Supply Chain Demo 2

Updated: 2022-01-04

NOTE: this is not any of the SC-1, SC-2, or SC-3; this is a model that precedes SC-1.

This demo is based on the model provided by Rutgers' School of Business team (Ben M./ Chen Weiwei / Aman Goswami /et al), specifically slide No. 2 in the PowerPoint file `Pharma-SC-Map_With-Disruptions.pptx` provided on 2021-12-07.

The code in this project represesnts a simple supply chain simulation implemented in Java using the MASON API and the MASON DES library

Prepare

You need a Java compile, which can be obtained from Oracle or OpenJDK. For convenience, I also use a Git client and Apache Ant (a `make`-like tool for Java).

- Check out the source code for this project from GitHub. Let's say you've put it to ~/mason/work on your machine:

```
mkdir ~/mason
cd ~/mason
mkdir work
cd work
git remote add origin https://github.com/eclab/DES-Supply-Chain-demo.git
git pull origin master
```

We'll call this location your *work directory*.

- Create also `~/mason/lib`, and put into it the MASON jar file, `mason.20.jar`, which you should download from the MASON site above.

Note: as of 2022-01-01, we are using a customized version of mason.20.jar, which has an extra class added to support skewed triangular distribution. Ask Vladimir or Sean Luke to provide you with this customized version.

- Obtain the DES library source code from its own GitHub repo (see the link above), compile it and package it into `des.jar`, and place the JAR file into the same `~/mason/lib` directory. Sample commands for doing this (assuming you place the DES library source code under `~/mason/git`):

```
#!/bin/csh

set m=~/mason

cd $m
mkdir git
cd git
git init
git remote add origin https://github.com/eclab/mason.git
git pull origin distributed-3.0

cd $m/git/code/contrib/des
javac -d $m/out -cp $m/lib/mason.20.jar sim/des/*.java
# javadoc  -d $m/javadoc -cp $m/lib/mason.20.jar:$m/out -sourcepath . sim.des
cd $m/out
jar cf des.jar sim/des/*.class
cp des.jar ../lib
```

Warning: the above may take forever, as GitHub will tend to give you megabutyes and gigabytes of stuff you don't need. So, alternatively, you can just ask me for precompiled des.jar, and put it into your ~/mason/lib

Compile

I compile with Apache Ant:

```
cd ~/mason/work
ant
```

Apache Ant is the tool that takes its commands from build.xml. If you don't want to use Ant, you can just compile the Java code from `src` using the `javac` command, and making use of the JAR files in the ../lib directory.

Run

You can run the simulation with a command like this:

```
./run2.sh -until 365 > tmp.log
```

The `-until 365` is a standard MASON option, which will cause the application to run for 365 units of simulation time (simulated days, in this model). Without this option, the application will likely never stop.

Command-line options

There are a few additional command-line options you can use with `run.sh`:

- `-verbose` --- causes various elements of the supply chain model to print various progress messages
- `-config config/pharma2.csv` -- specifies the name of the configuration file (see below)

You can also use any of the options that all MASON applications take, such as the already-mentioned `-until`. See the MASON manual (found at the same web site) for details.

Configuration file

The simulation tool takes the parameters of each element of the model from the configuration file, which, by default, is `conf/pharma.csv`

The configuration file is in CSV format. All lines beginning with a hash mark (#) are ignored, as well as all empty lines.

The data lines have the following format:

```
elementName,propertyName,val1[,val2,...]
```

The first column contains the name of the element of the network to which the property applies. The second column, the name of the specific property. The third column (and any subsequent) columns contain the value of the property.

Normally, there is just one value column in each row. However, some types of properties need several columns to be described. For example, the following line

```
Production,faulty,Uniform,0.1,0.2
```

means the that the parameter `faulty` for the element called `Production` describes a uniform distribution with the range [0.1,0.2]. (This is the distribution from which the model of the post-production QA step draws the number that determines the portion of faulty items in each produced lot. This particular distribution means that 10% to 20% of items in each lot are found faulty.)

Sample output

The output file, for a run without the `-verbose` option, may look like this:

```
 ./run2.sh -until 365 > t.tmp

 ===== Start ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=0.0; has 0.0 units of packagedDrug on hand. Has Received=0.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=0.0; of this, still on ships=0.0, QA discarded=0.0, QA released=0.0]
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=0.0; of this, still on ships=0.0, QA discarded=0.0, QA released=0.0]
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=0.0; of this, still on ships=0.0, QA discarded=0.0, QA released=0.0]
[Production.ApiProduction; stored inputs=([RawMaterial:0.0]). Ever started: 0.0 + (rework=0.0) = (in the works=0.0; in QA= 0.0; discarded=0.0;
[Production line (API): accepted 0 batches, totaling 0.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:0.0]). Ever started: 0.0 = (in the works=0.0; in QA= 0.0; discarded=0.0; good=0
[Production line (bulkDrug): accepted 0 batches, totaling 0.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:0.0]). Ever started: 0.0 = (in the works=0.0; in QA= 0.0; discarded=0.0
[Production line (packagedDrug): accepted 0 batches, totaling 0.0]
[Distributor.Distributor(packagedDrug): Shipping plan=0.0, has shipped=0.0, in 0 loads. Of this, 0.0 is still being shipped. Remains on hand=0
==============================================
===== Report at t=59.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=3800.0; has 0.0 units of packagedDrug on hand. Has Received=0.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=3800.0; of this, still on ships=3800.0, QA discarded=0.0, QA released=0.0]
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=3800.0; of this, still on ships=1900.0, QA discarded=188.0, QA released=1712
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=3800.0; of this, still on ships=3800.0, QA discarded=0.0, QA released=0.0]
[Production.ApiProduction; stored inputs=([RawMaterial:0.0]). Ever started: 0.0 + (rework=0.0) = (in the works=0.0; in QA= 0.0; discarded=0.0;
[Production line (API): accepted 0 batches, totaling 0.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:0.0]). Ever started: 0.0 = (in the works=0.0; in QA= 0.0; discarded=0.0; good=0
[Production line (bulkDrug): accepted 0 batches, totaling 0.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:1712.0]). Ever started: 0.0 = (in the works=0.0; in QA= 0.0; discarded=
[Production line (packagedDrug): accepted 0 batches, totaling 0.0]
[Distributor.Distributor(packagedDrug): Shipping plan=3800.0, has shipped=0.0, in 0 loads. Of this, 0.0 is still being shipped. Remains on har
==============================================
===== Report at t=119.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=7600.0; has 0.0 units of packagedDrug on hand. Has Received=0.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=7600.0; of this, still on ships=1900.0, QA discarded=304.0, QA released=5396.
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=7600.0; of this, still on ships=1900.0, QA discarded=475.0, QA released=5225
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=7600.0; of this, still on ships=1900.0, QA discarded=377.0, QA released=5323.0]
[Production.ApiProduction; stored inputs=([RawMaterial:96.0]). Ever started: 5300.0 + (rework=100.0) = (in the works=900.0; in QA= 100.0; disc
[Production line (API): accepted 54 batches, totaling 5400.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:1023.0]). Ever started: 4300.0 = (in the works=4300.0; in QA= 0.0; discarded=0.
[Production line (bulkDrug): accepted 43 batches, totaling 4300.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:5225.0]). Ever started: 0.0 = (in the works=0.0; in QA= 0.0; discarded=
[Production line (packagedDrug): accepted 0 batches, totaling 0.0]
[Distributor.Distributor(packagedDrug): Shipping plan=7600.0, has shipped=0.0, in 0 loads. Of this, 0.0 is still being shipped. Remains on har
==============================================
===== Report at t=179.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=11400.0; has 0.0 units of packagedDrug on hand. Has Received=0.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=11400.0; of this, still on ships=3800.0, QA discarded=399.0, QA released=7201
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=11400.0; of this, still on ships=3800.0, QA discarded=620.0, QA released=698
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=11400.0; of this, still on ships=3800.0, QA discarded=438.0, QA released=7162.0]
[Production.ApiProduction; stored inputs=([RawMaterial:1.0]). Ever started: 7200.0 + (rework=300.0) = (in the works=0.0; in QA= 100.0; discard
[Production line (API): accepted 75 batches, totaling 7500.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:562.0]). Ever started: 6600.0 = (in the works=4700.0; in QA= 100.0; discarded=2
[Production line (bulkDrug): accepted 66 batches, totaling 6600.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:5380.0]). Ever started: 1600.0 = (in the works=1000.0; in QA= 0.0; disc
[Production line (packagedDrug): accepted 16 batches, totaling 1600.0]
[Distributor.Distributor(packagedDrug): Shipping plan=11400.0, has shipped=0.0, in 0 loads. Of this, 0.0 is still being shipped. Remains on ha
==============================================
===== Report at t=239.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=15200.0; has 3000.0 units of packagedDrug on hand. Has Received=3000.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=15200.0; of this, still on ships=3800.0, QA discarded=680.0, QA released=1072
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=15200.0; of this, still on ships=3800.0, QA discarded=881.0, QA released=105
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=15200.0; of this, still on ships=1900.0, QA discarded=659.0, QA released=12641.0]
[Production.ApiProduction; stored inputs=([RawMaterial:20.0]). Ever started: 10700.0 + (rework=300.0) = (in the works=200.0; in QA= 0.0; disca
[Production line (API): accepted 110 batches, totaling 11000.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:2841.0]). Ever started: 9800.0 = (in the works=3900.0; in QA= 0.0; discarded=70
[Production line (bulkDrug): accepted 98 batches, totaling 9800.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:5319.0]). Ever started: 5200.0 = (in the works=400.0; in QA= 100.0; dis
[Production line (packagedDrug): accepted 52 batches, totaling 5200.0]
[Distributor.Distributor(packagedDrug): Shipping plan=15200.0, has shipped=3000.0, in 2 loads. Of this, 0.0 is still being shipped. Remains or
```

```
================================================
===== Report at t=299.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=19000.0; has 5000.0 units of packagedDrug on hand. Has Received=5000.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=19000.0; of this, still on ships=1900.0, QA discarded=1031.0, QA released=160
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=19000.0; of this, still on ships=3800.0, QA discarded=1103.0, QA released=14
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=19000.0; of this, still on ships=3800.0, QA discarded=757.0, QA released=14443.0]
[Production.ApiProduction; stored inputs=([RawMaterial:69.0]). Ever started: 16000.0 + (rework=300.0) = (in the works=2700.0; in QA= 300.0; di
[Production line (API): accepted 163 batches, totaling 16300.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:2243.0]). Ever started: 12200.0 = (in the works=3600.0; in QA= 100.0; discarded
[Production line (bulkDrug): accepted 122 batches, totaling 12200.0]
[Production.Packaging; stored inputs=([bulkDrug:0.0, PackagingMaterial:6397.0]). Ever started: 7700.0 = (in the works=1100.0; in QA= 0.0; disc
[Production line (packagedDrug): accepted 77 batches, totaling 7700.0]
[Distributor.Distributor(packagedDrug): Shipping plan=19000.0, has shipped=5000.0, in 4 loads. Of this, 0.0 is still being shipped. Remains or
================================================
===== Report at t=359.0 ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=22800.0; has 7000.0 units of packagedDrug on hand. Has Received=7000.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=22800.0; of this, still on ships=3800.0, QA discarded=1163.0, QA released=178
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=22800.0; of this, still on ships=3800.0, QA discarded=1345.0, QA released=17
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=22800.0; of this, still on ships=3800.0, QA discarded=848.0, QA released=18152.0]
[Production.ApiProduction; stored inputs=([RawMaterial:37.0]). Ever started: 17800.0 + (rework=300.0) = (in the works=1400.0; in QA= 100.0; di
[Production line (API): accepted 181 batches, totaling 18100.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:2852.0]). Ever started: 15300.0 = (in the works=3500.0; in QA= 0.0; discarded=1
[Production line (bulkDrug): accepted 153 batches, totaling 15300.0]
[Production.Packaging; stored inputs=([bulkDrug:100.0, PackagingMaterial:6955.0]). Ever started: 10700.0 = (in the works=700.0; in QA= 100.0;
[Production line (packagedDrug): accepted 107 batches, totaling 10700.0]
[Distributor.Distributor(packagedDrug): Shipping plan=22800.0, has shipped=7000.0, in 6 loads. Of this, 0.0 is still being shipped. Remains or
================================================
===== Finish ==================================
[HospitalPool.HospitalPool(packagedDrug): Has ordered=24700.0; has 9000.0 units of packagedDrug on hand. Has Received=9000.0]
[MaterialSupplier.RawMaterialSupplier(RawMaterial): Ever ordered=24700.0; of this, still on ships=1900.0, QA discarded=1410.0, QA released=213
[MaterialSupplier.PacMatSupplier(PackagingMaterial): Ever ordered=24700.0; of this, still on ships=3800.0, QA discarded=1481.0, QA released=19
[MaterialSupplier.ExcipientSupplier(Excipient): Ever ordered=24700.0; of this, still on ships=5700.0, QA discarded=848.0, QA released=18152.0]
[Production.ApiProduction; stored inputs=([RawMaterial:1590.0]). Ever started: 19800.0 + (rework=300.0) = (in the works=2900.0; in QA= 100.0;
[Production line (API): accepted 201 batches, totaling 20100.0]
[Production.DrugProduction; stored inputs=([API:0.0, Excipient:2352.0]). Ever started: 15800.0 = (in the works=3700.0; in QA= 0.0; discarded=1
[Production line (bulkDrug): accepted 158 batches, totaling 15800.0]
[Production.Packaging; stored inputs=([bulkDrug:100.0, PackagingMaterial:8519.0]). Ever started: 10900.0 = (in the works=700.0; in QA= 0.0; di
[Production line (packagedDrug): accepted 109 batches, totaling 10900.0]
[Distributor.Distributor(packagedDrug): Shipping plan=24700.0, has shipped=9000.0, in 7 loads. Of this, 0.0 is still being shipped. Remains or
================================================
```

Explanation of the model

While the PowerPoint slide and the subsequent email discussion with Viswanath, Aman, Ben, and others provided me with a lot of information about the models, I still needed to make a lot of "reasonable choices" on how exactly its elements operate.

The computational model currenty just models just a single type of medication. This should be fine if the underlying business model treats each product independently of each other, as we can simply run multiple instances of the computational model, one per medication. In real life, however, there likely are depedencies or common factors of various kinds, e.g. fingible storage space or production capacity, shared components (the same API can be used in pills or in liquid drugs), the same disruptions can disrupt the supply chains for multuple products etc.

All Java classes mentioned below are in the package edu.rutgers.pharma2 (and NOT in edu.rutgers.pharma, which represents an entirely different business model, provided by Abhisekh at GMU).

Main application class (Demo.java)

The main class. Starts and schedules HospitalPool and PharamCompany.

Units

All order sizes and product amounts in the model are in millions, so 1.9e3 actually represents 1.9 billion of doses of medication.

All time values are in days.

The Pharmacy/Hospital Pool (HospitalPool.java)

Configuration parameters:

```
HospitalPool,intervalBetweenOrders,30
HospitalPool,orderSize,1.9e3
```

The HospitalPool object, as a subclass of `Queue` represents the amount of product that has been supplied to it by the industry (specifically, the Distributor object).

At this point, this amount can be only added to (by push actions from Distributor); it is never reduced, as we do not model consumption (which itself could be driven by epidemic outbreaks etc, for example).

Every now and then, at a fixed interval (intervalBetweenOrders=30) the HospitalPool sends an order of a fixed size (orderSize=1.9e3) to PharmaCompany.

The Pharma Company (PharmaCompany.java)

This is the main class for a series of production elements, whose internal objects cover everything from supply pools to the distribution center.

Config:

```
PharmaCompany,orderDelay,Normal,1.14,0.1
```

The PharamaCompany class is Sink, which represents the functionality of receiving orders from the outside world.

Internally, it contains the following elements:

- Delay orderDelay; -- handles the order processig delay
- MMaterialSupplier rawMatSupplier, pacMatFacility, excipientFacility; -- each of these represents an external supplier of an input material, and a quality testing (QA) step
- Production apiProduction, drugProduction, packaging; -- each of these elements represents a production stage, plus the subsequent QA step
- Distributor distro; -- represents the Distribiution element.

When an order comes in from the outside world (HospitalPool) it is put into a Delay (with the delay distribition orderDelay=Normal(1.14,0.1). (In actual fact, the simulator has day-by-day stepping, so this means that for any single order the delay is usually 1, or, rarely 2 days. We could of course switch to scheduling in real-valued time, but I don't see a great benefit here).

Once an order comes out of the delay ("ripens", MASON DES parlance), the PharmaCompany object carries out the following actions:

- Places an appropriately sized order for the raw material with the Raw Material supplier. At present, the raw material order size is directly based on the size of the drug order. (We have 1:1 ratio of units, i.e. one unit of raw material is needed to make one unit of drug, absent any losses). In a later version, this can be enhanced in several ways:
  - taking into account likely losses (discard of faulty products by QA, at all QA stages), so that we won't be likely to run out of the materials beforer fully fulfilling the order;
  - taking into account any finished product that was over-produced previously (i.e. not "spoken for" by any earlier incoming order), so that the size of the current production batch can be reduced as compared to the current incoming prder;
  - taking into account any raw material or intermediate-stage products (such as the API) that we already happen to have on hand and that's not "spoken for" by the previous incoming orders.
- Places similar orders with the excipient facility and the packaging material suppier.
- Sends an instant message to Distributor (the Distribution center) appropriately adjusting the total amount of finished product that we have been asked to ship out to HospitalPool.

MaterialSupplier.java (rawMatSupplier, pacMatFacility, excipientFacility)

We have 3 instances of this java class in the PharamCompany object. They represent, respectively, the Raw Material Supplier Pool, the Excipients Facility, and the Packaging Material Facility, each with its own QA testing step.

Config:

```
RawMaterialSupplier,delay,Triangular,30,60,90
RawMaterialSupplier,faulty,Uniform,0.02,0.10

PacMatSupplier,delay,Triangular,30,60,90
PacMatSupplier,faulty,Uniform,0.02,0.10

ExcipientSupplier,delay,Triangular,30,60,90
ExcipientSupplier,faulty,Uniform,0.02,0.10
```

Each of these objects has a delay parameter (representing e.g. shipping time), and faulty-portion distribution parameter.

The PharmaCompany object, soon after creating each MaterialSupplier element, links it to a Receiver, which is one of the input buffers of one of the Production elements (discussed below).

At the QA stage, a random number is drawn from the faulty-portion distribution, and is used as the fraction of the product lot (which usually corresponds to the monthly order) that needs to be discarded as faulty. The rest of the lot iss released to the Receiver (a Production step) linked to this supplier element.

Production.java (API Production + API Testing; Drug Production + Drug Testing; Packaging + Packaging Testing)

The PharmaCompany object has 3 instance of the Production class as well. Each one represents one production stage plus the subesequent inspection stage.

Config:

```
ApiProduction,inBatch,100
ApiProduction,batch,100
ApiProduction,batchesPerDay,4
ApiProduction,faulty,Binomial,1,0.1
#--- What fraction of faulty product can be sent back to reprocessing, rather
#--- then discarded?
ApiProduction,rework,0.25
ApiProduction,prodDelay,Uniform,2,30
ApiProduction,qaDelay,Uniform,1,1

DrugProduction,inBatch,100,100
DrugProduction,batch,100
DrugProduction,batchesPerDay,4
DrugProduction,faulty,Binomial,1,0.1
DrugProduction,prodDelay,Triangular,40,80,100
DrugProduction,qaDelay,Uniform,1,1


Packaging,inBatch,100,100
Packaging,batch,100
Packaging,batchesPerDay,4
Packaging,faulty,Binomial,1,0.05
Packaging,prodDelay,Triangular,10,15,25
```

```
Packaging,qaDelay,Uniform,1,1
```

A Production element itself is a Steppable (so that it can be scheduled), and internally contains an array of Queue objects, representing the input buffers for the input ingredients:

```
sim.des.Queue[] inputStore;
```

and two delay objects, representing the production step and the QA step:

```
ProdDelay prodDelay;
QaDelay qaDelay;
```

During the initialization, the input of the input buffers is linked to the output of the QA stages of the appropriate suppliers, so that they can push the materials to these buffers as the pass the QA.

Conceptually, each production element carries out its production in batches of a fixed size (e.g. `ApiProduction,batch,100`). No more than a certain number of batches being started on each day (e.g. `ApiProduction,batchesPerDay,4`), which simulates the throughput the of production line. In our model the throughput is constant (e.g. up to 4 batches per day), simulating the fixed amount of the equipment on the shop floor and the fixed availability of labor. In real life, of course, those can vary, due to breakdowns of the equipment, workers falling sick, failing a drug or sobriety test, leaving the employment, or going on strike; modeling of such disruptions can be added at a later time.

To manufacture a batch of product, a certain amount of each involved input is needed. E.g., for drug production we have `DrugProduction,inBatch,100,100`, meaning that to make a batch (100 units) of the drug, one needs 100 units of the API and 100 units of the excipient. (The application knows which number stands for which input because of the parameters to the constructor of the relevant production object). Accordingly, at each time step (each simulated day), the Production object checks if the input buffers conatins a sufficient amounts of all necessary ingredients to start 1 or several batches, and if such ingredients are available, starts the max possible number of batches (limited by `batchesPerDay`, of course). Programmatically, this is done by putting the appropriate amount of the resource into the ProdDelay.

The output of the ProdDelay is linked to the input of the QaDelay. The QaDelay operates on the "entire batch" principle, i.e. each produced batch is tested and, with certain probability, is discarded, sent back to the production stage for reword (i.e. re-queued in the ProdDelay), or released to the next stage.

What the next stage for each Production element is, is configured during the construction of the production objects by the PharamCompanyt object. They are linked as per our PowerPoint slide, with the last one (Packaging) linked to the Distributor unit.

Distributor.java (Distribution)

Config:

```
Distributor,batch,1000
Distributor,interval,30
Distributor,shipOutDelay,Uniform,1,2
```

The operation of the Distributor object is slightly different from what's written on the PowerPoint slide, simply because I couldn't figure out how that was supposed to work.

The Distributor object is Queue, into which the last stage of the production process (i.e. the Packaging Test) pushes the finished product. The Distributor then decides when ship some of that product out, based on the following rules.

Our Distributor object works as follows. It continuously keeps track of how much product needs to be eventually sent to the HospitalPool (as instructed by the PharmaCompany object upon receipt of orders), and how much it has already sent. It can do sending only on specified days (say, once a month), and only in batches of fixed size. (This can simulate e.g. a company that uses a regularly sailing container ship or container train to send products from Tel Aviv to Elizabeth, NJ, or from Wuhan to Disuburg), and which prefers to send full container loads whenever possible. Accordingly, every now and then (with the interval set by Distributor,interval,30), it checks the "need to ship eventually" amount vs. "ever shipped", to determine the "still to be shipped" amount, i.e. to see if the company still has not fulfillied the shipment plan. If the "still to be shipped" amount is non-zero (i.e the plan has not been fully fulfilled yet), the Distributor checks if the amount of product on hand is enough to fill 1 or more full batches, and if so, ships the max possible number of full batches. Additionally, if the "still to be shipped" includes an "odd lot" (a partial batch), and the Distributor has enough product to supply that odd lot, it does so as well. (In other words, if the company has been ordered 3.5 container-loads of aspirin, then we have to send a half-full container at some point; but we'll only do it after the 3 full containers have been sent).

The "sending" is accomplished programmatically by putting resource into a Delay object (with the parameters Distributor,shipOutDelay,Uniform,1,2); the output of the Delay is connected to the input of the HospitalPool (which is a Queue), so that once the product comes out of the Delay, it incrments the amount of product in the HospitalPool.