线程及其同步设计结构：

使用继承Thread类或者实现Runnable接口来实现多线程的建立。

采用互斥控制：使得任何时刻只允许一个线程获得访问/执行权限。

synchronized(obj) {…} ：任意时刻只允许一个线程对对象obj进行操作

synchronized method(…){…} 任意时刻只允许一个线程调用方法method

任何线程在临界代码区(critical section)执行时

可能有多个其他线程在等待进入执行

所以执行结束前通过notify/notifyAll来让JVM调度一个线程进入并执行。

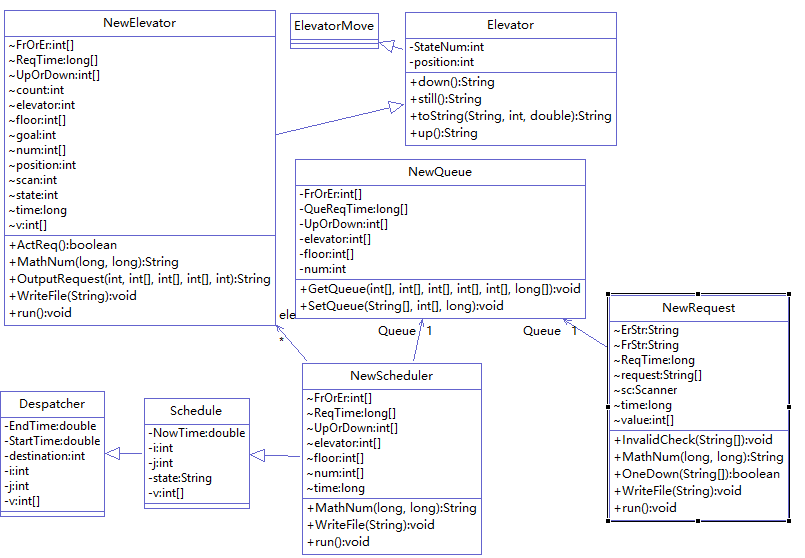
通过锁来实现进程同步及互斥：

J锁对象e的同步方法(“synchronized m(…){}”)

使用锁对象e来控制的同步代码段(“synchronized (e){…}”)

利用wait()来让一个线程在某些条件下暂停运行。例如，在生产者消费者模型中，生产者线程在缓冲区为满的时候，消费者在缓冲区为空的时候，都应该暂停运行。如果某些线程在等待某些条件触发，那当那些条件为真时，可以用 notify 和 notifyAll 来通知那些等待中的线程重新开始运行。不同之处在于，notify 仅仅通知一个线程，并且我们不知道哪个线程会收到通知，然而notifyAll会通知所有等待中的线程。

如果read-modify-write和check-then-act计算只涉及单一变量，就可以通过java.util.concurrent.atomic包中提供的Atomic\*\*\*\*类型来确保计算的原子性。

第三次电梯：（1）类图

1. 分析程序设计结构

类的个数：7

Despatcher类（40行）：属性6，方法1（Paln()30行）,

Elevator类（80行）：属性3，方法9

NewElevator类（200行）：属性12，方法6（NewElevator()4行，run()120行，ActReq()10行，MathNum()5行，WriteFile()21行，OutPutRequest()12行）

NewQueue类（87行）：属性5，方法2（GetQueue()38行，SetQueue()31行）

NewRequest类（156行）：属性8，方法6（NewRequest()4行，InvalidCheck()34行，MathNum()5行，OneDown()8行，run()18行，main()23行）

NewSchedule类（169行）：属性8，方法4（NewSchedule()5行，MathNum()5行，run()107行，WriteFile()22行）

Schedule类（215行）：属性5，方法2（OutputRequest()11行，Plan()200行）

线程协作： set

NewRequest

Main

Three Elevators

Queue

NewSchedule

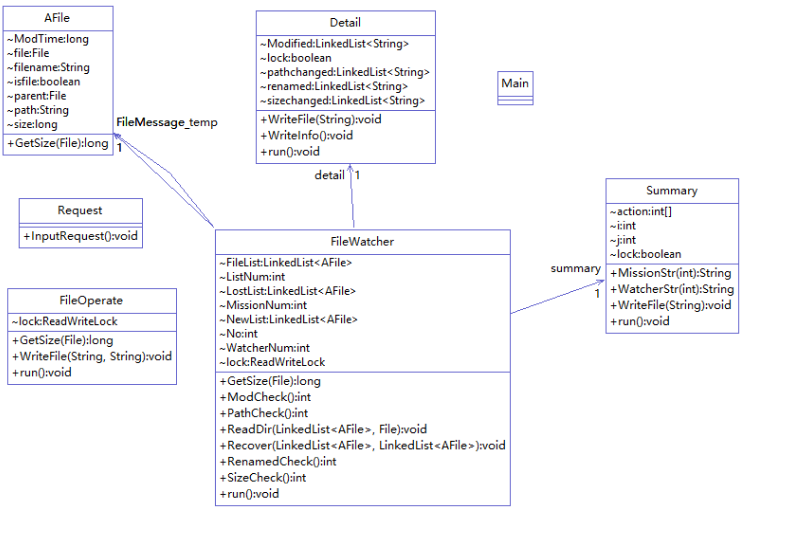
Get

NewRequest线程接收请求，并传递请求到Queue类中，NewSchedule从Queue类中得到新的请求，并调度给合适的电梯。

优点：一切调度工作交给NewSchedule线程，便于控制电梯。

缺点：NewSchedule线程所做的工作过于多，导致NewSchedule类代码冗长。

IFTTT:（1）类图：



1. 分析程序设计结构

类的个数：7.

AFile类（45行）：属性7,方法2（Afile()10行，GetSize()20行）

Detail类：（114行）：属性5，方法4（add()28行，run()23行，WriteFile()21行，WriteInfo()19行）

FileOperate类（125行）：属性1，方法4（FileOperate()3行，run()64行，WriteFile()21行，GetSize()19行）

FileWatcher类（510行）：属性11，方法9（FileWatcher()35行，run()56行，RenamedCheck()98行，PathCheck()97行，ModCheck()49行，SizeCheck()81行，Recover()10行，GetSize()19行，ReadDir19行）

Main类（110行）：属性0，方法1（main()96行）。

Request类（11行）：属性0，方法1（InputRequest()6行）

Summary类（91行）：属性4，方法5（add()5行，run()26行，WatcherStr()9行，WriteFile()21行，MissionStr()8行）

线程协作:

FileWatcher

Main

FileOperate

Summmary

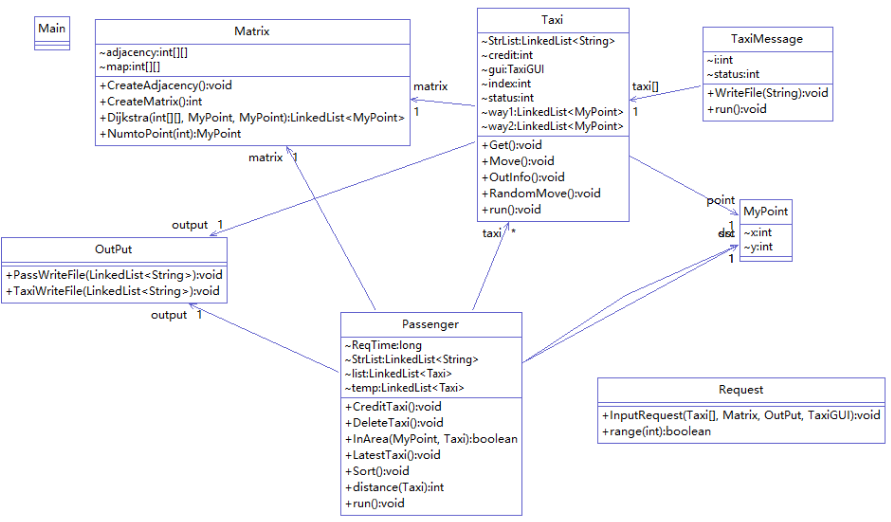
Detail

FileOperate线程负责对文件进行访问、修改，FileWacher线程监控文件变化状态并及时向Summary或Detail线程传递信息。

优点：各线程分工明确，不会造成某个类过于大。

缺点：多个线程，如果采取的锁模式没有考虑周到，可能造成效率低下

出租车：（1）类图



1. 分析程序设计结构

类个数：9

gui类

Main类（39行）：属性0，方法1（main()38行）

Matrix类（240行）:属性3，方法4（CreateAdfacency()35行，CreateMatrix()61行，Dijkstra()97行，NumtoPoint()10行）

MyPoint类（10行）：属性2，方法1（MyPoint()4行）

OutPut类（61行）：属性0，方法2（PassWriteFile()24行，TaxiWriteFile()24行）

Passenger类（209行）：属性9，方法8（Passenger()11行，CreditTaxi()12行，DeleteTaix()7行，distance(6行)，InArea()9行，LatestTaxi()11行，run()82行，Sort()29行）

Request类（74行）：属性0，方法2(InputRequest()54行，range()8行)

Taxi类（206行）：属性10，方法6（Taxi()7行，Get()7行，Move()62行，OutIlfo()7行，RandomMove(49行)，run()45行）

TaxiMessage类（78行）：属性3，方法3（TaxiMessage()5行，run()36行，WriteFIle()21行）

线程协作：

呼叫

100 Taxi

Main

Request

Passenger

对于每一条请求，产生一个Passenger线程，Passenger线程向出租车线程发出乘车请求。

分析程序bug：

这三次作业，我的bug主要有以下几点，第一，第三次电梯时，我的请求队列实际上没有加对锁，这使得程序出现了一系列的时对时错的bug，除此之外，我还犯了一些粗心的毛病，比如对正则表达式的理解还不充分，导致一些错误的输入没有报错。在IFTTT那一次的作业中，我的程序只要监控C盘或者D盘这样的目录，就有可能crash，这是因为系统盘中有一些隐藏的临时系统文件夹，如System Volume Information(系统卷标信息)等。

如何测试：

主要针对临界值，如第三次电梯时测试可以用多层捎带请求测试，测试出租车可以使用同时发出的请求来测试线程的安全问题。

心得体会：

在多线程编程时，共享的资源最好都加上锁，毕竟在不考虑性能的情况下，这是最安全的。编写多线程程序时，对于哪些资源是临界资源一定要在设计开始时就设计好，否则容易遗漏加锁。多个线程读写操作时，可以使用java的读写锁，避免使用synchronized方法导致的读之间互斥，以及性能低下。