

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 魏孝文 |
| 学号 | 1170500704 |
| 班号 | 软件一班 |
| 电子邮件 | 706461857@qq.com |
| 手机号码 | 18145640839 |

**目录**

[1 实验目标概述 1](#_Toc1988214)

[2 实验环境配置 1](#_Toc1988215)

[3 实验过程 1](#_Toc1988216)

[3.1 Poetic Walks 1](#_Toc1988217)

[3.1.1 Get the code and prepare Git repository 1](#_Toc1988218)

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

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc1988220)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc1988221)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc1988222)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc1988223)

[3.1.4.1 Make the implementations generic 2](#_Toc1988224)

[3.1.4.2 Implement Graph.empty() 2](#_Toc1988225)

[3.1.5 Problem 4: Poetic walks 2](#_Toc1988226)

[3.1.5.1 Test GraphPoet 2](#_Toc1988227)

[3.1.5.2 Implement GraphPoet 2](#_Toc1988228)

[3.1.5.3 Graph poetry slam 2](#_Toc1988229)

[3.1.6 Before you’re done 2](#_Toc1988230)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc1988231)

[3.2.1 FriendshipGraph类 2](#_Toc1988232)

[3.2.2 Person类 3](#_Toc1988233)

[3.2.3 客户端main() 3](#_Toc1988234)

[3.2.4 测试用例 3](#_Toc1988235)

[3.2.5 提交至Git仓库 3](#_Toc1988236)

[3.3 Playing Chess 3](#_Toc1988237)

[3.3.1 ADT设计/实现方案 3](#_Toc1988238)

[3.3.2 主程序ChessGame设计/实现方案 3](#_Toc1988239)

[3.3.3 ADT和主程序的测试方案 3](#_Toc1988240)

[3.4 Multi-Startup Set (MIT) 4](#_Toc1988241)

[4 实验进度记录 4](#_Toc1988242)

[5 实验过程中遇到的困难与解决途径 4](#_Toc1988243)

[6 实验过程中收获的经验、教训、感想 4](#_Toc1988244)

[6.1 实验过程中收获的经验和教训 4](#_Toc1988245)

[6.2 针对以下方面的感受 4](#_Toc1988246)

# 实验目标概述

本次实验训练抽象数据类型(ADT)的设计、规约、测试,并使用面向对象

编程(OOP)技术实现 ADT。具体来说:

①针对给定的应用问题,从问题描述中识别所需的 ADT;

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

③根据 ADT 的规约设计测试用例;

④ADT 的泛型化;

⑤根据规约设计 ADT 的多种不同的实现;针对每种实现,设计其表示

(representation)、表示不变性(rep invariant)、抽象过程(abstraction function)

⑥使用 OOP 实现 ADT,并判定表示不变性是否违反、各实现是否存在表示外泄(rep exposure);

⑦测试 ADT 的实现并评估测试的覆盖度;

⑧使用 ADT 及其实现,为应用问题开发程序;

⑨在测试代码中,能够写出 testing strategy 并据此设计测试用例。

# 实验环境配置

安装EclEmma

Lab2网址：https://github.com/ComputerScienceHIT/Lab2-1170500704

# 实验过程

请仔细对照实验手册，针对三个问题中的每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## Poetic Walks

实现图分别实现基于点的实现和基于边的实现，在性能上有区别，但是client使用起来没有区别。

### Get the code and prepare Git repository

如何从GitHub获取该任务的代码、在本地创建git仓库、使用git管理本地开发。

git clone <https://github.com/rainywang/Spring2019_HITCS_SC_Lab2.git>

mkdir Lab2 #创建本地项目文件夹

cd Lab2

git init #创建本地git仓库

提交代码命令：git add .

Git commit –m ‘[commit message]’

将提交同步到远程仓库：

git push origin master

### Problem 1: Test Graph <String>

以下各部分，请按照MIT页面上相应部分的要求，逐项列出你的设计和实现思路/过程/结果。

### Problem 2: Implement Graph <String>

分别给每个函数写相应的测试函数，这些测试函数放在了GraphInstanceTest.java文件中

测试的策略：

等价类划分：给图中添加的点，未存在，已存在

给图中添加的边的权值：>0, = 0,<0

然后在给个测试函数中将这些等价类进行笛卡尔乘积，得到相应的测试用例。

例如：在测add（）函数时，分别添加已存在的点和未存在的点，观察返回的值

在测set()函数时，分别添加两个已存在的点，边的权重>0,两个未存在的点，边的权重>0,一个已存在的边，但是边的权重<=0,分别观察函数返回值的情况。

#### Implement ConcreteEdgesGraph

boolean add(L vertex)

首先判断点集合中是否存在这个点，如果存在，就返回false,如果不存在，就向点集合中添加这个点，同时返回true.

int set(L source, L target, int weight)

首先遍历边的集合，看是否有与传入边相同的边，如果有，就先删除边，如果传入的weight>0,就将新的边加入集合，如果weight==0,就不做额外的处理。

如果边的集合中没有相同的边，就将新的边加入边的集合。

boolean remove(L vertex)

首先判断点的集合中是否有这个点，如果没有就返回false,.如果有就将这个点从点的集合中删除，然后遍历边的集合，将边的顶点中包含这个顶点的边删除。

Set<L> vertices()

新实例化一个点的集合，将原来点的集合clone到新的点集合，将新的点集合返回，这样可以避免数据的溢出

Map<L, Integer> sources(L target)

遍历边的集合，然后将边中包含target的边加入map中，返回

Map<L, Integer> targets(L source)

遍历边的集合，然后将边中包含source顶点的边加入map中，返回

#### Implement ConcreteVerticesGraph

boolean add(L vertex)

首先在点的集合中寻找是否这个点已经存在，如果存在，就返回false,如果不存在，就将这个点加入点的集合。

int set(L source, L target, int weight)

首先判断如果weight>0，就先将这两个点加入点的集合，然后分别寻找到source,target这两个点，将source点的出边添加或删除指定边，将target的入边添加或删除指定边。

boolean remove(L vertex)

首先判断点的集合中是否有这个点，如果没有，就返回false,如果有，就将这个点删除，然后将这个点的入边和出边都删除。

Set<L> vertices()

将图的点集合克隆后，将克隆后的对象返回。

Map<L, Integer> sources(L target)

将这个点的所有入边集合克隆后，将克隆后的对象返回

Map<L, Integer> targets(L source)

将这个点的所有出边集合克隆后，将克隆后的对象返回

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

#### Make the implementations generic

将原来的String 都修改成 泛型<L>

#### Implement Graph.empty()

返回一个新的实例化对象ConcreteVerticesGraph,因为这个对象的效率更高。

### Problem 4: Poetic walks

#### Test GraphPoet

**通过文件读入了几句话，然后输入几句话，比对返回的句子与人工计算出的句子是否相同。**

**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（需要筛选）**

#### Implement GraphPoet

GraphPoet(File corpus)

这个函数用来读入文件，然后根据文件内容读入的句子建立图，图的点是文件中的单词，图的边权重是两个单词前后相接情况出现的次数。

String getBridge(String source, String target)

这个函数输入两个相邻的单词，然后遍历图，寻找权重最大的bridge-word，并返回，如果没有符合条件的单词，就返回””. 这个函数中我使用了广度优先搜索，只需搜索两层的点即可。

String poem(String input)

这个函数拆解输入的input，然后分别调用getBridge函数，将返回的单词加在相邻两个单词之间即可。

#### Graph poetry slam

在这一步中，我直接运行了main函数，然后可以输出正确的结果，运行正常。

### Before you’re done

请按照[http://web.mit.edu/6.031/www/sp17/psets/ps2/#before\_youre\_done](http://web.mit.edu/6.031/www/sp17/psets/ps2/" \l "before_youre_done)的说明，检查你的程序。

如何通过Git提交当前版本到GitHub上你的Lab2仓库。

在这里给出你的项目的目录结构树状示意图。

Lab2

Src

P1

Graph

ConcreteEdgesGraph.java

ConcrteteVerticesGraph.java

Graph.java

Poet

GraphPoet.java

Main.java

mugar-omni-theater.txt

Test

P1

Graph

ConcreteEdgesGraph.java

ConcteteVerticesGraph.java

GraphInstanceTest.java

GraphStaticTest.java

Poet

Data.txt

GraphPoetTest.java

## Re-implement the Social Network in Lab1

这个任务让我们体会抽象数据类型的重要作用，只需要构造一次，就可以在其他程序中多次复用，大大提高编程的效率。

在这里简要概述你对该任务的理解。

### FriendshipGraph类

这个类中只需要一个P1中的graph对象，然后addVertex()，addEdge()函数都是调用graph中的add(),set()函数即可轻松完成。

getDistance()函数依然使用广度优先搜索，大部分程序都不需要修改，只需要将搜索某个点出边的代码替换成 Graph类中的targets()函数，即可轻松实现。

给出你的设计和实现思路/过程/结果。

### Person类

这个类只有一个字符串用来存储人的名字，然后有一个obseravor的函数，用来得到用户的名字。

给出你的设计和实现思路/过程/结果。

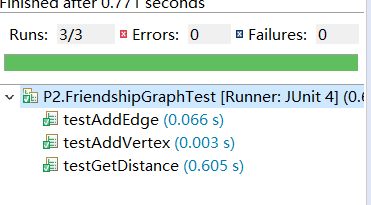
### 客户端main()

重新利用Lab1中已有的main客户端即可，由于所有Lab2中的所有实现都是没有改变addVertex、addEdge和getDistance三个函数，只需要重新使用即可使用。

给出你的设计和实现思路/过程/结果。

### 测试用例

测试策略主要是根据FriendshipGraph中的图的类型进行测试，主要分为空图的测试、仅有顶点的图的测试、无环图的测试和有环图的测试。除了以上的新增加的测试用例，还有在Lab1中使用的测试用例，来验证在不改变客户端仍然可以使用。测试结果如下图：



给出你的设计和实现思路/过程/结果。

### 提交至Git仓库

Lab2

Src

P2

FriendShipGraph.java

Person.java

Test

P2

FriendShipGraphTest.java

Edge.txt

如何通过Git提交当前版本到GitHub上你的Lab3仓库。

在这里给出你的项目的目录结构树状示意图。

## Playing Chess

### ADT设计/实现方案

设计了一个game接口和两个从game接口衍生的类，名为Go和Chess，还有一个命令行控制的主程序MyChessAndGoGame。

围棋程序GO：

Rep：

private int size,count1=0,count2=0;//size：棋盘大小，两个棋手在棋盘上的棋子数，mutable

private int board[][];//棋盘,mutable

private String P1,P2;//P1：棋手1，P2：棋手2 ,immutable

List<String> list1=new LinkedList<String>(),list2=new LinkedList<String>();//list1:棋手1的下棋历史，list2：棋手2的下棋历史,mutable

//Rep invariant:

//P1,P2 are player's names(a nonempty string of letters)

//The coordinates of the pieces are between 0 and 18

//Abstraction Function:

//setPlayer,put pieces,inquire,move pieces,remove pieces,

//safety from rep exposure:

//all fields are private;

//P1,P2,count1,count2 are immutable

//list1,list2 are mutable,so getList1(),getList2()make defensive copies to avoid sharing the rep's history with client

象棋程序chess：

**private** **int** size,count1=16,count2=16;//两个棋手在棋盘上的棋子数，mutable

**private** **final int** symbol[]= {1,2,3,4,5,6};//不同棋子的表示，1：国王king，2：皇后queen，3：象bishop，4：马knight，5：车rook，6：兵pawn immutable

**private** **int** board[][]= {{5,4,3,1,2,3,4,5},{6,6,6,6,6,6,6,6},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},{0,0,0,0,0,0,0,0},

{-6,-6,-6,-6,-6,-6,-6,-6},{-5,-4,-3,-1,-2,-3,-4,-5}};

//棋盘，开始摆好的，mutable

**private** String P1,P2;//P1：棋手1，P2：棋手2

List<String> list1=**new** LinkedList<String>(),list2=**new** LinkedList<String>();//走步历史,用string表示

**private** final **int** number[]= {1,1,2,2,2,8};//各个棋子的数量，immutable

**private** String pieces[]= {"king","queen","bishop","knight","rook","pawn"};//各个棋子的名称

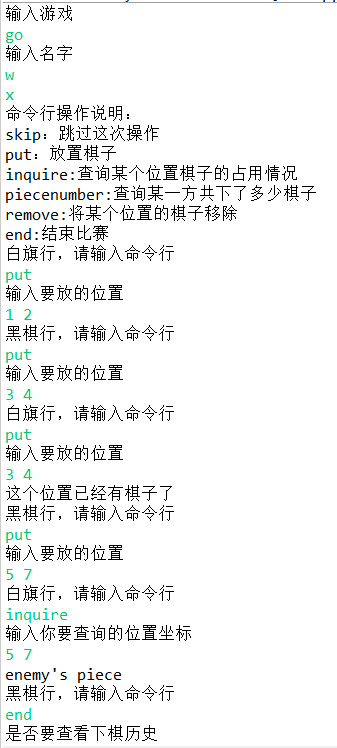
**private** List<Integer> white=**new** ArrayList<Integer>(Arrays.*asList*(1,1,2,2,2,8)),black=**new** ArrayList<Integer>(Arrays.*asList*(1,1,2,2,2,8));

//统计在棋盘上的棋子数量

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

必要时请使用UML class diagram（请自学）描述你设计的各ADT间的关系。

### 主程序MyChessAndGoGame设计/实现方案



主程序由三个函数组成：main函数，playGo函数，PlayChess函数

Main函数先读取用户需要玩哪个游戏，如果是围棋游戏的话先new一个GO对象，根据键盘上的输入对GO对象进行操作,比方说，如果用户输入的命令行是put，提示用户输入要下棋的坐标，调用GO的put方法，对GO对象的棋盘进行修改。

如果是Chess游戏，先new一个Chess对象，根据键盘上的输入对Chess对象进行操作。比方说，如果用户输入的命令行是eat，提示用户输入起始点和要吃掉的点的坐标，调用GO的remove方法，对Chess对象的棋盘进行修改。

怎么对ADT具体方法进行操作：

Game接口的方法：

1. setPlayer(String s1,String s2)
2. putPiece(String Player,String piece,int x,int y);
3. movePiece(String Player,int sourcex,int sourcey,int destx,int desty);
4. removePiece(String Player,int x,int y);
5. Inquire(String Player,int x,int y);

6） eatPiece(String Player,int sourcex,int sourcey,int destx,int desty);

GO类型的具体实现：

1. 棋盘用二维数组表示board[][]，用0表示棋盘上没有放置棋子的位置，用1表示棋盘上的白色棋子，用2表示棋盘上的黑色棋子，两个棋手分别用类中的变量P1和P2表示。
2. setPlayer(String s1,String s2)

让P1=s1,P2=s2。

1. putPiece(String Player,String piece,int x,int y);

检查这个棋子是不是这个棋手的，如果是而且这个位置没有棋子，将棋手棋子放置在这，返回true，否则输出错误信息返回false；

1. Inquire(String Player,int x,int y);

查询这个位置是不是这个棋手的棋子，如果没有棋子返回“没有棋子”，如果是这个棋手的返回“是这个棋手的棋子”，否则返回“敌方棋子”

1. removePiece(String Player,int x,int y);

如果这个位置是敌方棋子，移除，返回true，否则返回false；

.

Chess类型的具体实现：

1. 棋盘用二维数组表示board[][],用负数表示P1的白色棋子，用正数表示P2的黑色棋子, 1：国王king，2：皇后queen，3：象bishop，4：马knight，5：车rook，6：兵pawn。

将棋盘按照象棋摆放位置初始化。

1. setPlayer(String s1,String s2)

让P1=s1,P2=s2。

1. putPiece(String Player,String piece,int x,int y);

检查这个棋子是不是这个棋手的，如果是而且这个位置没有棋子，将棋手棋子放置在这，返回true，否则输出错误信息返回false；

1. movePiece(String Player,int sourcex,int sourcey,int destx,int desty);

如果起始位置的棋子是这个棋手的而且目标位置没有棋子的话，将起始位置的棋子移到目标位置上，返回true，否则返回false；

1. Inquire(String Player,int x,int y);

查询这个位置是不是这个棋手的棋子，如果没有棋子返回“没有棋子”，如果是这个棋手的返回“是这个棋手的棋子”，否则返回“敌方棋子”

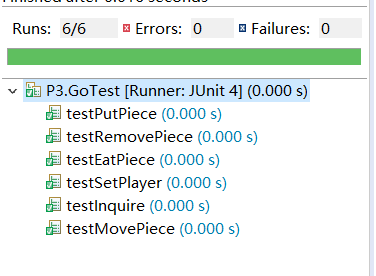
1. eatPiece(String Player,int sourcex,int sourcey,int destx,int desty);

如果起始位置的棋子是这个棋手的而且目标位置的棋子是敌人的，将目标位置的棋子替换为起始位置的棋子，起始位置的棋子清零，返回true，否则返回false。

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

### ADT和主程序的测试方案

对围棋的测试：



1.testSetPlayer()：

设置两个名字，对其进行检测。

2.testPutPiece()

放置棋子检测：

（1）.进行越界检测

（2）如果目标位置为空，那么放置为真，进行检测

（3）如果目标位置不为空，进行放置失败检测。

3. testInquire()

（1）如果目标位置没有棋子，进行检测。

（2）如果目标位置是自己的棋子，进行检测。

（3）如果目标位置是对手的棋子，进行检测。

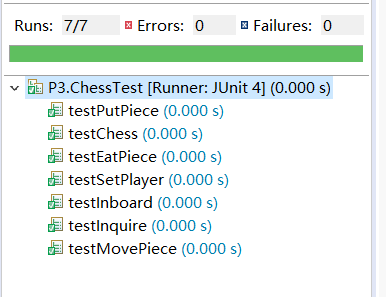
4. testRemovePiece()

（1）进行越界检测

（2）如果这个位置是自己的棋子或者没有棋子，进行检测。

（3）如果这个位置是敌方棋子，进行检测。

对象棋的检测：



1. testSetPlayer()：

设置两个名字，对其进行检测

2.testPutPiece()

如果这个棋子已经在棋盘上了，进行失败检测

如果这个目标位置有棋子，进行失败检测

否则进行成功检测

3.testInquire()

（1）如果目标位置没有棋子，进行检测。

（2）如果目标位置是自己的棋子，进行检测。

（3）如果目标位置是对手的棋子，进行检测。

4.testEatPiece()

如果开始位置是己方棋子，目标位置是对方棋子，进行成功检测

如果目标位置没有棋子或是自己棋子，进行失败检测

如果起始位置没有棋子或是敌方棋子，进行成功检测。

介绍针对各ADT的各方法的测试方案和testing strategy。

介绍你如何对该应用进行测试用例的设计，以及具体的测试过程。

## Multi-Startup Set (MIT)

请自行设计目录结构。

注意：该任务为选做，不评判，不计分。

# 实验进度记录

请使用表格方式记录你的进度情况，以超过半小时的连续编程时间为一行。

每次结束编程时，请向该表格中增加一行。不要事后胡乱填写。

不要嫌烦，该表格可帮助你汇总你在每个任务上付出的时间和精力，发现自己不擅长的任务，后续有意识的弥补。

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 3.21 | 18：00-21：00 | String类型的的编程 | 完成 |
| 3.23 |  | ConcreteEdgesGraph | 完成 |
| 3.24 |  | ConcreteVeritices | 延期 |
| 3.26 |  | 编写P1test | 延期 |
| 3.27-3.28 |  | P2 | 完成 |
| 3.29-3.31 |  | P3 | 完成 |
| 3.32 |  | 写实验报告 | 延期 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 泛型编程 | 在Csdn学习 |
| 关于接口和ADT的使用 | 在网上和中国大学mooc学习 |
| 实验三的设计 | 多看P1,P2设计找思路 |

# 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

## 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？

编写ADT需要考虑ADT的可复用性，应用场景则考虑代码的实际操作性能。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

应用泛型可以提高代码的可复用型，对client更友好，同时提高了编程难度。

1. 在给出ADT的规约后就开始编写测试用例，优势是什么？你是否能够适应这种测试方式？

进行黑盒测试，更早发现程序的问题，可以

1. P1设计的ADT在多个应用场景下使用，这种复用带来什么好处？

避免重复造轮子，可以将已有的代码进行复用来尽量节省时间，而且对于问题的抽象能力的培养有了更好的锻炼。编程的时候不再只注意某一个应用场景，而是从很多相似的应用场景中发现共性和已经编写好的代码之间的相似之处

1. P3要求你从0开始设计ADT并使用它们完成一个具体应用，你是否已适应从具体应用场景到ADT的“抽象映射”？相比起P1给出了ADT非常明确的rep和方法、ADT之间的逻辑关系，P3要求你自主设计这些内容，你的感受如何？

P3我是先设计一个game接口再衍生出两类棋类游戏，感觉抽象映射还是很有难度，对于自己编写设计ADT有很大的帮助

1. 为ADT撰写specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后编程中坚持这么做？

为了保证程序的安全形，避免rep被client知道并被他人攻击篡改。

1. 关于本实验的工作量、难度、deadline。

难度适中，deadline给的时间较为充足

1. 《软件构造》课程进展到目前，你对该课程有何体会和建议？

帮助我将编写代码的认识变为基于项目和工程，对具体的情景抽象出各种情景的相似处并编写ADT,提高代码的复用性，并且时刻注意代码的安全型。建议：最近学习感觉知识量很大又很碎片话，建议可以多给我们推荐一些读物。