



哈爾濱工業大學  
HARBIN INSTITUTE OF TECHNOLOGY

# 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	张景润
学号	1172510217
班号	1703002
电子邮件	2584363094@qq.com
手机号码	18530272728

## 目录

1 实验目标概述	1
2 实验环境配置	1
2.1 配置 EclEmma	1
2.2 Lab2 仓库地址	2
3 实验过程	2
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>	5
3.1.3.1 Implement ConcreteEdgesGraph	5
3.1.3.2 Implement ConcreteVerticesGraph	6
3.1.4 Problem 3: Implement generic Graph<L>	8
3.1.4.1 Make the implementations generic	8
3.1.4.2 Implement Graph.empty()	9
3.1.5 Problem 4: Poetic walks	9
3.1.5.1 Test GraphPoet	9
3.1.5.2 Implement GraphPoet	9
3.1.5.3 Graph poetry slam	10
3.1.6 Before you're done	10
3.2 Re-implement the Social Network in Lab1	11
3.2.1 FriendshipGraph 类	11
3.2.2 Person 类	12
3.2.3 客户端 main()	12
3.2.4 测试用例	13
3.2.5 提交至 Git 仓库	13
3.3 Playing Chess	14
3.3.1 ADT 设计/实现方案	14
3.3.1.1 Action 类	14
3.3.1.2 Board 类	14
3.3.1.3 Game 类	15
3.3.1.4 MyChessAndGoGame	16
3.3.1.5 Piece 类	17

3.3.1.6 Player 类	18
3.3.1.7 Position 类	18
3.3.2 主程序 MyChessAndGoGame 设计/实现方案	18
3.3.3 ADT 和主程序的测试方案	22
3.4 Multi-Startup Set (MIT)	24
4 实验进度记录	24
5 实验过程中遇到的困难与解决途径	25
6 实验过程中收获的经验、教训、感想	25
6.1 实验过程中收获的经验教训	25
6.2 针对以下方面的感受	25

# 1 实验目标概述

根据实验手册简要撰写。

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：

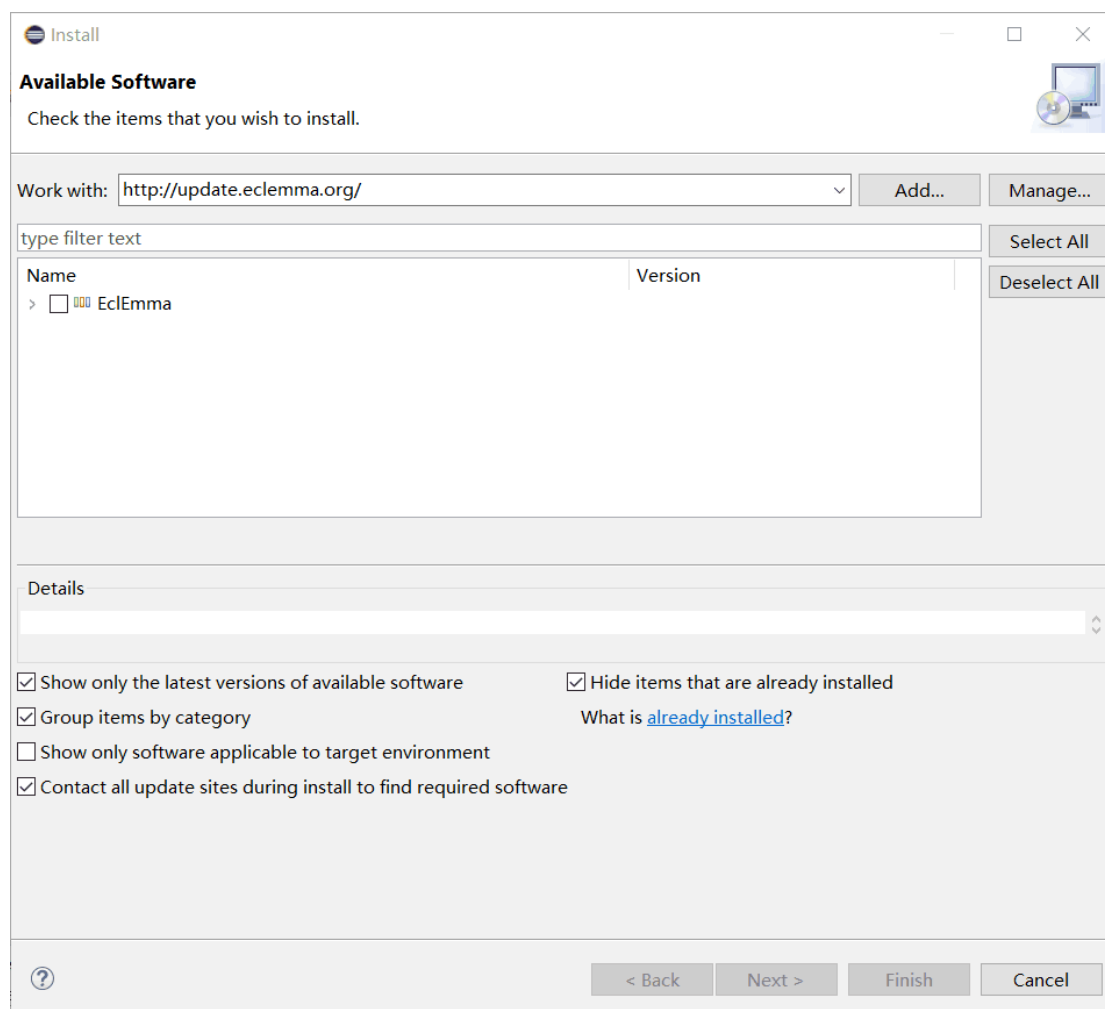
- 针对给定的应用问题，从问题描述中识别所需的 ADT；
- 设计 ADT 规约（pre-condition、post-condition）并评估规约的质量；
- 根据 ADT 的规约设计测试用例；
- ADT 的泛型化；
- 根据规约设计 ADT 的多种不同的实现；针对每种实现，设计其表示（representation）、表示不变性（rep invariant）、抽象过程（abstraction function）
- 使用 OOP 实现 ADT，并判定表示不变性是否违反、各实现是否存在表示泄露（rep exposure）；
- 测试 ADT 的实现并评估测试的覆盖度；
- 使用 ADT 及其实现，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

### 2.1 配置 EclEmma

依据 <https://www.eclemma.org/installation.html> 内容，从更新站点进行安装。

- 从 Eclipse 菜单中选择帮助 → 安装新软件；
- 在“安装”对话框中，在“工作日期”字段中输入 <http://update.eclemma.org/>；



- 检查最新的 EclEmma 版本，然后按“下一步”；
- 重启 eclipse，即可在 java 的透视图工具栏中找到 coverage 启动器，表示安装成功。



## 2.2 Lab2 仓库地址

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

<https://github.com/ComputerScienceHIT/Lab2-1172510217>

## 3 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！）。

### 3.1 Poetic Walks

在这里简要概述你对该任务的理解。

- 完善 Graph 接口类，并运用泛型的思想，将 String 拓展为泛型 L 类；
- 实现 Graph 类的方法：add、set、remove、vertices、sources、targets；
- 利用实现的 Graph 类，应用图的思想，实现 GraphPoet 类，如果输入的文本的两个单词之间存在桥接词，则插入该桥接词；若存在多个单一桥接词，则选取边权重较大者。

Finished after 0.068 seconds

Runs: 12/12    Errors: 0    Failures: 0

✓ ConcreteEdgesGraphTest [Runner: JUnit 5] (0.004 s)

- ✓ testEmptyGraph (0.000 s)
- ✓ testGet (0.000 s)
- ✓ testToString (0.000 s)
- ✓ testEdgeToString (0.000 s)
- ✓ testTargets (0.000 s)
- ✓ testAdd (0.004 s)
- ✓ testSet (0.000 s)
- ✓ testVertices (0.000 s)
- ✓ testInitialVerticesEmpty (0.000 s)
- ✓ testAssertionsEnabled (0.000 s)
- ✓ testRemove (0.000 s)
- ✓ testSources (0.000 s)

Finished after 0.047 seconds

Runs: 12/12    Errors: 0    Failures: 0

✓ ConcreteVerticesGraphTest [Runner: JUnit 5] (0.000 s)

- ✓ testAddEdge (0.000 s)
- ✓ testGet (0.000 s)
- ✓ testToString (0.000 s)
- ✓ testVertexToString (0.000 s)
- ✓ testTargets (0.000 s)
- ✓ testAdd (0.000 s)
- ✓ testSet (0.000 s)
- ✓ testVertices (0.000 s)
- ✓ testInitialVerticesEmpty (0.000 s)
- ✓ testAssertionsEnabled (0.000 s)
- ✓ testRemove (0.000 s)
- ✓ testSources (0.000 s)

### 3.1.1 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。

- 获取代码：下载自

[https://github.com/rainywang/Spring2019\\_HITCS\\_SC\\_Lab2/tree/master/P1](https://github.com/rainywang/Spring2019_HITCS_SC_Lab2/tree/master/P1)

- 在本地创建 git 仓库

1, 利用 Git Bash 输入如下命令行

`git clone https://github.com/ComputerScienceHIT/Lab2-1172510217.git`

2, 查看远程仓库的信息。git remote -v



```
MINGW64:/e/Lab2-1172510217

86185@LAPTOP-G3400N5D MINGW64 /e/Lab2-1172510217 (master)
$ git remote -v
origin https://github.com/ComputerScienceHIT/Lab2-1172510217.git (fetch)
origin https://github.com/ComputerScienceHIT/Lab2-1172510217.git (push)

86185@LAPTOP-G3400N5D MINGW64 /e/Lab2-1172510217 (master)
$
```

### 3.1.2 Problem 1: Test Graph <String>

分别测试 add、set、remove、vertices、sources、targets 方法

关于 add 方法: Testing strategy graph: 空+非空 vertex: 待加边不存在+待加边存在 结果: 若返回值为 true, 顶点数目++且顶点集中包含 vertex; 否则, 顶点集不变且顶点集中包含 vertex number of vertices increases by 1 else graph unmodified observe with vertices();

关于 set 方法:

```
// Testing strategy
// graph: 空+非空
// vertex: 待设置顶点已经存在+不存在
// weight: 为0时, 删除; 非零, 修改
```

// 返回值：为0，边不存在；否则，边存在，且返回值是原来的权重

关于remove方法：

// Testing strategy

// graph: 空+非空

// vertex: 待删顶点存在于顶点集中+不存在；待删顶点存在于边中+不存在

// 结果：若返回值为false，不存在该顶点，顶点集数目不变，边集不变；否则，在顶点集或者边集存在该顶点，顶点集数目

关于Vertices方法：

// Testing strategy

// graph: 空+非空

关于Sources方法：

// Testing strategy

// graph: 空+非空

// vertex: 待寻找顶点存在于顶点集中+不存在；

// edges: 待寻找顶点存在入边+不存在

// 结果：返回的map含有所有的入边

关于Targets方法：

// Testing strategy

// graph: 空+非空

// vertex: 待寻找顶点存在于顶点集中+不存在；

// edges: 待寻找顶点存在出边+不存在

// 结果：返回的map含有所有的出边

最后完善ConcreteEdgesGraphTest类以及ConcreteVerticesGraphTest类

### 3.1.3 Problem 2: Implement Graph <String>

#### 3.1.3.1 Implement ConcreteEdgesGraph

设计 1：表示不变量

// Abstraction function:

// AF(vertices, edges) = weighted graph with directed edges and vertices are

// different

// Representation invariant:

// All the weight of the edges > 0

// Safety from rep exposure:

// fields are private and final in order to keep the codes safe

// vertices is mutable, so vertices() make defensive copy to avoid rep exposure

// edges is immutable

设计 2：检查表示

```
private void checkRep() {
```

```
    assert vertices != null; // 检查是否顶点集为空
```



```

    for (L l : vertices) { // 检查是否加入了null顶点对象
        assert l != null;
    }
    for (Edge<L> edge : edges) { // 检查是否存在边权值不合格
        assert edge != null;
        assert edge.getWeight() > 0;
        assert vertices.contains(edge.getSource()); // 检查是否有野边
        assert vertices.contains(edge.getTarget()); // 检查是否有野边
    }
}

```

实现 Edge<String>类方法

public String getSource	得到入点
public String getTarget	得到出点
public Integer getWeight	得到权重
public String toString	类的 toString 表示

```

// Abstraction function:
// AF(source,target,weight) = an edge which is from source to target with a
// positive weight
// Representation invariant:
// weight > 0
// Safety from rep exposure:
// fields are private and final in order to keep the codes safe
// checkRep: check the rep invariant

```

设计 4：实现 ConcreteEdgesGraph<String>类属性

```

private final Set<L> vertices = new HashSet<>();
private final List<Edge<L>> edges = new ArrayList<>();

```

实现 ConcreteEdgesGraph<String>类方法

public Boolean add	调用 vertices 的 add 方法
public int set	分情况判断：边存在、边不存在；weight 为 0，weight 不为 0
public Boolean remove	分情况判断：待删顶点存在于顶点集中、不存在；存在于边集中、不存在
public Set<String> vertices	for 循环遍历，进行防御性复制
public Map<String,Integer> sources	遍历所有的边，判断是否含有出点
public Map<String,Integer> targets	遍历所有的边，判断是否含有入点
public(Override) String toString	重写该类的 toString 方法

### 3.1.3.2 Implement ConcreteVerticesGraph

设计 1：表示不变量

```

// Abstraction function:
// AF(vertices) = a directed weighted graph which has a map of its targets
and

```

```
// weight
// Representation invariant:
// all the weight of the edges > 0
// Safety from rep exposure:
// vertices field is private and final;
// vertices is mutable, so vertices() make defensive copy to avoid rep
// exposure.
```

设计 2: 检查表示

```
private void checkRep() {
    for (Vertex<L> vertex : vertices) {
        for (Integer weight : vertex.getRelationMap().values()) {
            assert weight > 0;
        }
    }
}
```

设计 3: 实现 Vertex<String>类属性

private String source	边的入点
private Map<L, Integer> relationMap = new HashMap<>();	入点的所有出点集合以及边的权重

```
// Abstraction function:
// a vertex with L and a map that record all the edges from it
// Representation invariant:
// all the weight > 0
// Safety from rep exposure:
// L is named final which is safe;
```

```
// the map is returned by copying a new map.
```

实现 Vertex<String> 方法

public Vertex	
public String getSource	得到入点
public void addEdge	在出边集中加一个顶点
public void deleteEdge	删除一条边
public Map<L, Integer> getRelationMap	得到所有的出边
public int getWeight	得到权重
public String toString	类的 toString 表示

设计4：实现 ConcreteVerticesGraph <String> 类属性

```
private final List<Vertex<L>> vertices = new ArrayList<>();
// Abstraction function:
// AF(vertices) = a directed weighted graph which has a map of its targets
and
// weight
// Representation invariant:
// all the weight of the edges > 0
// Safety from rep exposure:
// vertices field is private and final;
// vertices is mutable, so vertices() make defensive copy to avoid rep
exposure.
```

实现 ConcreteVerticesGraph <String> 类方法

public Boolean add	调用 vertices 的 add 方法
public int set	分情况判断：边存在、边不存在；weight 为 0，weight 不为 0
public Boolean remove	分情况判断：待删顶点存在于顶点集中、不存在；存在于边集中、不存在
public Set<String> vertices	for 循环遍历，进行防御性复制
public Map<String,Integer> sources	遍历所有的边，判断是否含有出点
public Map<String,Integer> targets	遍历所有的边，判断是否含有入点
public(Override) String toString	重写该类的 toString 方法

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

#### 3.1.4.1 Make the implementations generic

- 将具体类的声明更改为：

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

- 更新两个实现以支持任何类型的顶点标签，使用占位符 L 代替 String（如传入参数、返

回值等)。

### 3.1.4.2 Implement Graph.empty()

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

## 3.1.5 Problem 4: Poetic walks

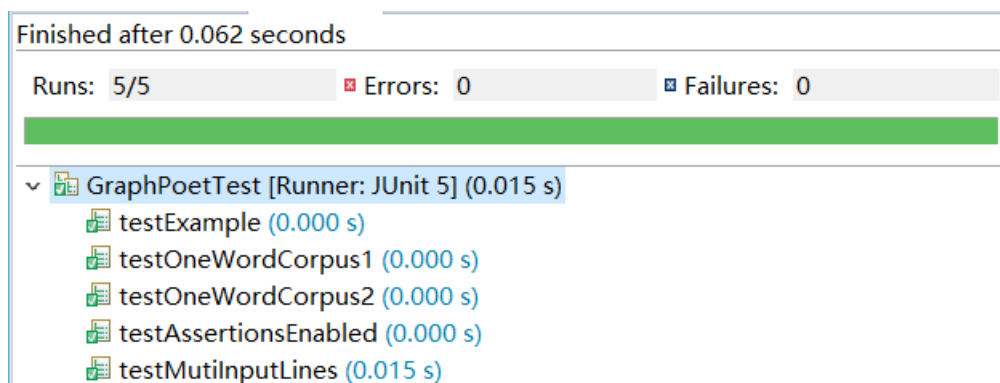
### 3.1.5.1 Test GraphPoet

测试用例均在 test 文件包的 poet 子文件里。

// Testing strategy

// corpus: 一个单词+多个单词+一行单词+多行单词

// input: 一个单词+多个单词+一行单词+多行单词+大小写



### 3.1.5.2 Implement GraphPoet

1, 表示不变量

// Abstraction function:

// a graph with the ConcreteEdgesGraph that represents a graph

// Representation invariant:

// vertices of the graph are non-empty case-insensitive strings

// of non-space non-newline characters

// Safety from rep exposure:

// graph field is private and final.

2, 检查不变量

```
private void checkRep() {  
    for (String vertex : graph.vertices()) { // 桥接词不区分大小写  
        String compare = vertex.toLowerCase();  
        assert vertex.equals(compare);  
        assert !vertex.equals(""); // 证明顶点内容单词非空  
    }  
}
```

```
    }
}
```

### 3, Graph方法

#### a) public GraphPoet(File corpus)方法

将文件中文本按照空格进行划分（调用String.split函数），将得到的所有单词加入到graph中，同时加边，权重是该边出现的次数

#### b) public String poem(String input)方法

将输入的文本按照空格进行划分（调用String.split函数），然后找到这些单词中相邻单词的桥接单词：一个单词调用targets方法，另一个单词调用sources方法，进行比对是否含有相同的单词；若两个单词含有多个桥接词，则进行选择一个权值较大者。同时选择StringBuilder将输出的所有单词拼接在一起（加上空格）

#### c) public String toString方法

重写此方法。将graph的顶点和边以及权重输出表示。

```
@Override public String toString() {
    String s = new String();
    for (String string : graph.vertices()) {
        s += string + " : ";
        for (String target : graph.targets(string).keySet()) {
            s += "(" + target + ", " + graph.targets(string).get(target) + ")";
        } // 输出出边和权重
        s += "\n";
    }
    return s;
}
```

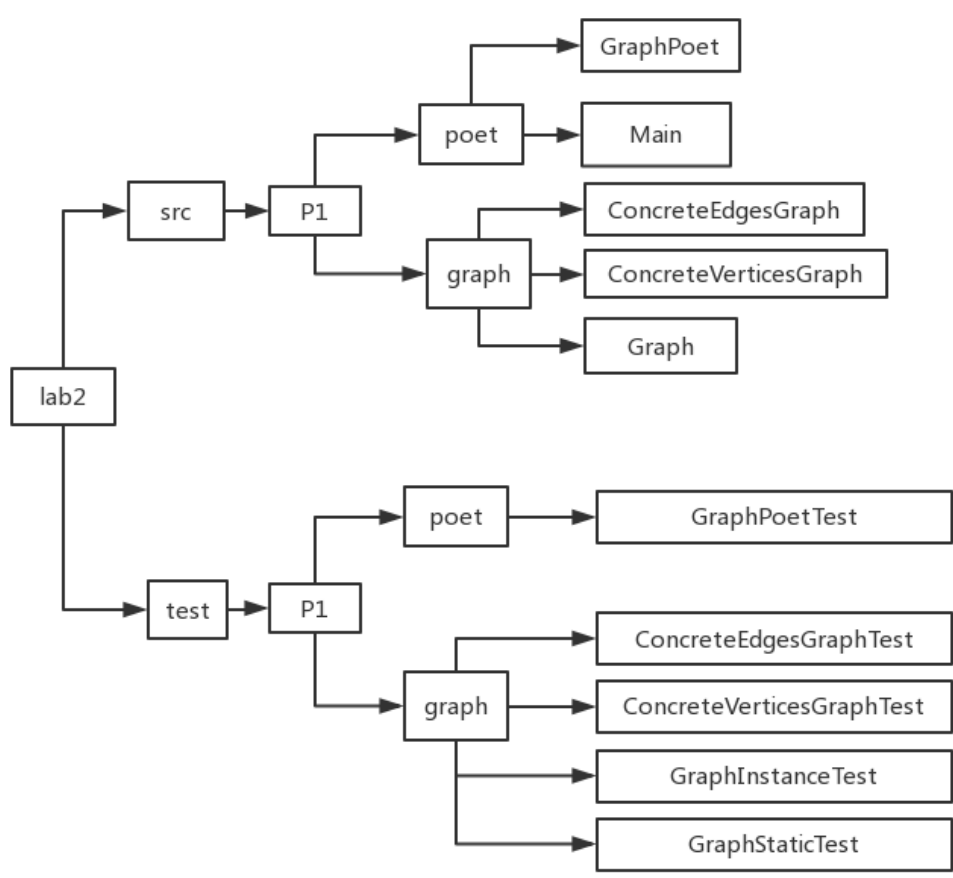
### 3.1.5.3 Graph poetry slam

更多的实现用例在 GraphPoet 中的 test 类中均存在。

故在这里不再赘述

### 3.1.6 Before you're done

- git add \*
- git commit -m "second commit"
- git push -u origin master
- 提交完成
- 项目树状图



3.2 Re-implement the Social Network in Lab1

充分利用 Lab1 的思想，并充分运用本次实验 P1 的类进行设计实验；  
其中 `getDistance` 算法利用了广度优先搜索队列的原理进行处理

3.2.1 FriendshipGraph 类

方法有

<code>addVertex</code>	调用 <code>Graph</code> 类中的 <code>add</code> 方法
<code>addEdge</code>	调用 <code>Graph</code> 类 <code>set</code> 方法，并将权重赋为 1
<code>getDistance</code>	利用广度优先遍历的思想，调用 <code>Graph</code> 类中的 <code>targets</code> 方法得到一个人的所有有联系的人
<code>main</code>	主函数，并进行简单的算法正确性验证

重要说明：在 `getDistance` 方法中，声明了两个 `Map` 结构，如下

```
Map<Person, Boolean> visited = new HashMap<>(); // 用于判断是否被访问
Map<Person, Integer> dis = new HashMap<>(); // 用于记录距离
```

在 `getDistance` 方法中, 声明了一个 `Queue` 结构, 如下

```
Queue<Person> queue = new LinkedList<>(); // 用于广度优先遍历
```

### 3.2.2 Person 类

Person 属性

```
private String name;
```

```
private static Set<String> allName = new HashSet<>(); // 用于储存所有的名字, 判断是否违背名字唯一原则
```

Person 方法

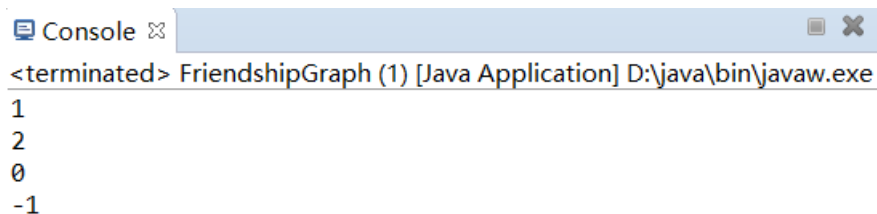
<code>Person</code>	初始化类, 输入是 <code>Person</code> 的 <code>String</code> 名字
<code>getName</code>	得到名字

### 3.2.3 客户端 `main()`

给出你的设计和实现思路/过程/结果。

```
public static void main(String[] args) {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person("Rachel");
    Person ross = new Person("Ross");
    Person ben = new Person("Ben");
    Person kramer = new Person("Kramer");
    graph.addVertex(rachel);
    graph.addVertex(ross);
    graph.addVertex(ben);
    graph.addVertex(kramer);
    graph.addEdge(rachel, ross);
    graph.addEdge(ross, rachel);
    graph.addEdge(ross, ben);
    graph.addEdge(ben, ross);
    System.out.println(graph.getDistance(rachel, ross));
    // should print 1
    System.out.println(graph.getDistance(rachel, ben));
    // should print 2
    System.out.println(graph.getDistance(rachel, rachel));
    // should print 0
    System.out.println(graph.getDistance(rachel, kramer));
    // should print -1
}
```

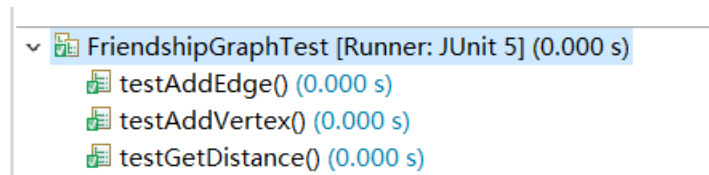
运行结果如下



```
<terminated> FriendshipGraph (1) [Java Application] D:\java\bin\javaw.exe
1
2
0
-1
```

### 3.2.4 测试用例

运行结果如下



设计思路

<code>testAddVertex</code>	判断加入是否成功+判断加入是否重复
<code>testAddEdge</code>	判断加入是否成功+加入是否多余
<code>testGetDistance</code>	判断为 0，为 -1，多个结果取最短

关于 testGetDistance 的一点说明

要验证假如有多个从 A 到 B 的通路的时候，取得路径的距离是最短的一条的。

- 实验验证设计关系图为

A->B->C

B->A->C

C->B->A

则 A 与 C 之间最小的距离应该是 1

- 实验验证设计关系图为

A->B->C->D

B->A->C->D

C->B->A->D

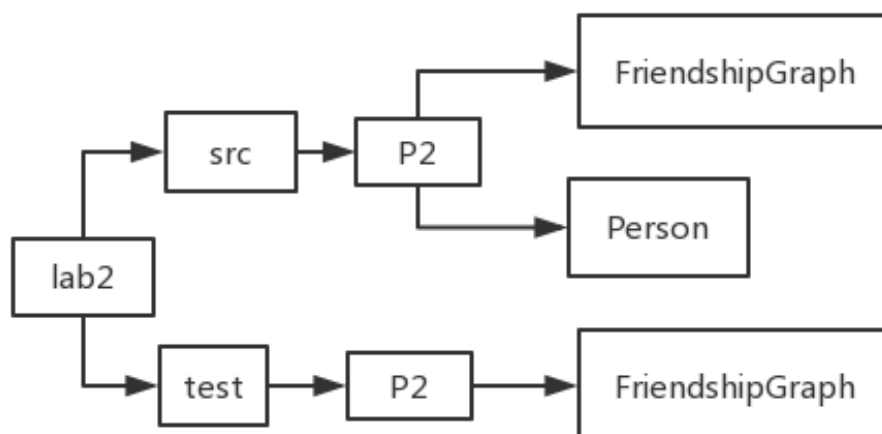
D->A->B->C

则 A 与 D 之间最短的距离也是 1

### 3.2.5 提交至 Git 仓库

- `git add *`  
`git commit -m "third commit"`  
`git push -u origin master`
- 树状图 (Process on)





### 3.3 Playing Chess

#### 3.3.1 ADT 设计/实施方案

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

##### 3.3.1.1 Action 类

方法有 `checkMovePiece` 和 `checkEatPiece`

`checkMovePiece`:

调用 `Position` 中的 `checkPosition` 函数检查坐标是否超出棋盘范围；调用 `Board` 中的 `getBoardOccupation` 函数检查 `position` 是否被占用以及被哪位选手占用。合法的条件是：原来的坐标被该选手占用且目标坐标位置无人占用且在棋盘范围内。

`checkEatPiece`

调用 `Position` 中的 `checkPosition` 函数检查坐标是否超出棋盘范围；调用 `Board` 中的 `getBoardOccupation` 函数检查 `position` 是否被占用以及被哪位选手占用。合法的条件是：原来的坐标被该选手占用且目标坐标位置对手占用。

##### 3.3.1.2 Board 类

属性

```
private Boolean[][] goBoard1 = new Boolean[19][19]; // 用于储存棋盘棋子存在情况: true 表示已被占用
```

```
private Boolean[][] goBoard2 = new Boolean[19][19]; // 用于储存棋手占用情况：
true 表示 player1
private Boolean[][] chessBoard1 = new Boolean[8][8]; // 用于储存棋盘棋子存在情
况：true 表示已被占用
private Boolean[][] chessBoard2 = new Boolean[8][8]; // 用于储存棋手占用情
况：true 表示 player1
```

方法

Board	初始化这几个二维数组
setGoBoard	将某个棋子下在围棋棋盘上
deleteChessBoardPiece	将棋子从象棋棋盘的二维数组表示上删除
addChessBoardPiece	将棋子的坐标位置表示在二维棋盘数组上
getOccupation	根据坐标判断该位置是否被占用
getOccupationPlayer	根据坐标判断该位置被谁占用
checkOccupation	检查某位置的占用情况

### 3.3.1.3 Game 类

属性

```
private Map<String, String> map // 用于输出棋盘时，将英文名字转化为中文名
private static Board goBoard = new Board(2); // 围棋棋盘
private static Board chessBoard = new Board(1); // 象棋棋盘
private Player player1; // 选手 1
private Player player2; // 选手 2
private static List<Piece> allPieces1 = new ArrayList<>(); // 玩家一所有棋子
private static List<Piece> allPieces2 = new ArrayList<>(); // 玩家二所有棋子
private String player1History = ""; // 玩家一的游戏历史
private String player2History = ""; // 玩家二的游戏历史
```

方法

public Game(int gameTye)	如果是象棋，则添加出初始的 32 个棋子
showGoMenu	输出围棋菜单选项
showChessMenu	输出象棋菜单选项
placePiece	在围棋棋盘上落子
checkPlacePiece	检查待落子坐标是否合法
extractPiece	在围棋棋盘上提子
checkExtractPiece	检查待提子位置坐标是否合法
movePiece	在象棋棋盘上移子
eatPiece	在象棋棋盘上吃子
showHistory	输出玩家游戏历史
setPlayer1/setPlayer2	设置玩家一/玩家二
getPlayer1/getPlayer2	得到玩家一/玩家二
getBoard	得到棋盘情况
getPiecesSize	得到棋子数目

getPieceByPosition	根据 Position 得到 Piece
changeHistory	改变玩家历史记录
printBoard	打印输出象棋或者围棋棋盘

### 3.3.1.4 MyChessAndGoGame

main 方法

整体运用 switch 语句，通过获取用户终端输入的数字，进行选择实现的功能；其中象棋与围棋的功能有所区别。switch 象棋部分主体的代码如下。

```
while (true) {
    switch (input) {
        case 1:
            System.out.println("请输入移子坐标");
            if (game.movePiece(in.nextInt(), in.nextInt(), in.nextInt(), in.nextInt(), player % 2 == 0)) {
                player++;
            }
            game.printBoard(1);
            break;
        case 2:
            System.out.println("请输入吃子坐标");
            if (game.eatPiece(in.nextInt(), in.nextInt(), in.nextInt(), in.nextInt(), player % 2 == 0)) {
                player++;
            }
            game.printBoard(1);
            break;
        case 3:
            System.out.println("跳过该选手");
            game.changeHistory("跳过", player % 2 == 0);
            player++;
            game.printBoard(1);
            break;
        case 4:
            in.close();
            System.exit(0);
        case 5:
            game.printBoard(1);
            break;
        case 6:
            System.out.println("请输入要选择的选手1或2");
            game.showHistory(in.nextInt() == 1);
            break;
        case 7:
            System.out.println("请输入查询坐标");
            Position position = new Position();
            position.setPosition(in.nextInt(), in.nextInt());
            game.getBoard(1).checkOccupation(position, 1);
            break;
        case 8:
            System.out.println("请输入要选择的选手1或2");
            System.out.println(game.getPiecesSize(in.nextInt() == 1));
            break;
        default:
            System.out.println("请输入正确!");
    }
    game.showChessMenu();
    input = in.nextInt();
}
```

switch 围棋主体部分如下

```

while (true) {
    switch (input) {
        case 1:
            System.out.println("请输入落子坐标");
            if (game.placePiece(in.nextInt(), in.nextInt(), player % 2 == 0)) {
                player++;
            }
            game.printBoard(2);
            break;
        case 2:
            System.out.println("请输入提子坐标");
            if (game.extractPiece(in.nextInt(), in.nextInt(), player % 2 == 0)) {
                player++;
            }
            game.printBoard(2);
            break;
        case 3:
            System.out.println("跳过该选手");
            game.changeHistory("跳过", player % 2 == 0);
            player++;
            break;
        case 4:
            in.close();
            System.exit(0);
        case 5:
            game.printBoard(2);
            break;
        case 6:
            System.out.println("请输入要选择的选手1或2");
            game.showHistory(in.nextInt() == 1);
            break;
        case 7:
            System.out.println("请输入查询坐标");
            Position position = new Position();
            position.setPosition(in.nextInt(), in.nextInt());
            game.getBoard(2).checkOccupation(position, 2);
            break;
        case 8:
            System.out.println("请输入要选择的选手1或2");
            System.out.println(game.getPiecesSize(in.nextInt() == 1));
            break;
        default:
            System.out.println("请输入正确!");
    }
    game.showGoMenu();
    input = in.nextInt();
}

```

几点简化的方法

1. 代码使用 int gameType 变量, 当其为 1 时, 表示为象棋, 否则表示为围棋; 同时使用 Boolean player 变量, 当其为 true 时, 表示为玩家一, 否则表示为玩家二。
2. 在功能的不断循环中, 当部分功能实现时, player 棋手默认控制权转变。但一些情况除外, 比如输出棋盘, 查看占用, 查看棋子总数, 查看游戏历史

### 3.3.1.5 Piece 类

属性

**private** Position piecePosition = new Position();// 棋子的坐标

**private** String pieceName = new String();// 棋子的类别

**private** Boolean player;// 棋子的拥有者: true 表示为棋手 1

方法

Piece	初始化一颗棋子
setPiecePosition	设置棋子的坐标

setPlayer	设置棋子的拥有者
getPiecePosition	得到坐标
getPlayer	得到棋子的拥有者
getPieceName	得到棋子的名字
setPieceName	设置棋子的名字

### 3.3.1.6 Player 类

属性

```
private String playName; // 玩家的姓名
```

方法

Player	设置玩家姓名
getPlayerName	返回玩家的姓名

### 3.3.1.7 Position 类

属性

```
private int px = -1; // 横坐标
```

```
private int py = -1; // 纵坐标
```

方法

setPosition	设置坐标属性
getPx	得到横坐标
getPy	得到纵坐标
positionEqual	判断 position 和当前位置是否重合
(override)toString	重写 toString
checkPosition	检查坐标是否越界，不在棋盘范围内

## 3.3.2 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。

设计和实现

- 读取输入的游戏类型：chess 或者 go
- 读取输入的玩家名字：player1 和 player2
- 读取玩家输入的数字：象棋中 1:移子 2:吃子 3:跳过 4:结束 5:棋盘 6:历史 7:查询占用 8:查询棋子总数。围棋中 1:落子 2:提子 3:跳过 4:结束 5:棋盘 6:历史 7:查询占用 8:查询棋子总数。进行性相应的功能匹配与调用

- **实现玩家交替进行**: 设置变量 `int player`, 每次实现一些功能, 就进行++操作, 同时根据 `player` 是奇数还是偶数进行判断是玩家一还是玩家二。
- **输出棋盘**: 每次相应的功能实现后, 都要输出相应的棋盘。
- **读取坐标**: 先读取的是横坐标, 其次读取的是纵坐标; 将这些坐标映射到功能要求的坐标上。

### 选择象棋

请先选择游戏: `chess`或`go`

`chess`

请输入棋手一的名字: `xiaoming`

请输入棋手二名字: `xiaohong`

### 移子

输入数字选择要实现的功能

\*\*\*\*\*  
1:移子 2:吃子 3:跳过 4:结束 5:棋盘 6:历史 7:查询占用 8:查询棋子总数  
\*\*\*\*\*

1  
请输入移子坐标

1 2

2 4

1:车	1:马	1:象	1:后	1:王	1:象	1:马	1:车
1:卒	1:卒	空	1:卒	1:卒	1:卒	1:卒	1:卒
空	空	空	空	1:卒	空	空	空
空	空	空	空	空	空	空	空
空	空	空	空	空	空	空	空
2:卒	2:卒	2:卒	2:卒	2:卒	2:卒	2:卒	2:卒
2:车	2:马	2:象	2:后	2:王	2:象	2:马	2:车

Player1:xiaoming      Player2:xiaohong

### 吃子

输入数字选择要实现的功能

\*\*\*\*\*  
1:移子 2:吃子 3:跳过 4:结束 5:棋盘 6:历史 7:查询占用 8:查询棋子总数  
\*\*\*\*\*

2  
请输入吃子坐标

7 1 2 4

1:车	1:马	1:象	1:后	1:王	1:象	1:马	1:车
1:卒	1:卒	空	1:卒	1:卒	1:卒	1:卒	1:卒
空	空	空	空	2:马	空	空	空
空	空	空	空	空	空	空	空
空	空	空	空	空	空	空	空
2:卒	2:卒	2:卒	2:卒	2:卒	2:卒	2:卒	2:卒
2:车	空	2:象	2:后	2:王	2:象	2:马	2:车

Player1:xiaoming      Player2:xiaohong

### 查询历史

输入数字选择要实现的功能

\*\*\*\*\*  
1:移子 2:吃子 3:跳过 4:结束 5:棋盘 6:历史 7:查询占用 8:查询棋子总数  
\*\*\*\*\*

6  
请输入要选择的选手1或2

1

移子:(1,2)->(2,1)

```
请输入要选择的选手1或2
```

```
2
```

```
吃子:(7,1)->(2,7)
```

### 查询占用

```
输入数字选择要实现的功能
```

```
*****
1:移子  2:吃子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
```

```
7
```

```
请输入查询坐标
```

```
1 2
```

```
该位置(1,2)未被占用
```

```
输入数字选择要实现的功能
```

```
*****
1:移子  2:吃子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
```

```
7
```

```
请输入查询坐标
```

```
1 3
```

```
该位置(1,3)已被棋手1占用
```

### 查询棋子总数

```
输入数字选择要实现的功能
```

```
*****
1:移子  2:吃子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
```

```
8
```

```
请输入要选择的选手1或2
```

```
1
```

```
16
```

```
输入数字选择要实现的功能
```

```
*****
1:移子  2:吃子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
```

```
8
```

```
请输入要选择的选手1或2
```

```
2
```

```
16
```

### 选择围棋

```
请先选择游戏: chess或go
```

```
go
```

```
请输入棋手一的名字: xiaoming
```

```
请输入棋手二的名字: xiaoguang
```

### 落子





## 查询棋子总数

```

输入数字选择要实现的功能
*****
1:落子  2:提子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
8
请输入要选择的选手1或2
1
0
输入数字选择要实现的功能
*****
1:落子  2:提子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
8
请输入要选择的选手1或2
2
0

```

## 查询历史

```

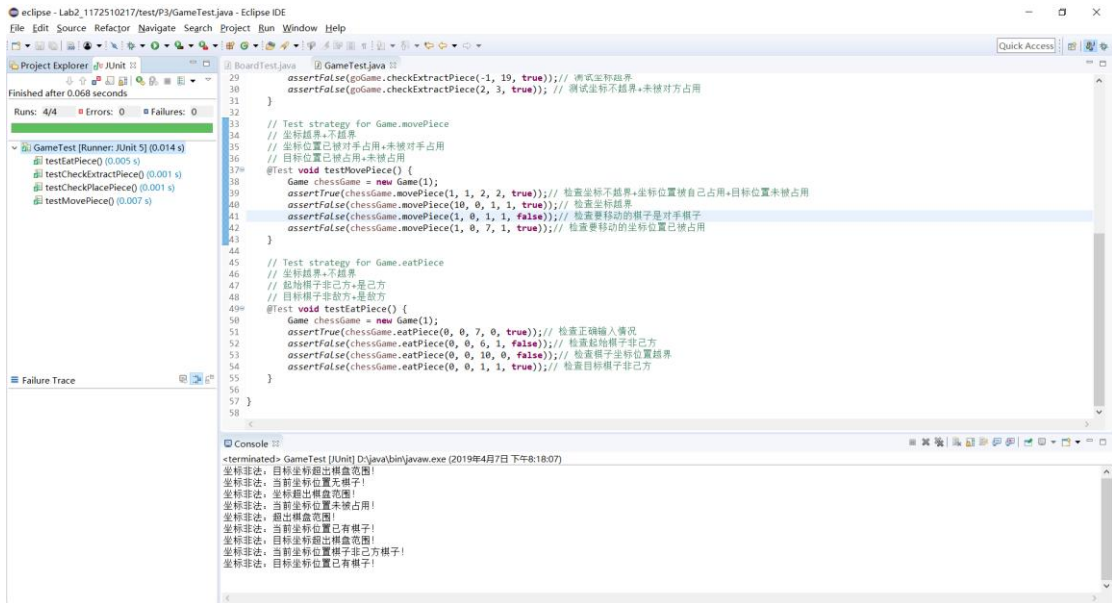
输入数字选择要实现的功能
*****
1:落子  2:提子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
6
请输入要选择的选手1或2
1
落子:(1,2)

输入数字选择要实现的功能
*****
1:落子  2:提子  3:跳过  4:结束  5:棋盘  6:历史  7:查询占用  8:查询棋子总数
*****
6
请输入要选择的选手1或2
2
提子:(1,2)

```

### 3.3.3 ADT 和主程序的测试方案

由于各个部分的调用关系，我们只需要测试部分类的部分方法即可  
Game 类测试截图如下。



```
// Test strategy for Game.checkPlacePiece
```

```
// 坐标越界+不越界
```

```
// 坐标位置已被占用+未被占用
```

```
// Test strategy for Game.checkExtractPiece
```

```
// 坐标越界+不越界
```

```
// 坐标位置已被对手占用+未被对手占用
```

```
// Test strategy for Game.movePiece
```

```
// 坐标越界+不越界
```

```
// 坐标位置已被对手占用+未被对手占用
```

```
// 目标位置已被占用+未被占用
```

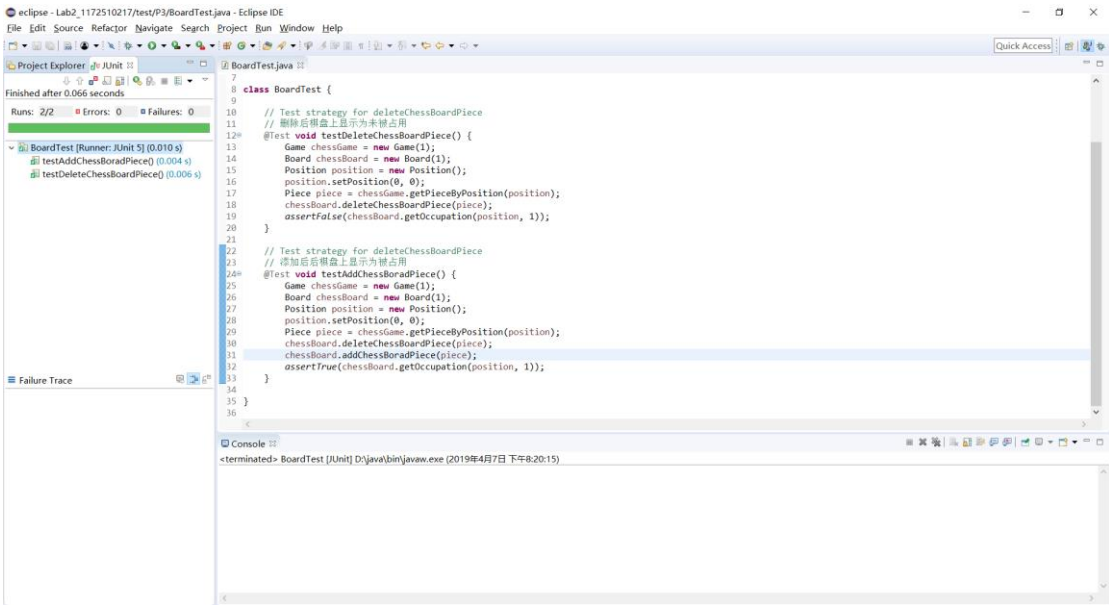
```
// Test strategy for Game.eatPiece
```

```
// 坐标越界+不越界
```

```
// 起始棋子非己方+是己方
```

```
// 目标棋子非敌方+是敌方
```

Board类测试截图如下



// Test strategy for deleteChessBoardPiece

// 删除后棋盘上显示为未被占用

// Test strategy for deleteChessBoardPiece

// 添加后后棋盘上显示为被占用

### 3.4 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。

## 4 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
19-03-19	晚上	了解 P1 题意	完成一半
19-03-20	晚上	了解 P1 题意	完成
19-03-21	晚上	完成 P1	完成三分之一
19-03-22	下午	完成 P1	完成二分之一
19-03-23	下午晚上	完成 P1	完成
19-03-24	晚上	完成 P2	完成
19-03-26	晚上	理解 P3	理解完成

19-03-28	晚上	完成一半类的构建	遇到问题，逻辑混乱
19-03-30	晚上	完成一半类的搭建	完成
19-04-01	晚上	完成 P3	遇到困难，出现 bug
19-04-02	晚上	完成 P3	完成

## 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
P1 问题理解难度大	百度翻译，谷歌翻译，必应翻译，自己翻译
P3 各部分调用逻辑紊乱	仔细构思，事前详细构思

## 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

- 1, 英语要好好学呀，专业术语也要好好学习；
- 2, 从零开始搭建一个 ADT，需要我们在书写代码前要详细构思，不能走一步看一步

### 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？  
前者调用关系比较多，后者逻辑关系相对较强
- (2) 使用泛型和不使用泛型的编程，对你来说有何差异？  
代码上差异不大，但应用的广泛程度上差别很大
- (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？  
减少了一些让我们有些不知所措的错误情况的出现  
暂时还不能太适应
- (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？  
应用范围更广
- (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？  
已经适应  
感受深刻，需要我们代码前比较全面的构思

- (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。  
工作量较大, 难度适中, deadline 合适
- (8) 《软件构造》课程进展到目前, 你对该课程有何体会和建议?  
实验性强, 需要我们不断地探索与实践