

1. Introduction

MediStock Pharmacy Inventory System is a simple program developed in Java with the intent to meet the requirements of a pharmacy. The main objective of the system is to have necessary stock inventory management such as add stock, record sales and chiefly monitoring medicine expiry date for patient safety and minimized its wastage due to expired.

The project utilizes **Object-Oriented Programming (OOP)** principles and the modern **Java 8 java.time API** for robust date management.

2. System Components and Design

The system is structured using three core Java classes, ensuring a clear **Separation of Concerns**:

Class Name	Role in the System	Key Responsibilities
Medicine.java	Model (Data Structure)	Encapsulates medicine data: name, batch, price, quantity, and LocalDate expiry date. Handles date parsing.
InventoryManager.java	Controller/Service (Business Logic)	Manages the ArrayList of Medicine objects. Contains methods for adding, selling, and checking expiry. Handles file I/O for data persistence.
MediStockApp.java	View (User Interface)	Contains the main method. Handles the command-line menu, user input (Scanner), and orchestrates calls to the InventoryManager.

2.1 Object-Oriented Principles Used

- **Encapsulation:** All fields in `Medicine.java` are private attributes (i.e., not publicly accessible - `quantityInStock`, and `expiryDate`) with public accessors - Getter & Setter.
- **Polymorphism:** The `Medicine` class implemented method that overrides `toString()` to return a formatted name for the Medicine.

3. Core Functionalities

3.1 Inventory Management (addNewMedicine & displayInventory)

- New medicines are instantiated as Medicine objects and stored in an ArrayList<Medicine> within the InventoryManager.
- The system includes a **data persistence mechanism** where the saveToFile() method is automatically called after every successful addition, writing the current inventory to inventory_data.txt.

3.2 Sales and Billing (sellMedicine)

- The sellMedicine method performs a case-insensitive search based on the medicine name.
- **Stock Validation:** It checks if quantityInStock \geq qtyWanted. If not, an "**Insufficient Stock**" error is returned.
- **Transaction:** If stock is sufficient, the quantity is deducted (setQuantityInStock), and a basic **bill receipt** is generated, displaying the item, quantity, and total cost (qtyWanted * pricePerUnit).

3.3 Expiry Safety Check (checkExpiryAlerts)

This is the most critical function, leveraging the java.time API for precision.

1. It determines the current date (LocalDate.now()).
2. For every medicine, it calculates the days until expiry using:

```
daysUntilExpiry = ChronoUnit.DAYS.between(today, expiryDate)
```

3. It generates alerts based on the following logic:

Condition	Alert Level	Status
daysUntilExpiry < 0	CRITICAL	EXPIRED (Recommends immediate discard)
0 \leq daysUntilExpiry \leq 30	WARNING	Expires Soon (Flags for prompt sale or removal)

4. Technical Implementation Highlights

4.1 Date Handling

The system strictly uses the `java.time.LocalDate` class, expecting dates in the **ISO format (YYYY-MM-DD)**. The `Medicine` constructor includes a try-catch block to handle a `DateTimeParseException`, defaulting the expiry date to today if the user enters an invalid format.

4.2 File I/O for Persistence

Data is saved to a plain text file named `inventory_data.txt`. The `saveToFile()` method uses a **try-with-resources** block for safe and automatic closing of the `FileWriter`, even if an `IOException` occurs.

Java

```
// Example of safe file writing in InventoryManager.java

try (FileWriter writer = new FileWriter("inventory_data.txt")) {
    // ... write medicine data
} catch (IOException e) {
    // ... handle exception
}
```

5. Conclusion

The MediStock Pharmacy Inventory System fills the bill for a core pharmacy inventory manager, hinging on its expiry tracking forte. With an structured oops design and modern JAVA APIs makes the code-base well organized, maintainable and strong enough handle date-sensitive information. The current solution has sample data (including one expired item) which is hard-coded so you can see the Critical Expiry Alert feature in action as soon as an app is opened.