Question 3

1. Instruction

1) Create a Java Project named “**2110215_Mock03**”.

2) Copy all folders in “**Mock03_toStudent**” to your project directory src folder.

3) You are to implement the following classes

1)	Piece	(package piece)
2)	Bishop	(package piece)
3)	Knight	(package piece)
4)	Movement	(package util)

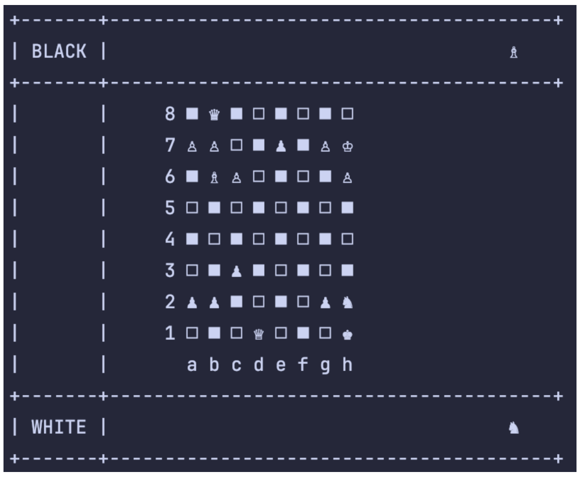
5) JUnit for testing is in package test.PieceTest

6) Save UML as **umlQ3.png** in root folder of your project.

7) Export jar file (**q3.jar**) that includes your source code and your .class files and put it in **root** folder of your project. YOU MUST EXPORT AND SUBMIT JAR FILE. IF YOU DO NOT, YOUR CODE WILL NOT BE MARKED.

2. Problem Statement : Chess Prototype

You are creating a chess prototype. The coordinates of the board (a1 to h8) are shown in the picture below.



```
+-------+-------------------------------------+
| BLACK |                                 ♗   |
+-------+-------------------------------------+
|       |     8 ■ ♛ ■ □ ■ □ ■ □
|       |     7 ♙ ♙ □ ■ ♟ ■ ♙ ♔
|       |     6 ■ ♗ ♙ □ ■ □ ■ ♙
|       |     5 □ ■ □ ■ □ ■ □ ■
|       |     4 ■ □ ■ □ ■ □ ■ □
|       |     3 □ ■ ♟ ■ □ ■ □ ■
|       |     2 ♟ ♟ ■ □ ■ □ ♟ ♞
|       |     1 □ ■ □ ♛ □ ■ □ ♚
|       |       a b c d e f g h
+-------+-------------------------------------+
| WHITE |                                 ♞   |
+-------+-------------------------------------+
```

The prototype consists of 3 types of pieces, a board, and a class that defines movements for each piece. You won’t be able to play the game, but you can test each piece’s legal moves in the JUnit test cases.

3. Implementation Details

3.1) Package Position

3.1.1) Class Position

/* ALREADY PROVIDED */

This class represents a position on a 2D grid, commonly used in games or simulations where positions are identified by a row and a column.

Fields

Name	Description
- int row	Stores the row index of the position. It is a zero-based index, meaning the first row is represented as 0.
- int col	Stores the column index of the position. It is a zero-based index, meaning the first column is represented as 0.

Constructors

Name	Description
+ Position(int row, int col)	Methods to initialize the row and col fields.
+ Position(String s)	From a string representation in the format of [a-h][1-8] (first character is a letter, second character is a number). In which 's' is strictly string with 2 characters length. Example: new Position("e4") will create a position that represents e4 on the board.

Methods

Name	Description
+ boolean equals(Object object)	Checks if this Position object is equal to another object.
+ int hashCode()	Returns a hash code value for the Position object.
+ String toString()	Converts the Position object to a string representation. Converts the column index to a letter and the row index to a digit.
+ Appropriate Getter and Setter	

3.2) Package Piece

3.2.1) Class Piece

You must implement this class from scratch.

The Piece class represents a generic base class for game pieces on a board. It provides common functionality and attributes for all game pieces. It also requires implementation of specific movement rules and copy behavior in subclasses.

Field

Name	Description
# boolean white	Indicates whether the piece is white. It is used to distinguish between white and non-white pieces.
# boolean moved	Tracks whether the piece has been moved from its initial position. This helps determine the piece's movement status. Initially the value of this field should be “false”.
# Position position	Represents the current position of the piece on the board.
# Board board	Refers to the board on which the piece is placed. It connects the piece to the game board, enabling interaction with other pieces and the game environment.

Constructors

Name	Description
+ Piece(boolean white, Position position, Board board)	Assign appropriate parameters to appropriate fields. Then, call “board.placePiece(Piece, Position)” to place the piece on the board.

Method

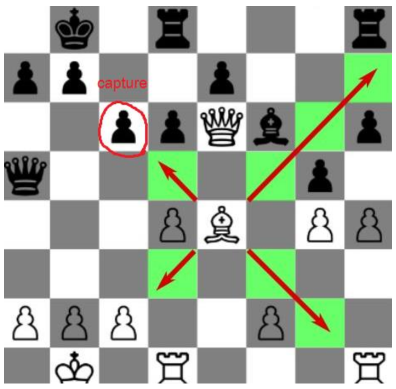
Name	Description
+ *Set<Position> getLegalMove()*	method to return a set of legal moves for **this** piece. Note: *Set* is used to prevent duplication of Positions. This method returns a different set, depending on the actual type of piece that calls this method. However, this method can be implemented here because we make use of polymorphism in class Movement. To implement this method: • Create a Movement object, with the same **position** and **board** as **this**. • Call getMovePositions(this) on the Movement object. This method updates the Movement object’s possible move positions. • Return the result from calling getMoves() on the Movement object.
+ *Object deepcopy()*	Create a deep copy of the piece and return it. This method returns a different object depending on the actual object that it copies. The method is too generic to be implemented

	here. But we need all Pieces to have this method.
+ String toString()	For the debugging process, return a string in a format of the subclass name. In this format “classname” + (“position”) Hint: you can get the subclass name by using method “getClass().getSimpleName()” Note: this class will not be tested on the JUnit.
+ void moveHandle(Position to)	Move the piece to a given position. This method calls hadMoved() to update the moved status and then sets the field position to the given position.
+ boolean hadMoved()	This method updates the moved field to true.
+ boolean equals(Object o)	Test whether **this** has the same contents as **o**. The fields that are objects must be compared using method **equals**. This is very important because it can be used to test whether the deepCopy() actually produces a new object.
+ “is” Method for all fields with boolean value	
+ Appropriate Getter and Setter Fields	

3.2.2) Class Bishop

You must implement this class from scratch.

The Bishop class represents a bishop piece. This piece moves diagonally. It cannot move beyond the piece it captures or the piece of the same faction (cannot lie on top of a piece of the same faction). The movement is handled by class Movement.



Constructors

Name	Description
+ Bishop(boolean isWhite, Position pos, Board board)	Assign variables to appropriate fields

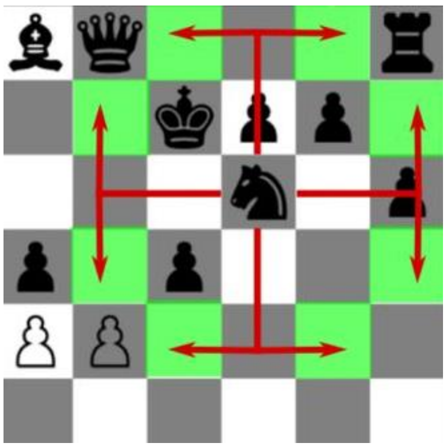
Methods

Name	Description
+ Object deepCopy()	Creates a new Bishop object with the same color, position, and board. If **this** bishop had been moved, the copy also updates the moved status. Return the new Bishop instance.

3.2.3) Class Knight

You must implement this class from scratch.

The Knight class represents a knight piece. This piece moves in L-shape. It cannot lie on top of a piece of the same faction. The movement is handled by class Movement.



Constructors

Name	Description
+ Knight(boolean isWhite, Position pos, Board board)	Assign variables to appropriate fields

Methods

Name	Description
+ Object deepCopy()	Creates a new Knight object with the same color, position, and board. If **this** knight had been moved, the copy also updates the moved status. Return the new knight instance.

3.3) Package Movement

3.3.1) Class Movement

You must fill-in a method in this class.

Field

Name	Description
- Position current	The current position from which moves are being calculated.
- Set<Position> moves	A set of potential moves calculated for the piece.
- Board board	Refers to the board on which the piece is currently placed. It connects the piece to the game board, enabling interaction with other pieces and the game environment.

Constructors

Name	Description
+ Movement(Position current, Board board)	Assign variables to appropriate fields

Methods

Name	Description
+ void lShapedMove()	Update the set “moves” to contain possible positions for a Knight move. A knight cannot move to the square occupied by the same side.
+ void diagonalMove()	Update the set “moves” to contain possible positions for a Bishop move. A bishop cannot move to/beyond the square occupied by the same side. Also, a bishop cannot move further than its first capture in a direction.
+ **void** getMovePositions(Piece p)	**You must fill in this method.** This method updates the possible moves set. It can be called with any type of Piece. If p is a Knight, execute lShapeMove(). else if p is a Bishop, execute diagonalMove() otherwise, do nothing.

4. Scoring Criteria (15, will be scaled down to 10)

4.1) JUnit Test

Class Bishop	**5 mark(s)**	
- bishopConstructorTest	2	mark(s)

-	bishopMoveStartingPositionTest	0.1	mark(s)
-	bishopAtCenterTest	0.1	mark(s)
-	bishopCaptureTest	0.2	mark(s)
-	bishopCantCaptureTest	0.2	mark(s)
-	bishopDiagonalFullDiagonalMovementTest	0.2	mark(s)
-	bishopEdgeCaseTest	0.2	mark(s)
-	deepCopyTest	2	mark(s)

Class Knight **5 mark(s)**

-	knightConstructorWithBoardTest	2	mark(s)
-	knightMoveStartingPositonTest	0.1	mark(s)
-	knightAtCornerMove	0.1	mark(s)
-	knightMoveObstructed	0.1	mark(s)
-	knightMoveAtBoardCenter	0.2	mark(s)
-	knightCapturableOpposePiece	0.2	mark(s)
-	knightCapturableOpposeKnight	0.1	mark(s)
-	knightCantMovByOwnPieces	0.1	mark(s)
-	knightCantMoveByOpposePieces	0.1	mark(s)
-	deepCopyTest	2	mark(s)

4.2) UML Diagram

UML of Piece, Bishop, Knight, and Movement class **(png file only -> umlQ3.png)** **5 mark(s)**

- all these Classes must be shown
- class details must be correct

The project must contain .jar file, **OTHERWISE THIS QUESTION WILL NOT BE GRADED**

- q3**.jar**
- the .jar file must have the source code inside
- the .jar file must have the .class files