the following association relationships:

- The `Move` class has a one-way association with `Player`.
- The `Player` class has a one-way association with `ChessMoveController`.
- The `ChessMoveController` class has a one-way association with `ChessGame`.
- The `ChessGame` class has a one-way association with `ChessGameView`.
- The `ChessGameView` class has a one-way association with `Player`.

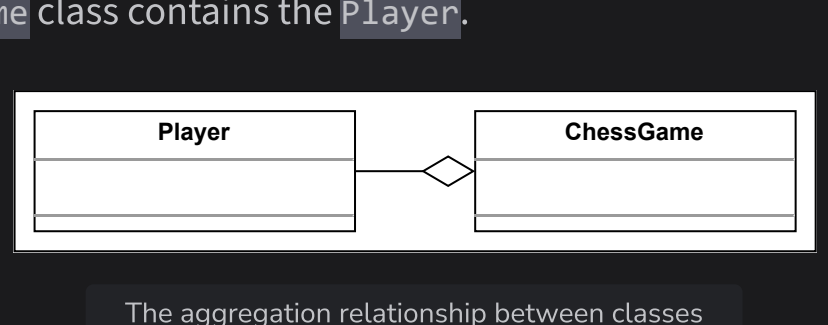


The association relationship between classes

Aggregation

The class diagram has the following aggregation relationships:

- The `ChessGame` class contains the `Player`.

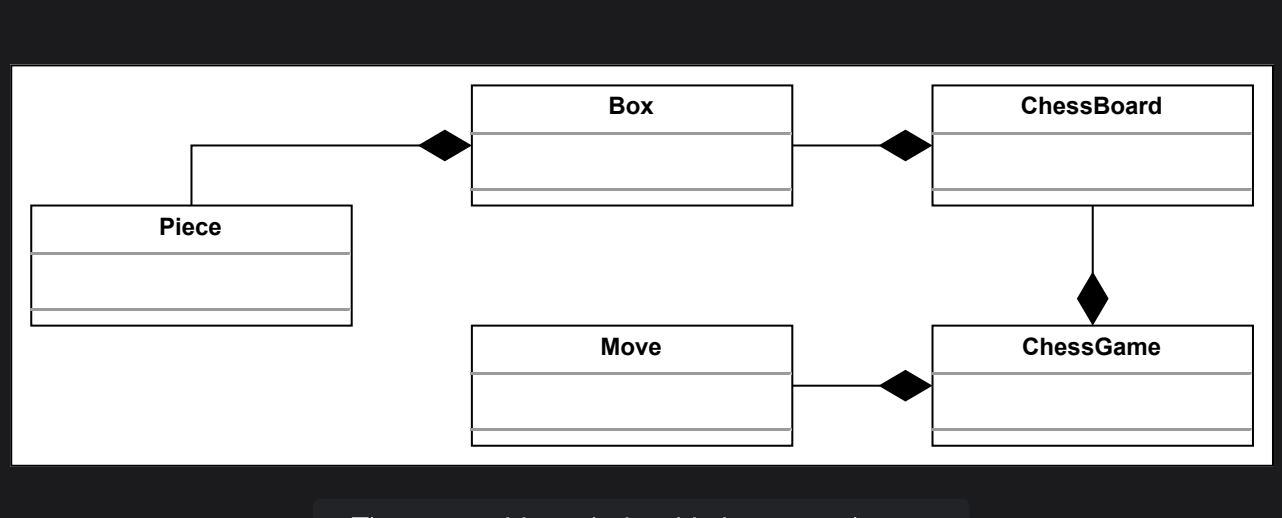


The aggregation relationship between classes

Composition

The class diagram has the following composition relationships.

- The `Box` class is composed of `Piece`.
- The `ChessBoard` class is composed of `Box`.
- The `ChessGame` class is composed of `ChessBoard`.
- The `ChessGame` class is composed of `Move`.



The composition relationship between classes

Inheritance

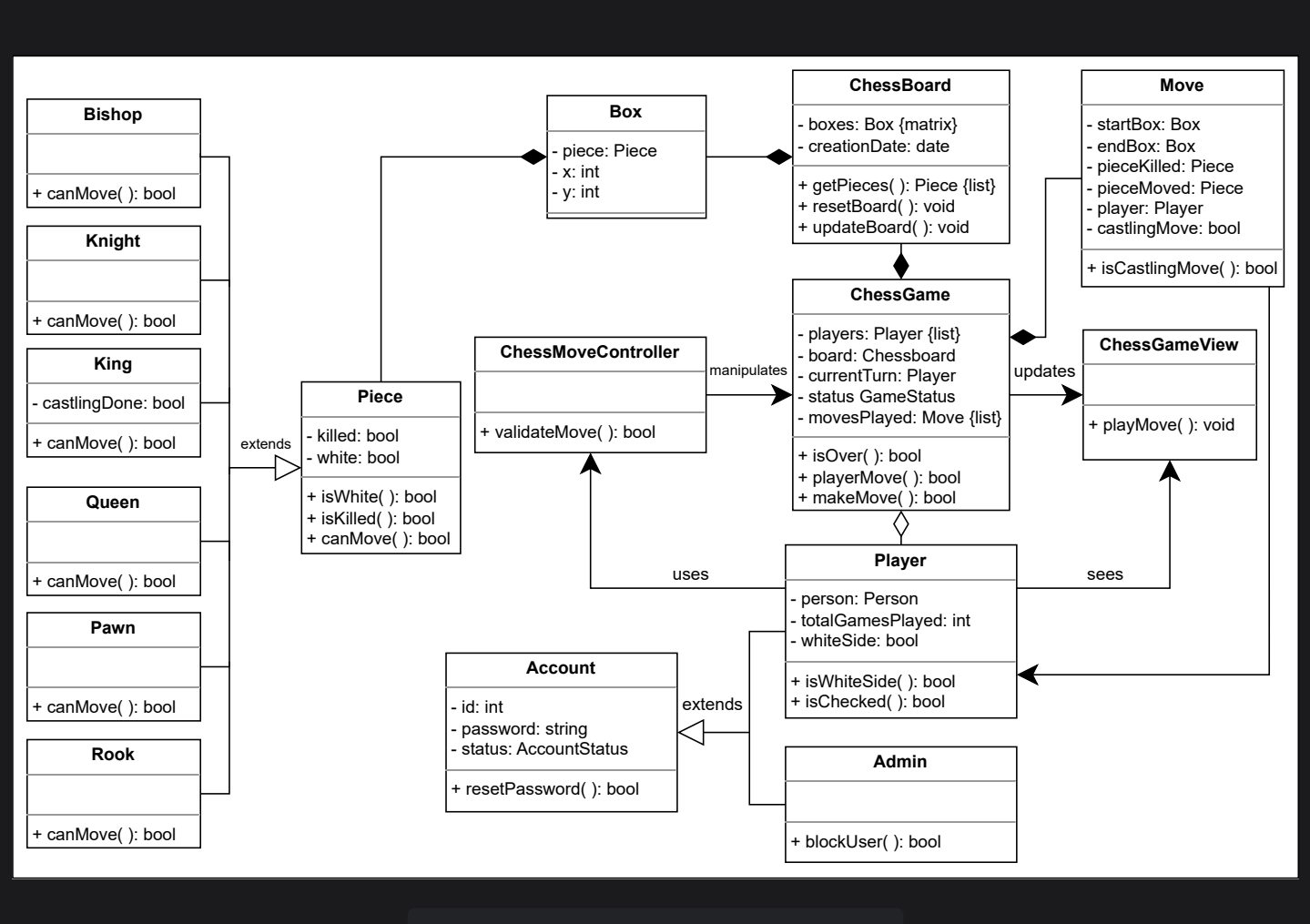
The following classes demonstrate an inheritance relationship:

- Both, `Admin` and `Player` extend the `Account` class.
- The `King`, `Queen`, `Knight`, `Bishop`, `Rook`, and `Pawn` classes extend the `Piece` class.

> **Note**: We have already discussed the inheritance relationship between classes in the component section above one by one.

Class diagram for the Chess game

Here's the complete class diagram for our chess game:



The class diagram of the chess game

Design pattern

The following design patterns have been used in the class diagram:

- **Singleton design pattern:** This pattern ensures the existence of a single instance of the chessboard at a given moment due to the shared nature of the chessboard as a resource. Multiple instances can cause the game state to become inconsistent.
- **Command design pattern:** This pattern is used to encapsulate the move logic for each chess piece. Each chess piece has its own implementation of the move command, which allows it to move according to the rules defined for it. For example, the knight moves in an L-shape pattern, or the rook can move only horizontally or vertically on any number of boxes.

The following design patterns can also be used to design chess:

- The Iterator design pattern would enable the game to move sequentially by allowing the pieces to behave in a uniform manner where the user does not need to know the specifications or underlying logic behind the moves of the pieces.
- The State design pattern ensures the encapsulation of the state logic of each piece, since all the chess pieces have their own respective implementations of checkmate states which makes them behave differently from each other depending on the situation.
- The Observer design pattern enables the chess pieces to act as observers where the chessboard is the subject. As soon as the state of the board changes, the pieces are notified to adapt to the changes accordingly. This decouples the pieces from the chessboard.

AI-powered trainer

At this stage, everything should be clear. If you encounter any confusion or ambiguity, feel free to utilize the interactive AI-enabled widget below to seek clarification. This tool is designed to assist you in strengthening your understanding of the concepts.