Ain Shams University
Faculty of Engineering
Department of Computer and Artificial Intelligence Engineering

Object-Oriented Computer Programming

Course Code: CSE241

Submitted by:

Karim Samer Mostafa 23p0439

Youssed Atef Elnaggar 23p0341

Nourhan Hesham Elsayed 23p0442

Jana Tamer Maher 23p0173

Omar Gamil Anwar 23p0438

Submitted on:
10/12/2024

Instructor:

Dr. Mahmoud Ibrahim Khalil

Project GitHub Link : [**https://github.com/karimsamer100/OOPprj**](https://github.com/karimsamer100/OOPprj)

Contents

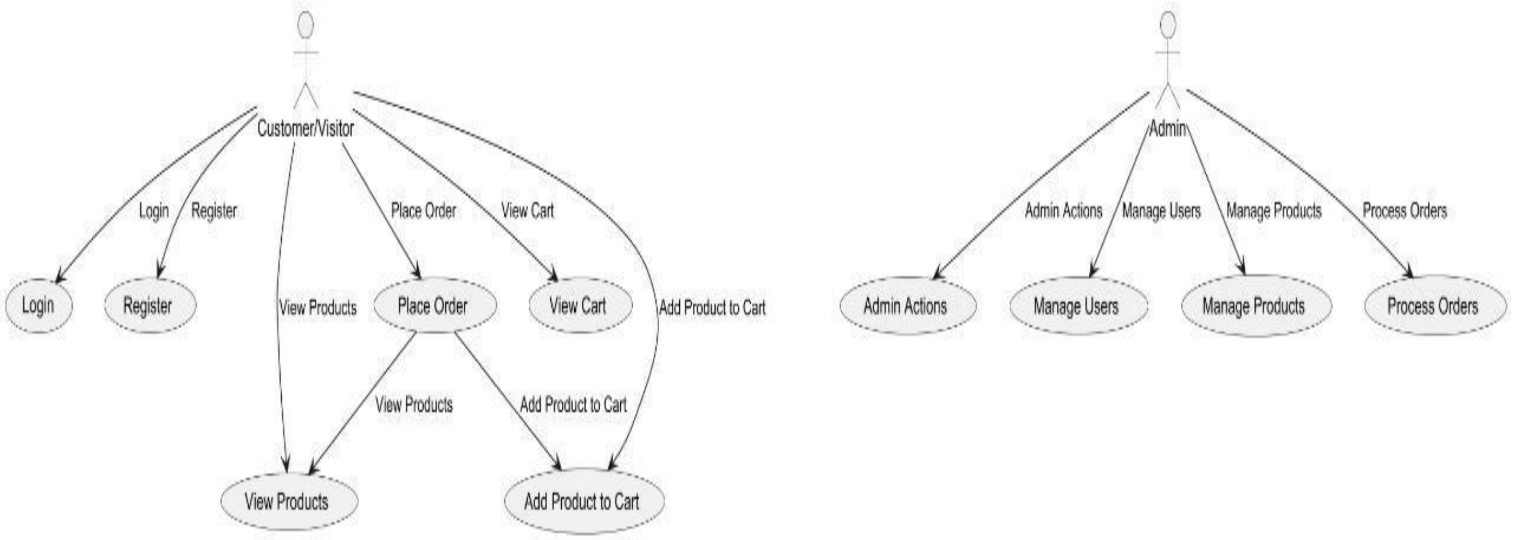
Introduction 2
System Design and Features........ 2
Customer Class........ 2
Admin Class 3
Category and Product Classes 4
Cart and Order Classes........ 5
Database Class: In-Memory Data Storage for E-Commerce System 5
Main Program (Javaproject Class) 8
Key Functionalities in the Main Program........ 9
Conclusion........ 13

E-Commerce Backend System Implementation Using Object-Oriented Programming (OOP)

Introduction

The evolution of e-commerce has revolutionized the way businesses interact with customers. With the advent of online shopping, it has become imperative to design and develop a robust and scalable backend system that can efficiently manage customer interactions, product inventory, order management, and administrative tasks. The aim of this project is to create a fundamental e-commerce backend system by utilizing Object-Oriented Programming (OOP) principles. OOP offers several advantages, including modularity, reusability, scalability, and maintainability, which are crucial for building complex and robust systems.

In this report, we explore the structure of the e-commerce backend system, focusing on the design and implementation of essential components such as customers, admins,



products, orders, and carts. We will also discuss the key features, the use of OOP principles, and the overall architecture that drives the system's functionality.

System Design and Features

The e-commerce backend system is built around several key entities, each represented as a class. These classes interact with each other to provide functionalities like customer management, product catalog management, order placement, and administrative control. Below is a detailed explanation of the classes and how they contribute to the system's overall structure and behavior.

Customer Class

The Customer class represents the consumer or user of the e-commerce platform. The attributes of this class are designed to model a real-world customer’s profile.

1. **Attributes:**
    - **username:** A unique identifier for the customer, essential for login and authentication.
    - **password:** A secure password for the customer's account (best practice would be to hash this password for security reasons).
    - **Date Of Birth:** The customer's birth date. This information can be used for age-related discounts or restrictions.
    - **balance (store Credit):** The customer's store credit, which can be used to make purchases.
    - **address:** The delivery address for orders placed by the customer.
    - **gender:** An enumerated value representing the gender (either MALE or FEMALE).
    - **interests:** A list of interests (like shopping preferences or categories) that can be used to recommend products to the customer.
2. **Methods:**
    - Display Info(): This method provides an abstraction for displaying a customer's details. It is overridden in the Customer class, allowing the system to print out the customer's details in a readable format.

3. **OOP Principles Applied:**
    - Encapsulation: The attributes of the customer are private to prevent direct manipulation, and getters/setters are used to manage access.
    - Polymorphism: The displayInfo() method can be invoked on different types of users (like Admin or Customer), and the specific version of the method is executed depending on the object type.

Admin Class

The Admin class represents administrative users who have privileged access to manage the e-commerce platform. This class provides functionality to manage users, products, and orders.

1. **Attributes:**
    - username: The admin's unique identifier.
    - password: Admin's secure password (again, ideally hashed).
    - dateOfBirth: Birth date of the admin.
    - role: The role of the admin (e.g., "Product Manager", "Order Manager"), which defines their specific duties.
    - workingHours: The number of hours the admin is scheduled to work, which can be useful for operational purposes.

2. **Methods:**
    - Display Info(): Displays detailed information about the admin's profile.
    - Show Users(List<Customer> customers): Displays all the customers in the system.
    - Show Products(List<Product> products): Displays the list of available products in the system.
    - Show Orders(List<Order> orders): Displays all the orders that have been placed.
    - Create Product(Scanner scanner): Allows the admin to add new products to the product catalog.
    - Update Product(Scanner scanner): Enables the admin to modify the details of an existing product.
    - Delete Product(Scanner scanner): Enables the admin to remove a product from the catalog.
3. **OOP Principles Applied:**
    - Encapsulation: The sensitive data like password is private, and the admin can only access their own information through controlled methods.
    - Abstraction: Admin actions such as managing users and products are abstracted into specific methods that hide the internal complexity.

Category and Product Classes

The Category and Product classes are used to organize and represent products in the system.
The Category class is a way of organizing products into logical groups, making it easier for customers to find what they are looking for. The Product class represents an individual product available for purchase.

1. Attributes:
    - id: A unique identifier for the category.
    - name: A name that represents the category, such as "Electronics", "Clothing", or "Books".

Product Class:

1. Attributes:
    - name: The name of the product.
    - price: The price of the product.
    - category: A Category object representing the product's category.
2. Methods:
    - CRUD operations (create, read, update, delete) allow the admin to manage products in the system.

3. OOP Principles Applied:
    - Encapsulation: The product details are encapsulated within the class, preventing direct manipulation and ensuring proper validation during CRUD operations.

Cart and Order Classes

The Cart and Order classes are central to the process of managing customer purchases.

Cart Class: The Cart class models a shopping cart where products are temporarily stored before an order is placed.

1. Attributes:
    - products: A list of Product objects that the customer has added to their cart.
2. Methods:
    - addProduct(Product product): Adds a product to the cart.
    - viewCart(): Displays all the products currently in the cart.
    - calculateTotal(): Calculates the total cost of the products in the cart.
    - clearCart(): Clears all items from the cart once the order is placed.

Order Class:The Order class represents a finalized purchase.

1. Attributes:
    - details: A string that holds order information such as the products ordered, payment method, and customer.
    - timestamp: The time the order was placed.

2. OOP Principles Applied:
    - Abstraction: The cart and order systems abstract away the complexity of the purchase process. Customers interact with simple methods such as adding products or placing orders, while the internal workings (calculating totals, storing orders) are hidden.

Database Class: In-Memory Data Storage for E-Commerce System

The Database class in the e-commerce backend system is responsible for acting as the central repository of all the data required by the system. This includes storing information about customers, admins, products, categories, and orders. Since this system is designed for in-memory storage, the Database class uses static ArrayList objects to manage and hold the data for different entities throughout the runtime of the application. This ensures that all data related to the application is readily available to the system's various components, such as customer and admin operations.

Attributes of the Database Class

The Database class contains four main attributes, each serving a distinct purpose for the storage and management of the system's data:

1. **customers (Static ArrayList<Customer>):** The customers attribute is a static ArrayList that holds all the Customer objects who have registered with the e-commerce platform. Each Customer object contains personal details, including the customer's username, password, balance, address, gender, interests, and other relevant attributes. Since the customers list is static, it persists across different instances of the Database class, ensuring that the list remains consistent and accessible throughout the application's lifecycle.

2. **admins (Static ArrayList<Admin>):** Similar to the customers list, the admins attribute is a static ArrayList that stores all the Admin objects. Each Admin object contains administrative details such as username, password, role, and working hours. The static nature of this list allows admins to be consistently managed across the entire system, ensuring that access control remains secure.

3. **products (Static ArrayList<Product>):** The products attribute is a static ArrayList designed to hold all the available products that customers can purchase. Each Product object in the list contains relevant information such as the product's name, category, price, description, and quantity available in stock. The products list is a key part of the e-commerce system, enabling customers to browse and purchase products while allowing admins to manage the catalog.

4. **orders (Static ArrayList<Order>):** The orders attribute is a static ArrayList that stores all the completed orders placed by customers. Each Order object contains details about the products ordered, the payment method used, the order status, and the customer who made the order. This attribute provides the historical record of all transactions, allowing both customers and admins to view past orders and manage fulfillment.

Methods of the Database Class

The Database class includes a key method that is responsible for initializing and populating the database with sample data for testing and development purposes. The initializeDummyData() method is essential for simulating a real-world e-commerce platform before actual users interact with the system.

1. **Initialize Dummy Data():** This method populates the static ArrayList attributes with sample data. It creates a few dummy Customer, Admin, Product, and Order objects, adding them to their respective lists. For instance, it might create:Dummy Customers: A set of sample customers with different usernames, passwords, addresses, and interests.
   - **Dummy Admins:** A few admin users with roles like "SuperAdmin" or "ProductManager," each having specific working hours and credentials.

This dummy data helps developers and testers simulate real-world interactions with the system, test the functionalities, and ensure everything works correctly without the need for an actual database at the beginning of the project. It can be removed or replaced when the system is ready for production deployment with a full-fledged database.

OOP Principles Applied

The design of the Database class makes extensive use of key Object-Oriented Programming (OOP) principles, particularly Encapsulation.

1. **Encapsulation:** Encapsulation is the core OOP principle applied in the Database class. It ensures that the data is hidden and accessed through well-defined methods, promoting modularity, security, and maintainability. By encapsulating the data in static ArrayList attributes, the class ensures that customers, admins, products, and orders are managed internally, and their data is not directly exposed to other parts of the program. This prevents external manipulation and ensures that data can only be modified or accessed via specific methods that provide controlled interactions.
    - The customers, admins, products, and orders lists are kept private to ensure that no class outside of the Database class can directly modify them. Instead, operations on the data, such as adding, removing, or retrieving elements, are done through appropriate methods like initializeDummyData(), ensuring that the integrity of the data is preserved.
    - The methods that interact with these static lists ensure that the data is manipulated safely and efficiently, using defined rules and validation checks. For example, when an admin adds a new product, the Product object is added to the products list through a well-defined method that checks for valid input and ensures the product is correctly added.

2. **Data Integrity and Security:** By encapsulating the data within the Database class and controlling how the data is accessed and modified, the class also enforces data integrity and security. Unauthorized access to or modification of critical data (such as customer orders or product inventory) is prevented by ensuring that only authorized methods within the Database class can modify the data.

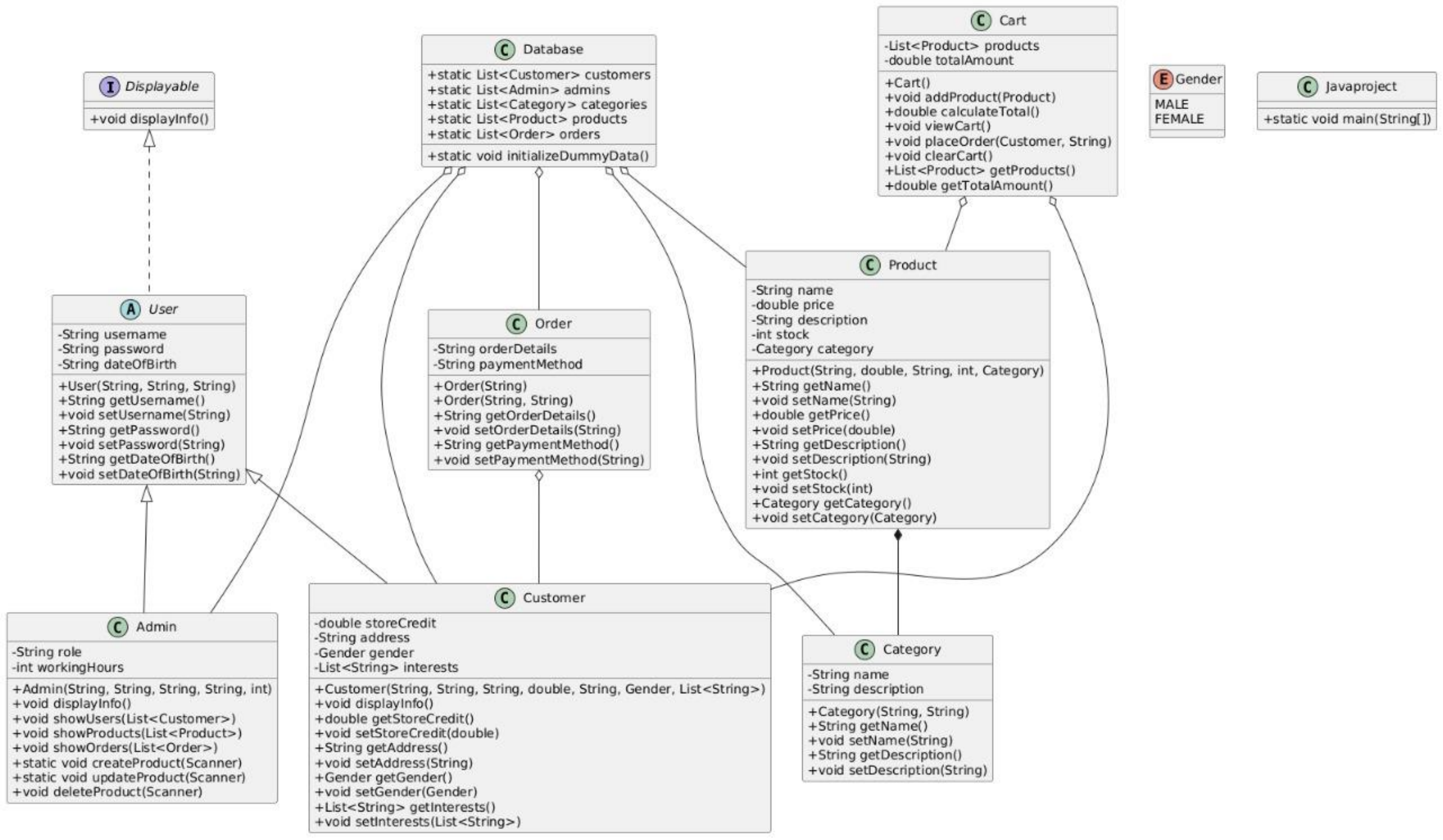
For example, customers cannot directly manipulate the admins or orders lists. Admins, on the other hand, can manage products but cannot modify the data associated with customers' private details. This separation of concerns reduces the risk of bugs or security breaches and ensures that the application functions as intended.

3. **Modularity:** The Database class also demonstrates modularity, another key OOP principle. By grouping the management of all data entities (customers, admins, products, orders) into a single class, the system becomes easier to maintain and extend. If any part of the database functionality needs to change, such as switching from an in-memory implementation to an actual database, the changes can be made in one central location without affecting other parts of the system.

4. **Maintainability:** Using OOP principles like encapsulation and modularity enhances the maintainability of the code. With encapsulated methods controlling the interactions with the static lists, any future updates to the data model (for example, adding more attributes to Customer or Product) can be managed within the Database class. The rest of the system will remain unaffected as long as the database methods are updated to handle these changes.

Main Program (Java project Class)

The Javaproject class serves as the main entry point for the e-commerce backend system. It brings together all the components of the system—customers, admins, products, carts, orders, and the database—into a cohesive user interface that facilitates interaction with the system. This class operates as the bridge between the user and the backend, providing the interface to carry out various actions such as logging in, managing products, handling carts, placing orders, and more.


Displayable
+void displayInfo()
Database
+static List<Customer> customers
+static List<Admin> admins
+static List<Category> categories
+static List<Product> products
+static List<Order> orders
+static void initializeDummyData()
Cart
-List<Product> products
-double totalAmount
+Cart()
+void addProduct(Product)
+double calculateTotal()
+void viewCart()
+void placeOrder(Customer, String)
+void clearCart()
+List<Product> getProducts()
+double getTotalAmount()
Gender
MALE
FEMALE
Javaproject
+static void main(String[])
User
-String username
-String password
-String dateOfBirth
+User(String, String, String)
+String getUsername()
+void setUsername(String)
+String getPassword()
+void setPassword(String)
+String getDateOfBirth()
+void setDateOfBirth(String)
Order
-String orderDetails
-String paymentMethod
+Order(String)
+Order(String, String)
+String getOrderDetails()
+void setOrderDetails(String)
+String getPaymentMethod()
+void setPaymentMethod(String)
Product
-String name
-double price
-String description
-int stock
-Category category
+Product(String, double, String, int, Category)
+String getName()
+void setName(String)
+double getPrice()
+void setPrice(double)
+String getDescription()
+void setDescription(String)
+int getStock()
+void setStock(int)
+Category getCategory()
+void setCategory(Category)
Admin
-String role
-int workingHours
+Admin(String, String, String, String, int)
+void displayInfo()
+void showUsers(List<Customer>)
+void showProducts(List<Product>)
+void showOrders(List<Order>)
+static void createProduct(Scanner)
+static void updateProduct(Scanner)
+void deleteProduct(Scanner)
Customer
-double storeCredit
-String address
-Gender gender
-List<String> interests
+Customer(String, String, String, double, String, Gender, List<String>)
+void displayInfo()
+double getStoreCredit()
+void setStoreCredit(double)
+String getAddress()
+void setAddress(String)
+Gender getGender()
+void setGender(Gender)
+List<String> getInterests()
+void setInterests(List<String>)
Category
-String name
-String description
+Category(String, String)
+String getName()
+void setName(String)
+String getDescription()
+void setDescription(String)

Key Functionalities in the Main Program

The main program in the Javaproject class is organized around several key functionalities that are essential for both customers and administrators to use the e-commerce platform effectively. Each of these functionalities is mapped to specific actions that the user can take, making it easy to navigate through the system.

1. Login and Registration
   One of the first steps when a user interacts with the system is logging in or registering. The system offers two types of users: customers and admins. Depending on the type of user, the system allows access to different functionalities.
   Login Process:
   The user is prompted to enter their credentials, including a username and password. The system verifies these credentials against the stored data in the database. There are several key steps in the login process:

- Prompting for Username and Password: The program prompts the user to enter their username and password.
- Checking Credentials: Once the credentials are entered, the program searches the customers or admins list (depending on the user type) to check if the credentials match any existing entries in the database.
- Successful Login: If the credentials are correct, the user is granted access to the system. Customers are directed to their main interface, while admins are given access to the admin controls.
- Failed Login: If the credentials are incorrect, an error message is displayed, and the user is prompted to try again.

Registration Process:

For new users (both customers and admins), the system offers a registration option. This process includes:

- Entering User Information: The user provides their personal details, such as username, password, email, date of birth, and any other relevant information.
- Storing Information in the Database: The newly registered user is added to the appropriate list in the database (either the customers or admins list).
- Confirmation Message: After successful registration, the system displays a confirmation message, and the user is redirected to the login page.

The login and registration system ensures that only authorized users can access sensitive features of the platform, such as managing products and orders.

2. Product Management

Product management is a crucial feature of the e-commerce backend system. For customers, the ability to browse and view products is a primary function. For admins, however, the system allows more advanced capabilities, including adding new products, updating existing ones, and deleting obsolete products from the catalog.

Customer Product Management:

For a customer, the primary interaction with the product catalog involves browsing available products. The customer can:

- View Product Details: The customer can view a list of all available products, which include essential information such as the product's name, category, price, and availability.
- Filter Products: The system may allow filtering of products based on categories (e.g., Electronics, Clothing, etc.) to enhance the shopping experience.
- Search Functionality: The customer may also search for specific products by name, category, or price range to quickly find what they need.

Admin Product Management:

Admins have the ability to manage the entire product catalog. This functionality provides the admin with full control over the system's inventory, ensuring that products can be added, updated, or removed as necessary. Admins can:

- Add New Products: Admins can enter product details such as the name, price, category, and description. This process also includes specifying the quantity available for sale.
- Update Existing Products: If a product's details change (e.g., price update, quantity change, or description update), the admin can modify the relevant information to keep the catalog up-to-date.
- Delete Products: If a product is discontinued or no longer available for sale, the admin can remove it from the catalog to ensure the customers only see active items.

Error Handling in Product Management:

- Invalid Input: The program checks for invalid input during product addition or updating (e.g., non-numeric values for price, missing product details). If such input is detected, the system displays an error message and prompts the user to enter valid data.
- Product Not Found: When updating or deleting a product, the system checks whether the product exists in the database. If the product is not found, an error message is displayed.

3. Cart Management

The cart is a critical feature for customers in an e-commerce system, allowing them to temporarily store products before placing an order. Cart management involves adding products to the cart, viewing the cart, and calculating the total price of the items.

Customer Cart Features:

- Add Products to Cart: Customers can add products to their shopping cart with a simple command. The product is added to a list in the cart, and the cart is updated with the new product's details.
- View Cart: Customers can view all the products currently in their cart, along with details such as product name, quantity, price per item, and total price.
- Remove Products from Cart: If a customer changes their mind about a product, they can remove it from the cart.
- Proceed to Checkout: When the customer is ready to purchase, they can proceed to checkout, where they will select their preferred payment method and place the order.

Order Placement and Payment:

Once the customer is satisfied with their cart, they can proceed to place an order. The payment method can be selected from a variety of options, such as:

- Store Credit: The customer can pay using the balance in their store account.
- Credit Card: The customer can use a credit card to complete the purchase.
- PayPal: The customer can also use PayPal to process the payment.

Once the payment is successful, the system generates an order, deducts the corresponding balance from the customer's store credit (if applicable), and clears the cart.

4. Admin Controls

Admins have a comprehensive set of tools to manage the e-commerce platform. The admin interface provides the following functionalities:

- View All Users: Admins can view a list of all customers registered on the platform.
- View All Products: Admins can view a complete list of products in the catalog.
- View All Orders: Admins can view a list of all orders placed by customers, along with the payment status and any other relevant details.

Additionally, admins can take action on these records, such as modifying user details, updating product information, and addressing issues related to orders. This level of access ensures that admins can effectively manage the platform's operations.

Error Handling in Admin Controls:

- Invalid Operations: Admin actions, such as deleting a product or modifying an order, are validated before execution. If an invalid operation is detected, such as trying to delete a non-existing product, the system displays an error message and asks the admin to re-enter the action.

5. Error Handling in the Main Program

Error handling is a critical aspect of any application, and the Javaproject class ensures that the system runs smoothly even in the event of unexpected errors. Several types of errors are anticipated and handled within the system, including:

Input Validation:

The system continuously validates user input to ensure that it is correct and appropriate. For example:

- Non-numeric Values for Price: When an admin tries to enter a product's price, the system checks if the input is a valid number.
- Invalid Username or Password: When a user logs in, the system verifies that the entered credentials match those stored in the database. If not, it prompts the user to try again.

Exception Handling:
The program uses try-catch blocks to handle exceptions that could arise during operations such as reading input, accessing the database, or performing calculations:

- Database Access Errors: If the program encounters any issues while accessing the database (e.g., trying to retrieve data from an empty list), an exception is thrown and caught, displaying a relevant error message to the user.
- Unexpected Errors: Other unforeseen errors, such as incorrect method calls or null reference errors, are caught by the catch block and handled gracefully, ensuring the program doesn't crash.

User Experience:
The error messages are designed to be user-friendly, helping the user understand the issue and providing clear instructions on how to proceed. For example, if a customer tries to add more products than are available in stock, the system will display an error message informing them of the issue.

User Class
The User class is a foundational element in the e-commerce system, designed to encapsulate the shared attributes and behaviors of all system users. It acts as a parent class for specific user types, such as Customer and Admin, promoting code reusability and maintainability.

The class includes private attributes for the user's username, password, and dateOfBirth, ensuring data security and encapsulation. These attributes are accessed and modified using public getter and setter methods. The class constructor initializes these attributes, allowing the creation of User objects with specific details.

Additionally, the displayInfo() method serves as a placeholder for displaying user information, which can be overridden by subclasses to provide more specific details about the user type. This method lays the groundwork for polymorphism, where derived classes can implement customized behaviors.

By encapsulating common properties and providing a base structure, the User class simplifies the management of user-related functionality across the system.

Conclusion

The Java project class is the backbone of the e-commerce backend system. It brings together a wide range of functionalities, such as login and registration, product management, cart management, and admin controls. Each feature is carefully designed to provide a seamless experience for both customers and administrators. By leveraging Object-Oriented Programming (OOP) principles, the system is modular, scalable, and maintainable.

The inclusion of robust error handling ensures that the system remains stable even when users make mistakes or unexpected issues arise. Through input validation and exception handling, the program can gracefully handle errors, ensuring a smooth user experience.

In summary, the main program ties all the components of the e-commerce system together, allowing users to interact with the platform effectively. The program is designed to be both flexible and extensible, making it a solid foundation for any e-commerce backend system.