经过详细的需求分析，我们决定使用Java作为物流园区模型的编程语言。Java是一个面向对象的语言，并且拥有许多功能强大的第三方类库，这意味着我们可以简化模型中复杂的处理过程而把注意力集中到仓位、巷道、订单、货车这几个类对象的成员变量和成员函数的构造，从而给我们的求解提供便利。除此之外，Java语言是多线程的，这一特性与我们模型中多个货车对象并行运行的需求非常契合。

整个物流园区的模型求解和路径优化可通过以下几个步骤实现：

1. **初始化园区内的仓位和巷道**

为了能像模型中一样，使园区内的仓位和巷道有良好的可操作性，在局部上，我们构造了仓位类和巷道类，在全局上，我们把园区的平面图划分为了两种矩阵，一种是仓位矩阵，一种巷道矩阵，矩阵中的每一项都是一个仓位对象或者巷道对象，这样我们就由上至下掌握了园区内的所有仓位和巷道，并能很轻松地进行操作。

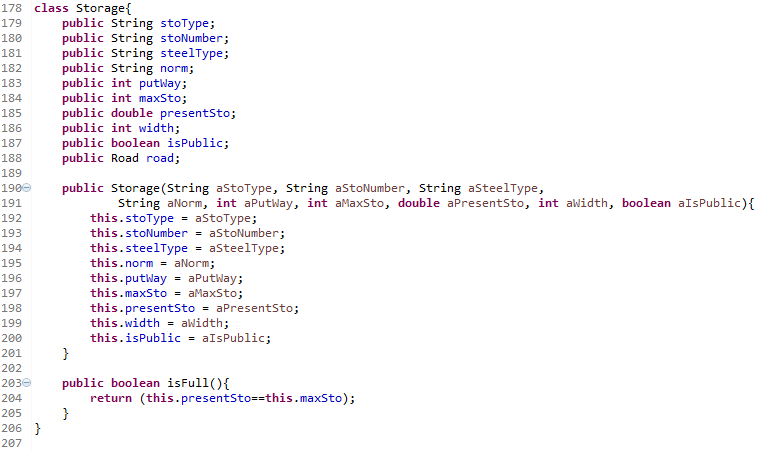


Figure 仓位类

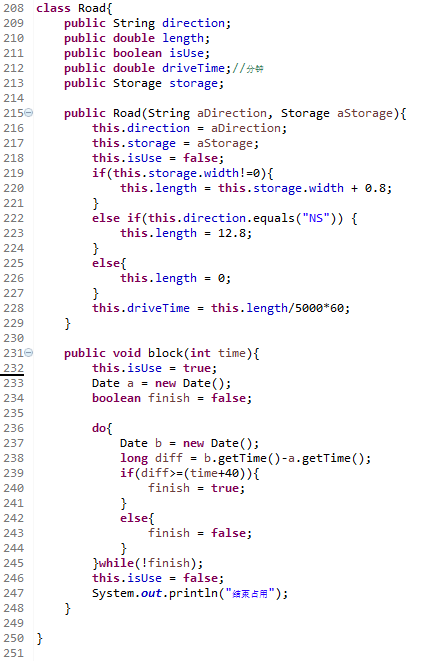


Figure 巷道类

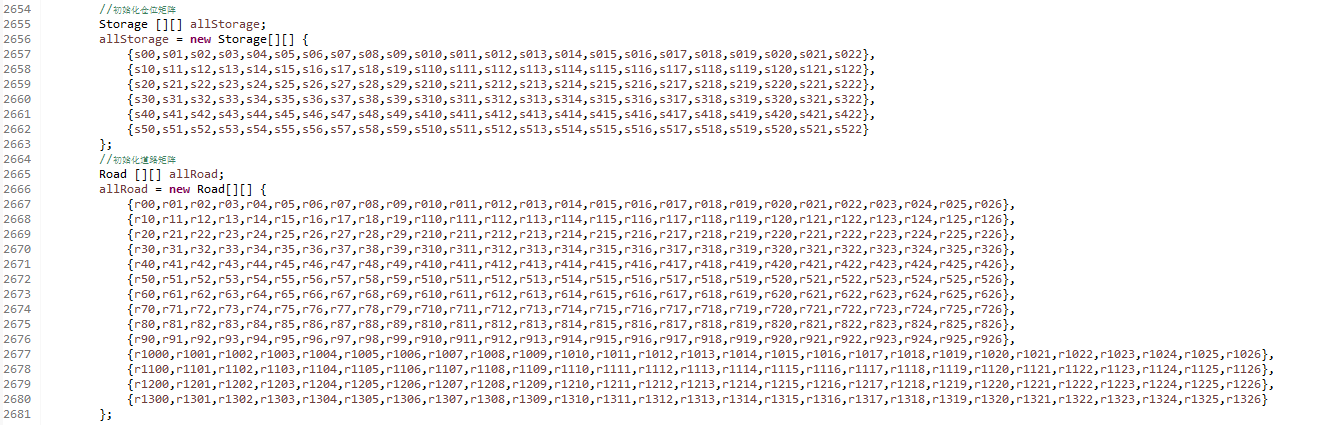


Figure 仓位矩阵和巷道矩阵

1. **读取入库订单和出库订单并实例化订单对象和货车对象**

由于数据量非常巨大，所以我们使用了一个叫做JXL的Java第三方类库，JXL是通过Java操作Excel表格的工具类库，支持Excel 95-2000的所有版本，能生成Excel 2000标准格式，支持字体、数字、日期操作，能够修饰单元格属性，支持图像和图表，同时这套API是纯Java的，并不依赖Windows系统，即使运行在[Linux](http://lib.csdn.net/base/linux)下，它同样能够正确的处理Excel文件。



Figure 通过JXL读取Excel文件并实例化订单对象和货车对象

对于每一个订单对象，我们把从Excel文件中读取到的订单信息通过构造函数传递给该对象作为它的成员变量的值，并且通过字符串匹配到该订单的目标仓位的编号。

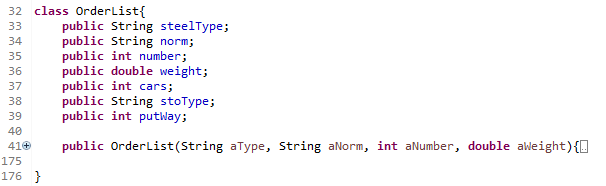


Figure 订单类



Figure 通过构造函数确定该订单的目标仓位的编号

实例化了订单对象之后，我们就可以根据订单对象实例化对应的货车对象。货车对象是我们整个模型中的核心，为了使货车对象能够遍历到园区中的每一个仓位和每一个巷道，我们把经过初始化的仓位矩阵和巷道矩阵赋予了货车对象中对应的成员变量。

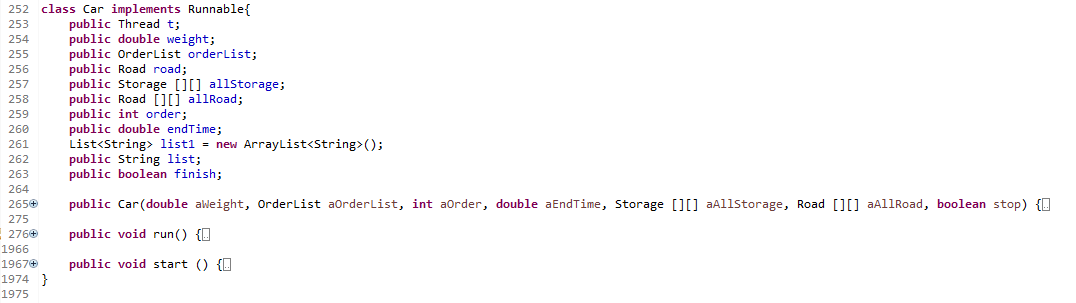


Figure 货车类

3. **为每个货车对象分配一个线程并启动**

通过货车类的代码可以看到，我们用货车类通过实现Runnable接口并重写Run()方法实现了多线程。为什么要实现多线程，这是因为园区中的所有货车都是不同的角色，通过多线程编程，我们才能实现对模型场景的还原，同时进一步提高模型的准确性。如果不使用多线程，所有的货车对象的运行方式实际上是串行的，这样就无法模拟出园区中货车之间对仓位资源和巷道资源的抢占和堵塞，整个模型的效果将严重失真。

为了更好地处理货车对象对仓位和巷道这样的公共资源的抢占，我们在程序中运用了锁和优先级调度的技术。

试想，面对数量众多的货车对象，公共资源的数量只有一个，当大量的货车对象对一个公共资源发起访问时，该公共资源只能先回应一个货车对象的访问请求，同时屏蔽其余货车对象的请求，从而实现一个货车对象对该公共资源的抢占。但是，由于这个抢占过程不是原子操作，有可能该公共资源还没有完成对其余货车对象的屏蔽就有多个货车对象认为自己已经完成了抢占，开始进行抢占后续的操作，从而导致了严重的错误。锁的技术就是为了解决这样的问题，只有获得某个公共资源锁的钥匙的货车对象才能发起对该公共资源的访问，避免了多个对象同时对一个公共资源的操作。

优先级调度则是为了解决饥饿问题。因为当多个对象对某个公共资源发起访问时，总是只能有一个对象实现对该资源的抢占，其余对象只能在抢占解除后再次尝试发起访问。但是这样就可能有运气不好的对象每一次都无法抢占到该资源，从而无限等待下去，导致饥饿问题。优先级调度，就是为每一个对象赋予优先级，当一个对象经过多次等待后，它的优先级就会越来越高，从而更容易抢占到该公共资源，避免饥饿问题。

4. **货车对象根据目标仓位匹配算法确定自己的目的地**

为每个货车对象分配一个线程并启动后，货车对象首先要根据自己对应的订单信息确定自己的目标仓位，再把目标仓位对应的巷道作为自己的目标巷道。



Figure 匹配专用仓位

寻找目标仓位的过程分为两步：第一步先匹配专用仓位，如果匹配到还有足够容量的专用仓位就把该仓位作为目标仓位；如果没有匹配到可用的专用仓位就开始寻找可用的公共仓位，这里以A类订单匹配B类公共仓位为例，当匹配到为空的B类公共仓位后，就把该公共仓位重新构造为自己这一类的专用仓位（每个仓位只能存放种类相同的钢材），然后把其作为自己的目标仓位。



Figure 匹配公共仓位

5. **根据目的地确定行驶路线并开始寻路**

园区内的巷道分为两种类型，一种是双向单车道，一种是单向双车道。由于考虑到安全超车的问题，所以两种巷道的寻路策略并不相同。为了在各个货车对象均会抢占仓位资源和巷道资源从而导致堵塞的情况下，寻找到全局情况下的最优行驶路线（堵塞时间最短），我们采用了动态规划算法来确定货车对象的行驶路线。

动态规划[算法](https://baike.baidu.com/item/%E7%AE%97%E6%B3%95)是通过拆分问题，定义问题状态和状态之间的关系，使得问题能够以递推（或者说分治）的方式去解决的一种算法。动态规划算法的基本思想与分治法类似，也是将待求解的问题分解为若干个子问题（阶段），按顺序求解子阶段，前一子问题的解，为后一子问题的求解提供了有用的信息。在求解任一子问题时，列出各种可能的局部解，通过决策保留那些有可能达到最优的局部解，丢弃其他局部解。依次解决各子问题，最后一个子问题就是初始问题的解。

能采用动态规划求解的问题的一般要具有3个性质：

（1）最优化原理：如果问题的最优解所包含的子问题的解也是最优的，就称该问题具有最优子结构，即满足最优化原理。

（2）无后效性：即某阶段状态一旦确定，就不受这个状态以后决策的影响。也就是说，某状态以后的过程不会影响以前的状态，只与当前状态有关。

（3）有重叠子问题：即子问题之间是不独立的，一个子问题在下一阶段决策中可能被多次使用到。（该性质并不是动态规划适用的必要条件，但是如果没有这条性质，动态规划算法同其他算法相比就不具备优势）

可以看到，我们设计的模型非常符合动态规划算法的这三个性质，故而我们使用了动态规划算法来求解货车对象的最优路径。



Figure 包含寻路算法的货车类中的run方法

（完整的寻路算法共有一千五百余行代码，由于版面原因不在此进行详细的贴图和代码描述）

1. **抵达目标仓位进行作业，完成后离开园区**

通过寻路算法，货车对象将得到当前全局条件下最优的行驶路线，并顺利到达目标巷道和目标仓位进行入库和出库操作。在完成操作后，货车对象会把园区出口作为目标巷道再次通过寻路算法规划出最优路径然后离开园区。

**通过上述步骤，我们完全还原了设计模型中的出入库流程，并在全局条件允许下，通过多线程技术、锁技术、优先级调度技术、动态规划算法、仓位匹配算法、寻路算法，配合订单类、仓位类、巷道类、货车类、仓位矩阵、巷道矩阵，实现了对物流园区模型的求解和货车路径的优化，在工程代码层面为模型设计提供了强有力的验证和参考。**