

**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 | 汤添凝 |
| 学号 | 1170300728 |
| 班号 | 1703007 |
| 电子邮件 | ttn912@126.com |
| 手机号码 | 18016240627 |

**目录**

目录

[1 实验目标概述 1](#_Toc19180)

[2 实验环境配置 2](#_Toc11490)

[3 实验过程 3](#_Toc29876)

[3.1 Poetic Walks 3](#_Toc10579)

[3.1.1 Get the code and prepare Git repository 3](#_Toc10148)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc22225)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc3802)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc2624)

[3.1.3.1.1 boolean add(String vertex) 4](#_Toc4344)

[3.1.3.1.2 public int set(String source, String target, int weight) 4](#_Toc10180)

[3.1.3.1.3boolean remove(String vertex) 5](#_Toc24952)

[3.1.3.1.4Map<String, Integer> sources(String target) 5](#_Toc32070)

[3.1.3.1.5Map<String, Integer> targets(String source) 6](#_Toc17802)

[3.1.3.1.6String toString() 6](#_Toc30304)

[3.1.3.1.7class Edge<String> 7](#_Toc28907)

[3.1.3.2 Implement ConcreteVerticesGraph 8](#_Toc7258)

[3.1.3.2.1 boolean add(String vertex) 8](#_Toc16974)

[3.1.3.2.2public int set(String source, String target, int weight) 8](#_Toc11101)

[3.1.3.2.3boolean remove(String vertex) 9](#_Toc9306)

[3.1.3.2.4Map<String, Integer> sources(String target) 9](#_Toc3603)

[3.1.3.2.5Map<String, Integer> targets(String source) 10](#_Toc16201)

[3.1.3.2.6String toString() 10](#_Toc23354)

[3.1.3.2.7class Vertex<String> 11](#_Toc18588)

[3.1.4 Problem 3: Implement generic Graph<L> 12](#_Toc3331)

[3.1.4.1 Make the implementations generic 12](#_Toc27032)

[3.1.4.2 Implement Graph.empty() 12](#_Toc10287)

[3.1.5 Problem 4: Poetic walks 12](#_Toc14934)

[3.1.5.1 Test GraphPoet 12](#_Toc31047)

[3.1.5.2 Implement GraphPoet 12](#_Toc32435)

[3.1.5.2.1 GraphPoet(File corpus) 13](#_Toc100)

[3.1.5.2.2 String poem(String input) 13](#_Toc31588)

[3.1.5.2.3 Graph poetry slam 14](#_Toc10333)

[3.1.6 Before you’re done 14](#_Toc11507)

[3.2 Re-implement the Social Network in Lab1 15](#_Toc3168)

[3.2.1 FriendshipGraph类 15](#_Toc14203)

[3.2.2 Person类 15](#_Toc26422)

[3.2.3 测试用例 16](#_Toc23890)

[3.2.4 提交至Git仓库 16](#_Toc25028)

[3.3 Playing Chess 17](#_Toc31756)

[3.3.1 ADT设计/实现方案 17](#_Toc25228)

[3.3.1.1 Piece 17](#_Toc19059)

[3.3.1.2 Position 17](#_Toc20025)

[3.3.1.3 Board 18](#_Toc10091)

[3.3.1.4 Player 19](#_Toc26998)

[3.3.1.5 Action 20](#_Toc2274)

[3.3.1.5 Game 21](#_Toc20135)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 22](#_Toc21296)

[3.3.2.1 StartGame() 22](#_Toc15930)

[3.3.2.2 Chesstime() 22](#_Toc25756)

[3.3.2.3 Gotime() 23](#_Toc1704)

[3.3.3 ADT和主程序的测试方案 24](#_Toc15838)

[4 实验进度记录 25](#_Toc26978)

[5 实验过程中遇到的困难与解决途径 25](#_Toc14290)

[6 实验过程中收获的经验、教训、感想 26](#_Toc19516)

[6.1 实验过程中收获的经验和教训 26](#_Toc30658)

[6.2 针对以下方面的感受 26](#_Toc17170)

# 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

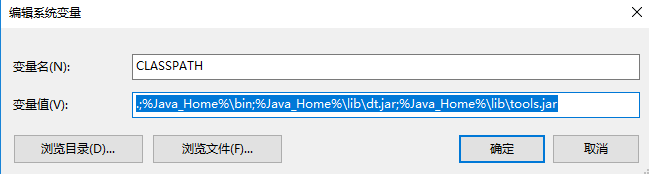
<http://web.mit.edu/6.031/www/fa18/getting-started/>中的安装Eclipse和Gitd的指南非常详细，按照文字和图片即可完成，但是JDK的官网里没有8、9、10版本的，所以我先下载了JDK11，后来在其他网站找到了JDK8版本。

java环境的配置：

安装好JDK后，计算机→属性→高级系统设置→高级→环境变量→系统变量→新建JAVA\_Home变量（填写JDK的安装路径如下图） 

接下来找到并编辑Path变量，在变量值最后输入“%JAVA\_HOME%\bin;%JAVA\_HOME%\jre\bin;”

最后新建CLASSPATH变量，变量值填写“ .;%JAVA\_HOME%\lib;%JAVA\_HOME%\lib\tools.jar ”，

系统变量就配置好了。

URL：<https://github.com/ComputerScienceHIT/Lab2-1170300728>

# 实验过程

## Poetic Walks

这个任务涉及到了图的定义和实现，在过往的算法设计课上已经有了些许经验，结合近期学习的类和继承的知识可以较为轻松地完成。

### Get the code and prepare Git repository

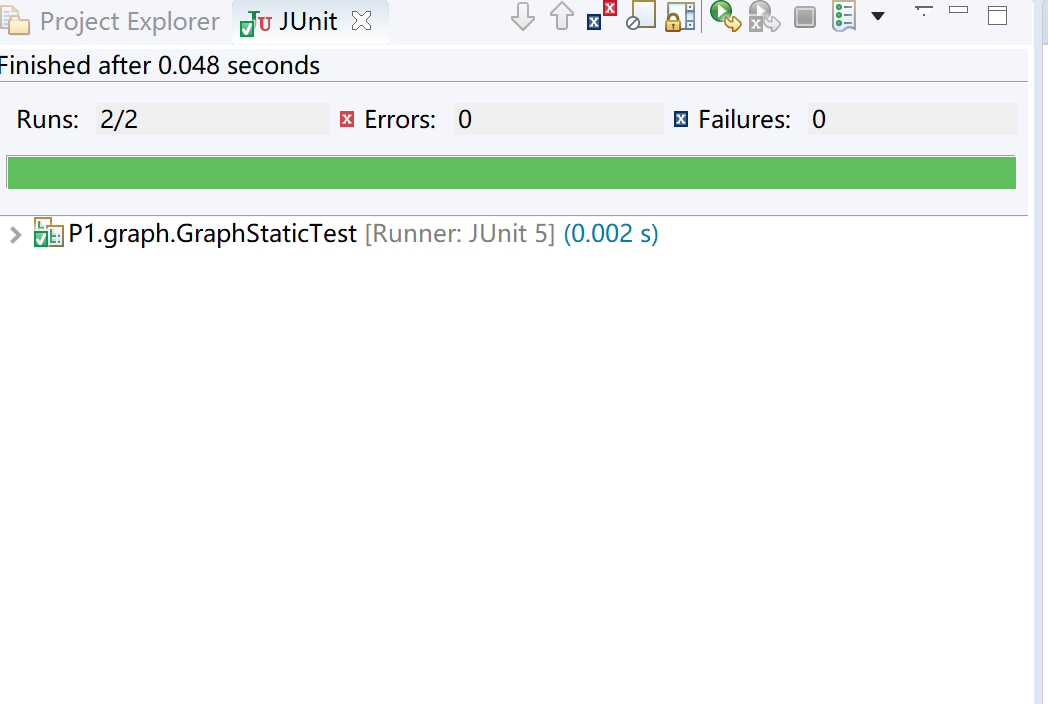
如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

### Problem 1: Test Graph <String>

这部分是对Graph本身的测试，比较简单。

值得一提的是GraphInstanceTest.java函数。它是用于检测空图的实例，也是从未有过的特殊测试类型。

这一部分的测试截图如下：



### Problem 2: Implement Graph <String>

这部分要求学生利用继承的关系设计一个基于点的图类ConcreteEdgesGraph和一个基于边的图类ConcreteVerticesGraph。下面我们分别详细阐述一下。

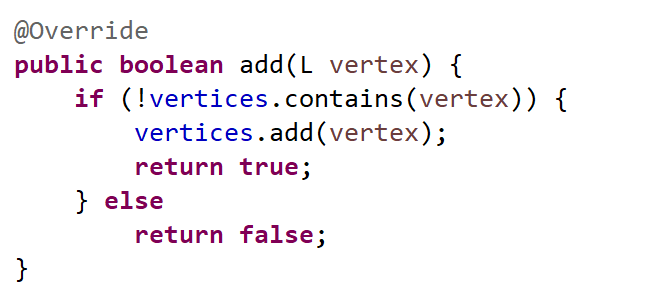
### 3.1.3.1 Implement ConcreteEdgesGraph

这个图类主要包含如下几个部分：

### 3.1.3.1.1 boolean add(String vertex)

就算是基于边的图也需要记录点的信息，否则构不成边，所以先要有一个添加点的函数来完成这一任务。

代码如下：

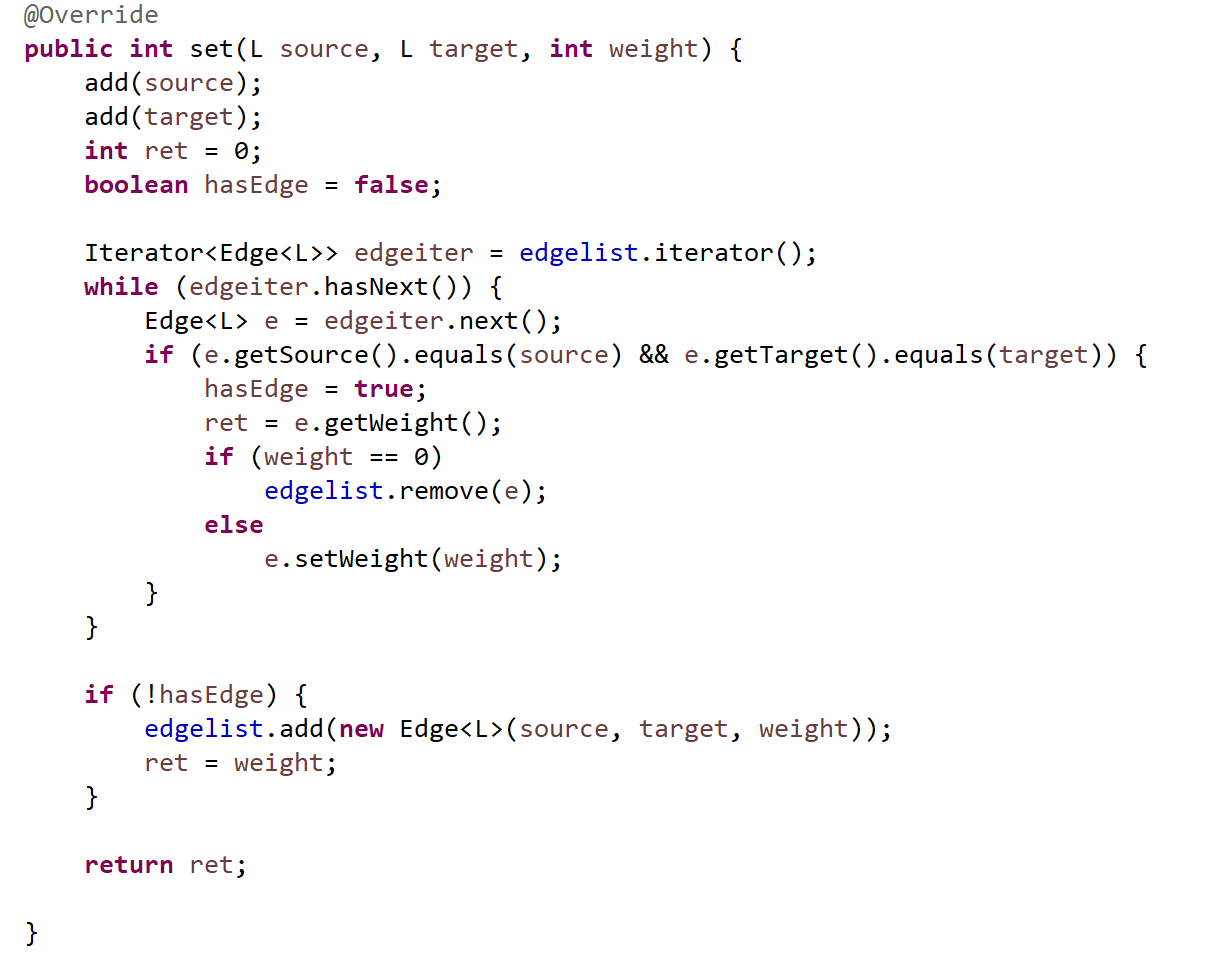


### 3.1.3.1.2 public int set(String source, String target, int weight)

由于是基于边的图，我们需要设计添加边的函数来完成图的基础。

需要注意的是，添加边有各种要求，比如边的两端点要存在，边不能已经存在等。

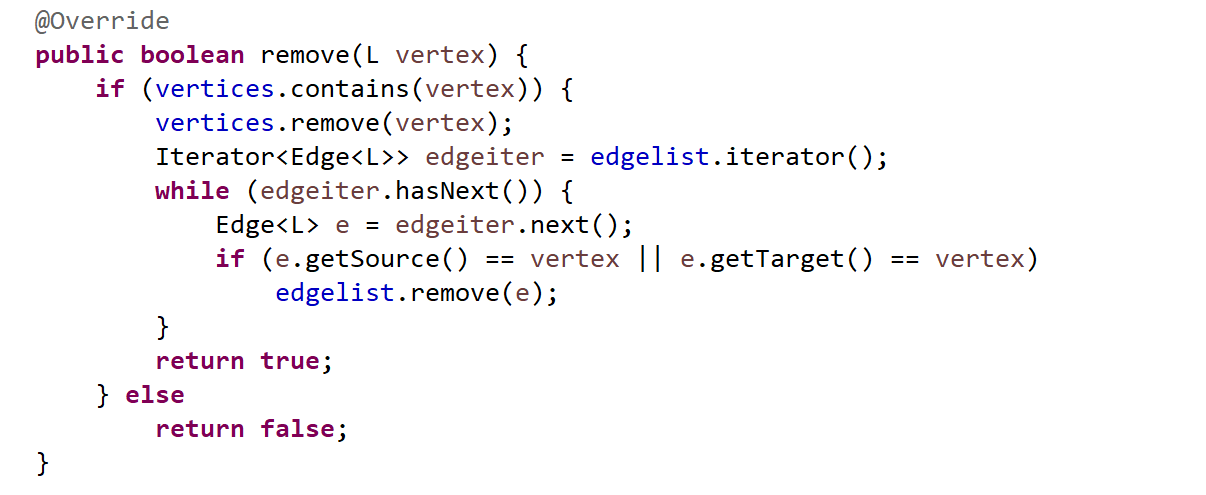
代码如下：

****

### 3.1.3.1.3boolean remove(String vertex)

在基于边的图中，删除点是比较困难的，需要考虑边的起点和终点两种情况。

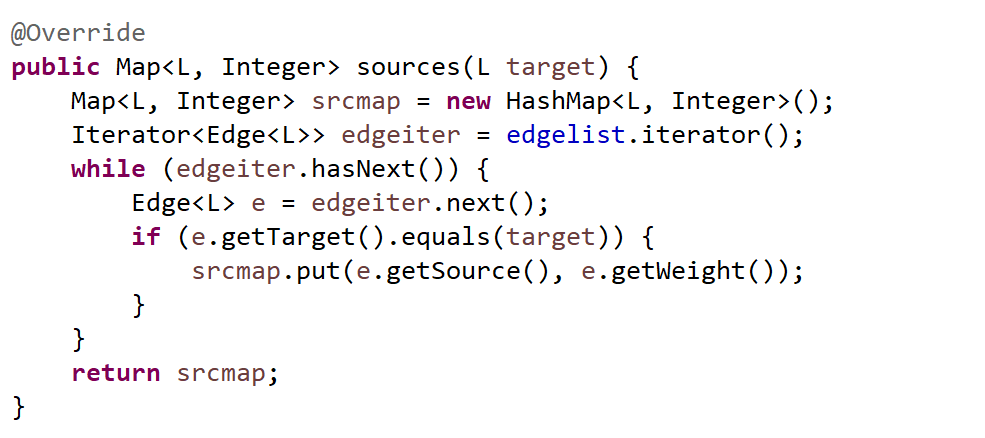
具体代码如下：

****

### 3.1.3.1.4Map<String, Integer> sources(String target)

这个函数的功能是返回一个以target为终点的map，记录能到达target的点以及这两点构成的边的权重。利用边集的遍历获取顶点信息可以轻松完成。

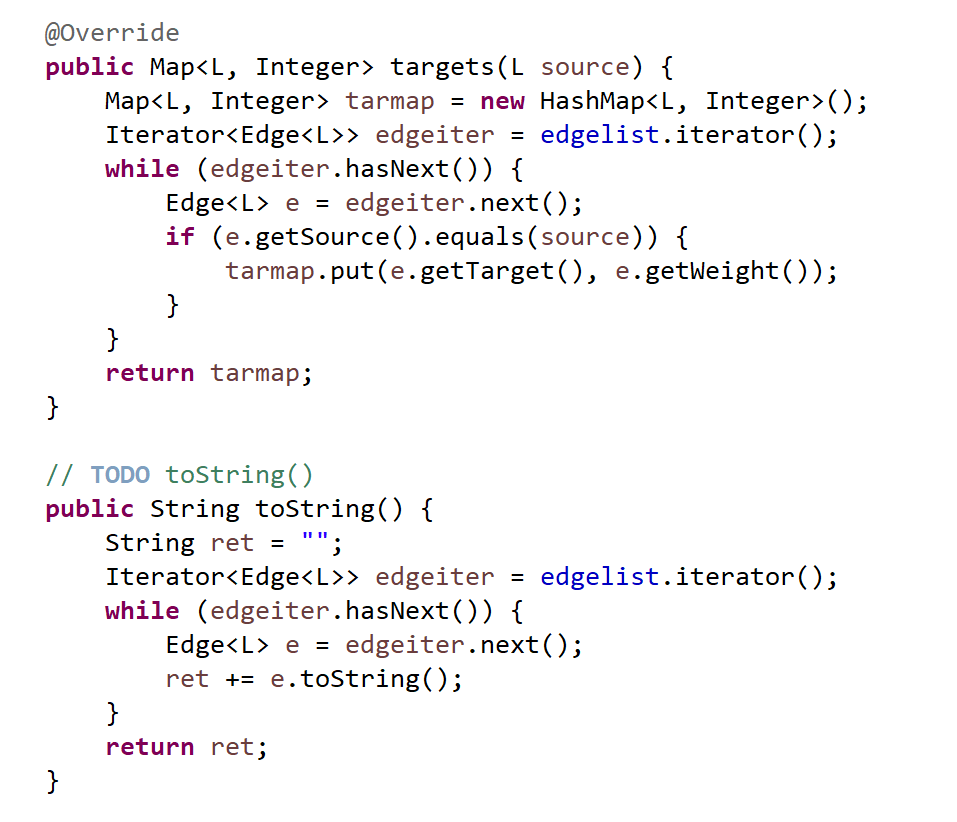
代码如下：

****

### 3.1.3.1.5Map<String, Integer> targets(String source)

这个函数的功能是返回一个以source为起点的map，记录source能到达的点以及这两点构成的边的权重。利用边集的遍历获取终点信息可以轻松完成。

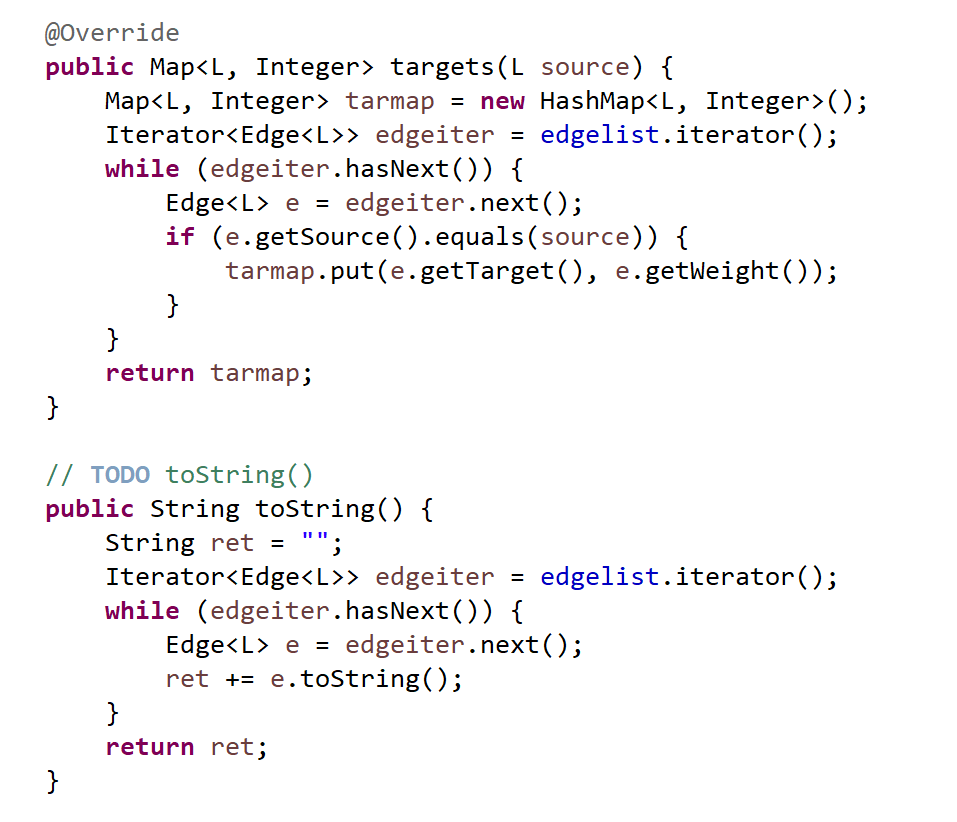
代码如下：

****

### 3.1.3.1.6String toString()

这个函数是针对本图的规格化输出方法。一张图需要一个合理的方式去清晰明了的输出图的结果。

代码如下：

****

### 3.1.3.1.7class Edge<String>

这个类是基于边的图的实现核心，其中包含各种对边的信息的调用、存储、计算方法，将图中的边记录为Edge类型的列表是基于边的图的设计核心思路。

具体如下：

****

### 3.1.3.2 Implement ConcreteVerticesGraph

这个图类主要包含如下几个部分：

### 3.1.3.2.1 boolean add(String vertex)

由于是基于顶点的图。记录顶点的信息尤为重要，所以先要有一个添加点的函数来完成这一任务。需要注意的是，这里的顶点需要设置各种初始化信息，以便之后调用和操作。

代码如下：



### 3.1.3.2.2public int set(String source, String target, int weight)

虽然是基于顶点的图，但我们仍然需要设计添加边的函数来完成图的基础结构。

代码如下：



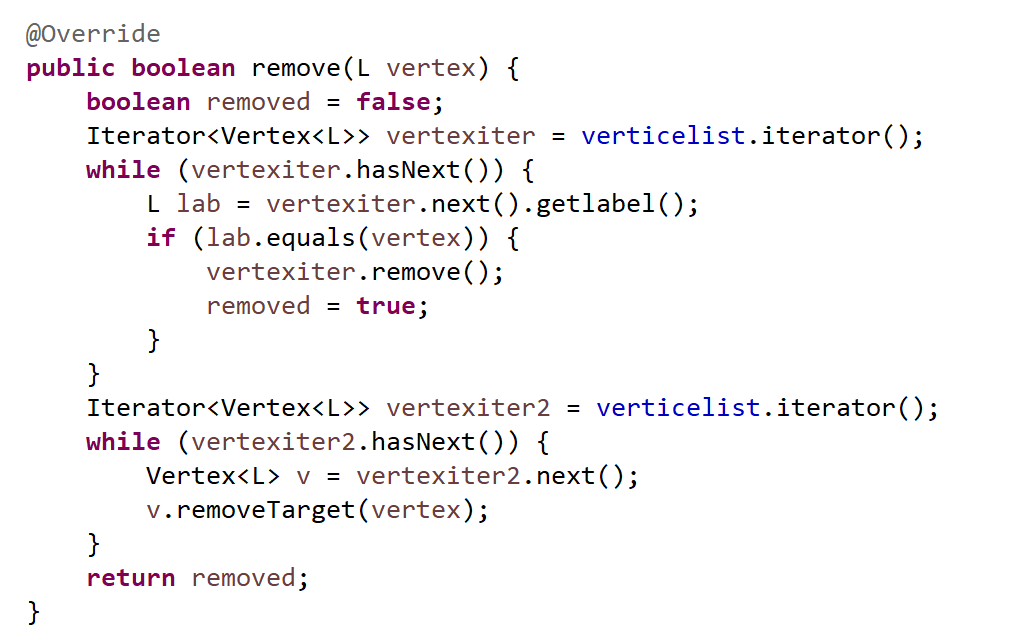
### 3.1.3.2.3boolean remove(String vertex)

在基于顶点的图中，删除点应该是比较简单的，但仍然需要考虑边的起点和终点两种情况。

然而，实际编程过程中，我遇到了很多问题，其中最具典型的是关于list的for循环遍历和remove函数的冲突。当我在for循环遍历的过程中使用remove时，程序会报错，因为此时我的list中有被remove的元素，而循环却不知道。所以需要将list转换为iterator类型以保证能够找到对应的元素并删除。其次，不能在循环过程中删除终点，这样会导致找到顶点后的终点不被删除。解决方案是用两次循环遍历，分别寻找终点和起点并删除。

值得一提的是在基于边的顶点删除中回避了这些问题。

具体代码如下：

****

### 3.1.3.2.4Map<String, Integer> sources(String target)

这个函数的功能是返回一个以target为终点的map，记录能到达target的点以及这两点构成的边的权重。利用顶点集的遍历获取边信息可以轻松完成。

代码如下：

****

### 3.1.3.2.5Map<String, Integer> targets(String source)

这个函数的功能是返回一个以source为起点的map，记录source能到达的点以及这两点构成的边的权重。利用顶点集的遍历获取边信息可以轻松完成。

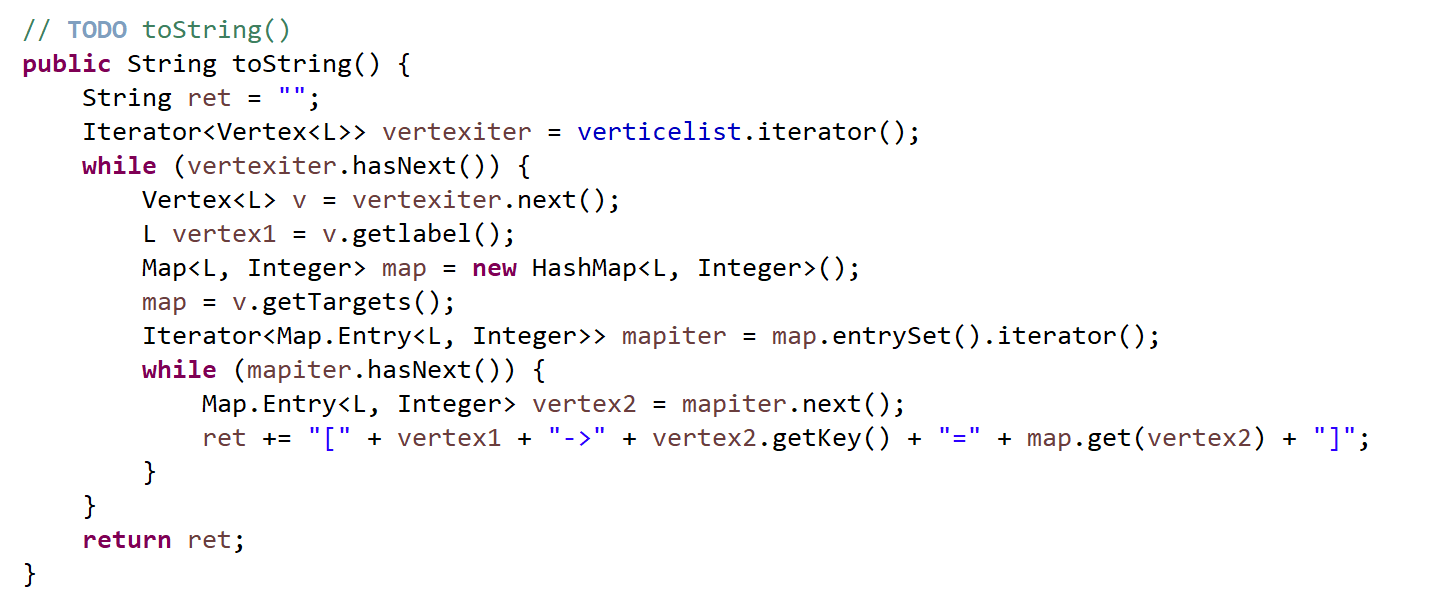
代码如下：

****

### 3.1.3.2.6String toString()

这个函数是针对本图的规格化输出方法。一张图需要一个合理的方式去清晰明了的输出图的结果。

代码如下：

****

### 3.1.3.2.7class Vertex<String>

这个类是基于顶点的图的实现核心，其中包含各种对顶点的信息的调用、存储、计算方法，将图中的顶点记录为Vertex类型的列表是基于边的图的设计核心思路。

具体如下：

****

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

泛型是程序设计中的常用类设计比较关键的思想，如果一个图只能够保存String类型的信息，那这个图的应用范围终究是相当狭窄的。这一个阶段我们要将已完成的两种类型的图修改为泛型可用。

#### Make the implementations generic

主要内容就是修改前面的操作中的String为泛型<L>此时会有各种报错，因为我们的代码很多地方默认了元素为String，依照泛型的方法进行修改即可。

由于报告撰写时间晚于实际实验时间，截图时已经完成了泛型的修改，此处不再截图。

#### Implement Graph.empty()

Empty其实相当简单，只需要新建一个对应的图即可。

### Problem 4: Poetic walks

这部分是MIT实验中最有意思的一部分，首先扩充了一些作诗技巧。实际使用这个技巧则需要我们引用前面所写的graph，也让我们枯燥的实验有了一些乐趣，是实验设计别出心裁的一章。

#### Test GraphPoet

测试初始的内容。

#### Implement GraphPoet

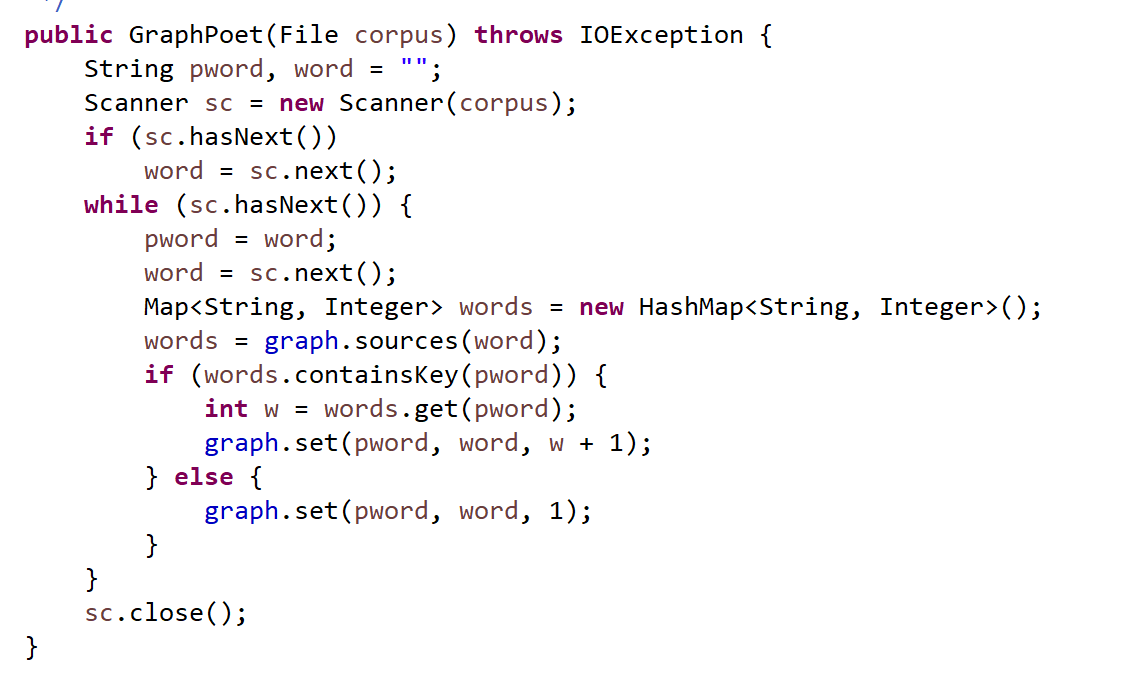
这一部分是Poetic walks的核心部分，包含如下几个核心函数：

### 3.1.5.2.1 GraphPoet(File corpus)

这一部分的核心内容是从文件中读取已有的诗歌信息，并记录在图中，获得一个类似MIT页面中所示的图。

由于前面的graph是自己写的，使用起来得心应手，可以轻松完成图的构建。

代码如下：



### 3.1.5.2.2 String poem(String input)

这一部分的要求是通过input的字符串内容，赠补出一个新的诗歌，并返回。简而言之就是找到input中相邻的各个单词并检索上一部中构建的图中是否存在这两个元素，且中间夹着另一个元素，如果有，加入这次的诗歌中。这也就是Poetic walks的核心含义。

代码如下：



### 3.1.5.2.3 Graph poetry slam

由于我一开始没看到提供了txt文件所以在撰写GraphPoemTest.java文件时，摘录了诗歌London的内容作为文件，与其他同学不一样，我这一步是在test中完成的，而main函数中则是老师提供的案例。

具体如下：

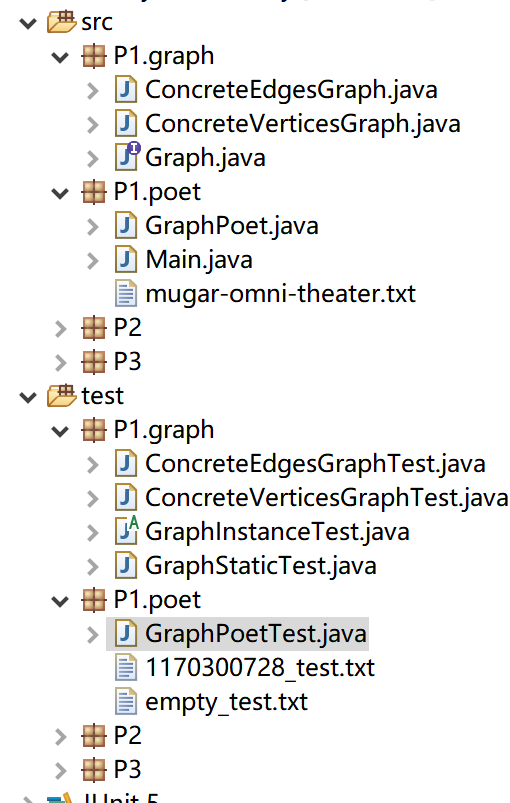


### Before you’re done

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/" \l "before_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。

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

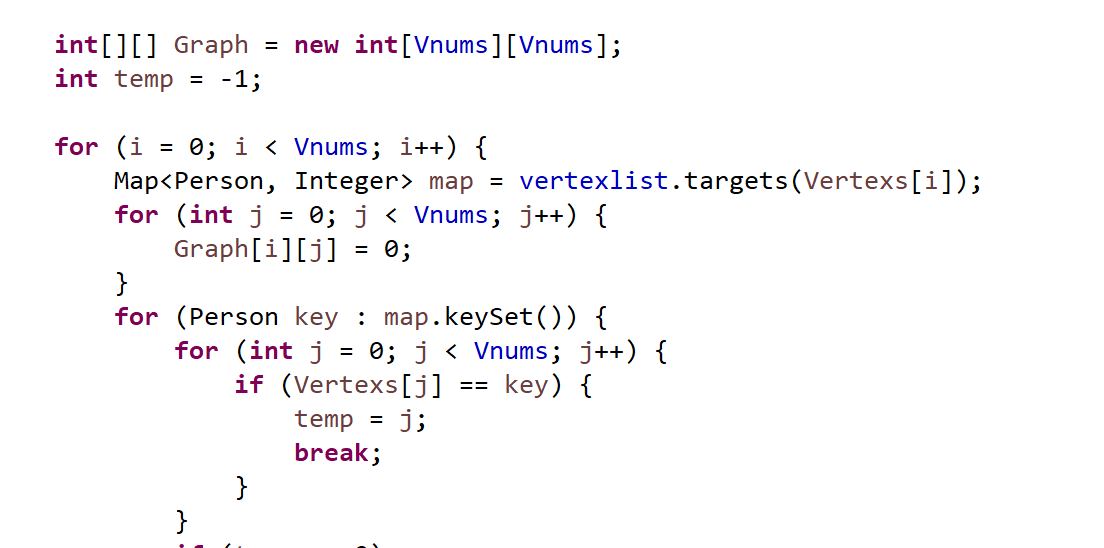
这一部分要求学生使用3.1中的graph完成Lab1中的Social Network问题，是对自己写的代码的使用的又一考验。

### FriendshipGraph类

要我构建一个社会关系网，对人与人之间的关系进行处理，得出正确的结果，同时要对输入的不同情况进行判断。

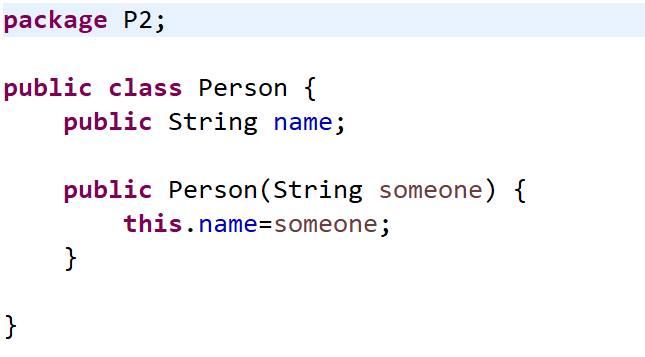
构建person图，利用之前的add和set函数，将人和关系加入到图当中，以备后续使用。

由于人数是不确定的，而且自己设计的图自知难以使用递归等算法遍历和查找，所以只得转化为邻接矩阵，结合dijsktra算法实现最短路径，部分代码如下：



### Person类

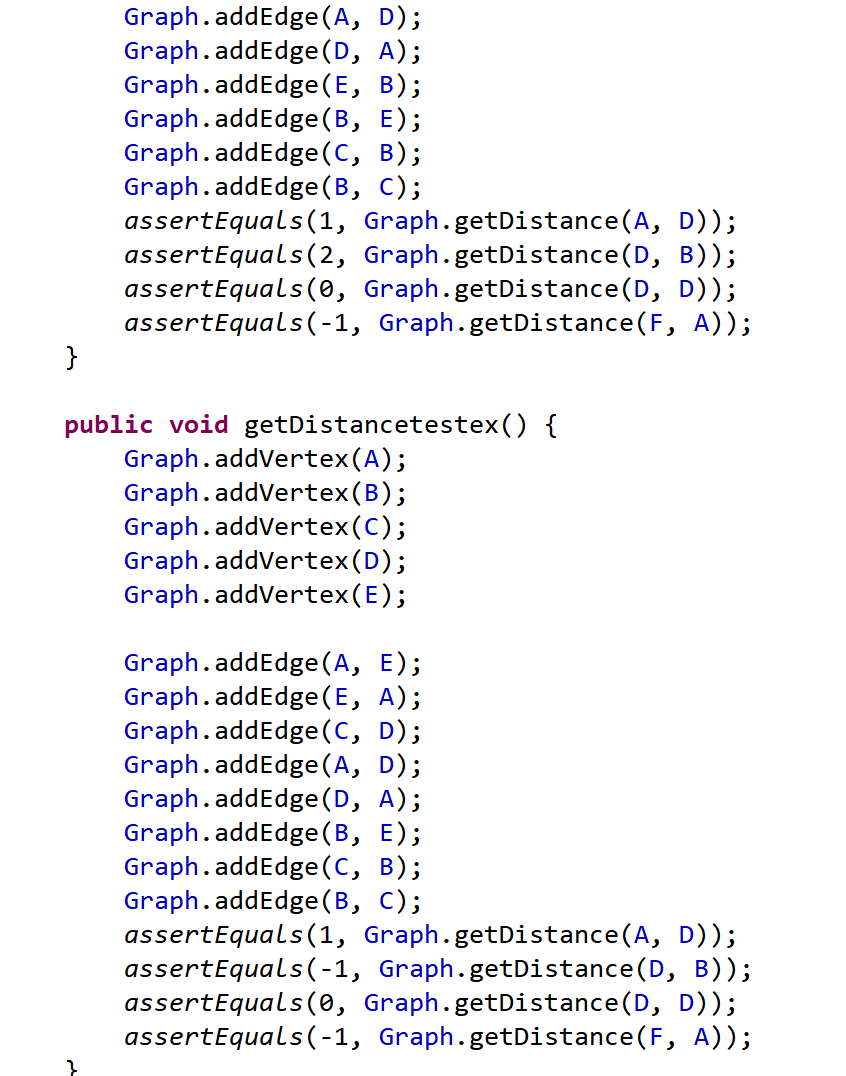
如下图所示，Person类被我很简单的完成，仅仅记录了姓名信息。

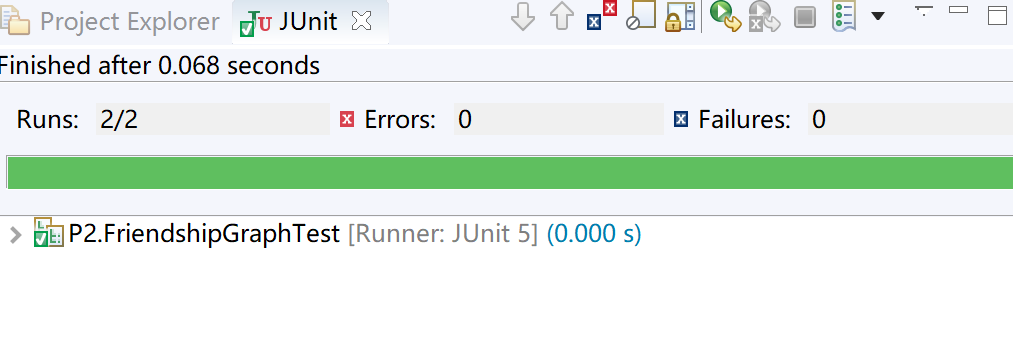


### 测试用例

这里我直接引用了上次实验的测试用例，因为函数所完成的任务是一致的，不用修改测试用例，也应当能通过测试。

部分代码和测试结果如下：

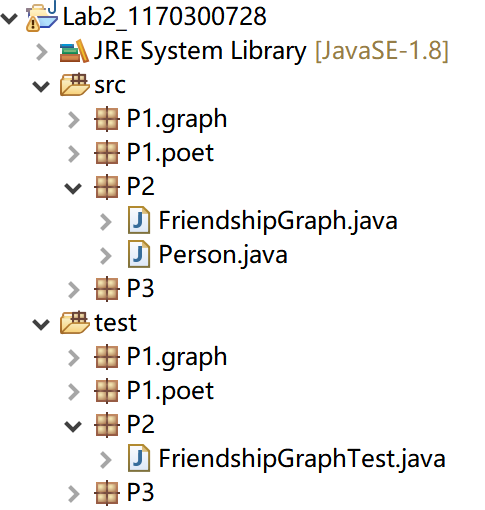




### 提交至Git仓库

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。



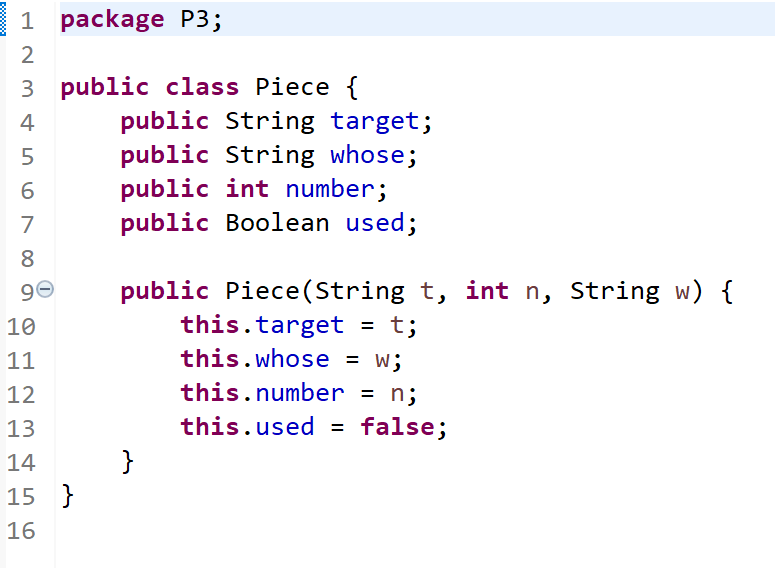
## Playing Chess

### ADT设计/实现方案

依据实验指导书的提示，我构建了Game、Player、Board、Piece、Position、Action六个ADT。

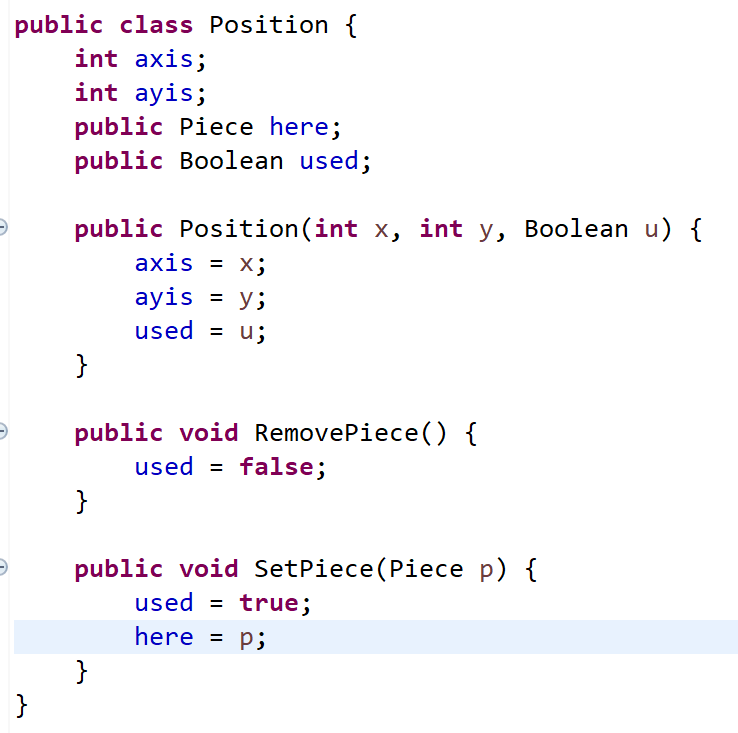
### 3.3.1.1 Piece

PieceADT中存储了单个棋子的信息，如所有者，是什么类型的棋子，是否被使用等信息，以便其他ADT调用。



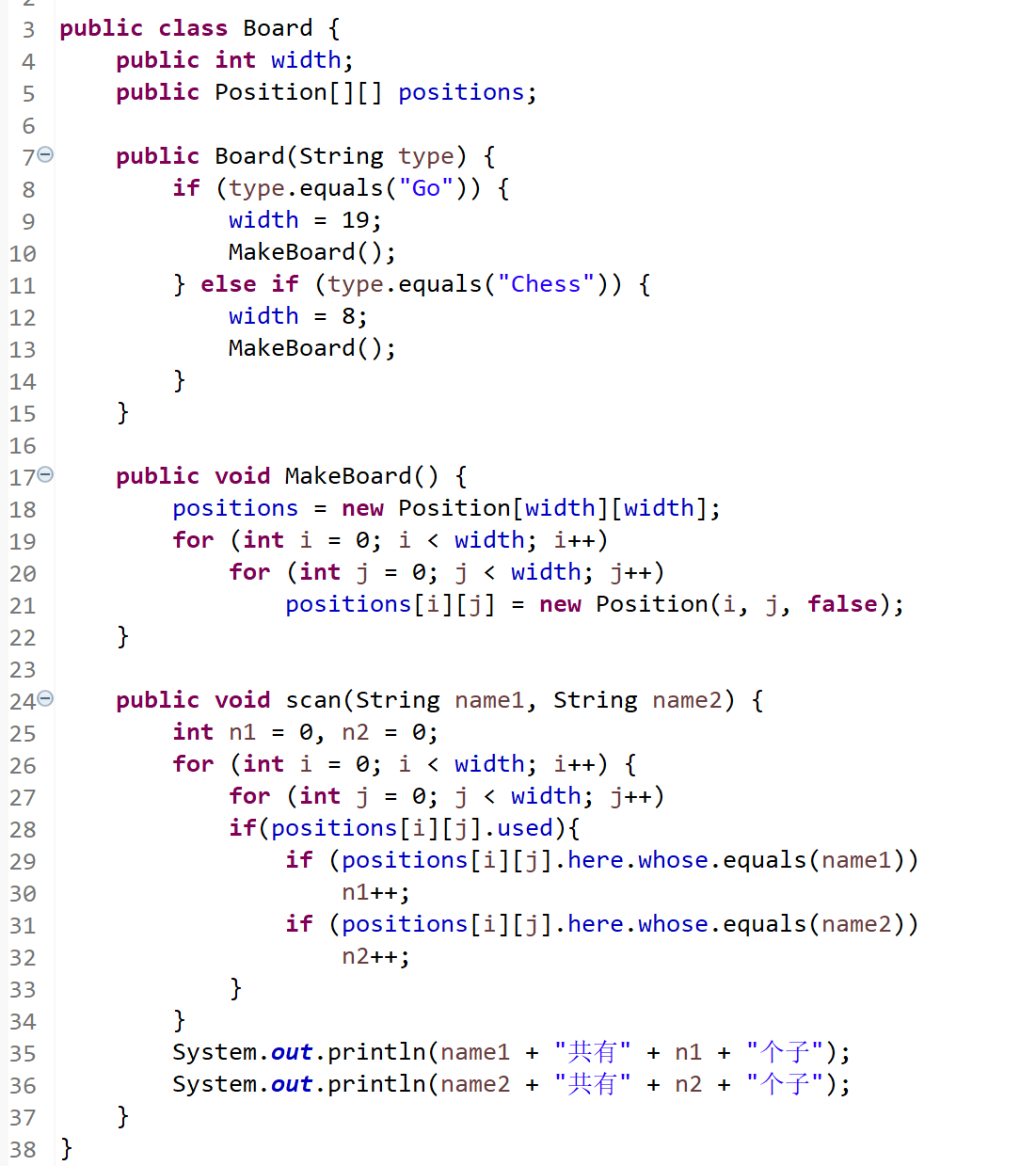
### 3.3.1.2 Position

PositionADT中存储了单个位置的信息，如在棋盘上的横纵坐标位置，是否有棋子，和所有的棋子的信息，并提供了落子和提子的底端实现。



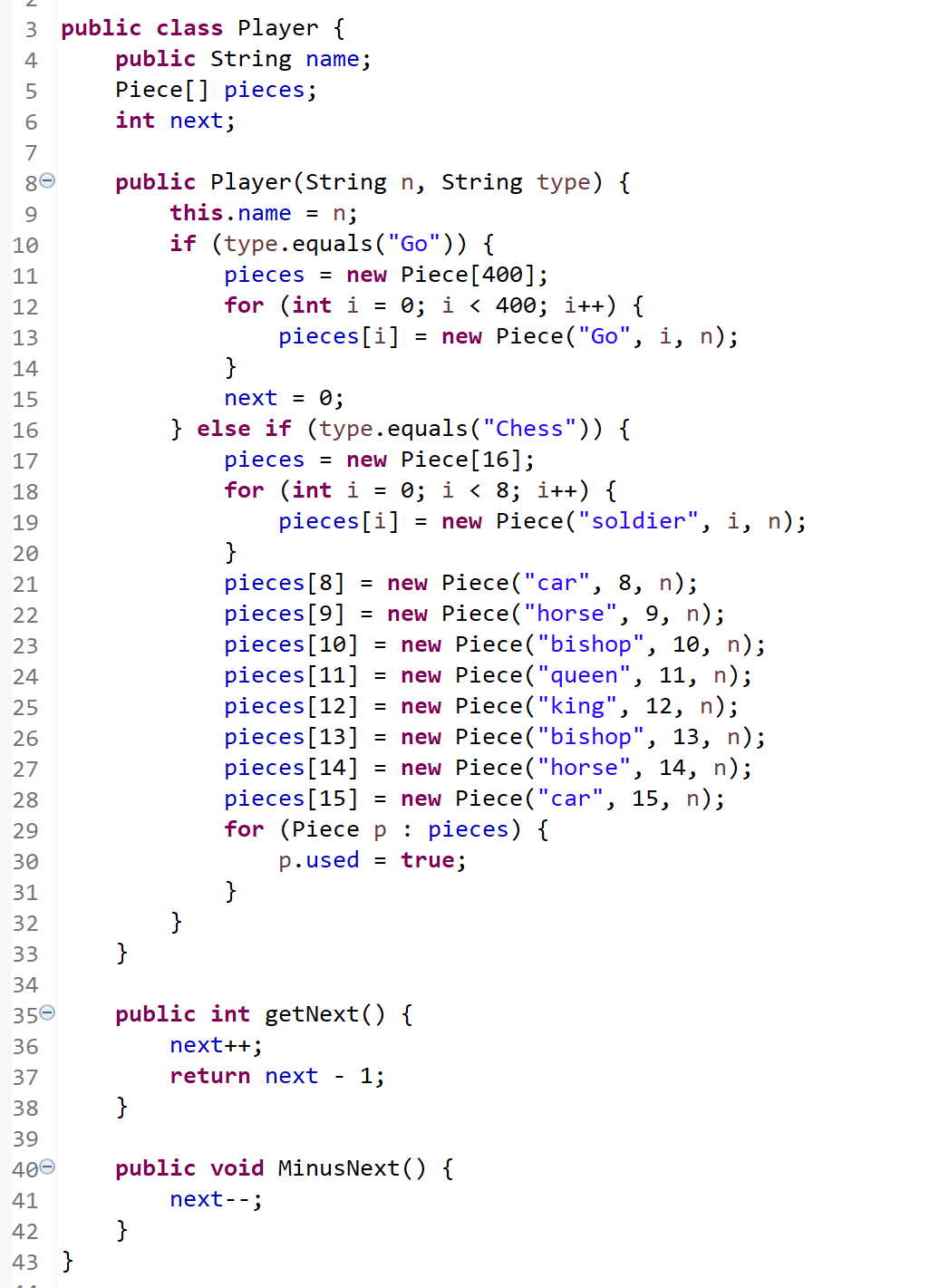
### 3.3.1.3 Board

BoardADT中存储了本局游戏的棋盘信息，如棋盘的宽度和对应棋盘大小的n\*n个Position构成的集合，以便对棋盘进行全局的把控。同时还提供了自动的制作棋盘的函数和检索棋盘查找两位玩家分别还有多少棋子的函数。



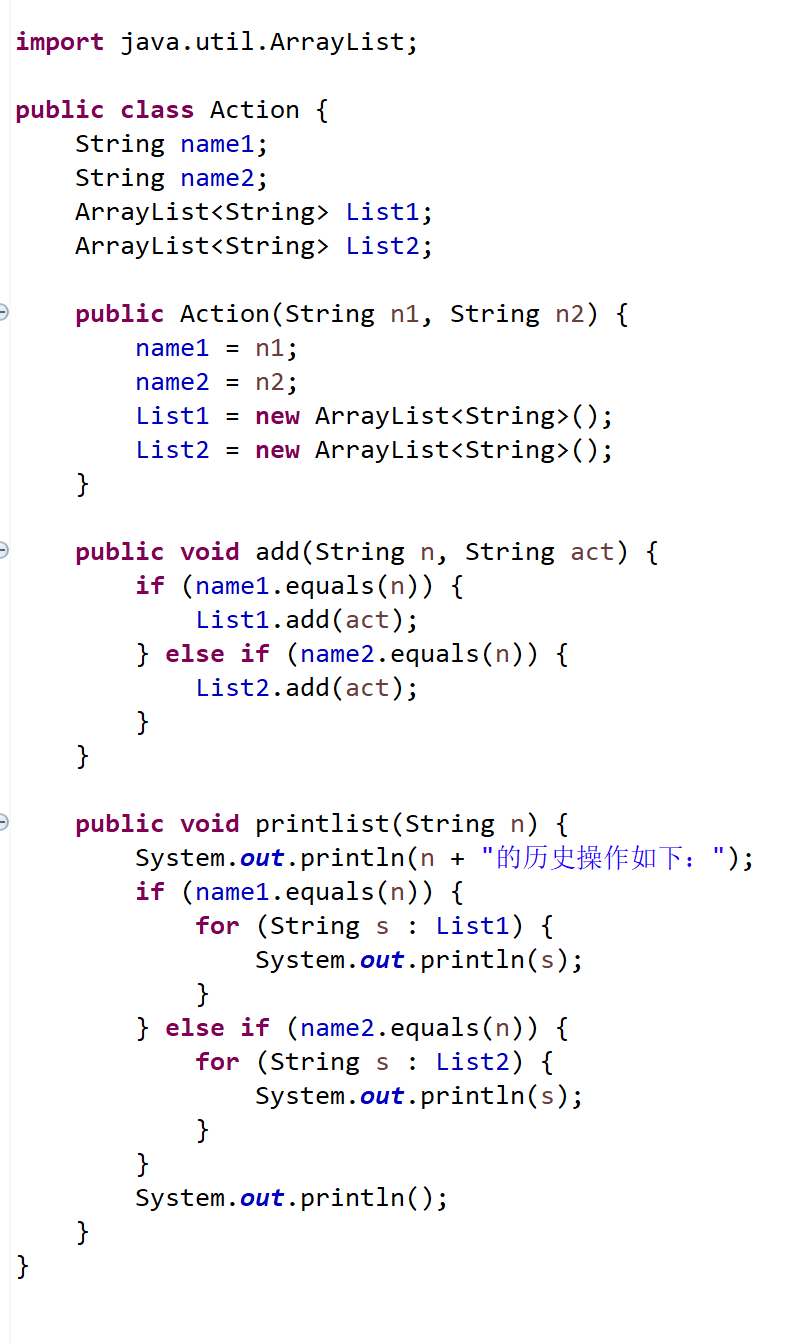
### 3.3.1.4 Player

PlayerADT中存储了单个玩家的姓名、持有的棋子以及如果是围棋，下一枚棋子的编号信息。在对玩家的初始化中，自动完成了游戏类型的初始化例如国际象棋中，玩家的棋子会对应的放置在棋盘两侧。



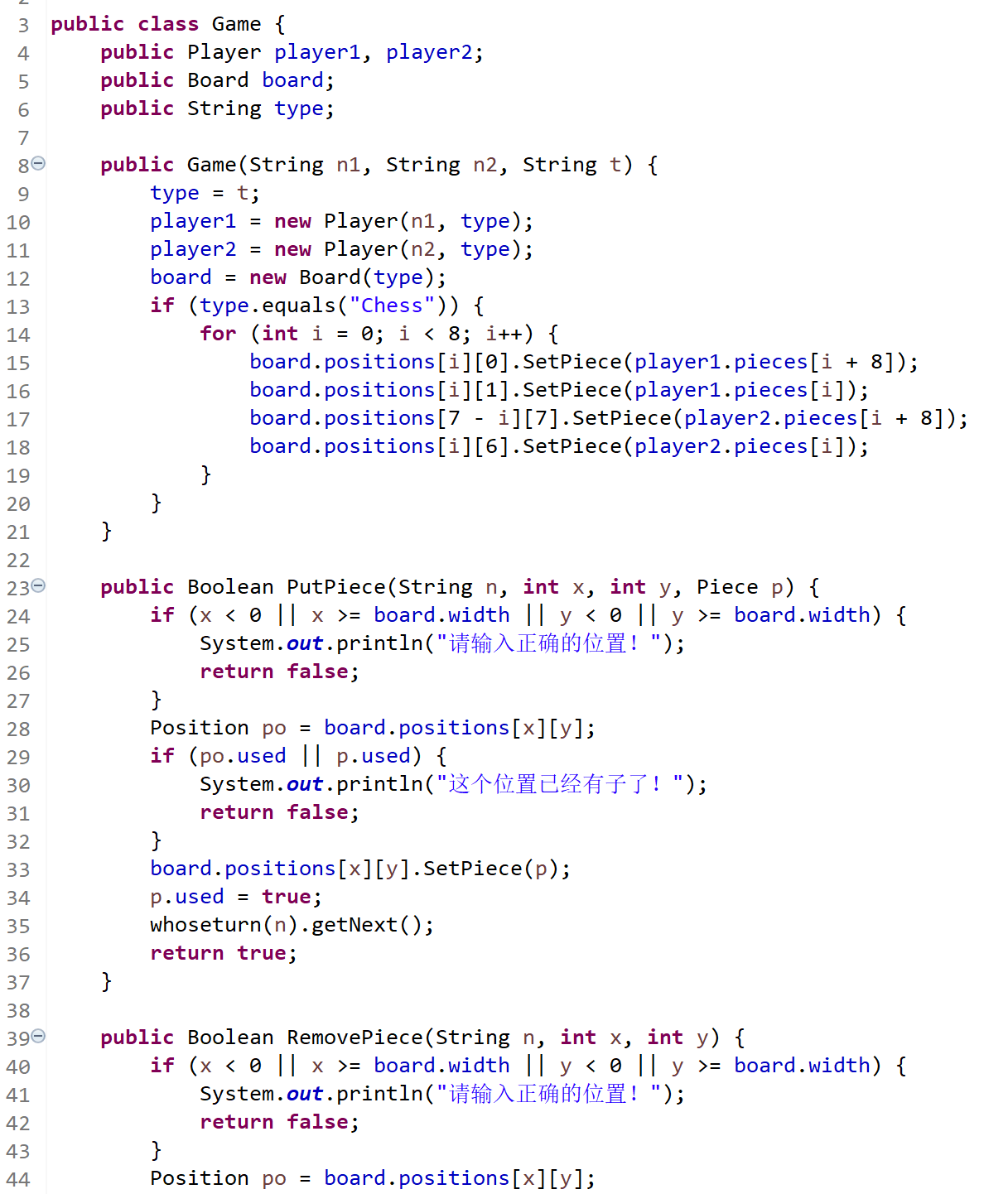
### 3.3.1.5 Action

ActionADT是一个用于记录游戏历史的类，其中保存了两位玩家的姓名和分别的历史操作的字符串信息。并提供了一个打印算法，可以在游戏结束后输出两位玩家的历史操作信息。



### 3.3.1.5 Game

GameADT是一个较为庞大的系统，其中存储了两个玩家信息，一个棋盘信息，和一个Action。提供了初始化游戏，提子，落子，移动棋子这几项游戏核心内容。本质上而言，它承担了游戏的玩法设计，可以根据函数调用来实现游戏的进行。

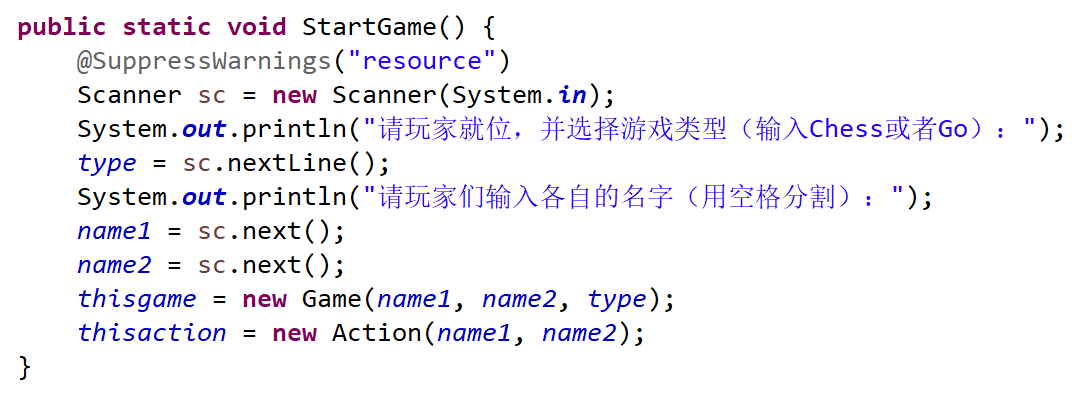


### 主程序MyChessAndGoGame设计/实现方案

这个主程序中包含StartGame()、Chesstime()和Gotime()三个用户交互的核心程序。下面一一详细描述：

#### **StartGame()**

这一部分的作用是用户登录，以及生成所要求的游戏的过程，由于无论是国际象棋还是围棋，都可以通过传输一段字符串给Game完成自动的生成，初始化这一操作也在这一步完成。



#### **Chesstime()**

当记录到用户信息以及游戏内容信息为Chess时调用该函数，国际象棋中只有“移动棋子”这一操作，所以提示信息中除了要求的查看棋盘某位置信息、查看棋盘上双方棋子个数等信息外只需要提示移动棋子，选择使用字符串输入来完成信息获取。

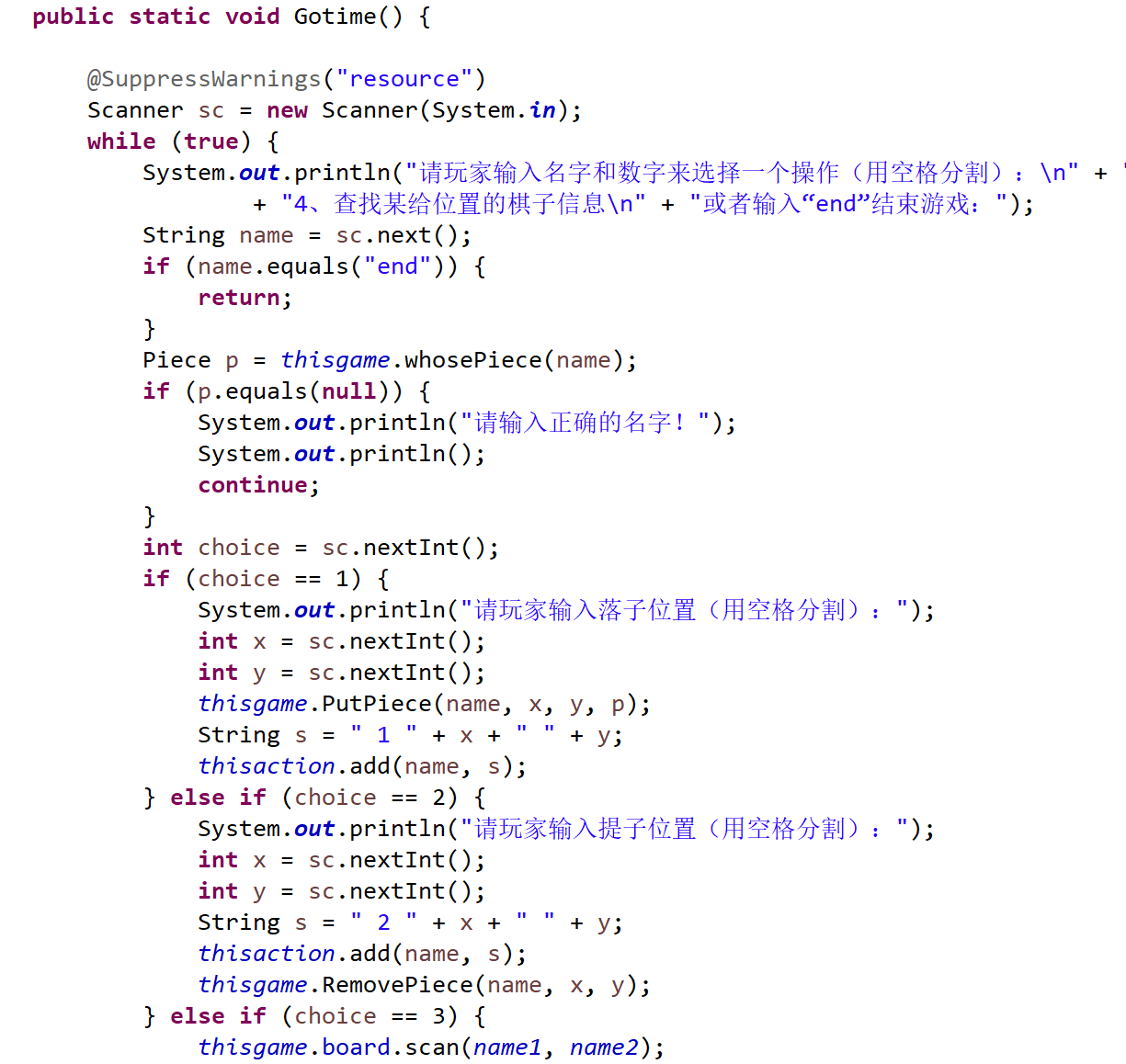
代码如下：



#### **Gotime()**

当记录到用户信息以及游戏内容信息为Go时调用该函数，围棋中有“提子”和“落子”两种操作，提示信息中除了要求的查看棋盘某位置信息、查看棋盘上双方棋子个数等信息外还要有“提子”和“落子”两项，内容较多，所以选择在输入当前玩家名字后输入一个数字来选择后续操作。

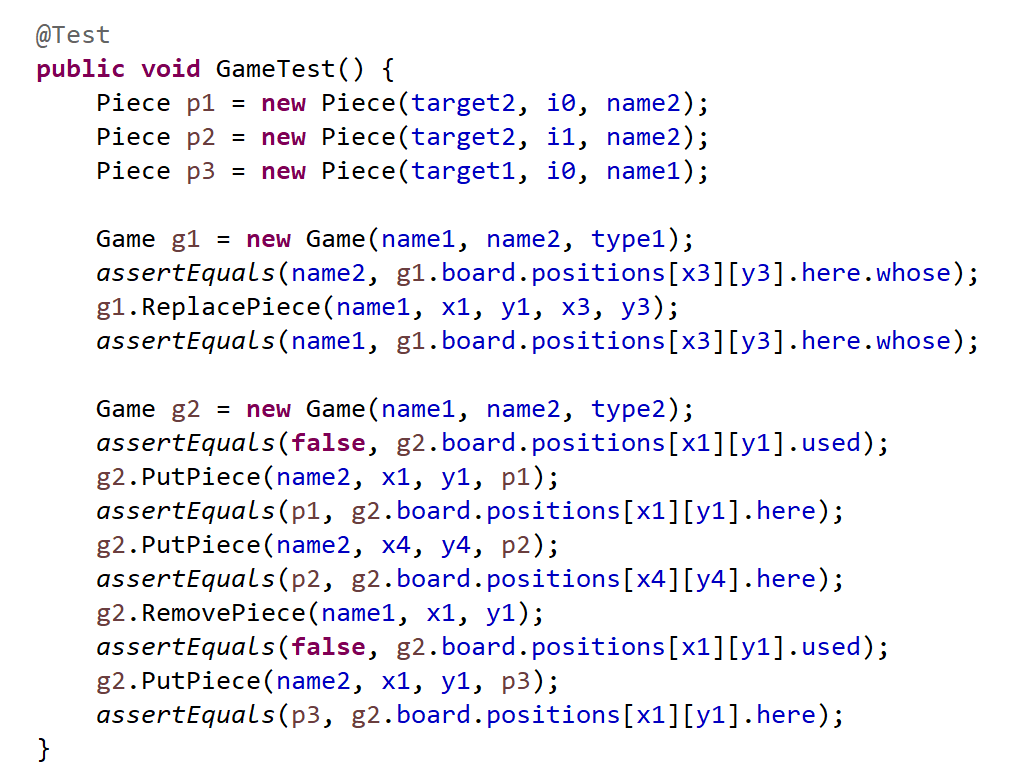
代码如下：

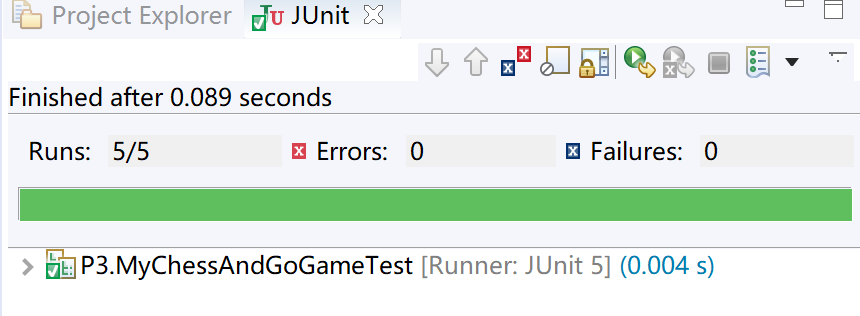


### 3.3.3 ADT和主程序的测试方案

测试文件主要通过自己设置各种字符串、棋子、位置信息模拟下棋的过程，针对Game、Player、Board、Position、Action做了测试样例。

代码以及运行结果如下：





# 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 |
| 2019-04-02 | 13:00-16:30 | 编写ConcreteEdgesGraph函数并进行测试 | 按计划完成 |
| 2019-04-03 | 14:00-16:00 | 编写ConcreteVerticesGraph函数并进行测试 | 按计划完成 |
| 2019-04-04 | 13:45-15:30 | 编写GraphPoet函数 | 遇到困难，延时完成 |
| 2019-04-05 | 13:00-17:00 | 编写FriendshipGraph函数 | 按计划完成 |
| 2019-04-06 | 13:00-19:00 | 编写问题3的棋类游戏函数 | 按计划完成 |
| 2019-04-07 | 10:45-11:30 | 编写棋类游戏的test文件 | 按计划完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| For循环遍历和remove的冲突问题 | 利用iterator来代替list进行循环遍历 |
| 如何编写简单图的最短路径算法 | 先转化为邻接矩阵，再通过学过的方法寻找最短路径 |
| 棋类游戏编写没有头绪 | 与同学交流、翻看去年学习Python时所做的贪吃蛇找寻灵感。 |

# 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

Java语言是面向对象的典型编程语言。这次实验遇到了许多java语言细节上的困难，但是经过查阅书籍、上网搜索或者寻求助教以及同学的帮助之后都成功解决了。

这次实验复习并巩固了以前学习的编程知识，加强了面向ADT编程的计算思维。还学会了各种eclipse的操作方法，有助于java编程。

## 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？

前者是从底层开始设计编程内容，而后者是从应用场景直接设计。由于实验内容简单，这次实验没有彻底体现前者的优势，但当编程规模巨大的时候，前者的思路会更为信息。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型编程会有更多限制，但是当编程逐渐深入，泛型的适用范围广的优势。

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？

优势在于可以根据规约依次设计测试用例，思路清晰。我能够适应。

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？

可以简化代码思路。也简化代码量，不必为了相似的内容写复数次类似的代码。

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？

是。有了P1、P2做基础，我比较顺利的完成了P3，自己测试的时候也挺有趣的。

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？

可以使编程工作少出错，减少失误，这当然也是今后需要坚持的。

1. 关于本实验的工作量、难度、deadline。

这次的内容难度不大，P2工作量不是很大，P1、P3是ADT设计工作量较大。deadline挺好的，有三周的时间可以做Lab1。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

能感受到这门课对学生的要求很高，学习难度也非常大，不过我相信在这门课的磨练之下，我的java编程能力会有很大进步。