

**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 | 张瑞豪 |
| 学号 | 1180800811 |
| 班号 | 1803002 |
| 电子邮件 | m17779349381@163.com |
| 手机号码 | 17779349381 |

**目录**

[1 实验目标概述 1](#_Toc29325498)

[2 实验环境配置 1](#_Toc29325499)

[3 实验过程 1](#_Toc29325500)

[3.1 Poetic Walks 1](#_Toc29325501)

[3.1.1 Get the code and prepare Git repository 1](#_Toc29325502)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc29325503)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc29325504)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc29325505)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc29325506)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc29325507)

[3.1.4.1 Make the implementations generic 2](#_Toc29325508)

[3.1.4.2 Implement Graph.empty() 2](#_Toc29325509)

[3.1.5 Problem 4: Poetic walks 2](#_Toc29325510)

[3.1.5.1 Test GraphPoet 2](#_Toc29325511)

[3.1.5.2 Implement GraphPoet 2](#_Toc29325512)

[3.1.5.3 Graph poetry slam 2](#_Toc29325513)

[3.1.6 Before you’re done 2](#_Toc29325514)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc29325515)

[3.2.1 FriendshipGraph类 2](#_Toc29325516)

[3.2.2 Person类 3](#_Toc29325517)

[3.2.3 客户端main() 3](#_Toc29325518)

[3.2.4 测试用例 3](#_Toc29325519)

[3.2.5 提交至Git仓库 3](#_Toc29325520)

[3.3 Playing Chess 3](#_Toc29325521)

[3.3.1 ADT设计/实现方案 3](#_Toc29325522)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 3](#_Toc29325523)

[3.3.3 ADT和主程序的测试方案 3](#_Toc29325524)

[4 实验进度记录 4](#_Toc29325525)

[5 实验过程中遇到的困难与解决途径 4](#_Toc29325526)

[6 实验过程中收获的经验、教训、感想 4](#_Toc29325527)

[6.1 实验过程中收获的经验和教训 4](#_Toc29325528)

[6.2 针对以下方面的感受 4](#_Toc29325529)

# 实验目标概述

根据实验手册简要撰写。

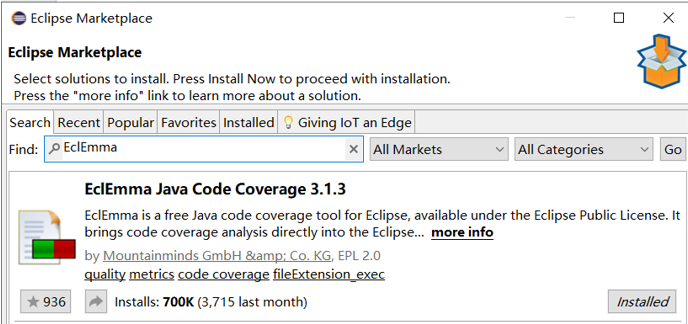
本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现ADT。具体来说：

* 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

## 配置EclEmma

1. 在Eclipse中选择 Help -> Marketplace
2. 搜索EclEmma，并且Install



1. 选择最新版进行安装即可。
2. 安装好后Eclipse菜单会出现图形表示安装成功



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/Lab2-1180800811>

# 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Poetic Walks

这个实验主要是让我们来掌握泛型的设计，并进一步理解抽象数据类型的设计、测试和规约，使用OOP技术来实现ADT，同时让我们进一步深入掌握测试优先的思想。这个任务还考察Java的一些集合类的使用，比如set、list、map等等。总的来说，这个任务的工作量很大，要设计的测试很多，但是难度不是很大，主要在于测试类的编写。

### Get the code and prepare Git repository

如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

**获取任务代码**:打开老师在lab2实验提供的网址，直接下载即可。

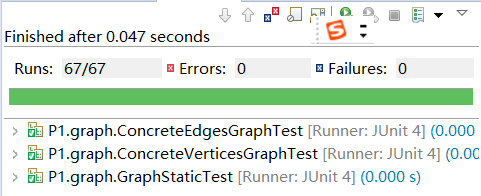
**在本地创建git仓库：**先git init 初始化创建一个本地仓库，然后git clone

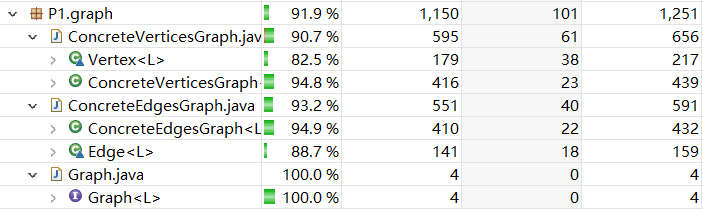
<https://github.com/ComputerScienceHIT/Lab2-1180800811.git> 即将本地仓库和远程仓库关联。

**使用git管理开发：**先使用git add 命令将lab2文件夹加入到缓冲区，再使用git commit将文件夹上传到本地仓库，再使用git push 将文件推送到远程仓库。

### Problem 1: Test Graph <String>

分别测试 add、set、remove、vertices、sources、targets 方法





**1. Test add**

**测试策略**：

graph: empty , not empty

vertex: existed , not existed

**结果**：返回true: 添加点成功。 原来的点集上会增加这个新加入的顶点，

返回false: 添加点失败。说明原来的点集包含这个待加入的顶点

**2. Test set**

**测试策略：**

graph: empty , not empty

source: existed , not existed

target: existed , not existed

weight: zero , positive

edge : existed , not existed

**结果：** 返回0 ：说明在原图中source 和 target 之间不存在边。若weight > 0 ，则在source 和 target 之间添加一条权重为weight的边；若weight = 0 ，

返回大于0的数 ：说明在原图中source 和 target 之间存在边，返回值即为原来的边的权重。若weight > 0 ,将原来的边的权重修改为weight ；若weight = 0 ，删去这条边。

**3. Test remove**

**测试策略：**graph : empty , not empty

vertex : not existed , existed with edges , existed without edges

**结果:** 返回true ：删除点成功，如果这个点原来有边相邻接，所有与这个点邻接的边全部删去。否则只删去该顶点。

返回false : 删除顶点失败， 说明原图中不存在该顶点。顶点集数目不变。边集数目不变。

**4. Test vertices**

**测试策略：**

**graph** ： empty ， not empty

**结果 ：** 返回图中所有的点的点集

**5 .Test source**

**测试策略：**

graph : empty , not empty

target ： existed with source to ,existed without source to ,not existed

**结果 ：** 返回所有以target顶点为target的顶点的点和权重的map集。

6. Test target

**测试策略：**

graph : empty , not empty

target ： existed with target to ,existed without target to ,not existed

**结果 ：** 返回所有以source顶点为source的顶点的点权重的map集。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

#### 1. 设计不变量

// Abstraction function:

// AF(vertices, edges) = a set of weighted directed edges

// from one source vertex to one target vertex that is different from source vertex

//

// Representation invariant:

// weight > 0,There are at most one directed edge between source and target

//

// Safety from rep exposure:

// vertices is mutable, so vertices() make defensive copy to avoid rep exposure

// and each of the fields are modified by key word final and private , so they can't be changed from outside

**2. 检查表示**

**public** **void** checkRep() {

**assert** vertices!= **null** ;//检查顶点集是否为空

**for**(L l : vertices) {

**assert** l != **null** ;//检查顶点是否为空

}

**for**(Edge<L> l : edges) {

**assert** l != **null** ;

**assert** l.getWeight() > 0;//检查权重是否大于零

}

}

**3.实现Edge类**

**3.1 实现Edge类的属性**

|  |  |
| --- | --- |
| **private** **final** L source | **边的入顶点** |
| **private** **final** L source | **边的出顶点** |
| **private** **final** **int** weight | **边的权重** |

**3.2 实现Edge类的方法**

|  |  |
| --- | --- |
| **public** Edge(L source,L target,**int** weight) | 构造方法 |
| **public** **void** checkRep() | 检查方法 |
| **public** L getTarget() | 得到入点 |
| **public** L getSource() | 得到出点 |
| **public** **int** getWeight() | 得到权重 |
| **public** **int** hashCode() | 重写hashCode方法 |
| **public** **boolean** equals(Object other) | 类的比较 |
| **public** String toString() | 类的toString表示 |

**4. 实现 ConcreteEdgesGraph<String>类**

**4.1 实现 ConcreteEdgesGraph<String>类的属性**

|  |  |
| --- | --- |
| **private** **final** Set<L> vertices = **new** HashSet<>(); | 图的所有顶点的集合 |
| **private** **final** List<Edge> edges = **new** ArrayList<>(); | 图的所有的边的集合 |

**4.2 实现 ConcreteEdgesGraph<String>类的方法**

|  |  |
| --- | --- |
| **public** ConcreteEdgesGraph() | 构造器初始化 |
| **public** **boolean** add (L vertex) | 添加顶点 |
| **public** **int** set(L source, L target, **int** weight) | Weight = 0 ，删除边， weight>0,修改边的权重，返回原来边的权重，或者返回0 |
| **public** **boolean** remove(L vertex) | 删除顶点 |
| **public** Set<L> vertices() | for 循环遍历，进行防御性复制，返回所有顶点集合 |
| **public** Map<L, Integer> sources(L target) | 遍历所有的边，返回入边顶点及其权重的集合 |
| **public** Map<L, Integer> targets(L source) | 遍历所有的边，返回出边顶点及其权重的集合 |
| **public** String toString() | 重写该类的 toString 方法 |
| **public** **public** **void** checkRep() | 检查方法 |

#### 3.1.3.2Implement ConcreteVerticesGraph

#### 1. 设计不变量

// Abstraction function:

// **TODO**

// Representation invariant:

// Each Vertex in vertices is different from other

//

// Safety from rep exposure:

// vertices is modified by keyword private and final

// so it can't be changed from outside

**2. 检查表示**

**public** **void** checkRep() {

**assert** vertices != **null** ;

**for**(Vertex<L> l : vertices) {

**for**(Integer weight : l.getRelationship().values())

**assert** weight >0; //权重大于零

}

**int** size = vertices.size();

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

**for**(**int** j = i+1 ; j <size ; j ++ ) {

**assert** !vertices.get(i).equals(vertices.get(j));//顶点各不相同

}

}

}

**3 实现Vertex类**

**3.1 实现Vertex类的属性**

|  |  |
| --- | --- |
| **private String source** | **边的入点** |
| **private Map<L, Integer> relationMap = new HashMap<>();** | **入点的所有出点集合以及边的权重** |

**3.2 实现Vertex类的方法**

|  |  |
| --- | --- |
| **public** Vertex(**final** L source ,**final** Map<L,Integer> map) | 带参数的构造器 |
| **public** Vertex(**final** L source ) | 不带参数的构造器 |
| **public** **void** checkRep() | 检查方法 |
| **public** L getSource() | 返回源顶点 |
| **public** **void** addEdge(**final** L target , **final** **int** weight) | 在源顶点和target顶点加一条边 |
| **public** **void** remove(**final** L target) | 删去与源顶点邻接的一个顶点及其边 |
| **Public** Map<L,Integer> getRelationship() | 返回该顶点的所有邻接顶点及其权重的map集 |
| **public** **int** getWeight(**final** L target ) | 返回源顶点与target顶点的权重 |
| **public** **boolean** equals(Object other) | 判断两个顶点是否相等 |
| **public** **int** hashCode() | 重写hashcode方法 |
| **public** String toString() | 重写类的tostring方法 |

**4. 实现ConcreteVerticesGraph <String>类**

**4.1 实现ConcreteVerticesGraph <String>类的属性**

|  |  |
| --- | --- |
| **private** **final** List<Vertex<L>> vertices = **new** ArrayList<>() | 所有的顶点集合 |

**4.2 实现ConcreteVerticesGraph <String>类的方法**

|  |  |
| --- | --- |
| **public** ConcreteVerticesGraph() | 构造器 |
| **public** **void** checkRep() | 检查方法 |
| **public** **boolean** add(L vertex) | 添加顶点 |
| **public** **int** set(L source,L target, **int** weight) | Weight = 0 ，删除边， weight>0,修改边的权重，返回原来边的权重，或者返回0 |
| **public** **boolean** remove(L vertex) | 删除顶点 |
| **public** Set<L> vertices() | for 循环遍历，进行防御性复制，返回所有顶点集合 |
| **public** Map<L, Integer> sources(L target) | 遍历所有的边，返回入边顶点及其权重的集合 |
| **public** Map<L, Integer> targets(L source) | 遍历所有的边，返回出边顶点及其权重的集合 |
| **public** String toString() | 重写该类的 toString 方法 |

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

#### Make the implementations generic

1.将具体类的声明修改为：



2.更新两个实现以支持任何类型的顶点标签，使用占位符 L 代替 String（如传入参数、返回值等）。

#### Implement Graph.empty()

**public** **static** <L> Graph<L> empty() {

**return** **new** ConcreteEdgesGraph<L>();

}

### Problem 4: Poetic walks

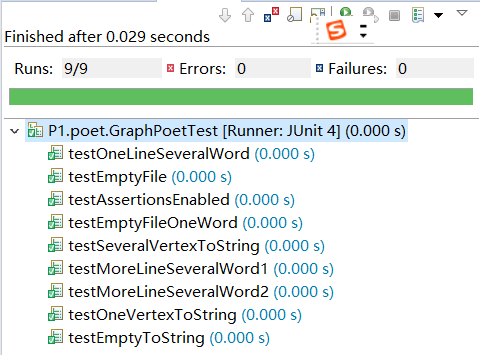
#### Test GraphPoet

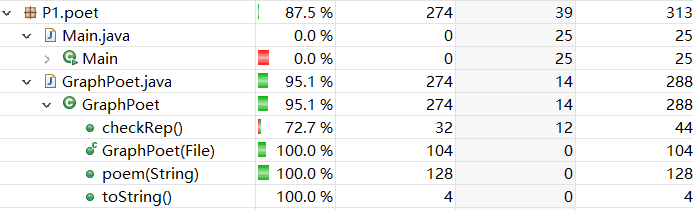
**测试策略：**

corpus : empty file , file with one line , file with several lines

Input : empty string , one word ,several words

toString: empty graph , graph with one vertex , graph with several vertex





#### Implement GraphPoet

1. 表示不变量

// Abstraction function:

// GraphPoet represents a word affinity graph which is generated with a corpus

// and vertices of is are case-insensitive words

// and edge weights of it are in-order adjacency counts.

// Representation invariant:

// graph != null , vertices of the graph not empty case-insensitive strings

// Safety from rep exposure:

// All fields are modified by private and final so that

// clients can't access the graph reference outside the class

1. 检查不变量

**public** **void** checkRep() {

**assert** graph != **null** ;

**for**(String vertex : graph.vertices()) {

**assert** !vertex.equals("");//判断顶点是非为空内容

**assert** vertex.equals(vertex.toLowerCase());//判断顶点是否都转换为小写

}

}

3.GraphPoet方法

将文件中文本按照空格进行划分（调用String.split函数），将得到的所有单词加入

到graph中，同时加边，权重是该边出现的次数。

4.poem方法

将输入的文本按照空格进行划分（ 调用String.split函数 ），然后找到这些单词

中相邻单词的桥接单词：一个单词调用targets方法，另一个单词调用sources

方法，进行比对是否含有相同的单词；若两个单词含有多个桥接词，则进行选

择一个权值较大者。同时选择StingBuilder将输出的所有单词拼接在一起（加

上空格

5. toString方法

直接调用graph的tostring方法

#### Graph poetry slam

测试策略

// Testing strategy

//

// Partition for the constructor of GraphPoet

// corpus : empty file , file with one line , file with several lines

//Partition for GraphPoet.poem

// input : empty string , one word , more than one word

//

//Partition for GraphPoet.toString

// input : empty graph , one vertex ,more than one vertex

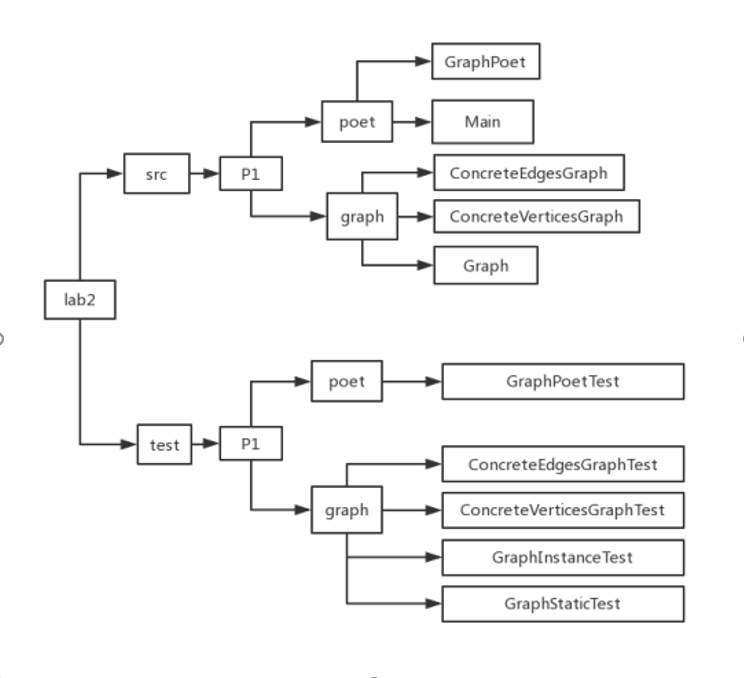
测试方法详见代码

### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序。

1. git init 初始化建立一个本地仓库
2. git clone <https://github.com/ComputerScienceHIT/Lab2-1180800811.git>将远程仓库和本地仓库关联
3. git add \* , 将Lab2里面所有文件添加到缓冲区
4. git commit -m “第一次提交” , 将缓冲区的文件提交到本地仓库
5. git push 将文件推送到远程仓库

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

主要就是利用本次实验P1中实现的图来实现Lab1中的人际关系图，getDistance利用广度优先的原理。

### FriendshipGraph类

方法

|  |  |
| --- | --- |
| **public** **boolean** addVertex(Person p) | 在人际关系图中添加一个顶点 |
| **public** **boolean** addEdge(Person p1 , Person p2) | 在人际关系图中的两个顶点之间添加一条边 |
| **public** **int** getDistance(Person person1, Person person2) | 给出人际关系图中两个顶点之间的最短距离。 |

**1.public** **boolean** addVertex(Person p)

首先判断原来的图中是否包含了该顶点，然后再调用ConcreteEdgesGraph<Person>类的add方法添加一个顶点即可。

**2.public** **boolean** addEdge(Person p1 , Person p2)

首先判断顶点p1和p2是否相同，然后直接可以调用ConcreteEdgesGraph<Person>类的set方法即可在顶点p1和p2添加一条权重为1的边。

**3.public** **int** getDistance(Person person1, Person person2)

①利用广度优先的原理，先将p1这个顶点压入队列中，再考察和p1邻接的所有顶点，如果邻接的顶点没有visit而且这个顶点恰好就是p2，直接输出距离即可。否则将这个没有visit的顶点压入队列。

②每次循环时，将队首元素取出并且从队列删除，以这个元素重新进行步骤①即可，循环直至队列为空。

**final** Map<Person, Integer> distance = **new** HashMap<>();//每个顶点到p1这个顶点的距离的map映射关系

**final** Map<Person, Boolean> visited = **new** HashMap<>();//判断每个顶点是否被访问过

Queue<Person> queue = **new** LinkedList<>();//队列

### Person类

1.属性

**private** String name ;//记录person的名字

2.方法

|  |  |
| --- | --- |
| Person(String name) | 构造器方法，初始化对象 |
| **public** String getName() | 得到person的名字 |
| **public** **boolean** equals(Object other) | 判断两个person是否相等 |
| **public** **int** hashCode() | 重写hashCode方法 |

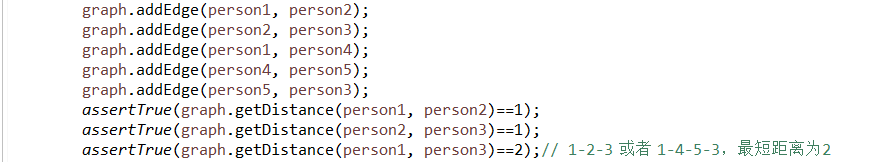
### 客户端main()



### 测试用例

测试策略：

|  |  |
| --- | --- |
| test addVertex | 加入的顶点在原来图中存在、加入的顶点在原来图中不存在、加入的顶点在原图中不存在，但是和原图中某个顶点重名 |
| test addEdge | 两个顶点相同、两个顶点不同 |
| test getDistance | 顶点到自身的距离、不连通的顶点的距离、两个顶点有多条路径去最短距离 |
|  |  |



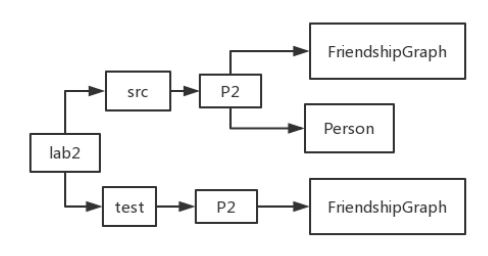
### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab3仓库。

1. git add \* , 将Lab2里面所有文件添加到缓冲区
2. git commit -m “第二次提交” , 将缓冲区的文件提交到本地仓库

③ git push 将文件推送到远程仓库

在这里给出你的项目的目录结构树状示意图。



## Playing Chess

### ADT设计/实现方案

设计了哪些ADT（接口、类），各自的rep和实现，各自的mutability/ immutability说明、AF、RI、safety from rep exposure。

必要时请使用UML class diagram（请自学）描述你设计的各ADT间的关系。

#### Position 类

1.属性

|  |  |
| --- | --- |
| **private** **int** x = -1 ; | 横坐标 |
| **private** **int** y = -1 ; | 纵坐标 |

// Abstraction Function:

// AF(x,y) = Represent a position with abscissa of x and ordinate of y

//

//Representation invariant:

// x and y are int

//

//Safety from rep exposure:

// the field of x and y is private

2.方法

|  |  |
| --- | --- |
| **public** **void** setPosition(**int** x,**int** y) | 设置位置的横纵坐标 |
| **public** **int** getX() | 得到位置的横坐标 |
| **public** **int** getY() | 得到位置的纵坐标 |
| **public** String toString() | 重写Position类的toString方法 |
| **public** **boolean** EqualPosition(Position s) | 判断两个位置是否相同 |
| **public** **boolean** FanWei(String type) | 判断某个位置是否超出了棋盘范围 |

#### Player类

1.属性

|  |  |
| --- | --- |
| **private** String Name | Player的名字 |

// Abstraction Function:

// Represent a name of a Person that is not null

//

//Representation invariant:

// Name != null

//

//Safety from rep exposure:

// name is immutable so there is no need for getName() to make defensive copy

// the field of name is private

2.方法

|  |  |
| --- | --- |
| **public** Player(String Name) | 构造器方法，初始化 |
| **public** String getName() | 得到player的名字 |

#### Piece类

1.属性

|  |  |
| --- | --- |
| **private** Position position = **new** Position() | 棋子的位置 |
| **private** String PieceName = **new** String() | 棋子的名字 |
| **private** **boolean** Owner | 棋子的拥有者，true为player1 |

// Abstraction Function:

// AF(position,PieceName,Owner) = Represent the position of the piece, the name of the piece and the owner of the piece

//Representation invariant:

// PieceName != null , position!=null

//

//Safety from rep exposure:

// PieceName is immutable so there is no need for getName() to make defensive copy

// All field is private

2.方法

|  |  |
| --- | --- |
| **public** Piece(**int** x ,**int** y , String Piecename, **boolean** owner ) | 初始化一颗棋子的构造器方法 |
| **public** **void** setPiecePosition(**int** px, **int** py) | 设置棋子的坐标 |
| **public** Position getPosition() | 得到棋子的位置 |
| **public** **boolean** getOwner() | 得到棋子的拥有者 |
| **public** **void** setPlayer(**boolean** player) | 设置棋子的拥有者 |
| **public** String getName() | 得到棋子的名字 |
| **public** **void** setPieceName(String pieceName) | 设置棋子的名字 |

#### Board类

1. 属性

|  |  |
| --- | --- |
| **private** **boolean**[][] goBoard1 = **new** **boolean**[19][19] ; | 围棋棋盘的占用情况， true表示被占用，false表示未被占用 |
| **private** **boolean**[][] goBoard2 = **new** **boolean**[19][19] ; | 围棋棋手的占用情况，true表示被player1占用，false表示被player2占用 |
| **private** **boolean**[][] chessBoard1 = **new** **boolean**[8][8]; | 象棋棋盘的占用情况， true表示被占用，false表示未被占用 |
| **private** **boolean**[][] chessBoard2 = **new** **boolean**[8][8]; | 象棋棋手的占用情况，true表示被player1占用，false表示被棋手2 |

// Abstraction Function:

// AF(goBoard1,goBoard2)= goBoard1 and goBoard2 represent the Board of go , goBoard1[i][j] represent whether the position of (i,j) is occupied by go piece,goBoard2[i][j] represent whose piece occupy the position of (i,j)

// AF(chessBoard1,chessBoard2)= chessBoard1 and chessBoard2 represent the Board of chess , chessBoard1[i][j] represent whether the position of (i,j) is occupied by chess piece,chessBoard2[i][j] represent whose piece occupy the position of (i,j)

//Representation invariant:

// goBoard1

//Safety from rep exposure:

// All field is private

2.方法

|  |  |
| --- | --- |
| **public** Board(String type) | 初始化一个棋盘的构造器方法 |
| **public** **void** setGoPiece(Piece piece) | 将一个棋子落在围棋棋盘上 |
| **public** **void** RemoveChessPiece(Piece piece) | 将一个棋子从象棋棋盘移除 |
| **public** **void** RemoveGoPiece(Piece piece) | 将一个棋子从围棋棋盘移除 |
| **public** **void** setChessPiece(Piece piece) | 将一个棋子落子象棋棋盘上 |
| **public** **boolean** getZhanYong(Position position , String type ) | 判断某个位置在某种棋盘上是否被占用 |
| **public** **boolean** getWhoZhanYong(Position position ,String type) | 得到某个被占用的位置的棋子是哪个player，true表示player1 |

#### Action类

1.属性

无

2.方法

|  |  |
| --- | --- |
| **public** **boolean** checkMove(**int** x, **int** y , **int** px , **int** py , Boolean player , Board chessboard) | 判断在象棋棋盘上移动棋子能否成功 |
| **public** **boolean** checkeatPiece(**int** x, **int** y , **int** px , **int** py , Boolean player , Board chessboard) | 判断象棋吃子能否成功 |
| **public** **boolean** checkTiPiece(**int** x , **int** y ,**boolean** player, Board goBoard ) | 判断围棋提子能否成功 |
| **public** **boolean** checkLuoPiece(**int** x , **int** y , Board goBoard,**boolean** player) | 判断围棋落子能否成功 |

**1.public** **boolean** checkMove(**int** x, **int** y , **int** px , **int** py , Boolean player , Board chessboard)

象棋移动棋子失败情况:

// 原位置和目标位置相同、目标位置被棋子占用、

// 当前的棋子不是被己方占用、当前需要移动的棋子的位置无棋子、

// 目标位置超出棋盘范围、当前棋子的位置超出棋盘范围

**2.public** **boolean** checkeatPiece(**int** x, **int** y , **int** px , **int** py , Boolean player , Board chessboard)

象棋吃子失败情况:

// 原位置和目标位置相同、目标位置的棋子不是敌方的

// 目标位置无棋子、当前的棋子不是被己方占用

// 当前需要移动的棋子的位置无棋子、目标位置超出棋盘范围

// 当前棋子的位置超出棋盘范围

**3.public** **boolean** checkTiPiece(**int** x , **int** y ,**boolean** player, Board goBoard )

围棋提子失败情况:

// 待提的子不是对方的棋子 、 待提子的位置无棋子、

// 待提子的位置超过了棋盘范围

**4.public** **boolean** checkLuoPiece(**int** x , **int** y , Board goBoard,**boolean** player)

围棋落子失败情况:

// 待落的子不是己方的棋子、待落子的位置已经有棋子

// 待落子的位置超过了棋盘范围

#### Game类

1.属性

|  |  |
| --- | --- |
| **private** String player1 = **new** String() | 玩家1 |
| **private** String player2 = **new** String() | 玩家2 |
| **private** Board chessBoard = **new** Board("chess") | 象棋棋盘 |
| **private** Board goBoard = **new** Board("go") | 围棋棋盘 |
| **private** String history1 = **new** String(); | 玩家1的下棋历史 |
| **private** String history2 = **new** String(); | 玩家2的下棋历史 |
| **private** List<Piece> pieces1 = **new** ArrayList<>() | 玩家1的所有棋子 |
| **private** List<Piece> pieces2 = **new** ArrayList<>() | 玩家2的所有棋子 |

2.方法

|  |  |
| --- | --- |
| **public** Game(String type) | 根据游戏类型初始化棋盘 |
| **public** **void** setPlayer1(String name) | 设置player1的名字 |
| **public** **void** setPlayer2(String name) | 设置player2的名字 |
| **public** String getPlayer1() | 得到player1 |
| **public** String getPlayer2() | 得到player2 |
| **public** **int** getSize(**boolean** player) | 得到player的棋子数目 |
| **public** String getHistory(**boolean** player) | 得到player的下棋历史 |
| **public** **boolean** LuoPieceGo(**int** x , **int** y ,**boolean** player ) | 在围棋棋盘上落子 |
| **public** **boolean** TiPieceGo(**int** x ,**int** y , **boolean** player ) | 在围棋棋盘上提子 |
| **public** **boolean** movePiece(**int** x ,**int** y ,**int** px, **int** py , **boolean** player) | 在象棋棋盘上移动棋子 |
| **public** **boolean** eatPiece(**int** x ,**int** y , **int** px , **int** py , **boolean** player ) | 在象棋棋盘上棋子吃子 |
| **public** Piece getPiece(Position position) | 根据棋子位置得到棋子 |
| **public** **void** addHistory(String s , **boolean** player) | 更改player的操作历史 |
| **public** **void** showBoard(String type) | 展示棋盘上棋子的分布 |

// Abstraction Function:

// Represent two players of the game ,and the type of the game ,

// the operation history of two players,and the pieces of two players

//Representation invariant:

// player1 != null ,player2 != null

//Safety from rep exposure:

// player1,player2,history1,history2 are immutable

// All field is private

#### MyChessAndGoGame类

MyChessAndGoGame类主要实现整个游戏。主要实现的是main方法。整体使用switch语句，通过获取用户终端输入的数字，进行选择实现的功能；其中象棋与围棋的功能略有所区别。具体main方法具体实现见代码。

### 主程序MyChessAndGoGame设计/实现方案

1.控制：运用switch语句，根据用户输入的数字来判断进行哪个操作。

2.玩家交替进行：每次操作都需要确定是哪个棋手操作，变量player为奇数表示当前是player2操作，为偶数表示player1进行操作。Player3表示目前操作的player的名字。

3.player变量：当操作是”吃子”、“移子”、“提子”、“落子”、“跳过”时，则player++，操作权转变。而当操作是“查询棋盘”、“查询棋子”、“查询棋子数目”、“历史”时，操作权不改变，player变量不会改变。

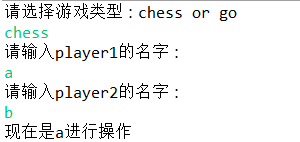
4.围棋棋子：围棋棋手1的棋子为“white”，围棋棋手2的棋子为“black”。

具体main方法流程

1. **首先，提示用户输入游戏类型->“chess” or “go”**



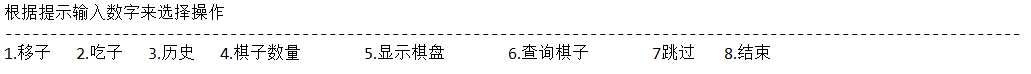
1. **再提示用户输入玩家的姓名**



1. **开始游戏，根据输入提示选择操作**

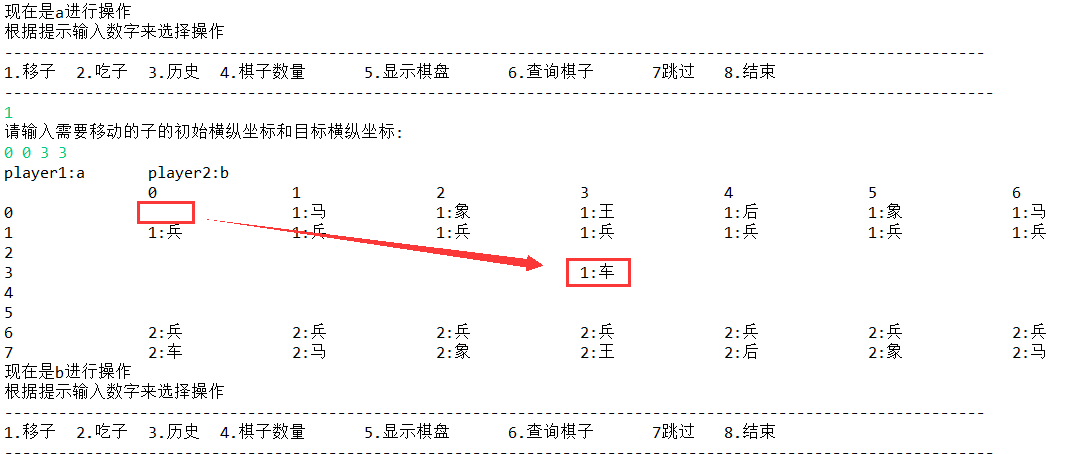
操作的具体实现见代码

**象棋操作**

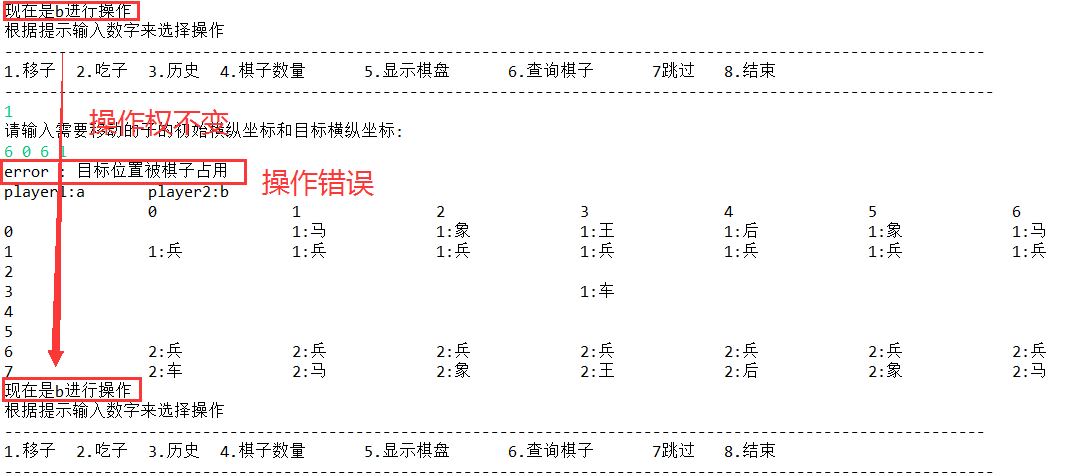


**1.移子**

给出需要移动的棋子的横坐标和纵坐标，以及目标位置的横坐标和纵坐标。

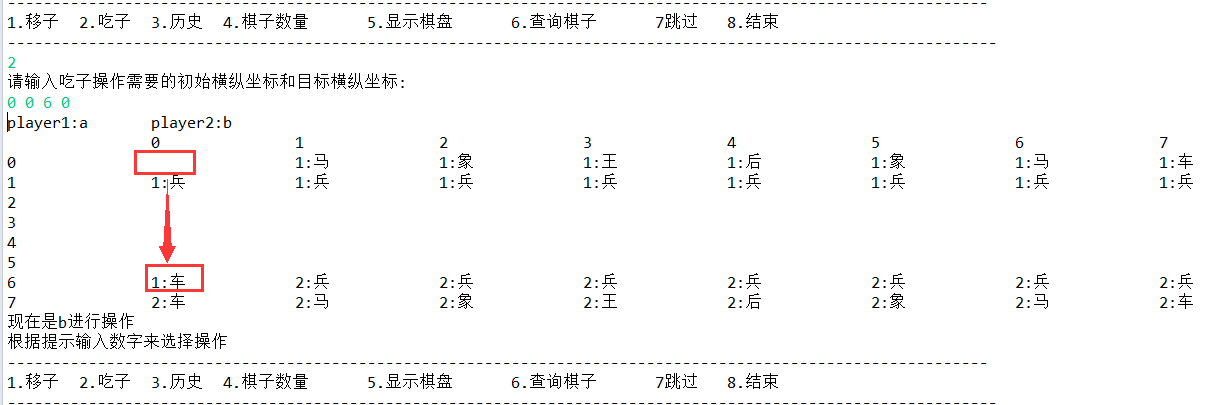


如果位置不合理则需重新进行操作，操作权不变

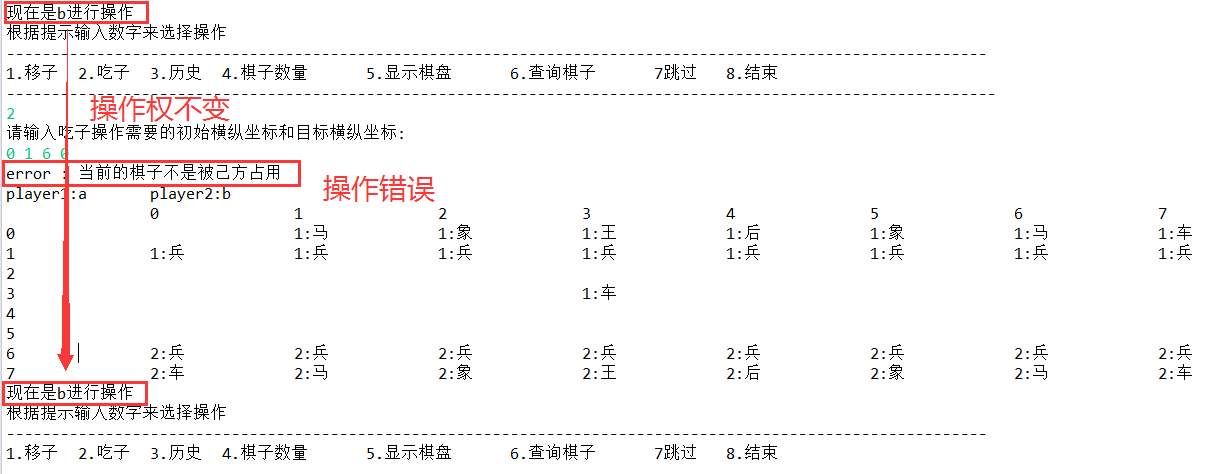


**2.吃子**

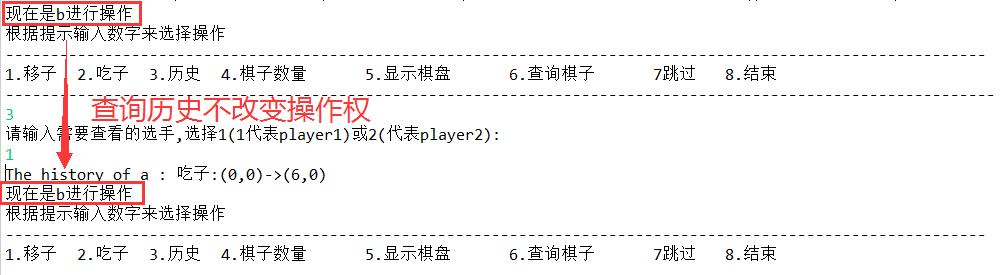
给出需要吃子的棋子的横坐标和纵坐标，以及被吃的子的横坐标和纵坐标。



如果位置不合理则需重新进行操作，操作权不变



**3.历史**

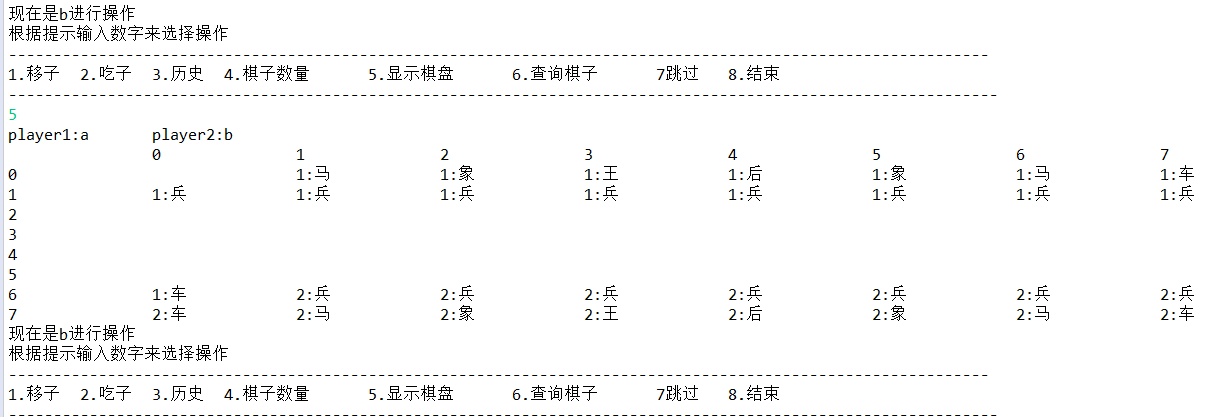


**4.棋子数量**



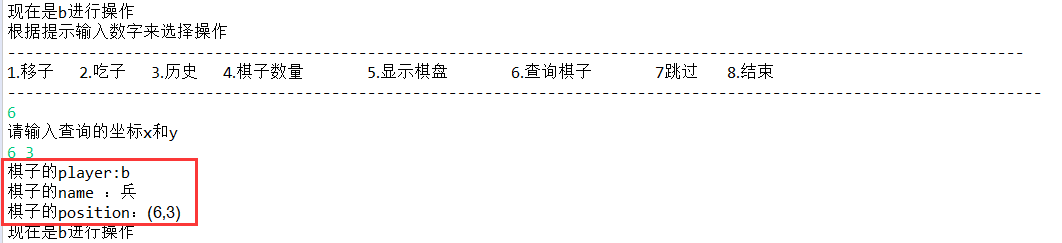
**查询棋子数量不改变操作权**

**5.象棋棋盘**

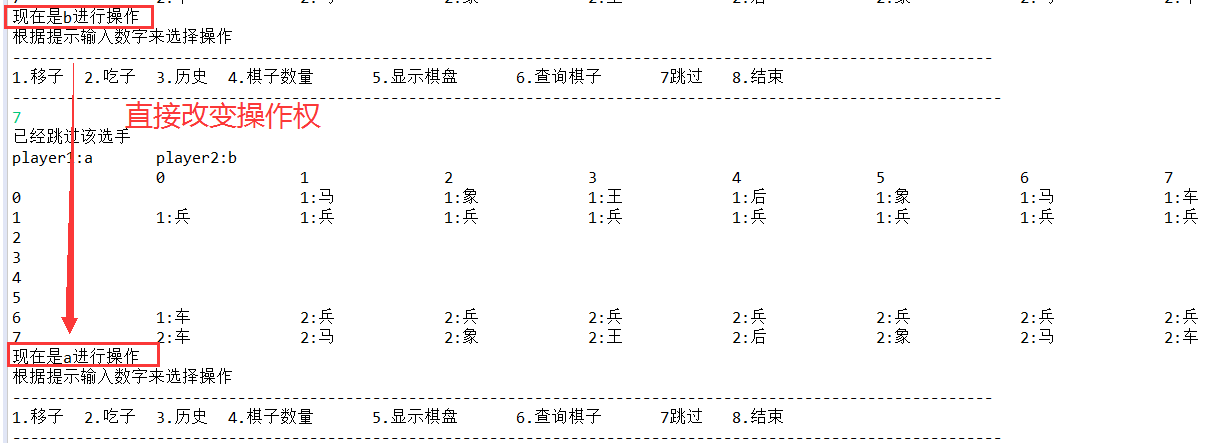


**6.查询棋子**

输入需要查询的棋子的横纵坐标即可，查询棋子操作不改变操作权

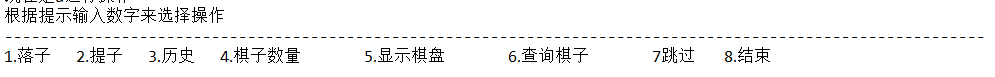


**7.跳过**



跳过操作，直接改变操作权即可

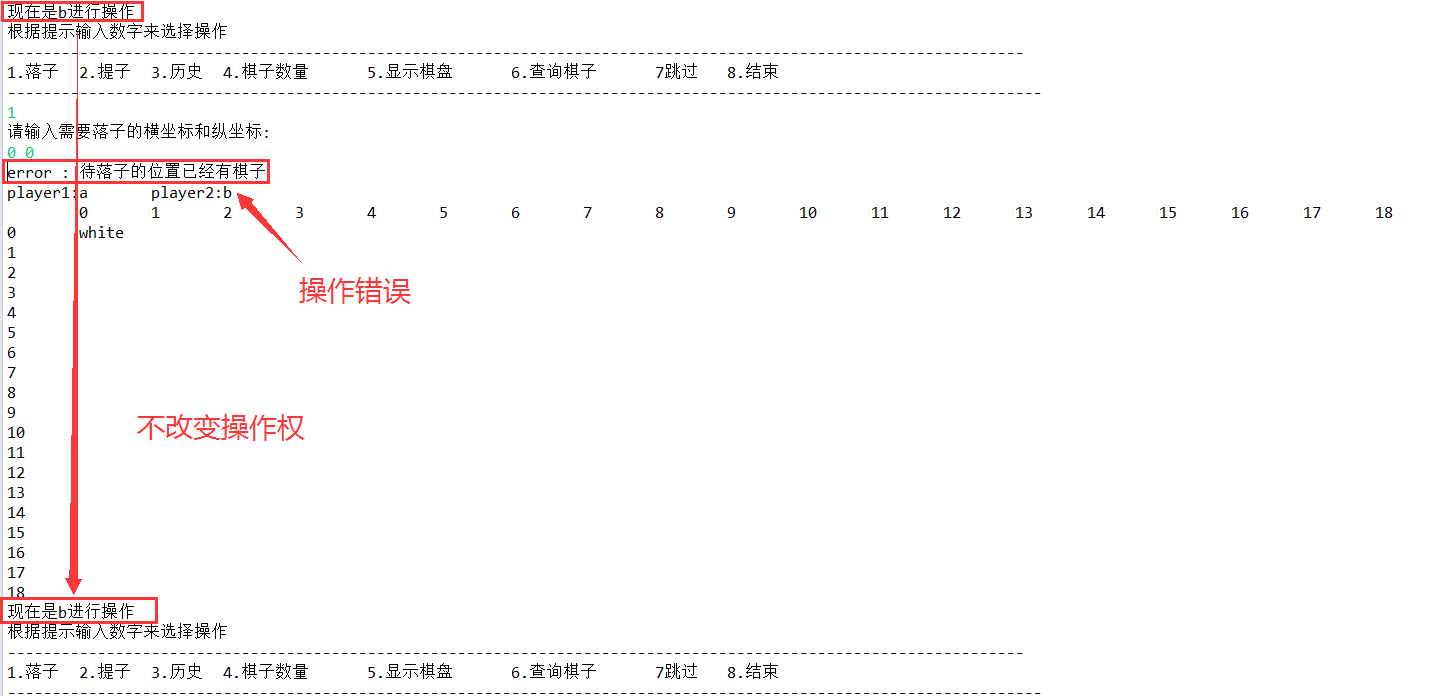
**围棋操作**



**1.落子**

输入需要落的子的横纵坐标即可。位置不合理需要重新输入





**2.提子**

输入需要提子的横纵坐标即可，位置不合理需要重新输入。





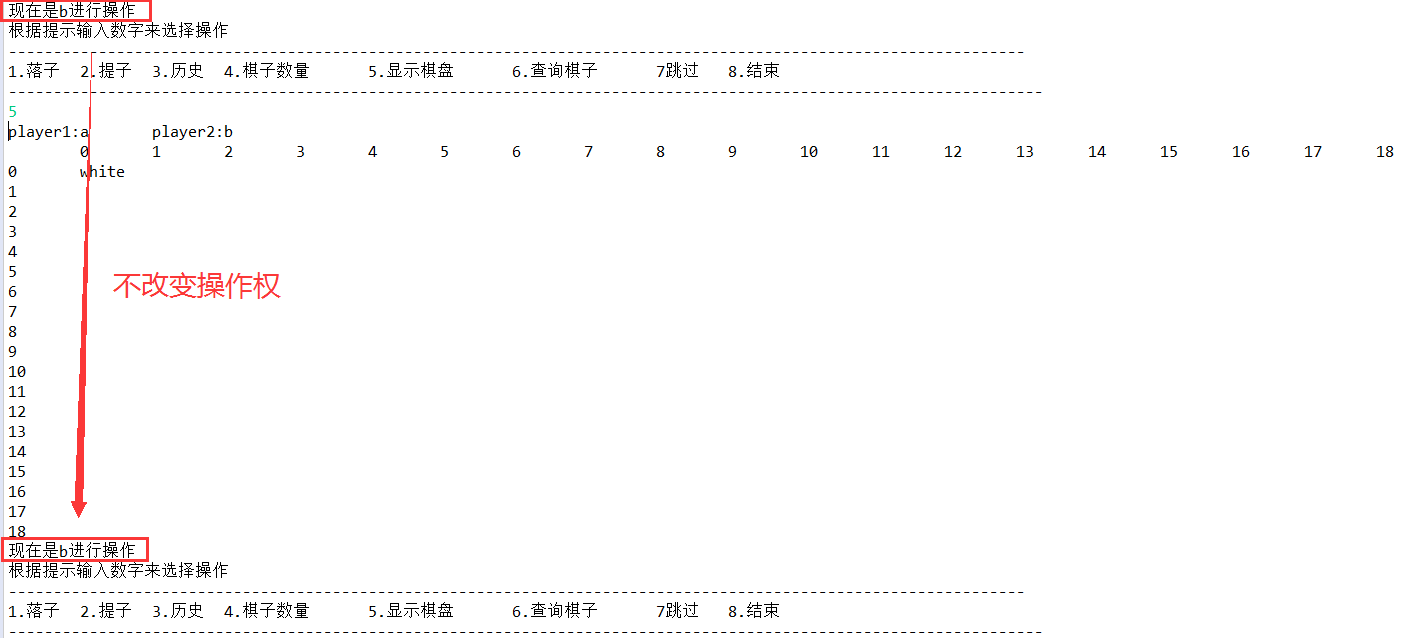
**3.历史**



**4.棋子数量**

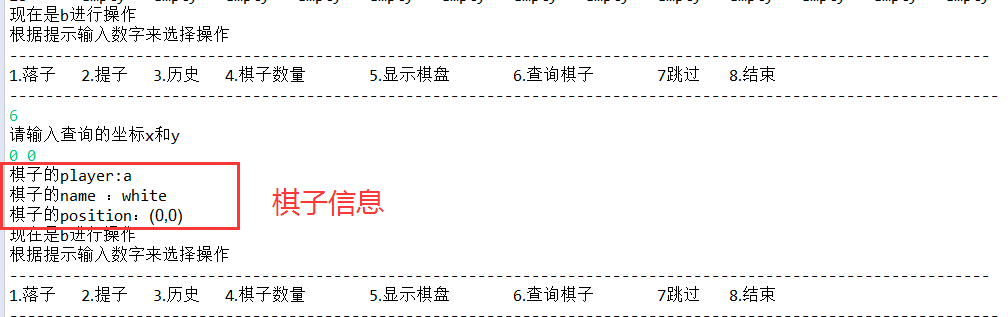


**5.围棋棋盘**

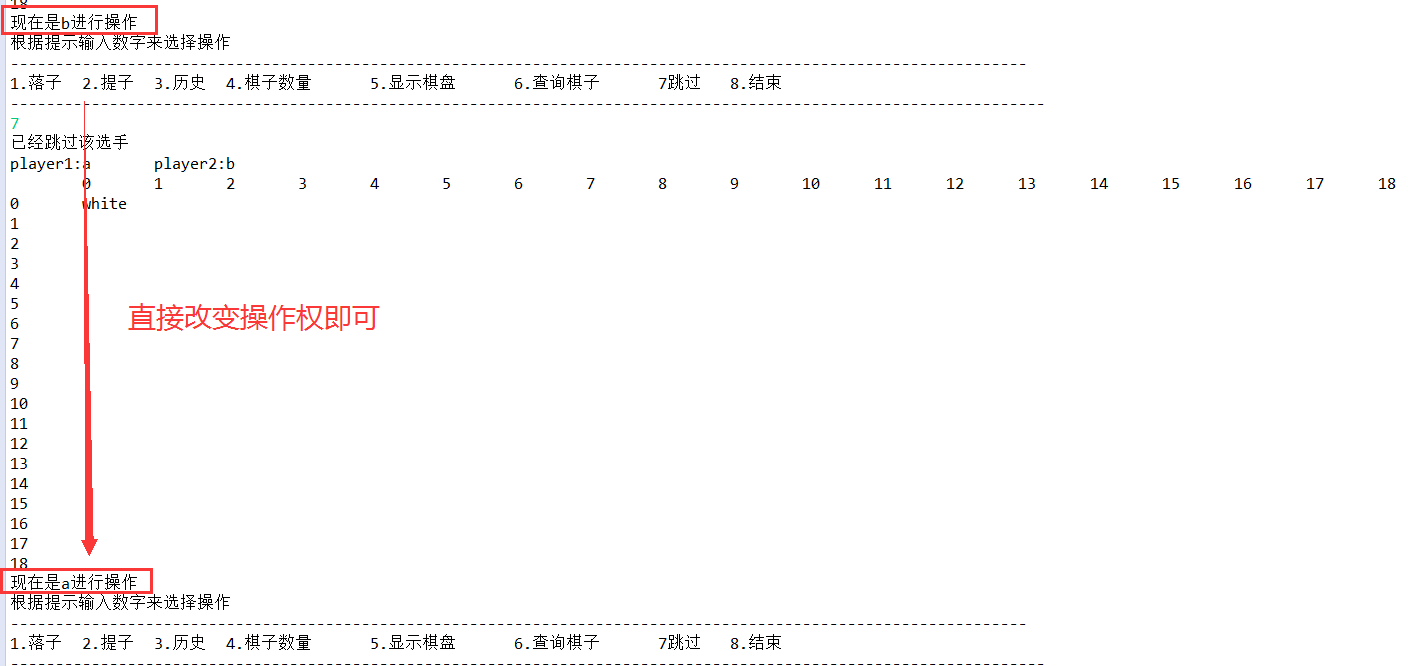


**6.查询棋子**

**输入需要查询的棋子的横纵坐标即可。**

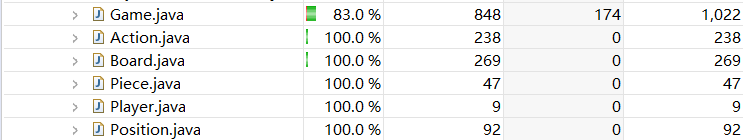


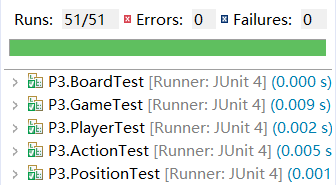
**7.跳过**



### ADT和主程序的测试方案

测试覆盖率





#### test Board类

testing strategy：

//Testing strategy

//Partition for Board.setGoPiece()

// input : go piece

//

//Partition for Board.setChessPiece()

// input : chess piece

//Partition for RemoveGoPiece()

// input : go piece

//

// Partition for RemoveChessPiece()

// input : chess piece

//

//Partition for getZhanYong()

// input : Position existed with piece , Position existed without piece

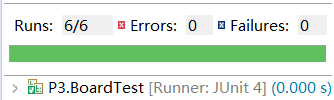
// Board : go board , chess board

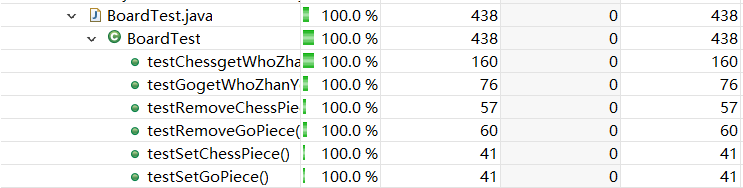
//

//Partition for getWhoZhanYong()

// Position existed with player1 , Position existed with player2

// Board : go board , chess board





#### test Action类

testing strategy：

//Testing strategy

//partition for checkMove()

// input : 原位置和目标位置相同、目标位置被棋子占用、

// 当前的棋子不是被己方占用、当前需要移动的棋子的位置无棋子、

// 目标位置超出棋盘范围、当前棋子的位置超出棋盘范围

// 目标位置和当前位置合理

//

//partition for checkeatPiece()

// input : 原位置和目标位置相同、目标位置的棋子不是敌方的

// 目标位置无棋子、当前的棋子不是被己方占用

// 当前需要移动的棋子的位置无棋子、目标位置超出棋盘范围

// 当前棋子的位置超出棋盘范围、 目标位置和当前位置合理

//

//partition for checkTiPiece()

// input : 待提的子不是对方的棋子 、 待提子的位置无棋子、

// 待提子的位置超过了棋盘范围、待提子的位置合理

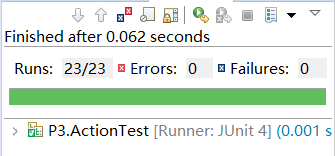
//

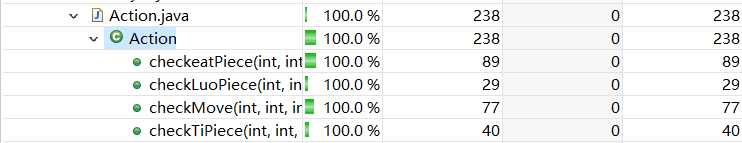
//partition for checkLuoPiece()

// input : 待落的子不是己方的棋子、待落子的位置已经有棋子

// 待落子的位置超过了棋盘范围、待落子的位置合理

//





#### test Game类

testing strategy：

// Testing strategy

// partition for LuoPieceGo()

// input : (x,y)位置合理，(x,y)位置不合理 ，

// player == true , player == false

//

// partition for TiPieceGo()

// input : (x,y)位置合理，(x,y)位置不合理 ，

// player == true , player == false

//

// partition for movePiece()

// input: (x1,y1),(x2,y2)位置合理，(x1,y1),(x2,y2)位置不合理

// player == true , ;player == false ;

//

// partition for eatPiece()

// input: (x1,y1),(x2,y2)位置合理，(x1,y1),(x2,y2)位置不合理

// player == true , ;player == false ;

//

// partition for getHistory()

// input : LuoPieceGo,movePiece,eatPiece,TiPiece,showBoard

// player : player1 , player2

//

// partition for getPiece()

// input : position with piece existed , position without piece existed

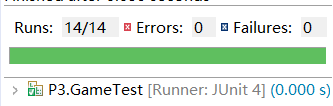
// board : goBoard , chessBoard

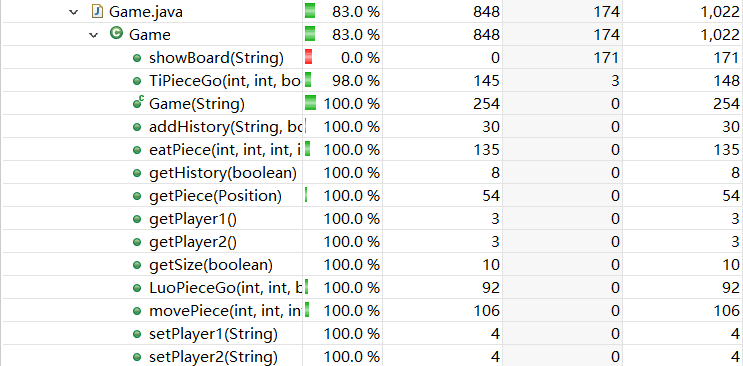
//

// partition for addHistory()

// input : null string , not null string

// player: true ,false





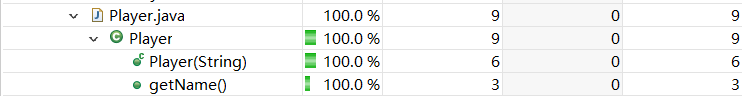
#### test Player类

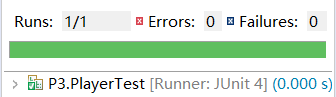
testing strategy：

//Testing strategy

//partition for Player.getName

// input : name





#### test Position类

testing strategy：

// Testing strategy

//

//partition for EqualPosition()

// input : equal position , not equal position

//

//partition for FanWei()

// input: board：go ,chess

// (x,y)超出范围，(x,y)未超出范围

//partition for setPosition()

// input : (x,y)超出范围，(x,y)未超出范围

//

//

//partition for getX()

// input : (x,y)

//

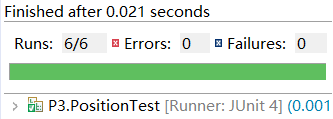
//partition for getY()

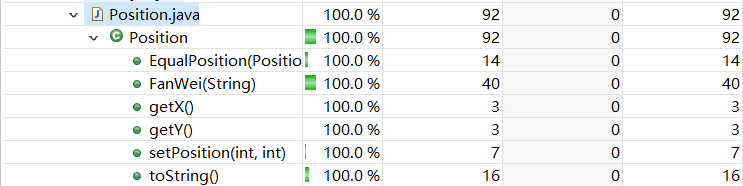
// input : (x,y)

//

//partition for toString()

// input :(x,y)





# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020-03-17 | 8:00-9:00 | 理解P1的题意 | 按时完成 |
| 2020-03-17 | 14:00-17:00 | 完成P1的测试用例 | 未完成 |
| 2020-03-19 | 19:00-21:30 | 完成P1GraphInstance测试用例 | 提前完成 |
| 2020-03-20 | 14:00-17:30 | 完成P1剩余测试用例 | 提前完成 |
| 2020-03-21 | 08:00-11:30 | 完成P1 | 延时完成 |
| 2020-03-21 | 19:00-22:30 | 完成P2 | 提前完成 |
| 2020-03-22 | 08:00-09:00 | 读懂P3题意 | 提前完成 |
| 2020-03-23 | 13:40-16:40 | 完成P3的Position、Player、Piece类 | 提前完成 |
| 2020-03-23 | 19:00-22:30 | 完成P3的Board类、以及Board类的测试用例和Action类的测试用例 | 未完成 |
| 2020-03-24 | 19:00-22:30 | 完成P3的Action类 | 提前完成 |
| 2020-03-25 | 全天 | 完成P3的Game类及其测试用例 | 未完成 |
| 2020-03-26 | 14:00-17:00 | 完成Game类及其测试用例 | 按时完成 |
| 2020-03-27 | 14:00-22:30 | 完成P3 | 提前完成 |
| 2020-03-27 | 20:00-22:30 | 完成报告 | 未完成 |
| 2020-03-28 | 全天 | 完成报告 | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| specification, invariants, RI, AF不会写 | 查阅博客，查阅老师上课的PPT |
| P1问题看不懂 | 结合Google翻译、老师讲解、别人的博客才搞懂 |
| P3问题bug太多 | 针对一个类的每个方法逐步进行测试。 |

# 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

1. 一定要先看懂题意，看懂每个实验的方法的具体含义。没看懂就下手会出很多错误
2. 一定要先写测试类，这次P3一开始没写测试类好多好多bug，后面写完测试类才慢慢的把bug修复。
3. 动手前一定要好好构思。

## 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？

面向ADT编程以对象为主体，当对象设计完成之后，应用实现起来更容易。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

具体实现的时候差异不大，但是应用的时候泛型设计的应用面明显要广太多

1. 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？

优势：知道规约之后，就可以知道出现错误的情况，这样就会根据规约内容编写测试类可以减少很多的时间去构思，同时也利于写出正确的程序。能适应。

1. 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？

应用面更广泛，可以满足更多的需求。

1. 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？

任务量特别大，需要构思的地方特别多。而且不同的类之间关联特别大，需要我们编写更详细的测试类来保证更少的错误发生。

1. 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？

意义: 撰写ADT的额使用规范等是方便别人了解自己的ADT，防治别人误用和破坏ADT。 时刻注意ADT是否有rep exposure为了防止变量泄露被外部人员改变。

1. 关于本实验的工作量、难度、deadline。

工作量很大，难度很大，deadline很充足。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

体会：这门课真的很好，学到了很多的东西。