

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 张景润 |
| 学号 | 1172510217 |
| 班号 | 1703002 |
| 电子邮件 | 2584363094@qq.com |
| 手机号码 | 18530272728 |

**目录**

[1 实验目标概述 1](#_Toc5564045)

[2 实验环境配置 1](#_Toc5564046)

[2.1 配置EclEmma 1](#_Toc5564047)

[2.2 Lab2仓库地址 2](#_Toc5564048)

[3 实验过程 2](#_Toc5564049)

[3.1 Poetic Walks 3](#_Toc5564050)

[3.1.1 Get the code and prepare Git repository 4](#_Toc5564051)

[3.1.2 Problem 1: Test Graph <String> 4](#_Toc5564052)

[3.1.3 Problem 2: Implement Graph <String> 5](#_Toc5564053)

[3.1.3.1 Implement ConcreteEdgesGraph 5](#_Toc5564054)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc5564055)

[3.1.4 Problem 3: Implement generic Graph<L> 8](#_Toc5564056)

[3.1.4.1 Make the implementations generic 8](#_Toc5564057)

[3.1.4.2 Implement Graph.empty() 9](#_Toc5564058)

[3.1.5 Problem 4: Poetic walks 9](#_Toc5564059)

[3.1.5.1 Test GraphPoet 9](#_Toc5564060)

[3.1.5.2 Implement GraphPoet 9](#_Toc5564061)

[3.1.5.3 Graph poetry slam 10](#_Toc5564062)

[3.1.6 Before you’re done 10](#_Toc5564063)

[3.2 Re-implement the Social Network in Lab1 11](#_Toc5564064)

[3.2.1 FriendshipGraph类 11](#_Toc5564065)

[3.2.2 Person类 12](#_Toc5564066)

[3.2.3 客户端main() 12](#_Toc5564067)

[3.2.4 测试用例 13](#_Toc5564068)

[3.2.5 提交至Git仓库 13](#_Toc5564069)

[3.3 Playing Chess 14](#_Toc5564070)

[3.3.1 ADT设计/实现方案 14](#_Toc5564071)

[3.3.1.1 Action类 14](#_Toc5564072)

[3.3.1.2 Board类 14](#_Toc5564073)

[3.3.1.3 Game类 15](#_Toc5564074)

[3.3.1.4 MyChessAndGoGame 16](#_Toc5564075)

[3.3.1.5 Piece类 17](#_Toc5564076)

[3.3.1.6 Player类 18](#_Toc5564077)

[3.3.1.7 Position类 18](#_Toc5564078)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 18](#_Toc5564079)

[3.3.3 ADT和主程序的测试方案 22](#_Toc5564080)

[3.4 Multi-Startup Set (MIT) 24](#_Toc5564081)

[4 实验进度记录 24](#_Toc5564082)

[5 实验过程中遇到的困难与解决途径 25](#_Toc5564083)

[6 实验过程中收获的经验、教训、感想 25](#_Toc5564084)

[6.1 实验过程中收获的经验和教训 25](#_Toc5564085)

[6.2 针对以下方面的感受 25](#_Toc5564086)

­­

# 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：

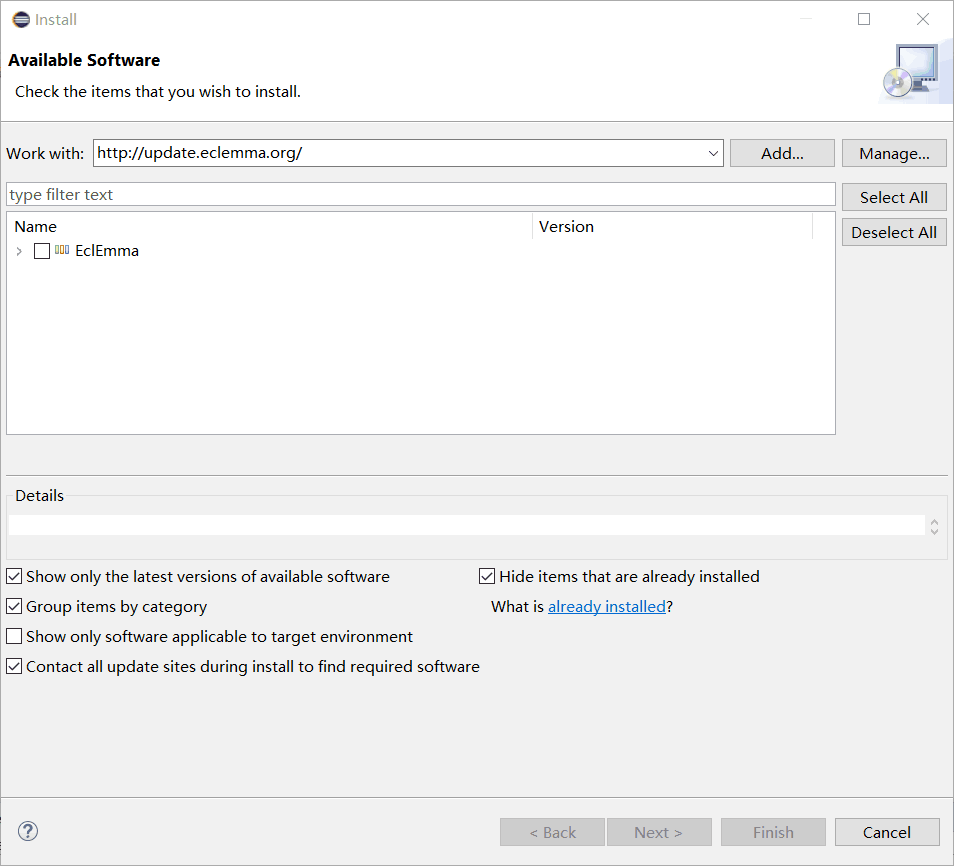
* 针对给定的应用问题，从问题描述中识别所需的ADT；
* 设计ADT规约（pre-condition、post-condition）并评估规约的质量；
* 根据ADT的规约设计测试用例；
* ADT的泛型化；
* 根据规约设计ADT的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
* 使用OOP实现ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
* 测试ADT的实现并评估测试的覆盖度；
* 使用ADT及其实现，为应用问题开发程序；
*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

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。

## Lab2仓库地址

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/Lab2-1172510217

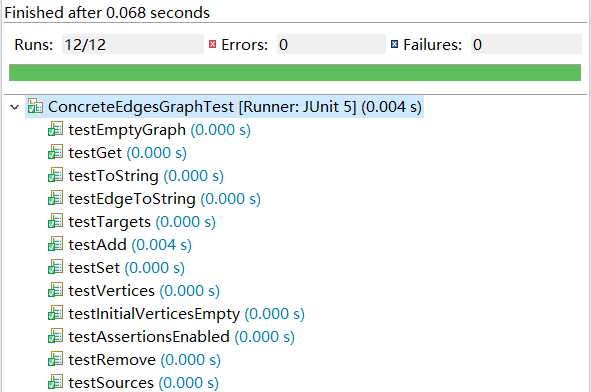
# 实验过程

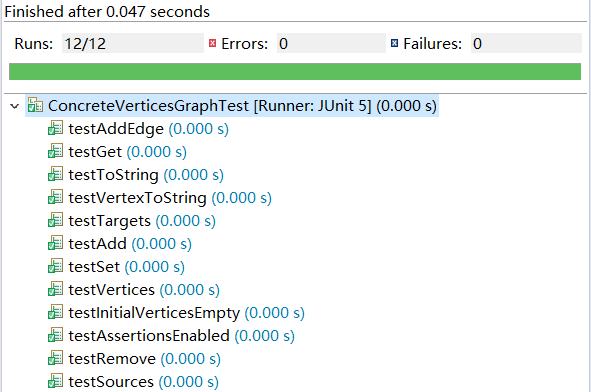
请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Poetic Walks

在这里简要概述你对该任务的理解。

* 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

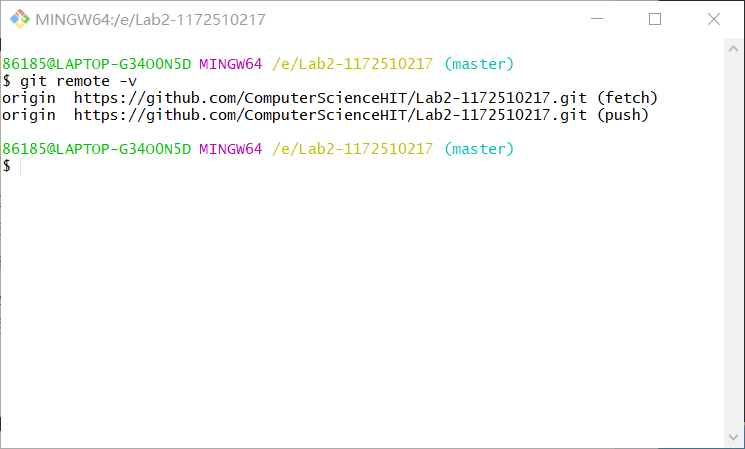
如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

* 获取代码：下载自  
  <https://github.com/rainywang/Spring2019_HITCS_SC_Lab2/tree/master/P1>
* 在本地创建git仓库

1. 利用Git Bash输入如下命令行

git clone <https://github.com/ComputerScienceHIT/Lab2-1172510217.git>

1. 查看远程仓库的信息。git remote -v



### Problem 1: Test Graph <String>

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

关于add方法：Testing strategy graph：空+非空vertex：待加边不存在+待加边存在 结果：若返回值为true，顶点数目++且顶点集中包含vertex；否则，顶点集不变且顶点集中包含vertex number of vertices increases by 1 else graph unmodified observe with vertices()；

关于set方法：

// Testing strategy

// graph：空+非空

// vertex：待设置顶点已经存在+不存在

// weight：为0时，删除；非零，修改

// 返回值：为0，边不存在；否则，边存在，且返回值是原来的权重

关于remove方法：

// Testing strategy

// graph：空+非空

// vertex：待删顶点存在于顶点集中+不存在；待删顶点存在于边中+不存在

// 结果：若返回值为false，不存在该顶点，顶点集数目不变，边集不变；否则，在顶点集或者边集存在该顶点，顶点集数目

关于Vertices方法：

// Testing strategy

// graph：空+非空

关于Sources方法：  
// Testing strategy

// graph：空+非空

// vertex：待寻找顶点存在于顶点集中+不存在；

// edges：待寻找顶点存在入边+不存在

// 结果：返回的map含有所有的入边

关于Targets方法：

// Testing strategy

// graph：空+非空

// vertex：待寻找顶点存在于顶点集中+不存在；

// edges：待寻找顶点存在出边+不存在

// 结果：返回的map含有所有的出边

最后完善ConcreteEdgesGraphTest类以及ConcreteVerticesGraphTest类

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

设计1：表示不变量

// Abstraction function:

// AF(vertices, edges) = weighted graph with directed edges and vertices are

// different

// Representation invariant:

// All the weight of the edges > 0

// Safety from rep exposure:

// fields are private and final in order to keep the codes safe

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

// edges is immutable

设计2：检查表示

**private** **void** checkRep() {

**assert** vertices != **null**;// 检查是否顶点集为空

**for** (L l : vertices) {// 检查是否加入了null顶点对象

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

}

**for** (Edge<L> edge : edges) {// 检查是否存在边权值不合格

**assert** edge != **null**;

**assert** edge.getWeight() > 0;

**assert** vertices.contains(edge.getSource());// 检查是否有野边

**assert** vertices.contains(edge.getTarget());// 检查是否有野边

}

}

­­实现Edge<String>类方法

|  |  |
| --- | --- |
| public String getSource | 得到入点 |
| public String getTarget | 得到出点 |
| public Integer getWeight | 得到权重 |
| public String toString | 类的toString表示 |

// Abstraction function:

// AF(source,target,weight) = an edge which is from source to target with a

// positive weight

// Representation invariant:

// weight > 0

// Safety from rep exposure:

// fields are private and final in order to keep the codes safe

// checkRep:check the rep invariant

设计4：实现ConcreteEdgesGraph<String>类属性

**private** **final** Set<L> vertices = **new** HashSet<>();

**private** **final** List<Edge<L>> edges = **new** ArrayList<>();

实现ConcreteEdgesGraph<String>类方法

|  |  |
| --- | --- |
| public Boolean add | 调用vertices的add方法 |
| public int set | 分情况判断：边存在、边不存在；weight为0，weight不为0 |
| public Boolean remove | 分情况判断：待删顶点存在于顶点集中、不存在；存在于边集中、不存在 |
| public Set<String> vertices | for循环遍历，进行防御性复制 |
| public Map<String,Integer> sources | 遍历所有的边，判断是否含有出点 |
| public Map<String,Integer> targets | 遍历所有的边，判断是否含有入点 |
| public(Override) String toString | 重写该类的toString方法 |

#### Implement ConcreteVerticesGraph

设计1：表示不变量

// Abstraction function:

// AF(vertices) = a directed weighted graph which has a map of its targets and

// weight

// Representation invariant:

// all the weight of the edges > 0

// Safety from rep exposure:

// vertices field is private and final;

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

设计2：检查表示

**private** **void** checkRep() {

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

**for** (Integer weight : vertex.getRelationMap().values()) {

**assert** weight > 0;

}

}

}

设计3：实现Vertex<String>类属性

|  |  |
| --- | --- |
| 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 |

// Abstraction function:

// a vertex with L and a map that record all the edges from it

// Representation invariant:

// all the weight > 0

// Safety from rep exposure:

// L is named final which is safe;

// the map is returned by copying a new map.

实现Vertex<String>方法

|  |  |
| --- | --- |
| public Vertex |  |
| public String getSource | 得到入点 |
| public void addEdge | 在出边集中加一个顶点 |
| public void deleteEdge | 删除一条边 |
| public Map<L, Integer> getRelationMap | 得到所有的出边 |
| public int getWeight | 得到权重 |
| public String toString | 类的toString表示 |

设计4：实现ConcreteVerticesGraph <String>类属性

**private** **final** List<Vertex<L>> vertices = **new** ArrayList<>();

// Abstraction function:

// AF(vertices) = a directed weighted graph which has a map of its targets and

// weight

// Representation invariant:

// all the weight of the edges > 0

// Safety from rep exposure:

// vertices field is private and final;

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

实现ConcreteVerticesGraph <String>类方法

|  |  |
| --- | --- |
| public Boolean add | 调用vertices的add方法 |
| public int set | 分情况判断：边存在、边不存在；weight为0，weight不为0 |
| public Boolean remove | 分情况判断：待删顶点存在于顶点集中、不存在；存在于边集中、不存在 |
| public Set<String> vertices | for循环遍历，进行防御性复制 |
| public Map<String,Integer> sources | 遍历所有的边，判断是否含有出点 |
| public Map<String,Integer> targets | 遍历所有的边，判断是否含有入点 |
| public(Override) String toString | 重写该类的toString方法 |

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

#### Make the implementations generic

* 将具体类的声明更改为：

public class ConcreteEdgesGraph<L> implements Graph<L> {...}

class Edge<L> { ... }

* 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

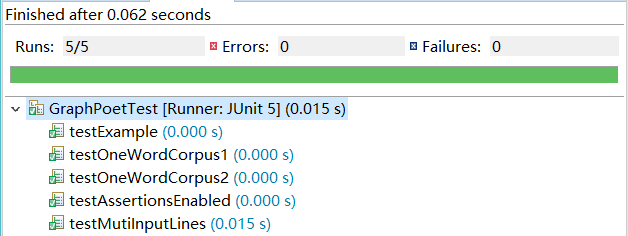
#### Test GraphPoet

测试用例均在test文件包的poet子文件里。

// Testing strategy

// corpus：一个单词+多个单词+一行单词+多行单词

// input：一个单词+多个单词+一行单词+多行单词+大小写



#### Implement GraphPoet

1. 表示不变量

// Abstraction function:

// a graph with the ConcreteEdgesGraph that represents a graph

// Representation invariant:

// vertices of the graph are non-empty case-insensitive strings

// of non-space non-newline characters

// Safety from rep exposure:

// graph field is private and final.

1. 检查不变量

**private** **void** checkRep() {

**for** (String vertex : graph.vertices()) {// 桥接词不区分大小写

String compare = vertex.toLowerCase();

**assert** vertex.equals(compare);

**assert** !vertex.equals("");// 证明顶点内容单词非空

}

}

1. Graph方法
2. public GraphPoet(File corpus)方法

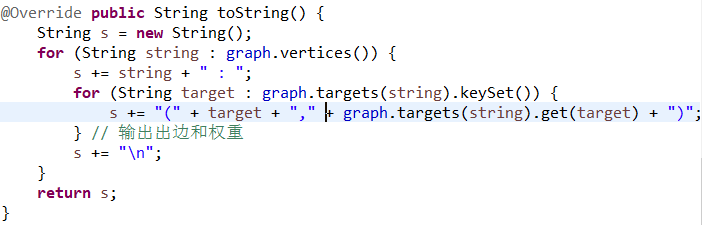
将文件中文本按照空格进行划分（调用String.split函数），将得到的所有单词加入到graph中，同时加边，权重是该边出现的次数

1. public String poem(String input)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）

1. public String toString方法

重写此方法。将graph的顶点和边以及权重输出表示。



#### Graph poetry slam

更多的实现用例在GraphPoet中的test类中均存在。

故在这里不再进行赘述

### Before you’re done

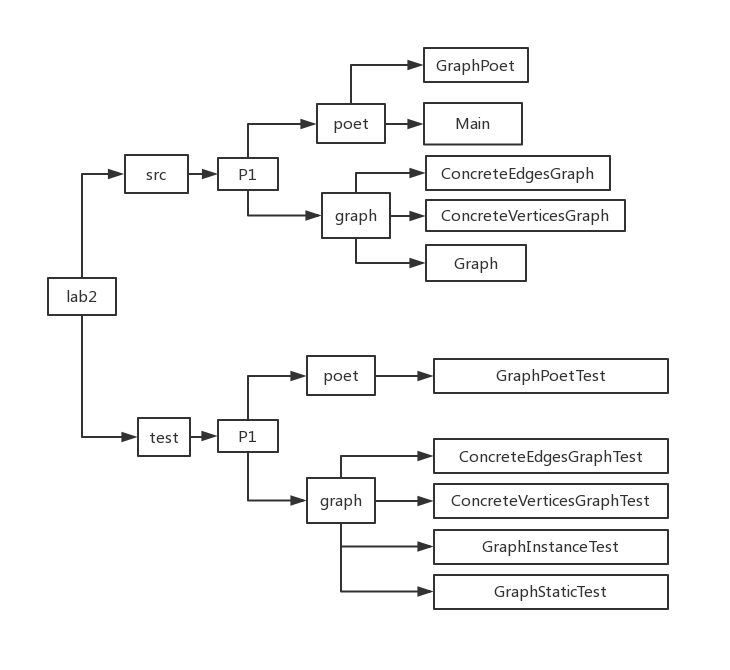
* git add \*

git commit -m ”second commit”

git push -u origin master

提交完成

* 项目树状图



## Re-implement the Social Network in Lab1

充分利用Lab1的思想，并充分运用本次实验P1的类进行设计实验；

其中getDistance算法利用了广度优先搜索队列的原理进行处理

### FriendshipGraph类

方法有

|  |  |
| --- | --- |
| addVertex | 调用Graph类中的add方法 |
| addEdge | 调用Graph类set方法，并将权重赋为1 |
| getDistance | 利用广度优先遍历的思想，调用Graph类中的targets方法得到一个人的所有有联系的人 |
| main | 主函数，并进行简单的算法正确性验证 |

重要说明：在getDistance方法中，声明了两个Map结构，如下

Map<Person, Boolean> visited = **new** HashMap<>();// 用于判断是否被访问

Map<Person, Integer> dis = **new** HashMap<>();// 用于记录距离

在getDistance方法中，声明了一个Quene结构，如下

Queue<Person> queue = **new** LinkedBlockingQueue<>();//用于广度优先遍历

### Person类

Person属性

**private** String name;

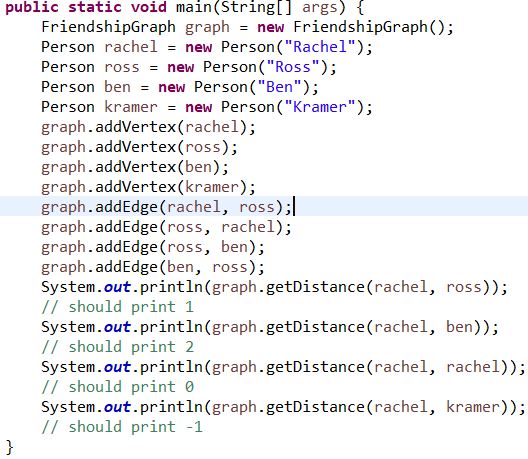
**private** **static** Set<String> *allName* = **new** HashSet<>(); // 用于储存所有的名字，判断是否违背名字唯一原则

Person方法

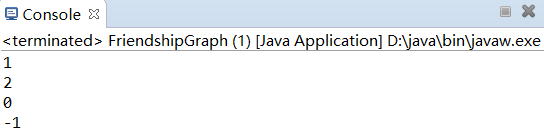
|  |  |
| --- | --- |
| Person | 初始化类，输入是Person的String名字 |
| getName | 得到名字 |

### 客户端main()

给出你的设计和实现思路/过程/结果。

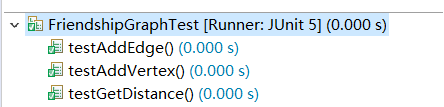


运行结果如下



### 测试用例

运行结果如下



设计思路

|  |  |
| --- | --- |
| testAddVertex | 判断加入是否成功+判断加入是否重复 |
| testAddEdge | 判断加入是否成功+加入是否多余 |
| testGetDistance | 判断为0，为-1，多个结果取最短 |

关于testGetDistace的一点说明

要验证假如有多个从A到B的通路的时候，取得路径的距离是最短的一条的。

* 实验验证设计关系图为

A->B->C

B->A->C

C->B->A

则A与C之间最小的距离应该是1

* 实验验证设计关系图为

A->B->C->D

B->A->C->D

C->B->A->D

D->A->B->C

则A与D之间最短的距离也是1

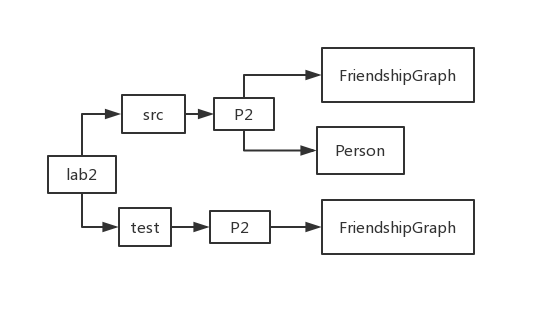
### 提交至Git仓库

* git add \*

git commit -m “third commit”

git push -u origin master

* 树状图（Process on）



## 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。

#### Action类

方法有checkMovePiece和checkEatPiece

checkMovePiece：

调用Position中的checkPosition函数检查坐标是否超出棋盘范围；调用Board中的getBoardOccupation函数检查position是否被占用以及被哪位选手占用。合法的条件是：原来的坐标被该选手占用且目标坐标位置无人占用且在棋盘范围内。

checkEatPiece

调用Position中的checkPosition函数检查坐标是否超出棋盘范围；调用Board中的getBoardOccupation函数检查position是否被占用以及被哪位选手占用。合法的条件是：原来的坐标被该选手占用且目标坐标位置对手占用。

#### Board类

属性

**private** Boolean[][] goBoard1 = **new** Boolean[19][19]; // 用于储存棋盘棋子存在情况：true表示已被占用

**private** Boolean[][] goBoard2 = **new** Boolean[19][19]; // 用于储存棋手占用情况：true表示player1

**private** Boolean[][] chessBoard1 = **new** Boolean[8][8]; // 用于储存棋盘棋子存在情况：true表示已被占用

**private** Boolean[][] chessBoard2 = **new** Boolean[8][8]; // 用于储存棋手占用情况：true表示player1

方法

|  |  |
| --- | --- |
| Board | 初始化这几个二维数组 |
| setGoBoard | 将某个棋子下在围棋棋盘上 |
| deleteChessBoardPiece | 将棋子从象棋棋盘的二维数组表示上删除 |
| addChessBoradPiece | 将棋子的坐标位置表示在二维棋盘数组上 |
| getOccupation | 根据坐标判断该位置是否被占用 |
| getOccupationPlayer | 根据坐标判断该位置被谁占用 |
| checkOccupation | 检查某位置的占用情况 |

#### Game类

属性

**private** Map<String, String> map // 用于输出棋盘时，将英文名字转化为中文名

**private** **static** Board *goBoard* = **new** Board(2); // 围棋棋盘

**private** **static** Board *chessBoard* = **new** Board(1); // 象棋棋盘

**private** Player player1; // 选手1

**private** Player player2; // 选手2

**private** **static** List<Piece> *allPieces1* = **new** ArrayList<>(); // 玩家一所有棋子

**private** **static** List<Piece> *allPieces2* = **new** ArrayList<>(); // 玩家二所有棋子

**private** String player1History = ""; // 玩家一的游戏历史

**private** String player2History = ""; // 玩家二的游戏历史

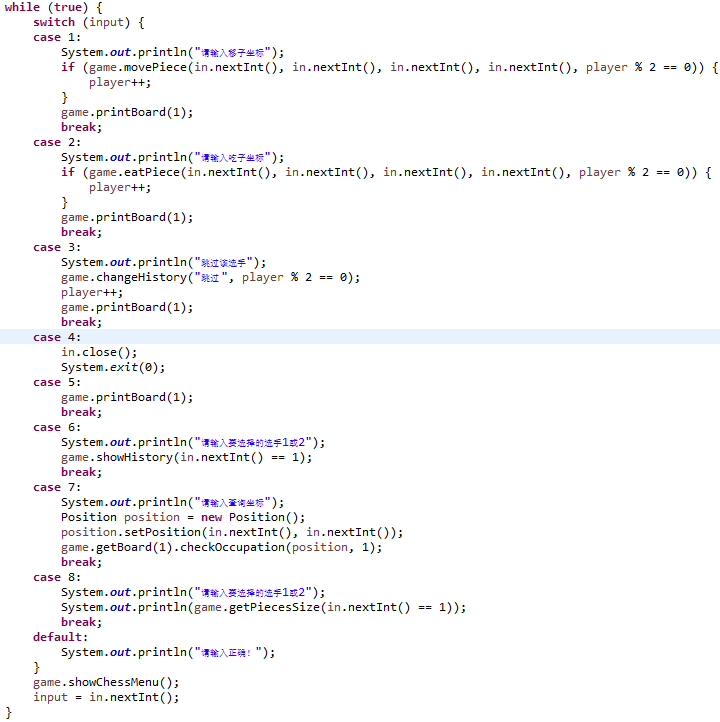
方法

|  |  |
| --- | --- |
| **public** Game(**int** gameTye) | 如果是象棋，则添加出初始的32个棋子 |
| showGoMenu | 输出围棋菜单选项 |
| showChessMenu | 输出象棋菜单选项 |
| placePiece | 在围棋棋盘上落子 |
| checkPlacePiece | 检查待落子坐标是否合法 |
| extractPiece | 在围棋棋盘上提子 |
| checkExtractPiece | 检查待提子位置坐标是否合法 |
| movePiece | 在象棋棋盘上移子 |
| eatPiece | 在象棋棋盘上吃子 |
| showHistory | 输出玩家游戏历史 |
| setPlayer1/setPlayer2 | 设置玩家一/玩家二 |
| getPlayer1/getPlayer2 | 得到玩家一/玩家二 |
| getBoard | 得到棋盘情况 |
| getPiecesSize | 得到棋子数目 |
| getPieceByPosition | 根据Position得到Piece |
| changeHistory | 改变玩家历史记录 |
| printBoard | 打印输出象棋或者围棋棋盘 |

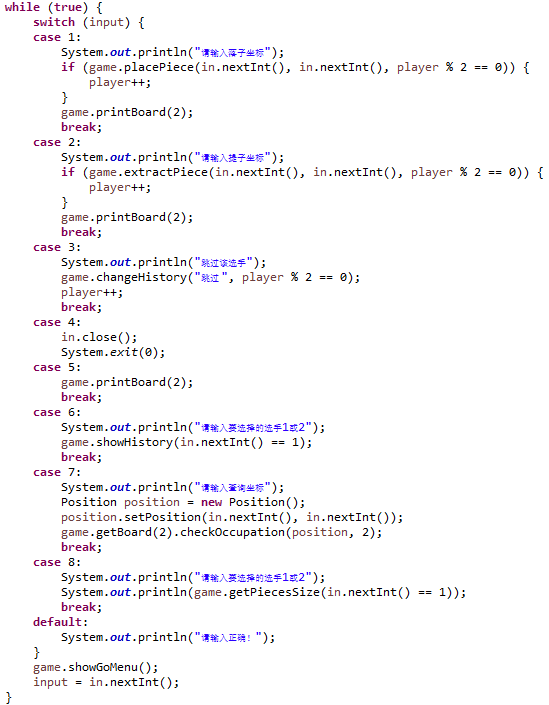
#### MyChessAndGoGame

main方法

整体运用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有所区别。switch象棋部分主体的的代码如下。



switch围棋主体部分如下



几点简化的方法

1. 代码使用int gameType变量，当其为1时，表示为象棋，否则表示为围棋；同时使用Boolean player变量，当其为true时，表示为玩家一，否则表示为玩家二。
2. 在功能的不断循环中，当部分功能实现时，player棋手默认控制权转变。但一些情况除外，比如输出棋盘，查看占用，查看棋子总数，查看游戏历史

#### Piece类

属性

**private** Position piecePosition = **new** Position();// 棋子的坐标

**private** String pieceName = **new** String();// 棋子的类别

**private** Boolean player;// 棋子的拥有者：true表示为棋手1

方法

|  |  |
| --- | --- |
| Piece | 初始化一颗棋子 |
| setPiecePosition | 设置棋子的坐标 |
| setPlayer | 设置棋子的拥有者 |
| getPiecePosition | 得到坐标 |
| getPlayer | 得到棋子的拥有者 |
| getPieceName | 得到棋子的名字 |
| setPieceName | 设置棋子的名字 |

#### Player类

属性

**private** String playName;// 玩家的姓名

方法

|  |  |
| --- | --- |
| Player | 设置玩家姓名 |
| getPlayerName | 返回玩家的姓名 |

#### Position类

属性

**private** **int** px = -1;//横坐标

**private** **int** py = -1;//纵坐标

方法

|  |  |
| --- | --- |
| setPosition | 设置坐标属性 |
| getPx | 得到横坐标 |
| getPy | 得到纵坐标 |
| positionEqual | 判断position和当前位置是否重合 |
| (override)toString | 重写toString |
| checkPosition | 检查坐标是否越界，不在棋盘范围内 |

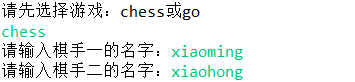
### 主程序MyChessAndGoGame设计/实现方案

辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

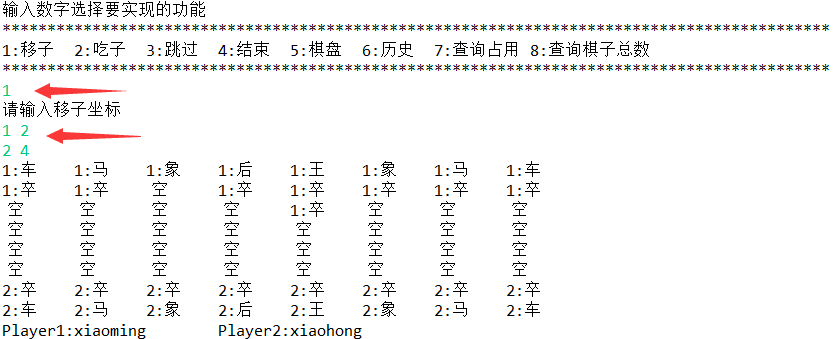
设计和实现

* 读取输入的游戏类型：chess或者go
* 读取输入的玩家名字：player1和player2
* 读取玩家输入的数字：象棋中1:移子2:吃子3:跳过4:结束5:棋盘6:历史7:查询占用8:查询棋子总数。围棋中1:落子2:提子3:跳过4:结束5:棋盘6:历史7:查询占用8:查询棋子总数。进行性相应的功能匹配与调用
* 实现玩家交替进行：设置变量int player，每次实现一些功能，就进行++操作，同时根据player是奇数还是偶数进行判断是玩家一还是玩家二。
* 输出棋盘：每次相应的功能实现后，都要输出相应的棋盘。
* 读取坐标：先读取的是横坐标，其次读取的是纵坐标；将这些坐标映射到功能要求的坐标上。

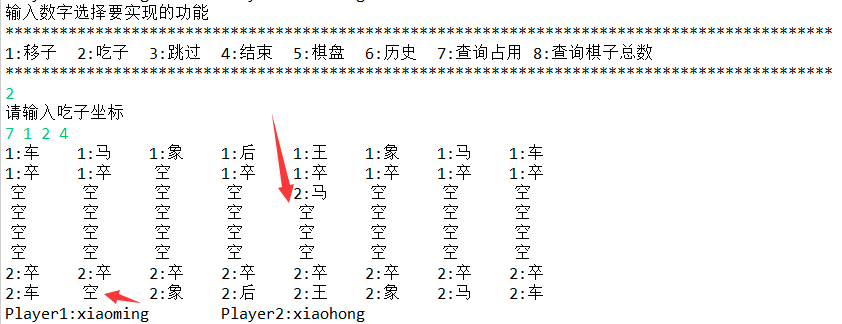
选择象棋



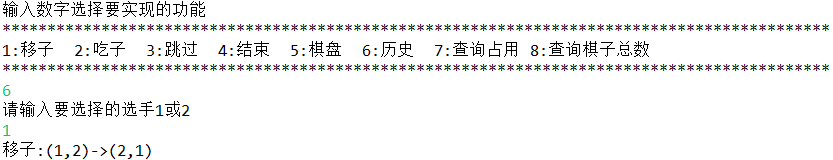
移子



吃子

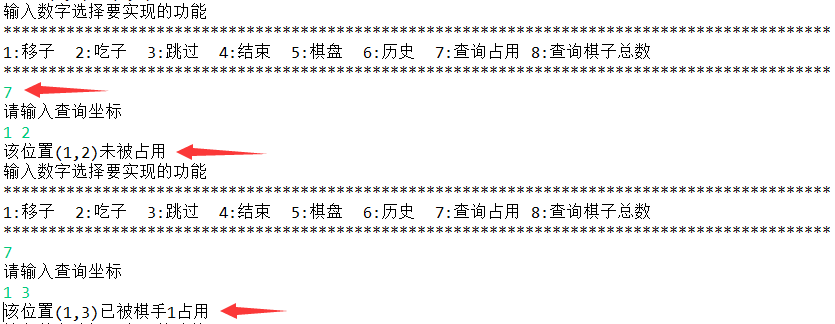


查询历史

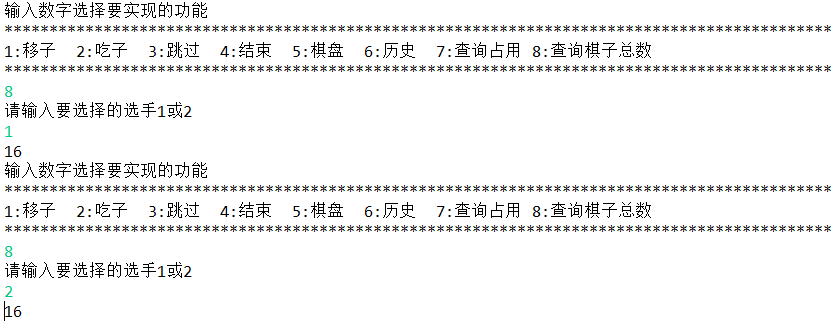




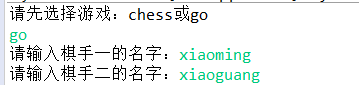
查询占用



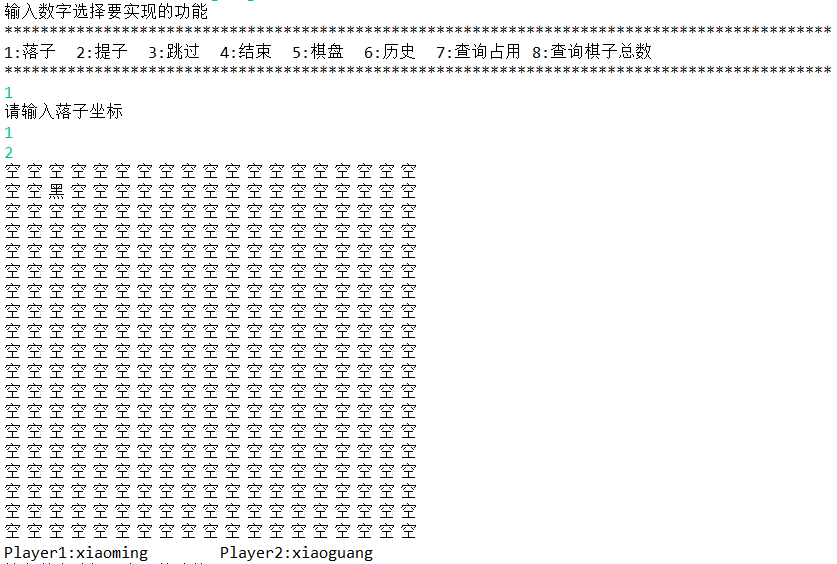
查询棋子总数



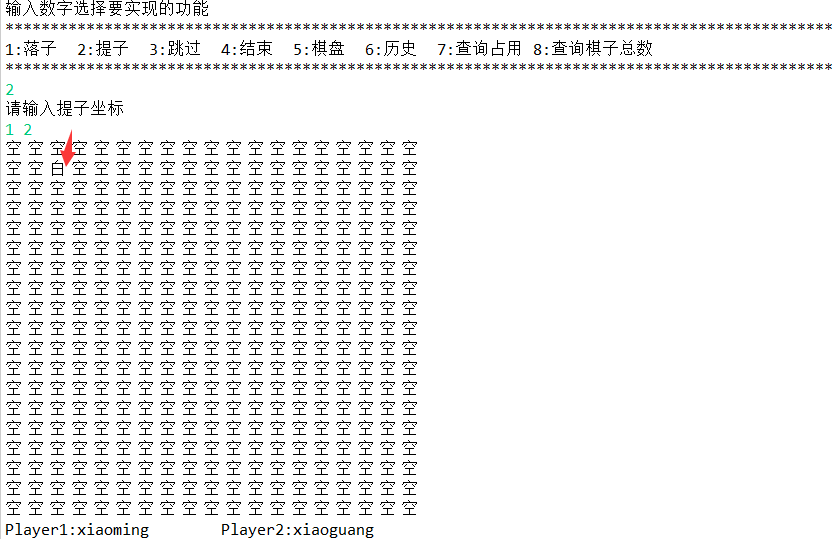
选择围棋



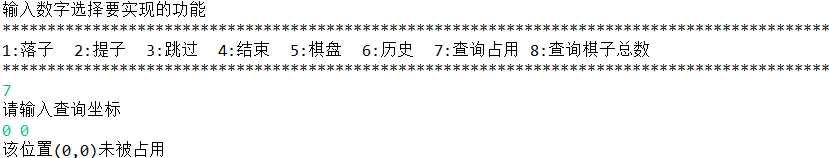
落子



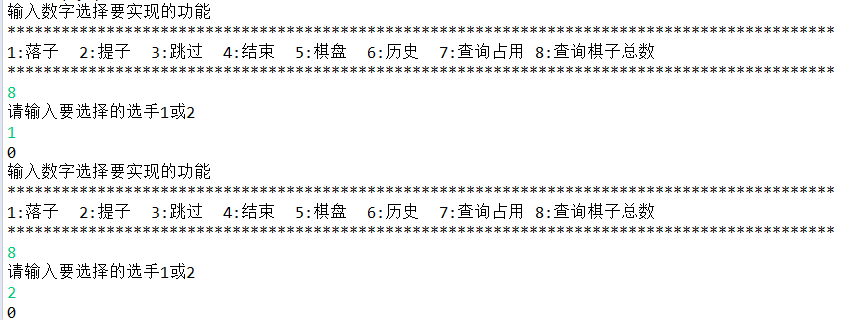
提子



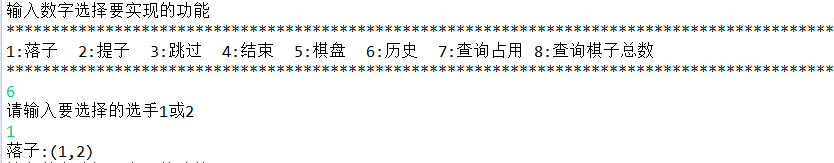
查询占用

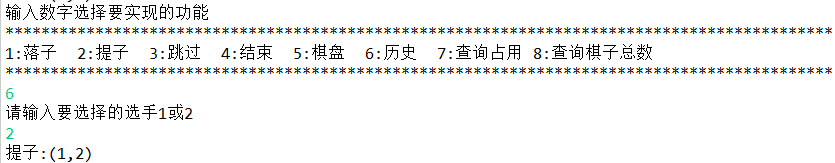


查询棋子总数



查询历史

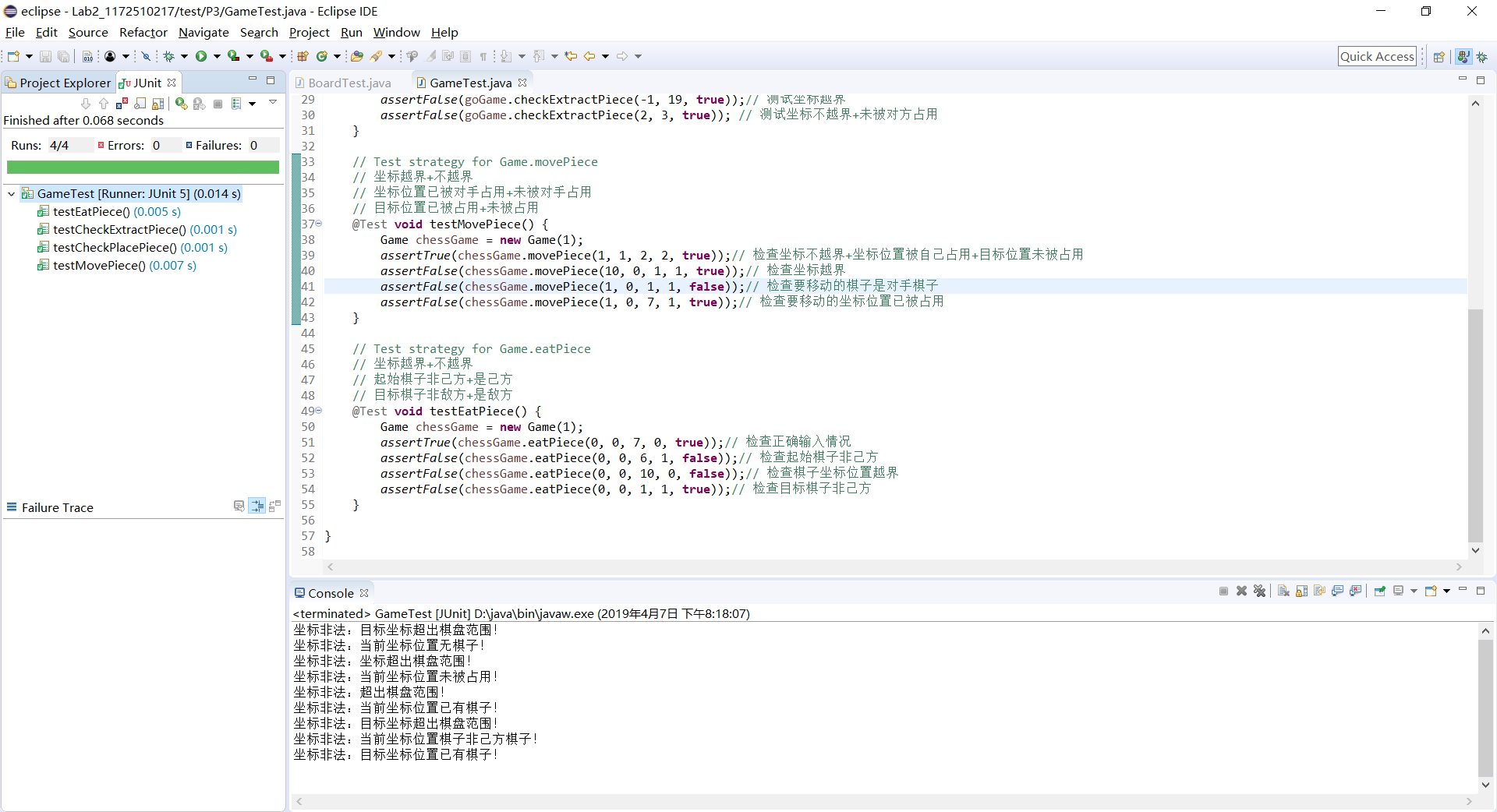




### ADT和主程序的测试方案

由于各个部分的调用关系，我们只需要测试部分类的部分方法即可

Game类测试截图如下。



// Test strategy for Game.checkPlacePiece

// 坐标越界+不越界

// 坐标位置已被占用+未被占用

// Test strategy for Game.checkExtractPiece

// 坐标越界+不越界

// 坐标位置已被对手占用+未被对手占用

// Test strategy for Game.movePiece

// 坐标越界+不越界

// 坐标位置已被对手占用+未被对手占用

// 目标位置已被占用+未被占用

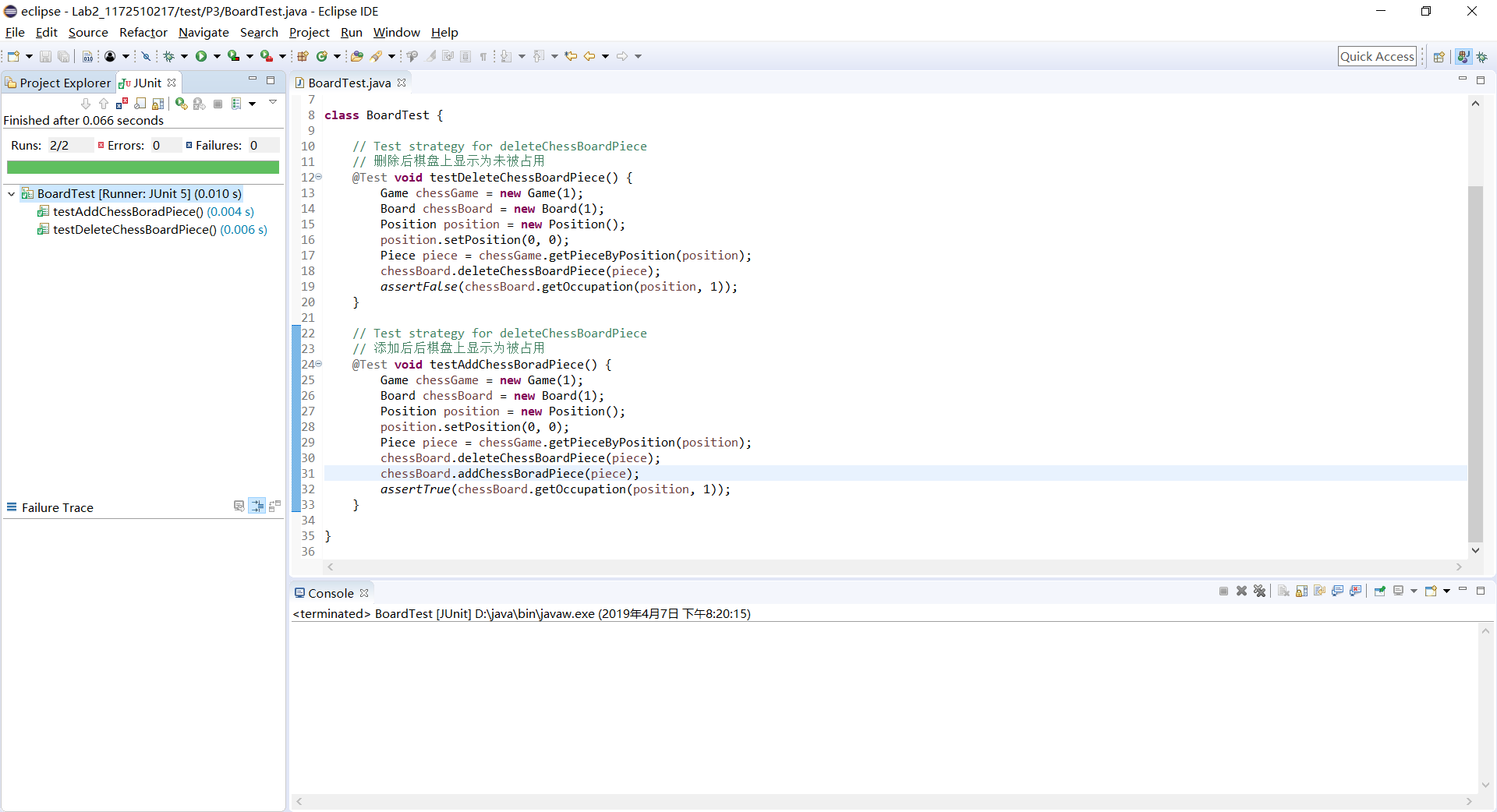
// Test strategy for Game.eatPiece

// 坐标越界+不越界

// 起始棋子非己方+是己方

// 目标棋子非敌方+是敌方

Board类测试截图如下



// Test strategy for deleteChessBoardPiece

// 删除后棋盘上显示为未被占用

// Test strategy for deleteChessBoardPiece

// 添加后后棋盘上显示为被占用

## Multi-Startup Set (MIT)

请自行设计目录结构。

注意：该任务为选做，不评判，不计分。

# 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 |
| 19-03-19 | 晚上 | 了解P1题意 | 完成一半 |
| 19-03-20 | 晚上 | 了解P1题意 | 完成 |
| 19-03-21 | 晚上 | 完成P1 | 完成三分之一 |
| 19-03-22 | 下午 | 完成P1 | 完成二分之一 |
| 19-03-23 | 下午晚上 | 完成P1 | 完成 |
| 19-03-24 | 晚上 | 完成P2 | 完成 |
| 19-03-26 | 晚上 | 理解P3 | 理解完成 |
| 19-03-28 | 晚上 | 完成一半类的构建 | 遇到问题，逻辑混乱 |
| 19-03-30 | 晚上 | 完成一半类的搭建 | 完成 |
| 19-04-01 | 晚上 | 完成P3 | 遇到困难，出现bug |
| 19-04-02 | 晚上 | 完成P3 | 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 |
| P1问题理解难度大 | 百度翻译，谷歌翻译，必应翻译，自己翻译 |
| P3各部分调用逻辑紊乱 | 仔细构思，事前详细构思 |

# 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. 英语要好好学呀，专业术语也要好好学习；
2. 从零开始搭建一个ADT，需要我们在书写代码前要详细构思，不能走一步看一步

## 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？

前者调用关系比较多，后者逻辑关系相对较强

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

代码上差异不大，但应用的广泛程度上差别很大

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？

减少了一些让我们有些不知所措的错误情况的出现

暂时还不能太适应

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？

应用范围更广

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？

已经适应

感受深刻，需要我们代码前比较全面的构思

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？
2. 关于本实验的工作量、难度、deadline。

工作量较大，难度适中，deadline合适

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

实验性强，需要我们不断地探索与实践