CS102A SPRING 22 ASSIGNMENT 5 CHESS

> Contributors:
>
> Leader and Overall Design: ZHU Yueming
>
> Source code: YAN Jiaqin
>
> Junit Design: LI Boyan, XIA Qiqi
>
> Document: DU Penghui
>
> Tester: GUO Denghao
>
> Publisher: WANG Ziming

DESCRIPTION

In this assignment, you need to implement a chess game as following description.

Hint: **You must implement all the given classes and methods.** Beyond that, you can design any classes and methods you need for assistance. **Utilize your creativity** to finish this game successfully! O(∩_∩)O

Click Here for a brief overview on chess rules.

QUESTIONS AND CLASSES INTRODUCTION

Question 1 ChessboardPoint (10 points)

ChessColor.java

```
public enum ChessColor {
    BLACK, WHITE, NONE;
}
```

ChessboardPoint.java

- This class helps to maintain chess locations in the game.
- **Variables to Define: x and y for recording chess location:**

```
 private int x; // x: Horizontal location of this chess
 private int y; // y: Vertical location of this chess 纵向
/*
For an example, look at the chessboard below
(Just an example for better understanding, not input format)

x\y 0 1 2 3 4 5 6 7
 0  R N B Q K B N R
 1  P P P P P P P P
 2  _ _ _ _ _ _ _ _
 3  _ _ _ _ _ _ _ _
 4  _ _ _ _ _ _ _ _
 5  _ _ _ _ _ _ _ _
 6  p p p p p p p p
 7  r n b q k b n r
*/
```

- **Method to Implement: constructor**

```
public ChessboardPoint(int x, int y)
```

  Setting the parameter value to private fields respectively.

- **Method to Implement: Two getters**

```
public int getX()
public int getY()
```

- **Method to Implement: toString.**

- For an instance, if you have a chess at position (4,2), your toString method is supposed to give a string "(4,2)" (without blank space)

```
public String toString()
```

- **Method to Implement: offset.**
  - You are given the coordinate change in x and y directions respectively as dx and dy. What you need to do is **Return a new object of ChessboardPoint type** to represent the new coordinate after the move. **If this leads a point out of chessboard, return null.**

```
public ChessboardPoint offset(int dx, int dy)
```

Hints: Do not modify or remove any methods or fields that have been already defined. You can add other methods or fileds that you think are necessary

Remind: For question 1, submit class "ChessColor", "ChessboardPoint", and other classes you used for assistance.

Question 2 ChessComponent (10 points)

ChessComponent.java

- This class is an abstract class
- This class helps to represent all the chess components on the chessboard.
- This class provides a rough description for all the chess pieces. Abstract methods should be left in abstract form for further implementation, **you don't need to write codes for them in detail**.
- **Variables to Define:**

```
private ChessboardPoint source; // Where the chess is
private ChessColor chessColor;  // What's the color
protected char name;      // What's the name
```

- **Method to Implement: constructor**

```
public ChessComponent()
```

An none parameter constructor should be defined, even though we do not care whether there is any statement in it.

- **Method to Implement: canMoveTo**
  - This abstract method tells **where this chess piece can move to**.
  - If **no** ChessboardPoint can be moved to, return an **reference of empty List** instead of null.

```
public abstract List<ChessboardPoint> canMoveTo();
```

- **Method to Implement: toString**
- Return the name of current chess piece.

```
@Override
public String toString() {
    return String.valueOf(this.name);
}
```

Hints: Do not modify or remove any methods or fields that have been already defined. You can add other methods or fileds that you think are necessary

Remind: For question 2, submit class "ChessColor", "ChessboardPoint", "ChessComponent", and other classes you used for assistance.

Question 3 ChessGame1(30 points)

- Before you do this task, you **need to declare all concrete chesses**. All concrete chess component that inherit the super class `ChessComponent` , including:

```
KingChessComponent
QueenChessComponent
RookChessComponent
BishopChessComponent
KnightChessComponent
PawnChessComponent
EmptySlotComponent
```

ChessGame.java

- This class provides a rough description for running a chess game. Abstract methods should be left in abstract form for further implementation, you don't need to write codes for them in detail.
- **Method to Implement: loadChessGame**
    - This abstract method loads chess game from given chessboard.

```
void loadChessGame(List<String> chessboard);
```

- **Method to Implement: getCurrentPlayer**
    - This abstract method returns the current player.

```
ChessColor getCurrentPlayer();
```

- **Method to Implement: getChess**
    - This abstract method returns the ChessComponent object in the given position.

```
ChessComponent getChess(int x, int y);
```

- **Method to Implement: getChessboardGraph**

- This abstract method returns the chessboard status.

```
String getChessboardGraph();
```

- **Method to Implement: getCapturedChess**
  - This abstract method returns all the chess pieces that are already captured.

```
public String getCapturedChess(ChessColor player);
```

ConcreteChessGame.java

- After reading the description for "ChessGame", you may have already understood the method structure of a chess game. Now you need to create a class named "ConcreteChessGame" and implement all the previously written abstract methods in detail.
- **Fields in ConcreteChessGame:**

```
// A 2-dimension array to store all the chess components
// should be initialized in your construct method.
// i.e. = new ChessComponent[8][8]
private ChessComponent[][] chessComponents;
// What's the current player's color, black or white?
// should be initialized in your construct method.
// by default, set the color to white.
private ChessColor currentPlayer;
```

- **Method to Implement: loadChessGame**

```
void loadChessGame(List<String> chessboard);
```

  - In this method, you need to load chess game from given chessboard. Here is an example of input chessboard:
  - R/r=rooks, N/n=knights, B/b=bishops, Q/q=queen, K/k=king, P/p=pawns, _=nothing. The upper case letter indicates black chess pieces, while the lower case letter means white chess pieces. "w" means the current player is white, while "b" means the current player is black.

```
# Sample parameter (List<String>):

RNBQKBNR
PPPPPPPP
________
________
________
________
pppppppp
rnbqkbnr
w
```

```
In another word:

List<String> chessboard;
chessboard.get(0) equals to "RNBQKBNR";
chessboard.get(1) equals to "PPPPPPPP";
chessboard.get(2) equals to "________";
chessboard.get(3) equals to "________";
chessboard.get(4) equals to "________";
chessboard.get(5) equals to "________";
chessboard.get(6) equals to "pppppppp";
chessboard.get(7) equals to "rnbqkbnr";
chessboard.get(8) equals to "w";
```

- **Method to Implement: getCurrentPlayer**

```
@Override
public ChessColor getCurrentPlayer() {
    return this.currentPlayer;
}
```

- **Method to Implement: getChessboardGraph**

```
public String getChessboardGraph()
```

  - This method returns the chessboard status. You should return a 8-rows String in the same format with input chessboard. Use `\n` to separate 2 rows. For an example:

```
# Sample return value (String):

RNBQKBNR
PPPPPPPP
________
________
________
________
pppppppp
rnbqkbnr

In another word your return String would be:

"RNBQKBNR\nPPPPPPPP\n________\n________\n________\n________\npppppppp\nrnbqkbnr
"
```

- **Method to Implement: getCapturedChess**

```
public String getCapturedChess(ChessColor player);
```

  - This method returns all the captured chess pieces. pieces should be **sort by the order** of "King>Queen>Rooks>Bishops>Knights>Pawns".

- At the begining of a game, the original count of those chess pieces of King, Queen, Rooks, Bishops, Knights, Pawns are 1, 1, 2, 2, 2, 8 respectively, so that the captured chesses are those whose have been lost from chessboard.

```
For an example, Black player(uppercase) lost 1 queen, 2 rooks;

Q 1
R 2

In another word:

String str = "Q 1\nR 2\n"

For an another example, White player(lowercase) lost 4 pawns, then you should
output:
p 4

In another word:

String str = "p 4\n"
```

- **Method to Implement: getChess**

```
public ChessComponent getChess(int x, int y)
```

This method returns the concrete chess component from `private ChessComponent[][] chessComponents;` in (x, y) location.

Hints: Do not modify or remove any methods or fields that have been already defined. You can add other methods or fileds that you think are necessary

Remind: For question 3, submit class "ChessColor", "ChessboardPoint", "ChessComponent", "ChessGame" and "ConcreteChessGame", and any other classes you used for assistance.

In this question, we suggest submit all subclasses of ChessComponent.

Question 4 ChessGame2 (50 points)

- Before you do this task, You should implement the abstract method `public abstract List<ChessboardPoint> canMoveTo();` and return all can move points according to the rules in concrete chess components respectively, which is to make different chess pieces moves in different ways.
- If **no** ChessboardPoint can be moved to, return an **reference of empty List** instead of null.

  For example: `EmptySlotComponent` is one of the subclasses of `ChessComponent`, which has no point to move , so that the implement of `canMoveTo` method in `EmptySlotComponent` would be

```
@Override
    public List<ChessboardPoint> canMoveTo() {
        return new ArrayList<>();
    }
```

- You also may want to **check out chess rules** mentioned at the beginning. Click Here for a brief overview on chess rules.

  To simplify this question, you can **ignore** those four special moves :

  - Capture the passing way Pawn. (吃过路卒)
  - Swap position if King and Rook. (王车易位)
  - Pawn promote to Queen, Rook, Bishop or Knight. (卒变后、车、相、马)
  - King cannot be defeated, and king cannot capture king （王不能被将，王不能吃王）

ChessGame.java

- **Method to Implement: moveChess**
  - This abstract method returns whether a chess piece at source can move to target. Detailed method is shown as follows.

```
boolean moveChess(int sourceX, int sourceY, int targetX, int targetY);
```

- **Method to Implement: getCanMovePoints**
  - This abstract method returns all the points that chess piece at "source" point can move to.

```
List<ChessboardPoint> getCanMovePoints(ChessboardPoint source);
```

ConcreteChessGame.java

All concrete move rules should be designed in concrete chess components，so that you should finish those two methods by using the theory of **polymorphism**.

More specifically, in following two methods, do not write down the similar code as follows:

```
if(chess instanceof KingChessComponent)
else if (chess instanceof QueenChessComponent)
else if (chess instanceof RookChessComponent)
......
```

Similar code as below is recommended to design. `chess` is an reference of `ChessComponent` and instantiated by its subclasses, so that the subclass invokes the `canMoveTo()` method is determined to the subclass instantiate the `chess`.

- **Method to Implement: moveChess**

```
public boolean moveChess(int sourceX, int sourceY, int targetX, int
targetY);
```

    - This method tells whether a chess at source position can moves to target position. can = true, can't = false. If the chess can be moved, then move it and switch the player.
    - Of course, the current player is not moving another player's pieces.
- Empty component has no where to move , so that it returns an empty **List** (which can be instantiated by ArrayList) instead of null.

Introcude of the method:

1. Check out whether the current ChessCompent (sourceX, sourceY) match the current player.
2. If the movement is legal, then move it, switch the chess, and return true.
    1. Remember to change the location of two chesses (source and target).
    2. Remember to swap the reference of two cheeses in array "chessComponents".
    3. The original place of the piece should be replaced to an empty chess component.
3. If the movement is illegal, then return false, and do not change anything.

- **Method to Implement: getCanMovePoints**

```
public List<ChessboardPoint> getCanMovePoints(ChessboardPoint source)
```

    - This method returns the points that a chess at source point can move to. **You are supposed to sort all the points in x-y ascending order. for an example: (1, 1) < (1, 3) < (2, 1)**
    - If no places to go, return an empty **List** (which can be instantiated by ArrayList) instead of null .
    - The structure of this method should be:

```
 public List<ChessboardPoint> getCanMovePoints(ChessboardPoint
source) {
        // 1. find chess according to source
        // 2. as below statement:
        List<ChessboardPoint> canMovePoints = chess.canMoveTo();
        // 3.sort canMovePoints by x - y ascending order
        return canMovePoints;
    }
```

      in this case, to calculate which points can be move to conveniently, you'd better pass the reference of `private ChessComponent[][] chessComponents;` to each subclass of `ChessComponent`

Extra hint for question 4:

- Do not modify or remove any methods or fields that have been already defined. You can add other methods or fileds that you think are necessary
- This can be a big assignment(but not so hard). You will find it helpful to code in elegant way, and

name every class & method & variable normatively.
- If you begin copy-and-paste your own code, you may be on a highway to bugs. You will be thankful to yourself if you can now use classes, methods and loops to make your assignment5 simple and clean.

Hints: Do not modify or remove any methods or fields that have been already defined. You can add other methods or fileds that you think are necessary

Remind: For question 4, submit class "ChessColor", "ChessboardPoint", "ChessComponent", "ChessGame" and "ConcreteChessGame", and any other classes you used for assistance.