

**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 | 卢兑玧 |
| 学号 | L170300901 |
| 班号 | 1703009 |
| 电子邮件 | nty0725@gmail.com |
| 手机号码 | 15663729008 |

**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：

* 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

访问GitHub Classroom 中的 URL，按照提示建立 Lab2 仓库并关联至学号。

访问Eclipse Marketplace下载EclEmma插件，发现已配置好，无需下载。

GitHub Lab2仓库的URL地址：<https://github.com/ComputerScienceHIT/Lab2-L170300901>

# 实验过程

## Poetic Walks

这个任务的需求1-3是让我们逐步实现两种有向图的数据结构在此基础上实现范型，完成测试。需求4则是实际应用，根据我们实现的图来生成诗。

### Get the code and prepare Git repository

$ git clone <https://github.com/ComputerScienceHIT/Lab2-L170300901>

$ cd /Users/ucla/Desktop/Lab2-L170300901

$ git init

$ git add .

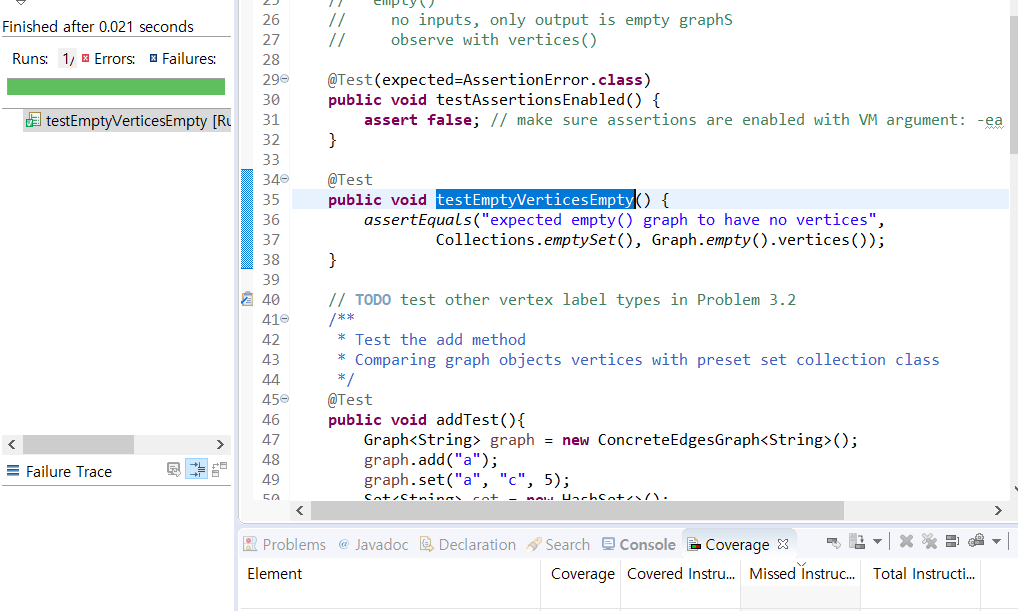
$ git commit –m “origin”

$ git remote add origin https://github.com/ComputerScienceHIT/Lab2-L170300901.git

$ git push -u origin master

### Problem 1: Test Graph <String>

实现GraphStaticTest.java：仅需实现testEmptyVerticesEmpty方法对Graph.empty().vertices()进行测试。可使用Collections.emptySet()新建新的空集合后与Graph.empty().vertices()比较检查是否为空；实现GraphInstanceTest.java：新建一张有向图仅需调用emptyInstance方法即可。测试结果如图所示：



### Problem 2: Implement Graph <String>

Graph*中抽象类的要求：*

* *set ：在有向图中新增一条边*

若已有边则返回过去的边的权值，若新增的边权值为0代表移除这条边。

* add ：在有向图中新增一个点。

若已有点则添加失败，返回false，否则返回true。

* remove ：从有向图中移除一个点

若集合中美有这个点则移除失败，返回true，否则返回true。

* vertices ：返回包含所有点的集合

若此图为空图，返回空的集合

* sources ：给定顶点，返回所有有指向这个点的起点

若找不到该顶点／符合条件的点，返回空的集合

* targets ：给定顶点，返回所有以这个点为起点的终点

若找不到该顶点／符合条件的点，返回空的集合

#### Implement ConcreteEdgesGraph

在这个类中有两个rep：

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

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

- vertices ：存储点的集合

- edges ：存储边的对象数组

自定义的类Edge需要满足以下条件

* 存储入边（String）
* 存储出边（String）
* 存储权值（int）
* private final L source;
* private final L target;
* private final int weight;

为了维护内部成员变量的封闭性，生成getter以获得成员变量的信息。

在实现这个问题时通过借助rep与自定义的Edge类，Graph中的抽象函数可以具体实现为：

* set ：在有向图中新增一条边—>向edges中新增一个Edge对象
* add ：在有向图中新增一个点—>向vertices中新增一个String对象
* remove ：从有向图中移除一个点—>从vertices中移除一个String对象
* vertices ：返回包含所有点的集合—>返回vertices集合
* sources ：给定顶点，返回所有有指向这个点的起点—>遍历edges，将所有以此点为终点的起点加入map后返回
* targets ：给定顶点，返回所有以这个点为起点的终点—>遍历edges，将所有以此点为起点的终点加入map后返回

#### Implement ConcreteVerticesGraph

在这个类中仅有一个rep来储存图中的点。

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

由于没有存储边的数据结构所以自定义的类Vertex需要满足以下条件

* 存储点的标签（String）
* 存储可到达的点的标签（String）
* 储存权值（int）

String label;

private Map<String, Integer> map = new HashMap<>();

为了维护内部成员变量的封闭性，生成getter以获得成员变量的信息。

在实现这个问题时通过借助rep与自定义的Edge类，Graph中的抽象函数可以具体实现为：

* set ：在有向图中新增一条边—>向edges中新增一个Edge对象
* add ：在有向图中新增一个点—>向vertices中新增一个String对象
* remove ：从有向图中移除一个点—>从vertices中移除一个String对象
* vertices ：返回包含所有点的集合—>返回vertices集合
* sources ：给定顶点，返回所有有指向这个点的起点—>遍历edges，将所有以此点为终点的起点加入map后返回
* targets ：给定顶点，返回所有以这个点为起点的终点—>遍历edges，将所有以此点为起点的终点加入map后返回

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

#### Make the implementations generic

由于Vertex和Edge中的label可能是字符串、整数、浮点数或其他类型，所以使用generic可以简化程序量，实现算法复用。<L>表示L可能是各种类型（String、Integer、Double……），实现范型很简单，只需用<L>代替<String>标签即可，所以类的声明变成如下形式：

public class ConcreteEdgesGraph<L> implements Graph<L>{ ... }

class Edge<L>{ ... }

类中的rep Edge、 List<Edge>也随之替换为 Edge<L>、List<Edge<L>。然后将代码中的String类型替换为L即可。

#### Implement Graph.empty()

这个需求非常简单，仅需在上面实现的两种图的基础上实现一个Graph的类函数empty生成一张空图即可。在这里选择ConcreteEdgesGraph。

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

return new ConcreteEdgesGraph<>();

}

### Problem 4: Poetic walks

#### Test GraphPoet

在这里需要编写对GraphPoet的测试用例。通过反复阅读文档才理解这个PoetWalk的需求（英语捉急）：

* 将语料库建立为一张有向图
* 输入一个语句，尝试在每两个单词A、B中间添加单词。这个单词应该是在有向图中A、B的权最大的bridge。
* 输出这条语句。

总结：编写测试用例时应覆盖以下特殊情况：

* 包含大写字母的corpus是否被处理为小写字母
* 输入的词汇包含大写字母后会被如何处理
* 输入的词汇带有标点符号会被如何处理
* 输入的corpus中有多个weight值时将选取哪个target
* 文件为空

#### Implement GraphPoet

这个问题的需求我们将其分为两部分：

* 将语料库建立为一张有向图
* 输入一个语句，尝试在每两个单词A、B中间添加单词。这个单词应该是在有向图中A、B的权最大的bridge，并输出这条语句。

首先应实现将语料库建立为一张有向图的需求。这个问题与Lab1-P1非常相似，应读取文件中的全部文本，然后将其分割为多个String对象组成一个List并返回即可。MIT给了以下提示：

* FileReader是从文件读取信息的标准类，在BufferedReader中包裹FileReader类使我们能够方便地使用readLine方法读取一行。
* Files.readAllLines是实用的函数，它将从路径中读取所有的行
* Scanner类被设计的功能为分块地读取文件，它提供很多特性，应谨慎的使用。

这里我们使用Scanner进行读取。先使用FileReader读取文件并用BufferedReader包裹，然后用scanner按单词读入，使用scanner的hasNext进行迭代，在每一次迭代中都讲单词加入List中，直至完成所有读取。

#### Graph poetry slam

此处相当于自由发挥，更新main函数，用自己的例子和自己的corpus来做poetic walk。然而没能找到cool的input，把自己的corpus放在与mugar-omni-theater.txt相同的文件下后运行得到以下输出：

Find something you love and explore opportunity in your life

>>>

Find something you really love and explore opportunity in your fantastic life

### Before you’re done

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)的说明检查程序。通过Git提交当前版本到GitHub上的Lab2仓库：

$ cd /Users/ucla/Desktop/Lab2-L170300901

$ gitadd .

$ git commit –m “finish P1”

$ git push

项目的目录结构树状示意图：

src

|——P1

|——graph

|——ConcreteEdgeGraph<L>.java

|——ConcreteVerticesGraph.java

|——Graph.java

|——poet

|——GraphPoet.java

|——Main.java

|——mugar-omni-theather

test

|——P1

|——graph

|——ConcreteEdgeGraphTest.java

|——ConcreteVerticesGraphTest.java

|——GraphInstanceTest.java

|——GraphStaticTest.java

|——poet

|——GraphPoetTest.java

## Re-implement the Social Network in Lab1

基于在 3.1 节 Poetic Walks 中定义的 Graph<L>及其两种实现，重新实现 Lab1中3.3节的FriendshipGraph类。

### FriendshipGraph类

在这里我们选择3.1中实现的ConcreteEdgesGraph<L>来重新实现FriendshipGraph类。其中L为Person类。首先使FriendshipGraph继承ConcreteEdgesGraph<L>类：

public class FriendshipGraph extends ConcreteEdgesGraph<Person> {..}

FriendshipGraph中应提供addVertex()、addEdge()和getDistance()三个方法：针对addVertex()和addEdge()需要尽可能复用 ConcreteEdgesGraph<L>中已经实现的 add()和set()方法，核心源代码如下所示：

private void addEdge(Person p1, Person p2) {

this.set(p1, p2, 1);

p1.addFriend(p2);

}

private void addVertex(Person p) { add(p); }

针对getDistance()方法，基于你所选定的 ConcreteEdgesGraph<L> 中的rep ： Set<L> vertices、List<Edge<L>> edges来实现。其中由于点的集合是Set，某些操作不太方便，所以将其转化为List。

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

List<Person> persons = new ArrayList<>(vertices());

其中bfs的思路与之前一样：首先将图使用二维数组储存，然后借用队列对图进行广度优先搜索得到距离。

储存和表示图需要新建空的二维数组，再遍历点的集合，获得每个点的targets（即认识的朋友），在图中将此点与targets连接即可。

List<Person> persons = new ArrayList<>(vertices());

for (Person person : persons) {// 遍历每个人

int i = persons.indexOf(person);// 每个人在list中的编号对应坐标

Map<Person, Integer> friends = targets(person);

for (Map.Entry<Person, Integer> entry : friends.entrySet()) {

int j = persons.indexOf(entry.getKey());

try {

graph[i][j] = 1;// 若i认识j 则graph[i][j]为1

} catch (ArrayIndexOutOfBoundsException e) {

System.err.println("捕捉到ArrayIndexOutOfBoundsException异常，是否未调用addVertex()将其加入图中？");

}

}

}

得到二维矩阵后借助Queue进行广度优先搜索：从一个点开始将其所有邻接且未访问过的顶点加入队列，直至队列为空或访问到了所需顶点：

### Person类

在Lab1中使用了List来存储此Person认识的人：

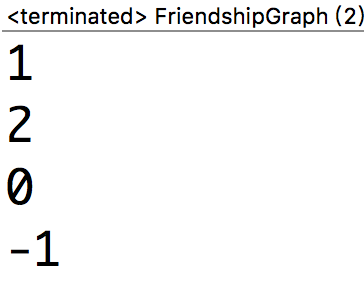
private List<Person> friends = new ArrayList<>();

但是由于Graph中提供了targets接口，所以可以弃用这个rep。之前的getFriends由graph.targets(person)来代替。

除此之外还在Person中重写了equals方法：若两个Person对象同名，则它们是同一个人（对象）。

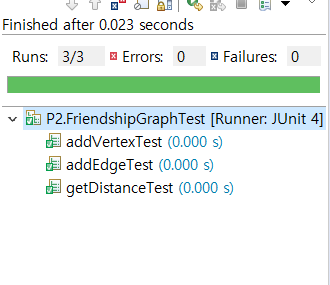
### 客户端main()

不变动 Lab1的3.3节给出的客户端代码(例如 main()中的代码)，同样的客户端代码仍可运行。

  
未改动异常判断，故在发生异常输入后仍可提供错误信息。

### 测试用例

重新执行在Lab1里所写的JUnit测试用例，测试在本实验里新实现的 FriendshipGraph 类的测试结果如下所示：



### 提交至Git仓库

通过Git提交当前版本到GitHub上的Lab2仓库：

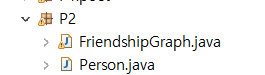
$ cd /Users/ucla/Desktop/Lab2-L170300901

$ git add .

$ git commit –m “finish P2”

$ git push

项目的目录结构树状示意图：

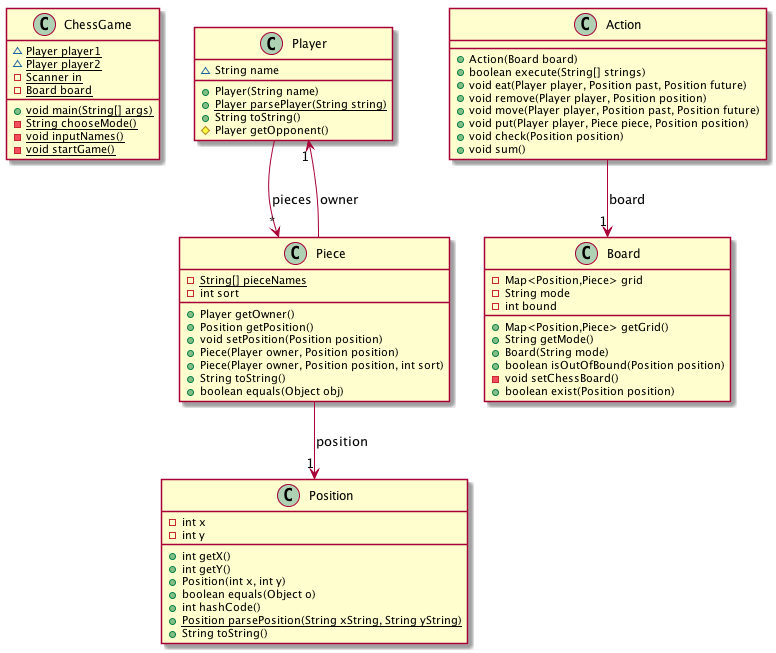


## Playing Chess

### ADT设计/实现方案

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

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

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

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 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
|  |  |
|  |  |
|  |  |

# 实验过程中收获的经验、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？
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？