

**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 | 王达 |
| 学号 | 1170300323 |
| 班号 | 1703003 |
| 电子邮件 | 775017232@qq.com |
| 手机号码 | 13100802368 |

[1 实验目标概述 1](#_Toc1988214)

[2 实验环境配置 1](#_Toc1988215)

[3 实验过程 1](#_Toc1988216)

[3.1 Poetic Walks 1](#_Toc1988217)

[3.1.1 Get the code and prepare Git repository 2](#_Toc1988218)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc1988219)

[3.1.3 Problem 2: Implement Graph <String> 2](#_Toc1988220)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc1988221)

[3.1.3.2 Implement ConcreteVerticesGraph 3](#_Toc1988222)

[3.1.4 Problem 3: Implement generic Graph<L> 3](#_Toc1988223)

[3.1.4.1 Make the implementations generic 3](#_Toc1988224)

[3.1.4.2 Implement Graph.empty() 3](#_Toc1988225)

[3.1.5 Problem 4: Poetic walks 4](#_Toc1988226)

[3.1.5.1 Test GraphPoet 4](#_Toc1988227)

[3.1.5.2 Implement GraphPoet 4](#_Toc1988228)

[3.1.5.3 Graph poetry slam 5](#_Toc1988229)

[3.1.6 Before you’re done 5](#_Toc1988230)

[3.2 Re-implement the Social Network in Lab1 5](#_Toc1988231)

[3.2.1 FriendshipGraph类 5](#_Toc1988232)

[3.2.2 Person类 6](#_Toc1988233)

[3.2.3 客户端main() 6](#_Toc1988234)

[3.2.4 测试用例 6](#_Toc1988235)

[3.3 Playing Chess 6](#_Toc1988237)

[3.3.1 ADT设计/实现方案 6](#_Toc1988238)

[3.3.2 主程序ChessGame设计/实现方案 7](#_Toc1988239)

[3.3.3 ADT和主程序的测试方案 10](#_Toc1988240)

[4 实验进度记录 11](#_Toc1988242)

[5 实验过程中遇到的困难与解决途径 12](#_Toc1988243)

[6 实验过程中收获的经验、教训、感想 12](#_Toc1988244)

[6.1 实验过程中收获的经验和教训 12](#_Toc1988245)

[6.2 针对以下方面的感受 13](#_Toc1988246)

# 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

本次实验与Lab1的实验环境基本一致，只需要在eclipse上配置EclEmma，右键点击项目，Coverage as运行，即可观察代码的覆盖程度。

我的GitHub Lab2仓库的URL地址：

https://github.com/ComputerScienceHIT/Lab2-1170300323

# 实验过程

## Poetic Walks

本实验的两个类ConcreteEdgesGraph，ConcreteVerticesGraph实现接口Graph。其中图的顶点类型设定为泛型。

Graph接口要求实现6种方法，add（向图中添加顶点），set（向图中添加边/更改边的权值/去掉边），remove（移除顶点），vertices（获取顶点集），sources(target)（获取以target作为目标结点的所有起始结点），targets(source)（获取以source作为起始结点的所有目标结点）

Poetic：给定一组句子，每个单词视作一个顶点，单词与单词之间视作一条有向边，以此构建有向图，并用Graph来表示。根据这个图来对输入的句子进行改造，如果输入句子中的两个单词在有向图中存在一个中间单词，那么将中间单词插入到输入的两个单词中。输出最后得到的句子。

### Get the code and prepare Git repository

<https://github.com/rainywang/Spring2019_HITCS_SC_Lab2/tree/master/P1>

从上述网址中获得代码，在github中根据提示操作将其关联起来。

### Problem 1: Test Graph <String>

在GraphInstanceTest测试文件中，完成对Graph接口中每一个方法的测试，这里只需要实现对emptyInstance（即测试用例为空）的情况。

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

这里需要以表示边的方式来实现Graph接口，首先需要实现Edge类。

Edge类中的属性有起始结点（source），目标结点（target），边的权重（weight）；

方法包括对三个属性的getter与setter方法，checkRep方法对边的合法条件进行检测，toString方法返回边的String类型表示。

ConcreteEdgesGraph类：

add方法：调用Set.add()方法添加顶点

set方法：首先判断顶点是否为空以及权重是否大于0，然后判断要操作的边是否存在，如果存在且weight==0则去掉这条边，如果weight!=0那么修改边的权值为weight。如果不存在这条边那么向边表中添加这条边。

remove方法：调用Set.remove()移除顶点，并移除与该顶点相关的所有边。

vertices方法：new一个set类型对象，使值与vertices相等，这样做能保证immutable的要求。

sources方法：定义一个map。遍历Edge列表，如果目标结点与target相等，那么将起始结点与该边权重作为key和value加入到map中，返回map

targets方法：定义一个map。遍历Edge列表，如果起始结点与source相等，那么将目标结点与该边权重作为key和value加入到map中，返回map

ToString方法：以String类型来表示这个图

测试：对上述方法选择合适的测试用例，进行测试。

#### Implement ConcreteVerticesGraph

这里需要以表示结点的方式来实现Graph接口，首先需要实现Vertex类。

Vertex类中的属性有vertex（结点），sources（Map类型，以vertex为目标结点时，起始结点与边权值的map集合），targets（Map类型，以vertex为起始结点时，目标结点与边权值的map集合）。

方法包括getter与setter方法，定义remove方法从sources或targets集合中移除一组顶点与权值的关系。

ConcreteVerticesGraph类：

add方法：调用List.add()方法添加顶点

set方法：首先判断顶点是否为空以及权重是否大于0，判断source和target是否存在，如果不存在向vertices集合中添加顶点，之后调用set方法来完成相应map的实现。

remove方法：遍历vertices中的所有顶点，找到要删除的结点，删除与该结点有关的所有边后删除该顶点。

vertices方法：将vertices中的结点添加到一个set中，返回set

sources方法：遍历vertices列表，找到指定的目标结点后，返回该结点所对应的属性sources。

targets方法：遍历vertices列表，找到指定的起始结点后，返回该结点所对应的属性targets。

ToString方法：以String类型来表示这个图

测试：对上述方法选择合适的测试用例，进行测试。

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

#### Make the implementations generic

将类声明改为：

**public** **class** ConcreteEdgesGraph<L> **implements** Graph<L>

**class** Edge<L>

**public** **class** ConcreteVerticesGraph<L> **implements** Graph<L>

**class** Vertex<L>

#### Implement Graph.empty()

代码如下所示：

**public** **static** <L> Graph<L> empty() {

**return** **new** ConcreteEdgesGraph<L>();

}

### Problem 4: Poetic walks

#### Test GraphPoet

配置文件：File file = **new** File("test/P1/poet/seven-words.txt");

代码如下：

@Test

**public** **void** testPoem() **throws** IOException {

File file = **new** File("test/P1/poet/seven-words.txt");

GraphPoet a = **new** GraphPoet(file);

*assertEquals*("", a.poem(""));

*assertEquals*("You are so smart", a.poem("You are so smart"));

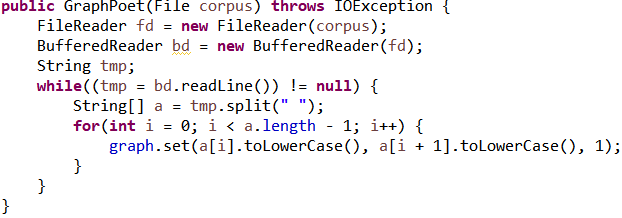
*assertEquals*("Seek to explore strange new life and exciting synergies!", a.poem("Seek to explore new and exciting synergies!"));

*assertEquals*("Seek out new life and strange new worlds and new civilizations", a.poem("Seek new and strange worlds and civilizations"));

}

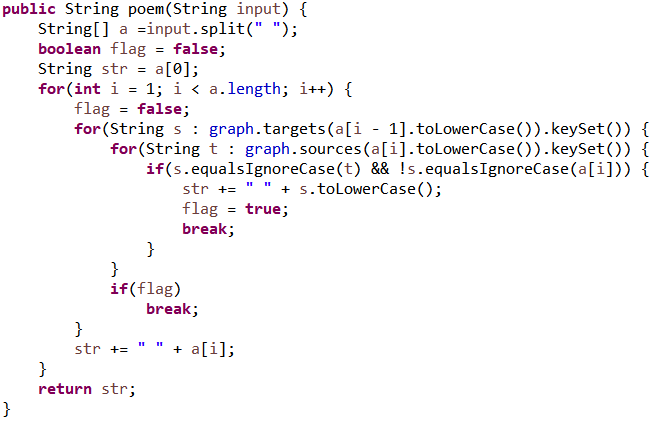
#### Implement GraphPoet

首先实现构造函数：



构造函数用于读取语料库文件中的语句，并将其绘制成图。

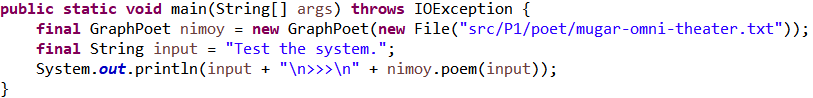
实现poem函数如下：



对输入语句中每两个临近的单词进行判断，如果在图中两个单词之间存在一个中间单词，那么将其加入到字符串当中，返回最后得到的字符串。

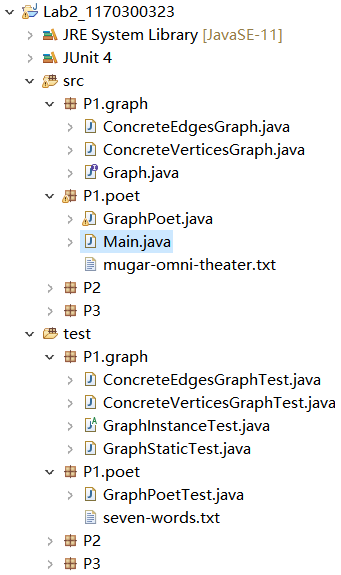
#### Graph poetry slam

可以对main函数进行扩展，实现相对复杂的字符串：



### Before you’re done

项目的目录结构树状示意图如下：



## Re-implement the Social Network in Lab1

在通过P1中的ConcreteEdgesGraph类或ConcreteVerticesGraph类来实现Lab1中P3：FriendshipGraph类。

### FriendshipGraph类

用P1的ConcerteEdgesGraph类定义一个图模型graph，用于表示关系网。addVertex向graph添加Person类型的顶点，addEdge向graph添加Edge<Person>类型的边，getDistance仍采用广度优先搜素的方式，在Person类中定义dep变量用于表示从一个人出发到某个人的深度，这样只需要从两个人中的一个人出发，计算另一个人的深度即为最短距离。

### Person类

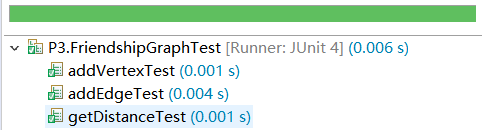
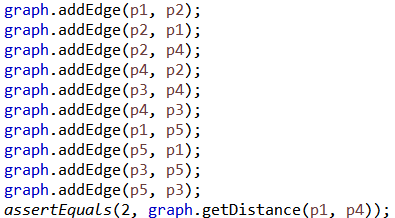
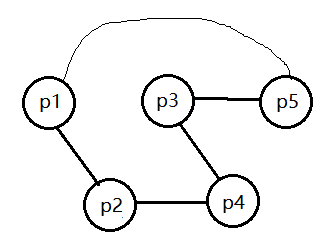
给Person类中分别定义了name表示人的姓名，dep变量用于表示广度优先搜索具体一个人的深度，vis变量用于表示一个人是否被访问过。

### 客户端main()

main方法采用实验一中给出的main方法进行测试。

### 测试用例

分别除了指导书中给的测试用例外，还对其他测试用例进行了测试，这里举一个例子：



其中p1和p4的最短距离为2，经过测试证明是正确的。

## Playing Chess

### ADT设计/实现方案

Position类：

属性：x，y分别表示横纵坐标，为int类型

方法：getX(), getY()用于获取横纵坐标，返回值为int类型

Piece类：

属性：color（int）：棋子颜色， type（int）：国际象棋棋子类型，

pos(Position)：棋子位置

方法：get、set方法略

public String color()：返回棋子颜色的String类型

public String type()：返回棋子类型的String类型

Player类：

属性：name（String）：姓名，color（int）：棋子颜色

方法：get、set以及color（同Piece类color）方法

Board类：

属性：

private Piece[][] a；用于表示棋盘上的棋子

其他为与围棋GUI有关的属性，请见对应属性的注释

（围棋GUI还定义了额外的类，详情请见Board类）

构造函数：构建初始棋盘

方法：

**public** Piece concretePiece(Position pos)：返回具体位置的棋子

其他均为与国际象棋GUI有关的方法，请见对应方法的注释

Action类：

属性：b（Board）：棋盘，p（Piece）：棋子

方法：getPiece，setPiece

围棋：

**public** **void** setToTarget(Player a)：放置一枚棋子

**public** **void** remove()：去掉一枚棋子

国际象棋：

**public** **void** change(Position pos)：改变一枚棋子的位置

**public** **void** eat(Position pos)：吃掉对方的一枚棋子

Game类：

属性：type（int）：游戏类型，b（Board）：棋盘

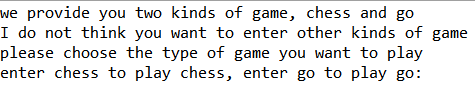
构造函数：产生新的游戏

### 主程序MyChessAndGoGame设计/实现方案

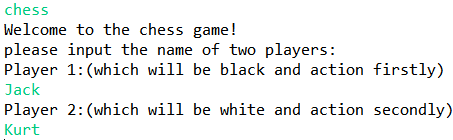
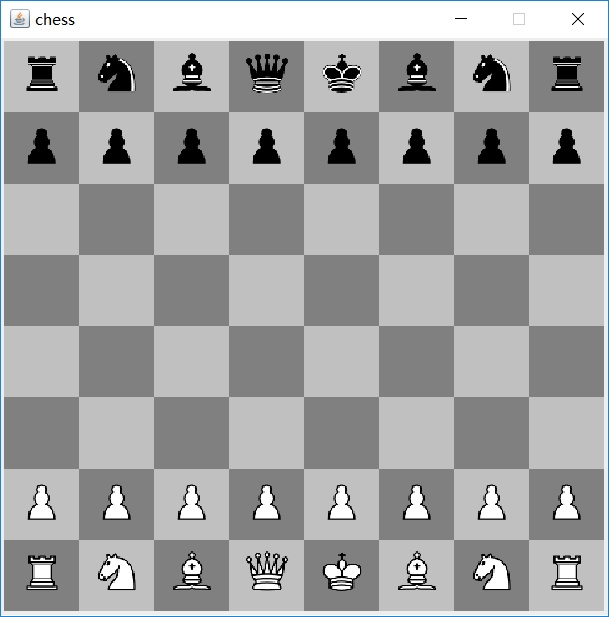
辅之以执行过程的截图，介绍主程序的设计和实现方案，特别是如何将用户在命令行输入的指令映射到各ADT的具体方法的执行。

一、游戏过程：

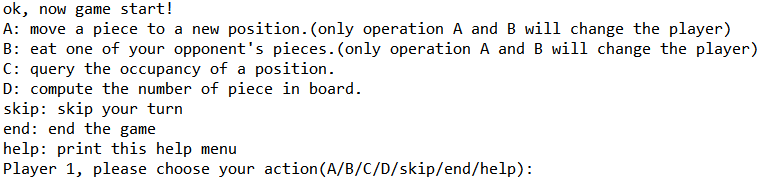
运行游戏后提示输入chess或go，如果输入的不是chess或go，则重新输入。



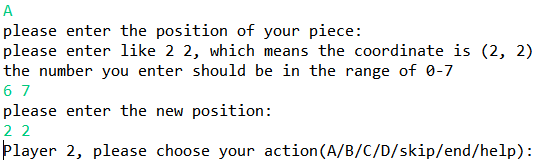
1. 输入chess后会显示GUI界面，之后输入两名玩家的姓名：



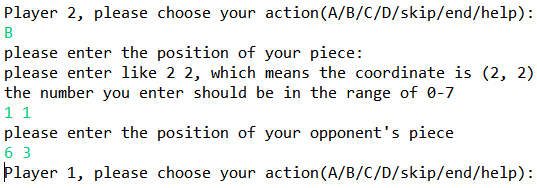
开始游戏，玩家1操纵黑棋，先行动：



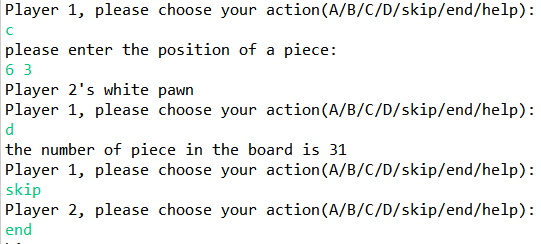
根据上述提示选择要操作的选项：



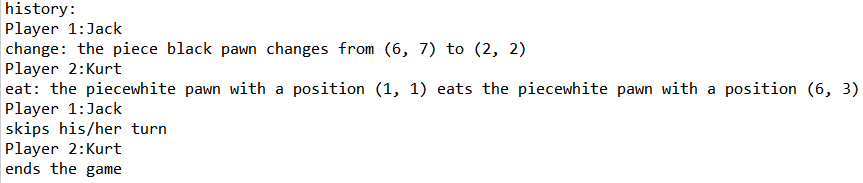
玩家1进行A操作，将坐标6 7的黑棋移动到2 2的空位上，这里调用了Action类的change方法，之后轮到玩家2操作：



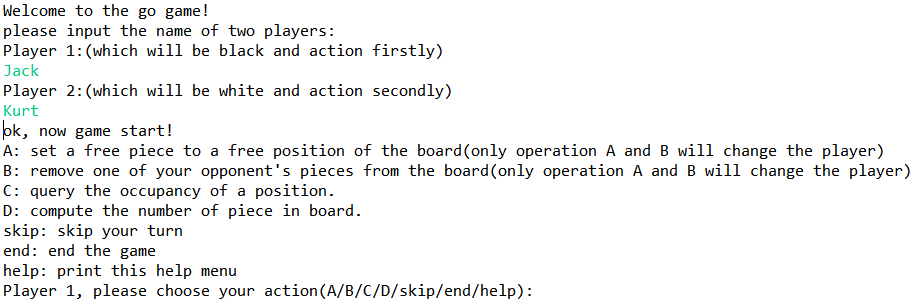
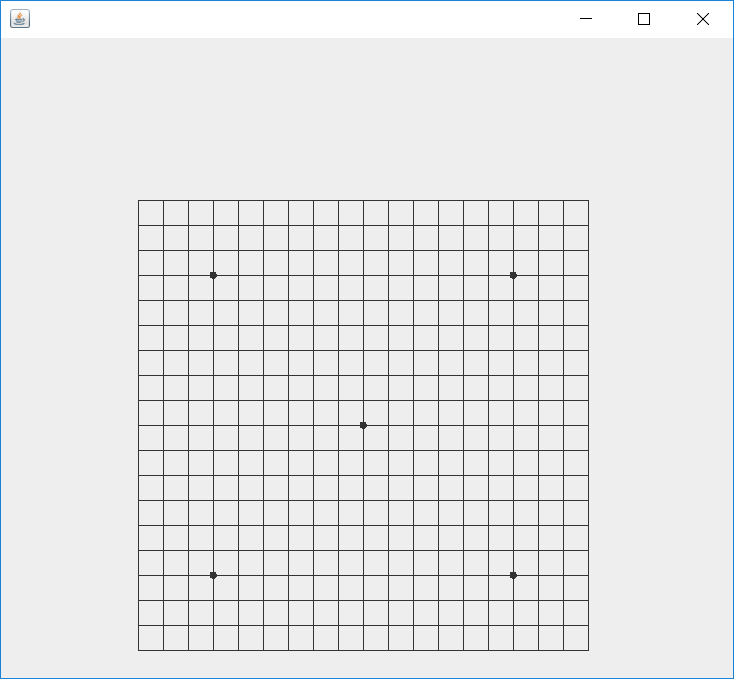
玩家2进行B操作，将坐标1 1的白棋吃掉6 3的黑棋，这里调用了Action类的eat方法，轮到玩家1操作：



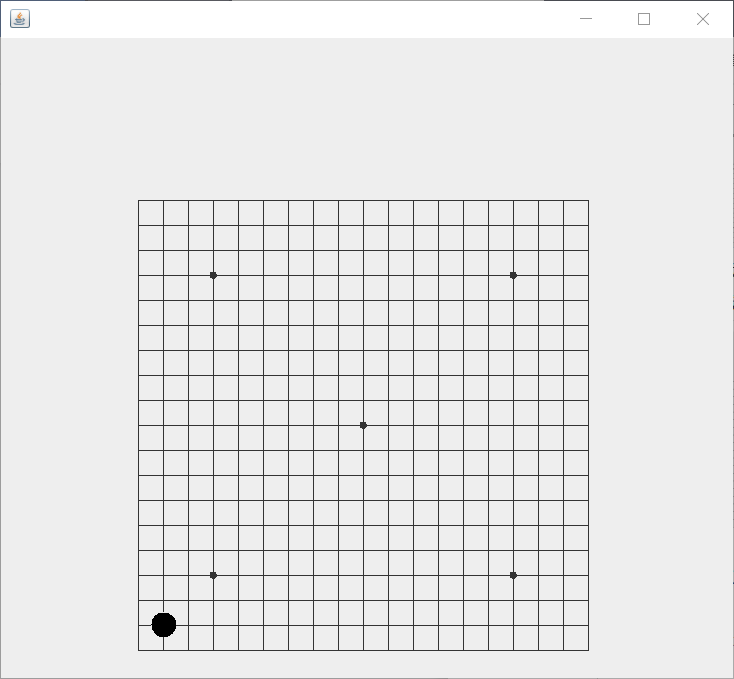
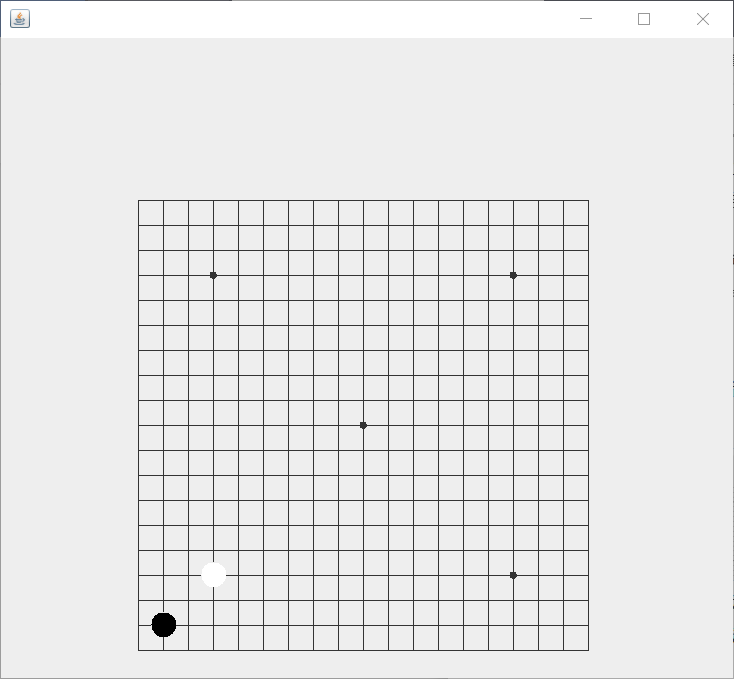
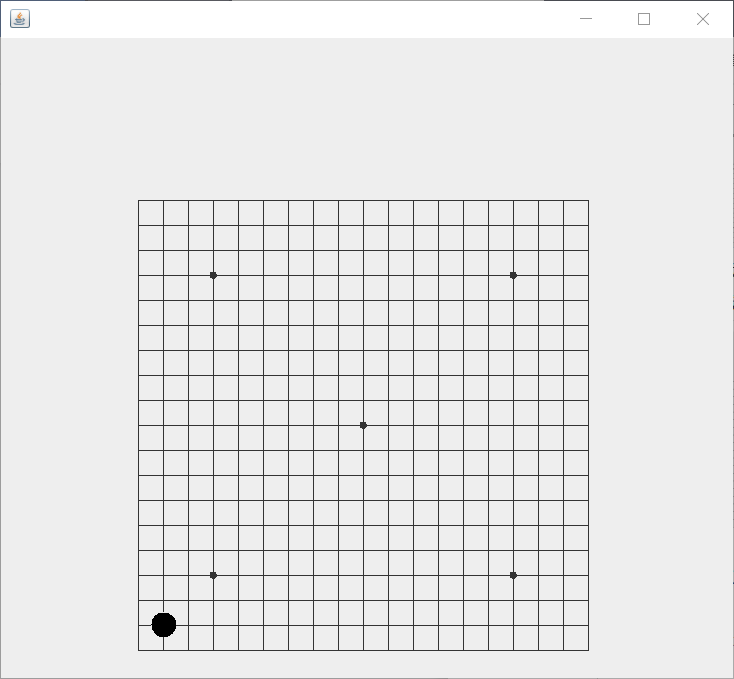
玩家1进行C操作，查看6 3的棋子，进行D操作，查看总共的棋子数，之后跳过回合，玩家2选择结束游戏，以下是本场游戏的历史：

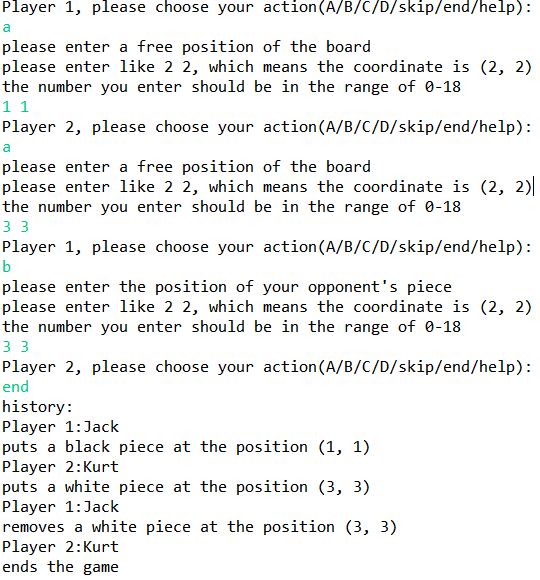


1. 输入go后会显示GUI界面，并输入玩家的名字，显示可行操作，轮到玩家1开始：



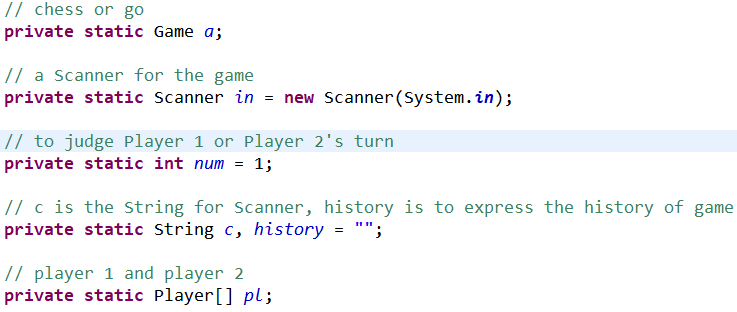
玩家1选择放置棋子，玩家2选择放置棋子，调用Action类的setPiece操作，玩家1选择吃掉刚才玩家2的棋子，调用Action类的remove操作。玩家2结束游戏。





1. MyChessAndGoGame类的实现：

属性：

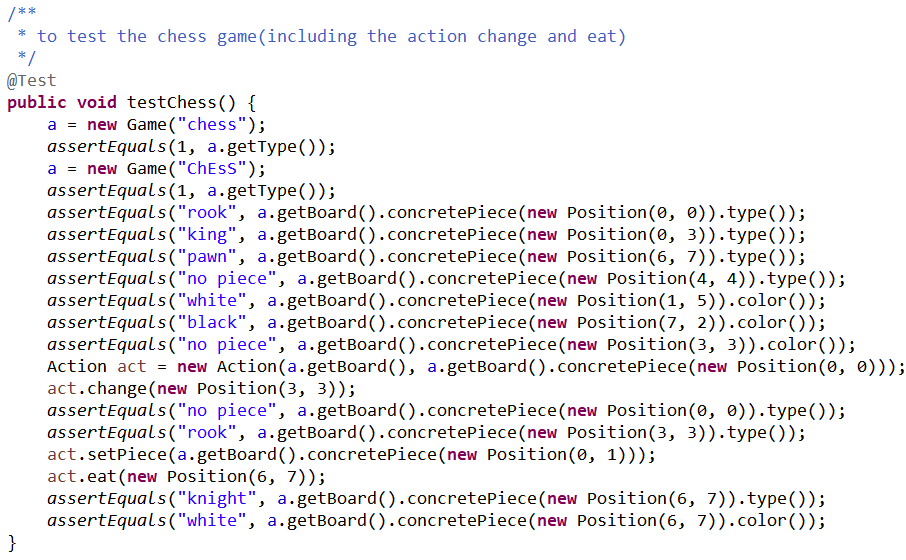


主函数：首先根据输入确定游戏的类型，通过一个大循环来表示游戏交互的过程，在其中放入相关的操作。

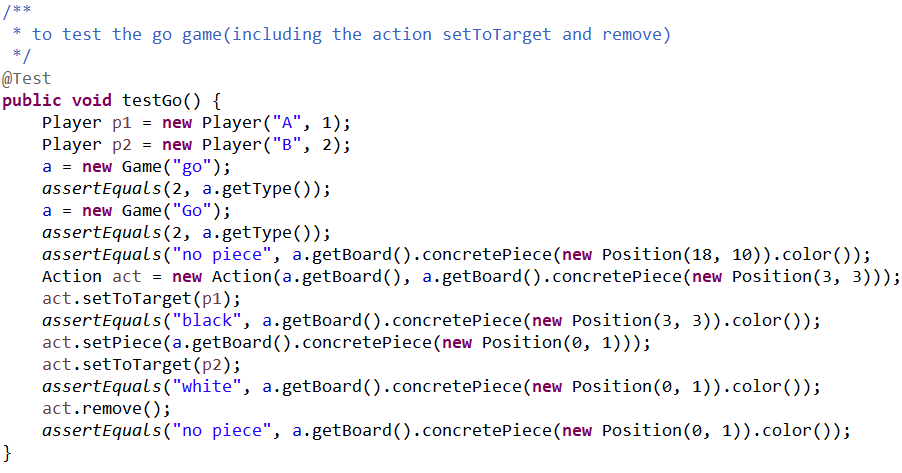
其他函数：用于辅助主函数，包括对检验输入是否合法，如输入的位置是否为自己的棋子等等。

### ADT和主程序的测试方案

以下是我分别对chess和go的测试：



在testChess函数中我测试了输入的正确性，初始棋盘棋子类型与颜色，经过移动后的棋子类型与颜色，吃子后的棋子类型与颜色。



在testGo函数中我测试了输入的正确性，初始棋盘棋子颜色，放子后的棋子颜色，移子后的棋子颜色。

# 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 | 截至时间 | 完成的任务 |
| 3-17 | 21:00 | 完成P1的Problem 1 |
| 3-19 | 12:00 | 完成P1的Problem 2的ConcreteEdgesGraph |
| 3-20 | 22:00 | 完成P1的Problem 2的ConcereteVerticesGraph |
| 3-21 | 15:00 | 完成P1的Problem 3 |
| 3-23 | 17:00 | 完成P1的Problem 4 |
| 3-24 | 20:00 | 完成P2 |
| 3-26 | 21:00 | 完成P3的chess部分 |
| 3-28 | 13:00 | 完成P3的go部分 |
| 4-4 | 11:00 | 完成实验报告并提交 |

# 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对于问题的入手 | 最开始看了P1的实验要求，并不清楚P1要求我们做什么，不过通过在Piazza上询问得知首先要完成测试文件，并依次实现边和顶点类。 |
| P1改用泛型 | 由于最初不熟悉泛型如何操作，因此通过看书以及上网查找资料，终于得知如何操作泛型，并成功的实现了泛型类。 |
| P1写诗问题 | 起初不明白问题要我们做什么，后来阅读要求，对实验的认知逐渐清晰，并完成。 |
| P2的getdistance | 我这里运用了广度优先搜索的策略，但运行得到的结果并不正确，后经多次调试得到了正确的结果。 |
| P3上手 | 这个实验让我们真正接触面向对象，但从零开始并不容易，要理解每一个类的作用以及要实现的方法。 |
| P3的两种游戏类型区分 | 两种游戏类型需要在每一个类中进行不同的操作，因此需要对游戏类型进行判定区分。 |
| P3主函数的编写 | 主函数要求完整实现游戏过程，需要对非法输入进行判断，而且编写一系列的语句，过程并不容易。 |
| 注释的编写 | 从这个实验开始，每一个函数都要有自己的注释，针对测试文件要编写test strategy，并编写rep invariant，abstraction function以及代码暴露处理措施。 |

# 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

实验中我学习了和接口，泛型有关的操作，并掌握了注释的编写技巧，能够独立的从零完成一个面向对象的程序编写。

我知道了要想完成实验，需要坚持不懈，只有这样才能攻克难的部分。

## 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是用功能来划分问题，这样能够节省一些不必要的步骤。而面向应用场景是按照步骤来实现问题，更加直观易懂。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型能够将类适用于各种类型，使得对象更加自由。但也会受到一定限制，例如不存在泛型数组，泛型不能实例化等等。

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？

只需要知道ADT实现的功能，就来编写测试用例，这样做可以使测试用例不受到代码的限制，也就是所谓的黑盒子测试。我可以适应这种测试方式。

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？

对ADT的应用，能够大幅度简化别的应用场景对具体功能的实现。

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？

我可以适应，在具体的应用中，很多情况都需要我们从零开始，这是对我们编写程序的锻炼。我的感受还可以。

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？你是否愿意在以后编程中坚持这么做？

能够更好的确定ADT实现的功能，表示不变量用于判断关于对象状态的条件，并防止出现代码暴露。我会坚持这么做。

1. 关于本实验的工作量、难度、deadline。

工作量和难度较大，deadline可以接受。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

希望老师能够提前讲一些和实验相关的知识。