

**2019年春季学期  
计算机学院《软件构造》课程**

**Lab 3实验报告**

|  |  |
| --- | --- |
| 姓名 | 林之浩 |
| 学号 | 1170300817 |
| 班号 | 1703008 |
| 电子邮件 | 630073498@qq.com |
| 手机号码 | 18065053516 |

**目录**

[1 实验目标概述 1](#_Toc7990549)

[2 实验环境配置 1](#_Toc7990550)

[3 实验过程 1](#_Toc7990551)

[3.1 待开发的三个应用场景 1](#_Toc7990552)

[3.2 基于语法的图数据输入 2](#_Toc7990553)

[3.3 面向复用的设计：CircularOrbit<L,E> 2](#_Toc7990554)

[3.4 面向复用的设计：Track 3](#_Toc7990555)

[3.5 面向复用的设计：L 4](#_Toc7990556)

[3.6 面向复用的设计：PhysicalObject 4](#_Toc7990557)

[3.7 可复用API设计 5](#_Toc7990558)

[3.8 图的可视化：第三方API的复用 5](#_Toc7990559)

[3.9 设计模式应用 7](#_Toc7990560)

[3.9.1 工厂设计模式： 7](#_Toc7990561)

[3.9.2 Builder设计模式： 7](#_Toc7990562)

[3.9.3 Iterator设计模式 8](#_Toc7990563)

[3.9.4 facade设计模式 8](#_Toc7990564)

[3.9.5 strategy设计模式 9](#_Toc7990565)

[3.9.6 备忘录设计模式 10](#_Toc7990566)

[3.10 应用设计与开发。 11](#_Toc7990567)

[3.10.1 TrackGame 11](#_Toc7990568)

[3.10.2 AtomStructure 17](#_Toc7990569)

[3.10.3 SocialNetworkCircle 22](#_Toc7990570)

[3.11 应对应用面临的新变化 26](#_Toc7990571)

[3.11.1 TrackGame 26](#_Toc7990572)

[3.11.2 AtomStructure 28](#_Toc7990573)

[3.11.3 SocialNetworkCircle 30](#_Toc7990574)

[3.12 Git仓库结构 31](#_Toc7990575)

[4 实验进度记录 32](#_Toc7990576)

[5 实验过程中遇到的困难与解决途径 32](#_Toc7990577)

[6 实验过程中收获的经验、教训、感想 33](#_Toc7990578)

[6.1 实验过程中收获的经验和教训 33](#_Toc7990579)

[6.2 针对以下方面的感受 33](#_Toc7990580)

# 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。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

<https://github.com/ComputerScienceHIT/Lab3-1170300817>

# 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

分析你所选定的多个应用场景的异同，理解需求：它们在哪些方面有共性、哪些方面有差异。

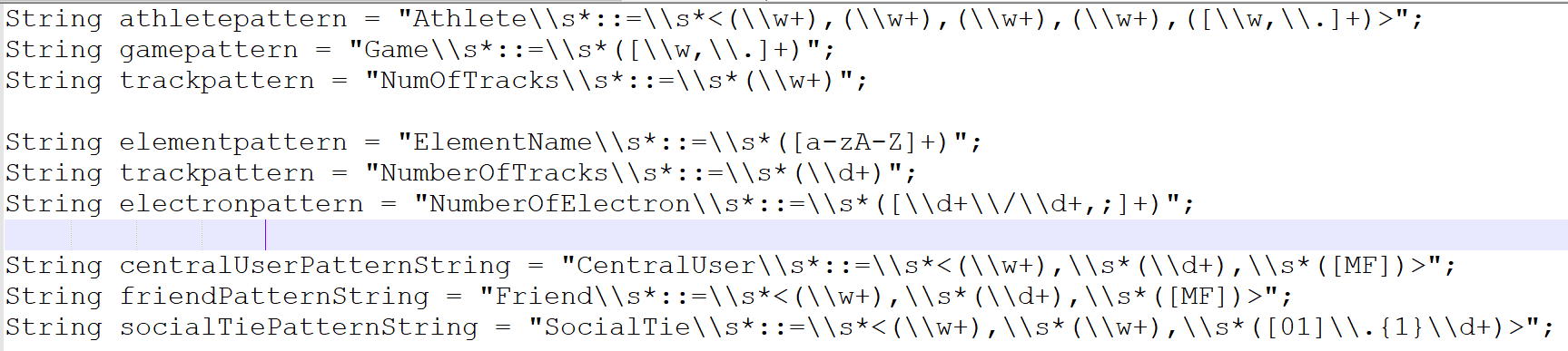
首先，他们都是多轨道结构，它们的轨道十分类似，都可以看成一个只有名字和半径属性的圆。它们的轨道物体也有一些相似之处：比如都要求有名字。他们都要求有对于轨道、轨道上的物体的一系列增删改查的操作的要求。

不同之处：

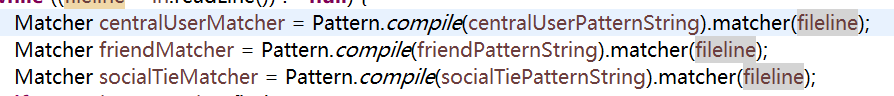
有的要求各个物体之间有有向或无向关系，有的各个物体之间没有区别比如电子，有的有区别比如运动员。

## 基于语法的图数据输入

使用的正则表达式如下：



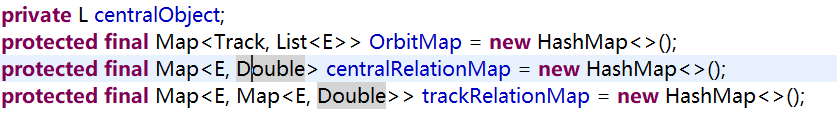
设计好正则表达式后，用它构造Pattern，Matcher实现捕获



## 面向复用的设计：CircularOrbit<L,E>



定义的数据结构如下



其中OrbitMap存放Track和E的对应关系，一个Track对应一个处于这个Track上的E链表。

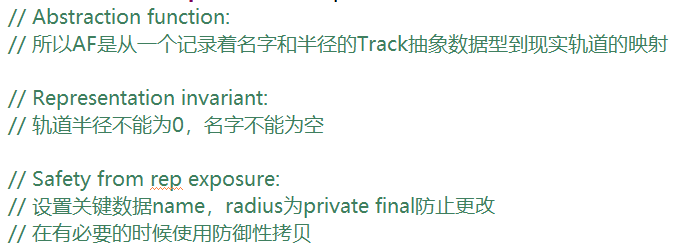
centralRelationMap存放中心物体的关系，map为<和中心物体有关系的物体，权重>

trackRelationMap存放物体和这个轨道有关系的物体的map，

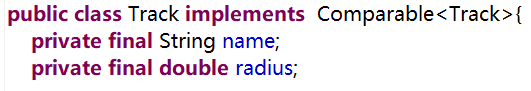
map为<轨道物体，<与这个轨道物体有关系的物体，权重>>

|  |  |
| --- | --- |
| 方法： | 功能： |
| setCentralObject(L centralObject); | 设置中心物体 |
| getCentralObject(); | 返回中心物体 |
| addTrack(Track t); | 增加一条轨道 |
| removeTrack(Track t); | 删除一条轨道 |
| getTrackNum(); | 返回轨道数目 |
| getObjectNumonTrack(Track t); | 统计一条轨道有多少物体 |
| addObjectToTrack(Track t, E object); | 向轨道上增加物体 |
| removeObjectOnTrack(Track t, E object); | 删去一个轨道上的某个物体 |
| addtrackRelation(E object1, E object2, double distance) | 在两个轨道物体之间新增关系 |
| addcentralRelation(E object, double distance); | 在轨道物体和中心物体之间新增关系 |
| getObjectDistributionEntropy(); | 计算轨道系统的信息熵 |
| getLogicalDistance(E e1, E e2); | 获得两物体间逻辑距离 |
| getDifference(CircularOrbit<L, E> c); | 比较当前orbit和目标orbit的不同  return 一个different对象，记录两个orbit的区别 |
| getSortedTracks(); | 获得当前orbit包含的所有轨道按半径排列成的链表 |
| drawpicture(); | 可视化方法 |
| contains(E e); | 判断当前Orbit是否包含某个元素e |
| getObjectTrack(E e); | 返回某个元素e所在的Object对象 |
| getCentralConnectedObject(); | 返回与中心连接的物体构成的集合 |
| getTrackConnectedObject(E object) | 返回与某个轨道物体连接的所有物体 |
| checkRep(){ | 检查合法性 |

## 面向复用的设计：Track



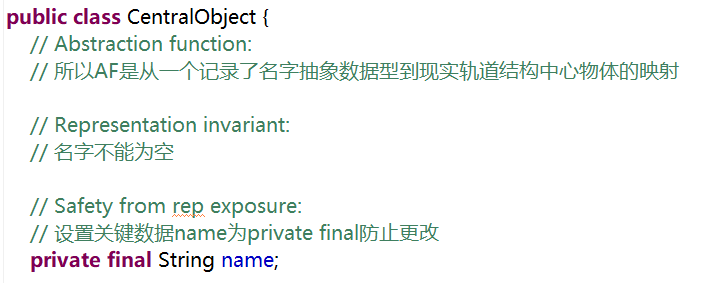
有两个域 name名字 和radius半径，实现Comparable依靠半径排序。



以下省略getter setter方法

|  |  |
| --- | --- |
| 方法 | 功能 |
| Track(String name, double radius) | 构造方法，传入名字，半径 |
| checkRep() { | 检查名字非空，半径大于0 |
| int compareTo(Track o) { | 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 |

## 面向复用的设计：L

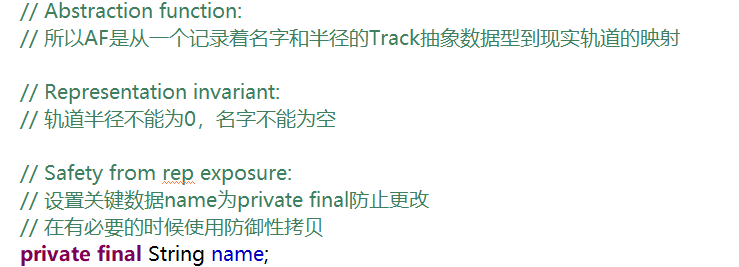


只有一个域名字，方法：

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

以及checkRep() 方法检查名字非空。

## 面向复用的设计：PhysicalObject



只有一个域名字，方法：

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

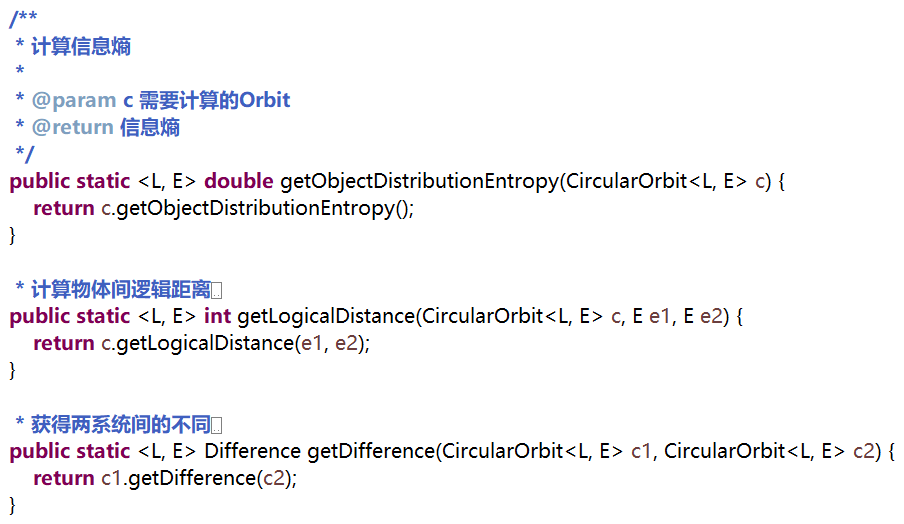
以及checkRep() 方法检查名字非空。

## 可复用API设计

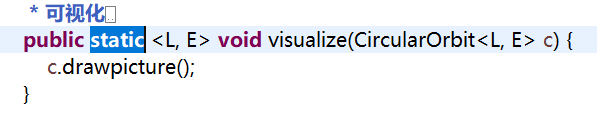
很简单，直接调用ConcreteCircularOrbit中的对应方法。

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。

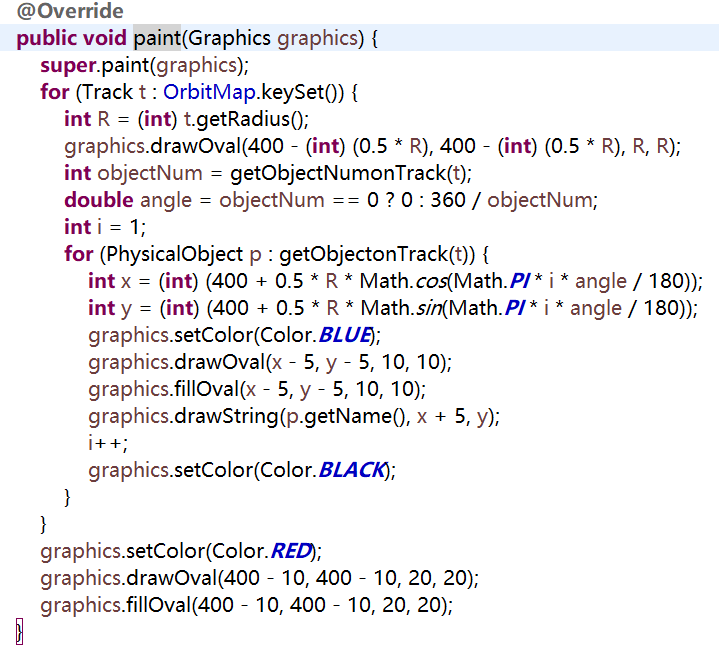


## 图的可视化：第三方API的复用



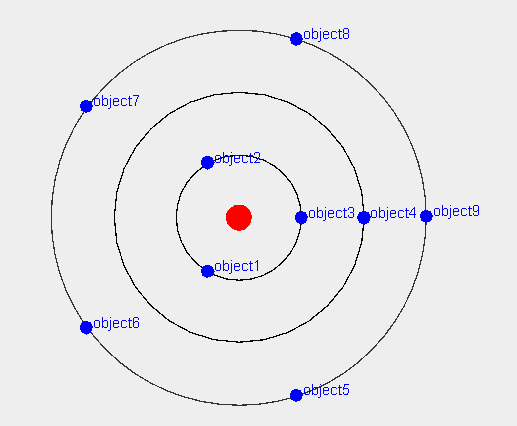
具体实现位于ConcreteCircularOrbit：

主要是构造一个JPanel，重写paint方法：



按轨道依次计算点的位置利用drawOval方法画圆，fillOval方法填色，drawString方法写名字。

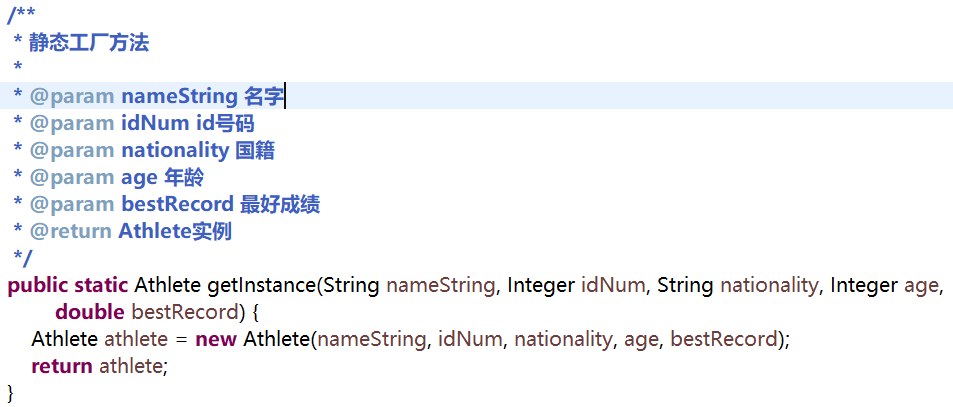
测试效果：



## 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Athlete即可调用



### Builder设计模式：

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



包含一个concreteCircularOrbit就是正在build过程中的Orbit

方法如下：

|  |  |
| --- | --- |
| 方法 | 功能 |
| getConcreteCircularOrbit() { | 返回构造完成的对象 |
| abstract void createCircularOrbit(); | 抽象方法，实现时创建具体类型的Orbit |
| bulidPhysicalObjects(L centralObj, Map<Track, List<E>> ObjectMap) | 根据传入的centralObj和ObjectMap初始化concreteCircularOrbit中的关系Map |
| bulidTracks(List<Track> trackList) { | 根据传入的trackList构造concreteCircularOrbit |

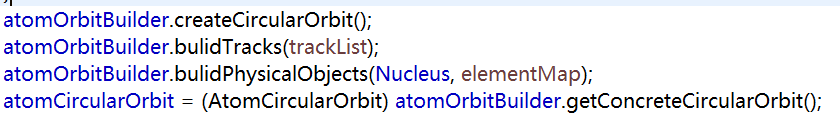
之后再根据具体应用实现子builder类：







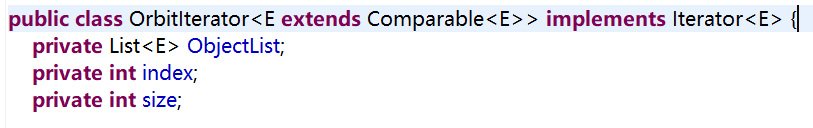
具体调用举例：



先初始化具体Orbit类，然后依次传入轨道列表，关系Map。最后完成构造输出。

### Iterator设计模式

设计OrbitIterator类实现Iterator接口。



ObjectList是保存所有物体的list，每次取next从ObjectList取值。

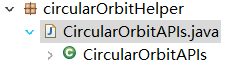
Index是迭代指针。

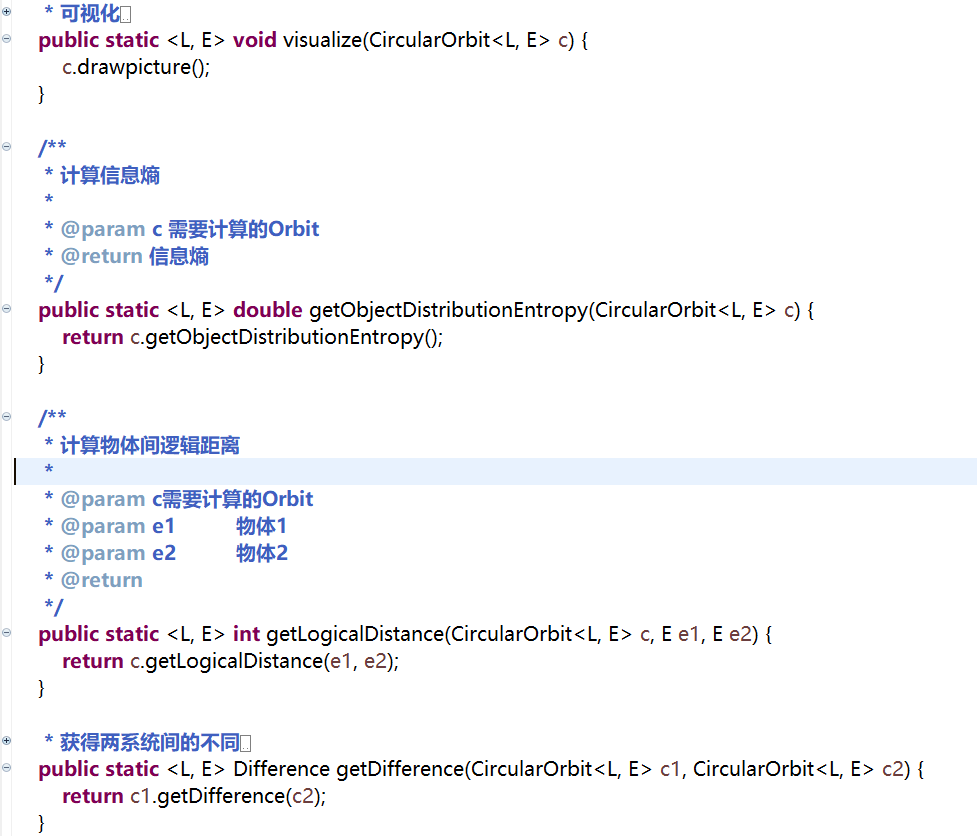
方法如下：

|  |  |
| --- | --- |
| 方法 | 功能 |
| OrbitIterator(Map<Track, List<E>> orbitMap) | 构造函数，根据关系Map orbitMap构造一个ObjectList作为迭代的输出 |
| hasNext() | 判断迭代指针是否等于size，如果是则没有下一个了 |
| next() | 取元素，Index++ |

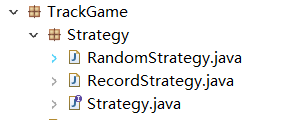
### facade设计模式

如图：





### strategy设计模式



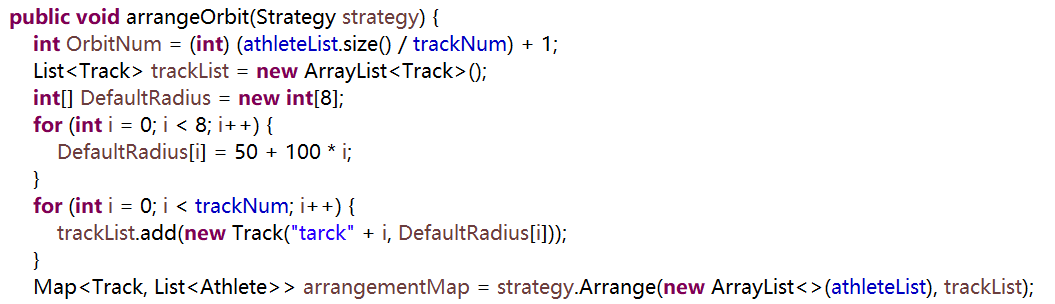
在TrackGame中设计一个strategy接口和两个具体实现的strategy类，分别能执行随机排序和按成绩排序。

只有一个方法



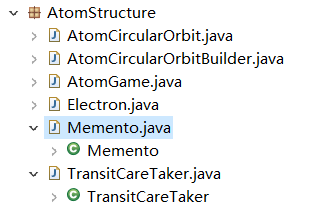
接收传入的运动员列表和赛道列表，返回构造好的关系Map。用于生成TrackCircularOrbit。

调用举例：

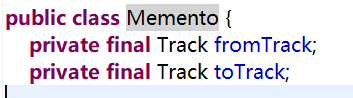


之后根据返回的安排好的关系Map。生成TrackCircularOrbit。

### 备忘录设计模式

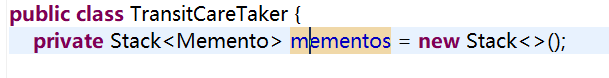


实现一个抽象ADT Memento



记录每次跃迁的起止轨道。

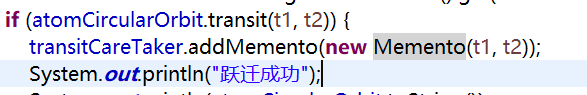
再实现一个TransitCareTaker类



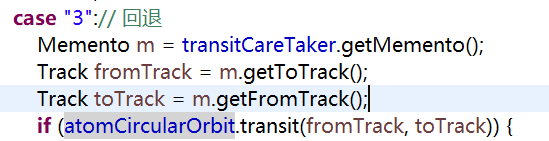
其中实现一个Memento的栈结构，契合恢复跃迁时先进先出的操作特点。

使用举例：

跃迁：



恢复：

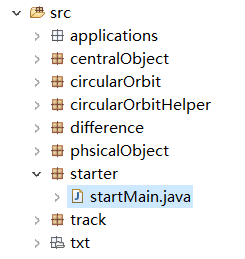


## 应用设计与开发。

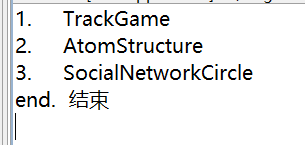
利用上述设计和实现的ADT，实现手册里要求的各项功能。

以下各小节，只需保留和完成你所选定的应用即可。

从starter下的startMain.java运行



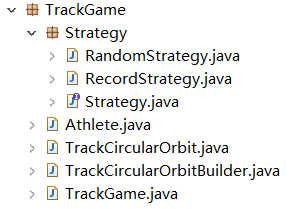
显示如下菜单：



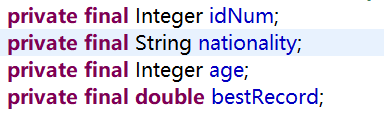
输入1，2，3进入功能

### 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

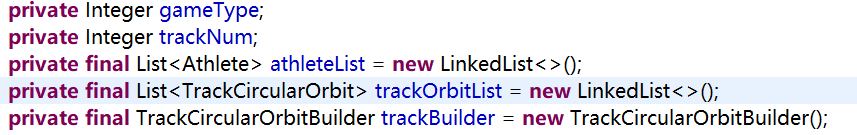
重写了toString方法输出多轨道结构

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。

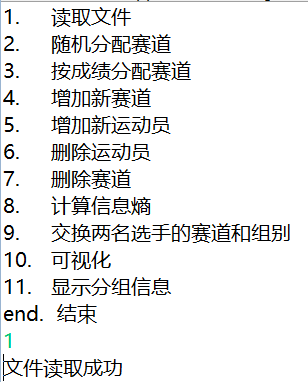
trackBuilder是TrackCircularOrbitBuilder的一个实例。用来build每一个TrackCircularOrbit对象。

方法:

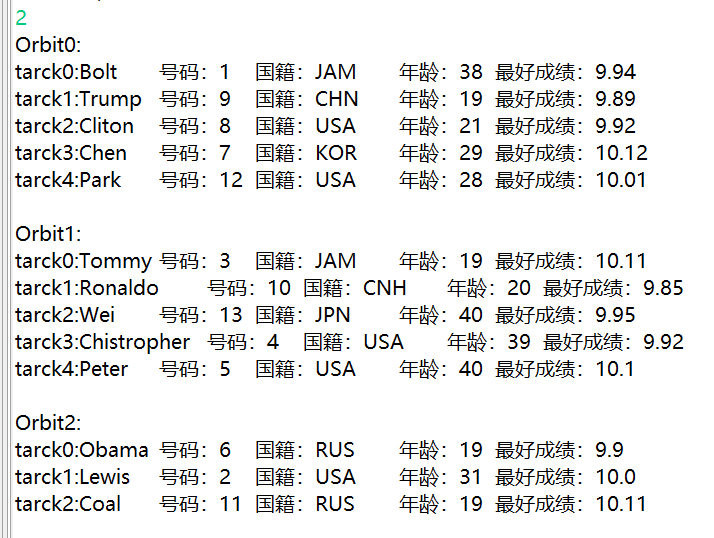
|  |  |
| --- | --- |
| 方法 | 功能 |
| GameMenu() { | 菜单 |
| gameMain | 功能主体，读取文件，构造每个TrackCircularOrbit，实现菜单中的每个功能。 |
| arrangeOrbit(Strategy strategy) | 策略安排方法，根据athleteList和trackNum和构造策略，生成构造方案 |

运行TrackGame.java文件出现菜单：

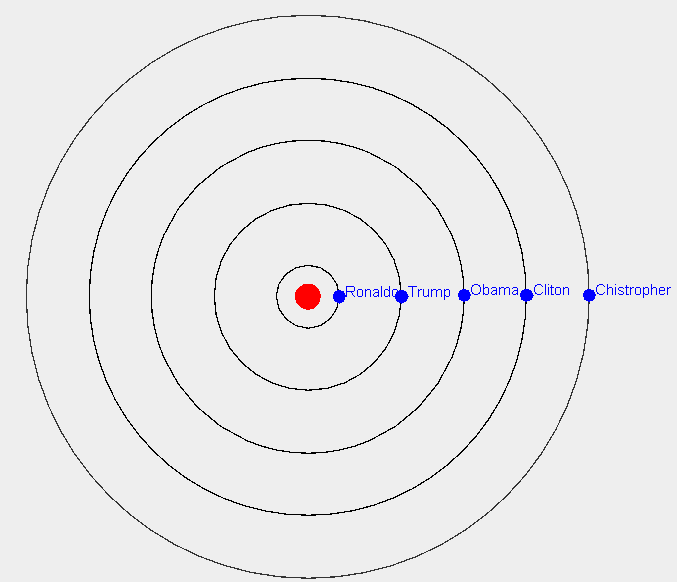
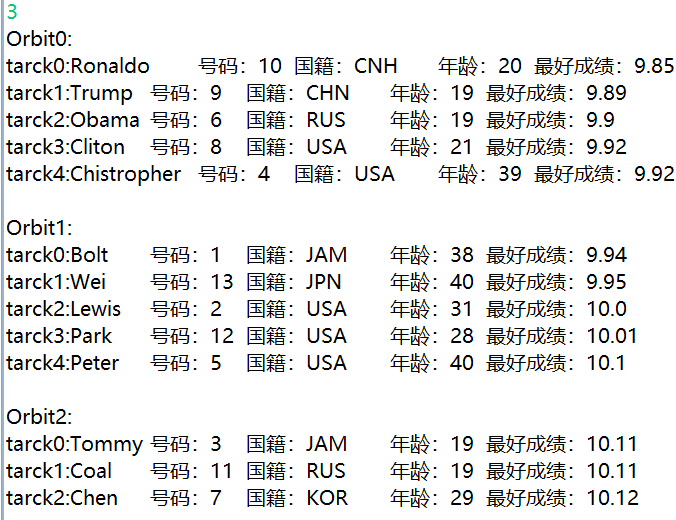
读取文件：

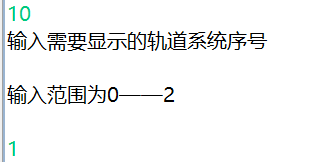


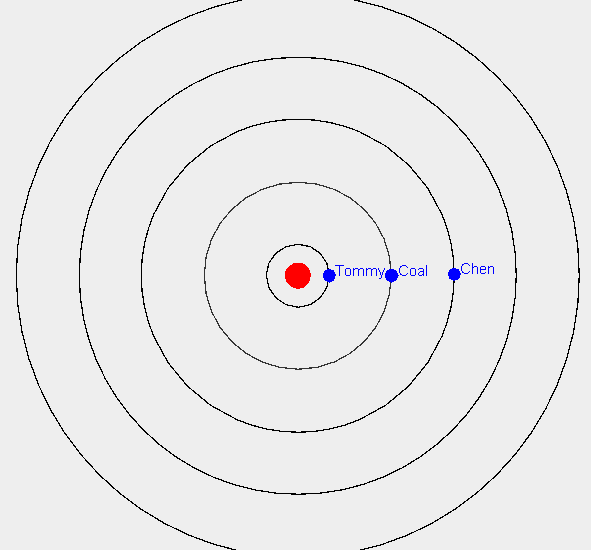
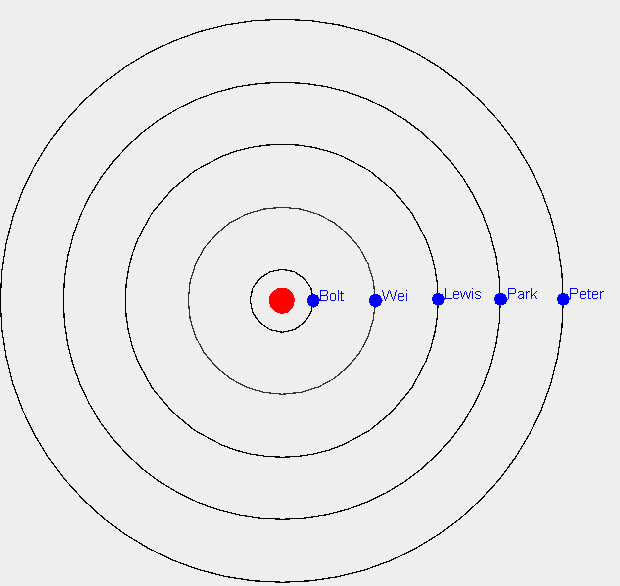
随机分配赛道：



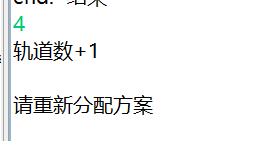
按成绩分配赛道：



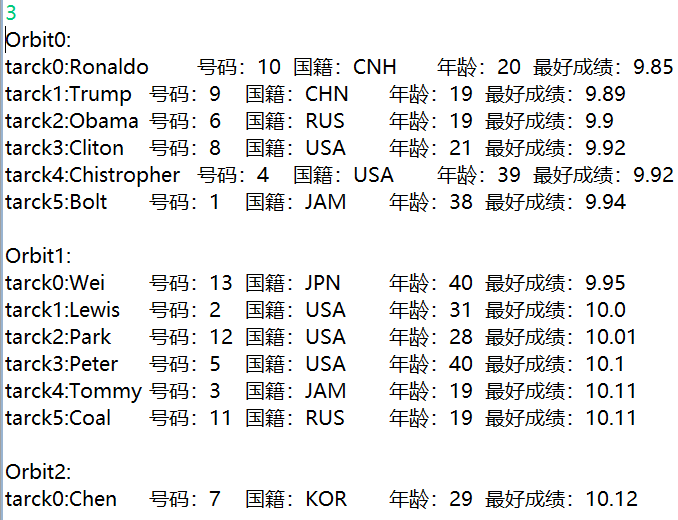
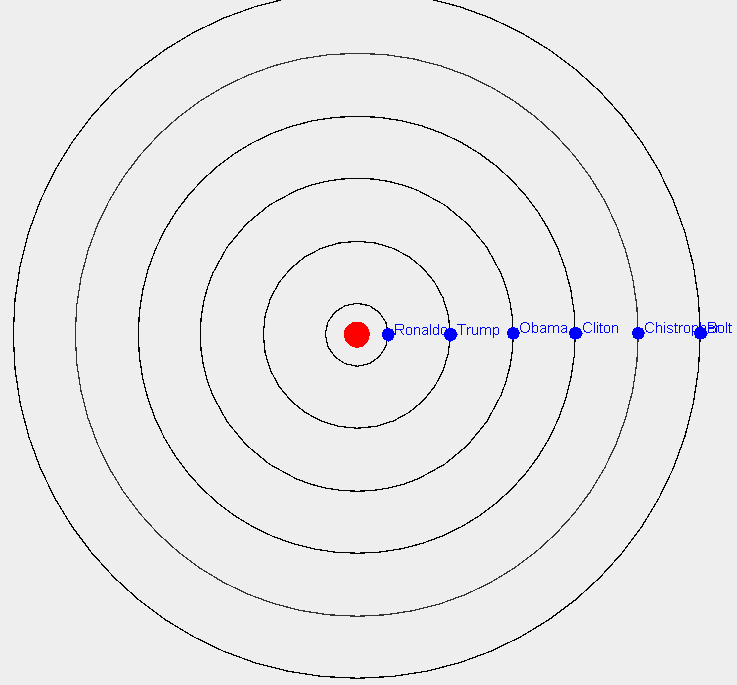
分别输入0 1 2显示三个图



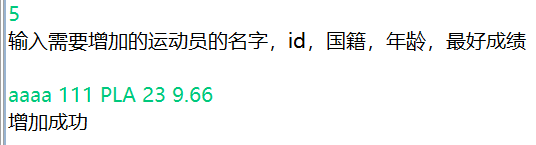
增加一条赛道：



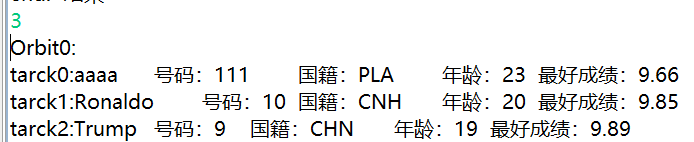
重新分配之后，可见轨道确实多了一条

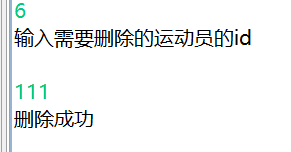
增加运动员：



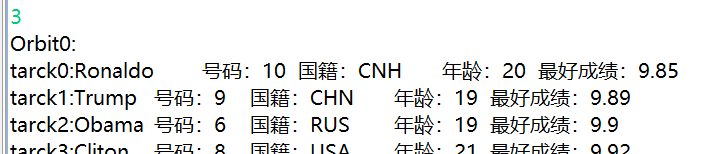
**可以看到，按成绩重新安排后出现了该选手，并且最快出场：**



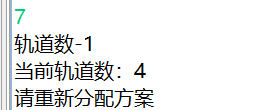
删除选手：



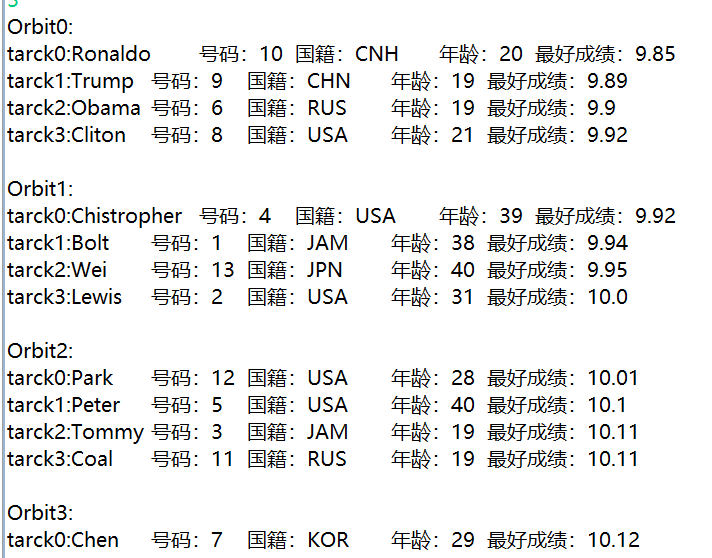
再次重新安排方案可见最快的选手aaaa已经移除：



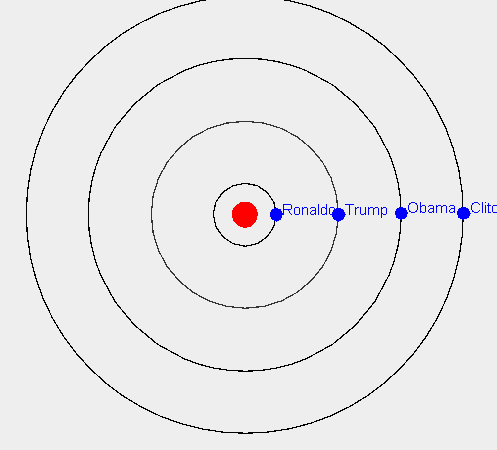
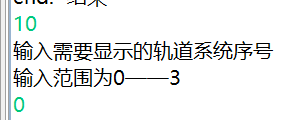
删除轨道：



可见赛道变为了四条

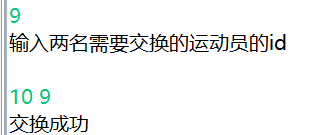


重新可视化：

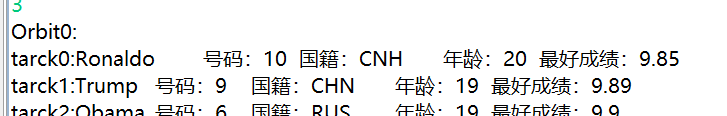


交换两名选手：

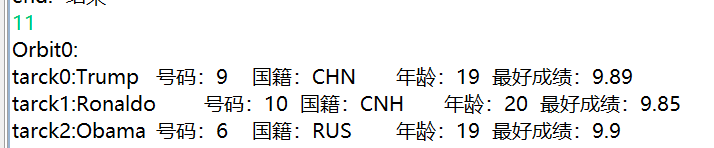
交换10号和9号：也就是按成绩排的头两名



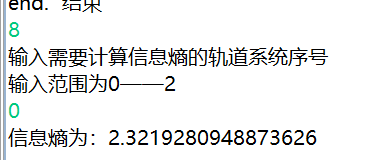
交换前：



交换后：

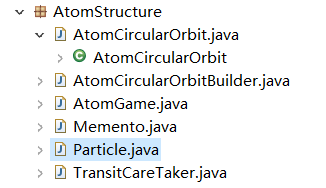


计算信息熵：



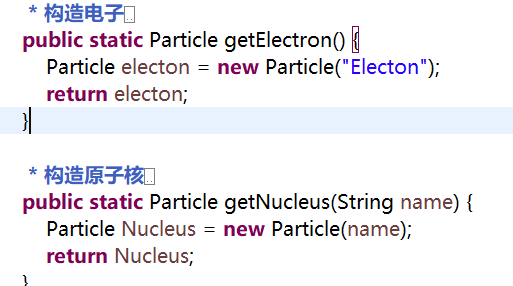
### AtomStructure

目录结构如下



其中：

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



getElectron返回一个没有区别的电子，

getNucleus返回一个有名字区别的原子核

Memento类和TransitCareTaker用来实现备忘录模式状态恢复功能，上面介绍过。

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

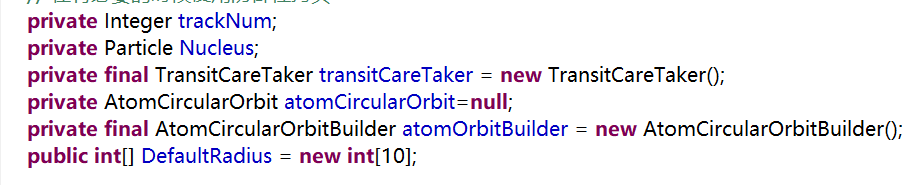
主要是重写了toString方法。新增方法如下：

|  |  |
| --- | --- |
| 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

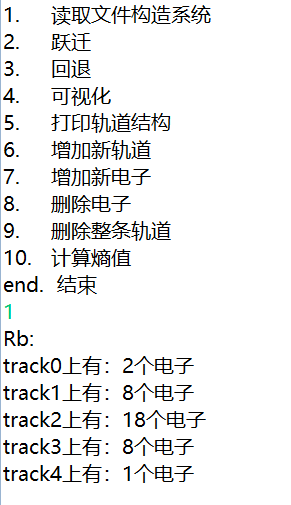
transitCareTaker是TransitCareTaker的一个实例。用来实现备忘录模式，具体是把每次跃迁记录成Memento保存下来，恢复时弹出上一个Memento。

atomCircularOrbit就是当前使用的轨道系统实例。

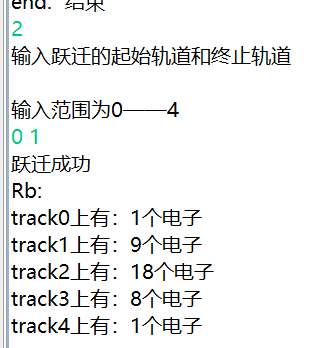
atomOrbitBuilder是AtomCircularOrbitBuilder的一个实例。用来build每一个AtomCircularOrbitBuilder对象。

功能演示：

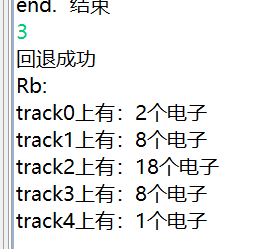
开始：读取文件并显示系统结构：



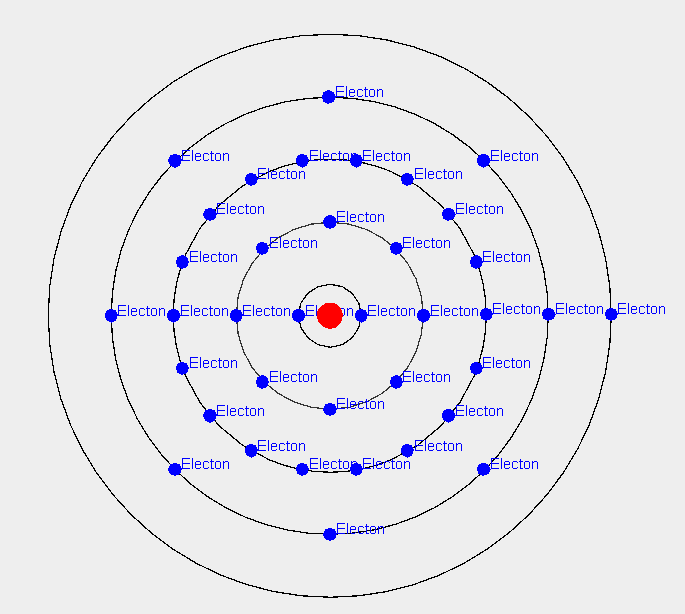
跃迁功能：



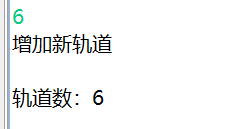
回退功能：



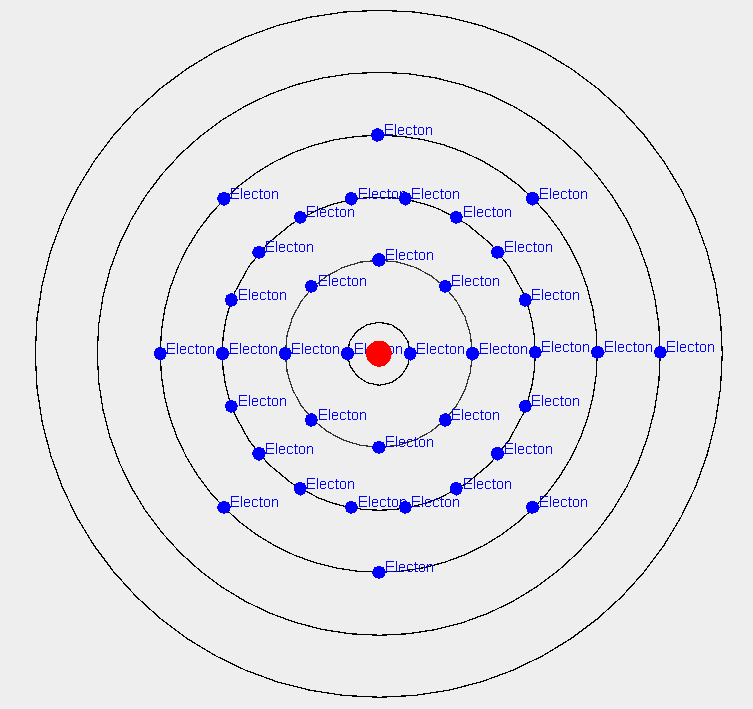
可视化：



增加新轨道：因为原子结构多条轨道没有区别，所以默认在外侧加一条：生成track5

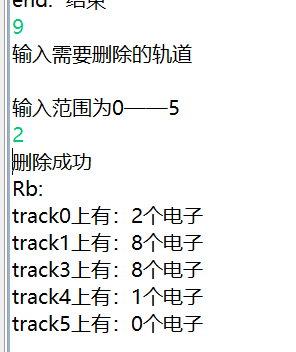


画图也可见外圈没有电子的新轨道：

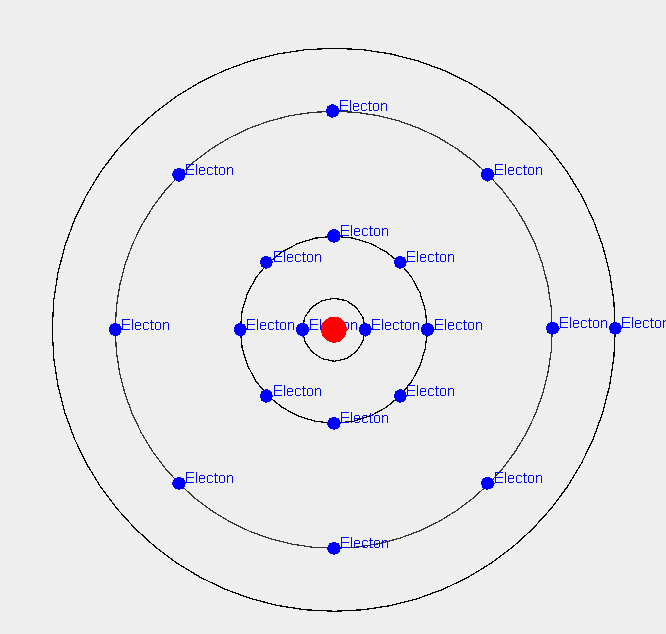


删除轨道：

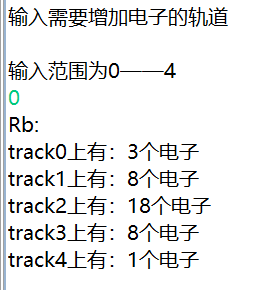
需要输入轨道序号：



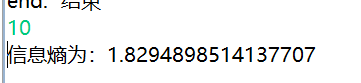
可见轨道2不见了



增加电子：因为电子之间互相没有什么不同，所以只需要指定增加电子的轨道：

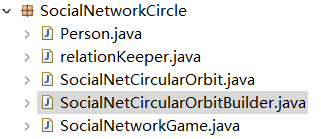


计算信息熵：



### SocialNetworkCircle

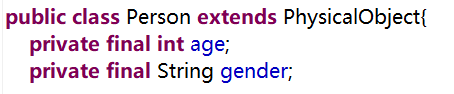
目录结构：



其中

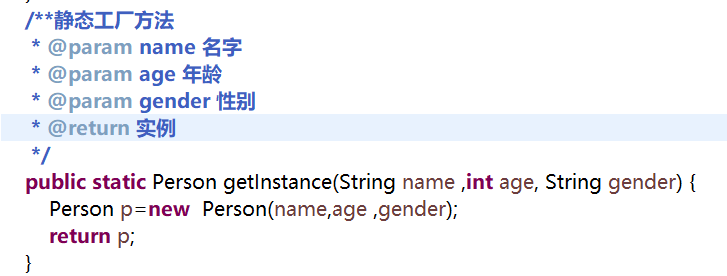
其中：

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：

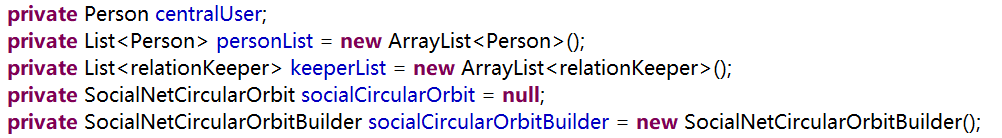
|  |  |
| --- | --- |
| 方法 | 功能 |
| reArrange | 重整关系网络 |
| 重写drawpicture方法 | 实现可视化，这个轨道结构要求可视化关系，所以加上画边的功能。 |
| 重写toString | 输出轨道结构 |

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：



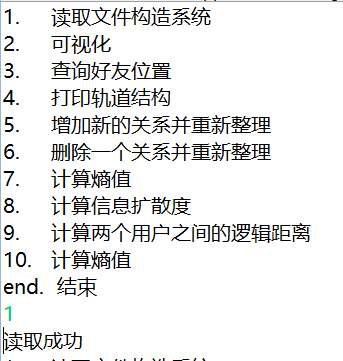
personList人列表，centralUser中心物体 ，keeperList是一个关系读取寄存器

socialCircularOrbit是当前操作的SocialNetCircularOrbit对象。

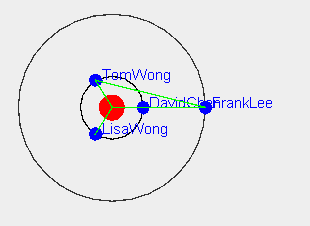
socialCircularOrbitBuilder是SocialNetCircularOrbitBuilder的一个实例。用来build每一个SocialNetCircularOrbit对象。

操作演示：

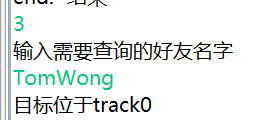
菜单



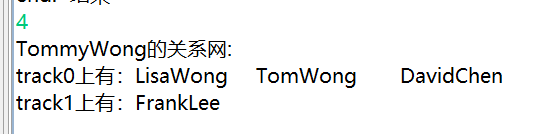
读取文件建立图：可视化：



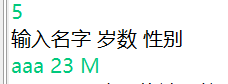
查询好友位置：



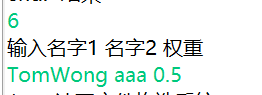
文字输出轨道结构：



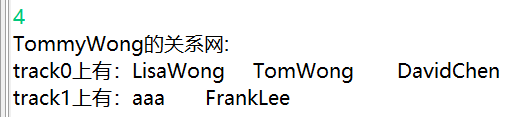
增加新的朋友：

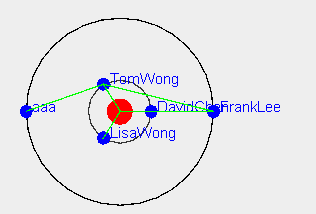


增加新的关系并重整结构：



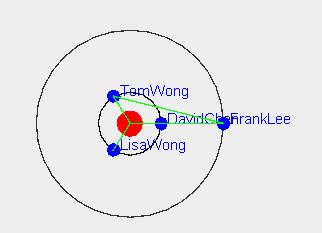
重新查看图：



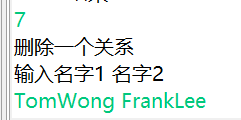
可见关系已经增加成功

删除关系并重整结构：

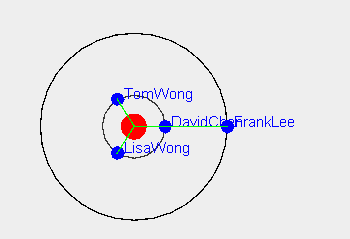
原结构：



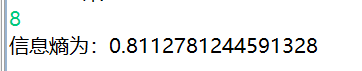
删除操作：



删除关系后：

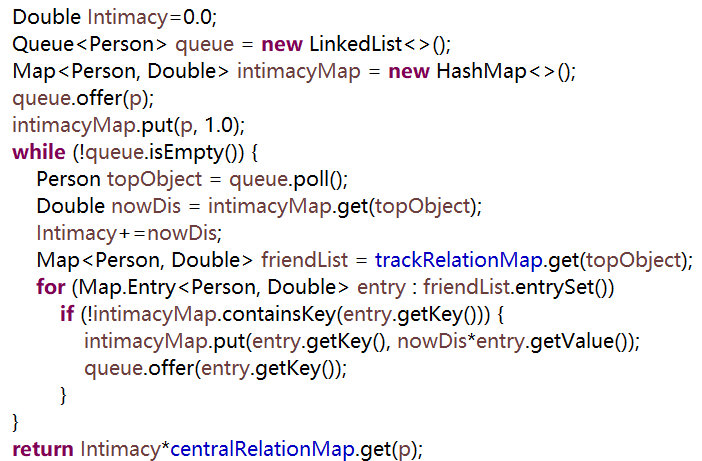
可见指定的关系已经被删除

计算信息熵：



计算信息扩散度：

代码：

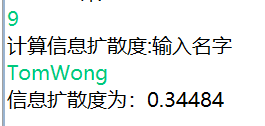


思路：主要利用队列+BFS算法，对于起始点：

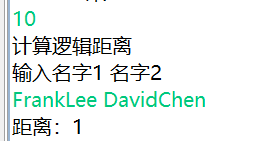
设置map为<起始点，中心点到起始点的权重>，随后对BFS算法找到的每个点，

设置map为<上一个点，上一个点的权\*上一个点到这个一个点的关系的权>。

例如图中存在关系a->b->c, a为中心点，则亲密度为：value(a,b)+ value(a,b)\* value(b,c)



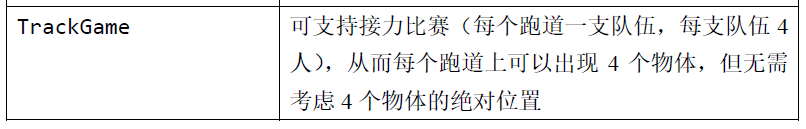
计算逻辑距离：



## 应对应用面临的新变化

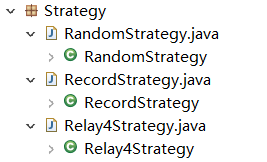
以下各小节，只需保留和完成你所选定的应用即可。

### TrackGame

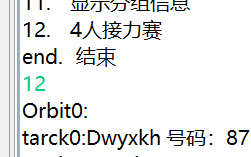


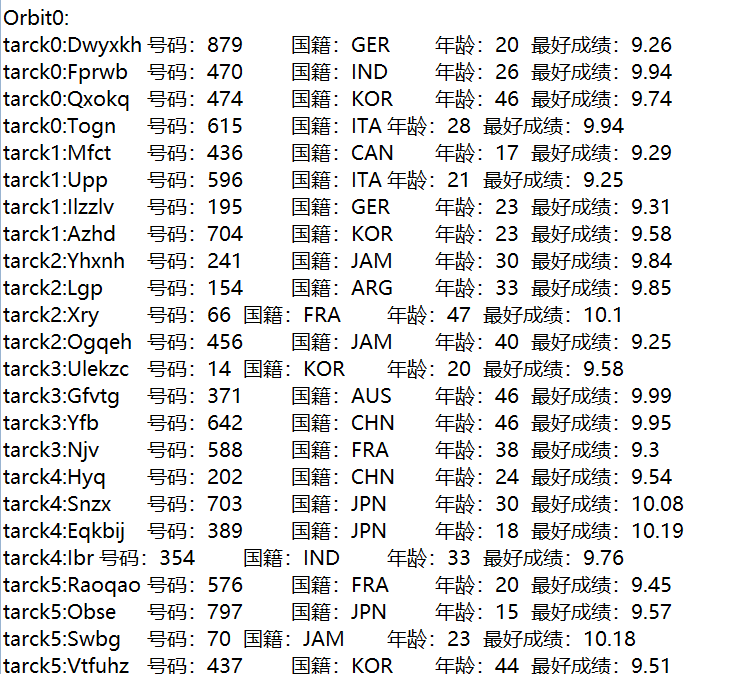
由于使用了strategy模式，

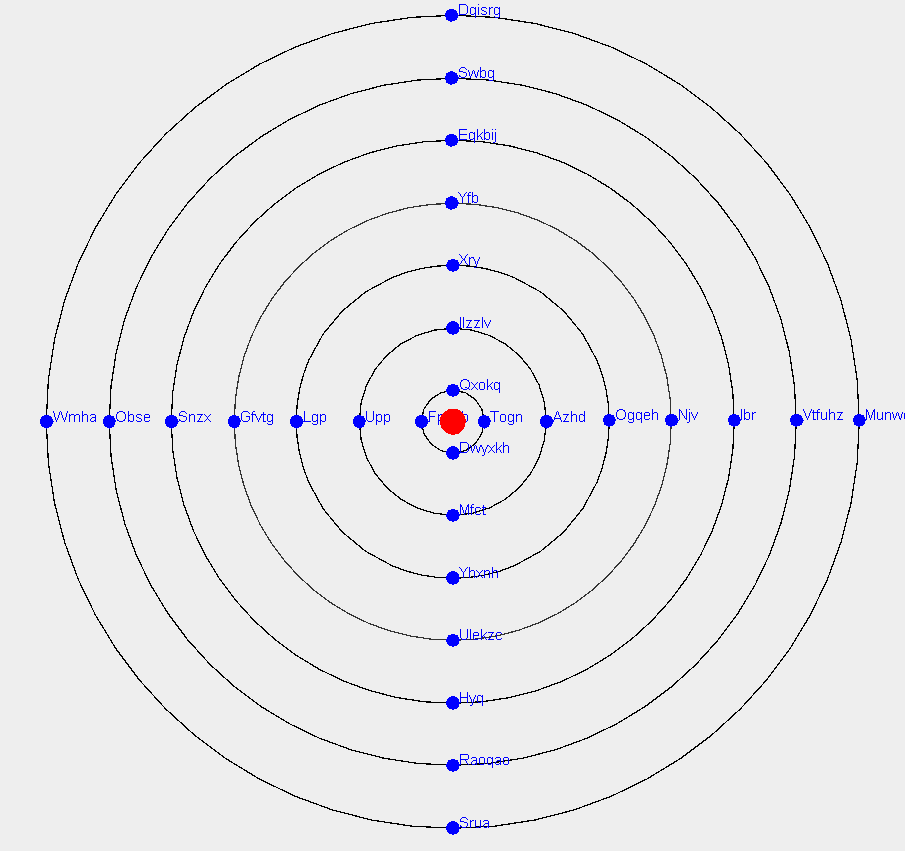
只需要新建一个strategy：Relay4Strategy继承自Strategy接口实现一次分配四个人分配就行了



运行展示：

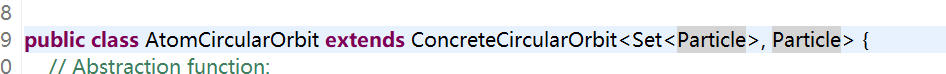




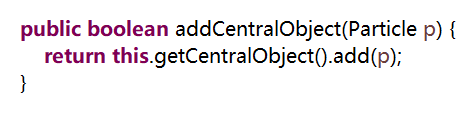


### AtomStructure

修改AtomStructure的声明



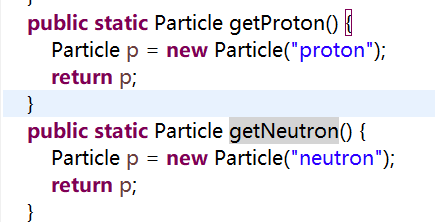
增加addCentralObject方法



修改AtomCircularOrbitBuilder中的bulidPhysicalObjects方法，在构造中心物体时传入一个集合。

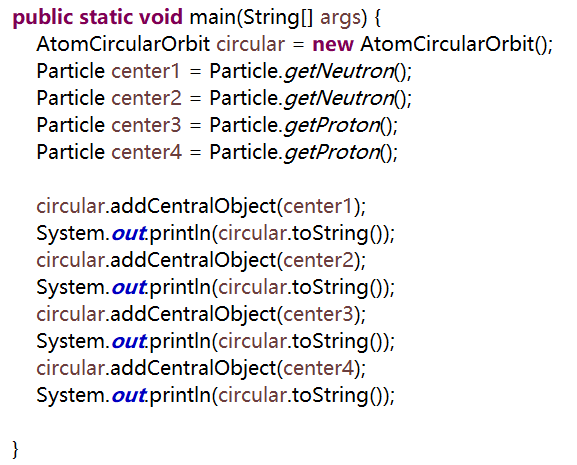
在文件读入时也改成读取中心物体，构造集合之后build。

增加Particle类的工厂方法使其能生成质子和中子

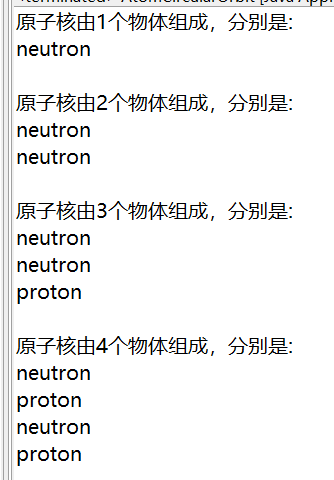


修改toString方法，输出原子核的具体情况。

写一个main方法测试下这些新的改动：

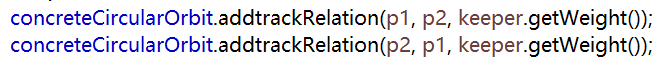


输出结果：



### SocialNetworkCircle

这个改动对于我的实现比较简单，因为之前用的就是有向关系，先前读取关系的时候就是相当于add两次相反的有向关系，这里只需要将改为只增加一个有向关系即可。

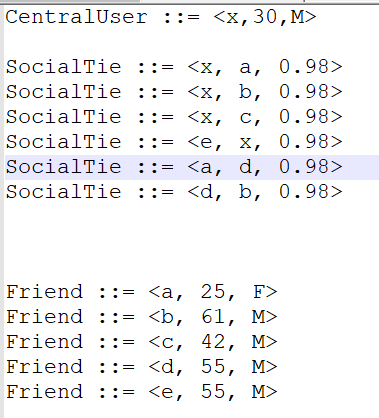


改为



写一个单元测试来测试：

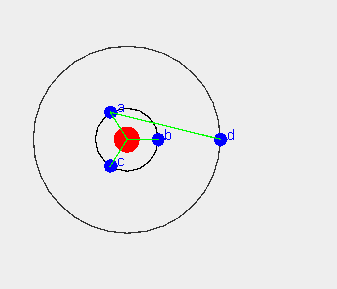
为了简单，构造一个十分简单的测试文件如下：



读取文件后判断：

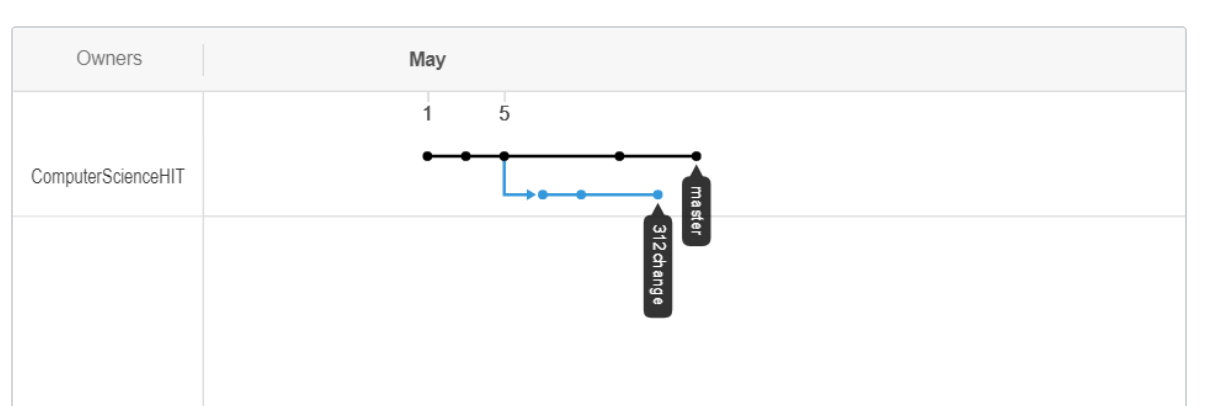
x->a, x->b, x->c, a->d都存在关系

d->b,e->x不存在关系如图：



## 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

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 |
| 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 |
| 正则表达式不熟练 | 上网学习 |
| Swing库不会用 | 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

拖延症严重，五一就干这事了，之前完成度很低。

可复用性考虑不足，change312还是改了不少东西。

上课效率一般，设计模式那块几乎是重新学了一遍。

## 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？
2. 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？
3. 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？
4. 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？
5. 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？
6. 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？
7. 你在完成本实验时，是否有参考Lab4和Lab5的实验手册？若有，你如何在本次实验中同时去考虑后续两个实验的要求的？
8. 关于本实验的工作量、难度、deadline。
9. 到目前为止你对《软件构造》课程的评价。
10. 复用极大减小了工作量，但也增加了思考量。
11. 时刻提醒自己注意自己的设计，注意防止泄露
12. 使用别人的API画图很方便很开心。感觉用几行代码就能做很牛逼的事情。
13. 合理运用设计模式十分有许多好处，针对不同的ADT使用恰当的设计模式很重要
14. 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。
15. 对物体的抽象
16. 参考了，但没有想法，还是先跟着课程学再说了
17. 难度中等但是工作量真的很大
18. 工作量很大。要求繁多。