CENG 1004 Spring 2023 Midterm Exam

In this homework you are to implement a command line chess program (a simpler version). You should submit your source code and documentation (described in Documentation Section) as described in Submission Section.

Class Diagram

A preliminary design is given below and example implementation of some classes are given at the Appendix Section of this document.

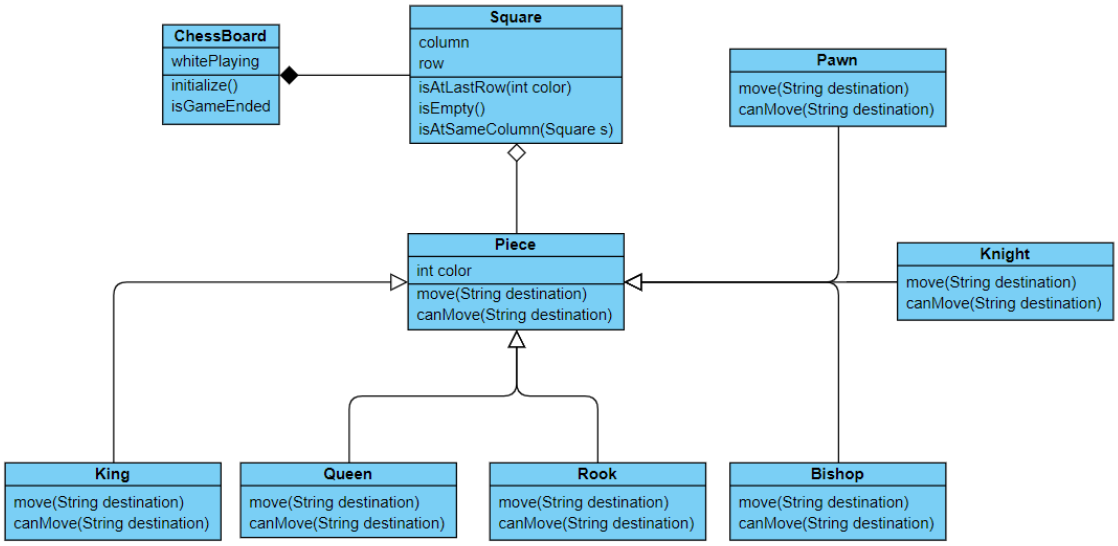

Figure 1 Class Diagram

The class diagram above shows the classes and their relations. ChessBoard class represents board in a chess game. A chessboard object contains 64 Square objects each having attributes to hold location information. In addition, a Square object may contain a Piece object. Piece class is super class of King, Queen, Rook, Bishop, Knight and Pawn classes. The diagram shows some important attributes and methods. In your implementation you may need to define further members.

Expected Functionality

You are expected to implement game partially. In your implementation game can be played with following features

- End game control : you can end the game when no pieces exist from one color.
- Promote: You can assume at the last row Pawn only becomes a Queen .

You don’t need to implement the following functionalities:

- Absolute pin: a piece can be played even it would put the King into check
- Check control: you can assume Kings can be captured as normal pieces.
- En Passant rule: after a Pawn moves two squares from its starting square, and it could have been captured by an enemy Pawn had it advanced only one square.
- Pawn is only promoted to Queen

In summary, you can assume the purpose of the game is to capture opponent’s all pieces, not to check mate.

Board Representation

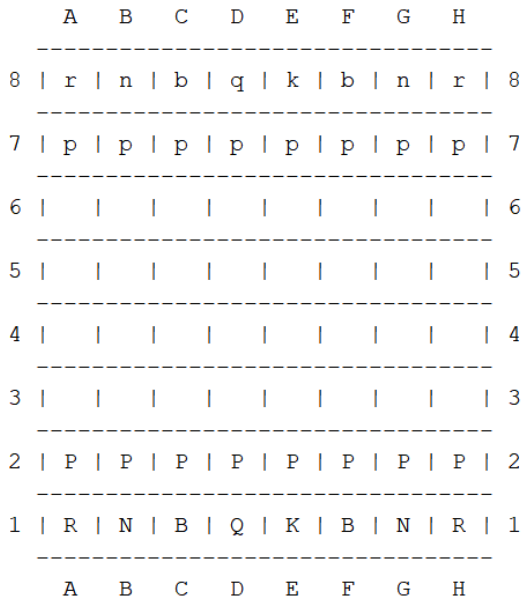

Figure 2 Board Representation

You should draw the board to command line as shown above.

- Coordinate information should be given at all sides as real life chess board.
- Square borders should be drawn with proper symbols as in the Figure 2
- Letters are used to represent pieces in Squares.
  - P for Pawn
  - R for Rook
  - N for Night

- B for Bishop
- Q for Queen
- K for King

➢ Use Uppercase letters for white pieces, lower case letters for black pieces.

Example Execution

```
It is White's turn
Enter the location of the piece:d2
Enter the new location of the piece:d4
    A   B   C   D   E   F   G   H
  ---------------------------------
8 | r | n | b | q | k | b | n | r | 8
  ---------------------------------
7 | p | p | p | p | p | p | p | p | 7
  ---------------------------------
6 |   |   |   |   |   |   |   |   | 6
  ---------------------------------
5 |   |   |   |   |   |   |   |   | 5
  ---------------------------------
4 |   |   |   | P |   |   |   |   | 4
  ---------------------------------
3 |   |   |   |   |   |   |   |   | 3
  ---------------------------------
2 | P | P | P |   | P | P | P | P | 2
  ---------------------------------
1 | R | N | B | Q | K | B | N | R | 1
  ---------------------------------
    A   B   C   D   E   F   G   H
```

Figure 3 White's first move

```
It is Black's turn
Enter the location of the piece:e7
Enter the new location of the piece:e5
    A   B   C   D   E   F   G   H
  ---------------------------------
8 | r | n | b | q | k | b | n | r | 8
  ---------------------------------
7 | p | p | p | p |   | p | p | p | 7
  ---------------------------------
6 |   |   |   |   |   |   |   |   | 6
  ---------------------------------
5 |   |   |   |   | p |   |   |   | 5
  ---------------------------------
4 |   |   |   | P |   |   |   |   | 4
  ---------------------------------
3 |   |   |   |   |   |   |   |   | 3
  ---------------------------------
2 | P | P | P |   | P | P | P | P | 2
  ---------------------------------
1 | R | N | B | Q | K | B | N | R | 1
  ---------------------------------
    A   B   C   D   E   F   G   H
```

Figure 4 Black's first move

```
It is White's turn
Enter the location of the piece:d5
Enter the location of the piece:e5
Enter the location of the piece:d4
Enter the new location of the piece:e5
    A   B   C   D   E   F   G   H
  ---------------------------------
8 | r | n | b | q | k | b | n | r | 8
  ---------------------------------
7 | p | p | p | p |   | p | p | p | 7
  ---------------------------------
6 |   |   |   |   |   |   |   |   | 6
  ---------------------------------
5 |   |   |   |   | P |   |   |   | 5
  ---------------------------------
4 |   |   |   |   |   |   |   |   | 4
  ---------------------------------
3 |   |   |   |   |   |   |   |   | 3
  ---------------------------------
2 | P | P | P |   | P | P | P | P | 2
  ---------------------------------
1 | R | N | B | Q | K | B | N | R | 1
  ---------------------------------
    A   B   C   D   E   F   G   H
```

Figure 5 White's second move (Pawn Capture)

In two attempts location of the white piece was not correct. In fact the main logic of the application is given in Main class which is available in Appendix. You will also find implementation of Pawn class. You should implement the missing classes to provide requested functionality. If you need, you can update Main and Pawn classes.

Grading Policy

A. Printing board : **10 Points** (Objective-1)

B. Creating Board and Square classes and defining their attributes: **10 Points** (Objective - 3)

C. Implementing the methods of Board and Square classes : **20 Points** (Objective - 3)

D. Creating classes in Piece hierarchy as described in UML and defining their attributes : **20 Points** (Objective - 4)

E. Implementing the methods in Piece hierarchy classes: **40 Points** (Objective - 4)

Documentation

For every item in the grading policy you should also provide requested information as described below. Documentation will be graded part of the (20 %) grading . For example, if you do not provide any explanation about printing board in your documentation, you can get at most 8 points instead of 10 Points.

A) Printing Board

Describe how you manage the print the board representation. How did you construct loops. What are the challenges? How did you manage to solve them?

B) Defining Board and Square Classes

A board can have 64 squares. How did you define the relation between Board and Square objects?

C) Implementing methods of Board and Square Classes

Describe each method in these classes except mutator and accessor methods. In each method description you should provide

- ✧ What does the method do ?
- ✧ One sentence description of each parameter
- ✧ What does it return?
- ✧ How did you implement the functionality?

Example:

```
public Square[] getSquaresBetween(Square s1, Square s2)
```

- ✧ Returns the squares between given squares if they are at the same row or same column or same diagonal
- ✧ s1: the beginning square ; s2: final square

- ✧ Array of squares as the same order from s1 square to s2 square not including s1 and s2, null is returned if no squares are in between
- ✧ First, whether the given squares are at the same row or same column or same diagonal is checked. If this is the case, based on the distances of squares an array is created. Arrays element are assigned with the squares by iterating from s1 Square to s2 Square.

D) Defining Piece Hierarchy

Explain how Main class benefits from polymorphism. Explain, which methods and classes can be defined abstract in Piece hierarchy. Is there a code reuse in your implementation?

E) Implementing methods in Piece Hierarchy

Do the same as described in C for the classes in Piece hierarchy

Submission

Zip your source folder as yourid.rar. Also save your documentation as yourid.pdf. If your id is "12345678" you will submit the following files:

- ✓ 12345678.rar : archive file containing your source code
- ✓ 12345678.pdf: documentation file containing explanations request in the Documentation Section

Submit your file through DYS system. You should be able to upload files in the homework announcement page.

Appendix

```
public class Main {

    public static void main(String[] args)  {
        Scanner reader = new Scanner(System.in);
        ChessBoard board = new ChessBoard();
        System.out.println(board);

        while (!board.isGameEnded()){
            System.out.println("It is " + (board.isWhitePlaying() ? "White" : "Black") + "'s turn");
            Piece piece = null;
            do {
                System.out.print("Enter the location of the piece:");
                String from = reader.next();
                piece = board.getPieceAt(from);
            }while(piece == null || piece.getColor()!=(board.isWhitePlaying() ? ChessBoard.WHITE :
ChessBoard.BLACK));

            String to = null;
            do {
                System.out.print("Enter the new location of the piece:");
                to = reader.next();
            }while(!piece.canMove(to));

            piece.move(to);
            System.out.println(board);
        }
        reader.close();
    }
}
```

Figure 6 Main.java

```
public class Pawn extends Piece {

    boolean initialLocation = true;

    public Pawn(int color, Square location) {
        super(color, location);
    }
    @Override
    public boolean canMove(String to) {
        boolean validMove = false;
        Square targetLocation = location.getBoard().getSquareAt(to);
        int rowDistance = targetLocation.getRowDistance(location);
        if (this.location.isAtSameColumn(targetLocation)) {
            if (color == ChessBoard.WHITE && rowDistance > 0 && rowDistance <= 2) {
                if (rowDistance == 2) {
                    if (initialLocation) {
                        //pawn is moving twice, check two squares in front are empty
                        Square[] between = location.getBoard().getSquaresBetween(location,
targetLocation);
                        validMove = targetLocation.isEmpty() && between[0].isEmpty();
                    }
                } else {
                    validMove = targetLocation.isEmpty();
                }
                return validMove;
            } else if (color == ChessBoard.BLACK && rowDistance < 0 && rowDistance >= -2) {
                if (rowDistance == -2) {
                    if (initialLocation) {
                        //pawn is moving twice, check two squares in front are empty
                        Square[] between = location.getBoard().getSquaresBetween(location,
targetLocation);
                        validMove = targetLocation.isEmpty() && between[0].isEmpty();
                    }
                } else {
                    validMove = targetLocation.isEmpty();
                }
            }
```

```
            // attacking diagonals
        } else if (this.location.isNeighborColumn(targetLocation)) {
            if (color == ChessBoard.WHITE && rowDistance == 1) {
                validMove = !targetLocation.isEmpty() && targetLocation.getPiece().getColor() ==
ChessBoard.BLACK;
            } else if (color == ChessBoard.BLACK && rowDistance == -1) {
                validMove = !targetLocation.isEmpty() && targetLocation.getPiece().getColor() ==
ChessBoard.WHITE;
            }

        }
        return validMove;
    }

    @Override
    public void move(String to) {
        Square targetLocation = location.getBoard().getSquareAt(to);
        //promoteToQueen
        if (targetLocation.isAtLastRow(color)) {
            targetLocation.putNewQueen(color);
        } else {
            targetLocation.setPiece(this);
        }
        //clear previous location
        location.clear();
        //update current location
        location = targetLocation;
        location.getBoard().nextPlayer();

    }
    @Override
    public String toString() {
        return color == ChessBoard.WHITE ? "P" : "p";
    }
}
```

Figure 7 Pawn.java

Note that, objects interact with others using their methods. They don't access the internal representation directly. Pawn object does not know how board is represented or how coordinates of a square represented. Classes should have minimum information about each others' internal details. You should follow this principle in your implementations. Also note that, if you introduce a new type of piece to chess, very few code should be affected. For example, if you create a Rook class no need to update the Main class and other Piece classes.