Walkthrough of CSC207 Project

Kodiak Jackson, Jason Li, Raymond (Ruize) Liu

Most Important Classes

1) OrderManager class

- It's the class that not only stores and has references to effectively every other object/class, but also contains the main method to run a file. This is by far our most important class.
- It also has the most instance variables and methods to keep track of
- However, it's built to rely on other classes to get the right information and execute precedingly, so it's more like a gigantic intermediary than a single class that does everything – which is good, as it's not as fatal if something breaks inside it

2) Warehouse class

- Stores data and objects similarly but to a lesser extent than order manager. However, it also is absolutely crucial for the system to run. It holds all the information for the translation/traversal tables, as well as the main intermediary between the Reserves & PickFace classes to any other class.
- In addition, it has many useful methods (mainly lookup ones) that prevalent classes like Picker will often call on

3) Picker, Sequencer, and Loader classes

- Aside from Replenisher (which honestly does very little overall), the other 3 Worker type classes are the objects that carry out most important events.
- Picker has the most important methods, and has the highest chance of making a mistake (thus it requires effectively the most monitoring out of the 3), and stores a ton of data in the end
- Sequencer's inspect method is one of its most important features, and required a lot of precision to prevent faulty detection/no detection. However, the rest is quite straight-forward.
- Loader is basically sequencer, but it also has an important method to load Pallets onto a Truck, which is arguably important as that is the final step for us to log and satisfy.

Event File Processing

1. The main method of OrderManager is called, and a new OrderManager object is created. It then immediately gets an orderCopy (the event file) using an orderPath, and also creates a Warehouse object, passing in translation/traversal file paths.
2. After this is set up, the OrderManager object (now abbreviated to ordMan) will call processWorkers.
3. The processWorkers method does a preliminary scan through of the event file, and picks out the necessary workers to create for this particular process. This way, we will definitely have just the right amount of workers for a entire file of jobs.
4. After creating all the Workers using a factory class, the processWorkers method ends.
5. However, all the workers are stored inside an ArrayList. This is extensively used by the workerLookUp method later to find a worker and give them a task.
6. ordMan will then call processOrders, and this is method which processes every single

event from the event file.

7. When a line is read (stripped, and turned into a String Array), ordMan looks for keywords like “Picker” or “Order” to determine the type of event that is happening.
8. The first set of events should be Orders. When this type of event is detected, ordMan calls its orderProcess method, which creates a new Order and stores it in a Queue.
9. Once the system finds the Queue has 4 Orders, it bundles them into a pickRequest and stores it inside another Queue.
10. Once the next event of a Picker being ready arises, the system will use workerLookUp (which literally loops and finds that particular worker from the worker list) to find that Picker. Then, it polls the first pickRequest in the queue and assigns those 4 orders to that Picker.
11. Then, the events keep going until the event Picker picks (some SKU) arisies. Then, the processOrders will call notifyObservers (using the Observer pattern), passing the event string. Since only Pickers have the keyword “Picker” and “picks”, it will realize the event is for them, and begin to pick its first Fascia.
12. To do this, it uses its own pick method, and calls lookUpSku from Warehouse to find the PickFace to pick from, gets the Fascia, and if its correct finishes + calls update on ordMan (this tells ordMan it can move onto the next event).
13. If incorrect, then ordMan will tell it to return the Fascia, and pick with the corrected SKU.
14. Eventually, the Picker will have 8 Fascia. At this point, it waits until the Marshalling event arises.
15. When this happens, a Sequencer will be looked up. The picker will then pass a copy of the correct SKUs and its current 8 Fascia to the Sequencer. The sequencer then uses its inspect method to find if all 8 are correctly picked.
16. If false, then all 8 are discarded, and Sequencer calls reject + pickAgain, prompting the picker to (before any new events happen) repick all 8 with correct SKUs.
17. If true
18. , then the Sequencer awaits the Sequences event. ordMan will look up a Loader first. The Sequencer will then create two Pallets (one front, one back), and give them to the Loader, along with a copy of the correct SKUs (the original version from picker).
19. Then, the Loader event will eventually come. In that instance, the Loader is looked up by ordMan again (it should already have the Pallets), and it uses its own inspect method.
20. If false, then the Loader does similar events to Sequencer. However, if successful, then it either (1) creates a new truck reference and loads to the empty truck or (2) loads to an existing truck under 40 Pallets in capacity. The event then ends
21. If at any point a Replenisher event is detected, ordMan looks up a Replenisher, and tells it to head to reserves and get a particular amount of Fascia *n* (using supply method). It then returns, confirms with ordMan, and finally gives all the Facsia to the correct PickFace(s) (using the refill method).
22. After all events end, there should be the correct logs and files outputted.

Data Storage
There's 3 main types of Data we are tracking, which are…
(1) Event Data (e.g. the data from 16orders)
(2) Translation-type Data (e.g. translation.csv, traversal_table.csv)
(3) Object Data (e.g. Workers, Pallets…)

[Event Data]
- Read by FileProcessor and returned as a 2D String Array. This data is then stored inside an ArrayList within the OrderManager, and processed Array by Array.

[Translation-type Data]
- Also read by FileProcessor, however, it directly sets the data inside translationData and traversalData, two 2D String Arrays stored inside Warehouse. OrderManager has access to them through Warehouse, but the true copy is only stored inside Warehouse.

[Object Data]
- Overall, it's stored within many places. However, OrderManager and Warehouse contains the most comparatively.
- OrderManager stores all Worker objects, the Warehouse, and all Orders (temporarily at least)
- Warehouse stores the Reserves room, as well as all the PickFaces
- Both Reserves and PickFaces store Fascia, and both Pickers and Sequencers can store Fascia
- Both Sequencers, Loaders, and Trucks can store Pallets (which also stores Fascia)

- Most of our data is stored either in 2D String Arrays, ArrayLists, or Queues. ArrayLists are the most prevalent one, and tend to store concrete & permanent data, whereas queues were used more as a temporary storage medium.

Design Patterns
[Observer Pattern]
- The main pattern which our whole system revolves around. Events processed by the OrderManager is automatically notified to Workers (which use observer), and using keywords to pinpoint which Worker should act – it automates the whole process very well.
- In addition, this makes it much harder to make a mistake compared to before.
- Furthermore, this decreases dependency of the Workers on each other like before, and instead just forms loose relations with the OrderManager. In this sense, it's much easier to introduce a new Worker and integrate them into the system.

[Factory Pattern]
- This was the pattern which maybe wasn't as necessary, but still useful nonetheless.
- It allows dependency injection, and obscures the various objects that are required by the OrderManger, only requiring a keyword for a particular worker.
- This way, it will be easier (once again) to add a new type of Worker in the future

Unit Test Coverage: 94%