Instruction

1. Using a git command to clone this repository to your computer
2. Open Eclipse and then "import existing project" from the cloned folder (In Step 1) into your Eclipse workspace
3. Rename your Eclipse Java project : **2110215_Exercise3_2017_{ID}_{FIRSTNAME}** (Example: **2110215_Exercise3_2017_5931234521_John**)
4. Implement all the classes and methods following the details given in the problem statement file which you can download from MyCourseVille
5. Check your codes with the provided JUnit Test Cases
6. After implemented all the classes, create a UML Diagram using **ObjectAid** and put the result image (UML.png) in your project folder
7. After finishing the program, export your project into a runnable jar file called **Exercise3_2017_{ID}** (Example: **Exercise3_2017_5931234521.jar**)
8. Commit and Push it to your GitHub repository.

Exercise 3: Abstract, Interface

1. Objective

- Understand abstract class
- Understand abstract method
- Understand Interface
- Understand the difference between abstract class and interface
- Learn to apply interface to class(es)

2. Problem Statement: Factory

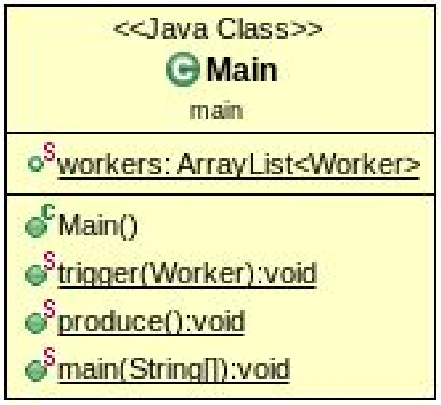
There are 3 types of Bot in this factory; MaterialProcessor, PartAssembler and Packager. Also, there are human workers who work as QualityCheckers to find faulty products.

Working Cycle is as follows, MaterialProcessor's job is processing raw materials into parts. Then, PartAssembler will assemble those parts to create finished product. After that, QualityChecker will check those products, before finally sending them to the Packager to prepare for shipping process.

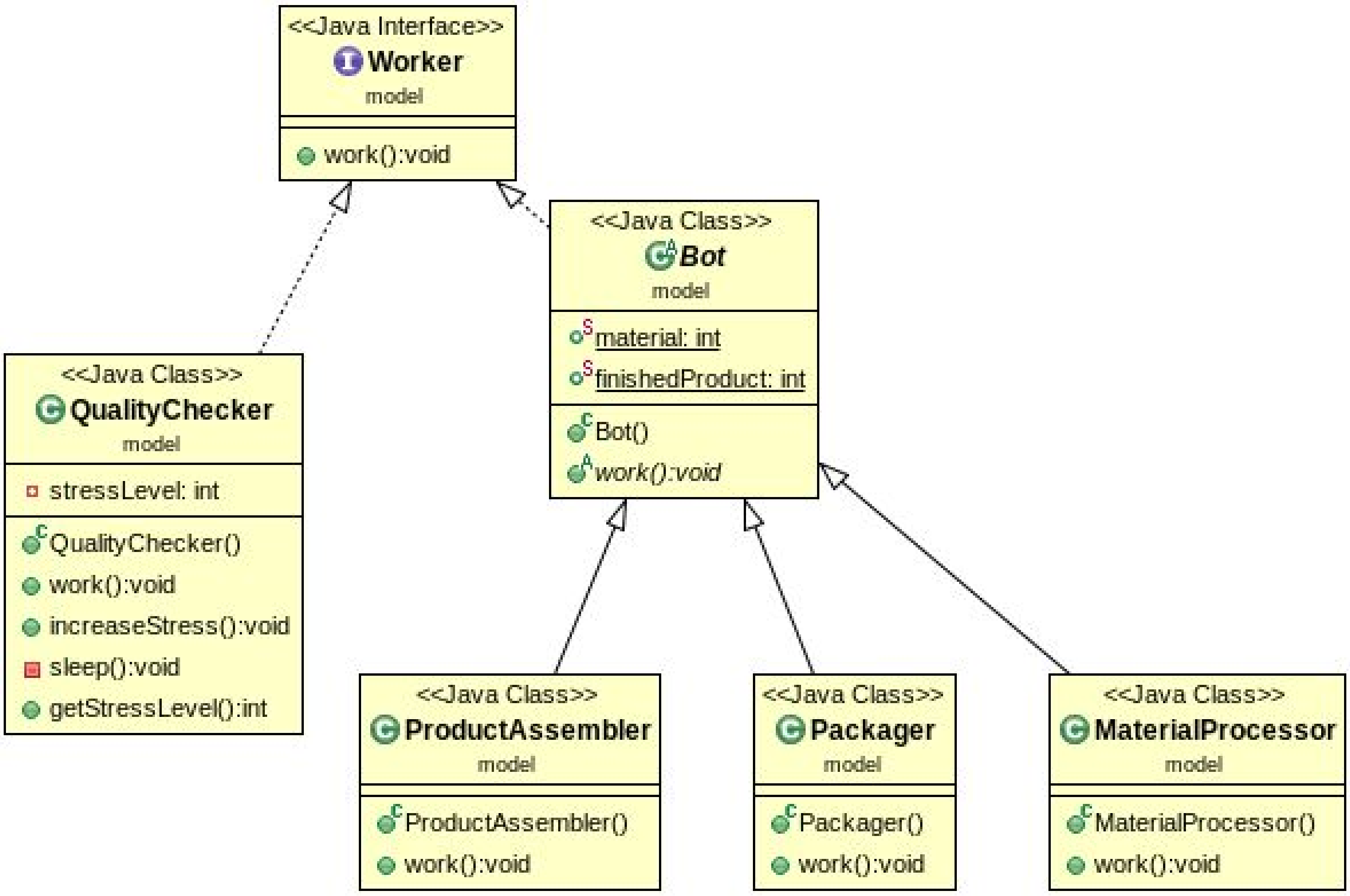
However, human workers cannot work as much as a Bot, so they will get tired after working for a couple or more working cycles. They will secretly take a nap somewhere which will slow down the whole cycle.

3. Class Diagram

3.1 Main package



3.2 Model package



4. Implement Details

4.1 Package : model

4.1.1 Interface “Worker” /*Create and Fill Everything*/

4.1.1.1 Method

+ void **work()**	Do nothing.

4.1.2 Abstract Class *“Bot”* /*Create and Fill Everything*/

Bot inherits work from Worker.

4.1.2.1 Field

+ int **<u>material</u>**	Number of raw material. Default =20. The value is shared across Objects.
+ int **<u>finishedProduct</u>**	Number of finished product. Default =0. The value is shared across Objects.

4.1.2.2 Method

+ void abstract ***work***	Do nothing.

4.1.3 Class "QualityChecker" /*Somewhat Provided*/

QualityChecker is a human worker, however it also inherits work from Worker.

4.1.3.1 Field

- int **<u>stressLevel</u>**	Default =1.

4.1.3.2 Method

- void **sleep()**	When this method is called, the worker will take a nap Reset stressLevel to 1.
+ void **work()**	If this worker's stressLevel is 5 or more, this worker will take a nap. Print "Checking a product" **/*Fill code here*/**
+ void **increseStress()**	stressLevel+1. **/*Fill code here*/**
+ int **getStressLevel()**	Getter for stressLevel. **/*Fill code here*/**

4.1.4 Class “MaterialProcessor” /*Provided*/

MaterialProcessor is a bot.

4.1.4.1 Method

+ void **work()**	Process 2 raw materials for parts.

4.1.5 Class “ProductAssembler” /*Provided*/

ProductAssembler is a bot.

4.1.5.1 Method

+ void **work()**	Assemble parts to create a product.

4.1.6 Class “Packager” /*Provided*/

Packager is a bot.

4.1.6.1 Method

+ void **work()**	Package product. Add 1 to finished product.

4.2 Package : main

4.2.1 Main /*Somewhat Provided*/

4.2.1.1 Field

+ ArrayList<Worker> **workers**	List of all the workers in the factory. Don’t forget to initialize it. **/*Fill code here*/**

4.2.1.2 Method

+ void **produce**()	Every worker in workers **work.** Call trigger() with each worker. **/*Fill code here*/**
+ void **trigger**(Worker)	If the worker is a QualityChecker, call increaseStress() ***Fill code here*/**
+ void **main**()	Instantiate some workers. Add workers to ArrayList. Call produce() to start the working cycle until raw materials run out.

Expected output when you run the completed program

```
Retrieved 2 mats from the warehouse. Remaining: 18
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 16
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 14
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 12
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 10
Assembled a product
Packaged a product
Zzzzz
Checking a product
Zzzzz
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 8
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 6
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 4
Assembled a product
Packaged a product
Checking a product
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 2
Assembled a product
Packaged a product
Zzzzz
Checking a product
Zzzzz
Checking a product
Retrieved 2 mats from the warehouse. Remaining: 0
Assembled a product
Packaged a product
Checking a product
Checking a product
Total Product today =10
```