



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	
学号	
班号	
电子邮件	
手机号码	

目录

1 实验目标概述	1
2 实验环境配置	1
3 实验过程	1
3.1 Poetic Walks	1
3.1.1 Get the code and prepare Git repository	1
3.1.2 Problem 1: Test Graph <String>	2
3.1.3 Problem 2: Implement Graph <String>	2
3.1.3.1 Implement ConcreteEdgesGraph	2
3.1.3.2 Implement ConcreteVerticesGraph	3
3.1.4 Problem 3: Implement generic Graph<L>	4
3.1.4.1 Make the implementations generic	4
3.1.4.2 Implement Graph.empty()	4
3.1.5 Problem 4: Poetic walks	4
3.1.5.1 Test GraphPoet	4
3.1.5.2 Implement GraphPoet	5
3.1.5.3 Graph poetry slam	5
3.1.6 Before you're done	5
3.2 Re-implement the Social Network in Lab1	6
3.2.1 FriendshipGraph 类	6
3.2.2 Person 类	6
3.2.3 客户端 main()	7
3.2.4 测试用例	7
3.2.5 提交至 Git 仓库	8
3.3 Playing Chess	9
3.3.1 ADT 设计/实现方案	9
3.3.2 主程序 ChessGame 设计/实现方案	11
3.3.3 ADT 和主程序的测试方案	14
3.4 Multi-Startup Set (MIT)	15
4 实验进度记录	16
5 实验过程中遇到的困难与解决途径	17
6 实验过程中收获的经验、教训、感想	17

6.1 实验过程中收获的经验教训	17
6.2 针对以下方面的感受	17

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。包括如下内容:

- 1.1. 设计 ADT
- 1.2. 设计 ADT 规约
- 1.3. 设计测试用例
- 1.4. ADT 范型化
- 1.5. 设计多种不同的实现,能够自行设计 Rep、RI、AF
- 1.6. 防止表示泄露
- 1.7. 测试代码覆盖度

2 实验环境配置

2.1. JUnit 配置与 Lab1 无区别。

2.2. 代码覆盖率测试工具, IntelliJ IDEA 已经集成了三款,使用 Run with coverage 即可。

GitHub Lab2 仓库 URL:

<https://github.com/ComputerScienceHIT/Lab2-1173710229.git>

3 实验过程

3.1 Poetic Walks

此任务循序渐进,从实现两个图开始,到范型化,再到设计简单的写诗应用程序。

3.1.1 Get the code and prepare Git repository

使用命令 `git clone + URL` 获取实验资源

3.1.2 Problem 1: Test Graph <String>

此部分只要求设计以 String 作为顶点标签的图的测试类。

测试类包含两种，一个是静态的，一个是实例的。其中静态的测试类的方法已经提供，可选性修改。实例的测试类需要我们实现，设计出测试的策略即可。

3.1.3 Problem 2: Implement Graph <String>

3.1.3.1 Implement ConcreteEdgesGraph

本任务要求使用两个已提供的 rep 实现对具体的边的图实现，这两个 rep 分别是顶点集合、边列表。

1. 完成 AF、RI:

我对 AF 的设计和规定如下:

```
// Abstraction function:
// This class represents a graph, implemented with concrete edges.
```

RI 如下:

```
// Representation invariant:
// 1. There should be no same edges in edges list, i.e. there shouldn't be
// two edges that their sources and targets are the same.
// 2. Source and target in each edge must also be in vertices set.
```

2. 完成 checkRep 方法:

根据 AF 和 RI，完成 checkRep。因为 RI 包含两个限制，所以要分别判断。

首先覆盖 Edge 的 equals 方法，然后在 checkRep 中使用 edges 创建一个新的 HashSet，因为 Set 的特性，相同的对象会添加失败，所以将这个 set 与 edges 的 list 大小做对比，如果相同，说明没有重复的边出现在 edges 列表中。

其次，边的 list 中出现的所有 source 和 target 都必须被 vertices 包含，所以需要将所有 source 和 target 分别取出，判断 vertices 是否 containsAll 即可。

3. add()方法:

此方法是向顶点列表中添加一个顶点，返回值就是 vertices.add 的返回值，添加与返回之间需要 checkRep。

4. set()方法:

此方法根据 source 和 target 向图中添加一条带权为 weight 的边，操作分为多种情况:

- 如果 weight 是 0，且 source 和 target 中有一者不存在，此时不得进行任何操作，然后返回 0;
- 如果 weight 是 0，且 source 和 target 都存在，说明存在 source 指向 target 的边，此时把这条边移除，然后返回 0;
- 如果 weight 大于 0，且原本不存在从 source 指向 target 的边，此时创建

一个这条边, 并且它的权是 `weight`。特殊的, 如果 `vertices` 集合中没有 `source` 或 `target`, 需要将这个顶点加入集合, 然后返回 0;

- 如果 `weight` 大于 0, 且原本存在从 `source` 指向 `target` 的边, 此时修改这条边的权, 并且返回旧的权值;
- 如果 `weight` 小于 0, 抛出异常。

基本实现思路是: 先判断 `weight` 的取值情况, 如果小于 0, 抛出异常, 大于等于 0 进行下一步操作。遍历 `edges` 列表, 寻找是否有从 `source` 指向 `target` 的边, 如果找到了且 `weight` 大于 0, 就修改权值并返回旧的权值, 如果 `weight` 等于 0, 就从 `edges` 中移除这条边。如果没有找到从 `source` 指向 `target` 的边, 且 `weight` 大于 0, 就创建新的边并且添加到 `edges` 列表中; 如果 `weight` 等于 0, 不做任何修改操作。

5. `remove()`:

此方法的功能是从顶点集合移除一条边, 和其相连的边也要移除, 所以首先用集合的 `remove` 方法移除顶点, 如果移除失败就不要执行后续操作, 检查表示不变情况后返回 `false`。如果成功移除, 就要遍历边的列表, 从其中移除 `source` 或 `target` 是这个顶点的边, 检查表示不变量后返回 `true`。

6. `vertices()`:

此方法要返回顶点集合, 为了保护变量, 进行防御式拷贝后返回。

7. `sources(L target)`:

返回边的终点是 `target` 的边和边的权组成的 `map`, 遍历边的集合, 取出符合条件的边和权加入待返回的 `map` 即可。这里由于 `Edge` 类已被设计为不可变, 所以外界取得这个 `map` 之后做的一切修改, 都不会影响内部。

8. `targets(L source)`:

和 `sources` 方法类似, 遍历边的列表, 取出符合条件的边和权加入待返回的 `map`。

9. `toString()`:

以字符串将这个图形象化, 以邻接表表示出各个顶点的连接关系。

3.1.3.2 Implement ConcreteVerticesGraph

本任务要求使用已提供的 `rep` 对具体的顶点实现一个 `graph`

1. 完成 AF、RI:

我对 AF 的设计和规定如下:

```
// Abstraction function:
// This class represents a graph which is implemented with concrete vertexes.
```

RI 如下:

```
// Representation invariant:
// 1. There should be no same edges in edges list, i.e. there shouldn't be
// two edges that their sources and targets are the same.
// 2. Source and target in each edge must also be in vertices set.
```

2. 完成 `checkRep` 方法:

3. `add()`:

创建一个新的顶点对象加入顶点列表, 成功返回 `true`。但是如果待加入顶

点的 label 已经出现在了那个 list 中, 就不要添加, 返回 false。

4. `set()`:

分类与 `ConcreteEdgesGraph` 类似, 不同的是每个分类下的具体实现。

若 `weight` 大于 0, 首先遍历找 `source` 和 `target`, 找到就记录, 否则新实例化一个对象。如果顶点列表中不包含这两个顶点, 就添加到列表中。随后记录旧的权值, 添加或修改边。

若 `weight` 等于 0, 寻找边, 找到就删除。

若 `weight` 小于 0, 抛出异常。

5. `remove()`:

首先找传入的顶点, 找不到图将不发生变化, 找到之后删除它和其邻边。

6. `vertices()`:

获取顶点标签集合, 与 `ConcreteEdgesGraph` 中的实现类似。

7. `targets(L source)`:

获取满足起点为 `source` 的顶点和边权组成的 map。根据顶点类的设计, 需要先获取到该 `source` 顶点, 然后才能获取其 `targets`。因此先遍历顶点列表获取对应顶点, 然后返回它的邻接顶点的标签拷贝集合。

8. `sources(L target)`:

遍历顶点列表, 找到顶点 `V` 满足: 与它邻接的顶点包括 `target`, 则将 `V` 和他们之间边的权添加到待返回的 map 中。

3.1.4 Problem 3: Implement generic Graph<L>

3.1.4.1 Make the implementations generic

最初实现的两个图都是基于 `String` 的, 本任务要求我们将它们泛型化。泛型化时, 除了将 `String` 改成 `L` 之外, 最好也将 `equals()` 和 `hashCode()` 覆盖掉, 这样在集合、Map 中添加已经存在的对象时能够更快的检查出重复。

3.1.4.2 Implement Graph.empty()

这是 `Graph` 类中的一个静态工厂方法, 手册要求我们任意选择一种实现方法, 并且返回该类的一个实例化对象, 这里我使用的是 `ConcreteEdgesGraph`。

3.1.5 Problem 4: Poetic walks

3.1.5.1 Test GraphPoet

这一切要求我们关注测试方法, 根据 spec 直接设计好该测试的所有步骤。

3.1.5.2 Implement GraphPoet

此处的目标是将给定的语料逐句解析，用图来刻画各个单词之间的关系，用边权来表示关系重复出现次数。之后用已经创建的单词关系图，结合输入语句，创造诗句。

解析步骤：

1. 读取文本文件的一行，用空格分割出每个单词，将所有相邻的两个单词都创建一个关系；
2. 存在一个特殊情况：如果该关系已经添加过，那么应该增加原来边的权重。

作诗步骤：

1. 创建一个 String 列表 poemWords，用来存储完成的诗中的单词；
2. 创建另一个 String 列表 inputWords，用来存储输入语句中的单词；
3. 将第一个单词加入 poemWords，因为它必定出现在诗中，同时记录此单词为 fWord
4. 从 1 号元素开始遍历 inputWords，每个元素记为 cWord；
5. cWord 获得 sources，fWord 获得 targets；
6. 取得重合的部分，选取边权之和最大的组合，将重合的单词加入 poemWords 中。

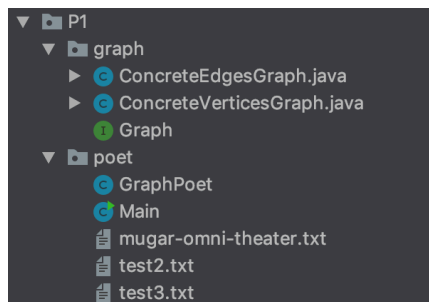
3.1.5.3 Graph poetry slam

本节可以自由发挥，自由创作。我从新闻网随意复制了一段文字，创建后打印的语句如图：

```
坚持了 基本原则
>>>
坚持了 科学社会主义 基本原则
```

3.1.6 Before you're done

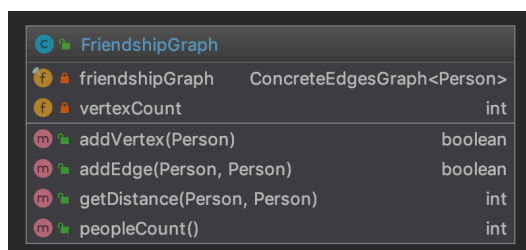
1. 所有方法都谢了文档注释
2. 所有类都完成了 AF RI 和 checkRep，也阐明了 safety from exposure
3. 所有覆盖父类的方法都使用了 @Override



3.2 Re-implement the Social Network in Lab1

用 P1 实现的两个具体图来重写实验 1 的朋友圈, 这里我使用了 ConcreteEdgesGraph。

3.2.1 FriendshipGraph 类



Member	Type
friendshipGraph	ConcreteEdgesGraph<Person>
vertexCount	int
addVertex(Person)	boolean
addEdge(Person, Person)	boolean
getDistance(Person, Person)	int
peopleCount()	int

此类有两个成员变量, friendshipGraph 是一个 ConcreteEdgesGraph 类的对象, vertexCount 是 int 类型变量。

以下是我定义的 AF RI 等的描述:

```
// AF
// This class represents a friendship graph in society.
// It is implemented by concrete edges graph, and the graph has its vertexes and edges.
// Every edge represents a connection between two persons.
// The two persons are the source and target vertex as the start and the end of an edge.

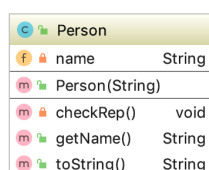
// RI
// true

// safety from exposure
// the mutator in this class can only modify the friendshipGraph object,
// In that object's instance methods, every change followed up with a checkRep() method
```

各个方法的设计:

1. addVertex: 向图中加入一个顶点
2. addEdge: 向图中加入一条边
3. getDistance: 计算两个 person 之间的距离
4. peopleCount: 返回参与当前交际圈的人数

3.2.2 Person 类



Member	Type
name	String
Person(String)	
checkRep()	void
getName()	String
toString()	String

Person 类有一个成员变量 name。

以下是我定义的 AF RI 等的描述:

```
// AF
// This class represents a person, each object has a name

// RI
// The name string can't be null or empty

// safety from exposure
// there's not mutator in this class. Only a constructor can mutate an object, but the checkRep() method follows.
```

各个方法:

1. getName: 向外暴露人名
2. checkRep: 检查当前类是否满足 RI
3. toString: 把此对象转化为可读的描述

3.2.3 客户端 main()

```
public static void main(String[] args) {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person( name: "Rachel");
    Person ross = new Person( name: "Ross");
    Person ben = new Person( name: "Ben");
    Person kramer = new Person( name: "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);

    assertEquals( expected: 1, graph.getDistance(rachel, ross));
    assertEquals( expected: 2, graph.getDistance(rachel, ben));
    assertEquals( expected: 0, graph.getDistance(rachel, rachel));
    assertEquals( expected: -1, graph.getDistance(rachel, kramer));
}
```

3.2.4 测试用例

按等价性划分为两部分测试:

1. 有向图:

```
@Test
public void testFriendshipGraph() {
    FriendshipGraph graph = new FriendshipGraph();
    Person rachel = new Person( name: "Rachel");
    Person ross = new Person( name: "Ross");
    Person ben = new Person( name: "Ben");
    Person kramer = new Person( name: "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);

    assertEquals( expected: 1, graph.getDistance(rachel, ross));
    assertEquals( expected: 2, graph.getDistance(rachel, ben));
    assertEquals( expected: 0, graph.getDistance(rachel, rachel));
    assertEquals( expected: -1, graph.getDistance(rachel, kramer));
}
```

2. 无向图:

```
/**
 * test for directed graph
 */
@Test
public void friendshipGraphTest2() {
    FriendshipGraph graph = new FriendshipGraph();

    Person rachel = new Person( name: "Rachel");
    Person ross = new Person( name: "Ross");

    Person ben = new Person( name: "Ben");
    Person kramer = new Person( name: "Kramer");

    Person a1 = new Person( name: "a1");
    Person b1 = new Person( name: "b1");
    Person c1 = new Person( name: "c1");
    Person alone = new Person( name: "alone");

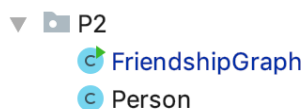
    graph.addVertex(rachel);
    graph.addVertex(ross);
    graph.addVertex(ben);
    graph.addVertex(kramer);
    graph.addVertex(a1);
    graph.addVertex(b1);
    graph.addVertex(c1);
    graph.addVertex(alone);
    graph.addEdge(rachel, ross);
    graph.addEdge(ross, rachel);
    graph.addEdge(ben, ross);
    graph.addEdge(a1, rachel);
    graph.addEdge(b1, a1);
    graph.addEdge(c1, ross);
    graph.addEdge(ross, c1);
    graph.addEdge(c1, kramer);

    assertEquals( expected: 1, graph.getDistance(rachel, ross));
    assertEquals( expected: 1, graph.getDistance(ross, c1));

    // directed graph, so ben can access to ross but ross cannot access to ben
    assertEquals( expected: 1, graph.getDistance(ben, ross));
    assertEquals( expected: -1, graph.getDistance(ross, ben));
}
```

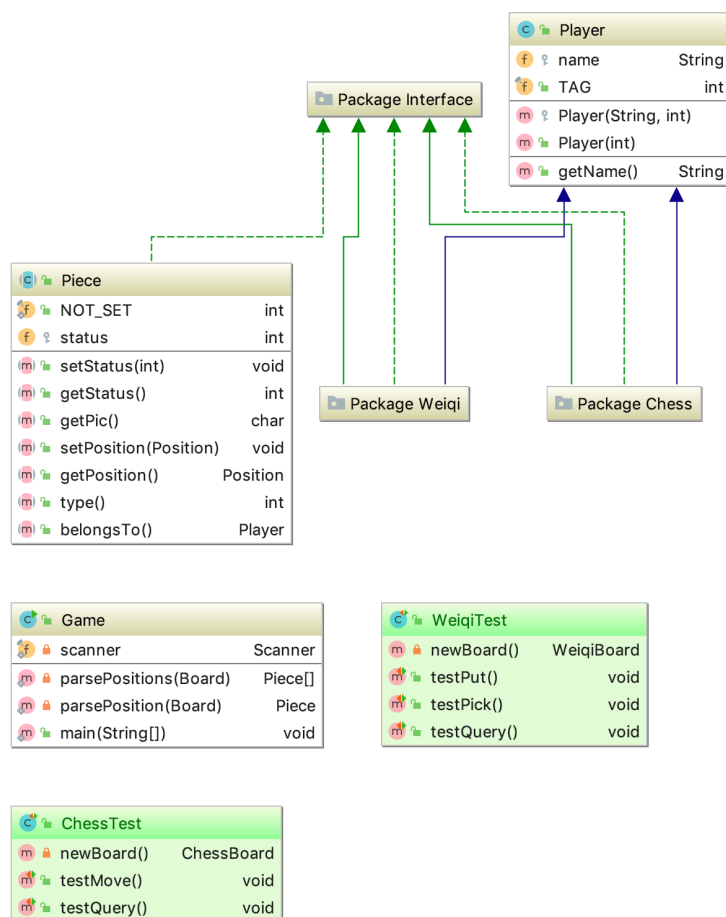
3.2.5 提交至 Git 仓库

使用 git commit 和 git push 将改动上传至 Git 仓库。P2 结构如图:

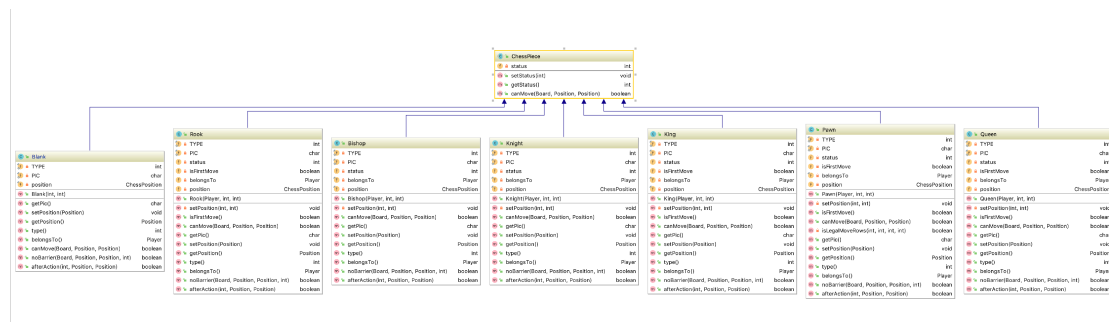


3.3 Playing Chess

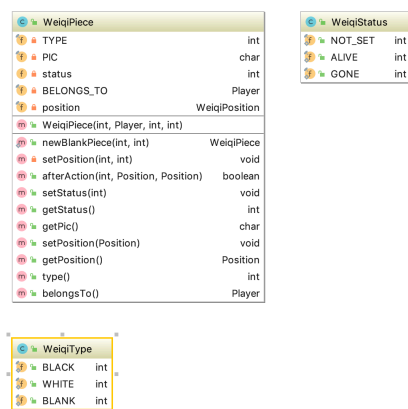
3.3.1 ADT 设计/实施方案



ADT 设计如上图，其中 Chess.Pieces 包结构如下图：



Weiqi.Pieces 包如图：



说明:

各个棋盘类都实现了 **Board** 接口，抽象棋子类实现 **Action**，具体棋子类继承抽象棋子类。

1. 玩家类:

```

/*
    AF
    this class represents a play in every game
*/

/*
    RI
    true
*/
  
```

2. 国际象棋棋子:

```

/*
    AF
    this abstract class is a model of every chess piece including pawn,
    bishop, king, knight, queen, rook, and these are also their type field.
    they all have their status and type(mentioned above). status represents a status in a chess game,
    for example, alive means the piece is still working. dead means it has benn eaten.
    they have their positions. position means the (row,col) in a chess board.
*/

/*
    RI
    true
*/

/*
    safety from exposure
    when returning a mutable field, it will be a copied version
*/
  
```

3. 国际象棋棋盘

```

// AF
// This class represents a chess board

// RI
// every cell on the board is not null;
// two players are not null and not same;
// current is in the players[]

// safety from exposure
// will return a mutable filed's copied value
  
```

4. 围棋棋盘类:

```

/*
    AF
    this class represents a weiqi board
    it has 19*19 places to put piece, and 2 players in one game
*/

/*
    RI
    every cell on the board is not null;
    two players are not null and not same;
    current is in the players[]
*/

/*
    safety from exposure
    will return a mutable filed's copied value
*/

```

5. 围棋棋子类:

```

/*
    AF
    this represents weiqi piece, including two types: black and white, distinguished by type field
*/

/*
    RI
    true
*/

/*
    safety from exposure
    when returning a mutable field, it will be a copied version
*/

```

3.3.2 主程序 ChessGame 设计/实现方案

首先输入 chess 或 go, 来确定要玩的游戏。

请输入你要进行的游戏
 国际象棋(chess)
 围棋(go)

不管输入 chess 还是 go, 都会要求输入玩家的名称:

```

chess
请黑方输入姓名:
Black
请白方输入姓名:
White

```

如果输入的是 chess, 之后会实例化国际象棋的棋盘对象, 游戏界面如图:

```

7|车|马|象|王|后|象|马|车|
6|兵|兵|兵|兵|兵|兵|兵|兵|
5| | | | | | | |
4| | | | | | | |
3| | | | | | | |
2| | | | | | | |
1|兵|兵|兵|兵|兵|兵|兵|兵|
0|车|马|象|王|后|象|马|车|
  0 1 2 3 4 5 6 7
当前是黑方（上方）执棋，请玩家Black输入操作，输入end结束
1. 行棋
2. 查询占用
3. 统计棋子数目

```

这时有三个选项供用户选择，

1. 此时如果输入 1，会要求用户输入两个坐标，格式为 “row,col row,col”，前一个坐标是起始点，后一个坐标是目标点，例如移动在 6,0 处的兵到 4,0，会调用棋盘的 move 方法：

```

      ^ ^ ^ ^ ^ ^ ^ ^
当前是黑方（上方）执棋，请玩家Black输入操作，输入end结束
1. 行棋
2. 查询占用
3. 统计棋子数目
1
请输入您的移动操作，输入坐标(row, col)，以空格分隔起点和终点，例如："3,4 3,5"
6,0 4,0

```

```

7|车|马|象|王|后|象|马|车|
6| |兵|兵|兵|兵|兵|兵|兵|
5| | | | | | | |
4|兵| | | | | | | |
3| | | | | | | |
2| | | | | | | |
1|兵|兵|兵|兵|兵|兵|兵|兵|
0|车|马|象|王|后|象|马|车|
  0 1 2 3 4 5 6 7

```

此时会交换出手，轮到白方玩家行棋。

2. 2号功能是查询棋盘占用情况，会调用棋盘的 queryCapture 方法：

```

7|车|马|象|王|后|象|马|车|
6| |兵|兵|兵|兵|兵|兵|兵|
5| | | | | | | |
4|兵| | | | | | |
3| | | | | | | |
2| | | | | | | |
1|兵|兵|兵|兵|兵|兵|兵|
0|车|马|象|王|后|象|马|车|

```

```

0 1 2 3 4 5 6 7
当前是白方（下方）执棋，请玩家White输入操作，输入end结束
1. 行棋
2. 查询占用
3. 统计棋子数目
2
请输入查询位置 row,col, 例如 3,4
1,0
(1,0)当前被White的 "兵" 棋子占用

```

3. 功能 3 是统计己方棋子的数目，映射到玩家的 toString 方法：

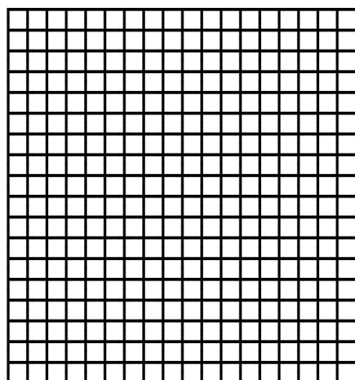
```

0 1 2 3 4 5 6 7
当前是白方（下方）执棋，请玩家White输入操作，输入end结束
1. 行棋
2. 查询占用
3. 统计棋子数目
3
White当前有16个棋子

```

4. 输入 end 结束游戏。

如果进行的游戏是围棋，之后会提供 6 种操作：



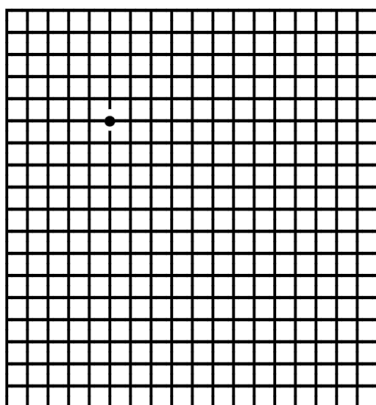
```

当前是黑方Black执棋
请输入要执行的操作：
1. 下子
2. 提子
3. 虚着（放弃本回合）
4. 查询占用
5. 统计我的棋子
结束游戏请输入end

```

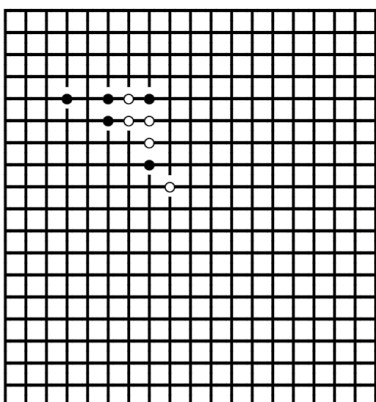
1. 下子，会调用 addPiece 方法当黑方执棋时，输入 5,5，会有如下变化：


```
1
请输入下子的坐标(row,col)如(1,2)
5,5
```



然后交换出手，轮到白方执棋。

2. 提子，提子只可以提取对方的棋子，将调用 `removePiece` 方法。此时是白方执棋，输入 5,5，会将刚才黑方放置的棋子提走。
3. 虚着，放弃本轮，直接轮到对方。
4. 统计己方棋子数目，调用 `queryCapture` 方法。操作例如：



```
当前是黑方Black执棋
请输入要执行的操作：
1. 下子
2. 提子
3. 虚着（放弃本回合）
4. 查询占用
5. 统计我的棋子
结束游戏请输入end
5
Black当前有5个棋子
```

（注：此前白方提子，将黑方 5,5 的棋子提走，所以黑方少了一个棋子）

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。

3.3.3 ADT 和主程序的测试方案

国际象棋测试策略

```
/*
    国际象棋测试
    this test contains these parts:
    1. test move function.
        1.1 use the return value of move function
        1.2 assert the old location is not the moved piece again
        1.3 assert the new location is the moved piece

    2. test querying capture situation
*/
```

1. 测试移动棋子方法, 包含三部分:
 - 1.1. 移动己方棋子到一个合法区域, 返回值应该为 `true`。
 - 1.2. 旧位置不再有该棋子。
 - 1.3. 新位置有该棋子。
2. 测试查询占用方法:

查询占用返回的字符串应当与构造的一样。

围棋测试策略

```
/*
    test for weiqi
    1. test put operation
        1.1 able to put
        1.2 can't put to the place

    2. test pick operation
        2.1 can pick away
        2.2 cannot pick away

    3. test query
*/
```

1. 测试放置棋子功能:
 - 1.1. 放置一个棋子到合法区域, 返回值为 `true`, 该位置可以查询到该棋子。
 - 1.2. 放置一个棋子到不合法区域, 返回值为 `false`, 不对棋盘产生任何影响。
2. 测试提子功能:
 - 1.1. 提取的目标为对方棋子, 则返回值为 `true`, 该棋子从棋盘上消失。
 - 1.2. 提取的目标为己方棋子或者该位置上没有棋子, 则返回值为 `false`, 不对棋盘产生任何影响。
3. 测试查询功能, 与构造的结果应当相同。

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
3.11	14:20-17:30	配置实验、完成大部分 P1	超时，实际只完成了一部分的 ConcreteEdgeGraph
3.13	14:00-15:30	完成 ConcreteEdgeGraph，并且完成 test	按计划完成
3.14	14:40-18:00	完成部分 ConcreteVertexGraph	按计划完成
3.14	19:10-21:00	完成 ConcreteVertexGraph	按计划完成，并且开始抽象出范型
3.15	11:00-11:30 14:30-16:50	完成 ConcreteEdgeGraph<L>和 ConcreteVertexGraph<L>并修复遗留的各种问题	按计划完成
3.16	10:40-12:50 15:50-17:35	摸索 P3，设计 P3 的相关类和接口	设计和细节基本完成
3.17	16:10-17:30	画出国际象棋棋盘，实现兵的棋子和规则	基本完成
3.17	21:10-22:10	1. 实现其他棋子的功能和规则，以及位置等等的具体实现。 2. Game 类的基本操作	尚未完成
3.18	14:00-15:30 17:35-21:40		
3.19	14:00-15:00 20:30-23:10		
3.20	14:00-17:00 19:00-22:00		
3.20	22:10-23:00	尝试 Poetic walk	尚未完成
3.21	15:30-16:40	完成 Poetic walk	完成
3.21	20:00-22:30	画出围棋棋盘，实现围棋的功能和 Game 中的操作	尚未完成
3.22	14:00-16:30		完成
3.24	10:40-11:00 12:40-16:30	修复 Bug，完善各种功能，完成某些阐述不清的注释	完成
3.25	14:30-17:30 19:20-20:10	完成遗漏的 AF、RI	完成
3.26	14:15-15:30 18:30-20:20 22:20-22:50	把出现表示泄露的类重新整理	未完成
3.27	16:00-17:50		完成

3.28	22:30-23:00	修复已发现的 bug	完成
3.29	10:50-11:30	写下棋的 test	未完成
3.30	17:00-17:30	修改错误, 目标通过所有棋盘测试	完成
4.1	14:20-15:10	实验报告	未完成
4.2	19:00-22:00		
4.4	15:30-17:30	完成实验报告	未完成
4.5	9:45-11:45		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
最初对可变和不可变的理解不够, 导致 P1-P3 非常多地方都发生了表示泄露	基本推翻了重写

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. 注意表示泄露问题
2. ADT 一定要提前设计好, 不要边写边设计, 这样只会导致不断的修改和重构

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?
面向 ADT 编程要求对 ADT 有良好的设计, 否则代码只会越来越复杂, 看向应用场景不要求那么复杂的 ADT, 把功能实现了即可。面向 ADT 不仅仅要求把点连成线, 还要求把线布置成面, 要不断考虑继承、组合、复用等等的一系列问题, 尽力写出高质量的代码。
- (2) 使用泛型和不使用泛型的编程, 对你来说有何差异?
泛型帮助我们写出更通用的代码, 比不使用泛型更具被复用能力。
- (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?
优势是更容易写出覆盖更广更严格的测试, 仍需要进一步适应这种测试

方式。

- (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?

能够使用同一段代码完成不同的功能, 使代码更简洁, 结构更清晰。

- (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?

体会到了设计 ADT 的不易, 想要设计出好的 ADT, 不仅仅是涂涂画画而已, 实践过程也会帮助改进 ADT。

- (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? 意义是时时刻刻提醒我们写代码时要遵守的东西, 时时刻刻注意安全性。以后也会坚持。

- (7) 关于本实验的工作量、难度、deadline。

虽说时间充裕, 但代码量非常大, 总计 3800 行+, (我的重复的代码很少, 合计 30 行以内)。相同性质的任务也不少, 感觉就像是把各个学校的各个实验一次性揉到了一起。

- (8) 《软件构造》课程进展到目前, 你对该课程有何体会和建议?

体会很深刻, 都是一次又一次推翻重做的血淋淋的教训。这个课程帮助我一点点改掉旧的编程习惯, 让我意识到什么是构造软件, 让我开始注意许多我不曾留意的编程细节。