Quentin Proulx

Convenience Store

Deliverable #1

The Scenario:

- This project operates as a convenience store management system. It will manage all the items and customers such that the store knows how many products to order and how much money to spend. It will manage the employee discounts, and whether a customer of a certain age is allowed to buy a product. It will also manage the utilities that the store chooses to buy. It will handle both rented and bought utilities and will manage their costs to the store. This can be applied to as many stores as necessary.

- The purpose of this program is to aid the management of a convenience store so that they're able to know exactly when and where their money is going and change what is being ordered to maximize their profits.

Design Paradigm

This section provides the specifications of the project*

Expected Output & Hierarchies

Interface

Returnable: This is for objects that can be returned for a price that is equal to or lower than what they paid for it.

Customer (abstract)

- Search for products (returns products with keyword and number of them)
- Return item to the store
- Purchase a product in the store, gives discount if employee (abstract)

Adult

- Purchase any product in the store

-

Minor

- Purchase only general products in the store

Store

- Hire somebody (adds them to a list of employees)
- Fire somebody (removes them to a list of employees)
- Calculate amount spent per year
- Acquire item
- Sell item

Item (abstract)

Utility

Purchased Utility

Rented Utility (Returnable)

- Calculate the return value of this item
- Calculate total amount paid for this item

Product

Adult Product

General Product (Returnable)

- Calculate the value of the object (initial price minus some degradation value)

More Specifications

Runtime Polymorphism:

You can find runtime polymorphism in the purchasing of items in the Customer class

- The customer has a purchase method which must be overridden to produce different results in the Adult versus Minor classes

Text IO:

You can find Text IO in the receipt system within the Customer class

- Upon receiving a receipt file, it must be scanned and verified as valid using Text IO so that it can be returned
- Upon having sold a product, a code must be added to a file so that the receipt can be checked in the future (the code must have a date, so it can be verified that it hasn't been more than 30 days since the purchase occurred)

Comparator Classes:

The Customer class should implement Comparator since it requires different sorting strategies depending on what type of customer it is

- The Customer class can be sorted differently based on whether it is an employee or not (it is determined to be an employee through a field in the class)

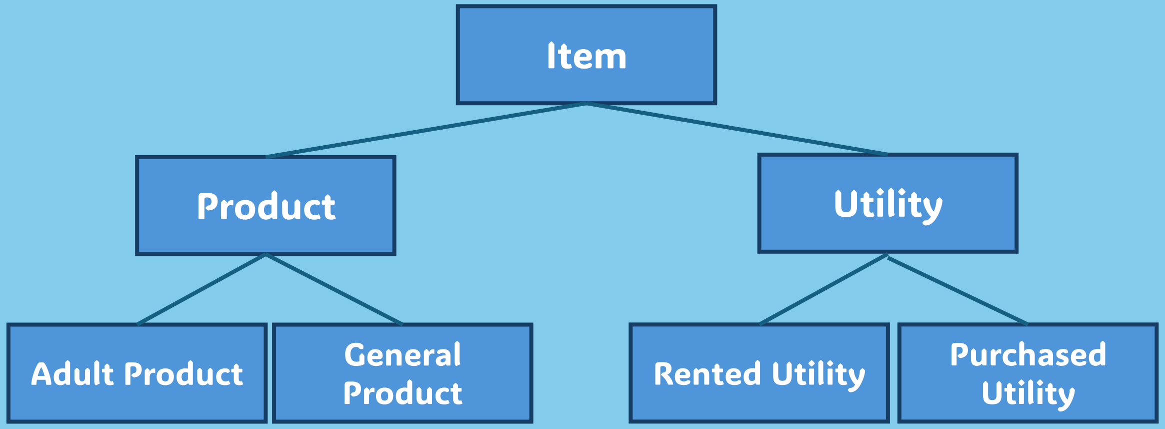
Comparable Classes:

The Item class should implement Comparator since it does not need any sorting strategies

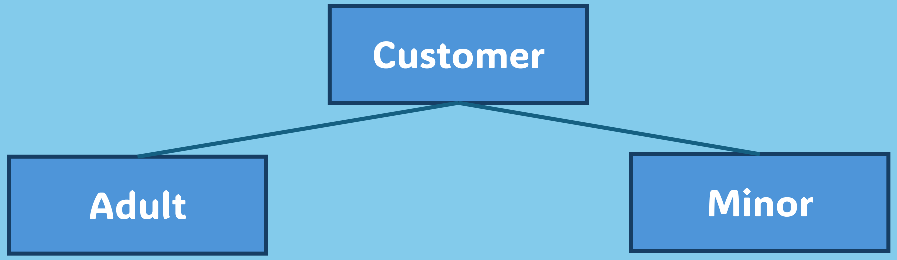
- Sorting them based off something like price would suffice for this class since there aren't any fields which would require a different means of sorting

Class Diagram

Item Inheritance



Customer Inheritance



Partial Implementation Phase

For Deliverable #2:

The general hierarchy structure must be complete with Customer, Adult, Minor, Item, Utility, Product, Rented Utility, Purchased Utility, Adult Product, and General Product all implemented. The Store class must also be finished. Customers should be able to purchase items, and the store should be able to acquire items. The returnable interface and methods should be implemented. The Comparator and Comparable interfaces for both the Customer, and Item subclasses must also be complete. The methods in the Store class should be functional.