



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2020 年春季学期
计算机学院《软件构造》课程

Lab 2 实验报告

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

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	2
3.1 Poetic Walks	2
3.1.1 Get the code and prepare Git repository	2
3.1.2 Problem 1: Test Graph <String>	3
3.1.3 Problem 2: Implement Graph <String>	5
3.1.3.1 Implement ConcreteEdgesGraph	5
3.1.3.2 Implement ConcreteVerticesGraph	5
3.1.4 Problem 3: Implement generic Graph<L>	8
3.1.4.1 Make the implementations generic	8
3.1.4.2 Implement Graph.empty()	9
3.1.5 Problem 4: Poetic walks	9
3.1.5.1 Test GraphPoet	9
3.1.5.2 Implement GraphPoet	11
3.1.5.3 Graph poetry slam	10
3.1.6 Before you're done	11
3.2 Re-implement the Social Network in Lab1	12
3.2.1 FriendshipGraph 类	12
3.2.2 Person 类	12
3.2.3 客户端 main()	13
3.2.4 测试用例	13
3.2.5 提交至 Git 仓库	14
3.3 Playing Chess	14
3.3.1 ADT 设计/实现方案	14
3.3.2 主程序 MyChessAndGoGame 设计/实现方案	19
3.3.3 ADT 和主程序的测试方案	26
4 实验进度记录	31
5 实验过程中遇到的困难与解决途径	32
6 实验过程中收获的经验、教训、感想	32
6.1 实验过程中收获的经验教训	32

6.2 针对以下方面的感受	32
---------------------	----

1 实验目标概述

根据实验手册简要撰写。

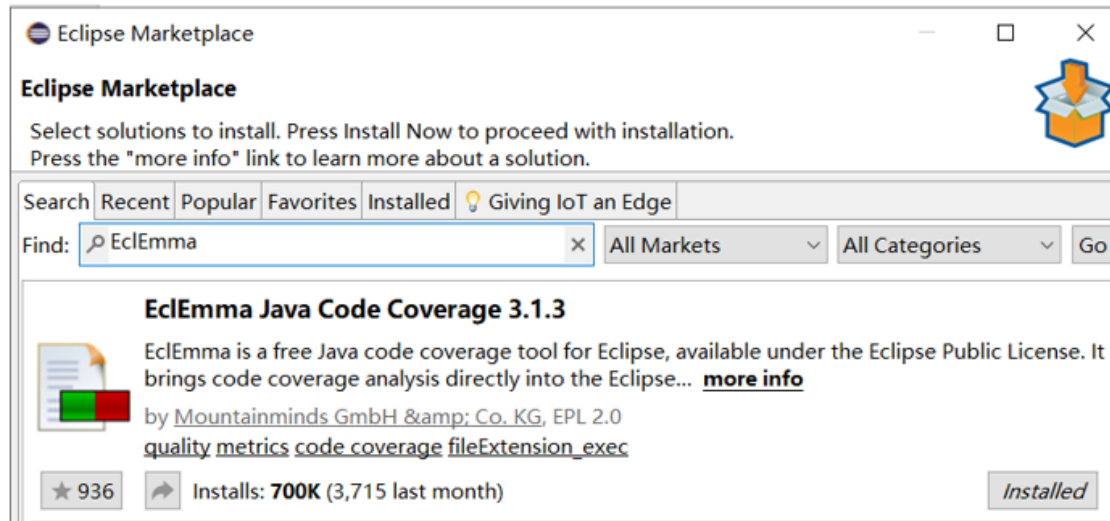
本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象编程（OOP）技术实现 ADT。具体来说：

- 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。

2 实验环境配置

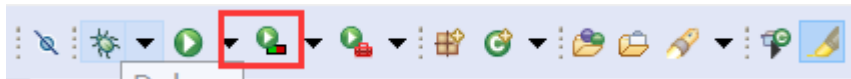
配置 EclEmma

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



③ 选择最新版进行安装即可。

④ 安装好后 Eclipse 菜单会出现图形表示安装成功



在这里给出你的 GitHub Lab2 仓库的 URL 地址（Lab2-学号）。

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

3 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！）。

3.1 Poetic Walks

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

3.1.1 Get the code and prepare Git repository

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

获取任务代码: 打开老师在 lab2 实验提供的网址，直接下载即可。

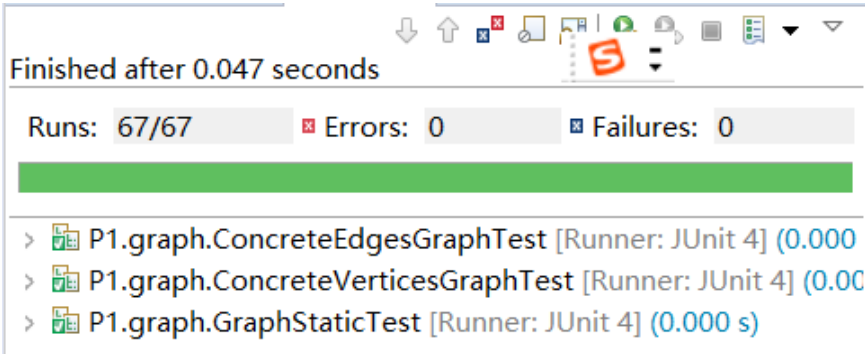
在本地创建 git 仓库: 先 git init 初始化创建一个本地仓库，然后 git clone

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

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

3.1.2 Problem 1: Test Graph <String>

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



Test Class	Success Rate	Passed	Failed	Total
P1.graph	91.9 %	1,150	101	1,251
ConcreteVerticesGraph.java	90.7 %	595	61	656
Vertex<L>	82.5 %	179	38	217
ConcreteVerticesGraph	94.8 %	416	23	439
ConcreteEdgesGraph.java	93.2 %	551	40	591
ConcreteEdgesGraph<L>	94.9 %	410	22	432
Edge<L>	88.7 %	141	18	159
Graph.java	100.0 %	4	0	4
Graph<L>	100.0 %	4	0	4

1. Test add

测试策略:

graph: empty , not empty
vertex: existed , not existed

结果: 返回 `true`: 添加点成功。 原来的点集上会增加这个新加入的顶点,
 返回 `false`: 添加点失败。说明原来的点集包含这个待加入的顶点

2. Test set

测试策略:

graph: empty , not empty
source: existed , not existed
target: existed , not existed
weight: zero , positive
edge : existed , not existed

结果: **返回 0** : 说明在原图中 source 和 target 之间不存在边。若 $\text{weight} > 0$, 则在 source 和 target 之间添加一条权重为 weight 的边; 若 $\text{weight} = 0$, **返回大于 0 的数** : 说明在原图中 source 和 target 之间存在边, 返回值即为原来的边的权重。若 $\text{weight} > 0$, 将原来的边的权重修改为 weight ; 若 $\text{weight} = 0$, 删去这条边。

3. Test remove

测试策略: **graph** : empty , not empty
vertex : not existed , existed with edges , existed without edges

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

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

4. Test vertices

测试策略:
graph : empty , not empty

结果 : 返回图中所有的点的点集

5 .Test source

测试策略:
graph : empty , not empty
target : existed with source to ,existed without source to ,not existed

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

6. Test target

测试策略:

graph : empty , not empty

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

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

3.1.3 Problem 2: Implement Graph <String>

3.1.3.1 Implement ConcreteEdgesGraph

1. 设计不变量

```
// Abstraction function:
// AF(vertices, edges) = a set of weighted directed edges
// from one source vertex to one target vertex that is different from
source vertex
//
// Representation invariant:
// weight > 0, There are at most one directed edge between source and
target
//
// Safety from rep exposure:
// vertices is mutable, so vertices() make defensive copy to avoid rep
exposure
// and each of the fields are modified by key word final and private ,
so they can't be changed from outside
```

2. 检查表示

```
public void checkRep() {
    assert vertices != null ;//检查顶点集是否为空
    for(L l : vertices) {
        assert l != null ;//检查顶点是否为空
    }
    for(Edge<L> l : edges) {
        assert l != null ;
        assert l.getWeight() > 0;//检查权重是否大于零
    }
}
```

3.实现 Edge 类

3.1 实现 Edge 类的属性

<code>private final L source</code>	边的入顶点
<code>private final L source</code>	边的出顶点

<code>private final int weight</code>	边的权重
---------------------------------------	------

3.2 实现 Edge 类的方法

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

4. 实现 ConcreteEdgesGraph<String>类

4.1 实现 ConcreteEdgesGraph<String>类的属性

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

4.2 实现 ConcreteEdgesGraph<String>类的方法

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

3.1.3.2 Implement ConcreteVerticesGraph

1. 设计不变量

```
// Abstraction function:
//  TODO
// Representation invariant:
//  Each Vertex in vertices is different from other
//
// Safety from rep exposure:
//  vertices is modified by keyword private and final
//  so it can't be changed from outside
```

2. 检查表示

```
public void checkRep() {
    assert vertices != null ;
    for(Vertex<L> l : vertices) {
        for(Integer weight : l.getRelationship().values())
            assert weight >0; //权重大于零
    }
    int size = vertices.size();
    for(int i = 0 ; i < size ; i++) {
        for(int j = i+1 ; j <size ; j ++ ){
            assert !vertices.get(i).equals(vertices.get(j)); //顶点各不相同
        }
    }
}
```

3 实现 Vertex 类

3.1 实现 Vertex 类的属性

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

3.2 实现 Vertex 类的方法

public Vertex(final L source ,final Map<L,Integer> map)	带参数的构造器
public Vertex(final L source)	不带参数的构造器
public void checkRep()	检查方法
public L getSource()	返回源顶点

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

4. 实现 ConcreteVerticesGraph <String>类

4.1 实现 ConcreteVerticesGraph <String>类的属性

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

4.2 实现 ConcreteVerticesGraph <String>类的方法

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

3.1.4 Problem 3: Implement generic Graph<L>

3.1.4.1 Make the implementations generic

1. 将具体类的声明修改为:

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

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

3.1.4.2 Implement `Graph.empty()`

```
public static <L> Graph<L> empty() {
    return new ConcreteEdgesGraph<L>();
}
```

3.1.5 Problem 4: Poetic walks

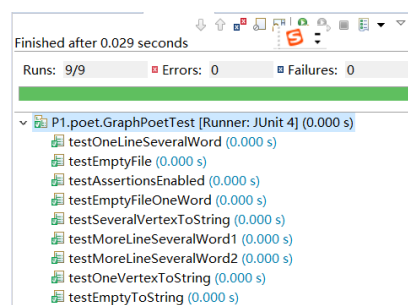
3.1.5.1 Test `GraphPoet`

测试策略:

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

Input : empty string , one word ,several words

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



▼ P1.poet	87.5 %	274	39	313
▼ Main.java	0.0 %	0	25	25
> Main	0.0 %	0	25	25
▼ GraphPoet.java	95.1 %	274	14	288
▼ GraphPoet	95.1 %	274	14	288
● checkRep()	72.7 %	32	12	44
● GraphPoet(File)	100.0 %	104	0	104
● poem(String)	100.0 %	128	0	128
● toString()	100.0 %	4	0	4

3.1.5.2 Implement `GraphPoet`

1. 表示不变量

```
// Abstraction function:
//   GraphPoet represents a word affinity graph which is generated
//   with a corpus
// and vertices of is are case-insensitive words
```

```

    // and edge weights of it are in-order adjacency counts.
    // Representation invariant:
    // graph != null , vertices of the graph not empty case-insensitive
strings
    // Safety from rep exposure:
    // All fields are modified by private and final so that
    // clients can't access the graph reference outside the class

```

2. 检查不变量

```

public void checkRep() {
    assert graph != null ;
    for(String vertex : graph.vertices()) {
        assert !vertex.equals(""); //判断顶点是非为空内容
        assert vertex.equals(vertex.toLowerCase()); //判断顶点是否都转换为小写
    }
}

```

3. GraphPoet 方法

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

4. poem 方法

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

中相邻单词的桥接单词：一个单词调用 `targets` 方法，另一个单词调用 `sources` 方法，进行比对是否含有相同的单词；若两个单词含有多个桥接词，则进行选择一个权值较大者。同时选择 `StringBuilder` 将输出的所有单词拼接在一起（加上空格）

5. toString 方法

直接调用 `graph` 的 `toString` 方法

3.1.5.3 Graph poetry slam

测试策略

```

// Testing strategy
//
// Partition for the constructor of GraphPoet
// corpus : empty file , file with one line , file with several
lines

```

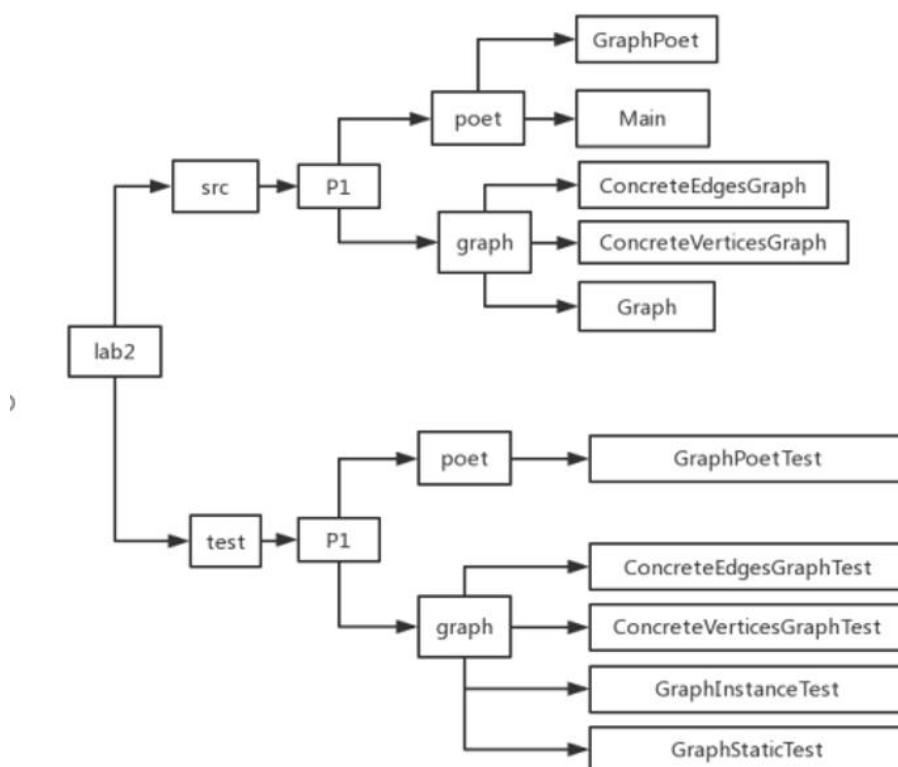
```
//Partition for GraphPoet.poem
//      input : empty string , one word , more than one word
//
//Partition for GraphPoet.toString
//      input : empty graph , one vertex ,more than one vertex
测试方法详见代码
```

3.1.6 Before you're done

请按照 http://web.mit.edu/6.031/www/sp17/psets/ps2/#before_youre_done 的说明, 检查你的程序。

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

在这里给出你的项目的目录结构树状示意图。



3.2 Re-implement the Social Network in Lab1

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

3.2.1 FriendshipGraph 类

方法

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

1. `public boolean addVertex(Person p)`

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

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

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

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

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

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

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

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

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

3.2.2 Person 类

1. 属性

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

2. 方法

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

3.2.3 客户端 main()

```

81 public static void main(String[] args) {
82     FriendshipGraph graph = new FriendshipGraph();
83     Person rachel = new Person("Rachel");
84     Person ross = new Person("Ross");
85     Person ben = new Person("Ben");
86     Person kramer = new Person("Kramer");
87     graph.addVertex(rachel);
88     graph.addVertex(ross);
89     graph.addVertex(ben);
90     graph.addVertex(kramer);
91     graph.addEdge(rachel, ross);
92     graph.addEdge(ross, rachel);
93     graph.addEdge(ross, ben);
94     graph.addEdge(ben, ross);
95     System.out.println(graph.getDistance(rachel, ross));
96     System.out.println(graph.getDistance(rachel, ben));
97     System.out.println(graph.getDistance(rachel, rachel));
98     System.out.println(graph.getDistance(rachel, kramer));
99 }
100 }

```

Coverage Console

<terminated> FriendshipGraph (2) [Java Application] D:\jdk\bin\javaw.exe (2020年3月27日 下午8:43)

```

1
2
0
-1

```

3.2.4 测试用例

测试策略:

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

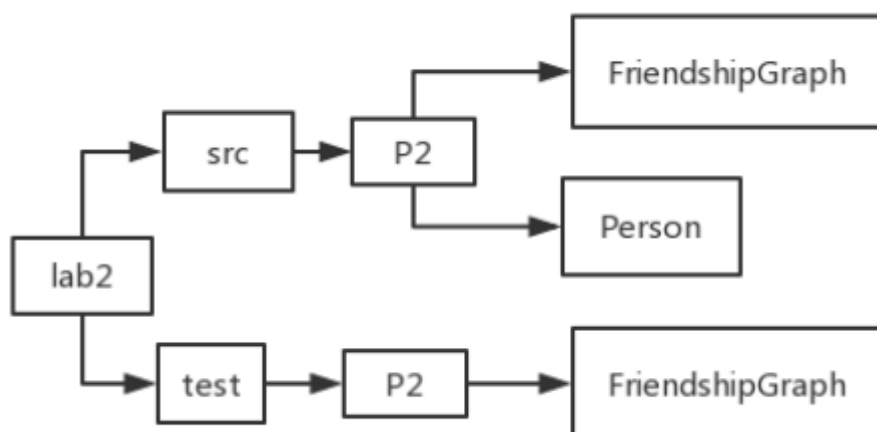

```
graph.addEdge(person1, person2);
graph.addEdge(person2, person3);
graph.addEdge(person1, person4);
graph.addEdge(person4, person5);
graph.addEdge(person5, person3);
assertTrue(graph.getDistance(person1, person2)==1);
assertTrue(graph.getDistance(person2, person3)==1);
assertTrue(graph.getDistance(person1, person3)==2); // 1-2-3 或者 1-4-5-3, 最短距离为 2
```

3.2.5 提交至 Git 仓库

如何通过 Git 提交当前版本到 GitHub 上你的 Lab3 仓库。

- ① `git add *`，将 Lab2 里面所有文件添加到缓冲区
- ② `git commit -m “第二次提交”`，将缓冲区的文件提交到本地仓库
- ③ `git push` 将文件推送到远程仓库

在这里给出你的项目的目录结构树状示意图。



3.3 Playing Chess

3.3.1 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/immaturity 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。

3.3.1.1 Position 类

1. 属性

<code>private int x = -1 ;</code>	横坐标
<code>private int y = -1 ;</code>	纵坐标

```
// Abstraction Function:
// AF(x,y) = Represent a position with abscissa of x and ordinate of y
//
//Representation invariant:
//   x and y are int
//
//Safety from rep exposure:
// the field of x and y is private
```

2. 方法

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

3.3.1.2 Player 类

1. 属性

<code>private String Name</code>	Player 的名字
----------------------------------	------------

```
// Abstraction Function:
// Represent a name of a Person that is not null
//
//Representation invariant:
//   Name != null
//
//Safety from rep exposure:
// name is immutable so there is no need for getName() to make
defensive copy
// the field of name is private
```

2.方法

<code>public Player(String Name)</code>	构造器方法，初始化
<code>public String getName()</code>	得到 player 的名字

3.3.1.3 Piece 类

1. 属性

<code>private Position position = new Position()</code>	棋子的位置
<code>private String PieceName = new String()</code>	棋子的名字
<code>private boolean Owner</code>	棋子的拥有者，true 为 player1

```
// Abstraction Function:
// AF(position,PieceName,Owner) = Represent the position of the piece,
the name of the piece and the owner of the piece
//Representation invariant:
// PieceName != null , position!=null
//
//Safety from rep exposure:
// PieceName is immutable so there is no need for getName() to make
defensive copy
// All field is private
```

2. 方法

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

3.3.1.4 Board 类

1. 属性

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

```

// Abstraction Function:
// AF(goBoard1,goBoard2)= goBoard1 and goBoard2 represent the Board of
go , goBoard1[i][j] represent whether the position of (i,j) is occupied by
go piece,goBoard2[i][j] represent whose piece occupy the position of (i,j)
// AF(chessBoard1,chessBoard2)= chessBoard1 and chessBoard2 represent
the Board of chess , chessBoard1[i][j] represent whether the position of
(i,j) is occupied by chess piece,chessBoard2[i][j] represent whose piece
occupy the position of (i,j)
//Representation invariant:
// goBoard1
//Safety from rep exposure:
// All field is private

```

2. 方法

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

3.3.1.5 Action 类

1. 属性

无

2. 方法

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

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

象棋移动棋子失败情况:

```
//          原位置和目标位置相同、目标位置被棋子占用、
//          当前的棋子不是被己方占用、当前需要移动的棋子的位置无棋子、
//          目标位置超出棋盘范围、当前棋子的位置超出棋盘范围
```

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

象棋吃子失败情况:

```
//          原位置和目标位置相同、目标位置的棋子不是敌方的
//          目标位置无棋子、当前的棋子不是被己方占用
//          当前需要移动的棋子的位置无棋子、目标位置超出棋盘范围
//          当前棋子的位置超出棋盘范围
```

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

围棋提子失败情况:

```
//          待提的子不是对方的棋子 、 待提子的位置无棋子、
//          待提子的位置超过了棋盘范围
```

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

围棋落子失败情况:

```
//          待落的子不是己方的棋子、待落子的位置已经有棋子
//          待落子的位置超过了棋盘范围
```

3.3.1.6 Game 类

1. 属性

<code>private String player1 = new String()</code>	玩家 1
<code>private String player2 = new String()</code>	玩家 2
<code>private Board chessBoard = new Board("chess")</code>	象棋棋盘
<code>private Board goBoard = new Board("go")</code>	围棋棋盘

<code>private String history1 = new String();</code>	玩家 1 的下棋历史
<code>private String history2 = new String();</code>	玩家 2 的下棋历史
<code>private List<Piece> pieces1 = new ArrayList<>()</code>	玩家 1 的所有棋子
<code>private List<Piece> pieces2 = new ArrayList<>()</code>	玩家 2 的所有棋子

2.方法

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

```
// Abstraction Function:
// Represent two players of the game ,and the type of the game ,
// the operation history of two players,and the pieces of two players
//Representation invariant:
// player1 != null ,player2 != null
//Safety from rep exposure:
// player1,player2,history1,history2 are immutable
// All field is private
```

3.3.1.7 MyChessAndGoGame 类

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

3.3.2 主程序 MyChessAndGoGame 设计/实现方案

1. **控制**：运用 `switch` 语句，根据用户输入的数字来判断进行哪个操作。
2. **玩家交替进行**：每次操作都需要确定是哪个棋手操作，变量 `player` 为奇数表示当前是 `player2` 操作，为偶数表示 `player1` 进行操作。`Player3` 表示目前操作的 `player` 的名字。
3. **player 变量**：当操作是“吃子”、“移子”、“提子”、“落子”、“跳过”时，则 `player++`，操作权转变。而当操作是“查询棋盘”、“查询棋子”、“查询棋子数目”、“历史”时，操作权不改变，`player` 变量不会改变。
4. **围棋棋子**：围棋棋手 1 的棋子为“white”，围棋棋手 2 的棋子为“black”。

具体 main 方法流程

- ① 首先，提示用户输入游戏类型->“chess” or “go”

```
请选择游戏类型 : chess or go
chess
```

- ② 再提示用户输入玩家的姓名

```
请选择游戏类型 : chess or go
```

```
chess
```

```
请输入player1的名字 :
```

```
a
```

```
请输入player2的名字 :
```

```
b
```

```
现在是a进行操作
```

- ③ 开始游戏，根据输入提示选择操作

操作的具体实现见代码

象棋操作

根据提示输入数字来选择操作

```
1.移子 2.吃子 3.历史 4.棋子数量 5.显示棋盘 6.查询棋子 7.跳过 8.结束
```

1.移子

给出需要移动的棋子的横坐标和纵坐标，以及目标位置的横坐标和纵坐标。

现在是a进行操作

根据提示输入数字来选择操作

```
1.移子 2.吃子 3.历史 4.棋子数量 5.显示棋盘 6.查询棋子 7.跳过 8.结束
```

```
1
请输入需要移动的子的初始横纵坐标和目标横纵坐标:
```

```
0 0 3 3
```

```
player1:a
```

```
player2:b
```

	0	1	2	3	4	5	6
0		1: 马	2: 象	3: 王	4: 后	5: 象	6: 马
1	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵
2							
3				1: 车			
4							
5							
6	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵
7	2: 车	2: 马	2: 象	2: 王	2: 后	2: 象	2: 马

现在是b进行操作

根据提示输入数字来选择操作

```
1.移子 2.吃子 3.历史 4.棋子数量 5.显示棋盘 6.查询棋子 7.跳过 8.结束
```

如果位置不合理则需重新进行操作，操作权不变

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

1
请输入需要移动的子的初始横纵坐标和目标横纵坐标:
6 0 6 1

error: 目标位置被棋子占用 操作错误

player1:a player2:b

0		1: 马	2: 象	3: 王	4: 后	5: 象	6: 马
1	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵
2							
3				1: 车			
4							
5							
6	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵
7	2: 车	2: 马	2: 象	2: 王	2: 后	2: 象	2: 马

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

2.吃子

给出需要吃子的棋子的横坐标和纵坐标，以及被吃的子的横坐标和纵坐标。

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

2
请输入吃子操作需要的初始横纵坐标和目标横纵坐标:
0 0 6 0

player1:a player2:b

0		1: 马	2: 象	3: 王	4: 后	5: 象	6: 马	7: 车
1	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵
2								
3								
4								
5								
6	1: 车	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵
7	2: 车	2: 马	2: 象	2: 王	2: 后	2: 象	2: 马	2: 车

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

如果位置不合理则需重新进行操作，操作权不变

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

2
请输入吃子操作需要的初始横纵坐标和目标横纵坐标:
0 1 6 0

error: 当前的棋子不是被己方占用 操作错误

player1:a player2:b

0		1: 马	2: 象	3: 王	4: 后	5: 象	6: 马	7: 车
1	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵	1: 兵
2								
3				1: 车				
4								
5								
6	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵	2: 兵
7	2: 车	2: 马	2: 象	2: 王	2: 后	2: 象	2: 马	2: 车

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

3.历史

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

3
请输入需要查看的选手, 选择1(代表player1)或2(代表player2):
1

The history of a : 吃子:(0,0)->(6,0)

现在是b进行操作
根据提示输入数字来选择操作

1. 移子 2. 吃子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束

4.棋子数量

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
4
请输入需要查看的选手, 选择1或2:
1
16
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
4
请输入需要查看的选手, 选择1或2:
2
15
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
```

查询操作不改变操作权

5.象棋棋盘

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
5
player1:a      player2:b
0              0
1              1: 马      2: 象      3: 王      4: 后      5: 象      6: 马      7: 车
1              1: 兵      1: 兵      1: 兵      1: 兵      1: 兵      1: 兵      1: 兵
2
3
4
5
6              1: 车      2: 兵      2: 兵      2: 兵      2: 兵      2: 兵      2: 兵
7              2: 车      2: 马      2: 象      2: 王      2: 后      2: 象      2: 马
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
```

6.查询棋子

输入需要查询的棋子的横纵坐标即可， 查询棋子操作不改变操作权

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7. 跳过  8. 结束
-----
6
请输入查询的坐标x和y
6 3
棋子的player:b
棋子的name : 兵
棋子的position: (6,3)
现在是b进行操作
```

7.跳过

```

现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7 跳过  8. 结束
-----
7
已经跳过该选手
player1:a      player2:b
0              0
1              1: 马      2      3      4      5      6      7
2              1: 兵      1: 兵      1: 象      1: 王      1: 后      1: 象      1: 马      1: 车
3
4
5
6              1: 车      2: 兵      2: 兵      2: 兵      2: 兵      2: 兵      2: 兵      2: 兵
7              2: 车      2: 马      2: 象      2: 王      2: 后      2: 象      2: 马      2: 车
现在是在a进行操作
根据提示输入数字来选择操作
-----
1. 移子  2. 吃子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7 跳过  8. 结束
-----

```

跳过操作，直接改变操作权即可

围棋操作

```

根据提示输入数字来选择操作
-----
1. 落子  2. 提子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7 跳过  8. 结束
-----

```

1. 落子

输入需要落的子的横纵坐标即可。位置不合理需要重新输入

```

现在是在a进行操作
根据提示输入数字来选择操作
-----
1. 落子  2. 提子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7 跳过  8. 结束
-----
1
请输入需要落子的横坐标和纵坐标:
0 0
player1:a      player2:b      3      4      5      6      7      8      9      10      11      12      13      14      15      16      17      18
0      white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子  2. 提子  3. 历史  4. 棋子数量  5. 显示棋盘  6. 查询棋子  7 跳过  8. 结束
-----

```

```

现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----
1
请输入需要落子的横坐标和纵坐标:
0 0
error : 待落子的位置已经有棋子
player1:a      player2:b
0      0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     15     16     17     18
0      white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----

```

操作错误

不改变操作权

2.提子

输入需要提子的横纵坐标即可，位置不合理需要重新输入。

```

现在是在a进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----
2
请输入需要提子的横坐标和纵坐标:
0 1
player1:a      player2:b
0      0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     15     16     17     18
0      white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----
10
现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----
2
请输入需要提子的横坐标和纵坐标:
0 1
error : 待提子的位置无棋子
player1:a      player2:b
0      0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     15     16     17     18
0      white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是在b进行操作
根据提示输入数字来选择操作
-----
1. 落子 2. 提子 3. 历史 4. 棋子数量 5. 显示棋盘 6. 查询棋子 7. 跳过 8. 结束
-----

```

(0,1)位置的子已经被提

操作错误

操作权不变，需要重新进行操作

3.历史

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
3
请输入需要查看的选手,选择1(代表player1)或2(代表player2):
1
The history of a : 落子: (0,0)提子:0,1)
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
3
请输入需要查看的选手,选择1(代表player1)或2(代表player2):
2
The history of a : 落子: (0,1)
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
```

4.棋子数量

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
4
请输入需要查看的选手,选择1或2:
1
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
4
请输入需要查看的选手,选择1或2:
2
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
```

棋子数量

5.围棋棋盘

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
5
player1:a      player2:b
0      1      2      3      4      5      6      7      8      9      10     11     12     13     14     15     16     17     18
0      white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7跳过  8.结束
-----
```

不改变操作权

6.查询棋子

输入需要查询的棋子的横纵坐标即可。

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7.跳过  8.结束
-----
6
请输入查询的坐标x和y
0 0
棋子的player:a
棋子的name : white
棋子的position: (0,0)
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7.跳过  8.结束
-----
```

棋子信息

7.跳过

```
现在是b进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7.跳过  8.结束
-----
7
已经跳过该选手
player1:a      player2:b
0             1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
0             white
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
现在是a进行操作
根据提示输入数字来选择操作
-----
1.落子  2.提子  3.历史  4.棋子数量  5.显示棋盘  6.查询棋子  7.跳过  8.结束
-----
```

直接改变操作权即可

3.3.3 ADT 和主程序的测试方案

测试覆盖率

>	Game.java	83.0 %	848	174	1,022
>	Action.java	100.0 %	238	0	238
>	Board.java	100.0 %	269	0	269
>	Piece.java	100.0 %	47	0	47
>	Player.java	100.0 %	9	0	9
>	Position.java	100.0 %	92	0	92

Runs: 51/51 Errors: 0 Failures: 0

> P3.BoardTest [Runner: JUnit 4] (0.000 s)

> P3.GameTest [Runner: JUnit 4] (0.009 s)

> P3.PlayerTest [Runner: JUnit 4] (0.002 s)

> P3.ActionTest [Runner: JUnit 4] (0.005 s)

> P3.PositionTest [Runner: JUnit 4] (0.001 s)

3.3.3.1 test Board 类

testing strategy:

```

//Testing strategy
//Partition for Board.setGoPiece()
// input : go piece
//
//Partition for Board.setChessPiece()
// input : chess piece

//Partition for RemoveGoPiece()
// input : go piece
//
// Partition for RemoveChessPiece()
// input : chess piece
//
//Partition for getZhanYong()
// input : Position existed with piece , Position existed without
piece
//          Board : go board , chess board
//
//Partition for getWhoZhanYong()
//          Position existed with player1 , Position existed with
player2
//          Board : go board , chess board

```

Runs: 6/6 ✖ Errors: 0 ❏ Failures: 0



> P3.BoardTest [Runner: JUnit 4] (0.000 s)

BoardTest.java	100.0 %	438	0	438
BoardTest	100.0 %	438	0	438
testChessgetWhoZha	100.0 %	160	0	160
testGogetWhoZhanY	100.0 %	76	0	76
testRemoveChessPie	100.0 %	57	0	57
testRemoveGoPiece	100.0 %	60	0	60
testSetChessPiece()	100.0 %	41	0	41
testSetGoPiece()	100.0 %	41	0	41

3.3.3.2 test Action 类

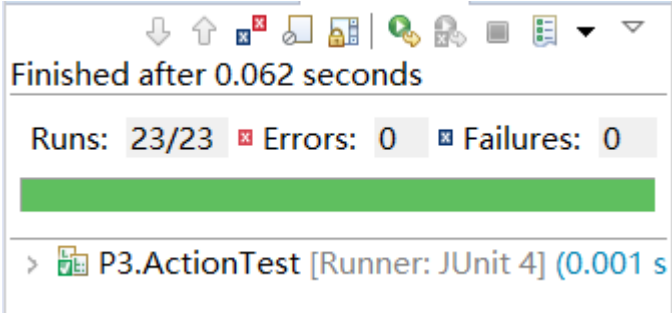
testing strategy:

```

//Testing strategy
//partition for checkMove()
// input : 原位置和目标位置相同、目标位置被棋子占用、
//          当前的棋子不是被己方占用、当前需要移动的棋子的位置无棋子、

```

```
//          目标位置超出棋盘范围、当前棋子的位置超出棋盘范围
//          目标位置和当前位置合理
//
//partition for checkeatPiece()
// input : 原位置和目标位置相同、目标位置的棋子不是敌方的
//          目标位置无棋子、当前的棋子不是被己方占用
//          当前需要移动的棋子的位置无棋子、目标位置超出棋盘范围
//          当前棋子的位置超出棋盘范围、 目标位置和当前位置合理
//
//partition for checkTiPiece()
// input : 待提的子不是对方的棋子 、 待提子的位置无棋子、
//          待提子的位置超过了棋盘范围、待提子的位置合理
//
//partition for checkLuoPiece()
// input : 待落的子不是己方的棋子、待落子的位置已经有棋子
//          待落子的位置超过了棋盘范围、待落子的位置合理
//
//
```



▼	Action.java	100.0 %	238	0	238
▼	➡ Action	100.0 %	238	0	238
	● checkeatPiece(int, int)	100.0 %	89	0	89
	● checkLuoPiece(int, in	100.0 %	29	0	29
	● checkMove(int, int, ir	100.0 %	77	0	77
	● checkTiPiece(int, int,	100.0 %	40	0	40

3.3.3.3 test Game 类

testing strategy:

```
// Testing strategy
// partition for LuoPieceGo()
// input : (x,y)位置合理, (x,y)位置不合理 ,
//          player == true , player == false
//
// partition for TiPieceGo()
```

```

// input : (x,y)位置合理, (x,y)位置不合理 ,
//         player == true , player == false
//
// partition for movePiece()
// input:    (x1,y1),(x2,y2)位置合理, (x1,y1),(x2,y2)位置不合理
//         player == true , ;player == false ;
//
// partition for eatPiece()
// input:    (x1,y1),(x2,y2)位置合理, (x1,y1),(x2,y2)位置不合理
//         player == true , ;player == false ;
//
// partition for getHistory()
// input :   LuoPieceGo,movePiece,eatPiece,TiPiece,showBoard
//         player : player1 , player2
//
// partition for getPiece()
// input :   position with piece existed , position without piece
existed
//         board : goBoard , chessBoard
//
// partition for addHistory()
// input :   null string , not null string
//         player: true ,false

```

Runs: 14/14 Errors: 0 Failures: 0

> P3.GameTest [Runner: JUnit 4] (0.000 s)

Game.java	83.0 %	848	174	1,022
Game	83.0 %	848	174	1,022
showBoard(String)	0.0 %	0	171	171
TiPieceGo(int, int, bo	98.0 %	145	3	148
Game(String)	100.0 %	254	0	254
addHistory(String, bc	100.0 %	30	0	30
eatPiece(int, int, int, i	100.0 %	135	0	135
getHistory(boolean)	100.0 %	8	0	8
getPiece(Position)	100.0 %	54	0	54
getPlayer1()	100.0 %	3	0	3
getPlayer2()	100.0 %	3	0	3
getSize(boolean)	100.0 %	10	0	10
LuoPieceGo(int, int, t	100.0 %	92	0	92
movePiece(int, int, in	100.0 %	106	0	106
setPlayer1(String)	100.0 %	4	0	4
setPlayer2(String)	100.0 %	4	0	4

3.3.3.4 test Player 类

testing strategy:

```
//Testing strategy
//partition for Player.getName
// input : name
```

Player.java	100.0 %	9	0	9
Player	100.0 %	9	0	9
Player(String)	100.0 %	6	0	6
getName()	100.0 %	3	0	3

Runs: 1/1

Errors: 0

Failures: 0

> P3.PlayerTest [Runner: JUnit 4] (0.000 s)


3.3.3.5 test Position 类



testing strategy:

```
// Testing strategy
//
//partition for EqualPosition()
// input : equal position , not equal position
//
//partition for FanWei()
// input: board: go ,chess
//          (x,y)超出范围, (x,y)未超出范围
//partition for setPosition()
// input : (x,y)超出范围, (x,y)未超出范围
//
//
//partition for getX()
// input : (x,y)
//
//partition for getY()
// input : (x,y)
//
//partition for toString()
// input :(x,y)
```

Finished after 0.021 seconds

Runs: 6/6 ✖ Errors: 0 ✖ Failures: 0

>  P3.PositionTest [Runner: JUnit 4] (0.001)

▼  Position.java	100.0 %	92	0	92
▼  Position	100.0 %	92	0	92
● EqualPosition(Position)	100.0 %	14	0	14
● FanWei(String)	100.0 %	40	0	40
● getX()	100.0 %	3	0	3
● getY()	100.0 %	3	0	3
● setPosition(int, int)	100.0 %	7	0	7
● toString()	100.0 %	16	0	16

4 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

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

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

5 实验过程中遇到的困难与解决途径

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

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验教训

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

6.2 针对以下方面的感受

- (1) 面向 ADT 的编程和直接面向应用场景编程，你体会到二者有何差异？
面向 ADT 编程以对象为主体，当对象设计完成之后，应用实现起来更容易。
- (2) 使用泛型和不使用泛型的编程，对你来说有何差异？
具体实现的时候差异不大，但是应用的时候泛型设计的应用面明显要广太多
- (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？
优势：知道规约之后，就可以知道出现错误的情况，这样就会根据规约内容编写测试类可以减少很多的时间去构思，同时也利于写出正确的程序。能适应。
- (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？
应用面更广泛，可以满足更多的需求。
- (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？
任务量特别大，需要构思的地方特别多。而且不同的类之间关联特别大，需要我

们编写更详细的测试类来保证更少的错误发生。

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

(7) 关于本实验的工作量、难度、deadline。
工作量很大, 难度很大, deadline 很充足。

(8) 《软件构造》课程进展到目前, 你对该课程有何体会和建议?
体会: 这门课真的很好, 学到了很多的东西。