

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 林之浩 |
| 学号 | 1170300817 |
| 班号 | 1703008 |
| 电子邮件 | 630073498@qq.com |
| 手机号码 | 18065053516 |

**目录**

[1 实验目标概述 1](#_Toc5484910)

[2 实验环境配置 1](#_Toc5484911)

[3 实验过程 2](#_Toc5484912)

[3.1 Poetic Walks 2](#_Toc5484913)

[3.1.1 Get the code and prepare Git repository 2](#_Toc5484914)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc5484915)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc5484916)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc5484917)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc5484918)

[3.1.4 Problem 3: Implement generic Graph<L> 8](#_Toc5484919)

[3.1.4.1 Make the implementations generic 8](#_Toc5484920)

[3.1.4.2 Implement Graph.empty() 9](#_Toc5484921)

[3.1.5 Problem 4: Poetic walks 10](#_Toc5484922)

[3.1.5.1 Test GraphPoet 10](#_Toc5484923)

[3.1.5.2 Implement GraphPoet 11](#_Toc5484924)

[3.1.5.3 Graph poetry slam 12](#_Toc5484925)

[3.1.6 Before you’re done 13](#_Toc5484926)

[3.2 Re-implement the Social Network in Lab1 14](#_Toc5484927)

[3.2.1 FriendshipGraph类 14](#_Toc5484928)

[3.2.2 Person类 16](#_Toc5484929)

[3.2.3 客户端main() 16](#_Toc5484930)

[3.2.4 测试用例 16](#_Toc5484931)

[3.2.5 提交至Git仓库 17](#_Toc5484932)

[3.3 Playing Chess 18](#_Toc5484933)

[3.3.1 ADT设计/实现方案 18](#_Toc5484934)

[3.3.1.1 Board类 18](#_Toc5484935)

[3.3.1.2 Action类 20](#_Toc5484936)

[3.3.1.3 Game类 21](#_Toc5484937)

[3.3.1.4 Piece类 22](#_Toc5484938)

[3.3.1.5 Player类 22](#_Toc5484939)

[3.3.1.6 Position类 22](#_Toc5484940)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 23](#_Toc5484941)

[3.3.3 ADT和主程序的测试方案 26](#_Toc5484942)

[3.4 Multi-Startup Set (MIT) 27](#_Toc5484943)

[4 实验进度记录 27](#_Toc5484944)

[5 实验过程中遇到的困难与解决途径 28](#_Toc5484945)

[6 实验过程中收获的经验、教训、感想 28](#_Toc5484946)

[6.1 实验过程中收获的经验和教训 28](#_Toc5484947)

[6.2 针对以下方面的感受 28](#_Toc5484948)

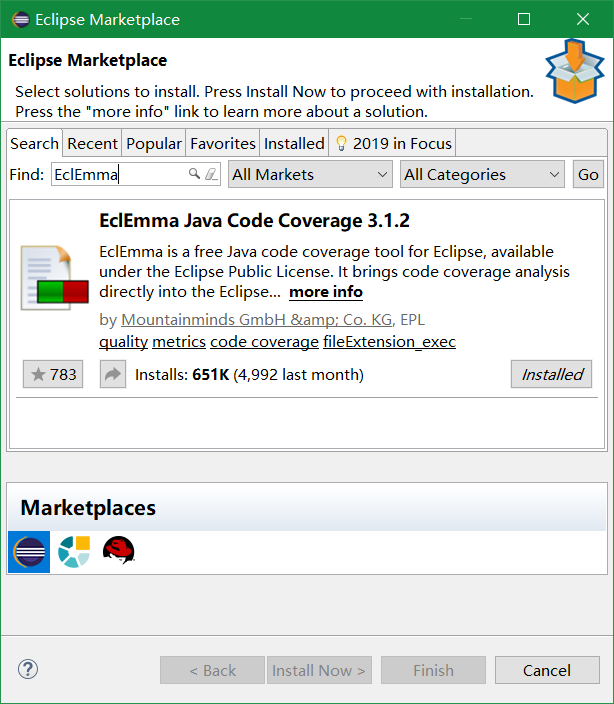
# 实验目标概述

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。

# 实验环境配置

进入EclEmma官网 <https://www.eclemma.org/>寻找安装方法，

发现可以使用eclipse自带的市场安装，打开搜索



找到EclEmma安装，简单学习下使用的技巧。

在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/Lab2-1170300817>

# 实验过程

## Poetic Walks

这个实验的主要目的是练习ADT的规约设计和ADT的不同实现。

### Get the code and prepare Git repository

 从<https://github.com/rainywang/Spring2019_HITCS_SC_Lab2/tree/master/P1> 下载得到实验代码，建立好project，进入目录，打开Git bush

依次输入

Git init

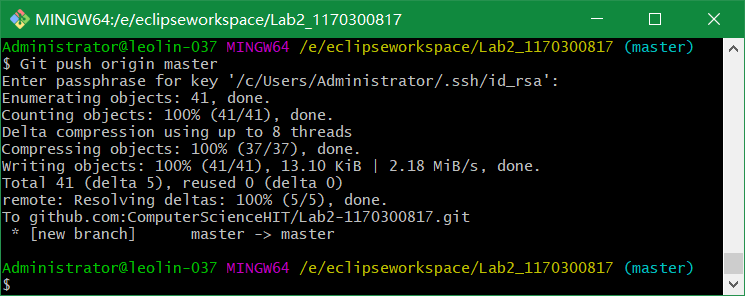
Git remote add origin git@github.com:ComputerScienceHIT/Lab2-1170300817.git

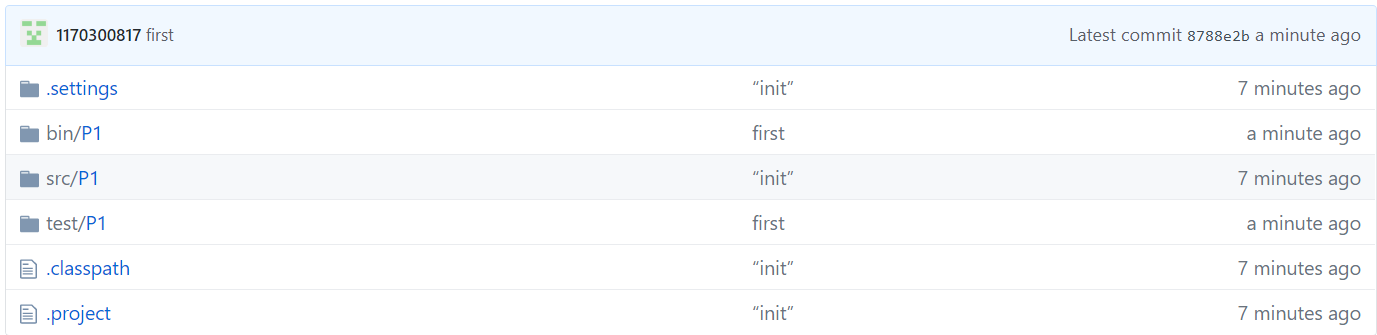
Git pull origin master

Git add .

Git commit -m “init”

Git push origin master





### Problem 1: Test Graph <String>

这部分主要是针对Graph<String>设计测试策略，编写测试用例主要利用等价类划分的思想进行测试，测试策略如下：



分别编写覆盖以上条件的测试用例。

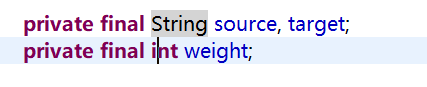
### Problem 2: Implement Graph <String>

分别用两种表示实现Graph。

#### Implement ConcreteEdgesGraph

###### Edge类实现

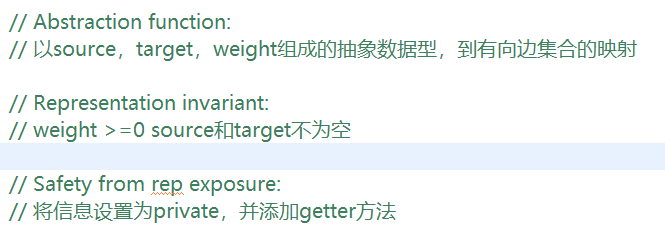
定义两个私有String类型source和target存放每个边的起止点，定义一个私有int类型weight保存这条边的权重（长度）



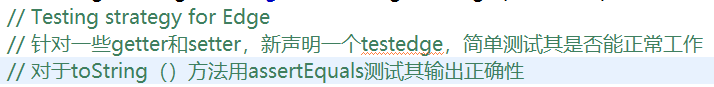
方法如下

|  |  |
| --- | --- |
| Edge | 构造方法，使用上述三个数据域声明一个新的边 |
| getSource | 返回source域 |
| getTarget | 返回target域 |
| getWeight | 返回weight域 |
| toString | 返回一个字符串表明这条边从哪个source到哪个target，weight是多少。 |
| checkRep | 检查表示不变量，其中source和target必须非空，weight必须大于0 |

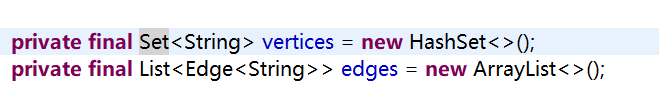
关于AF,RI和rep exposure：



关于测试策略：



###### ConcreteEdgesGraph实现

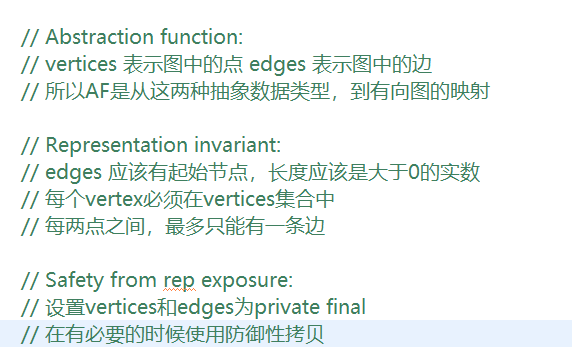


|  |  |
| --- | --- |
| vertices | 记录当前graph含有的点 |
| edges | 记录当前graph含有的边 |

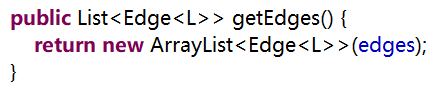
方法如下

|  |  |
| --- | --- |
| add | 如果顶点不为空，添加一个顶点 |
| getSource | 返回source域 |
| getTarget | 返回target域 |
| getWeight | 返回weight域 |
| toString | 对每条边调用toString方法，整合起来。 |
| checkRep | 检查表示不变量，其中source和target必须非空，weight必须大于0 |
| Set | 输入source，target，weight，确定一条有向边。  具体做法：如weight！=0，移去可能已经存在的相同起始点的边，然后加入新的边，如weight=0，寻找可能已经存在的相同起始点的边，删去。 |
| remove | 从vertices中删去，传入的参数vertex点，遍历edges，寻找是否有边的起点或者是终点是该vertex，删去。注意在使用迭代器遍历时要使用iterator.remove方法保证安全。 |
| vertices | 返回vertices集合 |
| sources | 参数：target。根据传入的target参数寻找以targe为终点的边。返回一个键值对为（点，权重）的map。实现：利用迭代器。 |
| targets | 参数：source。根据传入的source参数寻找以source为起点的边。实现同上。 |

关于AF,RI和rep exposure：



使用防御性拷贝的例子：





生成一个一模一样的复制对象返回，以保护原来的数据。

关于测试策略：



Test覆盖率：



#### Implement ConcreteVerticesGraph

##### Vertex类：

域：

**private** **final** String name;

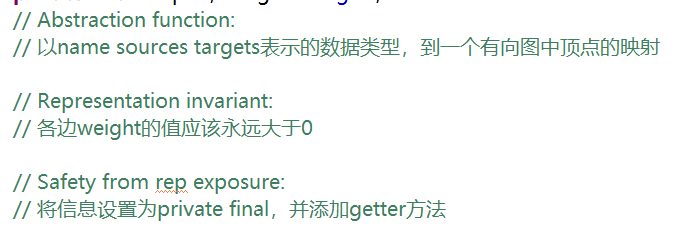
**private** **final** Map< String, Integer> sources;

**private** **final** Map< String, Integer> targets;

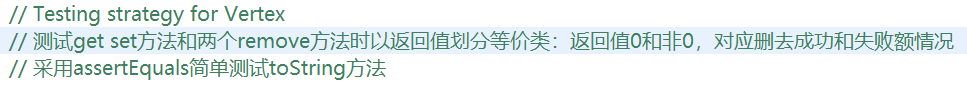
以下省略简单的getter和setter类不在赘述。

|  |  |
| --- | --- |
| Vertex | 构造方法，传入参数name创建新的点。 |
| checkRep | 检查表示不变性，各边weight的值应该永远大于0 |
| removeSource | 删去当前点的指定source |
| removeTarget | 删去当前点的指定target |
| toString | 输出能看懂的当前点信息 |
| setTarget | 为当前点新增一个target，  如果weight=0，删去当前点的target，成功返回删去target的weight，不存在返回0。如果weight！=0，为当前点新增一个target，长度为weight，如果该点已存在，返回旧的weight，否则返回0 |
| setSouce | 为当前点新增一个source，  如果weight=0，删去当前点的source，成功返回删去source的weight，不存在返回0。如果weight！=0，为当前点新增一个source，长度为weight，如果该点已存在，返回旧的weight，否则返回0  实现方法同上 |

关于AF,RI和rep exposure：



关于测试策略：



##### ConcreteVerticesGraph类

使用如下数据类型保存数据：

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

方法：

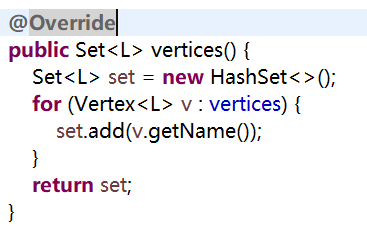
以下省略简单的getter和setter类的说明。

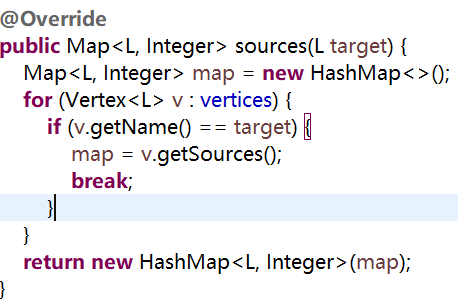
|  |  |
| --- | --- |
| add | 参数：vertex，判断vertices中无重复点就加入 |
| set | 参数：source, target, weight。先将可能不在vertices中的source点和target加入vertices。随后遍历vertices，找到source对它增加一个target，找到target为它增加一个source，并设置距离。 |
| remove | 参数：vertex。遍历vertices，如果当前点是vertex，删去（使用iterator.remove方法），如果不是，检查它的source和target是否包含vertex，如果有删去。 |
| sources | 参数：target。遍历vertices找到target点返回它的sources map（注意防御性拷贝） |
| targets | 参数：source。遍历vertices找到source点返回它的targets map（注意防御性拷贝） |
| toString | 输出一个看得懂的字符串表示当前图 |

关于AF,RI和rep exposure：

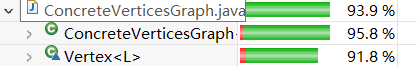


防御性拷贝使用举例：





测试覆盖率：

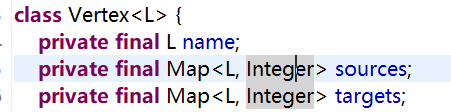


### Problem 3: Implement generic Graph<L>

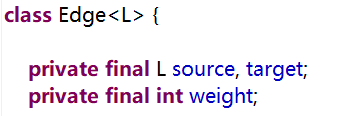
改成泛型实现

#### Make the implementations generic

使用泛型：



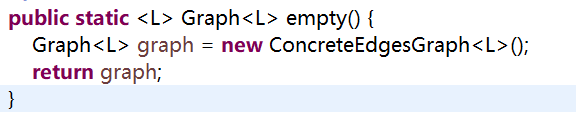






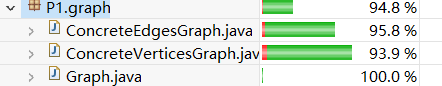
充分利用eclipse的改动错误功能快速修改成泛型实现。

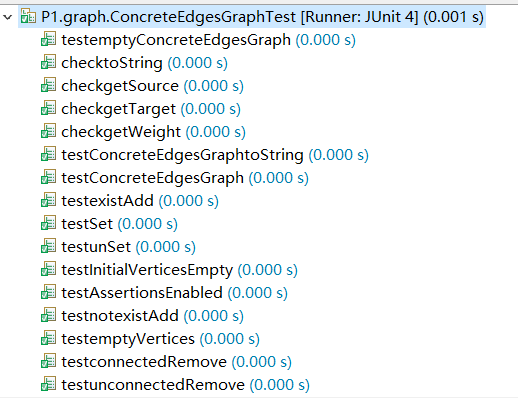
#### Implement Graph.empty()

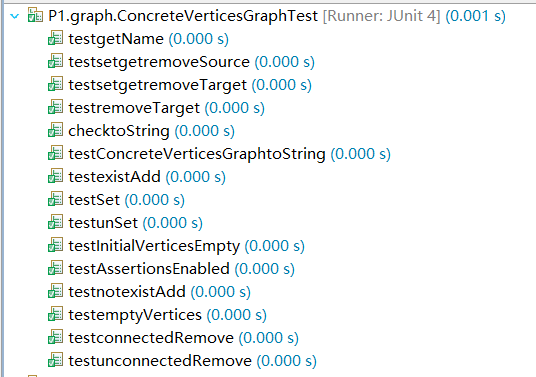


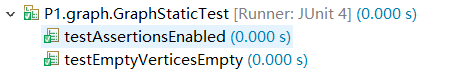
调用一个具体实现。

测试覆盖度：







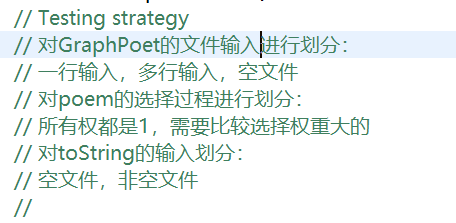


### Problem 4: Poetic walks

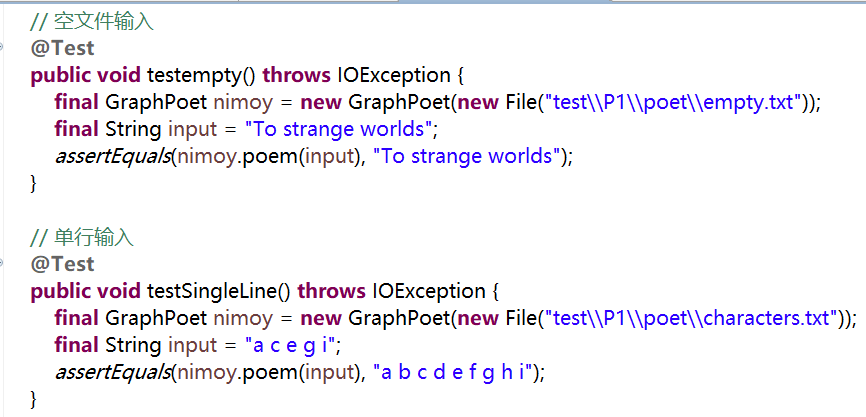
这个任务主要是要求利用之前实现的图结构，完成利用已经存在的诗句对一个新的句子进行扩充的任务。

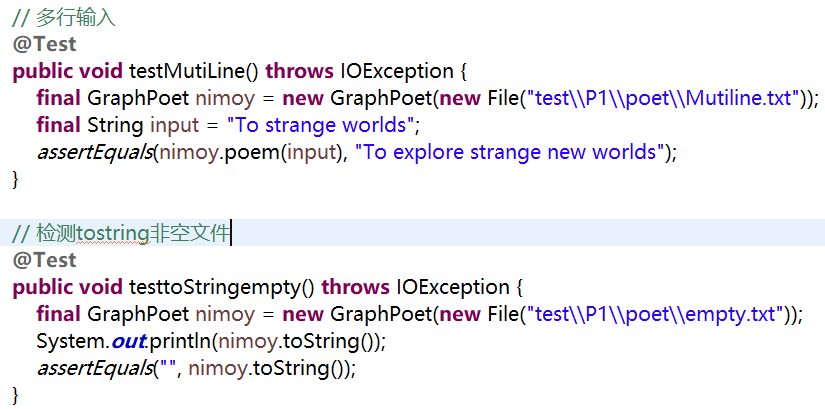
#### Test GraphPoet

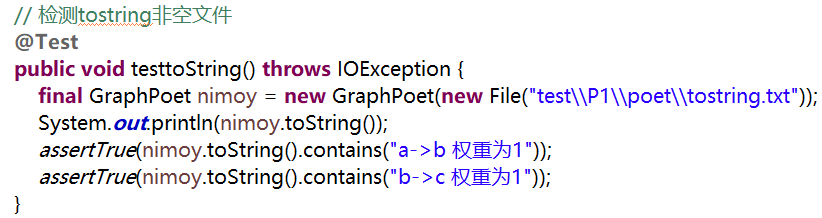
关于测试策略：

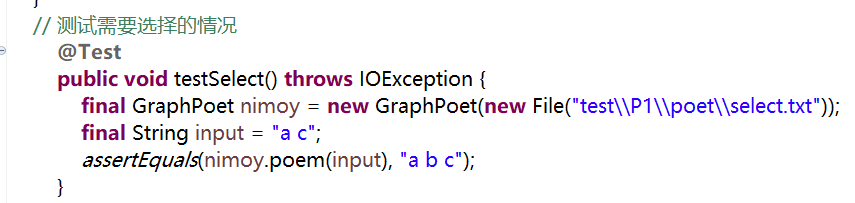


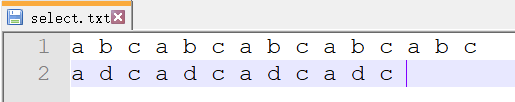
具体测试：











#### Implement GraphPoet

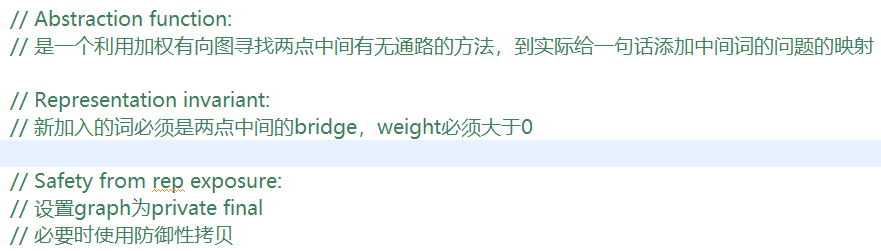
首先声明：



方法如下：

|  |  |
| --- | --- |
| GraphPoet | 参数：corpus文件路径。打开文件，读取文件输入，识别序列，构建图结构。具体：利用BufferedReader.readLine方法读取全部输入后用string.split以空格划分，保存在数组中，随后每次取相邻元素，在图中新增边。 |
| poem | 参数：input。  还是利用相同方法分割输入字符串，声明一个StringBuilder保存返回结果。每次读取一个词，然后以当前词为source，下一个词为target，在graph中寻找符合此条件的边，记录权值，结束后选择权值最大的，利用StringBuilder. Append方法，将节点名字加入字符串。 |
| toString | 调用ConcreteEdgesGraph的toString方法，输出图结构 |

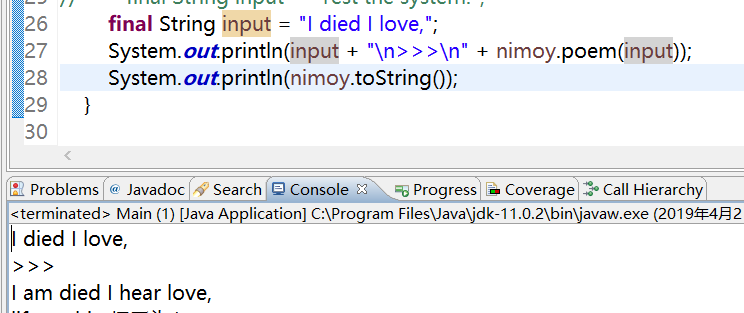
关于AF,RI和rep exposure：



#### Graph poetry slam

原始数据是泰戈尔的生如夏花

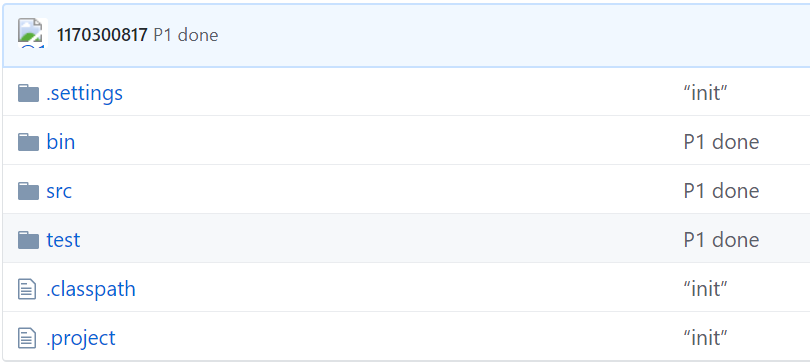
输入输出如下：



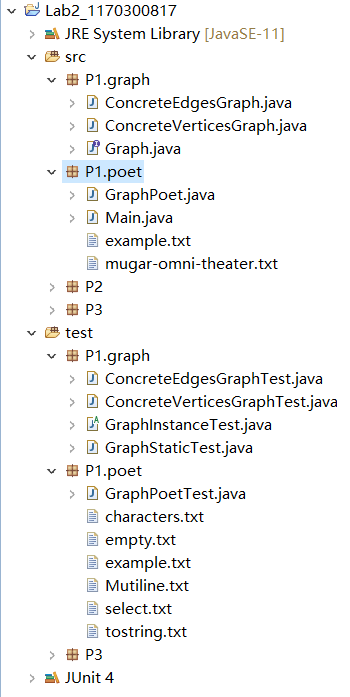
### Before you’re done

在工作目录下打开Git Gui

依次点击：rescan，stage change，输入commit信息，commut，push，就完成了。



目录结构如下：



## Re-implement the Social Network in Lab1

利用前面实现的graph，重新实现Social Network。

### FriendshipGraph类

将以前的邻接表结构改为新的有向图结构。

声明：



作为存放person点的有向图，这里采用ConcreteVerticesGraph这一实现，更加符合我们对关注的是每一个“人”的抽象。

方法如下：

|  |  |
| --- | --- |
| FriendshipGraph | 构造方法 |
| addVertex | 参数：newPerson。增加新的点，也就是增加新的人。 |
| addEdge |  |
| getDistance | 参数：Person1和Person2  获取二人距离，具体是使用队列+深度优先搜索，  distantMap保存其他人到Person1的距离  流程就是：入队自己，循环（出队，将其朋友全部入队，检查是否是Person2，不是加入distantMap，设置距离，是就返回） |

### Person类

Person类相比之前的邻接表实现可以大幅缩减了。

声明name存名字，和一个personlist表用来避免声明重复的人。



方法：

|  |  |
| --- | --- |
| Person |  |
| getName | 返回名字 |

### 客户端main()

|  |  |
| --- | --- |
| main | 实验要求提供的测试，运行结果如下 |

。

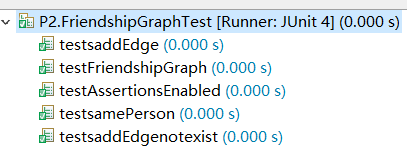
### 测试用例

测试策略：



因为之前要求声明重复人名后退出程序，于是将其和main部分注释掉后测试



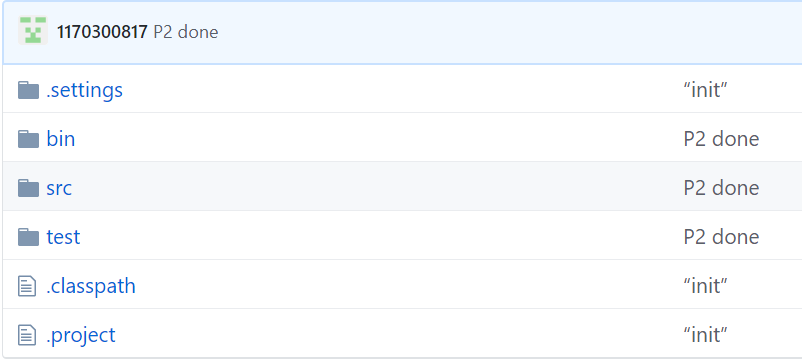


### 提交至Git仓库

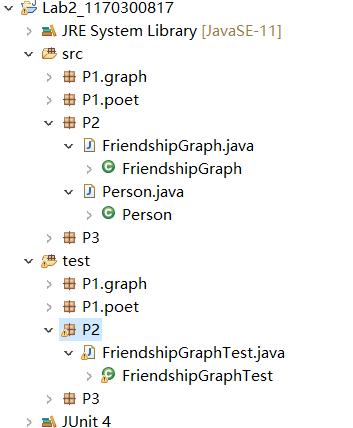
如何通过Git提交当前版本到GitHub上你的La2仓库。

在工作目录下打开Git Gui

依次点击：rescan，stage change，输入commit信息，commut，push，就完成了。



在这里给出你的项目的目录结构树状示意图。

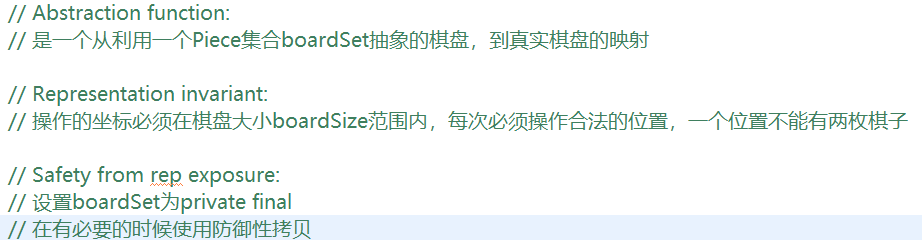


## Playing Chess

### ADT设计/实现方案

#### Board类

Board类是我的实现中最重要的类，它管理棋子，实现大部分移动增删操作。并报出大部分错误。



数据类型：

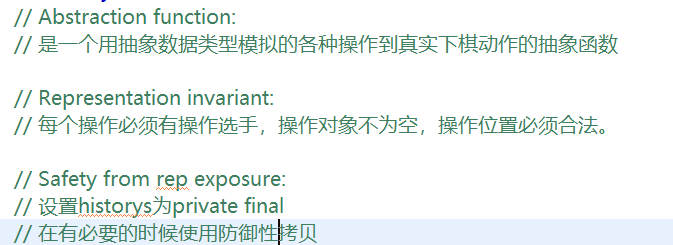
|  |  |
| --- | --- |
| boardSize | 棋盘大小 |
| boardSet | 储存棋子的集合 |
| playerA，playerB | 储存玩家命名，主要是为了返回一些说明字符串时更加明确 |

方法：

|  |  |
| --- | --- |
| getBoardSet | 返回boardSet，主要目的是方便测试 |
| Board | 参数Type。构造函数，根据传入的棋盘类型初始化boardSize。 |
| putPiece | 参数Piece piece。放置棋子，注意检查位置的有效性。 |
| checkPosivalid | 参数Position P。利用boardSize确认某位置P是否越界。 |
| removePiece | 参数Position P。移除某位置元素。注意检查位置的有效性。 |
| checkPos | 参数Position P。返回描述某个位置的string。查询操作所调用的函数。注意检查位置的有效性。 |
| countPiece | 统计双方含有的棋子数量，返回描述的字符串。 |
| checkPosempty | 参数Position P。查看一个位置是否为空，返回布尔值。 |
| move | 参数Player player, Position p1, Position p2。  移动棋子的函数，注意检查位置的有效性，还要检查移动的棋子归属是否合法。主要是利用迭代器在点的集合中搜索目标位置的点和起始位置的点进行操作。 |
| eat | 参数Player player, Position p1, Position p2。  吃子的函数，注意检查位置的有效性，还要检查移动的棋子归属是否合法，被吃的棋子的归属是否合法。实现类似move。 |

#### Action类

Action类会解析game类的调用之后调用board执行具体操作，并在操作成功时记录操作历史。



定义：



Historys数组记录历史，historycounter记录回合数。

方法：

|  |  |
| --- | --- |
| putPiece | 参数 Player player, Board board, Piece piece。调用board. putPiece放置棋子在指定的板上。成功则记录历史。 |
| removePiece | 参数 Player player, Board board, Position position。调用board. removePiece移除在指定的板上的指定位置的棋子。成功则记录历史。 |
| checkPos | 参数Board board, Position position。调用board. checkPos检查在指定的板上的指定位置的棋子。 |
| countPiece | 参数Board board。调用board. countPiece检查双方在指定的板上的棋子的数目。 |
| printHistory | 打印历史数组。 |
| move | 参数Player player, Board board, Position p1, Position p2。用board. Move移动棋子。 |
| eat | 参数Player player, Board board, Position p1, Position p2。用board. Eat实现吃子。 |

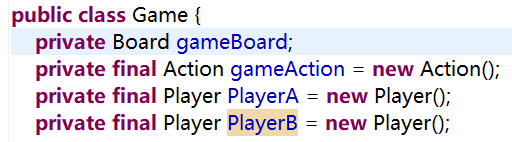
#### Game类

声明以下数据类型：

gameBoard是执行操作的棋盘对象。

gameAction是执行操作的action类的实例。

PlayerA和PlayerB是尚未被初始化的两个选手。

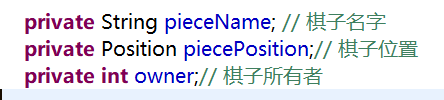


方法：（省略简单的getter和setter方法）

|  |  |
| --- | --- |
| getPlayer | 参数turn。根据turn返回此时操作的玩家。 |
| Game | 构造方法。参数type。根据传入的type区分初始化棋盘的类型，如果是国际象棋的话，要初始化32个棋子，并放置在棋盘的指定位置。 |
| setNames | 参数String nameA, String nameB。调用Player.setPlayerName初始化两名玩家的名字。 |
| addnewPiece | 参数Player player, Piece newPiece, Position P。  调用Piece.setPosition之后将点加入点集合中。 |
| removePiece | 参数Player player, Position P。调用Action.removePiece执行移除操作。 |
| checkPos | 参数Position p。调用Action.checkPos描述该位置情况。 |
| countPiece | 输出一个描述双方棋子个数的字符串 |
| printHistory | 调用Action. printHistory输出历史 |
| move | 参数Player player, Position p1, Position p2，调用Action. Move移动棋子 |
| eat | 参数Player player, Position p1, Position p2，调用Action. Move吃子 |

#### Piece类

声明如下数据结构：



方法：

（省略简单的getter和setter函数）

|  |  |
| --- | --- |
| Piece | 参数：String pieceName, int owner。构造方法赋予一个新棋子名字和所有者参数。 |
| setPosition | 参数：Integer x, Integer y。通过参数x，y新建一个位置。将目标棋子位置修改。 |

#### Player类

声明：

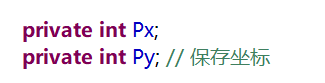


方法：

|  |  |
| --- | --- |
| getPlayerTurn | 返回玩家出手顺序 |
| setPlayerTurn | 参数：Integer turn，设置玩家出手顺序 |
| getPlayerName | 返回玩家名字 |
| setPlayerName | 参数：String playerName，设置玩家名字 |

#### Position类

声明：



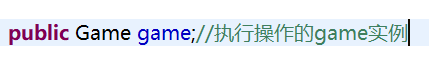
方法：

省略getter方法

|  |  |
| --- | --- |
| Position | 参数：int px, int py。构造方法 |
| equals | 参数：Position P。判断当前位置与目标位置是否相等 |

### 主程序MyChessAndGoGame设计/实现方案

声明：

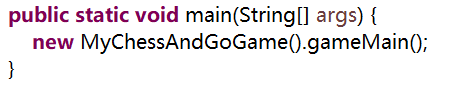


方法：

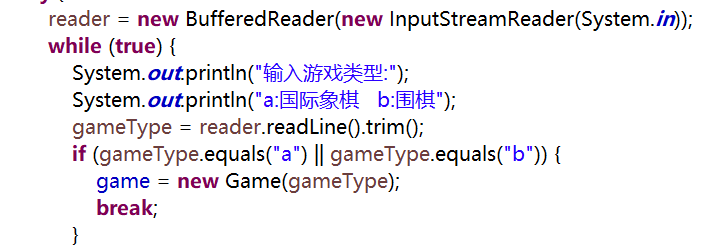
|  |  |
| --- | --- |
| goGameMenu | 输出围棋的菜单 |
| cheseGameMenu | 输出国际象棋菜单 |
| gameMain | 游戏尚未明确类型时的客户端函数 |
| goGame | 执行围棋操作 |
| cheseGame | 执行国际象棋的函数 |
| main | 声明一个MyChessAndGoGame实例，调用gameMain方法，启动游戏。 |

游戏流程：

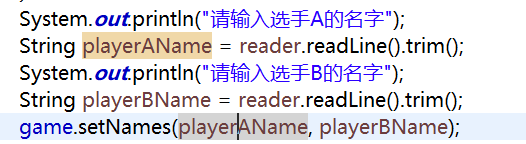
首先mian，新建实例，开始gameMain方法。



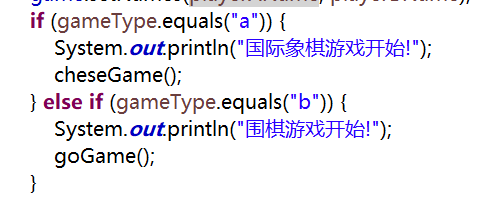
gameMain方法读取玩家输入选择游戏的种类，



初始化玩家名字，初始化棋盘，



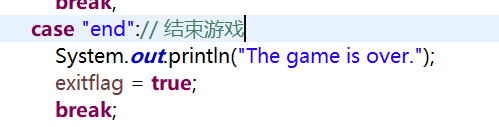
之后按游戏种类调用两个游戏方法：



在游戏函数中，针对落子，提子，吃子一类操作，都由客户端读取输入，分解用户输入，检查输入是否合法：（这里以落子为例，其他类似）

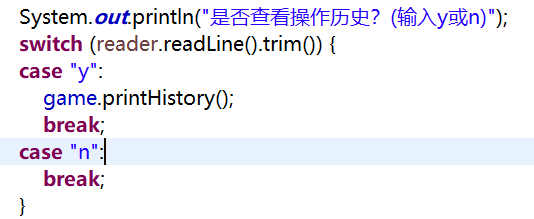


之后传递参数让game类执行操作，game类再由声明的action实例调用board类执行一系列落子，提子，吃子操作，并返回成功与否的布尔值。如果执行不成功输出错误提示信息，如果成功，由action类记录历史，返回真值，game类返回真值给客户端，客户端收到真值后，令TURN = (TURN + 1) % 2;，达到令TURN在0，1之间反复的效果。以此判断下一个选手是谁。



如果输入是end，修改exitflag之后，退出。

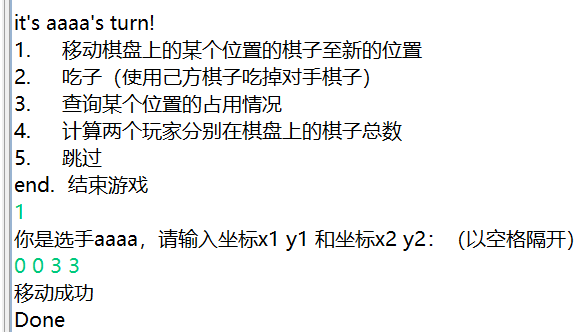
之后选择是否输出游戏历史：



演示截图：

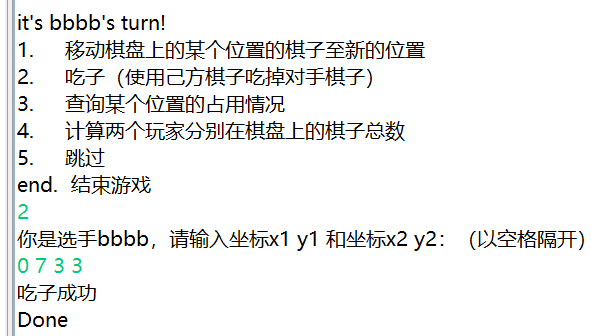
以国际象棋为例：

移动操作：

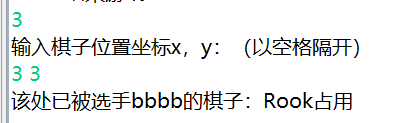


检查：

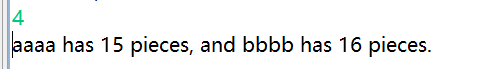
吃子



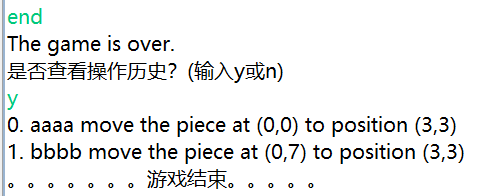
查看位置情况：



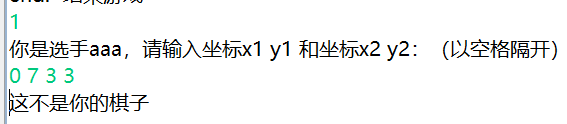
统计：

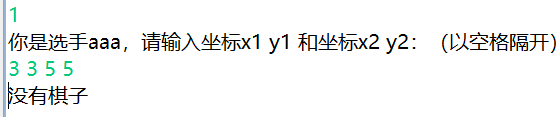


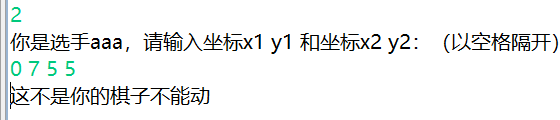
查看操作历史：

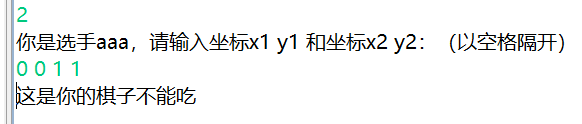


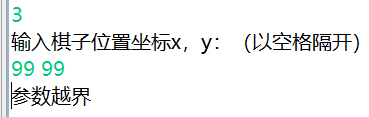
部分错误输入演示











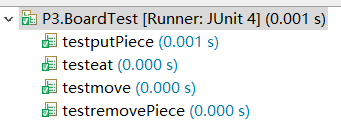
### ADT和主程序的测试方案

主程序主要使用手动测试的方法，针对参数越界，控制权不符合要求，输入参数不足等情况手动测试。

具体操作主要是在board类，所以对board类详细测试：

主要测试putPiece，removePiece，move，eat几个重点操作，





## Multi-Startup Set (MIT)

请自行设计目录结构。

注意：该任务为选做，不评判，不计分。

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.18 | 19:00-22:30 | P1 p1 | 看懂了实验要求，阅读了一些提供的reading |
| 3.19 | 19:00-22:30 | P1 p1 | 学习有关泛型的内容 |
| 3.20 | 9:30-22:30 | P1 p1,2,3 | 学习有关AF,RI,表示泄露的内容，完成P1.1和2 |
| 3.22 | 19:00-22:30 | P1 p3 | 完成 |
| 3.23 | 19:00-22:30 | P1 p4 | 完成 |
| 3.24 | 19:00-22:30 | 补全P1的test和说明 | 完成 |
| 3.25 | 19:00-22:30 | P2 ，设计P3思路 | 完成， |
| 3.26 | 19:00-22:30 | P3， | 未完成 |
| 3.27 | 9:30-22:30 | P3 | 未完成 |
| 3.28 | 9:30-22:30 | P3 | 完成 |
| 3.29 | 19:00-22:30 | 补全测试，撰写报告 | 补全测试，撰写报告 |
| 3.30 | 19:00-22:30 | 补全测试，撰写报告 | 补全测试，撰写报告 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 思考那些时候需要进行防御性拷贝，和出现表示泄露问题 | 回看ppt，参考MIT的readings |
| 思考P3如何设计让各类各司其职 | 没有合适解决，觉得自己类与类间分工不是很明确，无法做到体面配合。 |
|  |  |

# 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

设计多个类让他们互相配合的能力很差，感觉代码的逻辑有待提高。

## 针对以下方面的感受

1. 面向ADT的编程和直接面向应用场景编程，你体会到二者有何差异？
2. 使用泛型和不使用泛型的编程，对你来说有何差异？
3. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？
4. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？
5. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？
6. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？
7. 关于本实验的工作量、难度、deadline。
8. 《软件构造》课程进展到目前，你对该课程有何体会和建议？
9. 对实际物体进行抽象，与面向过程完全不同
10. 泛型更加灵活
11. 不是特别适应，还是习惯先写代码
12. 不用再写一遍针对某个类型的代码
13. 分析清楚抽象函数大大有助于我们编写代码，类似设计中的核心思想，以后要先总结出这个类的抽象函数是什么，再开始实现。自主设计多个类的功能较难，虽然最后是实现了要求，但是总觉的自己的实现很啰嗦，很繁杂，有一些没必要的函数沦为空壳。
14. 避免内部变量被修改，很有必要，不然会出现一些很难察觉的bug
15. 工作量不小，难度适中，deadline不紧不慢很合适。
16. 希望老师每个ppt在每一节开始之前就一次性修改好，避免中途修改，因为现在有很多同学使用平板设备在pdf上面做笔记，pdf更新进退两难