CSC207 Project Walkthrough

Peijun Ma
Andrew Olechtchouk
Tasbir Rahman
Venkata Ratna Sai Chaitanya Peesapati

March 31, 2017

Most Important Classes

Our most important classes are the manager classes. These classes are what allow entities like Worker, Picking Requests, and Orders to interact with the system. The MasterSystem class is the most important class of them all. This class brings everything together.

How Simulation Works

The two classes involved in the simulation are the Main, and Simulator classes. We kept these two separate so that more than one simulator can be called. Everything related to running a simulator is in the Main class. The Simulator class reads events from a file and executes those events into the system. First, all events in the file are placed into an ArrayList of strings where the elements are lines of the file. When running the simulation, we iterate through that list and use regular expressions to see if the line matches the syntax for an event. The first time a worker readies is when they are added to the system. After all events are done, the final.csv and order.csv files are created.

Data Storage

A lot of data is stored in the program. Every file that is read has its content stored as an arraylist of strings where each element is a line in the file. Everything that is to be written to a file is also stored in the same way. Inventory quantities are stored as a hashmap that maps an SKU to its quantity. Workers are stored with a hashmap as well but instead it maps name of worker to worker. This makes reading from the event file simpler.

Extensibility

Our program is as extensible as possible. It does not care for the floor layout nor the product 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 need 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. All subsystems can be accessed in this class. This reduces the number of dependencies a Worker class would have. Instead of a Picker having access to WorkerManager and WarehouseFloor, it can just access the MasterSystem.
- WorkerManager and PickingRequestManager classes are Observers, Worker and PickingRequest classes are Observables. This allows for workers to easily get assigned new picking requests when ready.
- 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.
- We have 100% unit test coverage.

Weakness:
-The use of implementation inheritance in our Worker class makes testing harder than it should be. In hindsight 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 popping 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 behaviors are different enough.
- PickingRequests 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.