

**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 | 刘璟烁 |
| 学号 | 1180500301 |
| 班号 | 1803001 |
| 电子邮件 | [corleonechina@outlook.com](mailto:corleonechina@outlook.com) |
| 手机号码 | 13102886813 |

**目录**

1. 实验目标概述 1
2. 实验环境配置 1
3. 实验过程 2
   1. Poetic Walks 2
      1. Get the code and prepare Git repository 2
      2. Problem 1: Test Graph <String> 2
      3. Problem 2: Implement Graph <String> 3
         1. Implement ConcreteEdgesGraph 3
         2. Implement ConcreteVerticesGraph 8
      4. Problem 3: Implement generic Graph<L> 8
         1. Make the implementations generic 8
         2. Implement Graph.empty() 8
      5. Problem 4: Poetic walks 8
         1. Test GraphPoet 8
         2. Implement GraphPoet 8
         3. Graph poetry slam 8
      6. Before you’re done 8
   2. Re-implement the Social Network in Lab1 8
      1. FriendshipGraph类 8
      2. Person类 8
      3. 客户端main() 9
      4. 测试用例 9
      5. 提交至Git仓库 9
   3. Playing Chess 9
      1. ADT设计/实现方案 9
      2. 主程序MyChessAndGoGame设计/实现方案 9
      3. ADT和主程序的测试方案 9
4. 实验进度记录 10
5. 实验过程中遇到的困难与解决途径 11
6. 实验过程中收获的经验、教训、感想 11
   1. 实验过程中收获的经验和教训 11
   2. 针对以下方面的感受 11

# 实验目标概述

第二次实验主要训练抽象数据类型(ADT)的设计、规约、测试，并使用面向对象 编程(OOP)技术实现 ADT。比如从给定应用问题中识别所需的ADT，设计ADT的spec、RI、AF、checkRep（），同时要涉及了范型ADT的设计，包括对其的不同实现的设计等；另外，练习了对ADT测试用例以及测试策略的设计，对rep exposure的考量等。

总的来说，对ADT 的设计及其OOP实现进行较为系统的练习。

# 实验环境配置

过程：

本次试验的工具基本在第一次实验过程中已经准备的比较到位了，因此本次试验的配置主要是对于课程已有项目的clone过程，通过实验指导上的url，我在本地仓库中使用 git clone url指令，将P1的源码克隆到本地，以此为基础进行实验。

问题与困难：

本次试验中P1提供的源码直接以项目的方式存在，我希望将其中的src和test文件单独插入到我已经建好的项目中，但在源代码文件顶部总是出现包环境有误的问题，后来我了解到是配置文件的问题。我是通过后来直接在提供的项目上进行实验这种粗暴的方式解决的，但我正在学习文件配置，希望以后能用更合理的方式进行处理。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

https://github.com/ComputerScienceHIT/Lab2-1180500301

# 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

我的理解：首先，这个任务的目标要求是让我们实现范型图ADT的设计，并完成其的两个具体实现，最后应用其中的实现，来完成一个让句子更有诗意的小工具；我认为，在实验的过程中，可以加深我们对ADT中规约、RI、AF等内容的理解与认识，同时也是为了锻炼我们的范型应用能力、对实际问题的解决能力。

### 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。

过程：我首先通过实验指南提供的网站： https://github.com/rainywang/Spring2020\_HITCS\_SC\_Lab2/tree/master/P1 找到P1的源代码，但如上面所说，我在这一过程中遇到了一些困难，并卡了良久：本次试验中P1提供的源码直接以项目的方式存在，我希望将其中的src和test文件单独插入到我已经建好的项目中，但在源代码文件顶部总是出现包环境有误的问题，后来我了解到是配置文件的问题。我最后直接在提供的项目上进行了实验，从而解决问题。

### Problem 1: Test Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

该任务要求对Graph类的部分测试进行设计。以下是设计和实现思路及过程：

1.静态方法 ：

empty（）：直接使用提供的测试用例，分析出具体思路：调用emptyInstance() 方法与vertices（）方法，与空集合进行比较。

2.实例方法：

add(L label): 通过两次调用该方法，并且两次都输入相同信息的不同label，希望第一次成功第二次失败（已加入）。

set(label,label,int):首先，输入的数字可以分为：非法（负数），0，正数；

同时，调用顺序也会影响结果。因此我的实现过程为，第一次用例输入负数，希望返回0；第二次为正数，希望返回0；第三次为0，希望返回第一次的正数。

remove(label):输入可分为两类：已经被加入的label，没有被加入的label；两个用例分别测试两种情况。

Vertices():该方法返回对象图的顶点集合，对象图可等价划分为：图中没有顶点，图中有若干个顶点。关于实现，我通过借助一个新定义的集合，辅助测试两种情况。

sources():该方法返回给定label的关联边的源顶点的map，对象定点划分：没有入边，有入边。借助自定义的map辅助测试两种情况。

targets():该方法返回给定label的关联边目标顶点的map，对象定点划分：没有出边，有出边。借助自定义的map辅助测试两种情况。

结果：全部实现，测试结果会在后面任务中进行。

### Problem 2: Implement Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

#### Implement ConcreteEdgesGraph

3.1.3.1.1对Edge规约的设计：

/\*\*

\* construct an edge

\* @param two labels and a positive number

\* @return an edge

\*/

public Edge（L sourceOfEdge, L targetOfEdge, int weightOfEdge)

/\*\*

\* get the source label of the edge

\* @param no input

\* @return the source label of this edge

\*/

public L getSourceOfEdge()

/\*\*

\* get the target label of the edge

\* @param no input

\* @return the target label of this edge

\*/

public L getTargetOfEdge()

/\*\*

\* get the weight of the edge

\* @param no input

\* @return the weight of this edge

\*/

public int getWeight()

/\*\*

\* reset the weight of this edge

\* @param a nonnegative number you want the weight to be

\* @return last weight of the edge

\*/

public int resetWeight(int newWeight)

@Override

public int hashCode()

@Override

public boolean equals（）

@Override

public String toString()

3.1.3.1.2对edge测试的设计：

Edge（L sourceOfEdge, L targetOfEdge, int weightOfEdge):：

将输入的权重分为：<0 =0 >0 分别进行测试，同时两个label可分为：不同 相同；

getSourceOfEdge(L)：将输入的label分为有入边和没有入边，分别进行测试。

getTargetOfEdge(L):将输入的label分为有出边和没有出边，分别进行测试。

getWeight():仅用一个边，测试其权重是否正确。

resetWeight(int):将输入数字分为：< 0 , > 0 , =0，分别进行测试

equals():测试方法与hashcode()、toString()相同，新建一个与调用方法的对象相同的对象，辅助进行测试。

3.1.3.1.3对两个类的方法实现设计思路与过程：

concreteEdgeGraph:

首先是类相关RI、AF等文档：

Abstraction function:

AF(vertices,edges) = the vertices and edges that made up this graph

Representation invariant:

the Object in vertices must be non-null

the Edge in edges must be different,the tail and head of the edge is different and they are both in vertices

Safety from rep exposure:

all fields are private and final

all the mutators' parameters are immutable

checkrep()：遍历rep确保没有对象为null，同时两重for循环遍历确保没有重复的边。

add(L label):直接调用rep的contains()和add()两个方法进行实现。

set(L source,L target,int number):首先检查两个对象是否相同，相同直接报错返回。随后检查是否已有该边，若有根据输入的weight来修改或删除边；否则，添加边或报错。

remove(L label):将label对应的顶点从相关rep中删除，并用iterator遍历边的list删除相关边。

public Set<L> vertices()：对成员变量list做防御式拷贝并返回；

sources（L label）:返回一个map，遍历边list将target为label的边的source label加入map的keyset，并将该边weight设为对应值。

targets（L label）:返回一个map，遍历边list将source为label的边的target label加入map的keyset，并将该边weight设为对应值。

containEdge(L source,L target)：遍历边的list，寻找该边source和target与参数相同的（信息一致）边，找到返回true否则false。

toString()：只要将图的信息表达清楚即可，因此设置适当的打印格式，将类中两个成员变量打印出来。

Edge<L>：

private final L sourceOfEdge;

private final L targetOfEdge;

private int weightOfEdge;

首先是类相关RI、AF等文档：

Abstraction function:

AF(sourceOfEdge,targetOfEdge,weightOfEdge) = the tail and head of the edge with its weight

Representation invariant:

sourceOfEdge and targetOfEdge must be non null,weightOfedge must be positive

Safety from rep exposure:

all the fields are private

ourceOFEdge and targetOfEdge are immutable so all the rep are safe

Public Edge(L sourceOfEdge, L targetOfEdge, int weightOfEdge):先检测weightOfEdge是否<=0，若是，报错exit；否则，根据输入的参数设置该边属性。

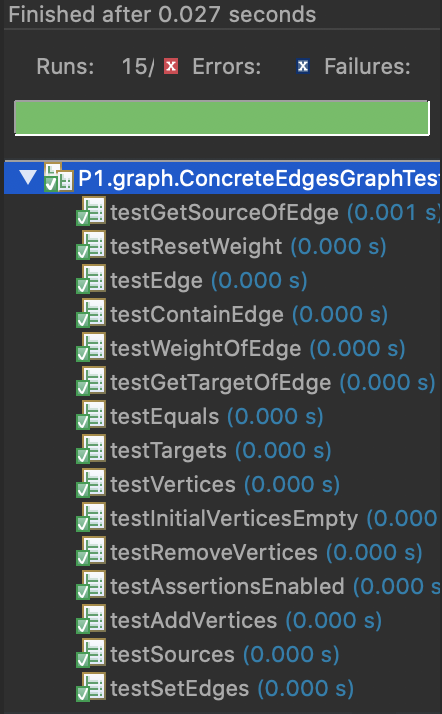
Private void checkRep():检查weight是否>0,两个label是否为null。

public L getSourceOfEdge():直接返回sourceOfEdge；

public L getTargetOfEdge():直接返回targetOfEdge；

Public int resetWeight(int weight):检查参数是否符合RI，若是则返回旧的weight，并修改该边weight；否则return0.

Public String toString():返回sourceOfEdge,和targetOfEdge的toString()形式。

结果：

#### 

#### Implement ConcreteVerticesGraph

**3.1.3.2.1对Vertex<L>规约的设计：**

/\*\*

\* get the name/information from this Vertex

\* @return this vertices' information

\*/

Public L getVertices()

/\*\*

\* get the vertices that have edge from it

\* @return the vertices and corresponding weight

\*/

public Map<Vertex<L>,Integer> getToVertices()

/\*\*

\* get the vertices that have edge to it

\* @return the vertices and corresponding weight

\*/

public Map<Vertex<L>,Integer> getFromVertices()

/\*\*

\* set the head vertex and weight to an edge

\* @param vertex,the vertex you want to set edge with it

\* @param weight, a nonnegative number

\* @return  whether it is changed

\*/

public boolean setToVertices(Vertex<L> vertex,Integer weight);

/\*\*

\* set the tail vertex and weight to an edge

\* @param vertex,the vertex you want to set edge with it

\* @param weight, a nonnegative number

\* @return  whether it is changed

\*/

public boolean setFromVertices(Vertex<L> vertex,Integer weight)

**3.1.3.2 Vertex测试的设计：**

public Vertex(L vertices)：与public L getVertices()一同测试，创建一个合法的用例，调用该方法判断是否正确。

public Map<Vertex<L>,Integer> getToVertices()：将调用方法的边进行划分，分为有出边和无出边，观察测试结果。

public Map<Vertex<L>,Integer> getFromVertices()：设计思想与上一个函数相同。

public boolean setToVertices(Vertex<L> vertex,Integer weight)：可讲weight划分为  0,>0,<0

，观察结果。

public boolean setFromVertices(Vertex<L> vertex,Integer weight)：测试方法与上一方法相同。

**3.1.3.3 concreteVertexGraph和Vertex类的实现：**

**concreteVertexGraph：**

RI、AF等文档：

// Abstraction function:

    //   AF(vertices) = the vertices that compose this graph

    // Representation invariant:

    // Vertex must be non-null,and there are no same vertices

    // Safety from rep exposure:

    //  the field is private and final

    //  all the mutators' parameters are immutable

    // used defensive copy in method sources() ,targets() ,vertices

方法：

private void checkRep()：二重循环遍历成员变量list，确保没有相同的vertex。

public boolean add(L vertex) ：调用list的contains和equals方法实现。

public int set(L source, L target, int weight)：首先调用findVertex方法找到source和target对应的vertex，随后判断是否存在相应的边，若是，根据参数weight删除或者修改相应的边，返回旧权值；否则，返回0，weight不为0时设置相应的边。

public boolean remove(L vertex) ：若图中含有该顶点，则将该顶点从该顶点出边集中所有顶点的入边集中移除，同时也将将该顶点从该顶点入边集中所有顶点的出边集中移除，从图的list中将该顶点移除。

public Set<L> vertices()：对成员变量list做防御式拷贝并返回；

public Map<L, Integer> sources(L target):返回一个map，遍历边list将target为label的边的source label加入map的keyset，并将该边weight设为对应值。

public Map<L, Integer> targets(L source):返回一个map，遍历边list将source为label的边的target label加入map的keyset，并将该边weight设为对应值。

public Vertex<L> findVertex(L label, List<Vertex<L>> vertices)：要在vertices中寻找label对应的vertex，只需对vertices进行遍历即可。

**Vertex<L>：**

private final L vertices;

private Map<Vertex<L>,Integer> toVertices =new HashMap<>();

private Map<Vertex<L>,Integer> fromVertices =new HashMap<>();

RI、AF等文档：

// Abstraction function:

    //   AF(vertices,toVertices,fromVertices) = the name/information of the vertex,the map that contain the vertex that have edge with it

    // Representation invariant:

    //   vertices must be non-null,toVertices or fromVertices contain only positive weight

    // Safety from rep exposure:

    //   vertices is private and final,also the type L is an immutable type

//  all fields are private,the observer method return the unmodifiableMap,and all the parameter of

// the mutators are immutable ,which guarantee the safety

方法:

public void checkRep():检查label是否等于null，对于toVertices和fromVertices中所有的键值做检查。

public L getVertices()：返回该vertex的label，由于其为不可变类型，不需防御式拷贝。

public Map<Vertex<L>,Integer> getToVertices()：对成员变量toVertices做防御使拷贝并返回。

public Map<Vertex<L>,Integer> getFromVertices()：对成员变量fromVertices做防御使拷贝并返回。

public boolean setToVertices(Vertex<L> vertex,Integer weight)：将vertex加入该点的出边对应的顶点集中，首先判断weight是否合法，随后检查是否已经存在对应的边，若是，则修改或删除toVertices中对应的顶点和键值；否则，根据weight是否为0来选择是否添加对应顶点。

public boolean setFromVertices(Vertex<L> vertex,Integer weight)：设计方法与上一方法相同。

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

#### Make the implementations generic

设计和实现思路：

由于之前已经完成了以String为label类型的Graph接口的两个具体 类的实现，所以现在只需要对这两个实现做一下小小的改动即可：将类中成员变量、方法原来与label相关的参数的类型改动为L（任意）；同时，将Vertex和Edge、两个具体实现类的类名后加上<L>标志用于指明为范型类；最后，也要在两个具体实现类中声明范型对象（比如一个Edge型对象）时，在变量名前加<L>：截屏2020-04-07 下午5.09.31即可。

过程中遇到的困难：

关于这个任务，由于之前对范型几乎没有了解，因此在完成任务之前我先在《Java核心技术》第一卷的相关部分大体学习了范型类的知识，并有了大体的完成思路。同时，上述思路的最后一部分最初也没有了解到，源代码没有报错，但是总有关于raw type的警告，后我请教老师得以补充，消除所有警告。

#### Implement Graph.empty()

设计思路：

这个任务的实现本身很简单，只需要在方法中新定义一个具体实现的graph对象即可；重要的是了解这种方法的意义：类如List范型类通过调用具体的构造函数而被迫承受一定数量的rep风险，通过这种方法能够隐藏你的对象的具体实现，从而更好的践行OOP的实现思想。

### Problem 4: Poetic walks

#### Test GraphPoet

设计及实现思路：

类中有两个方法，如果一同测试会比较复杂，难以看出bug出处。因此尝试在后面的部分实现一个observer来观察图，在这里则用它来协助测试constructor；确保constructor正确性后，再测试poem()方法，用两个txt文件测试来增强可靠性。

至于具体思路，可以从txt文件的句子中摘一部分简化略去其中个别单词，然后判断方法调用结果是否与完整版相同。

#### Implement GraphPoet

一．设计及实现思路：

该ADT的主要用途是生成文字图，并利用生成的文字图来对输入的句子进行扩展。

第一部分，对于给定文件，可以讲文件中的文本按行读取，每行拆分成小写化的单词的字符串数组，并遍历数组将单词加入图中；为了实现边的权值的累加，在每一次迭代的过程中，保留该word，在下一次迭代中与下一次的word构成一个边，插入该边或者更新已有边的权值。

第二部分，将输入的句子拆成单词组，从第一个开始，对于组内相邻两个单词 在图中找有无从前到后距离为2的，若有找weight最大的路，把这个weight最大的路径中间的节点的单词插入一个String的list，循环直到读到最后一个单词，不用再找，直接加进去。最后将遍历list，迭代将单词组合句子返回即可。

二．RI、AF等文档：

Abstraction function:

AF(graph) = a graph that composed by the strings that the file provided,which can make the sentences that client input a poem

Representation invariant:

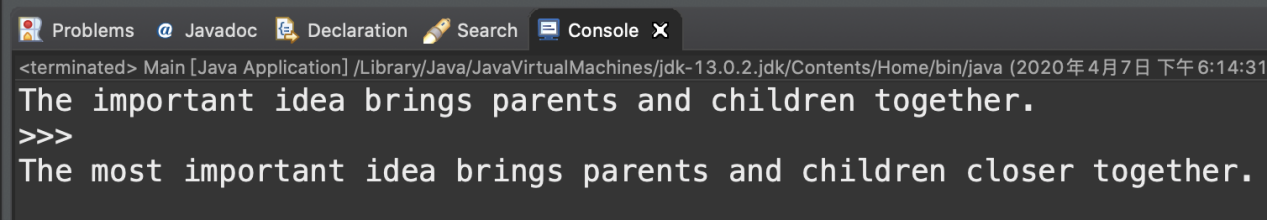
the String in graph must be non-null

Safety from rep exposure:

the field graph is private and final,there are no mutators in this class ,and all the label in the graph are immutable,thus it's safe from rep exposure

#### Graph poetry slam

设计思路：新插入一个更充实的文件，用其生成的图来进行语句的扩充，来时语句更有诗意。



### 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/%23before_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。

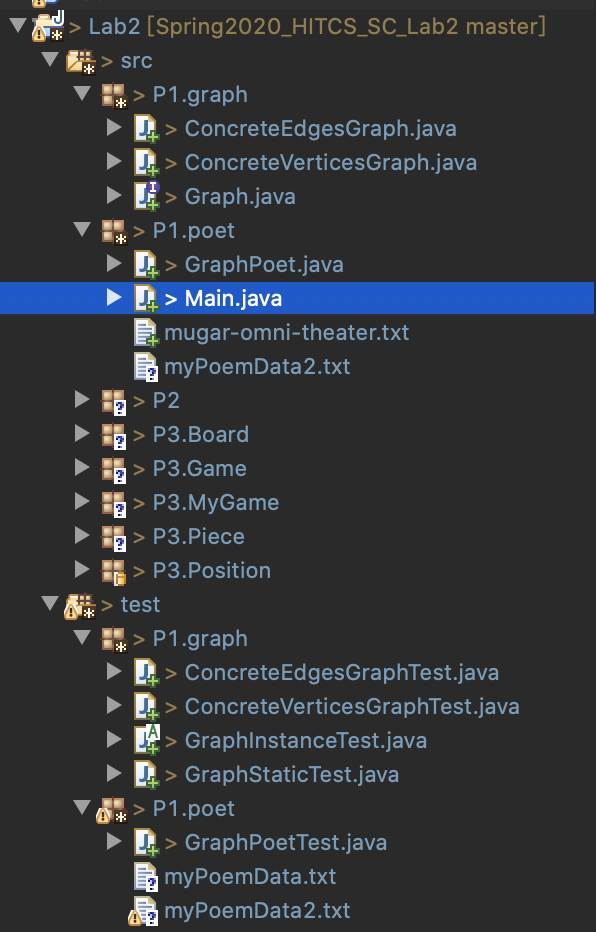
git add src

git add test

git commit -m “any message”

Git push origin master

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

理解概述：

这个任务基于在 3.1 任务 中定义的 Graph<L>及其两种 实现，重新实现 Lab1 中 的 FriendshipGraph 类，可以锻炼我们对ADT的实际应用能力，通过应用上一个任务实现的ADT，重构、简化了Lab1中的实现。

### FriendshipGraph类

设计实现思路及实现过程：

该类通过图的形式表述人际关系网,由此有如下设计

成员变量及RI:

private final Graph<Person> friendShipGraph = new ConcreteEdgesGraph<>();

Abstraction function:

AF(friendShipGraph) = the friendship graph composed by different people and there relationship

Representation invariant:

Person in friendShipGraph are different

Safety from rep exposure:

the field is private and final

the parameter add to the mutator is immutable

其中，成员变量friendShipGraph为存储Person类型结点的关系图，以上一任务中的具体实现来定义。

具体方法设计实现：

public boolean addVertex(Person newPerson)方法：直接调用friendShipGraph的add（)方法，将参数person加入图的成员变量中，如果出现已存在等问题，friendShipGraph中的方法会报错。

public int addEdge(Person fromPerson,Person toPerson)方法：同上一个方法的实现相近，直接调用friendShipGraph的set()方法，将两个结点构成的边加入图中，如果出现二者为相同结点、非图中结点等错误，set(）方法直接报错。

public int getDistance(Person fromPerson, Person toPerson)：先检查二者相同或者不在图中的情况，若是直接返回；检查通过后，用广度优先搜索，对friendShipGraph从fromPerson顶点进行广度优先遍历（利用队列），同时用一个Map来存储每个结点到fromPerson的距离，遍历过程中调用了上一任务中的targets()方法。

结果：实现结果将在3.2.3和3.2.4测试中展现。

### Person类

设计实现思路：

该ADT主要用于FriendShipGraph类中使用的immutable结点Person的定义，因此只要具有存储相应信息的成员变量，以及设计实现对应的observer、equals和hashcode方法即可。

过程：

先给出成员变量和RI等说明：

private final String personName;

Abstraction function:

AF(personName) = the name of a Person

Representation invariant:

true

Safety from rep exposure:

the field are private and final

person is String,so are guaranteed immutable

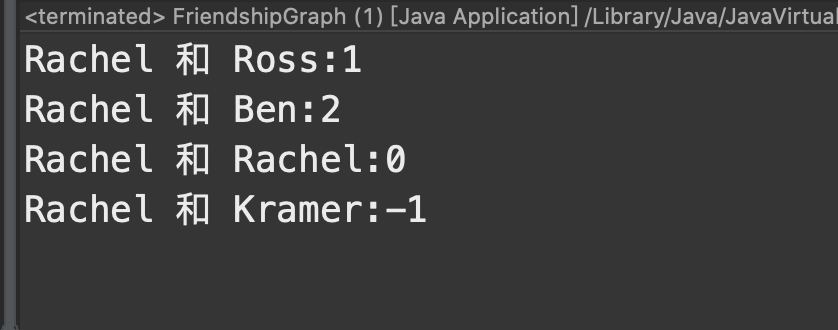
具体方法，重写toString()、equals()、HashCode()方法。

### 客户端main()

设计思路：

客户端主要用于测试以上两个类的功能，因此只需定义FriendShipgraph图，并定义数个Person类结点调用addVertex方法向图中加入，并调用addEdge方法加入数条边，随后任取几个顶点调用getDistance（）方法测试顶点间距离。

结果：



### 测试用例

设计和实现思路：

1.FriendShipgraph类：

public boolean addVertex(Person newPerson)：将输入划分为已添加过的结点和没添加过的检点，进行测试。

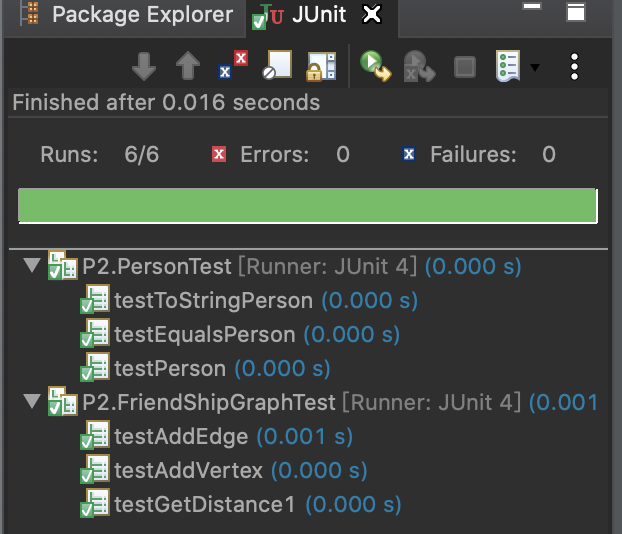
public int addEdge(Person fromPerson,Person toPerson)：将两个参数划分为：二者为相同结点，结点不在图中；对应边已添加，对应边未添加，

public int getDistance(Person fromPerson, Person toPerson)：将两个参数划分为：两者间没有路径，二者相同，二者间有一条边，二者间路径长度>1。

2.Person类：

constructor与toString()一同测试：新定义一个对象，并调用toString()方法测试其信息。

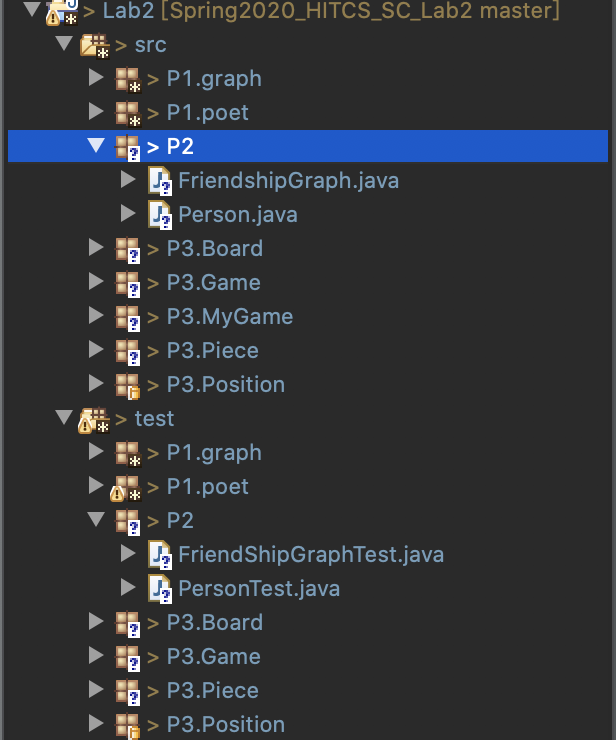
equals()：定义三个对象，两个信息相同，另外一个不同，两两比较。

结果：

### 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。

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。

设计ADT如下：

一．Board(抽象接口),ChessBoard,GoBoard:先来说明一下这几个ADT的作用，Board相关ADT为游戏提供特定的“棋盘”（实际上就是字符串的二维矩阵），并提供查看位置、打印棋盘等相关操作。

ChessBoard:

1.Mutability:mutable

2.Rep: private final String[][] chessBoard = new String[8][8];

3.Abstraction function:

AF(chessBoard) = the board that you will play the game on

4. Representation invariant:

the field must be non null,and the reference could not be changed

5.Safety from rep exposure:

the field is private and final,all the mutators do not change the rep invariant,the observer will make copy defensive

1. 设计及实现思路：

/\*\*

\* the constructor

\*/

public ChessBoard()

实现:调用下面提到的initailBoard()方法。

/\*\*

\* initial the game board

\*/

public void initailBoard()

实现：将成员变量chessBoard 中每个元素设为“empty”。

/\*\*

\* return the information that this place hold

\* for example,return "empty" or "playerA's white piece"

\* @param thisPlace,the position you want to check

\* @return the information that this place hold

\*/

public abstract String getPosition(Position thisPlace)

实现：若位置合法，对成员变量相应位置的元素做防御式拷贝并返回，否则报错。

/\*\*

\* print out the game board

\*/

public abstract void printBoard()

实现：按行和列循环遍历成员变量被打印输出，同时标记相应的行号和列号。

/\*\*

\* check if the position was legal to this board

\* @param where,the position you gonna check

\* @return true if legal,else false

\*/

public abstract boolean isLegalPlace(Position where)

实现：判断where的两个坐标是否在二维矩阵的相应范围内。

/\*\*

\* check if the position is empty on this board

\* @param where,the position you gonna check

\* @return true if empty,else false

\*/

public abstract boolean isEmpty(Position where)

实现：如果调用isLegalPlace确定位置合法且相应位置的信息为“empty”，返回true，否则false。

/\*\*

\* put the piece on a legal and empty position

\* @param whichPlayer,the player that the piece belongs to

\* @param whichPiece,the piece that you want to put

\* @param where,the place you want to put at

\* @return true if succeeded,else false

\*/

public boolean putPiece(Player whichPlayer,Piece whichPiece,Position where)

实现：调用isLegalPlace()和isEmpty()两个方法判断位置是否可以放置棋子，若是则在二维矩阵chessBoard的相应位置填写相关信息。

/\*\*

\* make a legal place where has existed a piece empty

\* @param where,the position you want to make it empty

\* @return true if succeeded,else false

\*/

public boolean makePositionEmpty(Position where)

实现：调用isLegalPlace()和isEmpty()两个方法判断位置是否符合spec，若符合则将其对应元素改为“empty”。

GoBoard：

1. mutability：mutable
2. Rep：private String[][] goBoard = new String[19][19];

3.Abstraction function:

// AF(goBoard) = the board that you will play the game on

4.Representation invariant:

// the field must be non null,and the reference could not be changed

5.Safety from rep exposure:

// the field is private and final

// all the mutators do not change the rep invariant

// the observer will make copy defensive

1. 设计与实现思路：

// constructor

public GoBoard()

实现：同ChessBoard相同

/\*\*

\* initial the game board

\*/

public void initailBoard()

实现：将成员变量goBoard 中每个元素设为“empty”。

/\*\*

\* return the information that this place hold

\* for example,return "empty" or "playerA's white piece"

\* @param thisPlace,the position you want to check

\* @return the information that this place hold

\*/

public abstract String getPosition(Position thisPlace)

实现：与ChessBoard相同,若位置合法，对成员变量相应位置的元素做防御式拷贝并返回，否则报错,

/\*\*

\* print out the game board

\*/

public abstract void printBoard()

实现：与ChessBoard相同,按行和列循环遍历成员变量被打印输出，同时标记相应的行号和列号。

/\*\*

\* check if the position was legal to this board

\* @param where,the position you gonna check

\* @return true if legal,else false

\*/

public abstract boolean isLegalPlace(Position where)

实现：与ChessBoard相同,判断where的两个坐标是否在二维矩阵的相应范围内。

/\*\*

\* check if the position is empty on this board

\* @param where,the position you gonna check

\* @return true if empty,else false

\*/

public abstract boolean isEmpty(Position where)

实现：与ChessBoard相同,如果调用isLegalPlace确定位置合法且相应位置的信息为“empty”，返回true，否则false。

/\*\*

\* put the piece on a legal and empty position

\* @param whichPlayer,the player that the piece belongs to

\* @param whichPiece,the piece that you want to put

\* @param where,the place you want to put at

\* @return true if succeeded,else false

\*/

public boolean putPiece(Player whichPlayer,Piece whichPiece,Position where)

实现：调用isLegalPlace()和isEmpty()两个方法判断位置是否可以放置棋子，若是则在二维矩阵goBoard的相应位置填写相关信息。

/\*\*

\* make a legal place where has existed a piece empty

\* @param where,the position you want to make it empty

\* @return true if succeeded,else false

\*/

public boolean makePositionEmpty(Position where)

实现：调用isLegalPlace()和isEmpty()两个方法判断位置是否符合spec，若符合则将其对应元素改为“empty”。

1. Position
2. mutability：immutable

2.Rep：private int x，private int y;

3.Abstraction function:AF(x,y) = the x and y coordinate of the position

4.Representation invariant:true

5.Safety from rep exposure:all fields are private，there are no mutator

6.设计及实现思路：

// constructor

public Position(int x, int y)

实现：将两个成员变量设置为对应参数值。

/\*\*

\* get the x coordinate of this position

\* @return,x coordinate of this position

\*/

public int getX()

实现：返回成员变量x即可。

/\*\*

\* get the y coordinate of this position

\* @return,y coordinate of this position

\*/

public int getY()

实现：返回成员变量y即可。

public int hashCode()、public boolean equals(Object obj)实现：使用与两个成员变量有关的IDE提供的默认实现即可。

三．Piece：用于定义作为棋子的对象，可以实现返回、修改棋子的位置，返回棋子名等。

1. mutability：mutable
2. Rep：

private final String nameOfPiece;//棋子的名字

private Player belongTo; //属于那个棋手

private Position itsPosition;//

3.Abstraction function:

AF(nameOfPiece,belongTo,itsPosition) = the name of the piece,who does it belongs to,and where is it

4. Representation invariant:all fields must be non null

5. Safety from rep exposure:

all fields are private ，all the mutators do not change the rep invariant，the observer will make defensive copy

1. 设计及实现思路：

// constructor

public Piece(String nameOfPiece,Player belong, Position where)

实现：设置成员变量并checkRep。

/\*\*

\* get the position of this piece

\* @return

\*/

public Position getPosition()

实现：对棋子的位置成员变量itsPosition做防御式拷贝并返回。

/\*\*

\* reset the position of this piece

\* @param newPosition

\*/

public void setPlace(Position newPosition)

实现：用输入参数的两个坐标重新设置棋子的位置itsPosition。

@Override

public String toString()

实现：返回棋子的名字即可，因为其他信息有相关函数进行返回，或不需要使用。

1. Player:用于定义表示玩家的对象，可以实现获取玩家姓名、判断某个棋子是否属于某个玩家、根据位置找到玩家的棋子等功能。
2. mutability：mutable

2.Rep：private final String playerName;

private Set<Piece> owningPieces = new HashSet<>();

3.Abstraction function:

AF(playerName,owningPieces) = the name of this player and his pieces

1. Representation invariant:the Piece in owningPiecews must be non-null

5.Safety from rep exposure:

all fields are private ，mutators are all protected and do not be exposed to the client

6.设计及实现思路：

//constructor

public Player(String playerOne)

实现：只需要设置成员变量playerName即可

/\*\*

\* get the name of this player

\* @return the name of this player

\*/

public String name()：

实现：返回成员变量layerName。

/\*\*

\* check out if this player has a piece which has similar inforamtion with this one

\*/

public boolean containsPiece(Piece thisPiece)

实现：遍历成员变量owningPieces，若其中有piece信息与其相同，则返回true否则false；

/\*\*

\* find a non null piece which belong to this player

\* @param the piece that has the same information with the piece you want to get

\* @return the real piece of this player

\*/

protected Piece findPiece(Piece thisOne)

实现：调用containsPiece方法确定是否含有该棋子，若是再遍历找到集合中对应的真正的棋子并返回，否则报错。

/\*\*

\* get one of the piece of this player by input its position

\* @param where , the position of the piece you want to get

\* @return the piece you want to get

\*/

protected Piece findPieceByPosition(Position where)

实现：根据输入的位置，遍历owningPieces集合中所有元素找到位置与参数相同的返回即可。

/\*\*

\* get the number of this player's remianing pieces

\* @return the number of this player's remianing pieces

\*/

public int getRemainingPieces()

实现：返回owningPieces的size即可。

/\*\*

\* add a new piece to this player

\* @param newPiece,the piece you want to add to him

\*/

protected boolean addPieceToHim(Piece newPiece)

实现：对owningPieces调用add()方法加入参数棋子并返回其返回值即可，同时也做一些鲁棒性处理。

/\*\*

\* remove a piece from this player

\* @param piece, the exact piece you want to move from him

\*/

protected boolean removePieceFromHim(Piece piece)

实现：对owningPieces调用remove()方法移除参数棋子并返回其返回值即可，同时也做一些鲁棒性处理。

五．Action、ChessAction、GoAction：用于定义负责协调棋子在棋盘上移动、吃子等游戏操作的对象，也就是实现游戏基本动作。其中，Aciton为接口，ChessAciton、GoAction为其具体实现。

ChessAction:

1. mutability：mutable
2. Rep：private final ChessBoard onThisBoard;
3. Abstraction function:

AF(onThisBoard) = the game board that we play on

4.Representation invariant:

onThisBoard must be non null,and the reference could not be changed

5.Safety from rep exposure:

the field is private and final，all the mutators do not change the rep invariant

1. 设计与具体实现：

// constructor

public ChessAction(ChessBoard onThisBoard)：

实现：将成员变量CHessBoard设置为参数棋盘。

/\*\*

\* put a piece on the board

\* @param whichPlayer,the player who would like to put the piece on board

\* @param whichPiece,the piece that will be put(must belong to this player)

\* @param where,the place to put

\* @return true if succeed,else false

\*/

public abstract boolean put(Player whichPlayer, Piece whichPiece, Position where);实现：检测位置where、棋子whichPiece的合法性，通过后调用成员变量onThisBoard的putPiece()方法和棋子的SetPlace()方法即可；否则报错。

/\*\*

\* test if the given player could move his piece from the given place

\* @param whichPlayer,the player that want to move his piece

\* @param fromWhere,from where you want to move

\* @return true if you there's a exact piece of yours at this place,else false

\*/

public boolean playerCouldMoveFrom(Player whichPlayer,Position fromWhere)

实现：该方法主要用于辅助后面的几个方法的实现。调用成员变量的isEmpty()方法和whichPlayer的containsPiece()方法，确认该位置有棋子且棋子属于该玩家即可。

/\*\*

\* move a piece of this player which has existed on the board to another place on this board

\* @param whichPlayer,the player who want to move his piece

\* @param fromWhere,the place of the piece that you want to move now

\* @param toWhere,the place you want to move it to

\* @return true if you succeeded,else false

\*/

public boolean move(Player whichPlayer,Position fromWhere, Position toWhere)

实现：先做一些鲁棒性处理比如判断判断两个位置是否相同、调用上面提到的playerCouldMoveFrom方法等，若没有问题调用findPieceByPosition(fromWhere)方法找到要移动的棋子，然后调用该类put(whichPlayer, newPiece, toWhere) 方法将该棋子放置在新位置，随后onThisBoard调用makePositionEmpty()方法将原位置清空。

/\*\*

\* remove the piece from the game,just to help the method eatPiece()

\* @param thisPlayer,whom the piece belongs to

\* @param thisPiece,the exact piece you want to remove

\*/

protected void remove(Player thisPlayer,Piece thisPiece)

实现：该方法是为下面的eatPeice()方法提供辅助，负责将棋子从游戏中移除。方法为protected类型，不会被客户端恶意调用，同时也只需被eatPiece()方法所负责。只需要对onThisBoard调用makePositionEmpty()将位置清空，并对被吃子的Player调用removePieceFromHim()方法移除该棋子即可。

/\*\*

\* help this player eat a piece of another player

\* @param whichPlayer,the player who want to eat other's piece

\* @param anotherPlayer,the player whose piece could be eaten

\* @param fromWhere,the position of the piece which will eat other piece

\* @param toWhere,the position of the piece that will be eaten

\* @return true if succeeded,else false

\*/

public boolean eatPiece(Player whichPlayer,Player anotherPlayer,Position fromWhere, Position toWhere)

实现：首先像上面说到的movePiece()方法一样做一些鲁棒性的处理，确认合法可行后，调用类中remove()方法移除对方棋子，并调用move()将该玩家的棋子移到对应位置即可。

GoAction：

1. mutability：mutable
2. Rep：private GoBoard onThisBoard;

3.// Abstraction function:

// AF(onThisBoard) = the game board that we play on

4. Representation invariant:

// onThisBoard must be non null,and the reference could not be changed

5.Safety from rep exposure:

// the field is private and final，all the mutators do not change the rep invariant

1. 设计和实现思路：

// constructor

public GoAction(GoBoard onThisBoard)

实现：和ChessAction相近，设置成员变量即可。

/\*\*

\* put a piece on the board

\* @param whichPlayer,the player who would like to put the piece on board

\* @param whichPiece,the piece that will be put(must belong to this player)

\* @param where,the place to put

\* @return true if succeed,else false

\*/

public abstract boolean put(Player whichPlayer, Piece whichPiece, Position where);

实现：和ChessAction相近，即检测位置where、棋子whichPiece的合法性，通过后调用成员变量onThisBoard的putPiece()方法和棋子的SetPlace()方法即可；否则报错。

/\*\*

\* this player move a piece of another player on this position

\* @param the player who wants to move other's piece,the position of the piece to be moved

\* @return true if succeed ,else false

\*/

public boolean removeGoPiece(Player thisPlayer,Player anotherPlayer,Position where)

实现：和ChessAction中的eatPiece()方法相近，先做一些鲁棒性处理，确认合法可行后调用onThisBoard的makePositionEmpty()方法将对应位置清空，并调用对方Player的removePieceFromHim()方法将被提的棋子移除。

1. Game、ChessGame、GoGame：用于接受客户端对实现游戏所需各个操作的调用，相当于面向客户端的一个平台，这几个类通过调用上述五个类中实现的方法来实现平台的操作。

ChessGame：

1. mutability：mutable
2. Rep：

private final Board gameBoard = new ChessBoard();

private final Action gameAction;

private final Player Player1 ,Player2;

1. Abstraction function:

AF(gameBoard,gameAction,Player1,Player2) = the board that you will play the game on,the two player that will play this game，and gameAction is the tool to play the game

1. Representation invariant:

all fields must be non null,and the reference could not be changed

1. Safety from rep exposure:

all fields are private and final，all the mutators do not change the rep invariant，the observer that will return an object is protected from the client

1. 设计和实现方法：

// constructor

public ChessGame(String player1Name,String player2Name)

实现：在构造器中用gameBoard来定义gameAction的具体对象，同时定义两个Player的具体对象，并调用下面说到的InitGame()方法来初始化游戏。

/\*\*

\* initial the game

\* @param Player1,this game's player1

\* @param Player2,this game's player2

\*/

private void initGame(Player Player1,Player Player2)

实现：该方法为私有方法，用于在棋盘上事先摆好双方的十六个棋子。在所有的相关位置上调用下面的putPieceOnBoar()方法即可。

/\*\*

\* put a piece that belongs to this player on board

\* @param the player's id,the piece's name,the place you want to put

\* @return true if succeeded ,else false

\*/

protected boolean putPieceOnBoard(Player newPlayer,String newpiece,Position newPosition)

实现：根据输入参数新定义一个棋子，并对newPlayer调用addPieceToHim()方法将该棋子加入该玩家的棋子集合中，随后对gameAction调用put()将棋子放置即可。

/\*\*

\* move a piece of this player from one position to another

\* @param newplayer ,the name of the player

\* @param fromX,the x coordinate of where the piece from

\* @param fromY,the y coordinate of where the piece from

\* @param toX,the x coordinate of where the piece to

\* @param toY,the y coordinate of where the piece to

\* @return true if succeed, else false

\*/

public boolean movePiece(String newplayer,String fromX,String fromY,String toX,String toY)

实现：先做鲁棒性处理；若该局游戏存在该玩家，则调用gameAction的move()方法并返回其返回值，否则报错。

/\*\*

\* used to provide tool to other method of this class

\* @param playerToFind,the player that has same information

\* @return

\*/

protected Player findRealPlayer(Player playerToFind)

实现：将参数与Player1和Player2两个成员变量做比较，若对应其中一者将其返回，否则返回null。

/\*\*

\* move a piece of one of the player to a place where has already

\* existed a piece that belongs to another player

\* @param newplayer's name, the x and y coordinate of from where and to where

\* @return true if succeed,else false

\*/

public boolean eatPiece(String newplayer,String fromX,String fromY,String toX,String toY)

实现：调用上述的findRealPlayer()方法判断是否含有此玩家，若是，对gameAciton调用eat（）方法即可。

/\*\*

\* find out if there's a piece on this place

\* @param the x and y coordinate of the position

\* @return true if there have one, else return false

\*/

public abstract boolean occupationOfThisPosition(String X,String Y)

实现：先做鲁棒性处理，若坐标对应的位置不合法或者位置为空，返回false，否则对Player调用findPieceByPosition()方法判断该位置的棋子属于哪个玩家，并提示相关信息，返回true。

/\*\*

\* get the number of the left pieces

\* @param the name of the Player you want to check

\* @return the number of his remaining pieces

\*/

public abstract int getnumberOfRemainingPieces(String playerName)

实现：判断是否具有该玩家，若有对相应Player调用getRemainingPieces()方法返回即可。

/\*\*

\* print the board

\*/

public abstract void printBoard()

实现：调用gameBoard的printBoard()方法即可。

GoGame:

1. mutability：mutable
2. Rep：  
   private Board gameBoard = new GoBoard();

private Action gameAction;

private Player Player1 ,Player2;

3.Abstraction function:

// AF(gameBoard,gameAction,Player1,Player2) = the board that you will play the game on,the two player that will play this game

// and gameAction is the tool to play the game

4.Representation invariant:

// all fields must be non null,and the reference could not be changed

5.Safety from rep exposure:

// all fields are private and final

// all the mutators do not change the rep invariant

// the observer that will return an object is protected from the client

1. 设计及实现思路：

// constructor

public GoGame(String player1Name, String player2Name)

实现：与ChessGame相近，在构造器中用gameBoard来定义gameAction的具体对象，同时定义两个Player的具体对象，并调用下面说到的InitGame()方法来初始化游戏。

/\*\*

\* find the real id of the wanted player

\* @param cloned player who has the same information but different id

\* @return the real player of this game

\*/

protected Player findRealPlayer(Player playerToFind)

实现：与ChessGame相同的实现。

/\*\*

\* put a piece that belongs to this player on board

\* @param the player's name,the piece's name,the x and y coordinate of the place you want to put

\* @return true if succeeded ,else false

\*/

public boolean putPieceOnBoard(String newplayer,String newpiece,String X,String Y)

实现：先做鲁棒性处理判断玩家是否存在且该棋子是否是该玩家所能够操控的类型，若是，对Player调用addPieceToHim()将新定义的棋子加入其集合中，随后调用gameAction的put(newPlayer, newPiece, newPosition) 函数，返回其返回值，当然如果返回错误值需要将加入玩家集合的棋子再移除。

/\*\*

\* remove a existed piece of another player

\* @param newplayer's name, the x and y coordinate of from where and to where

\* @return true if succeed,else false

\*/

public boolean removeOthersPieceFromBoard(String newplayer,String X,String Y)

实现：根据输入参数找到对应的玩家，并对gameAction调用removeGoPiece()方法，将该位置对应的棋子删除即可。

以下三个方法的实现思路均与ChessGame相同：

/\*\*

\* find out if there's a piece on this place

\* @param the x and y coordinate of the position

\* @return true if there have one, else return false

\*/

public abstract boolean occupationOfThisPosition(String X,String Y)

/\*\*

\* get the number of the left pieces

\* @param the name of the Player you want to check

\* @return the number of his remaining pieces

\*/

public abstract int getnumberOfRemainingPieces(String playerName)

/\*\*

\* print the board

\*/

public abstract void printBoard()

### 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。

设计实现方案及执行过程展示：

1.大体设计思想：

先从控制台读取用户输入的两个玩家名，并根据玩家的输入确定游戏类型（Chess or Go），用两个玩家名来定义一个对应的游戏对象，并提示游戏开始。

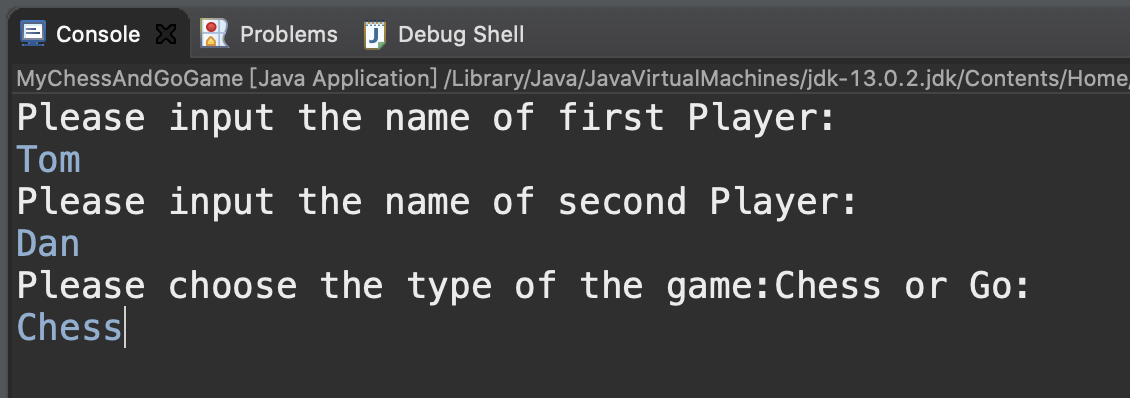
在游戏开始后，定义两个字符串数组类型的list，用于存储玩家指令的历史记录，并调用一个打印帮助信息的方法来给玩家进行提示。随后while循环，每次循环对应一个回合，每个回合进行两轮玩家操作：玩家一一轮，玩家二一轮。

每一轮操作玩家可随意选取该游戏中对应的操作（如Chess就可选取eat、move等，Go可选取put、remove等操作），程序读入对应的选择并调用Game类中相应的方法进行游戏操作，直到玩家选择end操作结束游戏。结束后跳出在开始我们说的while循环，并打印两个玩家的历史操作指令。

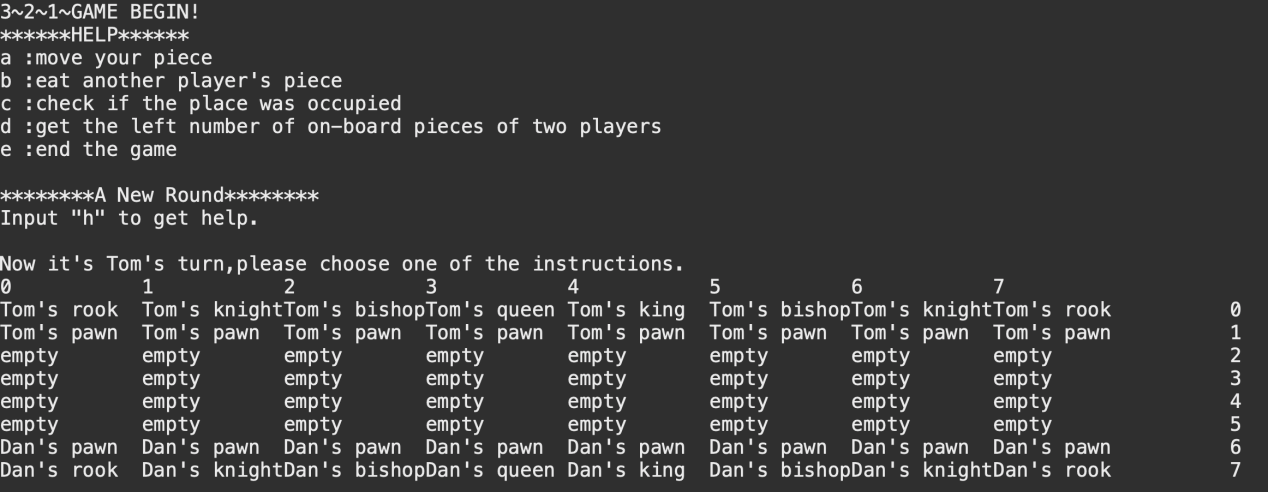
1. 对执行过程的介绍展示：

以ChessGame为例：

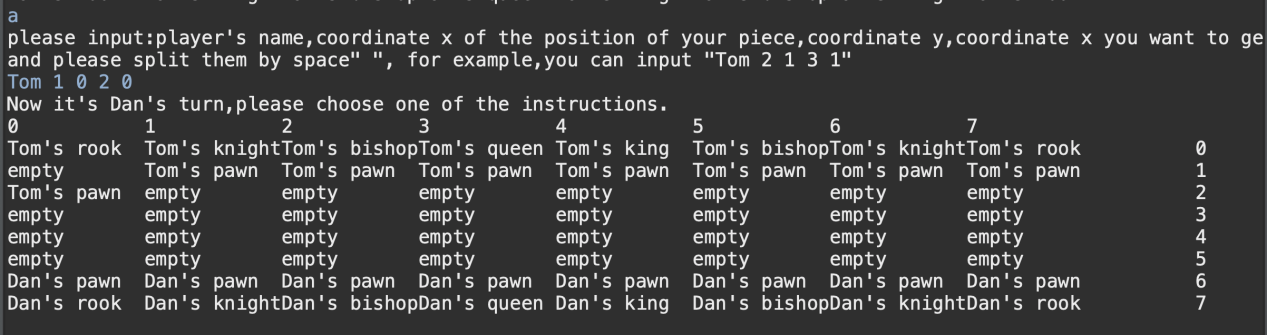
开始游戏后提示输入玩家名称以及游戏类型：



随后提示游戏开始，提供帮助指南，开始第一回合：

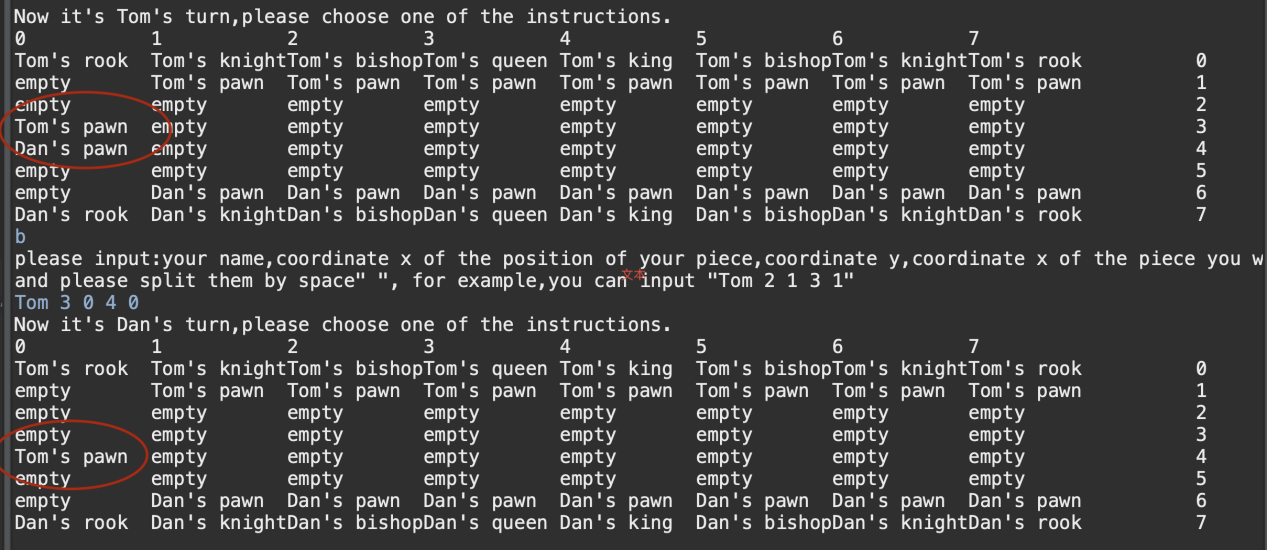


随后你可以根据帮助选择一个指令，并根据提示输入具体内容：可以看到，Tom的pawn前移了一格



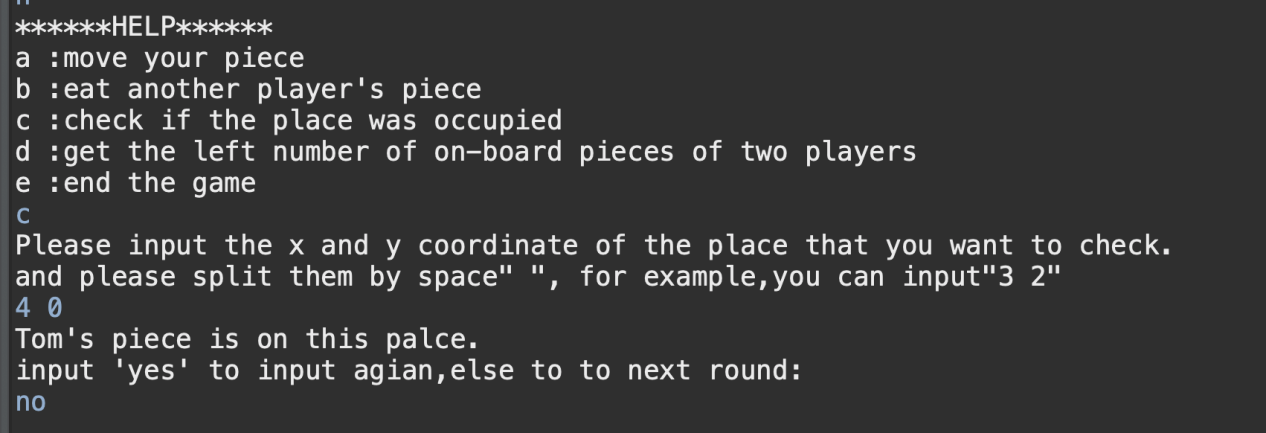
这里应该注意，我们在输入指令选择a后，又根据提示输入了”Tom 1 0 2 0”的指令，这里我们在程序中做了如下处理：将输入的这一行读入到一个字符串中，随后将其用String.split()方法存入一个字符串数组中，从而将数组元素作为参数一个个传入Game类的方法中。

若干回合后，我们调用了吃子的操作：



可以看到玩家Dan的一颗pawn被吃掉了。

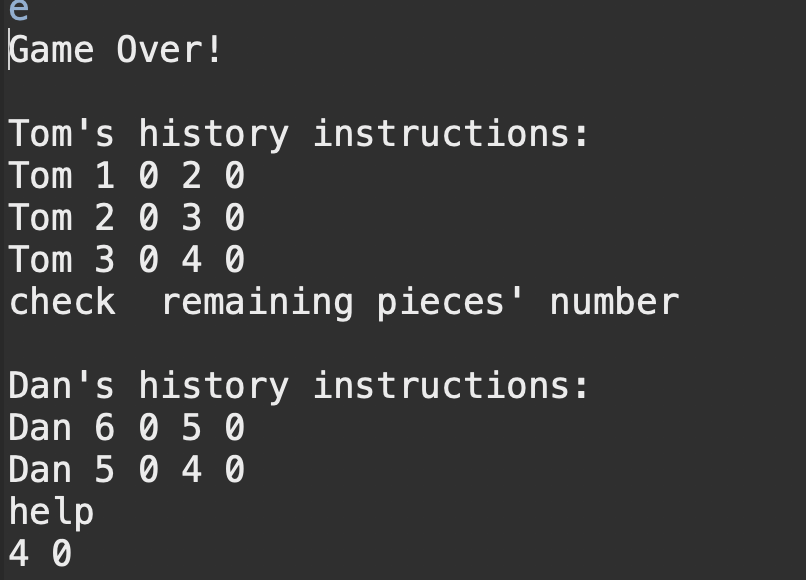
当然也可以选择查看位置指令：输入c，并根据提示输入位置4 0

这里给出了提示：Tom在这个位置有一个棋子，并询问我们是否还想查询。

我们再来查一下剩余数量：

可以看到Dan被吃掉一个棋子后，剩余棋子数少了1。

随后我们输入e结束游戏：



最后给出了玩家的操作指令。

### 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。  
一.Board、ChessBaord、GoBoard测试：

1.Instance：

initailBoard()和getPosition(Position p ):调用前者后，对后者进行多次调用，输入分别为：x = 0 y=0 , x> 0 y> 0。

Test strategy：

initialBoard()

\* test it together with getPosition()

\* invoke the initialBoard()

\* partition the position as follows:x =0,y=0 x>0,y>0

isLegalPlace(Position p)：将输入位置划分为：x=0 y=0, x<0 y<0, x和y均为正且在范围内，x和y均为正且在范围外。

Test strategy：

isLegalPlace()

\* partition the input as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

isEmpty(Position p)：将输入位置划分为：x=0 y=0, x<0 y<0, x和y均为正且在范围内，x和y均为正且在范围外；以及位置被占用，位置未被占用。

Test strategy：

isEmpty()

\* partition the input as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

\* and the position is occupied or empty

1. ChessBoardTest:

putPiece(Player, Piece, Position):将输入的位置position划分为:x =0,y=0 ；棋盘维度数>x>0,棋盘维度数>y>0； x<0,j<0 ； x>棋盘维度数,y>棋盘维度数；以及位置被占用，位置未被占用。并以此定义对应的棋子，作为调用该函数的输入。

Test strategy：

putPiceOnBoard()

// partition the position as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

// the place has a piece,the place does not has a piece

makePositionEmpty(Position)：同上一个测试基本相同：将输入的位置position划分为:x =0,y=0 ；棋盘维度数>x>0,棋盘维度数>y>0； x<0,j<0 ； x>棋盘维度数,y>棋盘维度数；以及位置被占用，位置未被占用。并以此定义对应的棋子，作为调用该函数的输入。

Test strategy：

makePositionEmpty()

// partition the position as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

// the place has a piece,the place does not has a piece

1. GoBoardTest（与ChessBoardTest测试思路相同，只是输入参数上做一些改动后）

putPiece(Player, Piece, Position):将输入的位置position划分为:x =0,y=0 ；棋盘维度数>x>0,棋盘维度数>y>0； x<0,j<0 ； x>棋盘维度数,y>棋盘维度数；以及位置被占用，位置未被占用。并以此定义对应的棋子，作为调用该函数的输入。

Test strategy：

putPiceOnBoard()

// partition the position as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

// the place has a piece,the place does not has a piece

makePositionEmpty(Position)：同上一个测试基本相同：将输入的位置position划分为:x =0,y=0 ；棋盘维度数>x>0,棋盘维度数>y>0； x<0,j<0 ； x>棋盘维度数,y>棋盘维度数；以及位置被占用，位置未被占用。并以此定义对应的棋子，作为调用该函数的输入。

Test strategy：

makePositionEmpty()

// partition the position as follows:x =0,y=0 length>x>0,length>y>0 x<0,j<0 x>length,y>length

// the place has a piece,the place does not has a piece

1. Position

Position(int ,int )与getX()、getY()方法一同测试；由于position对于输入并没有什么限制，只需要定义一个position对象并用后两个方法测试信息即可。

Test strategy：

// test these three methods together:Position(int x,int y),getX() , getY()

// define a new Position and observe its x and y coordinate

equals()与hashCode()：根据成员变量x和y的值判断二者是否相同，也只需要定义两个Position对象判断二者是否相等即可。

Test strategy：

// define two Position instance with same coordinate

// invoke equals by one of them,and input another,observe the result

1. Piece

Piece(String, Player, Position):与toString()一同测试，只需要定义一个对象，并调用toString()方法检查信息即可。

test strategy:

// test constructor with toString()

// define a new piece and observe its name by toString()

getPosition()：获取该对象的位置，只需要定义一个对象，并调用该方法检查信息即可。

test strategy:

// no inputs

// define a new piece,and invoke this method,observe the result

setPlace(Position p)：重新设置该棋子的位置，可以将输入的位置坐标划分为 x、y=0 ， x>0,y>0

test strategy:

// partition the inputs as follows:

// the x and y of the position =0 x>0,y>0

hashCode()与equals():比较将比较的两个对象划分为：二者所含信息相同，二者所含信息不同，

Test strategy:

// no inputs

// compare two piece with same information

// compare two piece with different information

1. Player

Player(String s)与 toString()：只需要定义一个对象，并调用toString()方法检查信息即可。

Test strategy:

// test them together,input a string to the constructor

// observe it by invoke the moethod toString()

addPieceToHim(Piece p):可将输入划分为：已加入过的棋子，未加入过的棋子。

Test strategy:

// test them together,input a string to the constructor

// observe it by invoke the moethod toString()

removePieceFromHim(Piece p)对输入的划分同上：已加入过的棋子，未加入过的棋子。

Test strategy:

// test them together,input a string to the constructor

// observe it by invoke the moethod toString()

findPiece(Piece p)：将输入划分为：已加入过的棋子，未加入过的棋子。

Test strategy:

// partition the inputs as: this piece has not been added to him,piece does not belongs to him

// observe the result

findPieceByPositon(Position p)：将输入的位置划分为该位置有他的棋子，该位置有别人的棋子，该位置为空。

Test strategy:

// partition the inputs as: there is a piece of his in this place,there is not a piece on this place

// or it is a emtpy place,observe the result

getRemainingPiece():对调用其的对象进行划分：玩家分别具有 0 ,1 ,2个棋子

Test strategy:

// partition the inputs as: 0 ,1 ,2 pieces in his set

1. Action、ChessAction、GoAction：

ChessAcitonTest:

put(Player p,Piece a,Position b):将输入划分为合法、非法位置，以及空位置。

Test strategy:

// partition the inputs as follows:

// position: illegal position,legal position, position that is not empty

//testPlayerCouldMove(Player, Position):将输入划分为非法位置、空位置及合法有效位置，以及该位置的棋子属于或不属于他。

Test strategy:

// partition: there is a piece on this place,isn't a piece on this place,illegal place

// and the piece belongs to another player

//movePiece(Player，Position, Position)：将输入划分为：起始位置为空、含有对方棋子、含有有效棋子；以及起始位置或终止位置非法、合法。

Test strategy:

// partition the inputs as :

// position: from position is empty, to position has his piece,from position have other's piece

// from position is illegal,to position is illegal

//remove():类内辅助方法，不会含有非法输入，仅需要测试是否可用即可。

Test strategy:

// this is a method being protected from client and only be used in eatPiece()

// so just invoke it one time ,and check the result

//eatPiece():将输入划分为：起始位置为空、含有对方棋子、含有有效棋子；以及起始位置或终止位置非法、合法。

Test strategy:

// partition the inputs as :

// position: from position is empty, to eat position has his piece,from position have other's piece

// from position is illegal,to position is illegal,from and to are same position

GoAction：

put(Player p,Piece a,Position b):将输入划分为合法、非法位置，以及空位置。

Test strategy:

// partition the inputs as follows:

// position: illegal position,legal position, position that is not empty

removeGoPiece(Player Player Position):将输入划分为：非法位置、空位置、含有己方棋子，含有对方棋子。

Test strategy:

// partition the inputs as follows:

// position: illegal position,legal position but empty, position that have his piece,position that have other's piece

1. Game、ChessGame、GoGame

ChessGame：

ChessGame(String,String)：这里构造器将与putPieceOnBoard方法一同测试，当然与观察方法测试也可行，将输入位置分为：位置为空、位置非法、有棋子。

Test strategy：

// test it with putPieceOnBoard(Player,String,Position)

// partition the inputs of putPieceOnBoard(Player,String,Position) as :

// place already has piece,place is empty,place is illegal

//findrealPlayer(String)：将输入分为：游戏中有玩家具有此名字，游戏中没有玩家具有此名字，

Test strategy：

// partition the inputs as :

// the input player has the same information with one of the game player,

// the input player has different information with any of the game player,

//movePieceOnBoard(String,String,String,String,String)：将输入划分为：起始位置为空、含有对方棋子、含有有效棋子；以及起始位置或终止位置非法、合法。

Test strategy：

// partition the inputs as :

// position: from position is empty, to position has his piece,from position have other's piece

// from position is illegal,to position is illegal

//eatPieceOnBoard(String name,String x1,String y1,String x2,String y2)：将输入划分为：起始位置为空、含有对方棋子、含有有效棋子；以及起始位置或终止位置非法、合法。

Test strategy：

// partition the inputs as :

// position: from position is empty, to position has his piece,from position have other's piece

// to position is illegal ,to position have others piece

//occupationOfThisPosition(String,String):将输入划分为：位置为空、位置有棋子、位置非法。

Test strategy：

// partition the inputs as:

// place:position has piece,position is empty,illegal position

//getNumberOfRemianingPieces(String)：在玩家被吃掉棋子前后分别测试。

Test strategy：

// test it twice,before and after you eat a piece of this player

GoGame：

GoGame(String,String)：与上一类相同策略，构造器将与putPieceOnBoard方法一同测试，当然与观察方法测试也可行，将输入位置分为：位置为空、位置非法、有棋子。

Test strategy：

// test it with putPieceOnBoard(Player,String,Position)

// partition the inputs of putPieceOnBoard(Player,String,Position) as :

// place already has piece,place is empty,place is illegal

//findrealPlayer(String)：将输入分为：游戏中有玩家具有此名字，游戏中没有玩家具有此名字，

Test strategy：

// partition the inputs as :

// the input player has the same information with one of the game player,

// the input player has different information with any of the game player,

removeOthersPiece(String name,String x,Stringy)：将输入对应的位置分为非法、空位置，位置含有他的棋子，含有对方的棋子。

Test strategy：

// partition the inputs as :

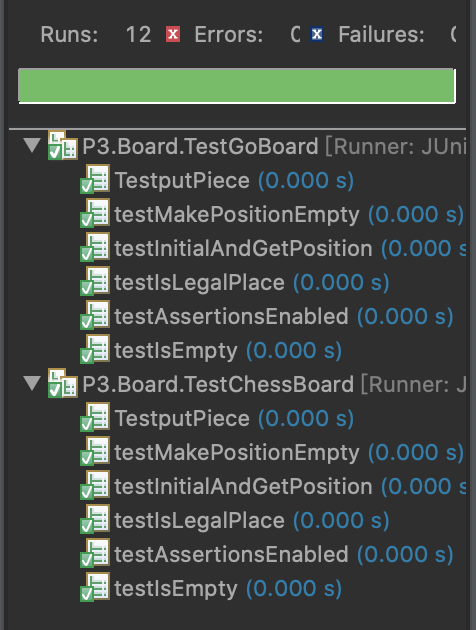
// position:position has his piece,position is empty,is illegal,has others piece

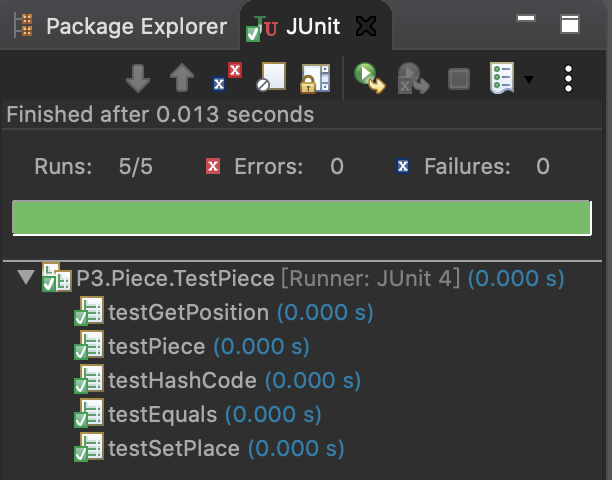
以下方法测试方案与ChessGame类相同：

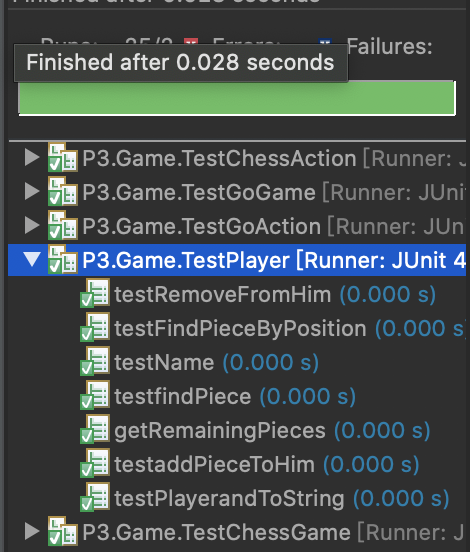
occupationOfThisPosition(String,String)

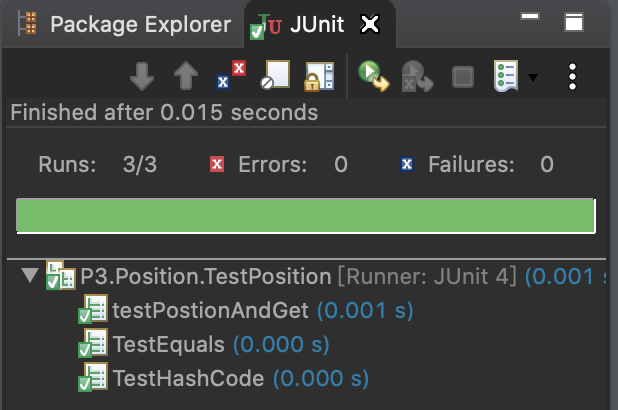
getNumberOfRemianingPieces(String)

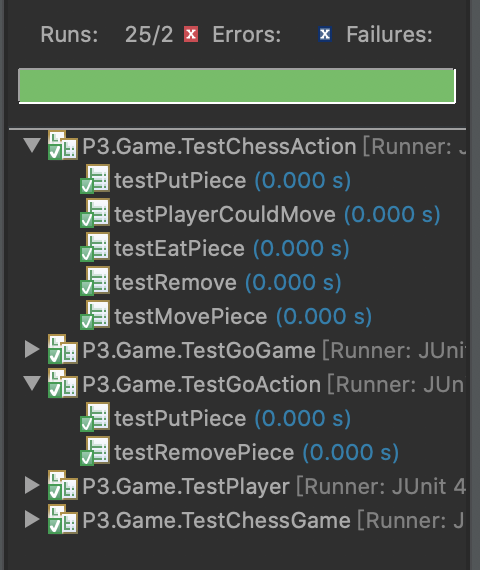
测试结果：

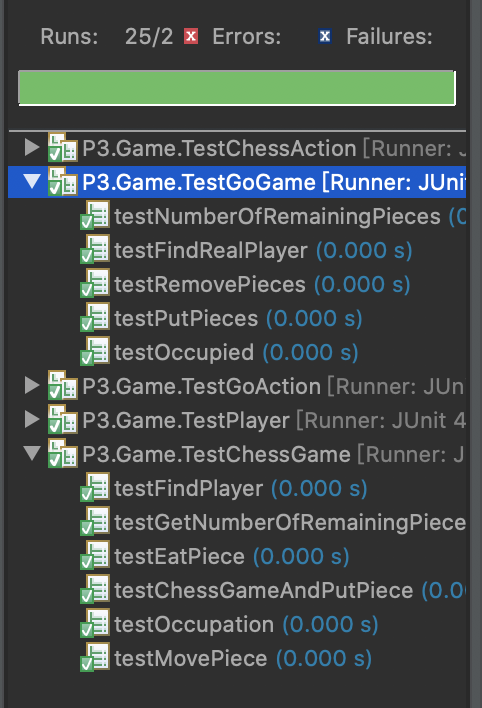
Board：

Piece：

Player：

Position：

Action：

Game:

1. 关于MyChessAndGoGame，我在该类中仅有两个额外的方法，均为打印输出帮助提示，因此只需在客户端中单独调用观察结果即可测试。对于Main()中的代码，我选择运行过程中查找错误。

# 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 |
| 3-18 | 7点-十点半 | 项目导入，查看任务 | 未完成，导入项目过程中出现问题难以解决 |
| 3-19 | 早8点-11点 | 完成P1的1、2部分 | 未完成，对任务要求的文档阅读花费很长时间 |
| 3-20 | 晚8点-11点 | 完成任务P1的2部分 | 完成 |
| 3-21 | 早9点-11点半 | 完成任务P1的3部分 | 未完成，阅读说明花费时间很长 |
| 3-21 | 下午2-5点 | 完成P1的3部分 | 完成 |
| 3-21 | 晚8-10点半 | 完成P1的四部分的测试用例编写 | 完成 |
| 3-22 | 下午2-5点 | 完成P1的四部分的实现 | 未完成，代码编写缓慢 |
| 3-22 | 晚8-10点半 | 完成P1的四部分的实现 | 完成 |
| 3-25 | 下午1点45到四点 | 完成P2的实现 | 完成 |
| 3-27 | 下午4点到七点 | 阅读P3说明文档，写出思路 | 完成部分任务，未完全将ADT设计出来 |
| 3-28 | 上午9点到11点 | 继续阅读文档=，设计思路 | 完成 |
| 3-29 | 下午2点半到六点半 | 实现其中的Position类、Piece类 | 完成 |
| 3-31 | 晚8点到十一点 | 实现Board类和Player类 | 未完成，任务量较大 |
| 4-1 | 下午实验课 | 继续实现上述两类 | 完成 |
| 4-2 | 下午2-6点 | 实现Action类 | 未完成，任务量较大 |
| 4-3 | 晚7点到11点 | 实现Game类 | 基本完成 |
| 4-4 | 上午9到11点半 | 实现Game类及测试 | 完成 |
| 4-4 | 晚7点到11点 | 实现客户端编写 | 未完成 |
| 4-5 | 上午9点到12点 | 实现客户端编写 | 完成 |
| 4-5 | 晚7点到9点 | 调试客户端 | 完成 |
| 4-6后 |  | 写报告 |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 范型的应用 | 查询《Java核心技术》相关知识，向老师询问相关技术方法，并实践应用， |
| 项目的引入，最初尝试将提供的项目拆分来放入我自己生成的项目中，但由于配置原因总是报错， | 直接在提供的项目上进行实验，后面修改相关目录结构即可。， |
| ADT的从零设计 | 多加思考各个类间的关系，并将思维导图写在纸上，帮助我成功实现。 |

# 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

经验教训总结：

通过这次的实验，我对于ADT的设计以及OOP的编程理念有了更为深刻体会，时也体会到了面向ADT的编程和直接面向应用场景编程的差异，也意识到了课程开始前的编程思维与现在具有的编程思维的转变。

这次实验任务一用到了范型编程，也让我深入学习了关于范型的知识，了解到复用性的益处。

在设计spec、RI、AF等文档说明的过程中，我对自己想要设计或者实现的ADT有了无比清晰的认识，这是以前没有体会过的，它们极大的帮助我减少了错误的产生。

同时，任务三的从零设计是一个十足有趣而又艰难的挑战，我感受到了协调ADT的难处所在，也深刻认识到利用ADT相互协调编程的益处所在。

当然，我也收获了一些教训。比如应该深刻地践行测试优先编程的理念，否则会在后期疲惫的编写测试用例，且错误可能较多。另外，持续的钻研实验对任务完成很有帮助，不应该断断续续的面对任务。

## 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

我体会到的是，面向ADT的编程可以通过多个类来实现不同的功能，每个类都容易实现也容易检查，但重在协调好各个类之间的关系；而直接面向应用场景编程则更顺序流程化，不好纠错，但协调各功能时较为简单。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用范型的编程更加具有安全性，对client暴露的信息更少。

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？

不必想到方法的具体实现，因此“破坏性”更高，测试也更有效。我目前还并没有完全适应这种方法，正在逐步适应。

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？

复用性是评价软件质量的标准之一，复用性强减少开发成本。

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？

通过不断思考、设计ADT与其AF，我基本适应掌握了抽象映射的方法；设计思考统筹各类、类中各个方法间的关系是一项艰巨的任务，需要良久的思考，但设计出以后任务就会轻松很多。

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？

使该类的功能更加明确，同时也给出规范，减少bug出现的概率，特别是有利于开发者之间的交流。

1. 关于本实验的工作量、难度、deadline。

个人认为，难度适当稍微偏上，但是可能是因为个人编程能力的原因，感觉工作量较大；由于我从第四周实验课就开始了实验，因此赶在了deadline前完成了实验，但时间并不十分充裕。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

通过可是课程的学习以及实验的进行，我对软件程序开发有了更深一步的理解，得到的收获让我感觉很充实。关于建议，我希望能将习题课提早些进行，这样可以帮助我们提早了解一些对实验有益的技术手段。