IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

Design of a Modular E-Commerce Product Management System Using Object-Oriented Java Class Hierarchies

Mrs. Punashri Patil[1], Yash Raut[2], Apurva Ransing[3], Vedika Patil[4]
Assistant Professor, Department of Information Technology[1]
Under Graduate Student, Department of Information Technology[2,3,4]
AISSMS's Institute of Information Technology, Pune, Maharashtra, India

Abstract: *With the rapid growth of online shopping websites and apps, it has become important to design software systems that are scalable, secure, and well-organized. Traditional procedural programming often creates systems where different parts are highly dependent on each other, making them difficult to modify or expand later. In this paper, we present the design and development of an object-oriented e-commerce product management system using Java. The system is built using core OOP concepts such as classes and objects, encapsulation using access modifiers, constructor and method overloading, static members, passing objects as parameters, recursion, and inner classes. A modular structure is created to handle main components like Product, User, Cart, Order, and Admin. Compared to a simple unstructured program, this approach makes the system easier to maintain, reuse, and extend in the future. The implementation also shows efficient handling of objects and better execution of product-related operations. Overall, this project demonstrates that a well-planned class-based design improves reliability and provides a strong foundation for future integration with databases and cloud services. During the implementation of this system practical understanding of the Object-Oriented concepts was gained through real coding and testing the output.*

Keywords: Object-Oriented Programming (OOP), Java, E-Commerce System, Class-Based Architecture, Encapsulation, Constructor Overloading, Static Members, Modular Software Design, Recursion, Inner Classes

I. INTRODUCTION

A. Background of Object-Oriented Programming

Modern software applications need a proper structure so that they can grow easily and be maintained without difficulty. Object-Oriented Programming (OOP) is a widely used approach because it allows programmers to model real-world things using classes and objects. This makes the program more organized and easier to understand. Important concepts of OOP such as encapsulation, abstraction, modularity, and reusability help in building systems that are flexible and simple to modify in the future. Java strongly supports object-oriented programming by providing features like class-based design, access modifiers for data protection, constructors for object initialization, and well-structured interaction between objects. In this project, these OOP concepts are applied to design a structured e-commerce product management system [1],[2].

B. Statement of the Problem

The rapid expansion of digital commerce platforms has significantly increased the complexity of managing products, users, transactions, and inventory within software systems. Traditional procedural programming approaches often lead to tightly coupled code structures, limited scalability, poor maintainability, and high redundancy. As the number of

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

product categories and business rules increases, modifying or extending the system becomes time-consuming and error-prone. There is a need for a modular, reusable, and scalable software architecture that can efficiently manage product hierarchies, pricing strategies, and transaction processing while maintaining data integrity and security.

Object-Oriented Programming (OOP) using Java provides mechanisms such as encapsulation, abstraction, inheritance, and polymorphism that can address these challenges. However, designing an optimized object-oriented architecture that effectively models real-world e-commerce entities while ensuring extensibility and maintainability remains a critical problem [12],[13].

Therefore, this research focuses on designing and implementing a Java-based E-Commerce product management system using OOP principles to improve code reusability, flexibility, security, and overall system scalability.

C. Limitations of Non-Structured System Design

In procedural programming, data and the functions that operate on that data are usually closely connected. Most of the time, both are written together in the same file. Because of this, making changes or adding new features can become difficult as the program grows. It may also become harder to debug and maintain the system. Although data can be protected to some extent, a non-modular design often leads to repetition of code. This redundancy reduces the overall maintainability of the software [8]. When data and functions are not properly organized, the program may become slower, less reliable, and harder to scale. As the application becomes larger and more complex, performance and efficiency can also decrease. This is one of the main reasons why object-oriented programming is preferred for building large and scalable systems.

D. Importance of OOP in E-Commerce Systems

E-commerce systems handle many activities such as managing products, interacting with users, processing payments, and handling orders. Because these systems deal with large amounts of data and multiple operations at the same time, proper organization and control are very important. By using an object-oriented, class-based approach, different parts of the system such as Product, User, Cart, and Order can be organized separately [9]. This makes the system easier to understand and manage. OOP also allows better security by using access modifiers to protect important data. Overall, it helps in building systems that are more scalable, organized, and easier to maintain as the application grows.

E. Objective and Contribution of the Study

This project focuses on designing and developing an e-commerce product management system using Java and object-oriented programming concepts. The system is built using important OOP features such as class and object creation, constructor and method overloading, use of static members, passing objects as parameters, recursion, and inner classes. The implementation shows that proper interaction between objects makes the system easier to manage and update. It also helps reduce repeated code and supports a modular structure, where each component has a specific responsibility. As a result, the system becomes more organized, maintainable, and efficient.

F. Organization of the Paper

The remaining sections of this paper are organized as follows. Section II discusses the literature review related to object-oriented system design and modular software development [3],[6]. Section III explains the proposed system architecture and includes class diagrams along with the relationships between different components. Section IV describes how various OOP concepts are implemented in the developed system. Section V explains the system's working, performance aspects, and key observations made during testing. Finally, Section VI summarizes the overall work, presents the main conclusions, and suggests possible improvements for future development.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

II. LITERATURE REVIEW

As online shopping continues to grow, e-commerce businesses require efficient software systems to manage large numbers of products and transactions. A proper system structure is important for organizing product data and handling operations smoothly. Traditional procedural programming often struggles with complex systems because it lacks modularity and code reusability. Object-Oriented Programming (OOP) is widely used for building scalable applications. It designs software using real-world entities as classes and objects, making systems easier to manage and modify. The four main principles of OOP—Encapsulation, Abstraction, Inheritance, and Polymorphism—help in creating flexible and maintainable software.

Encapsulation protects data by restricting direct access, while abstraction hides unnecessary implementation details. Inheritance allows reuse of existing code by creating new classes from existing ones. Polymorphism provides flexibility by allowing different objects to respond differently to the same method. These concepts reduce system complexity and improve software organization. Many studies suggest that OOP-based design improves maintainability and scalability in large applications. By dividing a system into independent classes, developers can update or extend specific parts without affecting the entire program. This makes OOP a suitable approach for developing structured and reliable e-commerce systems [9],[11].

Java is a popular language for developing enterprise and web-based systems because it strongly supports OOP and provides platform independence. In an e-commerce system, Java classes can represent components like Products, Customers, and Orders, helping maintain an organized structure. Using OOP in an e-commerce product management system improves code organization, reduces duplication, and makes the system easier to expand in the future.

III. METHODOLOGY

A. System Design Approach

The system is developed using an object-oriented approach where real-world entities are represented as classes. Instead of writing everything in one place like in procedural programming, the system is divided into different classes. Each class has its own data and functions. This makes the system more organized and easier to manage. It also helps in protecting data and allows controlled access to important information.

During development, the main entities of an e-commerce system such as Product, Customer, Cart, and Order are first identified. Their attributes and behaviours are then analysed. After that, a class diagram is created to show the relationships between these components, including association, inheritance, and dependency. This structured approach makes the system easier to understand, maintain, and expand in the future.

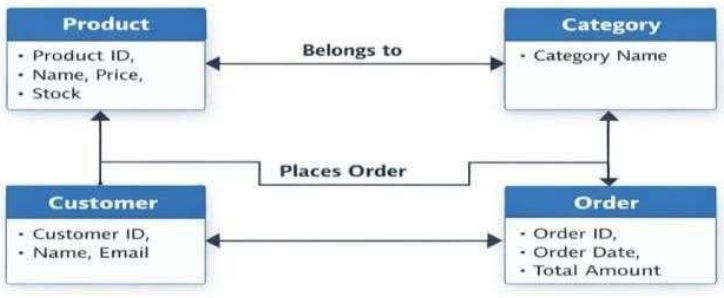

Fig. 3.1.1. Conceptual Architecture of an E-Commerce Product Management System using Object-Oriented Design.

B. Class Identification and Modelling

The main components of the e-commerce system are first identified and analysed. Based on this analysis, four main classes are created as the foundation of the system: Product, Customer, Cart, and Order. Each class stores its own data as attributes and defines its behaviour using methods. The Product class stores details such as product ID, name, price, and available quantity. The Customer class keeps user information required for placing orders. The Cart class manages

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

the products added by the customer and their quantities. The Order class handles order processing, billing, confirmation, and updates the product inventory after purchase. To ensure data security, the data members of each class are declared as private. Access to these variables is provided through public methods such as getters and setters. This follows the principle of encapsulation. Constructors are used to initialize the object data properly when an object is created. Constructor overloading is also used to allow flexible object creation. For example, a Product object can be created with basic details or with complete information. This is an example of compile-time polymorphism and improves the usability of the class design.

C. Implementation Environment

Because of the strong object-oriented capabilities of Java, along with its portability and extensive library support, the application was implemented using this programming language. Java provides built-in features such as access specifiers, exception handling, inheritance, and dynamic method dispatch, which allow effective demonstration of fundamental Object-Oriented Programming (OOP) principles during system development.

The application was developed using the Java Development Kit (JDK) and compiled as a console-based program. A console-based interface was selected to emphasize the logical interaction between classes and objects while avoiding the additional complexity associated with graphical user interface development.

This approach makes it easier to demonstrate how class instances interact with one another and how core programming concepts are implemented within the system. Furthermore, exception handling mechanisms were incorporated to improve the reliability and robustness of the application. These mechanisms manage runtime errors caused by invalid user inputs or situations where the system cannot process a request, such as insufficient product stock.

D. Testing and Validation

The interaction between classes and the logical flow of the system were verified through a series of systematic tests. These tests were designed to examine important functions such as product creation, shopping cart operations, and order processing. The system was tested under both normal working conditions and boundary situations to ensure reliable performance.

Several edge cases were also considered during testing. Scenarios such as negative product prices, zero stock availability, and invalid user inputs were checked using appropriate validation mechanisms. The billing output was carefully verified to ensure that calculations were accurate based on the provided input values. After each order was completed, the inventory levels were also reviewed to confirm that the product stock was updated correctly.

The modular architecture of the system makes debugging and maintenance easier. Since each class is designed to operate as an independent unit, errors can be identified and resolved without affecting other parts of the system. This separation of functionality is one of the key advantages of using an object-oriented design approach.

TABLE I. SYSTEM ENTITY AND CLASS ATTRIBUTE MAPPING

Entity Name	Java Class	Core Attributes (Encapsulated)	Primary Methods
Product	Product	id, name, price, totalProducts	display(), calculateFinalPrice()
Electronics	Electronics	warrantyYears, gstRate	calculateFinalPrice() (Overridden)
Customer	User	userId, userName, contact	updateProfile(), validateLogin()
Cart	Cart	productArray[], itemCount	addItem(), removeItem()
Order	Order	orderId, finalAmount, status	generateBill(), confirmOrder()

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

TABLE II. EXPERIMENTAL ENVIRONMENT AND IMPLEMENTATION TOOLS

Parameter	Specification / Tool
Programming Language	Java (JDK 17.0 or higher)
Paradigm	Pure Object-Oriented Programming (OOP)
Execution Environment	Java Virtual Machine (JVM)
Interface Type	Console-Based (CLI)
Input Handling	java.util.Scanner, Command-Line Arguments
Memory Management	Automatic Garbage Collection (Java)

TABLE III. TEST CASE VALIDATION AND PERFORMANCE OBSERVATIONS

Test Case ID	Feature Tested	Input Type	Expected Result	Status
TC-01	Encapsulation	Direct attribute access	Compile-time error	Passed
TC-02	Polymorphism	Electronic Product	Price + 18% GST	Passed
TC-03	Static Tracking	Multiple object creation	Incremental totalProducts count	Passed
TC-04	Recursion	recursiveDisplay()	Sequential stack-based output	Passed
TC-05	Error Handling	Negative price input	Exception / Validation message	Passed

IV. RESULTS AND DISCUSSION

A. Encapsulation

If data is freely accessible from anywhere in a program, it can create security risks and reduce system stability. In the e-commerce product management system, encapsulation is applied by declaring important attributes, such as product ID and product name, as private. Access to these variables is provided through public methods, which control how the data can be read or modified. This prevents unauthorized changes and helps protect the internal state of the object. As a result, the system becomes more secure, stable, and reliable.

B. Abstraction

When all internal details of a system are exposed, it becomes complex and difficult to understand. Abstraction helps reduce this complexity by showing only the necessary features to the user while hiding the internal implementation details. In this system, classes provide simple methods such as display() and calculateFinalPrice() that users can call without knowing how the internal calculations are performed. The user does not need to understand the internal logic to use these methods. This makes the system easier to use, improves clarity, and supports better modular design.

C. Inheritance

When similar code is written repeatedly in different parts of a program, it increases redundancy and reduces maintainability. Inheritance helps solve this problem by allowing one class to reuse the properties and methods of another class. In this system, the Electronics class inherits from the base class Product. The Product class contains common attributes and methods that are shared by all product types. By using inheritance, there is no need to rewrite the same code again in the subclass. This reduces duplication and makes the system easier to maintain and extend.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 6, Issue 4, March 2026
Impact Factor: 8.2

D. Polymorphism

When a system depends too much on fixed conditional statements, its behavior becomes rigid and less flexible. Polymorphism helps improve flexibility by allowing the same method to behave differently for different objects. In this system, method overriding is used to achieve polymorphism. For example, the calculateFinalPrice() method works differently for normal products and electronic products. The correct method is chosen automatically at runtime based on the type of object. This makes the system more flexible, scalable, and easier to extend in the future.

Fig. 4.1. Object-Oriented Programming Principles Applied in Java.

E. Code Implementation in Java (Representative Snippets)

1) Class Definition with Encapsulation and Static Variable:

```
class Product {
   private int id;
   private String name;
   protected double price;
   public static int totalProducts = 0;
   public Product(int id, String name, double price) {
      this.id = id; this.name = name; this.price = price; totalProducts++;
   }
   public void display() {
      System.out.println("ID: " + id);
      System.out.println("Name: " + name);
      System.out.println("Price: Rs." + price);
   }
   public double calculateFinalPrice() { return price; }
}
```

Encapsulation is shown in the Product class because the attributes id and name are declared as private. These variables cannot be accessed directly from outside the class and can only be modified through public methods, which helps protect the data from accidental changes. The totalProducts variable is declared as static, meaning it is shared by all objects of the Product class. A single variable exists at the class level to keep track of the total number of products. The constructor initializes the attributes when an object is created. By using the this keyword, the constructor refers to the current object and correctly assigns the values to the respective instance variables.

2) Inheritance and Method Overriding (Polymorphism):

```
class Electronics extends Product {
   private int warrantyYears;
```

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, March 2026 Impact Factor: 8.2

```
    public Electronics(int id, String name, double price, int warrantyYears) {
        super(id, name, price); this.warrantyYears = warrantyYears;
    }
    @Override public double calculateFinalPrice() {
        return price + (price * 0.18);
    }
    @Override public void display() {
        super.display();
        System.out.println("Warranty: " + warrantyYears + " Years");
        System.out.println("Final Price (with GST): Rs." + calculateFinalPrice());
    }
}
```

Inheritance is applied by deriving the Electronics class from the Product class. The super keyword is used to call the constructor of the parent class so that the base class attributes can be initialized and reused. The calculateFinalPrice() method is overridden in the Electronics class, which allows GST to be added specifically for electronic items. This demonstrates run-time polymorphism, where the same method can perform different operations depending on the type of object or product being used.

3) Method Overloading (Compile-Time Polymorphism):

```
public void setDetails(String name) { this.name = name; }
public void setDetails(String name, double price) {
    this.name = name; this.price = price;
}
```

The setDetails() method is overloaded by defining multiple versions with different parameter lists. This illustrates compile-time polymorphism, where the appropriate method is selected during compilation based on the number and type of arguments provided.

4) Object as Argument and Object as Return Type:

```
public boolean isSameProduct(Product p) { return this.id == p.id; }
public static Product createProduct() { return new Product(1, "Sample", 1000); }
```

The isSameProduct() method accepts an object as a parameter, demonstrating interaction between objects within the system. The createProduct() method returns a Product object, illustrating factory-style object creation and the concept of returning objects from methods.

5) Array of Objects and Recursion:

```
static Product[] products = new Product[5];
public static void recursiveDisplay(int index) {
    if (index == products.length || products[index] == null) return;
    products[index].display();
    recursiveDisplay(index + 1);
}
```

An array of objects enables structured storage of multiple product instances. The recursiveDisplay() method demonstrates recursion, where a function repeatedly calls itself until a specific termination condition is reached, illustrating the stack-based flow of execution.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, March 2026 Impact Factor: 8.2

6) Command-Line Arguments and Scanner Input:

```
public static void main(String[] args) {
   if (args.length > 0) {
      System.out.println("Welcome " + args[0]);
   }
   Scanner sc = new Scanner(System.in);
   System.out.print("Enter Product ID: ");
   int id = sc.nextInt();
}
```

By using command line arguments, external parameters can be provided to the program during execution. The Scanner class allows the program to accept dynamic user input, enabling the system to remain interactive and perform console-based product management operations.

From the previous examples, it is clear that the core Object-Oriented Programming principles have been implemented in an orderly fashion in the E-Commerce Product Management System. There is a full integration of these elements as part of a menu-driven console application.

V. CONCLUSION

This study focused on the design and implementation of a modular E-Commerce product management system using object-oriented programming principles in Java. Key OOP concepts such as encapsulation, constructor and method overloading, inheritance, method overriding, static members, recursion, object passing, command-line arguments, and arrays were implemented to develop the system. By developing this product management system, it has been established that the use of class hierarchies and structured object interaction provides a significantly higher level of organization, reusability, and maintainability compared to traditional unstructured procedural programming approaches. In addition, the modular design promotes logical grouping of the product entities while allowing controlled access to the data and executing dynamic behavior. As a result, this research demonstrates that structured class-based design in Java offers an efficient, secure, and flexible framework for developing e-commerce management systems.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to our guide and mentor for her valuable guidance and encouragement throughout the completion of this research work. Her expertise, insightful suggestions, and constructive feedback greatly contributed to improving the quality of this research paper.

We would also like to thank the faculty members of the Department of Information Technology at AISSMS's Institute of Information Technology for providing the necessary academic environment and support required for carrying out this study. Their motivation and assistance helped us successfully complete this research work.

REFERENCES

[1] H. M. Deitel and P. J. Deitel, Java: How to Program, 11th ed. Boston, MA, USA: Pearson, 2018.
[2] K. Sierra and B. Bates, Head First Java, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, 2005.
[3] H. Schildt, Java: The Complete Reference, 11th ed. New York, NY, USA: McGraw-Hill Education, 2019.
[4] B. Eckel, Thinking in Java, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall, 2006.
[5] G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, 2nd ed. Boston, MA, USA: Addison-Wesley, 2005.
[6] C. Larman, Applying UML and Patterns, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2004.
[7] I. Sommerville, Software Engineering, 10th ed. Boston, MA, USA: Pearson, 2016.
[8] R. S. Pressman and B. R. Maxim, Software Engineering: A Practitioner's Approach, 8th ed. New York, NY, USA: McGraw-Hill Education, 2015.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 6, Issue 4, March 2026 Impact Factor: 8.2

[9] K. K. Bajaj and D. Nag, E-Commerce: The Cutting Edge of Business, 2nd ed. New Delhi, India: Tata McGraw-Hill, 2005.
[10] G. Schneider, Electronic Commerce, 12th ed. Boston, MA, USA: Cengage Learning, 2017.
[11] A. K. Sharma and R. Kumar, "Object-Oriented Design Approach for E-Commerce Applications," International Journal of Computer Applications, vol. 182, no. 15, pp. 20–25, 2019.
[12] S. Aljawarneh, "Design and Implementation of an E-Commerce System Using Java Technologies," IJACSA, vol. 9, no. 3, pp. 1–7, 2018.
[13] M. Fowler, UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd ed. Boston, MA, USA: Addison-Wesley, 2004.

BIOGRAPHY

Mrs. Punashri Patil is currently working as an Assistant Professor in the Department of Information Technology at AISSMS's Institute of Information Technology, Pune, India. She has significant academic experience in teaching undergraduate engineering students and guiding them in various academic and research activities. Her areas of interest include Object-Oriented Programming, Software Engineering, and emerging technologies in the field of Information Technology. Yash Raut, Apurva Ransing and Vedika Patil are undergraduate students pursuing a Bachelor of Technology in Information Technology at AISSMS's Institute of Information Technology, Pune, India. Their interests include programming, software development, object-oriented system design and modern computing technologies