CSC207 Project Walkthrough

Peijun Ma
Andrew Olechtchouk
Tasbir Rahman
Venkata Ratna Sai Chaitanya Peesapati

March 30, 2017

Extensibility

Our program is as extensible as possible. It does not care for the floor layout nor the procuct type. As long as the set up files are provided it will run. For multiple different products in the same factory, if they can be mixed when loaded, one instance of MasterSystem will be enough to handle all of them. If they needs to be seprated, we can either just use one instance of MasterSystem per product or slightly change the PickingRequestManager class's generatePickingReq method. We did not add any new configuration files.

System boundaries

- The fax machine
- The computer
- The barcode scanners
- The WarehousePicking class
- The trucks

Design patterns

- Dependency Injection in MasterSystem class
- WorkerManager and PickingRequestManager classes are Observers, Worker and PickingRequest classes are Observables
- MasterSystemFactory and TestFactory are factories

Strength and weakness

Strength:
-Our project is very safe, we have exposed very few methods as public methods.
-It doesn't have string comparison anywhere, making all the if statements very concise.
-It is very modular, all the methods are short and we make use of helper methods.
-There is little to no duplicate code.
-It is highly customizable, there are few to none values hard coded.
-It is organized, classes are grouped into different packages.
-It is fast, the algorithms we used are mostly in constant time and at worst linear time, making the asymptotic runtime fast.
-It emulates the real life workflow very well, we have taken the real life situation into consideration when we designed this project.
-It is highly portable. It will run on almost any operating systems.

Weakness:
-The use of implementation inheritance in our Worker class makes testing harder than it should be. In hind sight we should have used composition over inheritance.
-The use of observer observable pattern on Worker and WorkerManager class is questionable, and the code should be more readable without using it.
-We loaded all text files into ram in favour of faster reading and writing, but as a trade off it takes up more space.
-In PickingRequestManager class the loadingArea and pallets data structures are separated. If we had more time, we would have made our own data structure to hold them both.

What are you particularly proud of?

- The getExpectedScan and getToBeScanned method in worker. It makes use of a LinkedList and poping items from the front of the list.
- We do not have a Fascia class, since we are not keeping track of the physical fascias. This makes the system more extensible.
- Our PickingRequest class implements Comparable, making loaders always pick the oldest PickingRequest to load.
- Our tests are really good. We were able to identity and fix bugs via unit tests.
- We make use of TestFactory in our unit tests to make testing easier.

Design decisions

- Replenisher class does not inherent Worker class. Their behavior are different enough.
- PickingRequest are not made until a picker calls ready method.
- A worker needs to ready again if there were no PickingRequest.
- Each rack has a max stock level, and Replenisher cannot add more skus than the max level. This is reflected in the WarehouseFloor.addSku() method.
- If an older PickingRequest failed in any of the steps, it has priority of the ones created after it.