“Nothing is particularly hard if you divide it into small jobs.” - Henry Ford

Introduction

In this lab, you will create FactoryRobots that help the workers to collect Resources. Workers will go to a FactoryRobotBin in order to get robots to assign them tasks. We will have a fixed amount of robots available to the workers to help collect resources. The workers will be able to create products faster because resource gathering is split up among multiple threads.

Part 0 – Factory Reorganization

Before we do anything in the lab, we will need to delete two walls.

```java
//for(int i = 0; i < 4; i++)
for(int i = 0; i < 3; i++)
{
    FactoryObject fw = new FactoryWall2(new Rectangle(i,9,1,1));
    mFObjects.add(fw);
    mFNodes[fw.getX()][fw.getY()].setObject(fw);
}
```

This way we won't have any additional pathing problems in our factory.
Part 1 – Creating the FactoryRobot

First make a class named FactoryRobot that extends FactoryWorker.

```java
public class FactoryRobot extends FactoryWorker
```

Create a constructor that simply calls the super constructor, and names the robot.

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

*Note: Also make sure to set the image.*

```java
mImage = ImageLibrary.getImage(Variables.resourceFolder + "robot" + Constants.png);
```

We want the robot to simply pick up a resource and drop it off at a location, so add two variables.

```java
private Resource mResource;
private FactoryNode mReturnNode;
```

Finally, we need to implement the run() method for the FactoryRobot class. We want the robot to have the same functionality in grabbing a resource as the FactoryWorker, so add the following snippet from the FactoryWorker’s run() method.

```java
mDestinationNode = mFactorySimulation.getNode(mResource.getName());
mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
mNextNode = mShortestPath.pop();
atLocation.await();
FactoryResource toTake = (FactoryResource)mDestinationNode.getObject();
toTake.takeResource(mResource.getQuantity());
```

We will then return the robot to the specified return location by appending this code to the bottom.

```java
mDestinationNode = mReturnNode;
mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
mNextNode = mShortestPath.pop();
atLocation.await();
```

Finally, we need a way for a worker to command a robot to get the resource, create a `getResource(Resource,FactoryNode)` method so the robot can get the resource, and return to the specified FactoryNode. `mThread` will throw an unimplemented variable error right now. Don't worry, we will fix this in a bit.

```java
public void getResource(Resource resource, FactoryNode node) {
    mResource = resource;
    mReturnNode = node;
    mThread = new Thread(this);
    mThread.start();
}
```

*Note: We want to control when to actually start the robot. Currently, all factory workers start themselves in their constructor. We will change this.*
Part 2 – Altering the FactoryWorker Class

We need to make the following change to the FactoryWorker Class. Add a protected data member named ‘mThread’.

    protected Thread mThread;

Also, assign a new thread to this member in the FactoryWorker constructor so that it is ready to go.

    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);
        mThread = new Thread(this);
    }

Lastly, create a getter for the thread.

    public Thread getThread() {
        return mThread;
    }
Part 3 – Creating the FactoryRobotBin

We want an area for the robots to be so that the workers can find them. We can create a Class that stores and maintains the FactoryRobots.

```java
public class FactoryRobotBin extends FactoryObject
```

Create a Constructor.

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

Create a Queue of FactoryRobots, and a Semaphore to control how many robots are being taken out.

```java
private Queue<FactoryRobot> robots;
private Semaphore robotPermit;
```

Since the FactoryRobotBin will have no robots in it to start, simply set the Semaphore count to 0.

```java
robots = new LinkedList<FactoryRobot>();
robotPermit = new Semaphore(0);
mImage = ImageLibrary.getImage(Constants.resourceFolder + "robotbin" + Constants.png);
```

Finally, we need to implement the only two functionalities of the FactoryRobotBin. Simply an addRobot(FactoryRobot) method, and a getRobot() method.

```java
public synchronized void addRobot(FactoryRobot fr) {
    robots.add(fr);
    robotPermit.release();
}
```

```java
public synchronized FactoryRobot getRobot() throws InterruptedException {
    if(robotPermit.tryAcquire()){
        return robots.remove();
    } else return null;
}
```
Part 4 – Finishing the FactoryRobot

Now that we have a bin to put the robots into, we can finish the run() method of the FactoryRobot. If the robot does not have a resource, simply have the robot navigate to the bin. Once there, the robot can add itself.

```java
public void run() {
    mLock.lock();
    try {
        if(mResource != null) {
            mDestinationNode = mFactorySimulation.getNode(mResource.getName());
            mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
            mNextNode = mShortestPath.pop();
            atLocation.await();
            FactoryResource toTake = (FactoryResource)mDestinationNode.getObject();
            toTake.takeResource(mResource.getQuantity());

            mDestinationNode = mReturnNode;
            mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
            mNextNode = mShortestPath.pop();
            atLocation.await();
            returnToBin();
        } else {
            mDestinationNode = mFactorySimulation.getNode("RobotBin");
            mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
            mNextNode = mShortestPath.pop();
            atLocation.await();
            ((FactoryRobotBin)mDestinationNode.getObject()).addRobot(this);
        }
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    mLock.unlock();
}
```

We just need to create the returnToBin() method, which will clear the robot's resource, and run itself again. We need to make sure however, that the robot only returns to the bin once the worker is at the robot’s return node. It must wait for the worker to be there if necessary. Create a new Condition ‘shouldWait’ and a monitor variable ‘workerHasArrived’ in FactoryRobot to determine if the robot should wait for the worker.

```java
private boolean workerHasArrived;
private Condition shouldWait;

workerHasArrived = false;
shouldWait = mLock.newCondition();
```
private void returnToBin() throws InterruptedException {
    if (!workerHasArrived) shouldWait.await();
    mResource = null;
    mReturnNode = null;
    workerHasArrived = false;
    mThread = new Thread(this);
    mThread.start();
}

Lastly, create a public method that the worker can use to allow the robot to return to the bin.

public void sendBack() {
    mLock.lock();
    workerHasArrived = true;
    shouldWait.signal();
    mLock.unlock();
}
Part 5 – Adding the Robots to the FactorySimulation

We need to add the FactoryRobotBin to the FactorySimulation. We can do this like any other FactoryObject by adding it to a node in the FactorySimulation constructor.

```java
//Make the FactoryRobotBin
FactoryRobotBin robotBin = new FactoryRobotBin(new Rectangle(3, 9, 1, 1));
mFNodes[3][9].setObject(robotBin);
mFNodeMap.put("RobotBin", mFNodes[3][9]);
mFObjects.add(robotBin);
```

Then, add an arbitrary amount of robots to the FactorySimulation as well.

```java
//Add robots
for(int i = 0; i < 20; i++)
{
    FactoryRobot fr = new FactoryRobot(i, this.getNode("RobotBin"), this);
    mFObjects.add(fr);
    mFWorkers.add(fr);
}
```

Lastly, we need to start all the FactoryWorkers at the bottom of the FactorySimulation constructor since we made alterations to the FactoryWorker earlier.

```java
for(FactoryWorker fw : mFWorkers) fw.getThread().start();
```

When the factory is run, the robots will immediately enter the bin so workers can use them.
Part 6 – Updating the FactoryWorker to use robots

Now that we have robots that can collect materials for the workers, we need workers to use the robots when available.

In the FactoryWorker’s run() method, **after the Worker gets to the resource room door**, navigate to the FactoryRobotBin. This snippet of code should come **right after you acquire a permit to go through the resource room door**.

```java
//find way to the resource room door
{
    mDestinationNode = mFactorySimulation.getNode("Workroom 2");
    mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
    mNextNode = mShortestPath.pop();
    atLocation.await();
    FactoryWorkroomDoor2 fwd2 = (FactoryWorkroomDoor2)mDestinationNode.getObject();
    fwd2.getWorkbench();
}

//AVENGERS I mean... ROBOTS ASSEMBLE!
ArrayList<FactoryRobot> robots = new ArrayList<FactoryRobot>();
ArrayList<Thread> robotThreads = new ArrayList<Thread>();
FactoryRobotBin robotBin;

//Go to the robot bin
{
    mDestinationNode = mFactorySimulation.getNode("RobotBin");
    mShortestPath = mCurrentNode.findShortestPath(mDestinationNode);
    mNextNode = mShortestPath.pop();
    atLocation.await();
    robotBin = (FactoryRobotBin)(mDestinationNode.getObject());
}
```

Now, the worker will attempt to grab a robot for each resource it needs to collect. If it is successful, it will send the robot off to collect the material. Otherwise the worker will go get the material itself since we don’t want workers to sit around doing nothing. After every material the worker collects, it will go back to the FactoryRobotBin to check if there are more robots available.

*Note: We want to keep a separate list of Threads so that we can check if the robot has finished its task.*
Once all the resources are collected, return the worker to the resource room's door. Once the worker arrives, it can signal all the robots to let them know that it has arrived. Lastly, the worker thread joins to the robot threads. Since these are the threads responsible for the task the worker originally assigned, the worker will be able to continue execution once all the robot's tasks are complete.
Your final factory should look like this:
Expand on This

No expansion this week. Study hard for your finals and do your best on your final project!

Some hints to get you started:

1. There are secret, hidden study areas in Doheny Library. Rumors say they are somewhere in the stacks...
2. Electrolyte filled drinks such as half-cut Gatorade and green and white teas boost concentration.
3. Lavender scented items relaxes a person's mind and promotes sleep.
4. Do not forget to eat before a test!
5. Breathe, relax. Everything works out in the end. We believe in you!

Good luck!

- Oceane, Ken, and the CSCI201 Staff