NADA GAMAL

Ecommerce System Design

APRIL 7, 2025

FAWRY RISE JOURNEY

Introduction

This report documents the design and implementation of a simplified e-commerce system using object-oriented principles. The system supports product definition, expiration and shipping features, cart management, and checkout with shipping calculations and error validation.

Design Overview

The system is designed using key OOP principles like encapsulation, composition over inheritance, and the Strategy design pattern. Product behaviors such as expiring or shipping are handled via interfaces and passed as optional behaviors, allowing extension without modifying the Product class (Open/Closed Principle).

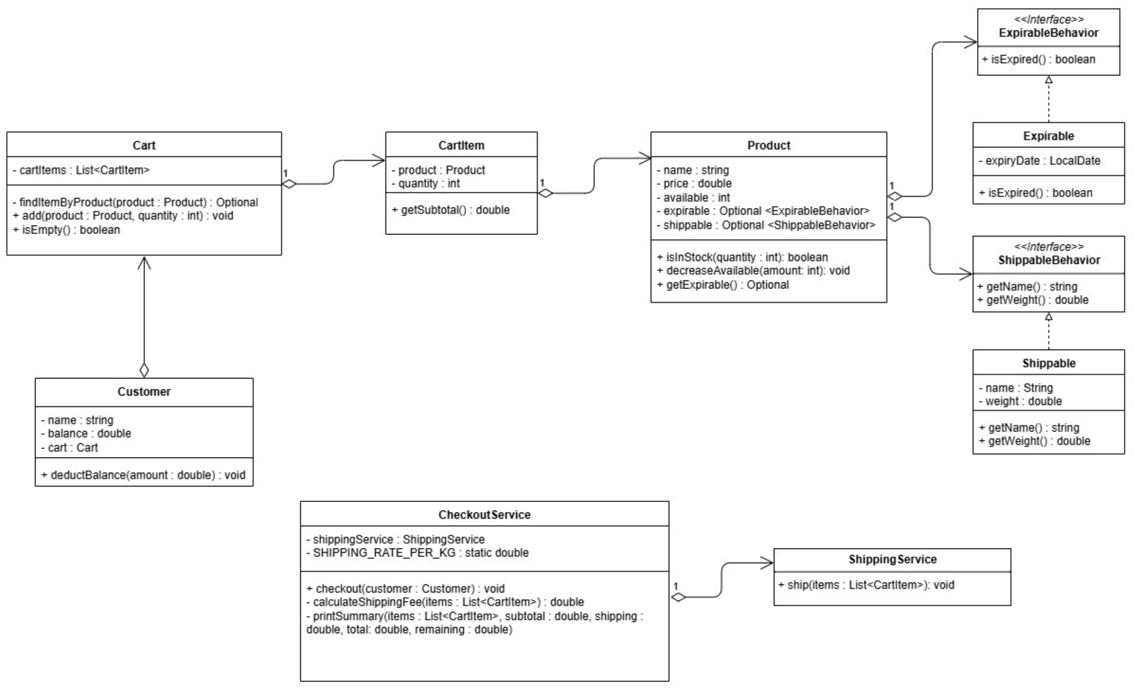
Class Descriptions

Class	Responsibility
`Product`	Represents a product with optional behaviors (shippable, expirable)
`Cart`	Holds items added by a customer
`CartItem`	Represents a product and its quantity in the cart
Customer	Owns a cart and a balance
CheckoutService	Coordinates checkout logic, including subtotal and shipping
ShippingService	Handles printing shipment details
ExpirableBehavior	Interface for checking if a product is expired
ShippableBehavior	Interface for accessing weight and name for shipping

Design Patterns Used

Strategy Pattern is used to allow dynamic assignment of behaviors like expiration and shipping logic via interfaces. This decouples behavior from the main product structure and follows the Open/Closed Principle.

UML Class Diagram



<<Interface>>
ExpirableBehavior
+ isExpired() : boolean
Expirable
- expiryDate : LocalDate
+ isExpired() : boolean
Cart
- cartItems : List<CartItem>
- findItemByProduct(product : Product) : Optional
+ add(product : Product, quantity : int) : void
+ isEmpty() : boolean
1
CartItem
- product : Product
- quantity : int
+ getSubtotal() : double
1
Product
- name : string
- price : double
- available : int
- expirable : Optional <ExpirableBehavior>
- shippable : Optional <ShippableBehavior>
+ isInStock(quantity : int): boolean
+ decreaseAvailable(amount: int): void
+ getExpirable() : Optional
1
1
<<Interface>>
ShippableBehavior
+ getName() : string
+ getWeight() : double
Shippable
- name : String
- weight : double
+ getName() : string
+ getWeight() : double
Customer
- name : string
- balance : double
- cart : Cart
+ deductBalance(amount : double) : void
CheckoutService
- shippingService : ShippingService
- SHIPPING_RATE_PER_KG : static double
+ checkout(customer : Customer) : void
- calculateShippingFee(items : List<CartItem>) : double
- printSummary(items : List<CartItem>, subtotal : double, shipping : double, total: double, remaining : double)
1
ShippingService
+ ship(items : List<CartItem>): void

Sample Output

Sample #1 Main Code

```
cart.add(cheese, 2);
cart.add(biscuits, 1);
cart.add(scratchCard, 1);
checkoutService.checkout(customer);
```

Console Output

```
** Shipment Notice **
2x Cheese 400g
1x Biscuits 700g
Total package weight 1.1kg

** Checkout Receipt **
2x Cheese 200
1x Biscuits 150
1x Scratch Card 20
----------------------
Subtotal 370
Shipping 30
Amount 400
Remaining 600
```

Sample #2 Main Code

```
Customer customer = new Customer("Nada", 1000);
Cart cart = customer.getCart();  // No items added

CheckoutService checkoutService = new CheckoutService(new ShippingService());

try {
    checkoutService.checkout(customer);
} catch (Exception e) {
    System.out.println("Checkout failed: " + e.getMessage());
}
```

Console Output

```
Checkout failed: Cart is empty.
```

Sample #3 Main Code

```
Customer customer = new Customer("Nada", 100);  // Low balance
Cart cart = customer.getCart();

cart.add(new Product("TV", 500, 1, Optional.empty(), Optional.empty()), 1);

CheckoutService checkoutService = new CheckoutService(new ShippingService());

try {
    checkoutService.checkout(customer);
} catch (Exception e) {
    System.out.println("Checkout failed: " + e.getMessage());
}
```

Console Output

```
Checkout failed: Insufficient balance.
```

Conclusion

This object-oriented design provides a flexible foundation for future extensions such as discount strategies, digital product delivery, and order history tracking. By isolating behaviors via interfaces, the system supports change and growth with minimal impact to core components.