Lab 02: Chess Design

Class Diagrams

	Board
- pieces	s : Pieces
- moves	s : list <move></move>
- lastMo	ove: Move
- setLas	stMove: Move
+ setPic + isVali + isOcc + apply + displa + reset	ece(pos: Position): Piece* ece(pos: Position, piece: Piece*): void dPosition(pos: Position): Bool eupied(pos: Position): Bool Move(move: Move): void ay (gout: ogstream): void (): void estMove()

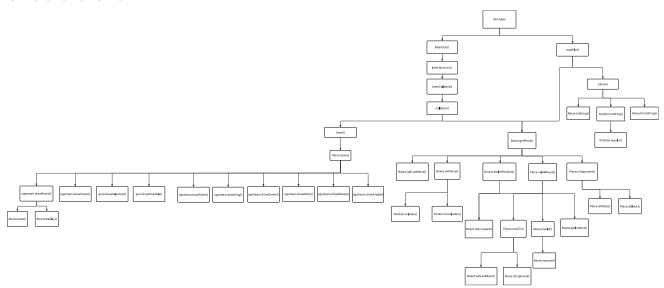
	Position	
- ro	ow: int (07)	
- c	ol: int (07)	
+ f + t	olndex(): int romIndex(i: int): Position oString(): string equals(p: Position): bool	

King
- hasMoved: bool
+ validMoves(board: Board): vector <move> + canCastle(board: Board): bool</move>

Move	
- from: Position	
- to: Position	
- piece: Piece	
- capturedPiece: Piece*	
- piece: Piece	
- promotion: char	
+ isValid(board: Board): bool + execute(board: Board): void + toString(): string + fromString(s: string): Move	t

Piece	
- isWhite: bool	
- pos: Position	
- pos: Position	
- captured: bool	
- hasMoved: bool	
+ isWhite(): bool + isBlack(): bool + isOpponent(piece): bool + validMoves(board: Board): vector <move> + moveTo(pos: Position): void + isCaptured(): bool + draw(gout: ogstream): void</move>	

Structure Chart



Pseudocode

```
FUNCTION checkPawnPromotion(pawn: Piece, board: Board):

IF pawn.isWhite() AND pawn.position.row == 7:
    board.setPiece(pawn.position) SET TO Queen(true, pawn.position)

ELSE IF pawn.isBlack() AND pawn.position.row == 0:
    board.setPiece(pawn.position) SET TO Queen(true, pawn.position)
```

```
FUNCTION checkEnPassant(pawn, board):
    SET moves TO <0>
    SET lastMove TO board.getLastMove()
    IF lastMove == null:
        RETURN moves
    SET fromRow TO lastMove.from.row
    SET toRow
                TO lastMove.to.row
    SET toCol
                TO lastMove.to.col
    IF pawn.color == WHITE:
        IF fromRow == 6 AND toRow == 4:
            IF pawn.position.row == 4:
                IF ABSOLUTE VALUE(pawn.position.col - toCol) == 1:
                    SET target TO Position(toCol, 5)
                    SET moves.add(Move(pawn.position, target, capture
TO lastMove.to))
    ELSE IF pawn.color == BLACK:
        IF fromRow == 1 AND toRow == 3:
            IF pawn.position.row == 3:
                IF ABSOLUTE VALUE(pawn.position.col - toCol) == 1:
                    SET target TO Position(toCol, 2)
                    SET moves.add(Move(pawn.position, target, capture
TO lastMove.to))
    RETURN moves
END FUNCTION
```

```
FUNCTION checkCastling(king, board):
    SET moves TO <0>
    IF king.hasMoved == true:
        RETURN moves
    SET row TO king.position.row
    SET col TO king.position.col
    SET rookKPos TO Position(7, row)
    SET rookK TO board.getPiece(rookKPos)
    IF rookK != null AND rookK.type == ROOK AND rookK.color == king.color AND
rookK.hasMoved == false:
        IF board.isEmpty(Position(5, row)) AND board.isEmpty(Position(6, row)):
            SET kingTarget TO Position(6, row)
            SET rookTarget TO Position(5, row)
            CALL moves.add(
                Move(king.position, kingTarget,
                      castleRookFrom TO rookKPos,
                      castleRookTo TO rookTarget )
            )
    SET rookQPos TO Position(0, row)
    SET rookQ TO board.getPiece(rookQPos)
    IF rookQ != null AND rookQ.type == ROOK AND rookQ.color == king.color AND
rookQ.hasMoved == false:
        IF board.isEmpty(Position(1, row)) AND board.isEmpty(Position(2, row)) AND
board.isEmpty(Position(3, row)):
            SET kingTarget TO Position(2, row)
            SET rookTarget TO Position(3, row)
            CALL moves.add(
                Move(king.position, kingTarget,
                      castleRookFrom TO rookQPos,
                      castleRookTo TO rookTarget )
            )
    RETURN moves
END FUNCTION
```