

**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 | 李大鑫 |
| 学号 | 1170300825 |
| 班号 | 1703008 |
| 电子邮件 | hahalidaxin@163.com |
| 手机号码 | 13075361573 |

**目录**

[1 实验目标概述 1](#_Toc1988214)

[2 实验环境配置 1](#_Toc1988215)

[3 实验过程 1](#_Toc1988216)

[3.1 Poetic Walks 1](#_Toc1988217)

[3.1.1 Get the code and prepare Git repository 1](#_Toc1988218)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc1988219)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc1988220)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc1988221)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc1988222)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc1988223)

[3.1.4.1 Make the implementations generic 2](#_Toc1988224)

[3.1.4.2 Implement Graph.empty() 2](#_Toc1988225)

[3.1.5 Problem 4: Poetic walks 2](#_Toc1988226)

[3.1.5.1 Test GraphPoet 2](#_Toc1988227)

[3.1.5.2 Implement GraphPoet 2](#_Toc1988228)

[3.1.5.3 Graph poetry slam 2](#_Toc1988229)

[3.1.6 Before you’re done 2](#_Toc1988230)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc1988231)

[3.2.1 FriendshipGraph类 2](#_Toc1988232)

[3.2.2 Person类 3](#_Toc1988233)

[3.2.3 客户端main() 3](#_Toc1988234)

[3.2.4 测试用例 3](#_Toc1988235)

[3.2.5 提交至Git仓库 3](#_Toc1988236)

[3.3 Playing Chess 3](#_Toc1988237)

[3.3.1 ADT设计/实现方案 3](#_Toc1988238)

[3.3.2 主程序ChessGame设计/实现方案 3](#_Toc1988239)

[3.3.3 ADT和主程序的测试方案 3](#_Toc1988240)

[3.4 Multi-Startup Set (MIT) 4](#_Toc1988241)

[4 实验进度记录 4](#_Toc1988242)

[5 实验过程中遇到的困难与解决途径 4](#_Toc1988243)

[6 实验过程中收获的经验、教训、感想 4](#_Toc1988244)

[6.1 实验过程中收获的经验和教训 4](#_Toc1988245)

[6.2 针对以下方面的感受 4](#_Toc1988246)

# 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

Java 环境：JDK11

IDE 软件：Eclipse

测试 工具：Junit5

VCS：git—repo: <https://github.com/ComputerScienceHIT/Lab2-1170300825>

硬件环境：Intel Core i7-6700HQ x64CPU,16G RAM,256G SSD +1T HDD.

# 实验过程

## 写在前面

**//实现过程中对于getter和setter使用新建对象的方法避免rep泄露，以下实现不进行解释。**

## Poetic Walks

分别新建两个类ConcreteEdgesGraph，ConcreteVerticesGraph 实现Graph接口。

Graph接口要求实现add（添加新节点），set（添加新边），remove（移除节点），vertices（获得所有的节点集合），sources（target）获得以target为目标节点的边的起始节点，targes(source)获得以source为起始节点的边的目标节点。

Poet：给定一组单词，对于两个相邻的单词a和b，认为存在一条由a到b的有向边，通过Graph接口构造有向图。再给定一条由单词组成的句子，如果句子中两个相邻单词之间在Graph图中有一个中间单词则将中间单词插入到两单词之间（如果有多个则随意插入一个）。

### Get the code and prepare Git repository

自 <https://github.com/rainywang/Spring2019_HITCS_SC_Lab2/tree/master/P1> 获得实验代码。

Git init

Git remote add origin [git@github.com:ComputerScienceHIT/Lab2-1170300825.git](mailto:git@github.com:ComputerScienceHIT/Lab2-1170300825.git)

在远程仓库新建Master分支

Git pull origin master

Git add .

Git commit -m “init”

Git push origin master

--git已经初始化

### Problem 1: Test Graph <String>

如果需要测试，默认ConcreteEdgesGraph implements Graph<String>，修改empty为：

Public Graph<String> empty() {

return new ConcreteEdgesGraph();

}

此时可以运行GraphStaticTest进行测试。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

1. 实现(generic) Edge<L>类

|  |  |
| --- | --- |
| 域 | 声明 |
| Private L source | 起始节点 |
| Private L target | 目标节点 |
| Private int weight | 边的权值 |

|  |  |
| --- | --- |
| 接口 | 说明 |
| Getter | 三个域的getter |
| judgeVertex | 便利函数，判断一个节点是否为这条边的端点之一 |
| @Override equas | 比较函数 |
| @Override hashCode | 生成hash值 |

1. 实现ConcreteEdgesGraph类

|  |  |
| --- | --- |
| 函数 | 实现思路 |
| add(L vertex) | 调用List.add，其返回结果为boolean满足spec定义。 |
| set(L source,L target,int weight) | 检查输入满足weight>=0。当weight>0时，如果当前没有该edge，则直接添加，返回值为0，如果已经有该edge，则记录旧值，修改为新值。如果weight==0，当删除的边不存在时返回0，当存在时，删除该节点，该边，如果此时两端点之中没有边与之相连了则删除节点。 |
| remove(L vertex) | 检查输入满足vertex存在vertices中，当存在时，删除该节点，同时删除所有与之相连的边，此时检查vertices中是否有点 没有连边，如果有则删除。 |
| Set<L> vertices | 返回vertices |
| Sources(L target) | 符合Map-filter-reduce的处理流程，Stream写法： |
| Targets(L source) | 符合Map-filter-reduce的处理流程，Stream写法： |

#### Implement ConcreteVerticesGraph

一、实现(generic) Vertex<L>类

|  |  |
| --- | --- |
| 域 | 声明 |
| Private L label | Label |
| Private Map<L,Integer> sources | 所有以label为目标节点的边，<起始节点lable,边的权重>. |
| Private Map<L,Integer> targets | 所有以label为其实节点的边，<目标节点label,边的权重> |

|  |  |
| --- | --- |
| 接口 | 说明 |
| Getter and Setter | 三个域的getter and setter |
| setSource(L source,int weight) | 检查输入满足source!=null，weight>=0。当weight==0时，调用this.removeSource，当weight>0时，调用Map.put修改source并且记录初始值。 |
| removeSource(L source) | 检查输入满足source!=null，调用sources.remove，并返回旧值（不存在则返回0）。 |
| setTarget(L target,int weight) | 检查输入满足target!=null,weight>=0，当weight=0时，调用this.removeTarget，当weight>0时，调用targets.put并返回旧值（不存在则返回0）。 |
| removeTarget(L target) | 检查满足target!=null。调用targets.remove()，返回旧值。 |
| remove(final L vertex) | 调用removeSource和removeTarget |

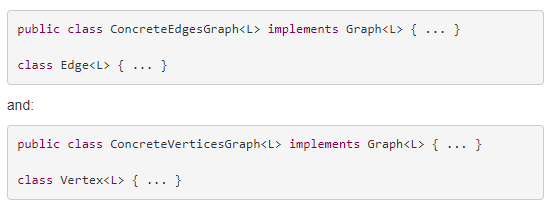
1. 实现ConcreteVertices类

|  |  |
| --- | --- |
| 函数 | 实现思路 |
| add(L vertex) | 检查输入满足vertex!=null。调用vertices.add，并返回结果 |
| set(L source,L target,int weight) | 检查输入满足weight>=0。如果source或者target不存在与vertices中则新建Vertex加入到vertices中，调用sourceVertex.setTarget，targetVertex.setSource。检查删除没有连边的节点。 |
| remove(L vertex) | 检查输入满足vertex!=null。如果输入节点不存在则返回false，否则，遍历vertices中的每一个节点调用v.remove在targes和sources中删除该节点，在vertices中删除该节点，检查删除没有连边的节点。 |
| Set<L> vertices | 符合Map-filter-reduce的处理流程，Stream写法： |
| Sources(L target) | 如果没有target则返回空集合，否则调用targetVertex.getSources(getter) |
| Targets(L source) | 如果没有source则返回空集合，否则调用targetVertex.getTargets(getter) |

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

#### Make the implementations generic

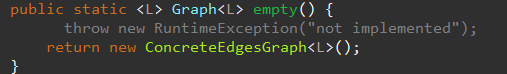
将类声明修改为：



将类的内部函数参数、返回值、域修改为L。

#### Implement Graph.empty()

修改Graph.empty为：

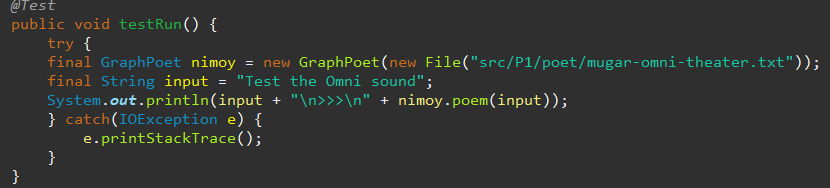


### Problem 4: Poetic walks

#### Test GraphPoet

配置文件为：new File(“src/P1/poet/mugar-omni-theater.txt”)

测试代码如下：



#### Implement GraphPoet

|  |  |
| --- | --- |
| 域 | 说明 |
| Private final Graph<String> affinityGraph | 配置文件生成的Graph |
| Private final List<String> corpusWord | 文集包含的单词 |

1. extractWordsFromFile(File corpus)

读取corpus配置文件，并从中提取中所有的单词。

使用split进行字符串分割

1. Graph<String> generateAffinityGraph

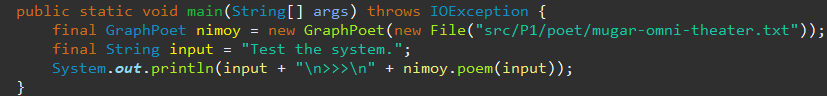
根据单词序列构造一个Graph，按照题目中对于图的定义，我们将前后两个相邻的单词之间添加一条从前向后的有向边。

1. public String poem(String input)

首先将input切割为单词序列inputWords，对于每两个相邻的节点v1和v2，如果v1.targets()与v2.sources()存在交集，则说明两个单词之间可以添加一个bridge，任意选择一个添加到两单词之间。

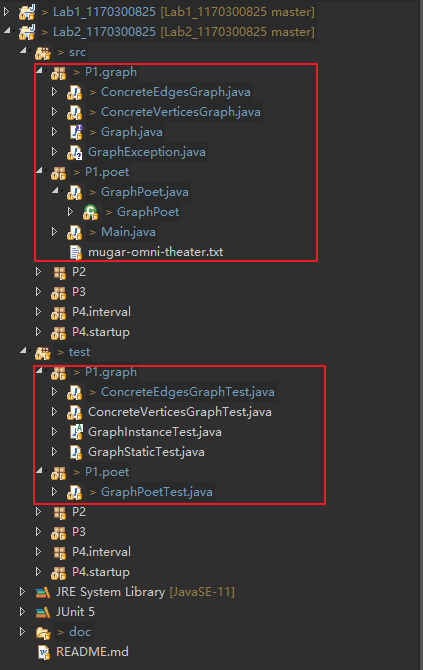
#### Graph poetry slam

Main函数实现如下：



### Before you’re done

项目的目录结构树状示意图：



## Re-implement the Social Network in Lab1

继承P1中ConcreteEdgesGraph<Person>或者ConcreteVerticesGraph<Person>类 实现FriendshipGraph，通过基本操作实现FriendshipGraph中addVertex，addEdge和getDistance三个接口，要求不能修改父类rep。

### FriendshipGraph类



|  |  |
| --- | --- |
| 函数 | 实现思路 |
| addVertex(Person newPerson) | 调用父类接口this.add(newPerson) |
| addEdge(Person pa,Person pb) | 调用父类接口this.set(pa,pb,1) |
| getDistance(Person stPerson,Person edPerson) | 使用BFS算法求得图中stPerson与edPerson之间的最短距离，BFS中需要遍历邻居节点，这时候只需要调用父类接口this.targets(topPerson)就可以获得topPerson的所有邻居节点。 |

### Person类

|  |  |
| --- | --- |
| 域 | 作用 |
| Private String nameString | 记录人名 |
| Private static Set<String> nameSet | 记录所有出现过的人名，在构造方法中如果出现重名情况则抛出SamNameException (extends Exception) |

### 客户端main()

测试代码如下：



### 测试用例

测试普通操作：包括addVertex，addEdge，包括重名情况

测试getDistance：测试不能达到目的地、有多条路径达到目的地的情况。

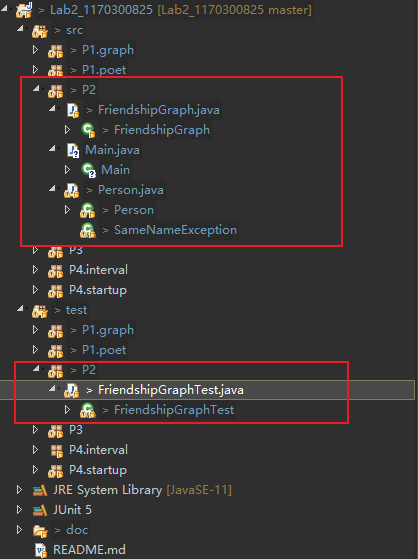
### 提交至Git仓库

Git add .

Git commit -m “P2 is DONE”

Git push origin master

项目的目录结构树状示意图如下：



## 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。

1. MyExp类

自定义异常类，继承自Exception，提供的接口有：

|  |  |
| --- | --- |
| 接口 | 作用 |
| Public String getExpMsg | 获得expMsg |
| Public void setExpMsg | 设置expMsg |
| @Override public String toString | 根据expMsg构造错误信息 |
| Public static void assertTrue(Boolean cond,String msg) throws MyExp | 检查cond是否正确，如果为false则抛出MyExp |

之所以自定义异常类，是因为使用Assert进行断言判断不合法情况的时候，AssertError只能在Junit下捕获到，当程序运行时则不能捕获，所以需要新建异常，使用MyExp来传递错误信息并进行回显。

1. Position’类

|  |  |
| --- | --- |
| 接口 | 作用 |
| Public int x() | 获得x |
| Public int y() | 设置y |
| Override public boolean equals(Oject that) | 判断两个Postion是否相等 |

坐标类，承载坐标信息x,y

1. Piece类

|  |  |
| --- | --- |
| 域 | 解释 |
| pieceState | 棋子状态，0为未放置，1为已经放置，2为放置之后被拿出棋盘且不可用 |
| pName | 棋子种类名称 |
| Px,py | 棋子在棋盘中所处的坐标 |

|  |  |
| --- | --- |
| 接口 | 作用 |
| Getter for (pName, px, py,pieceState) | Getter |
| Setter for (pName,pxy) | Setter |
| Public void rmFromBoard | 将该棋子从棋盘中移出，意味着不能再放入 |

棋子类，包含棋子基本信息，提供pieceState记录棋子状态。

1. Player类

|  |  |
| --- | --- |
| 域 | 解释 |
| Private String playerName | 玩家名称 |
| Private StringBuilder gameHistorySB | 玩家操作历史 |
| Private Set<Piece> playerPieces | 玩家所拥有的所有棋子 |

|  |  |
| --- | --- |
| 接口 | 作用 |
| Getter for (playerName,playerPieces,GameHistorySB) | Getter |
| Public Boolean addPiece(Piece) | 玩家添加棋子，如果玩家已有该棋子，则返回false。否则向playerPieces添加后返回true |
| Public Piece getAnyPieceByFilter(Predicate<? Super Piece> predicate) | 获取该玩家所有满足功能函数predicate要求的棋子之中的任意一个。如果没有，则返回null |
| Public Boolean isContainPiece(Piece) | 判断该玩家是否包含指定棋子 |
| Public void addHistory(String gameStep) | 向gameHistorySB中添加一步的操作 |

玩家类，注意一个玩家所拥有的棋子中包含许多相同类型名称的棋子，使用对象的引用来区别不同的对象，以下相同。

1. Board类

|  |  |
| --- | --- |
| 域 | 解释 |
| Private int boardType | 棋盘类型，0为放在格子里，1为放在交点上 |
| Private int boardSize | 棋盘大小，指的是棋盘上行或列所有的格子数目 |
| Private Piece boardPieces[][] | 存放棋盘上对应位置所放的棋子 |

|  |  |
| --- | --- |
| 接口 | 作用 |
| Public Piece getPieceAtCord(px,py) throws MyExp | 获取处于(px,py)位置的棋子，如果位置不合法抛出MyExp |
| Public void setPieceAtCord(int px,int py,Piece piece) | 将棋子piece放置在棋盘的(px,py)位置处。如果位置不合法则抛出MyExp |
| Public isCordAvailable(int cx,int cy) | 判断坐标(cx,cy)是否是一个合法坐标。棋盘最终的横纵可以放置的棋子数目为boardSize+boardType |
| Public isPieceInBoard(Piece piece) | 判断棋子piece是否处于棋盘之内。 |
| Public int getNumOfPlayerPiecesInBoard(Player player) | 获得player在棋盘中的棋子数目（也可以将接口交给Player实现） |

棋盘类。提供操作棋盘上棋子的接口。

这里除了检查坐标是否合法之外不进行操作的正确性检查，将检查放到Action进行合理性判断。

1. Action类

|  |  |
| --- | --- |
| 域 | 解释 |
| Private Board gameBoard | 游戏中棋盘对象的引用 |
| Private Player playerA,playerB | 游戏中玩家对象AB的引用 |

//以下接口作用省略检查边界的说明

//省略getter and setter的作用说明

|  |  |
| --- | --- |
| 接口 | 作用 |
| Public void putPiece(Player player,Piece piece,Position pos) | 将玩家player的未处于棋盘的piece棋子落到pos处。修改棋子piece位置，修改棋盘gameBoard上pos位置的棋子为piece，添加玩家历史。 |
| Public void movePiece(Player player,Position st,Position ed) | 将玩家player的已经处于棋盘上的位于st的棋子移动到空地址ed。将棋盘上st位置设置为null，ed位置设置为piece，修改piece的位置为ed。添加玩家历史 |
| Public void removePiece(Player player,Position pos) | 将用户player的位于棋盘上pos的棋子移出棋盘。调用piece.rmFromBoard，将棋盘上pos位置设置为null。添加玩家历史。 |
| Public void eatPiece(Player player,Position st.Position ed) | 使用用户player的位于棋盘st位置的棋子吃掉到对手的ed位置的棋子。调用edPiece.rmFromBoard，将棋盘ed位置设置为stPiece，将棋盘st位置设置为null，将stPiece坐标设置为ed。添加用户历史。 |

下棋动作类。

异常捕捉：对于每种下棋动作，使用MyExp.assertTrue()进行异常测试，使用try catch捕捉异常，并抛出异常。

下棋动作的操作对象为player，gameBoard，和piece。其中playerA、playerB和gameBoard对象是Game传入的引用，因为每种操作都是建立在这两类对象的基础上的。

1. Game类

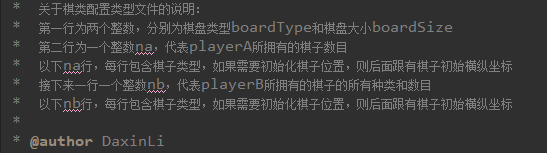
|  |  |
| --- | --- |
| 域 | 解释 |
| Private String gameType | 游戏类型 |
| Private Board gameBoard | 游戏棋盘 |
| Private Action gameAction | 游戏动作 |
| Private PlayerA,PlayerB | 游戏玩家，playerA为先手 |

//以下接口作用省略检查边界的说明

//省略getter and setter的作用说明

|  |  |
| --- | --- |
| 接口 | 作用 |
| Public void iniGameWithPlayerName(String paName,String pbName) | 通过传入的两个玩家的名字初始化Game中的各类对象。初始化：从gameType\_config.txt文件中读取游戏配置，初始化player，添加拥有的棋子，初始化棋盘，设置大小和类型。将gameBoard,playerA,playerB的引用传入Action新建gameAction对象。 |
| Public void pubPiece throws MyExp | 调用gameAction |
| Public void movePiece throws MyExp | 调用gameAction |
| Public void removePiece throws MyExp | 调用gameAction |
| Public void eatPiece throws MyExp | 调用gameAction |
| Public Player getOwnerAtCord(Positon pos) throws MyExp | 获得处于pos位置的棋子的所有者，获得pos位置的棋子piece，然后调用Player.isContainPiece判断属于playerA还是playerB。 |
| Public Piece getPieceAtCord(Position pos) throws MyExp | 获取处于pos位置的棋子piece，如果没有棋子则返回null。 |
| Public int getNumOfPlayerPiecesInBoard(Player player) | 获取用户在棋盘上的所有棋子数目，调用gameBoard. getNumOfPlayerPiecesInBoard |

游戏配置文件规范：



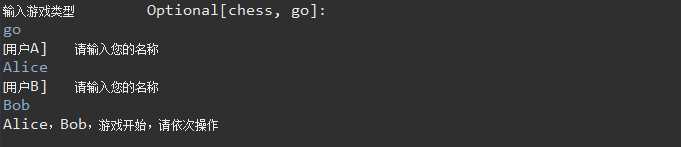
游戏类。

主要功能为：读取文件配置，初始化Player，Board，Piece对象，为调用类提供下棋动作的接口。实现读取文件配置，将其他操作下放到具体对象。

### 主程序MyChessAndGoGame设计/实现方案

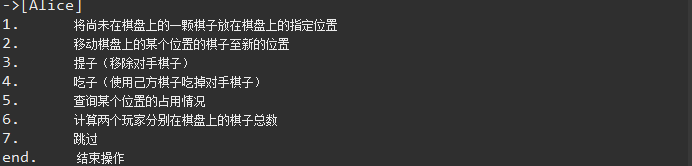
辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

1. 输入流程说明：
2. 获取游戏类型以及两个玩家的名称，如图



此时获取到gameType与playerAName，playerBName，创建game对象，并调用iniGameWithPlayerName读取文件配置初始化游戏。

1. 游戏操作菜单：



第一行为当前操作的玩家，其中1-4操作为下棋动作，操作成功后改变玩家，5-6为查询操作，查询后不改变当前玩家。

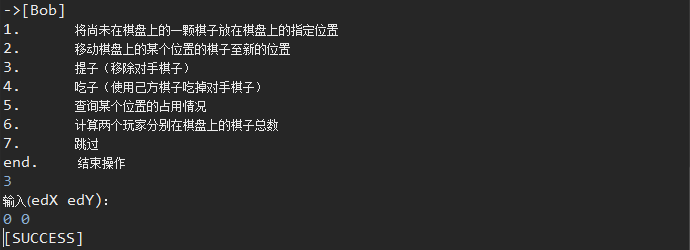
1. 输入(pName edX edY)，调用player.getAnyPieceByFilter获得满足pName类型的未处于棋盘上的棋子piece，处于调用game.putPiece将piece放到ed位置。



2—输入(stX stY edX edY)，调用game.movePiece将player的st位置的棋子移动到ed位置处。



3—输入(edX edY)，调用game.removePiece将ed位置的棋子移出棋盘。



4—输入(stX stY edX edY)，调用game.eatPiece，使用player st位置的棋子吃掉ed位置的敌方棋子

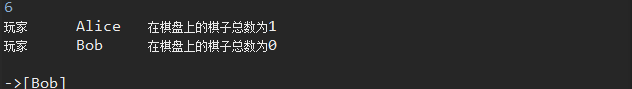


5—输入(edX edY) 调用game.getPieceAtCord game.getOwnerAtCord 获得ed位置的棋子和所有者player，打印棋盘占用情况（null则未被占用）。不改变玩家。





6—调用game. getNumOfPlayerPiecesInBoard，不改变玩家



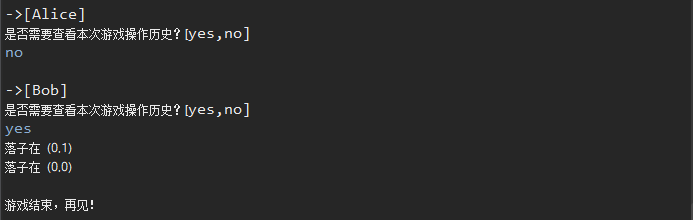
7—不操作，切换玩家。



end—结束操作。之后询问是否查看操作历史



1. 查看操作历史



调用并打印player. getGameHistoryString。

1. 输入异常处理

使用pNI代表当前操作的用户编号（0,1），针对输入数据不合法的基本思路是使用while(true)死循环套 输入模块，如果输入不合法则不改变当前用户编号，继续死循环，直到输入正确，此时pNI=(pNI+1)%2同时跳出死循环。

本程序处理的输入异常包括：输入个数不正确，输入数据不满足文档定义（抛出MyExp），输入数据类型错误。异常的处理方法为使用try catch捕获异常MyExp，NumberFormatException并打印相应错误信息。

### ADT和主程序的测试方案

介绍针对各ADT的各方法的测试方案和testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

测试主要针对重要ADT，测试方案如下：

1. testGame（包含测试Action）

测试主要包括测试Game 的配置文件载入，Game提供的各类操作接口，如putPiece，eatPiece等。

对于操作接口的测试，设计测试用例满足题目文档中所给出的各类边界条件。

1. testBoard

测试主要包括测试Board的越界情况处理以及getNumOfPlayerPiecesInBoard的函数测试。

## Multi-Startup Set (MIT)

代码已实现，报告 略。

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3-8 | 19:00-23:26 | 完成P1 | 完成 |
| 3-9 | 14:00-16:17 | 完成P2 | 完成 |
| 3-10 | 18:00-21:12 | 完成P4 | 完成 |
| 3-11 – 3-12 | 18:00-00:16 | 完成P3 | 没有完成 |
| 3-12 | 18:00-20:54 | 完成P3 | 完成 |
| 3-12 | 21:00-23:40 | 写文档 | 没有完成 |
| 3013 | 14:00-16:33 | 写文档 | 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 |
| 如何处理输入异常 | 自定义继承自RuntimeException的异常类，在继承类中提供static assertTrue检查cond条件真值，如果为假则抛出该异常。 |

# 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

## 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？

答：面向ADT的编程我们需要考虑更多边界情况，需要将具体的问题抽象为更加宽泛具有覆盖性的函数，同时对于ADT我们应该编写覆盖性较高的测试用例，而且ADT显然可以应用与更加广阔的场景，我们可以通过集成或者直接实例化将ADT应用与特定的场景之下。直接面向应用场景编程则不需要考虑那么多，直接为了解决特定的问题进行编程即可。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

答：考虑泛型编程需要考虑该泛型是否为可变类型，需要与Client做好约束。使用泛型编程有许多来自Object的接口我们不能使用，使用泛型编程需要考虑更多的实现场景。

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？

答：直接针对ADT的规约进行黑盒测试，这时候我们显然可以避免长期变成之后的麻木从而注意到更多更细节的编程用例。能。

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？

答：节省编程量。

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？

答：已经适应。首先需要自己从题目文档中抽取需求，然后针对题目需求设计接口，从而通过调用我们的接口就可以实现相应的需求。

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？

答：避免造成引用泄漏，形成潜在隐患。愿意。

1. 关于本实验的工作量、难度、deadline。

答：适中。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

答：希望以后可以学到更多的有用知识。