Warehouse

Group 0368

Overall Structure

System vs. Simulation

System:

- Accepts inputs (as strings)
- Parses strings and updates the states of the software representation of the warehouse
- Tracks locations and statuses of items, orders, and workers

Simulation:

- Orders.txt
- A single text file that collects all possible input from:
  - fax machine (orders)
  - workers (barcode scanners and punch-clock when they sign in)

Core Classes

- WarehouseSystem
  - Constructs everything, Main method
- JobManager
  - Parses orders and turns them into Jobs
- Location & WarehouseItem
  - Representations of locations and items in the warehouse, respectively
- Job
  - Stores data of four orders, passed between ProcessManagers throughout the warehouse
- *<<abstract>> ProcessManager*
  - Handles each step of process & its associated workers
  - Concrete: InventoryManager, PickingManager, SequencingManager, LoadingManager
- *<<abstract>> Worker*
  - Simulates the human worker's task within WarehouseSystem to track WarehouseItems
  - Concrete: Picker, Sequencer, Loader, Replenisher

Helper Classes

- Zone
  - Collection of locations, for quick searching
  - Used to organize locations so that zone names can be flexible
- FileHelper
  - Reads from input files and handles logging
- Pallet, Truck
  - Models the real-life pallet and trucks, to track item location and ordering

Walkthrough

Warehouse Setup

- WarehouseSystem constructor
  - Constructs JobManager and three ProcessManagers that constitute the system
- JobManager makes a translation hashmap mapping orders to skus
  - translation.csv -> FileHelper -> JobManager.skus
  - **Extensibility:** Changing input file changes sku location assignment
- InventoryManager makes the warehouse floor using input files
  - initial.csv & traversal_table.csv -> FileHelper -> InventoryManager.floor
  - Inventory_dimensions.csv -> assigns dimensions for # aisles, racks, levels, etc.
  - Creates Zones and creates Locations within Zones
  - **Extensibility:** Changes in input files changes floor

Four Orders Arrive -> Job

1. Orders are parsed by WarehouseSystem and sent to JobManager
2. JobManager collects 4 orders -> 8 skus -> creates a Job -> PickingManager
    a. Includes pickingOrder Location[]
3. PickingManager finds a ready Picker and gives them the Job
    a. If no Picker available, Job is added to jobsToDo queue
4. Picker gets location info and sends instruction to human worker

At this point, each Job is handled via a pattern of Input -> Processing -> Output.

Handling Each Job: Input -> Processing -> Output

All ProcessManagers follow the same pattern for worker job completion:

1. **Input:** Worker scans barcode -> string input through WarehouseSystem
2. WarehouseSystem alerts corresponding ProcessManager
3. ProcessManager checks input against Worker's current job and task
    a. If **worker error**: throw exception -> either **rescan** or **discardJob()** and restart at head of job queue
    b. If correct: Job's attributes are updated to reflect the task (e.g. added WarehouseItem)
4. ProcessManager alerts WarehouseSystem to update any other Managers
    a. E.g. InventoryManager: remove item from Location when picked
5. **Output:** Next instruction is sent to the worker
6. If this is the final sku in the Job, WarehouseSystem passes Job to next Manager
    a. If Job was loaded, then remove from JobManager's queue

Software Boundaries & Input Exceptions

Checks are in place to ensure input files don’t conflict with each other.

- Inventory Dimensions
  - The numbers of aisles per zone, racks per aisle, levels per rack
  - Maximum capacity per level & minimum capacity before replenish request
- Traversal Table
  - Non-Existent Locations (vs. inventory dimensions)
- Initial.csv
  - Overstocked Locations

Design Patterns

Model-View-Controller

Problem: A means to interface with human workers and update data accordingly

- **Model:** Job, Worker, Location
  - Store representations of warehouse data
- **View:** Log (console, for now)
  - Barcode reader instructions and event log output
- **Controller(s):** WarehouseSystem, ProcessManagers, JobManager
  - Parse input from orders.txt to update model
  - Manipulate model to output info to update view

Dependency Injection

Problem: Updating Job information between independent managers.

Injector	Content	Context
WarehouseSystem	Job	Each Worker and ProcessManager needs access to Job data
WarehouseSystem	Locations in InventoryManager.floor	Picker & PickingManager must update inventory levels once something is picked

Extensibility

Rearranging Warehouse Layout

- Inventory Dimensions
  - The numbers of aisles per zone, racks per aisle, levels per rack
  - Maximum capacity per level & minimum capacity before replenish request
- Traversal Table
  - Location of SKUs
- Translation Table
  - Orders and SKUs
- Initial.csv
  - Amount of product at each Location
  - Can be initialized from final.csv (next day in warehouse)

Diversifying Product Range

To add other products to the warehouse's line-up:

- Add product to translation table
- Edit or create method for reading the translation table
  - Each column = a product attribute
  - Each row = a unique product type
- Update skus per order:
  - Change one line of code in createJob()

Major Design Decisions & Changes

Make Abstract ProcessManager & Worker

Problem:

- Unique workers for each process in warehouse
- Each process has its own queue of tasks to be done
- Type of task queue must correlate with the worker's job

Our Solution:

- Avoid multiple queues inside one "master" class
- Avoid having to first find out worker type/task and then find the job in a queue
- Encapsulate worker types

Aisle, Rack, Level -> Zone and Location Objects

Problem:

- Originally had bloated cascade of classes to store the inventory

Our Solution:

- Consolidated to one single Location object that store these index values as well as the sku and inventory.
- Store all Locations in an ArrayList and use requested aisle, rack, level numbers to calculate the index of a location (avoids tedious iteration)
- Allows for immediate picking and restocking of Location without many getters
- Kept “Zone” object for flexibility of String name
    - Rather than having to use a static alphabet list, and limit zone name it single-digit letters

Critiquing the Design

Weaknesses

- Could have used strategy pattern to create an interface for each worker's nextTask()
  - Instead of abstract Worker and concrete Picker, Sequencer, etc. classes
- Lack of flexibility in warehouse dimensions
  - Every zone must have same dimensions (# aisles, racks, levels)
- No check for presence of all SKUs from translation table in traversal table
- Limited flexibility (requires editing private code) for adding item types

Strengths

- Clear chain of events to handle and log each step of order processing
- Encapsulation of attributes and methods, with minimal visibility
- Abstraction of ProcessManager and Worker allows for adding new types of processes and workers in client code
- Non-static Main method to run in multiple warehouses on same LAN
- Extensibility of warehouse layout via input files
- Exceptions to handle flawed input files and worker scanning mistakes

Test Coverage: System Boundaries

System Boundaries

- Inventory dimensions & capacity
- Job capacity (number of items, pallets, orders)
- Worker capacity (not multiple jobs)

Worker Mistakes

- Wrong SKU scanned
- SKUs out of order