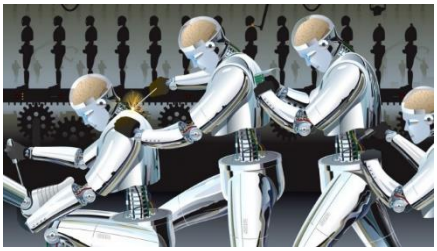
CENG 443 – Object-Oriented Programming Languages and Systems



Spring 2018 – Homework 2

Industry 4.0 – Smart Factory

Selim Temizer

Feedback : Between May 14th and May 18th, 2018
Due date : May 22nd, 2018 (Submission through COW by 23:55)

In this homework, we will build a simulation of a smart factory, where 4 types of robots work in coordination to produce new robots (some of which also join them to speed up the production). Each robot consists of the following 4 parts: a ***Base***, an ***Arm***, a ***Payload*** and a corresponding ***Logic*** chip. Possible payload – logic pairs are [***Gripper*** – ***Supplier***], [***Welder*** – ***Builder***], [***Camera*** – ***Inspector***] and [***MaintenanceKit*** – ***Fixer***]. Robot logics work as follows:

- ***Supplier***: If the production line is not full, randomly generates a robot part and places it in the next available location on the production line. If the production line is full, then empties a randomly selected slot on the production line. Next, wakes up builder robots that have been waiting.
- ***Builder***: Checks all the parts on the production line to see if there is a possibility to fulfill one production step. A production step might be to take an arm and attach it to a base, or to take a payload and attach it to a base-arm, or to finish a robot by adding the correct type of logic (corresponding to its payload), or to pick up a completed robot and add it alongside of working robots (to speed up production), or to store the completed robot if there is enough storage space. The builder logic also signals the factory to stop production if all storage space is full. The builder logic waits (to be notified by a supplier) if there is currently no possibility of fulfilling a production step.
- ***Inspector***: Checks each robot to see if it has lost one of its parts (due to wear and tear). If so, puts that robot in a *broken robots* list, and notifies the fixer robots.
- ***Fixer*** : Checks the *broken robots* list, and fetches the first broken robot from the list (if any). Then, fixes and wakes up the broken robot. If there are no broken robots in the list, then waits (to be notified by an inspector).

Most of the simulation code is already prepared for you. You just need to complete the parts marked with the word **TODO** in the ***Supplier.java***, ***Builder.java***, ***Inspector.java***, ***Fixer.java***, ***Factory.java***, and ***Runner.java*** files to finish up the simulation application. Do NOT use high level concurrency constructs. You should only use threading primitives like *synchronized*, *wait*, *notify*, and *notifyall*. At certain points in code, you may need to access (both get and set) some private instance fields of certain classes (especially, the ***Base*** class). Do NOT create getters and setters for that purpose. You will need to use and practice reflection in your code.

Note that when implemented properly, you shouldn't be getting any exceptions from the simulation at run time. A screenshot and the UML class diagram of a sample implementation are provided in the following part of this document as a guideline/reference.

What to submit? (Use ***only ASCII characters*** when naming all of your files and folders)

1. The 6 java files that need to be completed by you (in a directory named "***Source***").

2. Any other documentation that you would like to add (in a directory named "***Docs***").

There is no need to submit the files that are already implemented (we have copies of them). Just submit the 6 java files mentioned above. We should be able to compile and run your code simply by typing the following:

```
C:\...\Source>  javac  *.java
C:\...\Source>  java  Runner  <optional parameters>
```

Zip the 2 items above together, give the name <ID>_<FullName>.zip to your zip file (tar also works, but I prefer Windows zip format if possible), and submit it through COW. For example:

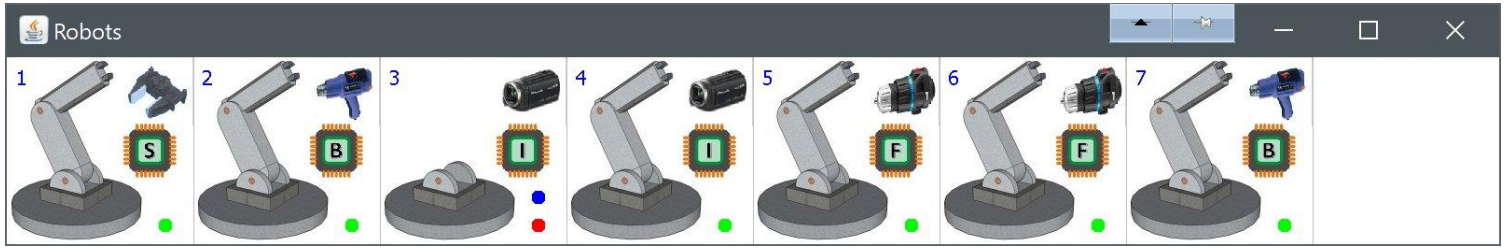
e1234567_SelimTemizer.zip

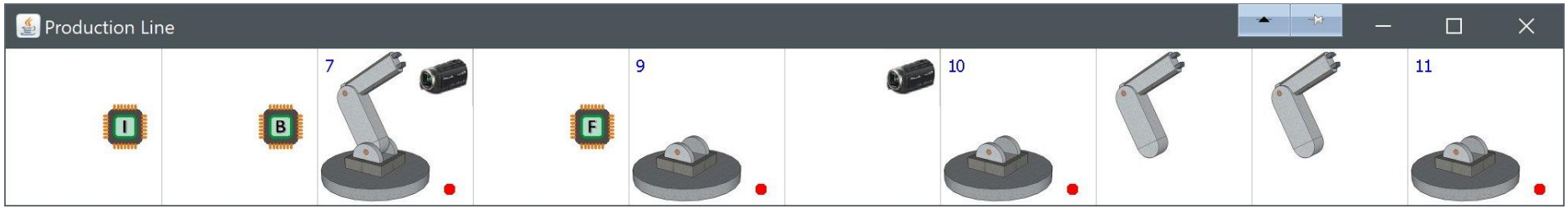
There are a number of design decisions and opportunities for creative extensions that are deliberately left open-ended in this homework specification. We have enough time until the deadline to discuss your suggestions and make further clarifications as necessary. There will be bonuses awarded for all types of extra effort. Late submissions will NOT be accepted, therefore, try to have at least a working baseline system submitted on COW by the deadline. Good luck.

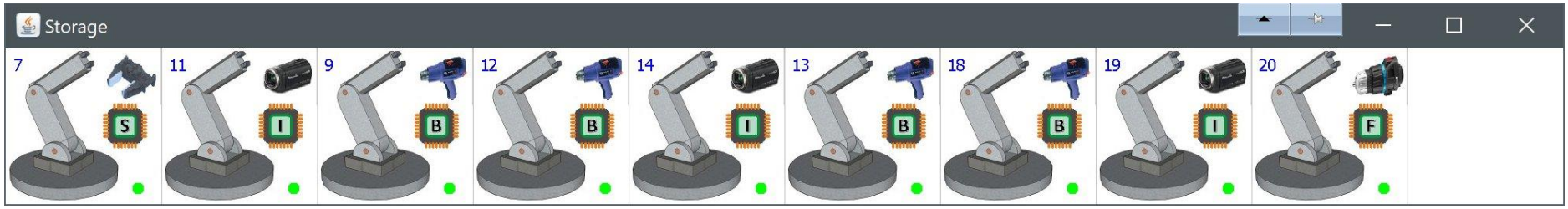
IMPORTANT: Late submissions (even for 1 minute) will not be accepted!

We will only grade submissions on COW, and system closes automatically at due time!

Sample Screenshot







UML Class Diagram