<<enumeration>>
Type

PAWN_WHITE = 0
ROOK_WHITE = 1
KNIGHT_WHITE = 2
BISHOP_WHITE = 3
QUEEN_WHITE = 4
KING_WHITE = 5
PAWN_BLACK = 6
ROOK_BLACK = 7
KNIGHT_BLACK = 8
BISHOP_BLACK = 9
QUEEN_BLACK = 10
KING_BLACK = 11
- VALOR: int (readOnly)
+ getValor(): int
+ getColor(): Turn
+ displayName(): String

<<enumeration>>
Turn

WHITE
BLACK
END
+ next(): Turn
+ endGame(): Turn

<<interface>>
Movable

+ moveTo(place: int, listMoves:List<Integer>, board:Board): boolean
+ getPossibleMoves(board: ChessPiece[][]): List<Integer>

<<interface>>
Positionable

+ getX(): int
+ getY(): int
+ setPosition(position: int): void

<<concrete>>
Board

pieces: ChessPiece[][]
turn: Turn

<>
ChessPiece

position: int
type: Type
n_moves: int
+ getX(): int
+ getY(): int
+ setPosition(position: int): void
+ isWhite(): boolean
+ moveTo(position: int, listMoves: List<Integer>, board: Board): boolean
+ getImgName(): String
# kingCheck(king: ChessPiece, piece: ChessPiece, toPosition: int, piecesMap: ChessPiece[][]): boolean
# isOpponent(other: ChessPiece): boolean

<<controller>>
MainController

- selectedPiece: ChessPiece
- selectedImage: ImageView
- possibleMoves: List<Integer>
- BOARD: Board (OnlyRead)
- BOARD_SQUARES: Region (OnlyRead)
- turn: Turn
- blockPromotion: boolean
+ initialize(): void
+ creatingBoard(): void
+ creatingPieces(): void
+ movingPieces(): void
+ toMark(): void
+ turnPawn(): void

<<concrete>>
Pawn

+ createPawn(position: int, type: Type): Pawn
+ promoteIfEligible(promotionType: Type): ChessPiece

<<concrete>>
King

+ createKing(position: int, type: Type): King
- isValidCastling(rook: ChessPiece, board: ChessPiece[][]): boolean
+ moveTo(position: int, listMoves: List<Integer>, board: Board): boolean
+ sequeMate(board: ChessPiece[][]): boolean

<<concrete>>
Rook

+ createRook(position: int, type: Type): Rook

<<concrete>>
Knight

+ createKnight(position: int, type: Type): Knight

<<concrete>>
Queen

+ createQueen(position: int, type: Type): Queen

<<concrete>>
Bishop

+ createBishop(position: int, type: Type): Bishop

<<utility, non-instantiable>>
PieceUtils

+ getX(position: int): int
+ getY(position: int): int
+ isSamePiece(a: ChessPiece, b: ChessPieces): boolean
+ isType(piece: ChessPiece, type: Type): boolean
+ isSameType(a: ChessPiece, b: ChessPieces): boolean
+ isSameBoardPosition(a: ChessPiece, b: ChessPieces): boolean
+ isSameColor(a: ChessPiece, b: ChessPieces): boolean
+ isPieceUnderAttack(a: ChessPiece, board: ChessPiece[][]): boolean
+ pathSafeForKing(to: int, king: ChessPiece, board: ChessPiece[][]): boolean

<<utility, non-instantiable>>
BoardUtils

+ findPieces(pieces: ChessPiece[][], type: type): List<ChessPiece>
+ getAlledas(referencia: ChessPiece, board: ChessPiece[][]): List<ChessPiece>
+ isPositionOccupied(pieces: ChessPiece[][], position: int): boolean
+ getPieceAt(pieces: ChessPiece[][], position: int): ChessPiece
+ isWithinBounds(position: int): boolean
+ copyBoard(original: ChessPiece[][]): ChessPiece [][]
+ isPathClearHorizontally(from: int, to: to, board: ChessPiece[][]): boolean

<<utility, non-instantiable>>
ImageFactoryUtils

+ createPieceImage(imagePath: String): ImageView
+ createTransparentImage( ): ImageView