Design and Implementation of a Lightweight E-commerce Order Processing and Visualization

Dr. J. Narendra Babu[1], Prashanth K N [2] Kanaka.R[3], Keerthana.H.B[4], Keerthana.N.L[5], Kiran Naik.K[6,] Manish.B.M.Thyvanige[7]
[1]Professor, Department of CSE-Data Science, Sapthagiri NPS University, India.
[2] Assistant Professor, Department of CSE-Data Science, Sapthagiri NPS University, India.
[3]Student,Departmentof CSE-Data Science, Sapthagiri NPS University,India.
[4]Student, Department of CSE-Data Science, Sapthagiri NPS University, India.
[5]Student, Department of CSE-Data Science, Sapthagiri NPS University, India.
[6]Student, Department of CSE-Data Science, Sapthagiri NPS University, India.

Abstract

This paper presents the design, implementation, and evaluation of a compact Java-based order processing module that demonstrates basic e-commerce concepts: product catalog, users, orders, order items, revenue computation, and a simple graphical visualization of order totals. The implementation follows an object-oriented approach, with clear separation of concerns between data models, business logic, and presentation. The paper includes a code walkthrough, sample dataset, runtime behavior, visualization details, and discussion of limitations and future improvements. The module can serve as an educational reference or a small component within a larger e-commerce system.

Keywords: e-commerce, order processing, Java, object-oriented design, data visualization, teaching module

1. Introduction

E-commerce systems require core functionality for representing products, managing users and orders, calculating revenues, and producing actionable summaries. Building small, illustrative modules helps students and developers understand these concepts before scaling to production-level systems. We present a compact Java program that implements product, user, order, and order item classes, plus console-driven analytics and a basic Swing-based bar chart visualization.

The contribution of this paper is twofold:

1. A clear, documented, pedagogical implementation of fundamental e-commerce functions in plain Java (no external frameworks).
2. A short evaluation using an internal sample dataset together with suggestions for extension toward production readiness and research directions.

2. Related Work

Numerous textbooks and tutorials cover e-commerce back ends, database design, and web-based checkout flows. This work focuses instead on the minimal in-memory models and simple desktop visualization to support teaching and rapid prototyping. It complements web-focused tutorials by highlighting clean object-oriented modeling and simple analytics.

3. System Architecture and Design

3.1. High-level overview

The program is organized into simple classes that map to domain entities:

- **Product** — immutable product descriptor (id, name, price).
- **User** — basic user identity (user ID, name).
- **Order Item** — association between a Product and a quantity, with helper for item-level total price.
- **Order** — collection of Order Items, with order ID, user ID, aggregation methods such as getOrderTotal().
- **Bar Chart Panel** — a Swing JPanel subclass that renders a bar chart of orders using Graphics2D.

- **main2** — the driver class that constructs sample data, provides a console menu, and launches the visualization.

This modular decomposition follows single-responsibility principles and keeps UI code separate from data modeling.

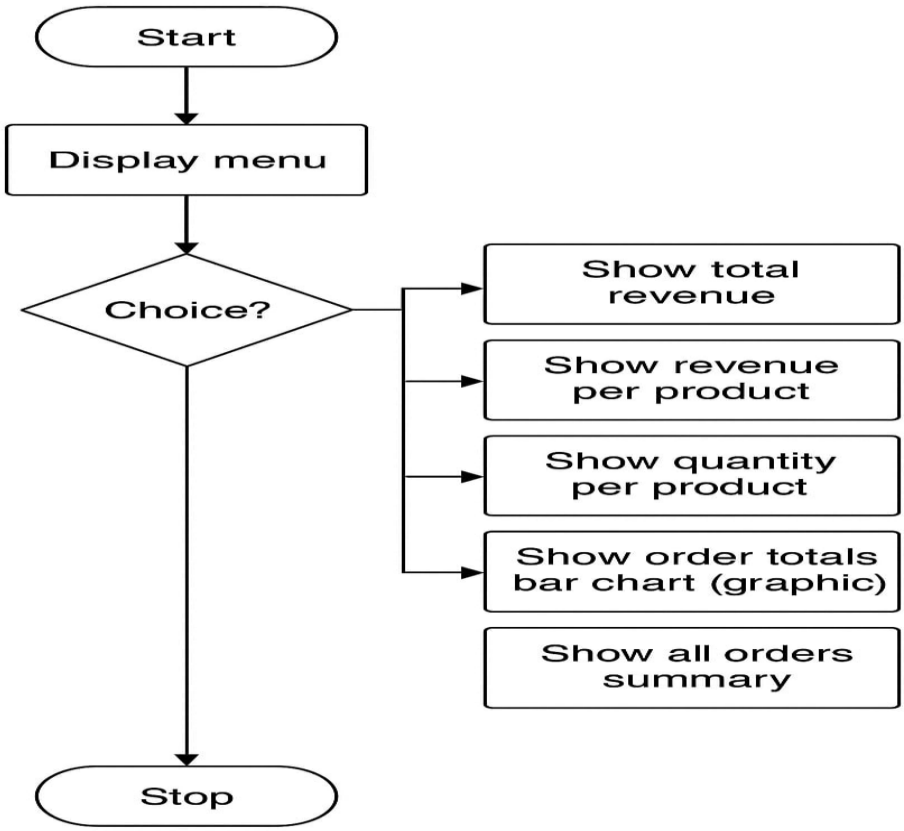
3.2. Class responsibilities and interactions

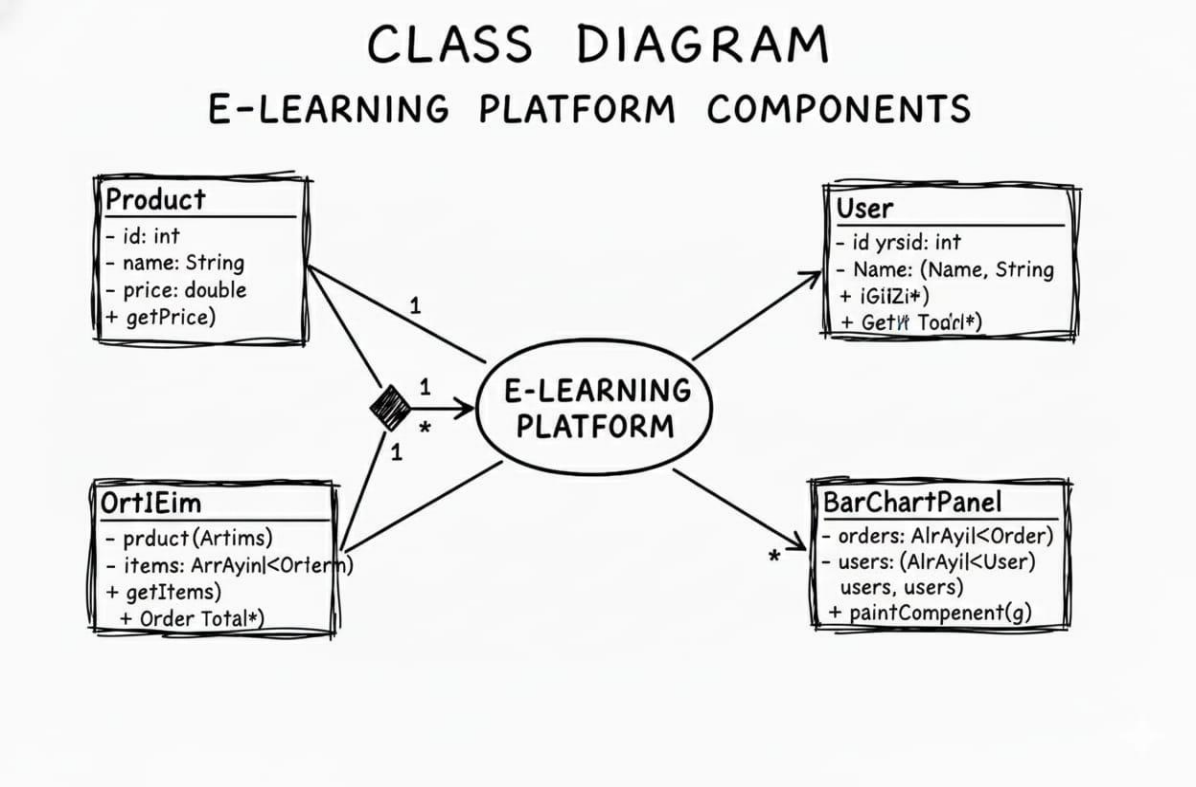
A UML-like summary (informal):

- main2 192 creates Product [], User [], Order[] and wires Order Items into orders.
- Order aggregates Order Items and exposes getOrderTotal() used by analytics and the BarChartPanel.
- Bar Chart Panel queries orders and users to label bars with user names and order IDs.

3.3. Data flow

All data is kept in-memory using Array List containers. The console menu invokes analytics by iterating the orders list and computing sums or counts.





4. Implementation Details and Code Walkthrough

This section explains the key implementation choices and critical methods.

4.1. Product, User, Order Item, and Order classes

- **Product** stores id, name, and price. Prices are double, which is sufficient for demonstration but not ideal for financial applications (see Section 6).
- **Order Item** links a Product and a quantity. getItemTotal() returns product.getPrice() * quantity.
- **Order** stores order ID, user ID, and a list of Order Item. getOrderTotal() sums item totals.

Code is straightforward and readable, emphasizing clarity for teaching.

4.2. Analytics implemented in main2 menu

The console menu provides 5 analytics options:

1. **Total Revenue** — sum of getOrderTotal() across all orders.
2. **Revenue per Product** — map product name to total revenue accrued across orders.
3. **Quantity per Product** — map product name to aggregated quantity sold.
4. **Order Totals Bar Chart** — launches a Swing frame that displays a bar chart of each order's total, labelled by order ID and user name.
5. **All Orders Summary** — prints order ID, user name, and order total for each order.

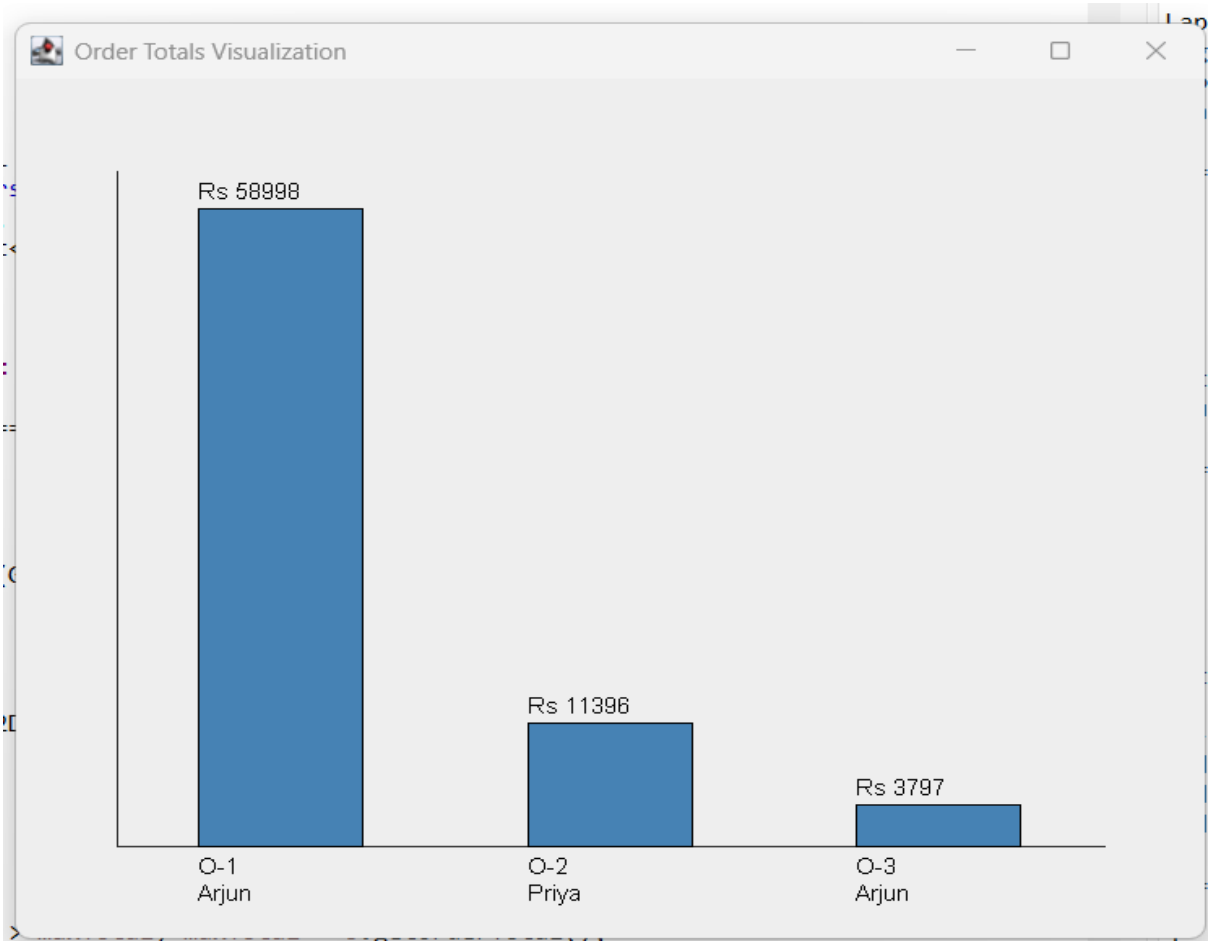
Implementation notes: - Java HashMap is used to accumulate per-product metrics. - The program uses Scanner for console input and a switch for menu handling.

4.3. Visualization with Swing (BarChartPanel)

The BarChartPanel extends JPanel and overrides paint Component(Graphics g). Key behavior:

- Computes maxTotal across the orders to scale bar heights.
- Computes barWidth dynamically: chartWidth / orders.size().
- Draws axes and iterates orders drawing filled rectangles with labels: price, order ID, and user name.

Limitations of the current visualization (elaborated in Section 6) include fixed colors, lack of axis ticks/gridlines, poor label placement for long names, and no dynamic resizing logic beyond a simple proportional height calculation.



5. Sample Data and Execution

The program builds a small sample dataset in main:

- Products: Laptop (55,000), Shoes (1,999), Smartwatch (3,499), Bag (899).
- Users: Arjun (101), Priya (102).
- Orders: three orders mixing products and quantities.

When launched, users can use the console menu to compute revenue metrics or open the graphical chart to inspect order totals.

5.1. Example outputs

- **Total Revenue:** computed as sum of order totals (e.g., Rs X — the actual numeric output is produced by running the program with provided data).

```
=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 1
Total Revenue: Rs 74191.0

=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 2

--- Revenue Per Product ---
Laptop: Rs 55000.0
Bag: Rs 2697.0
Shoes: Rs 5997.0
Smartwatch: Rs 10497.0

=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 3

--- Quantity Per Product ---
Laptop: 1 units
Bag: 3 units
Shoes: 3 units
Smartwatch: 3 units
```


- **Revenue per product:** printed as ProductName: Rs amount.
- **Quantity per product:** printed as ProductName: N units.

The bar chart shows a bar per order with label Rs <amount>, O-<orderId>, and the user name under the bar

```
=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 4
Launching Graphic Chart...

=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 5

--- Orders Summary ---
Order ID: 1 | User: Arjun | Order Total: Rs 58998.0
Order ID: 2 | User: Priya | Order Total: Rs 11396.0
Order ID: 3 | User: Arjun | Order Total: Rs 3797.0

=====MENU=====
1. Show Total Revenue
2. Show Revenue Per Product
3. Show Quantity Per Product
4. Show Order Totals Bar Chart (Graphic)
5. Show All Orders Summary
6. Exit
Enter choice: 6
Exiting... Thank you!
```

6. Discussion: Correctness, Limitations, and Production Considerations

6.1. Correctness and edge cases

- **Zero and negative values:** Code does not validate price or quantity. Negative prices or quantities could produce incorrect totals. Add validation in constructors or setters.
- **Floating point for currency:** Using double for monetary values can introduce rounding errors. For production, prefer Big Decimal with an explicit currency context.
- **Concurrency:** The in-memory Array List structures are not thread-safe. If multiple threads may modify orders, use concurrent collections or synchronization.

6.2. Usability and UX

- The console menu is synchronous and blocks while the Swing window is open. Consider separating UI threads (Swing runs on EDT) and ensuring non-blocking console I/O.
- The chart lacks dynamic features (tooltips, axis labels, colors). Use a charting library (e.g., JFreeChart) for richer visualizations.

6.3. Persistence and scaling

- Data is ephemeral and lost on exit. For real systems, persist data to relational databases (MySQL/Postgres) or NoSQL stores.
- The design must be extended for catalogs with hundreds/thousands of products and orders; use pagination, indexing, and queries rather than in-memory scans.

6.4. Security and validation

- Inputs from Scanner are not sanitized; validate user input to prevent crashes.
- For web or distributed systems, authenticate users and sanitize fields before display.

7. Extensions and Future Work

We propose multiple directions for improvement and research:

1. **Monetary precision:** Replace double with BigDecimal and implement currency-aware formatting.
2. **Persistent storage:** Integrate a relational database with DAOs (Data Access Objects) and JDBC or JPA.
3. **Web front-end:** Expose the module as RESTful APIs and build a web UI (React/Angular) for broader accessibility.
4. **Unit tests:** Add JUnit tests covering corner cases: empty orders, zero quantities, invalid inputs.
5. **Advanced analytics:** Add time-series revenue reports, top-k products, customer lifetime value estimation, and CSV/Excel export.
6. **Visualization libraries:** Replace custom Swing drawing with JFreeChart or JavaFX charts (supporting tooltips and interactive zoom).
7. **Concurrency and microservices:** Re-architect to allow concurrent processing and horizontal scaling.

8. Conclusion

This lightweight Java module demonstrates fundamental e-commerce building blocks: product modeling, order aggregation, revenue analytics, and a simple visualization. While intentionally simple for teaching and prototyping, it highlights practical considerations—currency handling, validation, persistence, and scalability—that must be addressed for production systems. The code serves as a foundation for classroom exercises and further research.

9. References

[1] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, “Design Patterns: Elements of Reusable Object-Oriented Software,” Addison-Wesley, 1994.
[2] Oracle Java Documentation — Swing, Oracle Corporation, 2024.
[3] J. Bloch, “Effective Java,” Addison-Wesley, 2008.
[4] P. Deitel and H. Deitel, “Java: How to Program,” 10th ed., Pearson, 2017.
[5] H. Schildt, “Java: The Complete Reference,” 11th ed., McGraw-Hill, 2018.
[6] C. S. Horstmann and G. Cornell, “Core Java,” Prentice Hall, 2015.
[7] I. Sommerville, “Software Engineering,” 10th ed., Pearson, 2015.
[8] R. C. Martin, “Clean Code: A Handbook of Agile Software Craftsmanship,” Prentice-Hall, 2008.
[9] E. Turban, J. Outland, D. King, et al., “Electronic Commerce: A Managerial and Social Networks Perspective,” Springer, 2018.
[10] S. Kumar and R. H. Goudar, “E-Commerce Security: A Survey,” International Journal of Computer Applications, 2012.
[11]Dr.J.NarendraBabu, upcoming Strengths on Internet of Things Journal of Advanced Research in Dynamical & Control systems Vol.11,N0.11,2019
[12]Dr.J.Naredra Babu, Enhancement of RVM Using FPGA Journal of Advanced Research in Dynamical & Control systems Vol.11,No.11,2019

[13]Dr.J.NarendrBabu, BrainSignal processing : Analysis, Technologies and Application Journal of Advanced Research in Dynamical & Control systems (JARDCS)Vol.11,No: 12,2019.

[14] Dr.J.NarendraBabu, Steganography based message hiding in image, International journal of advance research in science and engineering, vol.no.4, Issue No.12, December 2015, ISSN-2319-8354
[15] B. Bhasker, “Electronic Commerce: Framework, Technologies and Models,” McGraw-Hill, 2014.
[16] U. Shah and D. Patel, “Analysis of Billing System Automation in Modern Retail Environments,” IJCST, 2021.
[17] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, “Introduction to Algorithms,” MIT Press, 2009.
[18] M. Goodrich and R. Tamassia, “Data Structures and Algorithms in Java,” Wiley, 2014.
[19] R. Sedgewick and K. Wayne, “Algorithms,” Addison-Wesley, 2017.
[20] K. C. Laudon and C. G. Traver, “E-Commerce: Business, Technology, Society,” Pearson, 2023.
[21] D. K. Gangeshwer, “E-commerce or Internet Marketing: A Business Review from Indian Context,” IJUSEST, 2013.
[22] J. Han and M. Kamber, “Data Mining: Concepts and Techniques,” Morgan Kaufmann, 2012.
[23] Dr.J.Narendra Babu, et.al- Experimental detection of vehicles for toll gate billing, International journal of Science technology and management, vol.no.4, Issue No.12, December 2015, ISSN 2394-1537

Author Biography

Dr. J.Narendra Babu is a seasoned academician with over 28 years of experience in teaching and software industry, Dr.J.Narendra Babu currently serves as a Professor in the Department of Data Science at Sapthagiri NPS University. He holds a B.Tech, M.Tech and PhD degree. Dr.J.Narendra Babu has published extensively in reputed journals and conferences, Dr.J.Narendra Babu has played a key role in With mentoring students, supervising undergraduate projects, coordinating academic activities, and contributing to curriculum development