

**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 | 李震宇 |
| 学号 | 1170300110 |
| 班号 | 1703001 |
| 电子邮件 | [1137859144@qq.com](mailto:1137859144@qq.com) |
| 手机号码 | 18800419432 |

**目录**

[1 实验目标概述 1](#_Toc5197164)

[2 实验环境配置 1](#_Toc5197165)

[3 实验过程 1](#_Toc5197166)

[3.1 Poetic Walks 1](#_Toc5197167)

[3.1.1 Get the code and prepare Git repository 2](#_Toc5197168)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc5197169)

[3.1.3 Problem 2: Implement Graph <String> 2](#_Toc5197170)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc5197171)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc5197172)

[3.1.4 Problem 3: Implement generic Graph<L> 7](#_Toc5197173)

[3.1.4.1 Make the implementations generic 7](#_Toc5197174)

[3.1.4.2 Implement Graph.empty() 8](#_Toc5197175)

[3.1.5 Problem 4: Poetic walks 8](#_Toc5197176)

[3.1.5.1 Test GraphPoet 8](#_Toc5197177)

[3.1.5.2 Implement GraphPoet 8](#_Toc5197178)

[3.1.5.3 Graph poetry slam 9](#_Toc5197179)

[3.1.6 Before you’re done 9](#_Toc5197180)

[3.2 Re-implement the Social Network in Lab1 10](#_Toc5197181)

[3.2.1 FriendshipGraph类 10](#_Toc5197182)

[3.2.2 Person类 11](#_Toc5197183)

[3.2.3 客户端main() 11](#_Toc5197184)

[3.2.4 测试用例 11](#_Toc5197185)

[3.2.5 提交至Git仓库 11](#_Toc5197186)

[3.3 Playing Chess 11](#_Toc5197187)

[3.3.1 ADT设计/实现方案 11](#_Toc5197188)

[3.3.1.1 Position 11](#_Toc5197189)

[3.3.1.2 Piece 12](#_Toc5197190)

[3.3.1.3 Player 12](#_Toc5197191)

[3.3.1.4 Board 12](#_Toc5197192)

[3.3.1.5 Action 12](#_Toc5197193)

[3.3.1.6 Game 13](#_Toc5197194)

[3.3.1.7 MyChessAndGoGame 13](#_Toc5197195)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 13](#_Toc5197196)

[3.3.2.1 执行过程截图： 13](#_Toc5197197)

[3.3.2.2 主程序的设计和实现方案： 20](#_Toc5197198)

[3.3.3 ADT和主程序的测试方案 23](#_Toc5197199)

[3.3.3.1 ActionTest 23](#_Toc5197200)

[3.3.3.2 BoardTest 24](#_Toc5197201)

[3.3.3.3 PieceTest 24](#_Toc5197202)

[3.3.3.4 PlayerTest 25](#_Toc5197203)

[3.3.3.5 PositionTest 25](#_Toc5197204)

[3.3.3.6 主程序 25](#_Toc5197205)

[3.4 Multi-Startup Set (MIT) 26](#_Toc5197206)

[4 实验进度记录 26](#_Toc5197207)

[5 实验过程中遇到的困难与解决途径 27](#_Toc5197208)

[6 实验过程中收获的经验、教训、感想 27](#_Toc5197209)

[6.1 实验过程中收获的经验和教训 27](#_Toc5197210)

[6.2 针对以下方面的感受 27](#_Toc5197211)

# 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

在上个实验的基础上安装：eclEmma

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

# 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

实现一个图的ADT 。

作为一个图，它里面有两个元素：边、点，这个实验要从边、点两个不同的角度完成这个ADT的设计。

最后使用自己设计的这个ADT，完成Poetic walks的要求。

### Get the code and prepare Git repository

从Github上下载代码，import到eclipse中完成实验。

### Problem 1: Test Graph <String>

这部分要自己设计一些测试方法来对ADT进行测试。

主要就是对Graph中那些方法依次进行test，用assertequal方法即可。

分别测试add/set/remove/vertices/sources/target几个方法。

将运行结果与期望结果进行比较，相同则通过测试。

注意测试要尽可能的覆盖所有ADT方法的分支，考虑这种特殊情况。

### Problem 2: Implement Graph <String>

以两种不同的角度(边、点)设计ADT。

#### Implement ConcreteEdgesGraph

从边的角度完成对图ADT的设计。

##### 内部类Edge

###### 属性：

private final L source;

private final L target;

private final int weight;

分别为来源、目标、权重。

###### 构造方法：

public Edge(L source, L target, int weight)

使用this方法赋值即可。

private void checkRep()

检查不变量。要求source和target不能相同，weight不能为负。

使用assert断言即可。

###### get方法：

分别返回this.source、this.target、this.weight。

###### 重写：public boolean equals(Object e)

重写equals方法，认为source相同、target相同的边为相同的边。

###### 重写：public String toString()

重写toString方法，效果图：



##### ConcreteEdgesGraph类：

###### 属性：

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

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

分别为顶点集、边集。

顶点集存有所有的图中顶点的名称。

边集存有所有边的source、target、weight。

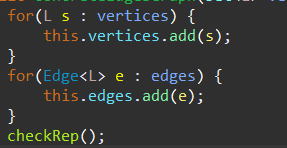
###### 构造方法：

public ConcreteEdgesGraph()

创建一个空的图。

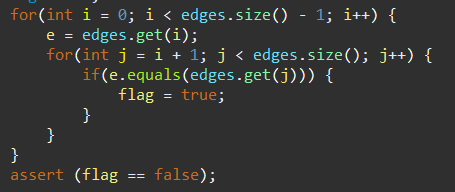
public ConcreteEdgesGraph(Set<L> vertices, List<Edge<L>> edges)

创建一个拥有顶点集vertices、边集edges的ADT，这里为了防止表示暴露，顶点集和边集都要复制一份。



###### private void checkRep()

检查不变量，不能有重复的边。使用assert断言。



以下方法皆重写：

###### public boolean add(L vertex)

添加顶点vertex。如果不存在重复，添加，返回true，如果重复则返回false。

###### public int set(L source, L target, int weight)

set方法。

首先判断weight大小，如果weight = 0则遍历边集找到这条 source -> target的边，记录其weight，将其删除，然后返回它的weight值。

如果weight>0，遍历边集，如果有这条source -> target的边，记录其原始weight，然后更改其weight，(由于edge是imuutable类型，所以这里使用remove再add的操作)并返回原始weight值。如果没有这条边，则new一个边加入，然后返回0。

###### public boolean remove(L vertex)

remove方法。

首先遍历顶点集，删除L顶点集。

然后便利edge集，如果找到了某条边，其source或target为L，就将其删除。

如果不存在这个顶点，则返回false，反之返回true。

###### public Set<L> vertices()

返回顶点集。为了保证immutable，返回一个副本。

###### public Map<L, Integer> sources(L target)

返回某个target的所有source、weight构成的map。

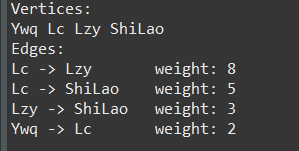
遍历边集，找到所有target为L的边，将其source weight作为键值对加入map中返回。

###### public Map<L, Integer> targets(L source)

与上同理。

###### public String toString()

效果：



#### Implement ConcreteVerticesGraph

从顶点的角度完成ADT

##### 内部类Vertex：(mutable)

###### 属性：

public Map<L, Integer> vertices = new HashMap<L, Integer>();

public L target;

一个map，记录source和weight，加上当前vertex顶点名便可以确定所有指向该顶点的边了。

###### **构造器**：

public Vertex(L name, L target, int weight)

//一个有一个指向它的名为name的节点，该边权重为weight。

public Vertex(L name)

//一个没有指向它的顶点的顶点。

public Vertex()

//空。

因为实际设计时，不需要直接传入一个map，所以没写。

###### private void checkRep()

检查不变量，要求所有权重大于零，且source的名字与该vertex的target名不相同。

使用assert断言。

###### 重写：public boolean equals(Object e)

重写equals，认为vertex属性中target名相同既为相同顶点。

###### 重写：public String toString()

重写toString，效果：



##### ConcreteVerticesGraph类：

###### 属性：

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

顶点集合

###### 构造方法：

public ConcreteVerticesGraph(List<Vertex<L>> vertices)

创建含有这些顶点信息的图。

public ConcreteVerticesGraph()

创建空图。

###### private void checkRep()

检查不变量，所有weight要大于零，所有边source和target不能相同。

以下方法全部是重写的

###### public boolean add(L vertex)

添加某个顶点，如果有重复返回false，如果不重复添加成功，返回true。

###### public int set(L source, L target, int weight)

设定边，首先判断weight，若大于零遍历寻找到这个边，然后暂存weight，设置weight，返回原weight。若等于零，则删掉边。这个与上边的一样，不赘述。

###### public boolean remove(L vertex)

删除某个顶点，即要删除顶点，也要删除所有与该顶点有关的边。

###### public Set<L> vertices()

返回所有顶点，注意要新建一个保证immutable。

###### public Map<L, Integer> sources(L target)

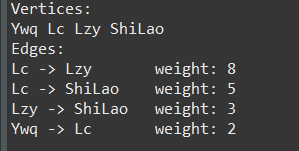
返回所有target的sources、weight的map。返回那个target的vertices即可，也是要新建一个副本返回，保证immutable。

###### public Map<L, Integer> targets(L source)

遍历寻找，同理不赘述。

###### public String toString()

效果：



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

将所有的String改成L。

#### Make the implementations generic

尝试使用Integer类型。

public void test() {

ConcreteEdgesGraph<Integer> g = new ConcreteEdgesGraph<Integer>();

g.add(new Integer(1));

Set<Integer> vertices = new HashSet<Integer>();

Set<Integer> verticesReturn = new HashSet<Integer>();

vertices.add(new Integer(1));

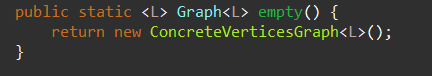
verticesReturn = g.vertices();

*assertEquals*("expected equal", vertices, verticesReturn);

}

#### Implement Graph.empty()

**使用顶点的那种实现方法。**



### Problem 4: Poetic walks

#### Test GraphPoet

**测试程序。**

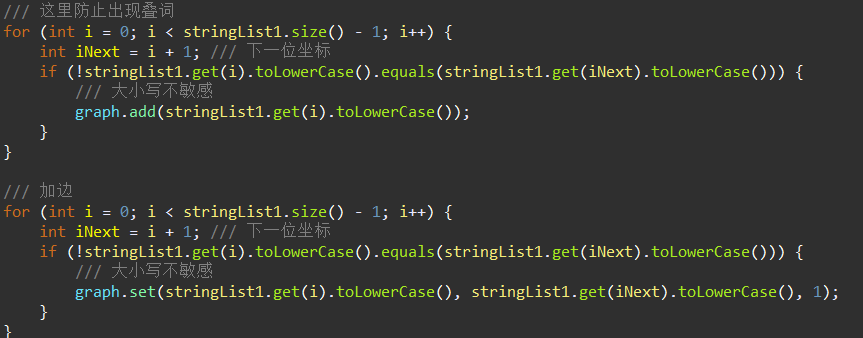
**就是创建一些样例，有特殊的有不特殊的，分别调用程序进行处理，判断结果是否相同，这样就可以发现程序的小问题了。**

#### Implement GraphPoet

首先是文件读入字符串，要处理输入的问题。然后需要处理标点符号问题。

然后按照每个单词，构建一个图。

构建过程：



#### Graph poetry slam

首先也是处理输入的字符串，分隔开，然后处理单词结尾处的标点符号。

使用队列的数据结构。进行一个深度为2的广度优先搜索，对不同的搜索结果进行不同的操作

关键操作：



函数意义：







### Before you’re done

目录结构：

Lab2

src P1 graph ConcreteEdegesGraphTest.java

ConcreteVerticesGraphTest.java

GraphInstanceTest.java

GraphStaticTest.java

poet mugar-omni-theater.text

test.txt

GraphPoetTest.java

test P1 graph ConcreteEdgesGraph.java

ConcreteVerticesGraph.java

Graph.java

poet GraphPoet.java

Main.java

mugar-omni-theater.txt

## Re-implement the Social Network in Lab1

写一个图的ADT，其中包括图的顶点和边。

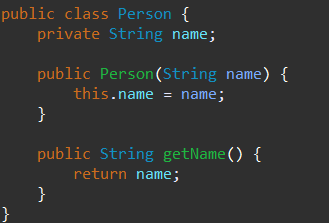
### FriendshipGraph类

将Lab1中的这个问题改一改，思路大体都一样。用刚刚写好的ADT实现对图的存储即可，getDistance的时候使用广度优先搜索就OK。

广度优先搜索：



### Person类



很简单的一个immutable类。

### 客户端main()

main函数就是Lab1中的main。

如果改成与用户交互的形式也很简单，不再赘述。

### 测试用例

和Lab1中的test一样，对一些各种情况进行讨论，预测和结果相同即可。

### 提交至Git仓库

文件目录示意图：

Lab2 src P2 FriendshipGraph.java

Person.java

test P2 FriendshipGraphTest.java

## Playing Chess

### ADT设计/实现方案

#### Position

mutable类型的类。

AF:表示二维平面上的坐标

RI:x、y大于等于0

Safety from rep exposure:这是一个mutable类型的类，可以进行修改。

#### Piece

immutable类型的类。

AF:存储棋子信息，包括名字、所处位置。

RI:名称字符串合法。所处位置合法。

Safety from rep exposure:返回List时返回新建的副本。使用private final属性。

#### Player

immutable类型的类

AF:存储棋手信息，包括名字、拥有的棋子、操作历史。

RI:名字、棋子信息合法。拥

Safety from rep exposure:返回List时返回新建的副本。使用private final属性。

#### Board

immutable类型的类

AF:表示棋盘，含有棋盘size，棋盘上存在的棋子，使用二维数组存储。

RI:棋盘size大于0，棋盘上存在的棋子需要合法。

Safety from rep exposure:返回List时返回新建的副本。使用private final属性。

#### Action

immutable类型的类

AF:表示棋手的操作，此处要将对player类对象的改动和board类对象的改动结合到一起。一起修改。

RI:player、board需要合法。操作输入坐标需要合法。

Safety from rep exposure:返回List时返回新建的副本。使用private final属性。

#### Game

immutable类型的类

AF:一场游戏中的所有元素。

RI:这些所有元素都需要合法。

Safety from rep exposure:返回List时返回新建的副本。使用private final属性。

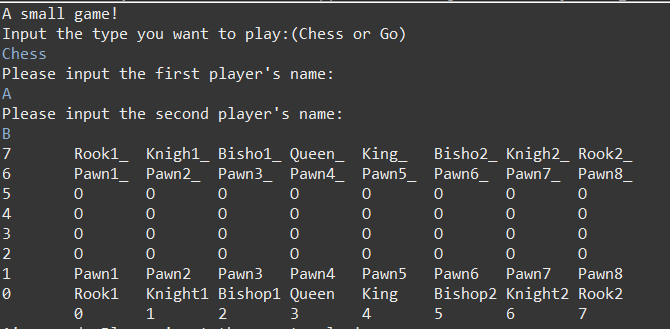
#### MyChessAndGoGame

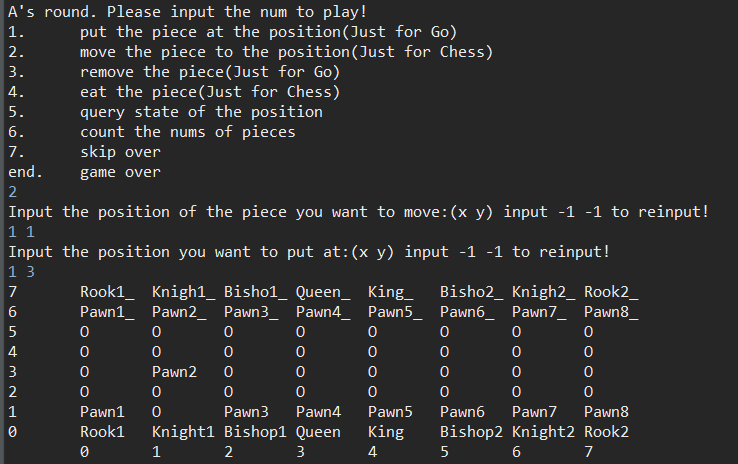
// 将在下面详细叙述。

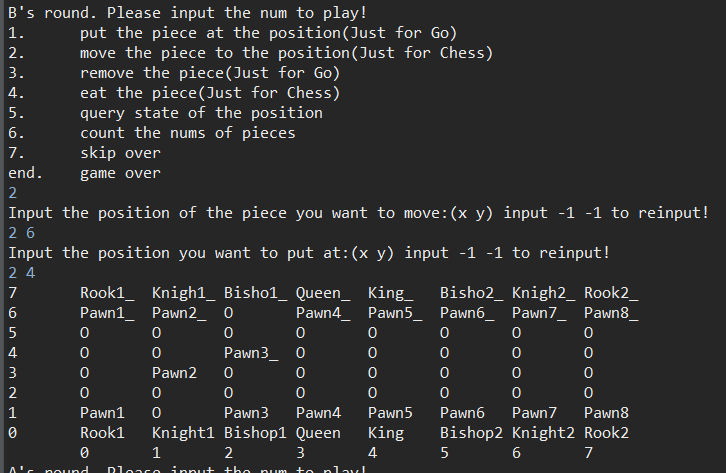
### 主程序MyChessAndGoGame设计/实现方案

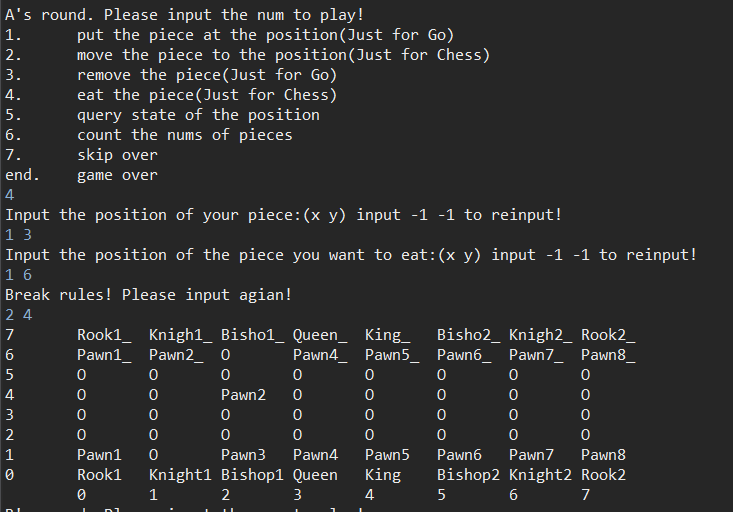
#### 执行过程截图：

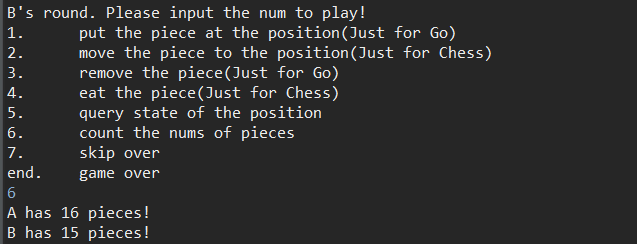
国际象棋：

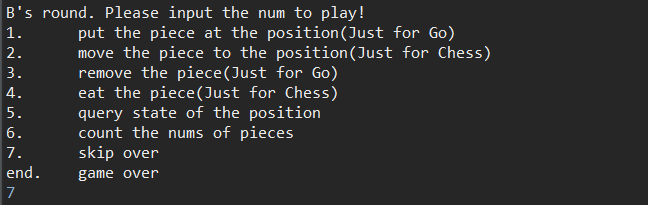


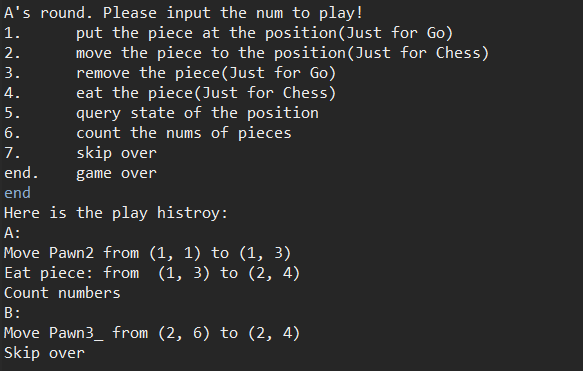




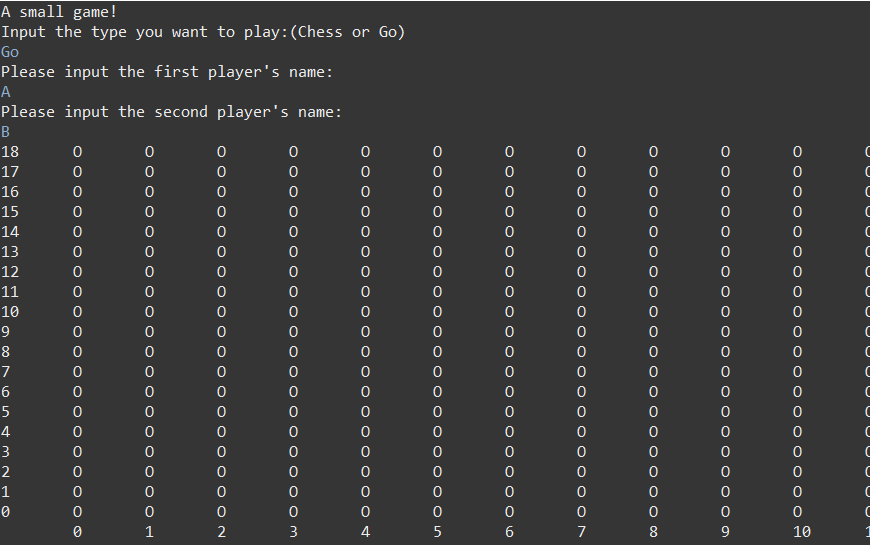


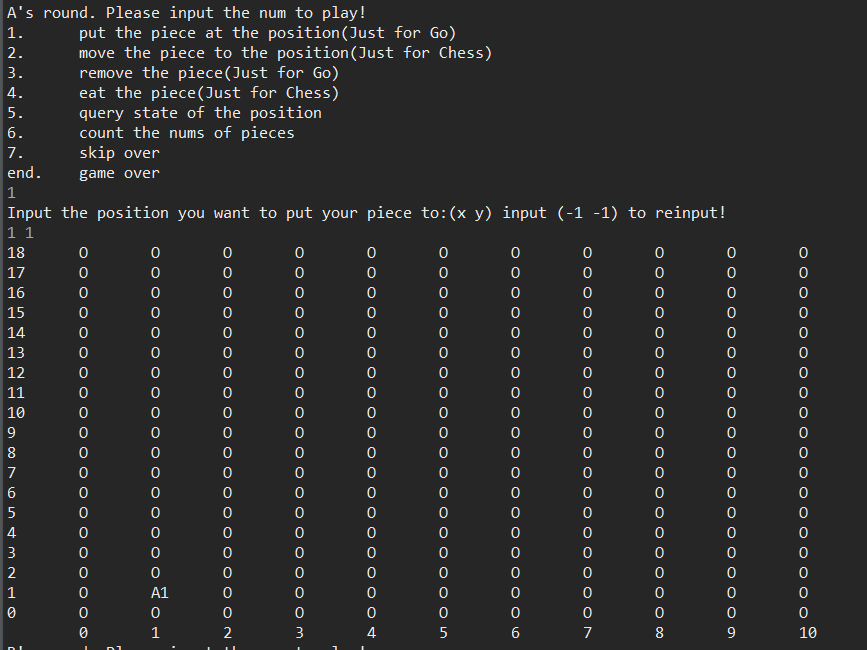


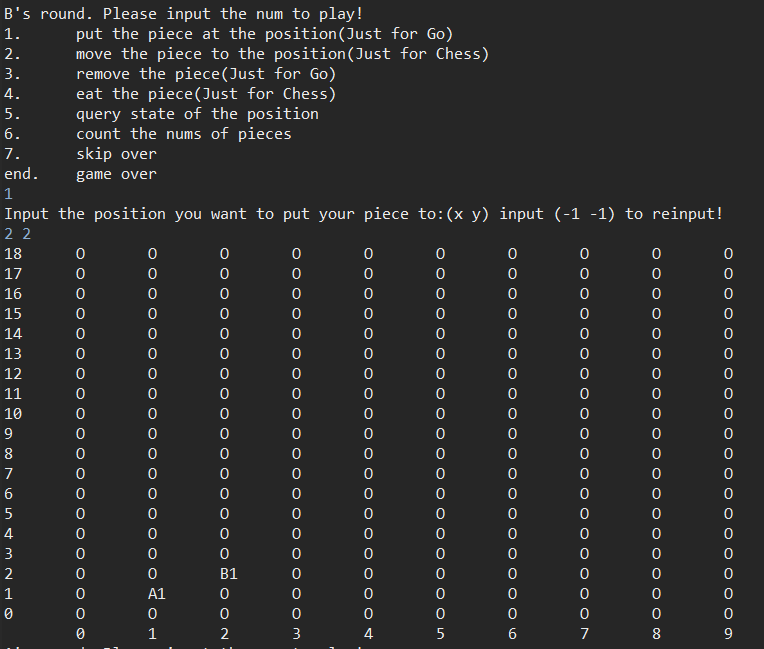


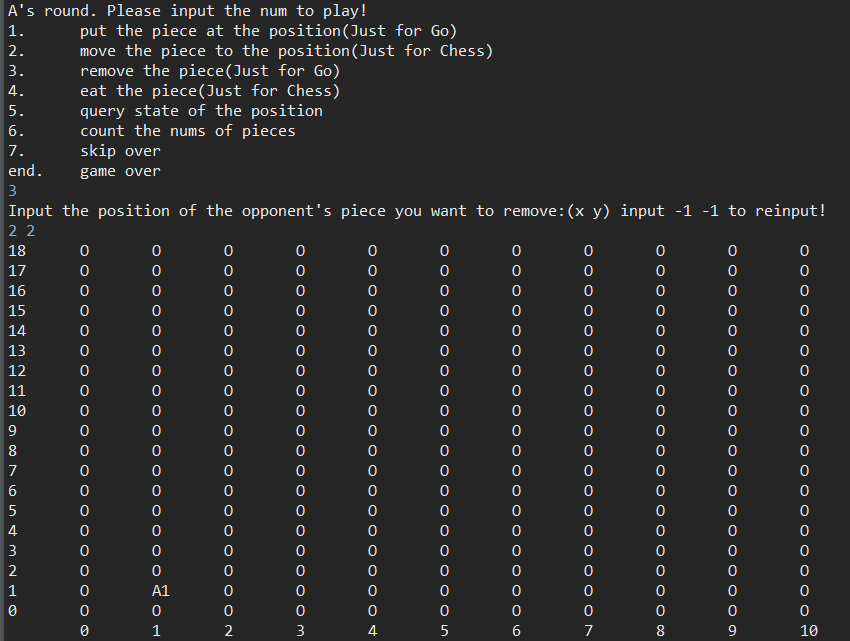


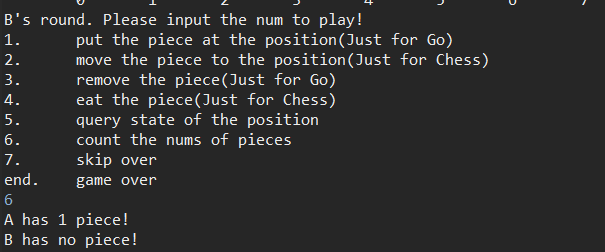
围棋：

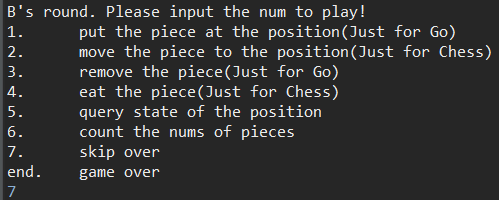


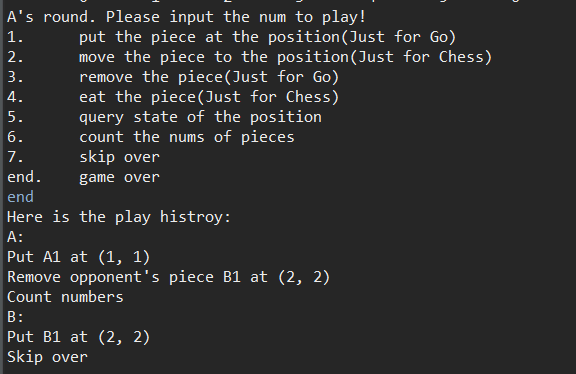






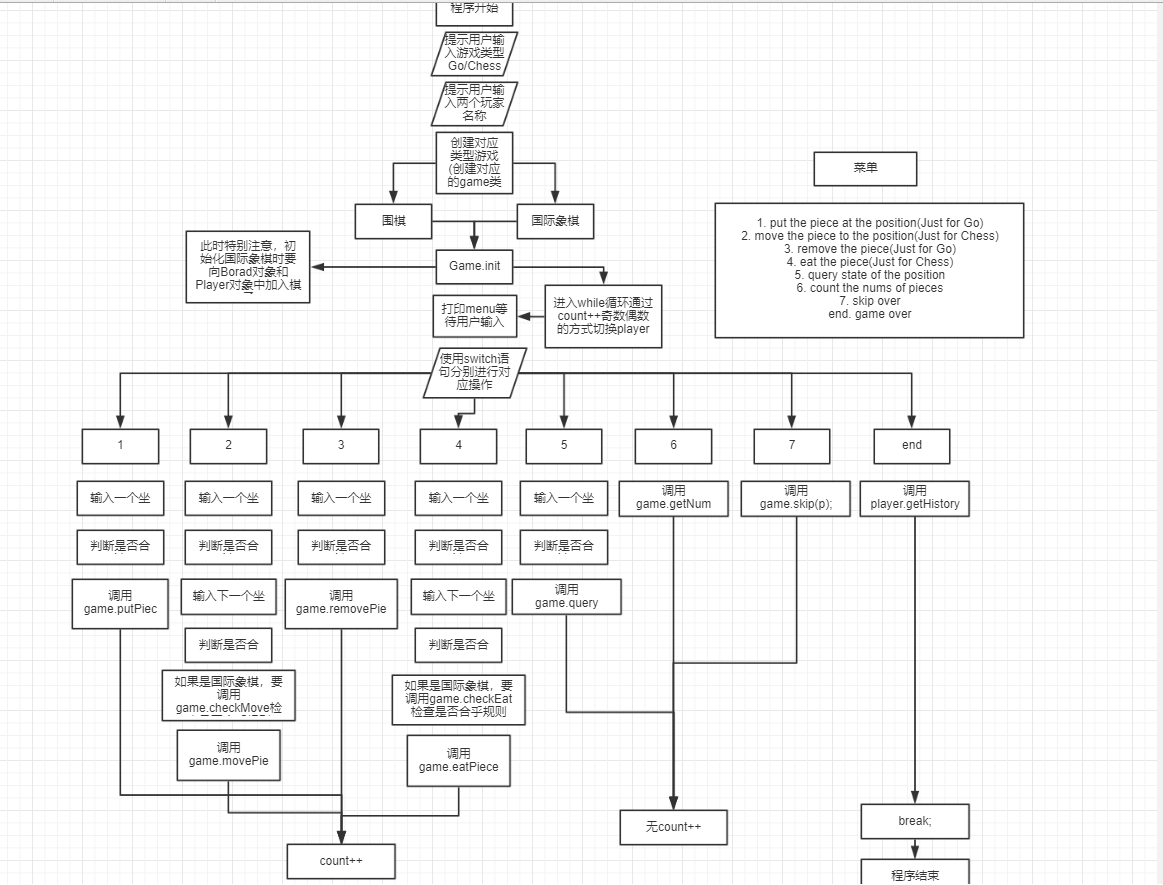






#### 主程序的设计和实现方案：

##### 大体轮廓：



##### 细节：

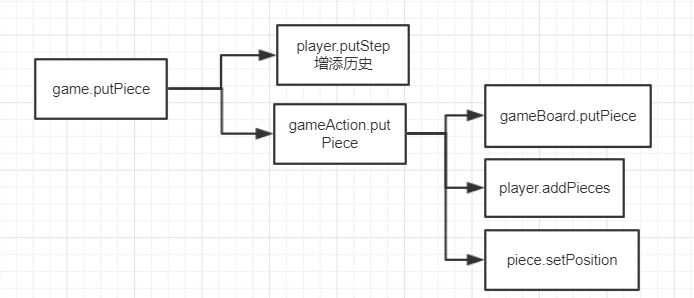
使用一个list存储player，每次在while循环中，切换player时使用count++%2作为索引，来选择调用list中的不同player轮流进行操作。

在各种game.xxx(操作)函数中，我们通过向函数传入player，然后对其进行处理的方法，为其增添history，方便以后在结束时直接调用player.getHistory。

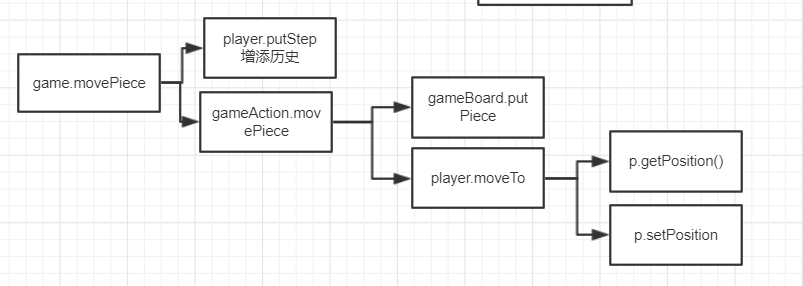
然后game.xxx增添完历史之后直接调用game中的action对象的方法，对game中的board类型的对象和player类型的对象进行修改，修改时就涉及到piece的修改和position的修改。当然，需要调用对应的put/set/move函数对其进行操作。

四个关键操作的流程图：

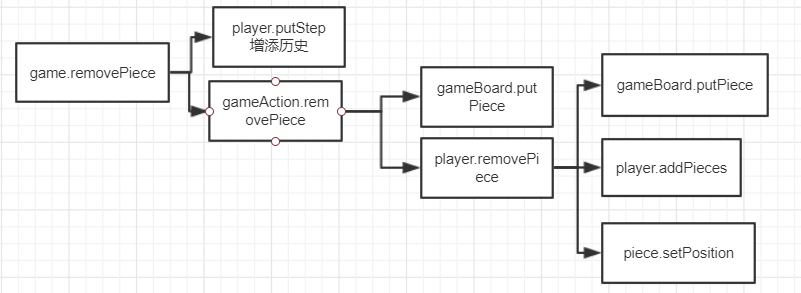
game.putPiece



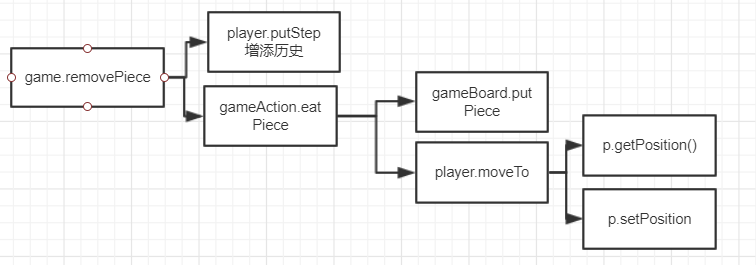
game.movePiece



game.removePiece



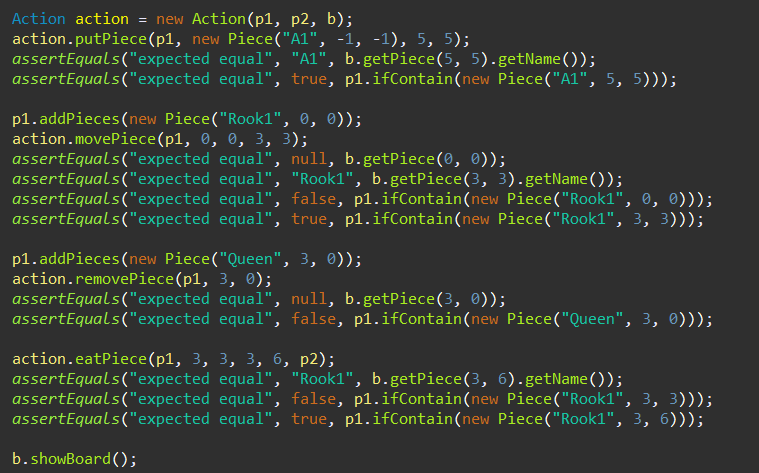
game.eatPiece



### ADT和主程序的测试方案

对每一个类设计测试样例，将期望的和运行结果进行对比即可。最后Game类的操作之比Action类多了一个putStep，不需要进行额外的测试了(就是调整以下字符串格式就ok了)。

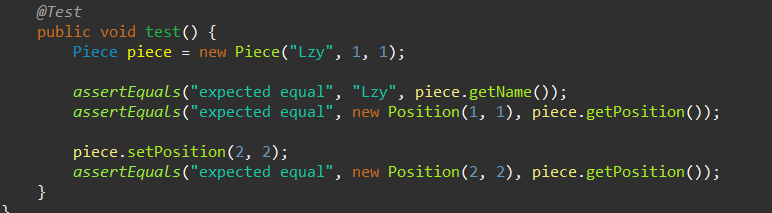
#### ActionTest



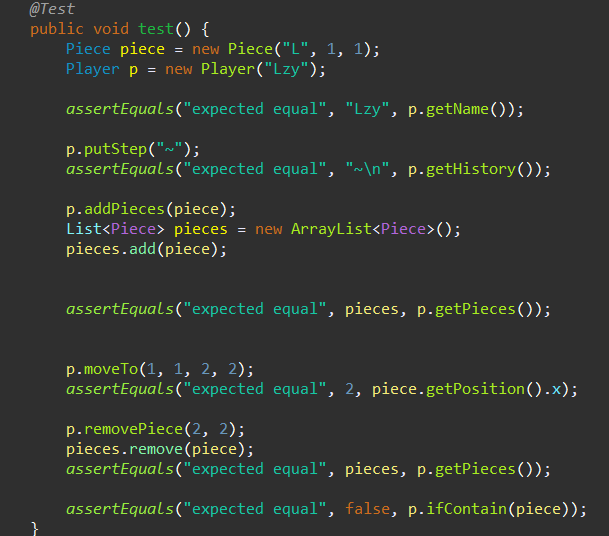
#### BoardTest



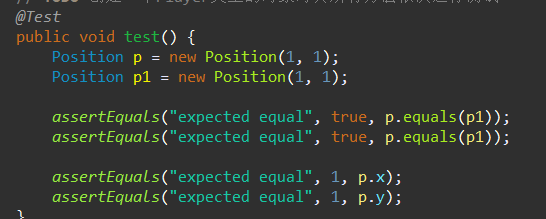
#### PieceTest



#### PlayerTest

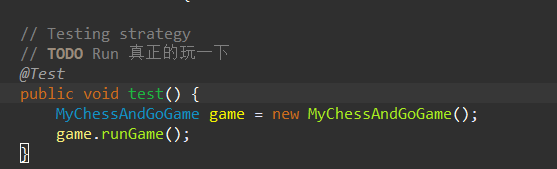


#### PositionTest



#### 主程序

主程序测试方案，就按照真实的游戏过程进行测试即可。(run即可)



## 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 |
| 3.13 | 晚间 | 阅读P1要求 | 完成 |
| 3.14 | 晚间 | 完成P1中的Con..VerticeGraph | 差点 |
| 3.15 | 晚间 | 完成P1中的Con..VerticeGraph | 完成 |
| 3.17 | 晚间 | 完成P1中的Con..EdgeGraph | 完成 |
| 3.18 | 晚间 | 完成P1中的问题三 | 完成 |
| 3.19 | 晚间 | 完成P1中的写诗 | 未完成 |
| 3.20 | 晚间 | 完成P1中的写诗 | 完成 |
| 3.22 | 晚间 | 完成P2 | 完成 |
| 3.22 | 晚间 | 阅读P3要求 | 完成 |
| 3.24 | 晚间 | 写Position、Piece、Player、Board | 完成 |
| 3.25 | 晚间 | 写Action、Game | 完成 |
| 3.26 | 晚间 | 写My…. | 未完成 |
| 3.27 | 晚间 | 写My…. | 完成 |
| 3.28 | 晚间 | 修改前面的类 | 未完成 |
| 3.29 | 晚间 | 修改前面的类 | 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 |
| 自己写ADT时封闭性不好。 | 尽量不暴露出去。使用老师教过的方法，实在不行就…暴露了。 |
| 自己写ADT时，一开始的构想不够完善，总是需要在写后面的时候再补充前面。 | 这是第一次写ADT，总有想的不到位的地方，应该仔细思索列一个表格之后再开始写代码。 |

# 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

在设计一个大工程的时候，一定一定一定要仔细构想，甚至把思路写在纸上再开始实现，不然的话会疯狂修改自己已经写完的东西。

## 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？

面向ADT的编程需要在写代码之前构想出大框架，然后慢慢实现。面向应

用编程感觉会简单一些，直接写就好。

使用泛型感觉操作更高难一些，但实际上还好。

更能注意到一些编写程序的细节方面的东西，不太适应。

增加代码的复用程度，不用再多谢了。

不太适应，感觉很难写。

更清楚的进行编程，不太愿意(但是好像不得不这么做。

感觉还好

emm，挺好的，就是有点秃。