1 Einleitung

bli bla blub

2 Klassendiagramme

2.1 Spiel

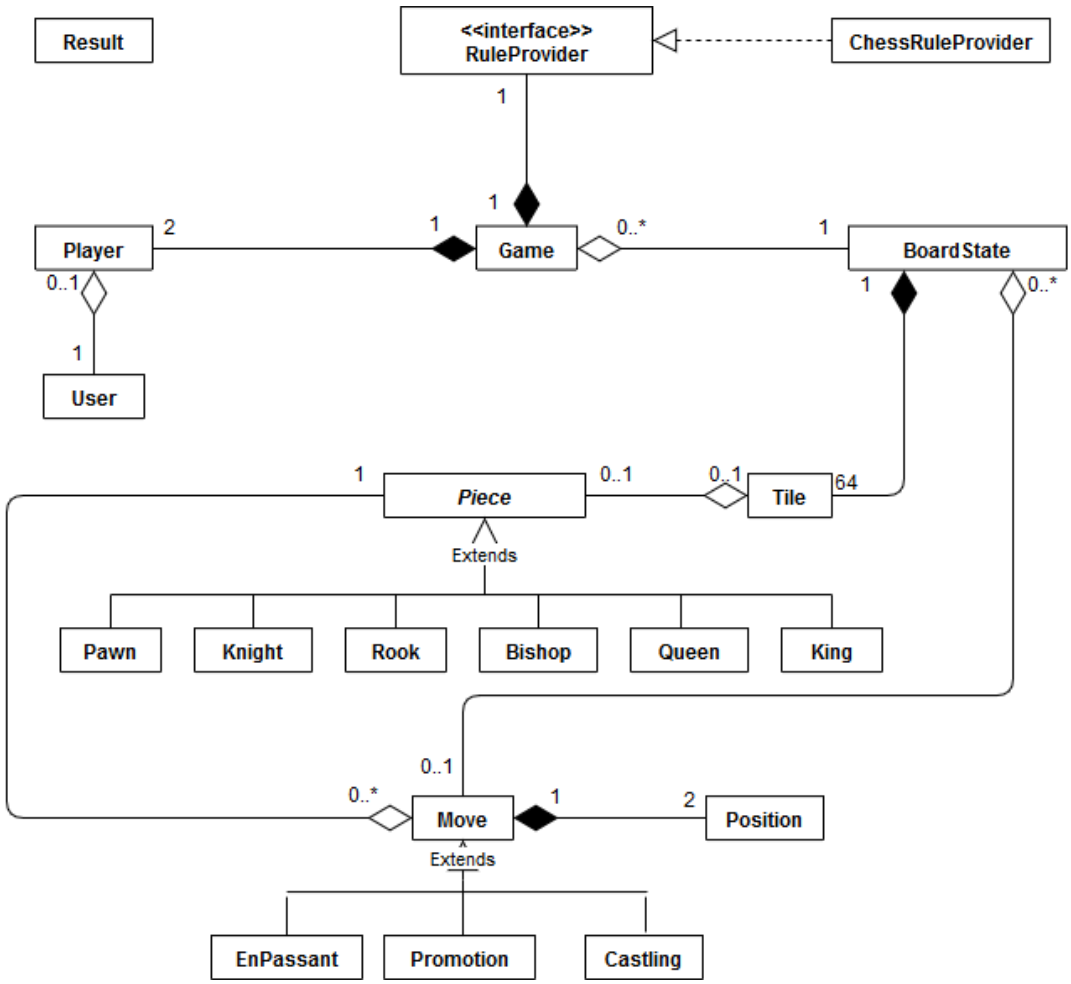

Abbildung 1: TotalGame

-
-

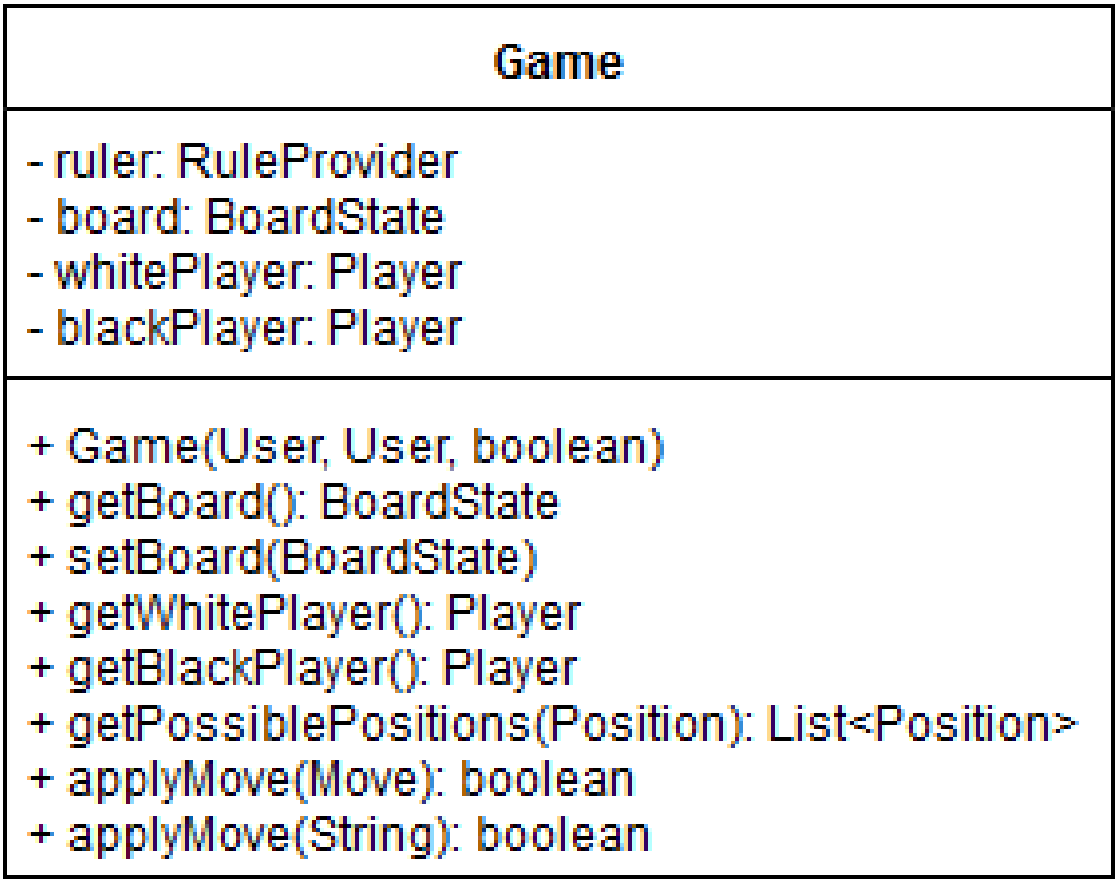

Abbildung 2: Game

- **Game**

  ruler: Objekt, welches die Spielregeln zur Verfügung stellt.

  board: Hier wird der aktuelle Spielstatus gespeichert.

  whitePlayer: Der Spieler mit den weißen Figuren.

  blackPlayer: Der Spieler mit den schwarzen Figuren.

  Game(User, User, boolean Konstruktor, welcher ein Spiel mit den Standard Schachregeln erzeugt. Der boolean sagt, ob der erste User der weiße Spieler sein soll, oder ob die Farbe gewürfelt werden soll.

  getBoard(): Gibt den Brettstatus des akutellen Spiels als Objekt zurück.

getBoard(BoardState): Setzt den Brettstatus neu.

getWhitePlayer(): Gibt den weißen Spieler zurück.

getBlackPlayer(): Gibt den schwarzen Spieler zurück.

getPossiblePositions(): Gibt alle möglichen Positionen als Liste zurück, auf welche eine Figur, welche sich auf der übergebenen Position befindet, ziehen kann. Zum Berechnen dieser wird das ruler Objekt benutzt.

applyMove(Move): Führt einen übergebenen Zug auf dem Brett aus.

applyMove(String):Führt einen als String übergebenen Zug auf dem Brett aus. Dazu muss der String erst in ein Zugobjekt umgewandelt werden.

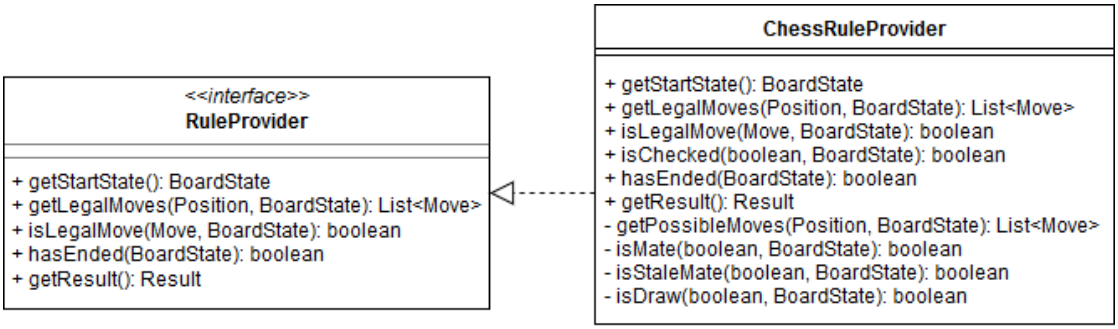

Abbildung 3: RuleProvider

- **RuleProvider**

  Interface, welches alle nötigen Regeln eines Spiels auf einem Schachbrett bereitstellt. Die zu implementierenden Methoden sind:

  getStartState(): Soll die Anfangskonfiguration eines Brettes für das jeweilige Spiel zurückgeben.

  getLegalMoves(Position, BoardState): Soll auf einem Brett ausgehend von einer Position die Zugmöglichkeiten der sich darauf befindenden Figur berechnen, entsprechend der implementierten Regeln.

  isLegalMove(Move, BoardState): Soll überprüfen, ob ein Zug gemäß der jeweiligen Regeln auf einem bestimmten Brett erlaubt ist.

  hasEnded(BoardState): Soll prüfen, ob ein gegebener Zug auf dem gegebenen Brett erlaubt ist.

  getResult(BoardState): Soll prüfen, ob ein Spiel beendet ist, wenn ja soll ein Ergebnis zurückgegeben werden, wenn nicht ein Null-Objekt.

- **ChessRuleProvider**

  Konkreter Regellieferer, welcher die genauen Schachregeln zur Verfügung stellt.

  getStartState(): Gibt die Standard Anfangsstellung eines Schachspiels als BoardState zurück.

  getLegalMoves(Position, BoardState): Gibt eine Liste an erlaubten Zügen ausgehen von einer ausgewählten Position und einem Brett zurück. Dazu werden zunächst mit ***getPossibleMoves(Position, BoardState)*** alle möglichen Züge berechnet. Anschließen wird jeder Zug auf einer Kopie des Brettes

simuliert, und mithilfe von ***isChecked(boolean, BoardState)*** überprüft, ob der selbe Spieler danach im Schach stünde (was den Zug ungültig machen würde).

- ***isChecked(boolean, BoardState):*** Überprüft ob ein Spieler auf dem gegebenen Brett im Schach steht. Ist der übergebene boolean true, wird Weiß überprüft, bei false Schwarz. Dazu wird geschaut, ob es eine gegnerische Figur gibt, welche durch ***getPosssibleMoves(Position, BoardState)*** einen Zug erhält, mit welchem der gegnerische König erreicht werden könnte.

- ***hasEnded(BoardState():*** Prüft, ob das Schachspiel zu Ende ist. Dazu wird geprüft, ob der zu ziehende Spieler laut ***isMate(boolean, BoardState)*** Matt gesetzt, laut ***isStaleMate(boolean, BoardState)*** Patt gesetzt oder ob laut ***isDraw(boolean, BoardState)*** andersweitig ein Unentschieden erreicht wurde.

- ***getResult()*** Ruft ***hasEnded(BoardState)*** auf, gibt aber bei beendetem Spiel das jeweilige Ergebnis als Result mit Begründung zurück. Ist das Spiel nicht beendet wird null zurückgegeben.

- ***getPossibleMoves(Position, BoardState):*** Gibt alle möglichen Züge einer Figur auf dem Brett zurück, ohne dabei zu berücksichtigen, ob der ziehende Spieler nach diesem Zug im Schach stehen würde.

- ***isMate(boolean, BoardState):*** Prüft, ob der übergebene Spieler Matt gesetzt wurde. Das ist der Fall, wenn der Spieler laut ***isChecked(boolean, BoardState)*** im Schach steht, und es für keine Figur einen nach ***getLegalMoves(Position, BoardState)***erlaubten Zug gibt.

- ***isStaleMate(boolean, BoardState):*** Prüft, ob der übergebene Spieler Patt gesetzt wurde. Das ist der Fall, wenn der Spieler laut ***isChecked(boolean, BoardState)*** nicht im Schach steht, und es für keine Figur einen nach ***getLegalMoves(Position, BoardState)***erlaubten Zug gibt.

- ***isDraw(boolean, BoardState):*** Prüft, ob ein anderweitiges Unentschieden erreicht wurde. Dies ist der Fall, wenn zu wenig Figuren auf dem Brett vorhanden sind, um Matt zu setzen, oder wenn 50 Züge lang keine Figur geschlagen und kein Bauer gezogen wurde.

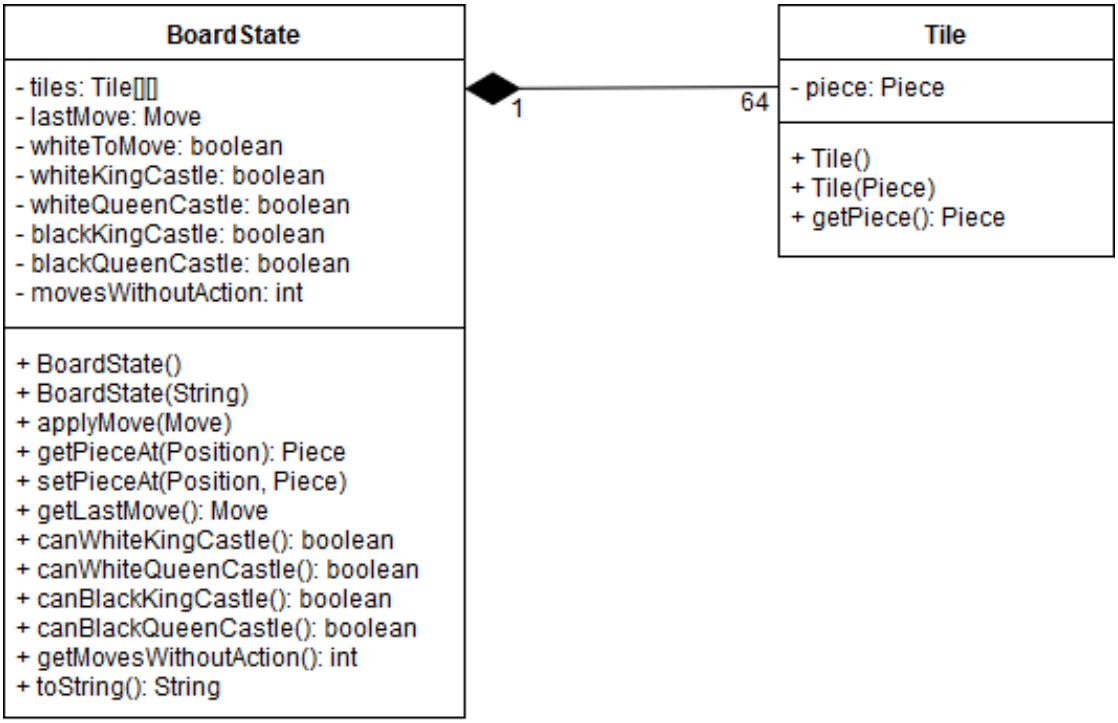

Abbildung 4: BoardState

- ***BoardState***

  Der gesamte Status eines Schachspiels er notwendig ist, um alle Regeln überprüfen zu können.

  tiles: Ein Array von Tiles, welches den Aufbau des Spielbretts darstellt

  lastMove: Der letzte gespielte Zug.

  whiteToMove: Speichert, ob Weiß am Zug ist.

  whiteKingCastle: Speichert ab, ob Weiß noch auf der Königsseite rochieren kann.

  whiteQueenCastle: Speichert ab, ob Weiß noch auf der Damenseite rochieren kann.

  blackKingCastle: Speichert ab, ob Schwarz noch auf der Königsseite rochieren kann.

  blackQueenCastle: Speichert ab, ob Schwarz noch auf der Damenseite rochieren

kann.

movesWithoutAction: Speichert die Anzahl der Züge in Folge, in welcher kein Bauer gezogen und keine Figur geschlagen wurde.

BaordState(): Erzeugt ein leeres Schachbrett und setzt alle Variablen auf ihre Standardwerte.

BoardState(String): Erzeugt ein Brett, ausgehen von einem String. In diesem müssen alle notwenidigen Informationen in einem bestimmten Format gespeichert sein.

applyMove(Move): Führt einen Zug auf dem Brett aus, indem es die Figur(en) wie im Move-Objekt vorgegeben bewegt.

getPieceAt(Position): Gibt die Figur an einer Position zurück.

setPieceAt(Position, Piece): Setzt eine Figur an eine bestimmte Position.

getLastMove(): Gibt ***lastMove*** zurück.

canWhiteKingCastle: Gibt zurück, ob Weiß noch auf der Königsseite rochieren kann.

canWhiteQueenCastle: Gibt zurück, ob Weiß noch auf der Damenseite rochieren kann.

canBlackKingCastle: Gibt zurück, ob Schwarz noch auf der Königsseite rochieren kann.

canBlackQueenCastle: Gibt zurück, ob Schwarz noch auf der Damenseite rochieren kann.

getMovesWithoutAction): Gibt ***movesWithoutAction*** zurück.

toString() Gibt den gesamten Brettzustand als String codiert zurück. Aus diesem String muss mithilfe des Konstruktors ein identisches Brett erzeugt werden können.

- ***Tile***

  Stellt eine Feld eines Schachbretts dar.

piece Die Figur, welche sich auf dem Feld befindet. Befindet sich keine Figur auf dem Feld, steht hier null.

Tile(): Konstruktor, welcher ein leeres Feld erzeugt.

Tile(Piece): Konstruktor, welcher ein Feld mit der übergebenen Figur darauf erzeugt.

getPiece(): Gibt ***piece*** zurück.

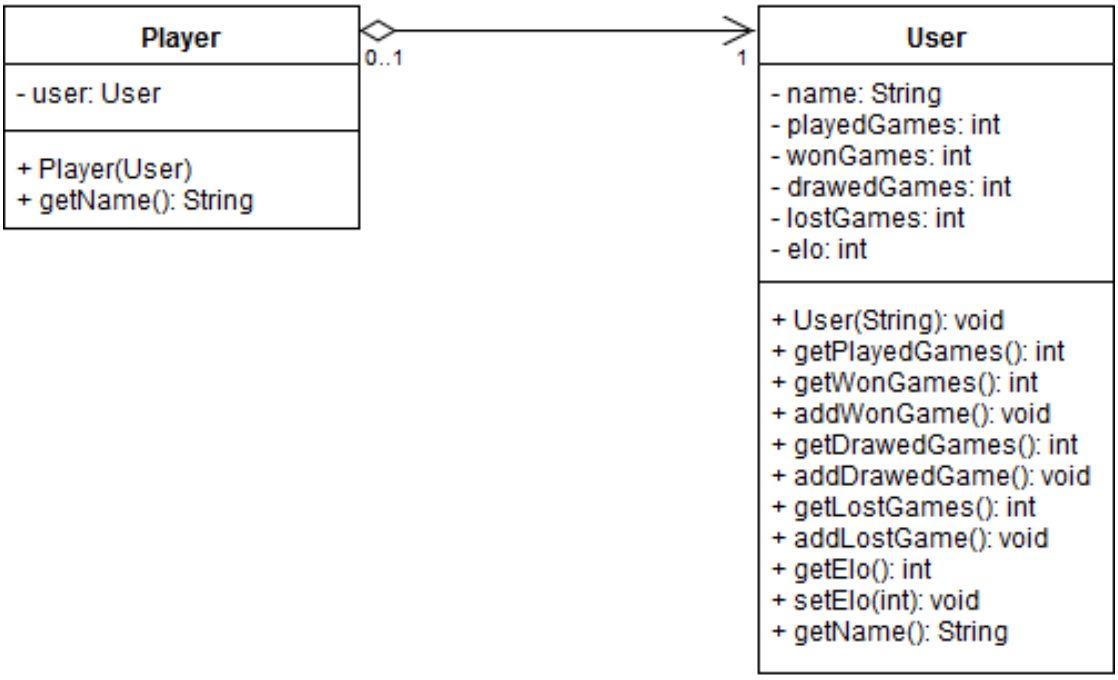

Abbildung 5: Player

-
-

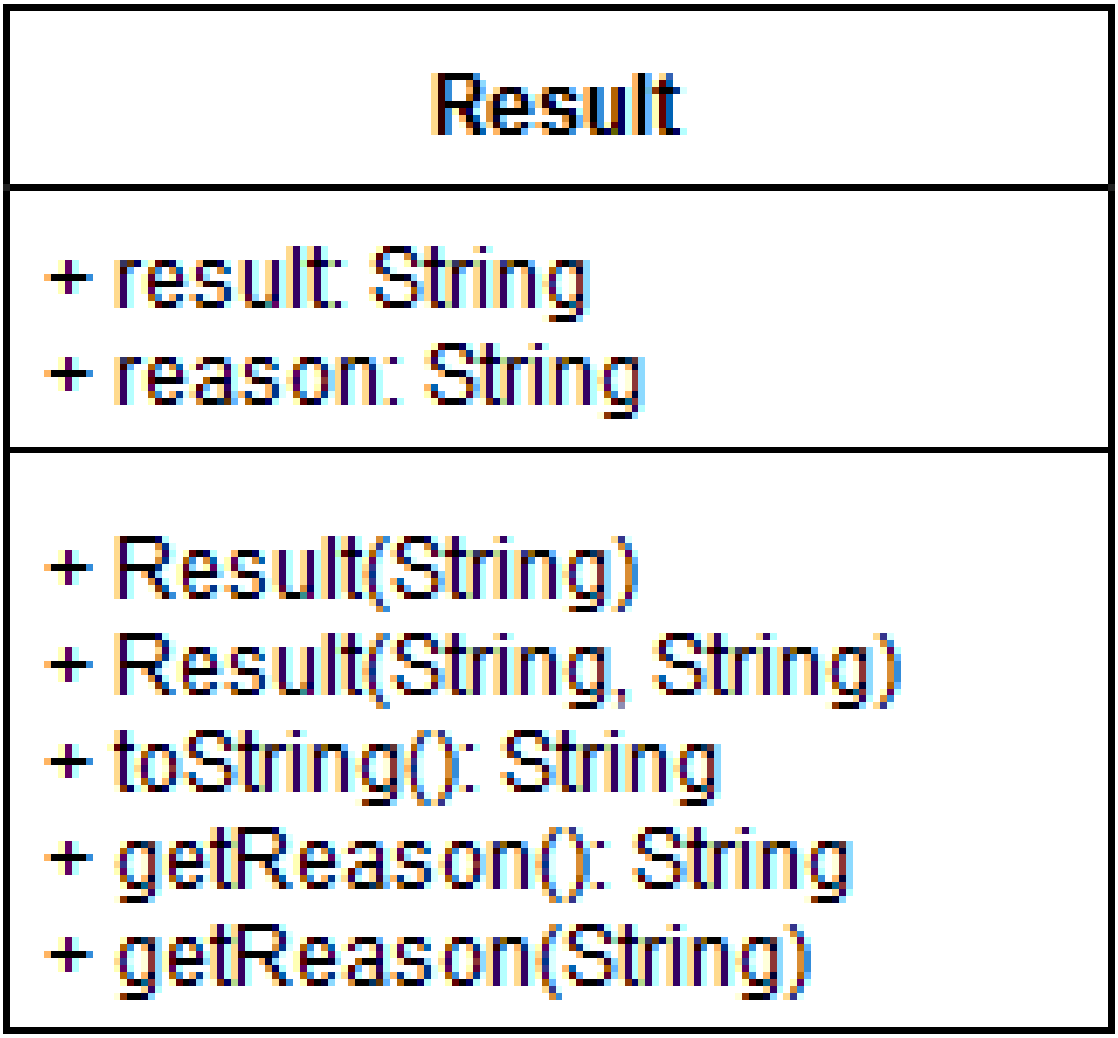

Abbildung 6: Result

-
-

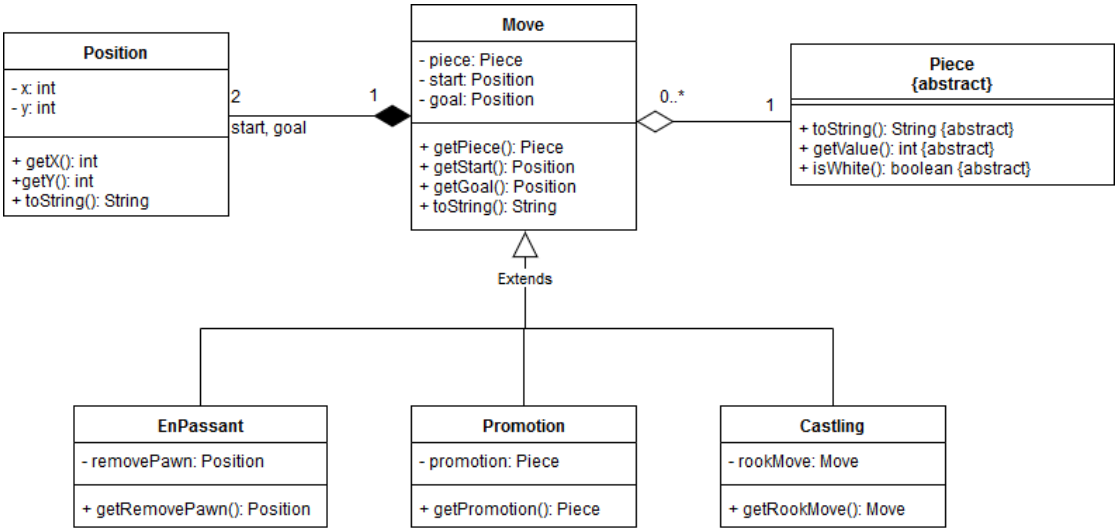

Abbildung 7: Move

-
-

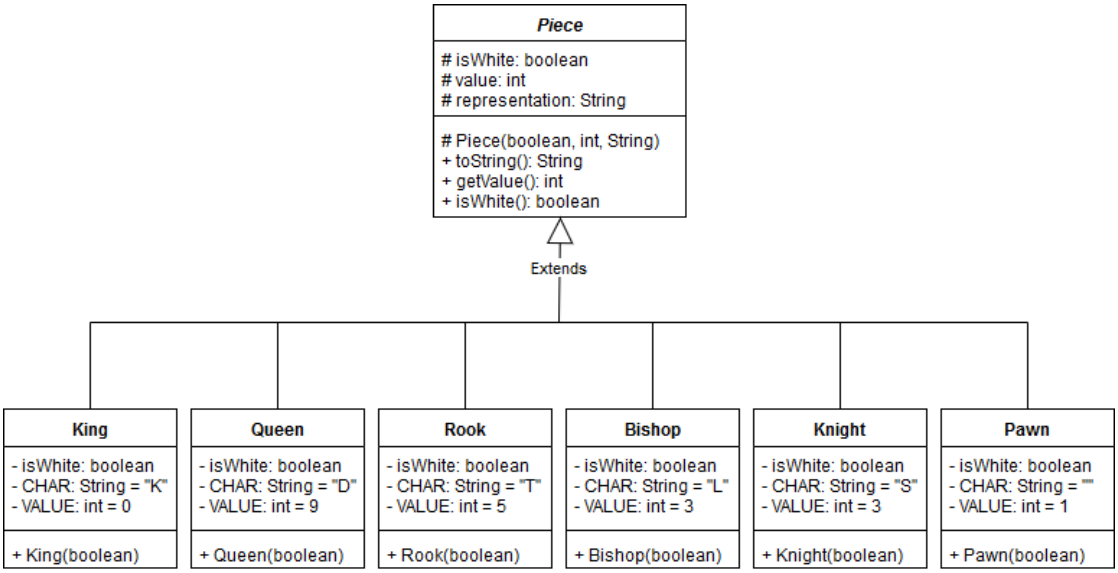

Abbildung 8: Pieces

-
-

2.2 GUI

2.3 Server

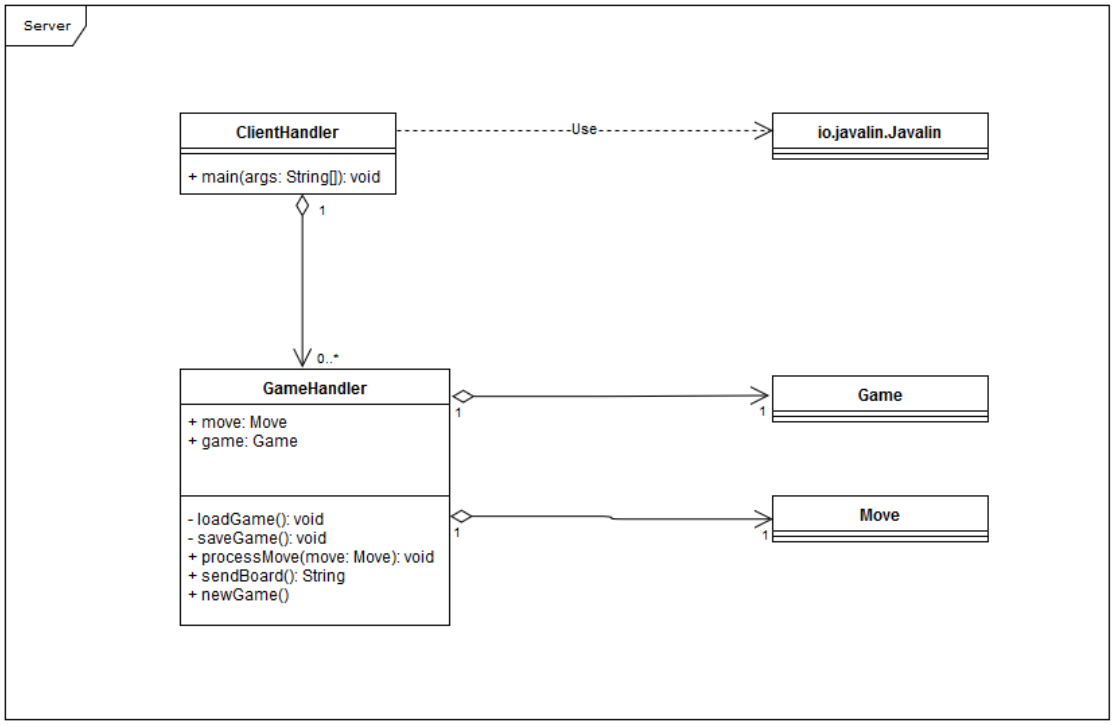

Abbildung 9: Server

- **ClientHandler** Verwaltet eingehende Anfragen an den Server.

  main(String[] args): Die main Methode des Servers. Hier wird der Javalin Server gestartet und für jede Anfrage ein neues GameHandler Objekt erzeugt. Diesem werden die gesendeten Anfragen übermittelt.

- **GameHandler**

  Move move: Der Zug, den der GameHandler übermittelt bekommen hat.

  Game game: Das Spiel, das aus der Datenbank geladen wurde.

  loadGame(): Lädt das aktuelle Spiel des Spielers aus der Datenbank und erzeugt daraus ein Game Objekt.

  saveGame(): Speichert das Game Objekt wieder als String in der Datenbank ab.

  processMove(): Führt zuerst loadGame() aus, überprüft den gegebenen Zug auf

Gültigkeit, wendet diesen an, sendet ihn an den anderen Spieler und führt anschließend saveGame() aus.

sendBoard(): Sendet das Brett des Spiels per Firebase Cloud Messaging(FCM).

newGame(): Legt einen neuen Eintrag in der Datenbank an und füllt ihn mit beiden Benutzernamen und der Startposition.

3 Sequenzdiagramm

Glossar

FCM Firebase Cloud Messaging https://firebase.google.com/docs/cloud-messaging/. 15