ONLINE SHOPPING CART

A Python Project Submitted

in

Information Technology

by

KESHAV KRISHAN {2401330130141}

Under the Supervision of

Mr. ASHOK THAKUR

Assistant Prof., MCA

Department Of

Information Technology

NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY,

GREATER NOIDA

(An Autonomous Institute)

Affiliated to

DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW

June, 2025

CERTIFICATE

This is to certify that **KESHAV KRISHAN** has successfully completed Project on **ONLINE SHOPING CART** in **INFORMATION TECHNOLOGY** at **Noida Institute of Engineering and Technology**.

19/06/2025

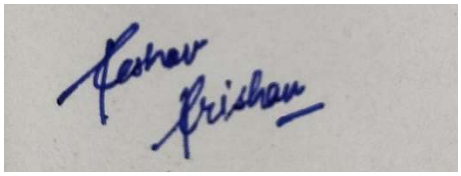
Date Signature Designation

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to **Noida Institute of Engineering and Technology** for allowing me to work on Project- "**ONLINE SHOPPING CART**". I am grateful to my Mentor **ASHOK THAKUR** for their guidance and support throughout the project. I also acknowledge the contributions of faculty and staffs who helped me complete this project.

Thank you all for your support and encouragement.

Sincerely,



KESHAV KRISHAN

1. Introduction

Project Overview

The **Online Shopping Cart** is a Python-based console application that mimics the behavior of a real-world shopping cart system used in e-commerce platforms. This system allows users to browse products, add or remove them from a cart, adjust quantities, view cost summaries including tax, and proceed to checkout. The project is designed using Object-Oriented Programming (OOP) principles to ensure scalability, reusability, and maintainability of the code.

Objective

- To design and implement a class-based architecture for managing products and shopping cart operations.
- To differentiate product types (physical and digital) using inheritance and polymorphism.
- To allow users to interact with the cart system through a command-line interface.
- To build a modular and intuitive system using encapsulation, abstraction, and composition.

Significance

Shopping carts are the backbone of any online retail application. This project simulates the core functions of such a system in a simplified educational context. It enhances understanding of real-world software engineering principles like class hierarchy, object interactions, and dynamic behavior based on data input. The significance lies in transforming abstract concepts of OOP into tangible application logic.

2. Literature Review

Background Study

Shopping carts were introduced to mimic the process of in-store shopping, allowing customers to collect and manage items before purchasing. With the rise of e-commerce, digital shopping carts have evolved into sophisticated systems capable of handling user sessions, inventory checks, promotions, and financial transactions.

Relevant Research

Research in software design patterns highlights the importance of **object-oriented programming** in managing systems with multiple interacting entities. E-commerce systems often rely on **inheritance** to create flexible product models, **composition** to group cart items, and **polymorphism** to standardize operations across different product types.

Key Findings

- Modular class structures increase maintainability and testability.
- Using encapsulation prevents accidental changes to critical internal state.
- A cart system must balance user convenience with inventory accuracy, requiring careful management of product quantities.
- Text-based interfaces, while simple, can effectively simulate user interactions and are excellent for prototyping.

3. Methodology

Research Design

The system was built using Python classes to represent real-world entities like `Product`, `PhysicalProduct`, `DigitalProduct`, `CartItem`, and `ShoppingCart`. Each class encapsulates attributes and behaviors relevant to its role in the shopping process. A procedural `main()` function serves as the user interface, allowing real-time interaction via keyboard inputs.

Data Collection Methods

Products were defined within the system as a pre-filled catalog. Users interactively select products by ID and specify quantities. Each user action triggers object method calls that update the internal state of the application — for example, reducing available stock or recalculating totals.

Data Analysis Techniques

The system calculates financial information such as subtotals, tax (at a fixed rate), and grand total. Each cart item contributes a subtotal based on product price and quantity. Class methods like `calculate_subtotal()` and `get_grand_total()` are used for these computations. The logic ensures accuracy in quantity management, disallowing invalid operations like over-purchasing or negative quantities.

4. CODE:

Online_shopping_cart.py U

.vscode > Online_shopping_cart.py > Product > name

```python
# ==================== Product Classes ====================
class Product:
    def __init__(self, product_id, name, price, quantity_available):
        # Base product initialization
        self._product_id = product_id
        self._name = name
        self._price = price
        self._quantity_available = quantity_available

    # Read-only properties for encapsulation
    @property
    def product_id(self): return self._product_id

    @property
    def name(self): return self._name

    @property
    def price(self): return self._price

    @property
    def quantity_available(self): return self._quantity_available

    # Set quantity with validation
    @quantity_available.setter
    def quantity_available(self, value):
        if value >= 0:
            self._quantity_available = value

    # Reduce stock if enough quantity is available
    def decrease_quantity(self, amount):
        if 0 < amount <= self._quantity_available:
            self._quantity_available -= amount
            return True
        return False

    # Increase stock
    def increase_quantity(self, amount):
        if amount > 0:
            self._quantity_available += amount
```

```python
    # Increase stock
    def increase_quantity(self, amount):
        if amount > 0:
            self._quantity_available += amount

    # Display basic product info
    def display_details(self):
        return f"ID: {self.product_id}, Name: {self.name}, Price: ${self.price}, Stock: {self.quantity_available}"

# Subclass for physical products with weight
class PhysicalProduct(Product):
    def __init__(self, product_id, name, price, quantity_available, weight):
        super().__init__(product_id, name, price, quantity_available)
        self._weight = weight

    def display_details(self):
        return (
            f"[Physical] ID: {self.product_id}, Name: {self.name}, Price: ${self.price}, "
            f"Stock: {self.quantity_available}, Weight: {self._weight}kg"
        )

# Subclass for digital products with download link
class DigitalProduct(Product):
    def __init__(self, product_id, name, price, quantity_available, download_link):
        super().__init__(product_id, name, price, quantity_available)
        self._download_link = download_link

    def display_details(self):
        return f"[Digital] ID: {self.product_id}, Name: {self.name}, Price: ${self.price}, Download Link: {self._download_link}"
```

```python
# ==================== Cart Item Class ====================
class CartItem:
    def __init__(self, product, quantity):
        self._product = product
        self._quantity = quantity

    @property
    def product(self): return self._product

    @property
    def quantity(self): return self._quantity

    @quantity.setter
    def quantity(self, value):
        if value >= 0:
            self._quantity = value

    # Calculate total price for this cart item
    def calculate_subtotal(self):
        return self.product.price * self.quantity

    def __str__(self):
        return (
            f"Item: {self.product.name}, Quantity: {self.quantity}, "
            f"Price: ${self.product.price}, Subtotal: ${self.calculate_subtotal():.2f}"
        )
```

```python
# ==================== Shopping Cart Class ====================
class ShoppingCart:
    TAX_RATE = 0.08  # 8% tax rate

    def __init__(self):
        self._items = {}  # key: product_id, value: CartItem
        self.catalog = self._create_sample_catalog()

    # Hardcoded catalog with 10 products
    def _create_sample_catalog(self):
        return {
            "P001": PhysicalProduct("P001", "Laptop", 999.99, 10, 2.5),
            "P002": PhysicalProduct("P002", "Smartphone", 499.99, 20, 0.3),
            "P003": PhysicalProduct("P003", "Headphones", 89.99, 15, 0.2),
            "P004": PhysicalProduct("P004", "Keyboard", 49.99, 30, 0.8),
            "P005": PhysicalProduct("P005", "Monitor", 179.99, 12, 4.0),
            "D001": DigitalProduct("D001", "Antivirus Software", 29.99, 100, "https://download.com/antivirus"),
            "D002": DigitalProduct("D002", "Photo Editor", 59.99, 100, "https://download.com/photoeditor"),
            "D003": DigitalProduct("D003", "Music Album", 9.99, 200, "https://download.com/music"),
            "D004": DigitalProduct("D004", "E-book", 14.99, 150, "https://download.com/ebook"),
            "D005": DigitalProduct("D005", "Online Course", 199.99, 50, "https://download.com/course")
        }
```

```python
    # Add a product to cart
    def add_item(self, product_id, quantity):
        product = self.catalog.get(product_id)
        if not product:
            print(f"❌ Product ID '{product_id}' not found.")
            return False
        if quantity <= 0:
            print("❌ Quantity must be greater than 0.")
            return False
        if not product.decrease_quantity(quantity):
            print(f"❌ Not enough stock. Available: {product.quantity_available}")
            return False

        if product_id in self._items:
            self._items[product_id].quantity += quantity
        else:
            self._items[product_id] = CartItem(product, quantity)
            print(f"✅ {quantity} x '{product.name}' added to cart.")

        return True

    # Remove a product from the cart
    def remove_item(self, product_id):
        if product_id in self._items:
            product = self._items[product_id].product
            product.increase_quantity(self._items[product_id].quantity)
            del self._items[product_id]
            print(f"🗑️ Removed '{product.name}' from cart.")
            return True
        return False

```

```python
    # Change quantity of a product in the cart
    def update_quantity(self, product_id, new_quantity):
        if product_id in self._items:
            item = self._items[product_id]
            product = item.product
            diff = new_quantity - item.quantity
            if diff > 0:
                if not product.decrease_quantity(diff):
                    print("❌ Not enough stock.")
                    return False
            else:
                product.increase_quantity(-diff)
            item.quantity = new_quantity
            print(f"🔁 Updated '{product.name}' to quantity {new_quantity}.")
            return True
        print("❌ Item not found in cart.")
        return False

    # Cart totals
    def get_total(self):
        return sum(item.calculate_subtotal() for item in self._items.values())

    def get_tax(self):
        return self.get_total() * self.TAX_RATE

    def get_grand_total(self):
        return self.get_total() + self.get_tax()

    # Empty all cart items
    def empty_cart(self):
        for item in list(self._items.values()):
            item.product.increase_quantity(item.quantity)
        self._items.clear()
        print("🧹 Cart emptied.")

    # Display contents of cart
```

```python
    # Display contents of cart
    def display_cart(self):
        print("\n🛒 --- Shopping Cart ---")
        if not self._items:
            print("Cart is empty.")
        for item in self._items.values():
            print(item)
        print(f"Subtotal: ${self.get_total():.2f}")
        print(f"Tax (8%): ${self.get_tax():.2f}")
        print(f"Total: ${self.get_grand_total():.2f}\n")

```

```python
    # Print available products by type
    def display_products(self):
        print("\n📦 --- Physical Products ---")
        for product in self.catalog.values():
            if isinstance(product, PhysicalProduct):
                print(product.display_details())
        print("\n💻 --- Digital Products ---")
        for product in self.catalog.values():
            if isinstance(product, DigitalProduct):
                print(product.display_details())
        print("")

    # Search for products
    def search_products(self, keyword):
        print(f"\n🔍 Search Results for '{keyword}':")
        found = False
        for product in self.catalog.values():
            if keyword.lower() in product.name.lower():
                print(product.display_details())
                found = True
        if not found:
            print("No products found.")

    # Print final checkout summary
```


```python
    # Print final checkout summary
```


```python
    def checkout(self):
        print("\n🧾 --- Checkout Summary ---")
        if not self._items:
            print("Cart is empty. Nothing to checkout.")
            return
        for item in self._items.values():
            print(item)
        print(f"Subtotal: ${self.get_total():.2f}")
        print(f"Tax: ${self.get_tax():.2f}")
        print(f"Grand Total: ${self.get_grand_total():.2f}")
        print("✅ Thank you for your purchase!")
        self._items.clear()

# ==================== Console Interface ====================
```

```python
# ==================== Console Interface ====================
def main():
    cart = ShoppingCart()
    while True:
        print("""
1. View All Products
2. Search Product by Name
3. Add to Cart
4. View Cart
5. Update Quantity
6. Remove Item
7. Empty Cart
8. Checkout
9. Exit
        """)
        choice = input("Select an option: ")
        if choice == '1':
            cart.display_products()
        elif choice == '2':
            keyword = input("Enter keyword to search: ")
            cart.search_products(keyword)
        elif choice == '3':
            pid = input("Enter product ID: ")
            try:
                qty = int(input("Enter quantity: "))
                if not cart.add_item(pid, qty):
                    print("❌ Failed to add item.")
            except ValueError:
                print("❌ Invalid quantity.")
        elif choice == '4':
            cart.display_cart()
        elif choice == '5':
            pid = input("Enter product ID to update: ")
            try:
                qty = int(input("Enter new quantity: "))
                if not cart.update_quantity(pid, qty):
                    print("❌ Update failed.")
            except ValueError:
```

```python
            except ValueError:
                print("❌ Invalid quantity.")
        elif choice == '6':
            pid = input("Enter product ID to remove: ")
            if not cart.remove_item(pid):
                print("❌ Product not found in cart.")
        elif choice == '7':
            cart.empty_cart()
        elif choice == '8':
            cart.checkout()
        elif choice == '9':
            print("👋 Thank you! Exiting...")
            break
        else:
            print("❌ Invalid choice.")

if __name__ == '__main__':
    main()
```

5. OUTPUT:

1. Options

```
1. View All Products
2. Search Product by Name
3. Add to Cart
4. View Cart
5. Update Quantity
6. Remove Item
7. Empty Cart
8. Checkout
9. Exit
```

2. 1. View All Products

```
Select an option: 1

📦 --- Physical Products ---
[Physical] ID: P001, Name: Laptop, Price: $999.99, Stock: 4, Weight: 2.5kg
[Physical] ID: P002, Name: Smartphone, Price: $499.99, Stock: 20, Weight: 0.3kg
[Physical] ID: P003, Name: Headphones, Price: $89.99, Stock: 15, Weight: 0.2kg
[Physical] ID: P004, Name: Keyboard, Price: $49.99, Stock: 30, Weight: 0.8kg
[Physical] ID: P005, Name: Monitor, Price: $179.99, Stock: 12, Weight: 4.0kg

💻 --- Digital Products ---
[Digital] ID: D001, Name: Antivirus Software, Price: $29.99, Download Link: https://download.com/antivirus
[Digital] ID: D002, Name: Photo Editor, Price: $59.99, Download Link: https://download.com/photoeditor
[Digital] ID: D003, Name: Music Album, Price: $9.99, Download Link: https://download.com/music
[Digital] ID: D004, Name: E-book, Price: $14.99, Download Link: https://download.com/ebook
[Digital] ID: D005, Name: Online Course, Price: $199.99, Download Link: https://download.com/course
```

2. Search Product by Name

```
Select an option: 2
Enter keyword to search: Smartphone

🔍 Search Results for 'Smartphone':
[Physical] ID: P002, Name: Smartphone, Price: $499.99, Stock: 20, Weight: 0.3kg
```

3. Add to Cart

```
Select an option: 3
Enter product ID: P001
Enter quantity: 1
✅ 1 x 'Laptop' added to cart.
```

4. View Cart

```
Select an option: 4

🛒 --- Shopping Cart ---
Item: Laptop, Quantity: 1, Price: $999.99, Subtotal: $999.99
Subtotal: $999.99
Tax (8%): $80.00
Total: $1079.99
```

5. Update Quantity

```
Select an option: 5
Enter product ID to update: P001
Enter new quantity: 2
🔁 Updated 'Laptop' to quantity 2.
```

6. Remove Item

```
Select an option: 6
Enter product ID to remove: P001
🗑️Removed 'Laptop' from cart.
```

7. Empty Cart

```
Select an option: 7
🧹 Cart emptied.
```

8. Checkout

```
Select an option: 8

🧾 --- Checkout Summary ---
Item: Smartphone, Quantity: 2, Price: $499.99, Subtotal: $999.98
Subtotal: $999.98
Tax: $80.00
Grand Total: $1079.98
✅ Thank you for your purchase!
```

9. Exit

```
Select an option: 9
👋 Thank you! Exiting...
PS C:\Users\kesha\.vscode>
```

3. Failed Condition

1. **If Product is not in the list**

```
Select an option: 2
Enter keyword to search: iPhone

🔍 Search Results for 'iPhone':
No products found.
```

2. **If Invalid Product ID**

```
Select an option: 3
Enter product ID: P008
Enter quantity: 2
❌ Product ID 'P008' not found.
❌ Failed to add item.
```

3. **If not enough stock**

```
Select an option: 3
Enter product ID: D001
Enter quantity: 20000000
❌ Not enough stock. Available: 98
❌ Failed to add item.
```

4. **If item not found in cart**

```
Select an option: 5
Enter product ID to update: D001
Enter new quantity: 2
❌ Item not found in cart.
❌ Update failed.
```

5. **If searched outside the list**

```
Select an option: 10
❌ Invalid choice.
```

6. Results and Discussion

Findings

The application allows users to:

- Browse products (with type-specific information like weight or download link)
- Search products by name
- Add products to the cart after validating available stock
- Update quantities or remove items from the cart
- View cart contents with a clear cost breakdown
- Simulate a checkout process with a thank-you message

Analysis

Using inheritance allowed the system to handle two types of products without duplicating code. For example, both PhysicalProduct and DigitalProduct inherit from Product, but provide unique display_details() behavior through method overriding.

Encapsulation was crucial to protect attributes like price and stock quantity from being directly altered. This ensured that all changes to product state happened through controlled methods, preserving data integrity.

Object composition made the cart flexible — each cart item references a product object and handles its own quantity and pricing logic. This modular design allows easy future expansion, such as supporting promotions or user accounts.

Implications

This project shows how OOP principles can power robust, clear, and extendable systems. It also provides a strong foundation for scaling the project further — such as adding GUI, user login, or payment simulation — without rewriting core logic.

7. Conclusion

This project successfully simulated a text-based online shopping experience using Python and object-oriented principles. Through carefully structured classes, the system achieves functional requirements like adding to cart, modifying cart contents, and viewing totals. The console interface allowed real-time interaction and testing of the application logic.

The project not only meets its educational goals of demonstrating OOP in practice but also lays a scalable groundwork for future enhancements. The structure and methodology used reflect industry-standard practices for modular and maintainable code.

8. References

- Python Official Documentation – https://docs.python.org
- GITHUB – https://github.com/
- E-Commerce Cart Design Patterns – Various academic and developer resources

9. Project(CODE) GITHUB link

https://github.com/Keshav-Krishan/Online-Shopping-Cart/blob/main/Shopping_cart.py