

A
Mini Project Report
On

Inventory management shop

Presented by

YUVRAJ NILESH DESHMUKH
SY[CSE]

Computer Science and Engineering
(2025 – 2026)

Guided By

Ms. N. L. Pariyal

(Department of Computer Science and Engineering)

Submitted to



MGM's College of Engineering, Nanded

Under

Dr. Babasaheb Ambedkar Technological University, Lonere

Certificate

This is to certify that the mini project entitled

Inventory management shop

Submitted By

YUVRAJ NILESH DESHMUKH

**In satisfactory manner as a partial fulfillment of
SY[CSE] in Computer Science and Engineering**

To

MGM's College of Engineering, Nanded

Under

**Dr. Babasaheb Ambedkar Technological University, Lonere
has been carried out under my guidance,**

Ms. N. L. Pariyal

Mini Project Guide

Dr. Mrs. A. M. Rajurkar

H.O.D

Computer Science & Engineering

Dr. Mrs. G. S. Lathkar

Director

MGM's College of Engg., Nanded

1. Abstract :-

The primary aim of this mini-project is to develop a simple, console-based Inventory Management System using Java that simulates the essential operations of a small shop. The system is designed to effectively manage product records, track stock levels by weight (kilograms), calculate the total value of stock, and perform key inventory functions such as adding, displaying, searching, and updating (simulating sales) product records.

2. Problem Statement:-

Small-scale inventory management in shops often relies on manual methods (registers, ledgers) for tracking stock. This traditional approach is prone to human errors, delays in stock reconciliation, and difficulties in calculating the real-time total value of goods. The absence of an automated system makes it challenging to accurately monitor stock levels, especially when items are measured by weight (kilograms), leading to issues like overstocking or stock-outs.

The problem addressed is: **How to design and implement a simple, efficient, and cost-effective Inventory Management System using core Java concepts to digitally record, display, search, and update product inventory by weight, ensuring accurate stock tracking and real-time total value calculation?**

3. Objective of the Project :-

The primary objectives of developing this console-based Inventory Management System are:

- To develop a simple, functional **Inventory Management System using core Java concepts**.
- To allow the user to **store and track multiple product records** (ID, Name, Quantity, Price) using parallel arrays.
- To enable efficient **searching of a product by its unique ID**.
- To implement logic for **updating stock levels** (simulating a sale) and ensuring that sales cannot proceed if stock is insufficient.
- To accurately **calculate the total monetary value of the remaining stock** after a sale.
- To identify and display the **product with the highest stock quantity**.
- To provide a clear, console-based demonstration of real-world inventory tracking principles for academic learning.

4. Scope of the Project :-

The project scope is strictly defined to focus on fundamental inventory management using core Java concepts.

- **Technology & Structure:** The system is built using **core Java** and relies on **parallel arrays** (in-memory) to hold all product data (ID, Name, Quantity in kg, Price).
- **Focus:** The system manages products measured by **weight (kilograms)**.
- **Key Operations:** It covers the primary life-cycle of a product in a small shop's inventory:
 - Inputting new product details.
 - Displaying the entire inventory and its total value.
 - Searching for specific products by ID.
 - Updating stock levels to simulate sales.
 - Identifying the product with the maximum stock.
- **Interface:** It is a **console-based** application, designed to be lightweight and suitable for academic demonstration.

5. Proposed System :-

The proposed system is a simple, automated solution designed to replace inefficient manual inventory tracking.

- **Objective:** To overcome the limitations of manual registers by providing **digital, accurate, and real-time** stock management⁷⁷⁷⁷.
- **Mechanism:** It uses the Java program to digitally map and track each product's attributes in dedicated data arrays⁸.
- **Key Advantage:** It introduces **stock validation logic** during the sale process to ensure the remaining quantity is sufficient before deduction, thus preventing over-selling errors⁹⁹⁹⁹.
- **Financial Insight:** The system provides an instant calculation of the **total stock value** ($\text{Quantity} \times \text{Price}$) for immediate reporting¹⁰.
- **Output:** The system outputs clear inventory records, search results, sale confirmations, and stock summaries to the cons

6. Existing System

Limitations of the Existing System

1. Manual Seat Management:

Traditional movie ticket booking relies on counters or registers, which can lead to errors and confusion in seat allocation.

2. Time-Consuming Process:

Manually checking seat availability and issuing tickets takes more time and becomes inefficient during peak hours.

6. Hardware Requirements :-

The system requires a minimum 1 GHz processor, 2 GB RAM, and about 100 MB free disk space. A keyboard and basic monitor are sufficient to run this console-based application. No high-end hardware is required

Feature	Traditional Inventory System	Computer-Based System (Our Project)
Speed	Slow and manual operations (writing ledgers)	Fast and automated data processing ³
Accuracy	Prone to human errors (miscalculation of total value)	High accuracy using program logic ⁴
Stock Management	Paper-based tracking and recording ⁵	Digital inventory using parallel arrays ⁶⁶
Availability Check	Manual checking of physical stock or register	Real-time stock validation during sale process ⁷
Total Value	Manual calculation	Automatic value calculation (Quantity * Price) 88888888
User Convenience	Difficult to manage and reconcile ⁹	Easy-to-use, console-based system ¹⁰

7. Feasibility Study :-

The Inventory Management System is highly feasible across all critical dimensions:

- **Technical Feasibility:** The system is technically feasible as it uses core Java concepts. It runs on any Java-enabled system and requires no complex networking or advanced libraries.

- **Operational Feasibility:** It is operationally feasible due to its simple input-based interaction. The operations (input, search, sale) are clear and align directly with a shop manager's typical needs.
- **Economic Feasibility:** The project is economically feasible because it utilizes free software tools (JDK) and requires no special hardware investment.

8. System Flow :-

¶ SYSTEM DESIGN

9. Use Case Diagram

Use Cases: Input new products, view full inventory, search product by ID, update stock (sale), find highest stock.

Description: The diagram outlines how the primary actor (the Shop Manager) interacts with the system to perform core inventory management functions, such as adding products, checking the current stock, and simulating sales through a simple console interface.

Program Variables and Their Purpose

VARIABLE	PURPOSE	DATA TYPE
sc	Accepts user input from the console.	Scanner
n	Stores the number of products to be managed.	int
productID[]	Stores the unique identification number for each product.	int [] ¹⁴
productName[]	Stores the name of each product.	String [] ¹⁵
quantityKg[]	Stores the current stock quantity (in kilograms).	double [] ¹⁶
pricePerKg[]	Stores the selling price per kilogram.	double [] ¹⁷

VARIABLE	PURPOSE	DATA TYPE
searchID	Stores the ID entered by the user for searching.	int
soldQty	Stores the quantity entered by the user for a sale.	double ¹⁸

10. Class Diagram

The project uses a single main Java class, **InventoryManagement**, to handle all system operations.

- **Attributes:** The class primarily holds the parallel arrays (**productID**, **productName**, **quantityKg**, **pricePerKg**) and the `Scanner` object.
- **Input Handling:** The `Scanner` object accepts user input for product details and transaction parameters.
- **Logic Section:** This is the `main` method, which contains all the logic for inventory input, stock validation, search, and maximum stock comparison.
- **Control Flow:** The `main` method dictates the sequential execution of the program's functions.

11. Sequence Diagram :-

The sequence diagram illustrates the order of interactions to complete the primary task: Stock Update/Sale.

User :- System: Start Program / Enter n

User :- System: Input All Product Details

System:- User: Display Inventory Records

User : Enter sellID and soldQty

System :- System (Logic): Validate Stock (Check if Qty >= soldQty)

System : Update quantityKg[i] -= soldQty

System :- User: Display Sale Confirmation and New Total Value

System :- System (Logic): Find Max Stock

System :- User: Display Product with Highest Stock

¶ IMPLEMENTATION

12. Introduction to Implementation

The implementation phase focused on developing the console-based Inventory Management System by translating the system design into functional Java code.

- It utilized **parallel arrays** to efficiently link different data types (integer IDs, string names, double quantities, double prices) for a single product record.
 - **Core Java concepts** such as arrays, loops, and conditional statements were used extensively.
 - The primary implementation logic focused on robust **stock validation** (`if (quantityKg[i] >= soldQty)`) to ensure that no negative stock could be generated.
-

13. Program Structure

The project code is structured into simple, logical components within the single `InventoryManagement` class:

- **Input Module:** Handles all data entry, including reading the number of products and iteratively collecting ID, Name, Quantity, and Price from the user.
- **Processing Logic Module:** This is the central unit that performs all computations and decisions, including:
 - Calculating the Total Value of stock.
 - Implementing the linear search for Product ID.
 - Handling stock validation and updating (sale deduction).
 - Finding the maximum value in the quantity array.
- **Output Module:** Manages all console output, displaying the full inventory, search results, sale confirmations, and the final stock summary.

This structure makes the system simple, efficient, and easy to understand.

14. Code snippet:-

```
import java.util.Scanner;

public class InventoryManagement {
    public static void main(String[] args) {
```

```

Scanner sc = new Scanner(System.in);

System.out.print("Enter number of products: ");
int n = sc.nextInt();
sc.nextLine(); // consume newline

int[] productID = new int[n];
String[] productName = new String[n];
double[] quantityKg = new double[n]; // quantity in kilograms
double[] pricePerKg = new double[n]; // price per kilogram

// Input product records
for (int i = 0; i < n; i++) {
    System.out.println("\nEnter details for Product " + (i+1));
    System.out.print("ID: ");
    productID[i] = sc.nextInt();
    sc.nextLine();
    System.out.print("Name: ");
    productName[i] = sc.nextLine();
    System.out.print("Quantity (in kg): ");
    quantityKg[i] = sc.nextDouble();
    System.out.print("Price per kg (in money): ");
    pricePerKg[i] = sc.nextDouble();
    sc.nextLine();
}

// Display all products
System.out.println("\n--- Inventory Records ---");
for (int i = 0; i < n; i++) {
    double totalValue = quantityKg[i] * pricePerKg[i];
    System.out.println("ID: " + productID[i] +
                       ", Name: " + productName[i] +
                       ", Quantity: " + quantityKg[i] + " kg" +
                       ", Price per kg: ₹" + pricePerKg[i] +
                       ", Total Value: ₹" + totalValue);
}

// Search product by ID
System.out.print("\nEnter Product ID to search: ");
int searchID = sc.nextInt();
boolean found = false;
for (int i = 0; i < n; i++) {
    if (productID[i] == searchID) {
        double totalValue = quantityKg[i] * pricePerKg[i];
        System.out.println("Product Found: " + productName[i] +
                           ", Quantity: " + quantityKg[i] + " "
                           + "kg" +
                           ", Price per kg: ₹" + pricePerKg[i]
                           +
                           ", Total Value: ₹" + totalValue);
        found = true;
        break;
    }
}
if (!found) {
    System.out.println("Product with ID " + searchID + " not
found.");
}

```

```

        // Update stock (simulate sale)
        System.out.print("\nEnter Product ID to sell: ");
        int sellID = sc.nextInt();
        System.out.print("Enter quantity sold (in kg): ");
        double soldQty = sc.nextDouble();
        for (int i = 0; i < n; i++) {
            if (productID[i] == sellID) {
                if (quantityKg[i] >= soldQty) {
                    quantityKg[i] -= soldQty;
                    double totalValue = quantityKg[i] * pricePerKg[i];
                    System.out.println("Sale successful! Remaining
stock: " + quantityKg[i] + " kg" +
                                         ", New Total Value: ₹" +
totalValue);
                } else {
                    System.out.println("Not enough stock available!");
                }
            }
        }

        // Find product with highest stock
        double maxStock = quantityKg[0];
        String maxProduct = productName[0];
        for (int i = 1; i < n; i++) {
            if (quantityKg[i] > maxStock) {
                maxStock = quantityKg[i];
                maxProduct = productName[i];
            }
        }
        System.out.println("\nProduct with highest stock: " +
maxProduct + " (" + maxStock + " kg)");

        sc.close();
    }
}

```

15. Output Sample :-

```

enter number of products: 3

Enter details for Product 1
ID: 101
Name: sugar
Quantity (in kg): 50
Price per kg (in money): 45

Enter details for Product 2
ID: 102
Name: peanuts
Quantity (in kg): 40
Price per kg (in money): 140

Enter details for Product 3
ID: 103
Name: rice
Quantity (in kg): 100

```

```

Price per kg (in money): 50

--- Inventory Records ---
ID: 101, Name: sugar, Quantity: 50.0 kg, Price per kg: ?45.0, Total
Value: ?2250.0
ID: 102, Name: peanuts, Quantity: 40.0 kg, Price per kg: ?140.0, Total
Value: ?5600.0
ID: 103, Name: rice, Quantity: 100.0 kg, Price per kg: ?50.0, Total
Value: ?5000.0

Enter Product ID to search: 101
Product Found: sugar, Quantity: 50.0 kg, Price per kg: ?45.0, Total
Value: ?2250.0

Enter Product ID to sell: 101
Enter quantity sold (in kg): 25
Sale successful! Remaining stock: 25.0 kg, New Total Value: ?1125.0

Product with highest stock: rice (100.0 kg)

```

16. Line-by-Line Explanation :-

Module 1: Input Module

This module manages all data entry required to initialize the inventory and perform subsequent transactions.

- **Function:** Accepts and stores all initial product details from the console.
- **Key Operations:**
 - Reads the size of the inventory (n).
 - Reads `productID`, `productName`, `quantityKg`, and `pricePerKg` for all n products.
 - Reads the ID required for searching.
 - Reads the ID and quantity required for the sale operation.
- **Java Tools:** `Scanner` class and an initial `for` loop.

Module 2: Display and Calculation Module

This module focuses on presenting the current state of the inventory and calculating financial metrics.

- **Function:** Displays all current inventory records and their total value.
- **Key Operations:**
 - Iterates through all parallel arrays.
 - Calculates the `totalValue` for each item.
 - Prints a formatted list showing all attributes, including the calculated value.
- **Java Tools:** `for` loop and the mathematical multiplication operator.

Module 3: Transaction Processing (Search and Sale) Module

This is the core logic module responsible for finding specific records and updating stock levels securely.

- **Function:** Handles searching for products and simulating sales while maintaining data integrity.
- **Key Operations:**
 - **Searching:** Uses a linear search loop to match the user-entered ID against the `productID` array.
 - **Stock Validation:** Before deduction, it uses an `if` condition to verify that `quantityKg[i]` is \geq `soldQty`.
 - **Stock Update:** If validation passes, it performs the deduction:
`quantityKg[i] -= soldQty`.
- **Java Tools:** `if/else` statements and `for` loops with the `break` keyword.

Module 4: Analysis and Summary Module

This module provides an analytical summary of the current inventory state.

- **Function:** Finds and reports the product that currently holds the largest quantity in stock.
- **Key Operations:**
 - Initializes `maxStock` and `maxProduct` with the first item's details.
 - Iterates through the rest of the `quantityKg` array, comparing each value to the current `maxStock`.
 - Updates `maxStock` and `maxProduct` whenever a larger quantity is found.
- **Java Tools:** `for` loop and comparison operator (`>`).

17. Working Procedure (Short)

Start: Program initiates; user enters the total number of products

Input: User enters {Id}, {Name} {Quantity} {kg}, and {Price} {\$/kg} for all {n} products, which are stored in parallel arrays.

Display & Value: Full inventory is displayed, with {Total Value} \$ calculated for each product.

Search: User enters {Product ID}; system performs linear search and displays matching details.

Sale/Update: User enters {ID} and {Quantity Sold}

Validation: System checks {Available Stock} {Sold Quantity}.

Action: If valid, stock is deducted; if invalid, an error is shown.

Summary: Program finds and displays the {Product with Highest Stock}

End: Program terminates.

18. Summary

The **Inventory Management System** is a successful, console-based Java application built using core programming concepts like **parallel arrays, loops, and conditional logic**. The system effectively manages product records, tracks stock by weight, and automatically calculates the total monetary value of the inventory. The key achievement is the **stock validation mechanism**, which prevents errors like over-selling. The project is technically and economically feasible, providing a simple, accurate, and practical solution for basic shop inventory control, making it a valuable academic exercise.

Conclusion

The **Inventory Management System** successfully achieved all project objectives. It is a functional and reliable Java application that automates inventory processes, significantly reducing the chances of human error compared to manual tracking. The effective use of fundamental programming constructs proves their ability to create practical, real-world solutions. The project confirms the efficiency of digital inventory management and serves as an excellent demonstration of applied Java programming.

