Menu-Driven E-Commerce Management System Using Python

M Akash [CB.EN.U4EEE22024]

Athish J D [CB.EN.U4EEE22007]

Darsan L M [CB.EN.U4EEE22010]

Abstract-This project presents a menu-driven e-commerce management system implemented in Python. The system effectively demonstrates key object-oriented programming concepts such as inheritance and polymorphism. It provides role-based functionality with distinct operations for customers and administrators. Customers can browse products, add items to their cart, place orders, and view purchase history. Administrators can manage inventory by updating stock and pricing. The design emphasizes code reusability and maintainability through a well-structured class hierarchy.

Keywords: E-commerce, Python, Object-Oriented Programming, Inheritance, Polymorphism, Inventory Management, Customer Interface, Admin Panel

1. Introduction

In today's digital marketplace, efficient management systems are essential for handling online transactions and product inventories. This report details the development of a menu-driven e-commerce management system built using Python. The project illustrates the application of object-oriented programming techniques to create a system that serves two distinct roles: the customer and the administrator. The customer interface allows users to view products, add them to a shopping cart, and complete purchases. The administrator interface provides functionalities to update inventory levels and adjust pricing, ensuring smooth operational management.

2. Methodology

The system is designed using object-oriented principles to promote modularity and reuse. The inheritance structure is central to the system's design:

- **User (Base Class)**
  - **Attributes:** name, uid
  - **Purpose:** Provides common attributes for both customer and admin classes.

- **Customer (Child Class, Inherits from User)**
  - **Attributes:** username, password, cart, orders
  - **Methods:** auth(), display_products(), add_to_cart(), purchase()
  - **Composition:** Uses a Cart object to manage product selections, an Order object to handle purchases, and a Payment module for processing transactions.

- **Admin (Child Class, Inherits from User)**
  - **Attributes:** login, password, cats (categories managed)
  - **Methods:** auth(), update_availability(), update_price(), view_availability()
  - **Association:** Interacts with the Inventory class to manage stock and pricing.

- **Independent Classes**
  - **Cart:** Manages product items and calculates the total cost.
  - **Payment:** Simulates transaction processing with a static method.

- **Order:** Generates receipts, deducts stock, and creates a delivery schedule.
- **Inventory:** Contains static methods to update stock levels and prices.
- **Delivery:** Tracks order delivery details and schedules.

The methodology section also explains the overall code flow:

- **Login and Main Menu:** Customers and administrators log in through separate interfaces.
- **Process Flow:** On successful authentication, users are directed to their respective menus where they can perform tasks (such as browsing products, adding items to the cart, making purchases, or updating inventory).

3. Code

```
import random
from datetime import datetime, timedelta

# Global availability to track stock and prices
GLOBAL_AVAILABILITY = {
   cat: {prod: {"price": price, "stock": 100} for prod, price in items.items()}
   for cat, items in {
      "Formal": {"Blazer": 3500, "Shoes": 2000, "Tie": 500},
      "Casuals": {"T shirt": 600, "Jeans": 1500, "Sneakers": 2500},
   }.items()
}

        #Base Classes

        class User:
           def __init__(self, name, uid):
              self.name = name
              self.uid = uid

        class Cart:
           def __init__(self):
              self.items = {}

           def add_item(self, product, qty):
              self.items[product] = self.items.get(product, 0) + qty

           def clear_cart(self):
              self.items.clear()

           def view_cart(self):
              total = sum(self.items[i] * GLOBAL_AVAILABILITY[cat][i]["price"]
                          for cat in GLOBAL_AVAILABILITY for i in self.items if i in
        GLOBAL_AVAILABILITY[cat])
              return self.items, total

        class Payment:
           @staticmethod
           def simulate_payment(total):
              card = input("Enter card number (16 digits): ").strip()
              if len(card) == 16 and card.isdigit():
                 return True, f"Payment of Rs.{total} successful using card ending {card[-4:]}."
     return False, "Payment failed."

        #Customer Classes
```

```
class Customer(User):
    def __init__(self, name, uid, username, password):
        super().__init__(name, uid)
        self.username = username
        self.password = password
        self.cart = Cart()
        self.orders = []

    def auth(self, username, password):
        return self.username == username and self.password == password

    def display_products(self):
        print("Available products with stock:")
        for cat, items in GLOBAL_AVAILABILITY.items():
            print(f"\nCategory: {cat}")
            for prod, details in items.items():
                print(f"{prod}: Rs.{details['price']} (Stock: {details['stock']})")

    def add_to_cart(self, product, qty):
        for cat, items in GLOBAL_AVAILABILITY.items():
            if product in items and (qty := int(qty)) <= items[product]["stock"]:
                self.cart.add_item(product, qty)
                return f"Added {qty} of {product} to cart."
        return "Invalid product or stock unavailable."

    def purchase(self):
        if not self.cart.items:
            return "Cart is empty."
        _, total = self.cart.view_cart()
        ok, msg = Payment.simulate_payment(total)
        if not ok:
            return msg
        order = Order(self, self.cart, total)
        self.orders.append(order.generate_receipt())
        self.cart.clear_cart()
        return msg + "\n" + order.generate_receipt()

class Order:
    def __init__(self, customer, cart, total):
        self.order_no = random.randint(1000, 9999)
        self.order_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.customer = customer
        self.cart = cart.items.copy()
        self.total = total
        self.delivery = Delivery(self.order_no)

    def generate_receipt(self):
        receipt = f"Order #{self.order_no} placed at {self.order_time}\n"
        for cat in GLOBAL_AVAILABILITY:
            for item, qty in self.cart.items():
                if item in GLOBAL_AVAILABILITY[cat]:
                    GLOBAL_AVAILABILITY[cat][item]["stock"] -= qty
                    receipt += f"{item}: {qty} x Rs.{GLOBAL_AVAILABILITY[cat][item]['price']} =
Rs.{qty * GLOBAL_AVAILABILITY[cat][item]['price']}\n"
        receipt += f"Total: Rs.{self.total}\n"
        receipt += f"Scheduled Delivery: {self.delivery.date}\n"
return receipt

#Admin Classes
```

```
class Admin(User):
    def __init__(self, name, uid, login, password, cats):
        super().__init__(name, uid)
        self.login = login
        self.password = password
        self.cats = cats

    def auth(self, login, password):
        return self.login == login and self.password == password

    def update_availability(self):
        return Inventory.update_stock(self.cats)

    def update_price(self):
        return Inventory.update_price(self.cats)

    def view_availability(self):
        return Inventory.view_products()

class Inventory:
    @staticmethod
    def update_stock(categories):
        try:
            cat = input(f"Select category {categories}: ").strip()
            if cat not in categories:
                return "Invalid category."
            print(f"Available in {cat}: {list(GLOBAL_AVAILABILITY[cat].keys())}")
            prod = input("Enter product name: ").strip()
            stock = int(input("Enter additional stock: "))
            if prod in GLOBAL_AVAILABILITY[cat]:
                GLOBAL_AVAILABILITY[cat][prod]["stock"] += stock
            else:
                price = float(input("Enter price: "))
                GLOBAL_AVAILABILITY[cat][prod] = {"price": price, "stock": stock}
            return f"Updated {prod}."
        except Exception as e:
            return str(e)

    @staticmethod
    def update_price(categories):
        try:
            cat = input(f"Select category {categories}: ").strip()
            if cat not in categories:
                return "Invalid category."
            print(f"Available products for price update in {cat}:")
            for product, details in GLOBAL_AVAILABILITY[cat].items():
                print(f"{product}: Rs.{details['price']}")
            prod = input("Enter product name to update: ").strip()
            if prod not in GLOBAL_AVAILABILITY[cat]:
                return "Invalid product."
            GLOBAL_AVAILABILITY[cat][prod]["price"] = float(input("Enter new price: "))
            return f"Updated {prod} price."
        except Exception as e:
            return str(e)

    @staticmethod
    def view_products():
return GLOBAL_AVAILABILITY
```

```
    #Delivery Management

    class Delivery:
        deliveries = {}

        def __init__(self, order_no):
            self.order_no = order_no
            self.date = (datetime.now() + timedelta(days=random.randint(1, 7))).strftime("%Y-%m-%d")
            Delivery.deliveries[order_no] = self

        def info(self):
            return f"Order {self.order_no} will be delivered on {self.date}"

        @classmethod
        def find_delivery(cls, order_no):
            delivery = cls.deliveries.get(order_no)
return delivery.info() if delivery else "Delivery not found."

    #Admin Menu

    def admin_login():
        login = input("Enter admin login: ").strip()
        password = input("Enter admin password: ").strip()
        for admin in admins:
            if admin.auth(login, password):
                print(f"Welcome, {admin.name}!")
                admin_menu(admin)
                return
        print("Invalid credentials. Please try again.")

    def admin_menu(admin):
        while True:
            print("1. Update Stock")
            print("2. Update Price")
            print("3. View Products")
            print("4. Logout")
            choice = input("\nEnter your choice (1/2/3/4): ").strip()
            if choice == '1':
                print(admin.update_availability())
            elif choice == '2':
                print(admin.update_price())
            elif choice == '3':
                print(admin.view_availability())
            elif choice == '4':
                 print("Logging out.")
                 break

    #Customer Menu ------------------------
    def customer_login():
        username = input("Enter username: ").strip()
        password = input("Enter password: ").strip()

        for customer in customers:
            if customer.auth(username, password):
                print(f"Welcome, {customer.name}!")
                customer_menu(customer)
                return
        print("Invalid credentials. Please try again.")

    def customer_menu(customer):
```

```
        while True:
            print("1. View Products")
            print("2. Add to Cart")
            print("3. View Cart")
            print("4. Purchase")
            print("5. View Orders")
            print("6. Logout")
            choice = input("Enter your choice (1/2/3/4/5/6): ").strip()

            if choice == '1':
                customer.display_products()
            elif choice == '2':
                product = input("Enter product name to add to cart: ").strip()
                qty = input("Enter quantity: ").strip()
                print(customer.add_to_cart(product, qty))
            elif choice == '3':
                items, total = customer.cart.view_cart()
                print("Your Cart:", items, f"\nTotal: Rs.{total}")
            elif choice == '4':
                print(customer.purchase())
            elif choice == '5':
                print("Your Orders:")
                for order in customer.orders:
                    print(order)
            elif choice == '6':
                print("Logging out.")
                break
            else:
    print("Invalid choice. Please try again.")

    #Login and Main Menu

    customers = [
        Customer("User1", "101", "user1", "pass1"),
        Customer("User2", "102", "user2", "pass2")
    ]

    admins = [
        Admin("Admin1", "201", "admin1", "pass1", ["Formal", "Casuals"]),
        Admin("Admin2", "202", "admin2", "pass2", ["Formal"]),
        Admin("Admin3", "203", "admin3", "pass3", ["Casuals"])
    ]

    def main_menu():
        print("Welcome to the Shopping Management System!")
        while True:
            print("\n1. Customer Login")
            print("2. Admin Login")
            print("3. Exit")
            choice = input("Enter your choice (1/2/3): ").strip()

            if choice == '1':
                customer_login()
            elif choice == '2':
                admin_login()
            elif choice == '3':
                print("Exiting. Thank you!")
                break
            else:
                print("Invalid choice. Please try again.")
```

```
    if __name__ == "__main__":
main_menu()
```

4. Results and Discussions

The code starts by importing necessary modules like datetime and timedelta to handle date and time operations, and random to generate random order numbers and simulate delivery timeframes. By enabling features like the creation of distinct order numbers and the computation of order delivery dates, these imports provide the program's framework. These import statements would be easily visible in a screenshot of the top section of the code editor.

The application creates a global dictionary named GLOBAL_AVAILABILITY after the imports. This dictionary is set up to save inventory information for the "Formal" and "Casuals" product categories. The price and a default supply of 100 units are stored for every product. In order to verify product availability and update stock levels, the system as a whole uses this global variable as a central store for product information. This crucial element would be highlighted in an editor screenshot showing the startup of GLOBAL_AVAILABILITY.

The code then defines two fundamental classes: User and Cart. Both customer and admin objects are built on top of the User class, a base class that offers shared information like the user's name and unique ID. The things that a consumer chooses are managed by the Cart class. Along with a total cost estimate, it offers ways to add products, clear the basket, and display the contents as of right now. In order to calculate the cost based on the product price and quantity, the view_cart method iterates over the products in the global availability dictionary. The User and Cart class definitions are shown in a snapshot in this section, along with inline comments that describe each method.

A static method called simulate_payment is then used to introduce the Payment class. By asking the user for a 16-digit card number, verifying the entry, and then giving a success message with the card's last four numbers if the input is accurate, this method mimics the payment process. The input returns a failure message if validation is unsuccessful. A terminal snapshot displaying the card number prompt and the following message can be used to record the intended outcome of this method.

The Customer class, which derives from User, contains the functions pertaining to customers. By adding features like a Cart object, a list to store order receipts, a username, and a password, this class expands on the base class. It contains functions to handle the purchase process, add items to the cart (while ensuring there is enough inventory), display available products by iterating through the global availability dictionary, and authenticate customers. In addition to simulating the payment, the purchase function creates an Order object, adds the order receipt to the customer's order history, and then removes the item from the cart. The product display, the updated cart after adding items, and the final receipt following a transaction would all be visible in screenshots of the customer interface.

The Order class plays a crucial role by generating a unique order number and capturing the exact time of order placement. After calculating the total cost and copying the contents of the cart, it instantiates a Delivery object to build a delivery schedule. By subtracting the acquired amounts from the global stock, summarizing the transaction data, and adding the scheduled delivery date, the generate_receipt method creates a comprehensive receipt. This

output would be easily demonstrated by a screenshot of an order receipt that is printed in the console.

The Admin class, which inherits from User and is further extended with characteristics for login credentials and the product categories they oversee, is defined for administrative functionality. It includes methods for calling Inventory class functionalities and authenticating the administrator. In order to update product stock, modify product prices, or just inspect the products that are available, the Inventory class is made up of just static methods. Error handling is incorporated to efficiently handle faulty inputs. After a successful login, a console snapshot would display the admin menu with choices to see goods, update prices, and update stock.

The Delivery class is dedicated to managing the shipping details of each order. It maintains this data in a class-level dictionary and, at initialization, determines a delivery date at random from one to seven days from the current date. The class function find_delivery makes it possible to retrieve the formatted string that the method info produces, which tells the user when the order is expected to be delivered, by entering an order number. A visual confirmation of this functionality would be a snapshot showing the shipping information for a specific order.

A series of menus and login features are used to coordinate user engagement. After successfully requesting credentials, the admin login function gives the admin access to an admin menu where they can see goods, update prices, update stock, or log out. Similar to this, the customer login feature verifies the user's information and, if successful, presents a menu with choices to see products, add items to the cart, view the cart, make a purchase, or view previous orders. The associated methods from the Customer or Admin classes are called by each of these operations. While later screenshots would show the specific options accessible in each user flow, a screenshot of the main menu would show the first prompt with the options for Customer Login, Admin Login, or Exit.

Lastly, lists of customer and admin objects with predetermined credentials are created by the software to set up sample data. Based on user input, the main_menu function then continually shows the main menu and directs users to the relevant login functions. Until the user chooses to stop using the application, this loop keeps going. A screenshot of the application's main menu is a great method to show where users enter the software and set the scene for subsequent user interactions. All cases were extensively discussed and the screenshots are added below

In a menu-driven e-commerce management system, the code is generally divided into logical portions that govern different operations. To offer a complete shopping experience, every element is expertly connected, from controlling product availability, processing payments, creating orders, handling user identification, and scheduling delivery. Along with demonstrating the program's functioning, the accompanying screenshots at each crucial stage offer visual proof of the outcomes achieved during execution.

```
Welcome to the Shopping Management System!

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  cust1
Enter password:  pass1
Invalid credentials. Please try again.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  user1
Enter password:  pass1
Welcome, User1!
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  1
Available products with stock:

Category: Formal
Blazer: Rs.3500 (Stock: 100)
Shoes: Rs.2000 (Stock: 96)
Tie: Rs.500 (Stock: 110)

Category: Casuals
T shirt: Rs.600 (Stock: 100)
Jeans: Rs.1400.0 (Stock: 100)
Sneakers: Rs.2500 (Stock: 100)
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  2
Enter product name to add to cart:  Jeans
Enter quantity:  2
```

```
Welcome to the Shopping Management System!

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  cust1
Enter password:  pass1
Invalid credentials. Please try again.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  user1
Enter password:  pass1
Welcome, User1!
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  1
Available products with stock:

Category: Formal
Blazer: Rs.3500 (Stock: 100)
Shoes: Rs.2000 (Stock: 96)
Tie: Rs.500 (Stock: 110)

Category: Casuals
T shirt: Rs.600 (Stock: 100)
Jeans: Rs.1400.0 (Stock: 100)
Sneakers: Rs.2500 (Stock: 100)
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  2
Enter product name to add to cart:  Jeans
Enter quantity:  2
```

```
Added 2 of Jeans to cart.
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  3
Your Cart: {'Jeans': 2}
Total: Rs.2800.0
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  4
Enter card number (16 digits):  1234567812345678
Payment of Rs.2800.0 successful using card ending 5678.
Order #4352 placed at 2025-04-06 14:33:36
Jeans: 2 x Rs.1400.0 = Rs.2800.0
Total: Rs.2800.0
Scheduled Delivery: 2025-04-11

1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  5
Your Orders:
Order #4352 placed at 2025-04-06 14:33:36
Jeans: 2 x Rs.1400.0 = Rs.2800.0
Total: Rs.2800.0
Scheduled Delivery: 2025-04-11

1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  6
```

```
Added 2 of Jeans to cart.
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  3
Your Cart: {'Jeans': 2}
Total: Rs.2800.0
1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  4
Enter card number (16 digits):  1234567812345678
Payment of Rs.2800.0 successful using card ending 5678.
Order #4352 placed at 2025-04-06 14:33:36
Jeans: 2 x Rs.1400.0 = Rs.2800.0
Total: Rs.2800.0
Scheduled Delivery: 2025-04-11

1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  5
Your Orders:
Order #4352 placed at 2025-04-06 14:33:36
Jeans: 2 x Rs.1400.0 = Rs.2800.0
Total: Rs.2800.0
Scheduled Delivery: 2025-04-11

1. View Products
2. Add to Cart
3. View Cart
4. Purchase
5. View Orders
6. Logout
Enter your choice (1/2/3/4/5/6):  6
```

```
Logging out.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  admin2
Enter password:  pass2
Invalid credentials. Please try again.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  2
Enter admin login:  admin1
Enter admin password:  pass1
Welcome, Admin1!
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  1
Select category ['Formal', 'Casuals']:  Formal
Available in Formal: ['Blazer', 'Shoes', 'Tie']
Enter product name:  Blazer
Enter additional stock:  12
Updated Blazer.
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  2
Select category ['Formal', 'Casuals']:  Casuals
Available products for price update in Casuals:
T shirt: Rs.600
Jeans: Rs.1400.0
Sneakers: Rs.2500
Enter product name to update:  Sneakers
Enter new price:  2300
Updated Sneakers price.
1. Update Stock
2. Update Price
```

```
Logging out.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  1
Enter username:  admin2
Enter password:  pass2
Invalid credentials. Please try again.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  2
Enter admin login:  admin1
Enter admin password:  pass1
Welcome, Admin1!
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  1
Select category ['Formal', 'Casuals']:  Formal
Available in Formal: ['Blazer', 'Shoes', 'Tie']
Enter product name:  Blazer
Enter additional stock:  12
Updated Blazer.
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  2
Select category ['Formal', 'Casuals']:  Casuals
Available products for price update in Casuals:
T shirt: Rs.600
Jeans: Rs.1400.0
Sneakers: Rs.2500
Enter product name to update:  Sneakers
Enter new price:  2300
Updated Sneakers price.
1. Update Stock
2. Update Price
```

```
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  2
Select category ['Formal', 'Casuals']:  Casuals
Available products for price update in Casuals:
T shirt: Rs.600
Jeans: Rs.1400.0
Sneakers: Rs.2500
Enter product name to update:  Sneakers
Enter new price:  2300
Updated Sneakers price.
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  3
{'Formal': {'Blazer': {'price': 3500, 'stock': 112}, 'Shoes': {'price': 2000, 'stock': 96}, 'Tie': {'price': 500, 'stock': 110}}, 'Casuals': {'T shirt':
{'price': 600, 'stock': 100}, 'Jeans': {'price': 1400.0, 'stock': 96}, 'Sneakers': {'price': 2300.0, 'stock': 100}}}
1. Update Stock
2. Update Price
3. View Products
4. Logout

Enter your choice (1/2/3/4):  4
Logging out.

1. Customer Login
2. Admin Login
3. Exit
Enter your choice (1/2/3):  3
Exiting. Thank you!
```

5. Conclusions

The Python-based Shopping Management System presented here serves as a robust and interactive simulation of a real-world e-commerce application. Through the implementation of object-oriented programming principles, the system effectively separates concerns into logical classes such as `User`, `Customer`, `Admin`, `Cart`, `Order`, `Inventory`, and `Delivery`, each handling a specific aspect of the workflow. Customers can browse products, add items to their cart, proceed to secure simulated payments, and receive digital receipts along with scheduled delivery dates. Simultaneously, admins are empowered with tools to manage stock levels and update product pricing, ensuring dynamic control over the inventory.

The system's menu-driven interface makes it user-friendly and easy to navigate for both customers and administrators. Exception handling and validation mechanisms contribute to a more reliable experience by preventing invalid inputs and ensuring consistency in data handling. Additionally, the use of global availability tracking ensures centralized management of stock and pricing, which mirrors real-world practices in inventory systems.

Overall, this project not only demonstrates practical application of Python programming but also highlights how a modular approach can lead to clean, maintainable, and scalable code. With potential for further enhancements like database integration, GUI development, or multi-user handling, this system lays a strong foundation for more advanced and commercial-level applications.

References

[1] A. Almazan, B. Radulovic and Z. Covic, "Transaction in object oriented programming," 2009 7th International Symposium on Intelligent Systems and Informatics, Subotica, Serbia, 2009, pp. 73-75, doi: 10.1109/SISY.2009.5291192. keywords: {Object oriented programming;Transaction databases;Internet;Informatics;Electronic commerce;Object oriented databases;Logic programming},

[2] https://www.geeksforgeeks.org/implementing-interactive-online-shopping-in-c/