

**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 | 肖行文 |
| 学号 | 1170300316 |
| 班号 | 1703003 |
| 电子邮件 | xiaoxingwen@stu.hit.edu.cn |
| 手机号码 | 13266573776 |

**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

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:

1. 子类型、泛型、多态、重写、重载
2. 继承、代理、组合
3. 常见的 OO 设计模式
4. 语法驱动的编程、正则表达式
5. 基于状态的编程
6. API 设计、API 复用

# 实验环境配置

从给定仓库地址下载代码，创建项目，并绑定这次Lab的仓库地址。

Lab3仓库地址：

<https://github.com/ComputerScienceHIT/Lab3-1170300316>

# 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

这三个应用场景都是同一类轨道系统，现列举轨道系统中每个部分的相似处。

中心：不可移动。

轨道：与距离有关，不同物体可处于轨道上。

轨道上的物体：与中心有一定关系，且相互之间也存在关系。

但它们也存在区别，比如StellarSystem是处于变化中的，而AtomStructure 和SocialNetwork是不移动的。SocialNetwork中的距离是抽象的距离，而StellarSystem中距离是实际的物理距离等，在应用同一套ADT的时候要进行转换。

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。

## 基于语法的图数据输入

用正则表达式对读入的数据进行适配。

1. StellarSystem

用<(.\*?),(.\*?),(.\*?)>对括号中的内容适配，第一个即为Stellar的名称，第二个即为半径，第三个即为质量。同理，<(.\*?),(.\*?),(.\*?),(.\*?),(.\*?),(.\*?),(.\*?),(.\*?)>对Planet进行适配。

1. AtomStructure

用::= (.\*?)$适配原子核和电子信息，本题输入不多。

1. SocialNetworkCircle

用<(.\*?),(.\*?),(.\*?)>适配每个人的信息。

## 面向复用的设计：CircularOrbit<L,E>

CircularOrbit是直接与GUI对接的一个类，也意味着基本上所有的用户操作和显示是针对CircularOrbit进行的。其需要有最基本的添加轨道、添加轨道物体、添加中心点物体的功能，然后将这些功能传入Track、PhysicalObject等类中。

使用工厂模式创建不同类型的Orbit，因为不同的Orbit有微小的差别，如StellarSystem中的每个轨道只有一个物体，而且是输入完成后对轨道进行排序。但是SocialCircle中每个轨道有多个物体，且是输入的时候自带层级，这就要求了接口上细微的差别。

CircularOrbit到每个场景的Orbit之间存在ConcreteCircularOrbit这座桥，用于实现三个具体场景类的共有方法，比如返回轨道数、命名轨道等。

## 面向复用的设计：Track

三个场景所用的Track功能类似，它是容纳PhysicalObject的一个容器。GetPhysicalObject、SetPhysicalObject等方法实现对其上的PhysicalObject的一系列操作，同时构造函数要用重载，方便随时补上空轨道、设置带层数属性的轨道等。

为了面向GUI的需求，Track中应有Init()和Update()进行绘图，除了绘制轨道本身，还要调用PhysicalObject的绘图。应该注意的是，在StellarSystem中，因为轨道的实际半径大小差异过大，在Track中还需要接受或者生成一个假的轨道半径，方便画图。

## 面向复用的设计：L

中心物体所承载的任务不多，主要存储中心物体的属性，并在图中心保持不动。

## 面向复用的设计：PhysicalObject

除了保存每个PhysicalObject的属性外，还应该有Position这个ADT。Position是极坐标，方便得对应物体的真实位置。PhysicalObject应该有画出自身的功能，画出自身应该是基于位置进行画图，这样如果移动PhysicalObject只需要改变它的属性Position就可以了。

## 可复用API设计

可复用的API实现了文档中要求的功能。

1. double getObjectDistributionEntropy(CircularOrbit c)

熵的计算需要调用轨道上的物体数量，用CircularOrbit提供的方法调用Track中的接口。

1. int getLogicalDistance (CircularOrbit c, E e1, E e2)

P3中实现了类似的功能，但是P3中实现的是有确定中心人物的逻辑最短路，意味着是单源最短路，改进了迪杰斯特拉实现。但是此处要求任意PhysicalObject的距离，便用Floyd算法求，将人物关系网视为二维矩阵。

1. double getPhysicalDistance (CircularOrbit c, E e1, E e2)

如上文中提到，每个PhysicalObject中都有属性Position(是极坐标)，使得任意两个PhysicalObject之间的距离可以由余弦定理求得。

1. Difference getDifference (CircularOrbit c1, CircularOrbit c2)

新建一个Difference类将两个Orbit之间的数值差异保存为数组，调用Orbit中的关于Track的方法可以求得各种差异。

## 图的可视化：第三方API的复用

1. CircularOrbitHelper

这个类的主要作用是展示一个启动窗口，然后启动某个场景的可视化程序。



1. StellarSystemVisual/AtomStructureVisual/SocialNetworkCircle

图的可视化是通过JPanel的组件实现的。如背景是一个铺满窗口的JLabel，包括PhysicalObject和轨道也是JLabel，因为JLabel可以是一张图片，而且可以方便移动位置。相比之下另一种绘图方式——使用paint()——移动的本质其实是重绘，动态的界面中效率低。

动态的GUI是由线程实现的，线程中对PhysicalObject的Position进行修改就实现了移动。

制作GUI的难度实际上很大，因为操作和帧需要同步，在这一帧执行的操作在一系列类中传递后也要保证在这一帧得到结果。

## 设计模式应用

1. 工厂方法

在调用每个场景具体的Orbit的时候，在CircularOrbit中使用工厂方法。

1. 迭代器

SocialNetworkCircle中由于要求逻辑最短路，需要用迪杰斯特拉算法，而Friend的对象是被存在集合中的，且集合在发生变化，故应用迭代器。

1. Bridge

ConcreteCircularOrbit是CircularOrbit和具体场景Orbit之间的一座桥，由ConcreteCircularOrbit提供共有方法，具体类对CircularOrbit中剩余接口进行补全。

1. 委托

为了使Thread作为线程控制GUI，要在Thread中生成一个GUI对象，同时也有在GUI中传入作为参数的Thread。这使得GUI和Thread可以互相影响、控制。

1. Observer

部分Observer需要对GUI中按钮情况进行观察，随时调用Orbit中方法修改或者打印结果。

## 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

功能1和3是可视化中实现的，如上文中提到，修改ADT中的Position可以实现有关位置的操作。为了计算行星间距离，因为Position存储极坐标，也是调用Position用余弦定理计算距离。

### AtomStructure

使电子跃迁的方式是删除起始轨道上的一个PhysicalObject，再于目标轨道上添加一个PhysicalObject，然后重新绘图。

### SocialNetworkCircle

为了实现功能3添加一个关系，将用户希望添加的关系以相同格式写入要读入的文件，如SocialNetworkCircle.txt，然后重新读入文件。功能4计算两个用户间距离通过迪杰斯特拉算法实现，实际上在Lab2中的Graph<L>可以实现这个功能。功能1是可视化的必备功能。为计算功能2信息扩散度，对指定人物遍历与其有关系的好友，找出高于某个亲密度的好友，继续计算下一层，对这样找出的好友数累加。

## 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

如上文所说，Track的图形化是通过JLabel实现的，如果想把Track变成椭圆的，实际上只用拉伸JLabel大小就可以。JLabel上的图片可以随着JLabel大小拉伸。

### AtomStructure

将中心视作半径为0的Track即可，因为Track上可以包含多个PhysicalObject。

### SocialNetworkCircle

在ADT Relation中，Relation其实是映射集合。原来如果要将两个Object a与b连接，需要有a到b的映射，也要有b到a的映射，现在只用建立一个单向的映射。



## Git仓库结构

对312Change单独建了一个分支，与master在测试和3.12要求部分不同



# 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 |
| 4.17 | 18:30-21:00 | 设计类之间关系，画UML图 | 完成 |
| 4.18 | 18:30-21:00 | 根据UML，写出ADT | 完成 |
| 4.25 | 21:00-23:00 | 完成P1的非GUI部分 | 完成 |
| 4.26 | 20:00-24:00 | 设计P1的GUI | 难度大，未完成 |
| 4.27 | 20:00-23:00 | 学习线程 | 完成 |
| 4.28 | 21:00-23:00 | 初步完成粗糙的P1的GUI | 完成，有拖影 |
| 4.29 | 21:00-23:00 | 完成P2 | 完成 |
| 5.1 | 18:00-22:00 | 完成P3 | 未完成，最短路设计出错 |
| 5.2 | 18:00-23:00 | 完成P3 | 完成 |
| 5.3 | 19:00-24:00 | 完成P1的GUI | 解决拖影问题 |
| 5.4 | 20:00-24:00 | 修改一些遗留bug | 完成 |
| 5.5 | 14:00-22:00 | 完成3.12 | 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 |
| 动态GUI | 上网查询到可以用线程Thread做出GUI动画，最终用Thread实现了行星运动模型。 |
| 最开始设计体系复杂 | 优先画出UML图，理清各类之间的关系，最后大部分ADT设计成功。 |
| 动态GUI由于运算量大无法刷新 | 用panel.removeAll()和panel.updateUI()重新再绘制UI，消除了动画留下的拖影。 |

# 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

1. 主要经验是制作GUI方面的，GUI方面要注意的问题特别多，如同步、消除拖影等，不过最终都找到了解决办法。
2. 找资料时要注意甄别，很多Java网络资料不靠谱。

## 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？

ADT使得工作量降低，复用不仅减少了代码量，还减少了可能产生的bug，对ADT接口的调用使程序更清晰。

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？

保证安全性，也防止程序被程序其它部分修改，更方便了开发者（包括自己）复用。愿意。

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？

难度很大，自己在用的时候会体会到乐趣，因为不用重写代码了。

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？

设计模式是很多人总结出来的高效的设计方法，可以提供程序效率，减少思考量。

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？

有点复杂和麻烦，而且这类编程的体验是可以被提高的。比如有很多的正则表达式生成器可以方便编程。

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？
2. 需要有清晰的项目规划，因为ADT在之后的项目要多次使用，如果ADT的接口或者层级关系有问题，到后来项目复杂的时候改起来非常麻烦。
3. ADT基本上需要绝对正确。在ADT投入使用前要详细测试ADT的正确性。很可能一个场景是正确的但另一个场景会触发ADT的小bug。
4. ADT要有清晰的说明，方法名要清晰。不仅仅是别人可能读不懂，类多了之后自己可能也读不懂。
5. 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？

参考了部分，在实验后期Lab4和Lab5上线后对安全性和结构进行了修改。

1. 关于本实验的工作量、难度、deadline。

工作量适当，难度除了GUI部分适当，GUI部分难度很大。

1. 到目前为止你对《软件构造》课程的评价。

软件构造非常倾向于构造，意味着要设计软件，不能凭感觉写。有了ADT、设计模式的知识后，可以写更长更复杂的程序了。