面向对象第三单元博客作业

很快来到了第三次OO作业的最后一周，不得不感叹时间真是过的飞快（尤其在写OO过程中）。在这一周我们主要完成了给定JML规格编写Path类和PathContainer类/Graph类/RailwaySystem类，并完成基于两个类上的各种增删查改功能。在PathContainer类中主要涉及的查询是图属性的查询（例如节点个数、路径条数、ID等），在Graph类中扩展出了边之间邻接关系的查询和最短路径的查询，RailwaySystem中主要涉及了有权图最短距离的查询，其中解决换乘点问题该次作业的一大难点。三次作业难度递增，依次扩展。为了性能的优化，不仅需要建立以上的类，在更高的难度上（例如Graph类/RailSystem类）的相关功能实现上我们还需要建立缓存类。对一些中间结果进行保存以减少查询次数和难度。从这几次作业中我能够体会到JML语言形成的规格给我们编写程序带来的好处以及特点；在性能优化的部分我也能体会到在程序开发中性能与功能性的折中与平衡。

本次的博客将就以下的几个部分对OO第三次作业进行总结：JML语言的理论基础和应用工具链情况、SMT solver的部署与方法验证、JML UnitNG/JML Unit的简要分析、我的架构设计及重构情况、代码实现的bug和修复情况，规格撰写和以及心得体会和我对整个作业的感想。

1. JML语言的理论基础及应用工具链情况
2. JML语言的理论基础

JML（Java Modeling Language）是用于对Java程序进行规格化设计的一种表示语言。JML是一种行为接口规格语言（Behavior Interface Specification Language, BISL），基于Larch方法构建。BISL提供了对方法和类型的规格定义手段。所谓接口即一个方法或者类型外部可见的内容。JML主要有Leavens教授在Larch上的工作，并融入了Betrand了一套行之有效的方法。通过JML及其支持工具，不仅可以基于规格自动构造测试用例，并整合了SMT Solver等工具以静态方式来检查代码实现对规格的满足情况。

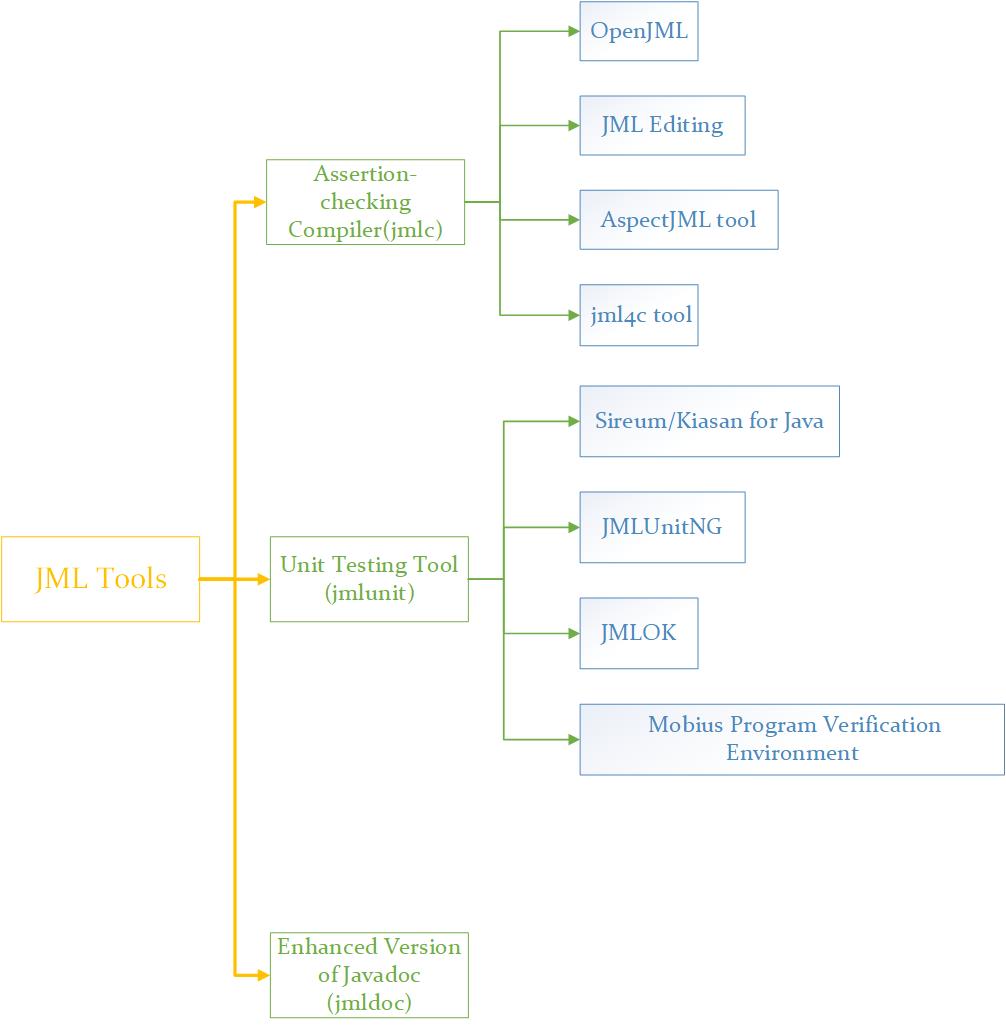
一般而言，JML主要有两种用法：

1. 开展规格化设计。这样交给代码实现人员的将不再是可能带有内在模糊性的自然语言描述，而是逻辑严格的规格。
2. 针对已有的代码实现，书写其对应的规格，从而提高代码的可维护性。这在遗留代码的维护方面具有特别重要的意义。

{以上内容节选自OO课程组JML（Level 0）使用手册}

1. JML的应用工具链情况

根据JML官网上面的信息，我们可以将JML的应用工具链总结为以下一张图。

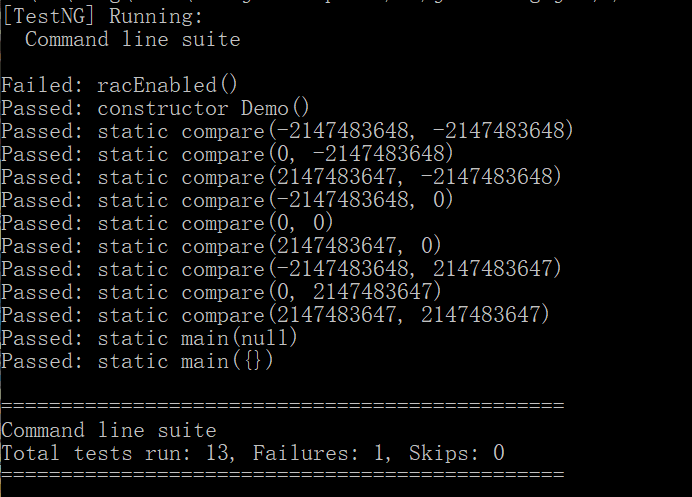


1. SMT solver部署与方法验证（选做）

按照一些博客尝试部署了一下，但是部署不出来。由于选做遂放弃。

1. JML UnitNG/JML Unit的部署和简要分析
2. 部署JML UnitNG
3. 针对JML UnitNG展开的分析

使用了path.java文件进行了分析，具体的命令不再展开，得到的编译结果如下：



说明我们的文件已经通过了规格检查。

我发现在实际应用中JML UnitNG在网上的资料并不多，不知道是不是因为使用频率过少。并且根据实测，大部分的情况都难以获得很好的检查效果。例如带构造函数的方法，带有一些简单JML语法的方法（\forall，\exists等）。在此比较质疑工具的实用性。因此本次的实验也并没有深入展开，留待后续在实际开发中继续探讨此工具的应用场景。

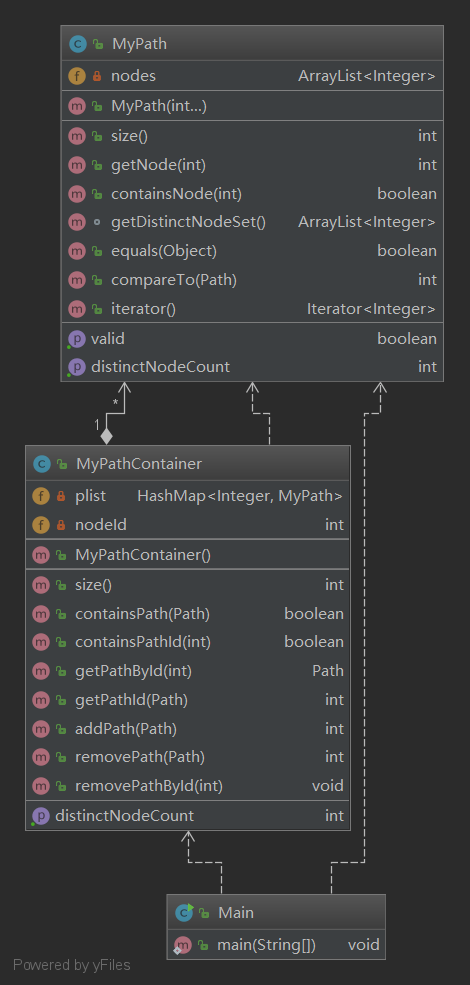
1. 我的架构设计及重构情况
2. 第一次作业——路径查询系统

·作业要求

本次作业需要实现一个简单的路径查询系统。要求我们根据JML规格来编写代码，能够完成路径的基本简单查询。这次作业非常简单不涉及任何高深的数据结构及算法。但是就因为他规格太过于简单，我就没有考虑其性能问题。而这个问题最坑的地方就是其优化问题。因为完全没有优化，仅仅保证了其正确性，没有保证其性能足够通过强测，导致我第一次强测仅仅得到了50分，这部分优化将在“Bug分析与修复”中详细介绍。

·本次作业的架构

因为本次作业非常简单，实际上看到规格基本上就可以大致写出来。这里只列举本次作业的类图。



JML第一次作业类图

1. 第二次作业——无向无权图查询系统

·作业要求

本次作业要求在上次作业的基础上再增加一些功能，使之变成一个图查询系统。三个添加的最主要功能为：

public boolean containsEdge(int i, int i1)

public boolean isConnected(int i, int i1)

public int getShortestPathLength(int i, int i1)

即为求邻接情况、连通情况及最短路径的问题。

·重构情况

从作业设计角度来看，本次作业是基于上次作业的增量设计。之所以这样说，最直观的一个角度就是因为本次要实现的Graph类是PathContainer类的一个子类。因此上次修复bug后的全部内容都可以保留并且不做修改。（除了维护addPath时新建的子类）。所以基本上不需要重构只需要做增量的设计。

·算法介绍

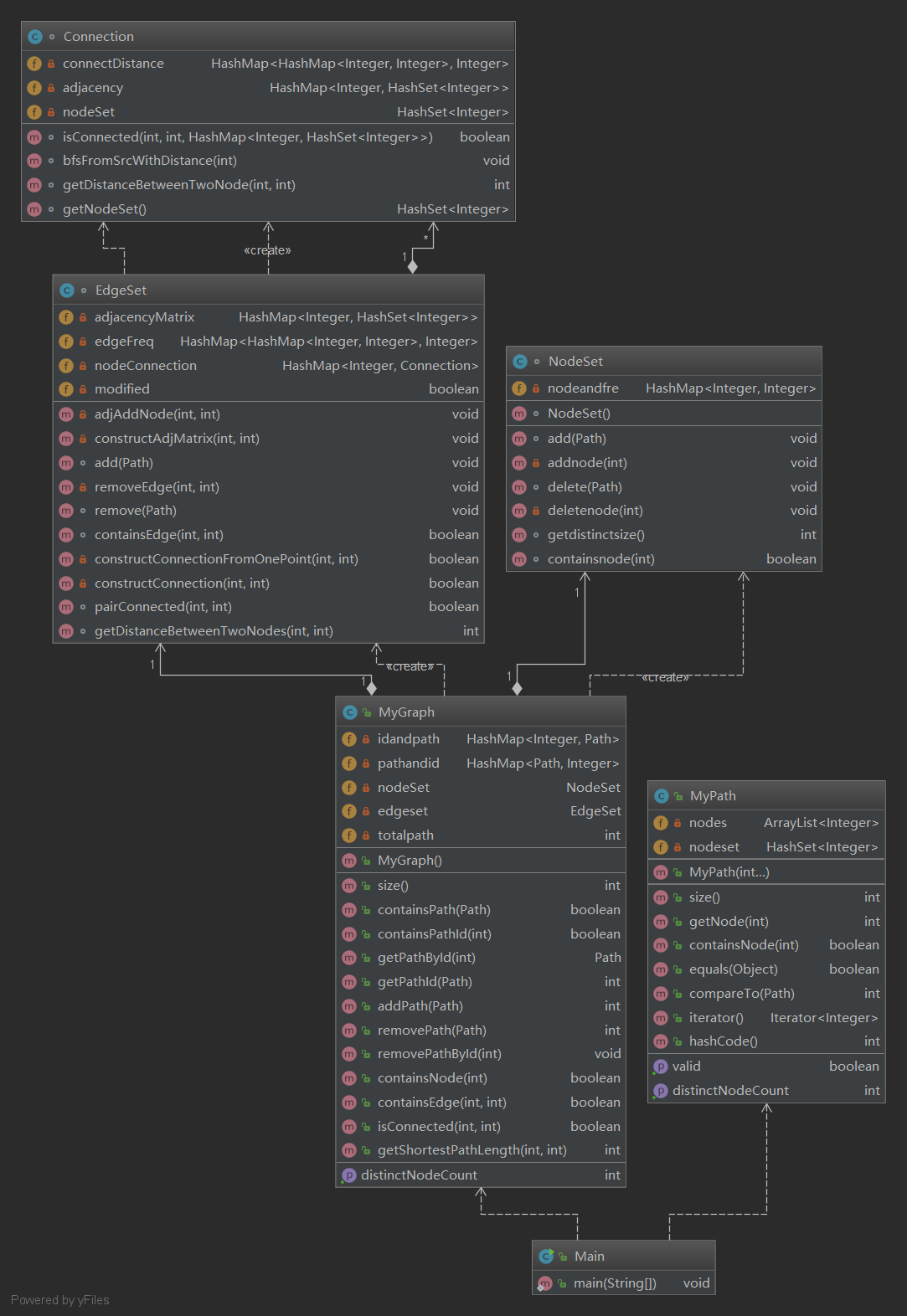
这次作业中涉及的最短路径问题是无向无权图的单源最短路径问题。所以我们用简单的BFS就能够解决。后来我又发现求两个节点连通的过程中也是使用的BFS。二者是同一的时间复杂度。因此不管是求连通还是求距离，我们都用BFS来求单源最短距离，并保留了二者的连通关系。这样就能够实现一个缓存系统，即查询求出来的距离和连通关系保留，直到图更新为止。当图更新，缓存清空。使用这样的方法能够大大提升程序的整体性能，没有必要进行冗余的计算。

·上次作业的重构情况

不需要重构，基本上只要在上次的基础上根据JML定义的规格进行增量设计即可。

·设计架构

下面的类图是我的设计架构。



简单解释：可以看到在Graph类下面有两个类，一个是NodeSet，一个是EdgeSet。NodeSet主要负责节点的查询，EdgeSet负责构建邻接矩阵以及构建连通分量类Connection。Connection中进行BFS。当时并没有考虑这样做的扩展性好不好，只是单纯的这样做了。实际上回过头来看这样做的扩展性是非常差的：在后面的有权图扩展中，我们有一些有权图的扩展，实际上他们都是图的算法，可以单独创一个类然后里面有算法这样做。换句话来说，我们的邻接矩阵和Connection类之间的关系太过密切，导致扩展性并不好。

1. 第三次作业——地铁查询系统（拆点的无向有权图）

·作业要求

本次作业要求实现一个地铁查询系统。实际上这个系统就是一个大型的无向有权图。比较麻烦的一点是相同的节点在不同的路径上面的意义不同。也就是同一个地铁站在不同的线路上面代表的意义不同，并且相同节点在不同的线路的站点之间还有权值问题。这一切都需要我们重新考虑算法。

·算法

这次作业中我使用了拆点的想法。即建立一个新的类表示站点。站点相等的唯一条件是二者是同一节点并且在同一条路径上——即将以上的两个值作为站点这一数据结构的key值。然后在建图的时候就将不同站点之间的权值以及相同节点不同路径站点上的权值都计算出来，存在邻接矩阵里。实际上求各种最短路径的区别仅仅是不同的计量方法的边的权值不同，而其主要的算法都是一样的——我使用的是单源最短路径算法（Dijstra算法）来跑有权图并将得到的所有距离缓存。所以不管是距离、“票价距离”、“不满意度距离”、“换乘距离”都可以使用同一个模型进行建模，只要我们求出了他们的邻接矩阵。因此可扩展性大大提高了。

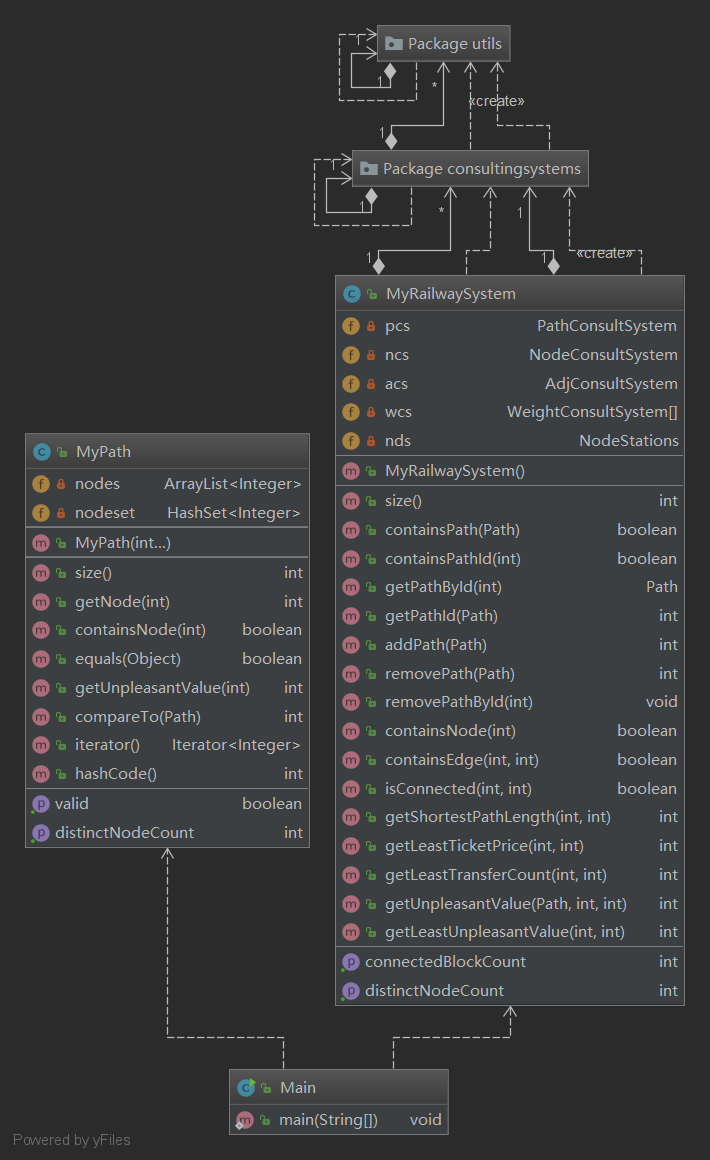
在判断连通块个数的过程中，为了降低程序的时间复杂度，进行了一个高层的抽象：将一个节点抽象为此节点对应的路径，即化路径为点。我们再在路径点组成的图上使用判断连通性的一些算法（例如BFS）来实现连通性的查询、连通块个数的查询。这样的BFS就能够将时间复杂度降低到了以路径条数为计量的时间复杂度上，简化了计算。

·重构情况

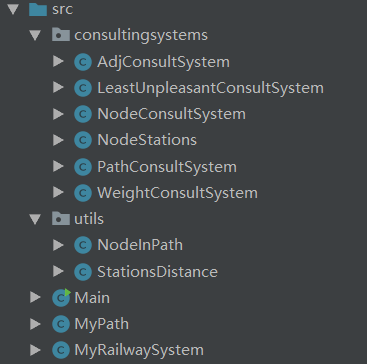
如上一部分的分析，我们上次作业的可扩展性并不好。所以在这部分我们基本上没有沿用上次作业关于查询的部分。全部进行了重构。

·设计架构

下面是本次作业的类图。



第三次作业类图



第三次作业程序结构

因为这里的utils包和consultingSystem包没法完整的显示，因此我出示下面的文件结构图，其能够让我们看到里面的内容。

AdjConsultSystem：承接第二次作业的EdgeSet。主要负责邻接情况的查询。

NodeConsultSystem：承接第二次作业的NodeSet。主要负责节点相关的查询。

PathConsultSystem：主要负责有关于路径的相关查询，如路径ID等。根据算法我们的连通性查询也在这个类中有所体现。

WeightConsultSystem：这是有权图的查询。所有的有权图通过两个参数建立分别：普通站点连接的权重、换乘权重。其在MyRailSystem中体现为工厂模式的一个数组。分别传入不同的参数来实现不同的查询功能。

LeastUnpleasantSystem：他继承了上面的WeightConsultSystem。因为这个系统中边的权值不能通过简单的传入参数来确定，而是需要具体的计算。所以我们重写其中的赋予权重函数。

1. 代码实现的bug和修复情况

主要的bug是第一次作业的Bug。在第一次作业实现的时候因为觉得规格非常简单，实现也很简单。但是实际上性能并不好。尤其是getDistinctNodeCount()方法在每次调用的时候都是重新计算的。

解决方法是写了NodeSet类。并且在不仅对path和id建立映射，也将id和path之间建立映射，这使查询的效果更好。

之后的两次作业因为比较在意性能方面的表现，所以并没有在强测和中测中暴露出bug。

1. 规格撰写和理解的心得体会

规格我们也在各种实验和作业中见到和写了不少了。一般来说需要我们写规格的情况是以下两种：

①在设计阶段：希望本方法（类）有相关的功能，但是只有文字描述不够清晰和规范，需要用规格来进一步的表述以更加贴近代码。

②在写好代码之后：根据代码用JML写规格。其实我始终没有搞懂这样做的目的是什么。直到我写博客的第一部分，在确定了规格的作用之后明白了：实际上这样做是需要我们根据写出的规格来判断这个程序是不是符合我们当初的要求。如果不符合先修改规格，再根据规格重写代码。这样能够做到在逻辑上尽量不出错。

关于写规格的感想，主要有以下几个方面：

①规格的写法很多，实际上规格是一种抽象的函数，只要经过规格的作用之后得到的结果和现象是我们需要的，就是合理的规格。

②规格撰写的时候要考虑参数的各种情况发生的各种特殊情况，要考虑规格的覆盖性。尽量把每个情况的行为都能规定好。。

③熟练。只有对JML这样一种语言足够熟练才能够理解其每条语句的内在含义和使用的作用范围，这些东西只有靠多看、多练才能够掌握。

④逻辑性。可以对应离散数学中的充要条件等等来写。

在写规格方面我还是有很多不足的。下面说一说规格在具体实现中的意义。

在具体实现的过程中也许我们处于性能的考虑、出于架构的考虑，不会按照规格规定的严格去做。但只要结果是符合规格的要求、理解规格的思想，就能够达到我们去写规格的目的，也能够实现规格的意义。从某种意义上来说，我感觉这三周的作业要比之前的作业简单。这或许是因为规格规定了具体的实现结果，能够使开发者在对一个功能进行开发的时候仅考虑其性能、实现方法而不再考虑为什么要进行这样的实现。这对平常写程序经常会没有思路的我来说是一种思路上的化简，也是一种从底向上的思想。这就使程序开发从创新性的工作逐渐变成了填空题。也符合自底向上的开发思想。

1. 作业感想

每次作业希望都是一个提升的过程。在JML的这一单元里面我有遗憾，例如第一次作业的不仔细不小心，被强测卡住了性能。这提醒我程序开发的过程中永远不能够马虎大意，不能只在乎功能而不考虑性能。也有很大的进步，例如本次作业中，尤其是第三次作业，我的逻辑划分还是非常清楚的，并且对于代码的重用、继承的使用、工厂模式的使用等等都有了比较熟练的掌握。这与前几周的煎熬也分不开。也体会到了JML的作用、程序规格化开发给程序员带来的好处。希望在今后的学习和工作中能够将这些思想和方法应用到实践中。最后还是希望最后一次作业能够圆满的完成，将之前程序训练的结果展示出来。