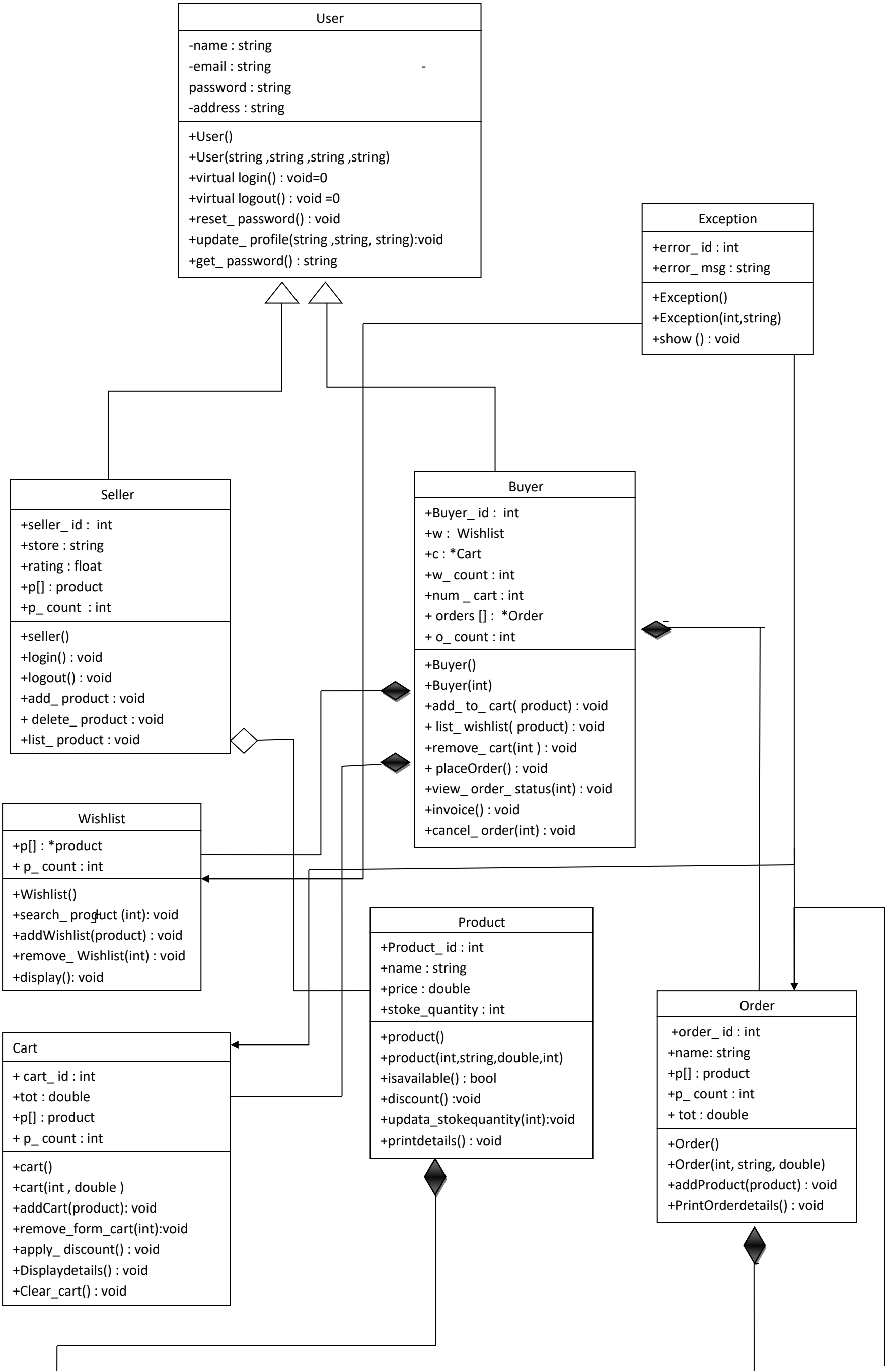
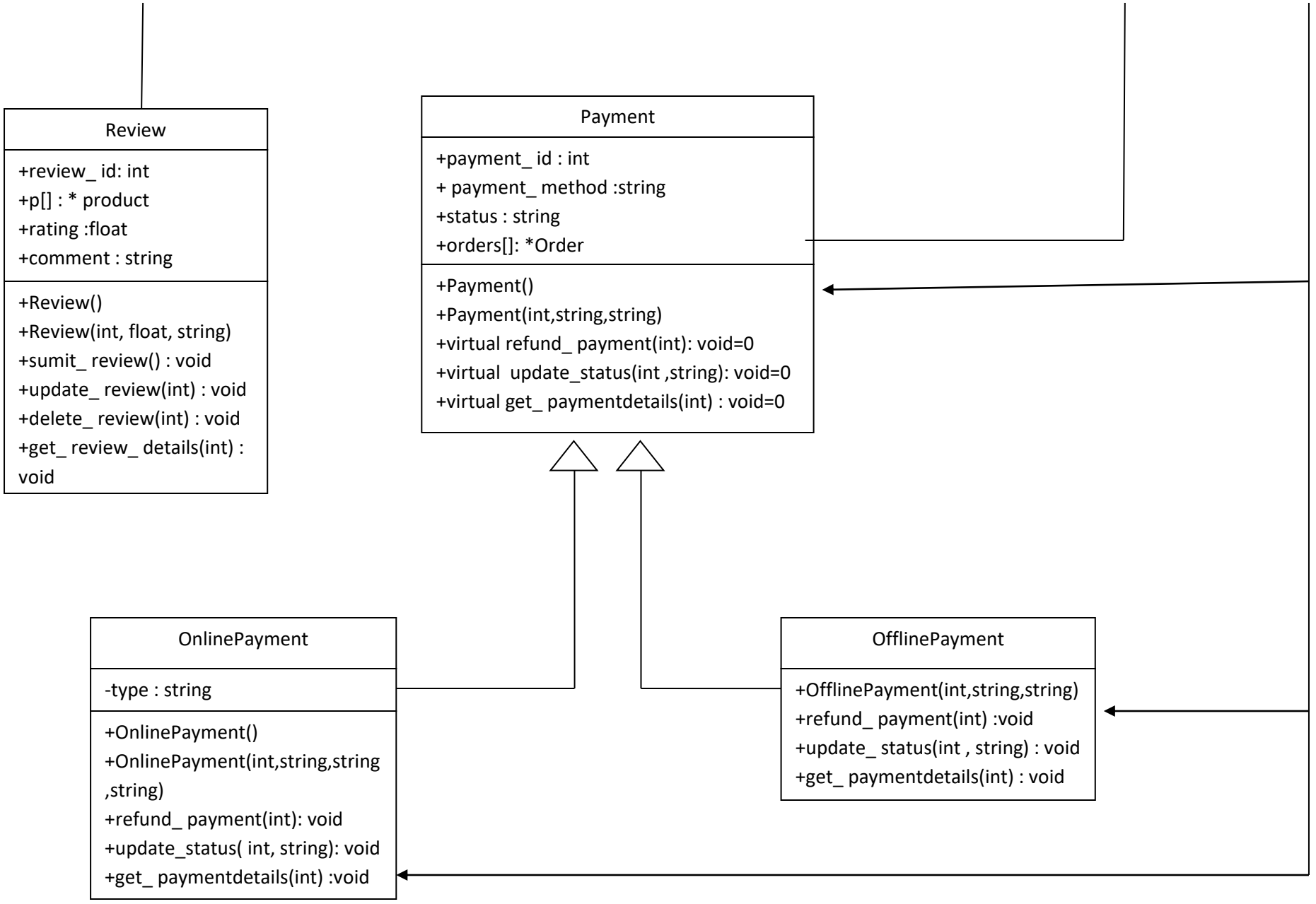
Problem definition (Description)

This online shopping code simulates the core functionalities of an e-commerce platform, encompassing multiple classes to handle different aspects of the system. The user class serves as the base class for both seller and Buyer, managing common user attributes like name, email, password, and address. The seller class allows sellers to add, delete, and list products, while Buyer includes functionalities to manage a shopping cart, wishlist, and orders. The product class represents individual items with attributes such as ID, name, price, and stock quantity. The Cart class enables buyers to add, remove, and view items, applying discounts based on total value. The Order class tracks orders placed by buyers, containing product details and total amounts. Payment classes (Payment, OnlinePayment, and OfflinePayment) handle different payment methods and statuses, providing options for refund and status updates. The Review class allows buyers to submit, update, delete, and retrieve product reviews. The main function demonstrates a sequence of operations, including seller login, product management, buyer wishlist and cart operations, placing orders, viewing order statuses, handling payments, and managing product reviews. The code also includes custom exception handling through the Exception class to manage various errors.

List of objects identified

1. Exception
2. User
3. product
4. seller
5. wishlist
6. Cart
7. Order
8. Buyer
9. Payment
10. OnlinePayment
11. OfflinePayment
12. Review


User
-name : string
-email : string
password : string
-address : string
+User()
+User(string ,string ,string ,string)
+virtual login() : void=0
+virtual logout() : void =0
+reset_ password() : void
+update_ profile(string ,string, string):void
+get_ password() : string
Exception
+error_ id : int
+error_ msg : string
+Exception()
+Exception(int,string)
+show () : void
Seller
+seller_ id : int
+store : string
+rating : float
+p[] : product
+p_ count : int
+seller()
+login() : void
+logout() : void
+add_ product : void
+ delete_ product : void
+list_ product : void
Buyer
+Buyer_ id : int
+w : Wishlist
+c : *Cart
+w_ count : int
+num _ cart : int
+ orders [] : *Order
+ o_ count : int
+Buyer()
+Buyer(int)
+add_ to_ cart(product) : void
+ list_ wishlist(product) : void
+remove_ cart(int) : void
+ placeOrder() : void
+view_ order_ status(int) : void
+invoice() : void
+cancel_ order(int) : void
Wishlist
+p[] : *product
+ p_ count : int
+Wishlist()
+search_ product (int): void
+addWishlist(product) : void
+remove_ Wishlist(int) : void
+display(): void
Product
+Product_ id : int
+name : string
+price : double
+stoke_quantity : int
+product()
+product(int,string,double,int)
+isavailable() : bool
+discount() :void
+updata_stokequantity(int):void
+printdetails() : void
Order
+order_ id : int
+name: string
+p[] : product
+p_ count : int
+ tot : double
+Order()
+Order(int, string, double)
+addProduct(product) : void
+PrintOrderdetails() : void
Cart
+ cart_ id : int
+tot : double
+p[] : product
+ p_ count : int
+cart()
+cart(int , double)
+addCart(product): void
+remove_form_cart(int):void
+apply_ discount() : void
+Displaydetails() : void
+Clear_cart() : void



Description of each class

1. user:

- This base class encapsulates common attributes and behaviors of any user in the system. Attributes include name, email, password, and address. The class provides fundamental operations such as login() to simulate user login, logout() for logging out, update_profile() to update user details, and reset_password() to change the user's password. The methods are designed to be overridden by subclasses.

2. product:

- This class models a product in the online store. Key attributes are product_id, name, price, and stock_quantity. Methods include is_available() to check if the product is in stock, update_stockquantity() to update the stock level, discount() to apply a discount based on the price, and printdetails() to display product information.

3.seller:

- Inherits from the user class, representing a seller on the platform. Additional attributes include sellerid, store, rating, and a list of products (p). Methods include login() and logout() for seller-specific login/logout operations, addProduct() to add new products to the seller's list, deleteProduct() to remove a product, and list_product() to display all products offered by the seller.

4. wishlist:

- Represents a buyer's wishlist. It maintains a list of products (p) that the buyer is interested in but has not yet purchased. Methods include searchProduct() to find a product by its ID, addWishlist() to add a product to the wishlist, remove_wishlist() to remove a product, and display() to show all products in the wishlist.

5. Cart:

- Models a shopping cart for a buyer. Attributes include cart_id, buyer_id, tot (total price), and a list of products (p). Methods include addCart() to add a product to the cart, removefrom_cart() to remove a product, applyDiscount() to apply a discount based on the total price, displaydetails() to show all products and the total price, and clear_cart() to empty the cart.

6. Order:

- Represents an order placed by a buyer. Attributes include order_id, buyer_name, a list of products (products), product_count, and total_amount. Methods include addProduct() to add a product to the order and printOrderDetails() to display the order's details, including each product and the total amount.

7. Buyer:

- Inherits from the user class and represents a buyer. Additional attributes include buyer_id, a wishlist, a Cart, and a list of orders (orders). Methods include add_to_cart() to add products to the cart, list_Wishlist() to add products to the wishlist, remove_from_cart() to remove products from the cart, place_order() to place an order for all items in the cart, view_order_status() to check the status of a specific order, invoice() to display all orders, and cancel_order() to cancel a specific order.

8. Payment:

- An abstract base class that models a payment. Attributes include payment_id, order, payment_method, and status. It declares virtual methods refund_payment(), update_payment_status(), and get_payment_details() which must be implemented by derived classes.

9. OnlinePayment:

- Inherits from the Payment class and represents an online payment. Additional attribute type specifies the type of online payment. Implements methods refund_payment() to process refunds, update_payment_status() to change the status of a payment, and get_payment_details() to retrieve details of a payment.

10. OfflinePayment:

- Also inherits from the Payment class and represents an offline payment. Implements methods refund_payment() to process refunds, update_payment_status() to change the payment status, and get_payment_details() to retrieve payment details.

11. Review:

- Represents a product review submitted by a buyer. Attributes include review_id, a pointer to the reviewed product (p), buyer_id, rating, and comment. Methods include submit_review() to submit a new review, update_review() to update an existing review, delete_review() to remove a review, and get_review_details() to display the details of a review.

Main Function

The main function of this program encapsulates an e-commerce system's operations, orchestrating interactions between users, products, orders, payments, and reviews. It initiates by simulating a seller's setup, adding products to their store. Then, it mimics a buyer's journey, including managing a wishlist and a cart, placing an order, and viewing order details and invoices. It also models payment processing, showcasing online payment details. Furthermore, it demonstrates the functionality of submitting, retrieving, updating, and deleting reviews for products. Exception handling ensures graceful error

management throughout these processes, enhancing user experience and system reliability. Through this succinct orchestration, the main function encapsulates the essence of an e-commerce ecosystem's core functionalities.

Uses of Standard Design Pattern

1. Factory Method Pattern:

- While not explicitly implemented as a pattern, the concept is implied in object creation throughout the code. Classes like Buyer, Seller, product, Order, and Payment serve as factories for creating instances of their respective types, encapsulating the creation logic.

2. Singleton Pattern:

- Although not explicitly applied in the code, the Singleton pattern ensures that only one instance of a class exists throughout the program's execution. This could be utilized, for instance, to manage a centralized configuration or resource pool within the application, ensuring global access and state consistency.