

**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 | 张乙 |
| 学号 | 1180301022 |
| 班号 | 1803010 |
| 电子邮件 | [1430268665@qq.com](mailto:1430268665@qq.com) |
| 手机号码 | 15839452898 |

**目录**

[1 实验目标概述 1](#_Toc29325498)

[2 实验环境配置 1](#_Toc29325499)

[3 实验过程 2](#_Toc29325500)

[3.1 Poetic Walks 2](#_Toc29325501)

[3.1.1 Get the code and prepare Git repository 2](#_Toc29325502)

[3.1.2 Problem 1: Test Graph <String> 3](#_Toc29325503)

[3.1.3 Problem 2: Implement Graph <String> 4](#_Toc29325504)

[3.1.3.1 Implement ConcreteEdgesGraph 4](#_Toc29325505)

[3.1.3.2 Implement ConcreteVerticesGraph 6](#_Toc29325506)

[3.1.4 Problem 3: Implement generic Graph<L> 10](#_Toc29325507)

[3.1.4.1 Make the implementations generic 10](#_Toc29325508)

[3.1.4.2 Implement Graph.empty() 11](#_Toc29325509)

[3.1.5 Problem 4: Poetic walks 12](#_Toc29325510)

[3.1.5.1 Test GraphPoet 12](#_Toc29325511)

[3.1.5.2 Implement GraphPoet 13](#_Toc29325512)

[3.1.5.3 Graph poetry slam 14](#_Toc29325513)

[3.1.6 Before you’re done 15](#_Toc29325514)

[3.2 Re-implement the Social Network in Lab1 16](#_Toc29325515)

[3.2.1 FriendshipGraph类 16](#_Toc29325516)

[3.2.2 Person类 19](#_Toc29325517)

[3.2.3 客户端main() 19](#_Toc29325518)

[3.2.4 测试用例 21](#_Toc29325519)

[3.2.5 提交至Git仓库 22](#_Toc29325520)

[3.3 Playing Chess 23](#_Toc29325521)

[3.3.1 ADT设计/实现方案 23](#_Toc29325522)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 26](#_Toc29325523)

[3.3.3 ADT和主程序的测试方案 32](#_Toc29325524)

[4 实验进度记录 35](#_Toc29325525)

[5 实验过程中遇到的困难与解决途径 35](#_Toc29325526)

[6 实验过程中收获的经验、教训、感想 36](#_Toc29325527)

[6.1 实验过程中收获的经验和教训 36](#_Toc29325528)

[6.2 针对以下方面的感受 36](#_Toc29325529)

# 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；

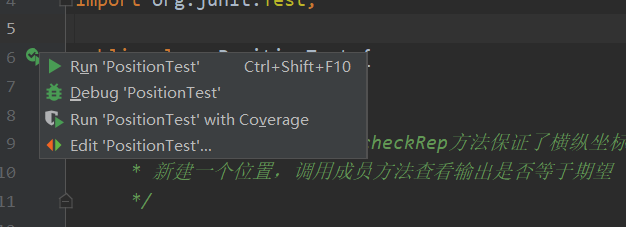
使用 ADT 及其实现，为应用问题开发程序；

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

IDEA本身支持运行测试代码时，显示被测试代码的覆盖率。

如下图Run with Coverage



GitHub Lab2仓库的URL地址：

https://github.com/ComputerScienceHIT/Lab2-1180301022

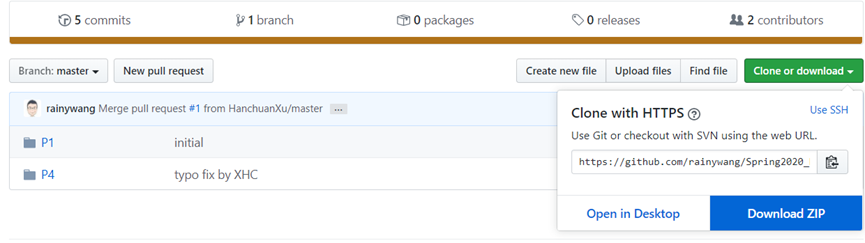
# 实验过程

## Poetic Walks

首先对图的ADT进行两种实现（从边和顶点出发），然后选择一种实现的ADT，根据输入的文件构建语料库，再利用构建的语料库，对输入字符串进行插入操作，并返回修改后的字符串。

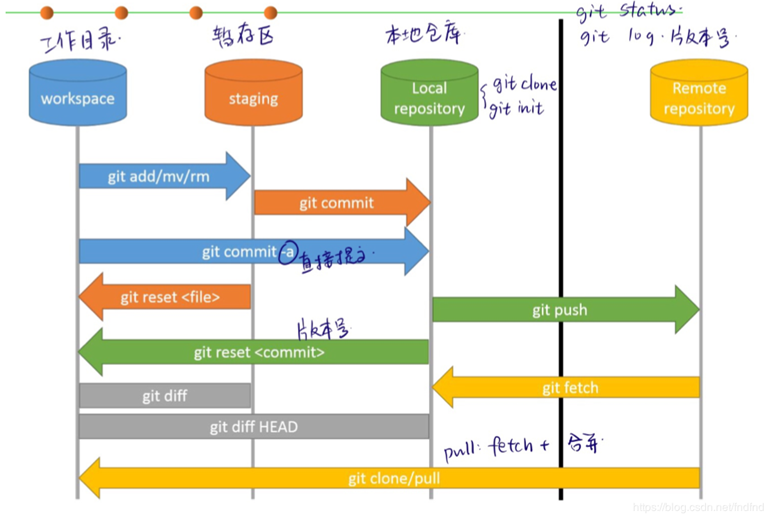
### Get the code and prepare Git repository

从GitHub获取该任务的代码:



在本地创建git仓库：在gitbash中输入命令git clone <https://github.com/ComputerScienceHIT/Lab2-1180301022>，将GitHub Classroom中的库链接到本地。

使用git管理本地开发：通过git pull或者git clone获取远程仓库的文件，通过git add、git commit、git push将本地仓库的文件上传到远程仓库。



### Problem 1: Test Graph <String>

测试图的add方法：

向图中添加一个点，并检查该点是否在图中。

再尝试向图中添加相同点，发现无法加入重复点。

测试图的set方法：

1.向图中添加两点，并设置两点之间的边值。第一次设置时返回0。

2.第二次设置时，返回第一次设置的值。

3.在set方法中输入未加入点，set结束后，查看是否在图中。

测试图的remove方法：

首先建立图 c-1-> a -2->b

删除a点，检查是否返回true，检查和a有关的边是否都被删除。

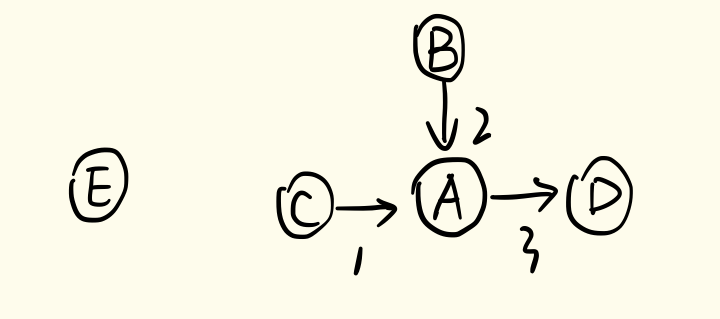
重复删除a点，检查是否返回false。

测试图的vertices方法：

向图中加入点，测试图中的点是否和加入的点相同，数目是否相等。

测试图的sources方法、targets方法：

构建如下图并检查各点的sources、targets是否符合预期。



具体的测试结果，在图的两种实现的测试中体现。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

// Abstraction function:

// AF(vertices, edges) = 含有顶点vertices、边edges的图

// Representation invariant:

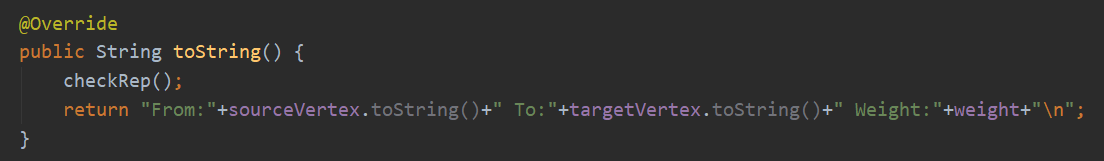
// edges中，所有的起点和终点都包含在vertices中

// Safety from rep exposure:

// 所有成员域都用private修饰

1. Edge类的实现

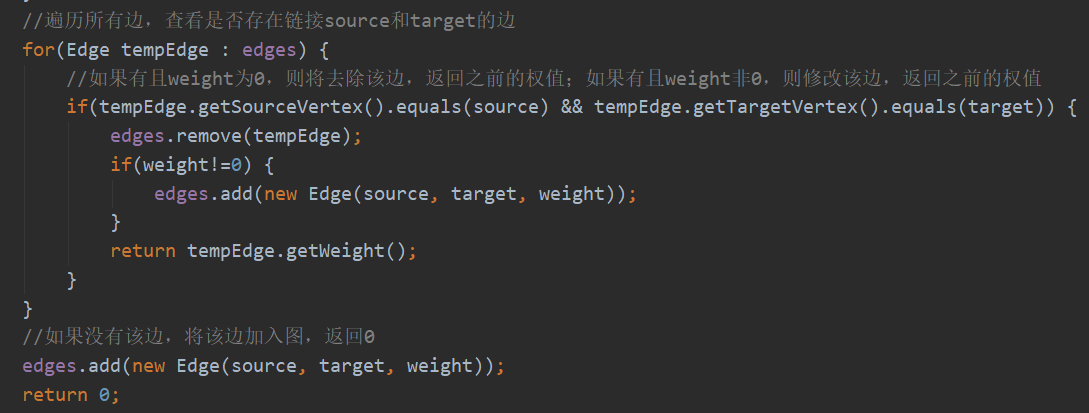
Edge类的成员域分别记录起始点、终止点和边权值。在构造方法中对边的实例进行设置。get方法的实现非常简单，不再赘述。此处简要介绍toString方法的实现。实验要求override该方法，提高可读性。因为每条边的结构比较简单，所以直接构造字符串返回即可。



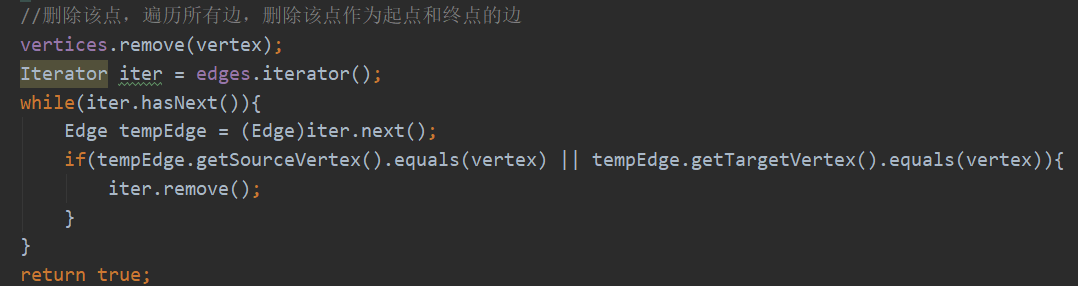
1. ConcreteEdgesGraph类中的方法实现

add：由于顶点是String类型，可以直接用vertices.contains(input)检测该点是否已经被加入到图中。如果已加入返回false；如果未加入则将输入点加入集合，返回true。

set：首先检测输入点是否已经加入图中。如果存在未在图中的点，则先将点加入到图中。遍历所有边，查看是否存在链接source和target的边。如果有且weight为0，则将去除该边，返回之前的权值；如果有且weight非0，则修改该边，返回之前的权值。如果没有该边，将该边加入图，返回0。



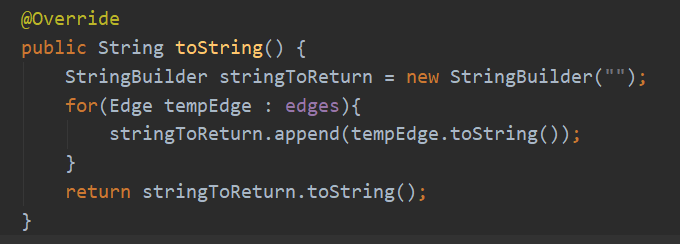
remove：如果输入点不在图中，直接返回false。否则，遍历边的集合，将起点或终点为待删除的点的若干边删除。注意到此处必须使用迭代器。



vertices：直接返回点集的复制即可。

sources、targets：两个方法本质相同。这里以sources为例。新建一个Map实例用于返回。首先检测输入点是否在图中，如果不在则返回空Map。否则，遍历边集edges，如果某条边的目标点和输入点相同，则将该边的起始点和权值记录到要返回的Map中。最后返回即可。

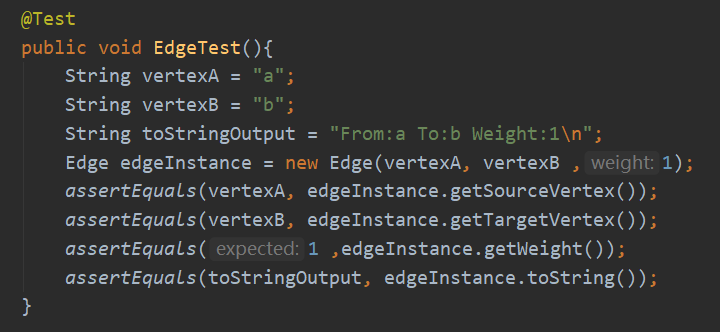
toString：因为涉及到字符串的频繁改变，所以使用StringBuilder对象完成字符串拼接。遍历图中的每条边，分别调用每条边的toString方法，并记录结果。最后调用StringBuilder.toString()，返回字符串。



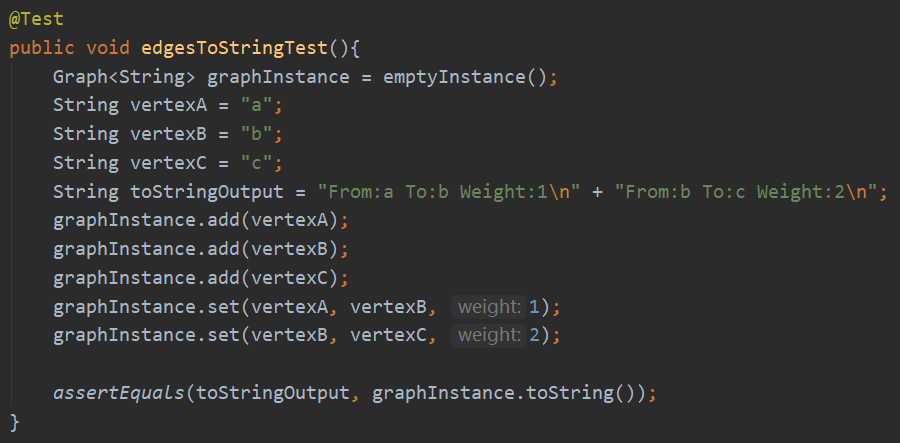
1. ConcreteEdgesGraphTest

其中，ConcreteEdgesGraph继承自Graph接口的方法的测试已经在上个问题中说明。这里只需测试Edge类的方法，和ConcreteEdgesGraph类的toString()方法。

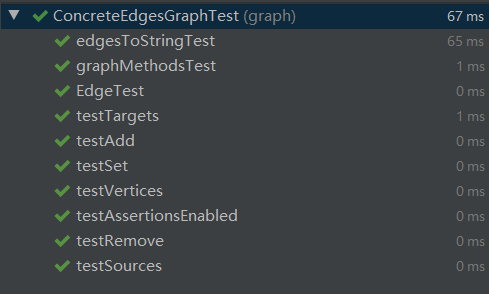
测试Edge类方法，新建一条边，测试其属性值是否等于输入，并测试toString()是否输出正确。



测试ConcreteEdgesGraph类的toString()方法，向新建的图中添加若干点和边，调用toString()方法查看输出是否正确。



测试结果：



#### Implement ConcreteVerticesGraph

// Abstraction function:

// AF(vertices) = 由vertices中的点和边信息构成的图

// Representation invariant:

// vertices!=null

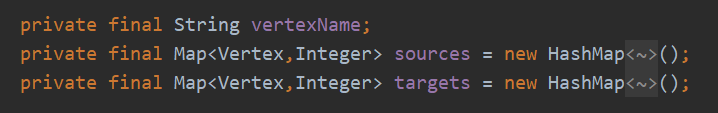
// Safety from rep exposure:

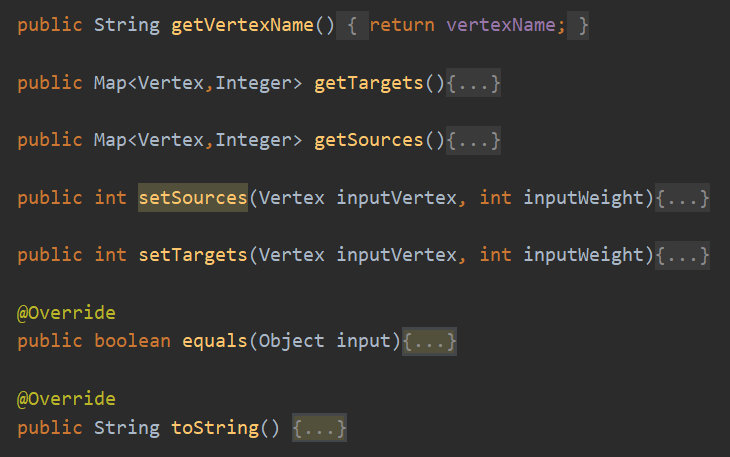
// 成员域用private修饰

// 返回时采用防御拷贝

1. Vertex类的实现

为了便于图的各种操作，我在Vertex类中设置了一下成员域和成员方法：





其中，setSources方法可以设置指向该点的边的情况，如增加一条边，改变一条已存在边的权值，删除一条边。但是需要保证传入的点已经在图中，且输入权值为非负整数。setTargets与之同理。

在此我重写了equals方法，使得两个Vertex实例如果拥有相同的vertexName，在集合中即视为同一对象，防止同名顶点重复加入图中。

toString方法实现如下：遍历该顶点的sources和targets，并将对应顶点和权值记录到一个StringBuilder实例中，最后通过StringBuilder.toString()输出字符串。

1. ConcreteVerticesGraph的方法实现

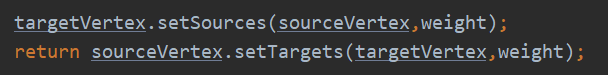
由于各个方法都是以字符串为参数调用，所以通过遍历vertices并检测vertexName是否为输入参数，确定顶点是否已经在图中。

add：

检测输入顶点是否已在图中，如果是返回false，如果不是，添加该点并返回true。

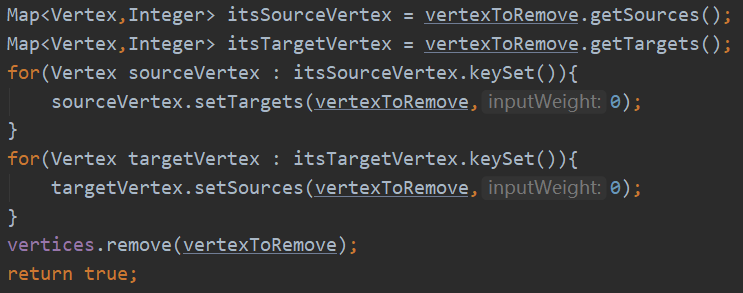
set：

遍历vertices，寻找输入点是否存在。如果vertices存在输入点，则获取该点的引用；如果不存在，则需新建点并加入到图中。通过Vertex类中的setSources方法和setTargets方法对输入两点的边进行设置。如果两点之间存在边，则在修改后返回修改之前的权值；否则返回0。



remove：

根据输入字符串，遍历vertices。如果不存在该点之间返回false。如果该点存在，则遍历该点的sources和targets，对于和该点连接的点，分别调用setTargets或者setSources，将边的权值设置为0（即是删除该边）。执行完后，在vertices中删除待移除点，返回true。



vertices：

新建一个Map，将vertices中的信息复制并返回。

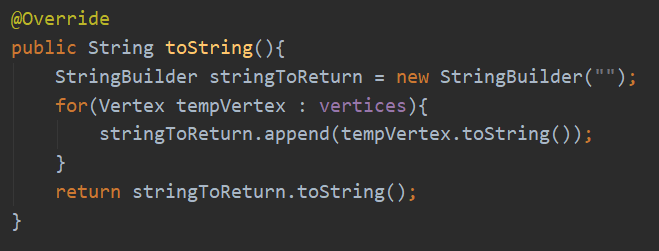
sources、targets：

两方法同理。以sources为例。

首先查看是否是图中的点，如果不是直接返回空map。如果是，新建一个Map，遍历该点的sources，并将其信息复制到新建的Map中，然后返回即可。

toString：

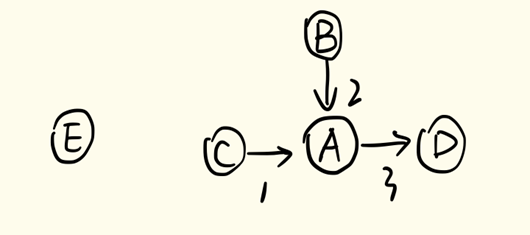
因为涉及到字符串的频繁改变，所以使用StringBuilder对象完成字符串拼接。遍历图中的每条边，分别调用每条边的toString方法，并记录结果。最后调用StringBuilder.toString()，返回字符串。



1. ConcreteVerticesGraphTest

其中，ConcreteVerticesGraph继承自Graph接口的方法的测试已经在上个问题中说明。这里只需测试Vertex类的方法，和ConcreteVerticesGraph类的toString()方法。

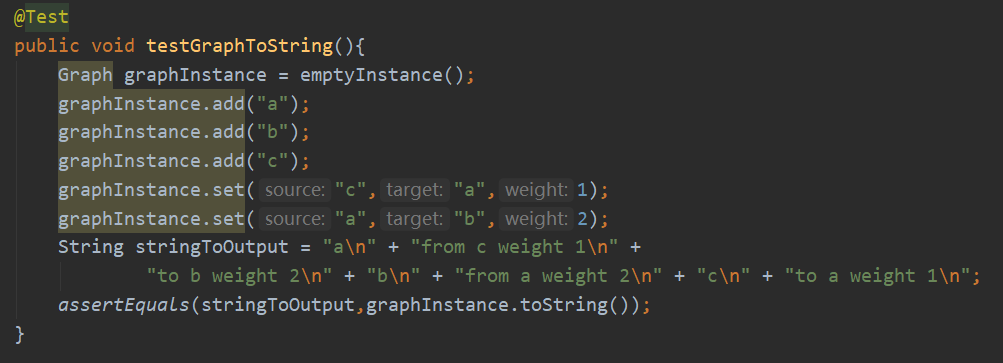
测试Vertex类的方法：



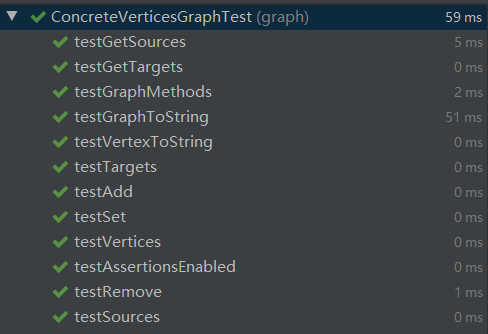
仍然采用该图作为测试用图。此时并不调用emptyInstance()，因为现在是在测试Vertex类，不应和ConcreteVerticesGraph类相交。建立该图后，对每个顶点分别调用getSources、getTargets、toString、setSources、setTargets方法，并检查输出是否符合预期。

测试ConcreteVerticesGraph类的toString()方法：

调用emptyInstance()并建立该图，然后调用ConcreteVerticesGraph.toString()，检查输出是否和预期相同。



测试结果：

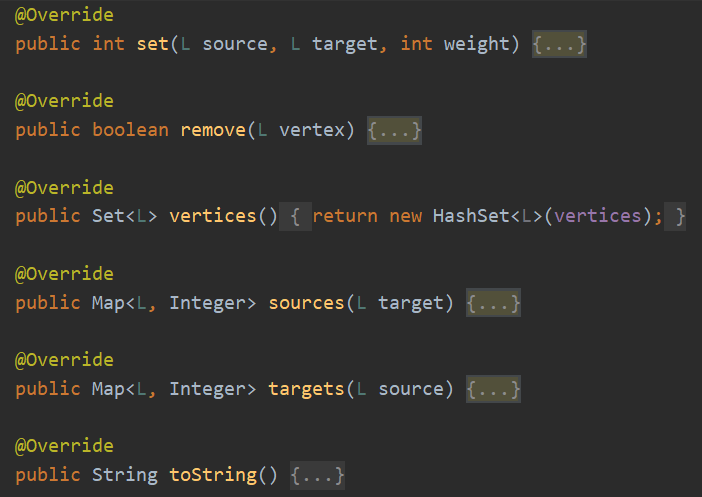


### Problem 3: Implement generic Graph<L>

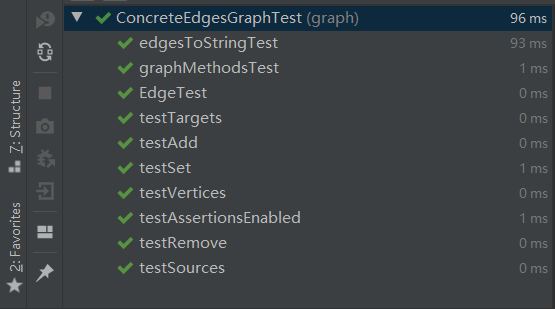
#### Make the implementations generic

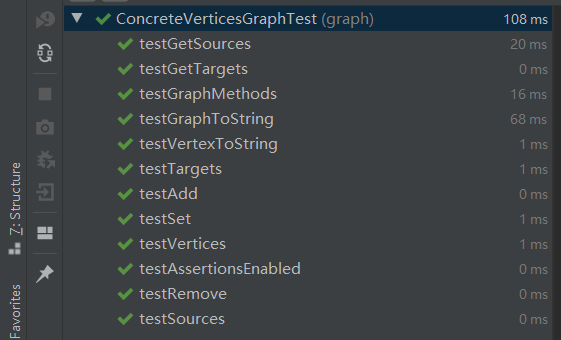
将上述两个类中的String标识符修改为L。根据实验报告，顶点的标识符L是不可变并且可以使用equals判断是否相等，所以在确认是否存在目标点时可以遍历点集并用equals方法判断。

ConcreteEdgesGraph、ConcreteVerticesGraph类修改后的方法：



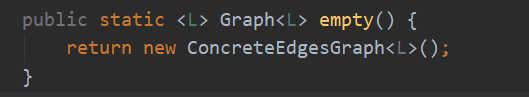
修改后运行ConcreteEdgesGraphTest、ConcreteVerticesGraphTest的结果通过，说明将泛型占位符L替换成String的运行结果是正确的。



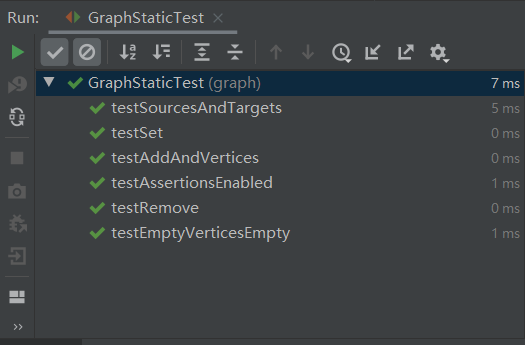


#### Implement Graph.empty()

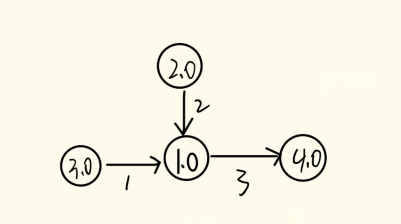
选取ConcreteEdgesGraph类构造实例。将empty方法修改成图中即可。



为了测试改写泛型是否成功，在GraphStaticTest测试类中将占位符L替换为Double类（Doub类是immutable的），改写ConcreteEdgesGraphTest类中的方法即可。测试结果如下。



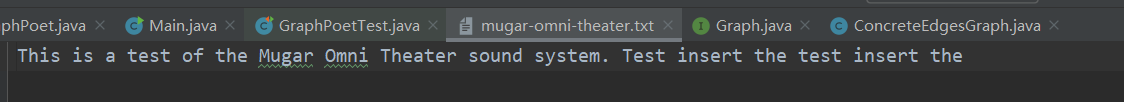
附：testSourcesAndTargets的测试用例

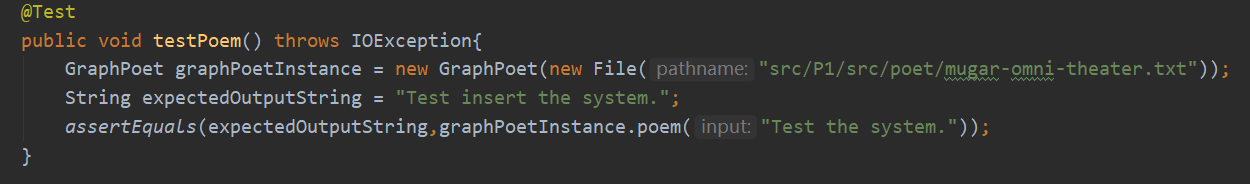


### Problem 4: Poetic walks

#### Test GraphPoet

通过修改所给测试用例，对poem方法进行测试：





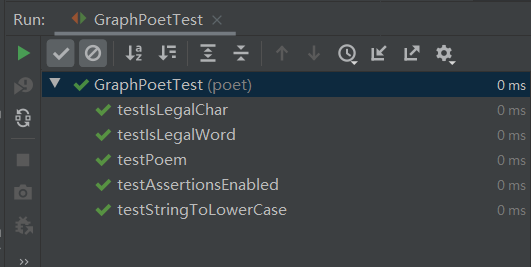
辅助函数的测试：

testIsLegalWord测试是否能正确检测输入字符串符合word的要求。

testIsLegalChar测试是否能正确检测输入字符符合word字符的要求。

testStringToLowerCase测试是否能将输入字符串中的大写字母转化成小写字母。

补全下述问题后的测试结果：



#### Implement GraphPoet

// Abstraction function:

// AF(graph) = 将输入语料库构成graph，并利用它完成诗句创作

// Representation invariant:

// graph != null

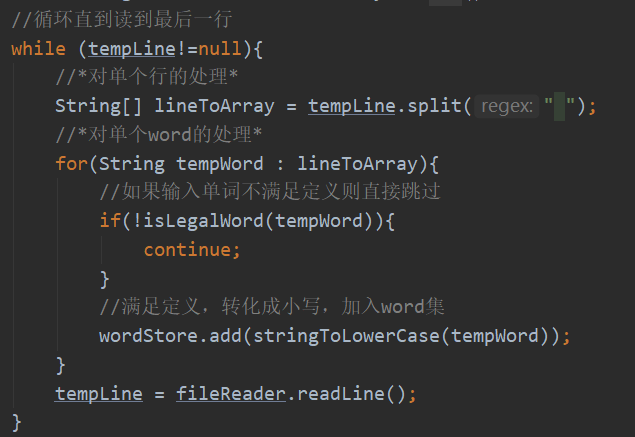
// Safety from rep exposure:

// 成员域用private修饰

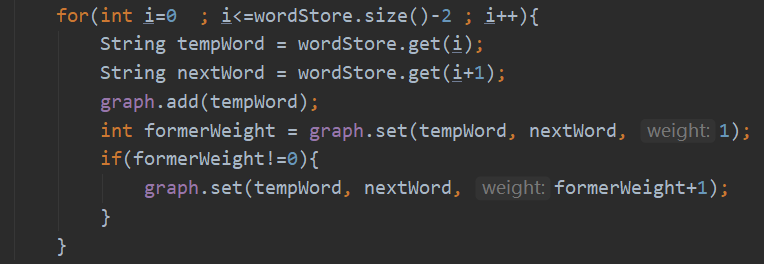
// 返回值类型都是不可变类

1. 构造方法：

如果无法打开输入文件，则抛出异常。如果没有抛出异常，按行读入输入文件，用String.split方法将每行拆分成单个word。对于单个的word，首先检测输入是否合法，如果输入非法则考察下个word；如果输入合法，调用stringToLowerCase将word中的大写字母转化成小写（如果存在），存入到字符数组中。

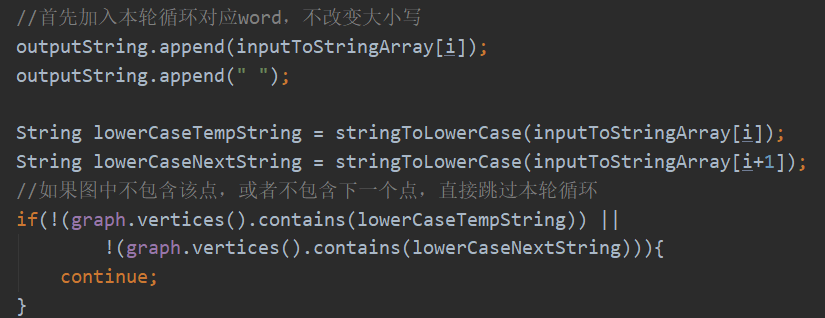


下面利用所得字符串数组，构造poetGraph。每轮循环考察数组中的一个word。将该字符串作为顶点加入到graph中。首先将该word和下一个word的边设置权值为1，如果之前不存在边，则完成操作（建立权值为1的边）；如果之前存在边，重新设置该边，将边的权值加1。



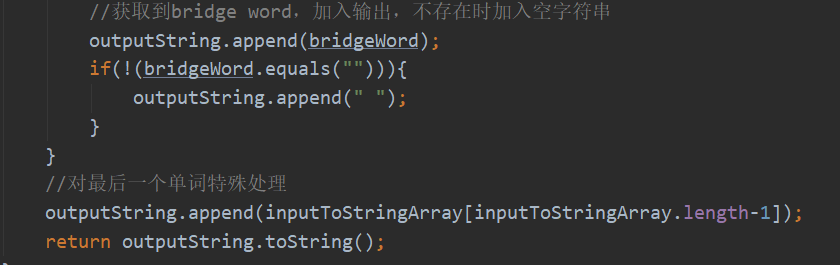
1. poem方法

使用String.split方法将输入字符串转化成字符串数组。在循环中考察除最后一个的所有word。首先将该word加入到输出字符串中。如果该word或者邻接的下个word在构建的图中不存在，则直接考察下个word。



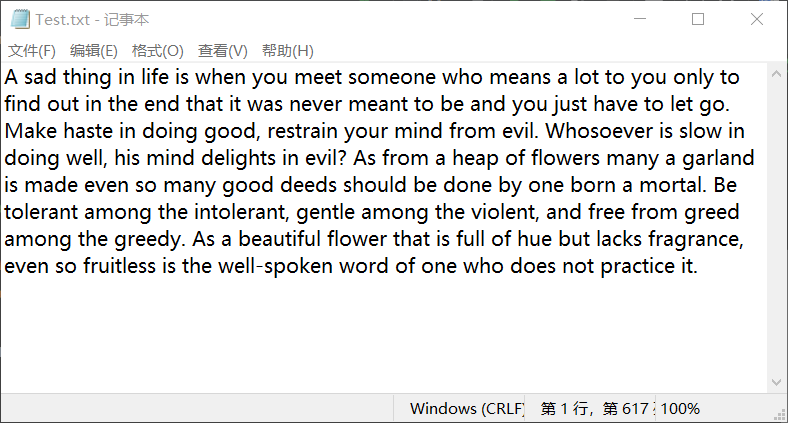
如果存在，将该word转化成小写，遍历该word的所有targets的targets（假设得到nextStringTargets），如果和邻接的下个word转化成小写的形式相等，则记录权值和bridgeWord。循环结束后，选择权值最大的bridgeWord加入输出。

最后，特殊处理数组中的最后一个字符串，直接加入输出字符串即可。

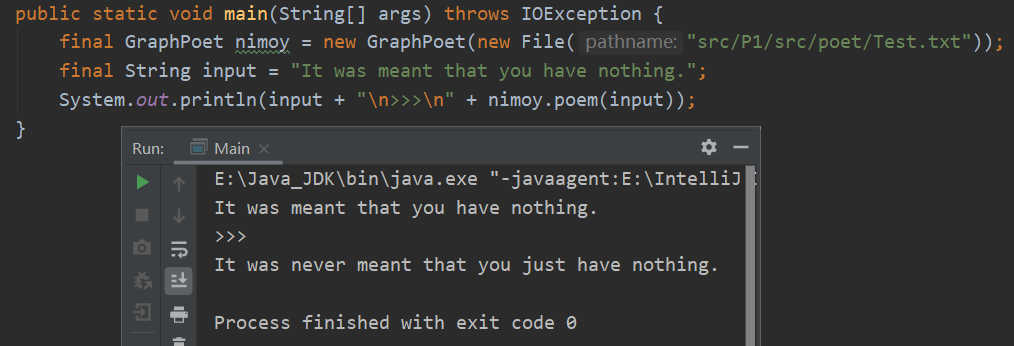


#### Graph poetry slam

选择的语料库：

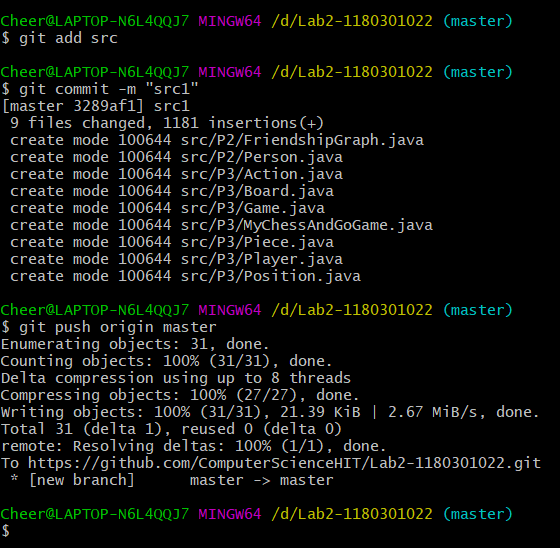


输入和输出：

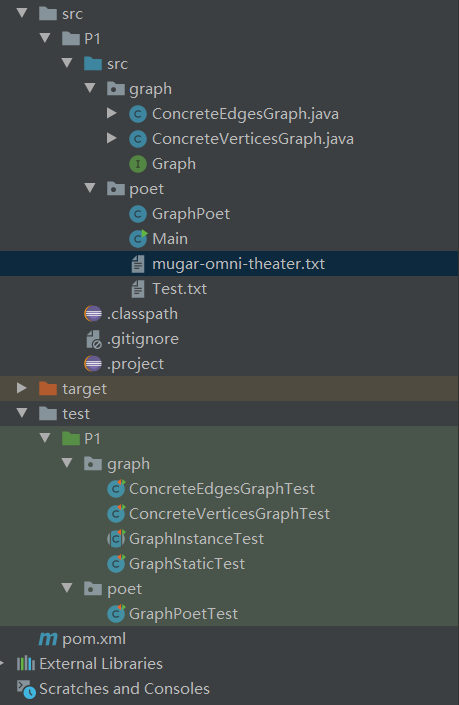


### Before you’re done

通过Git提交当前版本到GitHub上你的Lab2仓库：



项目的目录结构树状示意图：



## Re-implement the Social Network in Lab1

用Problem1实现的Graph，按照Lab1的要求重新实现FriendshipGraph类。这里我选择ConcreteEdgesGraph作为Graph的内部实现。

### FriendshipGraph类

Abstraction function:

AF(vertices, edges, isCounted) = 含有顶点vertices、边edges的图，且每个点都有考察标记

Representation invariant:

true

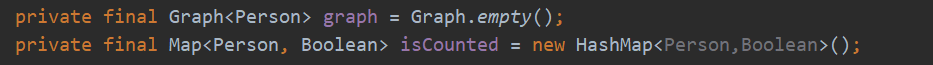
Safety from rep exposure:

所有成员域都用private修饰

使用防御拷贝

成员域：

其中isCounted在getDistance方法中用来标志某点是否已经被考察过。初始值都为false，即未被考察。

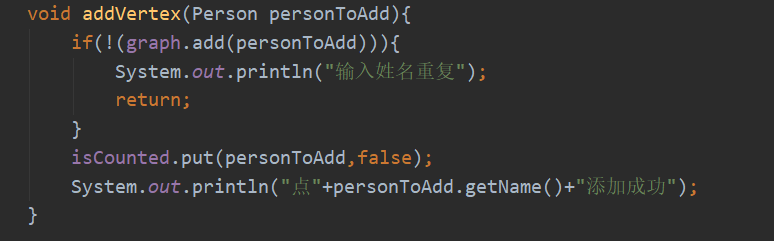


addVertex方法：

向图中加入顶点，如果输入重复点，则不重复加入，并输出错误信息。

如果非重复点，则加入图中，并在标志是否考察的Map（getDistance方法中用到）中添加项，并赋值false，输出提示信息。

因为在Person类中重写了hashCode和equals方法，所以Person的实例之间是可以用equals方法直接比较的，如果姓名相同则视为相同的点。



addEdge方法：

向图中添加边，如果输入点存在不在图中点，则输出错误信息并退出方法。

如果输入点均在图中，则按输入设置边，提示成功信息。

getDistance方法：

返回从起始点到终止点的最短距离。如果输入点存在不在图中的点，或者两点之间不可达，则输出-1；如果两点在图中可达，输出最短距离。



setIsCountedFalse方法：

将点是否考察过修改成否（false）。

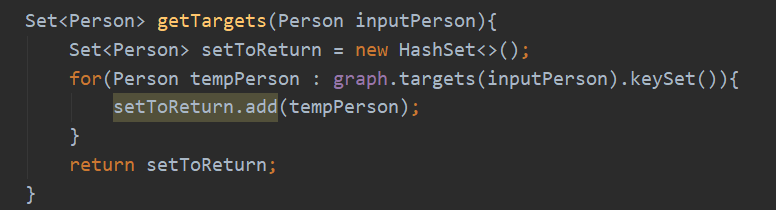
每次调用getDistance方法前都需要进行的初始化操作。

getPersons方法：

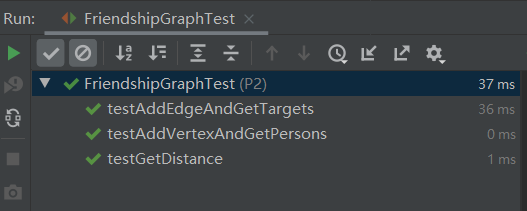
返回已经加入图中的点。

getTargets方法：

返回从输入Person出发的一条边，到达的点的集合。



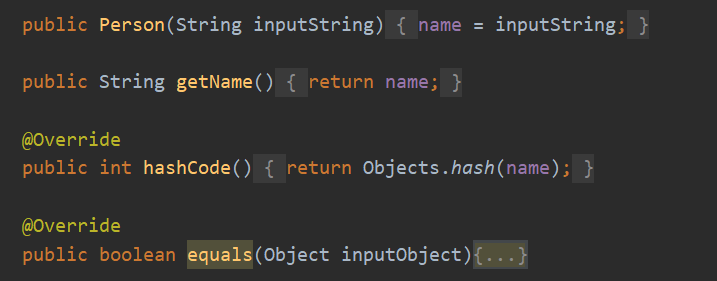
测试结果：



### Person类

相对Lab1，我删减了Person类的标志变量（转化成在FriendshipGraph中建立Map进行标记），只保留了姓名的成员域。

重写了hashCode和equals方法，方便在Person实例间使用equals进行比较，判断是否重复加入点。



// Abstraction function:

// AF(name) = 姓名为name的人

// Representation invariant:

// name!=""

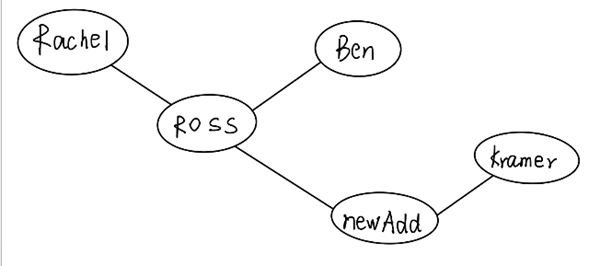
// Safety from rep exposure:

// 所有成员域都用private修饰

// String类是不可变类，可以直接返回而不产生暴露

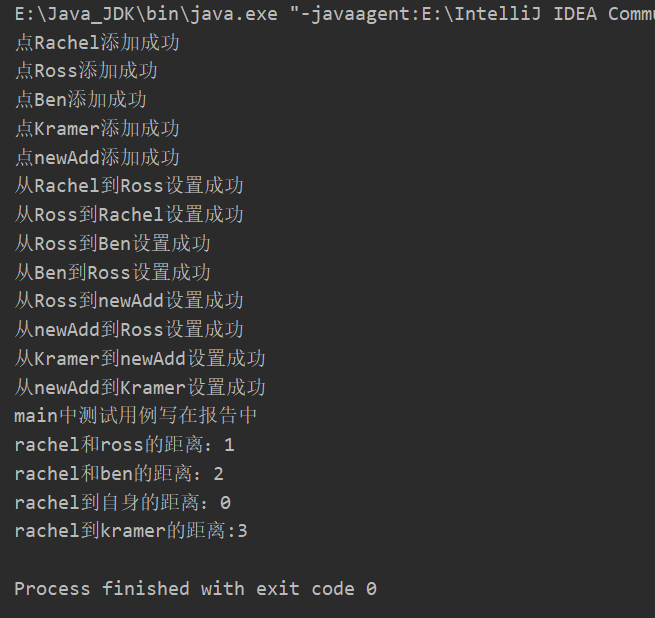
### 客户端main()

沿用了Lab1的客户端用例。





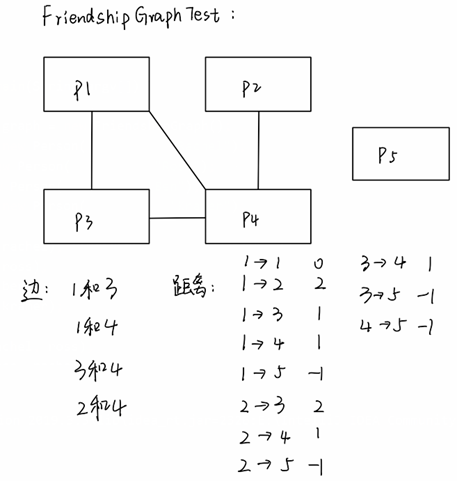
运行结果：



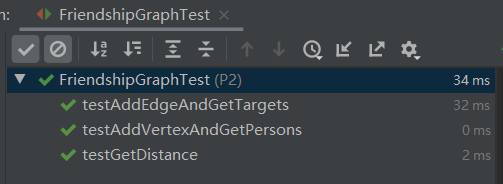
### 测试用例

测试思路：构建一个结构已知的图，用assert方法证明FriendshipGraph类中每个方法的正确性。

测试用例设计：

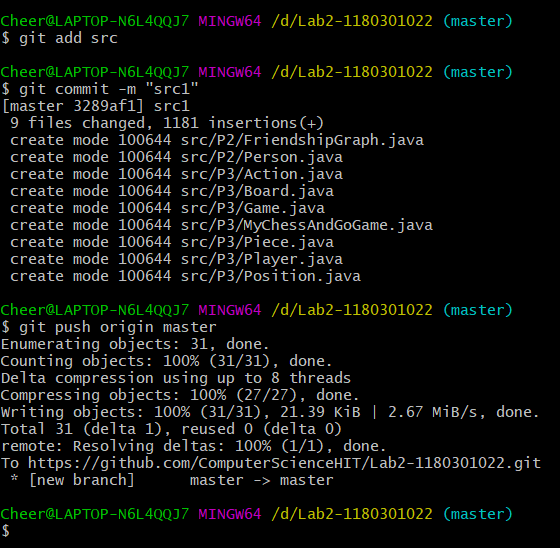


测试结果：

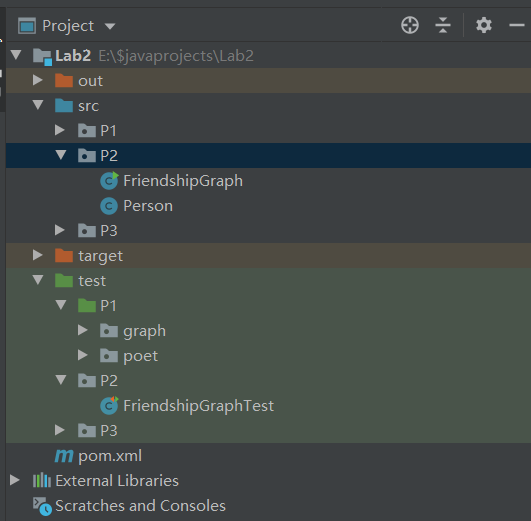


### 提交至Git仓库

如何通过Git提交当前版本到GitHub上你的Lab3仓库。



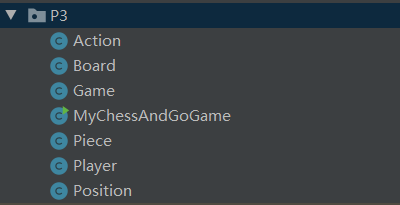
在这里给出你的项目的目录结构树状示意图。



## Playing Chess

### ADT设计/实现方案

设计的类：



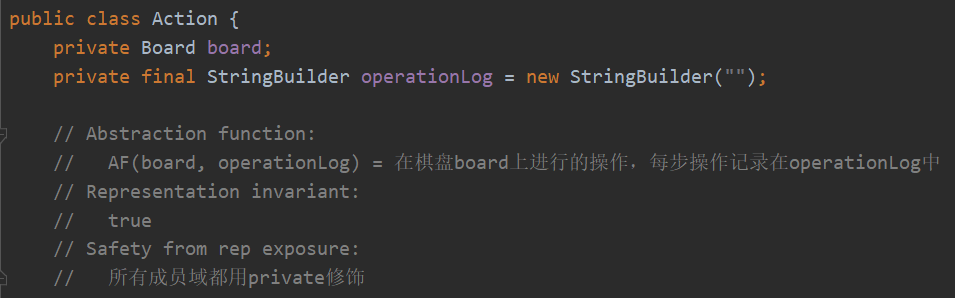
1. Action类：

棋类游戏的动作类。

属性：棋盘、操作历史记录。

方法：放子、移动、吃子、提子、查询位置占用情况、计算棋盘上的棋子数、获得历史记录。

是可变类型。



1. Board类：

棋类游戏的棋盘

属性：棋盘大小、棋盘上棋子的信息

方法：构造方法、获取所有棋子信息、加入棋子、删除棋子、检测位置合法性、计算棋子数目等

是可变类。



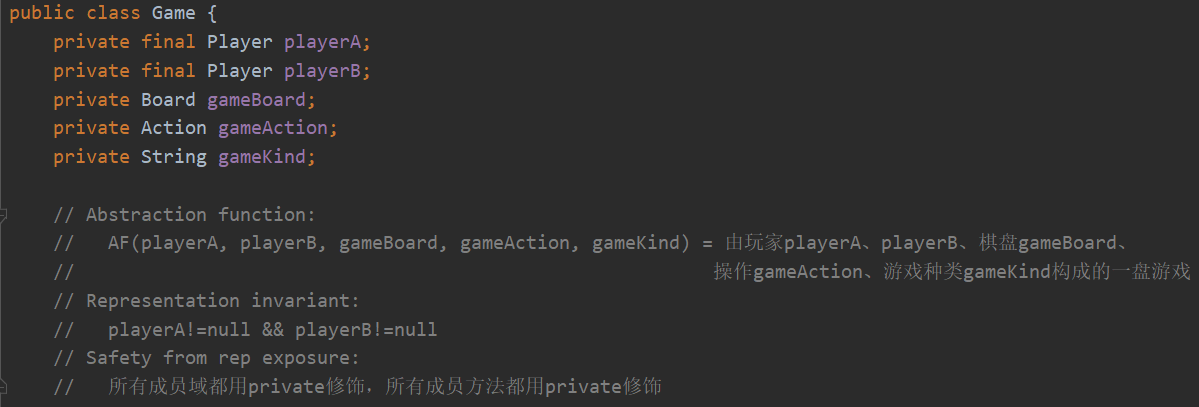
1. Game类：

创建并进行一盘新的棋类游戏

属性：两名玩家、一个棋盘、一个操作类、游戏种类

方法：初始化象棋/围棋游戏、启动游戏、打印菜单、执行操作等

是可变类



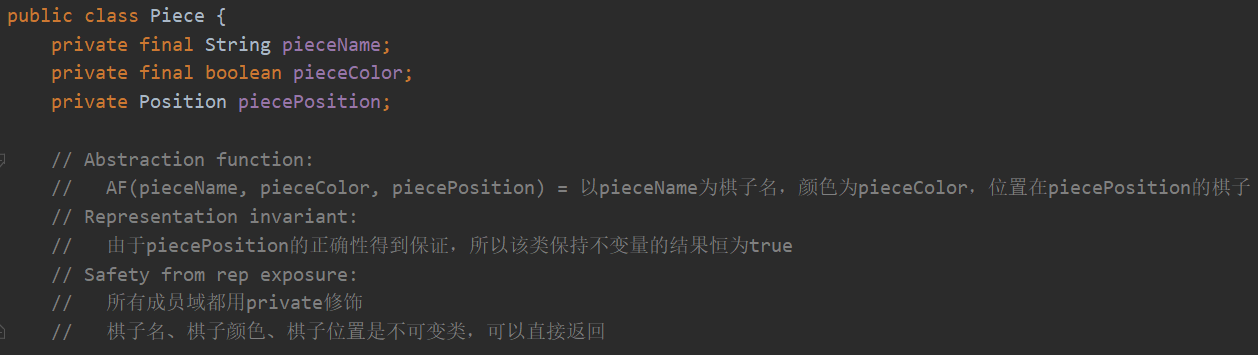
1. Piece类：

棋类游戏的棋子类型

属性：棋子名、棋子颜色、棋子位置

方法：构造方法、获得棋子名、获得棋子颜色、获得棋子位置

是不可变类



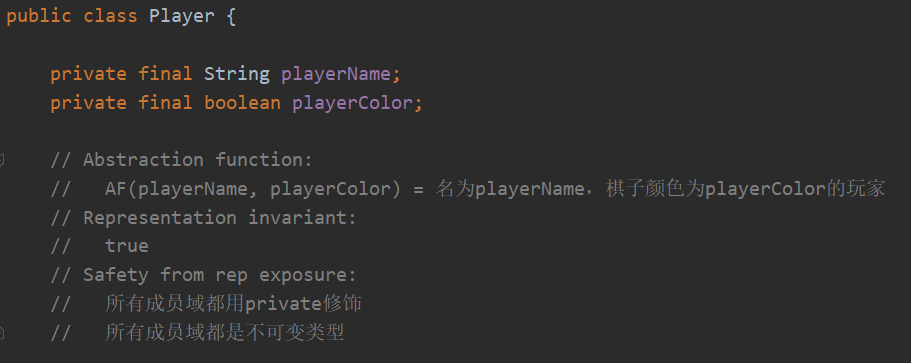
1. Player类：

棋类游戏的玩家类

属性：姓名、拥有的棋子颜色

方法：构造方法、获得玩家名、获得棋子颜色

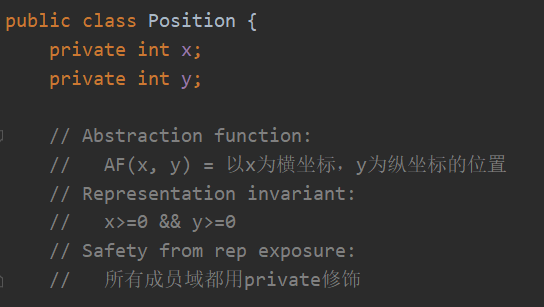
不可变类型



1. Position类：

记录、获取棋子位置

不可变类型

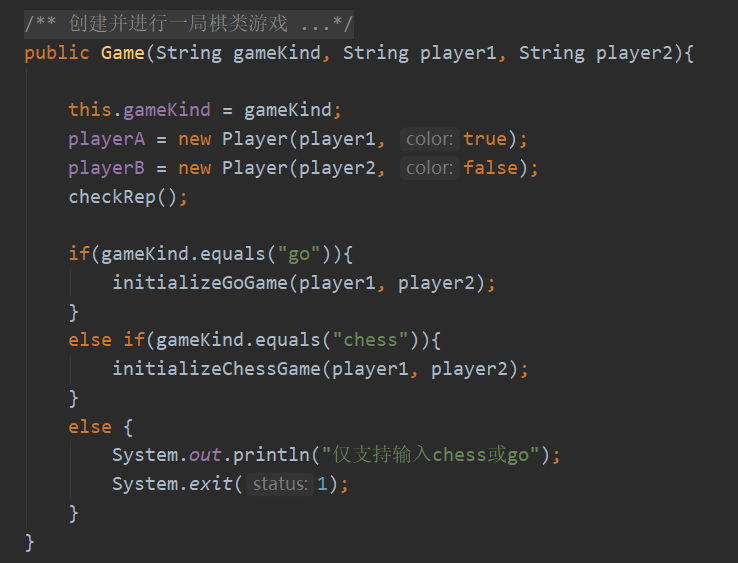


### 主程序MyChessAndGoGame设计/实现方案

我设计的MyChessAndGoGame类中的main方法只起到提示输入初始化游戏信息的作用，游戏的具体实现在Game类中。所以下面介绍Game类的方法。

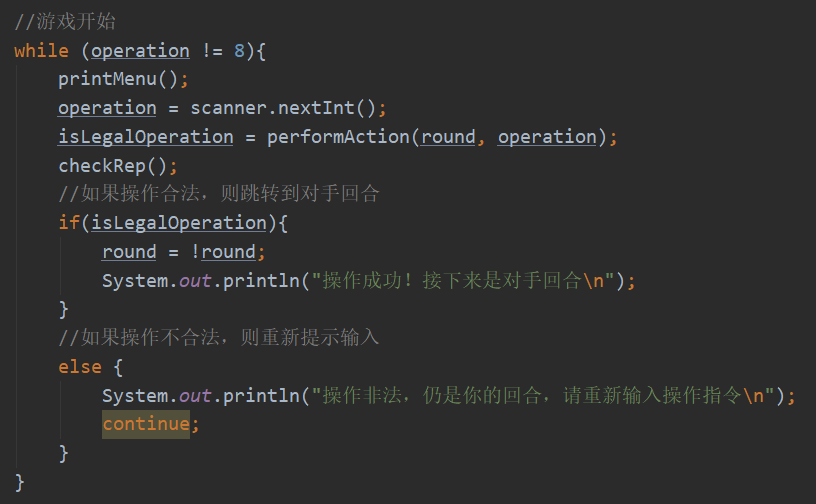
1. 初始化

获得输入信息后，对游戏进行初始化。由于两类游戏的棋盘大小和棋子的初始状态不同，而玩家信息相同，所以在创建玩家实例之后对两种游戏分别进行初始化。在初始化象棋游戏时，将象棋棋子添加到棋盘上。

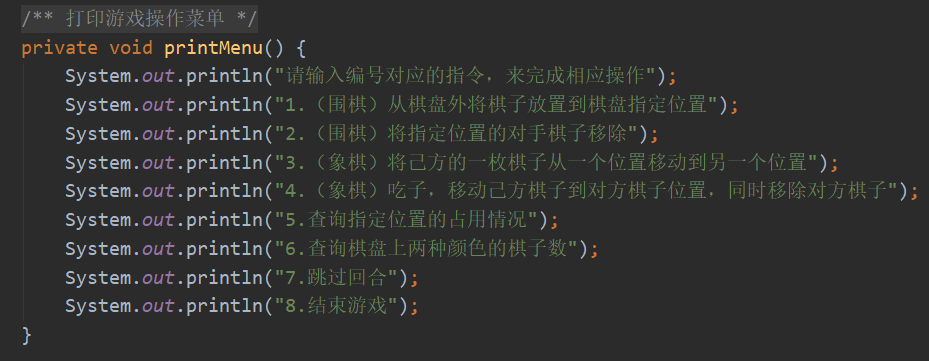


1. 进行游戏

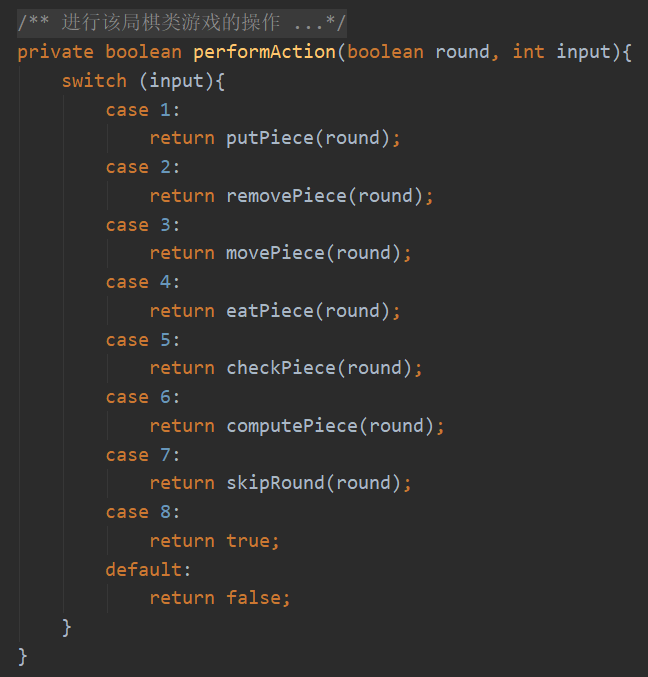
用int类型的变量operation记录用户输入的操作，从1到8每个数字代表一种操作，在终端中显示（通过调用printMenu方法）。然后将该值传入performAction方法，进行对应操作。该方法返回布尔值，标记是否操作成功。如果操作成功，则进入对方回合（round取反），否则仍是当前玩家的回合。



printMenu方法：

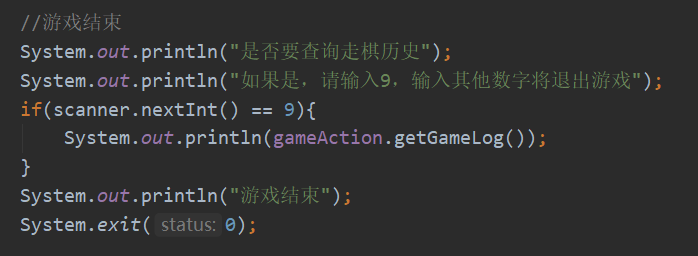


performAction方法：



其中，每种情况对应的操作不再一一赘述。使用switch结构使用户输入更加简洁。

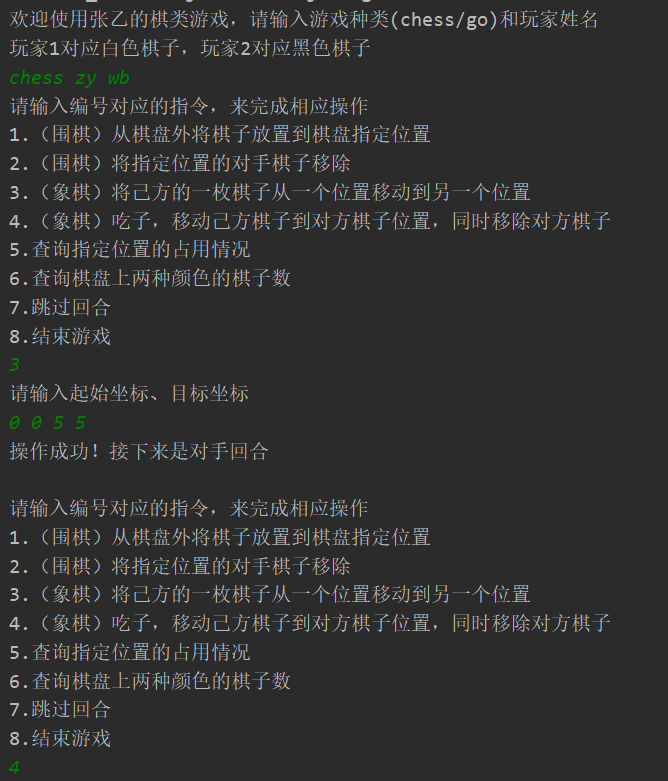
1. 游戏结束

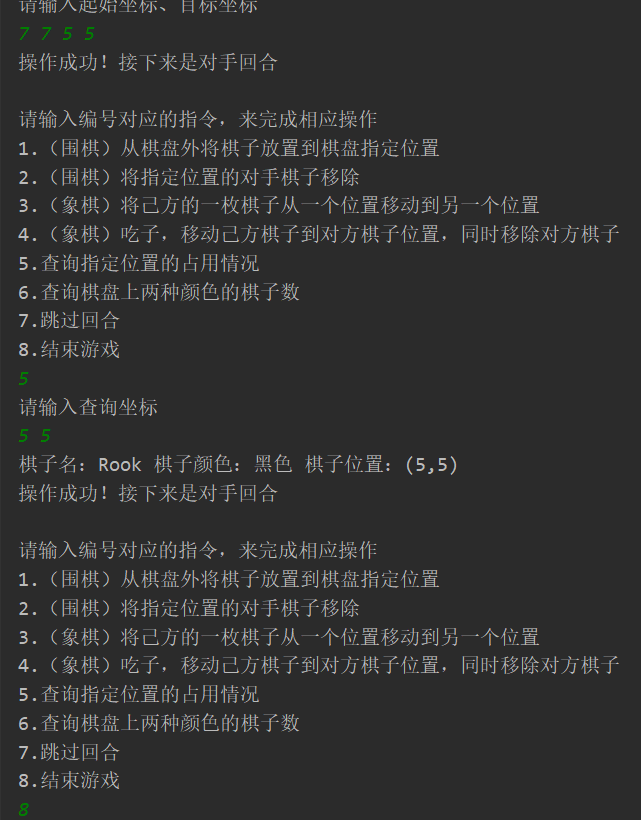


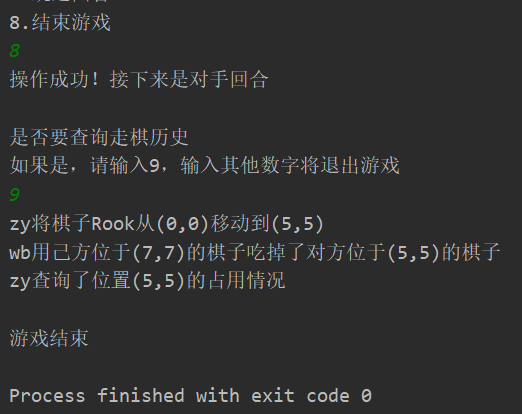
游戏结束时会提示玩家是否查询走棋信息。如果是，则打印走棋历史，否则直接退出游戏。

运行截图：覆盖了所有的操作

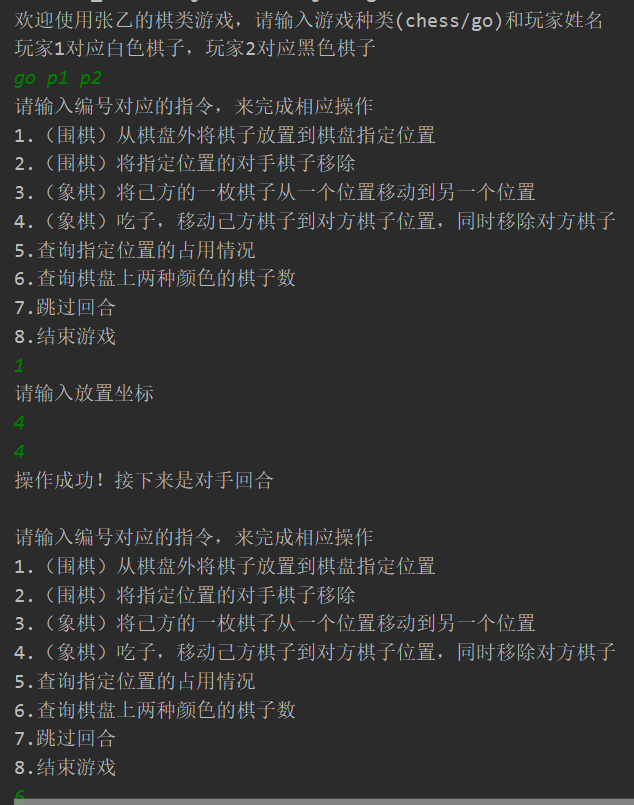
1. 象棋

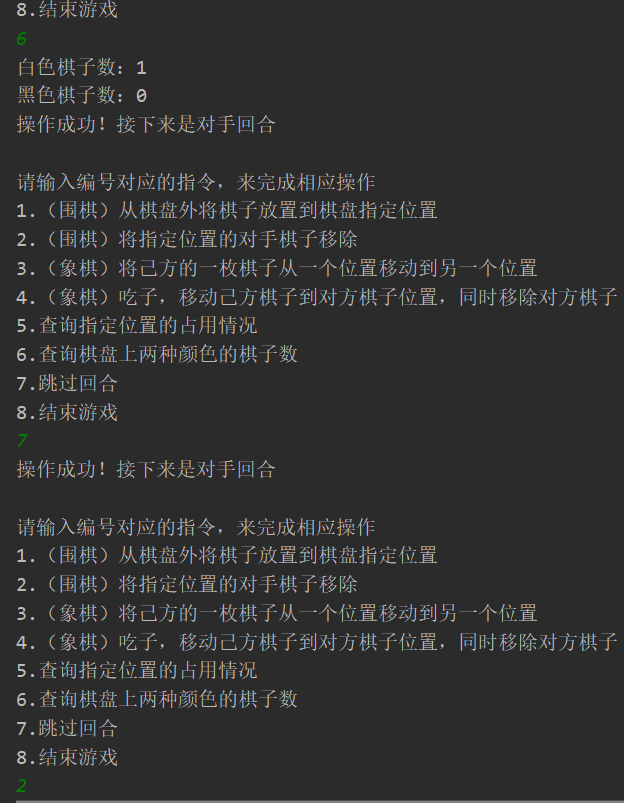


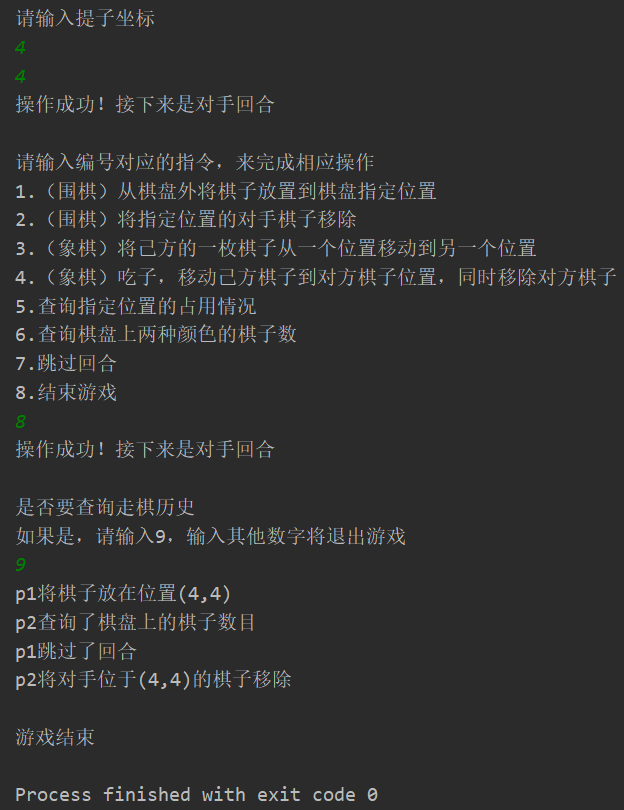




1. 围棋







### ADT和主程序的测试方案

1. Action类的测试

针对每种方法，对输入操作进行等价类划分

在尽可能简洁的情况下，提高代码覆盖率

按操作棋子颜色划分：操作棋子属于己方、操作棋子属于对方

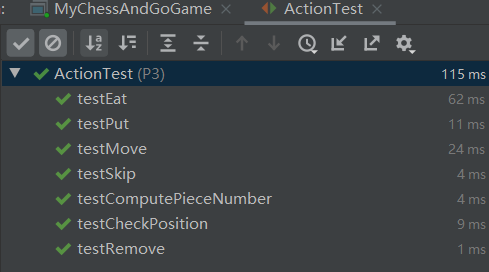
按棋子的位置划分：位置在棋盘上且该位置无棋子、位置在棋盘上且该位置有棋子、位置不在棋盘上

按操作棋子类型划分：操作棋子属于己方、操作棋子属于对方、输入位置无棋子

按目标位置划分：目标位置是己方棋子、目标位置是对方棋子、目标位置无棋子

按输入玩家的棋子颜色划分：黑色、白色





1. Board类测试

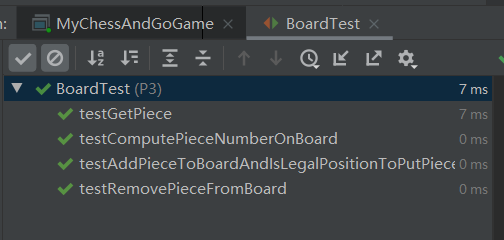
添加棋子分类：棋子不在棋盘上、棋子已在棋盘上

输入位置分类：位置在棋盘上且无棋子、位置在棋盘上且有棋子、位置不在棋盘上

移除棋子输入分类：待移除棋子在棋盘上、待移除棋子不在棋盘上

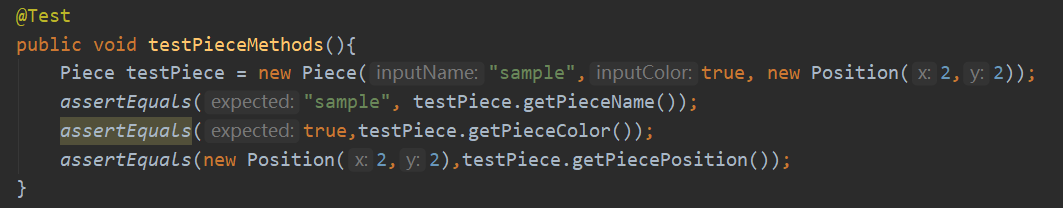
计算指定颜色棋子输入划分：黑色、白色

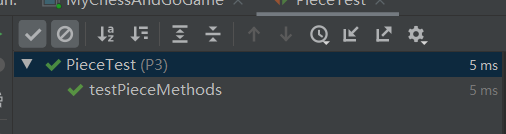




1. Piece类测试

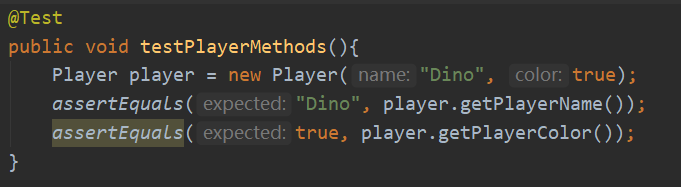
创建棋子实例，由于Piece类只有observer成员方法，则查看输出是否等于输入信息即可

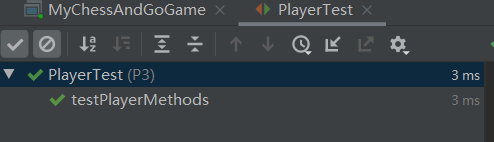




1. Player类测试

创建一个新的玩家实例，测试成员方法输出是否和期望相等

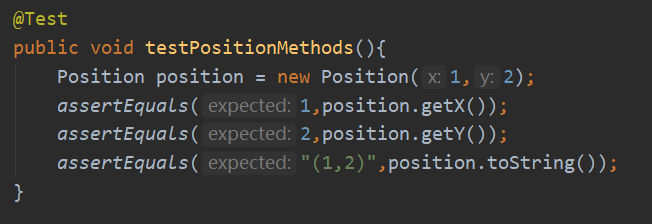


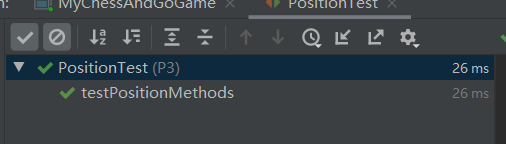


1. Position类测试

由于在Position类中的checkRep方法保证了横纵坐标非负，所有测试只考虑坐标为非负的情况

新建一个位置，调用成员方法查看输出是否等于期望





1. 主程序（Game类）测试：

对程序运行进行一系列健壮性的测试。由于Game类的方法都用private修饰，所以不用另外写测试类。

按操作棋子颜色划分：操作棋子属于己方、操作棋子属于对方

按棋子的位置划分：位置在棋盘上且该位置无棋子、位置在棋盘上且该位置有棋子、位置不在棋盘上

按操作棋子类型划分：操作棋子属于己方、操作棋子属于对方、输入位置无棋子

按目标位置划分：目标位置是己方棋子、目标位置是对方棋子、目标位置无棋子

按输入玩家的棋子颜色划分：黑色、白色

# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.19 | 16:00-19:30 | P1的测试和图的第一种实现（除AF、RI、表示暴露等没学的部分） | 按时完成 |
| 3.20 | 8:00-12:00 | P1图的第二种实现以及测试 | 按时完成 |
| 3.20 | 19:00-22:00 | P1的GraphPoet以及测试 | 延期半小时 |
| 3.25 | 13:00-17:30 | P2 | 按时完成 |
| 3.30 | 14:00-18:00 | P3类的测试和基本设计 | 提前一小时 |
| 3.31 | 8:00-11:30 | P3代码的完成和debug | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚开始接触实验时，不懂AF、RI和表示暴露的意思 | 老师讲到这点时，认真听讲，课后总结笔记并将知识点提炼后写成博客，帮助实验顺利完成。 |
| P3需要设计的类太多，无从下手 | 动手写程序之前，首先规划好各类之间的关系，明确关系之后再写难度明显降低。先完成简单的底层的类。 |
| 不明白如何使P3的客户端操作尽量简洁 | 经过思考后发现switch语句比较理想，只需要让用户输入一个数字就能执行相应操作 |

# 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

写函数规约十分有必要，好的函数规约可以使开发过程更加简单，并且注重于实际功能。写程序之前要先构思好大致结构，跳过思考而直接写代码效率低，甚至会出现写到一半需要重写的尴尬局面。如果不知道如何入手，可以先从底层的、基础的部分开始实现。

## 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的编程需要有AF、RI、避免表示暴露的约束，要将内部实现隐藏起来，不能直接操作对象的属性。面向应用场景编程允许直接操作数据。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型可以使应用面更广，而不仅仅局限于某一类型。

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？

在完成一个方法的代码后，可以直接进行测试。保证每个方法都是正确的，避免在调用时出现被调用方法发生错误，造成debug困难。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？

复用增加了ADT适用性，可用范围更广。

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？

适应。刚开始因为需要设计的类太多而觉得无从下手，后来听取同学建议，先设计简单的类，再利用已设计的类完成较为复杂的类。同时注意类的可变性。

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？

保证了类编写的正确性。愿意。

1. 关于本实验的工作量、难度、deadline。

工作量适中，难度中等偏上，ddl合适。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

希望老师在布置实验时，能够说明哪些部分是暂时没有讲到的，可以先空着等以后再写。这次实验很多同学都在问AF、RI、checkRep()等等（当时没学），我也疑惑这些东西是什么，因此浪费了一些时间。

这门课程能让我学到很多东西。写程序时很多不清楚的地方都会在课上一一解决，我很喜欢这门课程。