哈尔滨工业大学计算机科学与技术学院

2017 年秋季学期《软件工程》

Lab 7：OO 分析与设计

姓名	学号	联系方式
张宁	1150310607	3294349775@qq.com/17390605885
马玉坤	1150310618	mayukuner@126.com/18845895386

目 录

1 实验要求

简要复述实验手册中要求达到的实验目标与要求。

目标：针对 Lab6 之后的 Lab1 代码，使用 OO 思想对其进行重构。

过程：首先针对我们在实验一的用户需求,我们采用面向对象的编程思想来识别边界类控制类和实体类,然后设计每个类的方法,同时使用 StarUML 建立分析类图,领域类图和时序类图,根据我们建立的类图来重构我们的代码,同时完成回归测试,并将最终版本的代码提交到 github.

2 类识别

2.1 边界类

类名（中文）	类名（英文）	类的作用概述
Gui 部分	App	Gui 整体框架
画板基类	PanelBase	每一个功能画板的超类
生成新文本面板	PanelNewText	用来生成新文本的面板
打开文件面板	PanelOpenFile	打开文件的面板
打印图片面板	PanelPrint	打印图像的面板
查询桥接词面板	PanelQueryBridge	查询桥接词的面板
随机游走面板	PanelRandomAccess	随机游走算法的面板
存储图像面板	PanelSaveImage	存储图像到硬盘的面板
最短路径面板	PanelShortestPath	最短路求解模块的面板

2.2 控制类

类名（中文）	类名（英文）	类的作用概述
控制类的基类	CtrlBase	每个控制类的基类
生成新文本	CtrlNewText	用来生成新文本的控制类
打开文件	CtrlOpenFile	打开文件的控制类
打印图片	CtrlPrint	打印图像的控制类
查询桥接词	CtrlQueryBridge	查询桥接词的控制类
随机游走	CtrlRandomAccess	随机游走的控制类
最短路径	CtrlShortestPath	最短路求解的控制类

2.3 实体类

类名（中文）	类名（英文）	类的作用概述
边	WordEdge	存储图像中一条边的信息
节点	WordNode	存储图像中一个节点的信息
图	WordGraph	存储图的所有的边和节点的信息

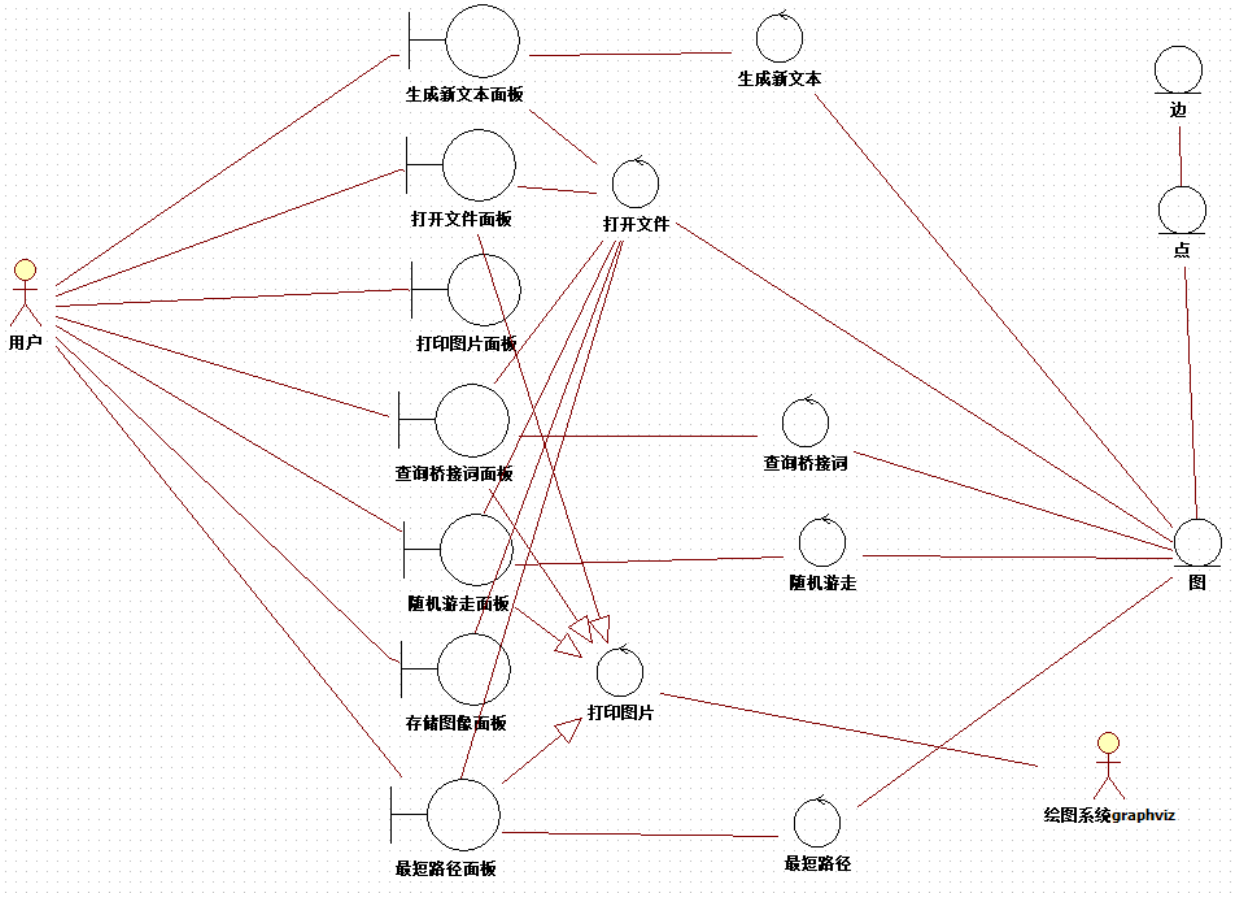
3 领域模型

3.1 分析类图

建立边界类、控制类、实体类之间的关联关系，使用 UML 类图形式描述（该图需要使用某种 UML 建模工具绘制）。

该分析类图中无需给出实体类的属性和方法。

注意：本图中出现的所有类，均应在第 2 部分的三个表格里有所定义；图中的类名均使用中文。

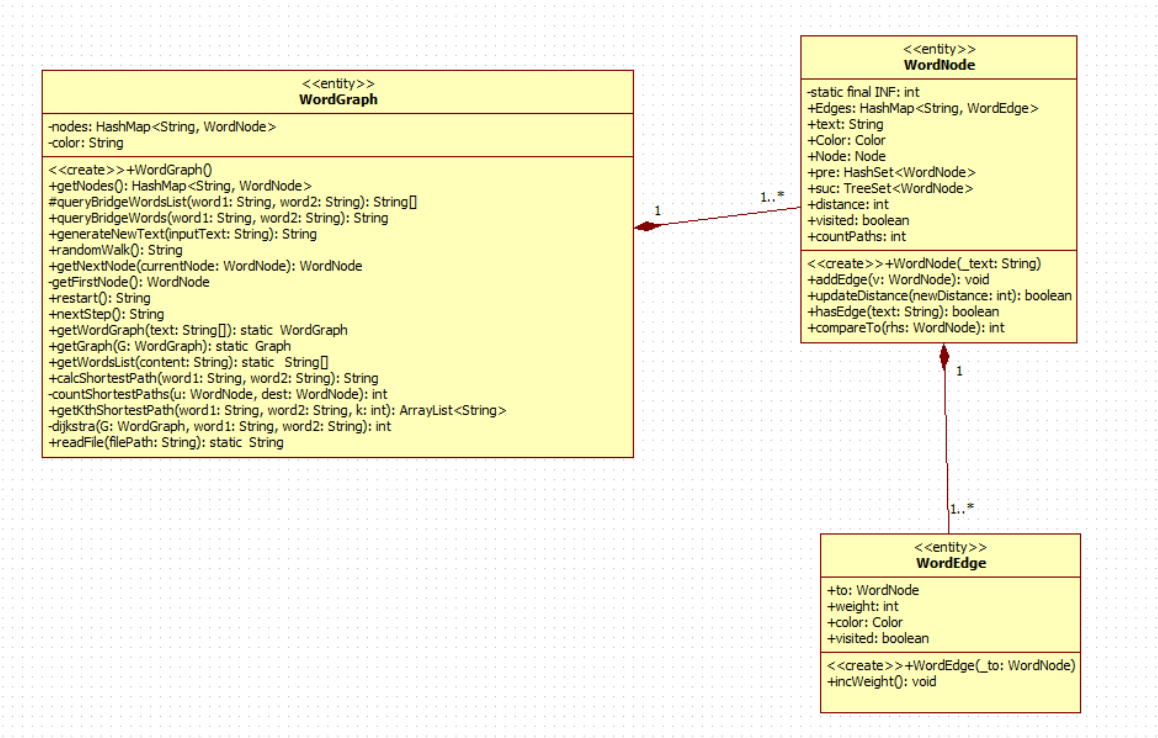


3.2 实体类图

仅针对实体类，建立 UML 类图，需详细刻画各类的全体属性集合（可见性、名称、数据类型）、全体操作集合（可见性、名称、返回值数据类型、参数列表）、类之间的关系（关联、组合、聚合、依赖、继承），以及关系的角色名、多重性、方向等信息。

该图需要使用某种 UML 建模工具绘制。

注意：本图中出现的所有实体类，均应在第 4.3 节的表格里有所定义；图中的类名、属性名、方法名、参数名均使用中文。



以下各小节，请给出实体类的详细设计，2.3 节表格里出现多少实体类，下面就应该有几个小节。撰写的时候，请将小节标题替换为实体类的中文名字+英文名字，例如：“有向图（DirectedGraph）”。

3.3 有向边（WordEdge）

属性定义

属性名（英文，与程序中的名字一致）	属性含义	数据类型	缺省值	可见性（public、private、protected）	类型（基本属性、关联属性、状态属性、派生属性）
to	记录有向边指向的节点	WordNode	无(类被创建的时候,必须给出指向的节点)	Public	关联属性
weight	weight 记录边上的权重	Int	1	Public	基本属性
color	color 记录边的颜色	Color	Color.***BLACK***	public	基本属性
visited	visited 记录边在随机游走过程之中是不是被访问过了	Boolean	False	public	状态属性

操作设计

操作名（英文，与程序中的名字一致）	操作含义与内部逻辑概述（即：该方法实现什么功能）	可见性	参数列表		返回值类型	类型（对属性 CRUD 的操作、状态更新操作、辅助操作）
			名称	数据类型		
WordEdge	初始化 WordEdge 对象	Public	_to	WordNode	无	辅助操作
incWeight	边权值自增（+1）	Public	无		无	对属性 CRUD 的操作

3.4 有向图节点（WordNode）

属性定义

属性名（英文，与程序中的名字一致）	属性含义	数据类型	缺省值	可见性（public、private、protected）	类型（基本属性、关联属性、状态属性、派生属性）
INF	表示整数的最大值	Int	Integer.MAX_VALUE	private	基本属性
edges	记录边上的信息	HashMap<String, WordEdge>	**new** HashMap<String, WordEdge>()	public	关联属性
text	text 记录节点的单词	String	对象实例化时给定	public	基本属性
color	color 记录节点的颜色	Color	Color.***WHITE***	public	基本属性
node	node 用来绘制图像	Node	null	public	关联属性
pre	记录节点在最短路计算中的前驱	HashSet <WordNode>	null	public	基本属性
suc	记录节点在最短路计算中的后继	TreeSet <WordNode>	null	public	基本属性
distance	记录节点在最短路计算中源点到该	int	null	public	基本属性

	点的距离				
visited	记录节点在最短路计算中是否被访问过	boolean	false	public	状态属性
countPaths	记录节点在最短路计算中到终点的不同的最短路径的数目	int	null	public	基本属性

操作设计

操作名（英文，与程序中的名字一致）	操作含义与内部逻辑概述（即：该方法实现什么功能）	可见性	参数列表		返回值类型	类型（对属性 CRUD 的操作、状态更新操作、辅助操作）
			名称	数据类型		
WordNode	初始化 WordNode 对象	public	_text	String	void	辅助操作
addEdge	添加一条连向 v 的边	Public	v	WordNode	void	对属性 CRUD 的操作
hasEdge	判断该点是否与 text 对应的节点连边	Public	text	String	boolean	辅助操作
compareTo	比较函数，在对节点进行排序时需要使用	Public	Rhs	WordNode	int	辅助操作
updateDistance	对该点的距离进行松弛，返回是否松弛成功	public	newDistance	int	boolean	对属性 CRUD 的操作

3.5 有向图（WordGraph）

属性定义

属性名（英文，与程序中的名字	属性含义	数据类型	缺省值	可见性（public、private、protected）	类型（基本属性、关联属性、状态属性、派生属性）

一致）					
nodes	图中的点	HashMap<String, WordNode>	**new** HashMap<String, WordNode>()	public	关联属性
color	记录图的颜色信息	String	“black”	public	基本属性

操作设计

操作名（英文，与程序中的名字一致）	操作含义与内部逻辑概述（即：该方法实现什么功能）	可见性	参数列表		返回值类型	类型（对属性 CRUD 的操作、状态更新操作、辅助操作）
			名称	数据类型		
WordGraph	构造函数	public	无		void	辅助操作
getNodes	获取有向图中的节点	public	无		HashMap<String, WordNode>	对属性 CRUD 的操作
getGraph	通过有向图生成 Graphviz 所用的图像，用于 GUI 显示	public	WG	WordGraph	Graph	辅助操作
getWordGraph	通过单词列表生成单词构成的有向图	public	text	String[]	WordGraph	辅助操作
getWordsList	对一段文本进行修改，生成单词列表	public	content	String	String[]	辅助操作
readFile	从文本文件中读取文本	public	filePath	String	String	辅助操作
calcShortestPath	计算最短路径	public	word1	String	String	对属性 CRUD 的操作
			word2	String		
countShortestPaths	计算从 u 到 dest 的最短路径个数	public	u	WordNode	int	对属性 CRUD 的操作
			dest	WordNode		
dijkstra	计算 G 中从 word1 到 word2 的最短路	private	G	WordGraph	int	对属性 CRUD 的操作
			word1	String		
			word2	String		

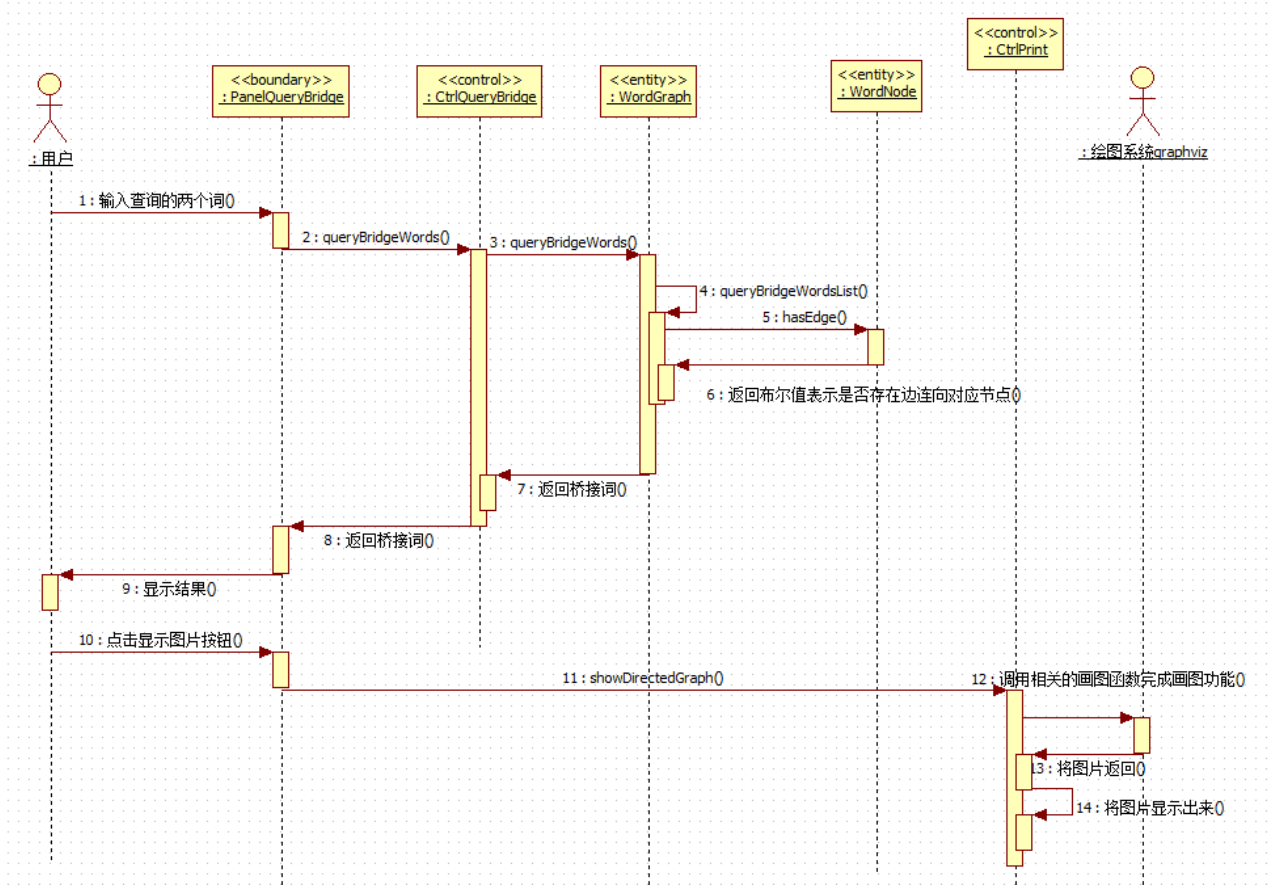
generateNewText	通过桥接词生成新文本	public	inputText	String	String	对属性CRUD的操作
getFirstNode	在有向图中随机获取一个结点作为随机游走的初始节点	public	无	无	WordNode	对属性CRUD的操作
getKthShortestPath	计算字典序第 k 小的最短路径	public	word1 word2 k	String String int	ArrayList<String>	对属性CRUD的操作
getNextNode	从当前节点随机游走到相邻的节点	public	currentNode	WordNode	WordNode	对属性CRUD的操作
nextStep	从当前节点随机游走到相邻的节点	public	无	无	void	对属性CRUD的操作
queryBridgeWords	查询两个单词的桥接词	public	word1 word2	String String	String	对属性CRUD的操作
queryBridgeWordsList	查询两个单词的桥接词列表	public	word1 word2	String String	String[]	对属性CRUD的操作
randomWalk	随机游走函数	public	无	无	String	对属性CRUD的操作
restart	重新选择节点开始随机游走	public	无	无	String	对属性CRUD的操作

4 时序模型

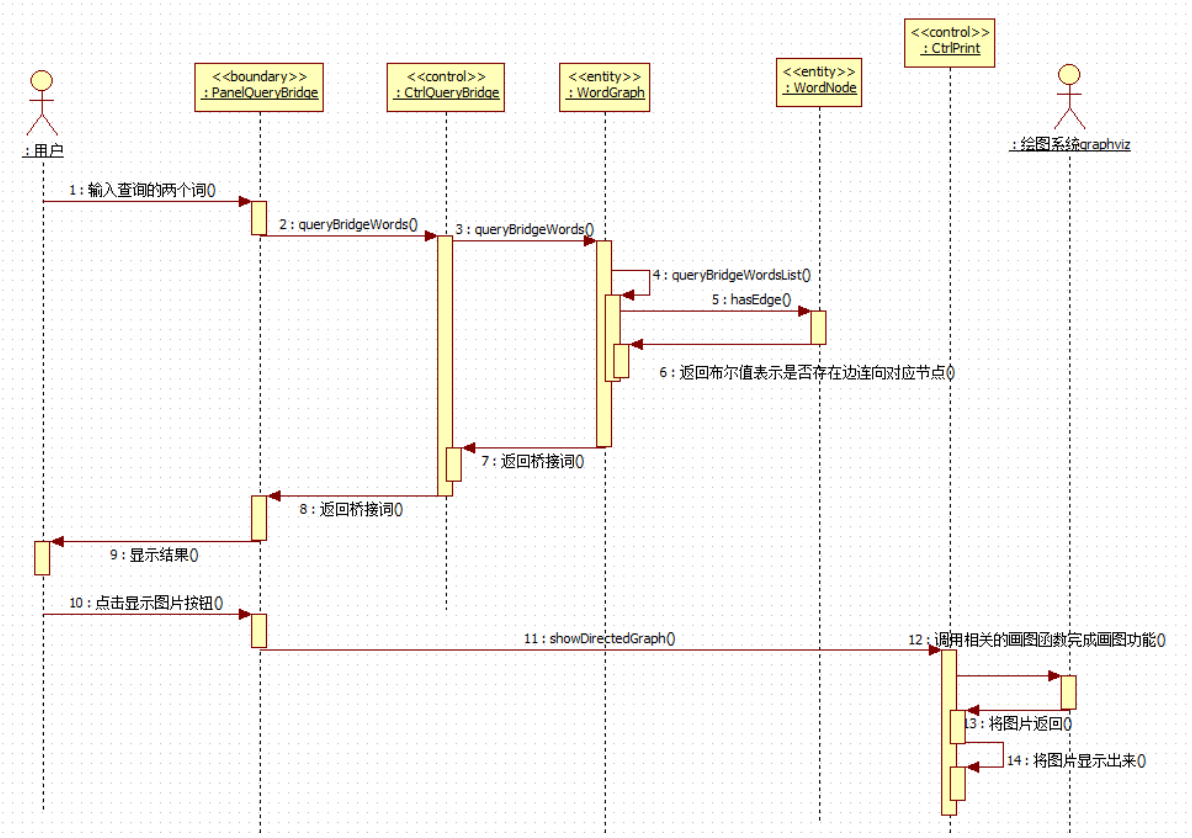
针对代码中的核心功能（生成有向图、查询桥接词、根据桥接词生成新文本、计算最短路径、随机游走），分别建立其时序模型。该模型中包含的所有对象必须在第 4 部分中被识别出来，包含的所有操作必须包含在第 3 节的类图和表格中。

该图需要使用某种 UML 建模工具绘制。

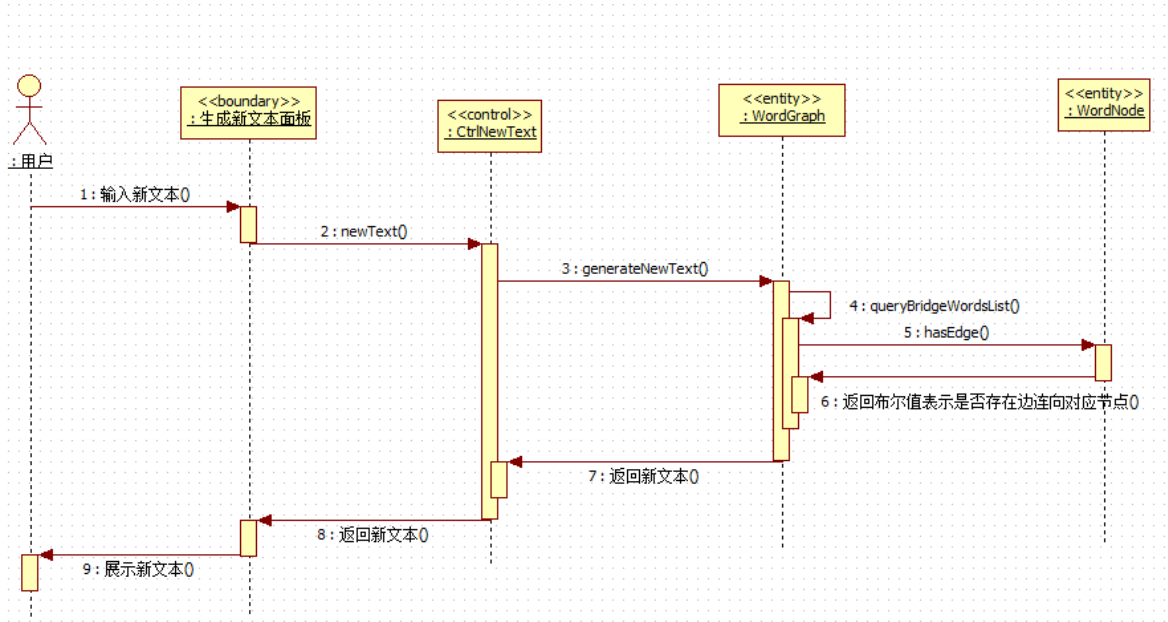
4.1 生成有向图


:用户
<<boundary>>
:PanelQueryBridge
<<control>>
:CtrlQueryBridge
<<entity>>
:WordGraph
<<entity>>
:WordNode
<<control>>
:CtrlPrint
:绘图系统graphviz
1:输入查询的两个词()
2:queryBridgeWords()
3:queryBridgeWords()
4:queryBridgeWordsList()
5:hasEdge()
6:返回布尔值表示是否存在边连向对应节点()
7:返回桥接词()
8:返回桥接词()
9:显示结果()
10:点击显示图片按钮()
11:showDirectedGraph()
12:调用相关的画图函数完成画图功能()
13:将图片返回()
14:将图片显示出来()

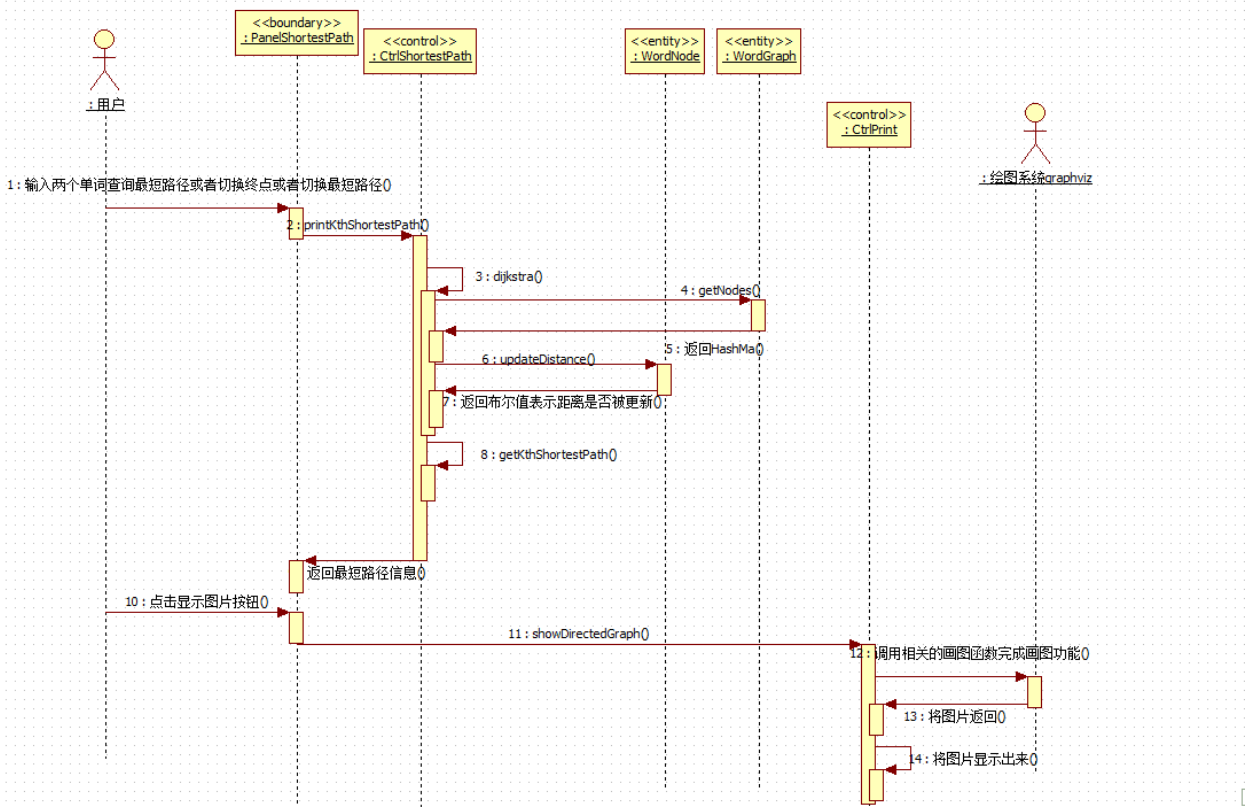
4.2 查询桥接词



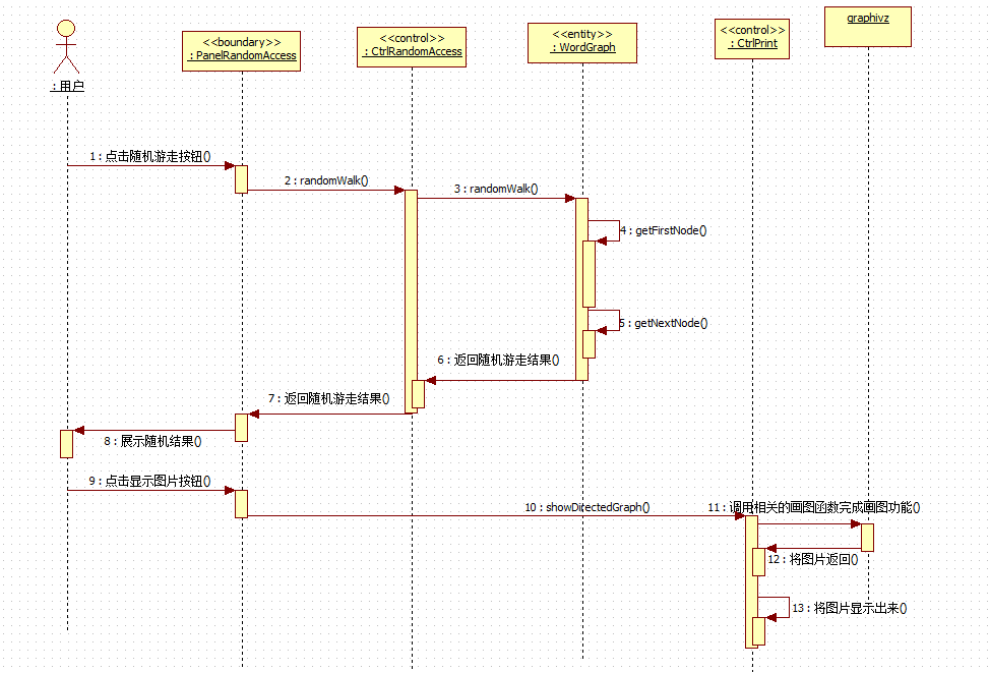
4.3 根据桥接词生成新文本



4.4 计算最短路径



4.5 随机游走



5 根据 OO 模型生成的类代码框架

使用 UML 建模工具，基于 3.2 节建立的实体类图，生成相应的类代码框架。以实体类为小节，贴出其相应的代码框架。

请将小节标题替换为实体类的中文名字+英文名字，例如：“有向图（DirectedGraph）”。

5.1 有向图（WordGraph）

```
public class WordGraph {
    private HashMap<String, WordNode> nodes;
    private String color;
    public void WordGraph() {

    }

    public HashMap<String, WordNode> getNodes() {

    }

    protected String[] queryBridgeWordsList(String word1, String word2) {

    }

    public String queryBridgeWords(String word1, String word2) {

    }

    public String generateNewText(String inputText) {

    }

    public String randomWalk() {

    }

    public WordNode getNextNode(WordNode currentNode) {

    }

    private WordNode getFirstNode() {
```

```
    }

    public String restart() {

    }

    public String nextStep() {

    }

    public static  WordGraph getWordGraph(String[] text) {

    }

    public static  Graph getGraph(WordGraph G) {

    }

    public static   String[] getWordsList(String content) {

    }

    public String calcShortestPath(String word1, String word2) {

    }

    private int countShortestPaths(WordNode u, WordNode dest) {

    }

    public ArrayList<String> getKthShortestPath(String word1, String word2, int k)
{

    }

    private int dijkstra(WordGraph G, String word1, String word2) {

    }

    public static  String readFile(String filePath) {

    }
}
```

5.2 有向边（WordEdge）

```
public class WordEdge {
    public WordNode to;
    public int weight;
    public Color color;
    public boolean visited;
    public void WordEdge(WordNode _to) {

    }

    public void incWeight() {

    }

}
```

5.3 有向图节点（WordNode）

```
public class WordNode {
    private int static final INF;
    public HashMap<String, WordEdge> Edges;
    public String text;
    public Color Color;
    public Node Node;
    public HashSet<WordNode> pre;
    public TreeSet<WordNode> suc;
    public int distance;
    public boolean visited;
    public int countPaths;
    public void WordNode(String _text) {

    }

    public void addEdge(WordNode v) {

    }

    public boolean hasEdge(String text) {

    }
```

```
        public int compareTo(WordNode rhs) {

        }

        public boolean updateDistance(int newDistance) {

        }
    }
```

6 对 Lab6 代码的重构

阐述你们的重构工作，即：相比 Lab6 中的代码，基于 OO 重新设计和实现的代码有哪些大的变化？逐项列出变化，并解释说明，必要时可给出代码来帮助佐证和理解。

最初我们采用了 OO 的思想来设计代码，但是当时并不知道什么边界类、控制类和实体类，我们把页面的 GUI 部分和本应该在控制类里面写的部分写在了一起，导致我们没有控制类，但是我们当时识别出了实体类,所以现在我们针对代码的主要修改就是把之前和界面耦合在一起的那些函数，抽象成类，形成一个高内聚低耦合的代码架构。

变化	说明	例子	
		修改前	修改后
新增控制类	Lab6 中的代码无控制类，边界类和实体类直接交互，耦合程度高。在 Lab7 中，我们对代码进行了重构，为每个用户故事创建了对应的控制类，并使用控制类连接边界类和实体类。	PanelNewText.java 中的 `private static String generateNewText(WordGraph G, String inputText)`方法用于生成新文本	新建了CtrlNewText控制类，其内部的`public static String newText(WordGraph G, String inputText)`方法用于边界类PanelNewText调用
将边界类中的大部分方法移到了实体类中	Lab6 中，边界类里有过多的方法，并与实体类耦合程度较高。在 Lab7 中，我们将大部分边界类中的方法移到了实体类中。	PanelNewText.java 中的 `private static String generateNewText(WordGraph G, String inputText)`方法用于生成新文本	在WordGraph类中实现了 `public String generateNewText(String inputText)`方法
新增了控制类的基类	将所有控制类的公共属性定义到了基类中，并设置为 static 方法	无控制类基类	控制类基类包含 `protected static String[] initialWords` 和 `public static WordGraph`

			`getInitialWordGraph()`
修改了边界类的基类	在 Lab6 中，我们将 PanelApp 重命名为 PanelBase。此外，边界类的基类 PanelApp 类过于冗余，含有过多方法，我们在 Lab7 中将这些方法移到了实体类 WordGraph 中。	PanelApp 类包含 getWordGraph，getGraph，setLookAndFeel，queryBridgeWordsList 四个成员方法。	PanelBase 类仅包含 myFont 属性，用于自定义各个面板的默认字体。

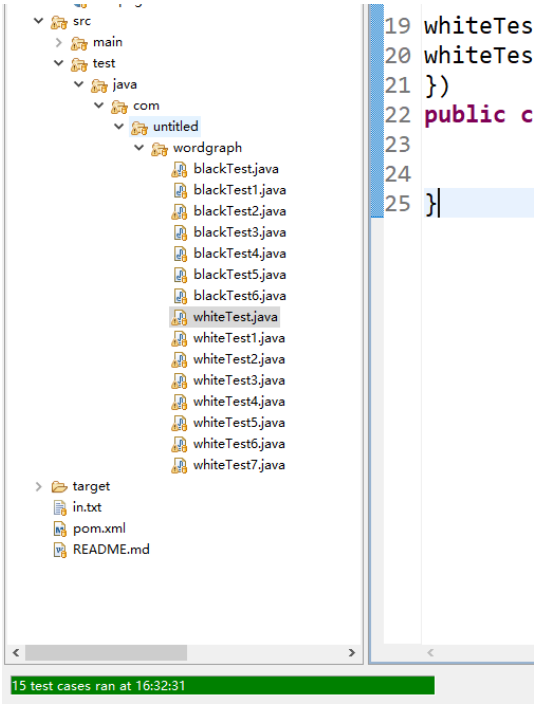
7 重构之后的回归测试

基于 Lab6 中设计和实现的各黑盒/白盒测试用例，进行回归测试，发现重构代码中存在的问题，并加以修复。

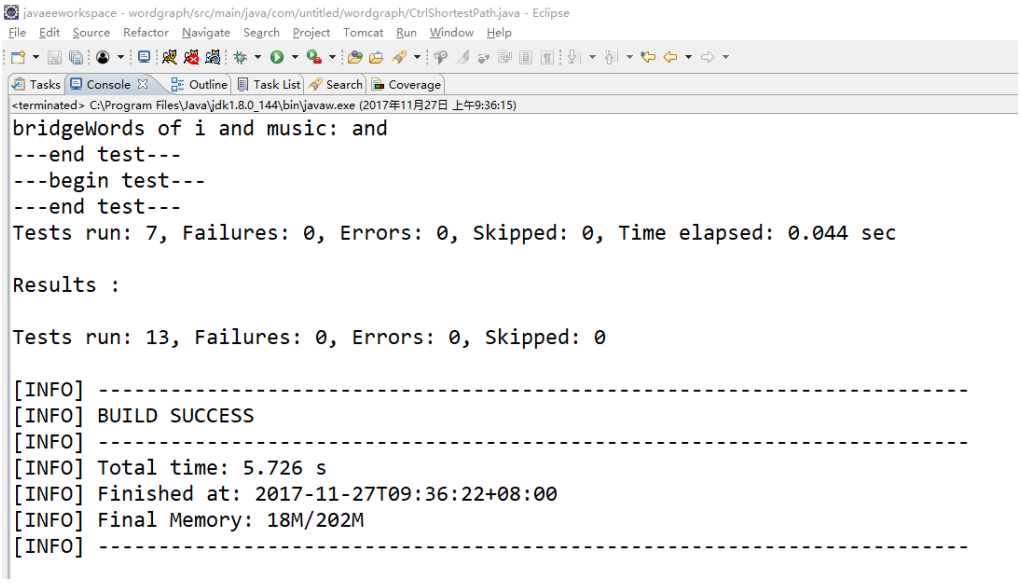
给出上述过程的截图作为佐证。

我们在针对代码重构之后,进行了白盒以及黑盒测试,测试结果显示我们的代码并没有产生错误.

下面是我们的持续集成工具 Infinitest 显示的结果（左下角）：

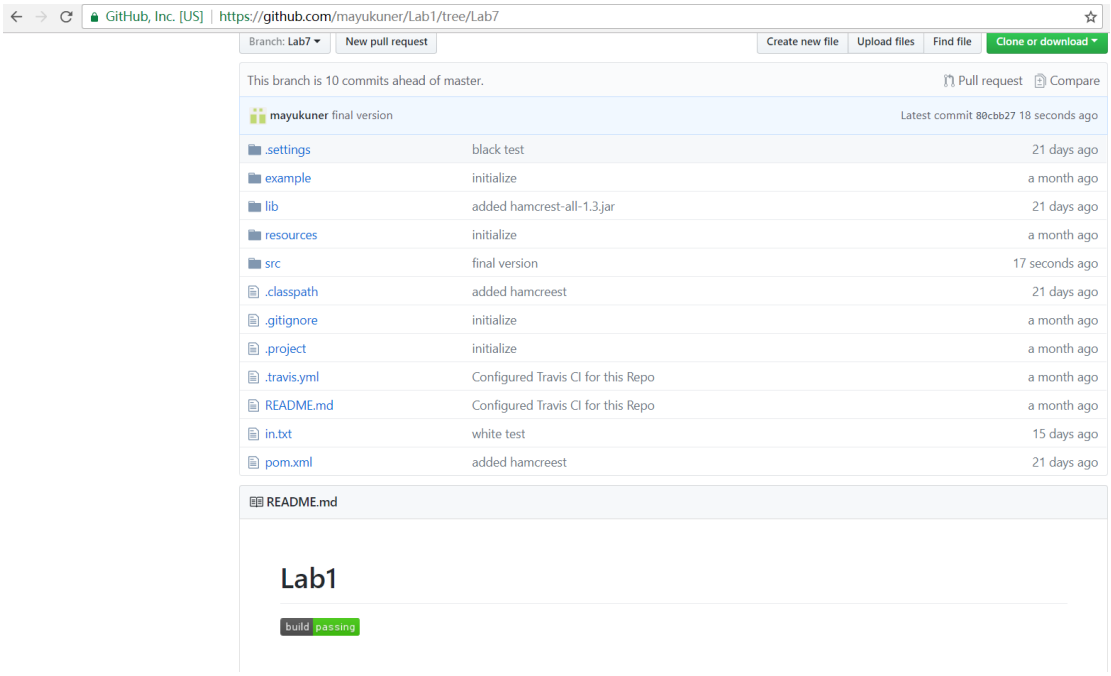


下面是我们的 maven test 之后的结果，如下图所示，13 个测试用例均已通过:



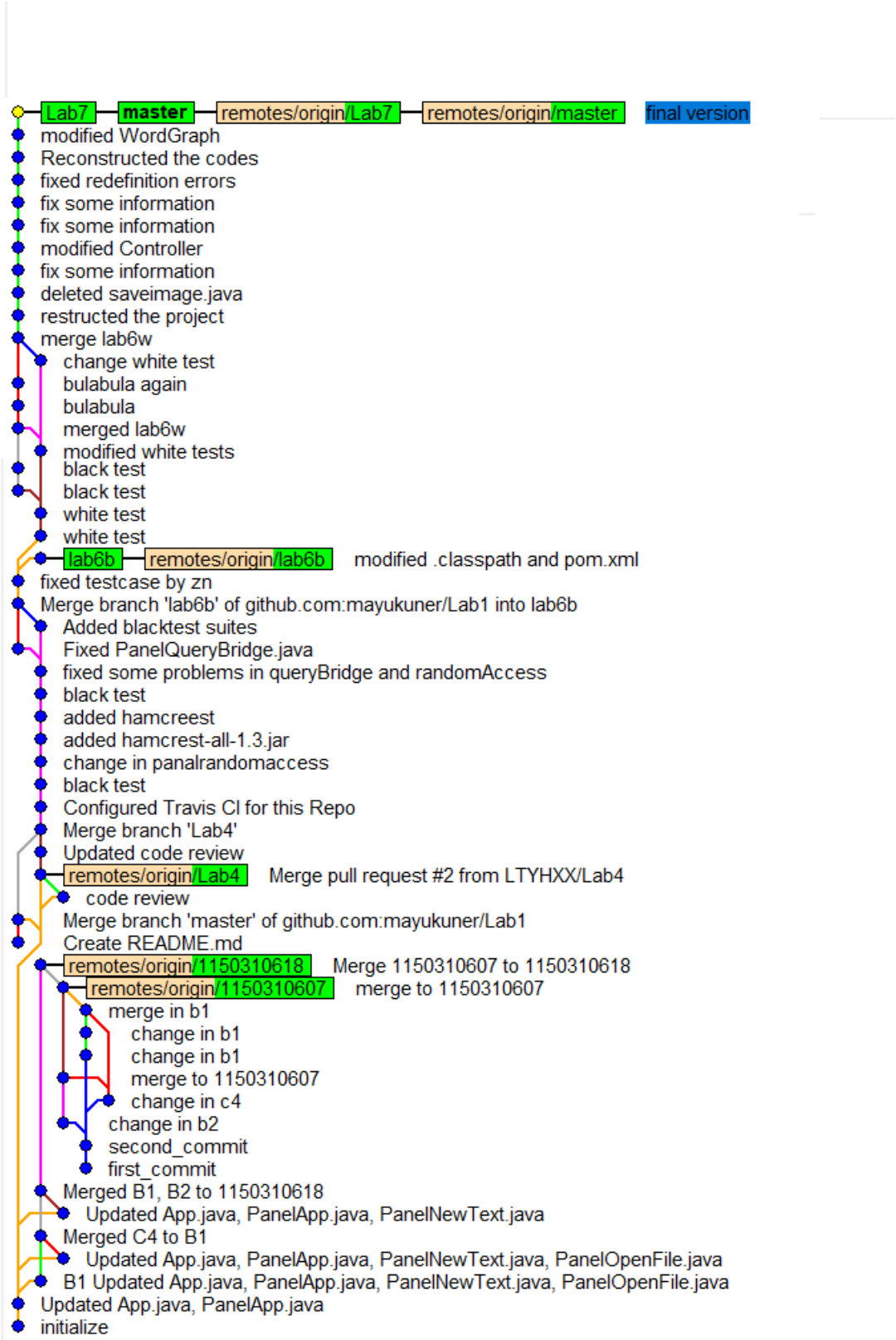
8 测试之后的 Git 提交

在 Git 仓库内建立 Lab7 分支，提交重构后的代码，push 到 GitHub 仓库。
给出本组 GitHub 的截图，证实你已成上述提交任务。



最后，给出本组 GitHub 上从 Lab1 开始、历经 Lab3、Lab4、Lab6、Lab7 之后，完整的提交历史（可以使用 GitHub 项目页面里“Graphs”下的“Network”，也可以使用 Git 指令获取提交历史的有向图）。

下图为在 Git GUI 中使用”Visualize All Branch History”功能所绘制的提交历史的有向图。


Lab7
master
remotes/origin/Lab7
remotes/origin/master
final version
modified WordGraph
Reconstructed the codes
fixed redefinition errors
fix some information
fix some information
modified Controller
fix some information
deleted saveimage.java
restructed the project
merge lab6w
change white test
bulabula again
bulabula
merged lab6w
modified white tests
black test
black test
white test
white test
lab6b
remotes/origin/lab6b
modified .classpath and pom.xml
fixed testcase by zn
Merge branch 'lab6b' of github.com:mayukuner/Lab1 into lab6b
Added blacktest suites
Fixed PanelQueryBridge.java
fixed some problems in queryBridge and randomAccess
black test
added hamcreest
added hamcrest-all-1.3.jar
change in panalrandomaccess
black test
Configured Travis CI for this Repo
Merge branch 'Lab4'
Updated code review
remotes/origin/Lab4
Merge pull request #2 from LTYHXX/Lab4
code review
Merge branch 'master' of github.com:mayukuner/Lab1
Create README.md
remotes/origin/1150310618
Merge 1150310607 to 1150310618
remotes/origin/1150310607
merge to 1150310607
merge in b1
change in b1
change in b1
merge to 1150310607
change in c4
change in b2
second_commit
first_commit
Merged B1, B2 to 1150310618
Updated App.java, PanelApp.java, PanelNewText.java
Merged C4 to B1
Updated App.java, PanelApp.java, PanelNewText.java, PanelOpenFile.java
B1 Updated App.java, PanelApp.java, PanelNewText.java, PanelOpenFile.java
Updated App.java, PanelApp.java
initialize

9 计划与实际进度

任务说明	计划时间长度（分钟）	实际耗费时间（分钟）	提前或延期的原因分析
画出三种图	100	200	由于我们之前已经写好了代码,我们并不希望我们的设计会导致源代码的大量修改 (虽然这样做和老师的初衷不一样),所以我们的设计过程既要考虑到 OO 的设计思想,又要以我们的源代码为基础,挺麻烦的.而且我们在实验 1 的时候实现了很多的附加功能,函数调用关系比较复杂,所以画时序图的时候挺麻烦的.
修改代码	100	50	我们设计完领域类图之后,我们的代码并没有很多的修改,所以修改的时间并不多
通过测试	20	10	我们的代码最初设计就是 OO,所以代码修改和重新测试都没有很难,轻松就通过了测试.
写报告	300	400	报告实在是太麻烦了,每一次都很难在预定的时间之内,写完如此冗长的报告,不过这好像是最后一次了吧.

10 小结

(1) 建模是否有必要？直接去写代码呗？边写代码边思考，岂不是省了建模的大量时间？

如果程序很简单,对于程序员来说,确实可以直接写代码,效率很高.

但是如果程序员不能依靠自己的智力直接处理的时候(但是人们并不总是有自知之明),就需要使用一些建模工具来帮助自己理清思路.否则极有可能出现自己在开发过程之中,发现自己思维上的一些错误,导致浪费了大量的时间和精力.

(2) 建模时如果能思考得很细节，其实就相当于写代码时的思考工作。如何能逼着自己在建模时想得更细节，你是否总结出什么方法？

可以使用小黄鸭工作法,程序员对着一只小黄鸭或者对着同伴,然后针对它/他,阐述自己的设计思路。实践证明，我们往往能够在这样的一个过程之中,发现自己设计之中的不合理之处。

此外，多尝试，多讨论，在讨论中发现问题也是一个非常好用的方法。

(3) “面向对象”的分析与设计方法，与传统的以“算法+数据结构”为单位的结构化

分析方法相比有什么好的地方和不好的地方？对比你重构前和重构后的代码，你认为这种重构的价值何在？

好的地方显而易见,可以提高不同开发人员间的协作效率，加快整个工程的开发速度。可以使得我们能够做出更大更健壮的软件。

不好的地方也有,就是面向对象的编程语言,对象这个概念太深刻了,数据封装带来了数据交互上不方便.

重构的价值在于我们的软件变得更加的高内聚/低耦合了,我们的软件开发采用了这样的方法之后,变得更加的健壮,出错之后,我们可以很容易的找到错误的原因。通过对工程的合理划分，降低不同模块间的耦合程度，可以提高我们之间协作的效率。

(4) 重构之后，你的代码是否引入了新的错误？为了规避重构过程中引入的新问题，你认为理想的重构过程应该如何做？

没有引入新的错误,非常幸运,我们的代码在重构之后,没有产生任何 bug,顺利运行.

我们在重构之前,应该设计好相应的 uml 图形,把分析类图,领域类图和时序图画出来,确保函数调用关系不出错,保证我们的代码具备了高内聚/低耦合的特征之后,才能开始对代码进行重构.

在重构的过程之中,使用持续集成工具,这样的话,一旦发现程序存在问题,那么我们就可以立即发现,将错误消除.

重构之后,对我们的程序进行一次完整的测试.

(5) 其他想法。

Staruml 这个软件我非常喜欢,软件操作起来比较舒服,但是还是有一些瑕疵,比如界面美观性就会差一点,还有一点就是并不是我们一开始就会使用的,如果这样的软件可以轻松的找到使用文档,那就再好不过了.

最初我们想的是使用一个 markdown 插件来画我们的图,具体的过程就和下面的图一样,通过一些简单的代码就可以描述时序图,但是后来用了 staruml 这个软件觉得还是算了,那样做纯粹是自己给自己找麻烦,staruml 更加的简单容易,之前我在画程序流程图的时候选择了使用 markdown 插件来画,因为我觉得线性的代码变得会使得修改起来变得非常简单,但是现在既然这个软件这么好用,就算了.

感觉这俩东西挺像 office 和 latex 的,一个所见即所得,一个通过编译源代码来生成最终的 pdf 文件,面对这两种方式,我们不必执着于情怀一定要选择 latex,好用就行,选择适合自己适合这个任务的工具.

```
sequenceDiagram
    Alice->John: Hello John, how are you?
    loop Every minute
        John-->Alice: Great!
    end
```