

**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 | 周牧云 |
| 学号 | 1180300315 |
| 班号 | 1836101 |
| 电子邮件 | 1036314134@qq.com |
| 手机号码 | 13912263240 |

**目录**

[1 实验目标概述 1](#_Toc37608905)

[2 实验环境配置 1](#_Toc37608906)

[3 实验过程 2](#_Toc37608907)

[3.1 Poetic Walks 2](#_Toc37608908)

[3.1.1 Get the code and prepare Git repository 3](#_Toc37608909)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc37608910)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc37608911)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc37608912)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc37608913)

[3.1.4 Problem 3: Implement generic Graph<L> 6](#_Toc37608914)

[3.1.4.1 Make the implementations generic 6](#_Toc37608915)

[Implement Graph.empty() 6](#_Toc37608916)

[3.1.5 Problem 4: Poetic walks 6](#_Toc37608917)

[3.1.5.1 Test GraphPoet 6](#_Toc37608918)

[3.1.5.2 Implement GraphPoet 7](#_Toc37608919)

[3.1.5.3 Graph poetry slam 7](#_Toc37608920)

[3.1.6 Before you’re done 7](#_Toc37608921)

[3.2 Re-implement the Social Network in Lab1 8](#_Toc37608922)

[3.2.1 FriendshipGraph类 8](#_Toc37608923)

[3.2.2 Person类 9](#_Toc37608924)

[3.2.3 客户端main() 9](#_Toc37608925)

[3.2.4 测试用例 10](#_Toc37608926)

[3.2.5 提交至Git仓库 11](#_Toc37608927)

[3.3 Playing Chess 12](#_Toc37608928)

[3.3.1 ADT设计/实现方案 12](#_Toc37608929)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 14](#_Toc37608930)

[3.3.3 ADT和主程序的测试方案 17](#_Toc37608931)

[4 实验进度记录 19](#_Toc37608932)

[5 实验过程中遇到的困难与解决途径 19](#_Toc37608933)

[6 实验过程中收获的经验、教训、感想 20](#_Toc37608934)

[6.1 实验过程中收获的经验和教训 20](#_Toc37608935)

[6.2 针对以下方面的感受 20](#_Toc37608936)

# 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

安装EclEmma:

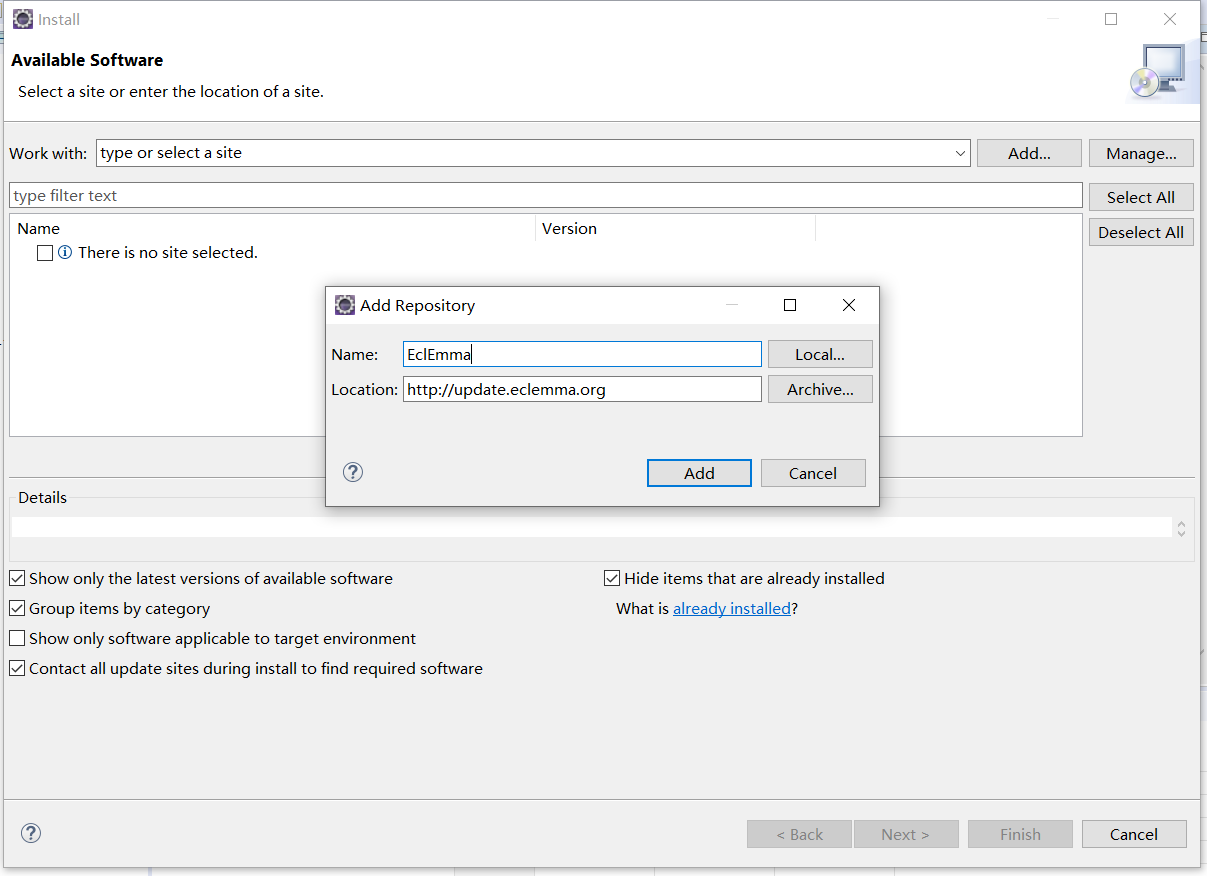
在eclipse中，选择help -> Install New Software -> add

在弹出的界面中填写：

Name: EclEmma

Location: <http://update.eclemma.org/>

即可成功安装。



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/Lab2-1180300315>

# 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

本问题有两个大的子问题：

1. 需要对题目给定的接口Graph<L>，完成他的两个实例类ConcretVerticesGraph和ConcreateEdgesGraph。

其中，ConcretVerticesGraph主要用Vertices类来实现，而ConcreateEdgesGraph主要用Edges来实现。

1. 对给定的文本文件，调用Graph<L>按照给定规则建立有向图，并利用该结构，对输入的字符串进行扩充，进行诗歌创作。

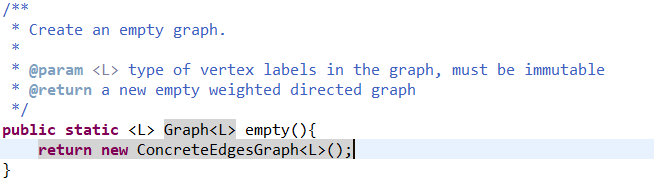
### Get the code and prepare Git repository

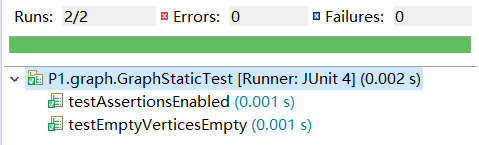
git clone <https://github.com/rainywang/Spring2020_HITCS_SC_Lab2.git>

### Problem 1: Test Graph <String>

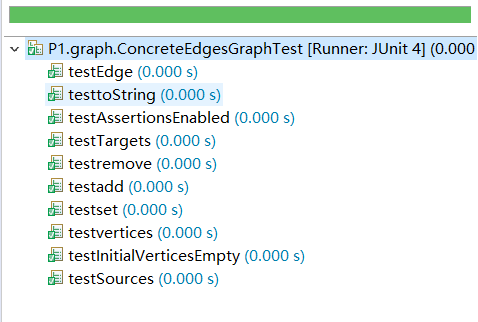
设计Junit测试

为了运行测试，首先需要更改Graph中的empty()

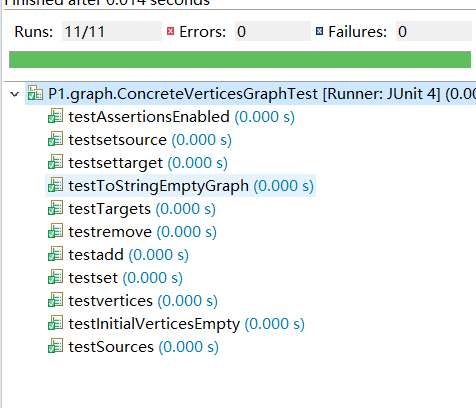




对Graph中的每个方法均设计测试









### 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

Edge类：

//fields

**private** **int** weight;//边的权重

**private** L source;//起点

**private** L target;//终点

//constructor

**public** Edge(L source, L target, **int** weight):增加一个edge

//methods

**public** **void** setweight(**int** weight):改变weight

**public** L source():返回source

**public** L target():返回target

**public** **int** weight():返回weight

//toString()

**public** String toString():返回字符串表示的edge，自动生成的就行

ConcreteEdgesGraph类:

//fields

**protected** **final** Set<L> vertices = **new** HashSet<>();//存储所有顶点

**private** **final** List<Edge<L>> edges = **new** ArrayList<>();//存储所有边

//methods

**public** **boolean** add(L vertex):增加一个顶点

**public** **int** findedge(L source, L target):找到边在list中的位置，返回一个index

**public** **int** set(L source, L target, **int** weight):增加一条有向边

**public** **boolean** remove(L vertex):移除一个顶点

**public** Set<L> vertices():返回包含所有顶点的集合

**public** Map<L, Integer> sources(L target):返回某个顶点为终点的所有边

**public** Map<L, Integer> targets(L source):返回某个顶点为起点的所有边

#### Implement ConcreteVerticesGraph

Vertex类：

//fields

**private** L name;//顶点的名字

**private** Map<L, Integer> allsource = **new** HashMap<>();//返回该顶点为起点的所有边

**private** Map<L, Integer> alltarget = **new** HashMap<>();//返回该顶点为终点的所有边

//constructor

**public** Vertex(L name):增加一个vertex

//methods

**public** L name():返回顶点名字

**public** Map<L, Integer> getsource():返回allsource

**public** Map<L, Integer> gettarget():返回alltarget

**public** **void** setsource(L source, **int** weight):在allsource中增加一个映射

**public** **void** settarget(L target, **int** weight):在alltarget中增加一个映射

//toString()

**public** String toString():自动生成

ConcreteVerticesGraph类:

//fields

**private** **final** List<Vertex<L>> vertices = **new** ArrayList<>();//储存所有顶点

//methods

**public** L name():返回顶点名字

**public** Map<L, Integer> getsource():返回allsource

**public** Map<L, Integer> gettarget():返回alltarget

**public** **void** setsource(L source, **int** weight):在allsource中增加一个映射

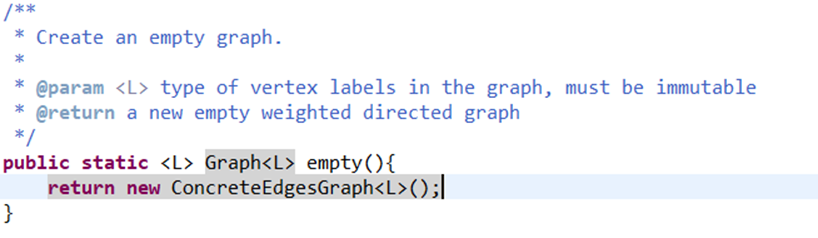
**public** **void** settarget(L target, **int** weight):在alltarget中增加一个映射

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

#### Make the implementations generic

将所有的String改为L就可以了

#### Implement Graph.empty()

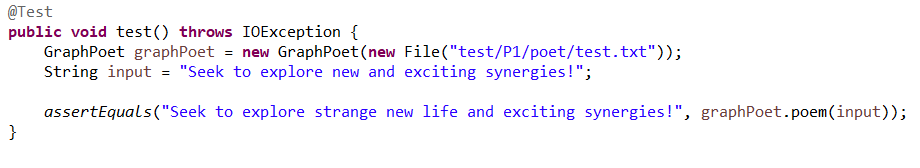


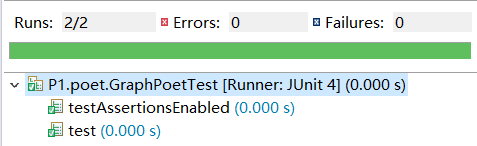
Edges和Vertexs选哪个无所谓

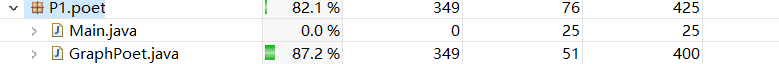
### Problem 4: Poetic walks

#### Test GraphPoet

直接使用题目给的来测试







#### Implement GraphPoet

//fields

**private** **final** Graph<String> graph = Graph.*empty*();

//methods

**public** GraphPoet(File corpus) **throws** IOException():从corpus中读取原文，处理后存在graph中。

**public** String poem(String input): **public** String poem(String input):基于graph将input改为诗歌并返回。

#### Graph poetry slam

更改一下原来给的路径就可以了



### Before you’re done

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

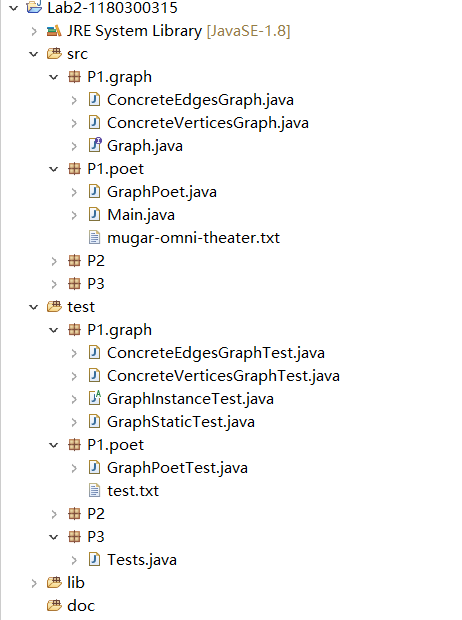
将文件放在本地仓库，然后输入指令

Git add .

Git commit

Git push

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

利用p1中完成的Graph来重新编写Lab1中的社交关系网。

感觉和之前差不多。。。只要把所有边的权重都设置为1就行。

再需要自己设计一个找长度的函数，和之前一样使用广度优先搜索就行。

### FriendshipGraph类

//fields

**private** **final** Graph<Person> graph = Graph.*empty*();

//methods

**public** **boolean** addVertex(Person person):增加一个人到关系网中

**public** **boolean** addEdge(Person source, Person target, **int** weight):在关系网中增加一个关系

**public** **int** getDistance(Person a, Person b):获取关系网中两个人的距离，使用BFS

### Person类

//fields

**private** String name;//人的名字

**public** **int** distance;//方便BFS而设置的距离

//constructor

**public** Person(String name):增加一个人

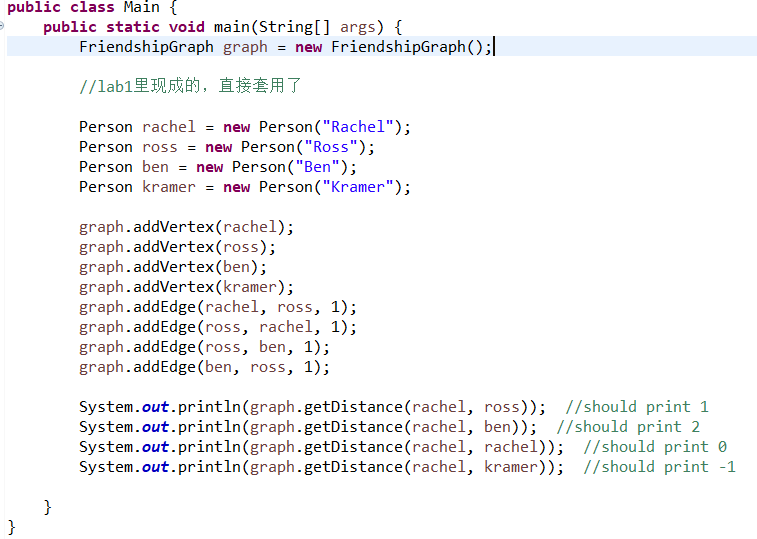
//methods

**public** String name():返回name

**public** **int** distance():返回distance

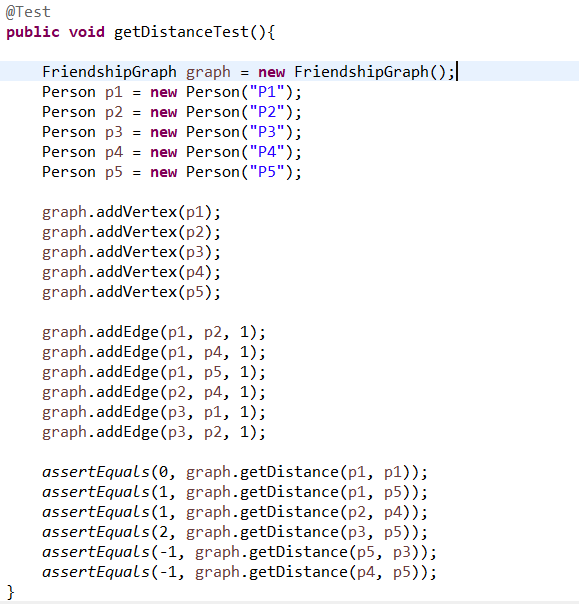
### 客户端main()

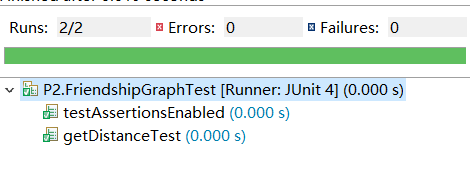
偷懒直接使用了之前lab1的main()了

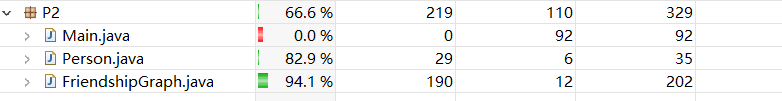


### 测试用例

也是之前设计的（反正都能用）

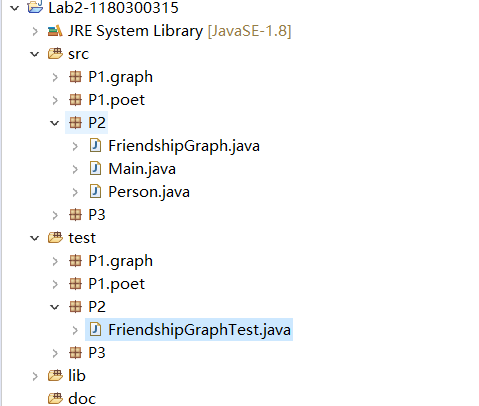






### 提交至Git仓库

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

Position类：

//fields

**private** **final** **int** x;//存横坐标

**private** **final** **int** y;//存纵坐标

//constructor

**public** Position(**int** x, **int** y):增加位置

//methods

**public** **int** x():返回横坐标

**public** **int** y():返回纵坐标

Player类：

//fields

**private** **final** String name; //储存玩家名字

//constructor

**public** Player(String name):增加一个玩家

//methods

**public** String name():返回玩家名字

Piece类：

//fields

**private** **final** String name;//储存棋子名字

//constructor

**public** Piece(String name):增加一个棋子

//methods

**public** String name():返回棋子名字

Board类：

//fields

**private** **final** **int** size;//储存棋盘的大小

**private** **final** Map<Position, Piece> board;//储存棋子与位置的对应关系

//constructor

**public** Board(**int** size):生成棋盘

//methods

**public** **int** size():返回size

**public** Map<Position, Piece> getboard():返回board

**public** **boolean** setPosition(Position position, Piece piece):在棋盘上增加棋子

**public** **boolean** removePosition(Position position):将棋子从棋盘上移除

**public** **boolean** isPositionAvailable(**int** x, **int** y):查看位置是否在棋盘上

Game类：

//fields

**private** **final** String type;//棋的种类

**private** **final** Board board;//棋盘

**private** **final** List<Piece> pieces;//已下棋子

**private** **final** List<Position> positions;//已下位置

**private** **final** Player player1, player2;//玩家

**private** **final** Map<Piece, Player> belong;//棋子与玩家的映射

//constructor

**public** Game(String type, String player1, String player2) **throws** IOException:生成一局游戏

//methods

**public** String type():返回type

**public** Board board():返回board

**public** List<Piece> pieces():返回pieces

**public** List<Position> positions():返回positions

**public** Map<Piece, Player> belong():返回belong

**public** Piece findPieces(String name):通过棋子名字从pieces中找到棋子

Position findPositions(**int** x, **int** y):通过坐标从positions中找到位置

**public** Piece queryPiece(**int** x, **int** y):通过坐标找到棋子

**private** Player findPlayer(String player):通过玩家名字找到玩家信息

**public** Player anotherPlayer(String player):通过玩家名字找到对手玩家信息

**public** **boolean** addPiece(String playername, String name, **int** x, **int** y):在棋盘上增加棋子

**public** **boolean** movePiece(String playername, **int** X1, **int** Y1, **int** X2, **int** Y2):在棋盘上移动棋子

**public** **boolean** removePiece(String playername, **int** x, **int** y): 在棋盘上删除棋子

**public** **boolean** setPiece(String playername, **int** X1, **int** Y1, **int** X2, **int** Y2): 在棋盘上吃子

**public** **int** queryallpiecesNumber():查询某个玩家拥有的棋子数

Action类：

//fields

**private** **final** Game game;

**private** **final** String player1;//玩家1

**private** **final** String player2;//玩家2

**private** **final** List<String> player1record;//玩家1历史步骤

**private** **final** List<String> player2record;//玩家2历史步骤

//constructor

**public** Action(String type, String play1, String play2) **throws** IOException :生成一个下棋动作

//methods

**public** Game game():返回这盘棋的局势

**public** **boolean** takeActionChess(String playername, **int** X1, **int** Y1, **int** X2, **int** Y2):执行一个国际象棋动作

**public** **boolean** takeActionGo(String playername, **int** X, **int** Y, **int** index):执行一个围棋动作

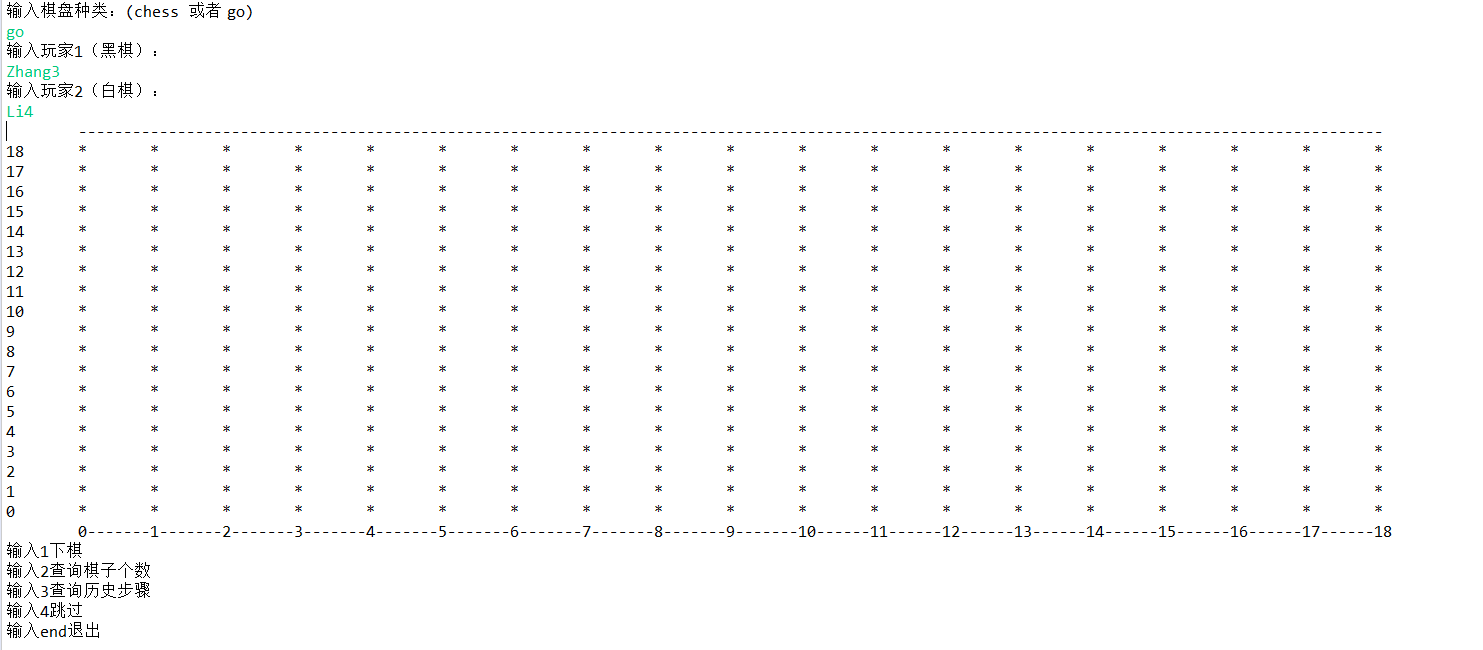
**public** List<String> queryRecord(String player):查询某个玩家历史步骤

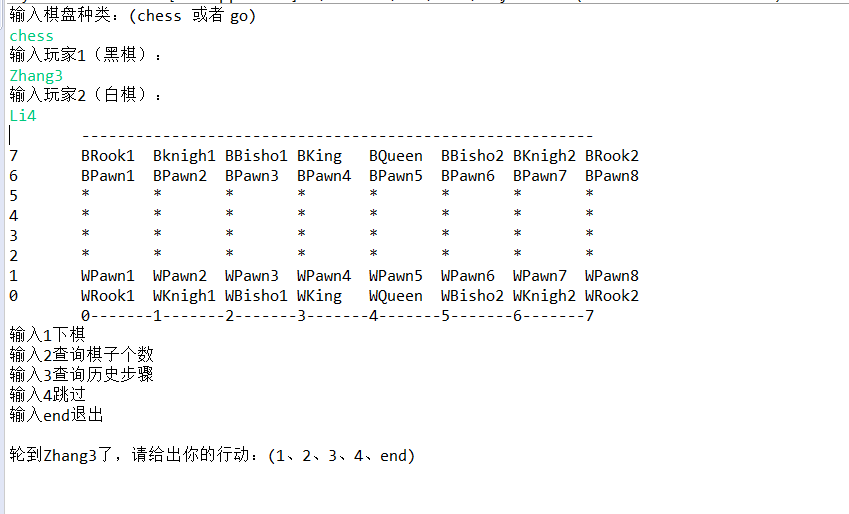
**public** **void** printBoard():打印棋盘

### 主程序MyChessAndGoGame设计/实现方案

首先你需要输入棋盘种类，然后输入玩家1和玩家2的名字

之后，程序会打印初始棋盘，并提示你如何继续操作

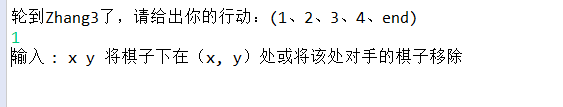


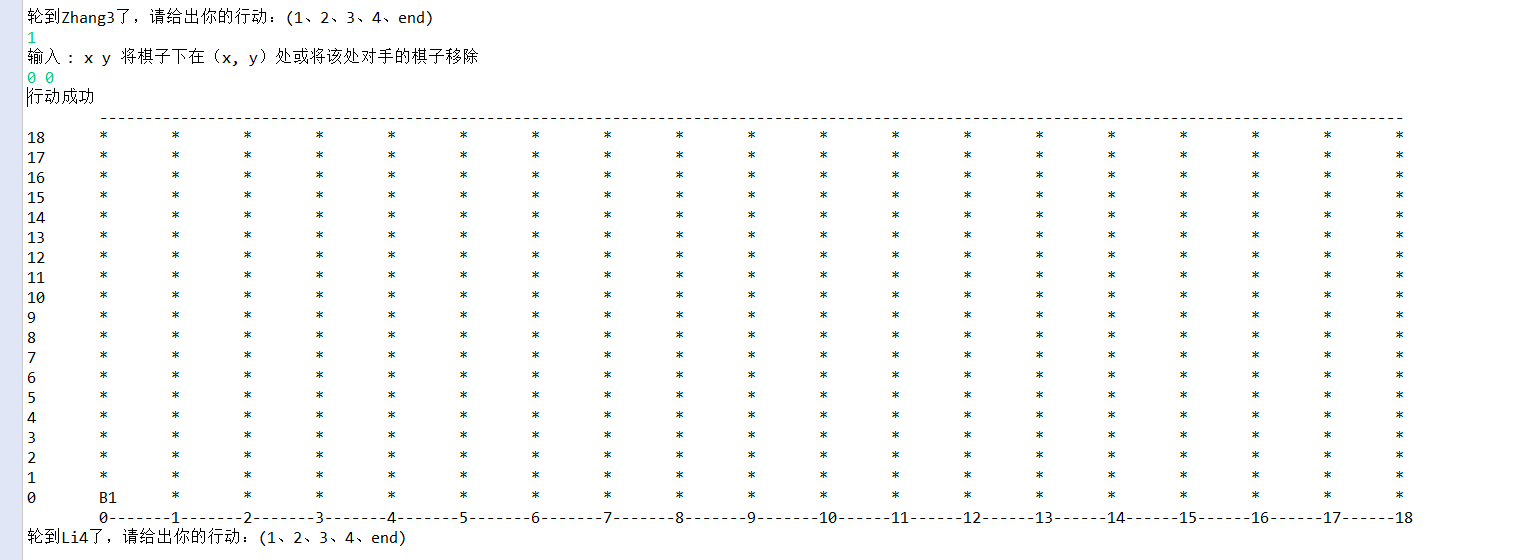


无论是哪种棋盘，都是由玩家1先手，并且玩家1拿黑棋子。

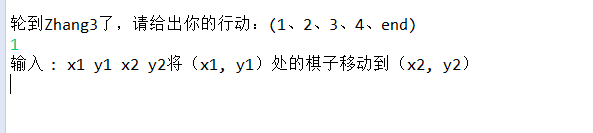
一、输入1之后会进行下棋操作

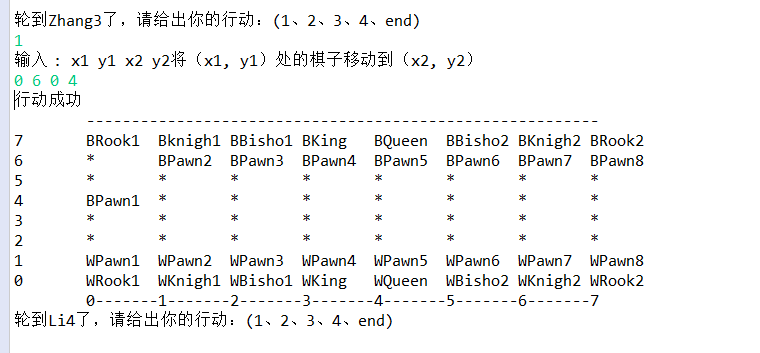
围棋中需要输入一个坐标，如果该坐标处没有棋子则会在该处下棋，并打印棋盘，如果该坐标处有对手棋子则可以将该棋子删除，其余情况会提示错误：





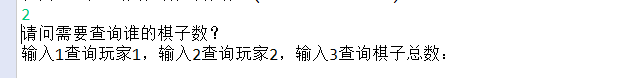
国际象棋中需要输入两个坐标，表示将第一个坐标内的己方棋子移动到第二个坐标处并打印棋盘，如果第二个坐标处有对手棋子则会进行吃子操作，其余情况会提示错误：



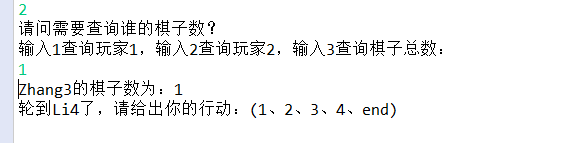


执行完一次操作后，会轮换到另一位玩家行动。

二、输入2之后可查看棋子数：

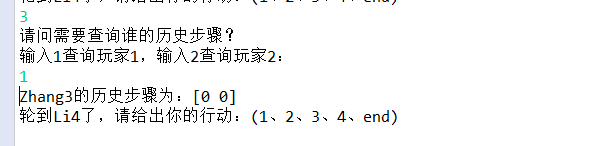


按照提示再次输入可查看相应的棋子数



执行次操作不会转换行动权，也就是说查看完之后还可继续执行其他操作

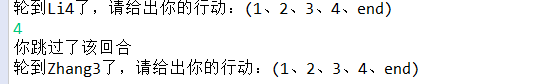
三、输入3可查看历史操作



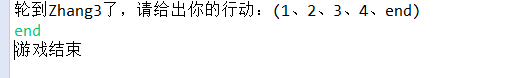
此操作可查看某个玩家所执行过的所有下棋行动，以坐标形式表现。

执行次操作同样不会转换行动权。

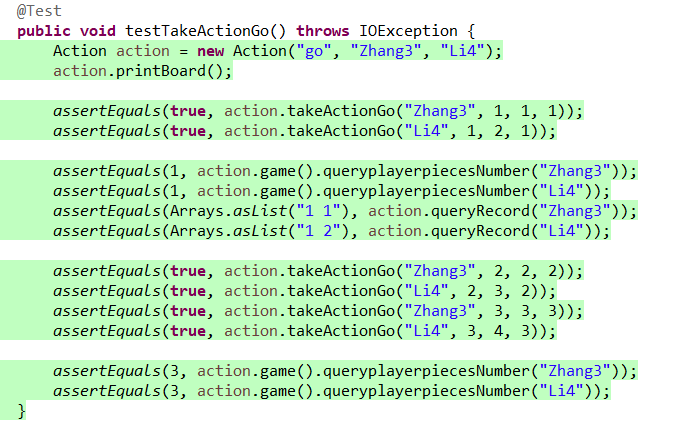
四、输入4即可跳过该回合

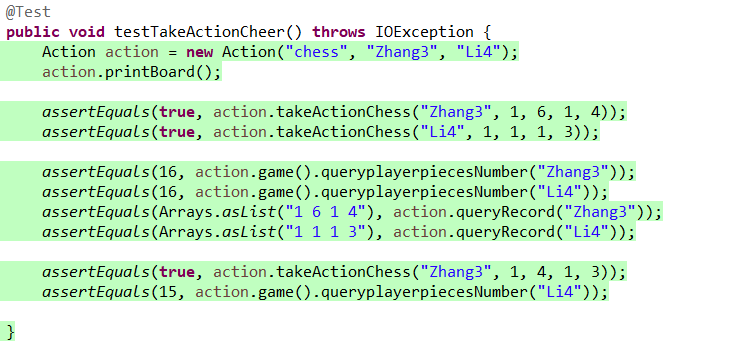


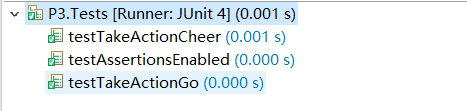
五、输入end可结束游戏

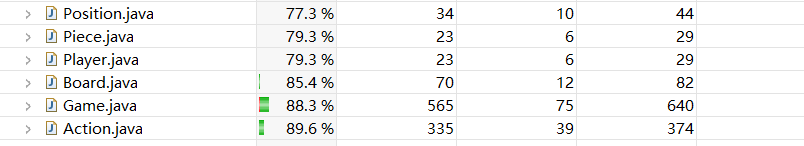


### ADT和主程序的测试方案









# 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 |
| 2020/3/16 | 18：30-20：15 | 完成GraphInstanceTest.java | 完成，后续有修改 |
| 2020/3/21 | 14：00-18：00 | 完成P1除GraphPoet.java外的所有内容 | 未完成， Vertices当中存在错误，暂时未解决 |
| 2020/3/23 | 15：00-18：00 | 完成P1 | 未完成，测试未通过 |
| 2020/3/24 | 18：30-20：15 | 完成P1剩余 | 完成 |
| 2020/3/31 | 18：30-20-15 | 完成P2 | 完成 |
| 2020/4/4 | 14：00-18：00 | 完成P3Test.java与Game.java | 未完成 |
| 2020/4/5 | 15：00-16：30 | 完成Game.java | 完成 |
| 2020/4/10 | 16：00-21：40 | 完成P3 | 未完成，主程序有部分功能出错 |
| 2020/4/11 | 8：00 -11：00 | 完成P3 | 未完成 |
| 2020/4/11 | 14：00-17：00 | 完成P3 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| P1的VertexGraph中set()方法无法正确实现 | 更改了设计思路，无论顶点存在与否都先加入进去，在add()函数中增加判断是否有重复，有重复则不执行 |
| P1的VertexGraph中remove ()方法无法正确实现 | 将List的remove()方法使用index删除，即可顺利删除了，  而之前使用object删除未成功 |
| P2中的getDistance函数出错，返回的全是-1 | 排查发现错误发生在add函数的遍历，需要调用vertices()方法，而不是直接从map中遍历key |
| P3的围棋功能无法实现 | 由于之前设计的方法中有通过棋子的名字来查找棋子，所以下围棋时也需要给每个棋子赋予不同的名字 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？

面向应用场景的编程实现起来更容易，但一但换个场景之后通用性很低，而面向ADT的编程实现起来相对复杂，但可以很方便地应用到多个应用场景，对于以后复用代码的工作量会减少很多

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

这次实验看起来差别不大。。。因为p1一开始不用泛型到后面改成泛型也不是很麻烦，说不定下次进行泛型编程的时候也可以先当作非泛型来编程之后再改成泛型？

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？

优势是更好查出错误，不然写完代码之后容易根据自己的代码设计测试用例，走进写代码时的思维圈子，跳脱不出来，设计出的测试用例找不出错误。

目前还不是很适应，说不定以后会很习惯。。。

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？

省时省力，确保了ADT的正确性之后就可以不关注内部实现，只关注具体实现效果就可以了，排查错误时也不用考虑ADT的问题

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？

较为适应了，不敢吹牛。

自主设计ADT的话自由程度很高，但也要花更多的时间去研究每个ADT之间的逻辑关系，可能自己的想法未必是最好的。照着现成的ADT框架去编写还是要容易很多的。。。

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？

更容易发现ADT内部实现的错误。

我比较懒，可能不愿意，但很明显必须得这么做。。

1. 关于本实验的工作量、难度、deadline。

感觉前两个实验都是做之前觉得不简单，等完成之后回顾发现好像并没有那么难的样子。我感觉我仍然需要更多的锻炼才能进步，毕竟起点太低了。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

暂时没有，只希望难度跨度不要太夸张