

**2024年春季学期  
计算学部《软件构造》课程**

**Lab 2实验报告**

|  |  |
| --- | --- |
| 姓名 | 姚冠希 |
| 学号 | 2022112625 |
| 班号 | 2237101 |
| 电子邮件 | 1224572938@qq.com |
| 手机号码 | 13392406469 |

**目录（更新失败）**

[1 实验目标概述 1](#_Toc97648181)

[2 实验环境配置 1](#_Toc97648182)

[3 实验过程 1](#_Toc97648183)

[3.1 Poetic Walks 1](#_Toc97648184)

[3.1.1 Get the code and prepare Git repository 1](#_Toc97648185)

[3.1.2 Problem 1: Test Graph <String> 1](#_Toc97648186)

[3.1.3 Problem 2: Implement Graph <String> 1](#_Toc97648187)

[3.1.3.1 Implement ConcreteEdgesGraph 2](#_Toc97648188)

[3.1.3.2 Implement ConcreteVerticesGraph 2](#_Toc97648189)

[3.1.4 Problem 3: Implement generic Graph<L> 2](#_Toc97648190)

[3.1.4.1 Make the implementations generic 2](#_Toc97648191)

[3.1.4.2 Implement Graph.empty() 2](#_Toc97648192)

[3.1.5 Problem 4: Poetic walks 2](#_Toc97648193)

[3.1.5.1 Test GraphPoet 2](#_Toc97648194)

[3.1.5.2 Implement GraphPoet 2](#_Toc97648195)

[3.1.5.3 Graph poetry slam 2](#_Toc97648196)

[3.1.6 使用Eclemma检查测试的代码覆盖度 2](#_Toc97648197)

[3.1.7 Before you’re done 2](#_Toc97648198)

[3.2 Re-implement the Social Network in Lab1 2](#_Toc97648199)

[3.2.1 FriendshipGraph类 2](#_Toc97648200)

[3.2.2 Person类 3](#_Toc97648201)

[3.2.3 客户端main() 3](#_Toc97648202)

[3.2.4 测试用例 3](#_Toc97648203)

[3.2.5 提交至Git仓库 3](#_Toc97648204)

[4 实验进度记录 3](#_Toc97648205)

[5 实验过程中遇到的困难与解决途径 3](#_Toc97648206)

[6 实验过程中收获的经验、教训、感想 4](#_Toc97648207)

[6.1 实验过程中收获的经验和教训（必答） 4](#_Toc97648208)

[6.2 针对以下方面的感受（必答） 4](#_Toc97648209)

# 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、表示不变性、抽象过程使用 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；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

简要陈述我配置本次实验所需环境的过程：

本次实验没有任何需要配置的环境。

在这里给出我的GitHub Lab2仓库的URL地址（HIT-Lab2-2022112625）：

https://github.com/ComputerScienceHIT/HIT-Lab2-2022112625

# 实验过程

## Poetic Walks

在这里简要概述你对该任务的理解：

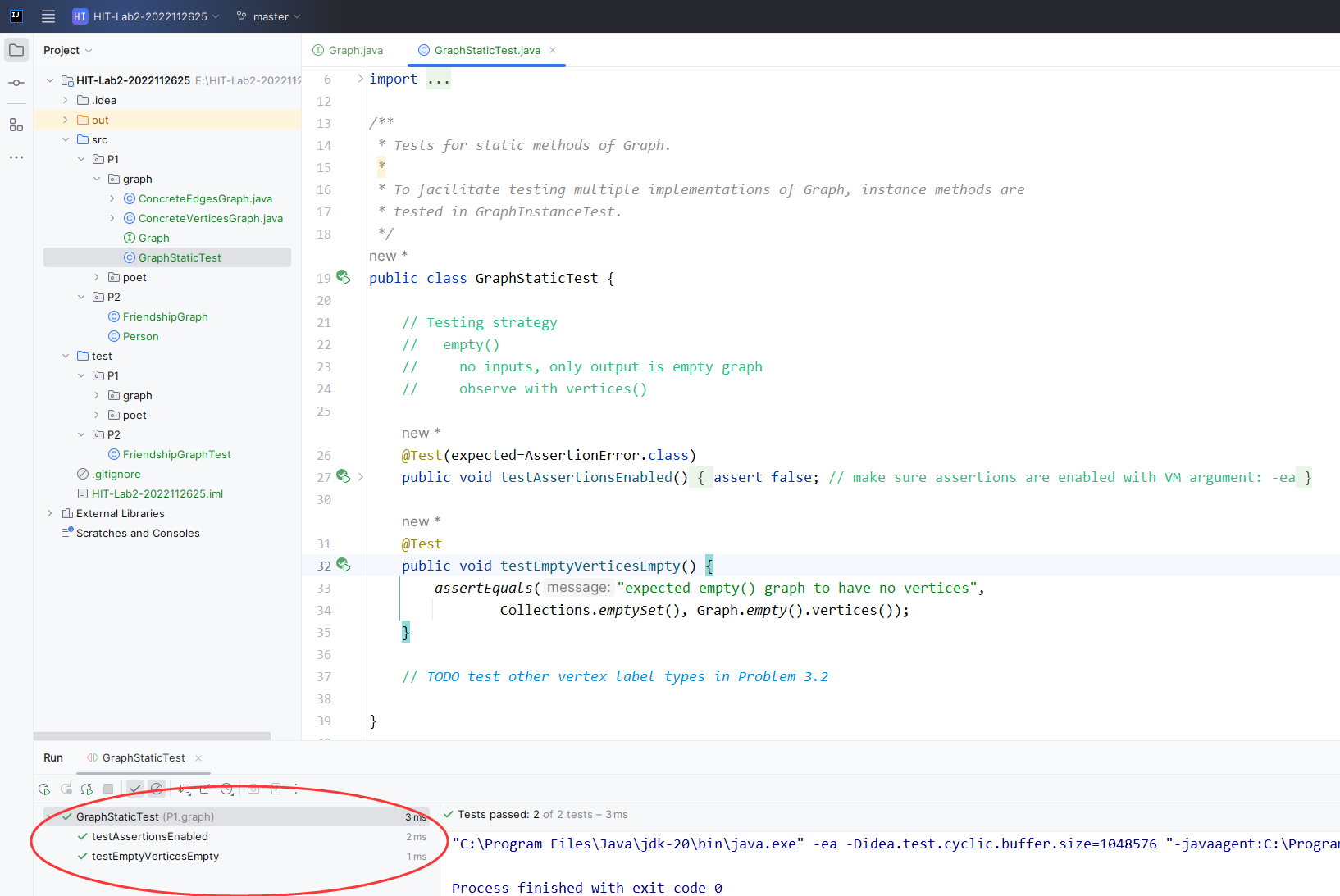
先写一个graph泛类，然后分别以顶点Vertex和边Edge作为对象实现具体类ConcreteEdgeGraph和ConcreteVertexGraph.要求该类能通过各种测试。

### 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：

同上次实验。

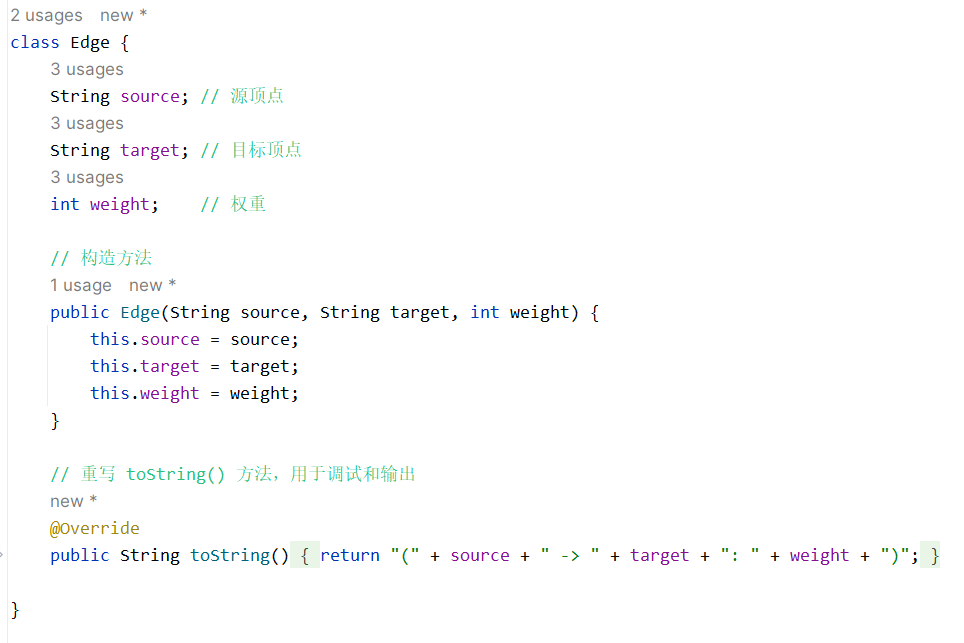
### Problem 1: Test Graph <String>



### Problem 2: Implement Graph <String>

与以下几个任务一起完成了，看下图即可。

#### Implement ConcreteEdgesGraph



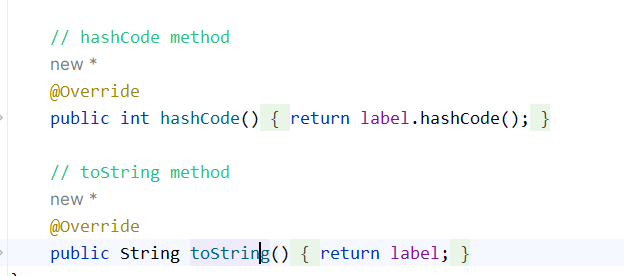
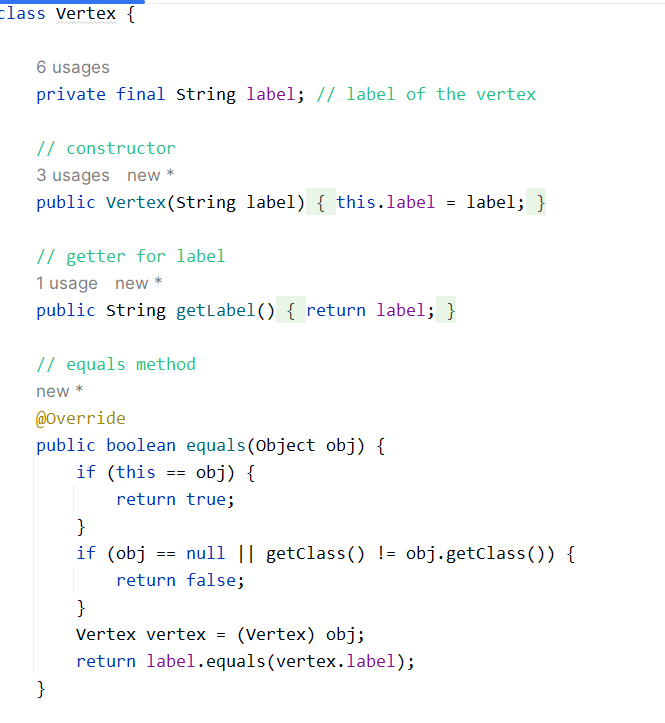
先写对象Edge，属性有两个顶点和边长（权重），方法有初始化方法和转字符方法。



然后以这个对象为基础写图的各种方法就行了，这里贴其中一个方法。

#### Implement ConcreteVerticesGraph

与上个任务大同小异，难度还小一点，毕竟顶点更符合人类思维。这里贴一张Vertex对象的属性和方法。



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

**分为以下两个部分。**

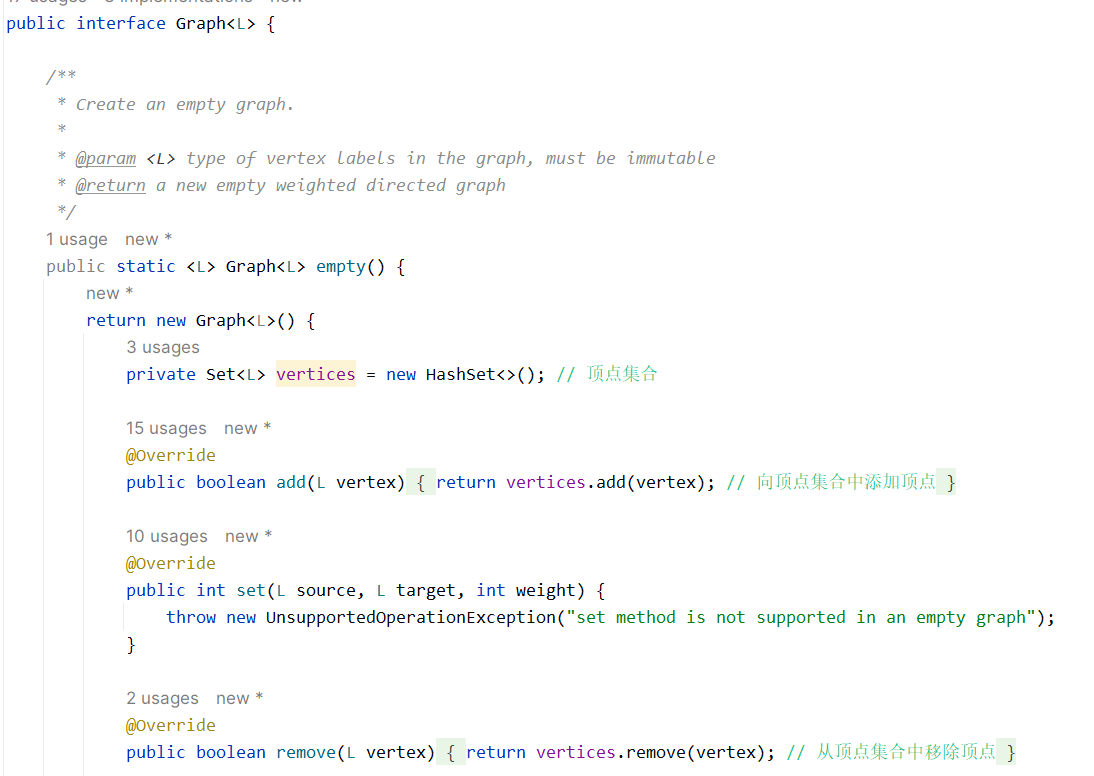
#### Make the implementations generic



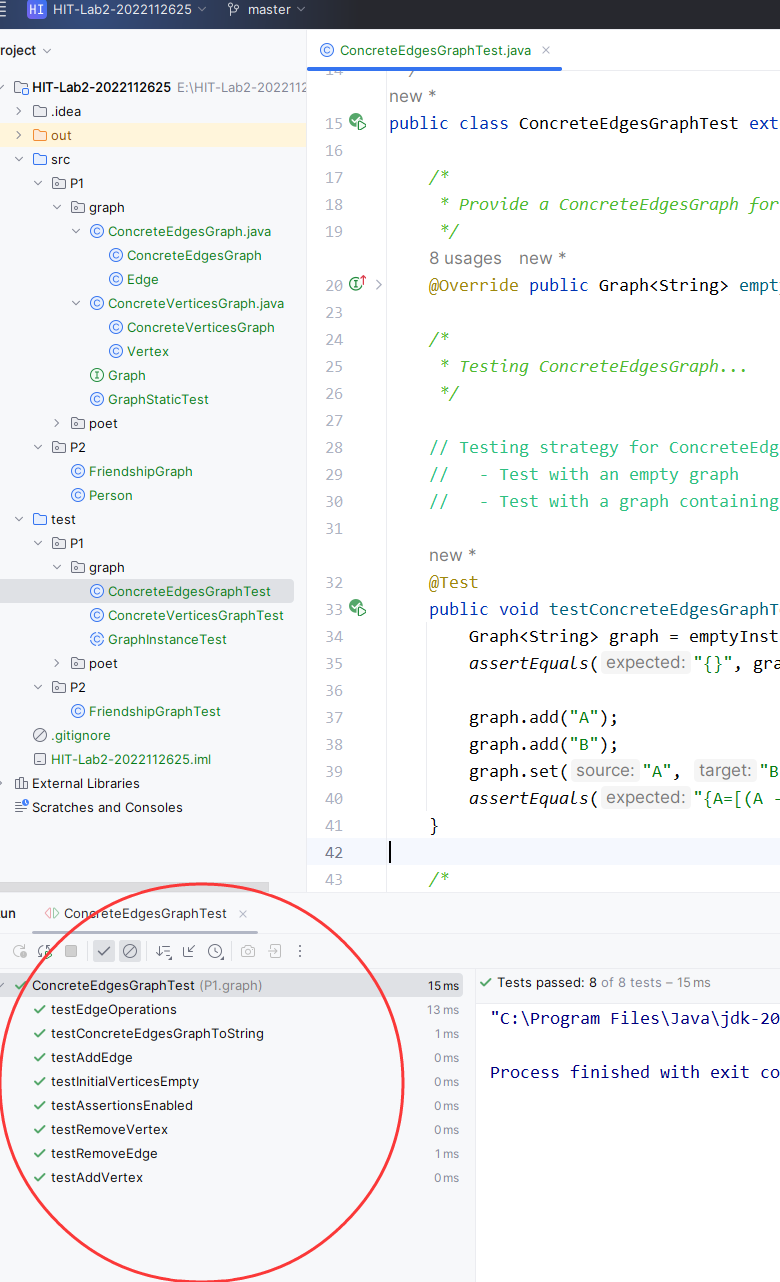


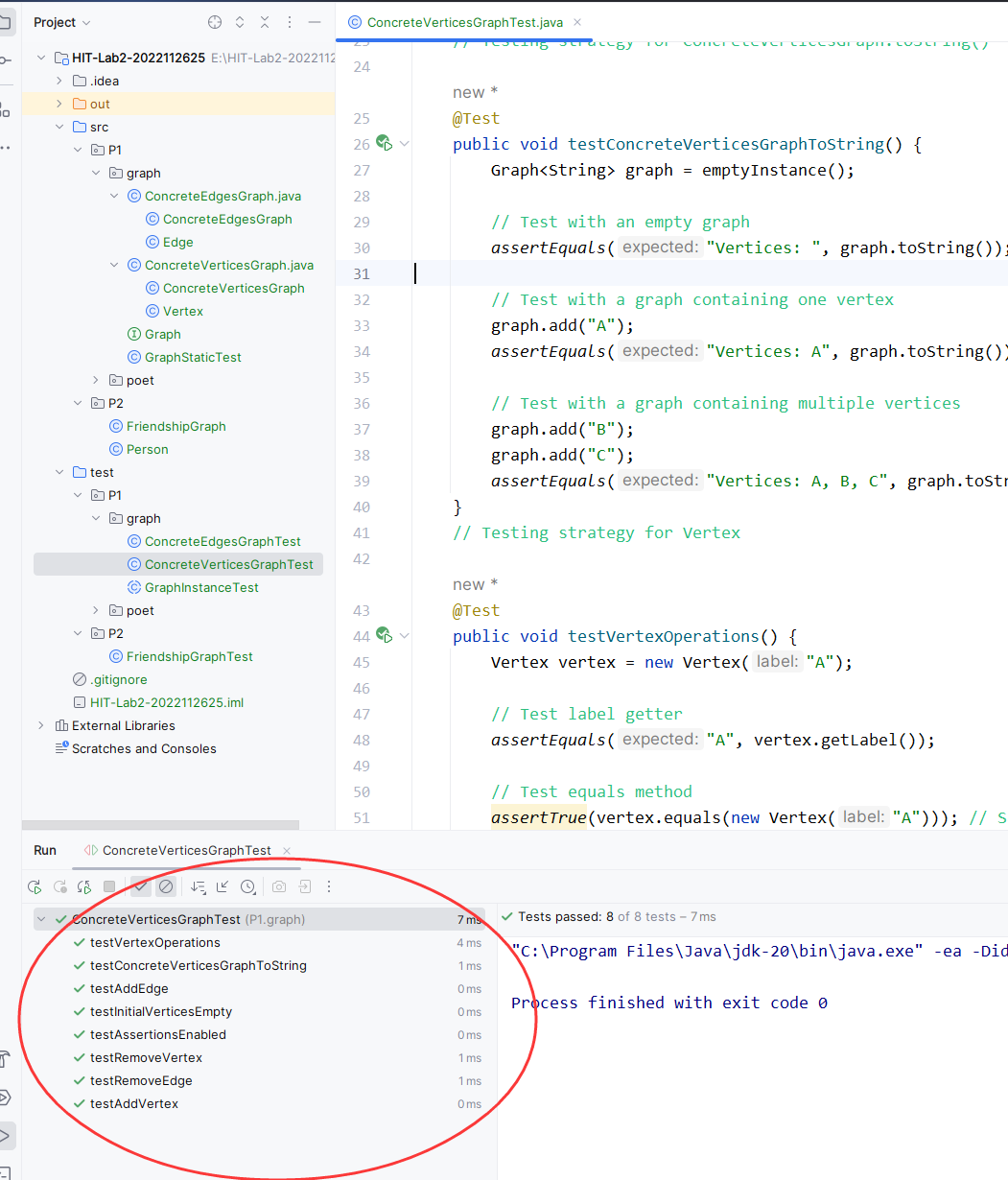
和之前两个任务一起做了。

#### Implement Graph.empty()



这两页补充一下test运行结果，全过：





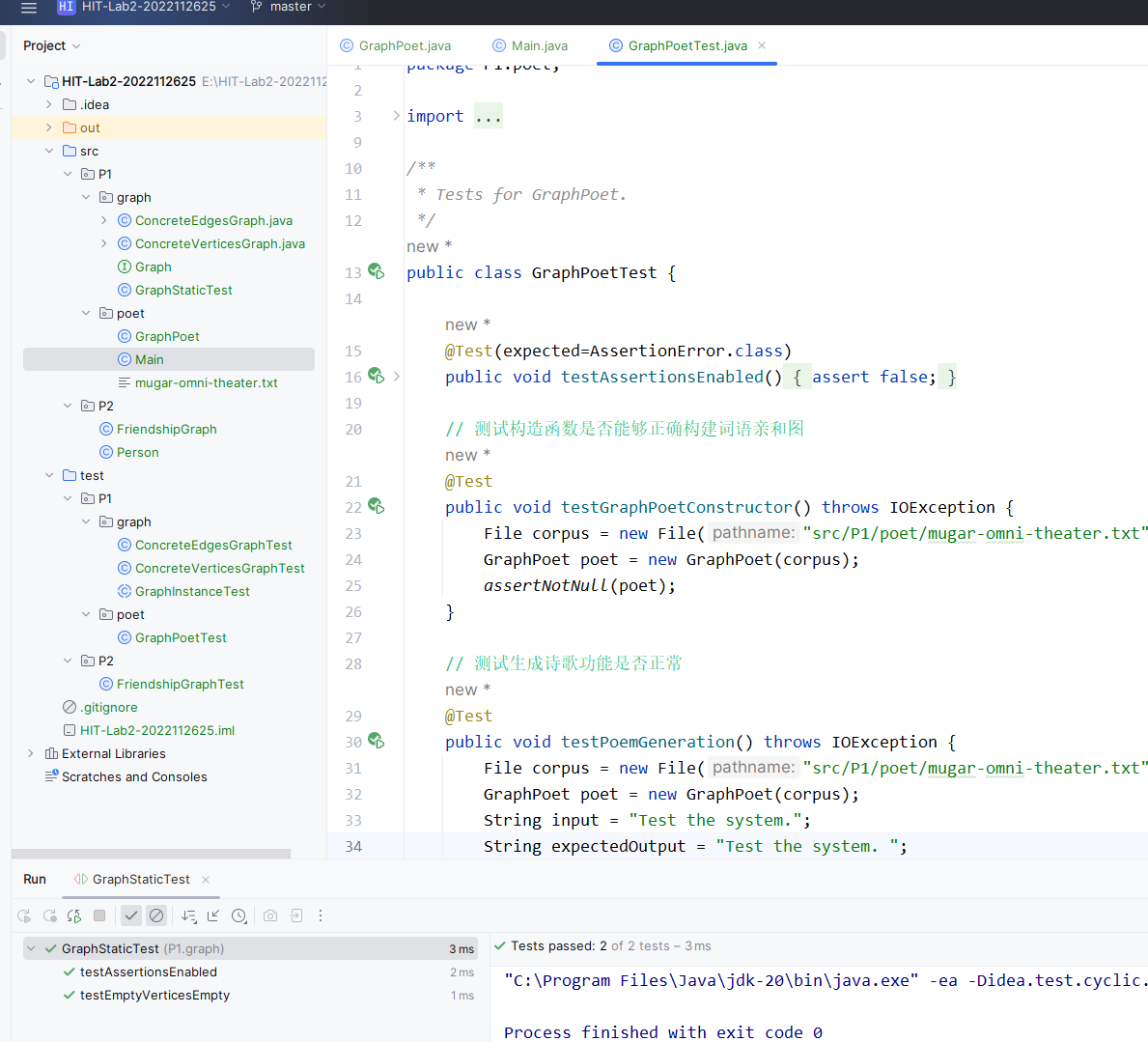
其中TestRemoveEdge在两个测试中都一直WA，改了很多次才过。第二个测试的TestAddEdge也老红色感叹号，改了一会儿。

### Problem 4: Poetic walks

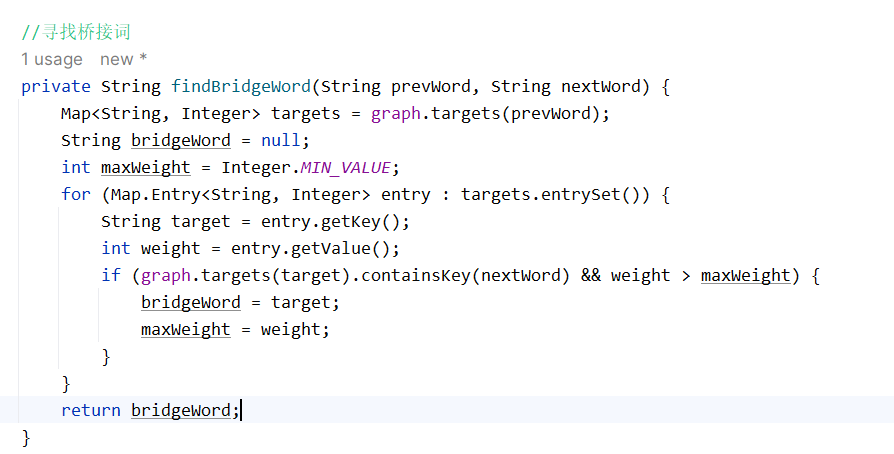
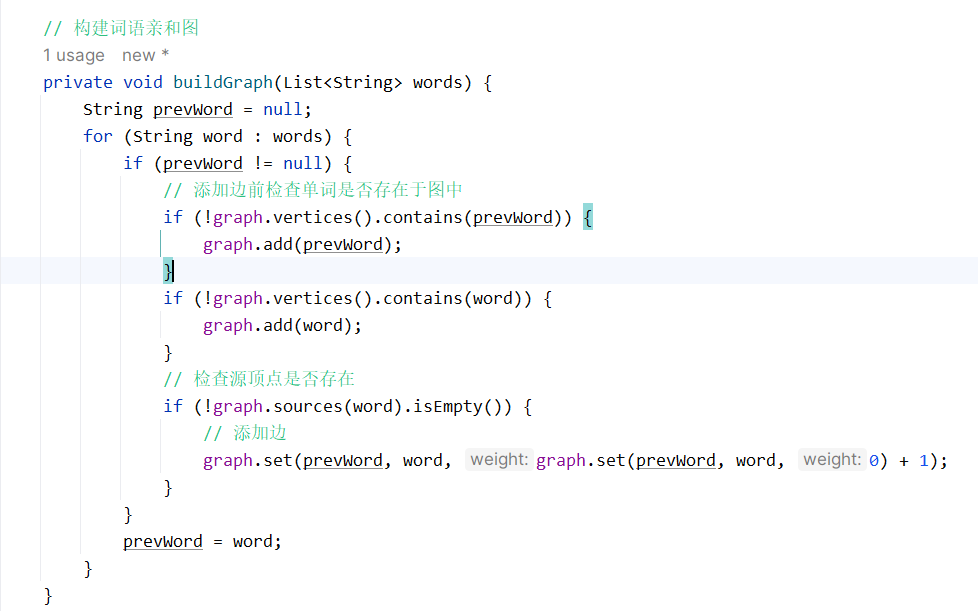
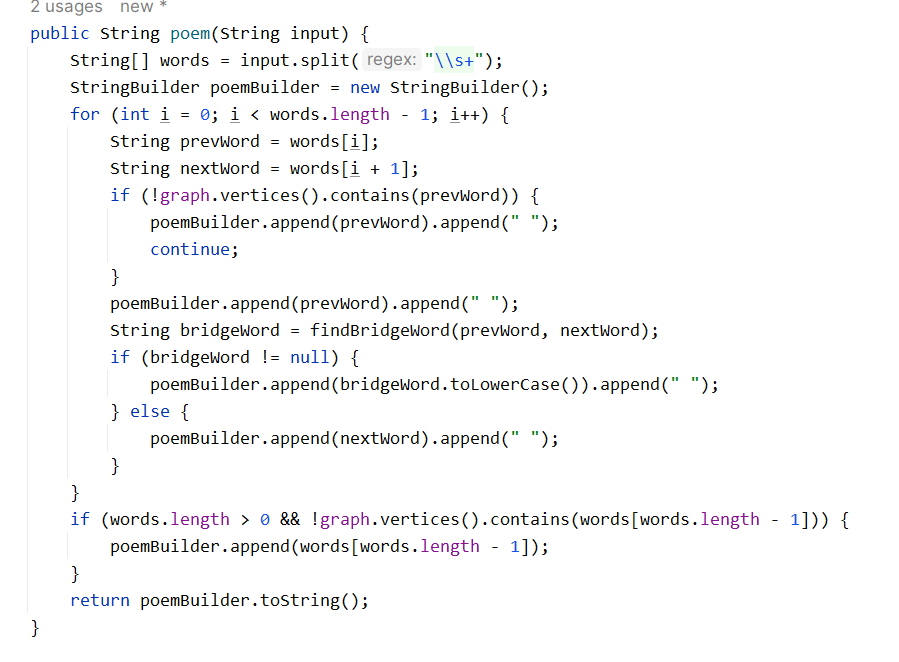
用上面完成的有权图结构制作电子诗人，利用数据集统计出不同词汇间的邻接权重图，以此判定单词间的关联程度，从而实现当输入一些单词时可以补全桥接词、返回一首完整的诗。

#### Test GraphPoet

**直接贴最后的test运行结果在这里吧。**



#### Implement GraphPoet



贴几个主要方法在这里。poem：接受一个字符串作为输入，将其分解为单词，并在每个单词之间寻找桥接词。最终，它返回一个修改后的字符串，其中桥接词被插入到原始单词之间。buildGraph:从给定的单词列表中构建词语亲和图。它遍历单词列表，将每个单词添加到图中，并为相邻的单词添加边，表示它们在语料库中相邻。findBridgeWord:寻找两个给定单词之间的桥接词。它检查第一个单词的目标单词，并查看这些目标单词是否与第二个单词有连接。如果是，则选择其中权重最大的单词作为桥接词。

### 使用Eclemma/Code Coverage for Java检查测试的代码覆盖度

### Before you’re done

### 在这里给出你的项目的目录结构树状示意图。

### 

## Re-implement the Social Network in Lab1

重做一遍Lab1P3，但是用到这次自己编完的包里的类。

### FriendshipGraph类

### 

### 

### Person类

### 

### 客户端main()

### 

### 测试用例

已经在上图中展示完毕了。

### 提交至Git仓库

同上次，直接提交上去即可。

# 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 |
| 2024.4.23 | 12:00-24:00 | P1的graph | 没按时完成 |
| 2024.4.24 | 0:00-1:30 | 接着P1的graph | 补完了 |
| 2024.4.24 | 12:00-16:30 | P1的poet | 比想象中快 |
| 2024.4.25 | 15:40-22:00 | P2和实验报告 | 有点磨蹭地完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 那个RemoveEdgeTest就在那里一直挂啊一直挂 | 改了很多词都不行，干脆换个思路整个重写了一遍这个方法又改了改就过了 |
| 那个Test of the system一直输出Test the system，我寻思这不相当于没变吗。 | 一开始是图就没都进去，后来把单词两边空格去掉就能读进去了，但还是报WA。最后使用了特殊方法解决（在代码里）。 |
| 不知道怎么在包里设置包 | 然后发现idea是可以内部把文件夹设置成包的，上次Lab1好像格式全是错的 |

# 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（必答）

通过这次实验，我对训练抽象数据类型（ADT）的设计、规约、测试有了初步了解，并使用面向对象编程（OOP）技术实现 ADT。我学会了从问题描述中识别所需的 ADT；设计 ADT 规约并评估规约的质量；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、表示不变性、抽象过程使用 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；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。

这次实验也让我对java和idea有了更深入的了解，对junit测试有了更多感悟。

## 针对以下方面的感受（必答）

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？

面向ADT的编程更侧重于抽象数据类型的设计和实现，强调数据的组织和操作，更注重程序的模块化和可复用性；而直接面向应用场景编程则更关注解决具体问题，可能更加贴近实际应用需求，但可能牺牲了通用性和可扩展性。

1. 使用泛型和不使用泛型的编程，对你来说有何差异？

使用泛型编程可以增加代码的灵活性和可重用性，因为泛型允许编写通用的算法和数据结构，适用于不同类型的数据；而不使用泛型则可能导致需要为每种数据类型编写特定的代码，增加了工作量和维护成本，同时可能降低代码的可读性和可维护性。

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？

在给出ADT的规约后立即编写测试用例的优势在于能够更加清晰地了解ADT的行为和功能，同时可以帮助确保实现的正确性和一致性。这种方式可以促使开发者更加专注于ADT的设计和实现，同时提前发现可能存在的问题，减少后期调试和修复的工作量。我可以适应这种测试方式，因为它有助于保证代码的质量和可靠性，同时提高了开发效率。

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？

不需要为每个应用场景重新设计和实现数据结构，节省了开发时间和资源；统一的ADT设计可以确保不同应用场景下的数据结构行为一致，降低了出错的可能性；维护一个通用的ADT实现比多个特定实现更容易，因为只需在一个地方进行修改和更新；经过验证的通用ADT实现已被广泛测试和使用，减少了引入新错误的风险；开发团队可以共享和重用相同的ADT实现，促进团队协作和知识共享。

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ADT撰写规约、不变式、表示不变式和抽象函数，以及避免表示暴露，有助于确保代码质量、防止数据结构不一致、隐藏实现细节和保护数据安全。这样做能提高代码可维护性和可读性，我愿意在以后的编程中坚持这种做法。

1. 关于本实验的工作量、难度、deadline。

工作量大；难度难；ddl还行，延期了。

1. 《软件构造》课程进展到目前，你对该课程有何收获和建议？

有很多收获。没有建议。