CSCI 201 Lab 8
Threads

I like fruit baskets because it gives you the ability to mail someone a piece of fruit without appearing insane. - Demetri Martin

Introduction

In this lab, you will add a mailbox and stockpersons (stockpeople?) to help replenish the factory’s resources. The mailbox will extend the FactoryObject class and will be located within the factory. The StockPerson class will extend the FactoryWorker class and will work similarly to the default worker.

Part 1 – Making a mailbox

Make a new class named FactoryMailbox that extends FactoryObject. This class should have a constructor that takes in a Vector of Resources – the resources of the factory.

Warning: import resource.Resource, which is the class we created, not java.util.Resource!

```java
public class FactoryMailbox extends FactoryObject {
    private Vector<Resource> available;

    protected FactoryMailbox(Vector<Resource> deliveries) {
        super(new Rectangle(0,0,1,1));
        available = deliveries;
    }
}
```

Note: Make sure to call the super constructor as shown above. This will ensure the mailbox is drawn correctly onto a node.

We must also set the image and label of the mailbox.

```java
mImage = ImageLibrary.getImage(Classes.resourceFolder + "mailbox" + Classes.png);
mLabel = "Mailbox";
```

We want the mailbox to return a random amount of a resource from the list of available resources. Do this by creating a new method.

```java
public Resource getStock() {
    int toStock = Math.abs(rand.nextInt() % available.size());
    int number = Math.abs(rand.nextInt() % 25 + 1);
    return new Resource(available.elementAt(toStock).getName(), number);
}
```

Note: Make sure to create a new Random instance named rand in the constructor.
Add a mailbox to the factory by creating a new instance of FactoryMailbox in the FactorySimulation constructor and adding it to node 0, 0.

```java
private FactoryMailbox mMailBox;
```

```
//create the mailbox
mMailBox = new FactoryMailbox(mFactory.getResources()); //Mailbox that can stock the factory resources
mObjects.add(mMailBox); //Add this object to be rendered
mNodes[0][0].setObject(mMailBox); //Link this object to node 0,0
mNodeMap.put("MailBox", mNodes[0][0]); //Make it easy to find the mailbox node
```

Now when the factory is run, the mailbox should appear.

![Mailbox image](image.png)

**Note:** Make sure to either remove the wall at location (0, 0) or add the mailbox after the wall to replace it.

Lastly, create a getter for the mailbox so the FactoryStockPersons will be able to access it.

```java
public FactoryMailbox getMailBox() {
    return mMailBox;
}
```
Part 2 – Making FactoryStockPersons

We want the FactoryStockPersons to act similar to the FactoryWorkers, so create a new class that extends FactoryWorker.

```java
public class FactoryStockPerson extends FactoryWorker
```

*Note: By extending FactoryWorker, we also implement Runnable since FactoryWorker implements Runnable.*

You will need to create a constructor for the FactoryStockPerson, and call its super constructor.

```java
FactoryStockPerson(int inNumber, FactoryNode startNode, FactorySimulation inFactorySimulation) {
    super(inNumber, startNode, inFactorySimulation);
    mLabel = "StockPerson " + inNumber;
}
```

*Note: It is important to know what will happen once this occurs. If you look in the FactoryWorker constructor, you will see that a new thread will be created, and the worker will automatically start running.*

```java
FactoryWorker(int inNumber, FactoryNode startNode, FactorySimulation inFactorySimulation) {
    super(new Rectangle(startNode.getX(), startNode.getY(), 1, 1));
    mNumber = inNumber;
    mCurrentNode = startNode;
    mFactorySimulation = inFactorySimulation;
    mLabel = Constants.workerString + String.valueOf(mNumber);
    new Thread(this).start();
}
```

Let's go ahead and add some stockpersons to the factory. Add three stockpersons inside the FactorySimulation constructor.

```java
//create some stockpersons
for(int i = 0; i < 3; ++i) {
    FactoryStockPerson sp = new FactoryStockPerson(i, mFNodes[0][0], this);
    mFObjects.add(sp);
    mFWorkers.add(sp);
}
```

If you run the factory, you will notice that the stockpersons are working! But they aren’t doing their correct job. They still have the behavior of a FactoryWorker.
Part 3 – Implementing the behavior of the stockpersons

Override the run() method in the FactoryStockPerson class. The run method will be fairly similar to that of the FactoryWorker.

The logic is fairly simple. If you don’t have a product to stock, find the node of the mailbox, and go to it. Then acquire a product once the mailbox is available for use. Once you do have a product, navigate to the product in the factory, and increment the amount.

Before anything happens, create a private member variable Resource in your Stockperson called mProductToStock.

```java
private Resource mProductToStock;
```

Then create the run() method as follows.

```java
@Override
public void run() {
    mLock.lock();
    try {
        while(true) {
            if(mProductToStock == null) {
                mDestinationNode = mFactorySimulation.getNode("MailBox");
                mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
                mNextNode = mShortestPath.pop();
                atLocation.await();
                while(!mDestinationNode.acquireNode()) Thread.sleep(1);
                mProductToStock = mFactorySimulation.getMailBox().getStock();
                Thread.sleep(1000);
                mDestinationNode.releaseNode();
            } else {
                mDestinationNode = mFactorySimulation.getNode(mProductToStock.getName());
                mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
                mNextNode = mShortestPath.pop();
                atLocation.await();
                FactoryResource toGive = (FactoryResource)mDestinationNode.getObject();
                toGive.giveResource(mProductToStock.getQuantity());
                mProductToStock = null;
            }
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    mLock.unlock();
}
```

Note: The actual movement of the StockPersons is handled in the FactoryWorker’s update(double) method. You do not need to override this method or make any changes.

Note: Change some of the FactoryWorker’s data to be “protected” instead of “private” in order to access it from the StockPerson.

Note: You will need to implement the FactoryResource.giveResource(int) method or simply use the takeResource(int) method and pass in the negated amount.
Part 4 – Giving the StockPersons boxes

For aesthetic purposes, change the image of the StockPerson depending on if it has a resource to deliver or not.

```java
mProductToStock = mFactorySimulation.getMailBox().getStock();
setImage = ImageLibrary.getImage(Variables.resourceFolder + "stockPerson_box" + Variables.png);
toGive.giveResource(mProductToStock.getQuantity());
setImage = ImageLibrary.getImage(Variables.resourceFolder + "stockPerson_empty" + Variables.png);
mProductToStock = null;
```

Now the StockPersons will look different than the workers, and it will be easy to tell if they have a resource to deliver or not.
Expand on This

Instead of the mailbox giving out infinite amounts of random resources whenever a stockperson asks, generate a **random** Resource every five seconds that goes by in real time, and **store** it in the mailbox. Then, when a stockperson comes to the mailbox, give the stockperson an **opportunity** to take a resource out of the mailbox and deliver it.

You will need the mailbox to be its own thread.

The worker **should** grab a resource type randomly, but if the mailbox is out of the given resource, it should force **the worker to wait until the resource is back in stock**.

Here are a couple of hints to get you going:

1. Check out how FactoryWorkers **get their task assignments**. The behavior of the expanded StockPerson is **similar** to how the Worker’s behave, but it will be different. In other words, do not simply copy paste the run() method.

2. Don’t forget to implement a **lock** in your mailbox, otherwise you will get concurrent modification exceptions. Concurrent modification exceptions are bad. Don’t let them happen.

3. In the FactoryMailbox’s new run() method, it’s going to have to generate a new Resource every five seconds. If only there was a way to get it to **sleep** for that time and not do anything...

4. Don’t forget to **start the mailbox thread** when it is created.

5. Also, don’t forget that when a Stockperson is waiting for a resource, don’t let it look like its holding a box!