## **Board** + COLUMN\_NOTATION : Character[] + PIECE\_NOTATION : Character[] + PIECE PRINT : Character[] + ROW\_NOTATION : Character[] + startFEN : String + testFEN : String + test2FEN : String + failSaveFEN : String + checkmateFEN : String + stalemateFEN : String + weirdPositionFEN : String + pawnDebudFEN : String + castleFEN : String + enPassantInitFEN : String + promotionFEN : String + ambitiousFEN : String + previousBoard : Board + previousMove : Move + notation : String + fen : String + board : Tile[][] + colorActive : int + castleAvailable : boolean[] + enPassantTileCoordinate : int[] + halfMoveCounter : int + fullMoveCounter : int + whiteMoves : LinkedList<Move> + blackMoves : LinkedList<Move> + whitePieces : LinkedList<Move> + blackPieces : LinkedList<Move> + whiteKingCoordinate : int[] + blackKingCoordinate : int[] + isWhiteInCheck : boolean + isBlackInCheck : boolean + isWhiteInCheckMate : boolean + isBlackInCheckMate : boolean + isWhiteInStaleMate : boolean + isBlackInStaleMate : boolean + castleCurrentAvailable : boolean[] + eliminateMove : boolean + searchDepth : int + boardStateEvaluation : double + treeStateEvaluation : double + optiMalMove : Move + nextBoardSet : TreeSet<Board> + Board(String fen) + Board(Board previousBoard, Move move, boolean eliminateMove, int searchDepth) + getPiece(int color) : LinkedList<Piece> + getMove(LinkedList<Piece> listPiece) : LinkedList<Move> + getKingPosition(int color) : int[] + isInCheck(int color) : boolean + assignPiece(int[] destinationCoordinate, int currentId) : Piece + limitMoveResultInCheck(int color): void + isInCheckMate(int color) : boolean + isInStalemate(int color) : boolean + checkCastleAvailable(int color): void + fenToBoard(String fen) : void + BoardToFen() : String + calculateValue() : double + printBoard() : void + printBoard1(): void + getNotation() : String + compareTo(Board b) : int

## **Diagramme UML: Board**

