Assignment 2

In Assignment 1, you created a chess board and placed a knight piece on the chess board. Further more, you allowed a knight piece to move around as long as the moves were legal:

- Setup a Chess Board (8x8) (with a grid and draw it textually perhaps)
- Introduce a Knight
    - placement on chessboard
    - legal / illegal moves

Overview:

Now in Assignment 2, we add more pieces and functionality to our Chess Game, namely:

1. Setup Chess Board and Initialize knight to its default place
2. Introduce 1 Queen, 2 Bishops, 2 Rook, 1 King and 8 pawns
    a. placement on chessboard
    b. legal / illegal moves
3. For all pieces
    a. Identify shadows (a possible move is blocked due to another piece in the path)
    - Identify threats

For all intents and purposes, we are adding pieces and functionality for only and only black pieces – not the white pieces.

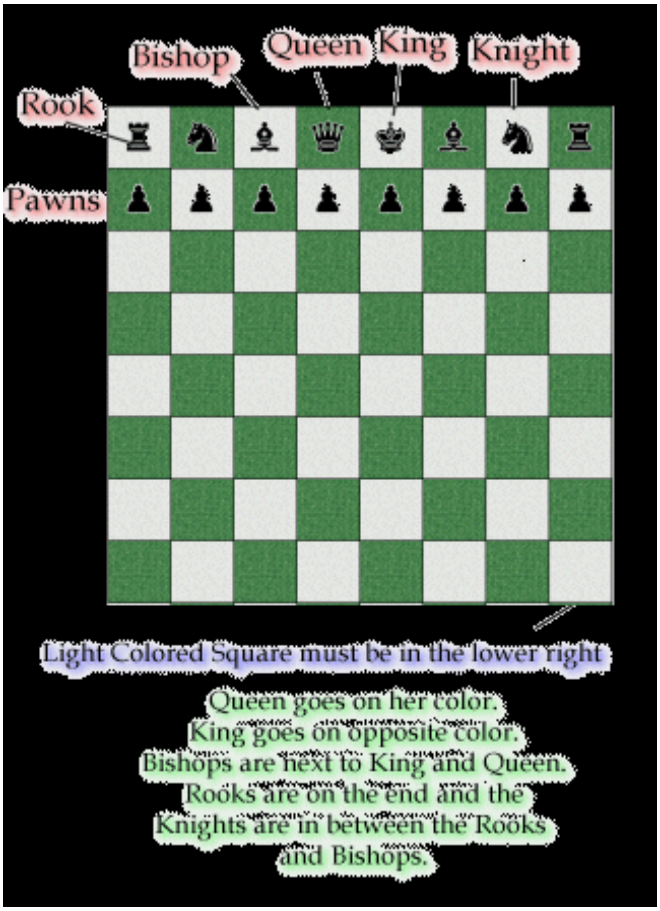


Figure 1 – Chess Game Layout

1. Chess Game Details

1.1. Initialize knights to their default place

This is primarily what you have already done for Assignment 1. If you have not done so, please go back and do Assignment 1. Please make sure of the following:

a) There should now be 2 knights
b) These knights should be initialized at location (0,1) and (0,6) on the Chess Board:

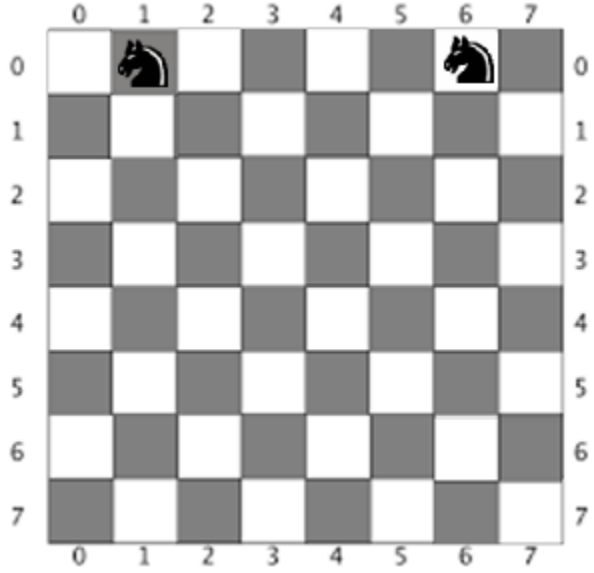


Figure 2 – Default location of Knights

1.2. Introduction of Remaining Chess Pieces

1.2.1. King

Though not the most powerful piece on the board, the king is the most vital, for once he is lost the game is lost. There is only 1 king piece.

The object of the game is to threaten the opponent's king in such a way that escape is not possible (checkmate). If a player's king is threatened with capture, it is said to be in check, and the player must remove the threat of capture on the next move. If this cannot be done, the king is said to be in checkmate, resulting in a loss for that player. Although the king is the most important piece, it is usually the weakest piece in the game until a later phase, the endgame. Players cannot make any move that places the king in check.

The king's default location is at (0, 4) as shown in Figure 3a. The king moves one square in any direction, as shown in Figure 3b.

Even though the king also participates in a special move called castling in a real game of chess, our set of assignments will **NOT** support castling.

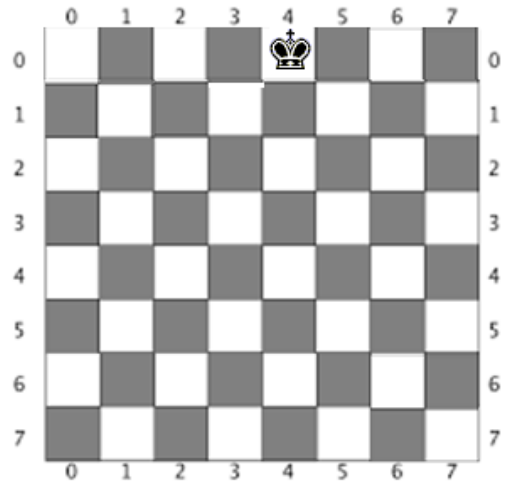


Figure 3a – Default location of King

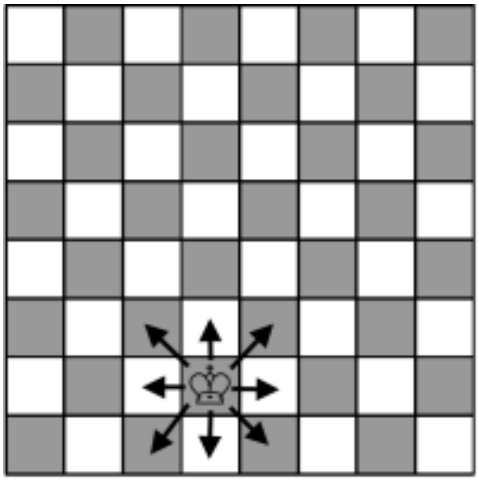


Figure 3b – King Possible Moves

1.2.2. Queen:

To many, a queen is the most powerful and most versatile piece on the chess board. There is only 1 queen piece.

A queen piece should start off at its default location at (0,3) as shown in Figure 4a. She can move as many squares as she desires and in any direction (barring any obstructions) – in horizontal, vertical, or diagonal direction, as shown in Figure 4b.

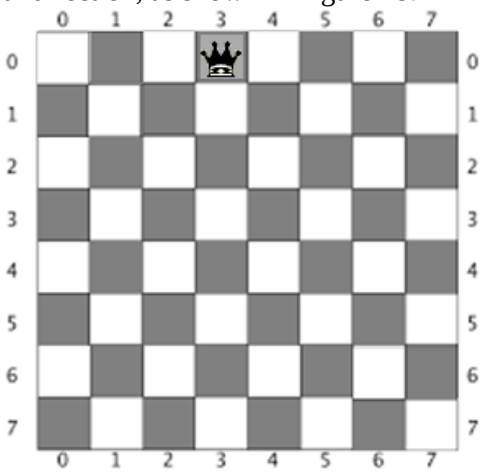


Figure 4a – Queen Default Location

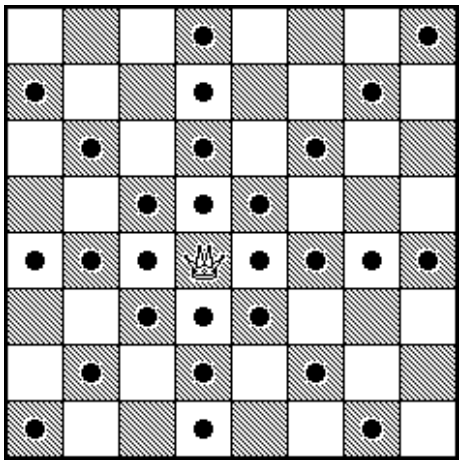


Figure 4b – Queen Possible Moves

She captures in the same way that she moves, replacing the unlucky opposing piece that get in her way. (She must, of course, stop in the square of the piece she has captured - unlike the knight the queen may not jump other pieces.)

1.2.3. Bishop

Each player begins the game with two bishops, one originally situated on a light square, the other on a dark square. Because of the nature of their movement, the bishops always remain on the same colored squares. A bishop on the black square can not occupy a white square and vice-versa.

The bishop pieces start off at their default locations at (0,2) and (0,5) as shown in Figure 5a.

The bishop may move any number of squares in a diagonal direction until it is prevented from continuing by another piece, as shown in Figure 5b.. It may then capture the opposing piece by landing on the square. It may not jump over pieces as can the knight.

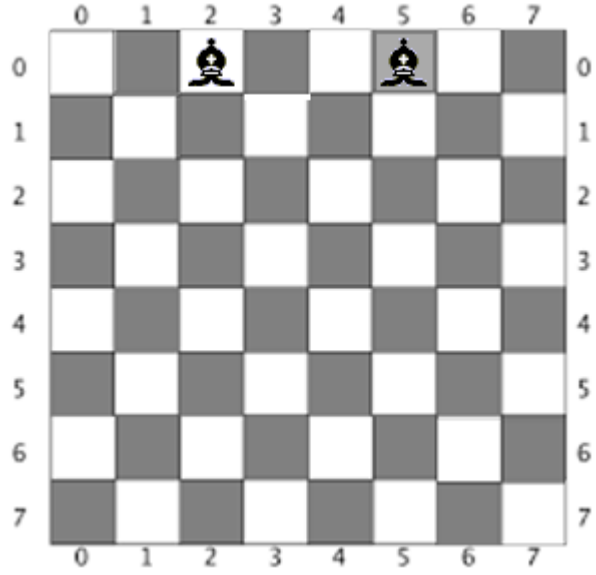


Figure – 5a Bishop Default Locations

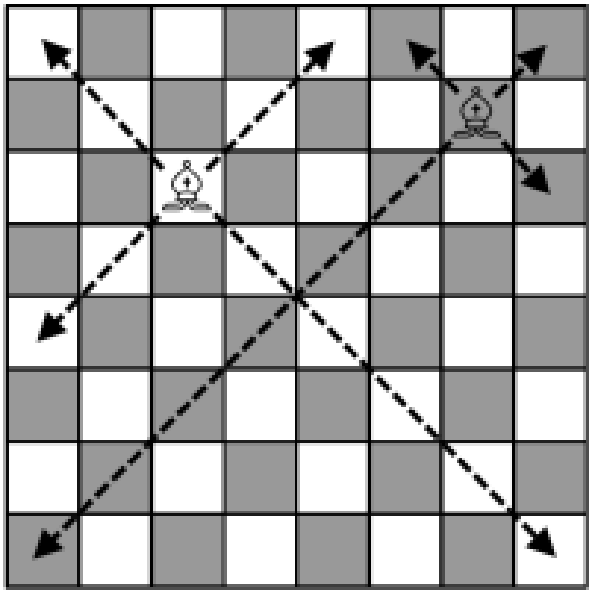


Figure 5b– Bishop Possible Moves

1.2.4. Rook

The rook, shaped like a castle, is one of the more powerful pieces on the board. Each player starts the game with two rooks, one in each of the corner squares on their own side of the board at (0,0) and (0,7) as shown in Figure 6a.

The rook can move any number of squares in a straight line along any column or row, through any number of unoccupied squares (see Figure 6b). They CANNOT move diagonally for any reason.

As with captures by other pieces, the rook captures by occupying the square on which the enemy piece sits. Even though the rook also participates in a special move called castling, our set of assignments will **NOT** support castling.

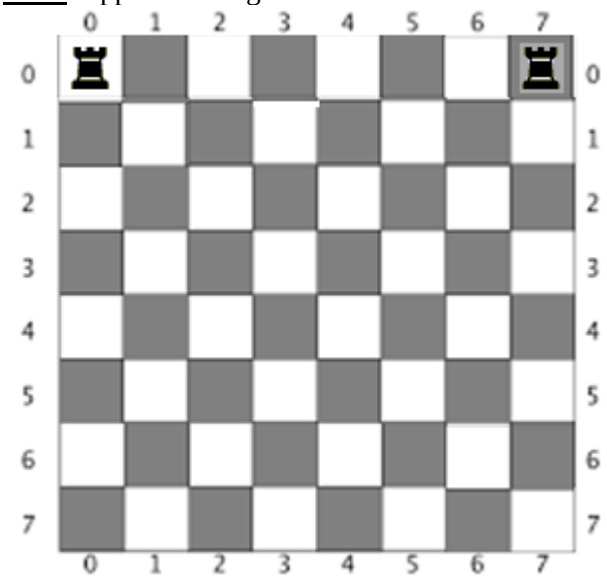


Figure 6a – Rook Default Locations

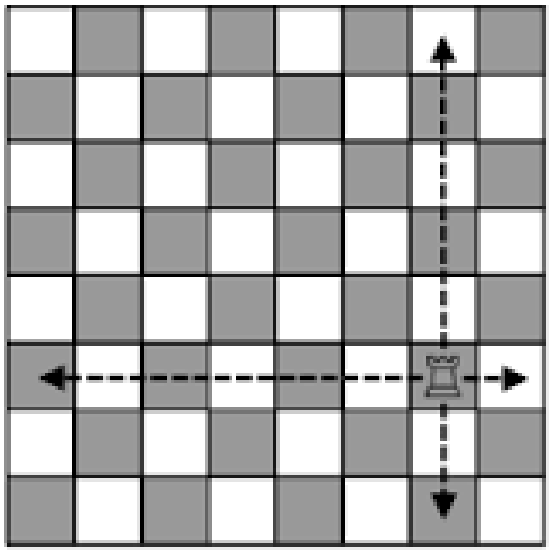


Figure 6b– Rook Possible Moves

1.2.5. Pawn

The pawn is the most numerous piece in the game of chess, and in most circumstances, also the weakest. It historically represents infantry, or more particularly, armed peasants or pikemen. Each player begins a game of chess with eight pawns, one on each square of the rank immediately in front of the other pieces.

Unlike the other pieces, pawns may not move backwards. Normally a pawn moves by advancing a single square, but the first time a pawn is moved, it has the option of advancing two squares. Pawns may not use the initial two-square advance to jump over an occupied square, or to capture. Any piece directly in front of a pawn, friend or foe, blocks its advance.

Unlike other pieces, the pawn does not capture in the same direction as it otherwise moves. A pawn captures diagonally, one square forward and to the left or right as shown in Figure 7b and 7c.

In a game of chess, there is a concept of pawn promotion – that is a pawn that advances all the way to the opposite side of the board (the opposing player's first rank) is promoted to another piece of that player's choice: a queen, rook, bishop, or knight of the same color. For our sets of assignments, we do **NOT** support / model / implement pawn promotion.

For our game, our default pawn placement is (1,0), (1,1), (1,2), (1,3), (1,4), (1,5), (1,6), and (1,7) as shown in Figure 7a.

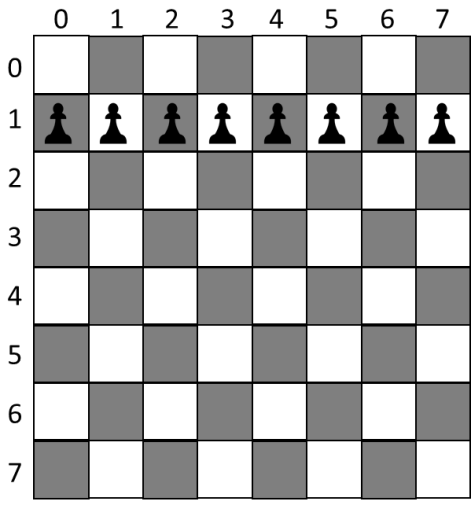


Figure 7a – Pawn Default Locations

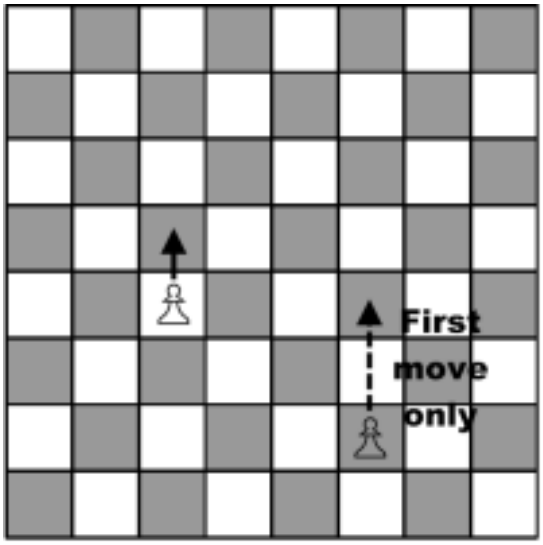
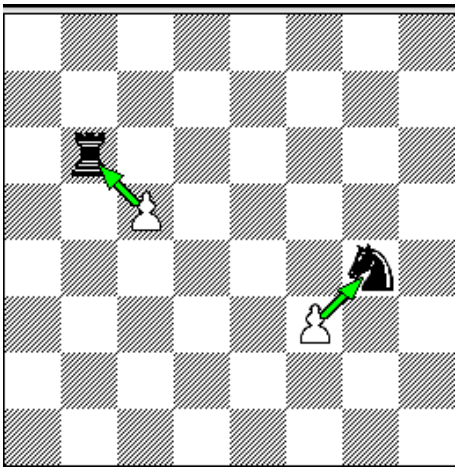


Figure 7b – Pawn Possible Moves



7c - Pawn Capture

1.3. For all pieces, identify shadows

For all pieces, make sure you identify shadows and do NOT allow a piece to move in a shadow. A shadow is location on the chess board where a piece may legally go, but is blocked due to another piece (either a friendly or an enemy piece).

For example, in the Figure 8a, possible legal moves for a rook are shown with orange arrows.

However, in Figure 8b, the same rook can not occupy the same spaces due to its own pieces in the way. The pawn at (4,2) does not allow the rook to occupy (4,0) and (4,1) and the bishop at (1,5) does not allow the rook to occupy (0,5).

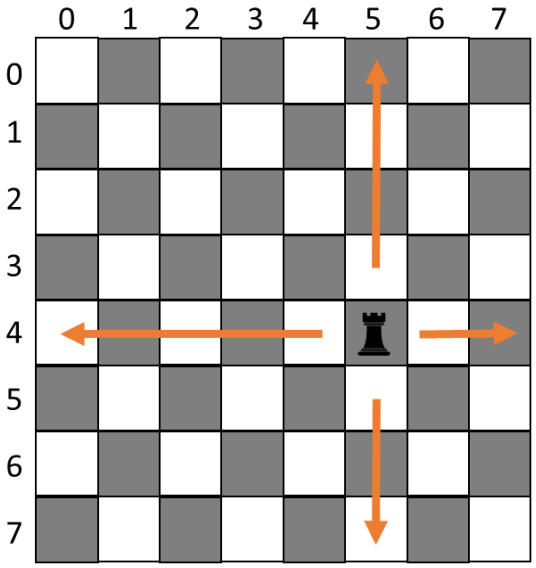


Figure 8a – Legal Moves for Rook

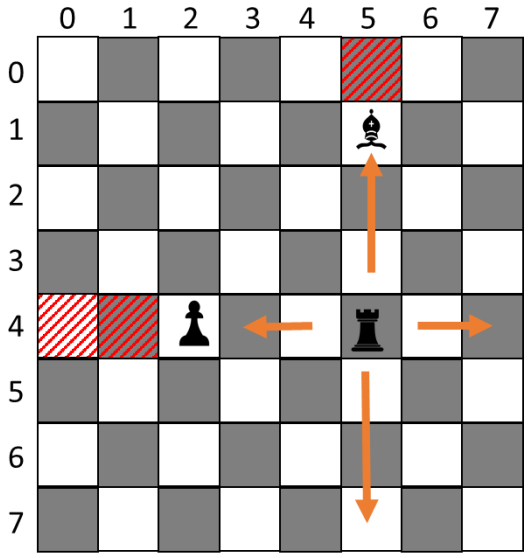


Figure 8b – Blocked moves of Rook

These are called shadows, where a piece is legally allowed to move but is blocked by another piece. All pieces except the knight can get blocked in similar fashion. **Knight piece does not get blocked!**

2. Classes

To help you develop the chess game, find below description of some of the classes you'll need

2.1. Class ChessBoard

Your ChessBoard class should allow you to store different pieces at different locations (that is what a real life chessboard does after all)

Variables:

You decide how to represent your chess board (perhaps a 2 D array, etc). Strictly speaking, you're free to choose whatever data structure you like, provided the methods pieceAt, addPiece, and removePiece all work like they're supposed to. Let's just say, for example, that you wanted to use a two-dimensional array to represent the spaces on the chess board (let us call it the "spaces" array).

Methods:

Public accessors and mutators methods for your private variables.

Additionally, it should have at least the following methods and a constructor.

Constructor: *public ChessBoard()*

The constructor will initialize your spaces array to an empty 8x8 array.

Method #1: public ChessPiece getPieceAt(ChessLocation location)

This method should return the piece at the specified location

Method #2: *public void placePieceAt (ChessPiece piece, ChessLocation location)*

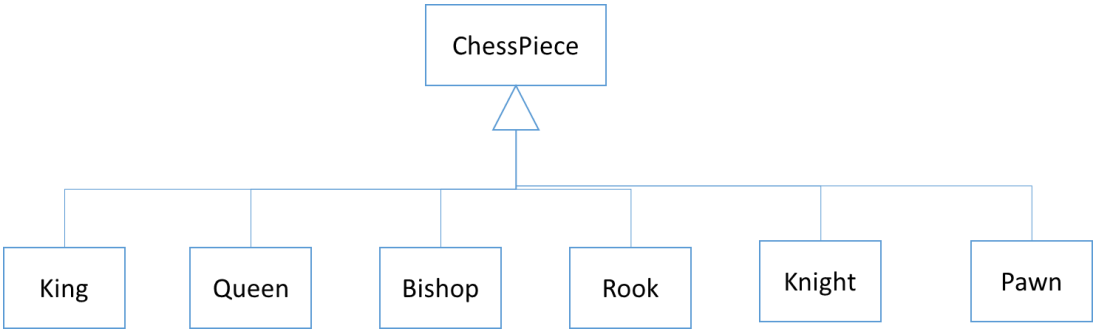
placePieceAt should place the given piece on your ChessBoard (i.e. in your "spaces" array) at the given location. **If the user attempts to add a piece to a location where one already exists,** *placePieceAt* **should overwrite the old piece with the new one.** Do not forget to update the piece's own location in its own class.

Method #3: *public void removePiece(ChessLocation location)*

The method *removePiece* should remove whatever piece is at the specified row and column from your "spaces" data structure, setting the value null at that row and column to be null. This method can be used to remove a piece on the board from the original location after a move. Do not forget to update the piece's own location to null.

In addition to these three required methods, you're free to add any other methods to the ChessBoard class in order to make your job easier. For instance, it might not be a bad idea to have toString method which draws out your ChessBoard

2.2. Class ChessPiece



Chess Piece will be a superclass for the classes: King, Queen, Bishop, Rook, Knight and Pawn.

Every piece needs to know where it is and which player it belongs to. Make sure you have at least the following:

Variables:

private ChessGame game
- To determine which game does this piece belong to

private String player
- Ideally Chess is a 2 person game identified by colours black and white. In this assignment, even though you do not have a 2nd person playing, I still want you to identify the owner of the piece (i.e. player = "player1" or "player2"). Keep it as "player1" for all your pieces

private ChessLocation location
- Current location of the chess piece

protected char id
- This identifies the piece on your board when you are displaying your chess board. The following should be your table:

Piece	**Mark**			**Piece**	**Mark**	
	Player 1	**Player 2**			**Player 1**	**Player 2**
King	K	k		Knight	N	n
Queen	Q	q		Rook	R	r
Bishop	B	b		Pawn	P	p

In this assignment, you do not have player 2, so you need not to worry about player 2 pieces

Methods

Public accessors and mutators methods for all your variables.

ChessPiece(String owner, ChessLocation initialLocation, ChessGame game)

- Initialize the owner, and the game to the given variables
- Initialize the location to null
- Place yourself on the chessboard with the initialLocation (hint: you need to get the chessboard reference from the game object passed)

public void moveTo(ChessLocationClass newLocation)

- Check the legality of the move
- If the move is legal, get ChessBoard from the object game, remove that piece from the old location and place that piece at the newLocation
- make sure the pieces can not be moved to indexes outside of the bounds of the chessboard

protected boolean checkLineOfSight(ChessLocation start, ChessLocation end)

- this is a complex function
- it should return if there is a line of sight from the position start to the position end. This means there are no obstructions from start to end i.e. there are no pieces between start to end.
- Between start to end can means:
    - you should check horizontally,
    - you should check vertically and
    - you should diagonally (perfect diagonal) with a slope of either 1 or -1

2.3. Class Bishop extends ChessPiece

No variables (except the ones it inherits from the superclass automatically)

Methods:

Constructor:

public Bishop(String player, ChessGame game, ChessLocation initial_ location)

- call the constructor of the super class with appropriate arguments
- check for the player and set the mark accordingly (player can either be “Player1” or “Player2”

public void moveTo(ChessLocation destination)

- check the legality of the move i.e. check if it is being asked to move diagonally or not
    - if it is being asked to move diagonally, check line of sight using the method from super class
        - if okay, call the moveTo of the super class, otherwise output an error message

hint: to check diagonality:

i) check if x coordinates of the source and destination are the same or different.

ii) check if y coordinates of the source and destination are the same or different.

2.4. Class Knight extends ChessPiece

No variables (except the ones it inherits from the superclass automatically)

Methods:

Constructor:

public Knight(String player, ChessGame game, ChessLocation initial_ location)

- call the constructor of the super class with appropriate arguments
- check for the player and set the mark accordingly (player can either be “Player1” or “Player2”

public void moveTo(ChessLocation destination)

- check the legality of the move using what you did in Assignment 1
- if okay, call the moveTo of the super class, otherwise output an error message

---

2.5. Class Queen extends ChessPiece

No variables (except the ones it inherits from the superclass automatically)

Methods:

Constructor:

public Queen(String player, ChessGame game, ChessLocation initial_ location)

- call the constructor of the super class with appropriate arguments
- check for the player and set the mark accordingly (player can either be “Player1” or “Player2”

public void moveTo(ChessLocation destination)

- check legality of the move (hint: use check line of sight using the method from super class)
    - if okay, call the moveTo of the super class, otherwise output an error message

---

2.6. Class King extends ChessPiece

No variables (except the ones it inherits from the superclass automatically)

Methods:

Constructor:
public King(String player, ChessGame game, ChessLocation initial_ location)
- call the constructor of the super class with appropriate arguments
- check for the player and set the mark accordingly (player can either be "Player1" or "Player2"

public void moveTo(ChessLocation destination)
- check legality of the move
    - if okay, call the moveTo of the super class, otherwise output an error message

---

2.7. Class Pawn extends ChessPiece

Variables:
Inherited from the superclass automatically

public boolean firstMove
this keeps track if this is the pawn's first move or not. If it is a first move, then it is allowed to move 2 spaces ahead. If not, then only 1 space

Methods:

Constructor:
public Pawn(String player, ChessGame game, ChessLocation initial_ location)
- call the constructor of the super class with appropriate arguments
- check for the player and set the mark accordingly (player can either be "Player1" or "Player2"
- set firstMove to false

public void moveTo(ChessLocation destination)
- check legality of the move
    - if okay, call the moveTo of the super class, otherwise output an error message
    (Just remember we are not capturing any pieces right now, so no need to check for a diagonal capture)
    Also remember to check for the first move exception

2.8. Class ChessLocation

Variables:

int row, col
- To determine where this knight is at the chess board

Methods:
Accessors and mutators methods for the private variables

Constructor:
ChessLocation(int x, int y)
- Initialize the row and column to the given x and y
- Make sure you check the given x and y do not fall out of the chess board indexes

equals (ChessLocation cp)
- Checks if the given location has the same row and column as that of itself

---

2.9. Class ChessGame:

If you are not familiar with the game of Chess, see Section 1 and the Appendix A for a game description*

Even though the games of chess work by alternating turns between a Player 1 and Player 2, you do **NOT** mirror that functionality (at least not yet). For this assignment, you assume there is only Player1 and all of its 16 pieces.

Variables
ChessBoard board
- Chess board to play the game on

String player1
- Models player 1

String player2
- Models player 2

Methods:

- Accessors and mutators methods for the private variables

Constructor: *public ChessGame(String player1, String player2)*
- Initialize the chess board
- Initialize the players
- Initialize all 16 pieces with their appropriate locations on their board and add it to your board (hint: this all can be done in one statement by calling each of the chess piece's constructor and using the "**this"** keyword)

2.10. Class PlayGame

Variables:
- none

Methods:
public static void main(String[] args)
- display an initial menu on the screen describe what the program is about
- make sure you talk about the row and columns are to be used for indexes and give example
- initialize a new chessGame with player "p1" and player "p2"
- retrieve and display the chessBoard from this initialized game
-
- Ask the user if they want to "move" or "quit"
    - If they want to quit, then the program gracefully ends with a goodbye message
    - If they want to "move"
        - Ask user for the source
        - Ask user for the destination
        - Check if there is a piece at the source (there should be one – if not, ask to retry)
        - Check if there is a piece at the destination
            - There should be none. We are playing still a single player game so you can not capture your own piece). If there is one, ask to retry
            - If the destination is empty, move the requested piece to a new location
    - Keep on doing this until "quit" is typed in
        - each time a piece is moved, display the board with the new locations

Deliverables:

From within BlueJ, create a JAR file containing your code (Project menu, click "Create JAR File").
When the "Create Jar File" dialog is displayed, make sure the "Include Source" and "Include BlueJ Project" checkboxes are both checked! If you forget to include the source code, you will get a zero, so make sure you double-check after submission as well. Multiple submissions are

allowed up until the deadline. Only the latest one will be graded

Submit the resulting Jar file on cuLearn.

Marking Scheme:

0	- not submitted - jar file submitted with out the source code - project file submitted only - zipped file submitted instead of the jar file
25	- submitted but doesn't compile or very incomplete
50	- compiles but crashes easily (i.e., terminates by throwing an exception) and/or doesn't follow the assignment requirements. For example, illegal moves are being allowed. - <= 3 of the 6 (King, Queen, Bishop, Knight, Rook and Pawn) pieces only work as outlined - Instructions are not included and/or are not clear on how to run the program
75	- compiles but crashes easily (i.e., terminates by throwing an exception) and/or doesn't follow the assignment requirements. For example, illegal moves are being allowed. - 3 < # of working pieces <= 5 of the 6 (King, Queen, Bishop, Knight, Rook and Pawn) pieces only work as outlined - Missing Javadoc - Missing minor things (such as out of bounds check etc)
100	- all 6 pieces work flawlessly - works, follows requirements - java doc exists for each

Appendix A – How the Game of Chess is Played

The Game

The basic idea behind chess is pretty simple. It's a two-player game played on an 8 by 8 grid (Figure 1), with each player controlling his or her own army of colored pieces (traditionally colored white and black). The game always begins with the pieces arranged as shown in Figure 1, and starts with White moving one of his/her pieces. The players then alternate turns, with each player moving one of their own pieces on their turn (the exact rules governing how each piece can move will be discussed later). If a player moves one of his/her pieces onto a square occupied by his/her opponent's piece, then the enemy piece is said to be "captured" and is removed from the board (Figure 7).

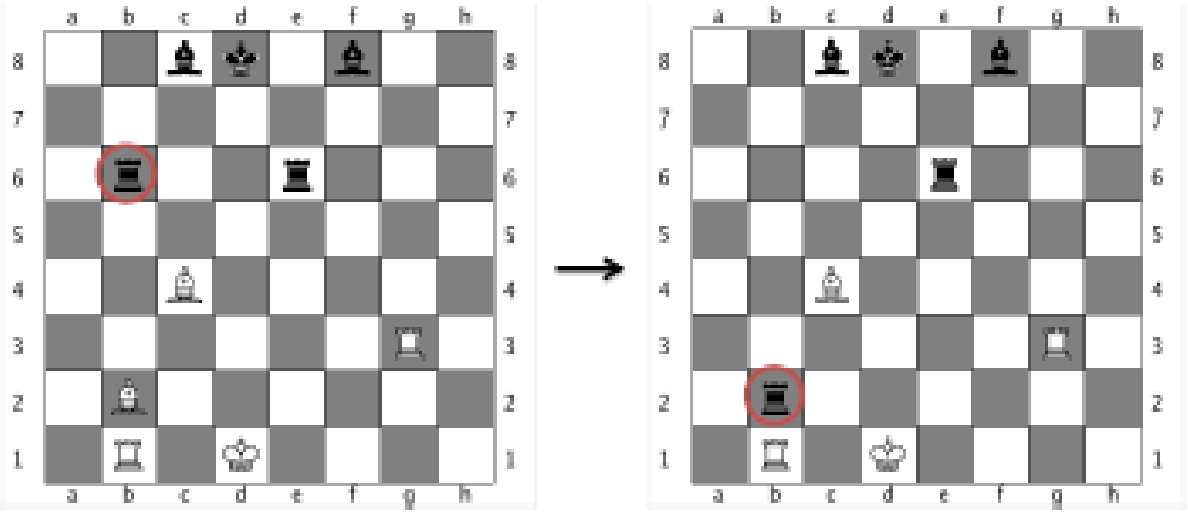


Figure 7. Example of capturing. The black Rook (originally at b6) moves to the same square as the white Bishop (located at b2), so the Bishop is "captured" and removed from the board.

The game ends when one player captures the other player's King. To this end, there are a couple of special scenarios known as *check* and *checkmate*. A player is said to be in *check* if their King is in danger of being captured by an opposing piece in one move. That is, a player is in *check* if his/her opponent could capture his/her King by making a single move. Once a player is in *check*, they are required to make a move to get themselves out of *check* (because if he/she did not, he/she would lose). Similarly, a player is not allowed to make a move that would put him/herself into *check* (again, doing so would immediately cause that player to lose). Sometimes, however, there are situations in which a player is in *check* and every legal move to available would also result in him/her being in check. This is known as a *checkmate* and results in the conclusion of the game, with the checkmated player losing.

Interestingly enough, however, the majority of high-quality chess games does not end with *checkmates*, but instead ends with draws, often the result of a condition called a *stalemate*. A stalemate occurs when a player is currently not in *check*, but any legal move left available would result in him/her moving into *check*. There are other ways to end a chess game by a draw, but we won't go into them here, and you don't need to worry about them for your game (they're fairly

infrequent, but hey, they make for great extensions. If you want to read about them check out http://en.wikipedia.org/wiki/Draw_(chess)).

The Pieces

The Knight

This is what a knight looks like on a chessboard:

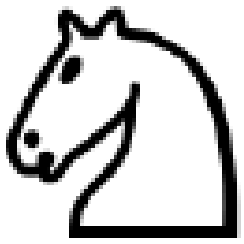


Figure 8. Image of a knight.

So what make a knight really interesting, and really easy, is how it moves. In particular the knight moves in L-shapes, moving first two squares in one direction and then one square in a perpendicular direction:

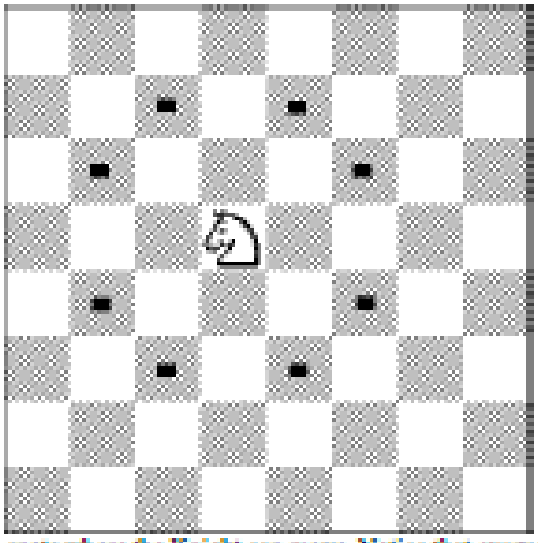


Figure 9. Diagram showing all of the spots where the Knight can move. Notice that every viable spot is two spaces away in one direction and then one space away in a perpendicular direction. Courtesy of www.thechesszone.com

The Knight is the only piece in chess that can jump over other pieces. What this means is, if you want to check to see if a given move with a Knight is valid, all you need to do is look at the state of the square the Knight is trying to move to.

The Queen

The Queen is considered the most powerful piece in chess. This is because of its extensive ability to move and capture pieces.

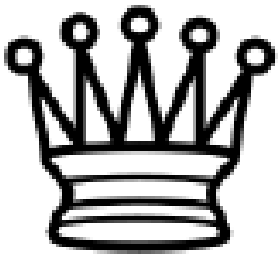


Figure 10. Image of a Queen. Like the King, it's depicted by a crown, but it's very different than a King. Don't get them confused.

A queen is allowed to move and capture enemy pieces along any straight line (horizontal, vertical, or diagonal) from the square that she currently occupies:

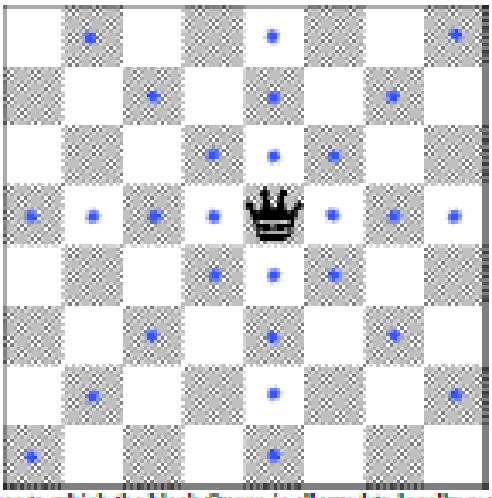


Figure 11. Diagram showing all squares to which the black Queen is allowed to legally move. Notice that every viable spot is on a straight line emanating out from the Queen's current location. Courtesy of www.thechesszone.com

The Bishop and Rook

Figure 12. Images of a Bishop (left) and Rook (right). The Rook is often called a Castle, but the official name is a Rook so that's what we're going to use.

Since we already talked about the Queen in a fair amount of depth, I'm going to go much faster through the Bishop and Rook. They can effectively be thought of as less-powerful variants on the Queen. In particular, the Bishop is a variant that can only move along diagonals, whereas the Rook is a variant that can only move along horizontal or vertical lines.

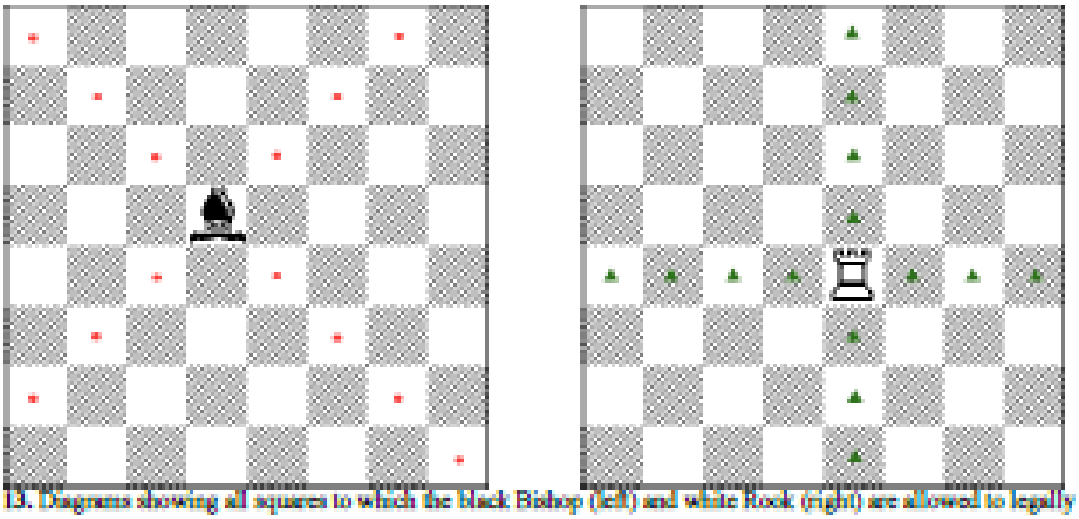


Figure 13. Diagrams showing all squares to which the black Bishop (left) and white Rook (right) are allowed to legally move. Notice that every viable spot is on a straight line emanating out from the Bishop or Rook's current location, but the straight lines allowed are different for the Bishop and Rook. Courtesy of www.thechesszone.com

The King

Figure 14. Image of a King.

Like the Knight, the King is also quite a simple piece. This is because its range of motion is quite limited. In particular, the King can only move one space, but it can move in any direction. To answer your question (if you haven't played Chess before), yes it is a little weird that the most important piece in the game is also one of the weakest as far as its ability to attack and move. That's just how it goes.

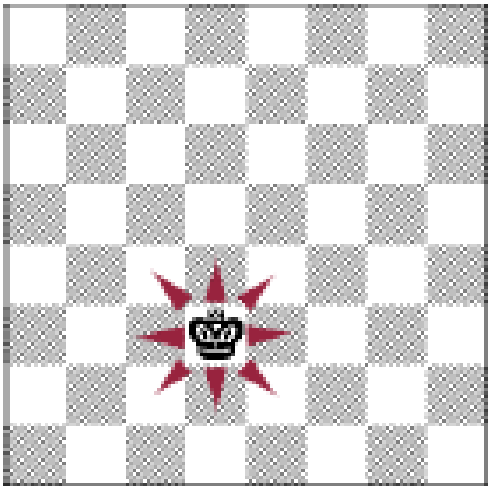


Figure 15. Diagram showing all the spaces where the black King can legally move. Courtesy of www.thechesszone.com

The Pawn

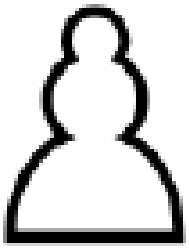


Figure 16 Image of a pawn. The picture of it is boring, but the way it moves is quite interesting.

Interestingly enough, the pawn, which is definitely the weakest piece in the game of Chess, is also the most complicated to get right. We'll go slowly to make sure everything's clear. The first interesting thing to say about pawns is that they only can move "forward." In this case forward is defined as towards the opponent's side of the board. In our orientation, this means that White pawns must always move up the screen, and Black pawns must always move down.

Now, ordinarily, pawns only move along a vertical line (always towards the opponent's side of the board) and they only move one space at a time. Except for two scenarios: a pawn's first move and capturing. For instance, each pawn is given the option of moving forward either one or two spaces on its first move (assuming that these paths lie unblocked by any pieces):

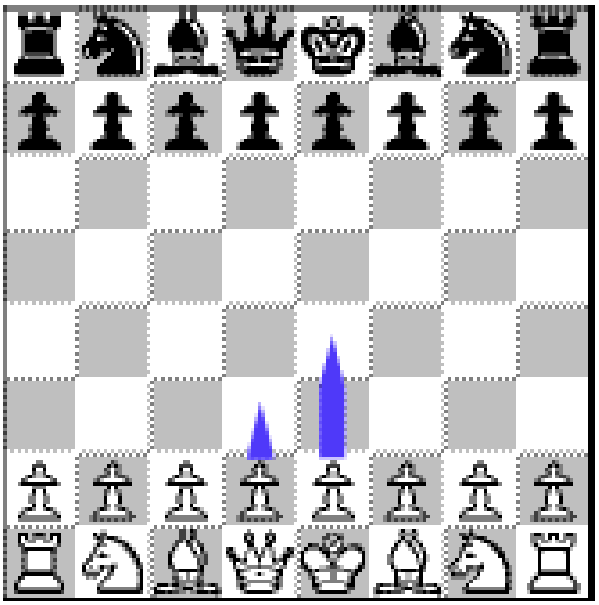


Figure 17. Diagram showing the unique movement of a pawn on its first move. Since the white Pawns shown have not been moved, they have the option of moving forward either one or two spaces (as indicated by the blue arrows) on their first move. This only applies if this move is not blocked by any piece, friend or enemy. Courtesy of www.thechesszone.com

Additionally, Pawns have another interesting wrinkle in that they capture differently than they move. In particular, while Pawns are only allowed to move along a vertical line forward, they capture on a diagonal line forward:

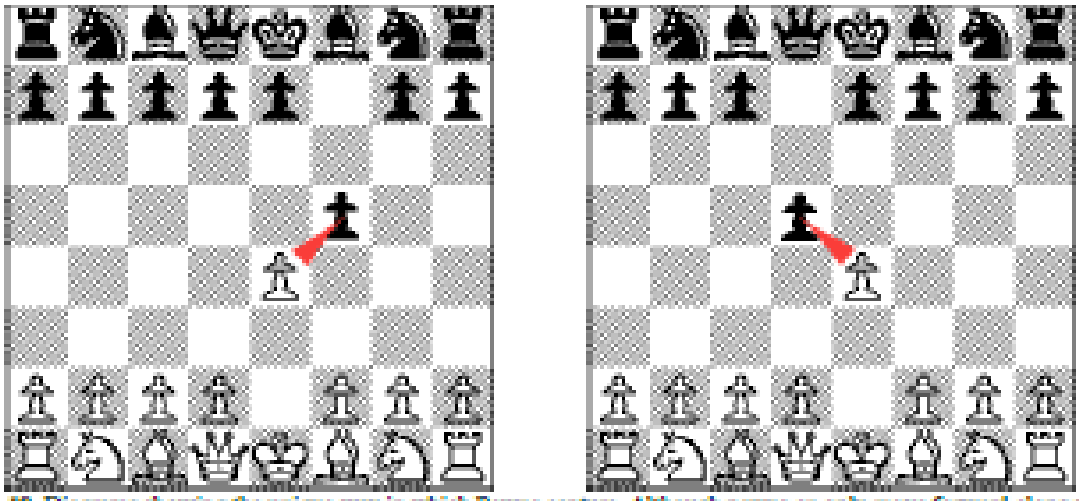


Figure 18. Diagrams showing the unique way in which Pawns capture. Although pawns an only move forward along vertical lines, they are only allowed to capture forward along diagonal lines.

This also means, however, that Pawns cannot capture directly forward:

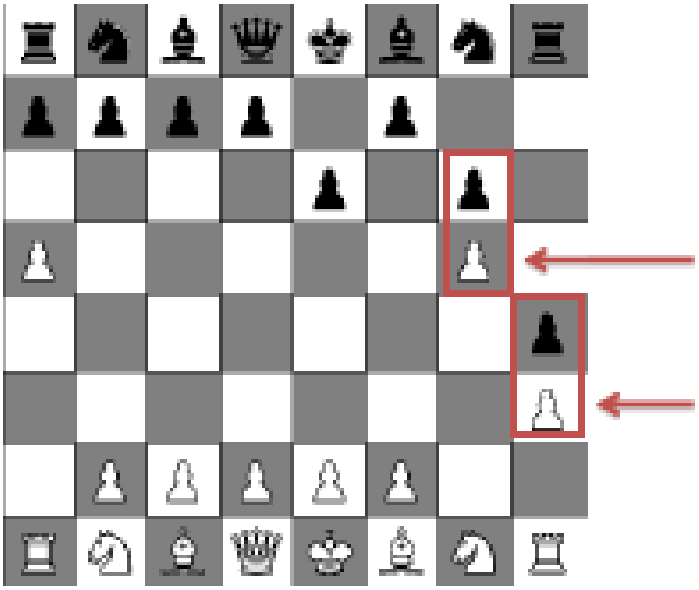


Figure 19. Diagram showing pawns who cannot capture each other (red boxes). Since pawns can only capture on diagonals, neither the black nor the white pawn in either of the red boxes can move in the current configuration.

Finally (and this you don't have to worry about), if a Pawn makes it all the way to the other side of the board, the player in command of the Pawn gets to replace that Pawn with their choice of a Knight, Bishop, Rook, or Queen (in this version of Chess is taken care of automatically for you – try it, it's kind of cool).