

**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 | 梁书育 |
| 学号 | 1180300208 |
| 班号 | 1803002 |
| 电子邮件 | 951009723@qq.com |
| 手机号码 | 18845773527 |

**目录**

[1 实验目标概述 1](#_Toc37344756)

[2 实验环境配置 1](#_Toc37344757)

[3 实验过程 2](#_Toc37344758)

[3.1 Poetic Walks 2](#_Toc37344759)

[3.1.1 Get the code and prepare Git repository 2](#_Toc37344760)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc37344761)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc37344762)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc37344763)

[3.1.3.2 Implement ConcreteVerticesGraph 4](#_Toc37344764)

[3.1.4 Problem 3: Implement generic Graph<L> 5](#_Toc37344765)

[3.1.4.1 Make the implementations generic 5](#_Toc37344766)

[3.1.4.2 Implement Graph.empty() 5](#_Toc37344767)

[3.1.5 Problem 4: Poetic walks 5](#_Toc37344768)

[3.1.5.1 Test GraphPoet 5](#_Toc37344769)

[3.1.5.2 Implement GraphPoet 5](#_Toc37344770)

[3.1.5.3 Graph poetry slam 6](#_Toc37344771)

[3.1.6 Before you’re done 6](#_Toc37344772)

[3.2 Re-implement the Social Network in Lab1 8](#_Toc37344773)

[3.2.1 FriendshipGraph类 8](#_Toc37344774)

[3.2.2 Person类 8](#_Toc37344775)

[3.2.3 客户端main() 8](#_Toc37344776)

[3.2.4 测试用例 8](#_Toc37344777)

[3.2.5 提交至Git仓库 9](#_Toc37344778)

[3.3 Playing Chess 9](#_Toc37344779)

[3.3.1 ADT设计/实现方案 9](#_Toc37344780)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 13](#_Toc37344781)

[3.3.3 ADT和主程序的测试方案 17](#_Toc37344782)

[4 实验过程中遇到的困难与解决途径 26](#_Toc37344783)

[5 实验过程中收获的经验、教训、感想 26](#_Toc37344784)

[5.1 实验过程中收获的经验和教训 26](#_Toc37344785)

[5.2 针对以下方面的感受 26](#_Toc37344786)

# 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：

1、针对给定的应用问题，从问题描述中识别所需的 ADT；

2、设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；

3、根据 ADT 的规约设计测试用例；

4、ADT 的泛型化；

5、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）

6、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）；

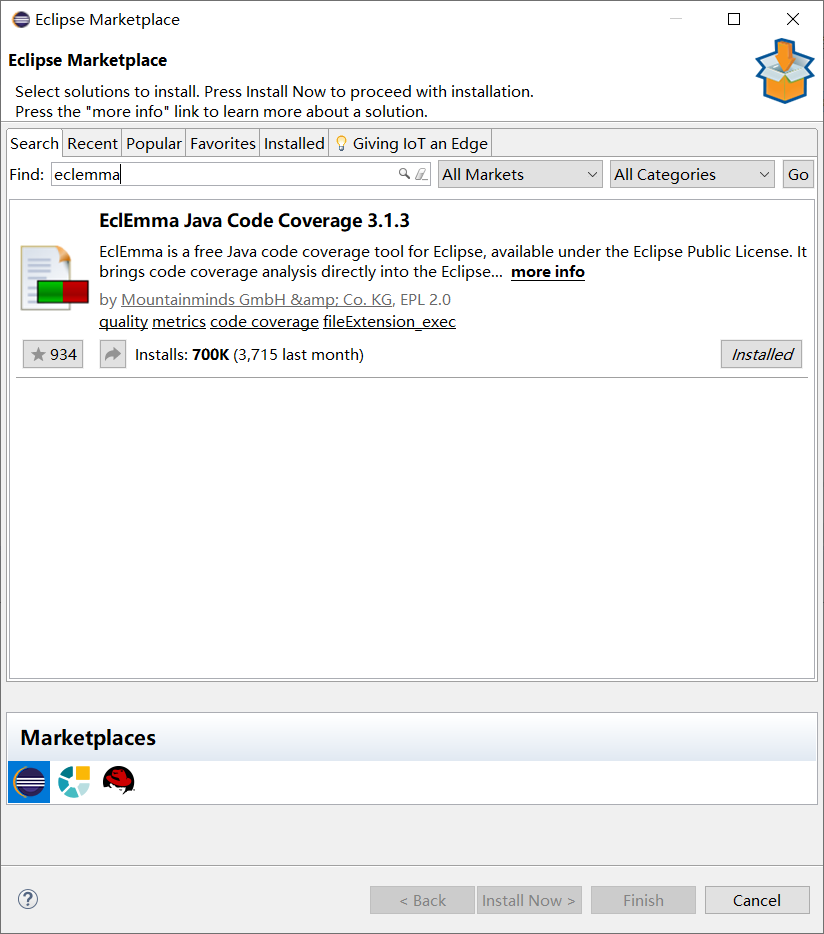
7、测试 ADT 的实现并评估测试的覆盖度；

8、使用 ADT 及其实现，为应用问题开发程序

9、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

在eclipse MarketPlace中搜索eclemma发现已经安装过这一插件，可以直接使用。



在这里给出你的GitHub Lab2仓库的URL地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/Lab2-1180300208>

# 实验过程

## Poetic Walks

该任务主要是对图这一数据结构进行两种实现——一种是记录点和点之间的边来构成图，另一种是通过记录点集来构成图。实现后对其功能进行测试，再利用写出的图数据结构来对句子进行处理，即为所谓的Poetic Walks，通过给定的语料来对给出的句子进行拓展。

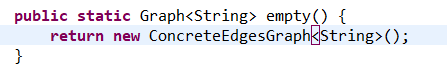
### Get the code and prepare Git repository

通过给出的如下的网址下载所需的代码，使用git的clone功能在本地创建git仓库。<https://github.com/rainywang/Spring2020_HITCS_SC_Lab2/tree/master/P1>

### Problem 1: Test Graph <String>

1、运行GraphStaticTest.java。

将Graph.java中的empty()方法补全为如下：



建立Graph接口的工厂方法，返回一个Graph的实现类（ConcreteEdgesGraph和ConcreteVerticesGraph等价）。

随后运行GraphStaticTest.java

2、为ConcreteEdgesGraph.java以及ConcreteVerticesGraph.java编写测试方法。

由于ConcreteEdgesGraph和ConcreteVerticesGraph是Graph的两个不同实现类，这两个类的测试方法在GraphInstanceTest.java中，用来对图数据结构的操作进行测试，两种实现应该能够通过同样的测试方法，而两种实现方法中特有的Edge类和Vertex类的测试应当分别放在ConcreteEdgesGraph.java和ConcreteVerticesGraph.java中。

Graph<L>的测试见文件GraphInstanceTest.java

Edge<L>的测试见文件ConcreteEdgesGraph.java

Vertex<L>的测试见文件ConcreteVerticesGraph.java

### Problem 2: Implement Graph <String>

#### Implement ConcreteEdgesGraph

ConcreteEdgesGraph是基于边集的Graph实现。

首先实现Edge类。Edge类的实现如下：

**rep：**

**private** **final** String source; 边的起始顶点

**private** **final** String target; 边的目标顶点

**private** **final** **int** weight; 边的权值

**method：**

**public** Edge(L source, L target, **int** weight) 建立边的constructor

**public String** getSource() 获得该边的起点

**public String** getTarget() 获得该边的终点

**public int** getWeight() 获得该边的权值

**public String** toString() 将该边的信息转化成String

有了Edge类就可以进一步实现ConcreteEdgesGraph了。ConcreteEdgesGraph类的实现如下：

**rep：**

**private** **final** Set<String> vertices = **new** HashSet<>() 该Graph的点集

**private** **final** List<Edge> edges = **new** ArrayList<>() 该Graph的边集

**method：**

**public** **boolean** add(String vertex)

向点集中加入新的点。

**public** **int** set(String source, String target, **int** weight)

遍历边集，如果存在一条从source到target的边，则将权值设为weight，返回旧的权值，如果要将权值设为0则删除这条边，如果不存在从source到target的边，则新建这样一条边并返回0.

**public** **boolean** remove(String vertex)

移除vertex顶点，并删去与其关联的所有边。点集中确实有vertex点并移除成功返回true，否则返回false。

**public** Set<String> vertices()

返回点集。

**public** Map<String, Integer> sources(String target)

返回将target点作为终点的所有边，将起点和权值加入到Map中返回。

**public** Map<String, Integer> targets(String source)

返回将source点作为起点的所有边，将终点和权值加入到Map中返回。

**public** String toString()

将图的信息转化为String类型

#### Implement ConcreteVerticesGraph

ConcreteVerticesGraph是基于点集的Graph实现。

首先实现Vertex类。Vertex类的实现如下：

**rep：**

**private** **final** String name; 顶点的名称

**private** Map<String, Integer> sources = **new** HashMap<L, Integer>();

所有以该顶点为终点构成的边的起点和权值形成的Map

**private** Map<String, Integer> targets = **new** HashMap<L, Integer>();

所有以该顶点为起点构成的边的终点和权值形成的Map

**method：**

**public** Vertex(L name) 建立Vertex的Constructor

**public** Map<L, Integer> getSources() 返回该Vertex的source Map

**public** Map<L, Integer> getTargets() 返回该Vertex的target Map

**public** L getName() 返回该Vertex的名称

**public** **void** setSources(Map<L, Integer> sources)

将sources设置为该Vertex的sources

**public** **void** setTargets(Map<L, Integer> targets)

将targets设置为该Vertex的targets

**public** String toString() 将该Vertex的信息转为String

有了Vertex类就可以进一步实现ConcreteVerticesGraph了。

ConcreteVerticesGraph类的实现如下：

**rep：**

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

Garph的点集

**method：**

**public** **boolean** add(String vertex)

向点集中加入新的点。

**public** **int** set(String source, String target, **int** weight)

遍历边集，如果存在一条从source到target的边，则将权值设为weight，返回旧的权值，如果要将权值设为0则删除这条边，如果不存在从source到target的边，则新建这样一条边并返回0.

**public** **boolean** remove(String vertex)

移除vertex顶点，并删去与其关联的所有边。点集中确实有vertex点并移除成功返回true，否则返回false。

**public** Set<String> vertices()

返回点集。

**public** Map<String, Integer> sources(String target)

返回将target点作为终点的所有边，将起点和权值加入到Map中返回。

**public** Map<String, Integer> targets(String source)

返回将source点作为起点的所有边，将终点和权值加入到Map中返回。

**public** String toString()

将图的信息转化为String类型

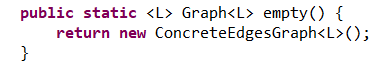
### Problem 3: Implement generic Graph<L>

#### Make the implementations generic

将之前的两个实现ConcreteEdgesGraph和ConcreteVerticesGraph中的String都换成泛型<L>即可，Edge和Vertex也应该改成泛型<L>，具体修改见代码。

#### Implement Graph.empty()

将此前的实现中的String换成L，如下：



### Problem 4: Poetic walks

这一个问题就是对我们之前实现的Graph类的验证。

#### Test GraphPoet

先编写GraphPoet的测试类。

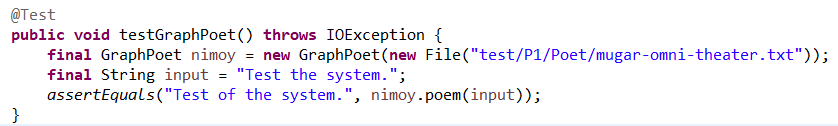
这里用的语料和例子是main中的例子。

使用This is a test of the Mugar Omni Theater sound system.作为语料来形成语料库。

用Test the system.作为输入。

期待其输出应该为Test of the system.。

测试代码如下：



#### Implement GraphPoet

首先应该有一个Graph来存储我们处理后的语料库。

具体的语料处理过程如下（实现于GraphPoet的构造方法中）：

1、将语料库中的句子用空格分开，形成一个一个分开的单词同时将其中的字母都转化为小写，便于下一步的比较。

2、把每个单词作为顶点加入到图的顶点集中（由于图中要求没有重复顶点，因此重复的单词只会出现一次）。

3、之后将每个单词和它在句中的前一个单词之间加一条边：前一个单词🡪当前单词，如果之前没有这样的边相连则将权值设为1，此后如果再次遇到这样的边权值就在此前的基础上加1

4、遍历过全部的单词，语料库就建立完成了。

之后的过程是结合语料库对输入句子进行扩展（实现于**public** String poem(String input)方法中）：

1、将输入的句子根据空格进行拆分成一个一个的单词。

2、对输入句子中的相邻的每一对单词，遍历语料库，寻找是否有单词恰好作为前一个单词的目标顶点和后一个单词的起始顶点，如果有则将其及其与前一个顶点的权值记录下来备用，最终遍历完语料库后将权值最大的这个点加入到原句的两个单词中间，如果遍历完成后仍然没有这样的单词则不插入任何单词。

3、遍历完整个输入句子整个拓展过程就完成了。

#### Graph poetry slam

更新main中的语料库以及输入句子，我在这里将语料库的句子改成了两个句子：

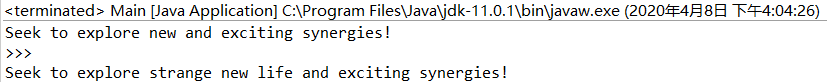
To explore strange new worlds

To seek out new life and new civilizations

将输入句子改成了：

Seek to explore new and exciting synergies!

最终的运行结果为：

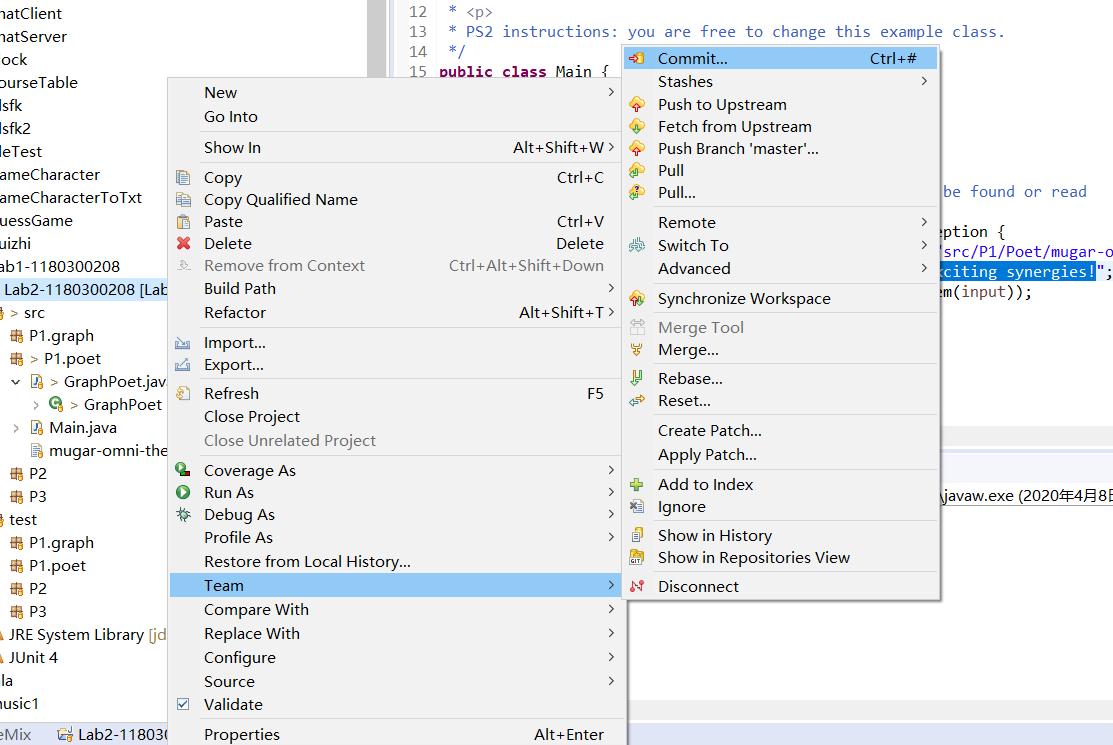


### Before you’re done

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。

使用了eclipse提供的提交方式如下图：



在这里给出你的项目的目录结构树状示意图。

项目的目录结构：

项目名称：Lab2-1180300208

src

P1

graph

ConcreteEdgesGraph.java

ConcreteVerticesGraph.java

Graph.java

poet

GraphPoet.java

Main.java

mugar-omni-theater.txt

test

P1

graph

ConcreteEdgesGraphTest.java

ConcreteVerticesGraphTest.java

GraphInstanceTest.java

GraphStaticTest.java

poet

GraphPoetTest.java

mugar-omni-theater.txt

## Re-implement the Social Network in Lab1

该任务就是用第一个任务完成的Graph来重新实现实验一中P3的FriendshipGraph。用Graph中的方法来实现FriendshipGraph中的addVertex()、addEdge()和getDistance()三个方法。

### FriendshipGraph类

利用Graph中提供的方法来重新实现FriendshipGraph中的方法，具体实现的思路在下面进行详述。

首先实例化一个Graph<Person>的people，以Person的对象作为朋友网络的节点。

下面实现FriendshipGraph中的三个关键方法。

**public** Set<Person> addVertex(Person person)

调用Graph中的add(L Vertex)为朋友关系网络中添加成员，在添加前需要判断是否有重复的人，如果有则退出整个程序。添加返回经过防御性拷贝的成员列表。

**public** Set<Person> addEdge(Person p1, Person p2)

添加p1 🡪 p2的边的关系，由于没有权值要求，则将边的权值统一设置为1来表示有从p1 🡪 p2的朋友关系（p1把p2当作朋友，单方面关系）。使用的是Graph中的set(Person source, Person target, int weight)来添加边。添加后返回经过防御性拷贝的p1的朋友列表。

**public** **int** getDistance(Person p1, Person p2)

寻找两个点之间的距离，使用的依旧是实验一中用到的广度优先搜索，利用队列这一数据结构，从p1出发，直到找到p2为止，这时找到p2必然是最短的距离。

### Person类

使用此前实验一中P3中的Person类即可，但是由于图的结构直接调用Graph中，所以去我取消了之前放在Person中的朋友列表friendList这一rep，只剩下了String类的name用来存储该person的名字。

方法只有用来向新实例化的Person对象传入name的构造方法和得到Person的名字的getName()方法。

### 客户端main()

与实验一的P3中的main()相同。运行结果也相同，这里不再赘述。

### 测试用例

与实验一的P3中的测试用例相同。测试结果也相同，这里不再赘述。

### 提交至Git仓库

使用了eclipse中的team 🡪 commit 🡪 commit and push进行提交。

在这里给出你的项目的目录结构树状示意图。

项目名称：Lab2-1180300208

src

P2

FriendshipGraph.java

Person.java

test

P2

FriendshipGraphTest.java

## Playing Chess

### ADT设计/实现方案

设计了哪些ADT（接口、类），各自的rep和实现，各自的mutability/ immutability说明、AF、RI、safety from rep exposure。

**1、Position**

是否可变: mutability

AF: 坐标为(x, y)的位置

RI: 对x和y没有要求

Safety from rep exposure: 返回时使用了防御性拷贝

rep:

**private** **int** x; 横坐标

**private** **int** y; 纵坐标

method:

**public** Position(**int** x, **int** y) 创建一个横坐标为x纵坐标为y的位置

**public** **boolean** setPosition(**int** x, **int** y) 将横纵坐标设为(x, y)

**public** **int** getX() 返回该位置的横坐标

**public** **int** getY() 返回该位置的纵坐标

**public** Position getPosition() 返回该位置的拷贝

**public** **boolean** equals(Object that) 判断该位置与that位置是否相同

**2、Piece**

**是否可变:** mutability

**AF:** owner的一个类型为type的棋子，位置为pos，如果在棋盘上则状态为true，不在棋盘上状态为false

**RI:** 对owner, type, pos, state没有特别要求

**Safety from rep exposure:** 使用了final关键字

**rep:**

**private** **final** String owner; 该piece的所有者的名称

**private** **final** String type; 该piece的类型

**private** Position pos; 该piece的位置

**private** **boolean** state; 该piece的状态(是否在棋盘上)

**method:**

**public** Piece(String owner, String type, **boolean** state)

新建一个所有者名称为owner类型为type状态为state的棋子

**public** String getOwner() 返回该piece的所有者名称

**public** String getType() 返回该棋子的类型

**public** Position getPos() 返回该棋子的位置

**public** **boolean** isOnBoard() 判断该棋子是否在棋盘上

**public** **boolean** setPos(**int** x, **int** y) 重新修改棋子的位置到(x, y)

**public** **void** setState(**boolean** state) 改变棋子的状态

**3、Player**

**是否可变:** mutability

**AF:** 一个名字为name的玩家，其棋子集合为pieceSet，走棋历史为history

**RI:** 名字name不能为空

**Safety from rep exposure:** 使用了final关键字

**rep:**

**private** **final** String name; 该玩家的名字

**private** **final** Set<Piece> pieceSet = **new** HashSet<>();

该玩家拥有的棋子集合

**private** **final** List<String> history = **new** ArrayList<>();

该玩家的走棋历史

**method:**

**public** Player(String name) 新建一个名字为name的player

**public** **boolean** addPiece(Piece piece)

向player添加棋子，若已有该棋子或者该棋子的owner与该player的name不同则返回false，如果没有则将其加到棋子集合中并返回true

**public** **int** pieceNumber() 返回该player当前在棋盘上的棋子数目

**public** **void** addHistory(String history)

将history加到该player的走棋历史中

**public** String getName() 返回该player的名字

**public** Set<Piece> getPieceSet() 返回该player的棋子集合

**public** List<String> getHistory() 获得该player的走棋历史

**4、Board**

**是否可变:** mutability

**AF:** 一个n\*n的棋盘，棋子放在格中或者格点处

**RI:** 如果是格点处放置棋子，应该有(n+1)\*(n+1)个放置位置；如果是格中放置棋子，应该有n\*n个放置位置

**Safety from rep exposure:** 使用了final标签不直接返回board而是直接打印，所以RI一旦确定讲不会被改变，无需对representation进行检查

**rep:**

**private** **final** **boolean** isCrossPlace; 该棋盘上的棋子是否落在格点上

**private** **final** **int** size; 该棋盘的大小(方形棋盘每行的方格数)

**private** **final** **int** actualSize; 该棋盘的实际大小(一行可容纳的棋子数)

**private** Piece[][] pieces; 棋盘存放棋子的二维数组

**private** **boolean**[][] hasPiece; 判断棋盘某一位置是否有棋子

**method:**

**public** Board(**int** size, **boolean** isCrossPlace)

新建一个大小、棋子放置是否在格点已知的棋盘

**public** Piece getPiece(**int** x, **int** y) 获得在(x, y)位置的棋子

**public** **boolean** xyHasPiece(**int** x, **int** y) 判断(x, y)处是否有棋子

**public** **void** setPiece(Piece piece, **int** x, **int** y)

将piece棋子放置在棋盘的(x, y)处

**public** **void** removePiece(**int** x, **int** y) 移除在(x, y)处的棋子

**public** **void** changePosition(**int** x1, **int** y1, **int** x2, **int** y2)

将(x1, y1)处的棋子移到(x2, y2)处

**public** **int** getActualSize() 获得每一行可容纳的棋子数

**public** **int** getSize() 获得每一行的方格数

**public** **boolean** getType() 获得棋子放置在棋盘的位置是格点还是格中

**public** **void** printBoard() 将棋盘打印到命令行

**5、Action (**interface**)**

**public** **static** Action empyt(String game)

工厂方法，用来实例化指定的类（chess或go）

**（1）ChessAction**

**是否可变: im**mutability

**AF:** chess游戏，由两位玩家和一个棋盘构成

**RI:** 两位玩家不能相同，名字也不能为空

**Safety from rep exposure:** 不将玩家和棋盘信息暴露出去

**rep:**

**private** Player player1; 玩家一

**private** Player player2; 玩家二

**private** Board board; 国际象棋棋盘

**method:**

**public** **void** createPlayer1(String player1) 创建玩家一

**public** **void** createPlayer2(String player2) 创建玩家二

**public** **void** createBoard() 创建国际象棋棋盘

**public** **boolean** placePiece(Player player, Piece piece, **int** x, **int** y)

国际象棋没有这个操作，提示后直接返回false

**public** **boolean** removePiece(Player player, **int** x, **int** y)

国际象棋没有这个操作，提示后直接返回false

**public** **boolean** movePiece(Player player,**int** x1,**int** y1,**int** x2,**int** y2)

玩家player的操作：将己方位于(x1, y1)的棋子移动到(x2, y2)的位置

要注意的情况： 给的两个坐标是否在棋盘范围内

起始位置和终点是否有棋子

起点位置处的棋子是否为己方棋子

**public** **boolean** eatPiece(Player player,**int** x1,**int** y1,**int** x2, **int** y2)

玩家player的操作：用己方位于(x1, y1)的棋子吃掉对方位于到(x2, y2)的棋子

要注意的情况： 给的两个坐标是否在棋盘范围内

起始位置和终点是否有棋子

起点位置处的棋子是否为己方棋子

终点位置处的棋子是否为对方棋子

**public** **void** printBoard() 打印此时棋盘

**public** Player getPlayer1() 获得玩家一

**public** Player getPlayer2() 获得玩家二

**public** Piece getPieceAtXY(**int** x, **int** y) 得到(x, y)处的棋子

**public** **void** printHistory(Player player) 打印player的走棋历史

**（2）GoAction**

**是否可变: im**mutability

**AF:** go游戏，由两位玩家和一个棋盘构成

**RI:** 两位玩家不能相同，名字也不能为空

**Safety from rep exposure:** 不将玩家和棋盘信息暴露出去

**rep:**

**private** Player player1; 玩家一

**private** Player player2; 玩家二

**private** Board board; 围棋棋盘

**method:**

**public** **void** createPlayer1(String player1) 创建玩家一

**public** **void** createPlayer2(String player2) 创建玩家二

**public** **void** createBoard() 创建围棋棋盘

**public** **boolean** placePiece(Player player, Piece piece, **int** x, **int** y)

玩家player的操作：将己方的piece棋子放置到(x, y)的位置

要注意的情况： 给的坐标是否在棋盘范围内

piece是否是己方的棋子

(x, y)处是否已经有棋子

piece是否是已经在棋盘上的棋子

**public** **boolean** removePiece(Player player, **int** x, **int** y)

玩家player的操作：将对方的位置在(x, y) 的棋子放置移除

要注意的情况： 给的坐标是否在棋盘范围内

piece是否是对方的棋子

(x, y)处是否有棋子

**public** **boolean** movePiece(Player player,**int** x1,**int** y1,**int** x2,**int** y2)

围棋没有这个操作，提示后直接返回false

**public** **boolean** eatPiece(Player player,**int** x1,**int** y1,**int** x2, **int** y2)

围棋没有这个操作，提示后直接返回false

**public** **void** printBoard() 打印此时棋盘

**public** Player getPlayer1() 获得玩家一

**public** Player getPlayer2() 获得玩家二

**public** Piece getPieceAtXY(**int** x, **int** y) 得到(x, y)处的棋子

**public** **void** printHistory(Player player) 打印player的走棋历史

**6、Game**

**是否可变: im**mutability

**AF:** 作为ChessAction和GoAction的包装供MyChessAndGoGame调用

**RI:** 一经创建就不会被修改

**Safety from rep exposure:** 使用final关键字

**rep: private** **final** Action action; 客户端程序要用到的操作

**method:** 除构造方法外，其他方法均为ChessAction和GoAction的包装，不再详述

**public** Game(String game, String player1, String player2)

用根据游戏名创建一个新的游戏，并传入两个玩家的名称

**public** Player getPlayer1()

**public** Player getPlayer2()

**public** Piece getPiece(**int** x, **int** y)

**public** **boolean** placePiece(Player player, Piece piece, **int** x, **int** y)

**public** **boolean** removePiece(Player player, **int** x, **int** y) {

**public** **boolean** movePiece(Player player,**int** x1,**int** y1,**int** x2,**int** y2)

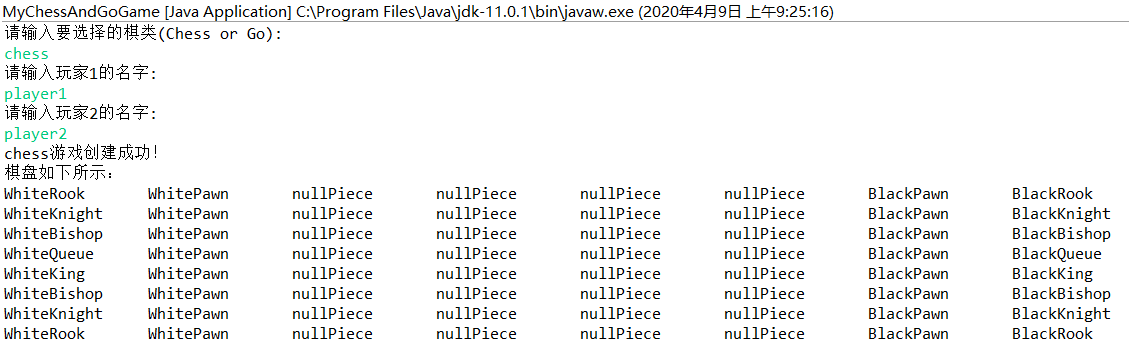
**public** **boolean** eatPiece(Player player,**int** x1,**int** y1,**int** x2, **int** y2)

**public** **void** printBoard()

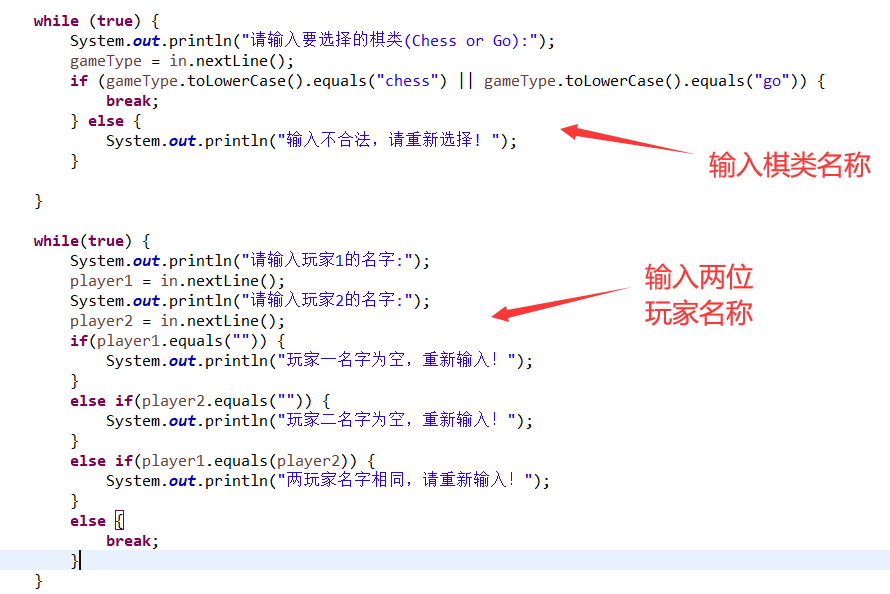
**public** **void** printHistory(Player player)

### 主程序MyChessAndGoGame设计/实现方案

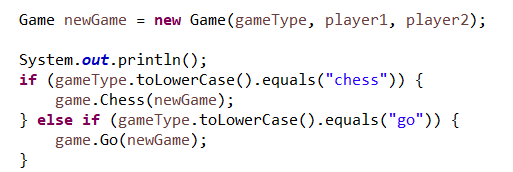
首先main方法读入游戏的名称（Chess或者Go）和两名玩家的名称，用这三个参数来建立一个新的游戏。为使代码整洁，将Chess和Go的操作放在不同的方法中，根据不同游戏类型进行调用。初始化游戏过程如下图所示：



对输入棋类和玩家的处理如下：

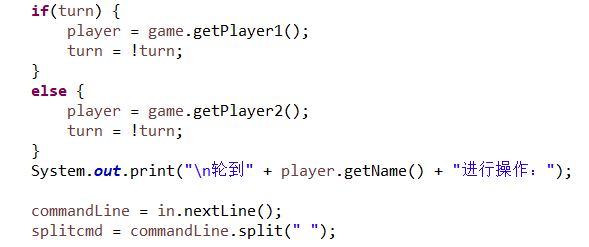


随后根据要实现的游戏类别以及玩家名称建立新的游戏，同时调用该游戏的操作方法，处理如下：



下面对Chess(Game game)和Go(Game game)两个方法进行解析。

除了命令不同和调用的操作方法不同外，两个方法对输入命令的解析和处理方式是一样的，都是在每一轮选择不同的玩家作为命令的发出者，将该玩家输入的命令通过空格拆分开，通过第一个单词来判断调用的方法，用剩下的作为参数来实现具体的棋子操作。对于两种游戏来说处理的方式是完全一样的，处理的代码如下图所示：



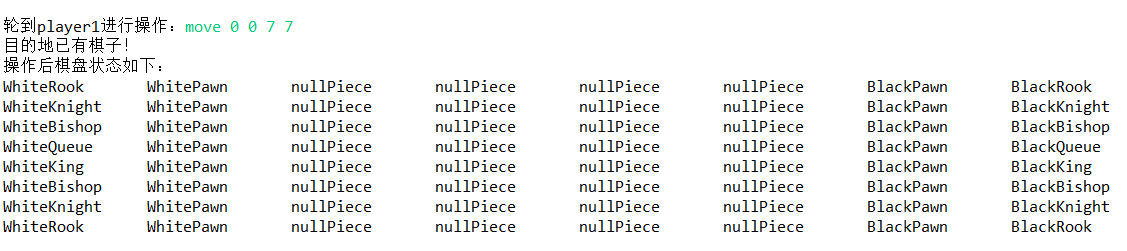
下面对两种游戏的操作分别进行说明：

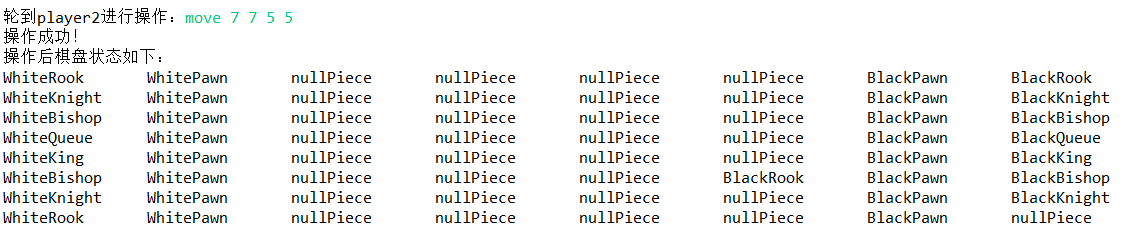
Chess的移子操作:

命令行输入格式为: move x1 y1 x2 y2

其意义为: 当前玩家将自己的在(x1, y1)处的棋子移到(x2, y2)处

操作实例如下:



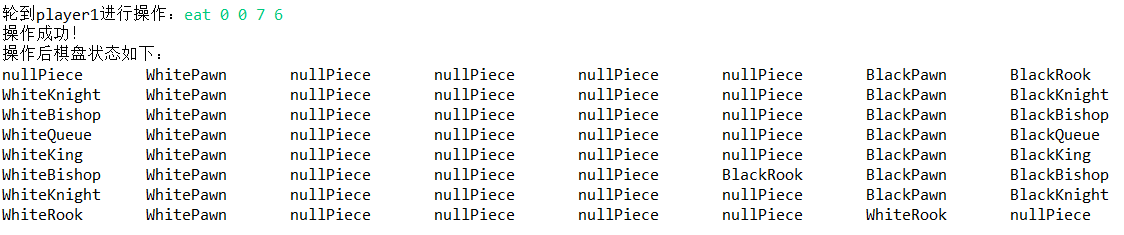


Chess的吃子操作:

命令行输入格式为: eat x1 y1 x2 y2

其意义为: 当前玩家用自己在(x1, y1)处的棋子替换掉对方在(x2, y2)处的棋子

操作实例如下:

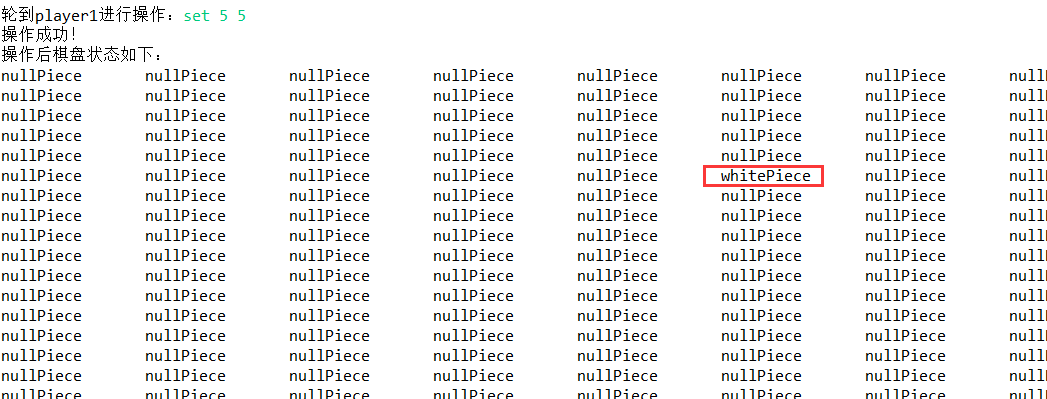


Go的落子操作:

命令行输入格式为: set x y

其意义为: 将当前玩家尚未在棋盘上的棋子放置在(x, y)处

操作实例如下: （只截取了部分棋盘进行展示）



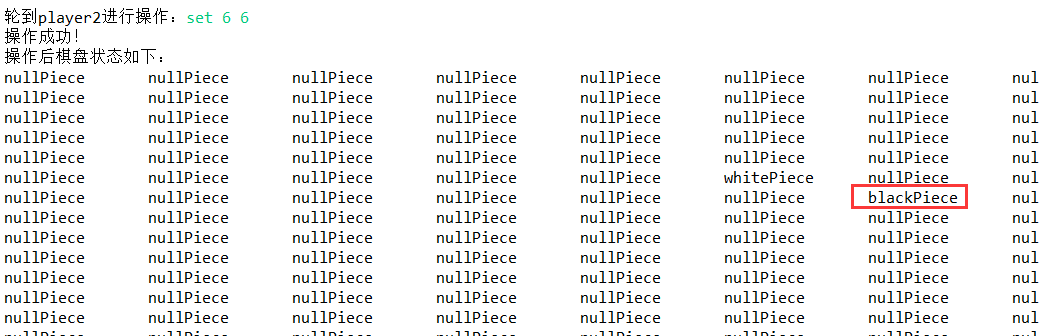
Go的提子操作:

命令行输入格式为: remove x y

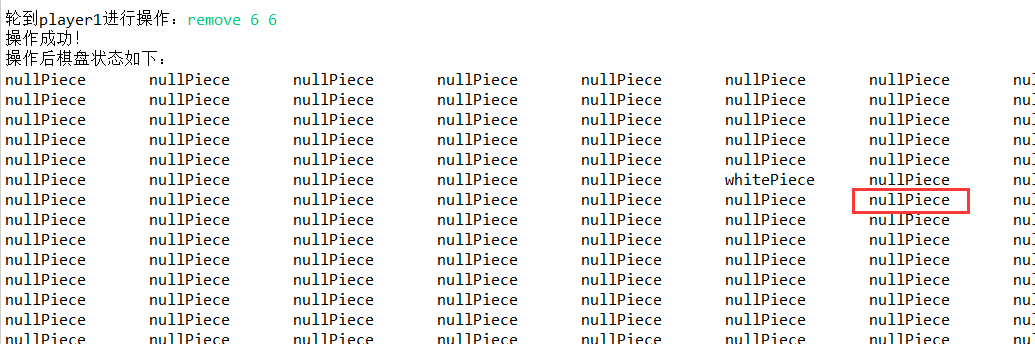
其意义为: 将对方玩家在棋盘上位置为(x, y)的棋子拿下棋盘

操作实例如下:

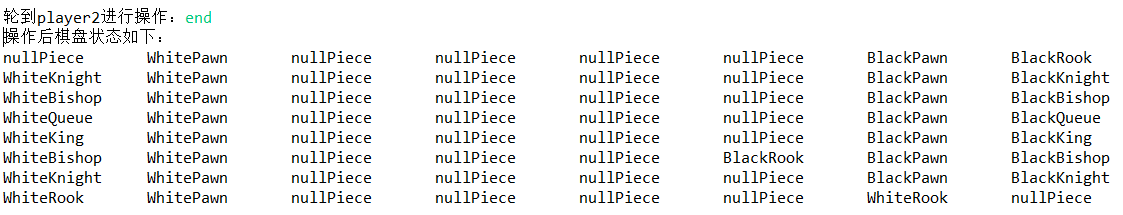
提子前：



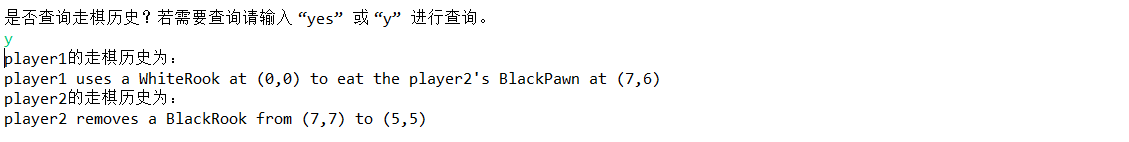
提子后：



双方任意一方输入end后游戏将被退出，如下图所示，会给出最后棋盘的样子。



之后会提示是否查询走棋历史，输入”yes”或者”y”将给出双方成功进行的走棋历史（即如果有步骤走棋失败将不会记录在走棋历史中），实例如下：



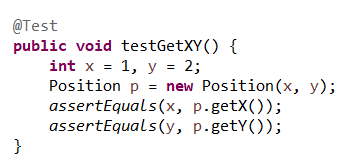
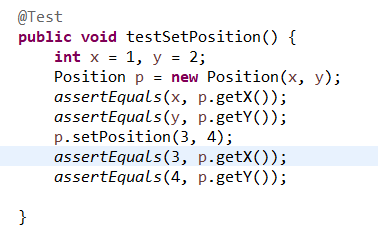
### ADT和主程序的测试方案

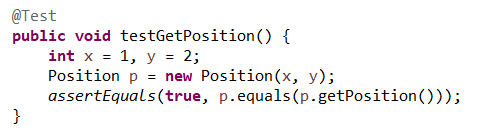
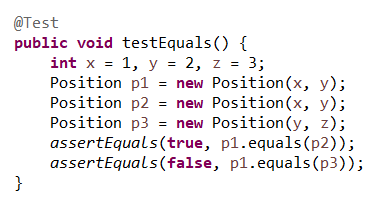
介绍针对各ADT的各方法的测试方案和testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

**1、PositionAction**

Test strategy: 按照逻辑对其进行测试即可，对x, y没有特别要求。

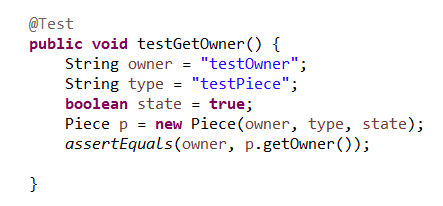
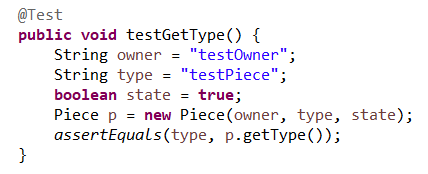


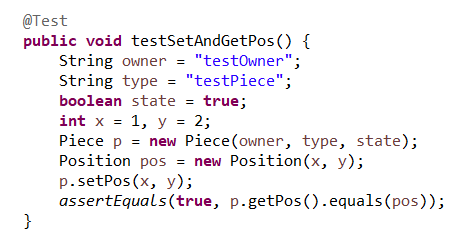
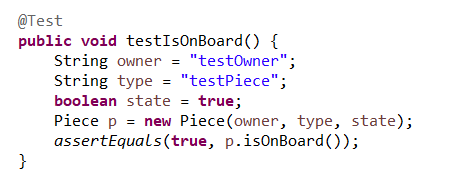
 

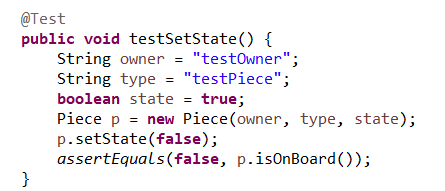
**2、PieceTest**

Test strategy: 由于对owner, owner, type, pos, state没有特别要求

因此简单测试逻辑即可

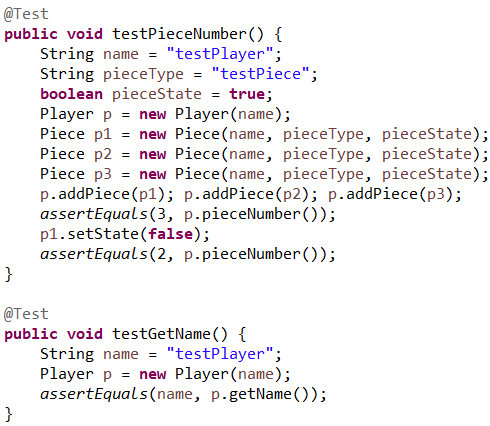
 



**3、PlayerTest**

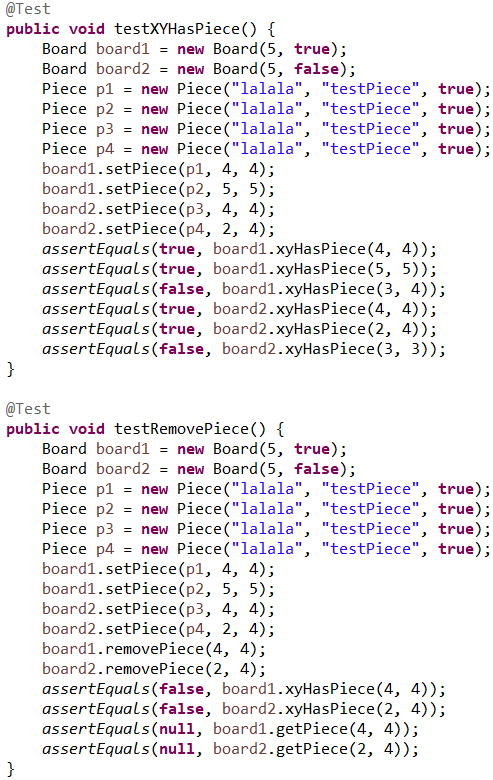
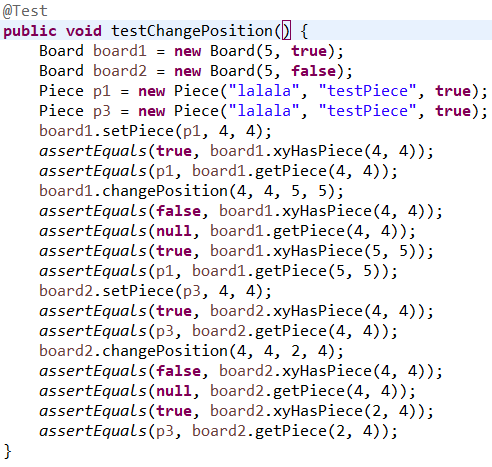
Testing strategy: 在添加棋子时要注意该棋子是否重复添加

该棋子是否属于该player

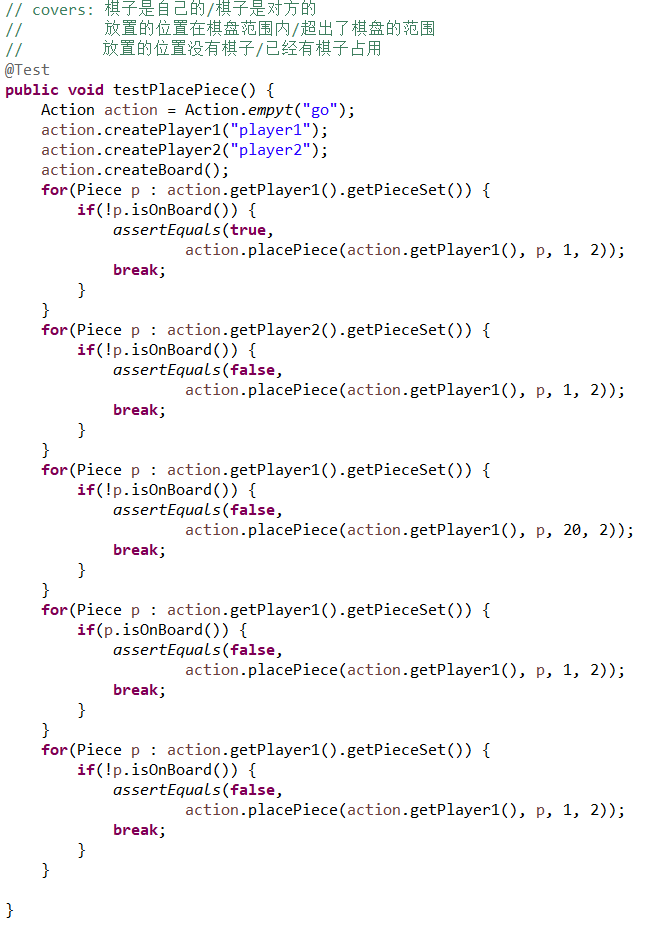
**4、BoardTest**

Testing strategy: 应该考虑不同状态的两种棋盘: 棋子落在方格中以及棋子落在格点处。而对棋盘上的棋子进行操作时应该考虑到位置是否在棋盘的范围之外，以及要操作的位置是否有棋子由更高级的操作进行判断，这里不做非法测试。

**5、ChessActionTest**

Testing strategy: 各项操作应当符合实验手册中给定的规则，每一项的设计思路会展示在每一个方法的测试之前。

**6、GoActionTest**

Testing strategy: 各项操作应当符合实验手册中给定的规则，每一项的设计思路会展示在每一个方法的测试之前。

**7、GameTest**

由于game只是ChessAction和GoAction的操作的包装，所有的操作只是调用ChessAction和GoAction的方法，因此无需进行重复测试。

项目名称：Lab2-1180300208

src

P3

Action.java

Board.java

ChessAction.java

Game.java

GoAction.java

MyChessAndGoGame.java

Piece.java

Player.java

Position.java

test

P3

BoardTest.java

ChessActionTest.java

GameTest.java

GoActionTest.java

PieceTest.java

PlayerTest.java

PositionTest.java

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 |
| 2020.03.16 | 20:00 – 20:30 | 创建目录 | 按计划完成 |
| 2020.03.21 | 08:00 – 11:00 | 阅读实验手册，研读网站内容，并尝试进行实验 | 按计划完成 |
| 2020.03.21 | 15:00 – 18:00 | 完成P1的1、2部分 | 按计划完成 |
| 2020.03.22 | 08:00 – 11:00 | 完成P1的3部分 | 按计划完成 |
| 2020.03.22 | 15:00 – 17:00 | 完成P2 | 按计划完成 |
| 2020.03.23 | 14:00 – 18:00 | 完成P1、P2中文档部分的编写 | 未按计划完成 |
| 2020.04.01 | 20:00 – 23:30 | 开始完成P3 | 按计划完成 |
| 2020.04.03 | 09:00 – 12:00 | 完成P1、P2的文档 | 按计划完成 |
| 2020.04.03 | 15:00 – 20:30 | 完成P3主体部分 | 按计划完成 |
| 2020.04.06 | 17:00 – 22:40 | 为P3添加测试 | 按计划完成 |
| 2020.04.09 | 13:00 – 17:00 | 最终完成实验报告 | 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 |
| 实验初期没有明白mutable与immutable的区别 | 通过老师后续的讲解明白了二者的区别 |
| 对AF、RI等概念不理解 | 通过老师后续的讲解掌握了这方面的知识 |
| 完成P3时对于各个类和接口的组织感到困惑 | 通过网上往届学长的分享的经验，加上自己的理解完成了各个ADT的组织，设计出了一套适用于棋类游戏的ADT |

# 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

经验：面对一个规模较大的项目时，有一个有条理的目录十分重要

教训：写代码前一定要有一个整体的规划，然后再向构建好的框架中填充代码，这样会极大地提高编码效率，能够规避直接编码带来的思考不成熟的问题

感想：这次的实验无论从代码量还是从难度上来讲，都远远高于实验一，对我们的编码能力提出了更高的要求，这次试验的完成，对我们来说是一次考验，同时也是一次很好的提升自己编程能力和软件构造能力的机会。

## 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，更加注重信息的隐藏与封装，更加注重不同情境下的复用，对同样的ADT可能还有不同的实现方式，与直接面对应用场景编程相比，面对ADT的编程更加独立，应用面更广。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型的编程和不使用泛型的编程在我看来没有大的差别，除了一些语句会有特殊的要求。但使用泛型的编程的使用面更广，可复用性更强。

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？

优势是可以通过测试用例不断指导自己编程的实现，能够更轻松地达到用户的需求。目前来讲，我还不太适应这种方式，但也在慢慢地适应。

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？

好处就是一次实现可以多次多场景使用，减少了编程的代码量，无需对任何相关的问题都从头开始进行编码。

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？

适应了从具体应用场景到ADT的抽象映射，P3要求自主设计，我认为还是有一定难度，对我个人而言是一个挑战。

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？

意义在于在大规模编程的时候能够减少bug的发生率，防止后期出现bug后发生难以排除的尴尬。我愿意在以后的编程中同样这样做，这样看似浪费时间，但正所谓磨刀不误砍柴工，这样的做法恰恰可以提高编程效率。

1. 关于本实验的工作量、难度、deadline。

本实验的工作量非常大，有一定难度，DDL在2020.4.12 23:55，时间还算充裕，可以从容完成。

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

软件构造为我们展示了编程的一个全新的角度，更加贴近于实际的应用场景，对我们帮助很大。

建议：无