

**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 | 强文杰 |
| 学号 | 1171000410 |
| 班号 | 1703005 |
| 电子邮件 | [672334335@qq.com](mailto:672334335@qq.com) |
| 手机号码 | 18800421389 |

**目录**

[1 实验目标概述 1](#_Toc7911115)

[2 实验环境配置 1](#_Toc7911116)

[3 实验过程 2](#_Toc7911117)

[3.1 待开发的三个应用场景 2](#_Toc7911118)

[3.2 基于语法的图数据输入 3](#_Toc7911119)

[3.3 面向复用的设计：CircularOrbit<L,E> 4](#_Toc7911120)

[3.4 面向复用的设计：Track 7](#_Toc7911121)

[3.5 面向复用的设计：L 7](#_Toc7911122)

[3.6 面向复用的设计：PhysicalObject 8](#_Toc7911123)

[3.7 可复用API设计 8](#_Toc7911124)

[3.8 图的可视化：第三方API的复用 10](#_Toc7911125)

[3.9 设计模式应用 13](#_Toc7911126)

[3.10 应用设计与开发 13](#_Toc7911127)

[3.10.1 StellarSystem 13](#_Toc7911128)

[3.10.2 AtomStructure 14](#_Toc7911129)

[3.10.3 SocialNetworkCircle 15](#_Toc7911130)

[3.11 应对应用面临的新变化 17](#_Toc7911131)

[3.11.1 StellarSystem 17](#_Toc7911132)

[3.11.2 AtomStructure 17](#_Toc7911133)

[3.11.3 SocialNetworkCircle 18](#_Toc7911134)

[3.12 Git仓库结构 20](#_Toc7911135)

[4 实验进度记录 20](#_Toc7911136)

[5 实验过程中遇到的困难与解决途径 21](#_Toc7911137)

[6 实验过程中收获的经验、教训、感想 22](#_Toc7911138)

[6.1 实验过程中收获的经验和教训 22](#_Toc7911139)

[6.2 针对以下方面的感受 22](#_Toc7911140)

# 实验目标概述

本次实验目标是编写具有可复用性和可维护性的软件，使用多种软件构造技术。如继承、代理、常见的OO设计模式等等。对于实验给定的五个具体应用，我们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（复用性）和更容易面向各种变化（可维护性）。

# 实验环境配置

简要陈述你配置本次实验所需环境的过程，必要时可以给出屏幕截图。

特别是要记录配置过程中遇到的问题和困难，以及如何解决的。

1. 根据助教在群里发的链接，完成Eclipse和JDK11的安装，然后去网上查询Java运行环境的配置，配置计算机的环境变量。

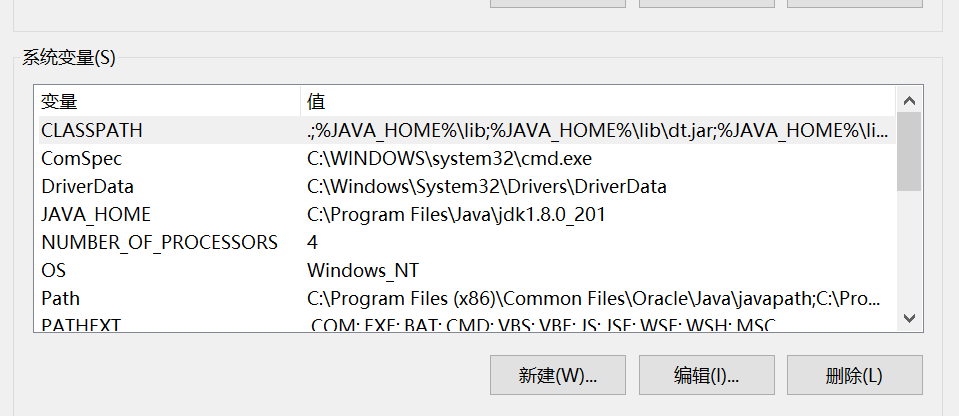


图2-1 环境变量配置

2.注册GitHub账号，并且授权GitHub classroom，获取lab3的GitHub classroom的URL地址。

3. Git代码配置管理。首先安装最新版Git，获取个人GitHub的URL地址，通过在Git bash输入指令建立管理本地仓库，将文件push到远程仓库。

git push origin master

**遇到的困难：**

1.在配置环境时，最早的参考教程是针对Mac OS，并且本地路径的设置出错，导致走了很多弯路。

2. 在安装Git之后，未第一时间在Git的安装目录注册user.name和user.email，导致后面输入命令时bash显示commend not found。

在这里给出你的GitHub Lab3仓库的URL地址（Lab3-学号）。

URL地址：https://github.com/ComputerScienceHIT/Lab3-1171000410.git

# 实验过程

请仔细对照实验手册，针对每一项任务，在下面各节中记录你的实验过程、阐述你的设计思路和问题求解思路，可辅之以示意图或关键源代码加以说明（但千万不要把你的源代码全部粘贴过来！）。

## 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）。

* StellarSystem
* 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。

**共性：**

1.都具备轨道系统中基本存在的对象，包括轨道、中心点物体、轨道物体.

2.都需要具备很多相同的操作：添加/删除轨道，在某一条轨道山添加/删除物体，从文件中读取轨道（读取方式不同）。

3.获得轨道系统的熵值，获得逻辑距离，比较两个同类型轨道的difference，检查轨道系统的合法化，GUI可视化。

**差异：**

1.StellarSystem的轨道物体考虑绝对位置，而其他两个系统无需考虑。

2.StellarSystem的轨道需要按照轨道半径进行排序，SocialNetworkCircle则是根据关系的级数进行轨道排序。

3.SocialNetworkCircle需要考虑复杂的映射关系，包括中心点物体映射轨道物体，和轨道物体之间的映射。并且在GUI可视化时，需要根据这些逻辑关系进行连线。

4.每个轨道系统都有着其特殊的功能，包括电子跃迁，计算行星物理距离和增加社交关系等。

## 基于语法的图数据输入

分别是在三个应用的读文件中设计正则表达式。然后对正则表达式进行解析匹配。

Pattern pattern = Pattern.*compile*(args[i]); // 构造正则表达式

Matcher mentioned = pattern.matcher(newFlieStr);//匹配正则表达式

**1.StellarSystem:**



模拟行星系统的正则表达式比较简单，括号捕获的顺序是从左到右，从外到内（嵌套）。

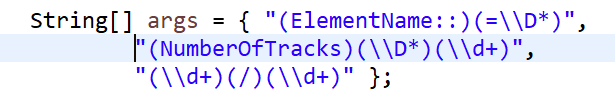
第一个括号内[\\w](file:///\\w)+是匹配多个字母，即一个单词，读取文件时每行先是一个单词，一个空格，一个'='，一个空格，后加一个max型的匹配，得到'='后所有信息项。

当mention.find时true时，

String[] args = mentioned.group(2).split("[\\s+ ,]");

将尖括号内的信息提取出来，如果后面的信息数目（字符串数组长度）为3，则读取到的是中心点的信息，否则读取的是行星的信息。

**2.AtomStructure**



对于原子轨道，我换了一种方式进行读取。

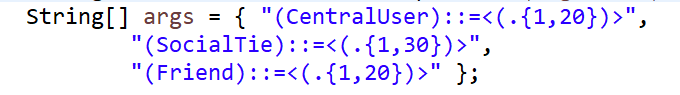
首先读入整篇txt，再去掉空格等空白符，使用如上的正则表达式进行全篇的匹配。

第一个正则表示式为了匹配原子核，其中group(2)获取后需要将多匹配到的NumberOfTracks字符串进行截去。

第二个正则表达式是为了获取group(3)中轨道的数目。

第三个正则表达式则是获取每级轨道的电子数目。

**3.SocialNetworkCircle**

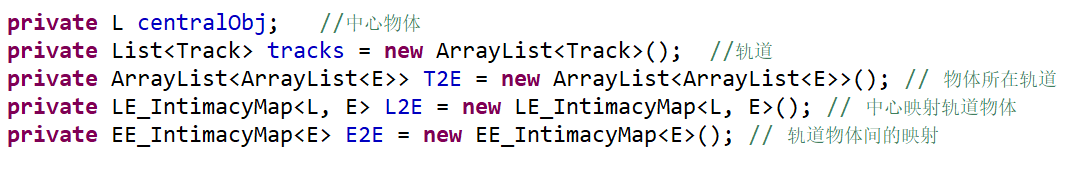


对于SocialNetworkCircle，因为读取SocialTie之前可能还没有读取Friend，因此我没有选择按行读取文件的方式，我依旧选择读取整个文件，去掉空格等空白符后全篇匹配。

按照字符串数组的顺序进行全篇匹配，保证在匹配SocialTie之前，系统中已经读取了Friend，获得mentioned.group(2)之后，再将其中的逗号分开，即可读取用户的信息，和建立用户之间的关系。

## 面向复用的设计：CircularOrbit<L,E>

**Fields:**



**域的设计思想：**

(1) 中心点物体；

(2) 一组轨道：将轨道存储在Arraylist中

(3) 每条轨道上的一组物体：使用了嵌套的Arraylist，因为T2E这个嵌套的list中，轨道的顺序和tracks轨道的顺序保持完全一致，因此不需要使用Map来表示这种映射关系，list的顺序存储已经隐含了track的序号。

(4) 中心点物体与轨道物体之间的关系：因为映射关系可能存在权值，因此使用自定义的类LE\_IntimacyMap来表示。

(5) 轨道物体之间的关系：因为映射关系可能存在权值，因此使用自定义的类EE\_IntimacyMap来表示。

// Abstraction function:

// represents a multi-track system where objects are in orbit

// central objects and orbital objects are mapped

// and there is a mapping between orbital objects.

// Representation invariant:

// centralObj is the central object of the system if it isn't null

// tracks is the list of tracks

// E2E is mapping between orbital objects

// T2E is track and object mapping

// L2E is central object and orbit objects mapping

// Safety from rep exposure:

// All fields are private and final

// tracks , T2E , L2E , E2E are mutable types, so operations use defensive copies and

// immutable wrappers to avoid sharing the rep's objects to clients

**方法及其实现：**

1.创建一个空的CircularObit对象

**public** **static** <L,E> CircularOrbit<L,E> empty(){

**return** **new** ConcreteCircularOrbit<>();

}

2.获得中心点物体：直接返回centralObj

**public** L getCentralObject();

3.获得轨道的集合：直接返回tracks的集合

**public** List<Track> getTracks();

4.获得所有轨道上的物体：直接返回T2E

**public** ArrayList<ArrayList<E>> getT2E();

5.获得中心物体和轨道物体的关系：直接返回L2E

**public** LE\_IntimacyMap<L, E> getL2E();

6.获得轨道物体之间的关系：直接返回E2E

**public** EE\_IntimacyMap<E> getE2E();

7.添加一条轨道：直接在tracks中添加track，并且T2E创建一条新的轨道，新轨道为空。

**public** **void** addTrack(Track track);

8.移除一条轨道：删除中心物体和轨道物体的映射，删除该轨道和其他轨道物体之间的映射，删除轨道物体，最后删除轨道。

**public** **void** removeTrack(Track t);

9.增加中心点物体

**public** **void** addCenterObject(L c);

10.增加轨道物体：从tracks获得轨道的序号，在T2E上找到对应的轨道，增加集合中的物体。

**public** **void** addTrackObject(Track t , E obj);

11. 删除轨道物体：从tracks获得轨道的序号，在T2E上找到对应的轨道，删除集合中的物体。

**public** **void** removeTrackObject(Track track, E obj);

12.增加中心点物体和轨道物体的关系：调用LE\_IntimacyMap中的put方法，增加图中的一条边。

**public** **void** addCenterTrackObjectsRelation(L c , E obj ,**float** intimacy);

13. 增加轨道物体之间的关系：调用EE\_IntimacyMap中的put方法，增加图中的一条边。

**public** **void** addTrackObjectsRelation(E obj1 , E obj2 ,**float** intimacy);

14. 删除中心点物体和轨道物体的关系：调用LE\_IntimacyMap中的remove方法，删除图中的一条边。

**public** **void** removeCenterTrackObjectsRelation(L c, E obj);

15. 删除轨道物体的之间关系：调用EE\_IntimacyMap中的remove方法，删除图中的一条边。

**public** **void** removeTrackObjectsRelation(E obj1, E obj2);

16.获得某个轨道上的物体：找到该轨道的序号i，返回T2E.get(i)

**public** ArrayList<E> getTrackObject(Track t);

17. 从外部文件读取数据构造轨道系统对象：待重写

**public** **void** readFile(File file);

**更多的共性操作：**

因为SocialNetworkCircle和AtomStructure都不需要考虑物体的绝对位置，而StellarSystem需要考虑物体的绝对位置。

因此设计了两个类WithoutPosition <L,E>和WithPosition <L,E>

它们继承了ConcreteCircularOrbit<L,E>

1.其中对于不需要考虑绝对位置的类，

方法transit()将 object 从当前所在轨道迁移到轨道 t：做法是遍历原轨道，将物体从原轨道进行删除，再添加到新轨道。

**public** **void** transit (E object, Track t)

2.对于需要考虑绝对位置的类，

**Field：**

**private** Map<E, Double> objectSitha= **new** HashMap<E , Double>(); //轨道物体的角度

**方法：**角度的setter、getter函数

将 object 从 当前位置移动到新的 sitha 角度所对应的位置，做法是将原物体与角度的映射删除，添加新的物体与角度的映射。

**public** **void** move(E object, **double** sitha)

**三个轨道系统的个性操作：**

先展示每个轨道系统与以上介绍的类之间的关系：

1.AtomStructure：

**public** **class** AtomStructure **extends** WithoutPosition<Nucleus, Electron> **implements** CircularOrbit<Nucleus, Electron>

2.StellarSystem

**public** **class** StellarSystem **extends** WithPosition<Stellar, Planet> **implements** CircularOrbit<Stellar, Planet>

3.SocialNetworkCircle

**public** **class** SocialNetworkCircle **extends** WithoutPosition<CentralUser, Friend> **implements** CircularOrbit<CentralUser, Friend>

它们分别都是继承了WithPosition或是WithoutPosition类，并且实现CircularOrbit类。

类中除了构造器，主要的方法就是重写读文件方法，构建不同的轨道系统。

## 面向复用的设计：Track

**Immutable类。**

**Fields**： **private** **final** **double** radius; //半径

**方法：**只有getter

## 面向复用的设计：L

表示中心点物体的父类CentralObject，L具体类都是**immutable**的。

1.其中CentralObject中的Field： **private** **final** String name; //名称

方法是getter

2.其余子类Stellar、CentralUser、Nucleus的Fileds包含它们的中心点物体的特定属性，

构造器中名称为super(name)，

方法也只有getter。

## 面向复用的设计：PhysicalObject

代表分布在不同轨道上的物体类PhysicalObject，将它设计为抽象类。

表示轨道物体的ADT都是immutable的。

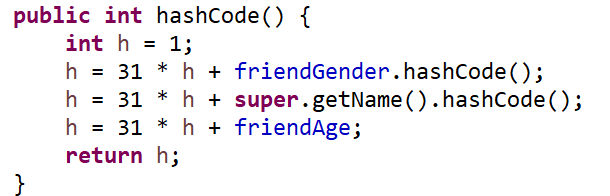
1.PhysicalObject类中的Field只有name，

方法是getter

2.子类Friend、Electron、Planet都构造器中名称为super(name)，其余的Field为各自独特的属性。

方法是getter。

特殊的是在Friend类中重写了hashCode()方法.



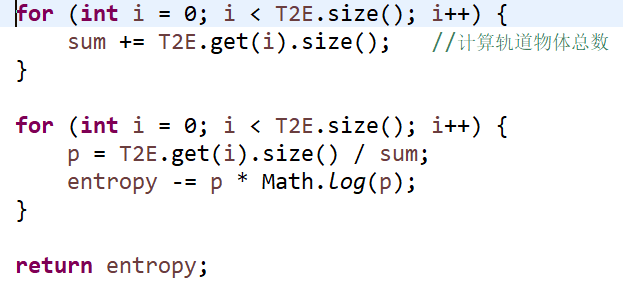
这是为了在SocialNetworkCircle的GUI可视化作图的时候，能通过每个Friend对应的不同hashCode，映射一个特定的位置，这样方便我们体现中心物体和轨道物体映射关系的时候，对特定点位置连接线段。

## 可复用API设计

1. **public** **double** getObjectDistributionEntropy(CircularOrbit c)

计算多轨道系统中各轨道上物体分布的熵值。

计算方法如图：



2. **public** **int** getLogicalDistance(CircularOrbit c, E e1, E e2)

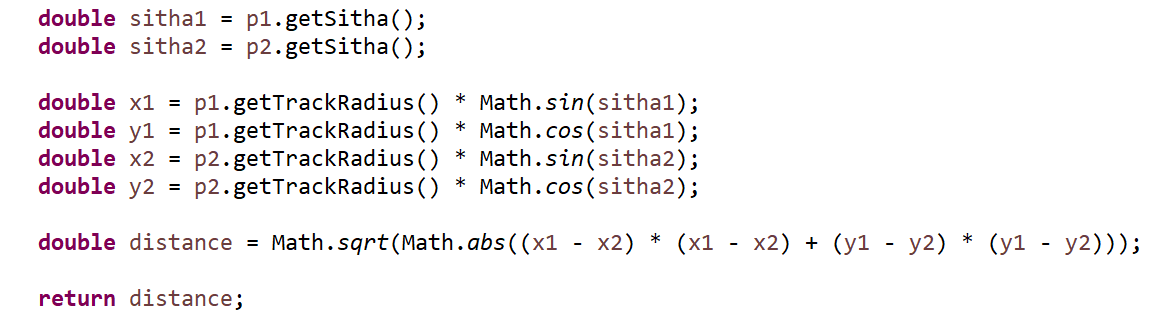
计算任意两个物体之间的最短逻辑距离。

和之前实验类似。因为这里的逻辑距离是指最少通过多少边可以连在一起，因此单位都是1，可以使用BFS对EE\_IntimacyMap进行搜索。其中获取某个朋友相邻的朋友的时候，可以调用EE\_IntimacyMap.getAdjacentObjects()方法返回相邻的物体。

3. **public** **double** getPhysicalDistance(CircularOrbit c, E e1, E e2)

计算任意两个物体之间的物理距离。针对StellarSystem。

将半径和角度的极坐标系转化为直角坐标系，计算两点之间的距离。

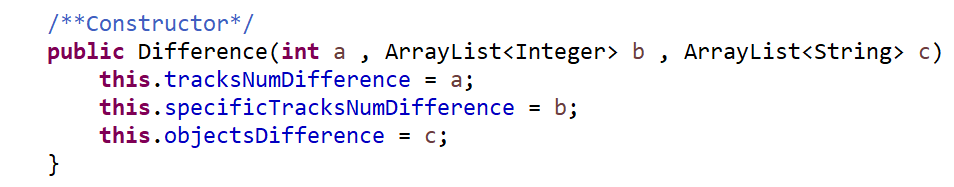


4. **public** Difference getDifference(CircularOrbit c1, CircularOrbit c2)

计算两个多轨道系统之间的差异。

Difference类的构造函数如下。Difference类中的方法设计为将构造器传入的参数打印出来，

即输出轨道数、轨道物体等不同。



在获取两个同类型轨道系统difference的时候，还是需要分三种情况进行讨论。

4.1.AtomStructure

轨道数的差异很易得，但是对于特定轨道物体数差异时，需要知道哪个系统的轨道数更多，这样轨道数少的系统在轨道不存在时，视其轨道物体数目为0即可。

4.2.StellarSystem

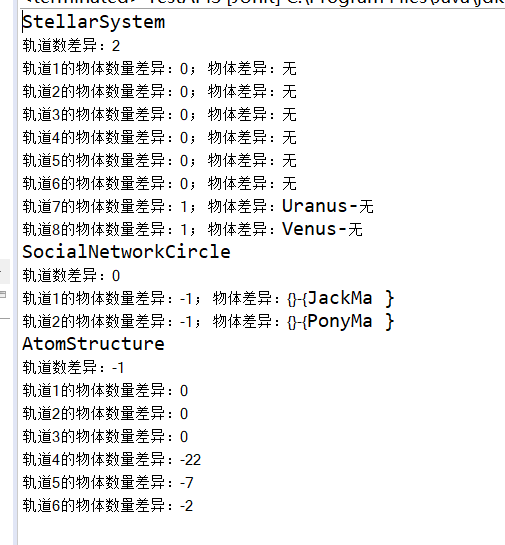
模拟行星的系统在找difference时和原子轨道系统差别不大。因为行星系统每个轨道上只有一个行星，因此我们在输出轨道物体数目差异的同时，将物体的name打印出来即可。需要注意的依然是对于轨道数较少的系统，当其比较到空轨道时，轨道物体数为0，轨道物体名为“无“。

4.3.SocialNetworkCircle

因为对于社交网络，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。

此时和StellarSystem的差异在于，找轨道物体差异的时候，需要对系统1某轨道的每个物体，在系统2对应轨道上找不同，并将它们的字符串连接起来。同样的，对于SocialNetworkCircle也需要根据系统轨道数的多少分情况讨论。

此功能添加在了应用中，下图是采用不同的轨道测试的输出图：



## 图的可视化：第三方API的复用

**1.静态图的实现**

由于AtomStructure的GUI实现比较简单，下面只介绍SocialNetworkCircle的GUI实现，此图较AtomStructure的GUI实现，添加了中心物体对于轨道物体的映射连线，和轨道物体之间映射的连线。

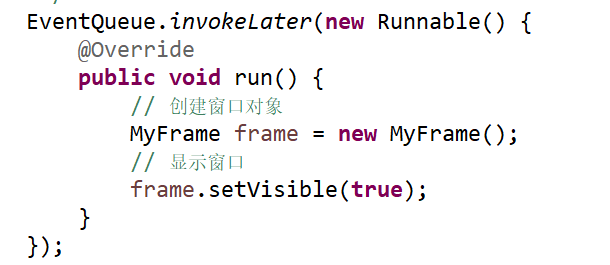
具体实现步骤：

1.窗口类：继承自JFrame，初始化分为以下几步：设置窗口标题和窗口大小；设置窗口关闭按钮的默认操作(点击关闭时退出进程)；把窗口位置设置到屏幕的中心；设置窗口的内容面板。

2.内容面板类：继承自JPanel。重写paintComponent方法，并在方法中调用drawArc(Graphics g)。

3. **public** **static** **void** visualizes(CircularOrbit c)

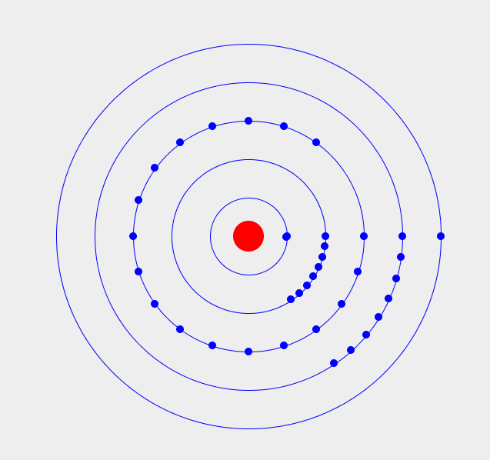
这个静态方法主要的目的时获取轨道系统GUI所需要的参数，包括轨道数目，中心物体和轨道物体的映射L2E，轨道物体之间的映射E2E。另外，在 AWT 的事件队列线程中创建窗口和组件,确保线程安全, 即 组件创建、绘制、事件响应 需要处于同一线程。



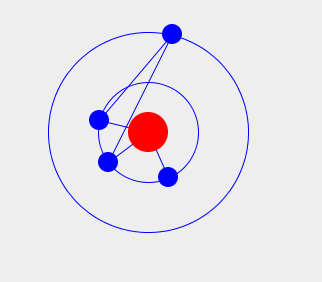
4.具体谈作图过程：因为是圆轨道，因此我们作图前需要将圆心位置确定，定为（400，400），对于每个轨道，我们使用drawArc画一个360度的圆弧，即圆。对于中心点物体，我们使用fillArc填充一个圆即可。

较为麻烦的是画出轨道物体，并且记录它们的位置进行连线。首先，轨道物体的圆心在轨道圆周上，另外物体自身也可使用fillArc填充为一个实心圆。对于轨道物体特定的位置，我们调用物体的hashCode方法，这样就可以将每个物体映射到一个确定的且唯一的位置。因为我们已经将轨道系统当作参数传入，并且获得了映射关系，因此调用drawLine先将中心物体和第一轨道的物体连线，再找轨道物体之间，如果存在相邻的关系，则将二者连线。

在GUI上可视化AtomStructure如下：



在GUI上可视化SocialNetworkCircle如下：



**2.动态图的实现**

动态图很多基础的实现和静态图是一致的，这里主要介绍我们究竟如何让图“动“起来。

**Circle类：**

**private** Point2D.Double center; //中心坐标

**private** **double** radius; //半径

**private** **int** tick; //记录在轨道的位置

**private** **int** orientation; // 1或-1

**private** **int** numOfTrack; //轨道

**private** String color; //颜色

其中初始的tick是根据轨道物体的角度设置，在轨道上物体运动的过程中，tick是不断加1的；orientation取1或-1，这是为了物体的角度增加或是减少（顺时针或是逆时针）；color颜色是根据planet颜色设置，其中包含了fillArc方法。

**Fields：**

**private** ArrayList<Circle> centers = **new** ArrayList<Circle>();

**private** ArrayList<Circle> orbiters = **new** ArrayList<Circle>();

centers和orbiters分别是轨道圆和orbiters的集合。

**主要的方法：**

**2.1 private** **void** updateOrbiterLoc()

根据orbiters的tick值，更新其位置。

**2.2** **public** **void** tick()

相当于时钟的“滴答“，所有轨道物体的tick值发生变化，变化大小为其orientation值（1或-1），每次tick都需要调用updateOrbitLoc方法更新其位置。

2.3 **public** **void** draw(Graphics2D g)

centers和orbiters分别调用Circle类中的drawCenter和drawOrbit方法进行画图。

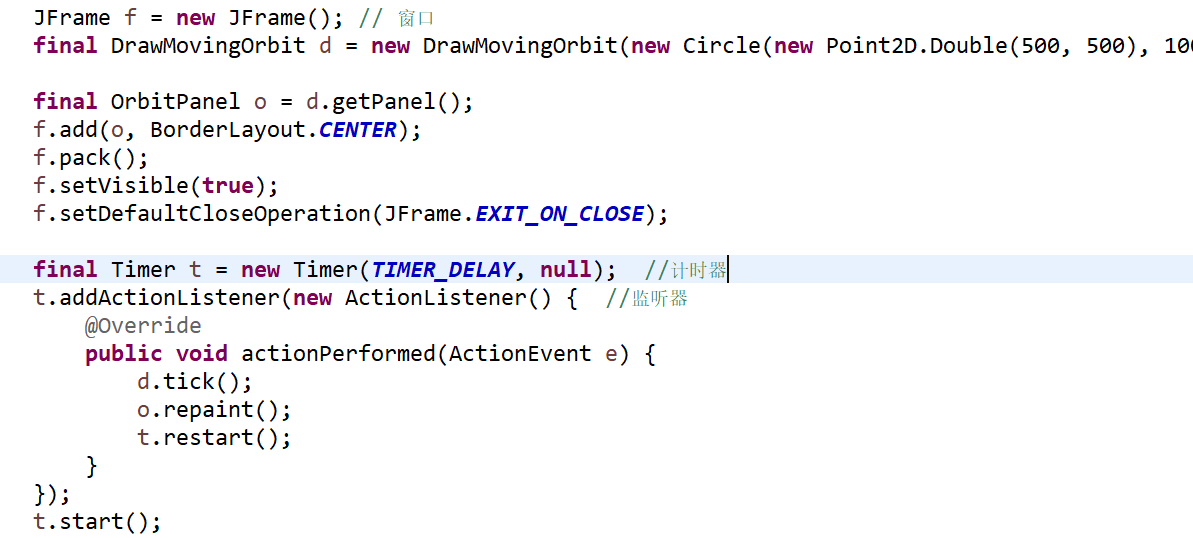
2.4 **public** DrawMovingOrbit(Circle center)

构造器，根据传入center进行画圆，其中每个轨道上都有个center半径1/8的orbiter ，根据初始的tick值，设置初始位置。

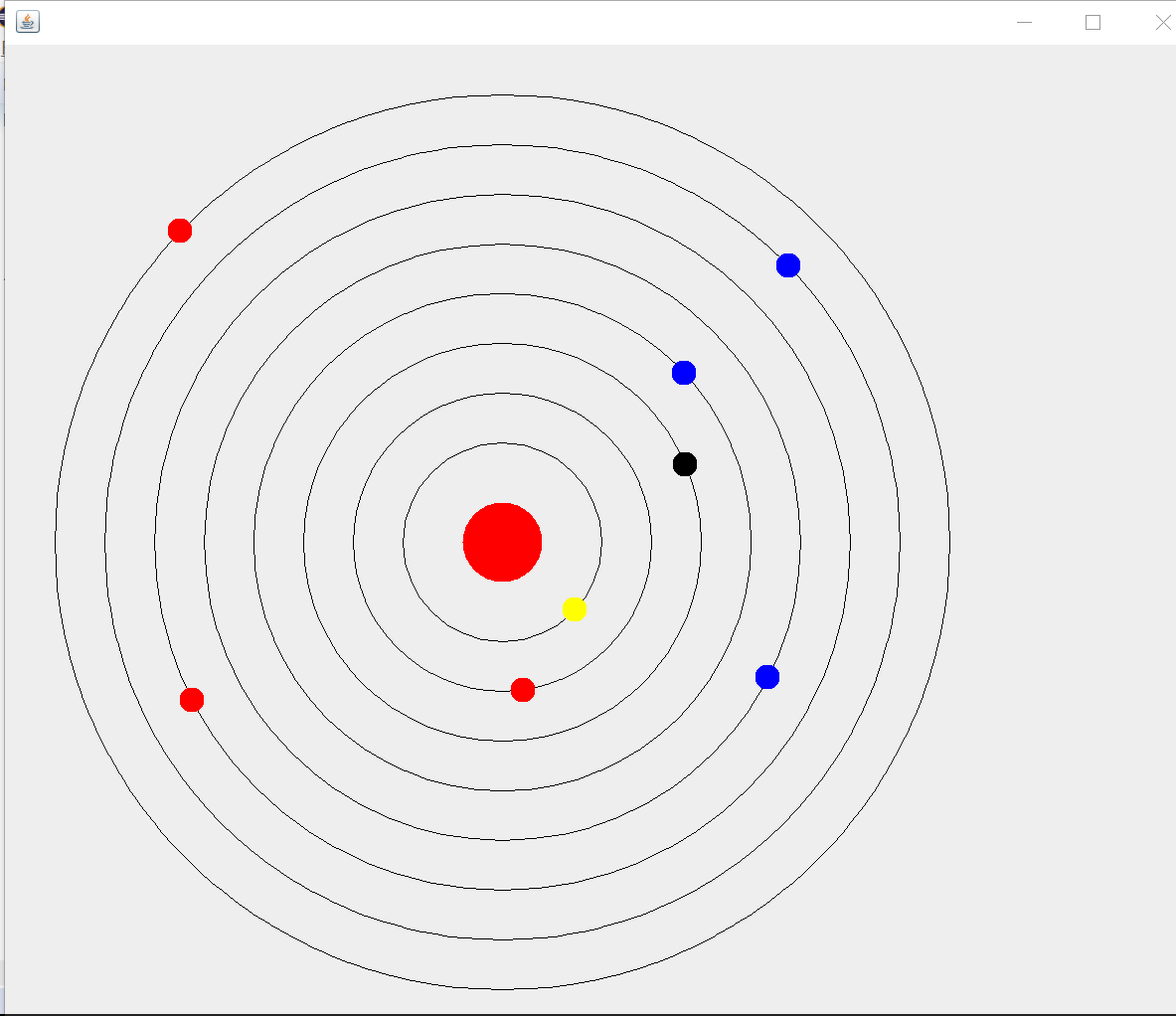
2.5 **public** **static** **void** visualizes(CircularOrbit c)

这个方法一方面是为了从多轨道系统中获取数据，另一方面，承载了主要的作图过程。

最主要的是线程定时器Timer和监听器的使用，为了在进行某一个操作的时候触发更新功能轨道物体位置的功能，重写了actionPerformed方法，在其中分别调用tick()更新轨道物体位置、repaint()重新作图、restart()重启计时器。



画出的动态图如下（不会截成动态的）：



## 设计模式应用

请分小节介绍每种设计模式在你的ADT和应用设计中的具体应用。

1.简单工厂设计模式

建立一个工厂类，对实现了同一接口CircularOrbit的一些类进行实例的创建，实质是由一个工厂类根据传入的参数，动态决定应该创建哪一个产品类（Multi-track system）的实例。

2.模板模式

因为三种轨道系统的相似之处非常多，因此我们可以将这些步骤封装起来，然后利用继承的方式来继承即可，当然对于readFile读文件方法可以自己重写实现。模板方法使得子类（Multi-track system）可以在不改变算法结构的情况下，重新定义算法中的某些步骤。

其余的设计模式没有用到。

## 应用设计与开发

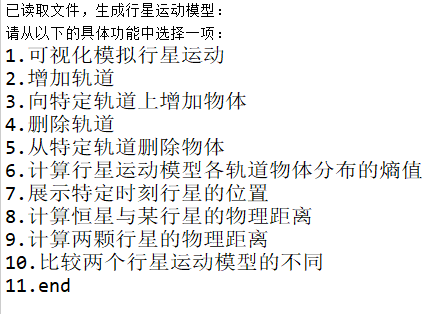
利用上述设计和实现的ADT，实现手册里要求的各项功能。

以下各小节，只需保留和完成你所选定的应用即可。

### StellarSystem

1.先读取文件，生成行星运动模型。在将生成的行星运动模型作为参数，传入检查合法性的函数中。合法性检查：中心点恒星不为null；一个轨道上行星的list大小只能为1；相邻轨道的半径之差不能小于两颗相应行星的半径之和。

2.打印菜单，让用户选择



3.通用功能：

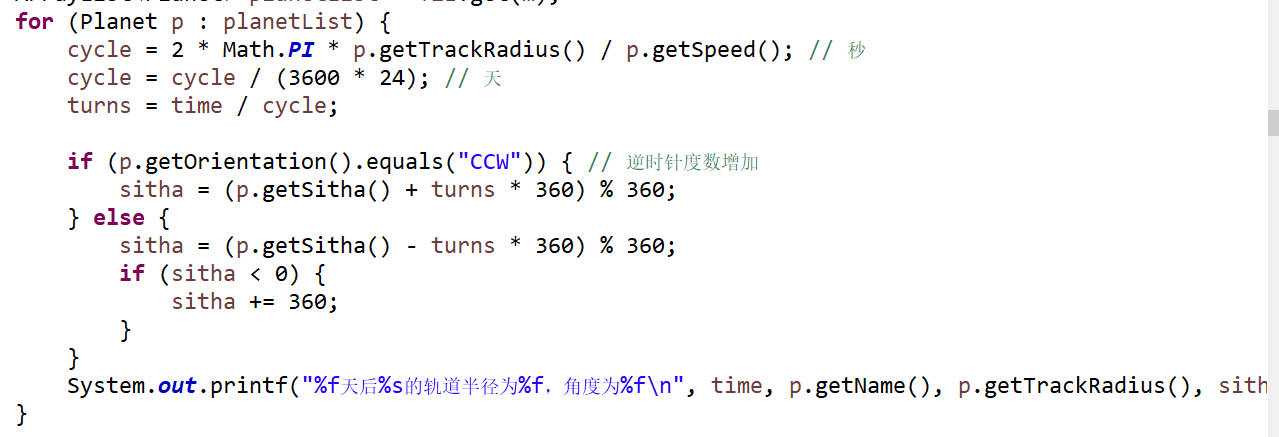
对于从轨道上添加删除物体，添加删除轨道这些功能，我们只需要让用户输入轨道半径来获取轨道，输入行星的参数等，再调用CircularOrbit接口提供的方法即可。

4.特殊功能

可视化模拟行星运动：DrawMovingOrbit.*visualizes*(ste); 调用DrawMovingOrbit类中的方法来画动图其中ste是传入的行星运动模型

计算行星运动模型各轨道物体分布的熵值：调用CircularOrbitAPIS中的getObjectDistributionEntropy方法即可。

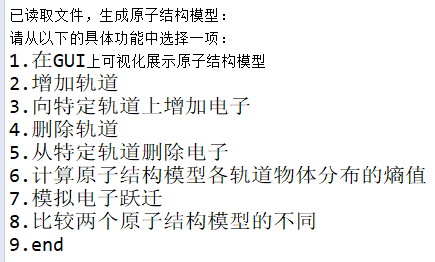
展示特定时刻行星的位置：让用户输入经历的天数，转化为秒，再根据行星公转的速度和方向，计算出某时刻行星的角度，轨道半径不变。



计算恒星与某行星的距离：直接返回轨道半径的大小即可。

计算两颗行星之间的距离：调用CircularOrbitAPIS中的getPhysicalDistance方法。

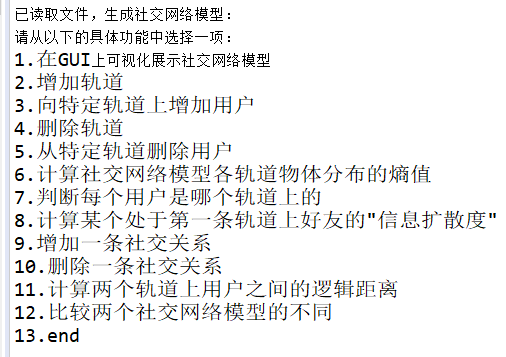
### AtomStructure



菜单上功能2-6的实现和行星模拟系统完全相类似。功能1在GUI上可视化展示原子结构模型只需要CircularOrbitHelper2.*visualizes*(ast);

特殊的功能只有模拟电子跃迁，实现方式是调用transit函数，将电子从原轨道删除，再添加到新轨道即可。

### SocialNetworkCircle



**1.对于前6个功能**，实现方式与AtomStrcture和StellarSystem完全相类似。只是在某个轨道上添加或删除用户的时候，需要添加或删除其对应的SocialTie，改变图的结构和映射关系，实现的手段就是调用LE\_IntimacyMap和EE\_IntimacyMap中的put和remove方法即可。

**2.特殊功能**

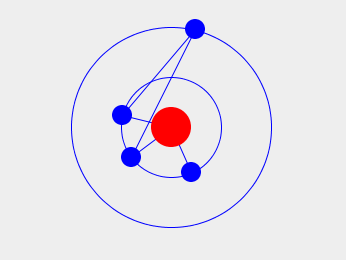
判断每个用户是哪个轨道上的：遍历所有的轨道，打印轨道用户

计算某个处于第一条轨道上好友的“信息扩散度“：先让用户从第一条轨道上选择好友，我们遍历该好友的所有朋友（不包括CentralUser），如果两人社交的亲密度大于设定值（暂设为0.5），则可将“信息扩散度”加1，最后打印出“信息扩散度”。

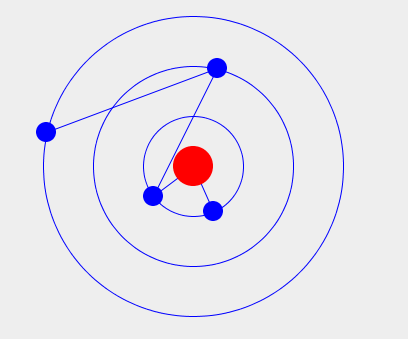
增加和删除一条社交关系：这是所有应用中最难的部分。我们先要判断SocialTie是轨道用户之间的，还是轨道用户和CentralUser之间的，分别调用EE\_IntimacyMap和LE\_IntimacyMap中的方法增加和删除图中的映射即可。难点在于如何调整图的结构。我的做法是，保留图中存在的所有映射，将轨道和轨道上的物体清空（对象还在），然后将SocialNetworkCircle中读取图是建立轨道的部分，封装到一个BuildRelation类中，这样在图的结构每次改变之后，保留映射关系和对象，重写建立轨道即可。

比较社交模型的不同：调用APIS中的方法。

原轨道图如下：



删除了TommyWong和DavidChen之间的SocialTie之后，DavidChen从第一轨道移到了第三轨道，如图所示：



**3.合法性检查**

还是调用CircularOrbitAPIS中的getLogicalDistance方法，获取每个人和中心点的距离，取最短距离，比较和所处的轨道序号是否相同。

在遍历的同时，如果第二轨道及更高轨道上的物体和第一轨道的物体之间没有联系，则它们不应出现在轨道系统中。

## 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。

### StellarSystem

首先我们需要自己写一个txt文件MyStellarSystem，将其中的半径那一项，改为半长轴a，半短轴b，和焦点c，根据这个文件我们建立行星运动系统的椭圆轨道。

修改Planet和Track的Fileds，将其中的轨道半径分别改为半长轴a，半短轴b，和焦点c，同时修改Constructor和getter方法。

其次就是修改读文件部分了，因为我们之前写的正则表达式就是按行读取，然后将等号后面的部分按照逗号分开，获取数据，因此这次处理改动不大，只需要将之前字符串分割的8项改为分割成10项即可。

然后我们需要根据提示的一些错误简单修改其他类，正确地传入参数。代码修改量和花费的时间不多。

最后一部分时考虑如何可视化模拟一个椭圆运动的行星轨道，这乍一看很难，其实程序要修改的地方并不多。首先是从轨道系统中获得各个行星的参数（由半径radius变化为长半轴、短半轴、焦点），为了方便处理，我们将读入的shortAxis设置为longAxis的一半。

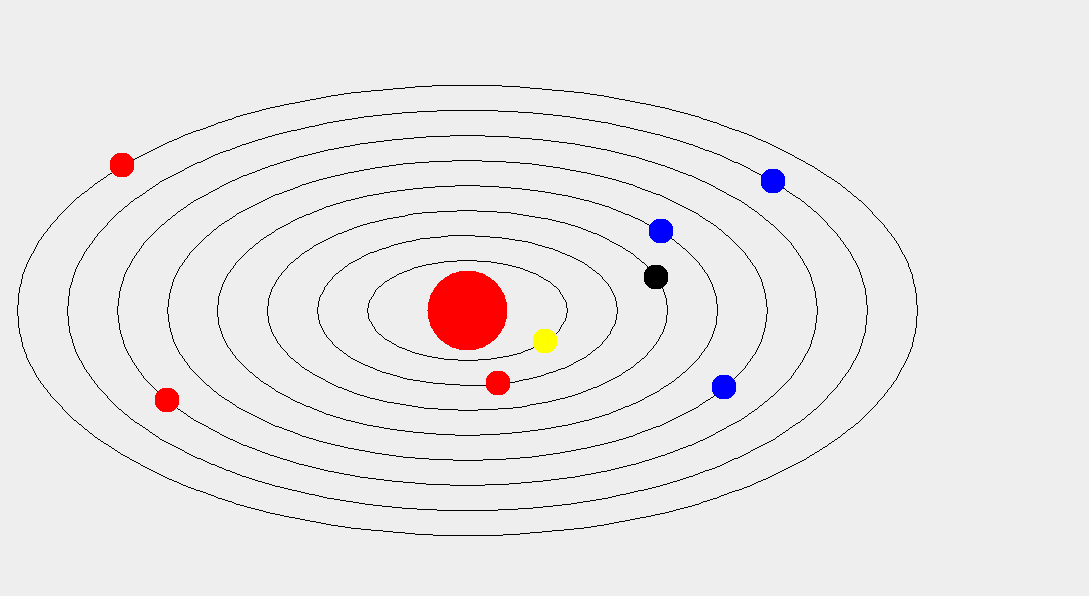
此时在绘制椭圆轨道时，只需将drawOval的参数x设置为centerx减去longAxis/2，参数width设置为longAxis；将参数y设置为center.y减去shortAxis/2，参数height设置为shortAxis。

更新轨道物体坐标的方法中，设置新的位置：

Point2D.Double d = **new** Point2D.Double();

d.setLocation(xCenter + longAxis \* Math.*cos*(tick \* ***DEG\_TO\_RAD***), yCenter + shortAxis \* Math.*sin*(tick \* ***DEG\_TO\_RAD***));

行星运动如下（依然没能截取动态图）：



### AtomStructure

当中心的原子核需要表示为质子和中子的结合，我们主要需要修改Nucleus类，因为我们并不需要质子和中子的对象，也不需要它们的集合，因此在Nucleus类的Fields中加入p和n分别表示质子和中子的数量即可。

对于txt文件，我们将第一行修改为Element ::= Rb,37,48

读文件的时候，构造的第一个正则表达式修改为

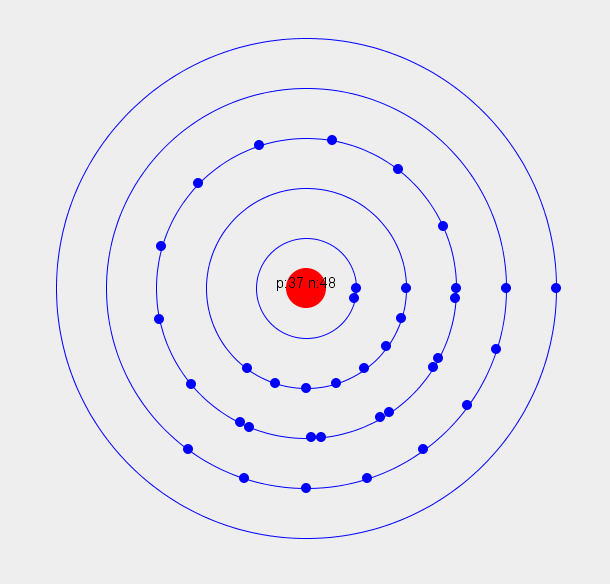
"(Element::=)(\\w+),(\\d+),(\\d+)"

若mentioned.find()为true，则mentioned.group(1)表示原子名称，mentioned.group(2)表示原子核质子数量的字符串，mentioned.group(3)表示原子核中子数目的字符串。

最后，我们在GUI可视化展示原子结构模型前，需要在CircularOrbitHelper2中获取原子核的质子数和中子数，然后绘制文本，将其显示在原子核上方即可。

这个变化主要修改了Nucleus类，在其中添加了新的域，修改了读文件，代码修改量较小。

新的GUI可视化模拟如下图所示：



### SocialNetworkCircle

**1.社交关系方向性：**由于我们为社交关系的方向性和亲密度写的EE\_IntimacyMap和LE\_IntimacyMap，在这里社交关系的方向性自动满足。

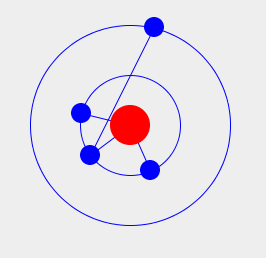
**2.忽略轨道物体到中心物体的关系：**这个实现起来也比较容易，当读文件SocialTie部分，CentralUser如果是用户2，则不执行任何操作，即不添加映射关系即可。

**3.忽略外层轨道到内层轨道的映射关系**。事实上，对于社交关系网络，在社交映射全部添加，社交图构建完毕之前，我们并不知道每个物体所处的轨道。因此，我们的做法是，先根据对象和映射关系，调用BuildRelation类中的方法第一次构建轨道。然后，对于已知的轨道，从内到外两层循环，如果存在外层轨道物体对内层轨道物体的映射，将其删除。最后我们先将轨道和轨道物体全部删除，再调用一次buildRelation方法，根据对象和新的映射关系重新构建轨道系统即可。

**重要的代码部分如下：**



**新的GUI可视化模拟如下**（此时已不存在FrankLee到DavidChen的SocialTie）：

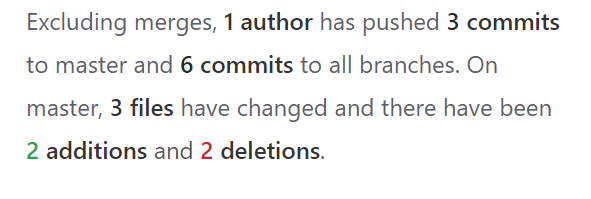


## Git仓库结构

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。



GitHub上项目仓库的Insight页面：



# 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 |
| 2019.4.10 | 14:00-18:00 | 阅读实验 | 未完成，实验刚开始理解较为困难 |
| 2019.4.11 | 18:30-23:00 | 通读实验要求 | 按计划完成 |
| 2019.4.12 | 15:30-23:00 | 完成CircularOrbit接口 | 比预计晚2小时 |
| 2019.4.15 | 13:00-18:00 | 完成Track类 | 比预计晚1小时 |
| 2019.4.16 | 19:00-23:00 | 完成轨道物体的设计 | 按计划完成 |
| 2019.4.18 | 14:00-17:30 | 完成中心物体的设计 | 按计划完成 |
| 2019.4.19 | 18:30-22:30 | 编写relation中的映射图 | 按计划完成 |
| 2019.4.20 | 14:00-17:00 | 设计具体的原子轨道及读文件 | 提前半小时完成 |
| 2019.4.21 | 17:30-23:00 | 设计具体的行星轨道及读文件 | 按计划完成 |
| 2019.4.22 | 13:30-15:30 | 设计SocialNetWork及读文件 | 未完成 |
| 2019.4.23 | 18:30-23:00 | 完成前一日内容，开始设计APIS | 按计划完成 |
| 2019.4.24 | 10:00-17:30 | 写完Difference | 比预计晚2小时 |
| 2019.4.25 | 18:30-23:00 | 写完APIS | 比预计晚1小时 |
| 2019.4.26 | 18:30-23:00 | AtomStructure的具体应用 | 未完成 |
| 2019.4.27 | 18:30-23:00 | StellarSystem的具体应用 | 按计划完成 |
| 2019.4.28 | 18:30-23:00 | SocialNetWork的具体应用 | 按计划完成 |
| 2019.5.1 | 15:30-23:00 | GUI可视化模拟 | 比预计晚2小时 |
| 2019.5.2 | 13:00-18:00 | 模拟行星运动，画动图 | 比预计晚1小时 |
| 2019.5.2 | 19:00-23:00 | 写测试 | 按计划完成 |
| 2019.5.3 | 14:00-17:30 | 完成312change | 按计划完成 |
| 2019.5.4 | 18:30-22:30 | 写报告的前半部分 | 按计划完成 |
| 2019.5.5 | 14:00-17:00 | 写报告的后半部分 | 提前半小时完成 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| 刚拿到实验，感到无从下手，难以理解题意 | 反复阅读实验指导书和文件注释中的提示，冷静下来考虑解题思路 |
| 对泛型理解不足，无法清楚认识到使用泛型的好处 | 翻阅Java编程思想，对于泛型部分的介绍仔细阅读，理解其中的实例 |
| 对设计模式的实现理解不足 | 在老师课堂讲解的基础上，去MIT网站上找到相关的文献资料进行阅读。 |
| 实验各个ADT的设计思路不明确 | 停下了编写代码的工作，仔细设计ADT |
| MyOrbitScenes运行时bug较多 | 设置断点，单步执行debug |
| 遍历集合的同时删除元素，发生错误 | 使用迭代器 |
| GUI可视化实现的时候遇到很多困难 | 仔细阅读书籍和查阅资料，学习了java的Graphics图形绘制 |
| 模拟行星运动难度大 | 在网上学习他人设计过的java动图绘制技术 |
| 描述映射关系复杂 | 定义描述relation的EE\_IntimacyMap和LE\_IntimacyMap |

# 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

经验：加深了自己对于泛型的理解和认识，提高了代码编写、ADT设计的能力，在找不出代码bug时，要学会设置断点进行debug，编写test测试文件时，有些方法的测试也能覆盖到其他的方法，避免重复测试增加工作量。

教训：在编写代码之前，最重要的是理解题意，这样可以避免走很多弯路。设计ADT时，多使用一些高级的设计模式，可以增加程序的可复用性和可维护性。

## 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？

答：我感觉，对于面向对象的编程，类是它的主要特点，程序执行过程中，先由主函数进入，定义一些类，根据需要，执行[类的成员函数](https://www.baidu.com/s?wd=%E7%B1%BB%E7%9A%84%E6%88%90%E5%91%98%E5%87%BD%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)，过程的概念被淡化了。当程序拥有良好的可复用性时，我们可以减少冗余的代码量，很多相似的实现通过复用 可以轻易解决。

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？

答：使编写的代码更加安全和可读性更强。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？

答：能体会到。开发给别人使用的API不是一件容易的事情，我们需要考虑很多的影响因素，不断完善自己的API。当然这件事也充满了乐趣，通过这个，我们不仅提升了自己，也提供了他人遍历。

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？

答：我认为，软件设计模式只是一个引导，在实际的软件开发中，必须根据具体的需求来选择：对于简单的程序，可能写一个简单的算法要比引入某种设计模式更加容易；但是对于大型项目开发或者框架设计，用设计模式来组织代码显然更好。

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？

答：语法驱动编程时候，程序设计的灵活性、逻辑性和功能性都得到加强，我们可以迅速地用极简单的方式达到字符串的复杂控制，给软件的构造提供了很大的便捷。

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？

答：ADT设计的难度主要在于考虑多个ADT之间的联系，并且从可复用性和可维护性的角度进行不断地完善优化是个漫长的过程。我们需要沉下心来，做出多种方案与尝试，最终即可克服困难。

1. 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？

答：时间问题，还未参考Lab4和Lab5的实验手册。

1. 关于本实验的工作量、难度、deadline。

答：工作量大，难度适中，deadline较合适。

1. 到目前为止你对《软件构造》课程的评价。

答：这是我非常喜欢的一门课，课程教会了我很多的知识，带我走进了软件构造的新世界。但是由于课程进度快，且ppt都是英文，因此课程难度较大，整个课程的设计应该还有可以完善的地方。