哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

2019 年春季学期

计算机学院《软件构造》课程

Lab 4 实验报告

姓名	沈子鸣
学号	1170301007
班号	1703010
电子邮件	2508754153@qq.com
手机号码	18800421860

目录

1 实验目标概述

本次实验重点训练学生面向健壮性和正确性的编程技能，利用错误和异常处理、断言与防御式编程技术、日志/断点等调试技术、黑盒测试编程技术，使程序可在不同的健壮性/正确性需求下能恰当的处理各种例外与错误情况，在出错后可优雅的退出或继续执行，发现错误之后可有效的定位错误并做出修改。

实验针对 Lab 3 中写好的 ADT 代码和基于该 ADT 的三个应用的代码，使用以下技术进行改造，提高其健壮性和正确性：

- 错误处理
- 异常处理
- Assertion 和防御式编程
- 日志
- 调试技术
- 黑盒测试及代码覆盖度

2 实验环境配置

本次实验环境配置与之前相同。

硬件环境：Intel Core i5-7200U，64 位; 4G;

软件环境：Windows 10 家庭中文版

IDE：Eclipse Java 2018-12

关于 Java：JDK 8, JRE 8

关于 Git：Git 2.20.1

```
C:\Users\Shen>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

```
C:\Users\Shen>git --version
git version 2.20.1.windows.1
```

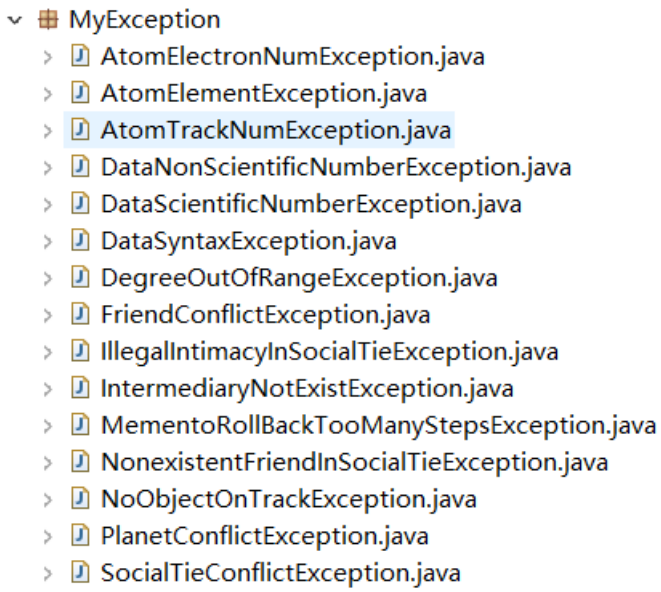
GitHub 仓库的 URL 地址：

https://github.com/ComputerScienceHIT/Lab4-1170301007

3 实验过程

3.1 Error and Exception Handling

3.1.1 自定义异常类总览



3.1.2 自定义异常类说明及使用简述

1. AtomElectronNumException
   该异常主要关于原子结构中的电子数目。数据文件中没有指明某些轨道上的电子数，或是轨道数与电子数的信息不匹配相等。在读文件时，涉及此异常的主要是判断 numberOfElectronString 变量的 length 是否符合要求。

```
numberOfElectronString = matcher3.group(1).split("/|;");
if (numberOfElectronString.length % 2 == 1) {
    throw new AtomElectronNumException(
            "Each track order number should match a number of electron! Exception line: "
                    + readLineCounter + ".");
}
if (numberOfElectronString.length != (numberOfTrack * 2)) {
    throw new AtomElectronNumException(
            "The number of electron information doesn't match the number of track! Exception line: "
                    + readLineCounter + ".");
}
```

2. AtomElementException
   该异常主要关于原子结构中的元素数据不合法。数据文件中元素数据并非只有 1 或 2 个英文字符，或第一个英文字符并非大写，或第二个英文字符，如果有的话，并非小写。在读文件时，涉及此异常主要是判断匹配到的元素字串是否符合要求。

```
if (matcher1.group(1).length() >= 3 || matcher1.group(1).length() <= 0) {
    throw new AtomElementException(
            "The element of this atom contains 0 or more than 2 characters. Exception line: "
                    + readLineCounter + ".");
} else if (!Character.isUpperCase(matcher1.group(1).charAt(0))) {
    throw new AtomElementException("The first character is not a upper case character! Exception line:
            + readLineCounter + ".");
}
if (matcher1.group(1).length() >= 2 && !Character.isLowerCase(matcher1.group(1).charAt(1))) {
    throw new AtomElementException(
            "The second character is not a lower case character! Exception line: " + readLineCounter
                    + ".");
}
```

3. AtomTrackNumException
   该异常主要关于原子结构总的轨道数目不合法。数据文件中轨道数目数据不是正整数。

```
if (Integer.parseInt(matcher2.group(1)) <= 0) {
    throw new AtomTrackNumException(
            "Number of track is not a positive integer! Exception line: " + readLineCounter + ".");
}
```

4. DataScientificNumberException
   该异常主要关于科学记数法的表示不合法，本实验中主要是在行星轨道中才会出现。科学记数法的尾数部分小于 1 或者大于等于 10，指数部分小于 4，都会触发该异常。

```
if (!(mantissa >= 1 && mantissa < 10)) {
    throw new DataScientificNumberException(
            "The mantissa is not between 1 and 10. Exception line: " + readLineCounter + ".");
}
if (exponent < 4) {
    throw new DataScientificNumberException(
            "The exponent is less than 4. Exception line: " + readLineCounter + ".");
}
```

5. DataNonScientificNumberException
   该异常主要关于非科学记数法的数字表示不合法，比如超过 10000 的数字并没有使用科学记数法表示。

```
if (d >= 10000) {
    throw new DataNonScientificNumberException(
            "A number greater than 10000 doesn't use scientific notation! The illegal number is t
                    + readLineCounter + ".");
}
```

6. DataSyntaxException
   该异常涉及的内容较多，全部是关于读取文件时，数据文件 与正则表达式不匹配的情况，这种情况不包括空行，仅考虑一些细微不符合语法的情况，所以，其判断条件应该是，该行数据中至少应该包括数据类型的声明，比如社交关系网络中的 Friend, SocialTie, CentralUser，行星轨道系统中的 Stellar 和 Planet 字串，在这种情况下，才认为是该数据中可能有细微的语法不匹配。

```
}
if (!centralUserMatch && !friendMatch && !socialTieMatch && line.contains("SocialTie")) {
    String[] syntaxTest = line.substring(line.indexOf("<") + 1, line.indexOf(">")).split(",
    int labelSyntaxCount = 0;
    for (int i = 0; i <= 1; i++) {
        if (Pattern.compile("\\s*[a-zA-Z0-9]*\\s*").matcher(syntaxTest[i]).matches()) {
            labelSyntaxCount++;
        }
    }
    if (labelSyntaxCount < 2) {
        throw new DataSyntaxException(
                "One of friend label syntax doesn't match! Exception line: " + readLineCoun
    }
    if (!Pattern.compile("\\s*0\\.\\d{1,3}\\s*|\\s*1\\.0{1,3}\\s*").matcher(syntaxTest[2]).
        throw new DataSyntaxException(
                "SocialTie intimacy syntax doesn't match! Exception line: " + readLineCount
    }
}
```

在读文件时，管理若干个 boolean 变量，以判断是否所有的情况都匹配失败，再判断该行有可能是哪种数据出现了细微的语法错误。

判断好是否可能有该异常发生后，再对该行的数据分量逐个解析，比如在判断行星数据中是哪个分量有语法错误时，考虑行星分量按顺序是名称、状态、颜色、行星半径、轨道半径、公转速度、公转方向和初始角度，有写分量的语法是一样的，比如名称、状态和颜色，这样并不能确保是哪一个分量有语法错误，只能解析出可能是哪些分量有语法错误。代码如下：

```
if (!stellarMatch && !planetMatch && line.contains("Planet")) {
    String[] syntaxTest = line.substring(line.indexOf("<") + 1, line.indexOf(">")).split(",");
    int labelSyntaxCount = 0;
    for (int i = 0; i <= 2; i++) {
        if (Pattern.compile("\\s*[a-zA-Z0-9]*\\s*").matcher(syntaxTest[i]).matches()) {
            labelSyntaxCount++;
        }
    }
    if (labelSyntaxCount < 3) {
        throw new DataSyntaxException(
                "One of planet label syntax doesn't match! It can be name, state or color. Exc
                        + readLineCounter + ".");
    }
    int numberSyntaxCount = 0;
    for (int i = 3; i <= 5; i++) {
        if (Pattern.compile(
                "\\s*[0-9]*\\s*|\\s*[0-9]*\\.[0-9]*\\s*|\\s*[0-9]\\.[0-9]*e[0-9]*\\s*|\\s*[0-9
                .matcher(syntaxTest[i]).matches()) {
            numberSyntaxCount++;
        }
    }
```

先解析是否有 3 个 label 类型数据匹配，如果不足 3 个，那么肯定就是名称、状态和颜色中的至少一个有语法错误，给出提示信息并抛出异常。解析 number 类型的数据也如此。

7. DegreeOutOfRangeException

该异常是关于初始角度溢出合法范围[0,360)，理论上来说，初始角度其实并没有一个严格的限制范围，所以当初始角度溢出合法范围时，可以考虑自动将其矫正为合法范围，模 360 取余即可。

```
originDegree = parseNumber(matcher2.group(8));
try {
    if (originDegree < 0 || originDegree >= 360) {
        throw new DegreeOutOfRangeException(
                "Original degree is less than 0 or no less than 360! Because degree is c
    }
} catch (DegreeOutOfRangeException e) {
    stellarSystemLogger.log(Level.WARNING, e.getMessage(), e);
    originDegree = originDegree % 360;
}
```

8. FriendConflictException
该异常是关于 Friend 数据信息冲突，也就是数据文件中给出了两个重名的 friend（认为两个 friend 重名即相等）。这个异常也不是很严重的异常，默认当读入一个已经存在的 friend 时，直接跳过不读即可。

```
for (PhysicalObject p : friends) {
    if (p.equals(friend)) {
        throw new FriendConflictException(
                "This friend has already existed! The latter conflicting friend isn't lo
                        + readLineCounter + ".");
    }
}
friends.add(friend);

            }
        } catch (FriendConflictException e) {
            socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
            System.out.println(e.getMessage());
        } catch (SocialTieConflictException e) {
            socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
            System.out.println(e.getMessage());
        }
    } // end while ((line = br.readLine()) != null)
```

9. IllegalIntimacyInSocialTieException
该异常是关于社交关系网络中的亲密度数据不合法。当亲密度溢出范围(0,1]时，认为亲密度不合法。

```
if (intimacy <= 0 || intimacy > 1) {
    throw new IllegalIntimacyInSocialTieException(
            "The intimacy is no greater than 0 or greater than 1! Exception line: "
                    + readLineCounter + ".");
}
```

10. IntermediaryNotExistException
该异常是关于社交关系网络应用中，在计算信息扩散度时，构造的中介人不存在。用一个 boolean 变量来维护中介人是否存在，判断该变量都确定是否引发异常。

```
}
for (PhysicalObject friend : socialNetworkCircle.getRelationBetweenCentralAndObject()[0].k
    if (friend.equals(intermediary)) {
        existIntermediary = true;
        break;
    }
}
try {
    if (!existIntermediary) {
        throw new IntermediaryNotExistException("This intermediary doesn't exist!");
    }
} catch (IntermediaryNotExistException e) {
    socialNetworkCircleLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
    break;
}
```

11. MementoRollBackTooManyStepsException
该异常是在做原子结构电子跃迁时，调用状态回溯功能，可能引发的回溯版本多于已经执行的跃迁步数，导致回溯失败。该异常仅在 Caretaker 类中触发并处理。

```
try {
    if (mementos.size() - 1 < i) {
        throw new MementoRollBackTooManyStepsException("Cannot roll so many back!");
    }
} catch (MementoRollBackTooManyStepsException e) {
    System.out.println("Roll back resetting fail! The state doesn't change.");
    return this.mementos.get(this.mementos.size() - 1);
}
return this.mementos.get(this.mementos.size() - i - 1);
```

12. NonexistentFriendInSocialTieException

该异常是在社交关系网络中，读入一个 SocialTie 但是该关系数据中包含一个不存在的 friend，从而引发异常。如果该异常发生的话，不应该终止文件读入，只要跳过该 SocialTie 的读入就可以解决了。

```
    if (!friendExisted) {
        throw new NonexistentFriendInSocialTieException("A nonexistent friend shows in a social t
    }
} catch (NonexistentFriendInSocialTieException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println("This social tie is skipped. Exception social tie: " + names[0] + "->" + r
    continue;
} catch (TwoSamePersonsInSocialTie e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println("This social tie is skipped. Exception social tie: " + names[0] + "->" + r
    continue;
}
```

13. NoObjectOnTrackException

该异常是关于在应用中使用删除某条轨道上的物体功能时，可能该条轨道上没有这个物体，或者特别地在原子结构系统中，该轨道上已经无电子。该异常仅在 deletePhysicalObjectFromTrack 方法中抛出。

```
@Override
public void deletePhysicalObjectFromTrack(E physicalObject, Track track) throws NoObjectOnTrack
    boolean exist = false;
    for (Track t : this.tracks) {
        if (t.equals(track)) {
            exist = true;
            this.objectsInTrack.get(t).remove(physicalObject);
            this.relationBetweenCentralAndObject[0].remove(physicalObject);
            this.relationBetweenCentralAndObject[1].remove(physicalObject);
        }
    }
    if (!exist) {
        throw new NoObjectOnTrackException("Delete fail! This track has no such object!");
    }
```

14. PlanetConflictException

该异常是在行星轨道系统中读取文件时，遇到了重名的行星（认为两个行星重名则相同），该异常不影响文件读取，提示用户并跳过该行星的读取即可。

```
for (PhysicalObject p : this) {
    if (p.equals(planetFactory.produce(planetName, planetState, planetColor, planetRadius,
            revolutionSpeed, revolutionDirection, originDegree))) {
        throw new PlanetConflictException(
                "This planet has already existed! Exception line: " + readLineCounter + ".
    }
}
```

15. SocialTieConflictException

该异常是在社交关系网络中读取文件时，遇到了相同的 SocialTie（认为

两个 SocialTie 连接的两个人相同时，则两个 SocialTie 相同），或者某一个 SocialTie 连接的两个人就是相同的，这两种情况都被认为是 SocialTie 非法。这种异常不影响文件继续读入，提示用户并跳过即可。

```
if (names[0].equals(names[1])) {
    throw new SocialTieConflictException(
            "The two persons between this social tie are the same! The social tie is
                    + readLineCounter + ".");
}
for (String[] s : adjacency.keySet()) {
    if (names[0].equals(s[0]) && names[1].equals(s[1])) {
        throw new SocialTieConflictException(
                "This social tie has already existed! The conflicting social tie is
                        + readLineCounter + ".");
    }
    if (names[1].equals(s[0]) && names[0].equals(s[1])) {
        throw new SocialTieConflictException(
                "This social tie has already existed! The conflicting social tie is
                        + readLineCounter + ".");
    }
}
```

3.1.3 各异常的处理（catch）

3.1.3.1 现场处理（try-catch）

我认为，能够现场处理的 exception 都应该是无关紧要的异常。比如：

1. DegreeOutOfRangeException

```
try {
    if (originDegree < 0 || originDegree >= 360) {
        throw new DegreeOutOfRangeException(
                "Original degree is less than 0 or no less than 360! Because degre
    }
} catch (DegreeOutOfRangeException e) {
    stellarSystemLogger.log(Level.WARNING, e.getMessage(), e);
    originDegree = originDegree % 360;
}
```

对非法的初始角度，模 360 取余继续操作即可。

2. PlanetConflictException

```
    } catch (PlanetConflictException e) {
        stellarSystemLogger.log(Level.WARNING, e.getMessage()
                + " The latter conflicting planet is skipped. Exception line: " + readLi
        System.out
                .println("The latter conflicting planet is skipped. Exception line: " +
    }
} // end while((line = br.readLine()) != null)
```

在读文件的过程中遇到冲突的行星，将后者跳过，继续读文件即可。

3. IntermediaryNotExistException

```
try {
    if (!existIntermediary) {
        throw new IntermediaryNotExistException("This intermediary doesn't exist!");
    }
} catch (IntermediaryNotExistException e) {
    socialNetworkCircleLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
    break;
}
```

这个异常相对其它现场处理的有点严重，它是在应用层面抛出，也是现场解决的，当在社交网络关系应用中求信息扩散度，而中介人不存在时，需要马上抛出异常，并停止当前功能调用。

4. FriendConflictException

```
    }
} catch (FriendConflictException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println(e.getMessage());
} catch (SocialTieConflictException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println(e.getMessage());
}
```

在读文件中遇到冲突的 Friend，将后者跳过，继续读文件即可。

5. SocialTieConflictException

```
    }
} catch (FriendConflictException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println(e.getMessage());
} catch (SocialTieConflictException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println(e.getMessage());
}
```

在读文件中遇到冲突的 SocialTie，将后者跳过，继续读文件即可。

6. NonexistentFriendInSocialTieException

```
try {
    boolean friendExisted = false;
    for (PhysicalObject p : friends) {
        if (p.getName().equals(names[0]) || p.getName().equals(names[1])) {
            friendExisted = true;
        }
    }
    if (!friendExisted) {
        throw new NonexistentFriendInSocialTieException("A nonexistent friend shows in
    }
} catch (NonexistentFriendInSocialTieException e) {
    socialNetworkCircleLogger.log(Level.WARNING, e.getMessage(), e);
    System.out.println("This social tie is skipped. Exception social tie: " + names[0]
    continue;
}
```

在读文件中遇到一个 SocialTie 中出现了不存在的 friend，只需将该 SocialTie 跳过即可。

7. MementoRollBackTooManyStepsException

```
public Memento<L, E> getMemento(int i) {
    try {
        if (mementos.size() - 1 < i) {
            throw new MementoRollBackTooManyStepsException("Cannot roll so many back!");
        }
    } catch (MementoRollBackTooManyStepsException e) {
        System.out.println("Roll back resetting fail! The state doesn't change.");
        return this.mementos.get(this.mementos.size() - 1);
    }
    return this.mementos.get(this.mementos.size() - i - 1);
}
```

在原子结构系统中使用跃迁状态回溯功能时，如果触发了该异常，当场处理即可。

3.1.3.2 向上抛出的异常（throws）

向上抛出的异常，一般都是比较严重的异常，其实在我的设计中，凡是向上抛出的异常，都是可以在现场解决的。但是有时候这种现场处理反馈的信息不够优雅，或者在测试中无法检测这些异常，所以就将大部分设计都改为了向上抛出，在上级 catch，做好 throws 声明。

1. AtomElementException
   在原子结构应用中，读文件时 try-catch。

```
} catch (AtomElementException e) {
    atomStructureLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

2. AtomTrackNumException
   在原子结构应用中，读文件时 try-catch。

```
} catch (AtomTrackNumException e) {
    atomStructureLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

3. AtomElectronNumException
   在原子结构应用中，读文件时 try-catch。

```
} catch (AtomElectronNumException e) {
    atomStructureLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

4. NoObjectOnTrackException
   在做删除轨道上的物体操作时，try-catch，三个应用中都有用到。

```
try {
    atomStructure.deletePhysicalObjectFromTrack(electronFactory.produce(),
            trackFactory.produce((double) sourceRadius));
} catch (NoObjectOnTrackException e) {
    System.err.println("Fail track: " + sourceRadius + ".");
    atomStructureLogger.log(Level.SEVERE, e.getMessage() + " Fail track: " + source
    break;
}
```

5. DataSyntaxException
   在做读取文件时 try-catch，主要是行星轨道系统和社交关系网络中要用

到。

```
} catch (DataSyntaxException e) {
    stellarSystemLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

6. DataScientificNumberException
   这个异常是在 StellarSystem 类中的 parseScientific 方法抛出的，经过 StellarSystem.readFromFile 调用，再经过应用代码调用，才 catch 处理。

```
} catch (DataScientificNumberException e) {
    stellarSystemLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

7. DataNonScientificNumberException
   该异常是在 StellarSystem 类中的 parseScientific 方法抛出的，在被调用的上一级直接 catch 处理。因为在应用代码和 readFromFile 代码中都有调用 parseNumber 方法，所有 catch 该异常处理的地方较多。但其实该异常并不严重，对大于 10000 的没有用科学记数法的数字照读不误，只是给出提示信息。

```
try {
    planetRadius = StellarSystem.parseNumber(matcher.group(4));
} catch (DataNonScientificNumberException e) {
    System.err.println(
            "There is at least one number greater than 10000 which doesn't use scient
    stellarSystemLogger.log(Level.WARNING, e.getMessage()
            + " There is at least one number greater than 10000 which doesn't use sc
            e);
```

8. IllegalIntimacyInSocialTieException
   该异常在社交关系网络应用中读文件时 try-catch。

```
} catch (IllegalIntimacyInSocialTieException e) {
    socialNetworkCircleLogger.log(Level.SEVERE, e.getMessage(), e);
    System.err.println(e.getMessage());
```

3.2 Assertion and Defensive Programming

3.2.1 checkRep()检查 invariants

checkRep 顾名思义，是用来检查 representaion invariants 合法性的，所以没有 rep 的 ADT 就没有 checkRep 了，比如说 API 类，Helper 类，applications 类等。

3.2.1.1 centralObject

3.2.1.1.1 Nucleus

```
private final char[] elementName;
```

```
* Representation invariant:
*   The length of elementName must be one or two.
*   If the length is one, elementName[0] must be a capital letter.
*   If the length is two, elementName[1] must be a lower-case letter.
*
```

```
// checkRep
private void checkRep() {
    assert (this.elementName.length == 1 && Character.isUpperCase(this.elementName[0]))
            || (this.elementName.length == 2 && Character.isUpperCase(this.elementName[0])
                    && Character.isLowerCase(this.elementName[1]));
}
```

3.2.1.1.2 Person

完全继承自 Friend.

3.2.1.1.3 Star

```
private final String name;
private final double radius;
@SuppressWarnings("unused")
private final double mass;
```

```
* Representation invariant:
*   name only consists of letters or numbers and shouldn't contain any blank space or other character.
*   radius must be positive.
*   mass must be positive.
```

```
// checkrep
private void checkRep() {
    assert !this.name.contains("[^a-zA-Z0-9]") && this.radius > 0 && this.mass > 0;
}
```

3.2.1.2 circularOrbit

3.2.1.2.1 ConcreteCircularOrbit<L, E>

```
protected L centralObject;
protected List<Track> tracks = new ArrayList<Track>();
protected Map<Track, List<E>> objectsInTrack = new HashMap<Track, List<E>>();
@SuppressWarnings("unchecked")
protected Map<E, Double>[] relationBetweenCentralAndObject = new Map[2];
protected List<Edge<E>> relationBetweenObjects = new ArrayList<Edge<E>>();
```

```
* Representation invariant:
*   tracks in tracks must be in ascend order of radius.
*   values in realtionBetweenCentralAndObject is more than 0 and no more than 1.
*
```

```
// checkRep
private void checkRep() {
    for (int i = 1; i < tracks.size(); i++) {
        assert tracks.get(i - 1).getRadius() <= tracks.get(i).getRadius();
    }
    for (Double intimacy : this.relationBetweenCentralAndObject[0].values()) {
        assert intimacy > 0 && intimacy <= 1;
    }
    for (Double intimacy : this.relationBetweenCentralAndObject[1].values()) {
        assert intimacy > 0 && intimacy <= 1;
    }
}
```

3.2.1.2.2 Edge<E>

```
private final E source;
private final E target;
private final double weight;
```

```
* Representation invariant:
*  source which represents an edge's start, shouldn't be null
*  target which represents an edge's end, shouldn't be null
*  weight which represents an edge's weight, should be a decimal from 0 to 1
```

```
// checkRep
private void checkRep() {
    assert weight > 0 && weight <= 1;
    assert source != null;
    assert target != null;
}
```

3.2.1.3 physicalObject

3.2.1.3.1 Electorn

```
private final double degree;
```

```
* Representation invariant:
*  degree is no less than 0 and less than 360, unit is degree
```

```
// checkRep
private void checkRep() {
    assert this.degree >= 0 && this.degree < 360;
}
```

3.2.1.3.2 Friend

```
protected final String name;
protected final int age;
protected final char sex;
private final double degree;
```

```
* Representation invariant:
*  name should be consist of only letters or numbers and shouldn't contain any blank space or any other characters.
*  age should be a positive integer.
*  sex should be either 'M' or 'F'. 'M' representing male and 'F' representing female.
*  degree is no less than 0 and less than 360, unit is degree
```

```
// checkRep
private void checkRep() {
    assert !this.name.contains("[^a-zA-Z0-9]");
    assert this.age > 0;
    assert this.sex == 'M' || this.sex == 'F';
    assert this.degree >= 0 && this.degree < 360;
}
```

3.2.1.3.3 Planet

```
private final String name;
private final String state;
private final String color;
private final double radius;
private final double speed;
private final boolean direct;
private final double degree;
```

```
* Representation invariant:
*  name should be consist of only letters or numbers and shouldn't contain any blank space or any other characters.
*  state should be consist of only letters or numbers and shouldn't contain any blank space or any other characters.
*  color should be consist of only letters or numbers and shouldn't contain any blank space or any other characters.
*  radius must be positive
*  speed must be non-negative
*  degree is no less than 0 and less than 360, unit is degree
```

```
// checkRep
private void checkRep() {
    assert !this.name.contains("[^a-zA-Z0-9]");
    assert !this.state.contains("[^a-zA-Z0-9]");
    assert !this.color.contains("[^a-zA-Z0-9]");
    assert this.radius > 0;
    assert this.speed >= 0;
    assert this.degree >= 0 && this.degree < 360;
}
```

3.2.1.4 Track

3.2.1.4.1 NormalTrack

```
private final double radius;

* Representation invariant:
*   radius must be positive

// checkRep
private void checkRep() {
    assert this.radius > 0;
}
```

3.2.2 Assertion 保障 pre-/post-condition

3.2.2.1 Pre-condition

关于 pre-condition，有些方法对传入的参数没有什么严格要求，所以就 assert parameter != null. 下面只说一些有对参数有严格意义上要求的。

```
 * @param c  circular orbit with relationship which is mostly social network
 *           circle
 * @param e1 source object in relationship
 * @param e2 target object in relationship
 * @return logical distance(length of the shortest path) from e1 to e2
 */
public int getLogicalDistance(CircularOrbit<L, E> c, E e1, E e2) {
    assert e1.getClass() == Friend.class && e2.getClass() == Friend.class : "e1 and e2 must be Friend class!";

 * @param source        a source friend
 * @param physicalEdges a list of edges representing all adjacency
 * @return a mapping from physical objects to which are adjacent from source, to
 *         intimacy of these relationship
 */
public static Map<PhysicalObject, Double> targets(PhysicalObject source, List<Edge<PhysicalObject>> physicalEdges) {
    assert source.getClass() == Friend.class : "source must be a friend";

 * @param c  circular orbit with accurate physical object position which is
 *           mostly stellar system.
 * @param e1 a physical object
 * @param e2 a physical object
 * @return physical distance from e1 to e2
 */
public double getPhysicalDistance(CircularOrbit<L, E> c, E e1, E e2) {
    assert e1.getClass() == Planet.class && e2.getClass() == Planet.class : "e1 and e2 must be Planet class!";

  * @param c circular orbit with accurate physical object position which is
  *           mostly stellar system.
  * @param e a physical object
  * @return physical distance from central object to e
  */
 public double getPhysicalDistanceFromCentralToObject(CircularOrbit<L, E> c, E e) {
     assert e.getClass() == Planet.class : "e must be Planet class!";
```

```
     * @param c1 a circular orbit
     * @param c2 a circular orbit
     * @return differences between these two circular orbits
     */
    public Difference getDifference(CircularOrbit<L, E> c1, CircularOrbit<L, E> c2) {
        assert c1.getCentralObject().getClass() == c2.getCentralObject()
                .getClass() : "Two circular orbits are not the same type!";

    /**
     * Observer. Get a memento in order to roll back to a historical version.
     *
     * @param i the number of version rolled back
     * @return the memento after i versions rolled back
     */
    public Memento<L, E> getMemento(int i) {
        assert i >= 0 : "version number can't be negative.";
```

```
@Override
public void addRelationshipBetweenCentralAndPhysical(E physicalObject, boolean fromCenctral, double weight) {
    assert physicalObject != null && weight > 0
            && weight <= 1 : "physicalObject can't be null and weight must be in range (0,1].";

@Override
public void addRelationshipBetweenPhysicalAndPhysical(E physicalObjectA, E physicalObjectB, double weight) {
    assert physicalObjectA != null && physicalObjectB != null && weight > 0
            && weight <= 1 : "physicalObject can't be null and weight must be in range (0,1].";
```

```
public List<String> removeComments(String[] source) {
    assert source.length >= 1 && source.length <= 100 : "The length of source is in the range [1,100]."; // new
    boolean inBlock = false;
    StringBuilder newline = new StringBuilder();
    List<String> ans = new ArrayList<String>(); // List<String> ans = new List();
    for (String line : source) {
        assert line.length() >= 0 && line.length() <= 80 : "The length of source[i] is in the range [0,80]."; // new
        int i = 0;
        char[] chars = line.toCharArray();
        if (!inBlock)
            newline = new StringBuilder();
        while (i < line.length()) {
            assert chars[i] >= 32 && chars[i] <= 127 && chars[i] != 34
                    && chars[i] != 39 : "There are no single-quote, double-quote, or control characters in the source
```

```
public TopVotedCandidate(int[] persons, int[] times) {
    assert persons.length == times.length && persons.length >= 1
            && persons.length <= 5000 : "1 <= persons.length = times.length <= 5000";
    A = new ArrayList<List<Vote>>(); // new ArrayList();
    Map<Integer, Integer> count = new HashMap<Integer, Integer>(); // new HashMap();
    for (int i = 0; i < persons.length; ++i) {
        assert persons[i] <= persons.length && persons[i] >= 0 : "0 <= persons[i] <= persons.length";
        assert times[i] >= 0
                && times[i] <= 10e9 : "times is a strictly increasing array with all elements in [0, 10^9]";
        if (i < times.length - 1) {
            assert times[i] < times[i + 1] : "times is a strictly increasing array with all elements in [0, 10^9]";
        }

    public int q(int t) {
        assert t >= A.get(0).get(0).time : "TopVotedCandidate.q(int t) is always called with t >= times[0].";
```

3.2.2.2 Post-condition

一般说来，ADT 的方法中，应该只有 mutator 方法有 post-condition。而 observer 方法是获取 ADT 的 rep，constructor 是初始化 rep，没什么 post-condition，这些的合法性都是由 checkRep 来检验的。还有一些应用层面的方法，返回值应该符合一些实际条件，才可认为是合法的。IMMUTABLE 的类中没有 mutator 方法，自然也就不用检查 post-condition 了。

3.2.2.2.1 一些 API 方法的 post-condition 检查

```
public double getObjectDistributionEntropy(CircularOrbit<L, E> c) {
    assert c != null;
    double entropy = 0;
    Map<Track, List<E>> objectsInTrack = c.getObjectsInTrack();
    int objectsNum = 0, trackNum = 0;
    int[] objectsNumEachTrack = new int[objectsInTrack.keySet().size()];
    for (List<E> l : objectsInTrack.values()) {
        objectsNum += l.size();
        objectsNumEachTrack[trackNum++] = l.size();
    }
    for (int i = 0; i < trackNum; i++) {
        double possibility = (double) objectsNumEachTrack[i] / objectsNum;
        entropy -= possibility * Math.log(possibility);
    }
    assert entropy >= 0 : "entropy must be non-negative";
    return entropy;
}
```

```
    }
    if (!markedMap.get((PhysicalObject) e2)) {
        assert false : "In social graph there must be a path between any two friends in circle system.";
        distance = -1;
    } else
        for (PhysicalObject e = (PhysicalObject) e2; !e.equals(e1); e = edgeTo.get(e))
            distance++;
    assert distance >= 0 : "distance can't be negative.";
    return distance;
}
```

```
public double getPhysicalDistanceFromCentralToObject(CircularOrbit<L, E> c, E e) {
    assert e.getClass() == Planet.class : "e must be Planet class!";
    double radius = 0;
    for (Track track : c.getTracks()) {
        for (E e1 : c.getObjectsInTrack().get(track)) {
            if (e1.getClass() == Planet.class && ((Planet) e1).getName().equals(((Planet) e).getName())) {
                radius = track.getRadius();
            }
        }
    }
    assert radius >= 0 : "physical distance can't be negative";
    return radius;
}
```

3.2.2.2.2 轨道系统 ADT 中的 mutator 方法的 post-condition 检查

```
@Override
public void addTrack(Track track) {
    assert track != null : "track can't be null!";
    int index = 0;
    for (Track t : this.tracks) {
        if (track.getRadius() > t.getRadius()) {
            index++;
        }
    }
    this.tracks.add(index, track);
    List<E> objects = new ArrayList<E>();
    objectsInTrack.put(track, objects);
    assert this.tracks.contains(track) && this.objectsInTrack.keySet().contains(track);
    checkRep();
}
```

```java
@Override
public void deleteTrack(Track track) {
    assert track != null : "track can't be null!";
    for (int i = 0; i < tracks.size(); i++) {
        if (track.equals(tracks.get(i))) {
            this.objectsInTrack.remove(this.tracks.get(i));
            this.tracks.remove(i);
            checkRep();
            break;
        }
    }
    assert !this.tracks.contains(track) && !this.objectsInTrack.keySet().contains(track);
    checkRep();
}
```

```java
@Override
public void addCentralObject(L centralObject) {
    assert centralObject != null : "centralObject can't be null!";
    this.centralObject = centralObject;
    assert this.centralObject.equals(centralObject);
    checkRep();
}
```

```java
@Override
public void addPhysicalObjectToTrack(E physicalObject, Track track) {
    assert physicalObject != null && track != null : "physicalObject and track can't be null!";
    int index = 0;
    for (Track t : this.tracks) {
        if (track.equals(t)) {
            for (E e : this.objectsInTrack.get(t)) {
                if (((PhysicalObject) physicalObject).getDegree() > ((PhysicalObject) e).getDegree()) {
                    index++;
                }
            }
            objectsInTrack.get(t).add(index, physicalObject);
            checkRep();
            break;
        }
    }
    assert this.objectsInTrack.get(track).contains(physicalObject);
    checkRep();
}
```

```java
@Override
public void addRelationshipBetweenCentralAndPhysical(E physicalObject, boolean fromCenctral, double weight) {
    assert physicalObject != null && weight > 0
            && weight <= 1 : "physicalObject can't be null and weight must be in range (0,1].";
    if (fromCenctral) {
        this.relationBetweenCentralAndObject[0].put(physicalObject, weight);
    } else {
        this.relationBetweenCentralAndObject[1].put(physicalObject, weight);
    }
    if (fromCenctral) {
        assert this.relationBetweenCentralAndObject[0].get(physicalObject).equals(weight);
    } else {
        assert this.relationBetweenCentralAndObject[1].get(physicalObject).equals(weight);
    }
    checkRep();
}
```

```java
@Override
public void addRelationshipBetweenPhysicalAndPhysical(E physicalObjectA, E physicalObjectB, double weight) {
    assert physicalObjectA != null && physicalObjectB != null && weight > 0
            && weight <= 1 : "physicalObject can't be null and weight must be in range (0,1].";
    Edge<E> edge1 = new Edge<E>(physicalObjectA, physicalObjectB, weight);
    Edge<E> edge2 = new Edge<E>(physicalObjectB, physicalObjectA, weight);
    this.relationBetweenObjects.add(edge1);
    this.relationBetweenObjects.add(edge2);
    assert this.relationBetweenObjects.contains(edge1) && this.relationBetweenObjects.contains(edge2);
    checkRep();
}
```

```
@Override
public void deleteRelationshipBetweenPhysicalAndPhysical(E physicalObjectA, E physicalObjectB) {
    assert physicalObjectA != null && physicalObjectB != null;
    Iterator<Edge<E>> iterator = this.relationBetweenObjects.iterator();
    while (iterator.hasNext()) {
        Edge<E> edge = iterator.next();
        if (edge.getSource().equals(physicalObjectA) && edge.getTarget().equals(physicalObjectB)) {
            iterator.remove();
        }
        if (edge.getSource().equals(physicalObjectB) && edge.getTarget().equals(physicalObjectA)) {
            iterator.remove();
        }
    }
    assert !this.relationBetweenObjects.contains(new Edge<E>(physicalObjectA, physicalObjectB, 0.5))
            && !this.relationBetweenObjects.contains(new Edge<E>(physicalObjectB, physicalObjectA, 0.5));
    checkRep();
}
```

```
@Override
public void deletePhysicalObjectFromTrack(E physicalObject, Track track) throws NoObjectOnTrackException {
    assert physicalObject != null && track != null;
    boolean exist = false;
    for (Track t : this.tracks) {
        if (t.equals(track)) {
            exist = true;
            this.objectsInTrack.get(t).remove(physicalObject);
            this.relationBetweenCentralAndObject[0].remove(physicalObject);
            this.relationBetweenCentralAndObject[1].remove(physicalObject);
        }
    }
    if (!exist) {
        throw new NoObjectOnTrackException("Delete fail! This track has no such object!");
    }
    Iterator<Edge<E>> iterator = this.relationBetweenObjects.iterator();
    while (iterator.hasNext()) {
        Edge<E> edge = iterator.next();
        if (edge.getSource().equals(physicalObject) || edge.getTarget().equals(physicalObject)) {
            iterator.remove();
        }
    }
    assert !this.objectsInTrack.containsKey(physicalObject);
    checkRep();
}
```

```
@Override
public void resetObjectsAndTrack() {
    this.objectsInTrack = new HashMap<Track, List<E>>();
    this.tracks = new ArrayList<Track>();
    assert this.objectsInTrack.isEmpty() && this.tracks.isEmpty();
    checkRep();
}
```

特别地，由于三个应用的轨道系统都是面向接口编程，所以具体类中有接口中定义的所有方法，但有些方法，具体类中是不需要也不应该实现的，比如说关于 relationship 的方法，在原子结构中就不应该实现，这种情况下，在具体类中重写这些不该有的方法，并 assert false。

比如：

```
/**
 * This method is non-existent in atom structure.
 */
@Override
public void addRelationshipBetweenPhysicalAndPhysical(PhysicalObject physicalObjectA,
        PhysicalObject physicalObjectB, double weight) {
    assert false : "shouldn't call this method";
}

/**
 * This method is non-existent in atom structure.
 */
@Override
public void deleteRelationshipBetweenPhysicalAndPhysical(PhysicalObject physicalObjectA,
        PhysicalObject physicalObjectB) {
    assert false : "shouldn't call this method";
}

/**
 * This method is non-existent in atom structure.
 */
@Override
public Map<PhysicalObject, Double>[] getRelationBetweenCentralAndObject() {
    assert false : "shouldn't call this method";
    return null;
}
```

3.3 Logging

本实验中，日志功能的实现调用了 Java 的库 java.util.logging。并且日志功能仅支持在应用中实现，不支持在测试时使用。

3.3.1 写日志

在主函数 Main 中维护三个应用的日志管理。

```
private static Logger atomStructureLogger = Logger.getLogger("AtomStructure Log");
private static Logger stellarSystemLogger = Logger.getLogger("StellarSystem Log");
private static Logger socialNetworkCircleLogger = Logger.getLogger("SocialNetworkCircle Log");
```

在后续的日志创建中，如果 Logger.getLogger(String name)的 name 参数和上述三个之一相同，那么日志变量将指向同一块内存区域，即共享同一 logger。

然后对日志配置。首先配置日志的输出语言为英文。

```
Locale.setDefault(new Locale("en", "EN"));
```

然后设置日志显示信息的最低级别（即包括了 INFO, WARNING 和 SEVERE）。

```
atomStructureLogger.setLevel(Level.INFO);
stellarSystemLogger.setLevel(Level.INFO);
socialNetworkCircleLogger.setLevel(Level.INFO);
```

接着对三个 logger 配置加入相应的文件管理，使得日志可以写入到文件中。

```
FileHandler atomStructureHandler = null, stellarSystemHandler = null, socialNetworkCircleHandler =
try {
    atomStructureHandler = new FileHandler("log/atomStructureLog.log", true);
    stellarSystemHandler = new FileHandler("log/stellarSystemLog.log", true);
    socialNetworkCircleHandler = new FileHandler("log/socialNetworkCircleLog.log", true);
} catch (SecurityException e) {
    e.printStackTrace();
    return;
} catch (IOException e) {
    e.printStackTrace();
    return;
}
```

对文件管理 handler 配置好文件写入的格式，采用 SimpleFormatter 格式文件，可读性高，但在检索时可能比较麻烦。

```
atomStructureHandler.setFormatter(new SimpleFormatter());
stellarSystemHandler.setFormatter(new SimpleFormatter());
socialNetworkCircleHandler.setFormatter(new SimpleFormatter());
atomStructureLogger.addHandler(atomStructureHandler);
stellarSystemLogger.addHandler(stellarSystemHandler);
socialNetworkCircleLogger.addHandler(socialNetworkCircleHandler);
```

这样日志的配置就做好了。在应用类中，只需要加一个全局静态变量，调用已经创建的相关的 logger，再设置无需从控制台输出日志内容即可。

```
private static Logger stellarSystemLogger = Logger.getLogger("StellarSystem Log");
```

```
stellarSystemLogger.setUseParentHandlers(false);
```

除了应用类，凡是应用类调用的函数如果有异常被 catch 的话，其所在的类中，也应该有这个 logger 管理变量。

在写日志时，遵循几个原则。

1. 应用中使用功能
   在应用中使用的任何功能，都应该在调用之后马上生成 INFO 调用信息，在功能成功结束后，生成 INFO 成功信息。如：

```
case 2: // Visualize
    stellarSystemLogger.log(Level.INFO, "Visualize.");
    CircularOrbitHelper.visualize(stellarSystem);
    stellarSystemLogger.info("Success: Visualize.");
    break;
```

2. 需要终止当前操作的异常
   如果遇到需要终止当前操作的异常，在 catch 结束前，应该记录 SEVERE 级别的日志信息，如：

```
} catch (FileNotFoundException e) {
    stellarSystemLogger.log(Level.SEVERE,
            "File not found. File: " + file.getPath() + ". " + e.getMessage(), e);
    System.err.println("File not found. File: " + file.getPath() + ". " + e.getMessage());
```

3. 不需要终止当前操作的异常
   如果遇到不需要终止当前操作的异常，在 catch 结束前，应该记录 WARNING 级别的日志信息，如：

```
try {
    planetRadius = StellarSystem.parseNumber(matcher.group(4));
} catch (DataNonScientificNumberException e) {
    System.err.println(
            "There is at least one number greater than 10000 which doesn't use scientif
    stellarSystemLogger.log(Level.WARNING, e.getMessage()
            + " There is at least one number greater than 10000 which doesn't use scier
            e);
```

4. Assertion error

对于应用中遇到的 Assertion error，应该记录下 SEVERE 级别的信息，如：

```
System.err.println("The length of shortest path from " + friend.getName()
        + " to center is: " + apis.getLogicalDistance(socialNetworkCircle,
                friendFactory.produce("center", 1, 'M'), friend));
System.err.println(friend.getName() + " is on the " + i + "-th track.");
socialNetworkCircleLogger.log(Level.SEVERE, "The length of shortest path from "
        + friend.getName() + " to center is: "
        + apis.getLogicalDistance(socialNetworkCircle,
                friendFactory.produce("center", 1, 'M'), friend)
        + ". But " + friend.getName() + " is on the " + i + "-th track. "
        + "If a friend in on the i-th track, then the shortest length from he/s
assert false : "If a friend in on the i-th track, then the shortest length from
```

下面给出一段日志的展示：

```
[2019-05-18 15:53:20] applications.Main main
INFO: AtomStructure application activate.
[2019-05-18 15:53:22] applications.AtomStructureAPP application
INFO: Read from input/AtomicStructure.txt. Restore present state.
[2019-05-18 15:53:22] applications.AtomStructureAPP application
INFO: Success: Read from input/AtomicStructure.txt. Restore present state.
[2019-05-18 15:53:25] applications.AtomStructureAPP application
INFO: Restore transition.
[2019-05-18 15:53:28] applications.AtomStructureAPP application
INFO: Success: Roll back 2 transition versions.

```

这种日志格式所给出的信息有：时间、类、方法、信息级别和日志信息。

3.3.2 日志查询

用 java.util.logging 做日志查询是一件麻烦事，如果用这个写的日志保存成为了 SimpleFormat 那就是更麻烦的事了。在外部查询可读性非常好，在程序内部查询就不得不用文件读取和语法匹配的知识了。

因为三个程序的日志格式都是一样的，所以，我将这日志查询功能封装在了 API 中。

```
/**
 * Log search.
 *
 * @param file a log file
 */
public void logSearch(File file) {
```

日志查询功能一共给出了 4 个查询标准：按时间段查询、按日志信息等级查

询、按类查询、按方法查询。

```
private static void logSearchMenu() {
    System.out.println("1. Time period.");
    System.out.println("2. Log level.");
    System.out.println("3. Class.");
    System.out.println("4. Method.");
}
```

按时间段查询是最麻烦的，因为要对文件中的时间信息做语法匹配，并作时间比较。这里我用了 java.sql.Date 和 java.sql.Time 类，用来构造时间信息。先令用户输入要查询的时间段，并以此构造基准时间对象：

```
System.out.println("Please input start date as format below:");
System.out.println("yyyy-mm-dd");
String startDateString = in.nextLine();
System.out.println("Please input start time as format below:");
System.out.println("hh:mm:ss");
String startTimeString = in.nextLine();
Date startDate = Date.valueOf(startDateString);
Time startTime = Time.valueOf(startTimeString);
System.out.println("Please input start date as format below:");
System.out.println("yyyy-mm-dd");
String endDateString = in.nextLine();
System.out.println("Please input start time as format below:");
System.out.println("hh:mm:ss");
String endTimeString = in.nextLine();
Date endDate = Date.valueOf(endDateString);
Time endTime = Time.valueOf(endTimeString);
```

再逐行读入 log 信息，因为 log 的信息格式是已知且固定的，所以只需要做简单的语法匹配，提取时间信息就好了。

```
String dateInfo = line1.substring(line1.indexOf("[") + 1, line1.indexOf("]")).split("
String timeInfo = line1.substring(line1.indexOf("[") + 1, line1.indexOf("]")).split("
Date logDate = Date.valueOf(dateInfo);
Time logTime = Time.valueOf(timeInfo);
```

然后根据时间段的比较，判断是否从控制台输出 log 信息。

```
if (startDate.equals(endDate) && logDate.equals(startDate)
        && (!logTime.before(startTime) && !logTime.after(endTime))) {
    System.out.println(line1);
    System.out.println(line2);
} else if (logDate.after(startDate) && logDate.before(endDate)) {
    System.out.println(line1);
    System.out.println(line2);
} else if (logDate.equals(startDate) && endDate.after(startDate) && !logTime.before(sta
    System.out.println(line1);
    System.out.println(line2);
} else if (logDate.equals(endDate) && startDate.before(endDate) && !logTime.after(endTi
    System.out.println(line1);
    System.out.println(line2);
}
```

按日志信息等级查询的话，只需要去解析日志等级信息就好了。

```
System.out.println("Please input log level(Info/Warning/Severe):");
String logLevel = in.nextLine();
try {
    while ((line1 = br.readLine()) != null) {
        line2 = br.readLine();
        if (line2 == null) {
            System.out.println("Abort log reading. The number of lines in log may
            break;
        }
        String logLevelInfo = line2.substring(0, line2.indexOf(":"));
        if (logLevelInfo.equalsIgnoreCase(logLevel)) {
            System.out.println(line1);
            System.out.println(line2);
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
break;
```

按类查询的话，类的结构是包.类，用户输入一个类，解析出所处的包和类，然后按行读入文件并解析相应位置的类信息，按包名和类名分别比较即可。

```
System.out.println("Please input class information(Package.Class):");
String info = in.nextLine();
String packageInfo = info.split("\\.")[0];
String classInfo = info.split("\\.")[1];
try {
    while ((line1 = br.readLine()) != null) {
        line2 = br.readLine();
        String packageClassInfo = line1.split(" ")[2];
        String logPackageInfo = packageClassInfo.split("\\.")[0];
        String logClassInfo = packageClassInfo.split("\\.")[1];
        if (logPackageInfo.equals(packageInfo) && logClassInfo.equals(classInfo)) {
            System.out.println(line1);
            System.out.println(line2);
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
break;
```

按方法查询，类似于按类查询，不过只有一个方法名，如果在不同的类中有重名方法，将全部输出。

```
System.out.println("Please input method name:");
String methodName = in.nextLine();
try {
    while ((line1 = br.readLine()) != null) {
        line2 = br.readLine();
        String logMethodName = line1.split(" ")[3];
        if (logMethodName.equals(methodName)) {
            System.out.println(line1);
            System.out.println(line2);
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
break;
```

3.4 Testing for Robustness and Correctness

在测试案例中，凡是需要用到读取文件的，测试文件中都有一个@Before 代码块，在每个测试块执行之前，先执行该代码块，读取相关测试文件。

3.4.1 Testing strategy

3.4.1.1 APIsTest

3.4.1.1.1 CircularOrbitAPIsTest

对 CircularOrbitAPIs 类中的 API 方法的测试，要对每一个 API 方法，在三个应用都测试一遍。对 getObjectDistributionEntropy 测试，由于其计算较为复杂，而三个应用中的物体分布情况又不同，所以直接调用默认文件测试即可。getLogicalDistance 和 targets 方法只需要在社交关系网络中测试即可。getPhysicalDistance 和 getPhysicalDistanceFromCentralToObject 方法只需要在行星轨道系统中测试即可。getDifference 也需要在三个应用中分别测试一遍。

```
/*
 * Test strategy
 *  @Before readFromFile
 *      Before any test all three circular orbits should be initialized from corresponding files.
 *  testGetObjectDistributionEntropyInAtomStructure
 *      This tests getObjectDistributionEntropy in atom structure.
 *      It calls getObjectDistributionEntropy and compare it's result with my expectation.
 *  testGetObjectDistributionEntropyInStellarSystem
 *      This tests getObjectDistributionEntropy in stellar system.
 *      It calls getObjectDistributionEntropy and compare it's result with my expectation.
 *  testGetObjectDistributionEntropyInSocialNetworkCircle
 *      This tests getObjectDistributionEntropy in social network circle.
 *      It calls getObjectDistributionEntropy and compare it's result with my expectation.
 *  testGetLogicalDistance
 *      This tests getLogicalDistance in social network circle.
 *      It calls getLogicalDistance and compare results in all legal paths with expectations.
 *  testTargets
 *      This tests static method targets in social network circle.
 *      It calls targets and choose FrankLee as input to see if he's targets fit my expectation.
```

```
*  testGetPhysicalDistance
*      This tests getPhysicalDistance in stellar system.
*      It calls getPhysicalDistance and choose planets in 3rd and 4th track as input to see if the physical distance fits expectation.
*  testGetPhysicalDistanceFromCentralToObject
*      This tests getPhysicalDistanceFromCentralToObject in stellar system.
*      It calls getPhysicalDistanceFromCentralToObject and test every planet's distance to central star to see if the result fits expectation.
*  testGetDiffereceAtomStructure
*      This tests getDiffereceAtomStructure in atom structure.
*      It calls getDifferece and test if the result fits the difference between two atom structures.
*  testGetDiffereceStellarSystem
*      This tests getDiffereceAtomStructure in stellar system.
*      It calls getDifferece and test if the result fits the difference between two stellar systems.
*  testGetDiffereceSocialNetworkCircle
*      This tests getDiffereceAtomStructure in social network circle.
*      It calls getDifferece and test if the result fits the difference between two social network circles.
*/
```

3.4.1.2 centralObjectTest

对 centralObject 中的 Person 类不予测试。该类是 Friend 的子类，方法和 rep 高度重合。

3.4.1.2.1 NucleusTest

Nucleus 类中只有重写的 toString 方法，这是用来获取原子核的元素名称的。只需要测这一个方法即可。

```
/*
 * Test strategy
 *  testToString
 *      This tests toString.
 *      toString can show the element name of nucleus. Call toString to see if the result fits expectation.
 */
```

3.4.1.2.2 StarTest

Star 类中除了 equals 和 hashCode 方法以外只有两个 Observer 方法，创建对象并调取 Observer 方法测试即可。

```
/*
 * Test strategy
 *  testGetName
 *      This tests getName.
 *      Call getName to see if the result fits expectation.
 *  testGetRadius
 *      This tests getRadius.
 *      Call getRadius to see if the result fits expectation.
 */
```

3.4.1.3 circularOrbitTest

该测试包中对三个轨道系统的方法分别测试。三个轨道系统均继承了接口中的方法，有些自己不应该有的方法，都做了 assert false，测试策略就是调用读取文件，并以文件中的数据为基础来测试各种方法。对不应该有的方法，要测试 AssertionError.

3.4.1.3.1 AtomStructureTest

```
/*
 * Test strategy
 *  All tests below are about tests in atom structure.
 *  @Before readFromFile
 *      Before any test, atomStructure should be initialized from AtomicStructure.txt.
 *      The test is all about atom structure from AtomicStructure.txt.
 *  testMementoMethod
 *      This tests Memento method in transition action.
 *      It does two transition to atom structure and saves these two states.
 *      And it restores these states to see if the transition can be undone.
 *  testAddTrack
 *      This tests addTrack.
 *      Add a track to atom structure to see the change before and after adding.
 *      Call getTracks to see if the test track is added.
 *  testDeleteTrack
 *      This tests deleteTrack.
 *      Delete a track in atom structure to see the change before and after deleting.
 *      Call getTracks to see if the track in test position is missed.
 *  testAddCentralObject
 *      This tests addCentralObject.
 *      Add a central object in atom structure to see the change before and after adding.
 *      Call getCentralObject to see if the central object is the test one after adding.
 *  testAddPhysicalObjectToTrack
 *      This tests addPhysicalObjectToTrack.
 *      Add a physical object to track to see the change before and after adding.
 *      Call getObjectsInTrack to see if the number of objects in corresponding track changes after adding.
 *  testDeletePhysicalObjectFromTrack
 *      This tests deletePhysicalObjectFromTrack.
 *      Delete a physical object from track to see the change before and after deleting.
 *      Call getObjectsInTrack to see if the number of objects in corresponding track changes after deleting.
 *  testResetObjectsAndTrack
 *      This tests resetObjectsAndTrack.
 *      Call resetObjectsAndTrack and then see if the tracks and objectInTracks is empty.
 *  testGetCentralObject
 *      This tests getCentralObject.
 *      See if the central object fits the data in file.
 *  testGetTracks
 *      This tests getTracks.
 *      See if the tracks fits the data in file.
 *  testGetObjectsInTrack
 *      This tests getObjectInTrack.
 *      See if the objects in tracks fits the data in file.
 *  testConstructSocialNetworkCircle
 *      Test that atom structure shouldn't have constructSocialNetworkCircle method.
 *  testAddRelationshipBetweenCentralAndPhysical
 *      Test that atom structure shouldn't have addRelationshipBetweenCentralAndPhysical method.
 *  testAddRelationshipBetweenPhysicalAndPhysical
 *      Test that atom structure shouldn't have addRelationshipBetweenPhysicalAndPhysical method.
 *  testDeleteRelationshipBetweenPhysicalAndPhysical
 *      Test that atom structure shouldn't have deleteRelationshipBetweenPhysicalAndPhysical method.
 *  testGetRelationBetweenCentralAndObject
 *      Test that atom structure shouldn't have getRelationBetweenCentralAndObject method.
 *  testGetRelationBetweenObjects
 *      Test that atom structure shouldn't have getRelationBetweenObjects method.
 */
```

3.4.1.3.2 StellarSystemTest

```
/*
 * Test strategy
 *  All tests below are about tests in stellar system.
 *  @Before readFromFile
 *      Before any test, stellarSystem should be initialized from StellarSystem.txt.
 *      The test is all about stellar system from StellarSystem.txt.
 *  testAddTrack
 *      This tests addTrack.
 *      Add a track to stellar system to see the change before and after adding.
 *      Call getTracks to see if the test track is added.
 *  testDeleteTrack
 *      This tests deleteTrack.
 *      Delete a track stellar system to see the change before and after deleting.
 *      Call getTracks to see if the track in test position is missed.
 *  testAddCentralObject
 *      This tests addCentralObject.
 *      Add a central object in stellar system to see the change before and after adding.
 *      Call getCentralObject to see if the central object is the test one after adding.
 *  testAddPhysicalObjectToTrack
 *      This tests addPhysicalObjectToTrack.
 *      Add a physical object to track to see the change before and after adding.
 *      Call getObjectsInTrack to see if the test object shows in the corresponding track after adding.
 *  testResetObjectsAndTrack
 *      This tests resetObjectsAndTrack.
 *      Call resetObjectsAndTrack and then see if the tracks and objectInTracks is empty.
*  testGetCentralObject
*      This tests getCentralObject.
*      See if the central object fits the data in file.
*  testGetTracks
*      This tests getTracks.
*      See if the tracks fits the data in file.
*  testGetObjectsInTrack
*      This tests getObjectInTrack.
*      See if the objects in tracks fits the data in file.
*  testParseNumber
*      This tests static method parseNumber in StellarSystem.
*      1. a number string without "e", i.e. non-scientific notation.
*      2. a number string with "e", i.e. scientific notatin.
*  testConstructSocialNetworkCircle
*      Test that stellar system shouldn't have constructSocialNetworkCircle method.
*  testSave
*      Test that stellar system shouldn't have save method.
*  testRestore
*      Test that stellar system shouldn't have restore method.
*  testAddRelationshipBetweenCentralAndPhysical
*      Test that stellar system shouldn't have addRelationshipBetweenCentralAndPhysical method.
*  testDeleteRelationshipBetweenPhysicalAndPhysical
*      Test that stellar system shouldn't have deleteRelationshipBetweenPhysicalAndPhysical method.
*  testGetRelationBetweenCentralAndObject
*      Test that stellar system shouldn't have getRelationBetweenCentralAndObject method.
*  testGetRelationBetweenObjects
*      Test that stellar system shouldn't have getRelationBetweenObjects method.
*/
```

3.4.1.3.3 SocialNetworkCircleTest

```
/*
 * Test strategy
 *  All tests below are about tests in social network circle.
 *  @Before readFromFile
 *      Before any test, atomStructure should be initialized from SocialNetworkCircle.txt.
 *      The test is all about atom structure from SocialNetworkCircle.txt.
 *  testAddTrack
 *      This tests addTrack.
 *      Add a track to atom structure to see the change before and after adding.
 *      Call getTracks to see if the test track is added.
 *  testDeleteTrack
 *      This tests deleteTrack.
 *      Delete a track in atom structure to see the change before and after deleting.
 *      Call getTracks to see if the track in test position is missed.
 *  testAddCentralObject
 *      This tests addCentralObject.
 *      Add a central object to see the change before and after adding.
 *      Call getCentralObject to see if the central object is the test one after adding.
 *  testAddPhysicalObjectToTrack
 *      This tests addPhysicalObjectToTrack.
 *      Add a physical object to track to see the change before and after adding.
 *      Call getObjectsInTrack to see if the number of objects in corresponding track changes after adding.
 *  testAddRelationshipBetweenCentralAndPhysical
 *      This tests addRelationshipBetweenCentralAndPhysical.
 *      Add a relationship between the central person and a friend. Use FrankLee as the test friend.
 *      This relationship shouldn't exist before adding. See if the relationship shows after adding.
 *  testAddRelationshipBetweenPhysicalAndPhysical
 *      This tests addRelationshipBetweenPhysicalAndPhysical.
 *      Add a relationship between two friends. Use LisaWong and TomWong as the test friends.
 *      This relationship shouldn't exist before adding. See if the relationship shows after adding.
 *  testDeleteRelationshipFromPhysicalToPhysical
 *      This tests deleteRelationshipFromPhysicalToPhysical.
 *      Delete a relationship between two friends. Use TomWong and FrankLee as the test friends.
 *      This relationship should exist before adding. See if the relationship disappears after adding.
 *  testResetObjectsAndTrack
 *      This tests resetObjectsAndTrack.
 *      Call resetObjectsAndTrack and then see if the tracks and objectInTracks is empty.
 *  testGetCentralObject
 *      This tests getCentralObject.
 *      See if the central object fits the data in file.
 *  testGetTracks
 *      This tests getTracks.
 *      See if the tracks fit the data in file.
 *  testGetObjectsInTrack
 *      This tests getObjectInTrack.
 *      See if the objects in tracks fit the data in file.
 *  testGetRelationBetweenCentralAndObject
 *      This tests getRelationBetweenCentralAndObject.
 *      See if all the relationship between the central person and friends fit the data in file.
 *  testGetRelationBetweenObjects
 *      This tests getRelationBetweenObjects.
 *      See if all the relationship between friends fit the data in file.
 *  testSave
 *      Test that social network circle shouldn't have save method.
 *  testRestore
 *      Test that social network circle shouldn't have restore method.
 */
```

3.4.1.4 physicalObjectTest

这个包中的测试代码测试了 physical object 的工厂类和 ADT 类，工厂类中主要测试工厂是否能正确制造出一个 physical object，ADT 类测试主要测试各 physical object 中的方法。由于所有的 physical object 共用一个接口，所有对于个别的 physical object 不应该具有的方法，要用 AssertionError 测试。

3.4.1.4.1 ElectronFactoryTest

```
/*
 * Test strategy
 *  testProduce
 *      This tests produce.
 *      Call produce to create an instance to see if the instance fits expectation.
 */
```

3.4.1.4.2 FriendFactoryTest

```
/*
 * Test strategy
 *  testProduce
 *      This tests produce.
 *      Call produce to create an instance to see if the instance fits expectation.
 */
```

3.4.1.4.3 PlanetFactoryTest

```
/*
 * Test strategy
 *  testProduce
 *      This tests produce.
 *      Call produce to create an instance to see if the instance fits expectation.
 */
```

3.4.1.4.4 ElectronTest

```
/*
 * Test strategy
 *  testGetDegree
 *      This tests getDegree.
 *      Because the degree of electron is generated randomly so there's no an expectation result.
 *      The degree test actually has been ensured by checkRep. This test has no necessary.
 *  testEquals
 *      This tests equals.
 *      Because all electrons have no difference, equals method is override that every object is the same.
 *      To see if equals works well, generate two electrons and assert they are equal.
 *  testHashCode
 *      This tests hashCode
 *      Because all electrons are equal, every electron's hash code should be the same, supposed to be -1.
 *      To see if hashCode works well, to see if a electron's hash code is -1.
 *  testGetDirect
 *      Test that electron shouldn't have getDirect method.
 *  testRadius
 *      Test that electron shouldn't have getRadius method.
 *  testName
 *      Test that electron shouldn't have getName method.
 *  testSpeed
 *      Test that electron shouldn't have getSpeed method.
 */
```

3.4.1.4.5 FriendTest

```
/*
 * Test strategy
 *  testGetName
 *      This tests getName.
 *      Call getName to see if the result fits expectation.
 *  testGetDegree
 *      This tests getDegree.
 *      Because the degree of friend is generated randomly so there's no an expectation result.
 *      The degree test actually has been ensured by checkRep. This test has no necessary.
 *  testEquals
 *      This tests equals.
 *      Two friends are equal only when they have the same name, age and sex.
 *      Generate two friends who have the same name, age and sex and see if they are equal.
 *      Generate two friends whose name, age and sex are not all the same, and see if they are not equal.
 *  testHashCode
 *      This tests hashCode
 *      Two friends' hash codes are equal only when they are equal.
 *  testGetDirect
 *      Test that friend shouldn't have getDirect method.
 *  testRadius
 *      Test that friend shouldn't have getRadius method.
 *  testSpeed
 *      Test that friend shouldn't have getSpeed method.
 */
```

3.4.1.4.6 PlanetTest

```
/*
 * Test strategy
 *  testGetName
 *      This tests getName.
 *      Call getName to see if the result fits expectation.
 *  testGetDegree
 *      This tests getDegree.
 *      Because the degree of planet is generated designative.
 *      Test if a planet's degree is the same as the initialized degree.
 *  testGetDirect
 *      This tests getDirect.
 *      Because the direct of planet is generated designative.
 *      Test if a planet's direct is the same as the initialized direct.
 *  testGetSpeed
 *      This tests getSpeed.
 *      Because the speed of planet is generated designative.
 *      Test if a planet's speed is the same as the initialized speed.
 *  testGetRadius
 *      This tests getRadius.
 *      Because the radius of planet is generated designative.
 *      Test if a planet's radius is the same as the initialized radius.
 *  testEquals
 *      This tests equals.
 *      Two planets are equals only when they have the same name with case sensitive.
 *      To see if equals works well, generate two planets who have the same name and see if they are equal.
 *  testHashCode
 *      This tests hashCode
 *      Two planets are equal only when they have the same name. So the hash code should link with name.
 *      To see if hashCode works well, see if a planet's hash code is the same as name's hash code.
 */
```

3.4.1.4.7 PlanetWithSatelliteTest

```
/*
 * Test strategy
 *  testGetName
 *      This tests getName.
 *      Call getName to see if the result fits expectation.
 *  testGetDegree
 *      This tests getDegree.
 *      Because the degree of planet is generated designative.
 *      Test if a planet's degree is the same as the initialized degree.
 *  testGetDirect
 *      This tests getDirect.
 *      Because the direct of planet is generated designative.
 *      Test if a planet's direct is the same as the initialized direct.
 *  testGetSpeed
 *      This tests getSpeed.
 *      Because the speed of planet is generated designative.
 *      Test if a planet's speed is the same as the initialized speed.
 *  testGetRadius
 *      This tests getRadius.
 *      Because the radius of planet is generated designative.
 *      Test if a planet's radius is the same as the initialized radius.
 */
```

3.4.1.5 readFromFileExceptionTest

本节涉及到的测试，针对三个应用的读文件功能，在读文件时可能遇到的各种语法错误或其它不合法行为，都给出的异常报错，并以一种优雅的方式处理。本节主要测试在读取“精心设计”的非法文件时，会不会报出预期的异常错误。

3.4.1.5.1 AtomStructureTest

```
/*
 * Test strategy
 *  Exceptions that may happen in a atomstructure are
 *      FileNotFoundException, AtomElementException, AtomTrackNumException and AtomElectronNumException.
 *  FileNotFoundException
 *      Let atomStructure read file exceptionAssertionTestFile/what.txt which doesn't exist.
 *  AtomElementException
 *      1. Element of atom is Rbc in AtomicStructureAtomElementException.txt.
 *      2. Element of atom is rb in AtomicStructureAtomElementException (2).txt.
 *      3. Element of atom is RB in AtomicStructureAtomElementException (3).txt.
 *  AtomTrackNumException
 *      Number of track is 0 in AtomicStructureAtomTrackNumException.txt.
 *  AtomElectronNumException
 *      1. Some tracks have no matched electron number information in AtomicStructureAtomElectronNumException.txt.
 *      2. The number of electron information doesn't match the number of track in AtomicStructureAtomElectronNumException (2).txt.
 */
```

3.4.1.5.2 StellarSystemTest

```
/*
 * Test strategy
 *  Exceptions that may happen in a stellarSystem are
 *      FileNotFoundException, DataSyntaxException and DataScientificNumberException.
 *  FileNotFoundException
 *      Let stellarSystem read file exceptionAssertionTestFile/what.txt which doesn't exist.
 *  DataSyntaxException
 *      1. Star name is Sun^^ in StellarSystemDataSyntaxException.txt.
 *      2. Star radius is 6.96392Ae5 in StellarSystemDataSyntaxException (2).txt.
 *      3. Star mass is 1.9885Ae30 in StellarSystemDataSyntaxException (3).txt.
 *      4. A planet's name is Earth= in StellarSystemDataSyntaxException (4).txt.
 *      5. A planet's state is Soli-d in StellarSystemDataSyntaxException (5).txt.
 *      6. A planet's color is Bl**ue in StellarSystemDataSyntaxException (6).txt.
 *      7. A planet's radius is 6378._137 in StellarSystemDataSyntaxException (7).txt.
 *      8. A track's radius is 1·.49e8 in StellarSystemDataSyntaxException (8).txt.
 *      9. A revolution speed is 29./783 in StellarSystemDataSyntaxException (9).txt.
 *      10. A revolution direction is CWB in StellarSystemDataSyntaxException (10).txt.
 *      11. A original degree is 0+ in StellarSystemDataSyntaxException (11).txt.
 *  DataScientificNumberException
 *      1. A track's radius is 11.49e8 in StellarSystemDataScientificNumberException.txt.
 *      2. A track's radius is 1.49e2 in StellarSystemDataScientificNumberException (2).txt.
 */
```

3.4.1.5.3 SocialNetworkCircleTest

```
/*
 * Test strategy
 *  Exceptions that may happen in a socialNetworkCircle are
 *      FileNotFoundException, IllegalIntimacyInSocialTieException and DataSyntaxException.
 *  FileNotFoundException
 *      Let socialNetworkCircle read file exceptionAssertionTestFile/what.txt which doesn't exist.
 *  IllegalIntimacyInSocialTieException
 *      An intimacy is 0.0 in SocialNetworkCircleIllegalIntimacyInSocialTieException.txt.
 *  DataSyntaxException
 *      1. CentralUser name is Tommy-Wong in SocialNetworkCircleDataSyntaxException.txt.
 *      2. CentralUser age is 30.0 in SocialNetworkCircleDataSyntaxException (2).txt.
 *      3. CentralUser sex is m in SocialNetworkCircleDataSyntaxException (3).txt.
 *      4. A friend's name in SocialTie is Tommy-Wong in SocialNetworkCircleDataSyntaxException (4).txt.
 *      5. A friend's name in SocialTie is Lisa_Wong in SocialNetworkCircleDataSyntaxException (5).txt.
 *      6. An intimacy in SocialTie is 0.98321 in SocialNetworkCircleDataSyntaxException (6).txt.
 */
```

3.4.1.6 trackTest

这部分只有一个测试文件，就是用来测 Track 这个 ADT 中的各种方法的。Track 这个 ADT 和之前的 physicalObject 还有 centralObject 都是 IMMUTABLE 的类，所以测试方法的话也主要是测 Observer 方法。而且 Track 作为轨道 ADT，涉及它的操作也很少。

```
/*
 * Test strategy
 *  testGetRadius
 *      This tests getRadius.
 *      Because the radius of track is generated designative. Test if a track's radius is the same as the initialized radius.
 *  testEquals
 *      Any two tracks who have the same radius should be concerned the same.
 *      To see if equals works well, generate two tracks who have the same radius and assert they are equal.
 */
```

3.4.1.7 debug

对 debug 三个程序的测试，是遵从了测试优先的原则。先弄懂算法原理，然后根据原理来设计测试案例，每次修改一个 bug 就运行一次测试，不断调试，最终使测试通过。

3.4.1.7.1 FindMedianSortedArraysTest

```
/*
 * Test strategy
 *  The algorithm is based on dichotomy. The factors which affect algorithm are A.length and B.length,
 *  whether the sum of A.length and B.length is odd or even, that critical situation where A.length or B.length
 *  may be 1. Besides, whether A appended by B is strictly sorted may affect the result. And some elements in A or B
 *  may be equal, which probably affects the result.
 *
 *  testSameLengthStrictlySorted
 *      A and B have the same length and A appended by B is strictly sorted.
 *  testNotSameOddLengthStrictlySorted
 *      A and B have different length. The whole length is odd. A appended by B is strictly sorted.
 *  testNotSameEvenLengthStrictlySorted
 *      A and B have different length. The whole length is even. A appended by B is strictly sorted.
 *  testSameLengthNotStrictlySorted
 *      A and B have the same length and A appended by B is not strictly sorted.
 *  testNotSameOddLengthNotStrictlySorted
 *      A and B have different length. The whole length is odd. A appended by B is not strictly sorted.
 *  testNotSameEvenLengthNotStrictlySorted
 *      A and B have different length. The whole length is even. A appended by B is not strictly sorted.
 *  testSmallLength
 *      Analyzing variables when debugging, when index variable equals 0 or length - 1, it may cause critial situation
 *      and it can affect result. Therefore, let A.length = 1 and test.
 *  testEitherEqualElementEvenLength
 *      Elements in either A or B are all equal. The whole length is even.
 *  testEitherEqualElementOddLength
 *      Elements in either A or B are all equal. The whole length is odd.
 *  testBothEqualElementEvenLength
 *      Elements in both A and B are all equal. The whole length is even.
 *  testBothEqualsElementOddLength
 *      Elements in both A and B are all equal. The whole length is odd.
 *
 */
```

3.4.1.7.2 RemoveCommentsTest

```
/*
 * Test strategy
 *  Overall, the test strategy tests default cases that the example indicates in spec, a few equivalence cases,
 *  and a few assertion cases.
 *
 *  testDefault
 *      A case that example 1 indicates in spec.
 *  testDefault2
 *      A case that example 2 indicates in spec.
 *  testBlockNestInLine
 *      Block comment nests in line comment.
 *  testLineNestInBlock
 *      Line comment nests in block comment.
 *  testLineNestInLine
 *      Line comment nests in line comment. In fact, only the first // works as comment notation. The rests are all comments.
 *  testBlockNestInBlock
 *      Block comment nests in block comment. In fact, only the first pair of block comment notation works. The rests are all comments.
 *  testSourceLengthAssertion
 *      Test AssertionError when the length of source is greater than 100.
 *  testSourceContentLengthAssertion
 *      Test AssertionError when the length of source[i] is greater than 80.
 *  testDoubleQuoteCharacterAssertion
 *      Test AssertionError when double-quote shows in source[i].
 *  testSingleQuoteCharacterAssertion
 *      Test AssertionError when single-quote shows in source[i].
 *  testControlCharacterAssertion
 *      Test AssertionError when control character shows in source[i].
 *  testBlockClosedAssertion
 *      Test AssertionError when a block comment has no end notation.
 */
```

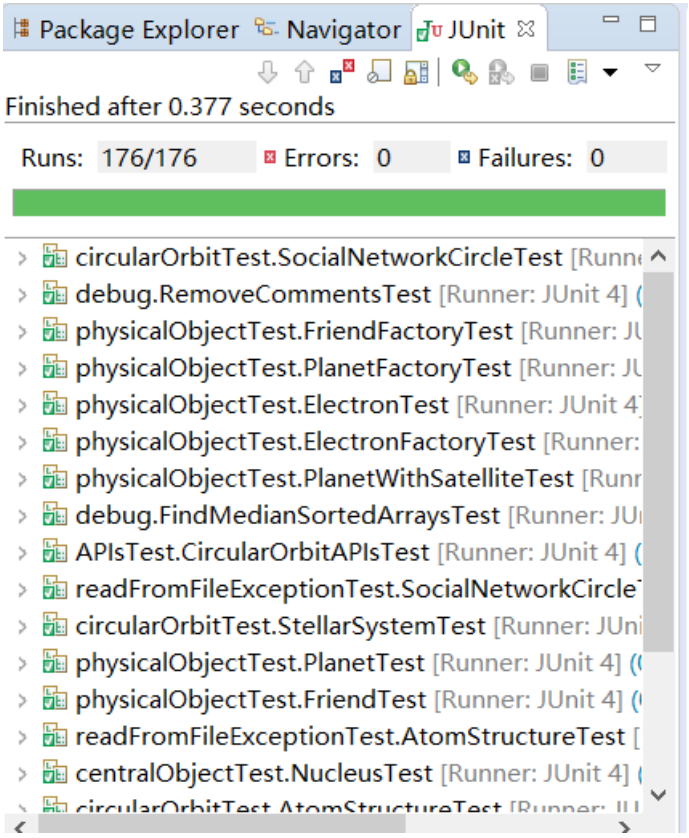
3.4.1.7.3 TopVotedCandidateTest

```
* Test strategy
*  The algorithm is based on dichotomy. The factors which may affect algorithm may be numbers of candidates,
*  order of votes, query time. The test strategy tests default cases that the example indicates in spec,
*  a few equivalence cases, and a few assertion cases.
*
*  testDefault
*      A case that example 1 indicates in spec.
*  testNonAlternateVotes
*      The order of votes is not alternate like the situation in default case.
*  testMultiCandidates
*      There are 4 candidates in persons. The order of votes is alternate.
*  testMultiCandidatesNonAlternateVotes
*      There are 4 candidates in persons. The order of votes is not alternate.
*  testCasualTimes
*      Vote times are casual not like the situation in default case.
*  testPersonsTimesNotSameLength
*      Test AssertionError when persons.length is not equal to times.length.
*  testPersonsLengthLessThanOne
*      Test AssertionError when persons.length is less than 1.
*  testTimesLengthGreaterThanFiveThousand
*      Test AssertionError when times.length is greater than 5000.
*  testPersoniLessThanZero
*      Test AssertionError when Persons[i] is less than 0.
*  testPersoniGreaterThanPersonLength
*      Test AssertionError when persons[i] is greater than persons.length.
*  testTimeiLessThanZero
*      Test AssertionError when times[i] is less than 0.
*  testTimeiGreaterThanABillon
*      Test AssertionError when times[i] is greater than 10e9.
*  testTimesNotStrictlyIncreasing
*      Test AssertionError when times is not a strictly increasing array.
*  testParameterTLessThanTimes0
*      Test AssertionError when parameter t in TopVotedCandidate.q(int t) is less than times[0].
```

3.4.2 测试用例设计

3.4.3 测试运行结果与 EclEmma 覆盖度报告

3.4.3.1 测试运行结果



3.4.3.2 Instruction Counters Coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Lab4-1170301007	60.9 %	9,867	6,339	16,206
src	48.9 %	5,267	5,497	10,764
applications	0.0 %	0	3,586	3,586
APIs	47.6 %	1,038	1,144	2,182
circularOrbit	87.3 %	2,712	393	3,105
MyException	10.0 %	24	216	240
physicalObject	86.3 %	402	64	466
atomTransitionMemento	84.9 %	236	42	278
centralObject	83.3 %	150	30	180
debug	97.5 %	635	16	651
track	92.1 %	70	6	76
test	84.5 %	4,600	842	5,442

3.4.3.3 Branch Counters Coverage

Element	Coverage	Covered Branches	Missed Branches	Total Branches
Lab4-1170301007	57.1 %	647	486	1,133
src	57.6 %	590	435	1,025
applications	0.0 %	0	164	164
APIs	49.8 %	105	106	211
circularOrbit	78.9 %	292	78	370
physicalObject	53.6 %	45	39	84
debug	83.8 %	109	21	130
centralObject	47.2 %	17	19	36
track	58.3 %	7	5	12
atomTransitionMemento	83.3 %	15	3	18
MyException		0	0	0
test	52.8 %	57	51	108

3.4.3.4 Complexity Coverage

Element	Coverage	Covered Complexity	Missed Complexity	Total Complexity
Lab4-1170301007	54.3 %	584	491	1,075
src	47.2 %	378	422	800
applications	0.0 %	0	122	122
circularOrbit	69.1 %	179	80	259
APIs	41.4 %	55	78	133
MyException	10.0 %	6	54	60
physicalObject	55.2 %	48	39	87
debug	71.6 %	53	21	74
centralObject	37.5 %	12	20	32
track	64.3 %	9	5	14
atomTransitionMemento	84.2 %	16	3	19
test	74.9 %	206	69	275

3.5 SpotBugs tool

3.5.1 Random object created and used only once

Bug: Random object created and used only once in new physicalObject.Friend(String, int, char)

This code creates a java.util.Random object, uses it to generate one random number, and then discards the Random object. This produces mediocre quality random numbers and is inefficient. If possible, rewrite the code so that the Random object is created once and saved, and each time a new random number is required invoke a method on the existing Random object to obtain it.

If it is important that the generated Random numbers not be guessable, you *must* not create a new Random for each random number; the values are too easily guessable. You should strongly consider using a java.security.SecureRandom instead (and avoid allocating a new SecureRandom for each random number needed).

Rank: Troubling (14), **confidence**: High
Pattern: DMI_RANDOM_USED_ONLY_ONCE
Type: DMI, **Category**: BAD_PRACTICE (Bad practice)

```
public Friend(String name, int age, char sex) {
    this.name = name;
    this.age = age;
    this.sex = sex;
    this.degree = new Random().nextDouble() * 360;
    checkRep();
}
```

该 bug 指出，如果要用 new Randon()来生成随机数的话，最好要把 Random 对象保存到一个变量上，每次调用这个变量来生成。如果每次生成随机数都 new Random()的话，可能会影响随机数的随机性。而且为了提高随机数的随机性，建议使用 SecureRandom 类。这里随机数的随机性不重要，所以将代码修改为：

3.5.2 equals() method does not check for null argument

Bug: centralObject.Person.equals(Object) does not check for null argument

This implementation of equals(Object) violates the contract defined by java.lang.Object.equals() because it does not check for null being passed as the argument. All equals() methods should return false if passed a null value.

Rank: Troubling (11), **confidence**: Normal
Pattern:
NP_EQUALS_SHOULD_HANDLE_NULL_ARGUMENT
Type: NP, **Category**: BAD_PRACTICE (Bad practice)

```
@Override
public boolean equals(Object person) {
    return person.getClass() == Person.class && this.name.equals(((Person) person).name);
}
```

这个 bug 在我的代码中有很多，在重写 equals 时，没有考虑到传参为 null 的情况，这是很容易被忽略的。这样的忽略违反了 java.lang.Object.equals()的规范，所以，将所有的重写 equals 都加上对参数为 null 的检查。

```
@Override
public boolean equals(Object person) {
    return person != null && person.getClass() == Person.class && this.name.equals(((Person) person).name);
}
```

3.5.3 Possible null pointer dereference of atomStructureHandler in applications.Main.main(String[]) on exception path

Bug: Possible null pointer dereference of atomStructureHandler in applications.Main.main(String[]) on exception path

A reference value which is null on some exception control path is dereferenced here. This may lead to a `NullPointerException` when the code is executed. Note that because SpotBugs currently does not prune infeasible exception paths, this may be a false warning.

Also note that SpotBugs considers the default case of a switch statement to be an exception path, since the default case is often infeasible.

Rank: Troubling (11), **confidence**: Normal
Pattern: NP_NULL_ON_SOME_PATH_EXCEPTION
Type: NP, **Category**: CORRECTNESS (Correctness)

```
atomStructureLogger.setLevel(Level.INFO);
stellarSystemLogger.setLevel(Level.INFO);
socialNetworkCircleLogger.setLevel(Level.INFO);
FileHandler atomStructureHandler = null, stellarSystemHandler = nu
try {
    atomStructureHandler = new FileHandler("log/atomStructureLog.l
    stellarSystemHandler = new FileHandler("log/stellarSystemLog.l
    socialNetworkCircleHandler = new FileHandler("log/socialNetwor
} catch (SecurityException e) {
    e.printStackTrace();
} catch (IOException e) {
    e.printStackTrace();
}
atomStructureHandler.setFormatter(new SimpleFormatter());
stellarSystemHandler.setFormatter(new SimpleFormatter());
socialNetworkCircleHandler.setFormatter(new SimpleFormatter());
atomStructureLogger.addHandler(atomStructureHandler);
```

这里有连续三行 bug。当上面的 try-catch 语句发生异常时，atomStructureHandler, stellarSystemHandler 和 socialNetworkCircleHandler 将成为指向 null 的变量。此时，存在 bug 的三行代码将出错。这也是容易被忽略的 bug，因为之前发生了比较严重的异常，应该在 catch 中结束当前方法，避免继续执行接下来的代码。

```
stellarSystemLogger.setLevel(Level.INFO);
socialNetworkCircleLogger.setLevel(Level.INFO);
FileHandler atomStructureHandler = null, stellarSystemHandler = nu
try {
    atomStructureHandler = new FileHandler("log/atomStructureLog.
    stellarSystemHandler = new FileHandler("log/stellarSystemLog.
    socialNetworkCircleHandler = new FileHandler("log/socialNetwor
} catch (SecurityException e) {
    e.printStackTrace();
    return;
} catch (IOException e) {
    e.printStackTrace();
    return;
}
atomStructureHandler.setFormatter(new SimpleFormatter());
stellarSystemHandler.setFormatter(new SimpleFormatter());
socialNetworkCircleHandler.setFormatter(new SimpleFormatter());
atomStructureLogger.addHandler(atomStructureHandler);
```

3.5.4 Dereference of the result of readLine() without nullcheck in APIs.CircularOrbitAPIs.logSearch(File)

Bug: Dereference of the result of readLine() without nullcheck in APIs.CircularOrbitAPIs.logSearch(File)

The result of invoking readLine() is dereferenced without checking to see if the result is null. If there are no more lines of text to read, readLine() will return null and dereferencing that will generate a null pointer exception.

Rank: Of Concern (15), **confidence**: Normal
Pattern: NP_DEREFERENCE_OF_READLINE_VALUE
Type: NP, **Category**: STYLE (Dodgy code)

```
case 2:
    System.out.println("Please input log level(Info/Warning/Severe):");
    String logLevel = in.nextLine();
    try {
        while ((line1 = br.readLine()) != null) {
            line2 = br.readLine();
            String logLevelInfo = line2.substring(0, line2.indexOf(":"));
            if (logLevelInfo.equalsIgnoreCase(logLevel)) {
                System.out.println(line1);
                System.out.println(line2);
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    break;
```

这个 bug 是这样的。因为 line2 是一个读文件的 String 变量，在没有检查它是不是 null 的情况下就调用 line2 的操作，如果它是 null，那么这里会报错。这个 bug 的导致原因是逻辑疏忽，因为在这里 line1 和 line2 都是从 log 日志中读取的，在我的 log 日志中，日志信息的存储格式都是每条日志信息是两行，所以我默认日志中有偶数行，也就是说，line1 不为 null 的情况下，line2 按理来说也不会是 null，但 spotbugs 不知道这些事情，所以为了保证万无一失，还是对 line2 做一个 null 检查。

```
System.out.println("Please input log level(Info/Warning/Severe):");
String logLevel = in.nextLine();
try {
    while ((line1 = br.readLine()) != null) {
        line2 = br.readLine();
        if (line2 == null) {
            System.out.println("Abort log reading. The number of lines :
            break;
        }
        String logLevelInfo = line2.substring(0, line2.indexOf(":"));
        if (logLevelInfo.equalsIgnoreCase(logLevel)) {
            System.out.println(line1);
            System.out.println(line2);
        }
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

3.5.5 Check for oddness that won't work for negative numbers

> **Bug**: Check for oddness that won't work for negative numbers in debug.FindMedianSortedArrays.findMedianSortedArrays (int[], int[])
>
> The code uses x % 2 == 1 to check to see if a value is odd, but this won't work for negative numbers (e.g., (-5) % 2 == -1). If this code is intending to check for oddness, consider using x & 1 == 1, or x % 2 != 0.
>
> **Rank**: Troubling (13), **confidence**: Normal
> **Pattern**: IM_BAD_CHECK_FOR_ODD
> **Type**: IM, **Category**: STYLE (Dodgy code)

在修改 bug 时，第一个 bug 程序中有一个要判断 A 和 B 长度总和奇偶性的检验，如果按如下的方式检验：

```
    }
    if ((m + n) % 2 == 1) { // if((m + n + 1) % 2 == 1)
        return maxLeft;
    }
```

Spotbugs 会提示这种对奇偶性的校验方式，对负数是无效的。建议改为与 1 做位与运算判断是否为 1 来检验奇偶性，或者对 2 取余是否为 0.故修改为：

```
    if ((m + n) % 2 != 0) { // if((m + n + 1) % 2 == 1)
        return maxLeft;
    }
```

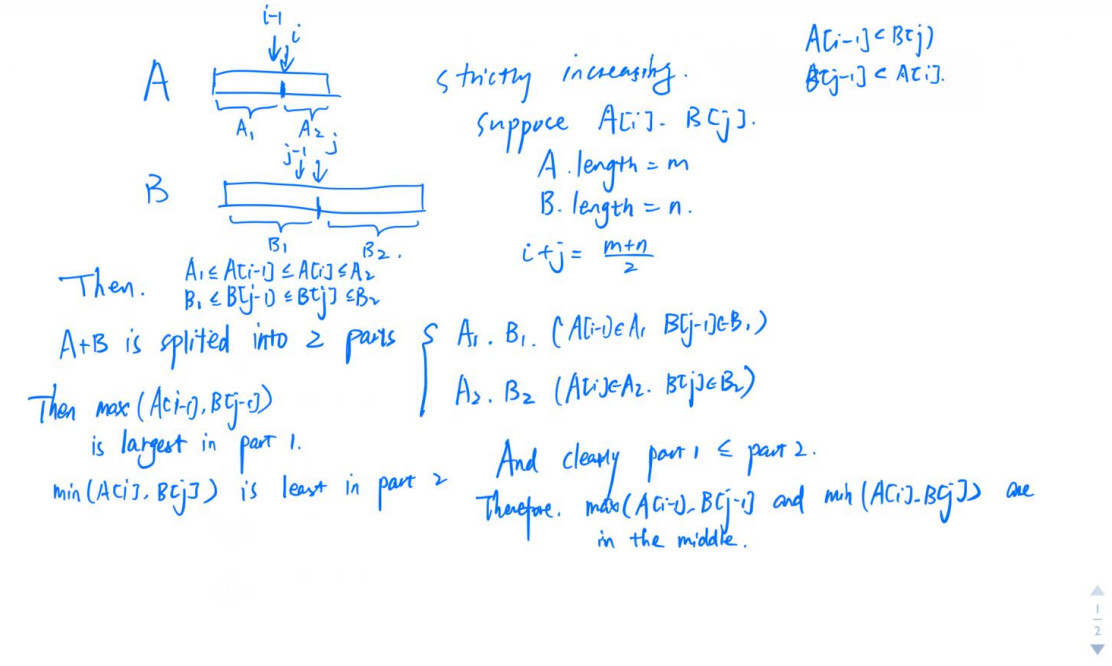
3.6 Debugging

3.6.1 理解待调试程序的代码思想

这三个程序真的是让人很头疼。注释不写，变量名不规范，第三个程序的 spec 写的也是一塌糊涂，顿时让人摸不着头脑。好在第一题是典型的算法题，第二题的逻辑不是很难懂，第三题需要花费很长时间才看懂。

3.6.1.1 FindMedianSortedArrays

该程序完整的写了一个算法，用来求两个严格有序的数组的中位数。该算法的思想是对这两个严格有序的数组 A 和 B 分别定界 i 和 j，记 A 长度为 m，B 长度为 n，应该满足，i+j = (m+n)/2，也就是说，A 和 B 的所有元素被分成了两部分，A 中索引小于 i 的加上 B 中索引小于 j 的元素组成了一部分，A 中索引大于等于 i 的加上 B 中索引大于等于 j 的元素组成了另一部分。如果取 A[i]和 B[j]中的较大者，那么这个较大者应该是所有元素中，较小的那一半的最大者，而 A[i]和 B[j]的较小者，则是所有元素中，较大的那一半的最小者。那么所有元素的中位数，在元素总和为偶数的情况下，便是 max(A[i-1],B[j-1])和 min(A[i],B[j])的算术平均值，如果是奇数个，那么也就是这两者其中之一了。



3.6.1.2 RemoveComments

这个算法是用来删除一段文本中的合法注释的。该算法的输入是一个字符数组，字符数组中的每个元素应该是一整行文本。该算法逻辑比较简单，就是解析字符数组中的每一个元素（每行文本），用 if-else 结构来判断是否为注释。如果

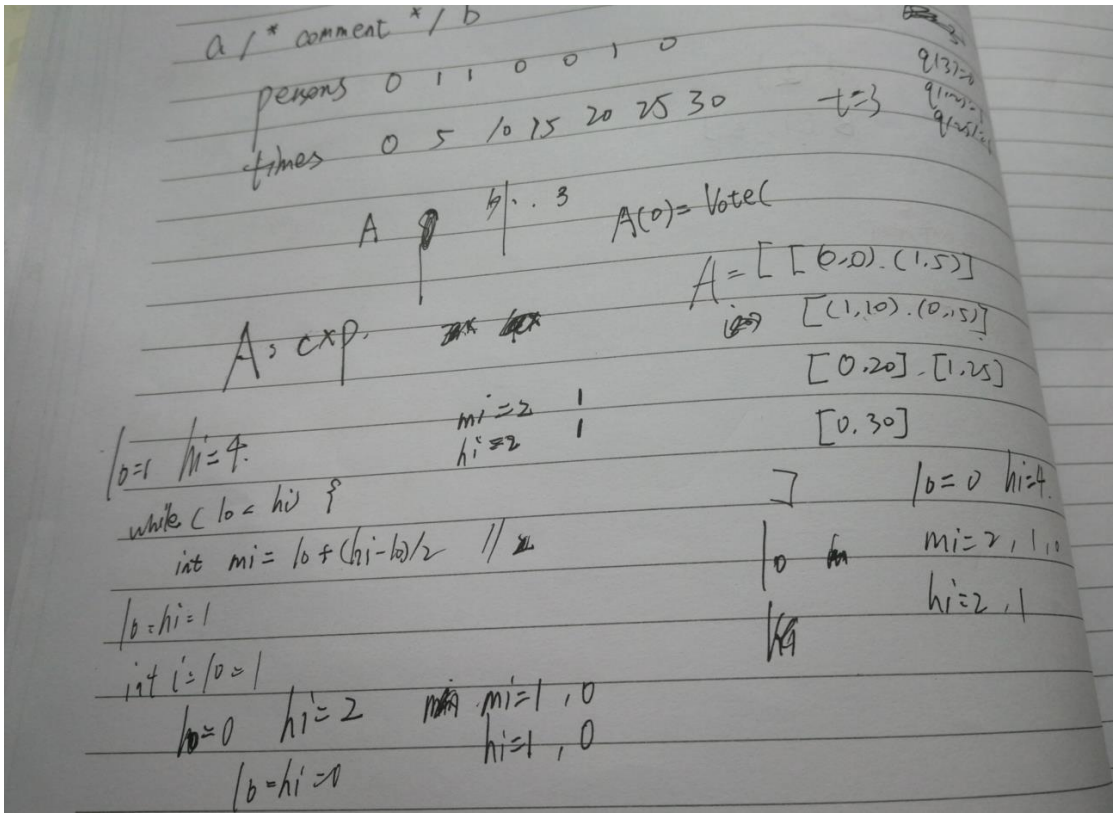
判定是注释，那就加入结果 ans，如果不是，那就解析下一行。

3.6.1.3 TopVotedCandidate

这个算法简直让人头大。首先 specification 就让人读不懂，嚼了一晚上代码，逻辑算是看懂了，但是 specification 还是一头雾水。直到刚才 piazza 上有人给解释，原来 spec 中的 input 的第一行是这个意思：

额，一个调用构造器，后边六次调用q函数，下边是参数，然后结果

在理解逻辑的过程中，我动用了 system.out 监视大法、debug 变量调试窗，最后发现还是没有人工 debug 好用。把所有变量写在纸上，把默认测试用例带到算法中，在纸上跑一边算法过程。



这个算法分为两部分，一部分是构造器，用来将输入构造成一个存有所有信息的二维数组变量 A，第二部分是方法 p，用来求结果。最难理解的是 A 的 shape 的具体含义。按照算法，A 是一个 c×p 的二维数组，c 是 candidate 最多的票数，而 p 是 candidates 人数。那么按照逻辑，如果投票时间是严格的递增序列，那么 A 中的每一个分量，也都是按序的投票结果，而且如果投票是交替的话，那么就会出现上图中右侧 A 的结构，如果不是交替的，那么在构建 A 的过程中，就会出现 A 的第一个纵分量，要比其它纵分量长度至少大 2 的时候。在最后的 A 结果中，如果要判断哪个人胜出，就看 A 的最大横分量中的那个票是谁（如果有多个票，看时间最近的），在某一时刻 t 要看谁胜出，就先在第一个纵分量上，找到比 t 小的时刻，然后就确定了一个横分量，在该横分量上再找比 t 的小的时刻，那么那个投票的结果，就是胜出结果了。

这个算法之所以正确，是因为，如果胜出者比失败者的票数至少大于 2，那

么对于我们找到的那个横分量来说，失败者的投票都不会出现在那个横分量里，甚至在之前的几个横分量中，失败者的投票都不会出现。而如果胜出者和失败者的票数相同，那么虽然二者的投票都会出现在那个找到的横分量中，但是由于 list 具有添加元素放在尾部的特点，也就是说，索引越大的投票，时间越晚，如果在第二个循环中从横分量的尾部开始逐步向前确定，那么最后确定到的那个投票结果，就是 t 时刻的胜出结果。

3.6.2 发现并定位错误的过程

3.6.2.1 FindMedianSortedArrays

首先阅读代码结构，一个明显的错误就是在判断 A 和 B 总长度奇偶性的时候就判断反了。在总长度为奇数时，只需要返回中间那个数就好了，这里面中间那个数就是较小的一部分中的最大值，maxLeft.

```
if ((m + n) % 2 != 0) { // if((m + n + 1) % 2 == 1)
    return maxLeft;
}
```

分析代码逻辑和算法正确性，结构应该是没问题的，错就错在了一些索引算错了。iMax 和 iMin 是用来定位 A 数组中的 i 索引，而 B 中的 j 索引，要用 A 和 B 的长度之和的一半来减去 i。分析对 i 的定位判断过程

```
if (i < iMax && B[j - 1] > A[i]) {
    iMin = i + 1;
} else if (i > iMin && A[i - 1] > B[j]) {
    iMax = i - 1;
}
/* iMin = iMax = i or all between i and j are equal*/
else {
    int maxLeft = 0;
```

如果 B[j-1]>A[i]的话，说明 A[i]太小了，得再往右定位，所以 iMin=i+1,如果 A[i-1]>B[j]的话，说明 A[i]太大了，得再往左定位，所以 iMax=i-1.

如果在若干次循环之后，iMax=iMin，也就是说 i 被定位，则顺利进入第三个分支求解结果，如果当 iMax!=iMin,但也进入到第三个分支，只能是前两个判断的后一个条件没有满足，那么在 A 中从 i 往右和 B 中的从 j 往左的元素都是相等的，在这种找中位数的话，也没必要作区分了。

接下来继续找 bug，对于 halfLen 的初始化，如果 A=3，B=4,那么 halfLen 到底是 3 还是 4 有待决定，下面在返回时（见第一个 bug 修复截图），当总长为奇数时，返回了 maxLeft，也就是返回了较小部分的最大值，由此可以判定，如果总长为奇数，那么较小部分的应该是多一个的。所以，halfLen 的赋值应该改为(1+m+n)/2。

```
int iMin = 0, iMax = m, halfLen = (1 + m + n) / 2; // halfLen = (m + n) / 2
```

再跑测试案例发现全部通过了。但是注意到对 i 的定界赋值，在下面的代码逻辑中有 A[i]的取值，容易举出一个例子，iMin=0,iMax=1,m=1 时，i 的定界会被置 1，如果这个时候执行到 A[i]的话，就会报 indexOutOfRange 的错误，不能把程序的正确性赌在会不会执行到 A[i]上，还是老实把这一行改掉。

```
int i = (iMin + iMax) / 2; // int i = (iMin + iMax + 1) / 2
```

3.6.2.2 RemoveComments

这个 debug 程序没用太多手段就做出来了。第一个最扎眼的错误就是 List 的初始化，修改之：

```
List<String> ans = new ArrayList<String>(); // List<String> ans = new List();
```

然后跑测试代码，竟然报错了，indexOutOfRange 异常，这是因为当在判断有无/* notation 时，对 i 的长度判断不对，修改之，

```
if (!inBlock && i + 1 < line.length() && chars[i] == '/' && chars[i + 1] == '*') { // && i+1 <=
                                                                                    // line.length()
} else if (inBlock && i + 1 < line.length() && chars[i] == '*' && chars[i + 1] == '/') { // && i + 1 <
                                                                                         // line.length()
```

再跑测试代码，失败，从测试代码中发现错误，发现该跑出结果的测试案例，system.out 打印结果是空值，再回去分析逻辑，发现一个 boolean 判断反了，应该是不在注释块内的文本加入到 ans 中。

```
if (!inBlock && newline.length() > 0) { // inBlock &&
    ans.add(new String(newline));
}
```

再跑，还是失败。发现很多这样的错误：

```
java.lang.AssertionError: expected:<[j, kk]> but was:<[/j, kk]>
```

在纸上跑一边逻辑，发现这个错误是因为，在判断了 block comments 的结束条件时，遇到*就将 inBlock 置为 false 了，紧接着再读下一个/的时候，认为它是在注释外的字符，也加入到 ans 中了，这是不正确的，所以这些地方都应该加上 i+=2 的跳跃代码，并加上 continue，拒绝执行 if-else 之后的 i++。

```
if (!inBlock && i + 1 < line.length() && chars[i] == '/' && chars[i + 1] == '*') { // && i+1 <=
                                                                                   // line.length()
    inBlock = true;
    i += 2; // new
    continue; // new
} else if (inBlock && i + 1 < line.length() && chars[i] == '*' && chars[i + 1] == '/') { // && i + 1 <
                                                                                        // line.length()
    inBlock = false;
    i += 2; // new
    continue; // new
```

再跑测试代码，发现一些简单的案例能过，但是一些精心设计的测试过不了，分析代码逻辑，显然，原 bug 程序只考虑了 block comments 而没有考虑 line comments。加上逻辑之后

```
} else if (!inBlock && i + 1 < line.length() && chars[i] == '/' && chars[i + 1] == '/') { // this is a
                                                                                         // new else
                                                                                         // if
    break;
```

再跑，测试案例全部通过，认为 debug 任务完成。
最后再加上一些限定 pre-condition 的 assert 语句吧。

3.6.2.3 TopVotedCandidate

在理解了代码逻辑之后，很容易修改这里的 bug 了。代码逻辑分析见 3.6.1.3.
首先是两个扎眼的 bug，list 和 map 的初始化。

```
A = new ArrayList<List<Vote>>(); // new ArrayList();
Map<Integer, Integer> count = new HashMap<Integer, Integer>(); // new HashMap();
```

然后是逻辑 bug，c 是候选人 p 的当前票数，当第一次遇到该候选人的时候，map 中没有键值 p，那么默认值应该是 0，而不是 1.

```
int c = count.getOrDefault(p, 0); // count.getOrDefault(p, 1);
```

读到候选人 p 的一票之后，应该增加其 value 值。

```
count.put(p, c + 1); // count.put(p, c);
```

构造器中的 bug 就这么多，然后是 q 函数中的 bug。在定位 A 的符合要求的横纵向量时用的是二分法，这样定位更快。但是对索引的判断会很模糊。首先容易发现 q 方法中的两个对 t 的判断不太一致，应该都是小于等于。

```
if (A.get(i - 1).get(mi).time <= t) // A.get(i).get(mi).time < t
```

然后有一个更新 lo 值也和另一个不一样，至于哪个需要改还需要进一步判断。在纸上跑一边默认测试案例的时候，可以得出，第一次循环结束后，将 lo 赋给 i，i 就是我们定位得到的横向量，但是这个 i 其实是第 i 行，要在 A 中取值的话，应该是 i-1 才行。

```
hi = A.get(i - 1).size(); // hi = A.get(i).size();
```

在接下来的取值过程中都对此修改。

```
if (A.get(i - 1).get(mi).time <= t) // A.get(i).get(mi).time < t
    lo = mi + 1;
```

还有第二次循环结束后的 j 赋值，把 lo 赋给 j，也是第 j 行，需要改为 j-1. 而且这里还有一个明显的错误，lo 本身是一个正整数，用 lo 和 0 取最大值，显然还是 lo 本身。

```
int j = lo; // int j = Math.max(lo, 0);
return A.get(i - 1).get(j - 1).person; // A.get(i).get(j).person;
```

至于刚才还没有决定哪个循环中的 lo 更新要修改，跑一边测试代码，并用 debug 工具跟踪变量，就可以决定了，在定位横向量的那个循环里面的要改。

```
while (lo < hi) {
    int mi = lo + (hi - lo) / 2;
    if (A.get(mi).get(0).time <= t)
        lo = mi + 1; // lo = mi
    else
        hi = mi;
}
```

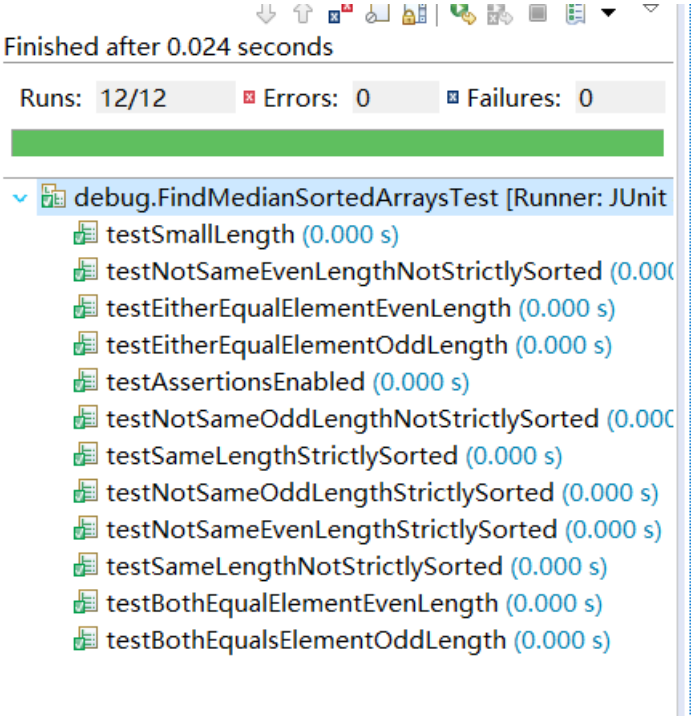
修改后跑测试案例，全部通过。
最后再用 assert 测一下 pre-condition。

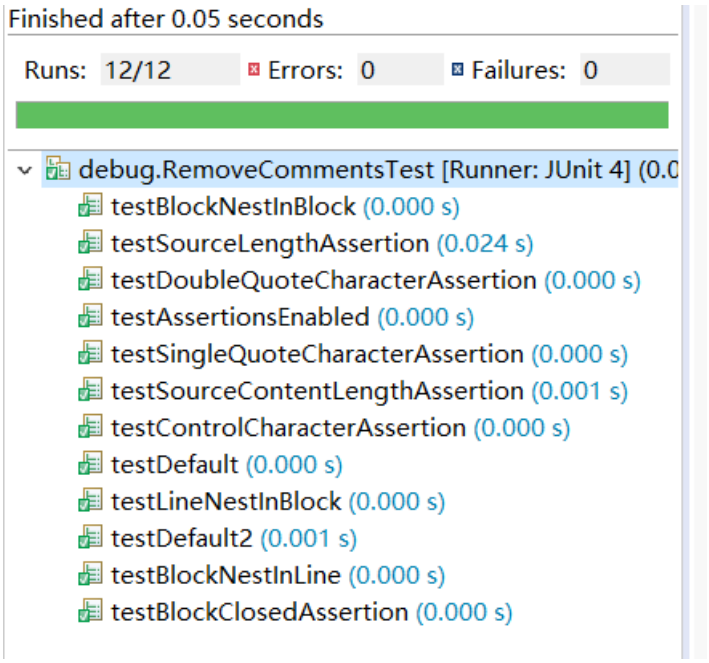
3.6.3 如何修正错误

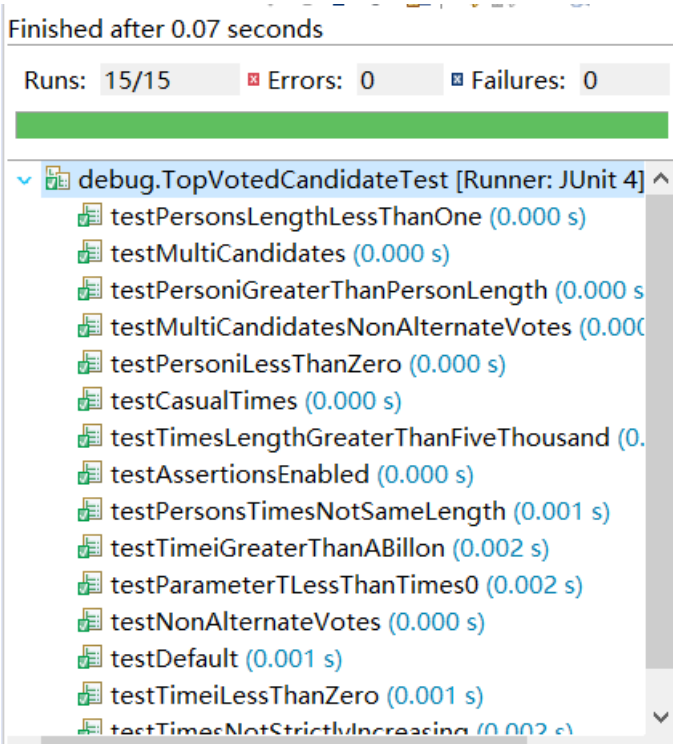
见 3.6.2.

3.6.4 结果

测试案例全部通过。







4 实验进度记录

日期	时间段	计划任务	实际完成情况
2019.5.14	13:30:18	Finish exception. No test file commit.	
2019.5.16	00:09:27	Implement logging. Using SpotBugs tool to fix a few bugs.	
2019.5.16	23:46:46	Implement exception file test. Add	

		defense for System.in.	
2019.5.18	00:08:47	Implement debug and debug test.	
2019.5.19	21:00	Implement pre/post-condition assertion. Finish report.	

5 实验过程中遇到的困难与解决途径

遇到的难点	解决途径
印象最深刻的是测试异常，因为我在设计异常的 try-catch 策略的时候，认为凡是能现场处理的就现场处理，不要把垃圾抛给上一级，可是发现现场处理的异常是测不到的，需要测一些其他的间接测试，这样很麻烦	把一写相对比较严重的异常向上一级抛出，这样测试也比较简单。把一些不那么严重的，现场解决的不必终止文件读入的，也就是可以允许的不合法的输入，默认将其以合情合理方式转成合法的，给用户提示信息。
做 log 日志的时候，用的是 java.util.logging，但是同一个 logger 要在不同的类中调用	只在第一级应用层中初始化所有 logger 和 filehandler，剩下的被调用的应用类都是只声明一个全局的 logger，并使这个 logger 的名称和第一级初始化的相同，这样这些类中的 logger 也都是第一级初始化的 logger。

6 实验过程中收获的经验、教训、感想

6.1 实验过程中收获的经验和教训

学会了自定义异常类的方法和策略，异常的处理方式，什么时候向上抛，什么时候现场处理，什么时候抛出 assertion error，但这次实验中应该没有 unchecked 异常。学会了使用最简单的 java.util.logging。而且本次实验，是我第一次使用 Java 内置的 debug 变量监视窗，感觉还是没有人工变量监视窗好用。至于覆盖度报告，也就是会生成了，也用不上。

6.2 针对以下方面的感受

(1) 健壮性和正确性，二者对编程中程序员的思路有什么不同的影响？

正确性是指导程序员编程的首要原则。只有 spec 中规定清晰的正确性，**程序员在编程时才有信心，“我能交付什么样的符合要求的代码”。**

健壮性是让一个程序员膨胀和头秃的次要因素。健壮性考虑的越多，程序员越头秃，但交付代码时程序员可以很膨胀地说，“这代码稳了”。

用所学的知识说，正确性是站在程序员自己的角度，健壮性是从用户的角度考虑的。

(2) 为了应对 1%可能出现的错误或异常，需要增加很多行的代码，这是否划算？（考虑这个反例：民航飞机上为何不安装降落伞？）

还是得看到底增大多少工作量吧，同时还得考虑这错误和异常严不严重。民航飞机那个问题我之前还真查过，就是成本高呗。但是要说是重要人物专机，那肯定得配最好的降落伞了，不管成本代价。

(3) “让自己的程序能应对更多的异常情况”和“让客户端/程序的用户承担确保正确性的职责”，二者有什么差异？你在哪些编程场景下会考虑遵循前者、在哪些场景下考虑遵循后者？

前者意味着程序员要付出更大的成本代价，后者虽然开发代价小，但是用户体验差。企业级的软件一般都应该是前者。而不那么重要的软件，一般会选择后者。

(4) 过分谨慎的“防御”（excessively defensive）真的有必要吗？如果你在完成 Lab5 的时候发现 Lab5 追求的是 I/O 大文件时的性能（时间/空间），你是否会回过头来修改你在 Lab3 和本实验里所做的各类 defensive 措施？如何在二者之间取得平衡？

我觉得必要与否得看场景。如果追求性能，那么这肯定是不必要甚至是不应该有的。具体怎么取舍，看甲方要求吧。Lab4 甲方说做防御，lab5 甲方说做性能，乙方没得选啊。

(5) 通过调试发现并定位错误，你自己的编程经历中有总结出一些有效的方法吗？请分享之。Assertion 和 log 技术是否会帮助你更有效的定位错误？

我觉得最有效的方法，就是 system.out 监视大法+人工变量监视窗…清晰明了，程序运行的历史跃然纸上，清晰可见…。Assertion 能定位错误，这个在 lab3 中就体会到了，至于 log，那得看这 log 质量写的怎么样了。

(6) 怎么才是“充分的测试”？代码覆盖度 100%是否就意味着 100%充分的测试？

否。正如 piazza 上老实的回答，覆盖度只是一个参考值，参考测试是否充分。如果我的程序逻辑不对，而测试的等价类划分也不合理，即使做

到了 100%覆盖度，那程序也是错的。

(7) Debug 一个错误的程序，有乐趣吗？

呵呵。没事的时候 debug 有乐趣，忙的时候 debug 这种没有注释和 spec 的代码，毫无乐趣。甲方给钱（分）也有乐趣，大量的乐趣。

(8) 关于本实验的工作量、难度、deadline。

本实验是基于 lab3 的，如果 lab3 做的好的话，这次工作量还不算大，但是工作很繁琐，重复的地方很多，很烦躁。难度有点难，有些地方规范还是没讲清楚，仅凭课上的学到的是不够的，我觉得软构这课本来就是讲软件开发中的一些规范，但是规范还是不够明确啊。Deadline 还行。

(9) 到目前为止你对《软件构造》课程的评价和建议。

规范不够明确，如果给一个明确的模板，再给出不规范的例子，我们会更清楚一些。

(10)期末考试临近，你对占成绩 60%的闭卷考试有什么期望或建议？//请严肃的提出，杜绝开玩笑，教师会认真考虑你们的建议。

对分数占比没意见。至于闭卷，我觉得这门课的知识覆盖面非常广，前所未有，比计算机系统设计的到知识还多，而闭卷考试的话，就考那么点东西，我们要复习的话肯定是要争取全面复习的，这复习量是不是太大了一些？这么多知识，一学期学过后考完之后，时间长了也就忘了。我意思是说，这门课中的很多东西就是熟能生巧的，也许有些地方开卷会好一些。毕竟以后我们在做软件的时候，有些地方真不知道了，上网搜一搜也就知道了，熟能生巧吧，就光这一学期的训练，还是远远不够熟练的，而考试的内容又不能只考实验中的内容吧。

综上，建议合理控制考试内容范围，主要考重点，范围不要太广，平衡开闭卷的考试难度。如果能缩小考试分数占比，对我们来说也是一件好事。