



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

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	郭茁宁
学号	1183710109
班号	1837101
电子邮件	gzn00417@foxmail.com
手机号码	13905082373

# 0 目录

<b>0 目录</b> .....	<b>2</b>
<b>1 实验目标概述</b> .....	<b>1</b>
<b>2 实验环境配置</b> .....	<b>2</b>
2.1 安装 Eclemma .....	2
2.2 GitHub Lab2 仓库的 URL 地址.....	3
<b>3 实验过程</b> .....	<b>4</b>
<b>3.1 Poetic Walks</b> .....	<b>4</b>
3.1.1 Get the code and prepare Git repository.....	4
3.1.2 Problem 1: Test Graph <String>.....	4
3.1.3 Problem 2: Implement Graph <String>.....	4
3.1.3.1 Implement ConcreteEdgesGraph.....	5
3.1.3.2 Implement ConcreteVerticesGraph.....	6
3.1.4 Problem 3: Implement generic Graph<L>.....	8
3.1.4.1 Make the implementations generic.....	8
3.1.4.2 Implement Graph.empty().....	8
3.1.5 Problem 4: Poetic walks.....	8
3.1.5.1 Test GraphPoet.....	8
3.1.5.2 Implement GraphPoet.....	8
3.1.5.3 Graph poetry slam.....	9
3.1.6 Before you're done.....	9
<b>3.2 Re-implement the Social Network in Lab1</b> .....	<b>10</b>
3.2.1 FriendshipGraph 类 .....	10
3.2.2 Person 类 .....	11
3.2.3 客户端 main() .....	11
3.2.4 测试用例.....	13
3.2.4.1 简单图测试.....	13
3.2.4.2 复杂图测试.....	13
3.2.4.3 Junit 测试结果 .....	14
3.2.5 提交至 Git 仓库.....	14
<b>3.3 Playing Chess</b> .....	<b>15</b>
3.3.1 ADT 设计/实现方案 .....	16
3.3.1.1 interface Game.....	16
3.3.1.1.1 class chessGame .....	19

3.3.1.1.2 class goGame .....	20
3.3.1.2 class Board.....	21
3.3.1.3 class Player.....	24
3.3.1.4 class Position.....	26
3.3.1.5 class Piece.....	27
3.3.1.6 interface Action.....	28
3.3.1.6.1 class chessAction .....	30
3.3.1.6.2 class goAction .....	31
3.3.2 主程序 MyChessAndGoGame 设计/实现方案.....	32
3.3.2.1 游戏流程图.....	32
3.3.2.2 程序演示.....	33
3.3.2.3 功能实现.....	35
3.3.3 ADT 和主程序的测试方案 .....	38
3.3.3.1 testInit().....	38
3.3.3.2 testPiece().....	38
3.3.3.3 testBoard.....	39
3.3.3.4 testPosition().....	40
3.3.3.5 testPlayer().....	40
3.3.3.6 testPut().....	41
3.3.3.7 testMove().....	41
3.3.3.8 testCaptureAndPut().....	42
3.3.3.9 ChessTest 测试结果 .....	43
3.3.3.10 GoTest 测试结果 .....	43
<b>4 实验进度记录 .....</b>	<b>44</b>
<b>5 实验过程中遇到的困难与解决途径 .....</b>	<b>45</b>
<b>6 实验过程中收获的经验、教训、感想 .....</b>	<b>46</b>
6.1 实验过程中收获的经验教训 .....	46
6.2 针对以下方面的感受 .....	46

# 1 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：

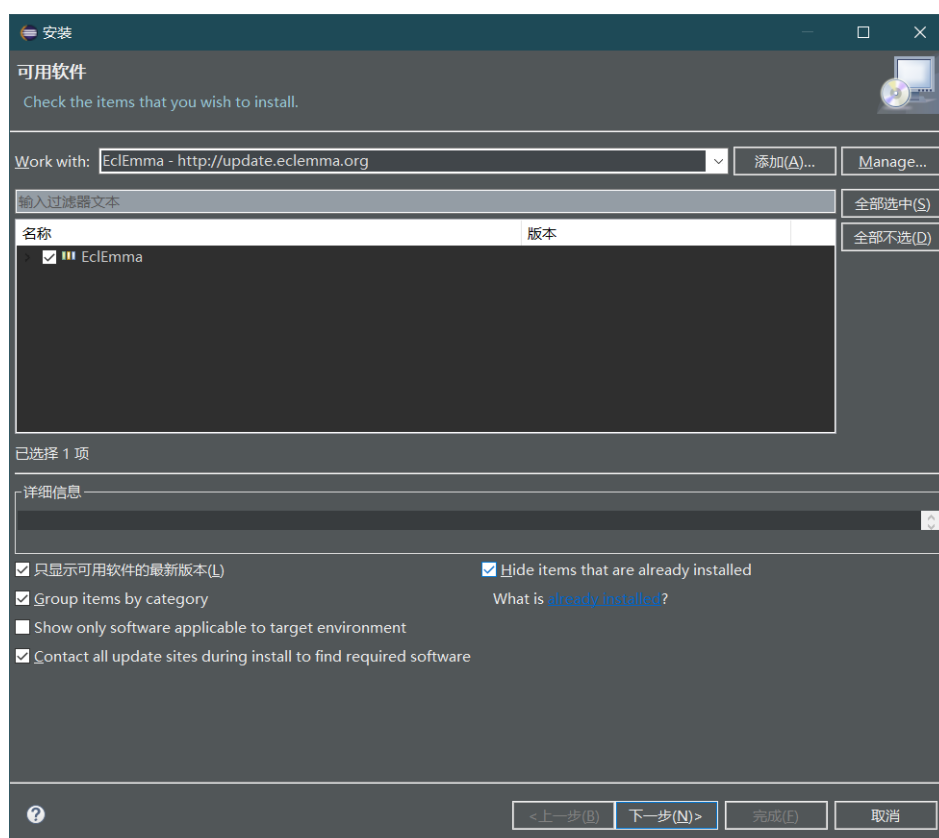
- 针对给定的应用问题，从问题描述中识别所需的 ADT；
- 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；
- 根据 ADT 的规约设计测试用例；
- ADT 的泛型化；
- 根据规约设计 ADT 的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
- 使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
- 测试 ADT 的实现并评估测试的覆盖度；
- 使用 ADT 及其实现，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。

## 2 实验环境配置

### 2.1 安装 EclEmma

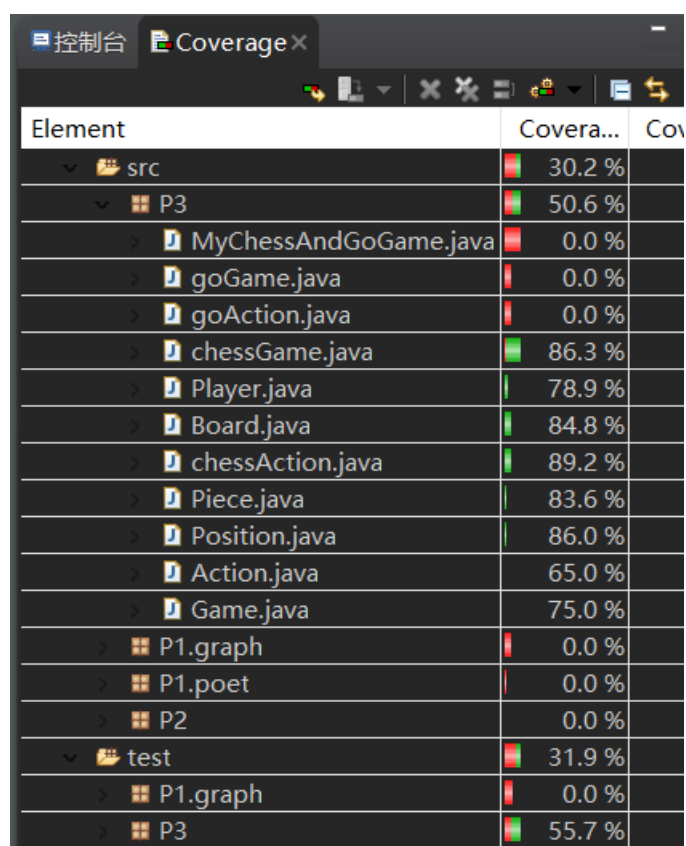
依据 <https://www.eclemma.org/installation.html> 内容，从更新站点进行安装。

- 从 Eclipse 菜单中选择帮助 → 安装新软件；
- 在“安装”对话框中，在“工作日期”字段中输入 <http://update.eclemma.org/>；



- 检查最新的 EclEmma 版本，然后按“下一步”；
- 重启 eclipse，即可在 java 的透视图工具栏中找到 coverage 启动器，表示安装成功。
- 使用效果





Element	Covera...	Cov
src	30.2 %	
P3	50.6 %	
MyChessAndGoGame.java	0.0 %	
goGame.java	0.0 %	
goAction.java	0.0 %	
chessGame.java	86.3 %	
Player.java	78.9 %	
Board.java	84.8 %	
chessAction.java	89.2 %	
Piece.java	83.6 %	
Position.java	86.0 %	
Action.java	65.0 %	
Game.java	75.0 %	
P1.graph	0.0 %	
P1.poet	0.0 %	
P2	0.0 %	
test	31.9 %	
P1.graph	0.0 %	
P3	55.7 %	

## 2.2 GitHub Lab2 仓库的 URL 地址

<https://github.com/ComputerScienceHIT/Lab2-1183710109>

## 3 实验过程

### 3.1 Poetic Walks

该任务主要是实验一个图的模块，并基于此使用。

- 完善 Graph 接口类，并运用泛型的思想，将 String 拓展为泛型 L 类；
- 实现 Graph 类的方法：add、set、remove、vertices、sources、targets；
- 利用实现的 Graph 类，应用图的思想，实现 GraphPoet 类，如果输入的文本的两个单词之间存在桥接词，则插入该桥接词；若存在多个单一桥接词，则选取边权重较大者。

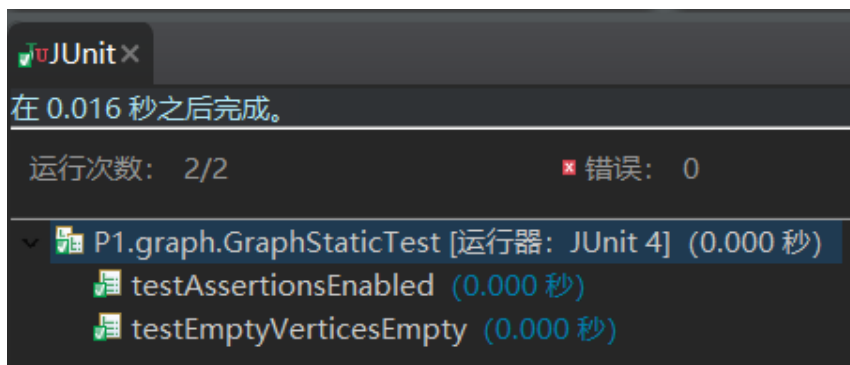
#### 3.1.1 Get the code and prepare Git repository

```
git clone https://github.com/rainywang/Spring2020_HITCS_SC_Lab2.git
```

```
guozn@DESKTOP-RQFQUOR MINGW64 ~/Desktop
$ git clone https://github.com/rainywang/Spring2020_HITCS_SC_Lab2.git
Cloning into 'Spring2020_HITCS_SC_Lab2'...
remote: Enumerating objects: 57, done.
remote: Total 57 (delta 0), reused 0 (delta 0), pack-reused 57
Unpacking objects: 100% (57/57), done.
```

#### 3.1.2 Problem 1: Test Graph <String>

测试静态方法生成 String 类型的 Graph。



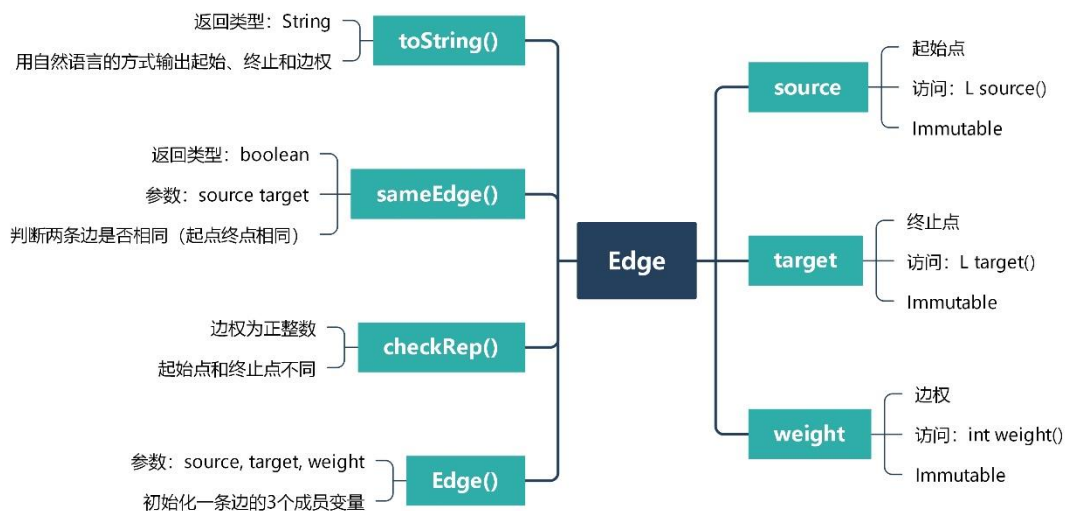
#### 3.1.3 Problem 2: Implement Graph <String>

该部分要求重写 Graph 里的方法，分别以点为基础的图和以边为基础的图。两种类型的图都经过同一个实例测试。

### 3.1.3.1 Implement ConcreteEdgesGraph

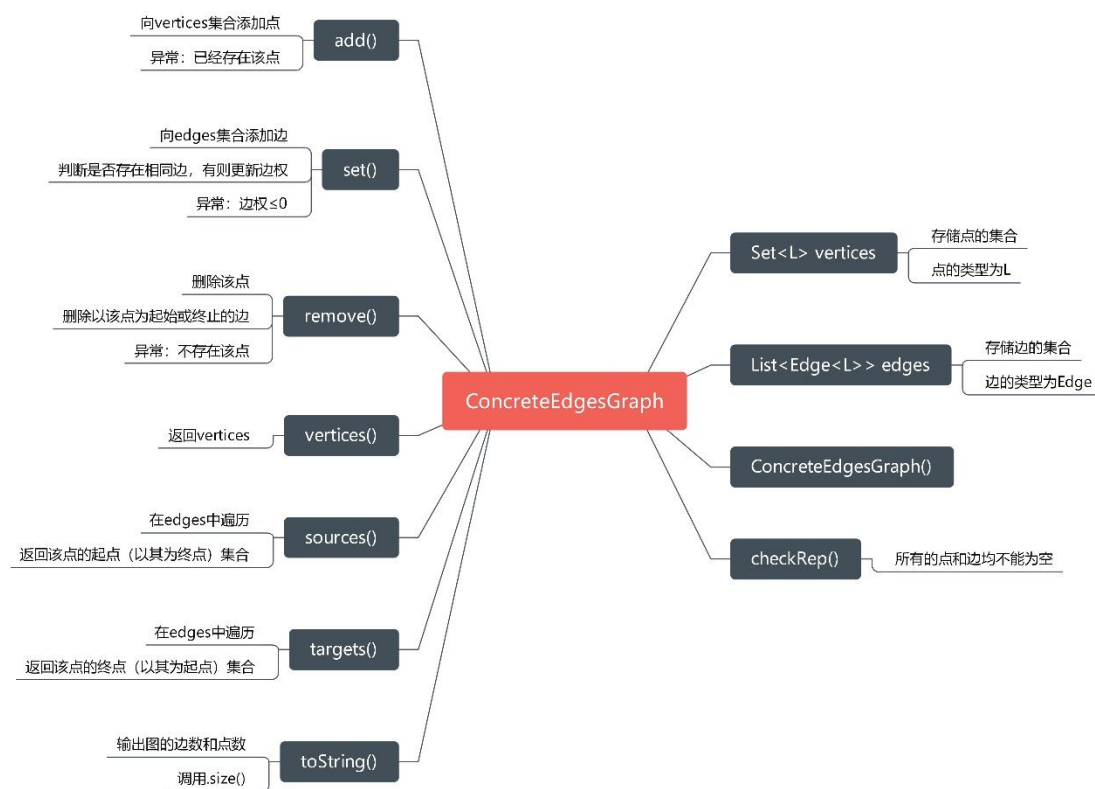
#### Edge 实现

Edge 的功能主要为存储边的 3 个信息。此外，为了 Graph 实现方便，增加了判断两条边是否相等的方法。



#### ConcreteEdgesGraph 实现

该类以 Edge 为基础重写 Graph<L>，用集合来存储点和边 (Edge)，每有 Edge 的增加就会影响到集合的更改，而点的删除也需要在集合中查询匹配。





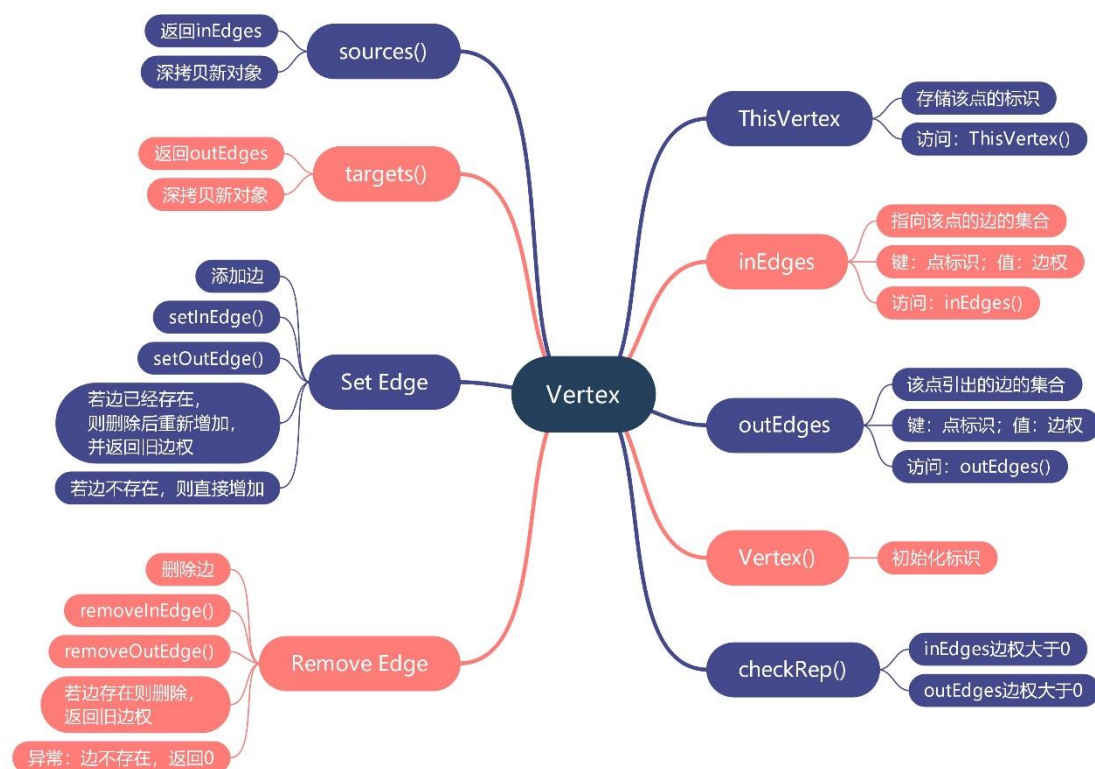
JUnit 测试:

All	11	Passed	11
P1.graph.ConcreteEdgesGraphTest			
> testToString	Passed	0s	
> testStructure	Passed	0s	
> testSourceTarget	Passed	0s	
> testAdd	Passed	0s	
> testSet	Passed	0s	
> testVertices	Passed	0s	
> testWhole	Passed	0s	
> testInitialVerticesEmpty	Passed	0s	
> testAssertionsEnabled	Passed	0s	
> testSetNegW	Passed	0.03s	
> testRemove	Passed	0s	

## 3.1.3.2 Implement ConcreteVerticesGraph

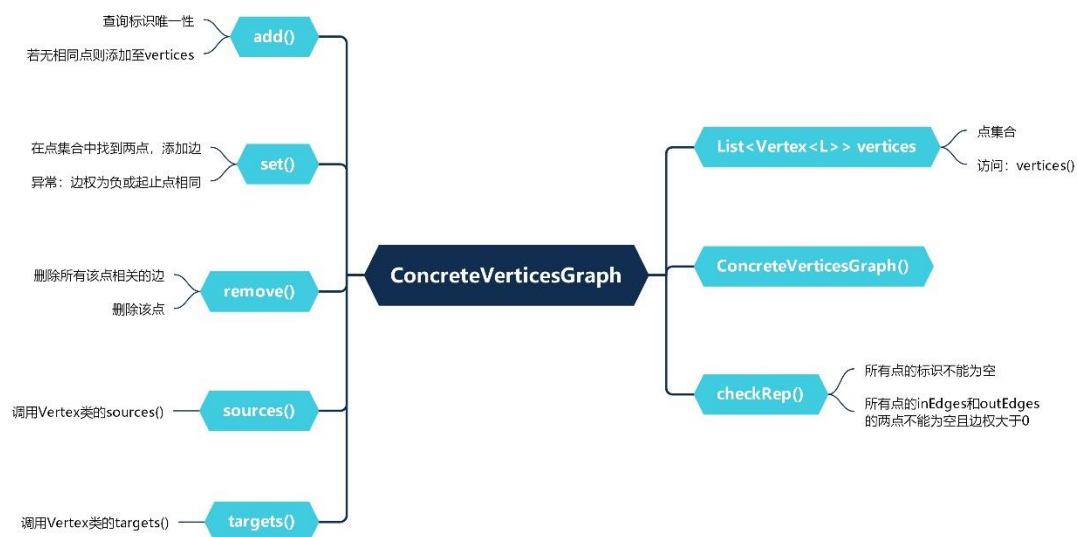
Vertex 实现

Vertex 是点的抽象类，包含 3 个信息：点的标识、指向该点的边、由该点引出的边。Vertex 需要能访问这 3 个信息，以及增加/删除进边/出边。



### ConcreteVerticesGraph 实现

ConcreteVerticesGraph 是以点为基础的图，每个点通过唯一的标识进行区分，set 和 remove 都依赖与 Vertex 类中的添加和删除操作，sources 和 targets 也调用了 Vertex 类的方法。



### Junit 测试:

P1.graph.ConcreteVerticesGraphTest			
> testVertices	Passed	0s	
> testToString	Passed	0s	
> testSourceTarget	Passed	0s	
> testAdd	Passed	0s	
> testSet	Passed	0s	
> testWhole	Passed	0s	
> testInitialVerticesEmpty	Passed	0s	
> testAssertionsEnabled	Passed	0s	
> testSetNegW	Passed	0.02s	
> testRemove	Passed	0s	

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

#### 3.1.4.1 Make the implementations generic

在程序中选择“重构”或选择“String”并选择更改所有匹配项（要注意 toString），即可实现泛化类型。

#### 3.1.4.2 Implement Graph.empty()

使 Graph.empty() 能返回一个新的空实例。代码如下：

```
public static Graph<String> empty() {  
    return new ConcreteEdgesGraph();  
}
```

### 3.1.5 Problem 4: Poetic walks

#### 问题简述：

给定一个语料库 corpus，根据 corpus 中的文本生成一个单词图，然后给定一条语句输入，在图中搜索词之间的关系，自动补全语句中可能可以完善的部分。

图的构建规则是，在 corpus 中，对每一个不一样的单词看作一个顶点，相邻的单词之间，建立一条有向边，相邻单词对出现的次数，作为这条有向边的权值。在输入信息补全时，对相邻单词 A 和 B 做检查，如果存在一个单词 C，在图中可以由前一个单词 A 通过这个单词 C 到达单词 B，那么就在 A 和 B 之间补全 C，补全的优先级按照权值越大者优先。

#### 3.1.5.1 Test GraphPoet

在基于预设的测试用例基础上，增加等价类划分的多种情况。

等价类划分：两个单词之间不存在连接词，两个单词之间只有一个连接词，两个单词之间有多个连接词。

此外还要注意句末的句号，测试当一个句子最后一个词是“桥”的一端。

#### 3.1.5.2 Implement GraphPoet

##### **1. 表示不变量和检查不变量**

该应用中的不变量是所有的点都不为空。

##### **2. 构造函数**

用文件输入单词，String.split() 分割为数组，通过 String.toLowerCase() 小写化。

接下来构建图，相邻的单词加边。首先要在加边前通过 Graph.add() 加点，加边时要判断是否存在：由于 Graph.set() 能返回之前加的边的值，以此来判断是否存在，存在则在之前的值加一（之前的边的值保存为 lastEdgeWeight）。

##### **3. Poem(String input)**

当相邻两个单词任意一个不在之前创建的图里，则将后者单词加入即可（再加个空格）当存在时，由于 Bridge 长度只能为 2，所以：分别求两个单词的 sources 和 targets，将该 Map 转换为 Set 求交集；若交集为空，则无桥，若交集不空，

则在交集中找最短的桥（可以在 Map 的 value 中查询 weight）。

### 3.1.5.3 Graph poetry slam

样例 “This is a test of the Mugar Omni Theater sound system.” 进行测试，测试成功。

```
Test the system.
>>>
Test of the system.
```

修改样例为 “This is a the Mugar system Omni Theater sound system test of the.”，测试成功。该样例用于测试极端情况。

```
Test the system.
>>>
Test of the mugar system.
```

### JUnit 测试



### 3.1.6 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/#before_youre_done) 的说明，检查你的程序。

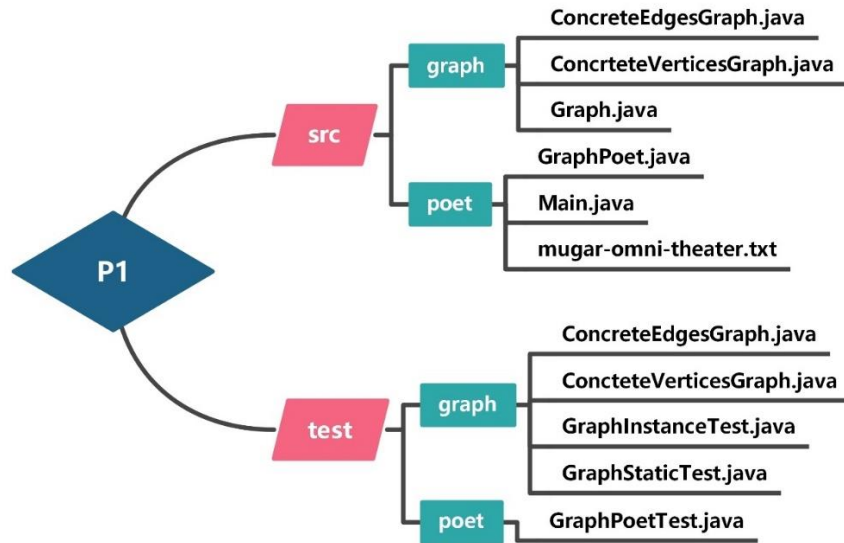
如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

```
guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git add *.docx

guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git commit -m "Update Report"
[master 2a73266] Update Report
1 file changed, 0 insertions(+), 0 deletions(-)

guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 71.81 KiB | 128.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:ComputerScienceHIT/Lab2-1183710109.git
1c1eb77..2a73266 master -> master
```

在这里给出你的项目的目录结构树状示意图。



## 3.2 Re-implement the Social Network in Lab1

这部分任务就是用我们在 3.1 中写的 ADT，把第一次实验中的 FriendshipGraph 重新实现一遍，图中的节点仍然是 Person 类型，所以泛型 L 一律为 Person。而对于已经写好的 FriendshipGraph 中的方法，要用 3.1 中的 Graph ADT 中的方法来实现它们。

### 3.2.1 FriendshipGraph 类

#### Graph<Person> graph:

直接调用 Graph 的静态方法 `empty()` 生成一个空的图。

#### boolean addVertex():

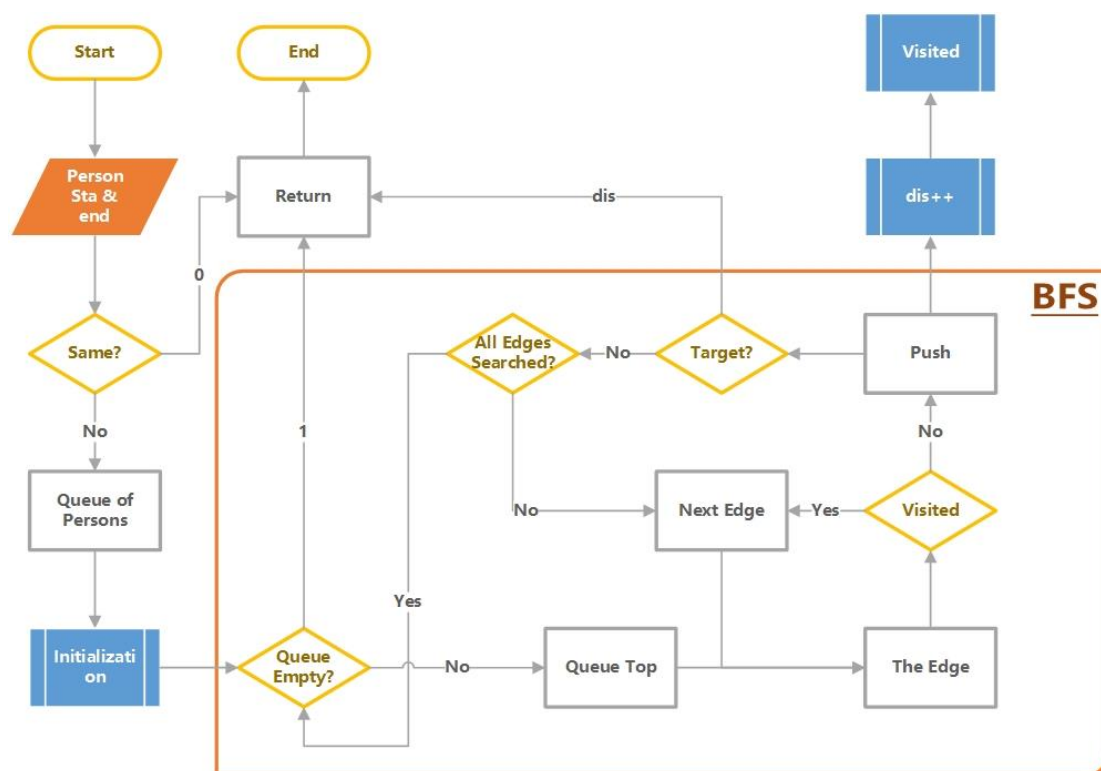
直接调用 `graph.add()` 添加点。

#### int addEdge():

调用 `graph.set()` 两次，添加双向边，默认权值为 1，并记录可能存在的旧边的权值。

#### int getDistance():

首先判断起止点是否相等。再新建 `Map<Person, Integer> dis` 表示从起始点开始到该 Person 的距离，以及 `Map<Person, Boolean> vis` 表示该 Person 是否访问过。将两个 Map 初始化后，把起点标记为已经访问（所有涉及这两个 Map 的操作均需要 `remove` 后再 `put`，后文不再阐述）。然后开始 BFS 搜索，找到终点为止。



### 3.2.2 Person 类

该类的目标是将每一个人对应到一个 Person 对象，并存储名字的信息。为了防止泄露，我将 String Name 设置为私有且不可变的。在构造函数中将 Name 初始化。

### 3.2.3 客户端 main()

```

public class FriendshipGraphTest {

    /**
     * Basic Network Test
     */
    @Test
    public void Test1() {
        final FriendshipGraph graph = new FriendshipGraph();

        final Person rachel = new Person("Rachel");
        final Person ross = new Person("Ross");
        final Person ben = new Person("Ben");
        final Person kramer = new Person("Kramer");

        assertEquals(true, graph.addVertex(rachel));
        assertEquals(true, graph.addVertex(ross));
        assertEquals(true, graph.addVertex(ben));
        assertEquals(true, graph.addVertex(kramer));

        assertEquals(0, graph.addEdge(rachel, ross));
        assertEquals(1, graph.addEdge(ross, rachel));
    }
}

```

```
        assertEquals(0, graph.addEdge(ross, ben));
        assertEquals(1, graph.addEdge(ben, ross));

        assertEquals(1, graph.getDistance(rachel, ross));
        assertEquals(2, graph.getDistance(rachel, ben));
        assertEquals(0, graph.getDistance(rachel, rachel));
        assertEquals(-1, graph.getDistance(rachel, kramer));
    }

    /**
     * Further Test
     */
    @Test
    public void Test2() {
        final FriendshipGraph graph = new FriendshipGraph();

        final Person a = new Person("A");
        final Person b = new Person("B");
        final Person c = new Person("C");
        final Person d = new Person("D");
        final Person e = new Person("E");
        final Person f = new Person("F");
        final Person g = new Person("G");
        final Person h = new Person("H");
        final Person i = new Person("I");
        final Person j = new Person("J");

        assertEquals(true, graph.addVertex(a));
        assertEquals(true, graph.addVertex(b));
        assertEquals(true, graph.addVertex(c));
        assertEquals(true, graph.addVertex(d));
        assertEquals(true, graph.addVertex(e));
        assertEquals(true, graph.addVertex(f));
        assertEquals(true, graph.addVertex(g));
        assertEquals(true, graph.addVertex(h));
        assertEquals(true, graph.addVertex(i));
        assertEquals(true, graph.addVertex(j));

        assertEquals(0, graph.addEdge(a, b));
        assertEquals(0, graph.addEdge(a, d));
        assertEquals(0, graph.addEdge(b, d));
        assertEquals(0, graph.addEdge(c, d));
        assertEquals(0, graph.addEdge(d, e));
        assertEquals(0, graph.addEdge(c, f));
        assertEquals(0, graph.addEdge(e, g));
        assertEquals(0, graph.addEdge(f, g));
        assertEquals(0, graph.addEdge(h, i));
        assertEquals(0, graph.addEdge(i, j));

        assertEquals(2, graph.getDistance(a, e));
        assertEquals(1, graph.getDistance(a, d));
        assertEquals(3, graph.getDistance(a, g));
        assertEquals(3, graph.getDistance(b, f));
        assertEquals(2, graph.getDistance(d, f));
        assertEquals(2, graph.getDistance(h, j));
        assertEquals(0, graph.getDistance(i, i));
        assertEquals(-1, graph.getDistance(d, j));
        assertEquals(-1, graph.getDistance(c, i));
    }
}
```

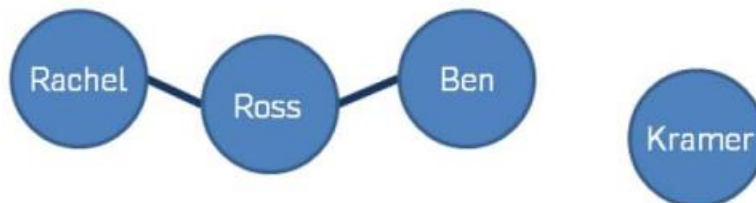


```
    assertEquals(-1, graph.getDistance(f, h));  
}  
}
```

### 3.2.4 测试用例

#### 3.2.4.1 简单图测试

根据题目中的社交网络图：

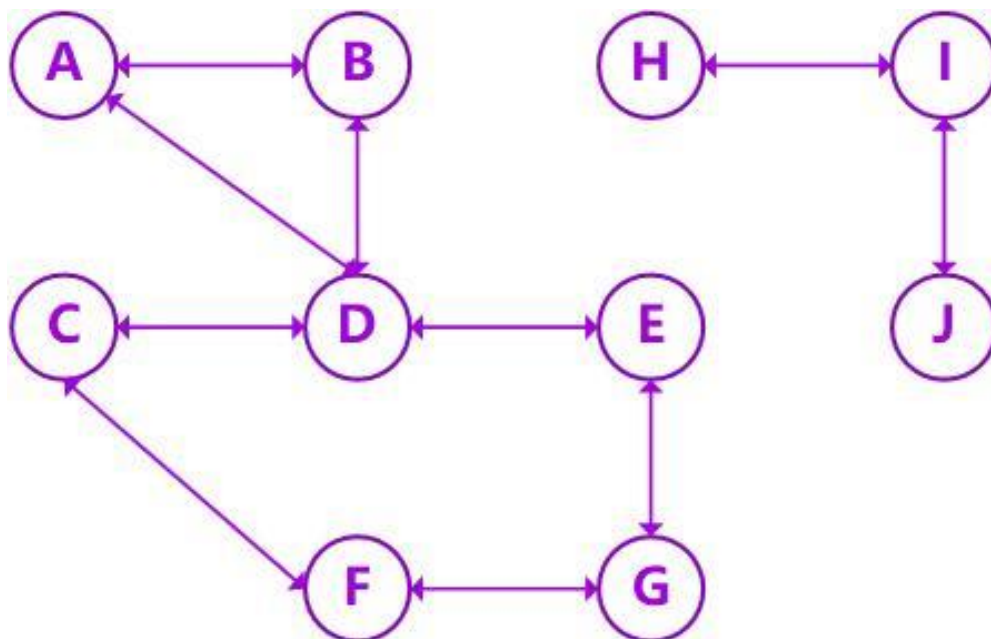


分别测试：

1. Rachel 和 Ross 距离是 1，Rachel 和 Ben 距离是 2
2. Rachel 和 Rachel 距离是 0
3. Rachel 和 Kramer 距离是-1

#### 3.2.4.2 复杂图测试

设计 10 个点、10 条边的社交网络图：



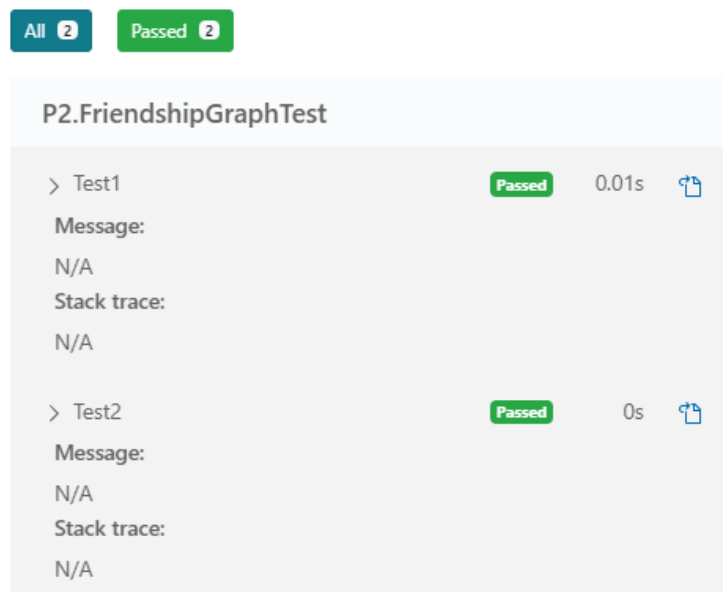
分别测试：

1. AE 距离 2，AD 距离 1，AG 距离 3，BF 距离 3，DF 距离 2，HJ 距离 2
2. II 距离 0
3. DJ 距离-1，CI 距离-1，FH 距离-1



### 3.2.4.3 Junit 测试结果

全部正确。



### 3.2.5 提交至 Git 仓库

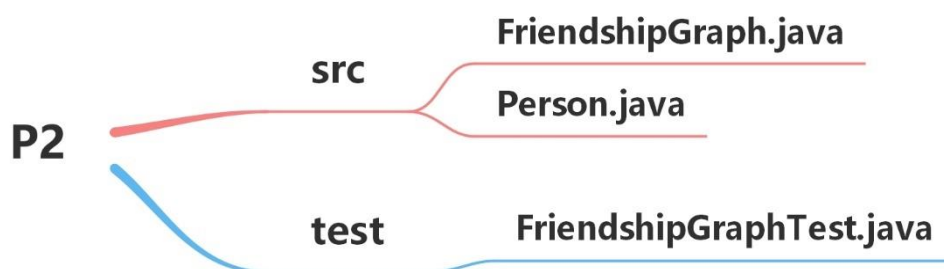
如何通过 Git 提交当前版本到 GitHub 上你的 Lab2 仓库。

```
guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git add *.docx

guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git commit -m "Update Report"
[master 2a73266] Update Report
1 file changed, 0 insertions(+), 0 deletions(-)

guozn@DESKTOP-RQFQUQR MINGW64 /d/GZN/HIT/个人文件/2020春软件构造/2020Labs/Lab2-1183710109 (master)
$ git push
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 71.81 KiB | 128.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:ComputerScienceHIT/Lab2-1183710109.git
1c1eb77..2a73266 master -> master
```

在这里给出你的项目的目录结构树状示意图。



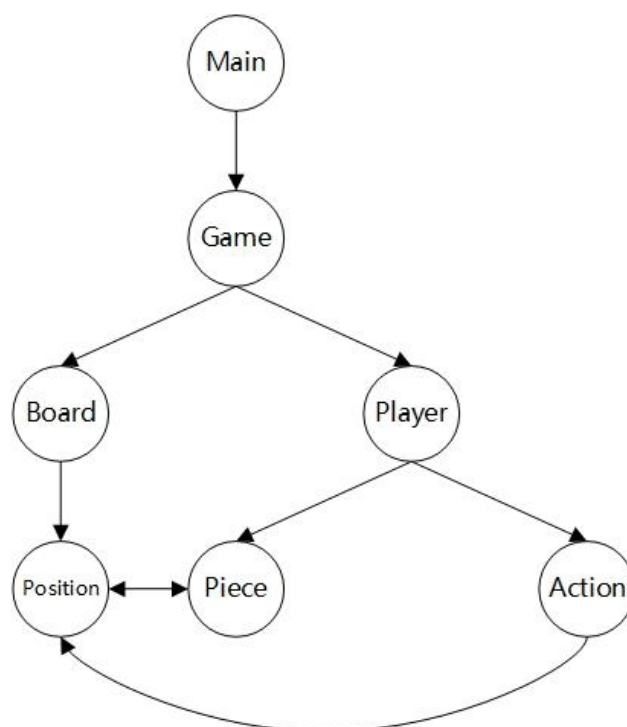
## 3.3 Playing Chess

### 问题简述：

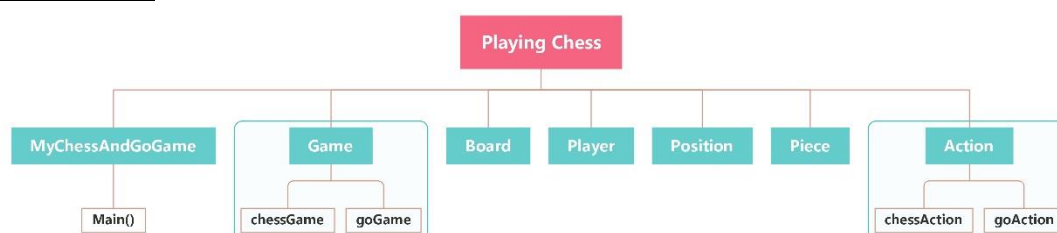
设计一款棋类游戏，同时支持国际象棋（Chess）和围棋（Go）。实现功能：

1. 选择游戏类型：创建 Game、Board
2. 输入玩家名字：创建 Player、Piece，其中 Piece 属于 Player
3. 开始游戏，轮流选择功能
4. 放棋：给定 player、piece、x、y
5. 移动棋（chess）：给定 player、piece、x1、y1、x2、y2
6. 提子（go）：给定 player、x、y
7. 吃子（chess）：给定 player、x1、y1、x2、y2
8. 查询某个位置占用情况：给定 x、y
9. 计算两个玩家分别的棋子总数
10. 跳过
11. 结束：输入“end”

### 整体架构



### 文件结构：



### 3.3.1 ADT 设计/实施方案

#### 3.3.1.1 interface Game

接口 Game 由 chessGame 和 goGame 实现，是 Main() 程序通向游戏对象的路口，通过一个接口把两种游戏分开，相同的操作类型在不同游戏中实现。

##### 接口 Game

所有已知实现类：  
chessGame, goGame

public interface **Game**

the interface Game is implemented by class chessGame and goGame. the Game can do the 7 chose of operations, be asked the players and board, and generate the pieces for initialize the players

##### 方法概要

所有方法	静态方法	实例方法	抽象方法
修饰符和类型	方法	说明	
<b>Board</b>	<b>board()</b>	get the object of the board	
boolean	<b>capture</b> (Player player, Position... positions)	Choice 3 capture a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player if the game is go, there is no target position	
<b>Player</b>	<b>choosePlayerByName</b> (java.lang.String playerName)	find the player whose name is equal to the given	
void	<b>end()</b>	Choice 7 enter "end" to end the game initiative	
java.lang.String	<b>gameType()</b>	ask the type of the game	
<b>Player</b>	<b>isFree</b> (Player player, int x, int y)	Choice 4 ask a position whether it is free the position should be within the board	
boolean	<b>move</b> (Player player, Piece piece, Position position)	Choice 2 FOR chess move or put a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player	
static <b>Game</b>	<b>newGame</b> (java.lang.String gameType)	to generate a particular game according to the String gameType	
java.util.Set<Piece>	<b>pieces</b> (boolean firstFlag)	the origin pieces of one particular game	
<b>Player</b>	<b>player1()</b>	ask the first player	
<b>Player</b>	<b>player2()</b>	ask the later player	
boolean	<b>put</b> (Player player, Piece piece, Position position)	Choice 1 put one particular piece on the board the player should be gaming the piece should belong to the player and be out of the board the position should be free and legally in the board	
boolean	<b>setPlayers</b> (Player p1, Player p2)	make players join in the game	
void	<b>skip</b> (Player player)	Choice 6 skip the choosing	
java.util.Map<Player, java.lang.Integer>	<b>sumPiece</b> (Player player)	Choice 5 ask the sum of the piece on the board	

Game 拥有 7 种操作所对应的方法，并且能支持访问下属的 Player（先后手访问、名字访问）和 Board，以及为玩家产生所对应 Piece 的功能。

7 种操作除了“end”均隶属于 Player 对象进行操作，其中的“放棋”、“移动”、“吃子/提子”均在 Action 接口的实现类中完成，在 Game 接口的实现类中判断是否执行成功即可。因此 3 种操作可以在 Game 的实现类中实现近乎标准化和统一（输入操作类型 String 即可），以“吃子/提子”（capture）为例：

```
@Override
public boolean capture(Player player, Position... positions) {
    if (player == null)
        return false;
    return player.doAction("capture", null, positions) != null;
}
```

在两种游戏中，差异较大的之一就是棋子。棋子属于玩家，但棋子是由一个特定类型的游戏所“产生”的，因此 Game 的两个实现类中差异最大的就是产生棋子的方法：

**pieces**

```
java.util.Set<Piece> pieces(boolean firstFlag)
```

the origin pieces of one particular game

参数:

firstFlag - true if the pieces belong to the first hand player, false if not

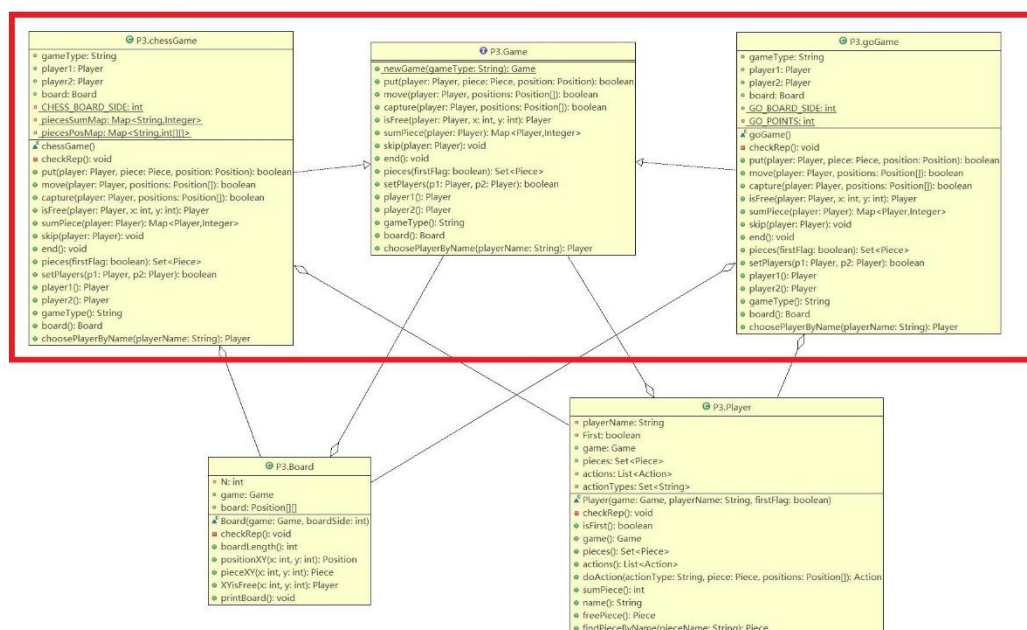
返回:

the set of pieces initialized

在 chess 中，黑白双方棋子除了颜色都相同，因此可以用 chessGame 静态成员变量预设好每个棋子的名字、数量和位置（黑白双方可以用公式颠倒）。然后依据预设的静态数据新建 16 个 Piece 对象，初始化 Position、Player，最后加入 Set<Piece> 中返回。goGame 中大致相同。

```
/**
 * the Map whose keys are the name of pieces, values are the numbers they are on board totally
 */
private static final Map<String, Integer> piecesSumMap = new HashMap<String, Integer>() {
    private static final long serialVersionUID = 1L;
    {
        put("P", 8);
        put("R", 2);
        put("N", 2);
        put("B", 2);
        put("Q", 1);
        put("K", 1);
    }
};
/**
 * the Map whose keys are the name of pieces, values are the coordinates of them
 */
private static final Map<String, int[][]> piecesPosMap = new HashMap<String, int[][]>() {
    private static final long serialVersionUID = 1L;
    {
        put("P", new int[][] { { 0, 1, 2, 3, 4, 5, 6, 7 }, { 1, 1, 1, 1, 1, 1, 1, 1 } });
        put("R", new int[][] { { 0, 7 }, { 0, 0 } });
        put("N", new int[][] { { 1, 6 }, { 0, 0 } });
        put("B", new int[][] { { 2, 5 }, { 0, 0 } });
        put("Q", new int[][] { { 3 }, { 0 } });
        put("K", new int[][] { { 4 }, { 0 } });
    }
};
@Override
public Set<Piece> pieces(boolean firstFlag) {
    Set<Piece> pieces = new HashSet<Piece>();
    for (Map.Entry<String, Integer> entry : piecesSumMap.entrySet()) {
        for (int i = 0; i < entry.getValue(); i++) {
            String pieceName = (firstFlag ? "W" : "B") + entry.getKey() + i; // eg. WB1 BR2 WP3
            Piece piece = new Piece(pieceName, firstFlag, (firstFlag ? player1 : player2));
            // get the coordinate of a specific piece
            int[] X = piecesPosMap.get(entry.getKey())[0];
            int[] Y = piecesPosMap.get(entry.getKey())[1];
            int x = X[i], y = (firstFlag ? Y[i] : CHESS_BOARD_SIDE - Y[i] - 1);
            // put the piece on the position
            piece.modifyPositionAs(board.positionXY(x, y));
            board.positionXY(x, y).modifyPieceAs(piece);
            // add the piece into the piece set of the player
            pieces.add(piece);
        }
    }
    return pieces;
}
```

Game 是 Board 和 Player 的父类。Board 的创建只能源于 Game 的构造函数，Player 的创建必须后于 Game 且玩家的 Piece 依赖于 Game 的函数。



上图上方三个分别是 Game 和 Game 的实现类 chessGame 和 goGame，Board 和 Game 隶属于 Game，在不同情况下调用两种实现类，且这两者无法联系，保护了对象的私有数据。

### 3.3.1.1.1 class chessGame

实现 chess 在 Game 中的功能。

#### 类 chessGame

java.lang.Object  
P3.chessGame

所有已实现的接口:  
Game

public class **chessGame**  
extends java.lang.Object  
implements Game

chessGame implements Game and it run the game for chess.

#### 字段概要

修饰符和类型	字段	说明
<b>Board</b>	<b>board</b>	
private static int	<b>CHESS_BOARD_SIDE</b>	
java.lang.String	<b>gameType</b>	
private static java.util.Map<java.lang.String,int[]>	<b>piecesPosMap</b>	the Map whose keys are the name of pieces, values are the coordinates of them
private static java.util.Map<java.lang.String,java.lang.Integer>	<b>piecesSumMap</b>	the Map whose keys are the name of pieces, values are the numbers they are on board totally
<b>Player</b>	<b>player1</b>	
<b>Player</b>	<b>player2</b>	

#### 构造器概要

构造器	说明
<b>chessGame()</b>	create a chess game

#### 方法概要

所有方法	实例方法	具体方法
修饰符和类型	方法	说明
<b>Board</b>	<b>board()</b>	get the object of the board
boolean	<b>capture(Player player, Position... positions)</b>	Choice 3 capture a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player if the game is go, there is no target position
private void	<b>checkRep()</b>	Rep: gameType can't be null players can't be null board can't be null
<b>Player</b>	<b>choosePlayerByName(java.lang.String playerName)</b>	find the player whose name is equal to the given
void	<b>end()</b>	Choice 7 enter "end" to end the game initiative
java.lang.String	<b>gameType()</b>	ask the type of the game
<b>Player</b>	<b>isFree(Player player, int x, int y)</b>	Choice 4 ask a position whether it is free the position should be within the board
boolean	<b>move(Player player, Piece piece, Position position)</b>	Choice 2 FOR chess move or put a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player
java.util.Set<Piece>	<b>pieces(boolean firstFlag)</b>	the origin pieces of one particular game
<b>Player</b>	<b>player1()</b>	ask the first player
<b>Player</b>	<b>player2()</b>	ask the later player
boolean	<b>put(Player player, Piece piece, Position position)</b>	Choice 1 put one particular piece on the board the player should be gaming the piece should belong to the player and be out of the board the position should be free and legally in the board
boolean	<b>setPlayers(Player p1, Player p2)</b>	make players join in the game
void	<b>skip(Player player)</b>	Choice 6 skip the choosing
java.util.Map<Player, java.lang.Integer>	<b>sumPiece(Player player)</b>	Choice 5 ask the sum of the piece on the board
从类继承的方法 <b>java.lang.Object</b>		
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait		

### 3.3.1.1.2 class goGame

实现 go 在 Game 中的功能。

#### 类 goGame

java.lang.Object  
P3.goGame

所有已实现的接口：  
Game

```
public class goGame
extends java.lang.Object
implements Game
```

goGame implements Game and it run the game for go.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
Board	board	
java.lang.String	gameType	
private static int	GO_BOARD_SIDE	
private static int	GO_POINTS	
Player	player1	
Player	player2	

#### 构造器概要

##### 构造器

构造器	说明
goGame()	create a go game

#### 方法概要

##### 所有方法

##### 实例方法

##### 具体方法

修饰符和类型	方法	说明
Board	board()	get the object of the board
boolean	capture(Player player, Position... positions)	Choice 3 capture a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player if the game is go, there is no target position
private void	checkRep()	Rep: gameType can't be null players can't be null board can't be null
Player	choosePlayerByName(java.lang.String playerName)	find the player whose name is equal to the given
void	end()	Choice 7 enter "end" to end the game initiative
java.lang.String	gameType()	ask the type of the game
Player	isFree(Player player, int x, int y)	Choice 4 ask a position whether it is free the position should be within the board
boolean	move(Player player, Position... positions)	Choice 2 FOR chess move or put a piece in a position the player should be gaming the source-piece should belong to the player and the target-piece should belong to another player
java.util.Set<Piece>	pieces(boolean firstFlag)	the origin pieces of one particular game
Player	player1()	ask the first player
Player	player2()	ask the later player
boolean	put(Player player, Piece piece, Position position)	Choice 1 put one particular piece on the board the player should be gaming the piece should belong to the player and be out of the board the position should be free and legally in the board
boolean	setPlayers(Player p1, Player p2)	make players join in the game
void	skip(Player player)	Choice 6 skip the choosing
java.util.Map<Player, java.lang.Integer>	sumPiece(Player player)	Choice 5 ask the sum of the piece on the board

#### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait



### 3.3.1.2 class Board

Board 是棋盘的对象，构造依赖于 Game 的构造。Position 的创建也依赖于 Board，Board 也存储这二维 Position 类型数组，并且拥有 final 变量 N 记录棋盘的边长。

#### 类 Board

java.lang.Object  
P3.Board

public class Board  
extends java.lang.Object

The Board contains N\*N Position objects. When the Board is created, the position is created too. It has methods to ask chosen position and its piece, and print the board on screen.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
Position[][]	board	
Game	game	
private int	N	

#### 构造器概要

##### 构造器

构造器	说明
Board(Game game, int boardSide)	Initialize a new Board and N*N Position

#### 方法概要

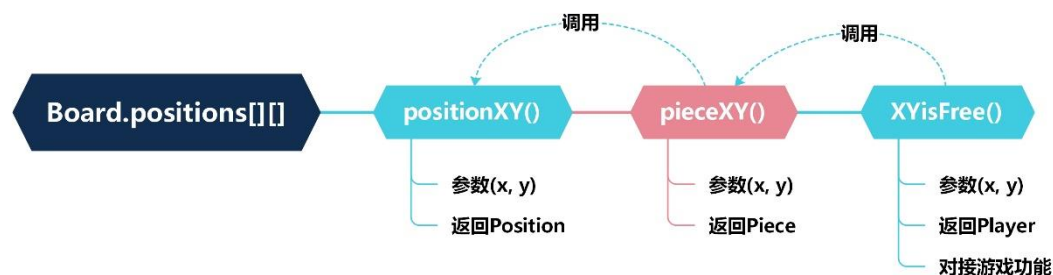
##### 所有方法 实例方法 具体方法

修饰符和类型	方法	说明
int	boardLength()	ask the length of the board
private void	checkRep()	Rep: N can't be negative board must be full
Piece	pieceXY(int x, int y)	ask the piece on (x, y) if there isn't null
Position	positionXY(int x, int y)	ask object of the position
void	printBoard()	print the Board '.' if one position has no piece or the piece' name if not
Player	XYisFree(int x, int y)	ask the player who owns the piece of (x, y) or null if not

##### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Board 的主要用于查询指定位置的 Position、Piece 和 Player，以及打印棋盘。查询 Position 可以直接访问 positions 成员变量，而查询 Piece 又要访问指定位置的 Position 不为空的 Piece，而查询 Player 又要查询不空的 Piece 的 Player。



在三个函数的实现中，按照调用关系，先后实现。在这里设计 Position 和 Piece 平级且捆绑，同为可变。



```

/**
 * ask object of the position
 * @param x the x of the asking position
 * @param y the y of the asking position
 * @return object of Position of the (x, y)
 */
public Position positionXY(int x, int y) {
    if (x < 0 || x >= this.N || y < 0 || y >= this.N)
        return null;
    return board[x][y];
}

/**
 * ask the piece on (x, y) if there isn't null
 * @param x the x of the asking position
 * @param y the y of the asking position
 * @return object of Piece of the (x, y)
 */
public Piece pieceXY(int x, int y) {
    if (positionXY(x, y) == null)
        return null;
    return positionXY(x, y).piece();
}

/**
 * ask the player who owns the piece of (x, y) or null if not
 * @param x the x of the asking position
 * @param y the y of the asking position
 * @return Player if (x, y) is occupied, null if it's free
 */
public Player XYisFree(int x, int y) {
    if (pieceXY(x, y) == null)
        return null;
    return pieceXY(x, y).player();
}

```

此外, 棋盘还具备打印功能。以下是实现方案和效果图。

```

/**
 * print the Board
 * '.' if one position has no piece
 * or the piece' name if not
 */
public void printBoard() {
    for (int i = 0; i < this.N; i++) {
        for (int j = 0; j < this.N; j++) {
            if (this.pieceXY(i, j) != null) {
                if (game.gameType().equals("chess")) {
                    /**
                     * the capital letter represents the white piece
                     * the little letter represents the black piece
                     */
                    System.out.print((this.pieceXY(i, j).isFirst() ? this.pieceXY(i, j).name().c
harAt(1)
                                : this.pieceXY(i, j).name().toLowerCase().charAt(1)) + " ");
                } else if (game.gameType().equals("go")) {
                    /**
                     * the 'B' represents the black pieces
                     * the 'W' represents the white pieces
                     */
                    System.out.print(this.pieceXY(i, j).name().charAt(0) + " ");
                }
            } else {
                // if there is no piece
                System.out.print(". ");
            }
        }
    }
}

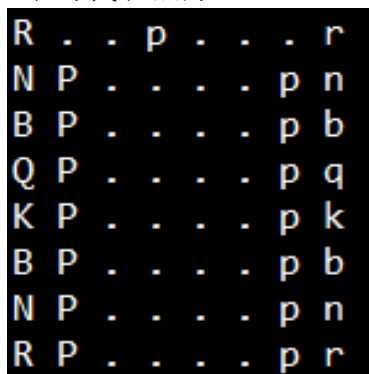
```

```

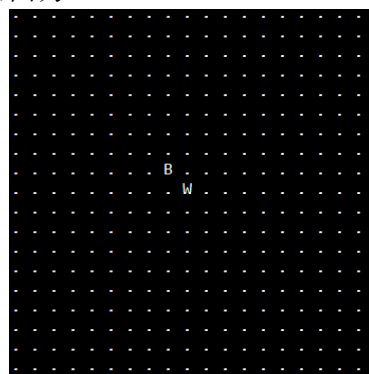
        System.out.println();
    }
}

```

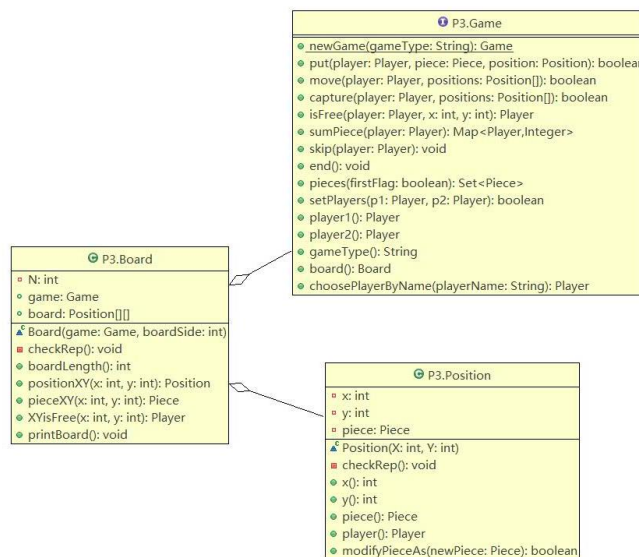
国际象棋：大写代表白方，小写代表黑方。



围棋：B 代表黑方，W 代表白方。



棋盘能够管理棋格/点，而根据要求棋盘是不能管理棋子的。因此 Board 是 Game 的子类，也是 Position 的父类。



### 3.3.1.3 class Player

Player 对象代表着玩家，有这 Boolean 标签区分先后手，拥有 Piece 并管理 Action。

### 类 Player

```
java.lang.Object
P3.Player
```

```
public class Player
extends java.lang.Object
```

the Player represent the gaming player, the player owns the pieces and actions as well as operating them.

### 字段概要

字段

修饰符和类型	字段	说明
private java.util.List<Action>	actions	the actions record in order
private java.util.Set<java.lang.String>	actionTypes	the Strings of the type of whole actions
private boolean	First	
Game	game	
java.util.Set<Piece>	pieces	
private java.lang.String	playerName	

## 构造器概要

## 构造器

构造器	说明
<code>Player(Game game, java.lang.String playerName, boolean firstFlag)</code>	initialize a Player and set his pieces

### 方法概要

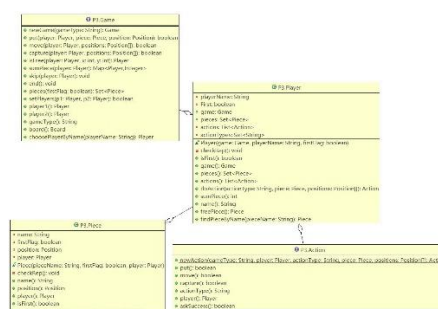
所有方法 实例方法 具体方法

修饰符和类型	方法	说明
java.util.List<Action>	actions()	ask for the actions after game
private void	checkRep()	Rep: playerName can't be blank game can't be null there is no the same piece in pieces
Action	doAction(java.lang.String actionType, Piece piece, Position... positions)	generate a new action and init the action type
Piece	findPieceByName(java.lang.String pieceName)	find a piece which owns the same name
Piece	freePiece()	ask any of the free pieces
Game	game()	ask for the object of the game
boolean	isFirst()	ask the player is first or later hand
java.lang.String	name()	ask one player's name
java.util.Set<Piece>	pieces()	ask for all of the pieces
int	sumPiece()	ask the number of pieces

## 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Player 的方法除了查询本对象的信息，还有寻找自己下属的棋子，以及执行并记录 Action。



寻找棋子时：当以棋子名字查询时，只需在成员变量 `Set<Piece> pieces` 中遍历，判断棋子的名字是否相等即可；而当查询任意一个空闲棋子时，则需判断其 `position` 是否为空。此外，Player 还能计算本方棋盘上棋子总数。

```
/**
 * ask the number of pieces
 * @return the number of pieces
 */
public int sumPiece() {
    int sum = 0;
    for (Piece piece : pieces) {
        // calculate the non-null piece
        if (piece.position() != null) {
            sum++;
        }
    }
    return sum;
}

/**
 * ask any of the free pieces
 * @return a free piece belonging to the player
 */
public Piece freePiece() {
    for (Piece piece : this.pieces) {
        // find a random free piece
        if (piece.position() == null)
            return piece;
    }
    return null;
}

/**
 * find a piece which owns the same name
 * @param pieceName String of the name of the piece
 * @return piece object of the piece name
 */
public Piece findPieceByName(String pieceName) {
    for (Piece piece : this.pieces) {
        // find the piece whose name is matched with the giving
        if (piece.name().equals(pieceName))
            return piece;
    }
    return null;
}
```

在执行 Action 的方法中，通过构造一个新的 Action 对象来执行。在 Action 对象的内部会进行完整的操作，玩家只需访问改动作对象的成功与否，然后加入 List 存储即可。

```
/**
 * generate a new action and init the action type
 * @param actionType String of the type of the action
 * @param piece the putting piece when the actionType is "put", null if not
 * @param positions the positions related to the action
 * @return the object of the action created
 */
public Action doAction(String actionType, Piece piece, Position... positions) {
    if (!ActionTypes.contains(actionType))
        return null;
    Action action = Action.newAction(this.game.gameType(), this, actionType, piece, positions);
    if (action.askSuccess())
        actions.add(action);
    else
        action = null;
    return action;
}
```

### 3.3.1.4 class Position

Position 代表着国际象棋棋盘的格子以及围棋棋盘的交叉点。

#### 类 Position

java.lang.Object  
P3.Position

public class **Position**  
extends java.lang.Object

Position represent a grid in chess or a point in go. A position will not be modify its coordinates once it has been created. When the position has piece on it, the position is related with the player.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
private Piece	piece	
private int	x	
private int	y	

#### 构造器概要

##### 构造器

构造器	说明
<b>Position</b> (int X, int Y)	create the position

#### 方法概要

##### 所有方法 实例方法 具体方法

修饰符和类型	方法	说明
private void	checkRep()	x, y must be non-negative
boolean	modifyPieceAs(Piece newPiece)	to update the Piece of the Position
Piece	piece()	
Player	player()	
int	x()	
int	y()	

##### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Position 隶属于 Board，一个对象的 x 和 y 是不可变的，但 Position 记录的 Piece 对象是可变的，提供了方法进行修改。

```
/**
 * to update the Piece of the Position
 * @param newPiece the new piece that is to modify it as
 * @return true if the Piece updated successfully, false if the new Piece is null
 */
public boolean modifyPieceAs(Piece newPiece) {
    this.piece = newPiece;
    checkRep();
    return true;
}
```

### 3.3.1.5 class Piece

Piece 代表着棋盘上的棋子，用一个唯一标识的 String 来区分每一个棋子，和一个 Boolean 来区分先后手，这两个信息是不可变的。

#### 类 Piece

java.lang.Object  
P3.Piece

public class **Piece**  
extends java.lang.Object

Piece represent one piece on the board. Every piece has its own name, and knows its position and owner player. the object can ask for the information, and modify the position.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
private boolean	<b>firstFlag</b>	
private java.lang.String	<b>name</b>	
private <b>Player</b>	<b>player</b>	
private <b>Position</b>	<b>position</b>	

#### 构造器概要

##### 构造器

构造器	说明
<b>Piece</b> (java.lang.String pieceName, boolean firstFlag, <b>Player</b> player)	

#### 方法概要

##### 所有方法 实例方法 具体方法

修饰符和类型	方法	说明
private void	<b>checkRep</b> ()	name can't be "" player can't be null
boolean	<b>isFirst</b> ()	ask the piece belongs to first player or later player
boolean	<b>modifyPositionAs</b> ( <b>Position</b> newPosition)	to update the position of the piece
java.lang.String	<b>name</b> ()	ask the piece's name
<b>Player</b>	<b>player</b> ()	ask who owns the piece
<b>Position</b>	<b>position</b> ()	ask the position of the piece

##### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

和 Position 相似，Piece 也提供了修改 Position 的方法，用于移动棋子等变化操作。

```
/**
 * to update the position of the piece
 * @param newPosition the new position that is to modify it as
 * @return true if the position updated successfully, false if the newPosition is null
 */
public boolean modifyPositionAs(Position newPosition) {
    this.position = newPosition;
    checkRep();
    return true;
}
```

### 3.3.1.6 interface Action

Action 代表着要求里的若干个操作, 通过一个 String 来区分, 在构造函数中就实现这一操作 (主要是放棋、吃子/提子、移动)。所需要的参数通过客户端的输入, 再由 Game 传递, Player 的方法 doAction() 中构造, 然后在 Action 对象中执行, 并记录执行的成败。Action 是基于 Player 实现的, 作用于 Position。

#### 接口 Action

所有已知实现类:  
chessAction, goAction

public interface Action

interface Action is implemented by chessAction and goAction which represent 2 types of game actions. An Action can execute 6 types of actions, including put, move, capture, AskIsFree, SumPiece and skip. As soon as the Action object is created, the action has been executed. The result of success will be stored in actionSuccess.

#### 方法概要

所有方法	静态方法	实例方法	抽象方法
修饰符和类型	方法	说明	
java.lang.String	actionType()	ask the action's type	
boolean	askSuccess()	ask weather the action is done successfully.	
boolean	capture()	capture a piece by another piece(chess) or a group of pieces(go)	
boolean	move()	move one piece to a chose position	
static Action	newAction(java.lang.String gameType, Player player, java.lang.String actionType, Piece piece, Position... positions)	generate a new Action of one piece do the action	
Player	player()	ask who does the action	
boolean	put()	put a piece onto a position	

接口 Action 有一个静态函数作为构造器:

```
/**
 * generate a new Action of one piece
 * do the action
 * @param gameType String of the type of the game
 * @param player the acting player
 * @param actionType String of the type of the action
 * @param piece the operating piece
 * @param positions the positions related to the action
 * @return an object of a type of Action(chessAction or goAction)
 */
public static Action newAction(String gameType, Player player, String actionType, Piece piece,
    Position... positions) {
    return gameType.equals("chess") ? (new chessAction(player, actionType, piece, positions))
        : (new goAction(player, actionType, piece, positions));
}
```

创建新的对象后, 依据 String actionType 执行操作:

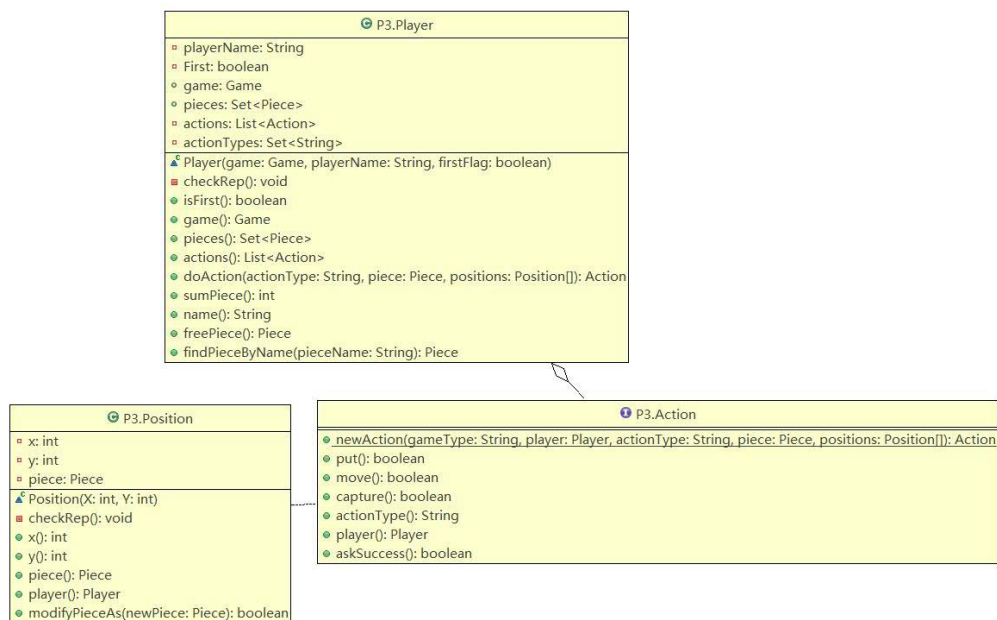
```
/**
 * create and finish the action
 * @param player the operating player
 * @param actionType String of the type of the action
 * @param piece the operating piece
 * @param positions the position related to the action
 */
chessAction(Player player, String actionType, Piece piece, Position... positions) {
    this.player = player;
    this.positions = positions;
    this.piece = piece;
    this.actionType = actionType;
    switch (actionType) {
        case "put":
            this.actionSuccess = (piece != null) && put();
            break;
        case "move":
```

```

        this.actionSuccess = move();
        break;
    case "capture":
        this.actionSuccess = capture();
        break;
    case "AskIsFree":
        this.actionSuccess = true;
        break;
    case "SumPiece":
        this.actionSuccess = true;
        break;
    case "skip":
        this.actionSuccess = true;
        break;
    default:
        this.actionSuccess = false;
    }
    checkRep();
}

```

chessAction 和 goAction 分别重写 Action 中的操作, 主要是 put()、move()、capture() 三个方法。最后通过方法的返回值将结果记录在 actionSuccess。





### 3.3.1.6.1 class chessAction

chessAction 代表着国际象棋中的操作。

#### 类 chessAction

java.lang.Object  
P3.chessAction

所有已实现的接口:  
Action

public class **chessAction**  
extends java.lang.Object  
implements Action

chessAction implements Action and it does the actions in chess game.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
private static java.util.Set<java.lang.String>	<b>ACTIONS</b>	the probable actions' names
private boolean	<b>actionSuccess</b>	true if the action if it succeeds
private java.lang.String	<b>actionType</b>	
<b>Piece</b>	<b>piece</b>	
<b>Player</b>	<b>player</b>	
private Position[]	<b>positions</b>	

#### 构造器概要

##### 构造器

构造器	说明
<b>chessAction</b> (Player player, java.lang.String actionType, Piece piece, Position... positions)	create and finish the action

#### 方法概要

##### 所有方法 实例方法 具体方法

修饰符和类型	方法	说明
java.lang.String	<b>actionType()</b>	ask the action's type
boolean	<b>askSuccess()</b>	ask weather the action is done successfully.
boolean	<b>capture()</b>	capture a piece by another piece(chess) or a group of pieces(go)
private void	<b>checkRep()</b>	Rep: actionType must be in {"put", "move", "capture"} player can't be null if the actionType == "put", piece can't be null
boolean	<b>move()</b>	move one piece to a chose position
<b>Player</b>	<b>player()</b>	ask who does the action
boolean	<b>put()</b>	put a piece onto a position

##### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

put() 操作与 go 类型, 将在 goAction 中举例。move() 操作和 capture() 操作中, 国际象棋比围棋多需要 1 个 Position 操作, 因此输入 Position 时就以不定项参数输入。两个操作大体类似, 主要区别于: move 的目标格要求为空, capture 需要移除目标格棋子。以 capture() 为例:

```
public boolean capture() {
    Position source = this.positions[0], target = this.positions[1];
    // capture requirement: 1. the target can't be null 2. the source can't be null
    // 3. the target must belong to the OPPOSITE 4. the source must belong to this player
    if (target.piece() != null && source.piece() != null && (!target.piece().player().equals(player))
        && source.piece().player().equals(player)) {
        target.piece().modifyPositionAs(null); // the piece capturing removed
        source.piece().modifyPositionAs(target); // captured piece move to the target
        target.modifyPieceAs(source.piece()); // move the piece, this must be done before source's piece be null
        source.modifyPieceAs(null); // set the source null
        return true;
    }
    return false;
}
```

### 3.3.1.6.2 class goAction

goAction 代表着围棋中的操作。

#### 类 goAction

java.lang.Object  
P3.goAction

所有已实现的接口:  
Action

public class **goAction**  
extends java.lang.Object  
implements Action

goAction implements Action and it does the actions in go game.

#### 字段概要

##### 字段

修饰符和类型	字段	说明
private static java.util.Set<java.lang.String>	<b>ACTIONS</b>	the probable actions' names
private boolean	<b>actionSuccess</b>	true if the action if it succeeds
private java.lang.String	<b>actionType</b>	
<b>Piece</b>	<b>piece</b>	
<b>Player</b>	<b>player</b>	
<b>Position[]</b>	<b>positions</b>	

#### 构造器概要

##### 构造器

构造器	说明
<b>goAction</b> (Player player, java.lang.String actionType, Piece piece, Position... positions)	create and finish the action

#### 方法概要

##### 所有方法 实例方法 具体方法

修饰符和类型	方法	说明
java.lang.String	<b>actionType()</b>	ask the action's type
boolean	<b>askSuccess()</b>	ask weather the action is done successfully.
boolean	<b>capture()</b>	capture a piece by another piece(chess) or a group of pieces(go)
private void	<b>checkRep()</b>	Rep: actionType must be in {"put", "move", "capture"}; player can't be null if the actionType == "put", piece can't be null
boolean	<b>move()</b>	move one piece to a chose position
<b>Player</b>	<b>player()</b>	ask who does the action
boolean	<b>put()</b>	put a piece onto a position

##### 从类继承的方法 java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

put() 操作在国际象棋和围棋中类似：客户端输入棋的名字（类型即可、无编号），然后 game 寻找该棋子后输入 action 对象；board 将 (x, y) 转换为 Position 对象后输入 action 对象；最后执行：（以 goAction 为例）

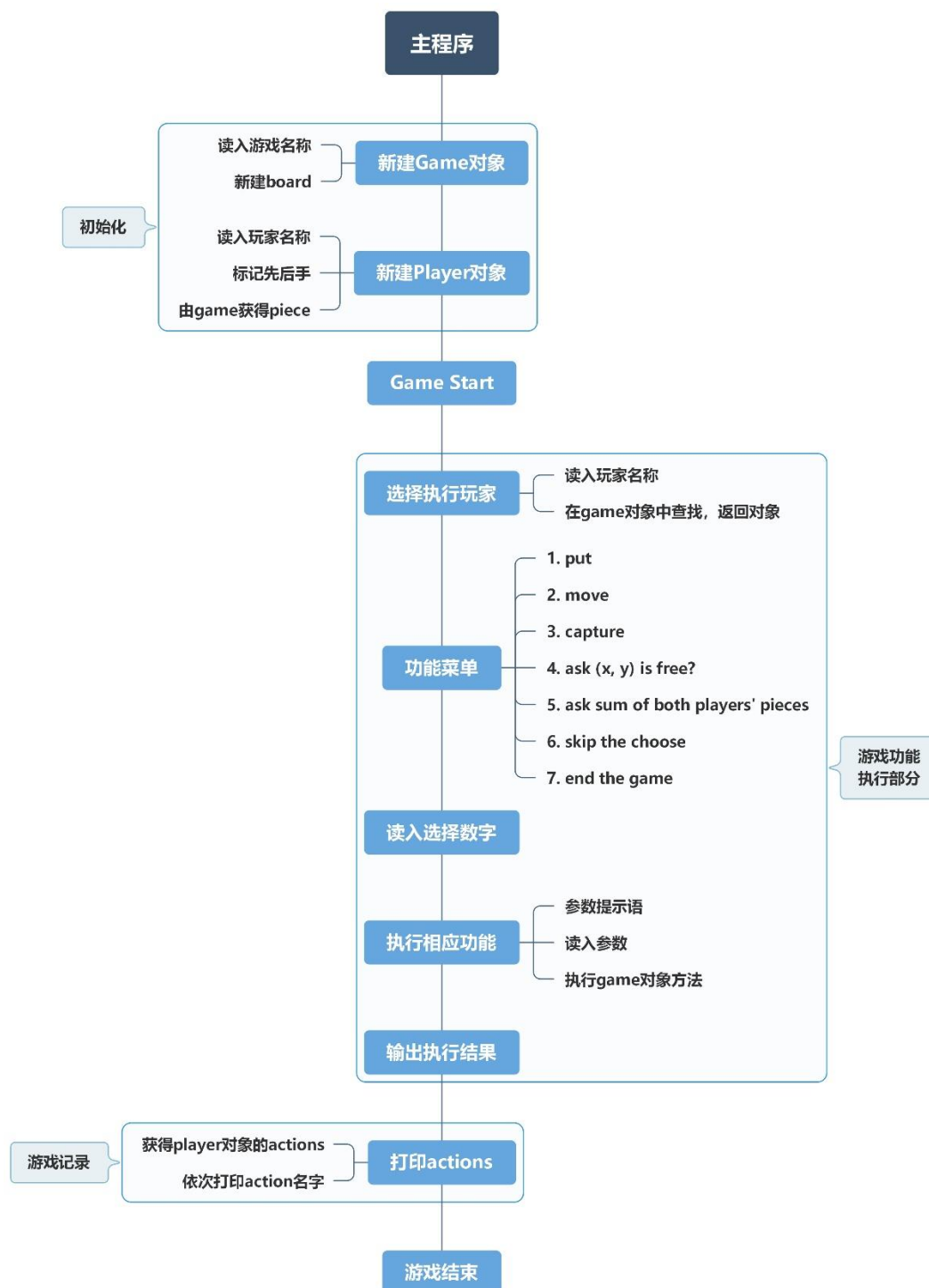
```
@Override
public boolean put() {
    Position target = this.positions[0];
    // put requirement:
    // 1. the piece of the target can't be null
    // 2. the putting piece can't be null
    // 3. the piece must belong to the player
    if (this.piece.position() == null && target.piece() == null && player.pieces().contains(pie
e)) {
        this.piece.modifyPositionAs(target);
        target.modifyPieceAs(this.piece);
        return true;
    }
    return false;
}
```

### 3.3.2 主程序 MyChessAndGoGame 设计/实现方案

Main() 主要分为 3 个步骤：

1. 新建游戏和玩家
2. 选择游戏功能并执行
3. 打印游戏记录

#### 3.3.2.1 游戏流程图



### 3.3.2.2 程序演示

#### 初始化:

给出提示语，输入游戏类型、玩家双方名字。若创建成功则显示“Game Start”。

```
D:\GZN\HIT\个人文件\2020春软件构造\2020Labs>java -jar MyChessAndGoGame.jar
Please choose a type of game (chess/go):
chess
Please write the player1's name (First):
p1
Please write the player2's name (Later):
p2
Game Start!
```

#### 游戏功能执行:

请用户选择玩家（输入名字），给出功能菜单，在用户选择后要求用户输入参数，执行功能，返回结果（true/false），打印当前棋盘。

下面给出一些操作的示例：

```
Please choose a player:
p1
Please choose an action type:
1. put
2. move
3. capture
4. ask: (x, y) is free?
5. ask: sum of both players' pieces
6. skip the choose
7. end the game
Your Choice is:2
The (x, y) of both source and target: 3 1 3 3
true
R P . . . p r
N P . . . p n
B P . . . p b
Q . . P . . p q
K P . . . p k
B P . . . p b
N P . . . p n
R P . . . p r
```

```
Please choose a player:
p2
Please choose an action type:
1. put
2. move
3. capture
4. ask: (x, y) is free?
5. ask: sum of both players' pieces
6. skip the choose
7. end the game
Your Choice is:4
The (x, y) of the questioning grid: 6 6
p2
R P . . . p r
N P . . . p n
B P . . . p b
Q . . P . . p q
K P . . . p k
B P . . . p b
N P . . . p n
R P . . . p r
```



输出记录：

```
All of the Actions Record are followed.

p1's Actions:
0: move
1: put

p2's Actions:
0: capture
That's All! See You Next Time!
```

## 3.3.2.3 功能实现

## 初始化：

```
/**
 * provide 2 choices on screen for users to choose chess or go.
 * generate new Game;
 * generate new Board;
 *
 * get 2 players' names printed on the screen.
 * generate 2 new Player;
 * generate new Piece belonged to Player;
 *
 * @param args FORMAT
 */
public static void main(String[] args) {
    // scan : 3 String
    System.out.println("Please choose a type of game (chess/go):");
    String gameType = input.nextLine();
    System.out.println("Please write the player1's name (First):");
    String playerName1 = input.nextLine();
    System.out.println("Please write the player2's name (Later):");
    String playerName2 = input.nextLine();
    // new objects
    final Game game = Game.newGame(gameType);
    final Player player1 = new Player(game, playerName1, true);
    final Player player2 = new Player(game, playerName2, false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);
    // gaming
    GAME(game);
    input.close();
    // actions
    printRecord(game, player1, player2);
    System.out.println("That's All! See You Next Time!");
}
```

## 功能执行：

```
/**
 * provide 7 choices including;
 * 1. to put a piece : put();
 * 2. to move a piece : move();
 * 3. to capture particular piece(s) : capture();
 * 4. ask whether a position is free or not : isFree();
```

```

    * 5. calculate the sum of the pieces on the
    * board : sumPiece();
    * 6. skip : skip()
    * 7. print "end" : end()
    * @param game the object of the game
    */
private static void GAME(Game game) {
    boolean endFlag = false;
    System.out.println("Game Start!");
    while (!endFlag) {
        System.out.println("Please choose a player:");
        String playerName = input.next();
        endFlag = playerActing(game, game.choosePlayerByName(playerName));
        game.board().printBoard();
    }
}

/**
 * The chosen player is operating one choice
 * @param game the object of the game
 * @param player the object of the operating player
 * @return true if the player choose ending the game, false if not
 */
private static boolean playerActing(Game game, Player player) {
    // menu
    System.out.println("Please choose an action type:");
    System.out.println("1. put");
    System.out.println(game.gameType().equals("chess") ? "2. move" : "");
    System.out.println("3. capture");
    System.out.println("4. ask: (x, y) is free?");
    System.out.println("5. ask: sum of both players' pieces");
    System.out.println("6. skip the choose");
    System.out.println("7. end the game");
    // input information
    String pieceName = "";
    int x1, y1; // source
    int x2, y2; // target
    // catch choice
    System.out.print("Your Choice is:");
    int choice = input.nextInt();
    while (choice > 0 && choice <= 7) { // prepare the probable wrong choice
        switch (choice) {
            case 1: // put
                if (game.gameType().equals("chess")) {
                    pieceName = input.next("Piece Name (eg. WQ0 BP2): ");
                }
                System.out.print("The (x, y) of the target: ");
                x1 = input.nextInt();
                y1 = input.nextInt();
                // choose a piece freely if go
                // get the particular piece if chess
                Piece puttingPiece = game.gameType().equals("chess") ? player.findPieceByName(pieceName)
                    : player.freePiece();
                // print result
                System.out.println(game.put(player, puttingPiece, game.board().positionXY(x1, y1)));
                return false;

```



```

        case 2: // move
            System.out.print("The (x, y) of both source and target: ");
            x1 = input.nextInt();
            y1 = input.nextInt();
            x2 = input.nextInt();
            y2 = input.nextInt();
            System.out.println(
                game.move(player, game.board().positionXY(x1, y1), game
                    .board().positionXY(x2, y2)));
            return false;
        case 3: // capture
            if (game.gameType().equals("chess")) {
                System.out.print("The (x, y) of both source and target: ");
                x1 = input.nextInt();
                y1 = input.nextInt();
                x2 = input.nextInt();
                y2 = input.nextInt();
                System.out.println(
                    game.capture(player, game.board().positionXY(x1, y1
                        ), game.board().positionXY(x2, y2)));
            } else if (game.gameType().equals("go")) {
                System.out.print("The (x, y) of the target: ");
                x1 = input.nextInt();
                y1 = input.nextInt();
                System.out.println(game.capture(player, game.board().positi
                    onXY(x1, y1)));
            }
            return false;
        case 4: // is free?
            System.out.print("The (x, y) of the questioning grid: ");
            x1 = input.nextInt();
            y1 = input.nextInt();
            Player here = game.isFree(player, x1, y1);
            System.out.println(here == null ? "Free" : here.name());
            return false;
        case 5: // sum of pieces
            Map<Player, Integer> sumPiece = game.sumPiece(player); // two p
            layers' sum of pieces
            System.out.println(game.player1().name() + ":" + sumPiece.get(g
                ame.player1()) + " pieces");
            System.out.println(game.player2().name() + ":" + sumPiece.get(g
                ame.player2()) + " pieces");
            return false;
        case 6: // skip
            game.skip(player);
            System.out.println("Skip");
            return false;
        case 7: // end
            game.end();
            System.out.println("The Game is ended.");
            return true;
    }
    System.out.println("Input WRONG, Please choose again:");
}
return false;
}

```

打印记录:



```

/**
 * after the game is ended, to print both players' records of the game.
 * @param game the object of the game
 * @param player1 the object of the first hand player
 * @param player2 the object of the later hand player
 */
private static void printRecord(Game game, Player player1, Player player2) {
    System.out.println("\nAll of the Actions Record are followed.");
    // get the record of the actions
    List<Action> actions1 = player1.actions();
    List<Action> actions2 = player2.actions();
    System.out.println("\n" + player1.name() + "'s Actions:");
    // print their action types
    for (int i = 0; i < actions1.size(); i++) {
        if (actions1.get(i) != null)
            System.out.println(i + ": " + actions1.get(i).actionType());
    }
    System.out.println("\n" + player2.name() + "'s Actions:");
    for (int i = 0; i < actions2.size(); i++) {
        if (actions2.get(i) != null)
            System.out.println(i + ": " + actions2.get(i).actionType());
    }
}
}

```

### 3.3.3 ADT 和主程序的测试方案

分别测试国际象棋和围棋，测试类分别为 ChessTest 和 GoTest。由于国际象棋的测试比围棋困难，约束也比围棋多，因此设计 ChessTest 后进行一定更改就可以测试围棋。以下以 chess 为例，设计 ADT 测试方案。

#### 3.3.3.1 testInit()

测试初始化游戏的操作：新建指定类型的 game、board，新建指定名字的 player，以及新建 piece 并将棋子赋予 player。

测试 game、player、piece 均不为空。

```

@Test
public void testInit() {
    final Game game = Game.newGame("chess");
    assertEquals(null, game);
    final Player player1 = new Player(game, "p1", true);
    assertEquals(null, player1);
    final Player player2 = new Player(game, "p2", false);
    assertEquals(null, player2);
    assertEquals(true, game.setPlayers(player1, player2));
    player1.pieces = game.pieces(true);
    assertEquals(Collections.EMPTY_SET, player1.pieces());
    player2.pieces = game.pieces(false);
    assertEquals(Collections.EMPTY_SET, player2.pieces());
}

```

#### 3.3.3.2 testPiece()

测试棋子是否被新建，以及是否赋予玩家、初始化位置。

```

@Test

```

```
public void testPiece() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);

    // Piece.modifyAsPosition
    for (Piece piece : player1.pieces()) {
        assertNotNull(piece.position());
        assertNotNull(piece.position().piece());
        assertEquals(piece, piece.position().piece());
    }
}
```

### 3.3.3.3 testBoard

测试棋盘的方法。

1. printBoard() 打印出棋盘;
2. positionXY() 返回的 Position 对象不为空, 并且当位置有棋子是判断棋子的名字是否和期望一样;
3. pieceXY() 返回的棋名字一样;
4. XYisFree() 分别测试有棋子和没棋子的情况

```
@Test
public void testBoard() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);

    // Board.printBoard()
    assertTrue(game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 3)));
    assertTrue(game.capture(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 3)));
    game.board().printBoard();

    // Board.PositionXY(x, y)
    assertNotNull(game.board().positionXY(0, 1));
    assertNotNull(game.board().positionXY(4, 6));
    assertEquals("BP4", game.board().positionXY(4, 6).piece().name());
    assertNull(game.board().positionXY(3, 3).piece());

    // Board.pieceXY(x, y)
    assertEquals("WR0", game.board().pieceXY(0, 0).name());
    assertEquals("BP5", game.board().pieceXY(5, 6).name());

    // Board.XYisFree(x, y)
    assertEquals(null, game.board().XYisFree(4, 5));
    assertEquals(player1, game.board().XYisFree(1, 1));
    assertEquals(player2, game.board().XYisFree(7, 6));
}
```

```
}
```

### 3.3.3.4 testPosition()

测试位置的方法。

1. 测试位置的棋子的名字；
2. 测试查看横纵坐标、所属玩家（当无棋子时，可能为空）；
3. 测试更改 position 的 piece

```
@Test
public void testPosition() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);

    // Position.piece()
    assertEquals("BQ0", game.board().positionXY(3, 7).piece().name());
    assertEquals("WK0", game.board().positionXY(4, 0).piece().name());

    // Position.x()
    assertEquals(4, game.board().positionXY(4, 0).x());

    // Position.y()
    assertEquals(5, game.board().positionXY(2, 5).y());

    // Position.player()
    assertEquals(player2, game.board().positionXY(6, 6).player());

    // Position.modify
    assertEquals(player1, game.board().XYisFree(1, 1));
    assertEquals(true, game.board().positionXY(1, 1).modifyPieceAs(null));
    assertNull(game.board().XYisFree(1, 1));
}
```

### 3.3.3.5 testPlayer()

测试玩家。

1. 测试双方先后手的标签；
2. 测试 player 所属游戏；
3. 测试 player 拥有的所有 pieces，是否在对应坐标的棋盘位置上存在；

```
@Test
public void testPlayer() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);
}
```

```

// Player.isFirst()
assertEquals(true, player1.isFirst());
assertEquals(false, player2.isFirst());

// Player.game()
assertEquals(game, player2.game());

// Player.pieces()
for (int i = 0; i < game.board().boardLength(); i++) {
    for (int j = 0; j < game.board().boardLength(); j++) {
        if (game.board().pieceXY(i, j) != null)
            assertEquals(true, player1.pieces().contains(game.board().pieceXY(i, j))
                || player2.pieces().contains(game.board().pieceXY(i, j)));
    }
}
}

```

### 3.3.3.6 testPut()

测试放棋操作。

1. 当选定棋子在棋盘上/不在棋盘上时；
2. 当棋子放的位置有/无棋子；

```

@Test
public void testPut() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);

    // put
    assertEquals(false, game.put(player1, player1.findPieceByName("WP0"), game.board().positionXY(0, 1)));
    assertEquals(false, game.put(player2, player2.findPieceByName("BN1"), game.board().positionXY(5, 4)));

    // After capture
    assertTrue(game.capture(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 1)));
    assertTrue(game.put(player1, player1.findPieceByName("WP0"), game.board().positionXY(0, 6)));
}

```

### 3.3.3.7 testMove()

测试移动棋子操作。

1. 选定格的棋子是否为本方；
2. 选定格的棋子是否存在
3. 目标格的棋子是否为空；
4. 移动之后选定格格子是否为空，而目标格是否不空且为之前选定格的棋子；

```

@Test
public void testMove() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
}

```

```

player2.pieces = game.pieces(false);

// move
assertEquals(true, game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 3)));
assertNull(game.board().positionXY(0, 1).piece());
assertNotNull(game.board().positionXY(0, 3).piece());
assertEquals(false, game.move(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 4)));
assertEquals(false, game.move(player2, game.board().positionXY(6, 0), game.board().positionXY(5, 2)));
assertEquals(true, game.move(player1, game.board().positionXY(1, 0), game.board().positionXY(2, 2)));
assertEquals(false, game.move(player1, game.board().positionXY(1, 0), game.board().positionXY(0, 2)));
assertNull(game.board().positionXY(0, 2).piece());
assertNull(game.board().positionXY(1, 0).piece());
assertEquals(false, game.move(player2, game.board().positionXY(3, 7), game.board().positionXY(4, 7)));
assertEquals(false, game.move(player1, game.board().positionXY(2, 2), game.board().positionXY(4, 1)));
assertNotNull(game.board().positionXY(4, 1).piece());
assertNotNull(game.board().positionXY(2, 2).piece());
assertEquals(true, game.move(player2, game.board().positionXY(0, 6), game.board().positionXY(0, 1)));
assertEquals(player2, game.board().positionXY(0, 1).piece().player());
assertEquals("BP0", game.board().positionXY(0, 1).piece().name());
assertEquals(player1, game.board().positionXY(0, 3).piece().player());

// sumPiece
assertEquals(16, player2.sumPiece());
}

```

### 3.3.3.8 testCaptureAndPut()

结合 put 操作测试 capture 吃子操作。

1. 吃子的起始格是否为本方；
2. 吃子的起始格是否为空；
3. 吃子的目标格是否为对方；
4. 吃子的目标格是否为空；
5. 查看吃子之后两个格子的棋子名字是否为期望；

```

@Test
public void testCaptureAndPut() {
    // init
    final Game game = Game.newGame("chess");
    final Player player1 = new Player(game, "p1", true);
    final Player player2 = new Player(game, "p2", false);
    game.setPlayers(player1, player2);
    player1.pieces = game.pieces(true);
    player2.pieces = game.pieces(false);

    // capture
    assertEquals(true, game.capture(player1, game.board().positionXY(0, 1), game.board().positionXY(0, 6)));
    assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(2, 1)));
    assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(1, 3)));
    assertEquals(true, game.capture(player1, game.board().positionXY(0, 6), game.board().positionXY(1, 6)));
    assertEquals(false, game.capture(player1, game.board().positionXY(1, 1), game.board().positionXY(1, 6)));
    assertEquals("WP0", game.board().pieceXY(1, 6).name());
    assertEquals(player1.findPieceByName("WP0"), game.board().pieceXY(1, 6));
    assertEquals(true, game.capture(player1, game.board().positionXY(2, 1), game.board().positionXY(2, 6)));
    assertEquals(13, player2.sumPiece());

    // put
    assertEquals("BP1", player2.findPieceByName("BP1").name());
    assertNull(game.board().positionXY(0, 4).piece());
    assertTrue(game.put(player2, player2.findPieceByName("BP0"), game.board().positionXY(0, 4)));
}

```

```
assertFalse(game.put(player2, player2.findPieceByName("BP1"), game.board().positionXY(0, 4)));
assertFalse(game.put(player1, player2.findPieceByName("BP1"), game.board().positionXY(0, 4)));
assertFalse(game.put(player2, player2.findPieceByName("BP0"), game.board().positionXY(0, 4)));
assertNull(player1.findPieceByName("BP1"));
assertFalse(game.put(player2, player1.findPieceByName("BP1"), game.board().positionXY(1, 4)));
assertNull(game.board().positionXY(8, -1));
assertFalse(game.put(player2, player2.findPieceByName("BP2"), game.board().positionXY(8, -1)));
}
```

### 3.3.3.9 ChessTest 测试结果

All	8	Passed	8
P3.ChessTest			
> testInit	Passed	0s	🔗
> testPiece	Passed	0s	🔗
> testBoard	Passed	0s	🔗
> testPosition	Passed	0s	🔗
> testPlayer	Passed	0s	🔗
> testPut	Passed	0.01s	🔗
> testMove	Passed	0s	🔗
> testCaptureAndPut	Passed	0s	🔗

### 3.3.3.10 GoTest 测试结果

All	7	Passed	7
P3.GoTest			
> testInit	Passed	0s	🔗
> testPiece	Passed	0s	🔗
> testBoard	Passed	0.02s	🔗
> testPosition	Passed	0s	🔗
> testPlayer	Passed	0s	🔗
> testPut	Passed	0.01s	🔗
> testCaptureAndPut	Passed	0s	🔗

## 4 实验进度记录

日期	时间段	计划任务	实际完成情况
2020-03-09	晚上	初始化项目	完成
2020-03-10	中午	Problem1 3.1.1-3.1.2	完成
2020-03-10	晚上	Problem1 3.1.3.1 Edge Graph	完成
2020-03-11	晚上	Problem1 3.1.3.2 Vertex Graph	完成
2020-03-12	下午	通过 Graph Instance Test	完成
2020-03-12	晚上	Problem1 3.1.5 Graph Poet	完成
2020-03-13	上午	Problem1 3.1.5 Test	完成
2020-03-13	上午	Problem2 3.2 整体完成	完成
2020-03-13	晚上	Problem3 设计框架 写 AF&RI	完成
2020-03-14	上午	Problem3 完成框架	完成
2020-03-14	下午	实现 Action 接口、Player、Board、Position、Piece 具体功能	完成
2020-03-14	晚上	完善上述功能，修改 bug	完成
2020-03-15	上午	实现 Game 接口、chessAction、goAction 功能	完成
2020-03-15	晚上	实现 chessGame、goGame 功能，调试测试用例	完成
2020-03-16	晚上	通过 chessGame 测试	完成
2020-03-17	下午	通过 goGame 测试	完成
2020-03-17	晚上	验收完成	完成

## 5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
在做 P1 时，ConcreteVerticesGraph 中仅给了一个 rep，实现起来有些蹩脚	因为图的表示需要点集和邻接表，而要求又只能用一个点集 rep，所以索性把每个顶点的邻接链表放到 Vertices 类中。
P1 中 Edge 类要求是一个 immutable 类，其中不能有 mutator 方法，而具体实现方法中很多操作要修改边相关。	凡是需要修改边的地方，全部改为清除后再添加
测试优先的编程习惯非常不适应	没办法了，既然测试优先是个好习惯，只能硬着头皮上了。最开始写 P1 测试时很蹩脚，不过后来就把测试当成数学里的分类讨论题来做了



## 6 实验过程中收获的经验、教训、感想

### 6.1 实验过程中收获的经验教训

- 设计一个完整的软件，需要从底层的 ADT 设计起；
- 检验 ADT 的正确性后才能检测软件的正确性
- ADT 的 Spec 在前期的设计非常重要，比 implementation 还要重要
- 要认真地写 spec，也包括细节的功能和注释

### 6.2 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？
  - ADT 编程更模块化，易于测试和调试，以及划分单元
- (2) 使用泛型和不使用泛型的编程，对你来说有何差异？
  - 泛型能兼容指定类型，所以总体来说没有差异
- (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？
  - 优势是能够在写测试用例的就把 AF 和 RI 考虑清楚，完成 Spec
  - 不是特别适应，在编程后还是会对测试用例有所补充
- (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？
  - 复用能减少开发成本，是单位量程序效益增大
- (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？
  - 能适应，学会了先写 Spec 再写 implementation
  - 设计 ADT 非常有意思，设计可能比实现更为重要
- (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？
  - 意义是提高程序的正确性和健壮性，规避错误
  - 愿意，我认为这对于测试、合作编程都很有意义
- (7) 关于本实验的工作量、难度、deadline。
  - 工作量适中，难度适中
  - 实话说……**deadline 太迟**（~~第一节实验课就验收了~~），三周就可以
- (8) 《软件构造》课程进展到目前，你对该课程有何体会和建议？
  - 讲基础理论和实际编程结合得不错，我建议实验可以拆散开来