1. 分析和总结自己三次作业来的设计策略及其变化
   1. 第一次作业

比较幸运第一次接触多线程的时候就可以遇到最后的方案，这次作业我运用的是synchronized+notified的策略，通过电梯线程和scheduler线程互相唤起来实现多线程的运行，而在线程不进行计算的时候就让现场wait。这样将线程的工作时间和强度降到最低，同时也保证了绝对的安全。

* 1. 第二次作业

由于要求限制，不得不放弃安全好用的线程互相唤醒的机制，使用不断循环扫描实现，通过每隔一定时间扫描一次进行一次快照，记录当前监控对象的状态，这种机制面临很多问题，比如如何保证一次扫描之间不会进行两次操作，好在最后对此进行了限制。

* 1. 第三次作业

单纯的synchronized，这次使用的县城方法比较简单，为了防止100个线程对系统资源利用过多，产生不可预料的后果，我将100辆车合并到了同一个线程，用轮询进行车的移动，车移动完之后在进行派单，虽然方法略显粗暴，但是效果奇好。再辅助以假时间的方法，可以解决非常多的问题。

1. 基于度量来分析自己的程序结构
   1. 第一次作业

有一些方法由于刚开始的时候没有考虑清楚，写的太过臃肿，这种现象主要是集中于带有大量条件判定的方法。关于条件判定的方法比如电梯选择，可稍待判断等可以继续优化。

优点：整体设计比较清晰

缺点：条件判定想得不够清楚带来冗余。

设计原则不足：存在GOD类

* 1. 第二次作业

与上次相比较略好一些，但是对于不同的触发器操作写到了一起。导致了方法的臃肿。

缺点：针对不同情况的操作写到了一个类。

设计原则不足：Dependency Inversion Principle，责任均衡分配原则

* 1. 第三次作业

比之前的好了很多，分派方法嵌套太深。

优点：责任分派相对均衡。

缺点：方法调用深度过深。

设计原则不足：重用原则，有功能重复的类。

1. 分析自己程序的bug
   1. 第一次作业

所交版本为调试版本，所有通知请求，未按照格式输出。

* 1. 第二次作业

无bug

* 1. 第三次作业

输出格式错误扣一分

将3000毫秒误计算为14个200ms的周期逻辑错误。

* 1. 基本都挺脑残的。。。没啥可分析的。

1. 分析自己发现别人程序bug所采用的策略
   1. 第一次作业

基本输出错误格式错误。

逻辑错误

仅靠前两项已经扣的有点多了，心软没有继续查下去

* 1. 第二次作业

边界测试，测出程序不能监控识别文件夹的最后一个文件

* 1. 第三次作业

对方由于版本问题，没有考虑gui带来的延迟导致了一系列错误。

同时依靠边界数据测出一个逻辑bug

* 1. 没有发现线程不安全的问题，拿到的基本上写的都不错，但是相比前三次作业而言，从中可学习的东西不是特别多，也可能我自己进步了。

1. 心得体会
   1. 毫无疑问notify+synchronized是现阶段最好的方法，即可以保证的线程的安全，又能降低线程的CPU占用。也更符合现实中的情况。
   2. 第六次作业全局锁真的是反人类的东西，可还不得不用，杨帅提出的对象全局共享很有意思，被我改造用到了第七次作业。
   3. 增强for循环还是个好东西，方便简单。学会使用@override等注解小技巧可以提高效率。
   4. 第七次作业明显简单一些，可以花时间考虑设计，这个时候难度降低有一定科学性。
   5. 千万不要临结束前一分钟提交，难保出不出意外。
   6. 由于之后的自由度大大提高，需要适应新的互测形势。
   7. 设计原则还需磨练。