

**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 | 许健 |
| 学号 | 1183710113 |
| 班号 | 1837101 |
| 电子邮件 | [941197279@qq.com](mailto:941197279@qq.com) |
| 手机号码 | 18945062342 |

**目录**

[1 实验目标概述 1](#_Toc34543435)

[2 实验环境配置 1](#_Toc34543436)

[3 实验过程 1](#_Toc34543437)

[3.1 Poetic Walks 1](#_Toc34543438)

[3.1.1 Get the code and prepare Git repository 2](#_Toc34543439)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc34543440)

[3.1.3 Problem 2: Implement Graph <String> 2](#_Toc34543441)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc34543442)

[3.1.3.2 Implement ConcreteVerticesGraph 5](#_Toc34543443)

[3.1.4 Problem 3: Implement generic Graph<L> 7](#_Toc34543444)

[3.1.4.1 Make the implementations generic 7](#_Toc34543445)

[3.1.4.2 Implement Graph.empty() 8](#_Toc34543446)

[3.1.5 Problem 4: Poetic walks 10](#_Toc34543447)

[3.1.5.1 Test GraphPoet 10](#_Toc34543448)

[3.1.5.2 Implement GraphPoet 10](#_Toc34543449)

[3.1.5.3 Graph poetry slam 11](#_Toc34543450)

[3.1.6 Before you’re done 11](#_Toc34543451)

[3.2 Re-implement the Social Network in Lab1 12](#_Toc34543452)

[3.2.1 FriendshipGraph类 12](#_Toc34543453)

[3.2.2 Person类 13](#_Toc34543454)

[3.2.3 客户端main() 13](#_Toc34543455)

[3.2.4 测试用例 13](#_Toc34543456)

[3.2.5 提交至Git仓库 14](#_Toc34543457)

[3.3 Playing Chess 14](#_Toc34543458)

[3.3.1 ADT设计/实现方案 14](#_Toc34543459)

[3.3.2 主程序MyChessAndGoGame设计/实现方案 21](#_Toc34543460)

[3.3.3 ADT和主程序的测试方案 25](#_Toc34543461)

[4 实验进度记录 27](#_Toc34543462)

[5 实验过程中遇到的困难与解决途径 28](#_Toc34543463)

[6 实验过程中收获的经验、教训、感想 28](#_Toc34543464)

[6.1 实验过程中收获的经验和教训 28](#_Toc34543465)

[6.2 针对以下方面的感受 28](#_Toc34543466)

# 实验目标概述

本次实验训练抽象数据类型（ADT）的设计、规约、测试，并使用面向对象

编程（OOP）技术实现 ADT。具体来说：

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

# 实验环境配置

安装EclEmma。在eclipse中的install new software中访问EclEmma的更新站点安装。

Lab2仓库的URL：<https://github.com/ComputerScienceHIT/Lab2-1183710113>

# 实验过程

## Poetic Walks

根据要求完成接口Graph<L> 的两个具体实现：ConcreteEdgesGraph和ConcreteVerticesGraph。Graph<L> 接口要求实现如下几个方法：

add：添加新节点；

set：添加新边；

remove：移除节点；

vertices：获得所有的节点集合；

sources(target)：获得以target为目标节点的边的起始节点；

targes(source)：获得以source为起始节点的边的目标节点。

Poet：给定一组单词作为语料库，对于两个相邻的单词a和b，认为存在一条由a到b的有向边，通过Graph接口构造有向图。再给定一条由单词组成的句子，如果句子中两个相邻单词之间在Graph图中有一个桥接词则将桥接词插入到两单词之间。

### Get the code and prepare Git repository

访问实验手册给出的URL地址，下载实验P1所需的代码。

在本地创建文件夹git，在文件夹下创建目录Lab2，在Lab2中打开git Bash：

使用命令git init创建Lab2的仓库。

### Problem 1: Test Graph <String>

根据Graph的spec，利用等价类划分的思想确定测试策略并编写测试用例

测试策略：

1. Graph.add方法测试：

点在图中，点不在图中

1. Graph.set方法测试：

按起点分：点在图中，点不在图中

按终点分：点在图中，点不在图中

按权重分：权不等于0，权等于0

1. Graph.remove方法测试：

点在图中，点不在图中

1. Graph.vertices方法测试：

图中没有点，图中有多个点

1. Graph.sources方法、Graph.targets方法测试：

给定点不在图中

给定点在图中：图中没有相应的点，图中有相应的点

### Problem 2: Implement Graph <String>

完成接口的两个实现，实现接口所声明的六大方法。

#### Implement ConcreteEdgesGraph

##### Edge类的实现

1. 数据域

在Edge类中设置保存边的起点、终点和权重的边的数据域，由于要求边是不可变的，所以把所有的数据域都使用final做了限定。声明如下：

private final int weight;

private final String source, target;

在一个Edge类的变量中，source和target不能为空，权重必须是正整数。

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 方法功能 |
| Edge(String source, String target, int weight) | 构造方法，创建一条边 |
| private void checkRep() | 检查不变量 |
| public String getSource() | 获得边的起点source |
| public String getTarget() | 获得边的终点target |
| public int getWeight() | 获得边的权重weight |
| public String toString() | 把边按照期望的形式输出 |

##### ConcreteEdgesGraph类的实现

1. 数据域

Set类的vertices变量用来保存所有的顶点，List类的edges变量用来保存所有的边。声明如下：

private final Set<String> vertices = new HashSet<>();

private final List<Edge> edges = new ArrayList<>();

1. 内部方法

（1）checkRep()：判断所有的点都不为空，所有的边都不为空，边数和点数满足条件：边数 <= 点数 \* (点数 - 1)。

（2）add(String vertex)：如果vertex不空且vertices域中不包含vertex，把vertex加入到vertices中，返回true，否则返回false。

（3）set(String source, String target, int weight)：先检查是否在edges中有以source为起点，target为终点的边。

如果有，把原来的边的权重记录下来并移除这条边，再判断weight是否为0，如果不为0，新建一条以source为起点，target为终点，weight为权重的边，将其加入到edges中，否则不做任何操作，此时相当于移除这条边，最后返回记录下来的原来的边的权重。

如果edges中不存在这样的边，那么先判断这两个顶点是否在vertices中，如果某个顶点不在vertices中，那么先把它加入到vertices中，然后再判断weight是否为0，如果不为0，新建一条以source为起点，target为终点，weight为权重的边，将其加入到edges中，否则不做任何操作。然后检查不变量，返回表示原来不存在边的0。

（4）remove(String vertex)：先判断vertex是否在vertices中，如果不在直接返回false，如果在其中则从vertices中删除vertex，然后遍历edges中的所有边，把与vertex有关(以其为起点或终点)的所有的边都删除。检查不变量然后返回true。

（5）vertices()：使用Collections.unmodifiableSet(vertices)方法把vertices封装成一个不可修改的Set类型返回给外部。

（6）sources(String target)：新建一个Map<String, Integer>类型的变量sources用来保存所有以target为终点的边的起点及权重，然后遍历edges，如果某个边的终点是target，那么把这条边的起点和权重加入到sources中，最后返回使用Collections.unmodifiableMap(sources)封装过的不可修改的Map类型的sources。

（7）targets(String source)：新建一个Map<String, Integer>类型的变量targets用来保存所有以source为起点的边的终点及权重，然后遍历edges，如果某个边的起点是source，那么把这条边的终点和权重加入到targets中，最后返回使用Collections.unmodifiableMap(targets)封装过的不可修改的Map类型的targets。

（8）toString()：如果vertices是空的，返回“”，也就是什么也没有。否则就把所有的顶点都加入到要返回的字符串中。然后再判断edges是不是空的，如果不是就把所有的边都加入到要返回的字符串中。最后把字符串返回。

##### ConcreteEdgeGraphTest实现

由于ConcreteEdgeGraphTest是由GraphInstanceTest继承而得的，所以只需要测试添加的ConcreteEdgeGraph.toString方法和Edge类中的方法即可。

ConcreteEdgeGraph.toString方法测试策略：空图，图内有点但没有边，图内既有点又有边。

Edge类的测试策略：

getter方法：测试对于一条边是否能够正常获得它的各个属性

toString方法：由于Edge的数据域不可能为空，只需要测试对于一条边能够按照预期的形式输出。

#### Implement ConcreteVerticesGraph

##### Vertex类的实现

1. 数据域

vertex变量用来保存顶点的label，Map类型的targets用来保存以vertex为起点的所有的边的终点和权重。声明如下：

**private final String vertex;**

**private final Map<String, Integer> targets;**

在一个Vertex变量中，vertex变量不能为空，targets中所有的终点不能为空，所有的权重必须是正整数。

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 方法功能 |
| **public Vertex(String vertex)** | 构造方法，创建一个顶点 |
| **private void checkRep()** | 检查不变量 |
| **public String getVertex()** | 获得顶点的名称vertex |
| **public Map<String, Integer> getTargets()** | 以vertex为起点的边集targets |
| **public int add(String target, int weight)** | 添加以vertex作为起点的边，其终点为target，权重为weight |
| **public int remove (String target)** | 删除到target这个终点的边，即从targets中移除target |
| **public String toString()** | 把边按照期望的形式输出 |

##### ConcreteVerticesGraph类的实现

1. 数据域

保存所有顶点的一个List变量vertices。声明如下：

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

1. 内部方法

（1）checkRep()：判断所有的点都不为空，且不包含重复点。

（2）add(String vertex)：如果vertex不空且vertices域中不包含vertex，把vertex加入到vertices中，返回true，否则返回false。

（3）set(String source, String target, int weight)：先检查是否在vertices中有source和target这两个点，如果没有target则先把这个点加入到vertices中。

如果没有source这个点，则把source也加入到vertices中，再判断权重是否为0，如果不为0，则把边加入到source的边集中，否则直接返回0。

如果存在source这个点，那么判断weight是否为0，如果不为0, 把边加入到source的边集中，返回原先边的权重；如果为0，则把以target为终点的这条边删除，返回原先边的权重。

（4）remove(String vertex)：先遍历vertices中所有的顶点，把以vertex为终点的边都删除，然后在vertices中找到名为vertex的点，将其删除并返回true，如果没有返回，说明不存在名为vertex的点，返回false即可。

（5）vertices()：遍历所有的顶点，取出顶点的名字加入到名字集合中，然后将这个集合用Collections.unmodifiableSet封装后返回。

（6）sources(String target)：新建一个Map<String, Integer>类型的变量sources用来保存所有以target为终点的边的起点及权重，然后遍历vertices，如果某个顶点vertex.getTargets包含到target的边，那么把这个点和这条边的权重加入到targets中。

最后返回使用Collections.unmodifiableMap(sources)封装过的不可修改的Map类型的sources。

（7）targets(String source)：遍历vertices找到source这个点，使用getTargets方法得到终点和对应边的集和，最后返回使用Collections.unmodifiableMap()封装过的不可修改的边集。

如果没找到，先创建new HashMap<L, Integer>()，然后返回它。

（8）toString()：遍历vertices中所有的顶点，调用Vextex中的toString方法输出以每个顶点为起点的所有边，把这个字符串加入到要返回的字符串中，最后把字符串返回。

##### ConcreteVerticesGraphTest的实现

由于ConcreteVerticesGraphTest是由GraphInstanceTest继承而得的，所以只需要测试添加的ConcreteVerticesGraph.toString方法和Vertex类中的方法即可。

ConcreteVertexGraph.toString方法测试策略：空图，图内有点但没有边，图内既有点又有边。

Vertex类的测试策略：

getter方法：测试对于一个点是否能够正常获得它的点的名称和以它为起点的边集

add方法：按加入的边分：原先不存在这条边，原先存在这条边

remove方法：按被删除的边分：原先不存在这条边，原先存在这条边

toString方法：按边分：只有顶点没有以之为起点的边，有以之为起点的边

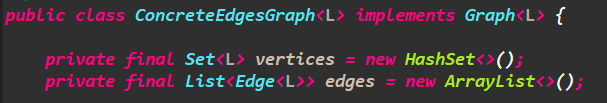
### Problem 3: Implement generic Graph<L>

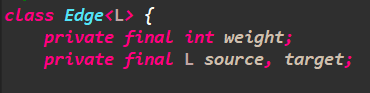
更改为泛型实现

#### Make the implementations generic

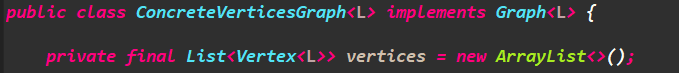
把所有的用到的String都替换成泛型实现

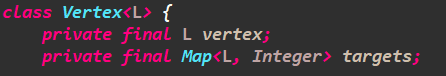
ConcreteEdgesGraph和Edge：





ConcreteVerticesGraph和Vertex：

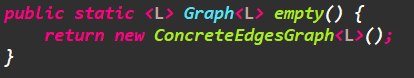




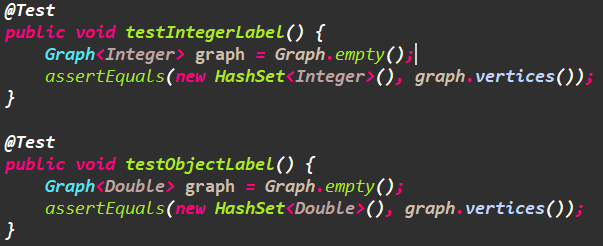
以及方法中涉及到的地方都改为泛型L。

#### Implement Graph.empty()

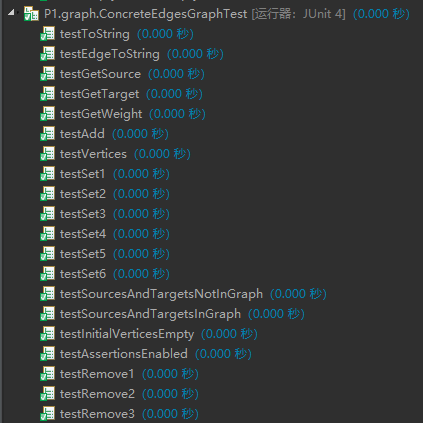
修改Graph.empty为：

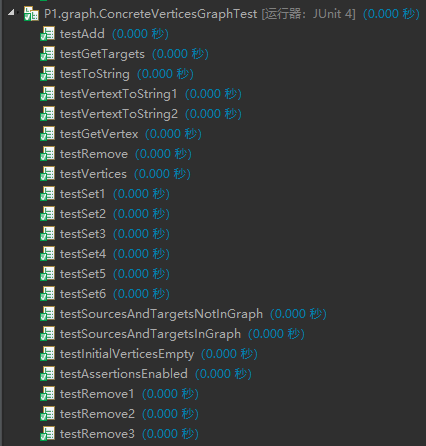


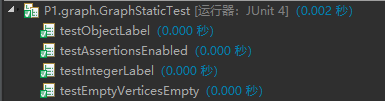
在GraphStaticTest中编写两个其他immutable类型的label测试：



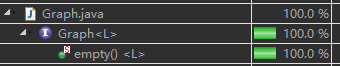
测试结果：

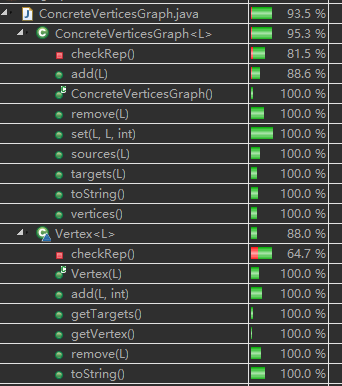


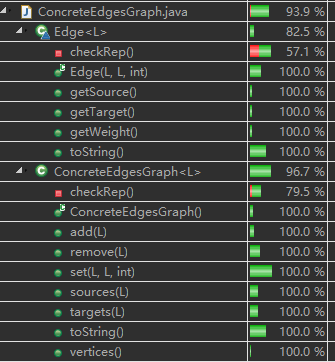




代码覆盖度：主要的损失在于由于checkRep是私有的没办法测试







### Problem 4: Poetic walks

#### Test GraphPoet

测试策略：

测试构造方法：空文件，单行输入，多行输入

测试poem方法：

按桥接点分：空输入，在语料库中没有桥接点的输入，在语料库中有桥接点的输入

按权重分：权重等于1，权重大于1

测试toString方法：空图，非空图

根据测试策略编写测试用例。

#### Implement GraphPoet

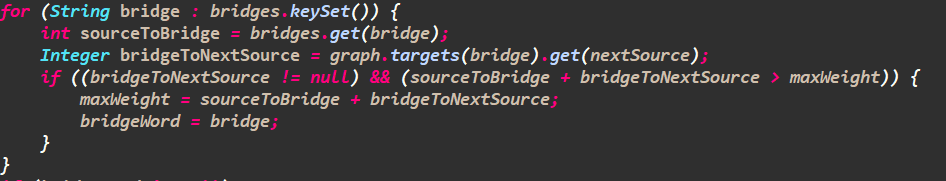
数据域：private final Graph<String> graph;由语料生成的一张图。

##### 实现构造方法GraphPoet

从文件中按行读入所有的语料然后根据要求以空格分割出所有的单词保存在一个名为words的List列表中，然后从words中依次读出第i个和第i+1个单词，如果存在以第i个单词名为起点，以第i+1个单词名为终点的边，那么就在原来的权重上加1，否则直接创建一条新的权重为1边，将其放入图中。

##### 实现poem

如果input为空，直接返回一个空字符串。否则，把输入的字符串以空格分割成单词存入List类型的words中，然后遍历words，判断第i个单词和第i+1个单词是否有桥接词的存在，即以第i个单词为起点，在创建好的graph中找到它的所有终点即为所有可能成为桥接词的点，再分别以所有的终点为起点找它们的终点，如果某个点的终点是我们的第i+1个单词，那么它就是我们找到的桥接词，再比较所有找到的桥接词的路径长度：第i个单词到桥接词的权重+桥接词到第i+1个单词的权重，取这个值最大时所经过的桥接词。



然后把找到的桥接词添加到待输出的字符串中，如果没有找到桥接词，则不需要这一步。

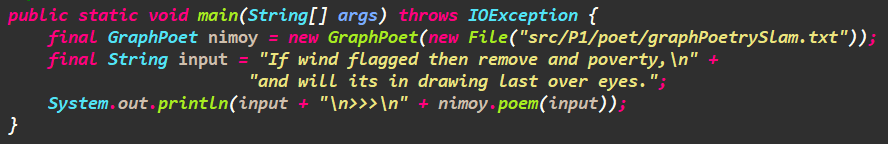
最后把得到的字符串返回。

##### 实现toString

直接调用ConcreteEdgesGraph的toString方法把所创建的图输出。

#### Graph poetry slam

客户端程序如下：



### Before you’re done

1. 通过Git提交当前版本到GitHub上的Lab2仓库：

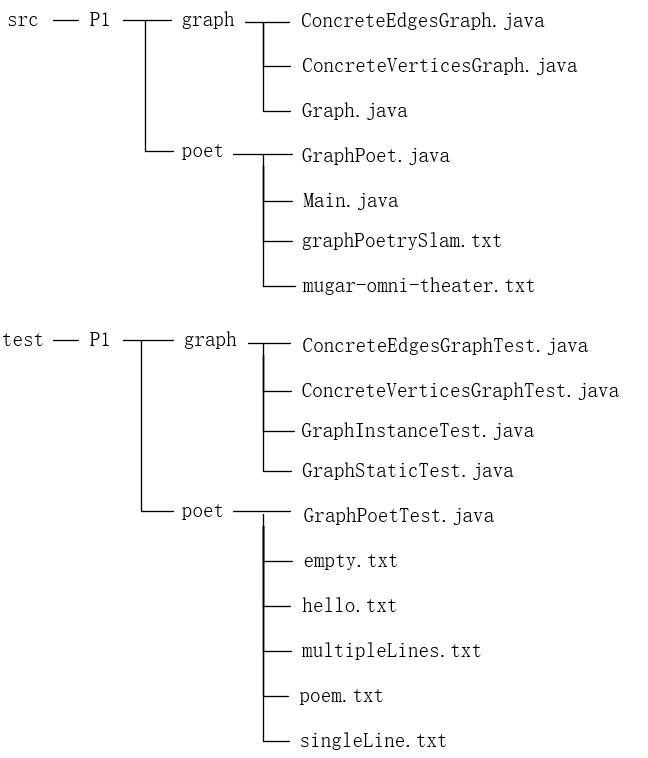
使用git add src/P1和git add test/P1将文件添加到仓库的缓存区；

使用git commit -m “message”命令将缓存区的文件提交到git仓库。

使用git remote add origin <https://github.com/ComputerScienceHIT/Lab2-1183710113> 将本地仓库与Github的远程仓库链接；

使用git push -u origin master命令将本地仓库推送到Github的仓库上。

1. 项目的目录结构树状示意图。



## Re-implement the Social Network in Lab1

利用P1部分实现的图结构重新实现Lab1中的FriendshipGraph。

### FriendshipGraph类

1. 数据域：private final Graph<Person> graph;是一张Person类的图。
2. 内部方法：

（1）FriendshipGraph()：构造方法，创建一张ConcreteVerticesGraph空图。

（2）addVertex(Person person)：调用ConcreteVerticesGraph.add方法直接向图中添加。

（3）addEdge(Person person1, Person person2) ：设置边权重为1，调用ConcreteVerticesGraph.set方法直接向图中添加边。

（4）getDistance(Person person1, Person person2)：先判断这两个人是不是同一个人，如果是直接返回0。否则使用BFS算法求两个人之间的最短距离。如果没找到则说明这两个人不存在关系，返回-1。

### Person类

1. 数据域：private final String name;代表这个人的名字。
2. 内部方法：

（1）Person(String name)：创建一个名为name的人。

（2）getName()：获得这个人的名字。

（3）equals(Person anotherPerson)：判断这两个人的name数据域是否相等，如果相等，返回true，否则返回false。

### 客户端main()



### 测试用例

测试策略：

addVertex方法：被添加的点在图中，被添加的点不在图中；

addEdge方法：添加单向边，添加双向边；

getDistance方法：到达自身，不能到达，单向到达，多条路径到达

### 提交至Git仓库

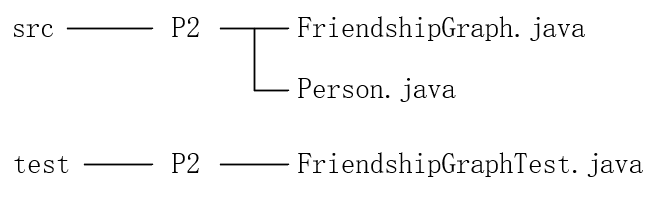
1. 通过Git提交当前版本到GitHub上的仓库。

使用git add src/P2和git add test/P2将文件添加到仓库的缓存区；

使用git commit -m “message”命令将缓存区的文件提交到git仓库；

使用git push -u origin master命令将本地仓库推送到Github的仓库上。

1. 项目的目录结构树状示意图。



## Playing Chess

### ADT设计/实现方案

#### Position类的实现

1. 数据域

用来表示位置的横纵坐标，声明如下：

private final int x;

private final int y;

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 作用 |
| public Position(int x, int y) | 构造方法 |
| public int getX() | 获得横坐标 |
| public int getY() | 获得纵坐标 |
| public boolean equals(Object position) | 判断两个位置是否相等 |
| public int hashCode() | 返回对象哈希值 |

1. 说明

Position是一个immutable类型，表示棋盘上的一个位置

Rep invariant:

x, y是整数

Abstraction function:

AF(x, y) = 一个从坐标到实际点的映射

Safety from rep exposure:

所有的数据域都是私有的且用final限定

#### Piece类的实现

1. 数据域

type表示棋子的类型，player表示棋子属于哪个棋手，声明如下；

private final String type;

private final Player player;

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 作用 |
| public Piece(String type, Player player) | 构造方法，创建棋子 |
| public String getType() | 获得棋子的类型 |
| public Player getPlayer() | 获得拥有该棋子的棋手 |
| public boolean equals(Object piece) | 判断两个棋子是否相等 |
| public int hashCode() | 返回对象哈希值 |

1. 说明

Piece是一个immutable类型，表示棋子

Rep invariant:

player不为空

Abstraction function:

AF(type, player) = 一个到实际的类型为type(针对国际象棋)，所属棋手为player的棋子

Safety from rep exposure:

所有的数据域都是私有的且用final限定

#### Board类的实现

1. 数据域

代表一个棋盘，一个位置对应一个棋子，声明如下：

private Map<Position, Piece> board;

1. 内部方法

（1）public Board(int size, Player white, Player black)：构造方法，初始化一张棋盘，根据size创建相应大小的棋盘，如果size为8，说明是国际象棋，然后从配置文件中读取棋子，放在棋盘上，如果size为18，说明是围棋，直接把所有位置初始化为空闲即可。

（2）public void put(Position position, Piece piece)：在棋盘上的合法position位置放置棋子piece。

（3）public void take(Position position)：从棋盘上的合法位置取走棋子，把棋盘上的position位置设置为空闲。

（4）public Map<Position, Piece> getBoard()：返回一个不可修改的棋盘状态。

（5）public Piece get(Position position)：获得棋盘上某个合法位置的状态，没有棋子就返回null，有棋子就返回该棋子。

1. 说明

Board是一个mutable的类型，代表一个棋盘。

Rep invariant:

board不为空

Abstraction function:

AF(board) = 一个真实的棋盘

Safety from rep exposure:

所有的数据域都是私有的

获得棋盘时使用Collections.unmodifiableMap()方法转换成了不可修改的类型

获得棋盘上的某个位置的棋子时，棋子是不可变的

#### Player类的实现

1. 数据域

表示棋手的名字，声明如下：

private final String name;

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 作用 |
| public Player(String name) | 构造方法，创建棋手 |
| public String getName() | 返回棋手姓名 |
| public boolean equals(Object anotherPlayer) | 判断两个棋手是不是同一个人 |
| public int hashCode() | 返回对象哈希值 |

1. 说明

Player是一个immutable类型，表示棋手

Rep invariant:

name不为空

Abstraction function:

AF(name) = 到真实的棋手的映射

Safety from rep exposure:

所有的数据域都是私有的且用final限定

#### Action类的实现

1. 数据域

表示一个棋盘，声明如下：

private Board board;

1. 内部方法

（1）public Action(int size, Player white, Player black)：构造方法，创建操作对象棋盘。

（2）public boolean putPiece(Player player, Piece piece, int x, int y, PrintWriter output)：针对围棋的落子操作，棋手把指定棋子放在指定位置。首先判断位置是否合法，再判断落子位置是否有空闲，再判断放置的棋子是不是属于棋手。如果条件都满足，就把棋子放在棋盘的相应位置返回true。否则打印提示信息并返回false。

（3）public boolean takePiece(Player player, int x, int y, PrintWriter output)：针对围棋的提子操作，棋手把指定棋子从指定位置取走。首先判断位置是否合法，再判断提子位置是否有空闲，再判断所提的棋子是不是属于棋手。如果条件都满足，就把棋子放在棋盘的相应位置返回true。否则打印提示信息并返回false。

（4）public boolean movePiece(Player player, int sourceX, int sourceY, int targetX, int targetY, PrintWriter output)：针对国际象棋的移动操作，棋手从起始位置移动棋子到目标位置。首先判断两个位置坐标是否合法，再判断目的位置是否空闲，再判断起始位置是否有棋子，最后判断移动的棋子是不是属于棋手。如果条件都满足，把原来的位置设为空闲，目标位置处放置原来位置的棋子。否则打印提示信息并返回false。

（5）public boolean eatPiece(Player player, int sourceX, int sourceY, int targetX, int targetY, PrintWriter output)：针对国际象棋的吃子动作，第一个位置的棋子吃掉第二个位置的棋子。首先判断两个位置坐标是否合法，再判断两个位置是不是同一位置，再判断目的位置是否空闲，再判断起始位置是否空闲，再判断起始棋子是不是属于棋手，最后判断目标棋子是不是属于对方棋手。如果条件都满足，把起始的位置设为空闲，目标位置处放置起始位置的棋子。否则打印提示信息并返回false。

（6）public boolean query(Player player,int x, int y, PrintWriter output)：棋盘上的查询操作，查询某位置是否空闲，如果不空闲输出是谁的什么棋子。首先判断位置是否合法。如果合法判断是否空闲，如果空闲，打印该位置空闲，否则打印该位置是哪一方棋手的什么类型棋子。否则输出提示信息并返回false。

（7）public int count(Player player, PrintWriter output)：遍历棋盘，计算棋盘上双方各有多少棋子。

（8）public Map<Position, Piece> getBoard()：调用Board.getBoard()方法，返回不可改变的棋盘状态。

1. 说明

是一个mutable的类型表示对国际象棋或围棋的操作。

Rep invariant:

棋盘不为空。

Abstraction function:

一个从抽象的操作到实际对棋盘上的棋子的动作的映射。

Safety from rep exposure:

所有的数据域都是私有的。

获得棋盘board的状态时调用了Board类的getBoard方法，得到了一个不可修改的棋盘。

#### Game接口的实现

内部方法

（1）public default void game(Scanner input, Player currentPlayer, Player waitPlayer) throws IOException：游戏的主体部分，通过调用这个方法可以开始一盘棋游戏。

（2）public void menu(Player player)：棋类游戏的菜单，不同的棋类有不同的菜单。

（3）public void play(int actNumber, Player player, Scanner input, PrintWriter output)：玩游戏的时候的动作，根据输入的actNumber的不同调用action的不同的方法。

（4）public default String history(File file) throws IOException：读取操作历史的文件，将其转化为字符串返回。

#### Go类的实现

1. 数据域

代表白方和黑方棋手，以及下棋行为，声明如下：

private final Player white;

private final Player black;

private Action action;

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 作用 |
| public Go(String player1, String player2) | 构造方法，创建围棋游戏 |
| public void game(Scanner input) throws IOException | 调用Game.game()，开始围棋的一盘棋游戏 |
| public void menu(Player player) | 实现Game接口中的menu |
| public void play(int actNumber, Player player, Scanner input, PrintWriter output) | 实现Game接口中的play |

1. 说明

Go是一个mutable类型。代表围棋游戏。

Rep invariant:

white不为空

black不为空

action不为空

Abstraction function:

AF(white,black,action) = 一个到现实中的围棋游戏的映射

Safety from rep exposure:

所有的数据域都是私有的

#### Chess类的实现

1. 数据域

代表白方和黑方棋手，以及下棋行为，声明如下：

private final Player white;

private final Player black;

private Action action;

1. 内部方法

|  |  |
| --- | --- |
| 方法 | 作用 |
| public Chess(String player1, String player2) | 构造方法，创建国际象棋游戏 |
| public void game(Scanner input) throws IOException | 调用Game.game()，开始国际象棋的一盘棋游戏 |
| public void menu(Player player) | 实现Game接口中的menu |
| public void play(int actNumber, Player player, Scanner input, PrintWriter output) | 实现Game接口中的play |

1. 说明

Chess是一个mutable类型。代表国际象棋游戏。

Rep invariant:

white不为空

black不为空

action不为空

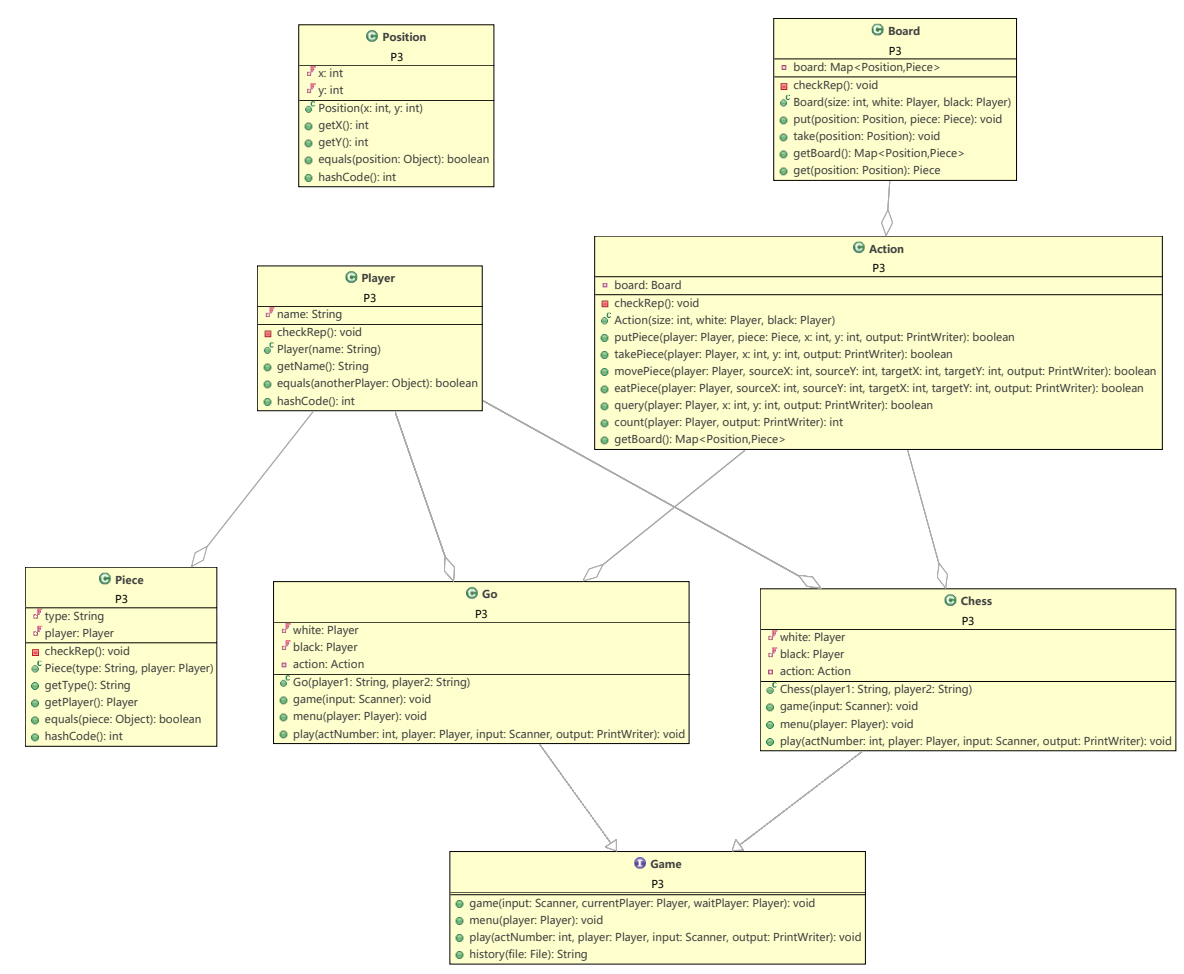
Abstraction function:

AF(white,black,action) = 一个到现实中的国际象棋游戏的映射

Safety from rep exposure:

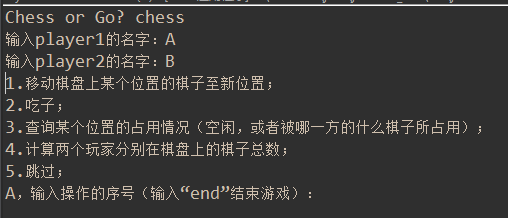
所有的数据域都是私有的

#### UML class diagram

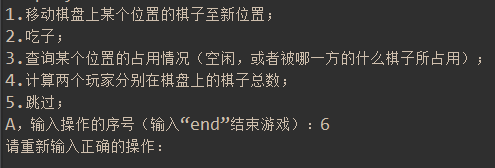


### 主程序MyChessAndGoGame设计/实现方案

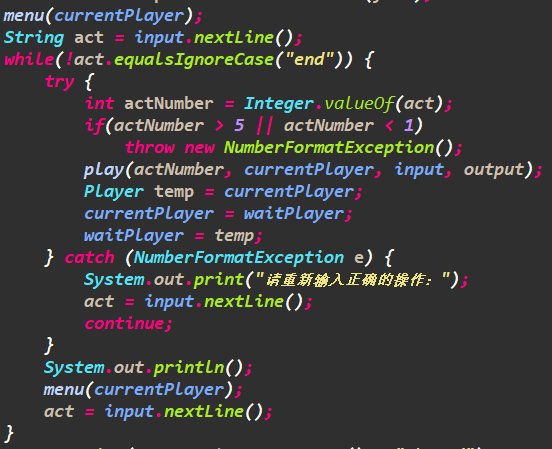
主程序的设计：先要求用户输入chess或者go来选择要进行的游戏，然后再要求用户输入两个玩家的名字，然后根据选择的游戏类型创建游戏，然后调用game方法进入游戏，开始游戏。如下图为进入chess游戏。



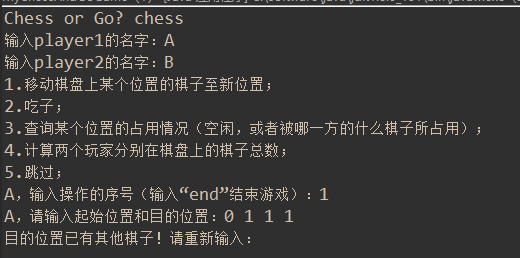
进入游戏后，打印出游戏菜单，用户输入所要做的操作的序号，如果用户输入错误，则要求用户重新输入，如下图。



此部分操作对应Game.game方法中下面的代码。当输入的操作不是end以及1-5的任意一个操作均抛出异常，然后提示用户输入正确的操作。



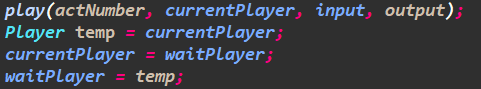
如果输入的操作，就会进入Chess.play，判断进入哪一个操作，接着会有提示信息，要求用户输入对应的信息，当用户输入的信息有误时输出相应的提示并要求用户重新输入，如下图。图为移动棋子的操作。

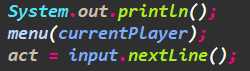


这部分对应Chess.play的如下部分代码，输入的位置会在action.movePiece中判断，如果返回的是false，说明输入错误，在返回之前会先打印出提示。然后进入else分支，提示用户重新输入，继续循环，知道action.movePiece返回true，也就是说用户的输入被执行了，就退出循环，结束本棋手的本轮操作。

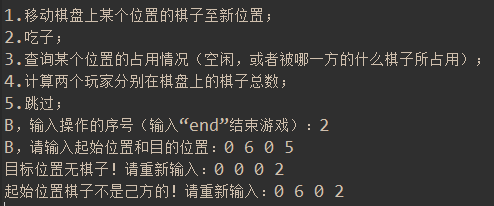


程序返回到game方法中继续执行，play结束后要先交换行动权，把A设为waitPlayer，把B设为currentPlayer。然后为B打印菜单，读取B的操作输入。





运行过程如下图，图为B吃掉A的棋子操作。



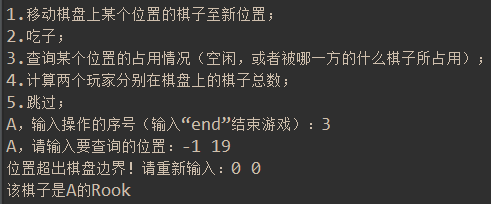
此时进入的是Chess.play的第二个操作的部分，这部分的执行逻辑与第一个操作没有区别。代码如图。



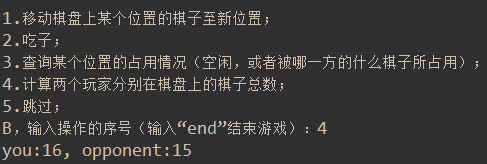
然后再次交替行动权，如此往复，直到有一方输入了end，就结束游戏。

游戏运行过程截图如下：

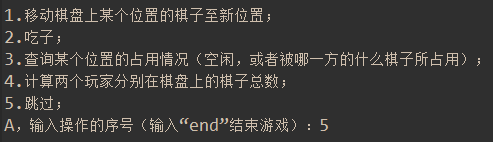
查询某位置的情况：



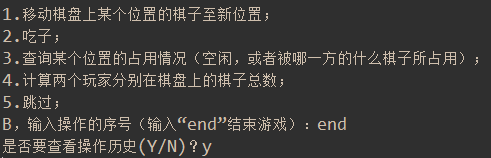
计算双方棋子在棋盘上的总数（之前的操作中A被B吃了一个棋子）：



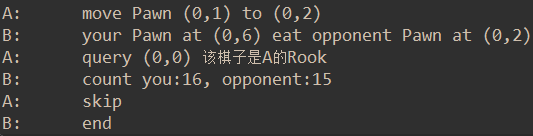
跳过操作：



输入end结束游戏：



最后查看历史：



### ADT和主程序的测试方案

#### 测试Action

这部分涉及到对棋盘的各种操作，是重点测试的对象，测试策略如下：

putPiece方法：

按位置是否为棋盘上的合法位置分：是棋盘上的位置，不是棋盘上的位置

按该位置是否空闲分：该位置空闲，该位置不空闲

按棋子分：棋子不是己方的，棋子是己方的

takePiece方法：

按位置是否为棋盘上的合法位置分：是棋盘上的位置，不是棋盘上的位置

按该位置是否空闲分：该位置空闲，该位置不空闲

所提棋子分：所提棋子是己方棋子，所提棋子不是己方棋子

movePiece方法：

按位置是否为棋盘上的合法位置分：是棋盘上的位置，不是棋盘上的位置

按目的位置是否空闲分：该位置空闲，该位置不空闲

按起始位置是否空闲分：该位置空闲，该位置不空闲

按移动的棋子分：棋子不是己方的，棋子是己方的

eatPiece方法：

按位置是否为棋盘上的合法位置分：是棋盘上的位置，不是棋盘上的位置

按两位置是否为同一位置分：是同一位置，不是同一位置

按目的位置是否空闲分：该位置空闲，该位置不空闲

按起始位置是否空闲分：该位置空闲，该位置不空闲

按移动的棋子分：棋子不是己方的，棋子是己方的

按被吃的棋子分：棋子是己方的，棋子不是己方的

query方法:

按查询的类型分：chess，go

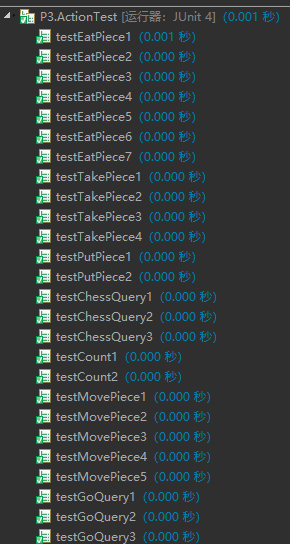
按位置是否为棋盘上的合法位置分：是棋盘上的位置，不是棋盘上的位置

按该位置是否空闲分：该位置空闲，该位置不空闲

count方法：

棋子数目为0，棋子数目不为0

测试结果：



#### 测试Position

由于Position是immutable的，所以只需要测试getX，getY方法能够正确返回坐标即可。

测试结果：



#### 测试Piece

测试策略：

getType方法：type为空(围棋)，type不为空(象棋)

getPlayer方法：测试返回的棋手是否正确

测试结果：



#### 测试Board

测试策略：

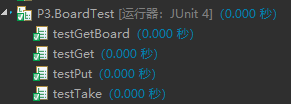
put方法：放在合法的空闲位置

take方法：该合法位置有棋子

getBoard方法：空棋盘，非空棋盘

get方法：按棋子分：有棋子，无棋子

测试结果：



#### 测试Player

测试getName方法能够正确返回棋手姓名。

测试结果：



#### 测试Go

测试策略：

game方法:

按输入类型：数字，字符串

按数字分：1-5，小于1或大于5

按字符串分：end，非end

按是否查看历史分：查看，不查看

menu方法：测试是否能打印出菜单

play方法:

按动作代号分：actNumber=1，2，3，4，5

按动作的位置数据是否正确分：正确，错误

history方法: 测试能够正确读取文件

测试结果：



#### 测试Chess

测试策略：

game方法:

按输入类型：数字，字符串

按数字分：1-5，小于1或大于5

按字符串分：end，非end

按是否查看历史分：查看，不查看

menu方法：测试是否能打印出菜单

play方法:

按动作代号分：actNumber=1，2，3，4，5

按动作的位置数据是否正确分：正确，错误

history方法: 测试能够正确读取文件

测试结果：



# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2020.3.17 | 20:30-23:00 | 完成编写P1的problem 1,2代码 | 按时完成 |
| 2020.3.18 | 18:30-23:00 | 完成编写P1的problem 3,4代码 | 按时完成 |
| 2020.3.19 | 13:00-14:30 | 完成编写P2的代码 | 延时10分钟完成 |
| 2020.3.23 | 15:00-16:30 | 构建P3部分的实现思路 | 按时完成 |
| 2020.3.23 | 20:00-23:00 | 完成编写P3的代码框架 | 延时半小时完成 |
| 2020.3.24 | 19:30-21:30 | 完成编写P3的除Chess和Go的部分 | 按时完成 |
| 2020.3.26 | 18:30-21:00 | 完成编写P3的Chess和Go的部分 | 按时完成 |
| 2020.3.27 | 10:00-12:00 | 完成P1的注释的编写 | 按时完成 |
| 2020.3.27 | 14:00-15:30 | 完成P1的problem1,2的报告 | 未完成 |
| 2020.3.27 | 20:00-23:00 | 完成P1.graph的报告的撰写 | 按时完成 |
| 2020.3.28 | 9:20-10:00 | 完成P1.poet的报告的撰写 | 按时完成 |
| 2020.3.28 | 11:00-12:00 | 完成P2的报告的撰写 | 按时完成 |
| 2020.3.29 | 13:00-17:00 | 完成编写P3的测试 | 按时完成 |
| 2020.3.30 | 15:00-17:00 | 完成报告 | 按时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| Scanner类使用System.in做参数创建的对象在调用一次close()方法之后其他的使用System.in做参数创建的对象输入都无效： java.util.NoSuchElementException | 只定义一次Scanner对象，然后把Scanner作为参数传递，在使用结束后一次关闭。 |
| 测试的时候发现屏幕输出System.out.println()与“\n“并不相等。 | 查阅资料后得知，println()的行为与操作系统相关，Windows下对应于“\r\n”。 |

# 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

在我们自己对一个问题进行完全自主的ADT设计时，要尽可能考虑周全，不要急于开始编写，要先设计好整个ADT的结构。如果一开始没有做好结构和方法设计，那可能在过程中会出现不好解决的问题以至需要全部推倒。

## 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？
2. 使用泛型和不使用泛型的编程，对你来说有何差异？
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？
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？
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？
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？
7. 关于本实验的工作量、难度、deadline。
8. 《软件构造》课程进展到目前，你对该课程有何体会和建议？
9. 面向ADT的编程我们需要考虑更多边界情况，需要将具体的问题抽象为更加宽泛具有覆盖性的函数，直接面向应用场景编程则不需要考虑那么多，直接为了解决特定的问题进行编程即可。
10. 使用泛型时不能使用特定的方法，但泛型可以更好的做到代码的复用，可以极大地避免多种类似对象的相同的操作的多次重写，同时，使用泛型编程需要考虑更多的实现场景。
11. 可以严格按照spec进行测试的编写，不需要考虑代码的实现，目前还不是很习惯这种编程方式。
12. 节省了很大的代码量，提高了开发的效率。
13. 已经适应。给定各ADT之间的关系后，只需要按照spec实现各个方法即可，而P3要求自主设计，就需要充分地考虑各个ADT之间的关系，做好各个ADT之间功能的分割，对各部分的逻辑关系必须理清。
14. 可以避免由于客户端的代码中对变量内容的修改造成的潜在的隐患。愿意。
15. 本实验工作量适中，难度适中，deadline很合适。
16. 课程内容需要亲自实践才会有深刻的印象。