

**2020年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 郭茁宁 |
| 学号 | 1183710109 |
| 班号 | 1837101 |
| 电子邮件 | gzn00417@foxmail.com |
| 手机号码 | 13905082373 |

# 目录

[0 目录 2](#_Toc35975544)

[1 实验目标概述 1](#_Toc35975545)

[2 实验环境配置 2](#_Toc35975546)

[2.1 安装EclEmma 2](#_Toc35975547)

[2.2 GitHub Lab2仓库的URL地址 3](#_Toc35975548)

[3 实验过程 4](#_Toc35975549)

[3.1 Poetic Walks 4](#_Toc35975550)

[3.1.1 Get the code and prepare Git repository 4](#_Toc35975551)

[3.1.2 Problem 1: Test Graph <String> 4](#_Toc35975552)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc35975553)

[3.1.3.1 Implement ConcreteEdgesGraph 5](#_Toc35975554)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc35975555)

[3.1.4 Problem 3: Implement generic Graph<L> 6](#_Toc35975556)

[3.1.4.1 Make the implementations generic 6](#_Toc35975557)

[3.1.4.2 Implement Graph.empty() 7](#_Toc35975558)

[3.1.5 Problem 4: Poetic walks 7](#_Toc35975559)

[3.1.5.1 Test GraphPoet 7](#_Toc35975560)

[3.1.5.2 Implement GraphPoet 7](#_Toc35975561)

[3.1.5.3 Graph poetry slam 7](#_Toc35975562)

[3.1.6 Before you’re done 8](#_Toc35975563)

[3.2 Re-implement the Social Network in Lab1 9](#_Toc35975564)

[3.2.1 FriendshipGraph类 9](#_Toc35975565)

[3.2.2 Person类 10](#_Toc35975566)

[3.2.3 客户端main() 10](#_Toc35975567)

[3.2.4 测试用例 11](#_Toc35975568)

[3.2.4.1 简单图测试 11](#_Toc35975569)

[3.2.4.2 复杂图测试 12](#_Toc35975570)

[3.2.4.3 Junit测试结果 12](#_Toc35975571)

[3.2.5 提交至Git仓库 13](#_Toc35975572)

[3.3 Playing Chess 14](#_Toc35975573)

[3.3.1 ADT设计/实现方案 15](#_Toc35975574)

[3.3.1.1 interface Game 15](#_Toc35975575)

[3.3.1.1.1 class chessGame 18](#_Toc35975576)

[3.3.1.1.2 class goGame 19](#_Toc35975577)

[3.3.1.2 class Board 20](#_Toc35975578)

[3.3.1.3 class Player 23](#_Toc35975579)

[3.3.1.4 class Position 25](#_Toc35975580)

[3.3.1.5 class Piece 26](#_Toc35975581)

[3.3.1.6 interface Action 27](#_Toc35975582)

[3.3.1.6.1 class chessAction 29](#_Toc35975583)

[3.3.1.6.2 class goAction 30](#_Toc35975584)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 31](#_Toc35975585)

[3.3.2.1 游戏流程图 31](#_Toc35975586)

[3.3.2.2 程序演示 32](#_Toc35975587)

[3.3.2.3 功能实现 34](#_Toc35975588)

[3.3.3 ADT和主程序的测试方案 37](#_Toc35975589)

[3.3.3.1 testInit() 37](#_Toc35975590)

[3.3.3.2 testPiece() 37](#_Toc35975591)

[3.3.3.3 testBoard 38](#_Toc35975592)

[3.3.3.4 testPosition() 38](#_Toc35975593)

[3.3.3.5 testPlayer() 39](#_Toc35975594)

[3.3.3.6 testPut() 40](#_Toc35975595)

[3.3.3.7 testMove() 40](#_Toc35975596)

[3.3.3.8 testCaptureAndPut() 41](#_Toc35975597)

[3.3.3.9 ChessTest测试结果 42](#_Toc35975598)

[3.3.3.10 GoTest测试结果 42](#_Toc35975599)

[4 实验进度记录 43](#_Toc35975600)

[5 实验过程中遇到的困难与解决途径 44](#_Toc35975601)

[6 实验过程中收获的经验、教训、感想 45](#_Toc35975602)

[6.1 实验过程中收获的经验和教训 45](#_Toc35975603)

[6.2 针对以下方面的感受 45](#_Toc35975604)

# 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现ADT。具体来说：

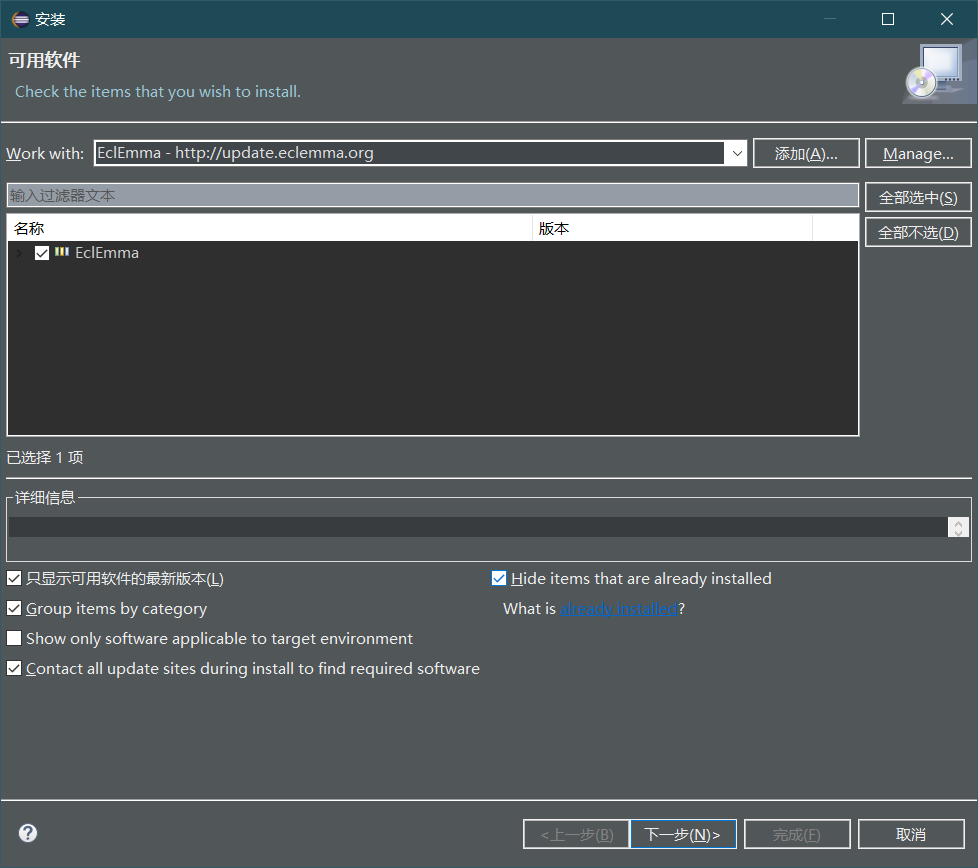
* 针对给定的应用问题，从问题描述中识别所需的ADT；
* 设计ADT规约（pre-condition、post-condition）并评估规约的质量；
* 根据ADT的规约设计测试用例；
* ADT的泛型化；
* 根据规约设计ADT的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
* 使用OOP实现ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
* 测试ADT的实现并评估测试的覆盖度；
* 使用ADT及其实现，为应用问题开发程序；
* 在测试代码中，能够写出testing strategy并据此设计测试用例。

# 实验环境配置

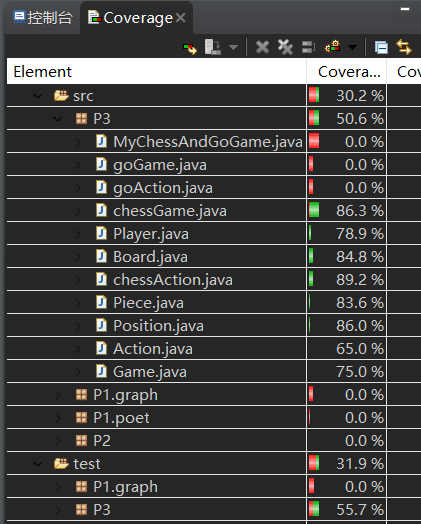
## 安装EclEmma

依据<https://www.eclemma.org/installation.html>内容，从更新站点进行安装。

* 从Eclipse菜单中选择帮助 → 安装新软件；
* 在“安装”对话框中，在“ 工作日期”字段中输入http://update.eclemma.org/；



* 检查最新的EclEmma版本，然后按“下一步”；
* 重启eclipse，即可在java的透视图工具栏中找到coverage启动器，表示安装成功。
* 使用效果



## GitHub Lab2仓库的URL地址

<https://github.com/ComputerScienceHIT/Lab2-1183710109>

# 实验过程

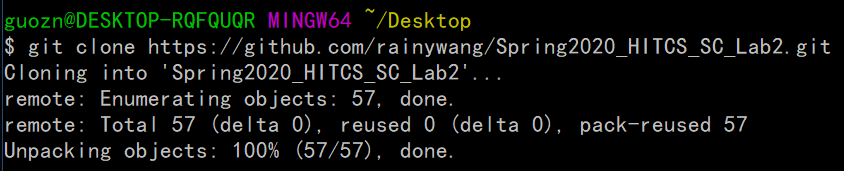
## Poetic Walks

该任务主要是实验一个图的模块，并基于此使用。

* 完善Graph接口类，并运用泛型的思想，将String拓展为泛型L类；
* 实现Graph类的方法：add、set、remove、vertices、sources、targets；
* 利用实现的Graph类，应用图的思想，实现GraphPoet类，如果输入的文本的两个单词之间存在桥接词，则插入该桥接词；若存在多个单一桥接词，则选取边权重较大者。

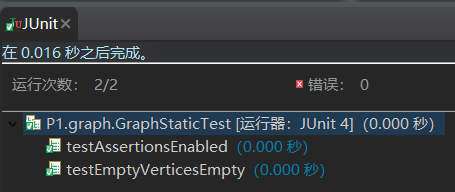
### Get the code and prepare Git repository

git clone https://github.com/rainywang/Spring2020\_HITCS\_SC\_Lab2.git



### Problem 1: Test Graph <String>

测试静态方法生成String类型的Graph。



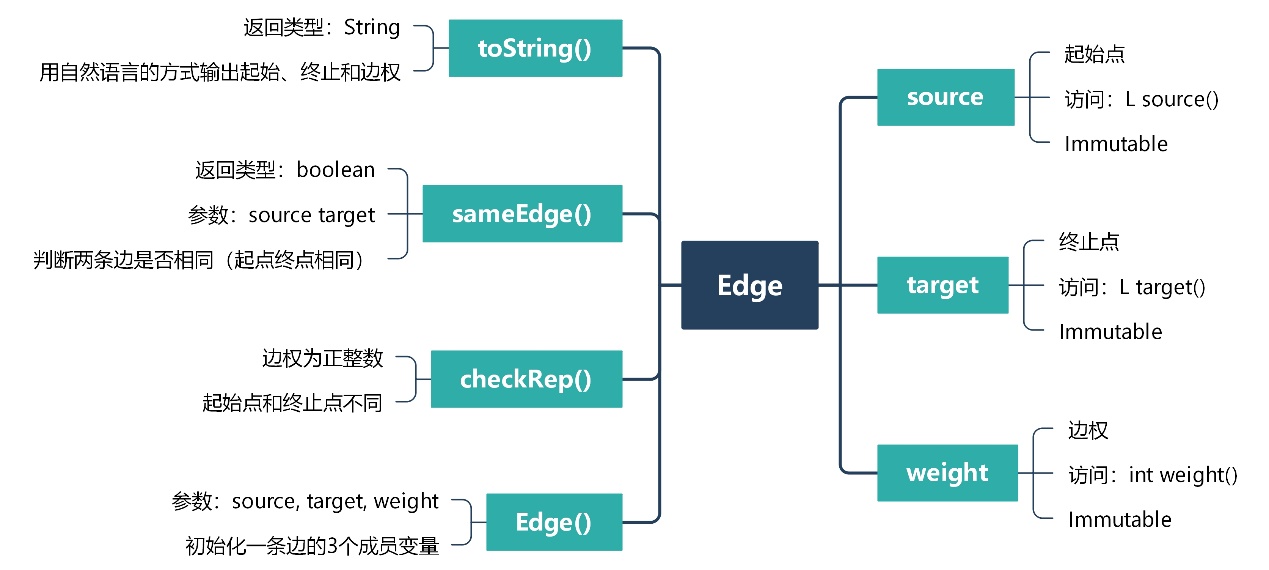
### Problem 2: Implement Graph <String>

该部分要求重写Graph里的方法，分别以点为基础的图和以边为基础的图。两种类型的图都经过同一个实例测试。

#### Implement ConcreteEdgesGraph

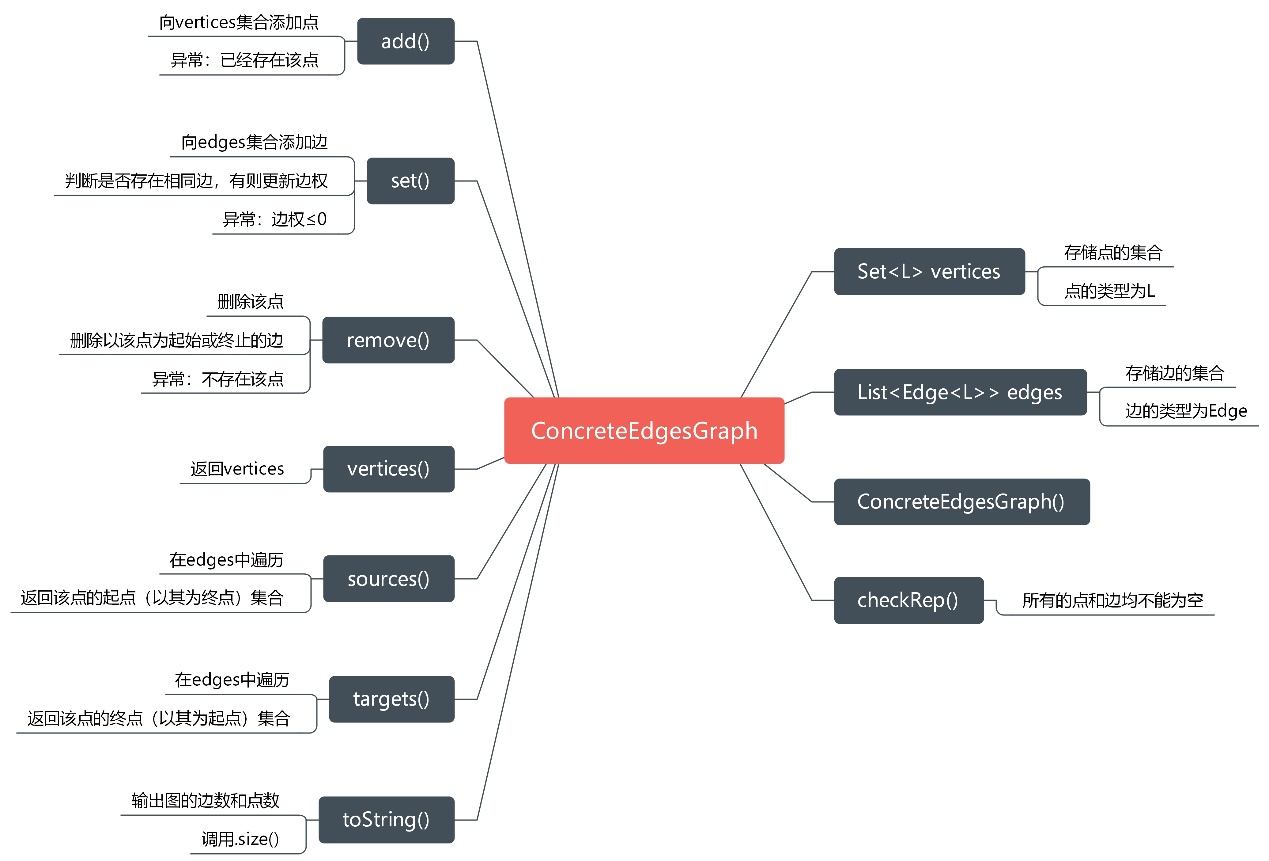
Edge实现

Edge的功能主要为存储边的3个信息。此外，为了Graph实现方便，增加了判断两条边是否相等的方法。



ConcreteEdgesGraph实现

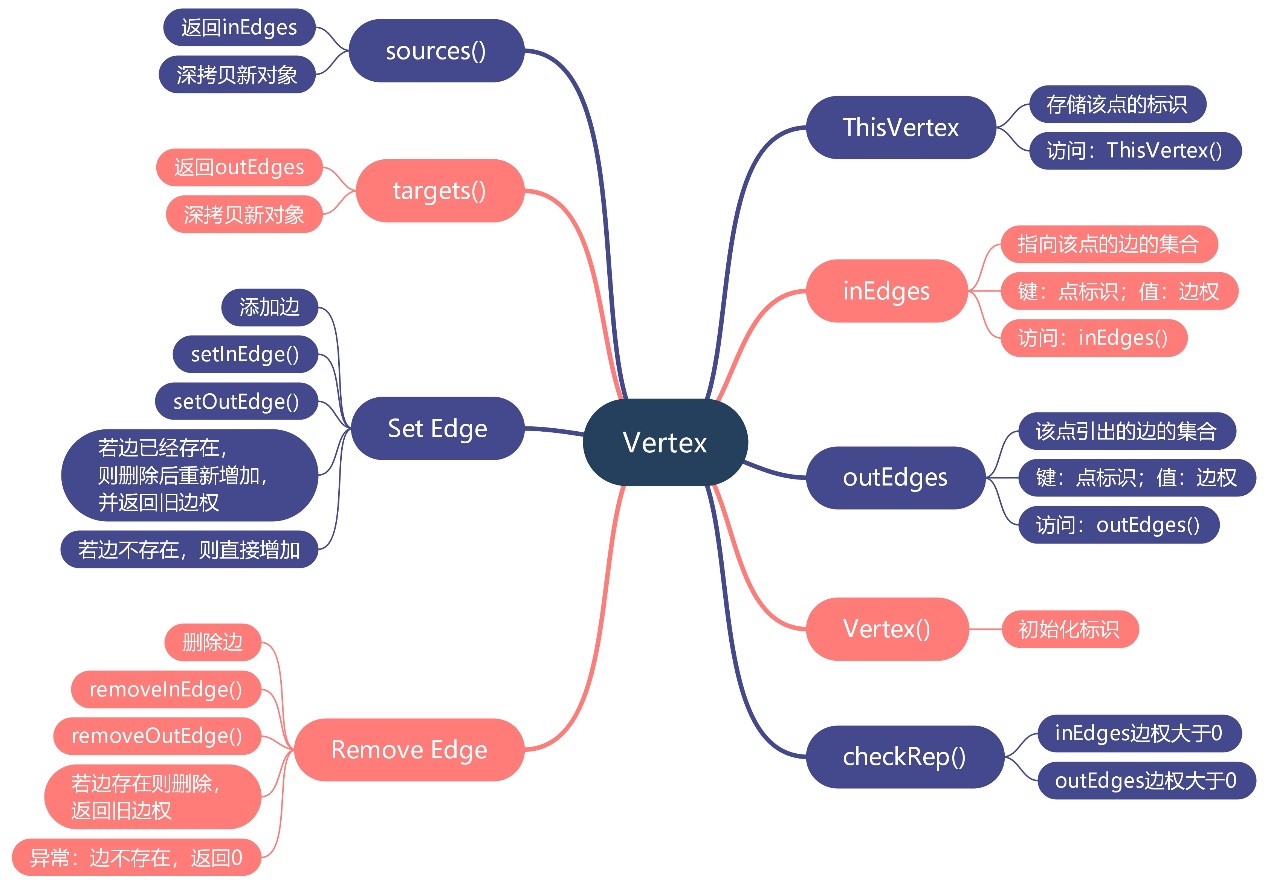
该类以Edge为基础重写Graph<L>，用集合来存储点和边（Edge），每有Edge的增加就会影响到集合的更改，而点的删除也需要在集合中查询匹配。



#### Implement ConcreteVerticesGraph

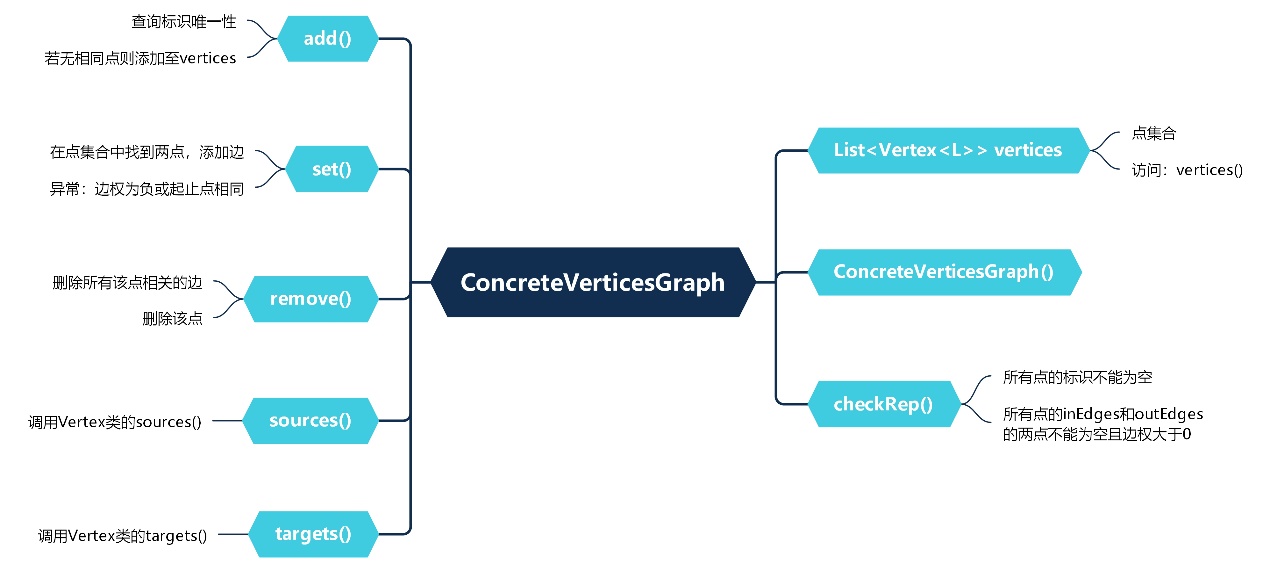
Vertex实现

Vertex是点的抽象类，包含3个信息：点的标识、指向该点的边、由该点引出的边。Vertex需要能访问这3个信息，以及增加/删除进边/出边。



ConcreteVerticesGraph实现

ConcreteVerticesGraph是以点为基础的图，每个点通过唯一的标识进行区分，set和remove都依赖与Vertex类中的添加和删除操作，sources和targets也调用了Vertex类的方法。



### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

在程序中选择“重构”或选择“String”并选择更改所有匹配项（要注意toString），即可实现泛化类型。

#### Implement Graph.empty()

使Graph.empty()能返回一个新的空实例。代码如下：

    public static Graph<String> empty() {

        return new ConcreteEdgesGraph();

    }

### Problem 4: Poetic walks

问题简述：

给定一个语料库corpus，根据corpus中的文本生成一个单词图，然后给定一条语句输入，在图中搜索词之间的关系，自动补全语句中可能可以完善的部分。

图的构建规则是，在corpus中，对每一个不一样的单词看作一个顶点，相邻的单词之间，建立一条有向边，相邻单词对出现的次数，作为这条有向边的权值。在输入信息补全时，对相邻单词A和B做检查，如果存在一个单词C，在图中可以由前一个单词A通过这个单词C到达单词B，那么就在A和B之间补全C，补全的优先级按照权值越大者优先。

#### Test GraphPoet

在基于预设的测试用例基础上，增加等价类划分的多种情况。

等价类划分：两个单词之间不存在连接词，两个单词之间只有一个连接词，两个单词之间有多个连接词。

此外还要注意句末的句号，测试当一个句子最后一个词是“桥”的一端。

#### Implement GraphPoet

1. 表示不变量和检查不变量

该应用中的不变量是所有的点都不为空。

1. 构造函数

用文件输入单词，String.split()分割为数组，通过String.toLowerCase()小写化。

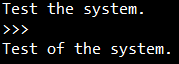
接下来构建图，相邻的单词加边。首先要在加边前通过Graph.add()加点，加边时要判断是否存在：由于Graph.set()能返回之前加的边的值，以此来判断是否存在，存在则在之前的值加一（之前的边的值保存为lastEdgeWeight）。

1. Poem(String input)

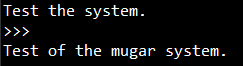
当相邻两个单词任意一个不在之前创建的图里，则将后者单词加入即可（再加个空格）当存在时，由于Bridge长度只能为2，所以：分别求两个单词的sources和targets，将该Map转换为Set求交集；若交集为空，则无桥，若交集不空，则在交集中找最短的桥（可以在Map的value中查询weight）。

#### Graph poetry slam

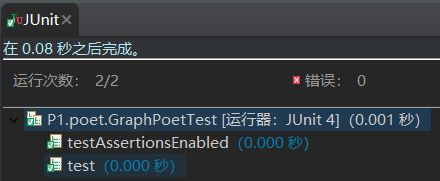
样例“This is a test of the Mugar Omni Theater sound system.”进行测试，测试成功。



修改样例为“This is a the Mugar system Omni Theater sound system test of the.”，测试成功。该样例用于测试极端情况。



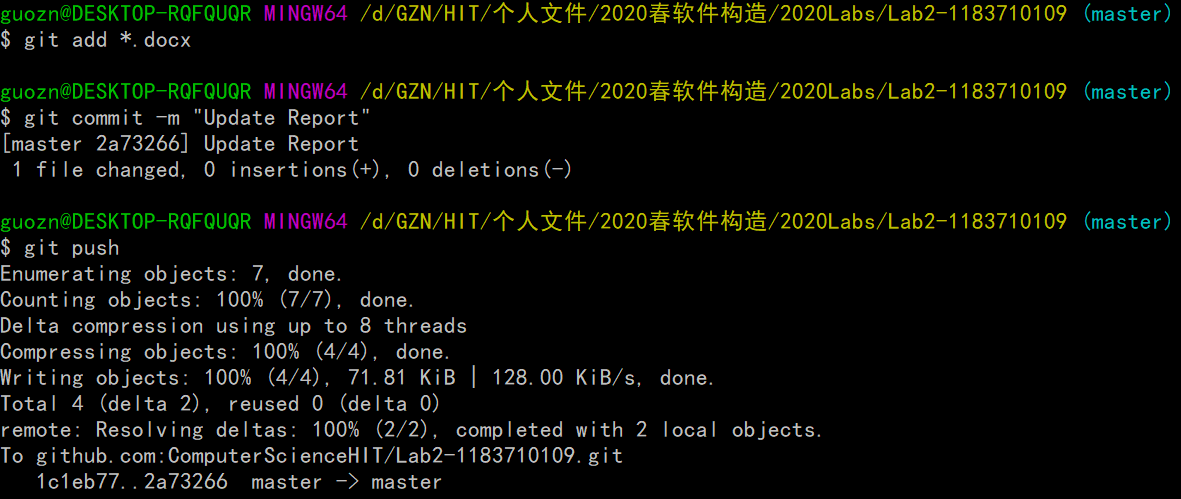
Junit测试



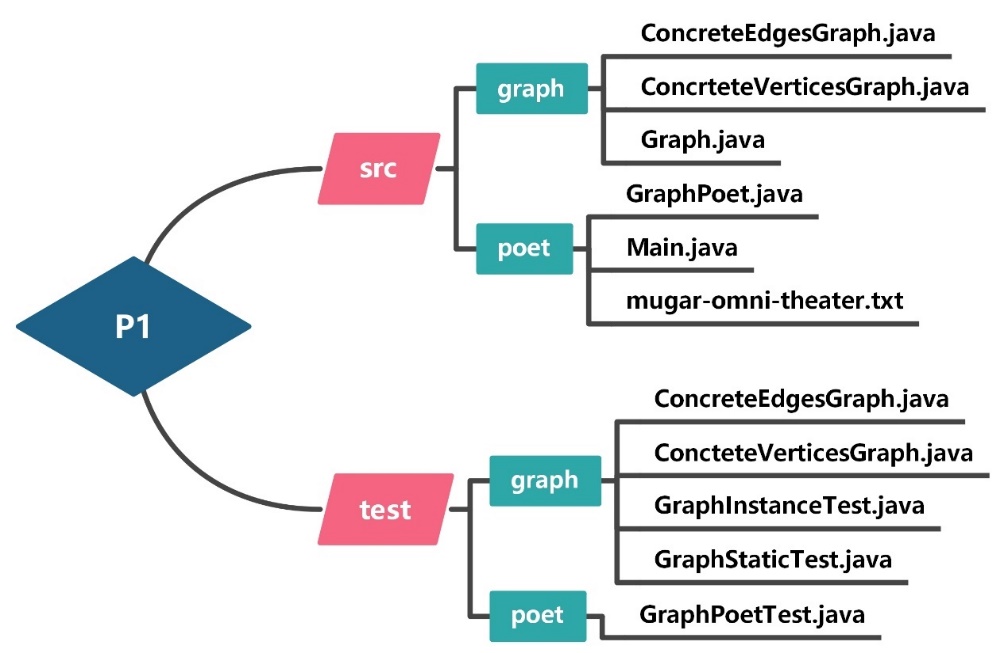
### Before you’re done

请按照<http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done>的说明，检查你的程序。

如何通过Git提交当前版本到GitHub上你的Lab2仓库。



在这里给出你的项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

这部分任务就是用我们在3.1中写的ADT，把第一次实验中的FriendshipGraph重新实现一遍，图中的节点仍然是Person类型，所以泛型L一律为Person. 而对于已经写好的FriendshipGraph中的方法，要用3.1中的Graph ADT中的方法来实现它们。

### FriendshipGraph类

Graph<Person> graph：

直接调用Graph的静态方法.empty()生成一个空的图。

boolean addVertex()：

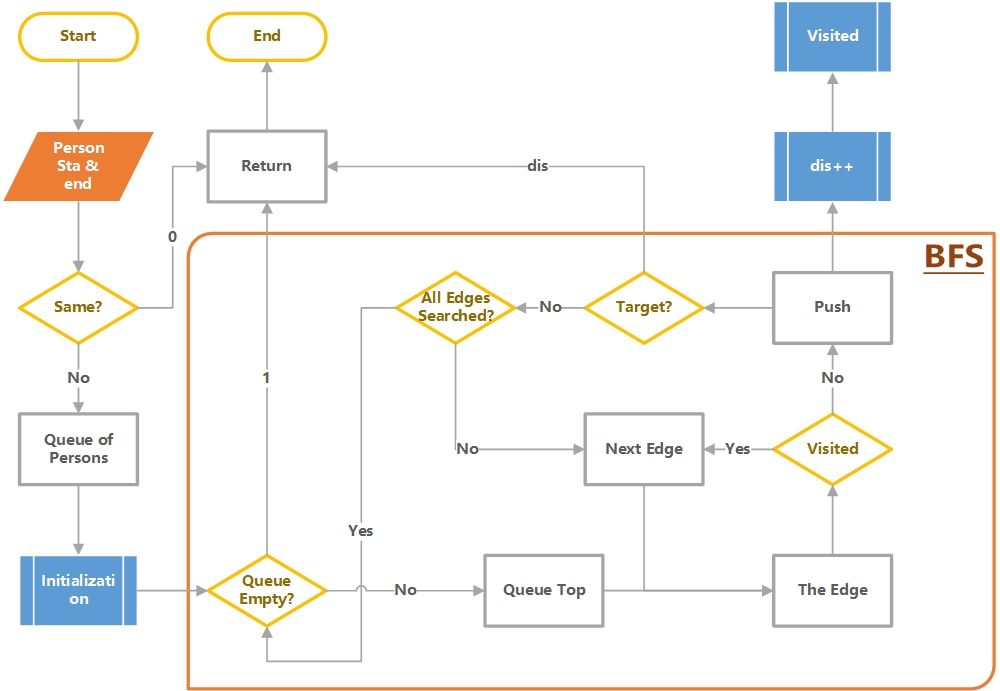
直接调用graph.add()添加点。

int addEdge()：

调用graph.set()两次，添加双向边，默认权值为1，并记录可能存在的旧边的权值。

int getDistance()：

首先判断起止点是否相等。再新建Map<Person, Integer> dis表示从起始点开始到该Person的距离，以及Map<Person, Boolean> vis表示该Person是否访问过。将两个Map初始化后，把起点标记为已经访问（所有涉及这两个Map的操作均需要remove后再put，后文不再阐述）。然后开始BFS搜索，找到终点为止。



### Person类

该类的目标是将每一个人对应到一个Person对象，并存储名字的信息。为了防止泄露，我将String Name设置为私有且不可变的。在构造函数中将Name初始化。

### 客户端main()

public class FriendshipGraphTest {

    /\*\*

     \* Basic Network Test

     \*/

    @Test

    public void Test1() {

        final FriendshipGraph graph = new FriendshipGraph();

        final Person rachel = new Person("Rachel");

        final Person ross = new Person("Ross");

        final Person ben = new Person("Ben");

        final Person kramer = new Person("Kramer");

        assertEquals(true, graph.addVertex(rachel));

        assertEquals(true, graph.addVertex(ross));

        assertEquals(true, graph.addVertex(ben));

        assertEquals(true, graph.addVertex(kramer));

        assertEquals(0, graph.addEdge(rachel, ross));

        assertEquals(1, graph.addEdge(ross, rachel));

        assertEquals(0, graph.addEdge(ross, ben));

        assertEquals(1, graph.addEdge(ben, ross));

        assertEquals(1, graph.getDistance(rachel, ross));

        assertEquals(2, graph.getDistance(rachel, ben));

        assertEquals(0, graph.getDistance(rachel, rachel));

        assertEquals(-1, graph.getDistance(rachel, kramer));

    }

    /\*\*

     \* Further Test

     \*/

    @Test

    public void Test2() {

        final FriendshipGraph graph = new FriendshipGraph();

        final Person a = new Person("A");

        final Person b = new Person("B");

        final Person c = new Person("C");

        final Person d = new Person("D");

        final Person e = new Person("E");

        final Person f = new Person("F");

        final Person g = new Person("G");

        final Person h = new Person("H");

        final Person i = new Person("I");

        final Person j = new Person("J");

        assertEquals(true, graph.addVertex(a));

        assertEquals(true, graph.addVertex(b));

        assertEquals(true, graph.addVertex(c));

        assertEquals(true, graph.addVertex(d));

        assertEquals(true, graph.addVertex(e));

        assertEquals(true, graph.addVertex(f));

        assertEquals(true, graph.addVertex(g));

        assertEquals(true, graph.addVertex(h));

        assertEquals(true, graph.addVertex(i));

        assertEquals(true, graph.addVertex(j));

        assertEquals(0, graph.addEdge(a, b));

        assertEquals(0, graph.addEdge(a, d));

        assertEquals(0, graph.addEdge(b, d));

        assertEquals(0, graph.addEdge(c, d));

        assertEquals(0, graph.addEdge(d, e));

        assertEquals(0, graph.addEdge(c, f));

        assertEquals(0, graph.addEdge(e, g));

        assertEquals(0, graph.addEdge(f, g));

        assertEquals(0, graph.addEdge(h, i));

        assertEquals(0, graph.addEdge(i, j));

        assertEquals(2, graph.getDistance(a, e));

        assertEquals(1, graph.getDistance(a, d));

        assertEquals(3, graph.getDistance(a, g));

        assertEquals(3, graph.getDistance(b, f));

        assertEquals(2, graph.getDistance(d, f));

        assertEquals(2, graph.getDistance(h, j));

        assertEquals(0, graph.getDistance(i, i));

        assertEquals(-1, graph.getDistance(d, j));

        assertEquals(-1, graph.getDistance(c, i));

        assertEquals(-1, graph.getDistance(f, h));

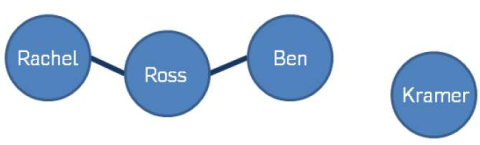
    }

}

### 测试用例

#### 简单图测试

根据题目中的社交网络图：

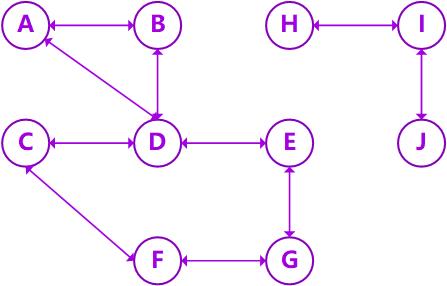


分别测试：

1. Rachel和Ross距离是1，Rachel和Ben距离是2
2. Rachel和Rachel距离是0
3. Rachel和Kramer距离是-1

#### 复杂图测试

设计10个点、10条边的社交网络图：

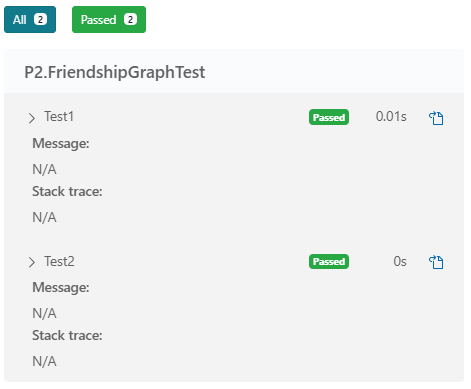


分别测试：

1. AE距离2，AD距离1，AG距离3，BF距离3，DF距离2，HJ距离2
2. II距离0
3. DJ距离-1，CI距离-1，FH距离-1

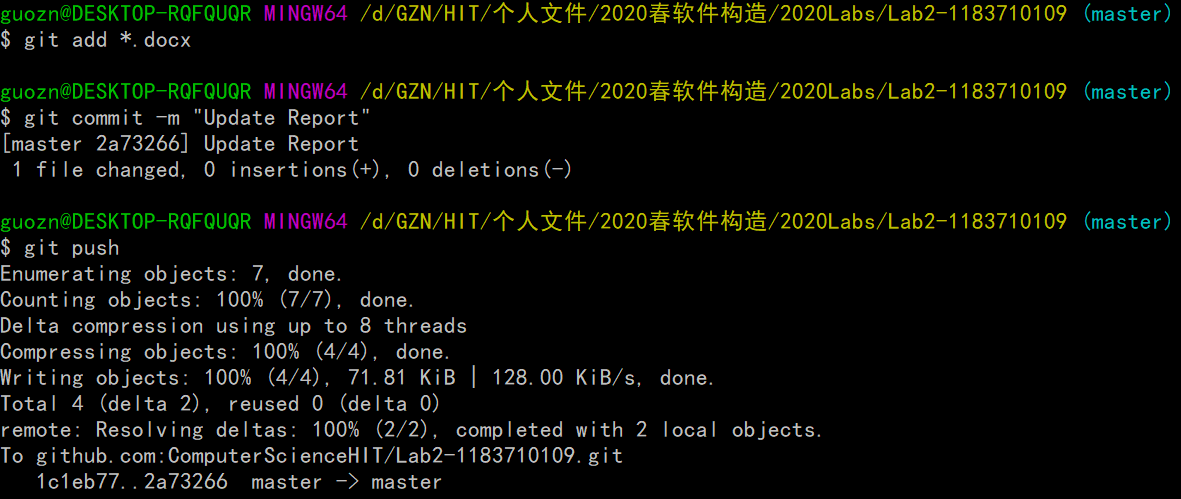
#### Junit测试结果

全部正确。

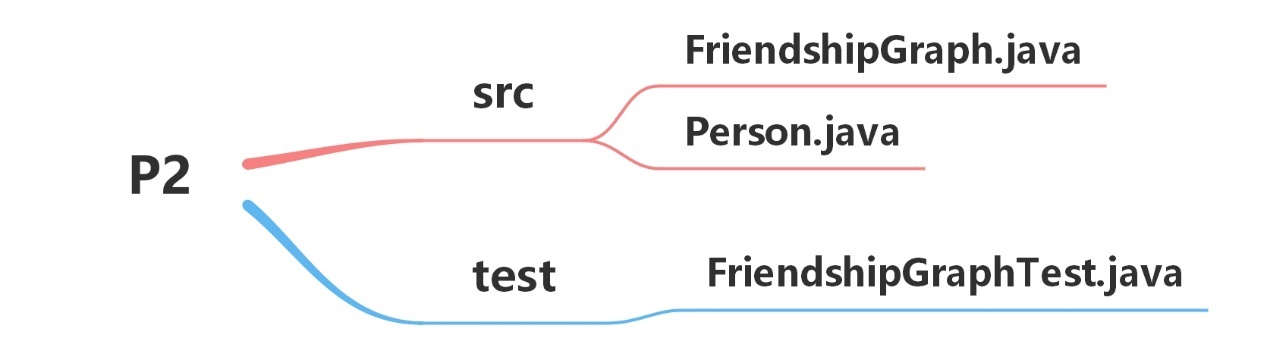


### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab2仓库。



在这里给出你的项目的目录结构树状示意图。



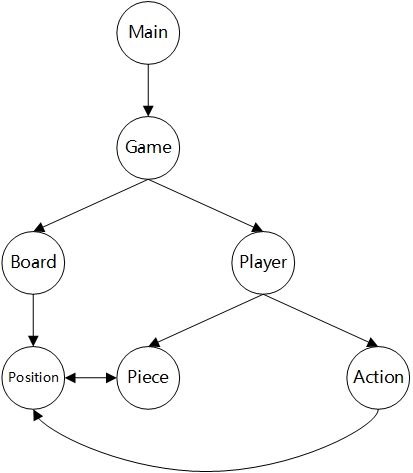
## Playing Chess

问题简述：

设计一款棋类游戏，同时支持国际象棋（Chess）和围棋（Go）。实现功能：

1. 选择游戏类型：创建Game、Board
2. 输入玩家名字：创建Player、Piece，其中Piece属于Player
3. 开始游戏，轮流选择功能
4. 放棋：给定player、piece、x、y
5. 移动棋（chess）：给定player、piece、x1、y1、x2、y2
6. 提子（go）：给定player、x、y
7. 吃子（chess）：给定player、x1、y1、x2、y2
8. 查询某个位置占用情况：给定x、y
9. 计算两个玩家分别的棋子总数
10. 跳过
11. 结束：输入“end”

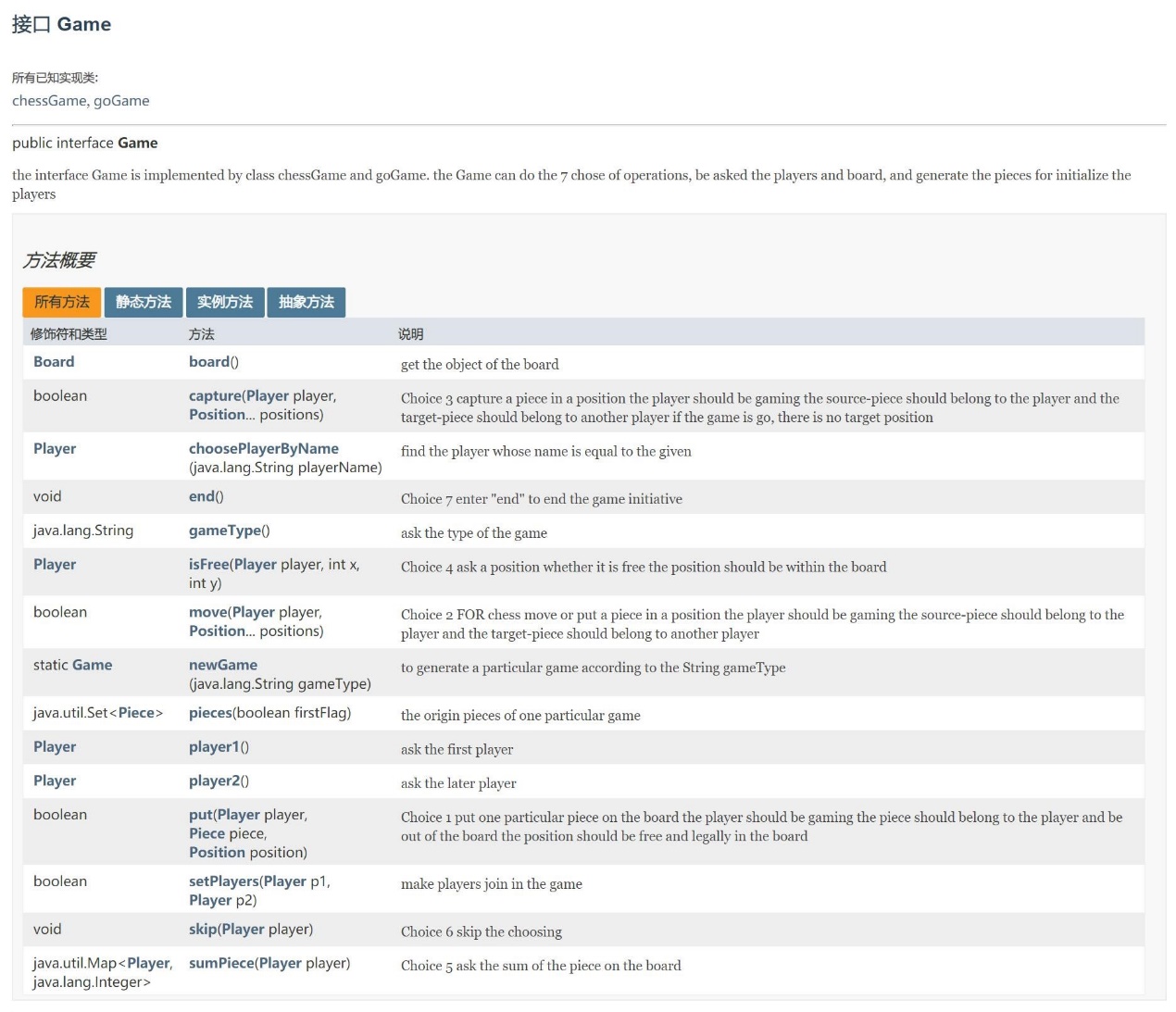
整体架构



### ADT设计/实现方案

#### interface Game

接口Game由chessGame和goGame实现，是Main()程序通向游戏对象的路口，通过一个接口把两种游戏分开，相同的操作类型在不同游戏中实现。



Game拥有7种操作所对应的方法，并且能支持访问下属的Player（先后手访问、名字访问）和Board，以及为玩家产生所对应Piece的功能。

7种操作除了“end”均隶属于Player对象进行操作，其中的“放棋”、“移动”、“吃子/提子”均在Action接口的实现类中完成，在Game接口的实现类中判断是否执行成功即可。因此3种操作可以在Game的实现类中实现近乎标准化和统一（输入操作类型String即可），以“吃子/提子”（capture）为例：

    @Override

    public boolean capture(Player player, Position... positions) {

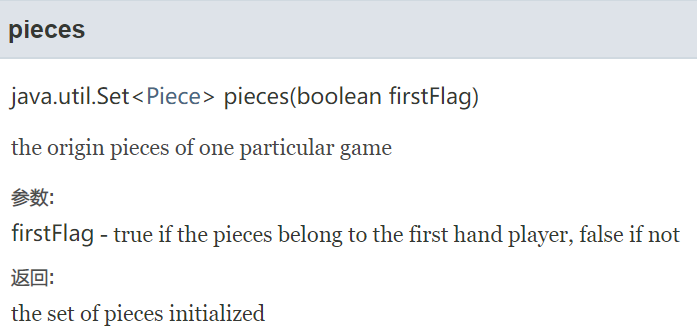
        if (player == null)

            return false;

        return player.doAction("capture", null, positions) != null;

    }

在两种游戏中，差异较大的之一就是棋子。棋子属于玩家，但棋子是由一个特定类型的游戏所“产生”的，因此Game的两个实现类中差异最大的就是产生棋子的方法：



在chess中，黑白双方棋子除了颜色都相同，因此可以用chessGame静态成员变量预设好每个棋子的名字、数量和位置（黑白双方可以用公式颠倒）。然后依据预设的静态数据新建16个Piece对象，初始化Position、Player，最后加入Set<Piece>中返回。goGame中大致相同。

    /\*\*

     \* the Map whose keys are the name of pieces, values are the numbers they are on board totally

     \*/

    private static final Map<String, Integer> piecesSumMap = new HashMap<String, Integer>() {

        private static final long serialVersionUID = 1L;

        {

            put("P", 8);

            put("R", 2);

            put("N", 2);

            put("B", 2);

            put("Q", 1);

            put("K", 1);

        }

    };

    /\*\*

     \* the Map whose keys are the name of pieces, values are the coordinates of them

     \*/

    private static final Map<String, int[][]> piecesPosMap = new HashMap<String, int[][]>() {

        private static final long serialVersionUID = 1L;

        {

            put("P", new int[][] { { 0, 1, 2, 3, 4, 5, 6, 7 }, { 1, 1, 1, 1, 1, 1, 1, 1 } });

            put("R", new int[][] { { 0, 7 }, { 0, 0 } });

            put("N", new int[][] { { 1, 6 }, { 0, 0 } });

            put("B", new int[][] { { 2, 5 }, { 0, 0 } });

            put("Q", new int[][] { { 3 }, { 0 } });

            put("K", new int[][] { { 4 }, { 0 } });

        }

    };

    @Override

    public Set<Piece> pieces(boolean firstFlag) {

        Set<Piece> pieces = new HashSet<Piece>();

        for (Map.Entry<String, Integer> entry : piecesSumMap.entrySet()) {

            for (int i = 0; i < entry.getValue(); i++) {

                String pieceName = (firstFlag ? "W" : "B") + entry.getKey() + i; // eg. WB1 BR2 WP3

                Piece piece = new Piece(pieceName, firstFlag, (firstFlag ? player1 : player2));

                // get the coordinate of a specific piece

                int[] X = piecesPosMap.get(entry.getKey())[0];

                int[] Y = piecesPosMap.get(entry.getKey())[1];

                int x = X[i], y = (firstFlag ? Y[i] : CHESS\_BOARD\_SIDE - Y[i] - 1);

                // put the piece on the position

                piece.modifyPositionAs(board.positionXY(x, y));

                board.positionXY(x, y).modifyPieceAs(piece);

                // add the piece into the piece set of the player

                pieces.add(piece);

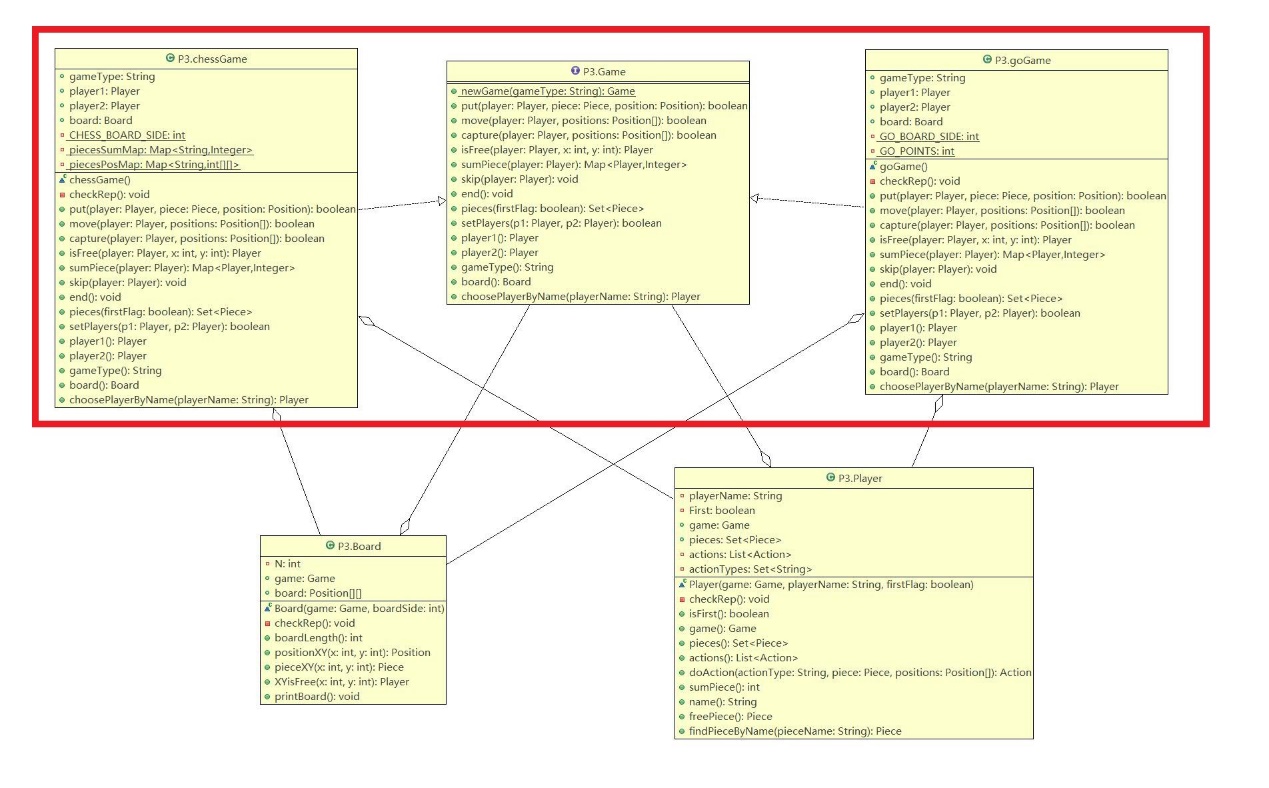
            }

        }

        return pieces;

}

Game是Board和Player的父类。Board的创建只能源于Game的构造函数，Player的创建必须后于Game且玩家的Piece依赖于Game的函数。



上图上方三个分别是Game和Game的实现类chessGame和goGame，Board和Game隶属于Game，在不同情况下调用两种实现类，且这两者无法联系，保护了对象的私有数据。

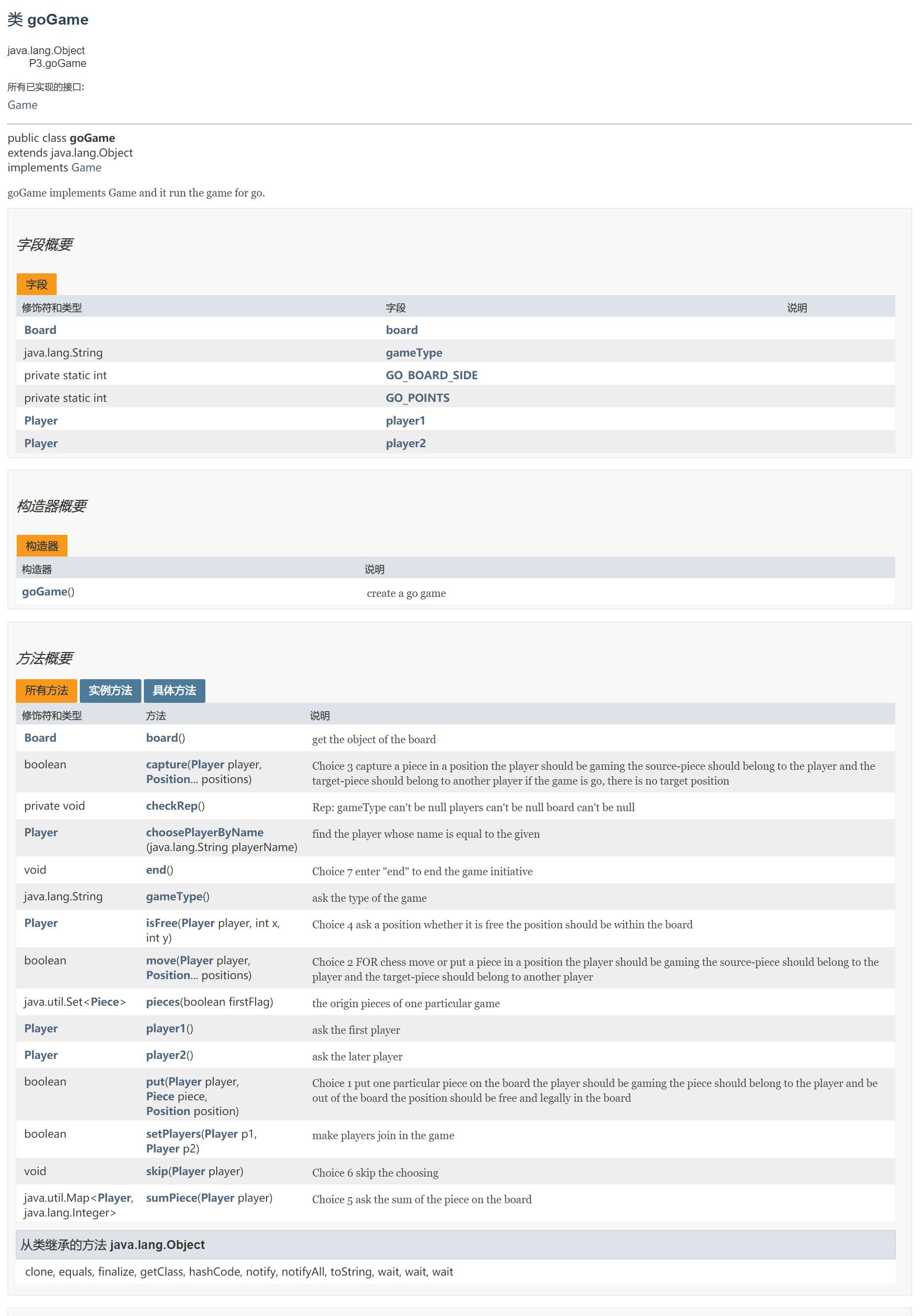
##### class chessGame

实现chess在Game中的功能。



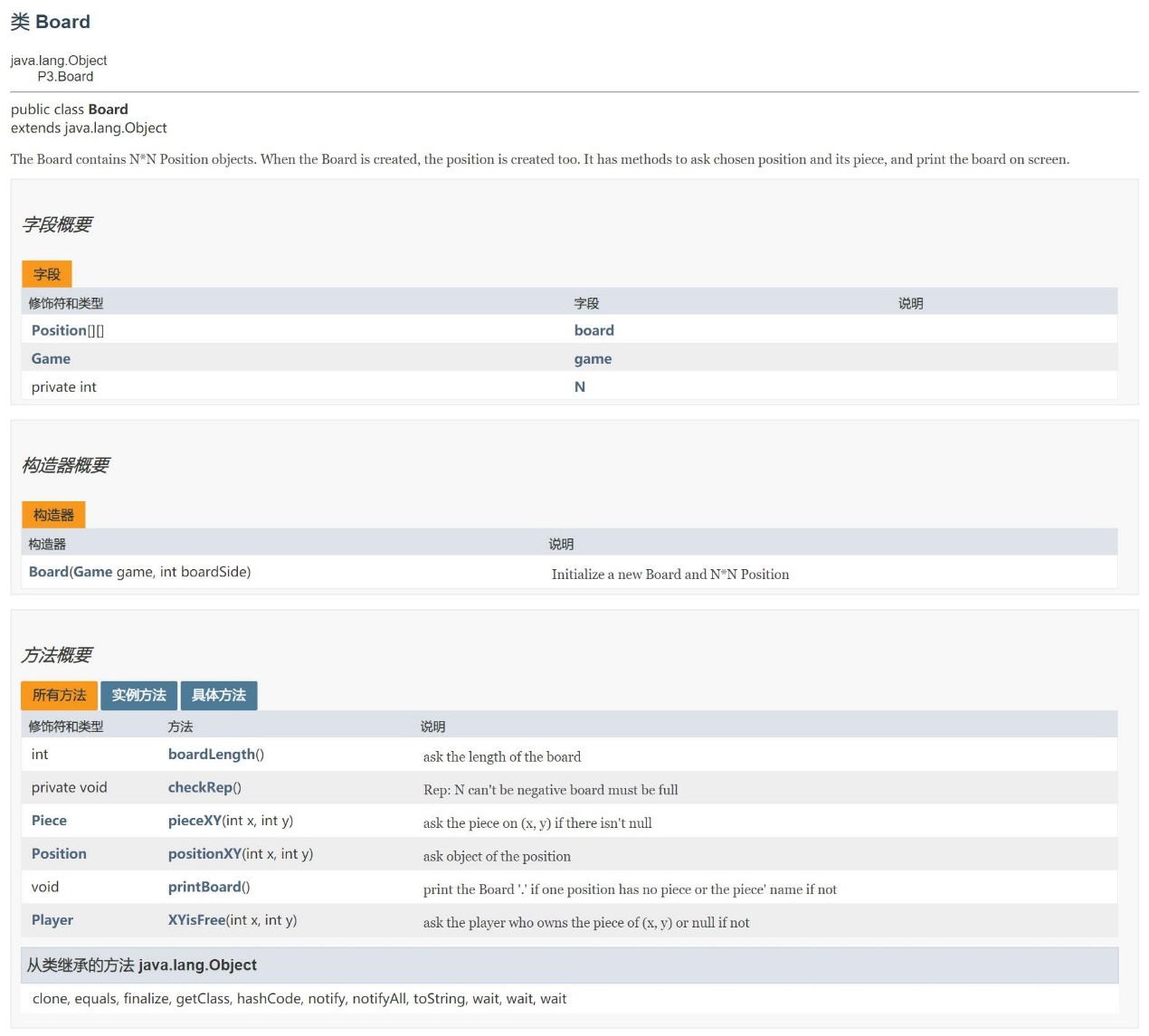
##### class goGame

实现go在Game中的功能。

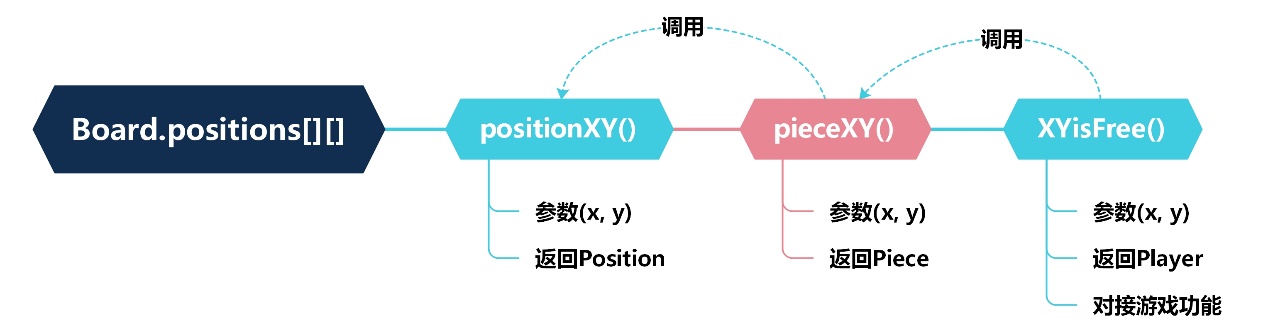


#### class Board

Board是棋盘的对象，构造依赖于Game的构造。Position的创建也依赖于Board，Board也存储这二维Position类型数组，并且拥有final变量N记录棋盘的边长。



Board的主要用于查询指定位置的Position、Piece和Player，以及打印棋盘。查询Position可以直接访问positions成员变量，而查询Piece又要访问指定位置的Position不为空的Piece，而查询Player又要查询不空的Piece的Player。



在三个函数的实现中，按照调用关系，先后实现。在这里设计Position和Piece平级且捆绑，同为可变。

    /\*\*

     \* ask object of the position

     \* @param x the x of the asking position

     \* @param y the y of the asking position

     \* @return object of Position of the (x, y)

     \*/

    public Position positionXY(int x, int y) {

        if (x < 0 || x >= this.N || y < 0 || y >= this.N)

            return null;

        return board[x][y];

    }

    /\*\*

     \* ask the piece on (x, y) if there isn't null

     \* @param x the x of the asking position

     \* @param y the y of the asking position

     \* @return object of Piece of the (x, y)

     \*/

    public Piece pieceXY(int x, int y) {

        if (positionXY(x, y) == null)

            return null;

        return positionXY(x, y).piece();

    }

    /\*\*

     \* ask the player who owns the piece of (x, y) or null if not

     \* @param x the x of the asking position

     \* @param y the y of the asking position

     \* @return Player if (x, y) is occupied, null if it's free

     \*/

    public Player XYisFree(int x, int y) {

        if (pieceXY(x, y) == null)

            return null;

        return pieceXY(x, y).player();

    }

此外，棋盘还具备打印功能。以下是实现方案和效果图。

    /\*\*

     \* print the Board

     \* '.' if one position has no piece

     \* or the piece' name if not

     \*/

    public void printBoard() {

        for (int i = 0; i < this.N; i++) {

            for (int j = 0; j < this.N; j++) {

                if (this.pieceXY(i, j) != null) {

                    if (game.gameType().equals("chess")) {

                        /\*

                         \* the capital letter represents the white piece

                         \* the little letter represents the black piece

                         \*/

                        System.out.print((this.pieceXY(i, j).isFirst() ? this.pieceXY(i, j).name().charAt(1)

                                : this.pieceXY(i, j).name().toLowerCase().charAt(1)) + " ");

                    } else if (game.gameType().equals("go")) {

                        /\*

                         \* the 'B' represents the black pieces

                         \* the 'W' represents the white pieces

                         \*/

                        System.out.print(this.pieceXY(i, j).name().charAt(0) + " ");

                    }

                } else {

                    // if there is no piece

                    System.out.print(". ");

                }

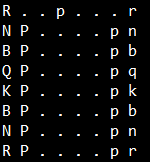
            }

            System.out.println();

        }

    }

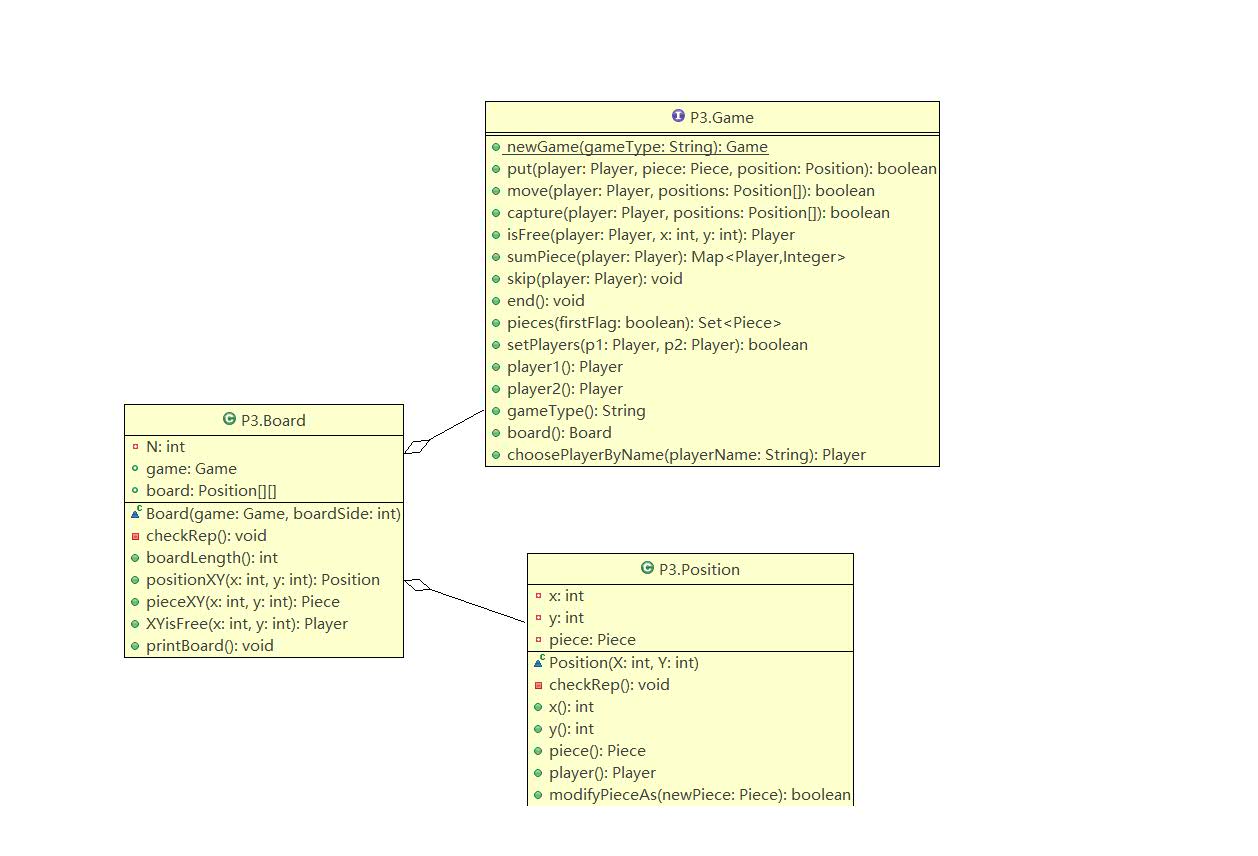
国际象棋：大写代表白方，小写代表黑方。



围棋：B代表黑方，W代表白方。

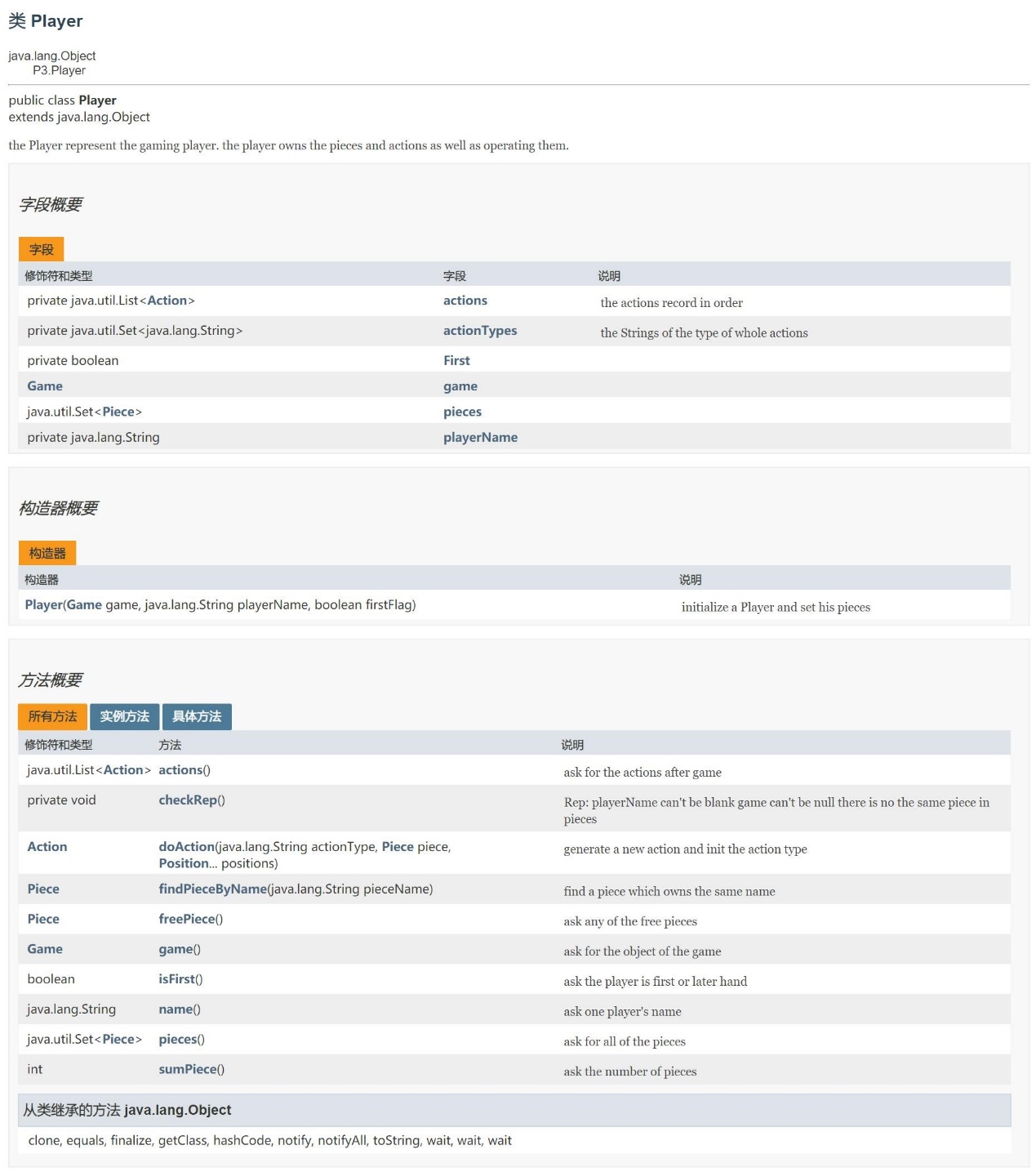


棋盘能够管理棋格/点，而根据要求棋盘是不能管理棋子的。因此Board是Game的子类，也是Position的父类。

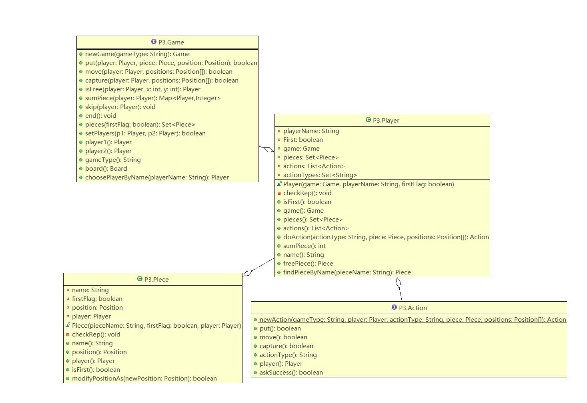


#### class Player

Player对象代表着玩家，有这Boolean标签区分先后手，拥有Piece并管理Action。



Player的方法除了查询本对象的信息，还有寻找自己下属的棋子，以及执行并记录Action。



寻找棋子时：当以棋子名字查询时，只需在成员变量Set<Piece> pieces中遍历，判断棋子的名字是否相等即可；而当查询任意一个空闲棋子时，则需判断其position是否为空。此外，Player还能计算本方棋盘上棋子总数。

    /\*\*

     \* ask the number of pieces

     \* @return the number of pieces

     \*/

    public int sumPiece() {

        int sum = 0;

        for (Piece piece : pieces) {

            // calculate the non-null piece

            if (piece.position() != null) {

                sum++;

            }

        }

        return sum;

    }

    /\*\*

     \* ask any of the free pieces

     \* @return a free piece belonging to the player

     \*/

    public Piece freePiece() {

        for (Piece piece : this.pieces) {

            // find a random free piece

            if (piece.position() == null)

                return piece;

        }

        return null;

    }

    /\*\*

     \* find a piece which owns the same name

     \* @param pieceName String of the name of the piece

     \* @return piece object of the piece name

     \*/

    public Piece findPieceByName(String pieceName) {

        for (Piece piece : this.pieces) {

            // find the piece whose name is matched with the giving

            if (piece.name().equals(pieceName))

                return piece;

        }

        return null;

    }

在执行Action的方法中，通过构造一个新的Action对象来执行。在Action对象的内部会进行完整的操作，玩家只需访问改动作对象的成功与否，然后加入List存储即可。

    /\*\*

     \* generate a new action and init the action type

     \* @param actionType String of the type of the action

     \* @param piece the putting piece when the actionType is "put", null if not

     \* @param positions the positions related to the action

     \* @return the object of the action created

     \*/

    public Action doAction(String actionType, Piece piece, Position... positions) {

        if (!actionTypes.contains(actionType))

            return null;

        Action action = Action.newAction(this.game.gameType(), this, actionType, piece, positions);

        if (action.askSuccess())

            actions.add(action);

        else

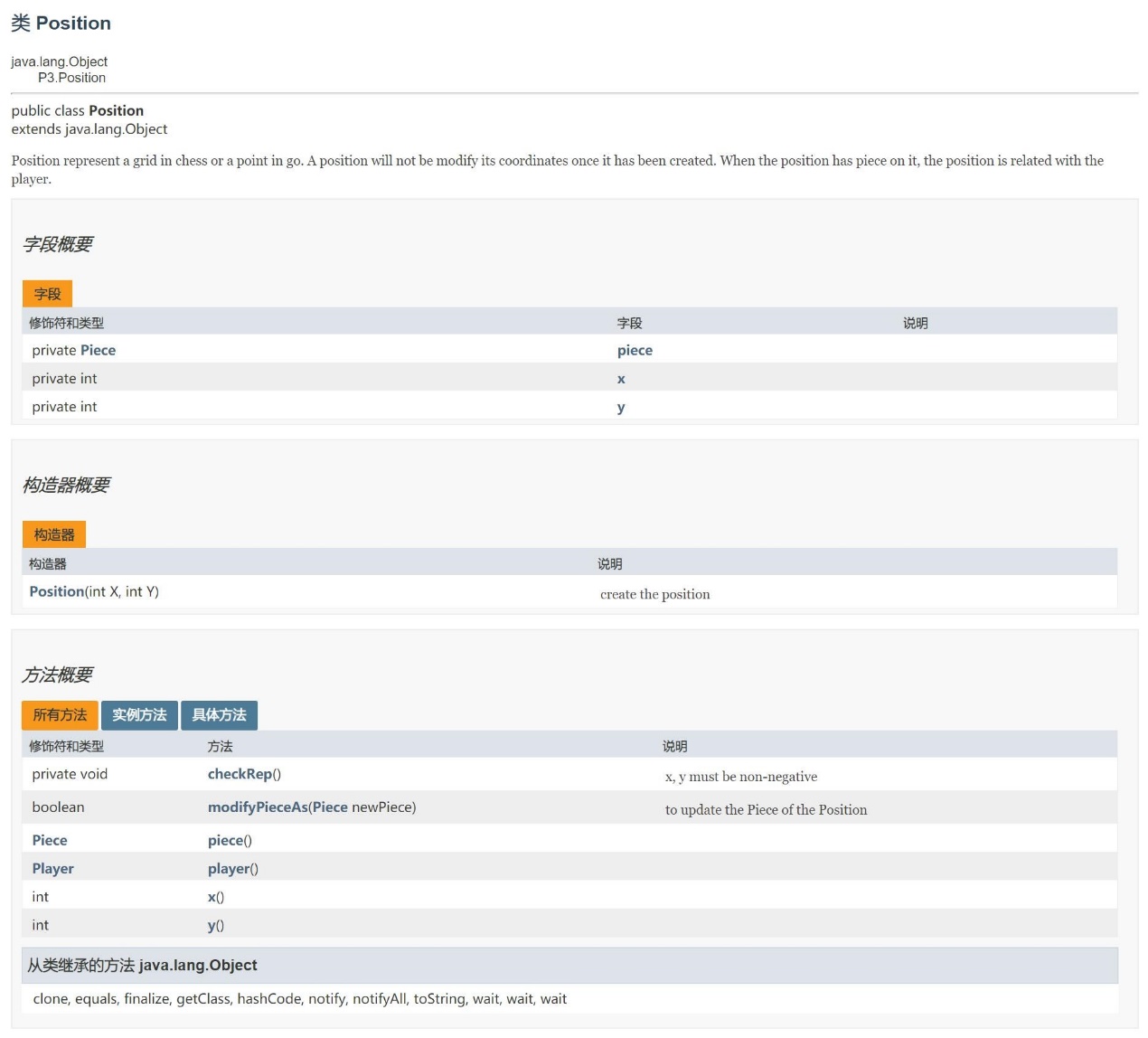
            action = null;

        return action;

    }

#### class Position

Position代表着国际象棋棋盘的格子以及围棋棋盘的交叉点。



Position隶属于Board，一个对象的x和y是不可变的，但Position记录的Piece对象是可变的，提供了方法进行修改。

    /\*\*

     \* to update the Piece of the Position

     \* @param newPiece the new piece that is to modify it as

     \* @return true if the Piece updated successfully, false if the new Piece is null

     \*/

    public boolean modifyPieceAs(Piece newPiece) {

        this.piece = newPiece;

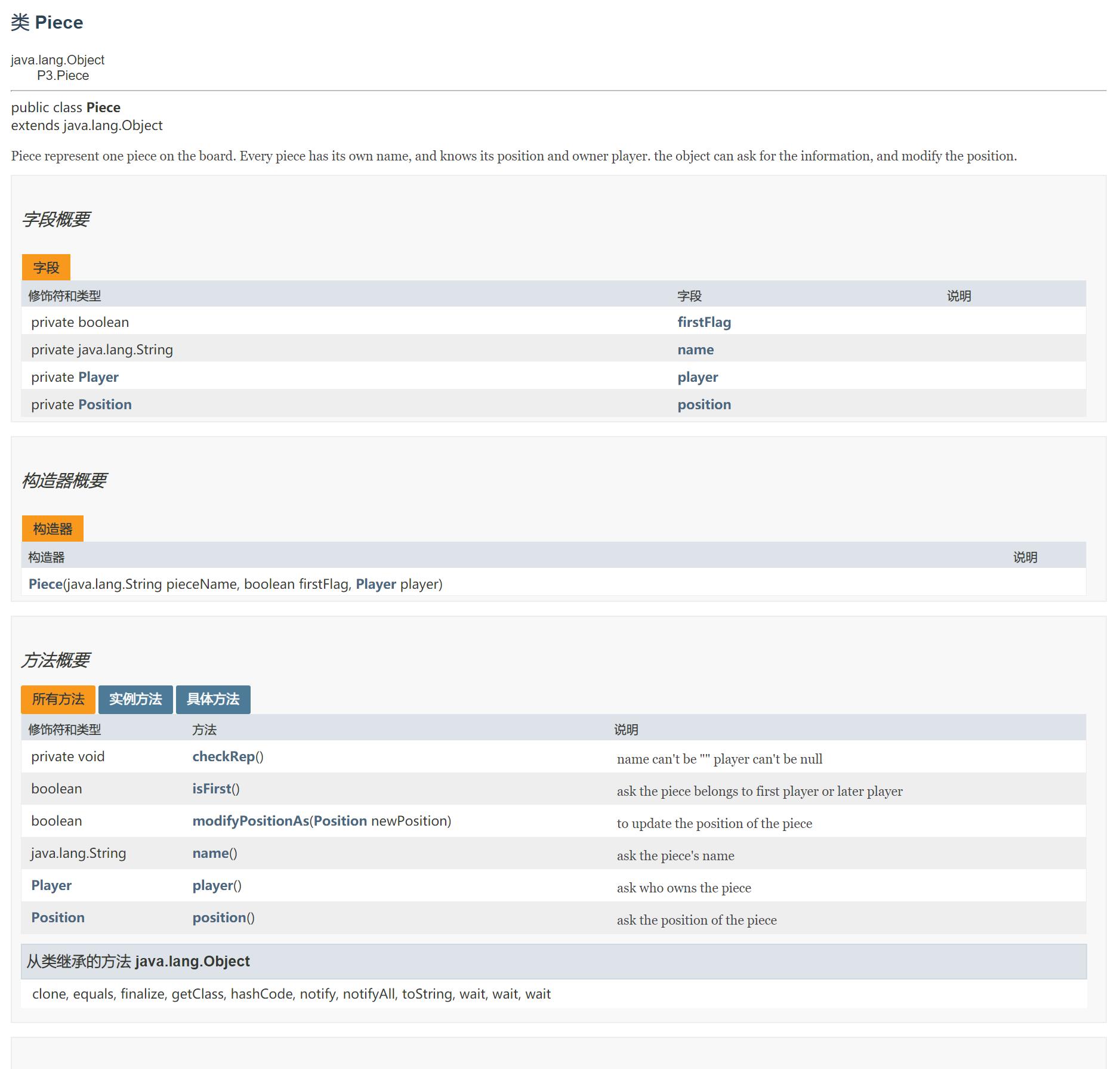
        checkRep();

        return true;

    }

#### class Piece

Piece代表着棋盘上的棋子，用一个唯一标识的String来区分每一个棋子，和一个Boolean来区分先后手，这两个信息是不可变的。



和Position相似，Piece也提供了修改Position的方法，用于移动棋子等变化操作。

    /\*\*

     \* to update the position of the piece

     \* @param newPosition the new position that is to modify it as

     \* @return true if the position updated successfully, false if the newPosition is null

     \*/

    public boolean modifyPositionAs(Position newPosition) {

        this.position = newPosition;

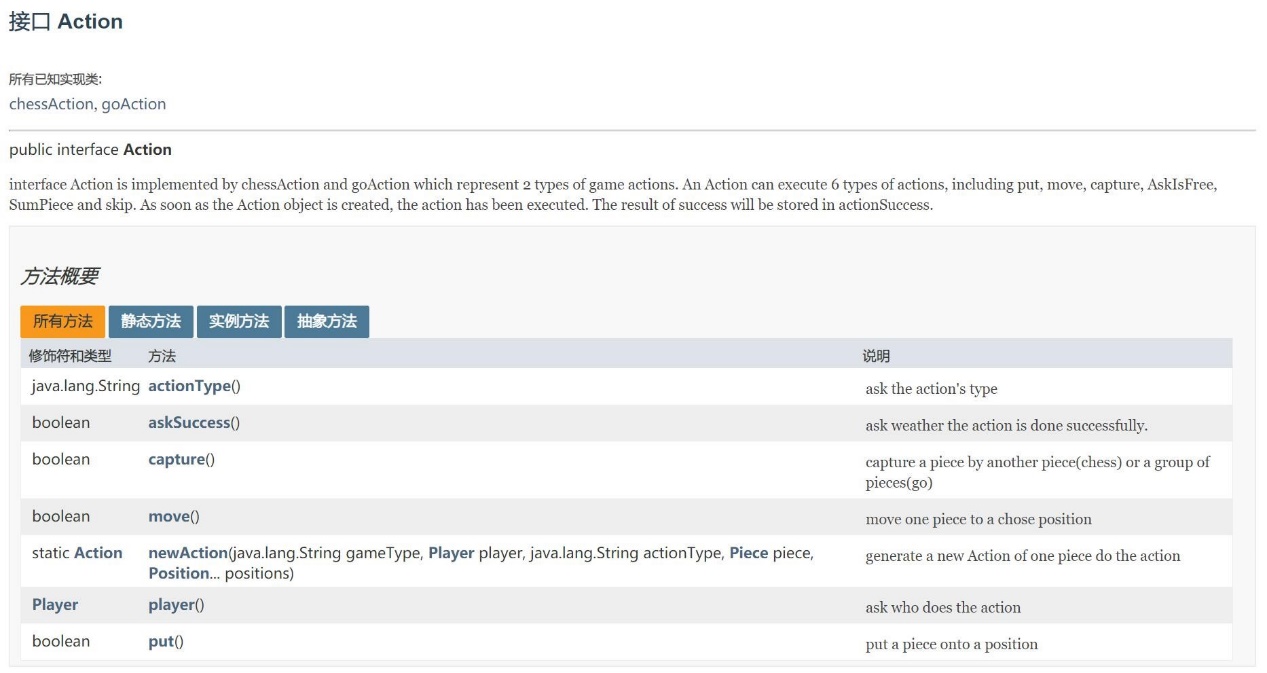
        checkRep();

        return true;

    }

#### interface Action

Action代表着要求里的若干个操作，通过一个String来区分，在构造函数中就实现这一操作（主要是放棋、吃子/提子、移动）。所需要的参数通过客户端的输入，再由Game传递，Player的方法doAction()中构造，然后在Action对象中执行，并记录执行的成败。Action是基于Player实现的，作用于Position。



接口Action有一个静态函数作为构造器：

    /\*\*

     \* generate a new Action of one piece

     \* do the action

     \* @param gameType String of the type of the game

     \* @param player the acting player

     \* @param actionType String of the type of the action

     \* @param piece the operating piece

     \* @param positions the positions related to the action

     \* @return an object of a type of Action(chessAction or goAction)

     \*/

    public static Action newAction(String gameType, Player player, String actionType, Piece piece,

            Position... positions) {

        return gameType.equals("chess") ? (new chessAction(player, actionType, piece, positions))

                : (new goAction(player, actionType, piece, positions));

    }

创建新的对象后，依据String actionType执行操作：

    /\*\*

     \* create and finish the action

     \* @param player the operating player

     \* @param actionType String of the type of the action

     \* @param piece the operating piece

     \* @param positions the position related to the action

     \*/

    chessAction(Player player, String actionType, Piece piece, Position... positions) {

        this.player = player;

        this.positions = positions;

        this.piece = piece;

        this.actionType = actionType;

        switch (actionType) {

            case "put":

                this.actionSuccess = (piece != null) && put();

                break;

            case "move":

                this.actionSuccess = move();

                break;

            case "capture":

                this.actionSuccess = capture();

                break;

            case "AskIsFree":

                this.actionSuccess = true;

                break;

            case "SumPiece":

                this.actionSuccess = true;

                break;

            case "skip":

                this.actionSuccess = true;

                break;

            default:

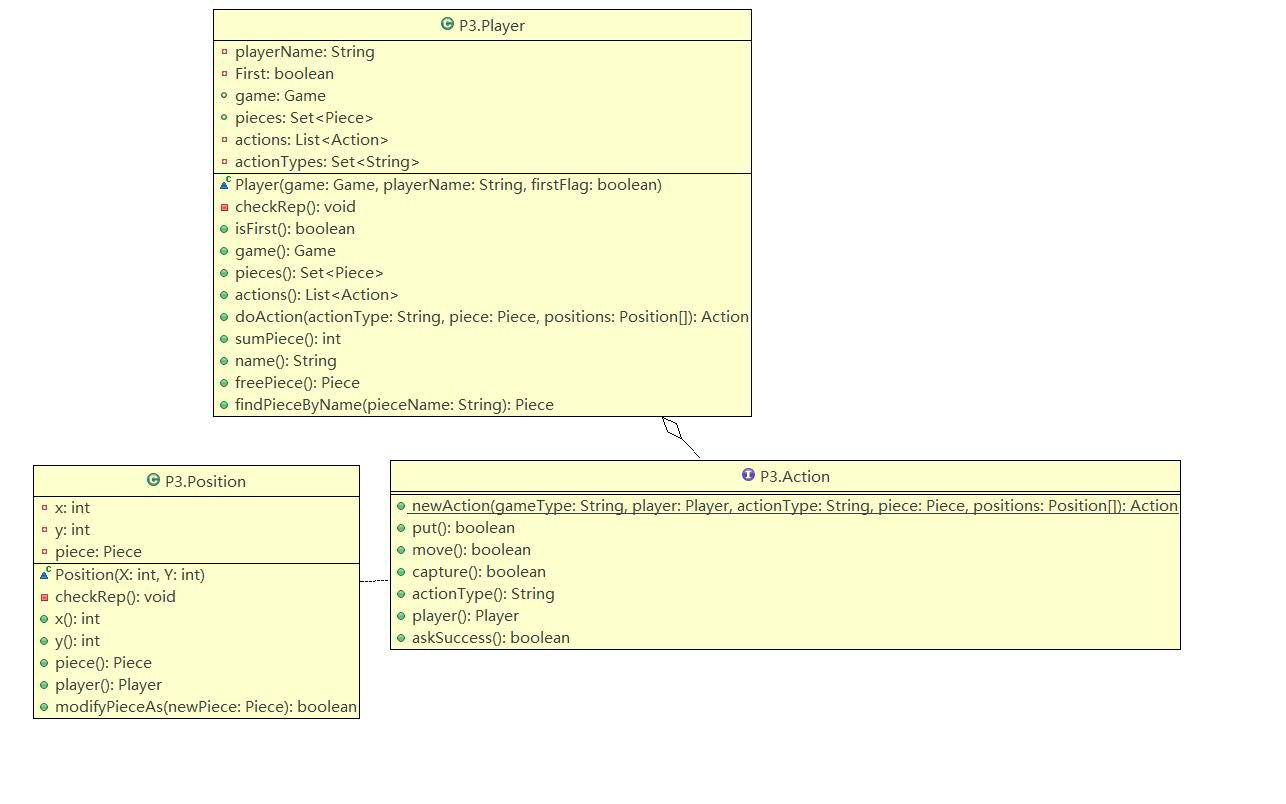
                this.actionSuccess = false;

        }

        checkRep();

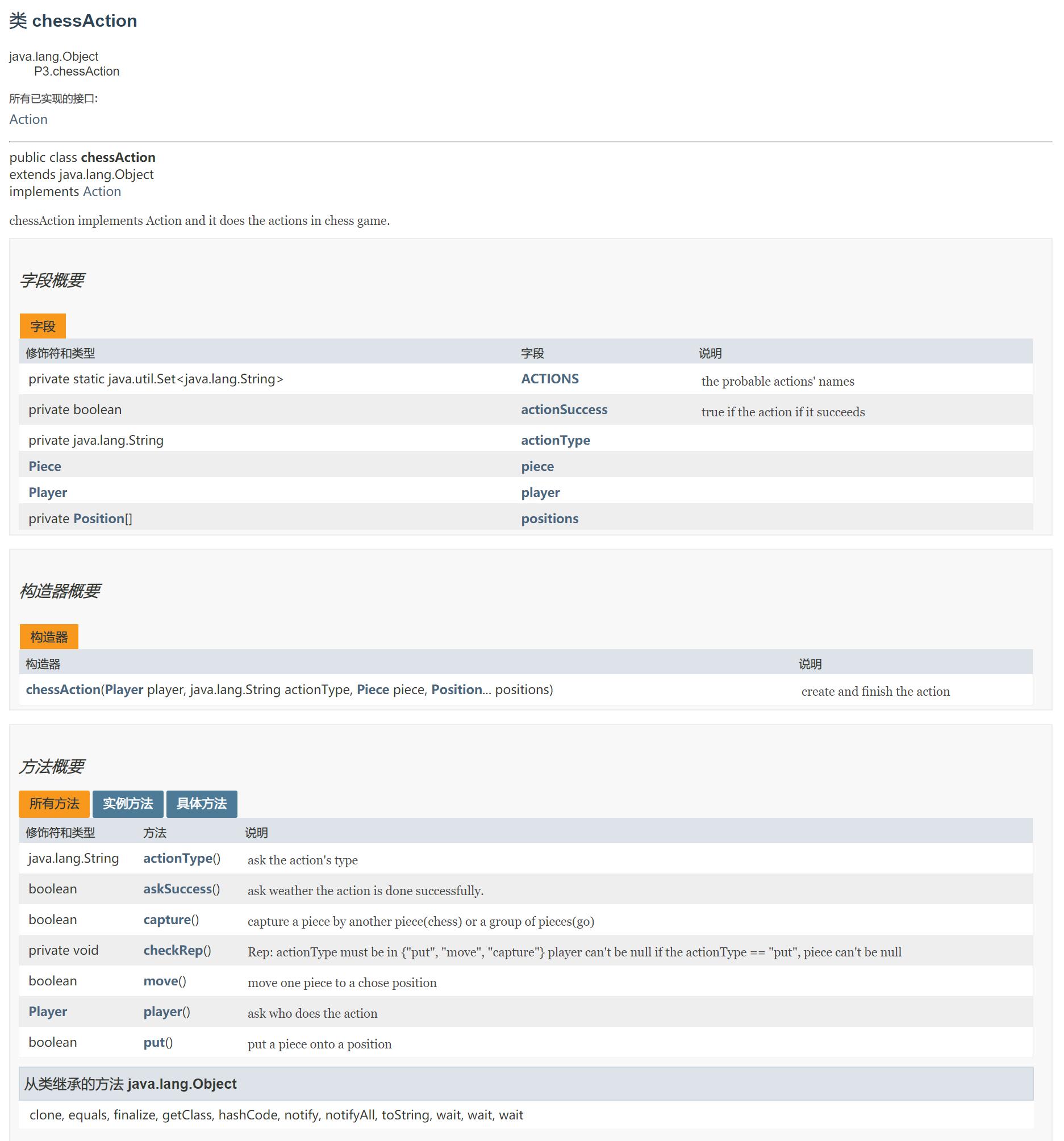
    }

chessAction和goAction分别重写Action中的操作，主要是put()、move()、capture()三个方法。最后通过方法的返回值将结果记录在actionSuccess。



##### class chessAction

chessAction代表着国际象棋中的操作。



put()操作与go类型，将在goAction中举例。move()操作和capture()操作中，国际象棋比围棋多需要1个Position操作，因此输入Position时就以不定项参数输入。两个操作大体类似，主要区别于：move的目标格要求为空，capture需要移除目标格棋子。以capture()为例：

    public boolean capture() {

        Position source = this.positions[0], target = this.positions[1];

        // capture requirement: 1. the target can't be null 2. the source can't be null

        // 3. the target must belong to the OPPOSITE 4. the source must belong to this player

        if (target.piece() != null && source.piece() != null && (!target.piece().player().equals(player))

                && source.piece().player().equals(player)) {

            target.piece().modifyPositionAs(null); // the piece capturing removed

            source.piece().modifyPositionAs(target); // captured piece move to the target

            target.modifyPieceAs(source.piece());// move the piece, this must be done before source's piece be null

            source.modifyPieceAs(null);// set the source null

            return true;

        }

        return false;

    }

##### class goAction

goAction代表着围棋中的操作。



put()操作在国际象棋和围棋中类似：客户端输入棋的名字（类型即可、无编号），然后game寻找该棋子后输入action对象；board将(x, y)转换为Position对象后输入action对象；最后执行：（以goAction为例）

    @Override

    public boolean put() {

        Position target = this.positions[0];

        // put requirement:

        // 1. the piece of the target can't be null

        // 2. the putting piece can't be null

        // 3. the piece must belong to the player

        if (this.piece.position() == null && target.piece() == null && player.pieces().contains(piece)) {

            this.piece.modifyPositionAs(target);

            target.modifyPieceAs(this.piece);

            return true;

        }

        return false;

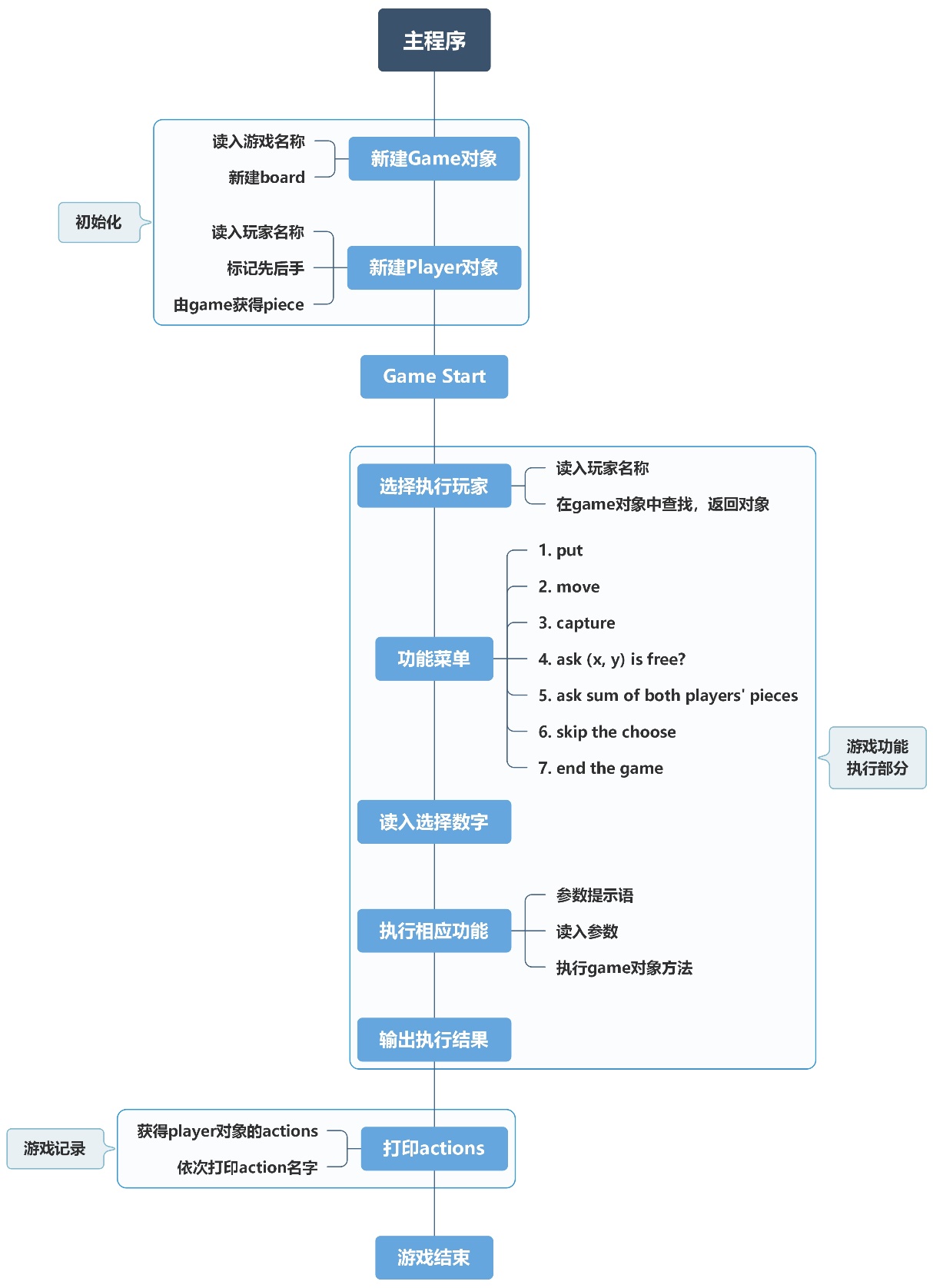
    }

### 主程序MyChessAndGoGame设计/实现方案

Main()主要分为3个步骤：

1. 新建游戏和玩家
2. 选择游戏功能并执行
3. 打印游戏记录

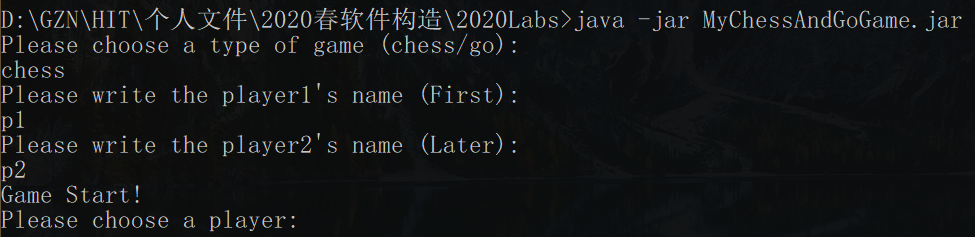
#### 游戏流程图



#### 程序演示

初始化：

给出提示语，输入游戏类型、玩家双方名字。若创建成功则显示“Game Start”。

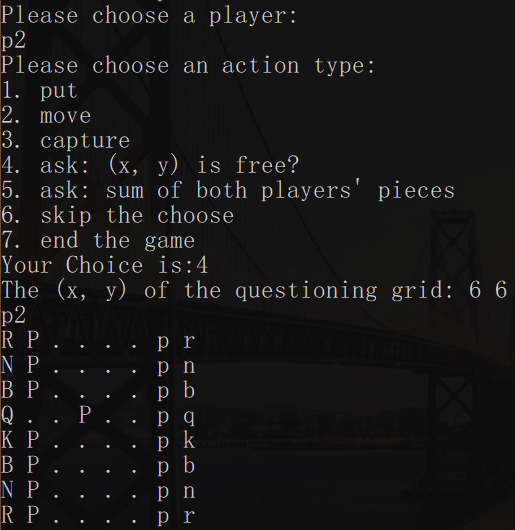


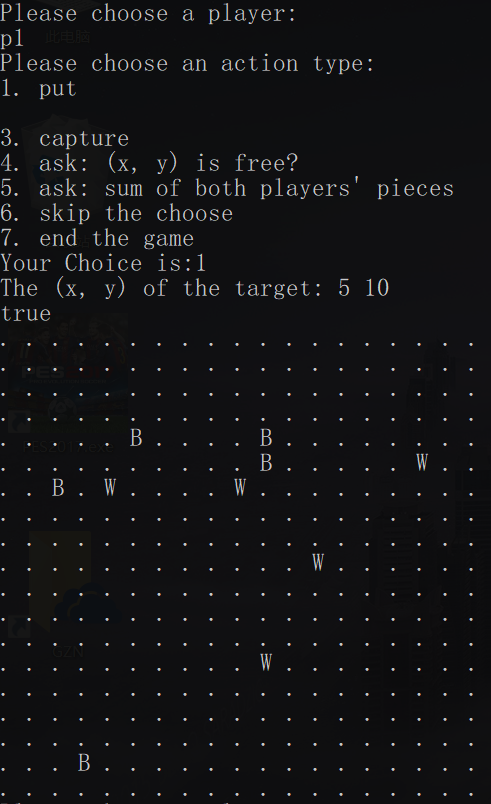
游戏功能执行：

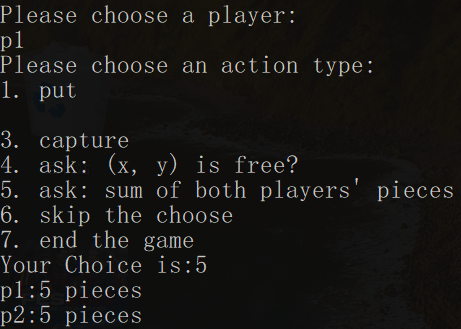
请用户选择玩家（输入名字），给出功能菜单，在用户选择后要求用户输入参数，执行功能，返回结果（true/false），打印当前棋盘。

下面给出一些操作的示例：









#### 功能实现

初始化：

    /\*\*

     \* provide 2 choices on screen for users to choose chess or go.

     \* generate new Game;

     \* generate new Board;

     \*

     \* get 2 players' names printed on the screen.

     \* generate 2 new Player;

     \* generate new Piece belonged to Player;

     \*

     \* @param *args* FORMAT

     \*/

    public static void main(String[] *args*) {

        // scan : 3 String

        System.out.println("Please choose a type of game (chess/go):");

        String gameType = input.nextLine();

        System.out.println("Please write the player1's name (First):");

        String playerName1 = input.nextLine();

        System.out.println("Please write the player2's name (Later):");

        String playerName2 = input.nextLine();

        // new objects

        final Game game = Game.newGame(gameType);

        final Player player1 = new Player(game, playerName1, true);

        final Player player2 = new Player(game, playerName2, false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // gaming

        GAME(game);

        input.close();

        // actions

        printRecord(game, player1, player2);

        System.out.println("That's All! See You Next Time!");

    }

功能执行：

    /\*\*

     \* provide 7 choices including;

     \* 1. to put a piece : put();

     \* 2. to move a piece : move();

     \* 3. to capture particular piece(s) : capture();

     \* 4. ask whether a position is free or not : isFree();

     \* 5. calculate the sum of the pieces on the

     \* board : sumPiece();

     \* 6. skip : skip()

     \* 7. print "end" : end()

     \* @param *game* the object of the game

     \*/

    private static void GAME(Game *game*) {

        boolean endFlag = false;

        System.out.println("Game Start!");

        while (!endFlag) {

            System.out.println("Please choose a player:");

            String playerName = input.next();

            endFlag = playerActing(game, game.choosePlayerByName(playerName));

            game.board().printBoard();

        }

    }

    /\*\*

     \* The chosen player is operating one choice

     \* @param *game* the object of the game

     \* @param *player* the object of the operating player

     \* @return true if the player choose ending the game, false if not

     \*/

    private static boolean playerActing(Game *game*, Player *player*) {

        // menu

        System.out.println("Please choose an action type:");

        System.out.println("1. put");

        System.out.println(game.gameType().equals("chess") ? "2. move" : "");

        System.out.println("3. capture");

        System.out.println("4. ask: (x, y) is free?");

        System.out.println("5. ask: sum of both players' pieces");

        System.out.println("6. skip the choose");

        System.out.println("7. end the game");

        // input information

        String pieceName = "";

        int x1, y1; // source

        int x2, y2; // target

        // catch choice

        System.out.print("Your Choice is:");

        int choice = input.nextInt();

        while (choice > 0 && choice <= 7) { // prepare the probable wrong choice

            switch (choice) {

                case 1: // put

                    if (game.gameType().equals("chess")) {

                        pieceName = input.next("Piece Name (eg. WQ0 BP2): ");

                    }

                    System.out.print("The (x, y) of the target: ");

                    x1 = input.nextInt();

                    y1 = input.nextInt();

                    // choose a piece freely if go

                    // get the particular piece if chess

                    Piece puttingPiece = game.gameType().equals("chess") ? player.findPieceByName(pieceName)

                            : player.freePiece();

                    // print result

                    System.out.println(game.put(player, puttingPiece, game.board().positionXY(x1, y1)));

                    return false;

                case 2: // move

                    System.out.print("The (x, y) of both source and target: ");

                    x1 = input.nextInt();

                    y1 = input.nextInt();

                    x2 = input.nextInt();

                    y2 = input.nextInt();

                    System.out.println(

                            game.move(player, game.board().positionXY(x1, y1), game.board().positionXY(x2, y2)));

                    return false;

                case 3: // capture

                    if (game.gameType().equals("chess")) {

                        System.out.print("The (x, y) of both source and target: ");

                        x1 = input.nextInt();

                        y1 = input.nextInt();

                        x2 = input.nextInt();

                        y2 = input.nextInt();

                        System.out.println(

                                game.capture(player, game.board().positionXY(x1, y1), game.board().positionXY(x2, y2)));

                    } else if (game.gameType().equals("go")) {

                        System.out.print("The (x, y) of the target: ");

                        x1 = input.nextInt();

                        y1 = input.nextInt();

                        System.out.println(game.capture(player, game.board().positionXY(x1, y1)));

                    }

                    return false;

                case 4: // is free?

                    System.out.print("The (x, y) of the questioning grid: ");

                    x1 = input.nextInt();

                    y1 = input.nextInt();

                    Player here = game.isFree(player, x1, y1);

                    System.out.println(here == null ? "Free" : here.name());

                    return false;

                case 5: // sum of pieces

                    Map<Player, Integer> sumPiece = game.sumPiece(player); // two players' sum of pieces

                    System.out.println(game.player1().name() + ":" + sumPiece.get(game.player1()) + " pieces");

                    System.out.println(game.player2().name() + ":" + sumPiece.get(game.player2()) + " pieces");

                    return false;

                case 6: // skip

                    game.skip(player);

                    System.out.println("Skip");

                    return false;

                case 7: // end

                    game.end();

                    System.out.println("The Game is ended.");

                    return true;

            }

            System.out.println("Input WRONG, Please choose again:");

        }

        return false;

    }

打印记录：

    /\*\*

     \* after the game is ended, to print both players' records of the game.

     \* @param *game* the object of the game

     \* @param *player1* the object of the first hand player

     \* @param *player2* the object of the later hand player

     \*/

    private static void printRecord(Game *game*, Player *player1*, Player *player2*) {

        System.out.println("\nAll of the Actions Record are followed.");

        // get the record of the actions

        List<Action> actions1 = player1.actions();

        List<Action> actions2 = player2.actions();

        System.out.println("\n" + player1.name() + "'s Actions:");

        // print their action types

        for (int i = 0; i < actions1.size(); i++) {

            if (actions1.get(i) != null)

                System.out.println(i + ": " + actions1.get(i).actionType());

        }

        System.out.println("\n" + player2.name() + "'s Actions:");

        for (int i = 0; i < actions2.size(); i++) {

            if (actions2.get(i) != null)

                System.out.println(i + ": " + actions2.get(i).actionType());

        }

    }

### ADT和主程序的测试方案

分别测试国际象棋和围棋，测试类分别为ChessTest和GoTest。由于国际象棋的测试比围棋困难，约束也比围棋多，因此设计ChessTest后进行一定更改就可以测试围棋。以下以chess为例，设计ADT测试方案。

#### testInit()

测试初始化游戏的操作：新建指定类型的game、board，新建指定名字的player，以及新建piece并将棋子赋予player。

测试game、player、piece均不为空。

    @Test

    public void testInit() {

        final Game game = Game.newGame("chess");

        assertNotEquals(null, game);

        final Player player1 = new Player(game, "p1", true);

        assertNotEquals(null, player1);

        final Player player2 = new Player(game, "p2", false);

        assertNotEquals(null, player2);

        assertEquals(true, game.setPlayers(player1, player2));

        player1.pieces = game.pieces(true);

        assertNotEquals(Collections.EMPTY\_SET, player1.pieces());

        player2.pieces = game.pieces(false);

        assertNotEquals(Collections.EMPTY\_SET, player2.pieces());

    }

#### testPiece()

测试棋子是否被新建，以及是否赋予玩家、初始化位置。

    @Test

    public void testPiece() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // Piece.modifyAsPosition

        for (Piece piece : player1.pieces()) {

            assertNotNull(piece.position());

            assertNotNull(piece.position().piece());

            assertSame(piece, piece.position().piece());

        }

    }

#### testBoard

测试棋盘的方法。

1. printBoard()打印出棋盘；
2. positionXY()返回的Position对象不为空，并且当位置有棋子是判断棋子的名字是否和期望一样；
3. pieceXY()返回的棋名字一样；
4. XYisFree()分别测试有棋子和没棋子的情况

    @Test

    public void testBoard() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // Board.printBoard()

        assertTrue(game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 3)));

        assertTrue(game.capture(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 3)));

        game.board().printBoard();

        // Board.PositionXY(x, y)

        assertNotNull(game.board().positionXY(0, 1));

        assertNotNull(game.board().positionXY(4, 6));

        assertEquals("BP4", game.board().positionXY(4, 6).piece().name());

        assertNull(game.board().positionXY(3, 3).piece());

        // Board.pieceXY(x, y)

        assertEquals("WR0", game.board().pieceXY(0, 0).name());

        assertEquals("BP5", game.board().pieceXY(5, 6).name());

        // Board.XYisFree(x, y)

        assertEquals(null, game.board().XYisFree(4, 5));

        assertEquals(player1, game.board().XYisFree(1, 1));

        assertEquals(player2, game.board().XYisFree(7, 6));

    }

#### testPosition()

测试位置的方法。

1. 测试位置的棋子的名字；
2. 测试查看横纵坐标、所属玩家（当无棋子时，可能为空）；
3. 测试更改position的piece

    @Test

    public void testPosition() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // Position.piece()

        assertEquals("BQ0", game.board().positionXY(3, 7).piece().name());

        assertEquals("WK0", game.board().positionXY(4, 0).piece().name());

        // Position.x()

        assertEquals(4, game.board().positionXY(4, 0).x());

        // Position.y()

        assertEquals(5, game.board().positionXY(2, 5).y());

        // Position.player()

        assertEquals(player2, game.board().positionXY(6, 6).player());

        // Position.modify

        assertEquals(player1, game.board().XYisFree(1, 1));

        assertEquals(true, game.board().positionXY(1, 1).modifyPieceAs(null));

        assertNull(game.board().XYisFree(1, 1));

    }

#### testPlayer()

测试玩家。

1. 测试双方先后手的标签；
2. 测试player所属游戏；
3. 测试player拥有的所有pieces，是否在对应坐标的棋盘位置上存在；

    @Test

    public void testPlayer() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // Player.isFirst()

        assertEquals(true, player1.isFirst());

        assertEquals(false, player2.isFirst());

        // Player.game()

        assertEquals(game, player2.game());

        // Player.pieces()

        for (int i = 0; i < game.board().boardLength(); i++) {

            for (int j = 0; j < game.board().boardLength(); j++) {

                if (game.board().pieceXY(i, j) != null)

                    assertEquals(true, player1.pieces().contains(game.board().pieceXY(i, j))

                            || player2.pieces().contains(game.board().pieceXY(i, j)));

            }

        }

}

#### testPut()

测试放棋操作。

1. 当选定棋子在棋盘上/不在棋盘上时；
2. 当棋子放的位置有/无棋子；

    @Test

    public void testPut() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // put

        assertEquals(false, game.put(player1, player1.findPieceByName("WP0"), game.board().positionXY(0, 1)));

        assertEquals(false, game.put(player2, player2.findPieceByName("BN1"), game.board().positionXY(5, 4)));

        // After capture

        assertTrue(game.capture(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 1)));

        assertTrue(game.put(player1, player1.findPieceByName("WP0"), game.board().positionXY(0, 6)));

    }

#### testMove()

测试移动棋子操作。

1. 选定格的棋子是否为本方；
2. 选定格的棋子是否存在
3. 目标格的棋子是否为空；
4. 移动之后选定格格子是否为空，而目标格是否不空且为之前选定格的棋子；

    @Test

    public void testMove() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // move

        assertEquals(true, game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 3)));

        assertNull(game.board().positionXY(0, 1).piece());

        assertNotNull(game.board().positionXY(0, 3).piece());

        assertEquals(false, game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 4)));

        assertEquals(false, game.move(player2, game.board().positionXY(6, 0), game.board().positionXY(5, 2)));

        assertEquals(true, game.move(player1, game.board().positionXY(1, 0), game.board().positionXY(2, 2)));

        assertEquals(false, game.move(player1, game.board().positionXY(1, 0), game.board().positionXY(0, 2)));

        assertNull(game.board().positionXY(0, 2).piece());

        assertNull(game.board().positionXY(1, 0).piece());

        assertEquals(false, game.move(player2, game.board().positionXY(3, 7), game.board().positionXY(4, 7)));

        assertEquals(false, game.move(player1, game.board().positionXY(2, 2), game.board().positionXY(4, 1)));

        assertNotNull(game.board().positionXY(4, 1).piece());

        assertNotNull(game.board().positionXY(2, 2).piece());

        assertEquals(true, game.move(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 1)));

        assertEquals(player2, game.board().positionXY(0, 1).piece().player());

        assertEquals("BP0", game.board().positionXY(0, 1).piece().name());

        assertEquals(player1, game.board().positionXY(0, 3).piece().player());

        // sumPiece

        assertEquals(16, player2.sumPiece());

    }

#### testCaptureAndPut()

结合put操作测试capture吃子操作。

1. 吃子的起始格是否为本方；
2. 吃子的起始格是否为空；
3. 吃子的目标格是否为对方；
4. 吃子的目标格是否为空；
5. 查看吃子之后两个格子的棋子名字是否为期望；

    @Test

    public void testCaptureAndPut() {

        // init

        final Game game = Game.newGame("chess");

        final Player player1 = new Player(game, "p1", true);

        final Player player2 = new Player(game, "p2", false);

        game.setPlayers(player1, player2);

        player1.pieces = game.pieces(true);

        player2.pieces = game.pieces(false);

        // capture

        assertEquals(true, game.capture(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 6)));

        assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(2, 1)));

        assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(1, 3)));

        assertEquals(true, game.capture(player1, game.board().positionXY(0, 6), game.board().positionXY(1, 6)));

        assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(1, 6)));

        assertEquals("WP0", game.board().pieceXY(1, 6).name());

        assertSame(player1.findPieceByName("WP0"), game.board().pieceXY(1, 6));

        assertEquals(true, game.capture(player1, game.board().positionXY(2, 1), game.board().positionXY(2, 6)));

        assertEquals(13, player2.sumPiece());

        // put

        assertEquals("BP1", player2.findPieceByName("BP1").name());

        assertNull(game.board().positionXY(0, 4).piece());

        assertTrue(game.put(player2, player2.findPieceByName("BP0"), game.board().positionXY(0, 4)));

        assertFalse(game.put(player2, player2.findPieceByName("BP1"), game.board().positionXY(0, 4)));

        assertFalse(game.put(player1, player2.findPieceByName("BP1"), game.board().positionXY(0, 4)));

        assertFalse(game.put(player2, player2.findPieceByName("BP0"), game.board().positionXY(0, 4)));

        assertNull(player1.findPieceByName("BP1"));

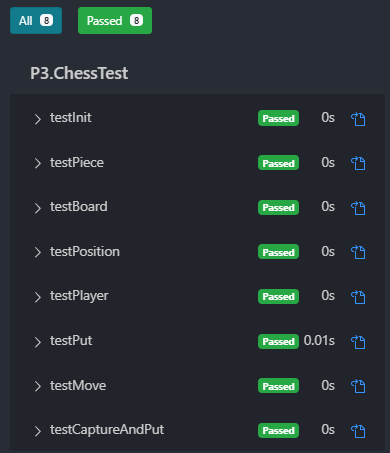
        assertFalse(game.put(player2, player1.findPieceByName("BP1"), game.board().positionXY(1, 4)));

        assertNull(game.board().positionXY(8, -1));

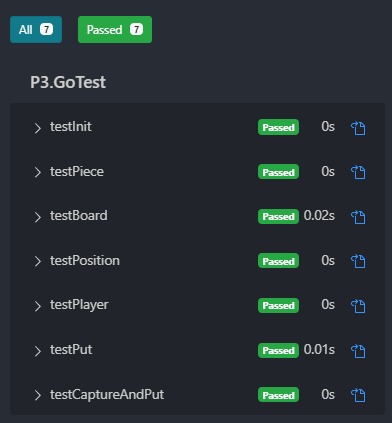
        assertFalse(game.put(player2, player2.findPieceByName("BP2"), game.board().positionXY(8, -1)));

}

#### ChessTest测试结果



#### GoTest测试结果



# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020-03-09 | 晚上 | 初始化项目 | 完成 |
| 2020-03-10 | 中午 | Problem1 3.1.1-3.1.2 | 完成 |
| 2020-03-10 | 晚上 | Problem1 3.1.3.1 Edge Graph | 完成 |
| 2020-03-11 | 晚上 | Problem1 3.1.3.2 Vertex Graph | 完成 |
| 2020-03-12 | 下午 | 通过Graph Instance Test | 完成 |
| 2020-03-12 | 晚上 | Problem1 3.1.5 Graph Poet | 完成 |
| 2020-03-13 | 上午 | Problem1 3.1.5 Test | 完成 |
| 2020-03-13 | 上午 | Problem2 3.2整体完成 | 完成 |
| 2020-03-13 | 晚上 | Problem3 设计框架 写AF&RI | 完成 |
| 2020-03-14 | 上午 | Problem3 完成框架 | 完成 |
| 2020-03-14 | 下午 | 实现Action接口、Player、Board、Position、Piece具体功能 | 完成 |
| 2020-03-14 | 晚上 | 完善上述功能，修改bug | 完成 |
| 2020-03-15 | 上午 | 实现Game接口、chessAction、goAction功能 | 完成 |
| 2020-03-15 | 晚上 | 实现chessGame、goGame功能，调试测试用例 | 完成 |
| 2020-03-16 | 晚上 | 通过chessGame测试 | 完成 |
| 2020-03-17 | 下午 | 通过goGame测试 | 完成 |
| 2020-03-17 | 晚上 | 验收完成 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
|  |  |
|  |  |
|  |  |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

* 设计一个完整的软件，需要从底层的ADT设计起；
* 检验ADT的正确性后才能检测软件的正确性
* ADT的Spec在前期的设计非常重要，比implementation还要重要
* 要认真地写spec，也包括细节的功能和注释

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？
2. 使用泛型和不使用泛型的编程，对你来说有何差异？
3. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？
4. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？
5. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？
6. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？
7. 关于本实验的工作量、难度、deadline。
8. 《软件构造》课程进展到目前，你对该课程有何体会和建议？