

Course: 420-SF2-RE PROGRAMMING

Section: 00001

Final Project Report – Convenience Store Management System

Submitted by: Quentin Proulx (6324569)

Semester: Winter 2025

Teacher: Yi Wang

Vanier College

May 10th, 2025

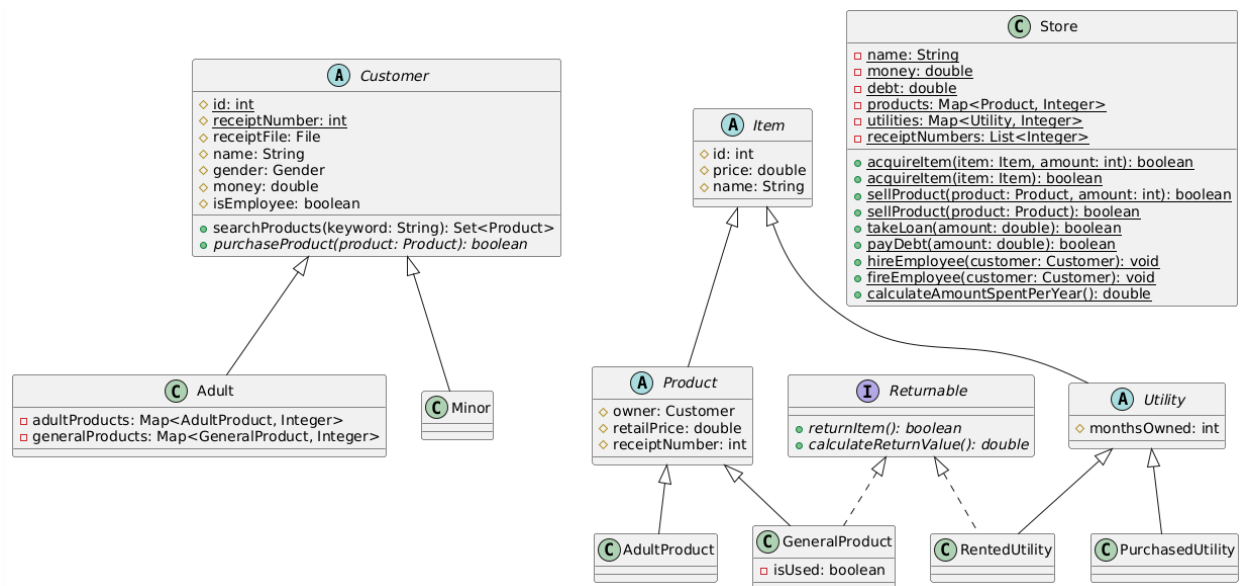
Table of Contents

Project Description.....	p.3
Program Features.....	p.3-5
How the Project Meets the Requirements.....	p.5-7
Screenshots.....	p.7-8
Challenges.....	p.8
Learning Outcomes.....	p.8-9

Project Description

This project operates as a simulated convenience store management system (backend). You can simulate the store's buying and selling products, simulate customers using and returning products, and simulate different types of products that may only be sold to a group of people above a certain age. You can also simulate the costs of different utilities that the store needs to buy and rent to operate. The purpose of this program is to aid the owner of a convenience store to maximize the profitability of the store.

UML Class Diagram



Program Features

The Store:

The store is an essential element of the program since it contains the supply of which every customer is to buy products from. Hence, it is very important to understand how it functions and know that it works.

Method 1: acquireItem()

This adds either a **Product** or a **Utility** to the **Store**'s supply of **Products** or **Utilities** to be bought or used by the store (in the case of **Utilities**)

Method 2: sellProduct()

The Store cannot sell its Utilities as they will be used immediately upon their purchase, and Rented Utilities shall be returned via the Returnable interface. However, the Store should have the option to sell products back to the supplier or other Stores for the original price (NOT retail price). That is the purpose of this method.

Methods 3 & 4: takeLoan() / payLoan()

The Store has the option to take a loan from the bank, however the net loan can not be over \$10,000; if it is, then they have an obligation to pay it back to whoever lent the money. These two methods are to handle the Store's debt and their ability to pay it off.

Methods 5 & 6: hireEmployee() / fireEmployee()

Employees of the Store are permitted an employee discount, and the Store must have employees to run. The hireEmployee() and fireEmployee() methods are to handle the Customers that work for the Store.

Method 7: calculateAmountSpentPerYear()

This method calculated how much the Store spends on Rented Utilities per year. This is to help the store keep track of its current profits vs. its untapped losses which could easily cause the Store to go into debt.

The Customer:

The Customer is another extremely important class within the program since they are the ones funding the Store and displaying a lot of what the program is designed to do in terms of runtime polymorphism and abstraction.

Method 1: searchProducts()

Takes a keyword and returns a set of every Product that includes the name of it within its name. This method is case insensitive, so it doesn't matter what case the letters of the name are in.

Method 2: purchaseProduct()

This is an abstract method which acts differently for both Minors and Adults. The key difference is that Adults can buy certain products which Minors can't (e.g. beer, cigarettes, lottery tickets, etc.) This method also produces a receipt to be written in this specific Customers receipt file (a file which is created in the resources folder upon the creation of the Customer). This receipt contains an ID which will later be used to return the product. Note that this adds a Product to the Customer's products map and removes a Product from the Store's product map.

Returnable:

This is an interface which is applied when an Item is returnable. It contains two methods which act differently depending on which class is using it. Each instance within this project is somewhat intuitively evident.

Method 1: returnItem()

This returns the Item to where it came from whether that be within the Store or within a third-party Store (in the case of the store returning a Rented Utility)

Method 2: calculateReturnValue()

Calculates how much value (positive OR negative) the Item is worth upon returning. For example, the Customer returning a product will be gaining money, however the Store returning a product of which it will have been renting will be losing money upon returning it.

How the Project Meets the Requirements

Hierarchies:

Customer (Superclass)

 Adult (Subclass)

 Minor (Subclass)

Item (Superclass)

 Product (Superclass)

 GeneralProduct (Subclass)

 AdultProduct (Subclass)

Returnable (Interface)

TextIO:

Customer:

Upon instantiation, every Customer has their own csv file in the resources folder with an ID as the name. This file contains all their receipts which are used to verify if they can return a General Product.

purchaseProduct():

This method writes a receipt in the Customer's csv file containing a receipt id which is also added to the Store to make sure that this specific receipt can't be used more than once.

GeneralProduct:

Since this Product, in specific, is Returnable, it must contain the returnItem() method which checks to see if its owner contains a valid receipt so that this Item can be returned.

returnItem():

This method reads every receipt in the Customer's csv file and compares it to the id of the item and checks whether the store contains the same receipt id as the Item, the Customer, and the Store. If so, the Product gets returned a refund is issued.

Comparable & Comparator:

Comparable:

The Item class is sorted via a comparable because sorting the Items by price will always be a reliable method of sorting.

Comparator:

The Customer class is sorted via a comparator because it should be sorted differently depending on whether the Customer is an employee or not.

Unit Testing:

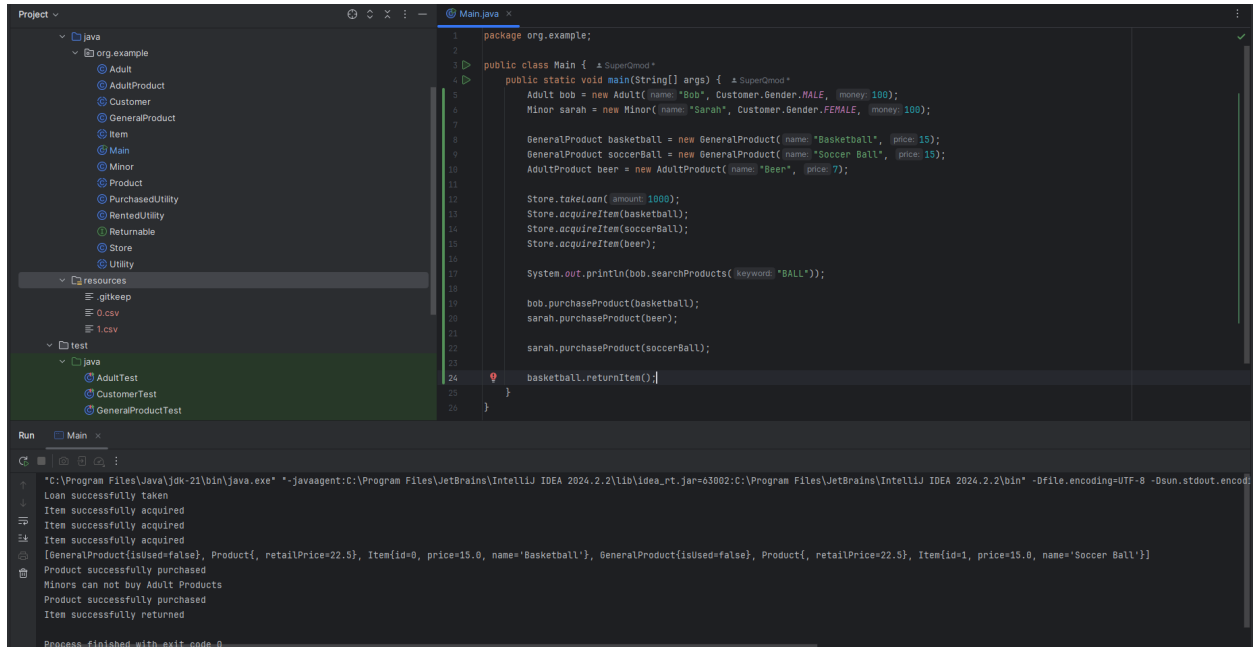
Every method is Unit Tested thoroughly (General cases and edge cases)

Stream:

The Customer uses Stream in the searchProducts() method. It filters the products and returns a set of all the products with a certain keyword in their names.

Screenshots:

General Execution and Output Examples



The screenshot displays an IDE with a project named 'org.example'. The left sidebar shows a package structure with classes like Adult, AdultProduct, Customer, GeneralProduct, Item, Minor, Product, PurchasedUtility, RentedUtility, Returnable, Store, and Utility. The main editor shows the 'Main.java' file with the following code:

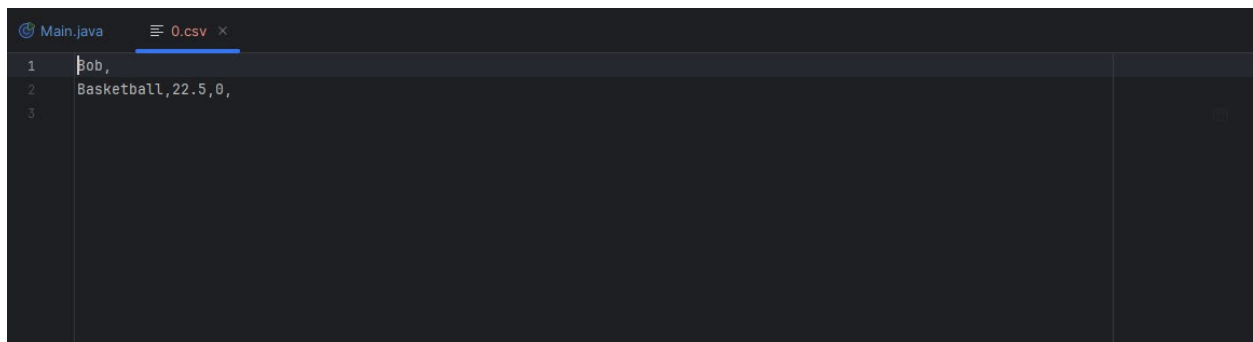
```
1 package org.example;
2
3 public class Main {
4     public static void main(String[] args) {
5         Adult bob = new Adult(name: "Bob", Customer.Gender.MALE, money: 100);
6         Minor sarah = new Minor(name: "Sara", Customer.Gender.FEMALE, money: 100);
7
8         GeneralProduct basketball = new GeneralProduct(name: "Basketball", price: 15);
9         GeneralProduct soccerBall = new GeneralProduct(name: "Soccer Ball", price: 15);
10        AdultProduct beer = new AdultProduct(name: "Beer", price: 7);
11
12        Store.takeLoan(amount: 1000);
13        Store.acquireItem(basketball);
14        Store.acquireItem(soccerBall);
15        Store.acquireItem(beer);
16
17        System.out.println(bob.searchProducts(keyword: "BALL"));
18
19        bob.purchaseProduct(basketball);
20        sarah.purchaseProduct(beer);
21
22        sarah.purchaseProduct(soccerBall);
23
24        basketball.returnItem();
25    }
26 }
```

The Run window at the bottom shows the following output:

```
"C:\Program Files\Java\jdk-21\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2024.2.2\lib\idea_rt.jar=63802:C:\Program Files\JetBrains\IntelliJ IDEA 2024.2.2\bin" -Dfile.encoding=UTF-8 -Dsun.stdout.encoding=UTF-8
Loan successfully taken
Item successfully acquired
Item successfully acquired
Item successfully acquired
[GeneralProduct{isUsed=false}, Product{, retailPrice=22.5}, Item{id=0, price=15.0, name='Basketball'}, GeneralProduct{isUsed=false}, Product{, retailPrice=22.5}, Item{id=1, price=15.0, name='Soccer Ball'}]
Product successfully purchased
Minors can not buy Adult Products
Product successfully purchased
Item successfully returned
Process finished with exit code 0
```

Note: 0.csv and 1.csv are the csv files created upon the creation of the Adult and the Minor

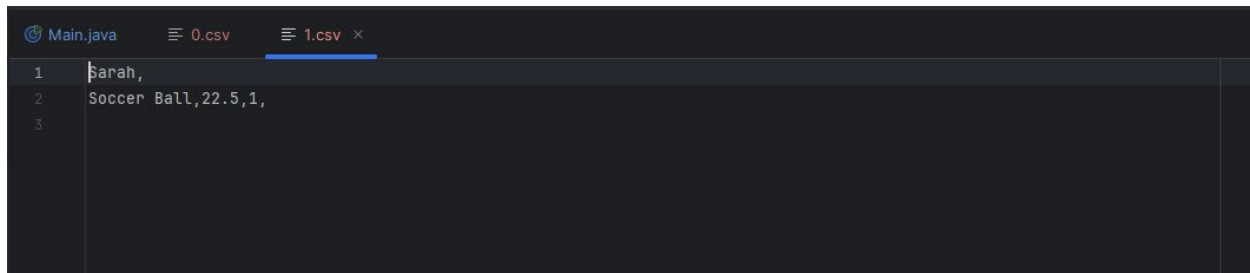
0.csv



The screenshot shows the '0.csv' file with the following content:

```
1 Bob,
2 Basketball,22.5,0,
3
```

1.csv



```
1 Sarah,  
2 Soccer Ball,22.5,1,  
3
```

Challenges

Challenge #1: The first major challenge I found myself facing during the course of this project was the initial organizational step; I found myself lost trying to figure out how everything would work out in the end at the beginning, and I didn't know how I would implement certain features (particularly the Returnable interface).

Challenge #2: I initially wanted the Store class to not be static, but I found that it would have made implementation of every other feature astronomically more difficult if I had made it so you were able to manage more than one Store. I think if I were to have redone the project with more time, I would have probably tried to find a way to work that in with the rest of the features.

Challenge #3: The purpose of this project was to act as a budget management system for a Convenience Store, however there are numerous elements that I didn't have time to finish in regards to budget management features. For example, I would've loved to make it so that the Store could calculate how much every employee had to have been paid, but I didn't think that far until the project was well over with (the only changes I made after the due date were readability enhancements and small bug fixes). This would have lead the way for other methods that would've helped the Convenience Store manage it's budget such as a "calculateTotalLossesPerYear" method.

Learning Outcomes

Gain #1: I learned the importance of project organization throughout the course of this project. Going into it, I only had a very vague idea of what I wanted to accomplish, and that made it very difficult for me to implement everything I wanted to implement efficiently. From now on, I think I am going to allocate a lot more time on the initial 'planning' stages of my projects than I did for this one.

Gain #2: I am now much more proficient in using GIT after this project and have learned different shortcomings I had been experiencing before (such as, for example, committing pushing unrelated changes in one go, making it harder for people to understand what I've changed).

Gain #3: I learned to overcome different bugs and glitches that stemmed from the structure of my program as opposed to simply one logical error in one method. This improves my debugging methodology and makes it so that in the future, I will be able to handle problems in my code much more efficiently (especially when it comes to issues that come from different classes interacting with each other).