



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 2019

目录

1	实验目标.....	1
2	实验环境.....	1
3	实验要求.....	2
3.1	基本概念.....	2
3.2	待开发的应用场景.....	3
3.3	基于语法的图数据输入.....	7
3.4	面向复用的设计: <code>CircularOrbit<L,E></code>	9
3.5	面向复用的设计: <code>Track</code>	10
3.6	面向复用的设计: <code>L</code>	11
3.7	面向复用的设计: <code>PhysicalObject</code>	11
3.8	可复用 API 设计.....	12
3.9	第三方 API 复用.....	13
3.10	设计模式应用.....	13
3.11	具体应用开发.....	14
3.12	新的变化.....	16
3.13	项目结构.....	18
4	实验报告.....	18
5	提交方式.....	19
6	评分方式.....	19

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、API 复用

本次实验给定了五个具体应用（径赛方案编排、太阳系行星模拟、原子结构可视化、个人移动 App 生态系统、个人社交系统），学生不是直接针对五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，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（可复用性）和更容易面向各种变化（可维护性）。

本次实验与后续的实验 4、实验 5 是一个整体，随着课程讲授进度逐步增加要求。在完成本实验时，学生可以提前了解实验 4 和实验 5 的具体要求，以便于在本实验的设计和编码过程中有意识的做好准备。不接受“为什么实验 4 和实验 5 要求对实验 3 代码进行这么多的修改”的抱怨。

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/*****

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包内开发，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

本实验需要设计并实现一个抽象数据类型 `CircularOrbit`，对现实中的各类具备“具有一个圆心/中心点、围绕圆心的多条环路/轨道、多个物体分布在圆心和不同环路上”特征的事物进行抽象，并在五个具体应用中使用它。

在设计该 ADT 时，需要对其进行泛型化，考虑中心点物体的不同类型、各环路上物体的不同类型，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。

请为每个你设计和实现的 ADT 撰写 `mutability/immaturity` 说明、`AF`、`RI`、`safety from rep exposure`。给出各 ADT 中每个方法的 `spec`。为每个 ADT 编写测试用例，并写明 `testing strategy`。

3.1 基本概念

多轨道系统 (Circular Orbit)	由唯一一个中心点和多条轨道构成的系统，在中心点和轨道上承载多个物体，物体在轨道上可保持静止或持续运动，物体与物体之间可存在联系。 将该类系统放在直角坐标系中，可看作是以原点 $(0,0)$ 为中心的多个同心圆。
中心点 (Central Point)	最多只有 1 个物体落在中心点，其位置保持不变，是坐标系中的原点 $(0,0)$ 。例如：“太阳”是太阳系的“中心点”所承载的唯一物体，而体育比赛的径赛场地的中心点无物体。
轨道 (Track)	围绕中心点的一条闭合曲线，一般为圆形或椭圆形或其他不规则形状，本实验中统一考虑为标准圆形。轨道的半径是指该轨道与中心点之间的距离。
物体 (Physical Object)	可放置/分配到中心点或轨道的具体事物。一个多轨道系统中可承载多个物体。物体只能出现在中心点或轨道上，不能出现在其他位置。 例如在太阳系里，“太阳”在中心点，八颗行星分处于不同的轨道上。在径赛场地，每条跑道被分配给最多一个“运动员”。在原子模型中，原子核周围的每条电子轨道上可能有多个电子。 有些应用中，物体需要区分（例如太阳系的八颗行星是不同的、竞赛比赛中的运动员是不同的）。有些应用中，

	物体无需区分（例如在原子结构中，各轨道上的电子可看作都是一样的）。
物体在轨道上的位置 (Position)	<p>在某些应用中，需要考虑物体在轨道上的绝对位置（例如在 t 时刻，某物体 a 处于轨道 r，a 的位置是极坐标系中的点 (ρ, θ)，轨道 r 的半径是 ρ，θ 为物体 a 的角度。该极坐标系的原点 $(0, 0)$ 为该轨道系统的中心点。</p> <p>在某些应用中，物体可以在轨道上跳跃，例如某电子从轨道 1 跳到轨道 2。</p> <p>在某些应用中，无需考虑物体的绝对位置，例如原子结构中电子的位置、移动生态系统中 App 的位置、社交网络中用户的位置。</p>
物体之间的关系 (Relation)	<p>在某些应用中，处于中心点的物体和其他物体之间、处于同一轨道的多个物体之间、处于不同轨道的多个物体之间，可能存在特定的关联关系。本实验里将关系限定为二元关系（即只能发生在两个物体之间）、无向关系。如果用可视化的方式表示出来，一条关系表示为连接两点之间的一条线（无箭头）。</p>

3.2 待开发的应用场景

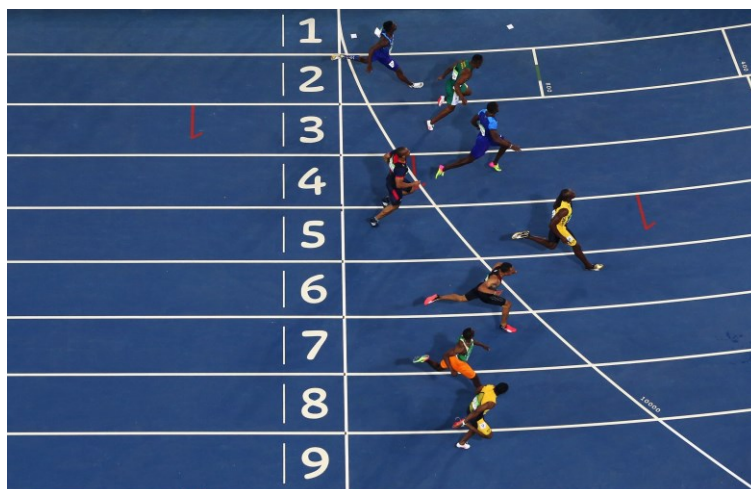
要为以下场景开发 Java 程序，故在设计和实现 ADT 的时候需充分考虑这些应用之间的相似性和差异性，使 ADT 有更大程度的复用和更容易面向各种变化。

本实验你需从以下的(1)和(2)中任选 1 个，(3)为必选，(4)和(5)任选 1 个，亦即：每人至少完成 3 个应用。鼓励完成 4 个或 5 个应用，但不额外计分，实验报告中只需要覆盖你希望评分的 3 个应用即可。

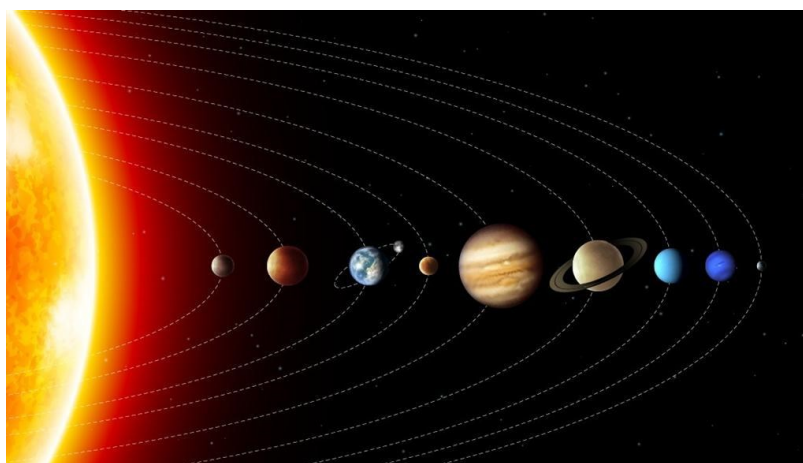
- (1) 径赛场地赛程编排 (**TrackGame**): 径赛是在田径场的跑道上进行的跑和走的竞赛项目的统称，例如 100 米、200 米、400 米、800 米等项目。一个标准的径赛场地通常由两个半径相等的半圆弯道和两个长度相等的直段组成。在本实验中，我们将场地形状简化成圆形。圆形周围划分为等距的若干跑道，在短距离赛跑比赛中每个跑道容纳 1 名运动员（本实验无需考虑长距离赛跑中运动员不区分跑道的情形）。场地中心点无物体，无需考虑各跑道的半径，将各运动员看作独立个体（彼此之间不存在关系），比赛过程中运动员不能切换跑道。

给定一组运动员，通过比赛方案编排算法，将他们分为 n 组，为每组内的每个运动员分配具体的跑道。从而，每一组运动员就形成了一个轨

道系统。



- (2) 行星运动模拟 (**StellarSystem**): 例如, 太阳系由太阳和八大行星构成, 太阳是中心点物体。每个行星与太阳的距离不同, 每个行星的属性也不同 (半径、质量、形态、公转速度、公转方向等)。本实验中, 将行星的运行轨道也简化为圆形, 行星围绕太阳进行固定速率的圆周运动, 周而复始, 需要根据初始位置、轨道半径、速度来计算特定时间行星在轨道上绝对位置。行星不能变换轨道 (不要考虑“流浪地球”的科幻情境☺), 各行星之间是独立的, 无需考虑之间的万有引力关系 (地球不会被木星捕获☺)。



- (3) 原子结构模型 (**AtomStructure**): 一个原子由原子核和绕核运动的电子组成, 原子的质量主要集中在由质子和中子构成的原子核上, 核外分布着电子。在物理学家 Rutherford 于 1909 年提出和 Bohr 于 1913 年改进的核结构模型中, 原子可看作是以原子核为中心的多条电子轨道的结构。本实验中, 将原子核作为整体看作中心点的物体, 无需考虑其内部的质

子和中子。无需考虑电子之间的关系。电子在轨道的分布遵循特定的原理，这里无需考虑电子的绝对位置（海森堡的测不准原理），也无需考虑电子在轨道上的运动。但电子可以在不同轨道之间跃迁。



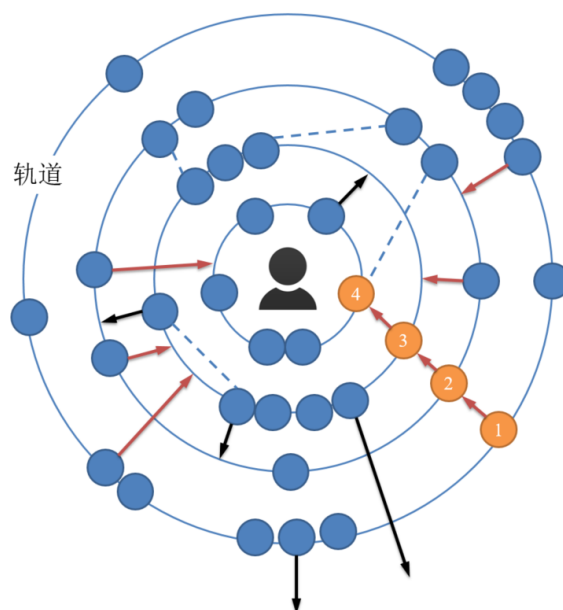
- (4) 个人 App 生态系统 (**PersonalAppEcosystem**): 用户日常使用手机上的各个 App，根据各个 App 的使用时间长度和频度，可以计算出每个 App 与该用户的“亲密度”。根据亲密度的不同，可以构造以该用户为中心的多级轨道结构，称之为“个人 App 生态系统”：围绕用户有多条轨道，与用户亲密度最高的若干 App 在最内层轨道，越往外的轨道上的 App 与用户的亲密度越低。

本实验中，你可以自行决定如何计算 App 的“亲密度”，但一定要考虑使用时长和使用频度这两个因素（使用的绝对时间越长、使用次数越多，则亲密度越高）。你可以自由选择设置轨道的策略（例如：计算出所有 App 的亲密度之后，获得亲密度的值域，然后将该值域均匀划分成 N 个子范围（ N 可以是已知参数，例如 $N=10$ ，也可以动态确定），再将亲密度落在第 i 个子范围内的所有 App 放在第 i 条轨道上。

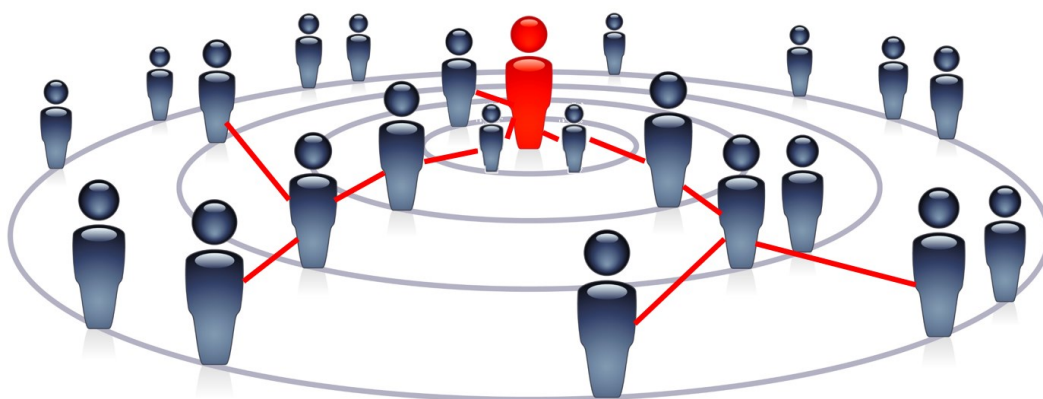
本实验中，根据预先指定的时间单位（小时、天、周、月等），将用户的 App 使用日志划分开来，然后分别构造 App 生态的轨道系统。如果对比相邻时间段的轨道系统，可以看出：随着时间的推移，某些 App 所处的轨道可能发生变化（向内、向外），代表着用户使用 App 习惯的变化（如图中的箭头所示）。图中橙色圆圈给出了一个极端的例子：某个 App 从最外层轨道逐步跳跃到最内层轨道。注意：下图中的所有箭头并非构造生态系统的必要因素，放在这里只是示意。

App 之间存在“合作”关系，亦即：当用户使用 App1 期间会跳转到

App2（例如在京东 App 里选择“微信支付”就会打开微信 App），如图中虚线所示的例子。



- (5) 社交网络的好友分布 (**SocialNetworkCircle**): 在一个社交网络中, 用户 a 有一级好友、二级好友、...、 n 级好友, 其中一级好友是指与 a 直接有社交关系的人, i 级好友是指与 a 无直接的社交关系, 但是与 a 的第 $i-1$ 级好友有直接社交关系的人。因此, 若以 a 为中心点, 则将其各级好友表示为多轨道的结构, 第 i 条轨道上分布着 a 的 i 级好友, 一个人只能出现在一条轨道上。显然的, 相邻的两条轨道上的人之间可存在社交关系, 同一条轨道上的人之间也可能存在社交关系。下图中的红色线表示了社交关系, 但并不完整。



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, 构造多轨道结构。

五个图的输入文件语法如下所示。

可从 https://github.com/rainywang/Spring2019_HITCS_SC_Lab3 下载示例文件, 但建议在你编程的时候, 自行设计更复杂的、包含更多可能情况的输入文件。

统一定义	<ul style="list-style-type: none"> ● label: 由大小写字母或数字构成, 不含有空格和其他符号 ● word: 由大小写字母构成, 不含有空格 ● sentence: 由大小写字母或数字构成, 可含有空格 ● number: 大于 10000 的数字按科学记数法表示 (例如 1.9885e30 表示 $1.9885 * 10^{30}$, 但 e 之前数字的整数部分必须在 1 到 9 的范围内, e 之后的数字只能是大于 3 的正整数), 小于 10000 的数字直接给出 (例如 5912, 103.193), 不能用科学计数法。小数点位数不限制。 <p>注: 输入文件中各行的次序是不确定的, 请不要让你的程序依赖于该次序; 文件中可能存在空行, 各行中不同分量之间可能存在空格, 这都是合法的, 程序需要能够正确处理。请不要使用未在本表中列出的任何标准进行文件解析 (例如: 观察到示例文件中某个属性的长度都为 3 或者小数点都保留 2 位)。</p>
Track Game	<ul style="list-style-type: none"> ● Game ::= 100 200 400 表明比赛项目, 只能这三个选择之一 ● NumOfTracks ::= 跑道数目, 4 到 10 之间的整数 ● Athlete ::= <姓名, 号码, 国籍, 年龄, 本年度最好成绩> 代表一个运动员, 其中姓名是 word, 号码和年龄是正整数, 国籍是三位大写字母, 成绩是最多两位整数和必须两位小数构成 (例如 9.10、10.05)。 ● Athlete ::= <Bolt,1,JAM,38,9.88> 运动员 1 ● Athlete ::= <Lewis,2,USA,39,10.00> 运动员 2 ● ...
Stellar System	<ul style="list-style-type: none"> ● Stellar ::= <名称, 半径, 质量> 处于中心点的恒星, 名称为 label 类型, 半径的单位为 km (无需在文件中表达), 质量的单位为 kg (也无需在文件中出现), 二者均为 number 类

	<p>型</p> <ul style="list-style-type: none"> ● Planet ::= <名称,形态,颜色,行星半径,轨道半径,公转速度,公转方向,初始位置的角度>, 其中名称、形态和颜色均为 label 类型, 行星半径、轨道半径、公转速度均为 number 类型, 初始位置的角度在 [0,360) 之间, 可以为整数或带小数, 公转方向为 CW 或 CCW, CW 为顺时针, CCW 为逆时针。 例如: Planet ::= <Earth,Solid,Blue,6378.137,1.49e8,29.783,CW,0> 表示一颗叫做 Earth 的蓝色固体星球。 ● ...
Atom Structure	<ul style="list-style-type: none"> ● ElementName ::= 元素名称, 例如 Na、C、O、Ca, 最多两位字母, 第一位大写, 第二位小写 (若有) ● NumberOfTracks ::= 轨道数目, 正整数 ● NumberOfElectron ::= 轨道号 1/该轨道上的电子数量; 轨道号 2/该轨道上的电子数量; ... 例如: NumberOfElectron ::= 1/2;2/6 表示第 1 条轨道上 2 个电子, 第 2 条轨道上 6 个电子
Personal App Ecosystem	<ul style="list-style-type: none"> ● User ::= 姓名, 类型为 Label ● App ::= <App 名称,公司,版本,"功能描述","业务领域">, 描述 App 的基本信息, 其中 App 名称和公司为 label 类型, 版本是类似于使用诸如“.”、“_”、“-”分割的 label, 例如 v3.2-3_001、2.30 等。功能描述和业务领域均带引号, 类型是 sentence 类型。 ● InstallLog ::= <date,time,App 名称>, 表示在该日期该时间将该 App 安装到手机上。date 为“YYYY-MM-DD”的形式, time 为“hh:mm:ss”的形式。 ● UsageLog ::= <date,time,App 名称,duration>, 一条日志, 表示用户在该日期该时间开始使用该 App, 使用时长为 duration, duration 单位为分钟, 正整数。 ● UninstallLog ::= <date,time,App 名称>, 表示在该日期该时间将该 App 从手机上卸载。 注意: 一个 App 可以有多条 install、usage 和 uninstall 的记录。 ● Relation ::= <App 名称 1,App 名称 2> 注: 该关系与时间无关, 故其中未描述时间信息。 ● ...

	<ul style="list-style-type: none"> ● Period ::= Hour Day Week Month 表示以多大的时间范围来生成个人 App 生态系统的模型，例如 Period ::= Day 表示根据每天（自然天）的日志生成一个模型，第二天的日志生成另一个模型。Hour、Week、Month 均按日历时间划分，例如 1 月/2 月、15-16 点/16-17 点等。
Social Network Circle	<ul style="list-style-type: none"> ● CentralUser ::= <姓名,年龄,性别>, 姓名类型为 label, 年龄为正整数, 性别取 M F 之一。 ● Friend ::= <姓名,年龄,性别> ● SocialTie ::= <用户 1 姓名,用户 2 姓名,社交亲密度>, 亲密度是范围在 (0,1] 内的小数, 最多 3 位小数位, 用户 1 和用户 2 可以是 CentralUser 或 Friend。

3.4 面向复用的设计: **CircularOrbit<L,E>**

设计新的接口 **CircularOrbit<L,E>**, 其中 **L** 和 **E** 分别代表多轨道系统的中心点物体类型和轨道物体的类型。

从你所选定的应用中进行 ADT 抽象, 设计 **CircularOrbit** 应提供的接口方法。需要写清楚为何设计每一个方法。例如:

- 创建一个空的 **CircularOrbit** 对象
- 增加一条轨道、去除一条轨道
- 增加中心点物体
- 向特定轨道上增加一个物体（不考虑物理位置）
- 增加中心点物体和一个轨道物体之间的关系
- 增加两个轨道物体之间的关系
- 从外部文件读取数据构造轨道系统对象
- ...

注意: 绝不仅仅只有这些操作, 需进一步抽象设计更完整的共性操作。例如: 原子结构中的电子可以跃迁, 社交网络中的人的地位可以变化（当增加两个用户之间的边的时候, 某个用户的轨道位置可能随之变化）, 因此可以设计一个操作（spec 仅供参考）将 **object** 从当前所在轨道迁移到轨道 **t**:

```
public void transit (E object, Track t);
```

再例如: 如果某轨道物体需考虑其绝对位置, 并可从一个位置移动到另一个位置, 则另一个用于支持共性行为的操作是（spec 仅供参考）, 将 **object** 从当前位置移动到新的 **sitha** 角度所对应的位置:

```
public void move(E object, double sitha);
```

进而，设计一个类 `ConcreteCircularOrbit` 来实现它。它的 `rep` 请自行设计。你也可以参照 Lab2 中 P1 的做法，为其设计多种 `rep` 方案，即开发多个实现类（可自行命名）。

提示：你的 `rep` 要表达以下方面：(1) 中心点物体；(2) 一组轨道；(3) 每条轨道上的一组物体；(4) 中心点物体与轨道物体之间的关系；(5) 轨道物体之间的关系。选择不同的数据结构存储这些信息，即产生不同的 `rep` 和不同的 ADT 实现方案。

在 `ConcreteCircularOrbit` 类中，需实现 `CircularOrbit` 中的所有方法。

在你所选定的至少三个应用中，你需要分别从 `ConcreteCircularOrbit` 派生出更具体的、面向应用的类，例如：

- `TrackGame`
- `StellarSystem`
- `AtomStructure`
- `PersonalAppEcosystem`
- `SocialNetworkCircle`

在这些具体的类中，你可以增加新的操作，以实现不同应用中的某些特殊功能（这些功能可能无法被 `CircularOrbit` 中的方法所完全覆盖）。

这是一个 `mutable` 的 ADT，请注意其安全性！

为以上完成的 `CircularOrbit`、`ConcreteCircularOrbit` 和三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）。后续各节同样要求，不再重复。

3.5 面向复用的设计：Track

定义 `ConcreteCircularOrbit<L,E>` 的 `rep` 使用的 `Track`，即轨道。

需要设计并实现该类的 `rep`（例如半径）和必要的操作。

这是一个 `immutable` 的 ADT，请务必注意其安全性！

注意：`Track` 作为 `ConcreteCircularOrbit` 内部使用的类，最好不要暴露给应用层的 `client` 端。

3.6 面向复用的设计: L

`CircularOrbit<L,E>`中的 L, 可以是你所设计的任何表征中心点物体的类。

对五个应用来说, 其中心点物体所需关注的属性可从 3.2 和 3.3 节的描述中抽取。请据此完成 L 的具体类的设计和实现, 都是 immutable 的。

- `TrackGame`: 无中心点物体
- `StellarSystem`: 中心点物体为恒星
- `AtomStructure`: 中心点物体为原子核
- `PersonalAppEcosystem`: 中心点物体为人
- `SocialNetworkCircle`: 中心点物体为人

3.7 面向复用的设计: PhysicalObject

实现 `ConcreteCircularOrbit<L,E>`中的 E, 即代表分布在不同轨道上的物体类 `PhysicalObject`。可以是接口, 也可以是抽象类。

需要定义各应用的共性数据表示 `rep`, 以及作用在其上的操作。

考虑到不同应用中包含不同类型的轨道物体, 需从 `PhysicalObject` 派生子类型, 通过 `override` 实现 `PhysicalObject` 中的各个接口方法或抽象方法, 也可根据应用需求来增加子类的特有属性和方法。

- `TrackGame`: 轨道物体为运动员
- `StellarSystem`: 轨道物体为行星
- `AtomStructure`: 轨道物体为电子
- `PersonalAppEcosystem`: 轨道物体为移动 App
- `SocialNetworkCircle`: 轨道物体为人

在实现这些 ADT 和具体类的时候, 你也可以考虑它们之间的相似性, 做进一步的抽象、提高复用度。例如: 第 1 个应用中的轨道物体、第 4 个应用中的轨道物体、第 5 个应用中的中心点物体和轨道物体都是“人”, 具有相似的属性, 是否可以复用?

这也是 immutable 的 ADT, 请务必注意安全!

进一步, 你需要考虑为“关系”设计 ADT, 用于支持中心物体与轨道物体之间的关系、两个轨道物体之间的关系。

3.8 可复用 API 设计

基于你所选定的三个应用构造的具体的多轨道结构，可在其上开展一系列共性计算。请针对 `ConcreteCircularOrbit<L,E>` 设计 API 并实现具体代码，包含在 `CircularOrbitAPIs.java` 中。

- 计算多轨道系统中各轨道上物体分布的熵值。有关熵的概念可参见 [https://en.wikipedia.org/wiki/Entropy_\(information_theory\)](https://en.wikipedia.org/wiki/Entropy_(information_theory))。通俗的说，如果所有物体都分布在同一条轨道上，其熵值最低；如果所有物体均匀的分布在每一条轨道上，整个系统的熵值最高。

```
double getObjectDistributionEntropy(CircularOrbit c)
```

- 计算任意两个物体之间的最短逻辑距离。这里的逻辑距离是指：`e1` 和 `e2` 之间通过最少多少条边 (`relation`) 即可连接在一起。两个物体之间若无关系，则距离无穷大。（你在 Lab1 的 P3 里已经实现了类似功能）

```
int getLogicalDistance (CircularOrbit c, E e1, E e2)
```

- 计算任意两个物体之间的物理距离。若物体有具体位置，则可在直角坐标系里计算出它们之间的物理距离。

```
double getPhysicalDistance (CircularOrbit c, E e1, E e2)
```

- 计算两个多轨道系统之间的差异。请自行设计 `Difference` 这个 ADT，需要表达出：轨道数的差异、具有相同次序的轨道上物体数量的差异和物体的差异（如果物体不需要区分，则无需给出物体的差异，只需给出数量差异）。`c1` 和 `c2` 必须为同类型的轨道才可以比较（例如不能比较一个太阳系和一个 100 米比赛）。

```
Difference getDifference (CircularOrbit c1, CircularOrbit c2)
```

以下给出两个例子来进一步说明“差异”需要表达什么内容：

- (1) 如果两个原子 `Rb` 和 `Er`，其电子分布分别是 `1/2;2/8;3/18;4/8;5/1` 和 `1/2;2/8;3/18;4/30;5/8;6/2`，那么二者之间的 `Difference` 对象应该类似于：

轨道数差异： -1

轨道 1 的物体数量差异： 0

轨道 2 的物体数量差异： 0

轨道 3 的物体数量差异： 0

轨道 4 的物体数量差异： -22

轨道 5 的物体数量差异： -7

轨道 1 的物体数量差异： -2

（对原子来说，其轨道上的所有电子不需要区分，故物体的差异无需表达）

(2) 两个 100m 比赛方案，第一组比赛 8 人参赛 ($a \setminus b \setminus c \setminus d \setminus e \setminus f \setminus g \setminus h$)，第二组比赛 5 人参赛 ($a/b/d/e/h$)，那么二者之间的 **Difference** 对象应该类似于：

轨道数差异：3

轨道 1 的物体数量差异：0；物体差异：无

轨道 2 的物体数量差异：0；物体差异：无

轨道 3 的物体数量差异：0；物体差异：c-d

轨道 4 的物体数量差异：0；物体差异：d-e

轨道 5 的物体数量差异：0；物体差异：e-h

轨道 6 的物体数量差异：1；物体差异：f-无

轨道 7 的物体数量差异：1；物体差异：g-无

轨道 8 的物体数量差异：1；物体差异：h-无

对应用 4 和应用 5 来说，一个轨道上有多个物体且需要区分，这种情况下表达“物体差异”可以用类似于集合的形式，例如 $\{a, b, c\} - \{d, f\}$ 意味着前者的该轨道上有 $a \setminus b \setminus c$ 三个物体没有在后者的同序号轨道上出现，而后者有 $d \setminus f$ 两个物体没有在前者出现。

3.9 第三方 API 复用

使用 JFreeChart、Gephi、JGraph/JGraphX/JGraphT、JUNG、G、甚至 Lab1 中所使用的 Turtle Graphics 等第三方 Java 库所提供的绘图 API，为你的应用添加可视化功能。尽量不要自己从 0 开始实现可视化，尽可能复用第三方 API。

在 `CircularOrbitHelper` 类中实现以下静态方法：

```
public static void visualize(CircularOrbit c)
```

请自学你所选定的第三方绘图库，并下载 jar 包添加至你的 lib 目录。

可视化的难度较大，请至少能达到以下要求：(1) 可以绘制出中心点和一组同心圆轨道；(2) 可以绘制出分布于轨道上的物体，无需考虑其绝对位置和动态性（例如跃迁、运动等）；(3) 可以绘制出物体之间的二元无向联系。

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 `Track`、`PhysicalObject` 等对象时，请使用 `factory method` 设计模式。设计其工厂类并实现之。
- (2) 构造 `ConcreteCircularOrbit` 对象时，针对不同应用中所需的不同类型的 L 和 E，使用 `abstract factory` 或 `builder` 设计模式。

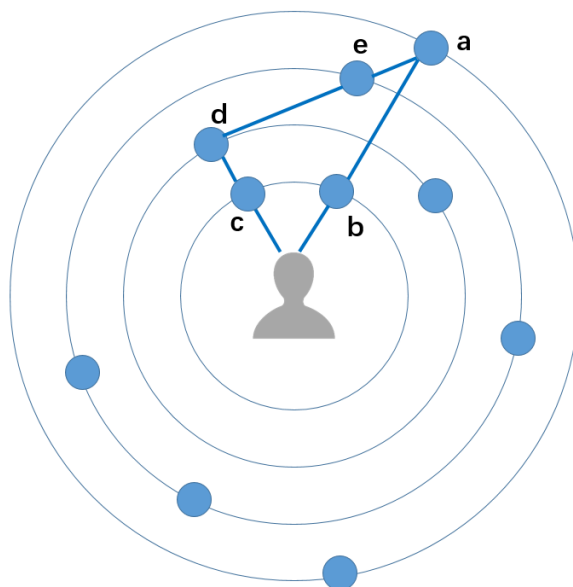
- (3) 请使用 Iterator 设计模式，设计迭代器，客户端可在遍历 `CircularOrbit` 对象中的各 `PhysicalObject` 对象时使用，遍历次序为：从内部轨道逐步向外、同一轨道上的物体按照其角度从小到大的次序（若不考虑绝对位置，则随机遍历）。
- (4) 在设计可复用 API 时，请遵循 `façade` 设计模式，将所有 API 放置在 helper 类 `CircularOrbitAPIs` 当中。
- (5) 在 `TrackGame` 应用中，请使用 `strategy` 设计模式实现多种不同的比赛方案编排策略，并可容易的切换不同的算法。——如果你没有选择该应用，则需要完成(6)。
- (6) 在 `StellarSystem` 应用中，请使用 `decorator` 设计模式，为某些行星增加一颗或多颗“卫星”——原需求中，轨道上的行星是单个的；在该新需求中，轨道上的行星可能携带一颗或多颗卫星。——如果你没有选择该应用，则需要完成(5)。
- (7) 在 `AtomStructure` 应用中，请使用 `state` 和 `memento` 设计模式管理电子跃迁的状态，并可进行状态的恢复。意即：可保存电子每次跃迁前后的轨道信息。
- (8) 可自由思考，在本实验中尝试着应用 `adaptor`、`proxy`、`bridge`、`composite`、`template`、`observer`、`visitor`、`mediator` 等设计模式。

3.11 具体应用开发

针对你选定的三个应用场景，开发一个程序，实现以下功能：

- (1) 读取数据文件，生成多轨道结构（3.3 节的要求）；
- (2) 在 GUI 上可视化展示文件读入的或程序产生的多轨道结构（3.9 节的要求）；
- (3) 用户提供必要信息，以增加新轨道、向特定轨道上增加物体；
- (4) 用户提供必要信息，以从特定轨道上删除物体、删除整条轨道；
- (5) 计算多轨道系统中各轨道上物体分布的熵值；
- (6) 每个应用所需的特殊功能：
 - **TrackGame:** **功能 1:** 给定 n 个运动员，自动编排比赛方案（分组、排道次），需实现至少 2 种方案（方案 1：完全随机；方案 2：根据运动员的本年度最好成绩从高到低排序，排名越靠前，则出场越晚，且更占据中央的赛道）；输出比赛方案（表格或可视化输出）；**功能 2:** 手工调整比赛方案（为两个选手更换赛道、更换组）。
 - **StellarSystem:** **功能 1:** 给定初始时刻（读入文件时）各行星的位置，计算输出 t 时刻各行星的位置；**功能 2:** 计算恒星与某颗行星之间、两颗行星之间的物理距离；**功能 3:** 可视化模拟行星运动。

- **AtomStructure**: 给定源轨道、目标轨道，模拟电子跃迁。
 - **PersonalAppEcosystem**: **功能 1**: 从个人使用 App 日志中恢复该结构。
注意: 从日志中无法直接获取有多少条轨道、各 App 分布在什么轨道上, 需要进行计算。具体计算方法见 3.2 节描述, 但是要确保不同时间段的轨道设置是完全一样的。**功能 2**: 获取两个时间段的轨道系统结构差异。
功能 3: 计算两个 App 之间的逻辑距离。
 - **SocialNetworkCircle**: **功能 1**: 从社交关系日志中恢复该结构, 判定每个用户是在哪个轨道上; **功能 2**: 计算某个处于第一条轨道上的某个好友的“信息扩散度”(即通过该好友可以间接认识多少个朋友, 需要考虑社交关系的亲密度因素, 具体计算方法请自定); **功能 3**: 增加/删除一条社交关系(然后重新调整该图的结构); **功能 4**: 计算两个轨道上的用户之间的逻辑距离。
- (7) 判断多轨道系统的合法性:
- **TrackGame**: 无中心点物体; 跑道数为 4-10 之间; 每一组的人数不能超过跑道数、每一组的每条跑道里最多 1 位运动员(但可以没有运动员); 如果第 n 组的人数少于跑道数, 则第 0 到第 n-1 各组的人数必须等于跑道数; 同一个运动员只能出现在一组比赛中。
 - **StellarSystem**: 中心点必须有一颗恒星; 一个轨道上只能有一个行星且不能没有行星; 相邻轨道的半径之差不能小于两颗相应行星的半径之和。
 - **AtomStructure**: 无。
 - **PersonalAppEcosystem**: 如果某个应用在某个时间段内被安装或被卸载, 那么它应该出现在该时间段的轨道系统中; 如果某个应用在某个时间段之前已被卸载, 则不应出现在该时间段的轨道系统中; 如果某个应用在某个时间点被安装且后续没有被卸载, 那么它就应始终出现在后续各个时间段的轨道系统中; 如果应用 1 和应用 2 分别出现在轨道 t1 和 t2 且前者更靠近中心点的用户, 则在该时间段内应用 1 的“亲密度”必然大于应用 2 的“亲密度”。
 - **SocialNetworkCircle**: 如果某个人出现在第 n 条轨道上, 那么他和中心点的人之间的最短路径是 n。例如下图 a 用户, 从中心点到它的最短路径是通过 b(而非 c-d-e-a), 那么 a 的位置应该是轨道 2(d 所在的轨道)。如果某个人与中心点用户不连通, 则不应出现在轨道系统中。



(8) 其他你认为有价值的功能

针对你所选定的三个具体应用分别考虑，自行扩展。

注：“开发一个程序”意味着：运行该程序，用户从三个选项中选择具体应用，然后用户选择执行该应用的具体功能。

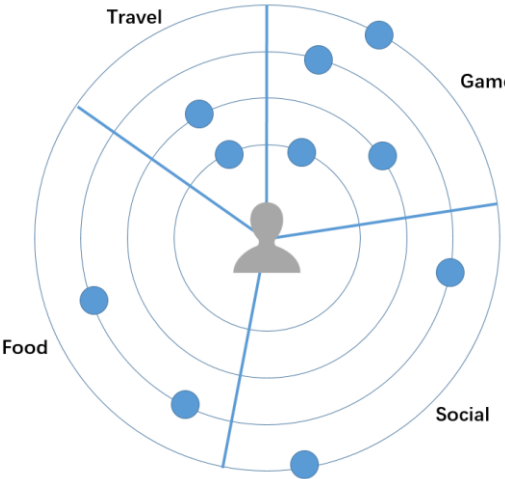
3.12 新的变化

3.11 节中开发出的应用，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，**在新分支“312change”**上完成本节任务。

在具体修改之前，请根据前面的 ADT 与应用设计，先大致估算一下你的程序需要变化多大才能适应这些变化。修改之后，再相对精确的度量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？这里的“代价”可用**“代码修改的量”**和**“修改所耗费的时间”**加以估计。

注意：提交到仓库之后，master 分支需仍然指向本节任务之前的结果，TA 会到仓库的 **312change** 分支上检查本节任务。

应用	需求变化
TrackGame	可支持接力比赛（每个跑道一支队伍，每支队伍 4 人），从而每个跑道上可以出现 4 个物体，但无需考虑 4 个物体的绝对位置

StellarSystem	行星轨道变成符合实际的椭圆形轨道（意即轨道的属性从“半径”变化成描述椭圆的多个参数）
AtomStructure	原子核需要表达为多个质子和多个中子，即处于中心点的物体可以是多个物体构成的集合
PersonalAppEcosystem	<p>为轨道系统增加“扇区”特性，根据其中包含的所有 App 的“业务领域”属性进行聚类，隶属于同一业务领域的所有 App 放在同一个扇区里。例如在下图中分为了四个领域：Game、Social、Food、Travel。</p> 
SocialNetworkCircle	<p>为社交关系增加方向性，多轨道中只包含从中心点向外的社交关系。此时输入文件中 <code>SocialTie ::= <用户 1, 用户 2, 社交亲密度></code> 表示从用户 1 指向用户 2 的社交关系。如果有代表着从轨道物体到中心点用户的关系，则自动忽略它们；如果有代表着从外层轨道向内层轨道的关系，也自动忽略它们。</p>

以下给出了 Git 的相关操作指南。

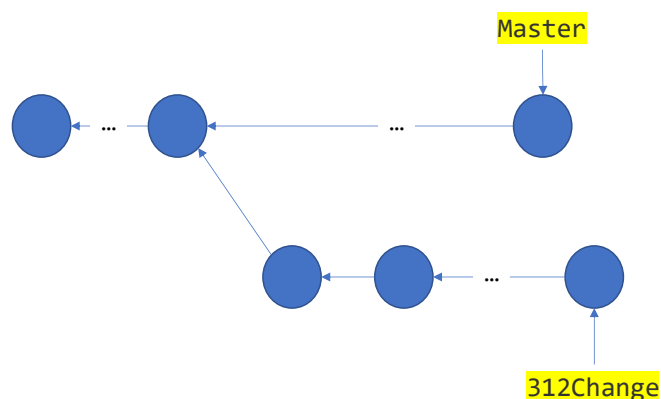
```

...最初在 master 上工作...
git checkout -b 312change 创建新分支
...按上面的要求进行代码修改...
git add *
git commit -m "312change"在该分支上提交
git checkout master          切换回 master 分支
...请不要使用 git merge 312change 进行合并修改
...请不要使用 git branch -d 312change 删除分支

```

你的 Git 仓库中的 Object Graph 应类似于下图所示。其中上方的 master 分支

包含了本节之前的开发结果，下方的分支是针对本节的变化需求的开发结果。



3.13 项目结构

项目名: Lab3-学号

<code>src</code>	<code>java</code> 文件自行组织包结构，做到有序、清晰 以下给出了示例，请参考，但可自行调整，必要时可以增加子目录
<code>circularOrbit</code>	包含 3.4 节各 ADT 相关的代码 <code>...java</code>
<code>track</code>	包含 3.5 节各 ADT 相关的代码 <code>...java</code>
<code>centralObject</code>	包含 3.6 节各 ADT 相关的代码 <code>...java</code>
<code>physicalObject</code>	包含 3.7 节各 ADT 相关的代码 <code>...java</code>
<code>APIs</code>	包含 3.8 节和 3.9 节的相关代码 <code>CircularOrbitAPIs.java</code> <code>CircularOrbitHelper.java</code>
<code>applications</code>	包含 3.11 节各应用的相关代码 <code>...java</code>
<code>otherDirectory</code>	你所需的其他目录，可自行增加
<code>test</code>	JUnit 测试程序目录，与 <code>src</code> 结构保持一致
<code>lib</code>	程序所使用的的所有外部库文件
<code>doc</code>	实验报告
	其他辅助目录

注: 如果手册中给出的接口/类无法满足你的设计需要, 请自行增加相应的包、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。
- 实验报告文件可采用 Word 或 PDF 格式，命名规则：Lab3-学号-Report。

5 提交方式

截止日期：第 10 周周日夜间 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，教师和 TA 对学生在 GitHub 上的代码进行测试，阅读实验报告，做出相应评分。