哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Lab Manuals for

Software Construction

Lab-3

Reusability and Maintainability oriented Software Construction

School of Computer Science and Technology

Harbin Institute of Technology

Spring 2018

目录

1 实验目标

本次实验覆盖课程第 3、5、6 章的内容，目标是编写具有可复用性和可维护性的软件，主要使用以下软件构造技术：

- 子类型、泛型、多态、重写、重载
- 继承、代理、组合
- 常见的 OO 设计模式
- 语法驱动的编程、正则表达式
- 基于状态的编程
- API 设计

本次实验给定了四个具体应用（Lab 2 中的 GraphPoet、Lab 1 中的 SocialNetwork、网络拓扑结构 NetworkTopology、电影网络 MovieGraph），学生不是直接针对四个应用分别编程实现，而是通过 ADT 和泛型等抽象技术，开发一套可复用的 ADT 及其实现，在 Lab 2 所完成的抽象数据类型 `Graph<L>`的基础上，进一步扩展至 `Graph<L,E>`，充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用和更容易面向各种变化（可维护性）。

2 实验环境

实验环境设置请参见 Lab-0 实验指南。

本次实验在 GitHub Classroom 中的 URL 地址为：

https://classroom.github.com/a/aMg3ti15

请访问该 URL，按照提示建立自己的 Lab3 仓库并关联至自己的学号。

本地开发时，本次实验只需建立一个项目，统一向 GitHub 仓库提交。实验包含的多个任务分别在不同的目录内开发，具体目录组织方式参见各任务最后一部分的说明。请务必遵循目录结构，以便于教师/TA 进行测试。

3 实验要求

实验 2 实现了一个 `Graph<L>`和它的两个实现类 `ConcreteEdgesGraph<L>`、

`ConcreteVerticesGraph<L>`，并用 `String` 代替 `L` 实现了 `GraphPoet` 应用。

在本次实验里，你需要针对现实当中的各种图应用，进一步扩展 `Graph<L>` 及其实现，从而使其抽象能力更强、适应现实中不同情况需求的能力更强。

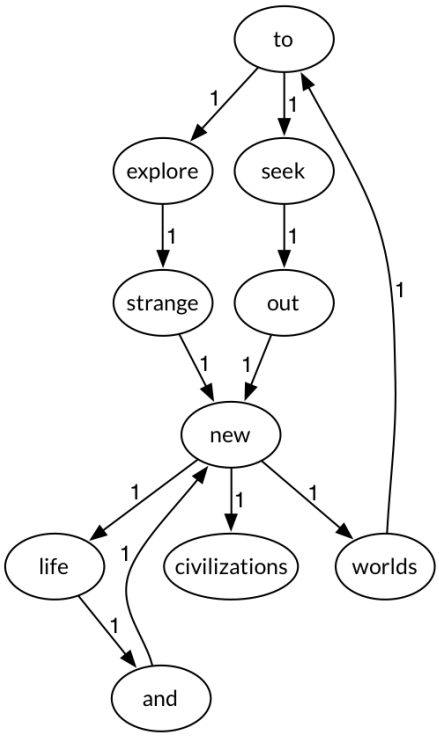
3.1 基本概念

Undirected graph 无向图	A graph in which edges have no orientation
Directed graph 有向图	A graph in which edges have orientations
Mixed graph 混合图	A graph in which some edges may be directed and some may be undirected.
Simple graph 简单图	An undirected graph in which both multiple edges and loops are disallowed.
Multigraph 多重图	An undirected graph in which there are two or more edges that connect the same two vertices.
Simpledigraph 单重有向图	A directed simple graph.
Multidigraph 多重有向图	A directed multigraph.
Weighted graph 带权图	A graph in which a number (the weight) is assigned to each edge
Singlemode graph 单模图	A graph in which the types of all vertices are the same.
Multimode graph 多模图	A graph in which the types of different vertices are different, e.g., some vertices are “computers” while others are “servers” or “routers”.
Hyperedge 超边	A non-empty subsets of vertices (see Wikipedia).

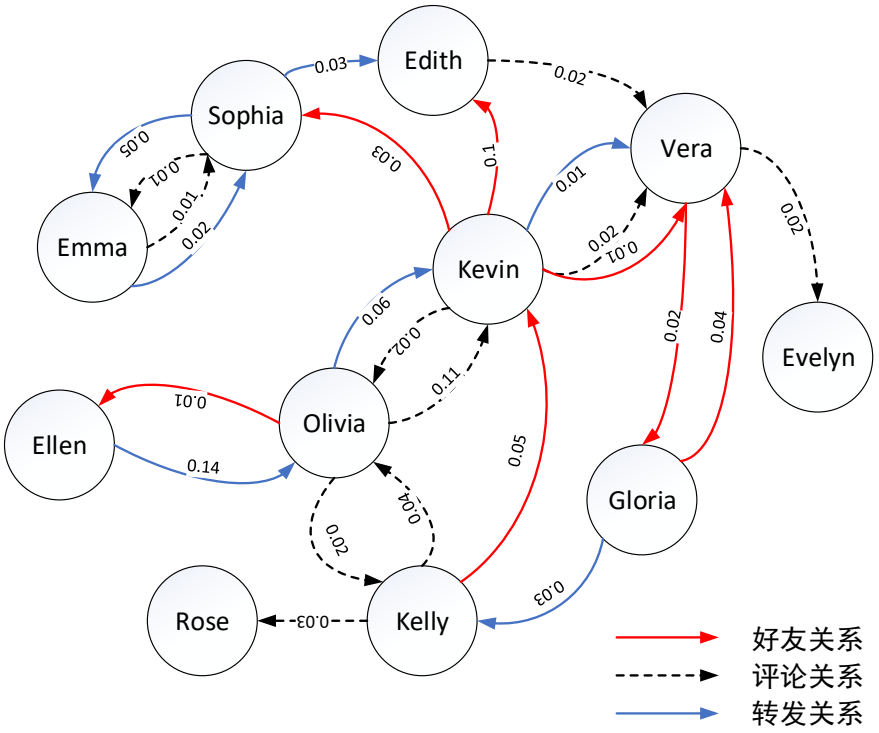
3.2 待开发的应用场景

你需要为以下场景分别开发 Java 程序，故在设计和实现 ADT 的时候需要充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用和更容易面向各种变化。

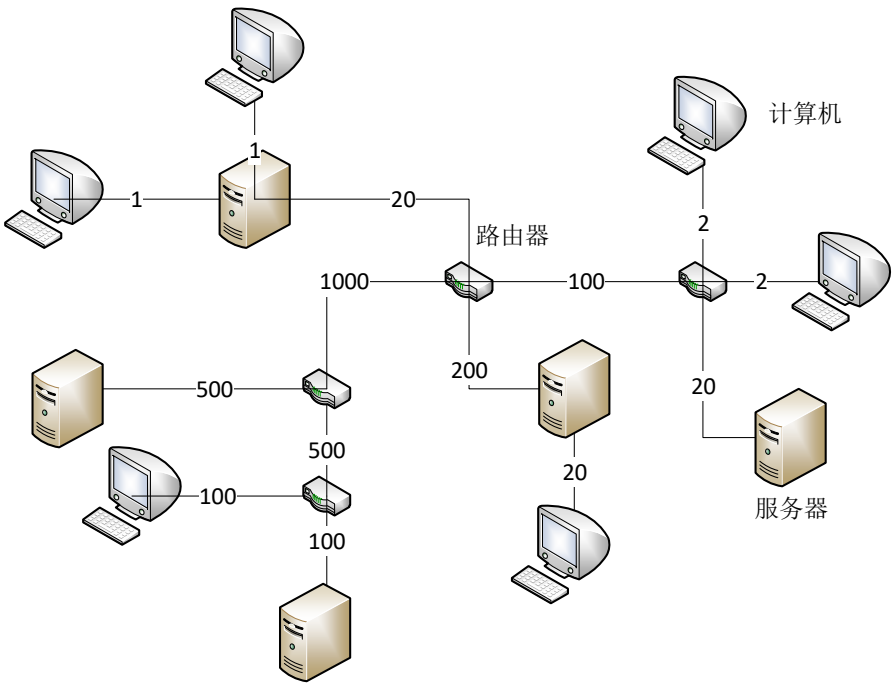
(1) 单词网络 `GraphPoet`：具体参见 Lab 2 中 3.1 节的说明，节点为“单词”（label 为该单词的文本字符串，无其他属性），边为两个单词在文本中的相邻关系，边的权重是相邻出现的次数（值域为正整数）。图中可以出现 loop，即一条边的起点和终点为同一个节点。它是单重有向图、带权图、单模图。



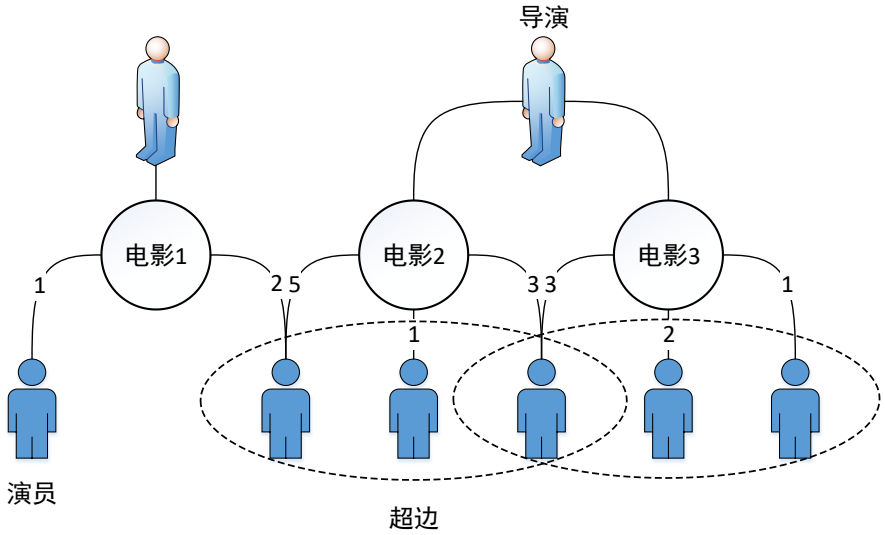
(2) 微博社交网络 `SocialNetwork`：节点为“用户”（label 为用户姓名，其他属性包括：性别、年龄），边为两个用户的社交关系。两个用户之间最多可存在 3 种类型的社交关系边，分别表征“好友关系”（A 关注了 B，边的方向为 A→B）、评论关系（A 曾评论过 B 的微博，边的方向为 A→B）、转发关系（A 曾转发过 B 的微博，边的方向为 A→B）。边的权重表示二者通过特定社交关系类型进行交互的频度，取值范围为(0,1]，图中所有边的权值之和=1。图中不能出现 loop，即一个人与自己不能是好友/评论/转发关系。该图是一个多重有向图、带权图、单模图，由此可知任意两个节点之间的边的总数最多为 6（3 种类型的社交关系边，2 个方向）。



(3) 网络拓扑图 `NetworkTopology`：节点为“计算机”、“服务器”、“路由器”（label 为主机名，属性包括：IP 地址），边为它们之间的网络连接关系（但计算机之间不能直接相连、服务器之间不能直接相连），权重为网络连接的带宽（例如 1、20、100）。图中不能出现 loop，即一条边的起点和终点为不能为同一个节点。它是简单图、带权图、多模图。



(4) 电影网络 MovieGraph：节点为“电影”（label 为电影名，还有三个属性：上映年份、拍摄国家、IMDb 评分）、“演员”（label 为姓名，属性：年龄、性别）、“导演”（label 为姓名，属性：年龄、性别），有两种无向边：演员 A 参演了电影 M、导演 D 执导了电影 M。这是多重图、多模图，演员和电影之间的边有权值（表示 A 在 M 中的角色次序，用正整数表示），导演和电影之间的边无权值。图中不能出现 loop，即一条边的起点和终点不能为同一个节点。参演过同一部电影的所有演员形成一条超边，超边无权值。



注：在以上四个应用中，图中均不能出现具有相同 label 的节点、具有相同 label 的边。

3.3 基于语法的图数据输入

以下语法（grammar）用于撰写特定格式的输入文件，你的程序读入该文件并使用正则表达式 parser 对其进行解析，从中抽取信息，构造图结构。

表 1 图的语法定义

```
GraphType =
   “GraphPoet”|“SocialNetwork”|“NetworkTopology”|“MovieGraph”
```
图的类型，上述四种取值之一

`GraphName = Label` 该图的标题，字符串

`VertexType = type`$_1$`,…,type`$_n$

包含的节点类型列表，每个 type 的取值来自于表 3。如果是单模图，这里只会出现一个类型。

Vertex = <$Label_1$,$type_1$,<$attr_1$,...,$attr_k$>>
...
Vertex = <$Label_m$,$type_m$,<$attr_1$,...,$attr_k$>>
每行代表一个节点，type 为节点类型，来自于 VertexType 定义的列表。不同节点的标签不能重复。$attr_1$,...,$attr_k$ 为该节点的属性值列表，具体信息参见表 3，属性值列表的次序与表 3 中的属性次序一致。如果该节点无属性值，则简化为 Vertex = <$Label_m$,$type_m$>。

EdgeType = $type_1$,…,$type_n$
图中包含的边的类型列表，每个 type 取值来自于表 5。

Edge = <Label, type, Weight, StartVertex, EndVertex, Yes|No>
...
Edge = <Label, type, Weight, StartVertex, EndVertex, Yes|No>
每行代表一条简单边，type 为边的类型，来自于 EdgeType 的列表；Yes|No 表示该条边是否为有向：若为有向，则该边为 StartVertex -> EndVertex。StartVertex 和 EndVertex 是在该行出现之前已经定义的 Vertex 的标签（Label）。不同边（包含简单边和超边）的标签不能重复。Weight 为边的权值（正数），当其值为“-1”的时候表示它为不带权的边。

HyperEdge = <Label, Type, {$Vertex_1$, …, $Vertex_n$}>
...
HyperEdge = <Label, Type, {$Vertex_1$, …, $Vertex_n$}>
每行代表一条超边，Type 为出现在 EdgeType 列表中的类型，$Vertex_1$,…,$Vertex_n$ 表示超边包含的节点列表（n>1），且各个 Vertex 均为之前已经定义的 Vertex 的标签。

Label：由 word character(\w，即[a-zA-Z_0-9])构成。

以下给出一个示例：

GraphType = “NetworkTopology”
GraphName = “LabNetwork”

VertexType = “Computer”, “Router”, “Server”
Vertex = <“Computer1”, “Computer”, <“192.168.1.101”>>

```
Vertex = <“Server1”, “Server”, <“192.168.1.2”>>
Vertex = <“Router1”, “Router”, <“192.168.1.1”>>

EdgeType = “NetworkConnection”
Edge = <“R1S1”, “NetworkConnection”, “100”, “Router1”,
        “Server1”, “No”>
Edge = <“C1S1”, “NetworkConnection”, “10”, “Computer1”,
        “Server1”, “No”>
```

另一个示例：

```
GraphType = “MovieGraph”
GraphName = “MyFavoriteMovies”

VertexType = “Movie”, “Actor”, “Director”
Vertex = <“TheShawshankRedemption”, “Movie”, <“1994”, “USA”,
           “9.3”>>
Vertex = <“FrankDarabont”, “Director”, <“59”, “M”>>
Vertex = <“MorganFreeman”, “Actor”, <“81”, “M”>>
Vertex = <“TimRobbins”, “Actor”, <“60”, “M”>>
Vertex = <“TheGreenMile”, “Movie”, <“1999”, “USA”, “8.5”>>
Vertex = <“TomHanks”, “Actor”, <“62”, “M”>>

EdgeType = “MovieActorRelation”, “MovieDirectorRelation”,
           “SameMovieHyperEdge”
Edge = <“SRFD”, “MovieDirectorRelation”, “-1”,
       “TheShawshankRedemption”, “FrankDarabont”, “No”>
Edge = <“GMFD”, “MovieDirectorRelation”, “-1”,
       “TheGreenMile”, “FrankDarabont”, “No”>
Edge = <“SRMF”, “MovieActorRelation”, “1”,
       “TheShawshankRedemption”, “TimRobbins”, “No”>
Edge = <“SRTR”, “MovieActorRelation”, “2”,
       “TheShawshankRedemption”, “MorganFreeman”, “No”>
Edge = <“GMTH”, “MovieActorRelation”, “1”, “TheGreenMile”,
       “TomHanks”, “No”>
HyperEdge = <“ActorsInSR”,“SameMovieHyperEdge”,
```

{“TimRobbins”, “MorganFreeman”}>

3.4 基于语法的图操作指令输入（选做，额外加分）

在应用运行过程中，用户可使用以下图操作命令向应用发出指令。请实现这些指令。（若有精力，可自行设计其他有价值的指令并实现之）

表 2 图操作的语法定义

操作语法	说明

`vertex --add label type`	增加一个节点，类型为 type（见表 3）。如果该类型的节点不应出现在该图中，则提示失败。如果图中已有该 label 的节点，则提示失败。
	例如针对上面的示例 1：
	`vertex --add “NewVertex1” “Computer”`
	`vertex --add “NewVertex2” “Person”`
	第一条语句可执行成功
	第二条语句执行失败。

`vertex --delete regex`	删除其 label 满足 regex 条件的所有节点，以及所有受影响的边，意即：如果满足 regex 条件的某个节点是某条边的两端之一，则该边被删除；如果某节点属于某条超边，若该节点删除后该条超边仍可合法存在，则该超边继续保留，否则就删除之。 删除前需经由用户确认。 例如：`vertex --delete “[a-zA-Z]\d{3}[a-zA-Z]”` 删除 label 中有三个连续数字且数字前后为字母的节点
`edge --add label type [weighted=Y|N] [weight] [directed=Y|N] v1, v2`	增加一条边，类型为 type（见表 5），标签为 label，[…]表示可选参数，v1 和 v2 表示两个在图中已存在的节点。 需要根据当前图的类型、type 所表征的边的类型、v1/v2 的节点类型，判断该条新增边是否合法；若不合法，则提示失败，不增加新边。 例如： ● 若该图 v1 和 v2 之间已存在同类型的边，

	则增加边失败。 ● 若 v1 和 v2 的类型均为 Computer，指令中边的 type="NetworkConnection"，那么增加边失败。 例如： edge --add "NewEdge1" "WordNeighborhood" weighted=Y 2 directed=Y "This" "is" 在单词网络中增加一条 this 和 is 之间的有向边，权值 2
edge --delete regex	删除其 label 满足 regex 条件的所有边（包括简单边和超边），但节点不应受影响。
hyperedge --add label type $vertex_1$, ..., $vertex_n$	将一组节点聚合起来成为一条超边，其标签为 label；若该超边已经存在，则直接将这些节点加入该超边。若该图不应包含超边，则提示失败。若这些节点不符合图中超边的定义（例如某个 actor 节点与其他 actor 并非在同一部电影中出演），则提示失败。 例如： hyperedge --add "ForeverYoung" "SameMovieHyperEdge" "ZhangZiyi", "HuangXiaoming", "WangLihong"

3.5 面向复用的设计：Graph<L,E>

实验 2 中所设计的泛型 Graph<L>仅对节点的类型进行了参数化，这不够满足 3.2 节中给出的多种图应用场景需要。为此你需要设计新的接口：

Graph<L,E>

其中 L 和 E 分别代表节点和边的类型。

该接口提供的操作如下：

- public static <L,E> Graph<L,E> empty() 构造一张图的空实例
- public boolean addVertex(L vertex) 向图中增加一个节点
- public boolean removeVertex(L vertex) 从图中删除一个节点
- public Set<L> vertices() 返回图的节点集合
- public Map<L, Integer> sources(L target) 与Lab2中Graph接口的同名操作含义相同；如果与target相连的边包括无向边，则无向边的另一端节点也需包含在返回值Map中；不需考虑超边。
- public Map<L, Integer> targets(L source) 与Lab2中Graph接

口的同名操作含义相同；如果与source相连的边包括无向边，则无向边的另一端节点也需包含在返回值Map中；不需考虑超边。

- public boolean addEdge(E edge)增加一条边（包括超边）
- public boolean removeEdge(E edge) 删除一条边（包括超边）
- public Set<E> edges() 返回边的集合（包括超边）

接下来需要设计一个抽象类来实现接口 Graph<L,E>：

abstract class ConcreteGraph<L,E>

其内部 rep 为 Collection<L>、Collection<E>，分别为节点和边的集合，Collection 的具体子类型选择取决于你自己的设计思考。

在四个应用中，你需要分别从 ConcreteGraph<L,E>派生出更具体的、面向应用的类，分别为：

GraphPoet<L,E>

SocialNetwork<L,E>

NetworkTopology<L,E>

MovieGraph<L,E>

并将 L 和 E 替换为各应用中你设计的具体节点类与边类。

为以上完成的 Graph<L,E>、ConcreteGraph<L,E>和四个具体应用类设计和编写 JUnit 测试用例。

注意：本次实验中需要你构造的所有 ADT 接口及其实现类，均需按实验 2 的要求撰写 AF、RI、Safety from rep exposure，以及每个方法的 specification（precondition、post-condition 等）。

另外，在本文档中没有明确说明的情况下，你设计的 ADT 应遵循图的定义与约束条件。例如：Collection<E>中出现的每一条边，其节点均应出现在 Collection<L>中。

后续各节不再重复这两个要求。

3.6 面向复用的设计：Vertex

定义 Graph<L, E>中的 L，即节点抽象类 Vertex

abstract class Vertex

Vertex 中定义各应用的共性数据表示以及作用在其上的操作，至少应覆盖以下方面：

属性：

- `String label` 外部可观察的标签信息

操作：

- `Vertex(String label)` 构造函数
- `abstract void fillVertexInfo (String[] args)` 为特定应用中的具体节点添加详细属性信息，所需的信息参见表 3 的最后一列，参数的次序与表 3 保持一致。因为不同的 `Vertex` 子类型的属性不同，故该操作可设计为 `abstract`，在具体子类里实现之。
- `public String getLabel()` 返回节点 `label` 的取值
- `toString()` 如果针对不同子类型有不同的 `string` 表示，则需要在各个子类型中 `override` 该函数。
- `equals()` 类似于 `toString()`
- `hashCode()` 类似于 `toString()`
- 其他你认为可抽象至 `Vertex` 的共性操作

这是一个 mutable 的 ADT，请务必注意安全性！

在各个应用中，需从 `Vertex` 派生子类型，通过 override 或 overload 等方式，支持应用的具体需求。下表中各子类的方法集合，取决于你自己的设计，以满足应用需求为准，但必须要仔细斟酌考虑复用性；各子类的属性集合，请不要扩展。

表 3 各应用的具体 Vertex 子类型

应用	子类型	含义	Label	属性
单词网络	`Word`	单词	文本	
社交网络	`Person`	用户	姓名	1. 性别（`M/F`，字符串） 2. 年龄（正整数）
网络拓扑	`Computer`	计算机	主机名	1. IP 地址（字符串，用“.”分割为四部分，每部分的取值范围为`[0,255]`）
	`Server`	服务器		
	`Router`	路由器		
电影网络	`Movie`	电影	电影名	1. 上映年份（四位正整数，范围为`[1900, 2018]`） 2. 拍摄国家（字符串） 3. `IMDb` 评分（`0-10` 范围内的数值，最多 2 位小数）
	`Actor`	演员	姓名	1. 年龄（正整数） 2. 性别（`M/F`，字符串类型）
	`Director`	导演		

（选做，额外加分）在某些应用中，图的节点有一系列的状态，在不同状态

下执行操作后可转为其他状态。请使用 State/Memento 模式，根据下表列出的信息为 Vertex 的某些子类型增加状态管理的功能，能够恢复节点对象之前的状态。为此，需在相关子类中增加 save()，restore()，getState()三个操作，分别用于保存当前状态、恢复之前状态、获得当前状态。

表 4 各应用中的实体的状态及其转换

应用	状态	状态转换
WordPoet	无	无
SocialNetwork	deactive 休眠 active 活跃 locked 被封禁	● deactive->locked 通过 lock()操作 ● active->locked 通过 lock()操作 ● locked->active 通过 unlock()操作 ● active->deactive 通过 deactive()操作 ● deactive->active 通过 active()操作
NetworkTopology	close 关机 open 运行中	● close->open 通过 open()操作 ● open->close 通过 close()操作
MovieGraph	无	无

3.7 面向复用的设计：Edge

定义 Graph<L，E>中的 E，即边抽象类 Edge

```
abstract class Edge
```

Edge 中定义各应用的共性数据表示以及作用在其上的操作，至少应覆盖以下方面：

属性：

- Collection vertices 该边中包含的所有节点，请自行设计采用何种具体 Collection 子类型作为 Edge 的 Rep
- String label 外部可观察的标签信息，字符串
- double weight 权值，若为带权边，则为非负数；若为无权边则该属性为-1

操作：

- Edge(String label, double weight) 构造函数
- toString() 不同类型的边，其字符串表示应有不同，需表征出端点、方向、权重、label，具体形式自行设计，在子类型中 override；
- equals() 类似于 toString()

- `hashCode()` 类似于 `toString()`
- 其他你认为可抽象至 `Edge` 的共性操作

考虑到不同图应用中包含不同类型的边（有向边、无向边、超边），在各个应用中，需从 `Edge` 派生子类型，支持应用的具体需求。具体信息如下表所示。

表 5 各具体边的子类型

应用	边的类型	单重/多重	方向	是否带权	是否超边
单词网络	`WordNeighborhood`	单重	有向	是	否
微博社交网络	`FriendTie` 通过关注建立的好友关系	多重	有向	否	否
	`CommentTie` 通过评论建立的社交关系		有向	是	否
	`ForwardTie` 通过转发建立的社交关系		有向	是	否
网络拓扑	`NetworkConnection`	单重	无向	是	否
电影网络	`MovieActorRelation` 演员和电影之间的关系	单重	无向	是	否
	`MovieDirectorRelation` 导演和电影之间的关系	单重	无向	否	否
	`SameMovieHyperEdge` 参演过同一部电影的演员之间的关系	单重	无向	否	是

根据具体应用的需求，在某些特定子类型中实现以下函数：

- `public boolean addVertex(Vertex v1, Vertex v2)` 针对非超边：如果是有向边，该操作将 `v1` 作为 `source`，将 `v2` 作为 `target`，即 `v1->v2`；如果是无向边，无需考虑 `v1` 和 `v2` 的次序；
- `public boolean addVertices(List<Vertex> vertices)` 如果是代表超边的子类，该函数添加 `vertices` 中的所有节点到该超边；

这也是 mutable 的 ADT，请务必注意安全性！

3.8 可复用 API 设计

基于四个应用构造的具体图结构，可在其上开展一系列关于图的计算。请针对 `ConcreteGraph<L,E>`设计 API 并实现具体代码。请遵循 façade 设计模式，将所有 API 放置在一个 helper 类 `GraphMetrics` 当中。

(1) 阅读 https://en.wikipedia.org/wiki/Centrality 或查阅其他资料，了解“中心度”的含义，完成 degree centrality、closeness centrality、betweenness centrality 三种中心度的度量函数 API（不需考虑超边）：

- `static double degreeCentrality(Graph<L,E> g, L v)` 计算图 g 中节点 v 的 degree centrality
- `static double degreeCentrality(Graph<L,E> g)` 计算图 g 的总体 degree centrality
- `static double closenessCentrality(Graph<L,E> g, L v)` 计算图 g 中节点 v 的 closeness centrality
- `static double betweennessCentrality(Graph<L,E> g, L v)` 计算图 g 中节点 v 的 betweenness centrality

针对有向图，再增加两个度量函数 API：

- `static double inDegreeCentrality(Graph<L,E> g, L v)` 计算图 g 中节点 v 的 indegree centrality
- `static double outDegreeCentrality(Graph<L,E> g, L v)` 计算图 g 中节点 v 的 outdegree centrality

(2) 阅读 https://en.wikipedia.org/wiki/Distance_(graph_theory)或其他资料，了解距离相关的各指标，完成以下 API（不需考虑超边）（选做，额外加分）：

- `static double distance(Graph<L, E> g, L start, L end)` 节点 `start` 和 `end` 之间的最短距离（需要区分有向图和无向图）
- `static double eccentricity(Graph<L,E> g, L v)`
- `static double radius(Graph<L,E> g)`
- `static double diameter(Graph<L,E> g)`

3.9 第三方 API 的复用（选做，额外加分）

使用 JUNG 包中所提供的 API，为你的四个应用添加可视化功能。在 `GraphVisualizationHelper` 类中实现以下静态方法：

```
public static void visualize(Graph<L,E> g)
```

请自学 http://jung.sourceforge.net 并下载 JAR 包添加至你的项目。

3.10 设计模式应用

在具体设计过程中，请考虑以下建议，尝试着使用设计模式：

(1) 构造 Vertex 对象时，请使用 factory method 设计模式，构造以下工厂类：

```
abstract class VertexFactory
class WordVertexFactory extends VertexFactory
class PersonVertexFactory extends VertexFactory
...
class DirectorVertexFactory extends VertexFactory
```

在 VertexFactory 及各个子类中实现工厂方法：public Vertex createVertex(String label, String type, String[] args)，args 是不同 Vertex 子类型所需的额外属性信息。

客户端使用的时候，采用以下方式调用：

```
WordVertexFactory.createVertex(label, type, args) 或
new WordVertexFactory().createVertex(label, type, args)
```

(2) 构造 Edge 对象时，也使用 factory method 设计模式，构造以下工厂接口或工厂类：

```
abstract class EdgeFactory
class WordNeighborhoodFactory extends EdgeFactory
class FriendTie extends EdgeFactory
...
class SameMovieHyperEdgeFactory extends EdgeFactory
```

在 EdgeFactory 及各个子类中实现工厂方法：public Edge createEdge (String label, String type, List<Vertex> vertices)，vertices 是边的节点集合，简单边是 2 个节点（有向边需考虑次序），超边是多于 2 个节点。

客户端使用的时候，采用以下方式调用：

```
FriendTie.createEdge(label, type, vertices) 或
new FriendTie().createEdge(label, type, vertices)
```

(3) 构造 Graph 对象时，请使用 abstract factory 或 builder 设计模式，针对不同应用中所需的不同类型的节点和不同类型的边，设计 4 个 Graph 工厂类，生成相应的具体 ConcreteGraph 对象。

```
GraphFactory
SocialNetworkFactory
GraphPoetFactory
MovieGraphFactory
NetworkTopologyFactory
```

客户端可直接使用以下方式创建图：

```
Graph g = GraphPoetFactory.createGraph(String filePath) 或
Graph g = new GraphPoetFactory().createGraph(String filePath)
```

这些工厂方法从磁盘文件读取遵循语法的数据，进而将这些数据构造为图结构。具体包括：读取文件、进行基于正则表达式的解析、文件数据的“合法性验证”、如合法则构造图。

(4) 3.4 节给出的用户指令种类较多，你需要为每个指令编写解析器函数，但对四个应用来说，最好不要让应用先识别用户输入的指令类型，再去调用相应的解析器函数。建议使用 façade 设计模式，在 `ParseCommandHelper` 类中封装指令解析和执行功能，对外提供统一的静态方法：

```
public static void parseAndExecuteCommand (String command)
```

(5) 在客户端使用 `GraphMetrics` 类对某节点进行 centrality 度量的时候，有三种不同的算法可以调用（`degreeCentrality`、`closenessCentrality`、`betweennessCentrality`），请使用 Strategy 设计模式进行改造，使客户端可以容易的切换不同的算法。

(6) （选作，额外加分）除了上述建议之外，本次实验中还有很多场合应用其他设计模式，例如：

- 将超边看作是一种复合节点，从而可应用 Composite 模式进行设计。
- 考虑到不同应用中的 `Edge` 对象有不同侧面的特征（如方向、权重、维度、是否超边等），可使用 decorator 模式进行对象构造，让不同应用中的边具有不同的特征值，这就避免了创建表 5 里如此多的子类型。

  `SimplestEdge` 无向无权边
  `DirectedEdge` 有向边
  `WeightedEdge` 加权边

(7) （选作，额外加分）可自由思考，在本实验中尝试着应用 adaptor、proxy、

bridge、Template，iterator、observer、visitor、mediator 等设计模式。

3.11 应用开发

针对四个图应用场景，开发四个可独立运行的程序，可以为命令行程序，也可以为 GUI 程序。

四个应用均包含以下功能：

- 读取包含图数据的磁盘文件，验证其合法性，并解析为图结构；
- 在 GUI 上可视化展示图结构（可选，额外计分）；
- 增加节点、删除节点，修改节点的信息（Label、其他属性）；
- 增加边、删除边、修改边的 label、权重、方向；向超边中增加节点、从超边中去除节点；
- 选定某一节点，计算其三种中心度、eccentricity；如果为有向图中的节点，计算 indegree 和 outdegree 中心度；
- 计算图 degree centrality、radius 和 diameter；
- 选定两个节点，计算它们之间的 distance；
- 执行用户输入的图处理指令（表 2）（可选，额外计分）。
- 其他你认为有价值的功能（需针对四个具体应用分别考虑，额外计分）。

3.12 新的变化（任选两个变化，多做额外计分）

3.11 节中开发出的四个应用，面临着下表的变化。请考查原有的设计，是否能以较小的代价适应这些变化。修改原有设计，使之能够应对这些变化。在修改之前，请确保你之前的开发已经 commit 到 Git 仓库。

修改之后，评估一下你在该节之前所做的 ADT 设计以多大的代价适应了该表中的每一个变化？代价：修改的量

表 6 面临的新变化

应用	需求变化
单词网络	凡是权重<n 的有向边，不应出现在图中，n 为用户确定的参数，正整数，n>1
微博社交网络	为节点增加权重，表征该用户的社交影响力，权值来自于该节点的 indegree centrality
网络拓扑图	增加一种新的节点类型：无线路由器（`WirelessRouter`），其他规则不变
电影网络	增加一种新类型的边，A-B 表示电影 A 和电影 B 具有至少 1 个共同的演员或导演，该边是无向边，权重为共同演员或导演的个数

3.13 项目结构

项目名：Lab3_学号	
src	java 文件
graph	子目录
Graph.java	Graph<L,E>接口
ConcreteGraph.java	Graph 抽象类
GraphPoet.java	四个具体 Graph 类
SocialNetwork.java	
NetworkTopology.java	
MovieGraph.java	
vertex	子目录
Vertex.java	抽象节点类
Word.java	各种具体节点类
Person.java	
Computer.java	
Server.java	
Router.java	
Movie.java	
Actor.java	
Director.java	
edge	子目录
Edge.java	抽象边类
WordEdge.java	各种具体边类
FriendConnection.java	
CommentConnection.java	
ForwardConnection.java	
NetworkConnection.java	
MovieActorRelation.java	
MovieDirectorRelation.java	
SameMovieHyperEdge.java	
helper	子目录
GraphMetrics.java	图度量 API
ParseCommandHelper.java	读取并解析用户输入指令
application	子目录，四个具体应用
GraphPoetApp.java	

	SocialNetworkApp.java	
	NetworkTopologyApp.java	
	MovieGraphApp.java	
test		JUnit 测试程序子目录，可以包含子目录
lib		程序所使用的所有外部库文件，内部不分子目录
doc		实验报告
其他辅助目录		

注：如果上述接口/类无法满足你的设计需要，请自行增加相应的子目录、接口、类，但不要更改上述包结构和接口/类名。

4 实验报告

针对上述四个编程题目，请遵循 CMS 上 Lab3 页面给出的**报告模板**，撰写简明扼要的实验报告。

实验报告的目的是记录你的实验过程，尤其是遇到的困难与解决的途径。不需要长篇累牍，记录关键要点即可，但需确保报告覆盖了本次实验所有开发任务。

注意：

- 实验报告不需要包含所有源代码，请根据上述目的有选择的加入关键源代码，作为辅助说明。
- 请确保报告格式清晰、一致，故请遵循目前模板里设置的字体、字号、行间距、缩进；
- 实验报告提交前，请“目录”上右击，然后选择“更新域”，以确保你的目录标题/页码与正文相对应。

5 提交方式

截止日期：第 10 周周日（2018 年 5 月 6 日）夜间 23:55。截止时间之后通过 Email 等其他渠道提交实验报告和代码，均无效，教师和 TA 不接收，学生本次实验无资格。

源代码：从本地 Git 仓库推送至个人 GitHub 的 Lab3 仓库内。

实验报告：除了随代码仓库（doc）目录提交至 GitHub 之外，还需手工提交至 CMS 实验 3 页面下。

6 评分方式

TA 在第 6-8 周实验课上现场验收：学生做完实验之后，向 TA 提出验收申请，TA 根据实验要求考核学生的程序运行结果并打分。现场验收并非必需，由学生主动向 TA 提出申请。

Deadline 之后，教师使用持续集成工具对学生在 GitHub 上的代码进行测试。教师和 TA 阅读实验报告，做出相应评分。