

**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 | 郭子阳 |
| 学号 | 1170300520 |
| 班号 | 1703005 |
| 电子邮件 | guoziyang0033@gmail.com |
| 手机号码 | 18800420598 |

**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 2](#_Toc1988216)

[3.1 Poetic Walks 2](#_Toc1988217)

[3.1.1 Get the code and prepare Git repository 2](#_Toc1988218)

[3.1.2 Problem 1: Test Graph <String> 2](#_Toc1988219)

[3.1.3 Problem 2: Implement Graph <String> 3](#_Toc1988220)

[3.1.3.1 Implement ConcreteEdgesGraph 3](#_Toc1988221)

[3.1.3.2 Implement ConcreteVerticesGraph 3](#_Toc1988222)

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

[3.1.4.1 Make the implementations generic 3](#_Toc1988224)

[3.1.4.2 Implement Graph.empty() 4](#_Toc1988225)

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

[3.1.5.1 Test GraphPoet 4](#_Toc1988227)

[3.1.5.2 Implement GraphPoet 4](#_Toc1988228)

[3.1.5.3 Graph poetry slam 4](#_Toc1988229)

[3.1.6 Before you’re done 4](#_Toc1988230)

[3.2 Re-implement the Social Network in Lab1 5](#_Toc1988231)

[3.2.1 FriendshipGraph类 6](#_Toc1988232)

[3.2.2 Person类 6](#_Toc1988233)

[3.2.3 客户端main() 6](#_Toc1988234)

[3.2.4 测试用例 6](#_Toc1988235)

[3.2.5 提交至Git仓库 6](#_Toc1988236)

[3.3 Playing Chess 7](#_Toc1988237)

[3.3.1 ADT设计/实现方案 7](#_Toc1988238)

[3.3.2 主程序ChessGame设计/实现方案 8](#_Toc1988239)

[3.3.3 ADT和主程序的测试方案 9](#_Toc1988240)

[3.4 Multi-Startup Set (MIT) 9](#_Toc1988241)

[4 实验进度记录 9](#_Toc1988242)

[5 实验过程中遇到的困难与解决途径 9](#_Toc1988243)

[6 实验过程中收获的经验、教训、感想 10](#_Toc1988244)

[6.1 实验过程中收获的经验和教训 10](#_Toc1988245)

[6.2 针对以下方面的感受 10](#_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

实验环境：

系统：macOS Mojave 10.14.3，JDK11.0.2

ide：Eclipse IDE for Eclipse Committers 2018-12 (4.10.0)

编辑器：Visual Studio Code 1.31.1

构建工具：Gradle，Travis CI

版本管理：git

代码托管：Github

配置过程：

本次构建改用了Gradle构建工具，使用Gradle构建需要在项目根目录书写配置文件build.gradle，并且指定Java源文件和测试文件的目录。

由于Travis-CI指定使用oraclejdk11，而Travis-CI的默认Gradle由于版本问题，无法识别jdk11，故在本地使用Gradle Wrapper后再在Travis上构建。

Github Lab2仓库URL地址：

https://github.com/ComputerScienceHIT/Lab2-1170300520

# 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

该问题要求通过实现Graph接口，来完成一个完整的图结构，并使用该结构完成对诗歌的润色工作，以加强对接口和范型的认识。

### Get the code and prepare Git repository

将仓库从Github上clone下来：

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

并将自己的作业仓库clone下来

git clone <https://github.com/ComputerScienceHIT/Lab2-1170300520.git>

将所需要的文件夹复制到作业仓库，并将本地的祖业仓库与远程仓库同步：

git add .

git commit -m “…”

git push origin master

### Problem 1: Test Graph <String>

该问题似乎是采用了“先写测试用例、再实现具体方法”的顺序，要求首先完成问题的测试用例的编写。

所有静态方法的测试都在GraphStaticTest.java中，实例方法的测试在GraphInstanceTest.java中。

GraphInstanceTest是一个抽象类，针对Graph的不同的实现类需要编写具体的测试类，都需要继承自GraphInstanceTest类。

通过查看Graph类中的接口规约，即可很容易地编写各个测试类。由于暂时没有具体的实现，是无法通过测试的。

### Problem 2: Implement Graph <String>

该问题需要实现Graph接口的两个具体实现类：ConcreteEdgesGraph和ConcreteVerticsGraph，实现的范型要求指定为String。

#### Implement ConcreteEdgesGraph

ConcreteEdgesGraph是对Graph的一个具体实现。类中使用一个Set来存储所有的顶点，使用List来存储所有的边。边的类Edge需要自行完成，由于边类是不可变的，所以将边的属性source、target和weight都置为final。

Edge类和ConcreteEdgesGraph类都需要实现toString()方法以便于测试输出。

有一个注意点，在实现remove()方法删除一个顶点时，除了需要从set中删除该顶点外，还需要遍历edges来删除所有与被删除顶点相关的边。

#### Implement ConcreteVerticesGraph

ConcreteVerticesGraph是对Graph的一个具体实现。类中使用一个List来存储所有的顶点，顶点的类Vertex需要自行实现，由于无法修改ConcreteVerticesGraph的属性，只得在Vertex中加入一个Map，来描述顶点与顶点之间边的关系。最终实现的是一种类似邻接表的结构。

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

前面的两个具体实现类是基于String类型具体实现的，不具有通用性，我们要想让这两个类更加具有通用性的话，就要使用范型。

#### Make the implementations generic

将两个类的声明改成如下这个样子：

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

class Edge<L> { ... }

和

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

class Vertex<L> { ... }

并且将实现类中所有涉及具体的String都改成代表范型的L。完成之后，原先的测试类将依然可用。

#### Implement Graph.empty()

实现Graph中的静态方法empty()方法，创建一个空的Graph，由于无法创建接口，只得创建实现类，所以只需要选择一个Graph的实现类，实例化之后返回即可。

### Problem 4: Poetic walks

使用以上实现的图结构，根据现有的语料库构建图模型，对输入的文本进行润色。

#### Test GraphPoet

首先完成GraphPoet的测试类，根据接口的规约对每个接口进行响应的测试即可。

#### Implement GraphPoet

实现GraphPoet类，主要需要实现两个方法，一个是GraphPoet类的构造方法，构造方法参数为一个File，代表了语料库文件，构造方法用于通过语料库中的语句构建一个图结构，存储语料库中的单词关系，忽略大小写。

第二个需要实现的方法是poem方法，该方法传入一个String，用于通过图结构对该语句进行润色，在必要处添加适当单词。唯一需要注意的就是输入与输出的大小写需要保持一致。

#### Graph poetry slam

该问题要求修改main()方法，使用不同的输入对程序进行测试，也可以自定义语料库。

### Before you’re done

检查程序的Javadoc注释与注解的正确性，检查无误后即可push到github。

项目目录结构如下：

.

├── README.md

├── build.gradle

├── build.xml

├── doc

│   └── Lab-2\ Report\ Template.docx

├── gradle

│   └── wrapper

│   ├── gradle-wrapper.jar

│   └── gradle-wrapper.properties

├── gradlew

├── lib

│   ├── hamcrest-core-1.3.jar

│   └── junit.jar

├── src

│   └── P1

│      ├── graph

│      │   ├── ConcreteEdgesGraph.java

│      │   ├── ConcreteVerticesGraph.java

│      │   └── Graph.java

│      └── poet

│      ├── GraphPoet.java

│      ├── Main.java

│      └── mugar-omni-theater.txt

└── test

└── P1

   ├── graph

   │   ├── ConcreteEdgesGraphTest.java

   │   ├── ConcreteVerticesGraphTest.java

   │   ├── GraphInstanceTest.java

   │   └── GraphStaticTest.java

   └── poet

   └── GraphPoetTest.java

## Re-implement the Social Network in Lab1

Lab1中有一个使用图结构来表示社交网络关系到问题，该问题是对它的重新实现，使用问题1中的图结构即可。

### FriendshipGraph类

该类是实现一个描述社交关系的图结构，多数的方法都可以在ConcreteVerticesGraph类中有对应的接口，难点在于实现getDistance()方法。该方法实现广度优先搜索，与之前的实现不同的是，Lab1使用的是邻接矩阵存储图，现在使用邻接表，其余类似。

### Person类

Person类和原来的实现基本一致，只是添加了一个静态的ArrayList用于存储name，这样即可判断是否有name相同的Person。

### 客户端main()

main()方法依旧使用示例代码。

### 测试用例

测试依旧使用Lab1的测试用例，用来测试所有的public方法，主要测试getDistance()方法。

### 提交至Git仓库

检查程序的Javadoc注释与注解的正确性，检查无误后即可push到github。

项目目录结构如下：

.

├── README.md

├── build.gradle

├── build.xml

├── doc

│   └── Lab-2\ Report\ Template.docx

├── gradle

│   └── wrapper

│   ├── gradle-wrapper.jar

│   └── gradle-wrapper.properties

├── gradlew

├── lib

│   ├── hamcrest-core-1.3.jar

│   └── junit.jar

├── src

│   ├── P1

│   │   ├── graph

│   │   │   ├── ConcreteEdgesGraph.java

│   │   │   ├── ConcreteVerticesGraph.java

│   │   │   └── Graph.java

│   │   └── poet

│   │   ├── GraphPoet.java

│   │   ├── Main.java

│   │   └── mugar-omni-theater.txt

│   └── P2

│      ├── FriendshipGraph.java

│      └── Person.java

└── test

├── P1

│   ├── graph

│   │   ├── ConcreteEdgesGraphTest.java

│   │   ├── ConcreteVerticesGraphTest.java

│   │   ├── GraphInstanceTest.java

│   │   └── GraphStaticTest.java

│   └── poet

│   └── GraphPoetTest.java

└── P2

   └── FriendshipGraphTest.java

## Playing Chess

### ADT设计/实现方案

Action接口：对玩家行为的抽象

Board接口：对棋盘的抽象

ChessAction接口：对象棋玩家行为的抽象，继承自Action接口

GoAction接口：对围棋玩家行为的抽象，继承自Action接口

Game接口：对一局游戏的抽象

Piece接口：对棋子的抽象

Player接口：对玩家的抽象

Position类：描述了棋盘上的一个位置

ChessBoard类：描述了象棋棋盘的类，实现了Board接口

ChessGame类：描述了一局象棋游戏的类，实现了Game和Action接口

ChessPiece类：描述了一个象棋棋子的类，实现了Piece接口

ChessPlayer类：描述了一个象棋玩家的类，实现了Player接口

GoBoard类：描述了围棋棋盘的类，实现了Board接口

GoGame类：描述了一局围棋游戏的类，实现了Game和Action接口

GoPiece类：描述了一个围棋棋子的类，实现了Piece接口

GoPlayer类：描述了一个围棋玩家的类，实现了Player接口

### 主程序MyChessAndGoGame设计/实现方案

主程序主要通过用户输入，调用各个方法。由于方法已在接口中声明，直接调用接口方法即可。

运行示例如下：

请选择游戏类型（chess 或 go）：chess

请输入两个玩家的名字：

zhangsan

lisi

开始游戏！

当前是zhangsan的回合

1. 移动棋子

2. 吃子

3. 查询占用

4. 计算棋子总数

5. 跳过本轮

你的选择：1

请输入被移动棋子的位置（x 与 y）：0 0

请输入目标位置（x 与 y）：2 0

当前是lisi的回合

1. 移动棋子

2. 吃子

3. 查询占用

4. 计算棋子总数

5. 跳过本轮

你的选择：2

请输入你的棋子的位置（x 与 y）：7 0

请输入对方棋子的位置（x 与 y）：2 0

当前是zhangsan的回合

1. 移动棋子

2. 吃子

3. 查询占用

4. 计算棋子总数

5. 跳过本轮

你的选择：end

游戏结束！是否要查看走棋历史（y or n）？y

zhangsan 将棋子 zhangsan: 车从(0,0)移动到了(2,0)

lisi 将棋子lisi: 车从(7,0)移动到了(2,0)，吃掉了对手的zhangsan: 车

### ADT和主程序的测试方案

分别测试GoGame和ChessGame 的各个方法，使用自动化测试。

## 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

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 2019-3-12 | 19:00-21:13 | 完成P1 | 完成 |
| 2019-3-12 | 21:30-23:00 | 完成P2 | 完成 |
| 2019-3-14 | 16:30-19:00 | 完成P3 | 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 |
| 试图使用Gradle构建 | 在线学习 |

# 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

深入理解了ADT编程和面向接口编程，并规范了编程中的注释与注解的书写，学会了使用Gradle构建。

## 针对以下方面的感受

本次实验主要考察ADT编程以及范型的使用、接口的设计。ADT和使用范型可以提升代码的复用性，使代码语义化，结构清晰。本实验难度较简单，只需理解ADT和范型即可完成。