



哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2021 年春季学期 计算学部《软件构造》课程

Lab 2 实验报告

姓名	张瑞
学号	1190201421
班号	1936603
电子邮件	1190201421@stu.hit.edu.cn
手机号码	15736059288

目录

1	实验目标概述	3
2	实验环境配置	3
3	实验过程	3
3.1	Poetic Walks	3
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	4
3.1.3.2	Implement ConcreteVerticesGraph	5
3.1.4	Problem 3: Implement generic Graph<L>	6
3.1.4.1	Make the implementations generic	6
3.1.4.2	Implement Graph.empty()	6
3.1.5	Problem 4: Poetic walks	6
3.1.5.1	Test GraphPoet	6
3.1.5.2	Implement GraphPoet	7
3.1.5.3	Graph poetry slam	7
3.1.6	使用 Eclemma 检查测试的代码覆盖度	8
3.1.7	Before you're done	8
3.2	Re-implement the Social Network in Lab1	9
3.2.1	FriendshipGraph 类	9
3.2.2	Person 类	9
3.2.3	客户端 main()	9
3.2.4	测试用例	9
3.2.5	提交至 Git 仓库	9
4	实验进度记录	10
5	实验过程中遇到的困难与解决途径	11
6	实验过程中收获的经验、教训、感想	11
6.1	实验过程中收获的经验教训	11
6.2	针对以下方面的感受	12

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

本次实验的环境之前已经搭建好，此处不再赘述。需要指出的是，本次实验需要在 Eclipse IDE 中安装配置 EclEmma，按实验手册操作时发现 Eclipse 已经默认配置好了。

在这里给出你的 GitHub Lab2 仓库的 URL 地址（Lab2-学号）。

<https://github.com/ComputerScienceHIT/HIT-Lab2-1190201421>

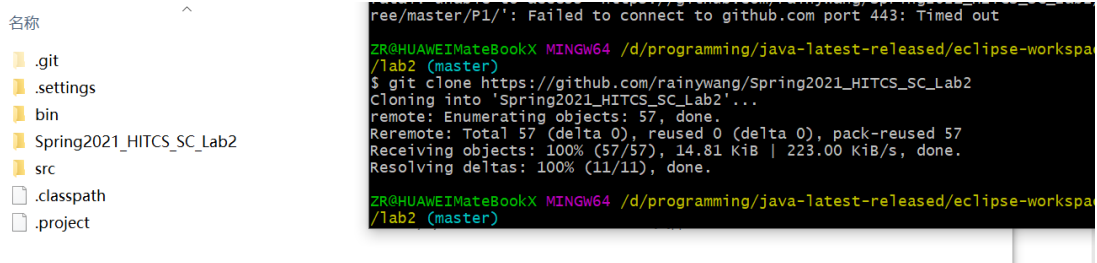
3 实验过程

3.1 Poetic Walks

先用两个类 ConcreteEdgesGraph 和 ConcreteVerticesGraph 分别实现 Graph 接口。然后将 ADT 泛型化，进行从 String 到 L 的转变。最后运用已经实现的 Graph 模型实现一个应用——Poetic walks。

3.1.1 Get the code and prepare Git repository

先选好工作区的位置，然后在 git bash 中输入“git init”建立本地仓库，再“git clone https://github.com/rainywang/Spring2021_HITCS_SC_Lab2”即可获取远程仓库的代码：



3.1.2 Problem 1: Test Graph <String>

本部分要求完成测试用例，其中 GraphStaticTest 的测试用例已经给出，需要完成的是 GraphInstanceTest。我们需要对 Graph 中的各个实例方法都给出测试用例，首先根据规约进行等价类划分，写出测试策略：

```
// Testing strategy:
//
// add():
//   vertex already existed, did not exist
// set():
//   weight <0, =0, >0
//   source already existed, did not exist
//   target already existed, did not exist
//   source and target are the same, different
// remove():
//   vertex already existed, did not exist
//   vertex has only edges to it, only edges from it, both edges to and from it, no edges to or from it
// vertices():
//   the graph is empty, not empty
// sources():
//   vertex already existed, did not exist
//   vertex has no source, has source(s)
// targets():
//   vertex already existed, did not exist
//   vertex has no target, has target(s)
```

此处的方法中，add()为 creator，set()和 remove()是 mutator，vertices()、sources()和 targets()是 observer。

接下来按照“覆盖每个取值”的策略为每个方法构造测试用例，其中，对于 creator 和 mutator，应该用 observer 来观察操作后的结果是否正确；对于 observer，应该先用 creator 和 mutator 来改变对象再观察结果是否正确。

3.1.3 Problem 2: Implement Graph <String>

3.1.3.1 Implement ConcreteEdgesGraph

首先完成对 Edge 的设计，确定 rep 有 source、target 和 weight，方法包含

getSource()、getTarget()和 getWeight(), 并且重写 toString(), 给出 Edge()的构造方法和 checkRep 方法。并完成 spec、AF、RI、safety from Rep Exposure 的撰写。

然后完成对 Edge 的测试用例的书写, 先进行等价类划分, 写出测试策略, 然后按照“覆盖每个取值”的策略为每个方法构造测试用例。

接下来完成对 ConcreteEdgesGraph 中重写的 toString 方法的测试用例。

最后完成 ConcreteEdgesGraph 的 spec、AF、RI、safety from Rep Exposure, 并实现 Edge 和 ConcreteEdgesGraph。

值得注意的是, 我在写代码过程中遇到了在 for 循环里进行删除操作进而引发错误的问题, 该段错误代码如下:

```
@Override public boolean remove(String vertex) {
    boolean b = vertices.remove(vertex);
    for (Edge edge : edges){
        if(edge.getTarget().equals(vertex) || edge.getSource().equals(vertex)){
            edges.remove(edge);
        }
    }
    checkRep();
    return b;
}
```

错误的原因是删除了元素, 但循环的计数会继续, 从而导致遍历出现遗漏, 没把该删的元素删完。最后我采用了按 List 内在顺序从大到小遍历的方法, 避免删除操作带来遗漏:

```
@Override public boolean remove(String vertex) {
    boolean b = vertices.remove(vertex);
    int n = edges.size();
    for (int i = n-1; i >= 0; i--) {
        if(edges.get(i).getSource().equals(vertex) || edges.get(i).getTarget().equals(vertex)) {
            edges.remove(i);
        }
    }
    checkRep();
    return b;
}
```

3.1.3.2 Implement ConcreteVerticesGraph

首先完成对 Vertex 的设计, 确定 rep 有 label、sources 和 targets, 方法包含 setSource()、setTarget()、getLabel()、getSources()和 getTargets(), 并且重写 toString(), 给出 Vertex()的构造方法和 checkRep 方法。并完成 spec、AF、RI、safety from Rep Exposure 的撰写。

然后完成对 Vertex 的测试用例的书写, 先进行等价类划分, 写出测试策略, 然后按照“覆盖每个取值”的策略为每个方法构造测试用例。

接下来完成对 ConcreteVerticesGraph 中重写的 toString 方法的测试用例。

最后完成 ConcreteVerticesGraph 的 spec、AF、RI、safety from Rep Exposure, 并实现 Vertex 和 ConcreteVerticesGraph。

有了实现 ConcreteEdgesGraph 和 ConcreteEdgesGraphTest 的经验，这一段任务完成得较为顺利，就是实现 ConcreteVerticesGraph 的方法的时候，发现好像更复杂了一点，还是花了不少时间。

3.1.4 Problem 3: Implement generic Graph<L>

3.1.4.1 Make the implementations generic

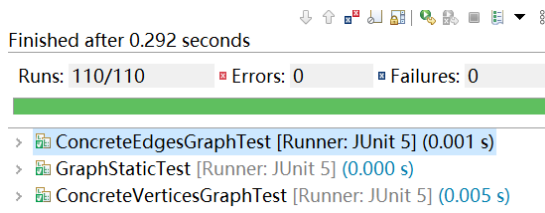
将前面已经实现的 String 类型改为泛型 L，这一步十分简单，将 String 替换为 L（注意不能替换 toString 部分输出时用到的 String 类型），将 Vertex 改为 Vertex<L>，将 Edge 改为 Edge<L>，ConcreteEdgesGraph 变为 ConcreteEdgesGraph<L>，ConcreteVerticesGraph 变为 ConcreteVerticesGraph<L>。

3.1.4.2 Implement Graph.empty()

我采用的是 ConcreteEdgesGraph<L>来实现 Graph<L>：

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

然后为了测试 Graph<L>的正确性，我在 GraphStaticTest 中为 IntegerGraph 和 DoubleGraph 设计了几个简单的测试用例，最终所有测试用例都通过了：



3.1.5 Problem 4: Poetic walks

3.1.5.1 Test GraphPoet

为了方便测试，我在 GraphPoet 中添加了几个 observer，分别是 vertices()、sources()和 targets()。

然后完成对 GraphPoet 的测试用例的书写，先进行等价类划分，写出测试策略，然后按照“覆盖每个取值”的策略为每个方法构造测试用例。

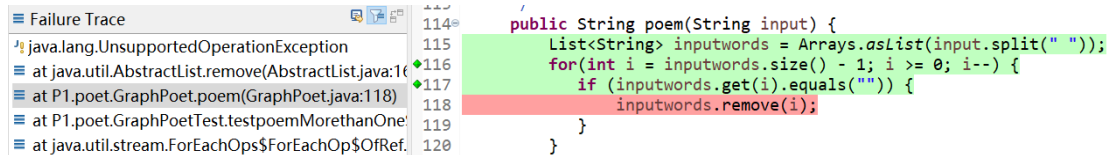
因为添加的 vertices()、sources()和 targets()，以及待实现的 toString()我都准备直接采用 GraphPoet 中 rep 对应的 graph 内在的方法来实现，对于它们的正确性

已在前面得到验证，故此处未再设计测试用例。

3.1.5.2 Implement GraphPoet

完成测试用例之后，接着完成 GraphPoet 的 AF、RI、safety from Rep Exposure，并实现各个方法。

值得注意的是，我在写代码过程中遇到了下列报错：



```
114 public String poem(String input) {
115     List<String> inputwords = Arrays.asList(input.split(" "));
116     for(int i = inputwords.size() - 1; i >= 0; i--) {
117         if (inputwords.get(i).equals("")) {
118             inputwords.remove(i);
119         }
120     }
121 }
```

经查阅资料发现，Arrays.asList 转换得到的 ArrayList 继承了 AbstractList 类，而这个类的 set、remove、add 方法都定义为了“throw new UnsupportedOperationException();”，故不能进行删除操作，然后我通过再创建一个 List 的办法解决了此问题：

```
List<String> inputword = Arrays.asList(input.split(" "));
List<String> inputwords = new ArrayList<>();
inputwords.addAll(inputword);
for(int i = inputwords.size() - 1; i >= 0; i--) {
    if (inputwords.get(i).equals("")) {
        inputwords.remove(i);
    }
}
```

3.1.5.3 Graph poetry slam

简单引用了泰戈尔的一句话构造了一个句子：



```
29 final GraphPoet example = new GraphPoet(new File("src/P1/poet/example.txt"));
30 final String input2 = "Some like flowers, but some like leaves.";
31 System.out.println(input2 + "\n>>>\n" + example.poem(input2));
```

example - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

Let life be beautiful like summer flowers, and death like autumn leaves.

< 第 1 行, 第 1 列 100% Windows (CRLF) UTF-8

```
Some like flowers, but some like leaves.
>>>
Some like summer flowers, but some like autumn leaves.
```

3.1.6 使用 Eclemma 检查测试的代码覆盖度

Element	Cover...	Covered In...	Missed Ins...	Total Instr...
lab2	94.1 %	4,325	273	4,598
src	90.4 %	1,812	193	2,005
test	96.9 %	2,513	80	2,593
P1.graph	96.9 %	2,155	70	2,225
ConcreteVerticesGra	93.8 %	608	40	648
ConcreteEdgesGraph	89.7 %	210	24	234
GraphInstanceTest.ja	99.7 %	884	3	887
GraphStaticTest.java	99.3 %	453	3	456
P1.poet	97.3 %	358	10	368
GraphPoetTest.java	97.3 %	358	10	368

3.1.7 Before you're done

请按照 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。

先进入工作区的位置，然后在 git bash 中输入“git init”建立本地仓库，再输入“git remote add lab2 https://github.com/ComputerScienceHIT/HIT-Lab2-1190201421”添加远程仓库，然后输入“git add <filename>”，“git commit -m “****””，“git push -u lab2 master”即可向远程仓库提交代码。

在这里给出你的项目的目录结构树状示意图。

```

v lab2 [lab2 master]
  > JRE System Library [JavaSE-1.8]
  > JUnit 5
  > src
    > P1.graph
      > ConcreteEdgesGraph.java
      > ConcreteVerticesGraph.java
      > Graph.java
    > P1.poet
      > GraphPoetTest.java
      > Main.java
      > example.txt
      > mugar-omni-theater.txt
  > test
    > P1.graph
      > ConcreteEdgesGraphTest.java
      > ConcreteVerticesGraphTest.java
      > GraphInstanceTest.java
      > GraphStaticTest.java
    > P1.poet
      > GraphPoetTest.java
      > bothcases.txt
      > emptyfile.txt
      > lowercases.txt
      > oneword.txt
      > uppercases.txt

```


3.2 Re-implement the Social Network in Lab1

运用已经实现的 Graph 模型重新实现 Lab1 中的 FriendshipGraph。

3.2.1 FriendshipGraph 类

Lab1 中已经有了 FriendshipGraph 的一个实现，但这里需要复用已经设计出来的 Graph<L>，故进行了一定修改。Rep 直接使用 Graph<L>，这里我的 Graph<L>选择的是 ConcreteEdgesGraph<L>的实现方式。至于方法，除了要求的 addVertex、addEdge 和 getDistance，我还加入了 checkRep、getPeople 和 getFriends，其中 getPeople 用于查看 graph 中的 Person，getFriends 用于查看 Person a 的朋友（单向关系）。

然后按照 ADT 的设计方法，完善了 Lab1 中的 FriendshipGraph，写出所有的 spec、AF、RI、safety from Rep Exposure。

3.2.2 Person 类

同样的，Lab1 中也已经有了 Person 的一个实现，此处继续使用这个类。但由于朋友关系能交给 graph 去记录，故略去了 Person 中和 friend 相关的内容。同时，由于 Person 是一个 immutable 的类，我重写了 equals、hashCode 和 toString，还写了 spec、AF、RI、safety from Rep Exposure，实现了 checkRep 方法。

3.2.3 客户端 main()

此处按实验手册照搬 Lab1 中的代码。

3.2.4 测试用例

我先是按实验手册要求照搬了 Lab1 中的测试用例，后来发现当时设计的测试用例覆盖度不够，就对 FriendshipGraph 中各个方法进行等价类划分，写出测试策略，然后按照“覆盖每个取值”的策略为每个方法构造测试用例。

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。

先进入工作区的位置，然后在 git bash 中输入“git init”建立本地仓库，再输入

“git remote add lab2 https://github.com/ComputerScienceHIT/HIT-Lab2-1190201421”
 添加远程仓库，然后输入“git add <filename>”,“git commit -m “***””,“git push -u
 lab2 master”即可向远程仓库提交代码。

在这里给出你的项目的目录结构树状示意图。

```

  v src
    > P1.graph
    > P1.poet
    v P2
      > FriendshipGraph.java
      > Person.java
  v test
    > P1.graph
    > P1.poet
    v P2
      > FriendshipGraphTest.java
  
```

4 实验进度记录

日期	时间段	计划任务	实际完成情况
2021-5-25	14:00-15:30	完成 P1 的 Problem1	lab2 相关内容尚未讲授，没太弄明白，未能完成
2021-6-1	13:45-15:30	继续完成 P1 的 Problem1	按计划完成
2021-6-8	13:45-15:30	重新修改已完成的 P1 的 Problem1，完成 ConcreteEdgesGraph 的 Edge 的设计与测试用例	按计划完成
2021-6-9	10:00-12:00	完成 ConcreteEdgesGraph	基本完成，有部分测试用例未通过
2021-6-9	15:00-18:00	修复 ConcreteEdgesGraph 中的问题，完成 ConcreteVerticesGraph 中的 Vertex 的设计与测试用例	按计划完成
2021-6-10	13:00-16:00	完成 ConcreteVerticesGraph	基本完成，有部分测试用例未通过
2021-6-10	16:00-18:00	修复 ConcreteVerticesGraph 中的问题，完成 Problem3	按计划完成

2021-6-11	10:00-11:45	完成 GraphPoet 的设计与测试用例	按计划完成
2021-6-11	16:00-19:00	完成 Poetic walks	完成，但是实现 GraphPoet 后发现还需进一步完善测试用例
2021-6-11	20:00-22:00	完成 P1	按计划完成
2021-6-12	19:00-20:00	完成 P2	完成了实验手册上的任务，但添加测试用例额外花了时间

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
对于一些 Java 的类不熟悉	模仿老师已给代码，还不会的就上网搜索
Git 的加载是个问题，不太稳定	反复加载，坚持不懈
不会 build	问天问地，抱大佬大腿

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

在本次实验中对 Java 更加的熟悉了，代码能力有一定提升，也学会了更多关于 Java 类的细节。而且我还是把在循环遍历中删除元素的坑踩了一遍，对这个点的印象更加深刻了。此外，我还学习到了 `Arrays.asList` 转换得到的 `ArrayList` 继承了 `AbstractList` 类，而这个类的 `set`、`remove`、`add` 方法都定义为了“`throw new UnsupportedOperationException();`”，故不能进行增加和删除操最后，我还是没克服喜欢拖延，把事情堆到ddl快到了再开始赶的恶习，下次一定提前做好进度。

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？

前者更为抽象，设计出来的成果能在多个应用场景下复用，二次开发的成本低；后者虽然更具体，但一旦有类似场景需要再设计时，重复工作较多。

(2) 使用泛型和不使用泛型的编程，对你来说有何差异？

泛型理解起来更抽象一点，但代入一个具体的类型还是很好理解的。而且使用泛型编程方便以后的复用。

(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？

能考虑到更全面的测试策略，思路更清晰。我还比较适应，就是写测试用例写得有点崩溃。

(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？

能利用已有成果，迅速完成多个场景的应用。一次开发，多次复用，效率更高。

(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？

什么 P3？本次实验无 P3。（但是下次就有了，希望我能适应……）

(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？

既能使他人更容易读懂自己的代码，也能使自己以后能快速回想起设计与实现的思路。时刻注意程序的正确性，同时保证安全性，不会被客户端恶意破坏。

(7) 关于本实验的工作量、难度、deadline。

比起 Lab1 工作量更大、难度有所提升，但也还在可承受范围内，deadline 较为合理，虽然我懒，又拖延了。

(8) 《软件构造》课程进展到目前，你对该课程有何体会和建议？

感觉还是能提升不少编程能力，让自己能编写出更高质量的程序。问题就是学时太少，导致实验时间较紧张，而且把课程和考试月排在一起压力较大。