

**2024年春季学期  
计算机学院《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 王渊 |
| 学号 | 8200880108 |
| 班号 | 2137102 |
| 电子邮件 | 1029912611@qq.com |
| 手机号码 | 13694876416 |

**目录**

[1 实验目标概述 1](#_Toc3922069)

[2 实验环境配置 1](#_Toc3922070)

[3 实验过程 1](#_Toc3922071)

[3.1 待开发的三个应用场景 1](#_Toc3922072)

[3.2 基于语法的图数据输入 1](#_Toc3922073)

[3.3 面向复用的设计：CircularOrbit<L,E> 1](#_Toc3922074)

[3.4 面向复用的设计：Track 2](#_Toc3922075)

[3.5 面向复用的设计：L 2](#_Toc3922076)

[3.6 面向复用的设计：PhysicalObject 2](#_Toc3922077)

[3.7 可复用API设计 2](#_Toc3922078)

[3.8 图的可视化：第三方API的复用 2](#_Toc3922079)

[3.9 设计模式应用 2](#_Toc3922080)

[3.10 应用设计与开发 2](#_Toc3922081)

[3.10.1 TrackGame 2](#_Toc3922082)

[3.10.2 StellarSystem 2](#_Toc3922083)

[3.10.3 AtomStructure 2](#_Toc3922084)

[3.10.4 PersonalAppEcosystem 2](#_Toc3922085)

[3.10.5 SocialNetworkCircle 2](#_Toc3922086)

[3.11 应对应用面临的新变化 2](#_Toc3922087)

[3.11.1 TrackGame 3](#_Toc3922088)

[3.11.2 StellarSystem 3](#_Toc3922089)

[3.11.3 AtomStructure 3](#_Toc3922090)

[3.11.4 PersonalAppEcosystem 3](#_Toc3922091)

[3.11.5 SocialNetworkCircle 3](#_Toc3922092)

[3.12 Git仓库结构 3](#_Toc3922093)

[4 实验进度记录 3](#_Toc3922094)

[5 实验过程中遇到的困难与解决途径 3](#_Toc3922095)

[6 实验过程中收获的经验、教训、感想 4](#_Toc3922096)

[6.1 实验过程中收获的经验和教训 4](#_Toc3922097)

[6.2 针对以下方面的感受 4](#_Toc3922098)

# 实验目标概述

目标：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

# 实验环境配置

环境配置没有遇到问题。

# 实验过程

## 待开发的三个应用场景

首先请列出你要完成的具体应用场景（至少3个，1和2中选一，3必选，4和5中选一，鼓励完成更多的应用场景）。

* TrackGame
* AtomStructure
* SocialNetworkCircle

相同之处：

首先，他们都是多轨道结构，它们的轨道十分类似，都可以看成一个只有名字和半径属性的圆。它们的轨道物体也有一些相似之处：比如都要求有名字。他们都要求有对于轨道、轨道上的物体的一系列增删改查的操作的要求。

不同之处：

有的要求各个物体之间有有向或无向关系，有的各个物体之间没有区别比如电子，有的有区别比如运动员，电子一个轨道可以有多个等。

## 基于语法的图数据输入

使用的正则表达式如下：

track game：

final Pattern ATHLETE\_PATTERN = Pattern.*compile*("Athlete\\s\*::=\\s\*<(\\w+),(\\w+),(\\w+),(\\w+),([\\w,.]+)>");  
final Pattern GAME\_PATTERN = Pattern.*compile*("Game\\s\*::=\\s\*([\\w,.]+)");  
final Pattern TRACK\_PATTERN = Pattern.*compile*("NumOfTracks\\s\*::=\\s\*(\\w+)");

AtomStructure：

String elementPattern = "ElementName\\s\*::=\\s\*([a-zA-Z]+)";  
String trackPattern = "NumberOfTracks\\s\*::=\\s\*(\\d+)";  
String electronPattern = "NumberOfElectron\\s\*::=\\s\*([\\d+/,;]+)";

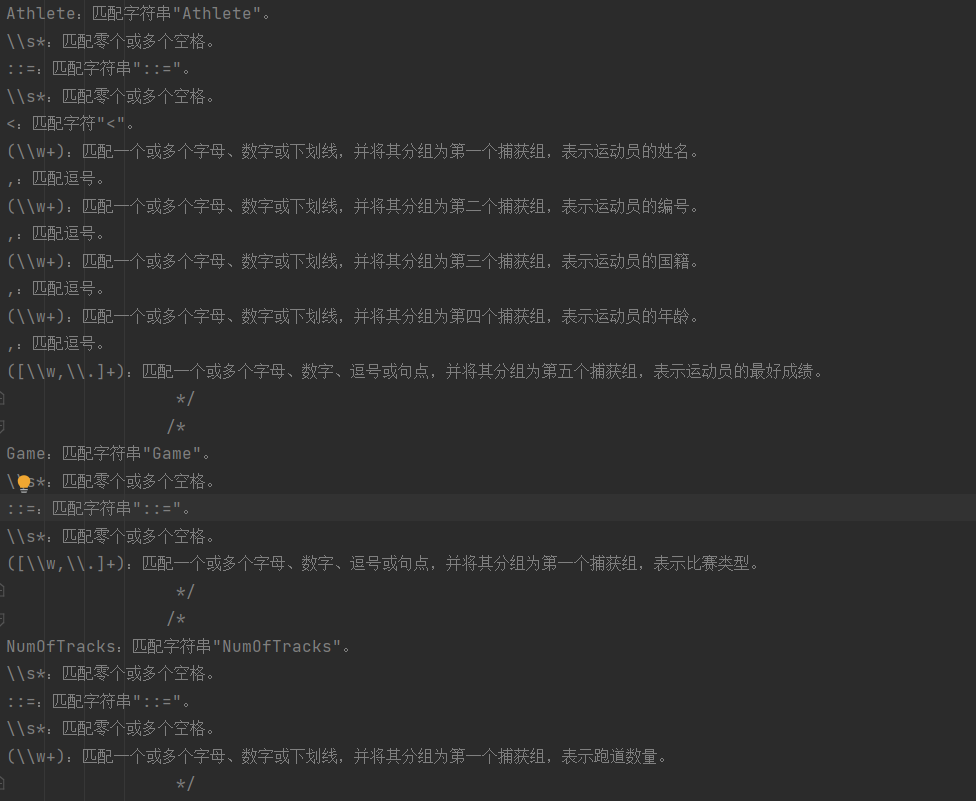
SocialNetworkCircle：

String centralUserPatternString = "CentralUser\\s\*::=\\s\*<(\\w+),\\s\*(\\d+),\\s\*([MF])>";

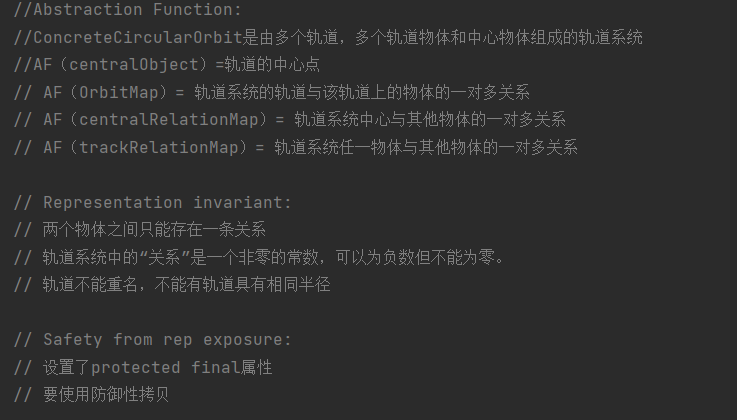
String friendPatternString = "Friend\\s\*::=\\s\*<(\\w+),\\s\*(\\d+),\\s\*([MF])>";

String socialTiePatternString = "SocialTie\\s\*::=\\s\*<(\\w+),\\s\*(\\w+),\\s\*([01]\\.{1}\\d+)>";

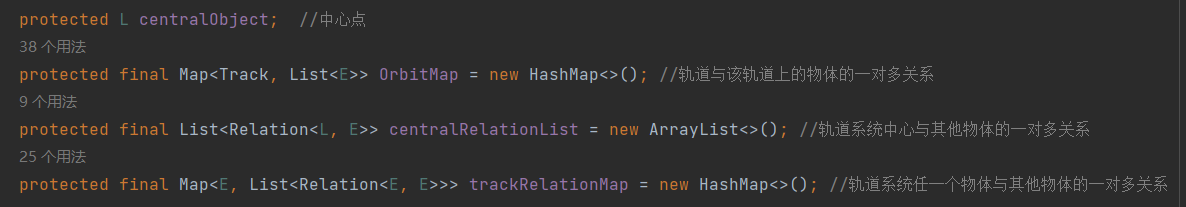
为了方便记忆和理解部分注释解释如下：



## 面向复用的设计：CircularOrbit<L,E>



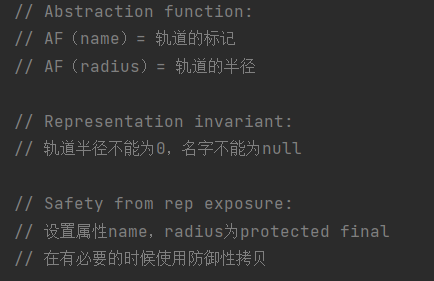
定义的数据结构如下：



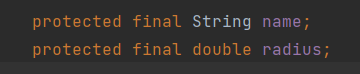
定义的方法如下：

|  |  |
| --- | --- |
| 方法： | 功能： |
| setCentralObject(L centralObject) | 设置中心物体 |
| addTrack(Track t) | 增加一条轨道 |
| getCentralObject() | get轨道中心物体 |
| removeTrack(Track t) | 删除一条轨道 |
| getTrackNum(); | 返回轨道数目 |
| getTrackObjectNum(Track t) | 统计一条轨道有多少物体 |
| getObjectOnTrack(Track t) | 返回一条轨道的所有物体 |
| addObjectToTrack(Track t, E object) | 添加一个物体到某个轨道上 |
| removeObjectOnTrack(Track t, E object) | 从某个轨道上删除一个物体 |
| addtrackRelation(E object1, E object2, double weight) | 为两个轨道系统上的物体添加关系（距离） |
| addCentralRelation(E object, double weight) | 为轨道系统的中心物体和某个轨道上的物体添加关系（距离） |
| removetrackRelation(E object1, E object2) | 删除轨道系统两个物体的关系 |
| removeCentralRelation(E object) | 为轨道系统的中心物体和某个轨道上的物体删除关系 |
| transit(E oldObj, E newObj, Track t) | 将 oldObj 从当前所在轨道 迁移到 t |
| move(E oldObject, E newObject) | 将 object 从当前位置移动到新的角度所对应的位置 |
| calculateEntropyOfOrbit() | 计算该轨道系统的信息熵 |
| getLogicalDistance(E object1, E object2) | 获得两物体间逻辑距离 |
| contains(E object) | 判断当前轨道系统是否包含某个物体 |
| getTrackOfObject(E object) | 返回某个物体object所在的轨道对象Track |
| checkRep() | 检查合法性 |
| getCentralConnectedObject() | 返回与中心连接的物体 |
| getTrackConnectedObject(E object) | 返回与某个轨道物体连接的所有物体 |
| getDifference(CircularOrbit<L, E> that) | 比较当前轨道系统和轨道系统c |
| getSortedTracks() | 获得当前轨道系统包含的所有轨道的按半径排列成的list |
| drawPicture() | 将轨道可视化 |
| checkOrbitAvailable() | 判断系统的合法性(主要用于继承给子类) |
| MyIterator(Map<Track, List<E>> orbitMap) | 迭代器 |

## 面向复用的设计：Track

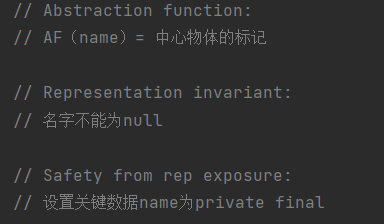


有两个域 name名字 和radius半径，实现Comparable依靠半径排序。



|  |  |
| --- | --- |
| 方法 | 功能 |
| getInstance(String name, double radius) | 静态工厂方法，构造名字为track，半径自定义的轨道 |
| checkRep() | 检查名字非空，半径大于0 |
| int compareTo(Track t) | 重写Comparable接口的compareTo方法，依据轨道半径对轨道进行大小比较，进而排序 |
| getName() | 获得name |
| getRadius() | 获得轨道半径 |

## 面向复用的设计：L

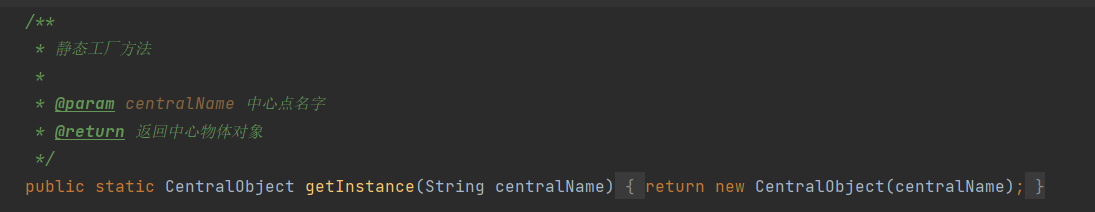


只有一个域名字，方法：

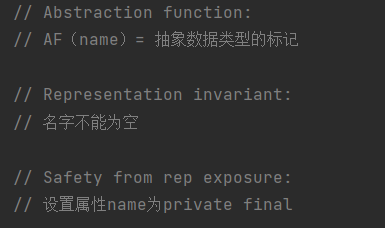
Name的getter方法和CentralObject(String name) 构造方法

以及checkRep() 方法检查名字非空。

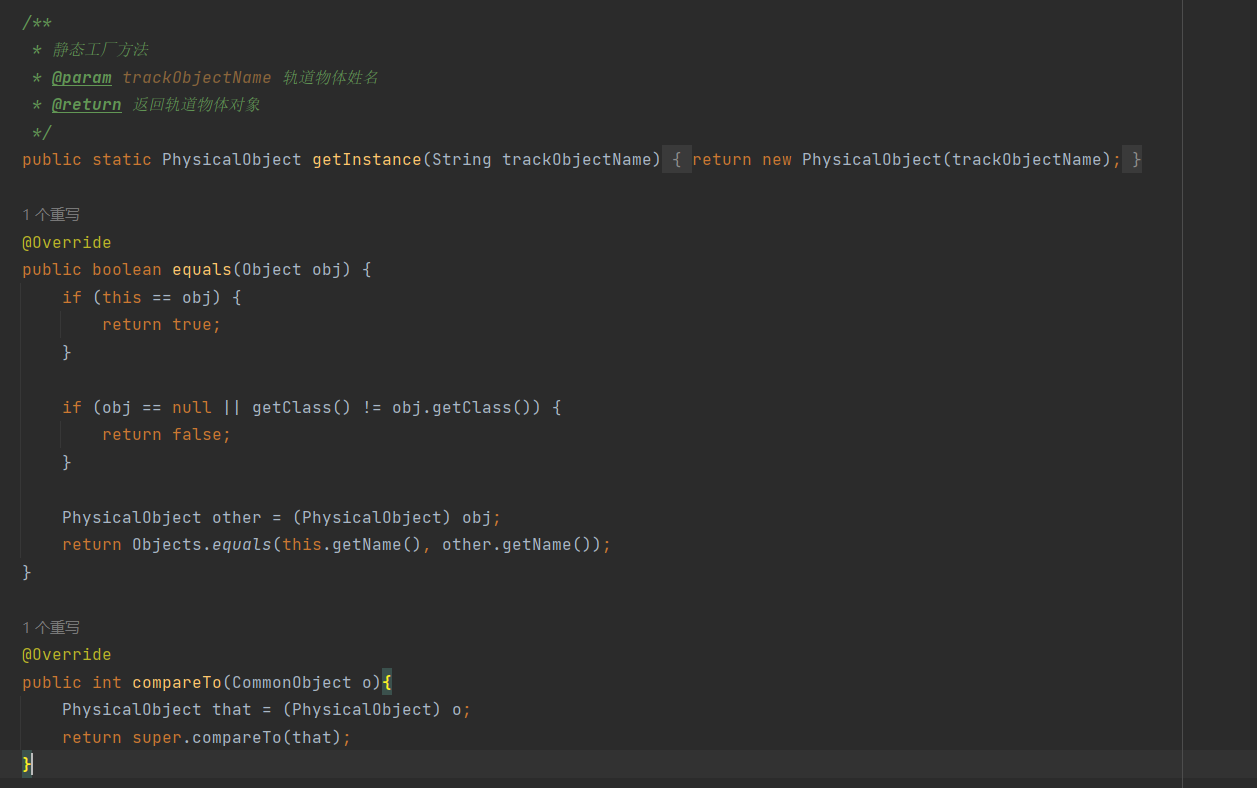
自定义了一个静态工厂方法：



## 面向复用的设计：PhysicalObject



定义了静态工厂方法，equals方法和compareto方法。



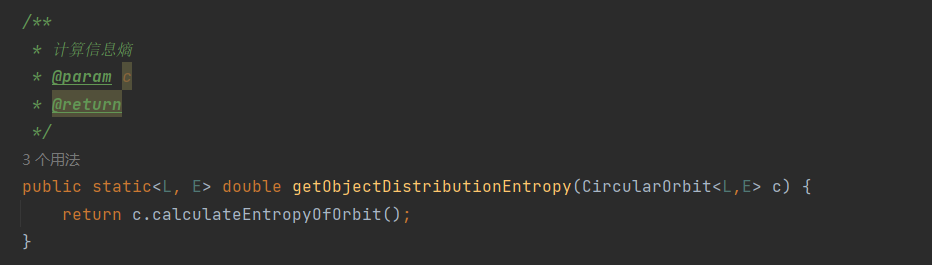
## 可复用API设计

直接调用ConcreteCircularOrbit中的对应方法。

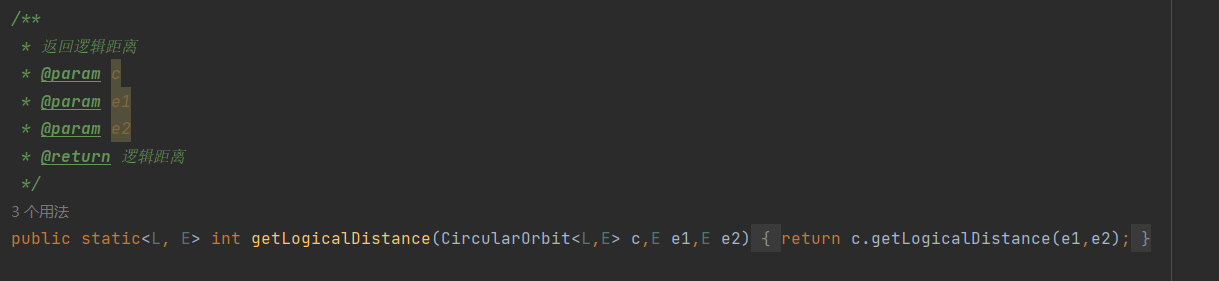
有一个getPhysicalDistance方法没有实现，因为选择的是应用135对物体的物理位置都没有要求。

全部设置成static方法可以不用实例化CircularOrbitAPIs就使用这些方法。

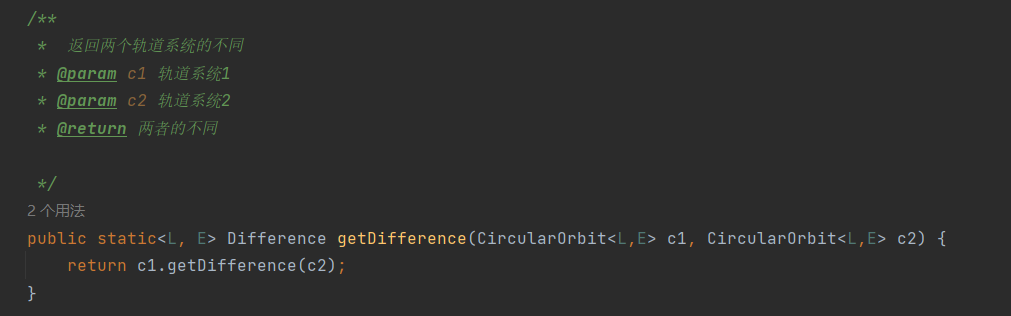
计算信息熵：



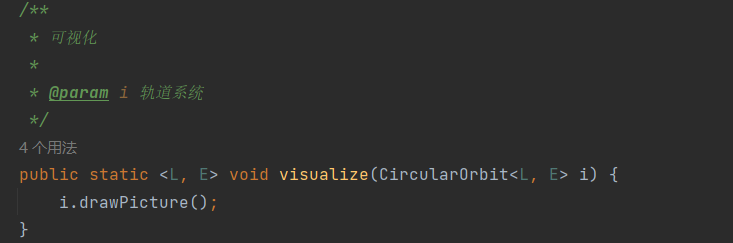
返回逻辑距离：



返回两个轨道的不同：



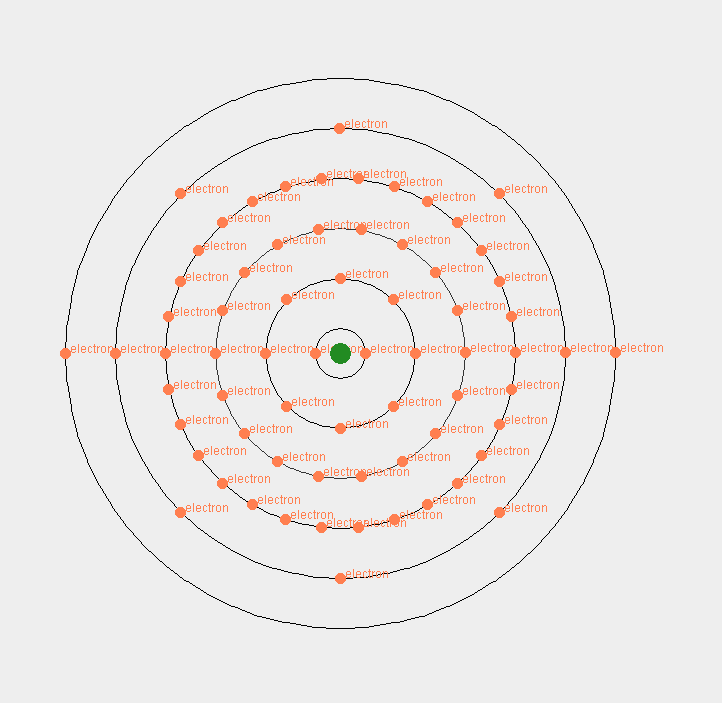
## 图的可视化：第三方API的复用



具体实现位于ConcreteCircularOrbit：

实现思路：具体实现思路是通过创建一个JFrame窗口，设置其大小和退出方式，然后在窗口中添加一个JPanel，重写其paintComponent方法，在其中绘制轨道和物体。通过循环遍历轨道系统中的每个轨道，获取该轨道上物体的数量和角度，并通过Math库中的函数计算出每个物体在界面上的位置，最后使用Graphics类中的方法绘制出来。

测试效果如图：

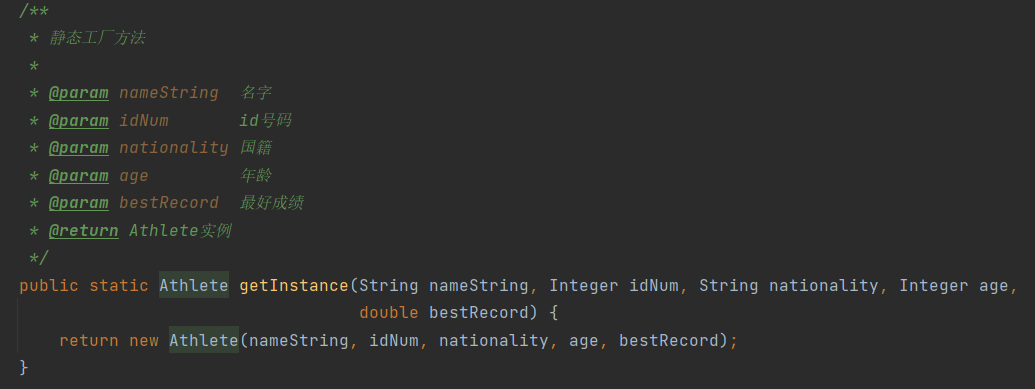


## 设计模式应用

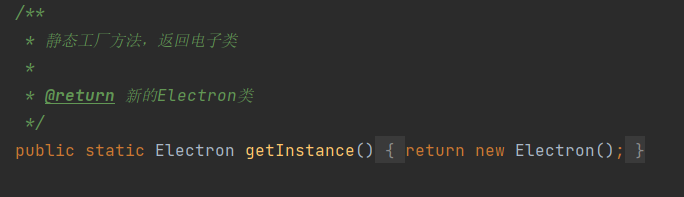
### 工厂设计模式：

在每个具体Object类中实现一个静态的getInstance方法，不需要实例化即可调用。

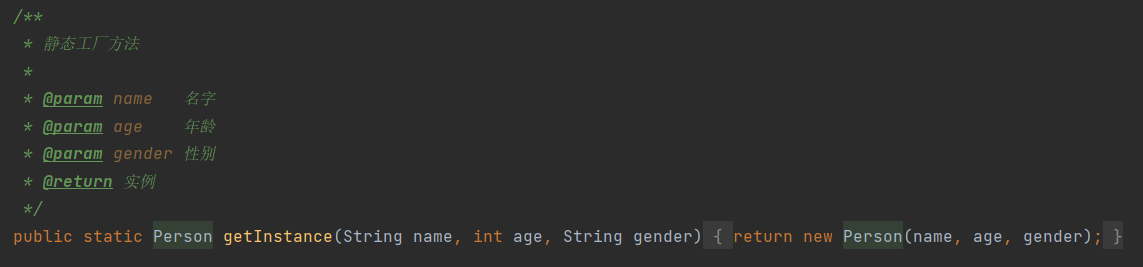
运动员：



电子：



人：



### Builder设计模式：

设计抽象CircularOrbitBuilder类作为具体builder类的父类。



之后再根据具体应用实现子builder类：







### Iterator设计模式

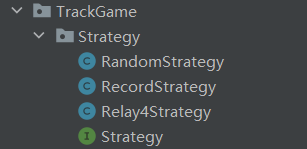
在ConcreteCircularOrbit中实现迭代器。

1. */\*\**
2. \* 迭代器实现
3. \*/
4. @Override
5. public Iterator<E> iterator() {
6. return new MyIterator(OrbitMap);
7. }
8. */\*\**
9. \* 内部类，生成迭代器，遍历轨道系统内所有的轨道物体，顺序为顺序为从内轨道逐步向外，
10. \* 同一轨道物体按名称排序
11. \*/
12. private static class MyIterator<E extends Comparable<E>> implements Iterator<E> {
13. private final List<E> ObjectList;
14. private int index;
15. private int size;
16. */\*\**
17. \* 构造方法，初始化 ObjectList 作为 OrbitIterator 的迭代输出
18. \*
19. \* @param orbitMap 迭代对象
20. \*/
21. public MyIterator(Map<Track, List<E>> orbitMap) {
22. index = 0;
23. size = orbitMap.values().stream().mapToInt(List::size).sum();
24. ObjectList = orbitMap.entrySet()
25. .stream()
26. .sorted(Map.Entry.comparingByKey())
27. .flatMap(entry -> entry.getValue().stream().sorted())
28. .collect(Collectors.toList());
29. }
30. */\*\**
31. \* 判断是否有下一个
32. \*
33. \* @return 是返回 true
34. \*/
35. @Override
36. public boolean hasNext() {
37. return index < size;
38. }
39. */\*\**
40. \* 获取下一个迭代对象的方法
41. \*
42. \* @return 下一个 E 对象
43. \*/
44. @Override
45. public E next() {
46. return ObjectList.get(index++);
47. }
48. }

实现思路：

在该迭代器中，使用了内部类MyIterator，它实现了Iterator接口，其中包含了ObjectList、index、size三个属性。ObjectList是一个List类型，存储了轨道系统中所有的物体，并按照轨道顺序和名称排序。index是一个计数器，表示当前遍历到的物体在ObjectList中的下标，size表示ObjectList的大小。MyIterator类中实现了hasNext()和next()两个方法，用于判断是否还有下一个物体并获取下一个物体。在构造方法中，首先将轨道系统中的物体按照轨道顺序和名称排序，然后将排序后的物体存入ObjectList中。最后，通过stream()和lambda表达式实现了对轨道系统进行遍历和排序的功能。

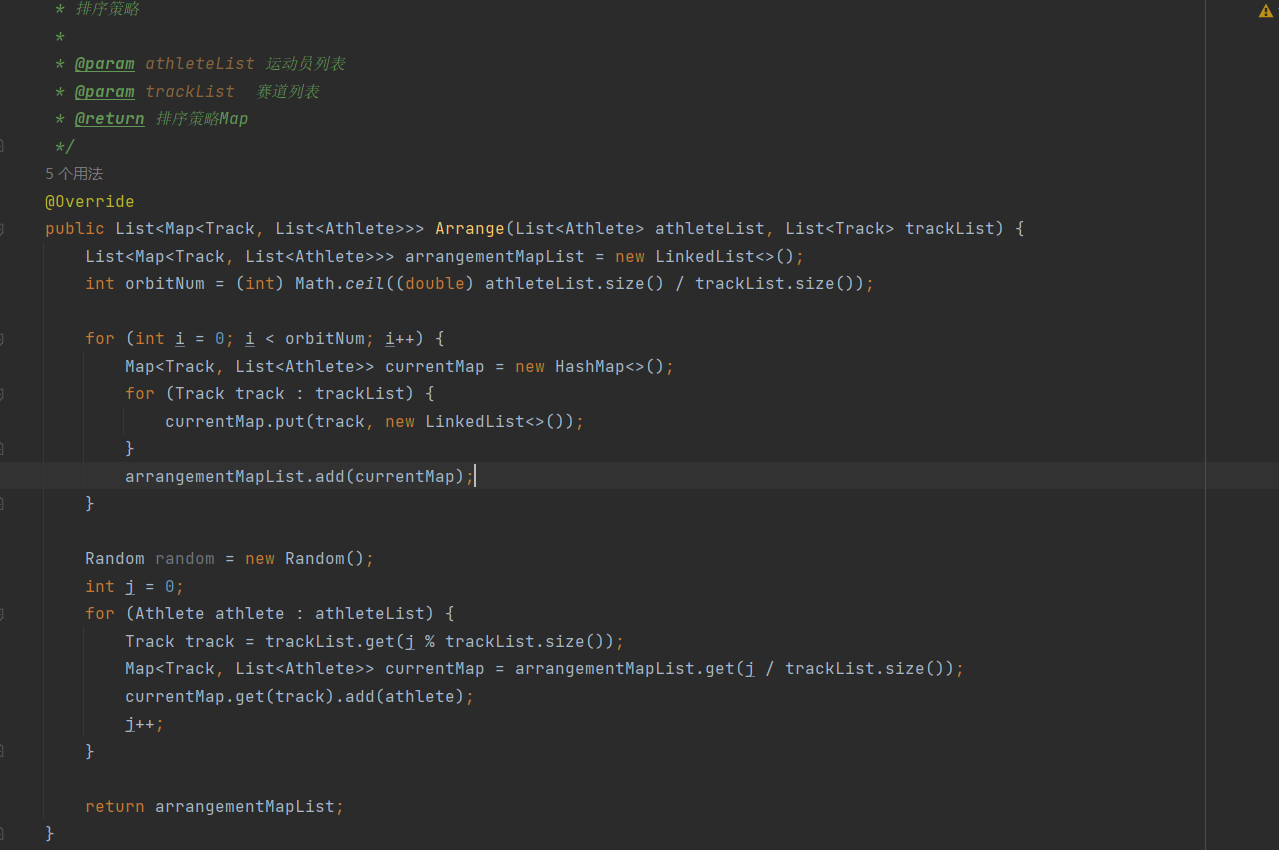
### strategy设计模式



在TrackGame中设计一个strategy接口和三个具体实现的strategy类，分别能执行随机排序，按成绩排序和四人接力。

接口接收传入的运动员列表和赛道列表，返回构造好的关系Map。用于生成TrackCircularOrbit。

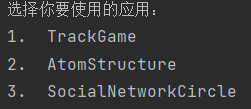
调用举例：



之后根据返回的安排好的关系Map。生成TrackCircularOrbit。

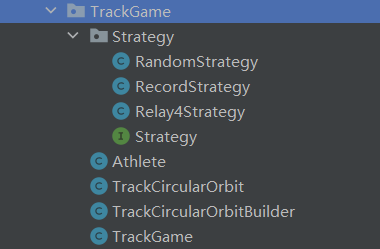
## 应用设计与开发

在main类里调用三个应用。

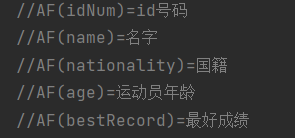


### TrackGame

目录如下：



Athlete类继承自PhysicalObject作为轨道物体，具有以下新增属性



TrackCircularOrbit类继承自ConcreteCircularOrbit作为具体轨道结构

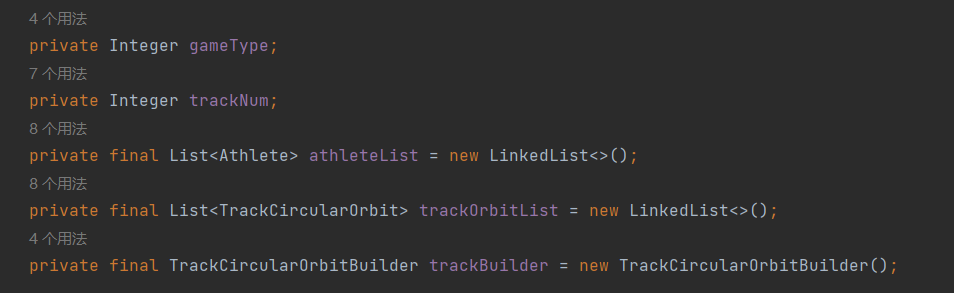
重写了toString方法输出多轨道结构

重写了checkRep()方法按要求检查轨道特性

TrackCircularOrbitBuilder类继承自CircularOrbitBuilder

主要是重写了createCircularOrbit方法，生成TrackCircularOrbit。

TrackGame类：具有下列域：



GameType轨道类型，trackNum轨道数量，athleteList运动员列表

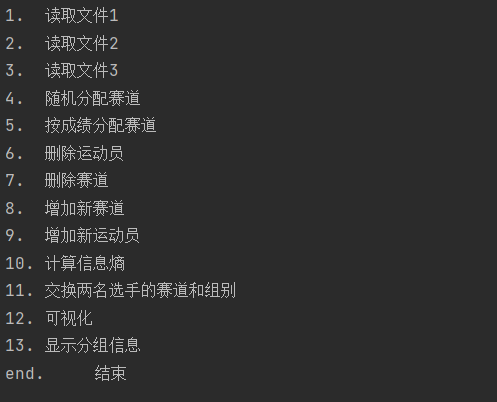
trackOrbitList是一个TrackCircularOrbit的list结构，因为每次安排比赛生成多个轨道结构，用这个list存放生成的所有TrackCircularOrbit。

trackBuilder是TrackCircularOrbitBuilder的一个实例。用来build每一个TrackCircularOrbit对象。

方法:

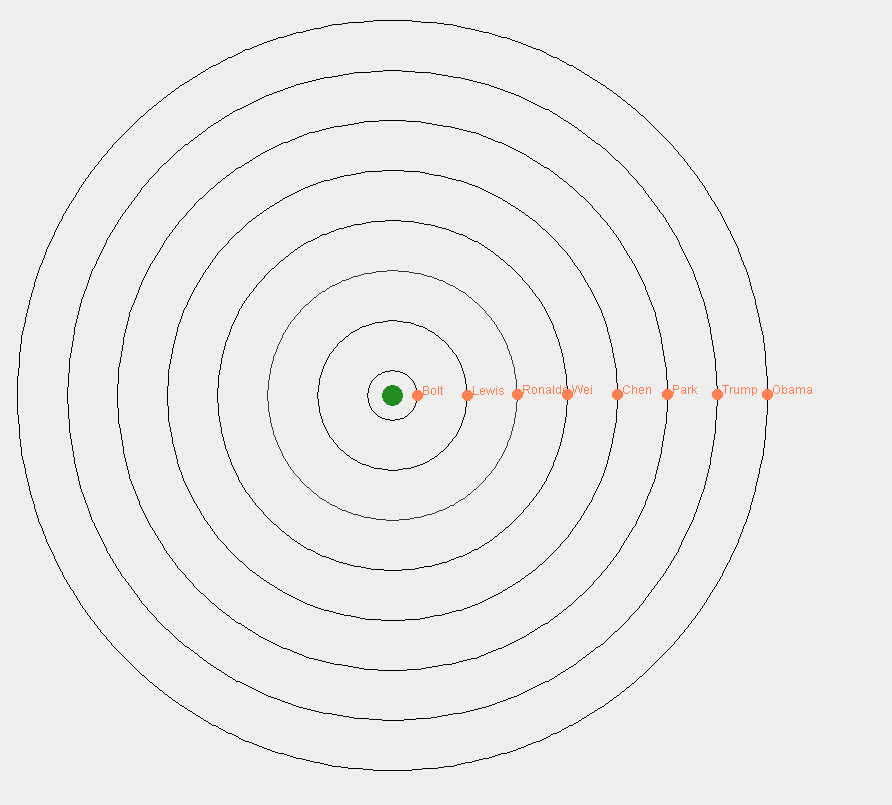
|  |  |
| --- | --- |
| 方法 | 功能 |
| GameMenu() { | 菜单 |
| gameMain | 功能主体，读取文件，构造每个TrackCircularOrbit，实现菜单中的每个功能。 |
| arrangeOrbit(Strategy strategy) | 策略安排方法，根据athleteList和trackNum和构造策略，生成构造方案 |

运行应用会出现以下菜单：

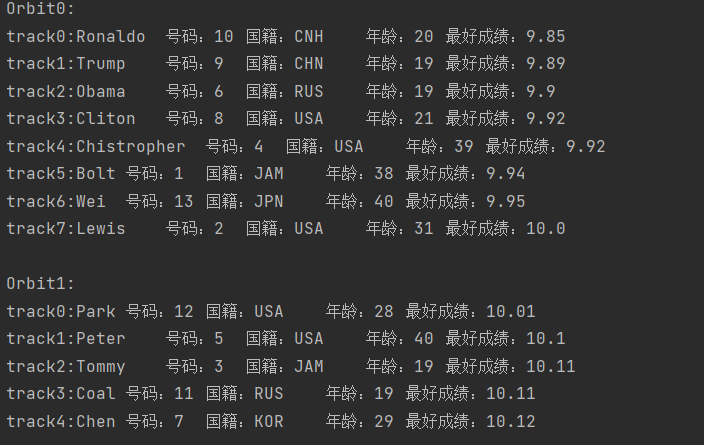


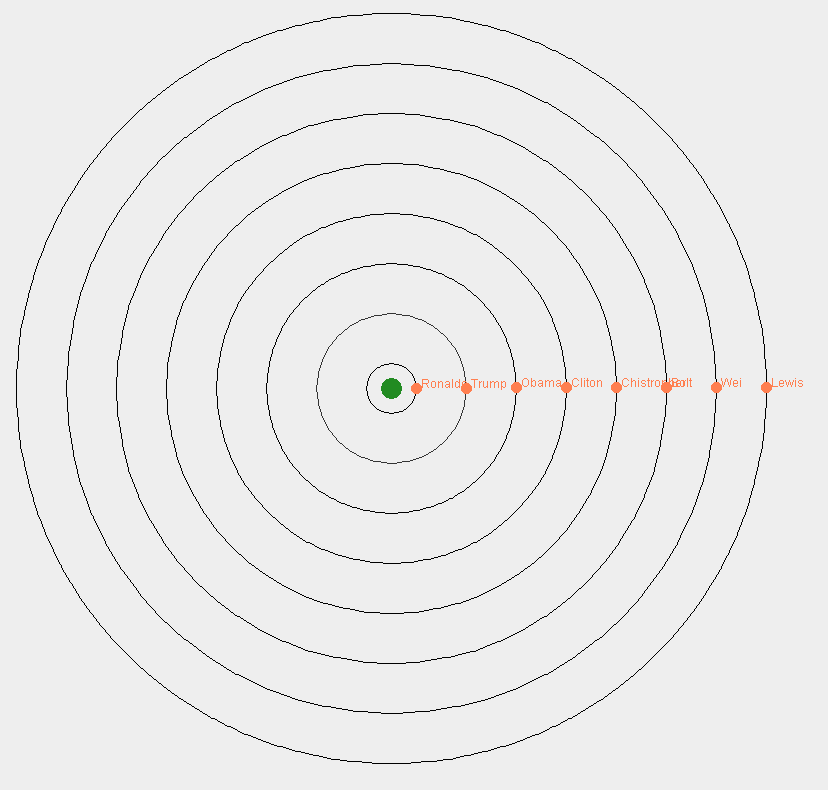
1 2 3.都是读取文件

4为随机分配赛道：



5为按成绩分配赛道：

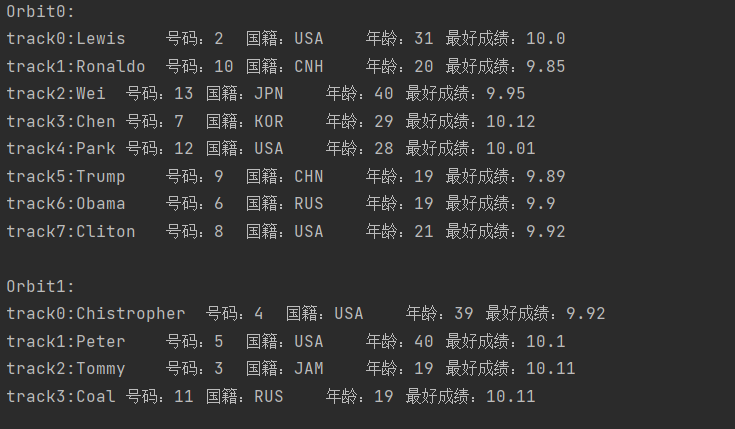




删除运动员：

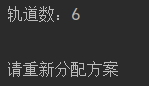


重新分配后可见1号选手已经删除：

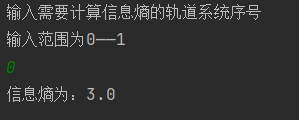


增加和删除轨道;

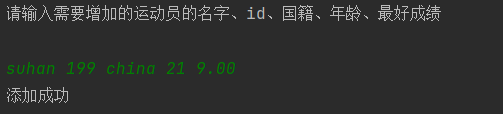




计算信息熵：



增加运动员：

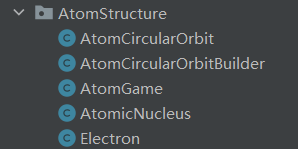


重新分组后可看到，suhan排名第一：



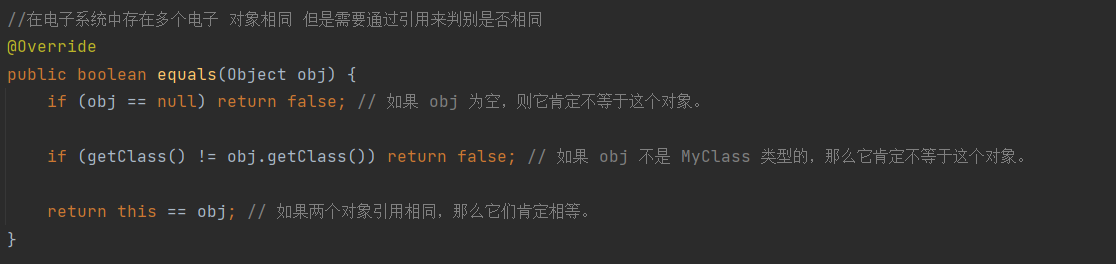
### AtomStructure

目录结构如下



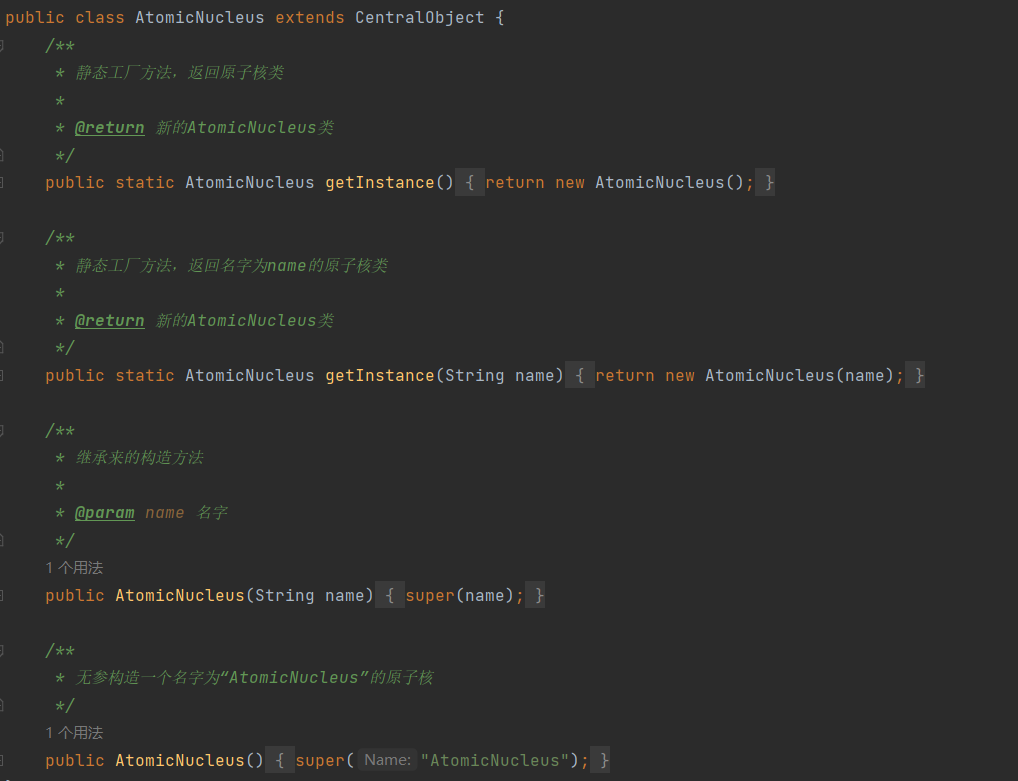
其中：

Electron类继承自PhysicalObject作为轨道物体，具有以下新方法：



具有以下特性：原子系统的电子，电子视为同一物体，不考虑不同电子的不同。

AtomicNucleus类继承自CentralObject作为中心物体，具有以下新方法：



AtomCircularOrbit类继承自ConcreteCircularOrbit作为具体轨道结构

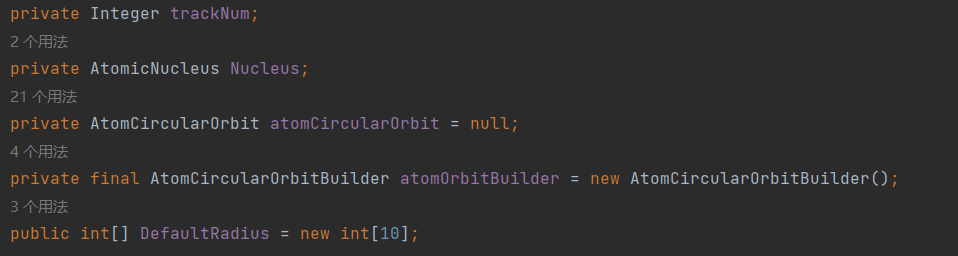
主要是重写了toString方法。新增方法如下：

|  |  |
| --- | --- |
| 方法 | 功能 |
| transit(Track t1, Track t2) { | 电子跃迁方法 |
| removeElectron(Track t) | 从某条轨道删去一个电子，因为电子互相之间没有区别，所以只需一个轨道参数。 |
| 重写toString | 输出轨道结构 |

AtomCircularOrbitBuilder类继承自CircularOrbitBuilder

主要是重写了createCircularOrbit方法，生成AtomCircularOrbit。

AtomGame类：具有下列域：



trackNum轨道数量，Nucleus原子核

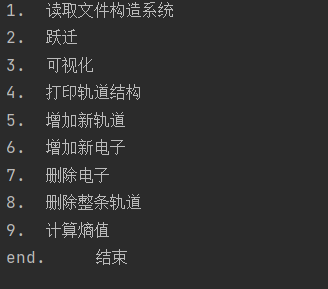
transitCareTaker是TransitCareTaker的一个实例。用来实现备忘录模式，具体是把每次跃迁记录成Memento保存下来，恢复时弹出上一个Memento。

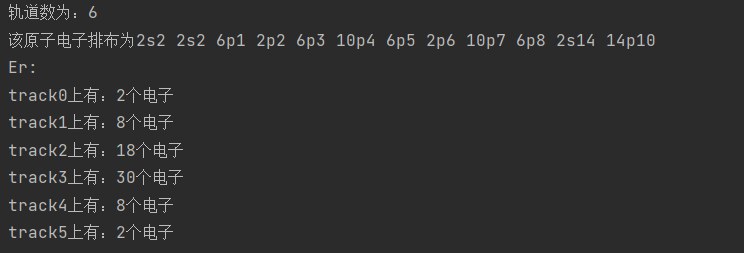
atomCircularOrbit就是当前使用的轨道系统实例。

atomOrbitBuilder是AtomCircularOrbitBuilder的一个实例。用来build每一个AtomCircularOrbitBuilder对象。

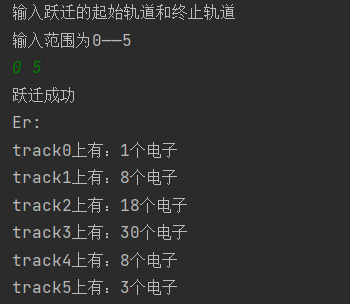
功能演示：

开始：读取文件并显示系统结构：

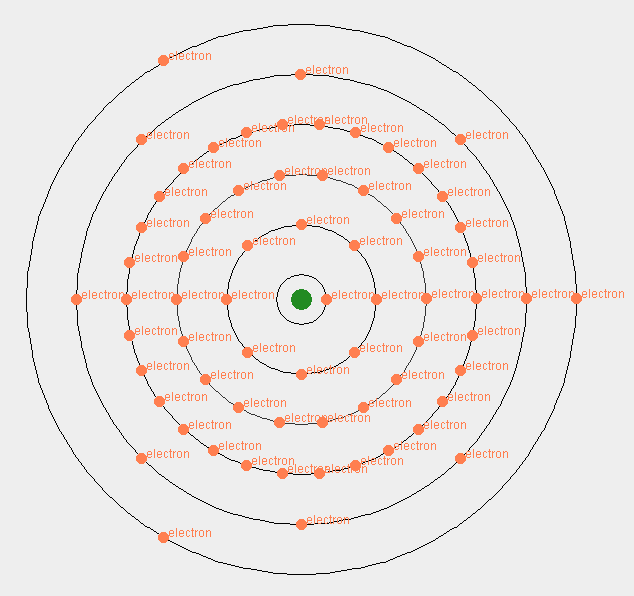




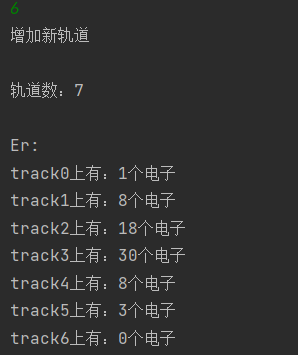
跃迁功能：



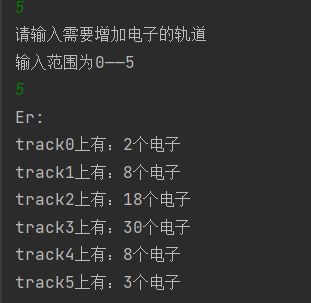
可视化功能：



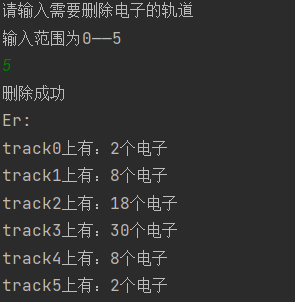
增加新轨道：



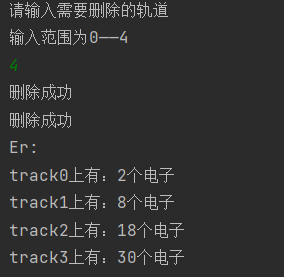
增加新电子：



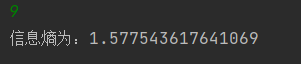
删除电子：



删除轨道：

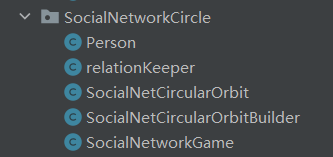


计算熵值：



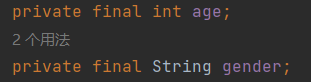
### SocialNetworkCircle

目录结构：



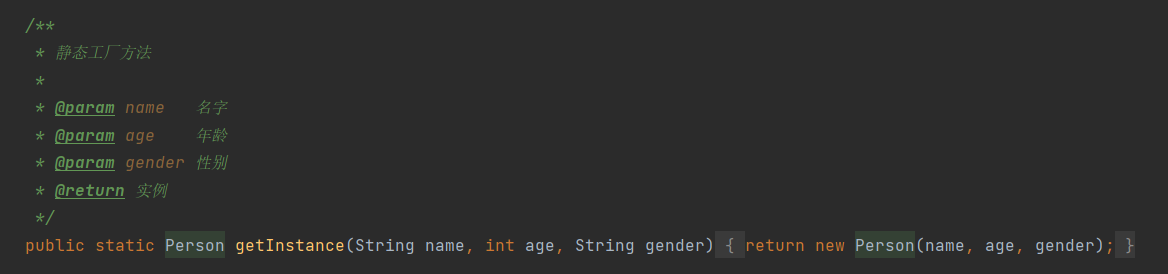
其中：

Person类继承自PhysicalObject作为轨道物体，具有以下新域：



Age保存年龄，gender是性别

提供一个静态工厂方法



relationKeeper类和是一个用来保存文件读入的类，因为读取文件读入的过程没法将读取到的人名 马上与实例对应起来，所以构造relationKeeper保存每个关系的人名string。

SocialNetCircularOrbit类继承自ConcreteCircularOrbit作为具体轨道结构

新增方法如下：

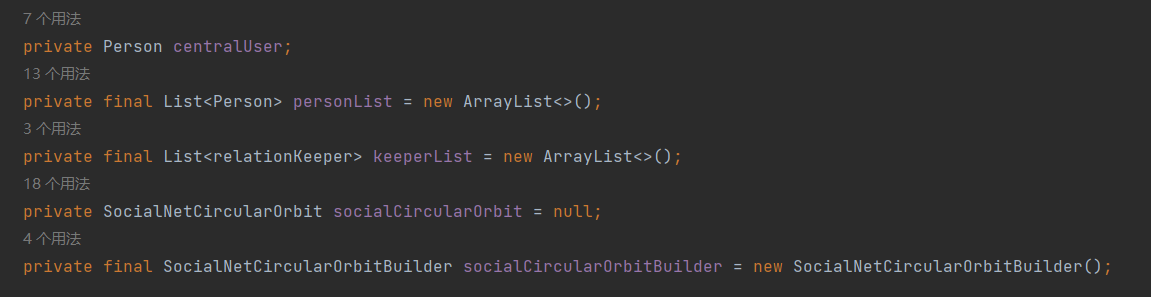
|  |  |
| --- | --- |
| 方法 | 功能 |
| reArrange | 重整关系网络 |
| 重写drawpicture方法 | 实现可视化，这个轨道结构要求可视化关系，所以加上画边的功能。 |
| 重写toString | 输出轨道结构 |

SocialNetCircularOrbitBuilder类继承自CircularOrbitBuilder

主要是重写了createCircularOrbit方法，生成SocialNetCircularOrbit。

新增bulidRelations，从读取的文件输入构造人际关系图，随后通过reArrange构造轨道关系图。

SocialNetworkGame类：具有下列域：



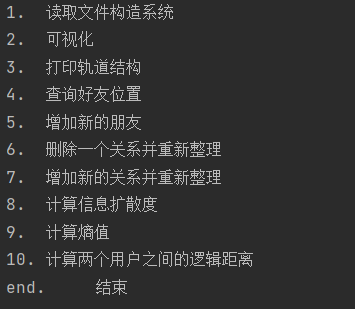
personList人列表，centralUser中心物体 ，keeperList是一个关系读取寄存器

socialCircularOrbit是当前操作的SocialNetCircularOrbit对象。

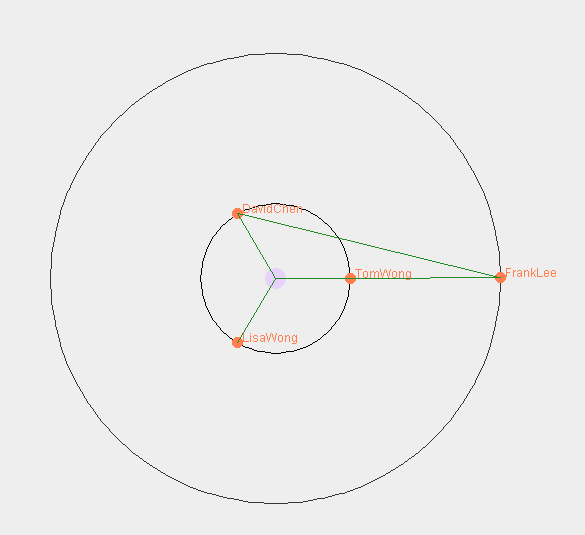
socialCircularOrbitBuilder是SocialNetCircularOrbitBuilder的一个实例。用来build每一个SocialNetCircularOrbit对象。

操作演示：

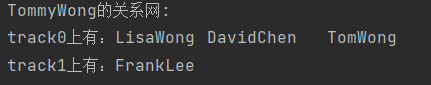
菜单



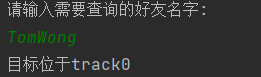
可视化：



文字输出轨道结构：



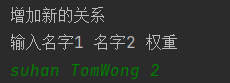
查询好友位置：



增加新的朋友：

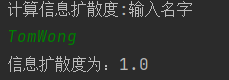


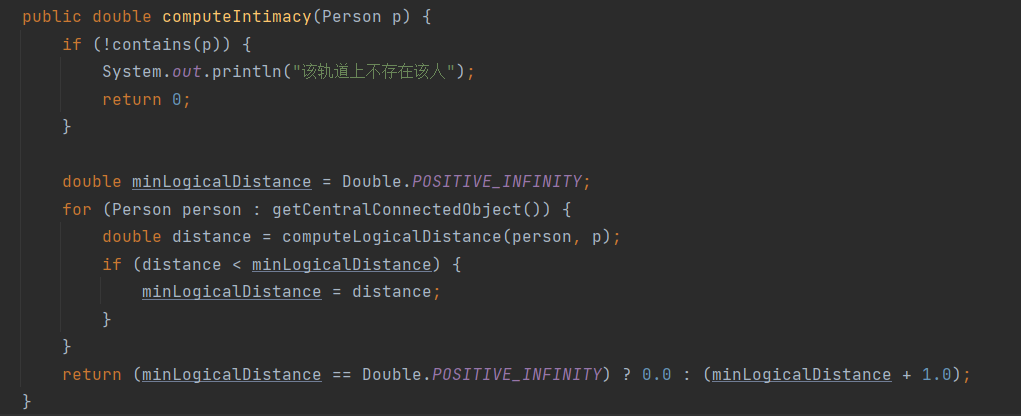
增加新的关系：



计算熵值:  


计算信息扩散度：





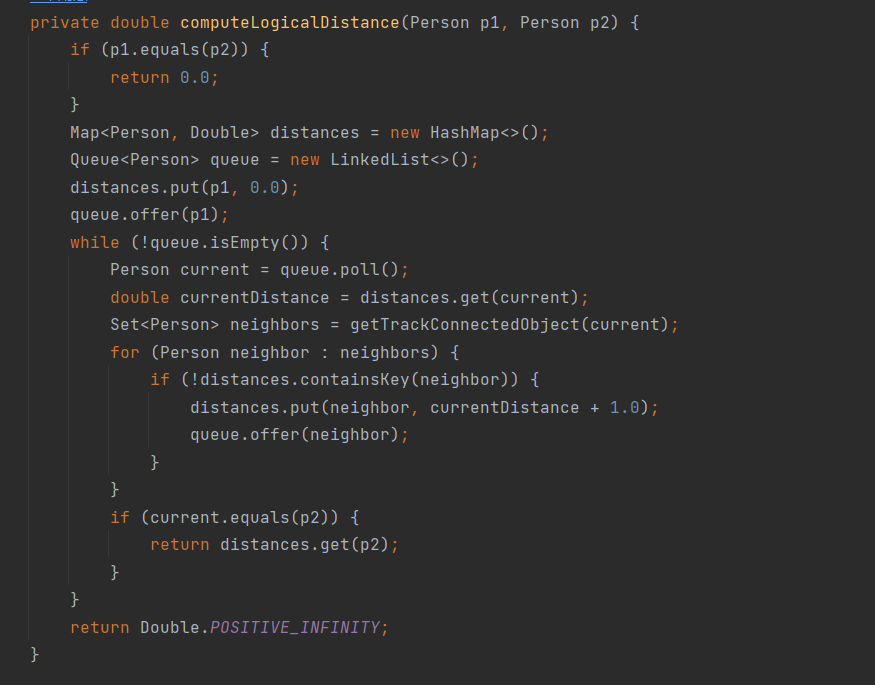
思路如下：

首先，代码通过contains(p)方法判断该人是否在当前轨道上，如果不在则输出提示信息并返回0。接下来，代码通过getCentralConnectedObject()方法获取当前轨道上与轨道中心相连的物体，这里假设这些物体都是人。然后，代码使用for循环遍历这些人，计算每个人与目标人之间的逻辑距离，并记录其中最小值。最后，代码返回逻辑距离与1之和，如果最小逻辑距离为无穷大，则返回0。其中，+1的操作是为了避免出现0的情况，保证返回值始终大于0。

计算逻辑距离：



代码：



思路：

代码中使用了广度优先搜索（BFS）算法，首先判断两个人是否相同，如果是，则返回0。接下来，代码使用distances存储每个人到起点人p1的距离，初始化时将p1的距离设为0，并将p1加入队列中。然后，代码进入while循环，不断从队列中取出当前人current，并获取current到p1的距离currentDistance。接着，代码使用getTrackConnectedObject(current)方法获取当前人current在轨道系统中邻居（即与其相连的其他人），并遍历这些邻居。对于每个邻居，如果其距离p1未被计算过，则将其距离设为currentDistance+1.0，并将其加入队列中。如果当前人current是目标人p2，则返回distances中p2的距离。如果遍历完所有与p1直接或间接相连的人后仍未找到p2，则返回无穷大。

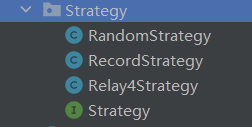
## 应对应用面临的新变化

以下各小节，只需保留和完成你所选定的应用即可。

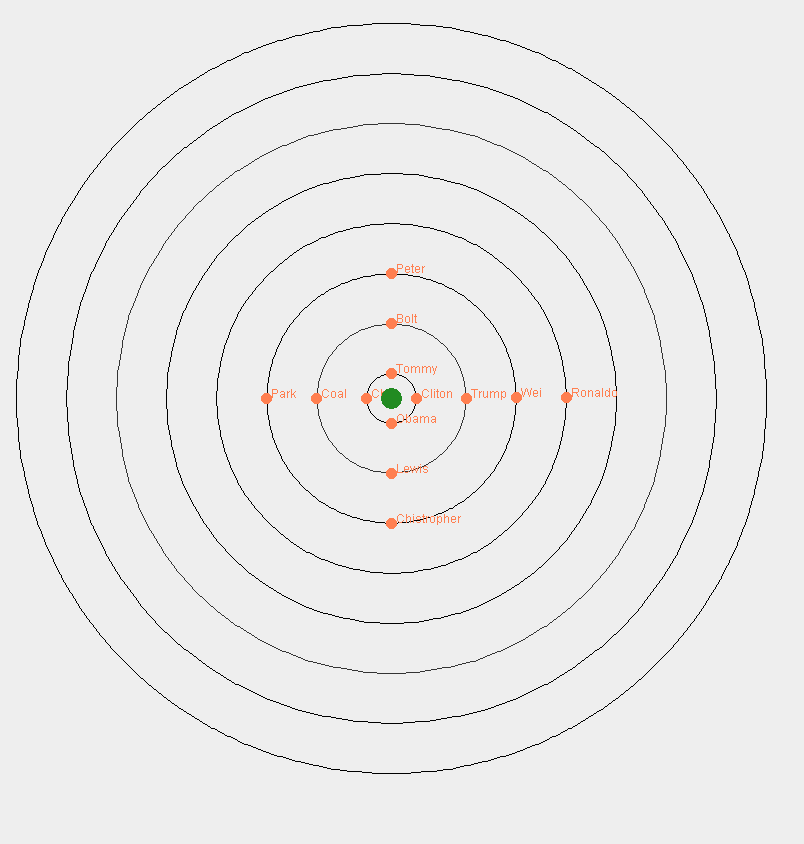
### TrackGame

由于使用了strategy模式，

只需要新建一个strategy：Relay4Strategy继承自Strategy接口实现一次分配四个人分配就行了



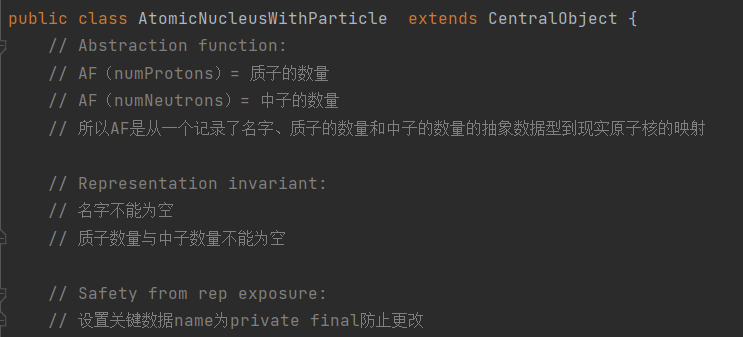
运行展示：



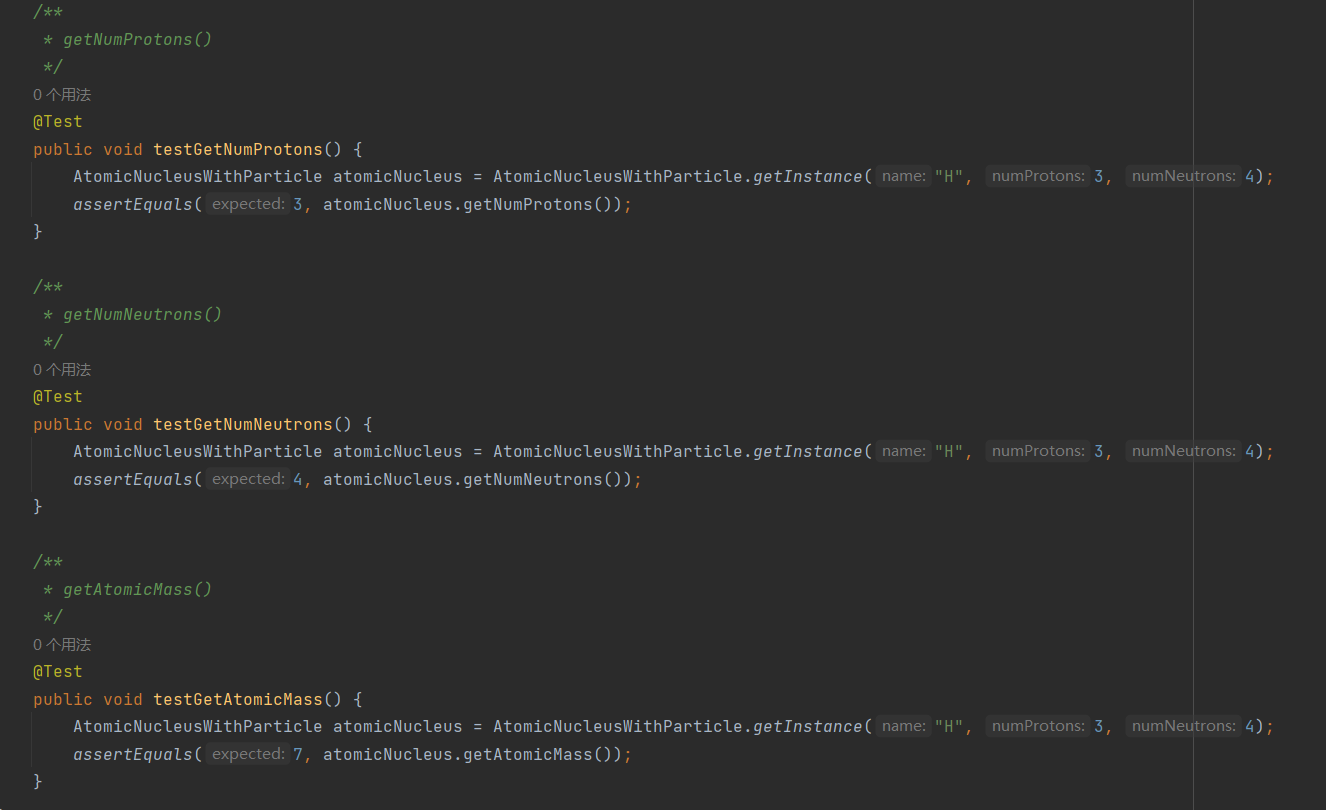
### AtomStructure

原子核表达为多个质子和多个中子的组合：设计中子质子类，添加

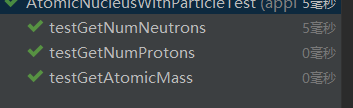
AtomicNucleusWithParticle，添加中子质子数量的属性。



同时进行测试：



测试均通过：



### SocialNetworkCircle

为关系增加方向性：在原来的实现中，无向边使用两条有向边表示，这里只

需要改一下，添加/删除关系的时候只需要操作一条边即可。

在读入边的时候，如果存在轨道物体向中心点的边则不操作，然后将其他所

有的边加入，这里也包括了外层轨道到内层轨道的边，但是这里我们保留它，因

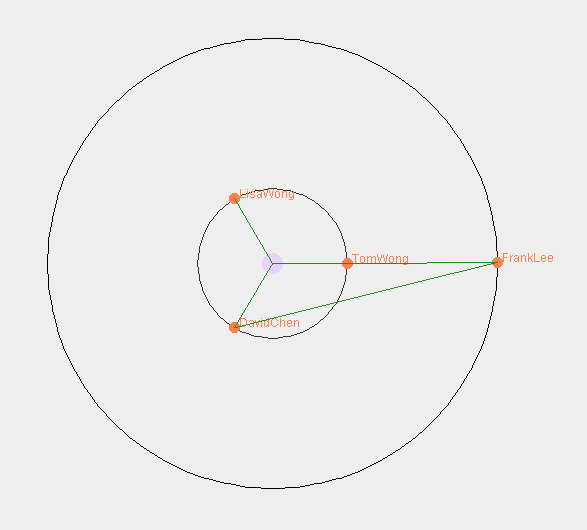
为从实际应用角度出发，虽然一条边是从外层轨道到内层轨道的，但是在添加/

删除关系之后这条边也可能变成内到外的边。

这里我们求出所有点所在的轨道，根据所在轨道的半径即可判断内外（也可

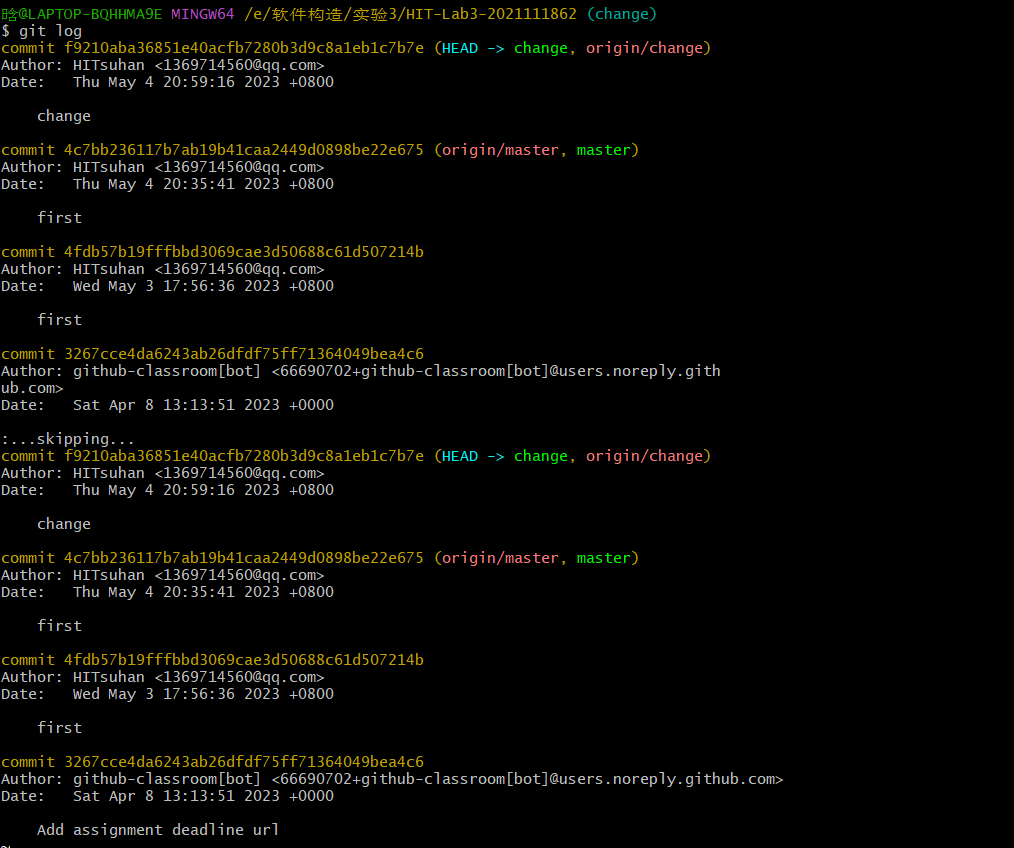
以求出物体距离中心点的亲密度），每次更改关系结构之后需要重新求一遍距离，

所以将求距离操作设计在 reArrange 中。



## Git仓库结构

请在完成全部实验要求之后，利用Git log指令或Git图形化客户端或GitHub上项目仓库的Insight页面，给出你的仓库到目前为止的Object Graph，尤其是区分清楚312change分支和master分支所指向的位置。



# 实验进度记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日期 | 时间段 | 计划任务 | 实际完成情况 |
| 4.15 | 14-22 | 翻看实验手册，补充知识 | 完成 |
| 4.16 | 14-22 | 建立项目，设计文件结构 | 完成 |
| 4.17 | 18-23 | 完成3.4 | 完成 |
| 4.19 | 15-23 | 完成3.5 3.6 3.7 | 完成 |
| 4.21 | 15-23 | 完成3.8 | 未完成 |
| 4.22 | 19-22 | 3.8 | 完成 |
| 4.25 | 18-22 | 3.9 | 完成 |
| 4.28 | 18-22 | 3.10 | 未完成 |
| 4.29 | 18-22 | 3.10 | 完成 |
| 5.1 | 8-22 | App1 | 未完成 |
| 5.2 | 8-22 | App1，App2 | 完成app1 |
| 5.3 | 8-22 | App2，app3 | 完成app2 |
| 5.4 | 8-22 | App3，补充test | 完成app3 |
| 5.5 | 8-22 | 撰写实验报告 | 完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 正则表达式不会用 | 上网学习咨询同学 |
| 任务量太大了 | 通宵写代码 |
| 不会可视化 | 咨询同学，上网学习 |

# 实验过程中收获的经验、教训、感想

## 实验过程中收获的经验和教训

在修改change312时，我没有考虑到代码的可复用性，导致修改的次数比较多。

## 针对以下方面的感受

1. 重新思考Lab2中的问题：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？本实验设计的ADT在五个不同的应用场景下使用，你是否体会到复用的好处？

面向ADT的编程和面向应用场景的编程是两种不同的编程思路。面向ADT的编程主要关注数据结构和抽象数据类型的设计和实现，强调代码的可重用性和可扩展性。而面向应用场景的编程则更加注重实际应用场景的需求和特征，强调代码的实用性和可读性。复用的好处：提高代码的可维护性，提高代码的可扩展性。

1. 重新思考Lab2中的问题：为ADT撰写复杂的specification, invariants, RI, AF，时刻注意ADT是否有rep exposure，这些工作的意义是什么？你是否愿意在以后的编程中坚持这么做？

确保ADT行为符合预期：通过撰写specification、invariants、RI和AF，可以清楚地描述ADT的行为和功能，以及其与外部环境的交互方式，从而确保ADT行为符合预期。

预防错误和漏洞：通过定义invariants和RI，可以确保ADT的状态和数据结构的完整性和一致性，从而预防错误和漏洞的出现。

提高代码的可维护性：通过撰写specification、invariants、RI和AF，可以提高代码的可读性和可维护性，使代码更易于理解和修改。

支持代码的复用：通过撰写AF，可以清晰地描述ADT的抽象数据结构和操作，从而支持代码的复用，提高代码的可重用性。

愿意这么做。

1. 之前你将别人提供的API用于自己的程序开发中，本次实验你尝试着开发给别人使用的API，是否能够体会到其中的难处和乐趣？

使用别人的API很方便，前提是对别人的API要有一定的理解。

1. 在编程中使用设计模式，增加了很多类，但在复用和可维护性方面带来了收益。你如何看待设计模式？

设计模式在编程中是非常有价值的，可以带来多方面的收益，尤其是在复用和可维护性方面。

1. 你之前在使用其他软件时，应该体会过输入各种命令向系统发出指令。本次实验你开发了一个解析器，使用语法和正则表达式去解析输入文件并据此构造对象。你对语法驱动编程有何感受？

避免了自己输入的繁琐过程。可以很方便快捷得处理海量数据。

1. Lab1和Lab2的大部分工作都不是从0开始，而是基于他人给出的设计方案和初始代码。本次实验是你完全从0开始进行ADT的设计并用OOP实现，经过三周之后，你感觉“设计ADT”的难度主要体现在哪些地方？你是如何克服的？

设计ADT需要具备较强的抽象能力，能够将问题抽象成抽象数据类型，并设计相应的操作。在设计ADT时，我会尽可能地抽象问题，将问题转化为抽象数据类型，并设计相应的操作。

1. 关于本实验的工作量、难度、deadline。

工作量特别特别大，基本上用了两周时间，难度还可以，ddl真的很难受。

1. 到目前为止你对《软件构造》课程的评价。

挺好