

**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 |  |
| 学号 |  |
| 班号 |  |
| 电子邮件 |  |
| 手机号码 |  |

**目录**

[1 实验目标概述 1](#_Toc7961049)

[2 实验环境配置 1](#_Toc7961050)

[3 实验过程 1](#_Toc7961051)

[3.1 待开发的三个应用场景 1](#_Toc7961052)

[3.2 基于语法的图数据输入 2](#_Toc7961053)

[3.3 面向复用的设计：CircularOrbit<L,E> 2](#_Toc7961054)

[3.4 面向复用的设计：Track 2](#_Toc7961055)

[3.5 面向复用的设计：L 2](#_Toc7961056)

[3.6 面向复用的设计：PhysicalObject 2](#_Toc7961057)

[3.7 可复用API设计 3](#_Toc7961058)

[3.8 图的可视化：第三方API的复用 3](#_Toc7961059)

[3.9 设计模式应用 6](#_Toc7961060)

[3.9.1 factory method 6](#_Toc7961061)

[3.9.2 abstract factory 6](#_Toc7961062)

[3.9.3 Iterator 7](#_Toc7961063)

[3.9.4 facade 7](#_Toc7961064)

[3.9.5 decorator 7](#_Toc7961065)

[3.9.6 state 7](#_Toc7961066)

[3.9.7 memento 7](#_Toc7961067)

[3.10 应用设计与开发 8](#_Toc7961068)

[3.10.1 StellarSystem 8](#_Toc7961069)

[3.10.2 AtomStructure 9](#_Toc7961070)

[3.10.3 SocialNetworkCircle 9](#_Toc7961071)

[3.11 应对应用面临的新变化 10](#_Toc7961072)

[3.11.1 StellarSystem 10](#_Toc7961073)

[3.11.2 AtomStructure 10](#_Toc7961074)

[3.11.3 SocialNetworkCircle 10](#_Toc7961075)

[3.12 Git仓库结构 10](#_Toc7961076)

[4 实验进度记录 10](#_Toc7961077)

[5 实验过程中遇到的困难与解决途径 11](#_Toc7961078)

[6 实验过程中收获的经验、教训、感想 11](#_Toc7961079)

[6.1 实验过程中收获的经验和教训 11](#_Toc7961080)

[6.2 针对以下方面的感受 11](#_Toc7961081)

# 实验目标概述

根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）。

# 实验环境配置

* JGraphX

下载地址: <https://github.com/jgraph/jgraphx/raw/master/lib/jgraphx.jar>

* repo

<https://github.com/ComputerScienceHIT/Lab3-1170300431.git>

# 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）。

* StellarSystem
* AtomicStructure
* SocialNetworkCircle

共性, 都是轨道系统, 都能完成一些轨道系统上的共有操作;

差异:

* StellarSystem 要求可以模拟行星运动;
* AtomicSystem各个物体之间没有区别;
* SocialNetworkCircle物体之间有关系, 并且物体的轨道是由关系决定的.

## 基于语法的图数据输入

观察各个样本文件, 利用正则表达式和字符串分割获得信息; 在LoadFromFile中创建PhysicalObject对象或建立对象之间的关系.

## 面向复用的设计：CircularOrbit<L,E>

设计CircularOrbit接口, 实现轨道增删, 物体增删, 移动物体, 查询物体等功能. 抽象类ConcreteCircularOrbit继承了CircularOrbit接口. 针对每个具体应用, 分别有StellarSystem, AtomStructure, SocialNetworkCircle 继承ConcreteCircularOrbit.

## 面向复用的设计：Track

Track表示一个轨道. 它应当包含描述一条轨道的所有参数, 并提供获取这些参数的接口. Track应当被具有”轨道”属性的对象持有.

因为Track是内部类, 所以在接口参数和返回值中, 需要表示轨道的地方应当采取另外的表示方式. 而显然, Track实际上是一个表示轨道参数的数组. 故这些地方用一个浮点数组来代替Track, 而Track内部增加与浮点数组互相转换的显式接口(Track::new, Track::getRect, Track::getRect\_alt).

## 面向复用的设计：L

L是CircularOrbit的第一个参数, 表示中心物体的类型. 使用泛型使得CircularOrbit可复用, 可以不必修改地用于多种中心物体. 此处使L继承PhysicalObject

## 面向复用的设计：PhysicalObject

PhysicalObject表示一个实际的物体. 可能是中心物体或是轨道物体. PhysicalObject为抽象类, 它拥有的属性是. 他们的类型分别是String, Track, double. 可以看到PhysicalObject持有Track的对象. 这表示PhysicalObject处于某一轨道上. PhysicalObject提供它们的封装.

面向不同的应用, 应当有不同的类继承PhysicalObject表示某一特定物体, 如行星, 电子, 用户; 恒星, 原子核, 中心用户.

## 可复用API设计

* getObjectDistributionEntropy

以各个物体的轨道半径为统计量, 得到这些统计量的概率为, 熵为.

* getLogicalDistance

使用了Lab2的Graph. 借由Lab2的getDistance实现.

* getPhysicalDistance

由于物体本身带有表示位置的属性, 使用余弦定理即可.

* getDifference
  + 对于两个给定的轨道, 分别调用getTrack得到轨道的差异, 包括轨道数目差异, 和轨道参数差异.
  + 对于两条参数相同的轨道, 分别调用getObjectsOnTrack得到它们所包含的物体, 若某一物体只在一条轨道上出现, 那么这个物体上存在差异.
  + 将得到的差异输出到控制台.

## 图的可视化：第三方API的复用

使用JGraphX设计CircularOrbit, 主要使用的接口有addCell, moveCell, addEdge;



图 1 AtomicStructure.

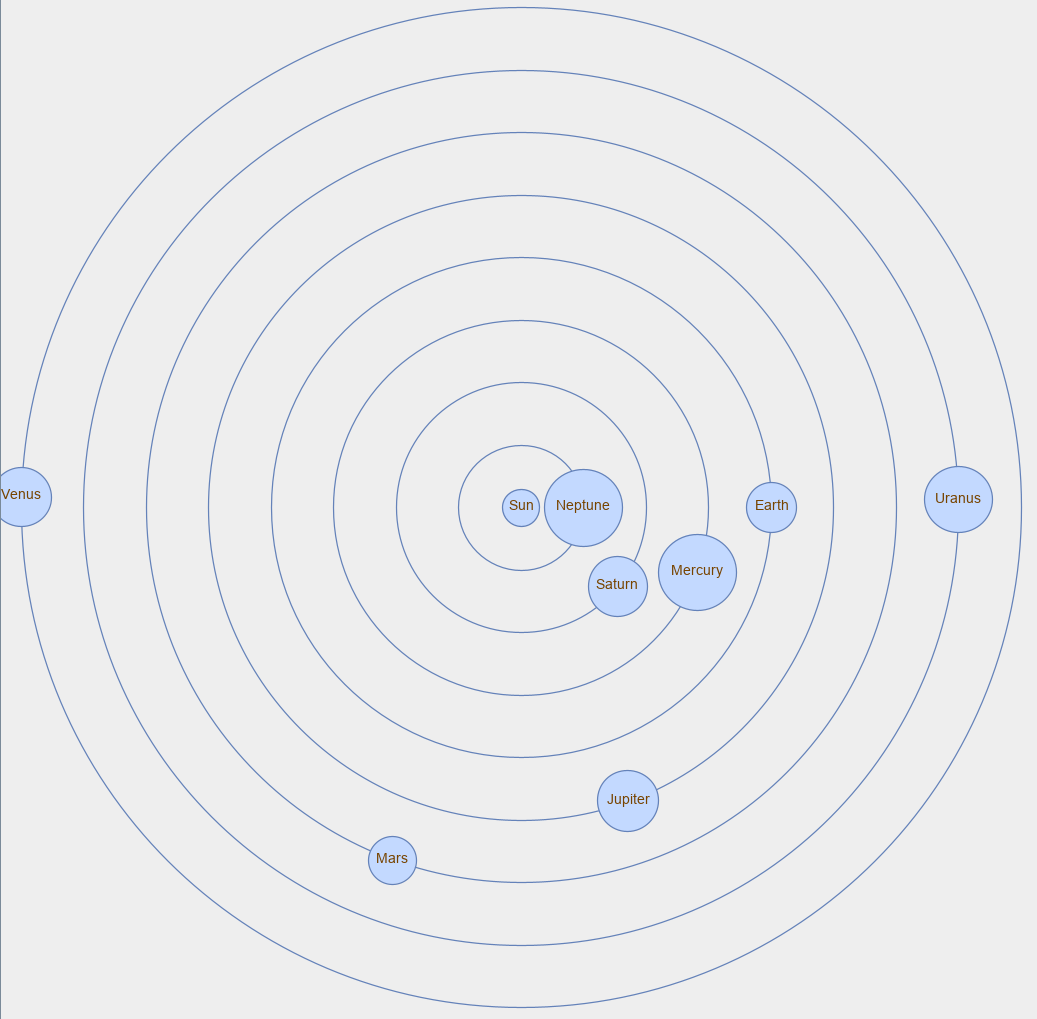


图 2 StellarSystem

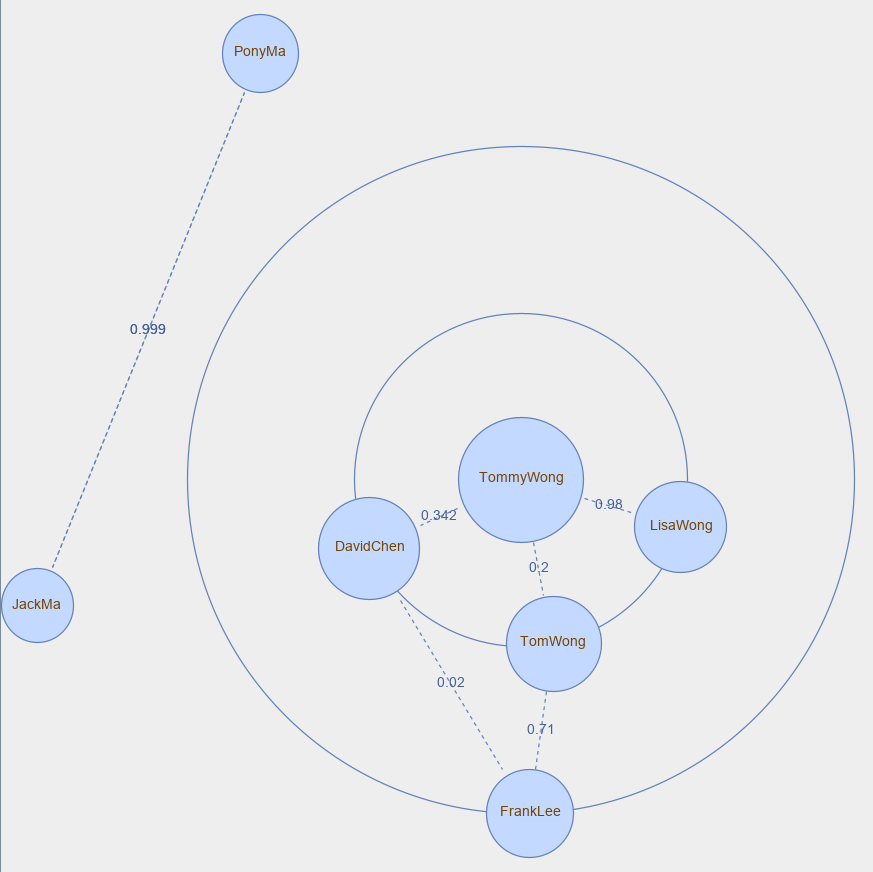


图 3 SocialNetworkCircle[[1]](#footnote-1).

## 设计模式应用

### factory method

PhysicalObject的创建可以使用静态工厂方法

1. **public** **static** PhysicalObject produce(@NotNull String[] args);

其中args是描述参数的字符串数组.

### abstract factory

CircularOrbit的实例化应当使用CircularOrbitFactory工厂接口. 它由DefaultCircularOrbitFactory继承, 有两个方法: Create, CreateAndLoad. 分别是创建一个空系统, 和从文件中载入系统.

### Iterator

CircularOrbit继承Iterable接口, 支持轨道从内向外 角度从小到大遍历.

### facade

所有 API 放置在 helper 类 CircularOrbitAPIs 当中.

### decorator

实现PlanetarySystem继承Planet, 并持有Planet的数组表示卫星.

1. **final** **class** PlanetarySystem **extends** Planet{
2. **private** Set<Planet> satellites;
4. PlanetarySystem(@NotNull Planet center);
6. **public** **boolean** addSatellite(@NotNull Planet satellite);
8. @NotNull
9. **public** Planet[] satellites();
11. @Nullable
12. **public** Planet query(@NotNull String name);
13. }

### state

电子Electron持有ElectronState对象表示电子是否跃迁. 基态电子只能向高轨道跃迁, 激发态电子只能向低轨道跃迁. ElectronState是一个接口, Ground类和Excited类继承ElectronState, 分别表示基态和激发态.

### memento

AtomStructure应用了memento设计方法. 它持有一个caretaker对象用于管理memento; 当跃迁transit被调用时, 一个memento被创建, 它持有当前轨道上电子的拷贝. 若跃迁操作出现异常, 则recover方法被调用, 从caretaker中取出对应的memento, 并恢复它. 于是AtomStructure便回复到了调用前的状态, 保证了异常安全.

## 应用设计与开发

### StellarSystem

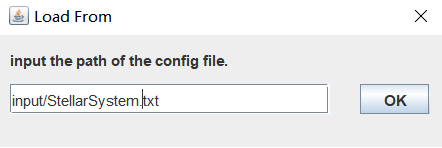


图 4 从文件读入

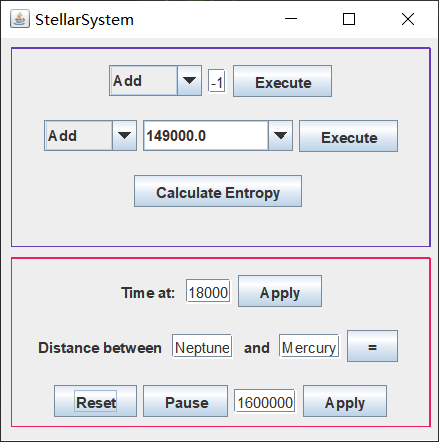


图 5 StellarSystem的控制界面

蓝色框是通用部分, 包括(下同, 略)

* 第一行, 增删轨道
* 第二行, 增删物体
* 第三行, 计算熵值.

红色框, StellarSystem特殊功能

* 第一行, 计算特定时间各行星位置
* 第二行, 计算两行星间物理距离
* 第三行, 重置, 暂停/继续, 行星运行速度.

### AtomStructure

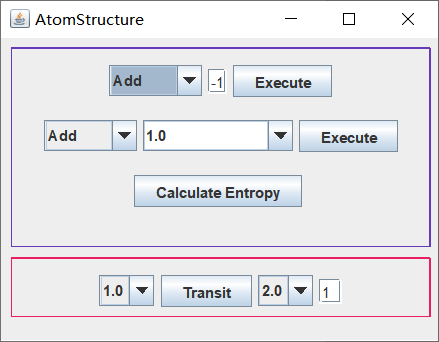


图 6 AtomicStructure控制界面

红色框: 电子跃迁.

### SocialNetworkCircle

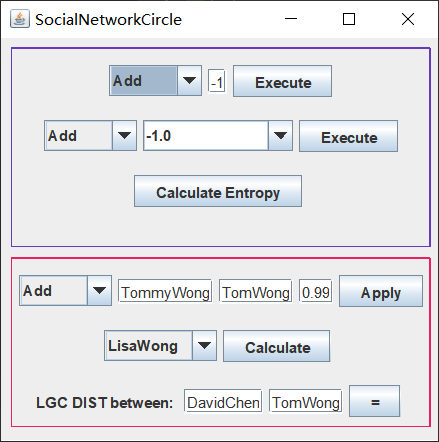


图 7 SocialNetworkCircle的控制界面

红色框

* 第一行, 增加或删除一条关系
* 第二行, 计算第一轨道某个人的拓展度(权值累乘大于0.02的人数)
* 第三行, 计算某两个人间的逻辑距离.

## 应对应用面临的新变化

### StellarSystem

* 更改Track的构造函数, 使其接受长度为二的浮点数组(长半轴 短半轴)
* 更改Planet的构造函数, 从接受浮点数(半径)改为接受浮点数组.
* 更正CircularOrbitHelper的绘图方式, 从用单一浮点数(半径)绘制圆变为用浮点数组绘制椭圆.

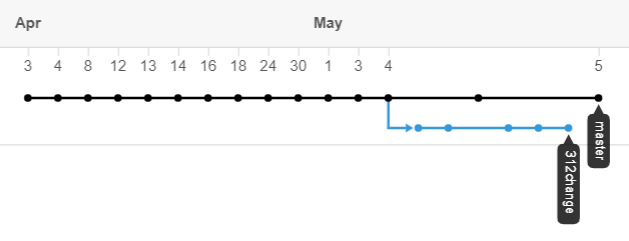
### AtomStructure

在Kernel类中增加字段protron和neutron, 并提供getter和setter.

### SocialNetworkCircle

更改setRelation方法的实现, 使其增加单条边(a->b)而不是两条(a->b & b->a); SocialNetworkCircle的私有方法updateR中, 遍历所有边, 若有从高轨道指向低轨道的边, 则删除它.

## Git仓库结构



# 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 |
| 4-3 |  | 初始化仓库, 设计ADT | 一部分 |
| 4-12 |  | 实现StellarSystem的ADT部分 | 完成 |
| 4-17 |  | CircularOrbitHelper | 一部分 |
| 4-18 |  | 模拟行星运动 | 完成 |
| 4-24 |  | AtomStructure的ADT和GUI显示 | 完成 |
| 5-2 | 全天 | 所有GUI, 以及各个具体实现的控制面板 | 完成 |
| 5-3 | 全天 | 大规模底层变动 | 完成 |
| 5-4 | 全天 | 测试和注释 | 完成 |
| 5-5 |  | bug fix | 可能吧 |

# 实验过程中遇到的困难与解决途径

|  |  |
| --- | --- |
| 遇到的难点 | 解决途径 |
| Track是immutable类 | 不把Track作为容器, 而只作为一个属性给PhysicalObject持有. |
| PhysicalObject可变后, 计算Hash和判断Equal出现困难 | 在Position和Radius字段之外, 设置两个新的不可变的字段pos\_init和R\_init, 在hash, equals, comparator中使用pos\_init和R\_init来代替可变的pos和R. |

# 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

## 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？ 是
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？

确保在用户操作之后, ADT的数据结构不被破坏.

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？

开发API是实现复用的必然选择.

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？

在复用等方面的确有一定的优势, 但应当慎用.

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？

是合理且高效的.

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？

如何设计才能保证它的作用, 并且不与其他已有的发生重叠; 如何尽可能地封装, 而减轻内部实现对外部的影响.

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？
2. 关于本实验的工作量、难度、deadline。

听说很多人放弃了假期...

1. 到目前为止你对《软件构造》课程的评价。

1. 注, SocialNetworkCircle中不在轨道上的点表示该点不与中心点连通, 即它不在CircularOrbit内部. [↑](#footnote-ref-1)