Inventory System Project Report

Inventory Management System – Data Producer Module

February 18, 2026

Contents

1 Introduction **2**

2 Project Architecture Overview **2**

3 Model Layer **2**
- 3.1 Abstract Class: Product . . . 2
- 3.2 Subclass: Electronics . . . 3
- 3.3 Subclass: Clothing . . . 3

4 Demonstration of Polymorphism **3**

5 I/O Layer: CSVProductReader **3**

6 Application Layer (Main) **4**

7 Design Decisions **4**
- 7.1 Minimal Intrusion Principle . . . 4
- 7.2 Separation of Concerns . . . 4
- 7.3 Extensibility . . . 4

8 Role Within a Larger Inventory System **5**

9 Future Work: Toward a Full Inventory Simulation System **5**
- 9.1 Inventory State Management . . . 5
- 9.2 Transaction Modeling . . . 5
- 9.3 Service Layer Introduction . . . 6
- 9.4 Warehouse and Location Modeling . . . 6
- 9.5 Reorder and Safety Stock Logic . . . 6
- 9.6 Inventory Metrics and Reporting . . . 6
- 9.7 Simulation Engine . . . 7
- 9.8 Advanced Extensions . . . 7
- 9.9 Architectural Evolution . . . 7

10 Conclusion **8**

1 Introduction

This document describes the structure and functionality of a Java-based Inventory System developed as part of a modular enterprise architecture. The current implementation represents the **Data Producer layer**, responsible for reading product data from CSV files, storing it in memory, performing calculations, and preparing it for persistence in a relational database through JDBC.

The system demonstrates object-oriented principles including abstraction, inheritance, polymorphism, and separation of concerns.

2 Project Architecture Overview

The system is organized into the following logical components:

- **Model Layer** – Defines the product hierarchy.
- **I/O Layer** – Handles CSV data ingestion.
- **Application Layer (Main)** – Provides console interaction and business logic.
- **Data Consumer Layer (JDBC)** – Responsible for database persistence (external to this report).

3 Model Layer

3.1 Abstract Class: Product

The `Product` class represents a generic inventory item and defines attributes common to all product types:

- Category
- Product Name
- Quantity
- Unit Cost
- Margin

It provides shared behavior:

- Inventory value calculation
- Abstract unit price calculation
- Abstract packing/shipping cost calculation

This design enforces a contract requiring subclasses to implement pricing and packing logic.

3.2 Subclass: Electronics

The `Electronics` class extends `Product` and represents electronic goods.

Responsibilities:

- Implements pricing based on margin.
- Implements packing cost based on weight.

The overridden `getShippingCost(double weight)` method demonstrates subclass-specific behavior by computing packing requirements using a weight parameter.

3.3 Subclass: Clothing

The `Clothing` class extends `Product` and represents apparel products.

Responsibilities:

- Implements pricing based on margin.
- Implements packing cost based on volume.

The overridden `getShippingCost(double volume)` method demonstrates polymorphism through volume-based packing calculation.

4 Demonstration of Polymorphism

The system uses polymorphism through:

- A shared `List<Product>` collection.
- Overridden implementations of `getUnitPrice()`.
- Overridden implementations of `getShippingCost(double)`.

At runtime, the correct subclass method is invoked depending on the actual object type, demonstrating dynamic method dispatch.

5 I/O Layer: CSVProductReader

The `CSVProductReader` class is responsible for:

- Reading CSV files from the resources directory.
- Validating mandatory fields.
- Parsing numeric values safely.
- Instantiating appropriate subclass objects.
- Returning lists of `Product`.

This class isolates file parsing logic from business logic, improving maintainability and modularity.

6 Application Layer (Main)

The `Main` class provides a console-based interface with options to:

1. Import Electronics CSV
2. Import Clothing CSV
3. Display Electronics
4. Display Clothing
5. Compute Total Inventory Value
6. Display Packing Costs
7. Exit

Packing costs are calculated by requesting per-unit weight or volume from the system administrator. This demonstrates how subclass-specific parameters influence overridden methods without modifying shared data structures.

7 Design Decisions

7.1 Minimal Intrusion Principle

The project intentionally avoids modifying:

- CSV parsing logic
- Existing constructors
- Shared in-memory product structures

This ensures compatibility with downstream JDBC persistence logic.

7.2 Separation of Concerns

- Model classes handle domain logic.
- I/O classes handle file ingestion.
- Main handles interaction and orchestration.
- JDBC layer handles persistence.

7.3 Extensibility

The design supports future enhancements:

- Storing actual weight/volume per product.
- Implementing real shipping calculations.
- Adding additional product subclasses.
- Integrating with a warehouse management system.

8 Role Within a Larger Inventory System

Within an enterprise architecture, this module functions as:

- A **data ingestion component**.
- A **business rule processor**.
- A **pre-persistence validation layer**.

It prepares structured, validated product objects for insertion into an SQL database through JDBC, forming part of a multi-tier inventory management system.

9 Future Work: Toward a Full Inventory Simulation System

The current implementation represents a foundational data ingestion and processing module within a broader inventory architecture. While it successfully demonstrates object-oriented design, CSV ingestion, and preparation for database persistence, several extensions would transform it into a fully operational inventory management and simulation system.

9.1 Inventory State Management

Future versions should support dynamic stock movement rather than static inventory records. This includes:

- Stock In (receiving goods from suppliers)
- Stock Out (sales or shipments to customers)
- Inventory adjustments (corrections, damages, audits)

This would require controlled methods for increasing and decreasing product quantities with validation to prevent negative inventory levels.

9.2 Transaction Modeling

A realistic inventory system must record the reason behind quantity changes. Introducing an `InventoryTransaction` class would enable:

- Transaction type tracking (IN / OUT / ADJUSTMENT)
- Timestamp recording
- Supplier or customer reference
- Historical auditing

This enables traceability and supports reporting and compliance requirements.

9.3 Service Layer Introduction

Currently, the main application coordinates logic directly. A future architectural improvement would introduce an `InventoryService` layer responsible for:

- Processing transactions
- Validating stock levels
- Applying business rules
- Triggering low-stock alerts

This improves separation of concerns and scalability.

9.4 Warehouse and Location Modeling

To simulate real-world logistics, the system could introduce:

- Multiple warehouses
- Storage locations or bins
- Capacity constraints
- Volume-based storage optimization

This allows inventory to be tracked not only by product but also by physical location.

9.5 Reorder and Safety Stock Logic

A practical inventory system typically includes:

- Reorder points
- Safety stock levels
- Automatic purchase order generation
- Lead time simulation

These features help prevent stockouts and optimize supply chain efficiency.

9.6 Inventory Metrics and Reporting

Future enhancements may compute operational metrics such as:

- Inventory turnover rate
- Days of inventory on hand
- Fast-moving and slow-moving items
- Dead stock analysis
- Historical inventory valuation

These analytical tools provide strategic business insight.

9.7 Simulation Engine

To evolve into a full inventory simulator, an event-driven engine could be implemented to model:

- Random customer demand
- Supplier lead times
- Seasonal demand variations
- Backorders and stockouts

A simulation loop operating over simulated time periods (e.g., days or weeks) would allow the study of system behavior under varying operational conditions.

9.8 Advanced Extensions

Additional long-term enhancements may include:

- FIFO/LIFO inventory valuation
- Batch tracking
- Expiration date management
- Returns processing
- Role-based access control
- Graphical user interface integration

9.9 Architectural Evolution

With these enhancements, the system would evolve from a static data ingestion module into a layered architecture consisting of:

- Presentation Layer (Console or GUI)
- Service Layer (Business Logic)
- Domain Model (Products, Transactions, Warehouses)
- Persistence Layer (JDBC / SQL Database)
- Simulation Engine (Optional Analytical Component)

Such an evolution would position the project as a fully functional inventory management and simulation platform suitable for academic research, enterprise prototyping, or operational training environments.

10 Conclusion

The Inventory System demonstrates key object-oriented principles while maintaining architectural stability. The addition of subclass-specific packing calculations illustrates polymorphism without disrupting the data production workflow.

The system is modular, extensible, and suitable for integration into a broader enterprise inventory solution.