Introduction

In this lab, you will create a FactoryWorkbench class. The workers will use this in order to build the Product after they collect the necessary Resources. Only one worker will be able to use this Workbench at any given time. In addition to this, you will lower the amount of starting resources in order to have the FactoryWorkers wait at the FactoryResource until the FactoryStockPerson refills the Resource.

Part 1 – Creating the FactoryWorkbench class

Create a new class called FactoryWorkbench in the client package that extends FactoryObject.

```java
public class FactoryWorkbench extends FactoryObject
```

You will need to create a constructor as well.

```java
public FactoryWorkbench(Rectangle inDimensions) {
    super(inDimensions);
}
```

Lastly, we need to make sure to set the label for the object, as well as set its image.

```java
mLabel = "Workbench";
mImage = ImageLibrary.getImage(Constants.resourceFolder + "workbench" + Constants.png);
```

Now, in the FactorySimulation constructor, add a workbench to the bottom left corner of the factory.

```java
FactoryWorkbench fw = new FactoryWorkbench(new Rectangle(0,mFactory.getHeight()-1,1,1));
mFNodes[0][mFactory.getHeight()-1].setObject(fw);
mFNodeMap.put("Workbench",mFNodes[0][mFactory.getHeight()-1]);
mFObjects.add(fw);
```

Note: Remember to always attach objects to a node, give a String as a key for easy look-up if desired, and add the object to mFObjects.

Now, we should have a workbench in the bottom left corner.
Part 2 – Navigating to the Workbench

Now that the workbench has been placed in the factory, we should have the workers navigate to it once they collect the necessary Resources to make a product.

We will want to place some code in-between where the resources are collected, and when we return the worker to the taskboard in the FactoryWorker’s run() method.

We can go ahead and add the following:

```java
mDestinationNode = mFactorySimulation.getNode("Workbench");
mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
mNextNode = mShortestPath.pop();
atLocation.await();
Thread.sleep(1000); //working
```

You’ll notice though, that multiple workers use the workbench at a time. This isn’t very realistic for the simulation. The factory workers should wait until the Workbench is available and use it one at a time.

Part 3 – Making the workers wait for the Workbench

Since we only want one worker to access the Workbench at a time, let’s create a lock in the FactoryWorkbench class.

```java
private Lock mLock;
```

Initialize it in the constructor.

```java
mLock = new ReentrantLock();
```

We don’t want other classes to be able to access this lock directly, but we do want them to be able to lock and unlock this lock. Create two public methods to help with this:

```java
public void lock() {
    mLock.lock();
}

public void unlock() {
    mLock.unlock();
}
```

Now, we can update the FactoryWorker’s run() method so that the worker must acquire this lock before building the product, and release the lock when it is done.

Also, we should have the worker pathfind to a node next to the Workbench instead. That way, they can wait to acquire the lock, and then move onto the Workbench. Then we will only see one Worker on the workbench at a time.
Note: Notice how we save a reference to the node that contains the Workbench. Then, once we path-find to it, we remove the workbenchNode from the path. Once the worker reaches a node next to the Workbench, it will try to acquire the workbench's lock. Once successful, it will move onto the workbench.

Lastly, we will add an assemble(Product) method to the workbench. Instead of sleeping 1 second for every product, we will sleep an amount based on how many resources it takes to make the product.

```java
workbench.assemble(mProductToMake);

public void assemble(Product mProductToMake) throws InterruptedException {
    for(int i = 0; i < mProductToMake.getResourcesNeeded().size(); ++i) {
        Thread.sleep(500);
    }
}
```

Now the workers will wait their turn before building a product at the Workbench.
Part 4 – Limiting resources

In the factory.txt file, change all the resource counts to 50.

```java
9 -- these are all of the resources in the factory with the quantity of each resource and location
10 Resource|Motherboard|50|5|8
11 Resource|Processor|50|5|2
12 Resource|Hard Drive|50|5|16
13 Resource|Memory|50|5|11
14 Resource|Box|50|5|4
```

If you were to run the factory now, you will notice that the workers take out resources even if a FactoryResource is empty.

Create a new lock in FactoryResource, and a new Condition.

```java
Lock mLock;
Condition isEmpty;

mLock = new ReentrantLock();
isEmpty = mLock.newCondition();
```

Now, before blindly taking resources in the takeResources(int) method, we can check if the amount is available to take first. If the resource amount is available, we can proceed as normal, otherwise we must wait on the isEmpty condition.

```java
public void takeResource(int amount) throws InterruptedException {
    mLock.lock();
    while(amount > mResource.getQuantity()) isEmpty.await();
    mResource.deductFromQuantity(amount);
    mLock.unlock();
}
```

Now the FactoryWorker threads will pause their execution when they can’t do anything.

We need someway however, to wake up waiting threads. Add a signalAll() method call when resources are added by the StockPerson thread.

```java
public void giveResource(int amount) {
    mLock.lock();
    mResource.deductFromQuantity(-1*amount);
    isEmpty.signalAll();
    mLock.unlock();
}
```

Note: The reason we have a while-loop in takeResource(int) method is because when signaled, it is possible that the condition still isn’t met. The reason we are using signalAll() instead of SignalAll() is because it is possible to have two or more workers be able to grab the resources that were added.

You should notice workers halting at resources, and resuming execution when a StockPerson drops off more resources.
In case you feel the factory will take a long time to finish. You can speed things up by adding more starting resources, adding more stockpersons, or decreasing the time interval that the server sends resources to the factory.

In case you want to generate resources faster, modify this line:

```java
public void run() {
    while(true) { //always
        try {
            Thread.sleep(7500); //
            int toStock = Math.abs(rand.nextInt() % resources.size());
            int number = Math.abs(rand.nextInt() % 25 + 1);
            for(ServerClientCommunicator communicator : communicators) {
```
Expand on This

The FactoryWorkers get quite dirty after they make the products (Don't ask us, we're the ones getting the complaint emails from them). Implement a FactoryWashroom that the FactoryWorkers can go to once they are done making the product at the Workbench.

For sanitation reasons, the workers should not all pile on top of the washroom when they are waiting for it. Use the techniques learned in this lab to implement this behavior. The time needed to use the restroom should be proportional to the difficulty of the product that has been made. All you need to make this washroom is in the main lab.

No washroom icon is provided. Make your own!