Music Store Management System
Project Overview

This project is a **Management System for a Music Store**. It’s designed to handle the basics of running a store, like keeping track of inventory, orders, and employees.

It includes a login system with different roles for each user:

- **Employee**: Can take orders, manage inventory, etc.
- **Admin**: Has full access, including managing users.
- **Accountant**: Can view stats and financial info for the store.

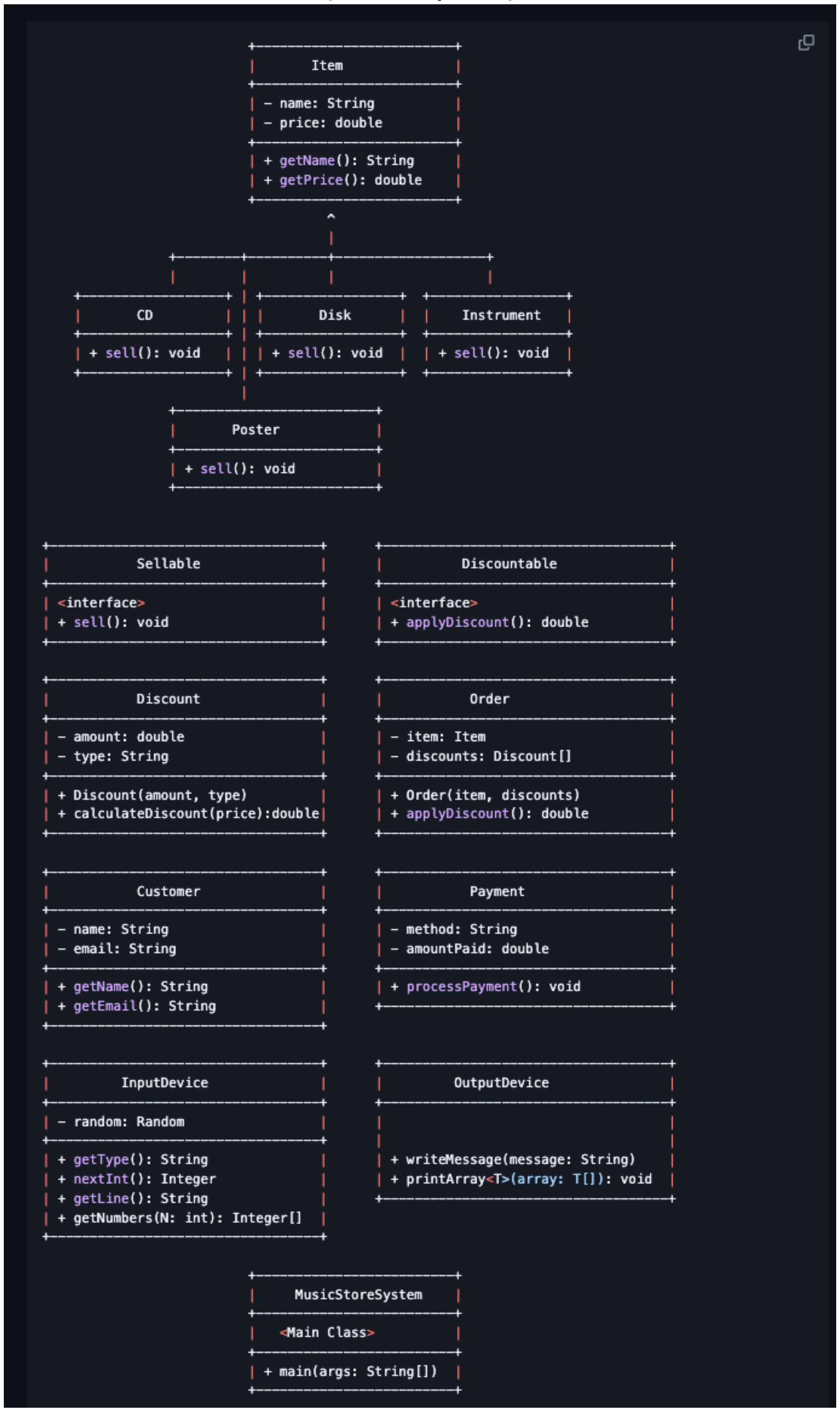
Demo time !
(switch to IntelliJ)

Development Journey

The first thing I did was imagine what the final project would look like, then break it down into smaller, individual parts. For example, one main feature is **Order** management.

I sketched out some **UML diagrams** to get a clearer picture of how everything would connect, like classes and methods. Here's what Initialy I came up with:

(initial plan)



```
                          +-------------------------+
                          |       Item              |
                          +-------------------------+
                          | − name: String          |
                          | − price: double         |
                          +-------------------------+
                          | + getName(): String     |
                          | + getPrice(): double    |
                          +-------------------------+
                                    ^
                                    |
                +--------+----------+-------------------+
                |        |          |                   |
    +------------------+ | +-----------------+  +-----------------+
    |       CD         | | |       Disk      |  |    Instrument   |
    +------------------+ | +-----------------+  +-----------------+
    | + sell(): void   | | | + sell(): void  |  | + sell(): void  |
    +------------------+ | +-----------------+  +-----------------+
                         |
                +-------------------------+
                |       Poster            |
                +-------------------------+
                | + sell(): void          |
                +-------------------------+


+----------------------------------+      +------------------------------------+
|           Sellable               |      |         Discountable               |
+----------------------------------+      +------------------------------------+
| <interface>                      |      | <interface>                        |
| + sell(): void                   |      | + applyDiscount(): double          |
+----------------------------------+      +------------------------------------+

+----------------------------------+      +------------------------------------+
|           Discount               |      |          Order                     |
+----------------------------------+      +------------------------------------+
| − amount: double                 |      | − item: Item                       |
| − type: String                   |      | − discounts: Discount[]            |
+----------------------------------+      +------------------------------------+
| + Discount(amount, type)         |      | + Order(item, discounts)           |
| + calculateDiscount(price):double|      | + applyDiscount(): double          |
+----------------------------------+      +------------------------------------+

+----------------------------------+      +------------------------------------+
|           Customer               |      |          Payment                   |
+----------------------------------+      +------------------------------------+
| − name: String                   |      | − method: String                   |
| − email: String                  |      | − amountPaid: double               |
+----------------------------------+      +------------------------------------+
| + getName(): String              |      | + processPayment(): void           |
| + getEmail(): String             |      +------------------------------------+
+----------------------------------+

+----------------------------------+      +------------------------------------+
|         InputDevice              |      |         OutputDevice               |
+----------------------------------+      +------------------------------------+
| − random: Random                 |      |                                    |
+----------------------------------+      |                                    |
| + getType(): String              |      | + writeMessage(message: String)    |
| + nextInt(): Integer             |      | + printArray<T>(array: T[]): void  |
| + getLine(): String              |      +------------------------------------+
| + getNumbers(N: int): Integer[]  |
+----------------------------------+

                          +-------------------------+
                          |     MusicStoreSystem    |
                          +-------------------------+
                          |  <Main Class>           |
                          +-------------------------+
                          | + main(args: String[])  |
                          +-------------------------+
```

Once I had a plan, I dove into the code.

Challenges and Fixes

Some things didn't go as planned:
For example:

- **Password Hashing**: I ran into an annoying bug with the User hashPassword method, where passwords would accidentally get double-hashed, locking users out. It took a long time to find the issue, but I fixed it by making sure hashing only happens once.

Things that went as planned:

- **User Roles**: For roles (Employee, Admin, Accountant), I created an **Enum** to make role assignments clear and consistent right from user creation.
- **All the methods that handle** user specific functionalities.

Testing

Testing also helped a lot, especially for **constructors** and **file storage** to make sure data saved and loaded correctly.

Project Diagram
(switch to IntelliJ)

Data and Configuration
I went by using JSON files for saving data.

Data is saved to files so it’s there when you reopen the app.

In the future I am planning using a database for storing data.

Exception Handling and Validation

I’ve set up some error handling to manage things like missing files (FileNotFoundException), I/O issues, and others. Also **custom exceptions** that helps in specific situations.

Example:

- InvalidItemException when we add items to the inventory
- *AuthenticationException* used when we try to login the user.

Input Validation

Every input from users gets validated to avoid data issues and unexpected Exceptions.

Future Ideas for the Project

Here's what I'm thinking:

- **Graphical User Interface (GUI) Upgrade**: Currently, the system runs in the terminal, but I plan to transform it into a more user-friendly graphical user interface.

- **Support for any type of item + a way to change Order statuses.**

- **Enhanced Reporting for the Accountant Role**: I'd like to add more detailed financial and sales reports, including monthly revenue summaries and inventory turnover rates, so the accountant can gain better insights.

- **PDF Report Generation**: The Accountant Role should be able to generate PDF reports summarizing the store's monthly performance.

- **Database Integration**: Move data storage to a database for better data management and scalability.

- **Code Structure Improvements**: Organize the code with a more consistent and structured convention.

Any Questions are welcome !