UGANDA CHRISTIAN UNIVERSITY

A Centre of Excellence in the Heart of Africa

FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY

DEPARTMENT OF COMPUTING AND TECHNOLOGY

ADVENT 2024 SEMESTER OOP COURSEWORK PROJECT REPORT

PROGRAM: BSCS, BSDS 2:1

COURSE: Object-Oriented Programming

COURSE LECTURER: Ian Raymond Osolo

PROJECT TITLE: **Online Shopping Cart System Using OOP**

Submitted by

S/N	Reg Number	Name	Signature
1.	S23B23/051	OKIDI NORBERT	
2.	S23B23/074	BUWEMBO DAVID DENZEL	
3.	S23B23/027	KIISA ANGELA	
4.	S23B23/046	NZIRIGA ISAAC NICKSON	
5.	S23B23/029	KISUZE GARETH NEVILLE	
6.	S23B23/041	NAJJUMA TEOPISTA	

Date Submitted: ……

1.0 Abstract *(Half a page)*

This project report presents the design and implementation of an **Online Shopping Cart System** built using **Object-Oriented Programming (OOP)** principles in Python. The system is designed to simulate a basic e-commerce experience, allowing customers to browse products, add items to their shopping cart, create orders, and process payments. By leveraging OOP principles such as **Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism**, the system is organized into modular and reusable components, enhancing both functionality and maintainability.

The system includes five primary classes—**Product**, **Customer**, **Cart**, **Order**, and **Payment**—each representing a core component of the shopping experience. Through encapsulation, each class maintains control over its attributes, with methods provided for secure access and manipulation of data. Complex processes, such as calculating totals in the cart and verifying payments, are abstracted within each class, providing a simplified interface for users while hiding intricate backend operations. Polymorphism is used within the **Payment** class hierarchy, where different payment methods (`CreditCardPayment` and `PayPalPayment`) inherit from the base **Payment** class, each implementing the `process_payment` method differently to support multiple payment types.

This modular design approach results in a scalable and secure system that could be expanded with additional features, such as discount handling, multiple shipping methods, and enhanced payment processing. Encapsulation secures data, while abstraction simplifies complex functions for ease of use. Inheritance reduces code redundancy by enabling shared functionality, and polymorphism adds flexibility, allowing the system to handle multiple types of payments seamlessly.

The project demonstrates how OOP principles can structure a real-world application, dividing complex processes into manageable parts, improving readability, and ensuring future extensibility. This system serves as a model for small-scale e-commerce applications and provides a strong foundation for developers to build more advanced and secure online shopping solutions.

2.0 Introduction, problem statement, and project objectives *(1-2 Pages)*

Introduction:
The Online Shopping Cart System project presented here addresses these requirements by using **Object-Oriented Programming (OOP)** principles in Python to create a system that organizes and manages product selection, order processing, and payment handling.

An online shopping cart is a user's primary tool to interact with the e-commerce platform, allowing them to add products, manage quantities, view prices, and proceed to checkout. However, building a fully functional shopping cart system is a complex task, involving multiple interconnected processes, such as managing inventory, securing user data, calculating totals, and processing payments. In traditional programming, managing this complexity often results in large, monolithic codebases, which are difficult to understand, maintain, and extend. By applying OOP principles, however, these tasks can be broken down into manageable units, making the system more modular, scalable, and maintainable.

The primary focus of this project is to develop a **modular and extensible shopping cart system** using Python's OOP features. In this context, OOP principles like **Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism** enable us to design a system where each component performs a specific, isolated function and interacts with other components through well-defined interfaces. This approach not only promotes reusability but also enhances security, readability, and future scalability.

This project encompasses five core classes—**Product**, **Customer**, **Cart**, **Order**, and **Payment**—each representing a different aspect of the e-commerce experience. These classes encapsulate specific data and behaviors, allowing them to interact with each other while keeping internal details secure. For example, the `Product` class manages individual product details, including `product_id`, `name`, `price`, and `stock`, while the `Cart` class handles the operations of adding items, calculating totals, and displaying cart content. By organizing these tasks into separate classes, we can clearly define each component's role and ensure that changes or extensions to one component do not disrupt the entire system.

Problem Statement:
An online shopping cart system has unique requirements for efficient data handling and secure transaction processing. The complexity of managing different entities, such as products, customers, carts, and payments, makes it challenging to build a coherent, error-free system. Without a well-defined structure, this complexity can lead to inefficient, hard-to-maintain code that is prone to errors and difficult to expand. For instance, managing inventory across different orders and payments can be prone to synchronization issues if not handled carefully. Additionally, providing an easy-to-navigate shopping experience requires that complex tasks, like total calculation and payment processing, are presented to users in a simplified form.

The aim of this project is to address these challenges by using OOP to break down these components into specific classes, each responsible for distinct tasks within the system. By leveraging OOP principles, this project demonstrates how software design can benefit from structured programming, making it easier to handle multiple aspects of an online shopping cart system in a modular and organized way.

Project Objectives:
The main objectives of this project are to build a Python-based Online Shopping Cart System that uses OOP principles to manage product selection, customer information, shopping cart functionality, order processing, and payments in a cohesive, extensible framework. Key objectives include:

1. **Efficient Product Management**: Create a `Product` class to handle essential product details, such as `name`, `price`, and `stock`. This class ensures that all products can be consistently managed and easily accessed for inventory purposes.
2. **Customer Information Handling**: A `Customer` which holds common attributes like `name`, `email`, and `address`. This design ensures that customer data is securely handled and can be easily extended for different user types (e.g., admins or sellers) if needed in the future.
3. **Shopping Cart Functionality**: Develop a `Cart` class that encapsulates all operations related to a customer's shopping cart. This class enables adding/removing products, calculating the total cost, and viewing cart details in an organized manner. Abstraction is applied to simplify cart management, so users can interact with their cart without dealing with internal calculations.
4. **Order Processing**: An `Order` class allows customers to place an order based on the contents of their cart. Each order is assigned a unique `order_id`, and the total is calculated based on the cart items. This modular approach makes order management straightforward and provides the flexibility to integrate additional order features in the future.
5. **Payment Processing with Multiple Payment Methods**: The `Payment` class acts as a base class, which is inherited by `CreditCardPayment` and `PayPalPayment` subclasses. These subclasses implement the `process_payment` method uniquely, supporting different payment types. This setup demonstrates polymorphism by allowing the system to handle multiple payment methods through a single interface.

Significance of Object-Oriented Programming Principles in the Project:
Using OOP principles significantly enhances the system's modularity, reusability, and scalability. Each principle contributes uniquely to achieving these objectives:

- **Encapsulation**: Encapsulation is applied across all classes to control data access. By marking attributes as private, such as `_name` and `_price` in `Product`, each class hides its internal data from outside interference, only allowing access through dedicated methods. This approach secures sensitive data, reducing the risk of accidental modification and ensuring that data remains consistent across different parts of the system.
- **Abstraction**: The system's complexity is simplified through abstraction, where only essential operations are exposed to users while complex details are hidden. For example, the `process_payment` method in each `Payment` subclass abstracts the intricacies of payment processing, allowing users to initiate payments without knowing the underlying details.
- **Inheritance**: Inheritance promotes code reuse by creating base classes with shared attributes and methods. For instance, `CreditCardPayment` and `PayPalPayment` inherit from `Payment`, allowing shared functionality and enabling unique behaviors for each payment type.
- **Polymorphism**: The system uses polymorphism to handle different payment methods seamlessly. By implementing the `process_payment` method differently in `CreditCardPayment` and `PayPalPayment`, polymorphism enables the `process_payment` method to execute different actions based on the payment type. This approach enhances flexibility, making it easy to add or modify payment types without changing the core system.

3.0 Methods, tools, and designs used for the project *(1-2 Pages)*

Methods:
This project utilizes **Object-Oriented Programming (OOP)** principles to provide a structured design for each component of the shopping cart system:

1. **Encapsulation**: Attributes are marked as private using underscores (e.g., `_product_id`, `_price`), restricting direct access to the data. Access is controlled through methods and properties, ensuring that data is only accessed or modified through specific interfaces.
2. **Abstraction**: Each class offers a simplified interface for complex operations. For example, the `Cart` class abstracts away the complexities of managing items, while the `process_payment` method in the `Payment` subclasses hides details of payment verification.
3. **Inheritance**: The `Payment` class is inherited by `CreditCardPayment` and `PayPalPayment`, which allows these subclasses to share and extend the `process_payment` functionality.
4. **Polymorphism**: The system uses polymorphism through the `Payment` class, allowing different payment types (`CreditCardPayment` and `PayPalPayment`) to be processed in a unified way via the `process_payment` method. Each subclass implements `process_payment` differently, providing customized handling for different payment types.

Tools and Environment:

- **Python 3.1--above**: Used for writing and executing code.
- **IDE (VS Code)**: For developing, testing, and debugging the code.

Design:
The system is divided into specific classes:

- **Product** handles product-related data and methods.
- **Customer** inherits from **Person**, managing customer-specific information.
- **Product Catalog** for displaying products in catalog
- **Cart** is responsible for handling shopping cart operations.
- **Order** generates unique orders and calculates totals.
- **Payment** and its subclasses handle payment processing and success/failure messages.

4.0 Results *(2-5 Pages)*

(Description of the project developed, justifying how it solves the problem and achieves the set objectives.)

Project Overview

The Online Shopping Cart System is designed to simulate a typical e-commerce experience, allowing customers to:

- Add products to a shopping cart,
- View cart details and item totals,
- View product catalog,
- Place orders, and
- Process payments.

Problem-Solving Approach

The system was designed to solve the core problems associated with managing online shopping transactions by organizing each part of the shopping experience into distinct, encapsulated classes. Each class has its own set of responsibilities, attributes, and methods, allowing for controlled data handling and secure, efficient operations. Below is a breakdown of how each component contributes to achieving the project objectives and how OOP principles were applied to meet the requirements of a typical e-commerce system.

Components and Their Contributions

1. **Product Class**
    - **Objective**: To manage product details, such as product ID, name, price, and stock.
    - **Functionality**: The `Product` class encapsulates all product-related data and provides methods to retrieve information and update stock levels. Private attributes (`_product_id`, `_name`, `_price`, and `_stock`) are used to control access, ensuring that product data remains secure and consistent.
    - **Result**: This class meets the objective of providing secure and efficient product management. By exposing only essential methods (e.g., `get_product_info`, `update_stock`), it prevents unintended modifications to product data, which is crucial for maintaining accurate inventory.
2. **Customer Class**
    - **Objective**: To handle customer-specific details securely and efficiently.
    - **Functionality**: The `Customer` class provides common attributes (`_name`, `_email`, `_address`).
    - **Result**: This setup achieves the goal of secure customer data management. Encapsulation is applied to protect customer information.
3. **Cart Class**
    - **Objective**: To manage shopping cart operations, including adding products, removing products, calculating the total cost and display the checkout summary,
    - **Functionality**: The `Cart` class provides methods for adding and removing products, viewing cart contents, calculating the total amount and display the checkout summary. This class hides the complexity of cart management from the user by abstracting calculations and product handling through simple methods.
    - **Result**: This class meets its objective by providing a cohesive and organized way to handle the shopping cart. Through **Abstraction**, the cart operations are simplified for the user, allowing them to focus on product selection without dealing with backend calculations or inventory checks. This approach ensures a smooth user experience and maintains data integrity.
    -

4. **Order Class**
    - **Objective**: To process orders, assign unique IDs, and manage order totals.
    - **Functionality**: The `Order` class encapsulates order-related attributes and methods. Each order is assigned a unique `order_id`, generated through a static variable `order_count`. The `place_order` method calculates the total amount based on the contents of the cart.
    - **Result**: This class meets its goal of streamlined order processing. By encapsulating order details, it protects the integrity of order information while ensuring a consistent process for each transaction. The use of static variables to generate unique IDs also simplifies tracking and improves reliability in order management.
5. **Payment Class (and Subclasses)**
    - **Objective**: To handle multiple types of payments, each with a unique process for verification.
    - **Functionality**: The `Payment` class is designed as a base class, which is then extended by `CreditCardPayment` and `PayPalPayment` subclasses. These subclasses each implement the `process_payment` method, demonstrating **Polymorphism**. This design allows the system to handle different payment methods while maintaining a consistent interface.
    - **Result**: The Payment class and its subclasses achieve the objective of providing flexible payment processing. Polymorphism allows different payment methods to be processed using the same interface, which simplifies integration and makes it easier to add or modify payment types in the future.

Achievement of Project Objectives

The project successfully meets the objectives set out at the beginning, including efficient product and customer management, intuitive shopping cart operations, organized order processing, and flexible payment handling. Here's how each objective is achieved:

1. **Efficient Product Management**:
   By encapsulating product attributes and controlling access to stock levels, the `Product` class provides an organized approach to managing product information. The methods within the class allow secure access and modification of product data, which supports consistent and accurate inventory management.
2. **Shopping Cart Functionality**:
   The `Cart` class effectively abstracts cart operations, allowing customers to add, view, and calculate totals without dealing with backend complexities. This class structure provides a clean interface for cart management, making the shopping experience user-friendly and efficient. By handling totals and managing stock updates, the Cart class ensures that both the user and the system have a consistent view of the cart contents.
3. **Order Processing with Unique Identifiers**:
   The `Order` class assigns unique IDs to each order and calculates the total based on cart contents. By encapsulating order details, the class secures transaction data and maintains consistency. The use of unique order IDs and static variables ensures that each transaction is accurately tracked, improving reliability for both the user and system administrators.
4. **Flexible and Secure Payment Processing**:
   Through the `Payment` class and its subclasses (`CreditCardPayment` and `PayPalPayment`), the system provides a polymorphic payment structure that can handle different payment types with the same interface.

Justification of Problem Solving

The Online Shopping Cart System solves the core challenges of organizing and managing e-commerce transactions by implementing OOP principles effectively:

- **Encapsulation** is used to protect sensitive data, such as product prices and customer information, ensuring they are only accessed through controlled methods. This approach maintains data integrity and supports consistent handling across the system.
- **Abstraction** hides the complexities of backend operations, making it easy for users to interact with their shopping cart, view totals, and complete orders without needing to understand the underlying calculations.
- **Inheritance** promotes reusability, reducing redundancy by allowing shared attributes and methods across related classes, such as `Customer` and `Person`, and `Payment` and its subclasses.
- **Polymorphism** adds flexibility to the payment system, allowing multiple types of payments to be processed using a single interface, thus enhancing the extensibility of the codebase.

Overall Impact and Benefits

This structured, OOP-driven approach results in a modular, organized, and extensible system, providing a foundation that could be easily expanded with additional features such as discounts, various shipping options, and more payment methods. The principles applied in this project make the system maintainable, scalable, and adaptable to future requirements, ensuring that it can grow with the evolving needs of a modern e-commerce platform. By separating responsibilities into specific, isolated classes, the system also improves collaboration among developers, allowing individual components to be developed and maintained independently.

5.0 Conclusion & Recommendation (1 Page)

Conclusion:
The **Online Shopping Cart System** developed using Python and Object-Oriented Programming (OOP) principles serves as a functional and flexible e-commerce solution that simulates the core processes of an online shopping experience. By leveraging the four fundamental principles of OOP—**Encapsulation**, **Abstraction**, **Inheritance**, and **Polymorphism**—the project achieves a modular and organized design that enhances the system's maintainability, scalability, and extensibility. Each class encapsulates specific data and behavior, ensuring secure data handling and controlled access to sensitive information, such as product details and customer information.

The project's structure demonstrates how a complex set of operations can be broken down into independent, reusable components, making it easier to manage and expand. Encapsulation protects data integrity within each class, while abstraction simplifies interactions for the user by hiding backend complexities, such as order calculations and payment processing. Inheritance is used to streamline shared functionality across related classes, reducing redundancy, and ensuring code reusability. Finally, polymorphism in the `Payment` subclasses allows for the seamless integration of multiple payment methods, a feature crucial for real-world applications where flexibility and customer choice are essential.

Recommendations for Future Development:
While the current system meets the fundamental requirements of a shopping cart application, there are several enhancements that could be incorporated to improve functionality and provide a more realistic e-commerce experience:

1. **Discount and Promotion Management**:
   Adding a system for handling discounts, coupons, and promotional codes would enhance the customer experience and incentivize purchases. A `Discount` class could be introduced to apply percentage-based or fixed-amount discounts at checkout, providing users with more options and improving sales potential.
2. **Database Integration**:
   Storing data in a persistent database would enable the system to handle a larger volume of products, customers, and orders. By integrating a relational database like MySQL or PostgreSQL, the system could store customer profiles, past orders, product inventories, and payment histories securely. This addition would make the system scalable and suitable for real-world applications that require consistent and reliable data storage.
3. **Enhanced Payment Processing**:
   Currently, the system supports basic payment processing via `CreditCardPayment` and `PayPalPayment`. Expanding this feature to include additional payment gateways, such as digital wallets, bank transfers, or cryptocurrency, would cater to a wider audience and improve user convenience. Incorporating secure protocols (e.g., encryption for sensitive data) would also strengthen transaction security.
4. **User Authentication and Role Management**:
   Implementing a user authentication system with different roles (e.g., customers, admins, sellers) would add a layer of security and personalization. An admin could manage product inventory, monitor sales, and generate reports, while a customer could track their order history and save items to a wishlist. By creating a role-based access control (RBAC) system, the shopping cart could support multi-user functionality and cater to diverse use cases.
5. **Order Tracking and Notifications**:
   Adding a feature for order tracking and notifications would improve customer engagement and transparency. After an order is placed, the system could notify the user via email or SMS about the order status (e.g., confirmed, shipped, delivered). This functionality would enhance the user experience and build trust in the platform by keeping customers informed.

6. **Product Categorization and Search Functionality**:
   As the product catalog grows, categorizing products and adding a search feature would make navigation easier for customers. A `Category` class could be introduced to group products into relevant categories (e.g., Electronics, Clothing, Home Goods). Implementing a search function with filters (price range, ratings, etc.) would also improve accessibility and help customers find products quickly.
7. **Analytics and Reporting Features**:
   Adding analytics and reporting capabilities would be beneficial for business insights and decision-making. The system could generate reports on sales trends, customer preferences, and inventory levels. A `Report` class could handle these functions, offering daily, weekly, or monthly summaries to help administrators optimize operations.

6.0 References (if Any)

NONE

7.0 Appendices *(if any, including screenshots, codes, etc)*

Appendix A: Key Code Snippets

The following code snippets illustrate the core classes and methods that form the basis of the Online Shopping Cart System, with a focus on how each class applies specific OOP principles:

1. **Product Class with Encapsulation**

   The `Product` class encapsulates product-related attributes and methods, ensuring data security and controlled access to the attributes. Only specific methods can access or modify attributes like `_price` and `_stock`, which are set as private to prevent accidental modification.

```python
# Product Class
class Product:
    """Represents a product with ID, name, price, and stock availability."""

    def __init__(self, product_id, name, price, stock):
        # Encapsulated product attributes
        self._product_id = product_id
        self._name = name
        self._price = price
        self._stock = stock

    def get_product_info(self):
        """Returns a string with product information."""
        return f"Product: {self._name}, Price: {self._price}, Stock: {self._stock}"

    def update_stock(self, quantity):
        """Updates the stock by reducing it by the quantity sold."""
        if quantity <= self._stock:
            self._stock -= quantity
        else:
            print("Not enough stock available")

    @property
    def price(self):
        """Returns the price of the product."""
        return self._price

    @property
    def name(self):
        """Returns the name of the product."""
        return self._name

```

2. **Customer Class Demonstrating Inheritance**

```python
# Customer Class
class Customer:
    """Represents a customer with ID, name, email, and address."""

    def __init__(self, customer_id, name, email, address):
        # Encapsulated customer attributes
        self._customer_id = customer_id
        self._name = name
        self._email = email
        self._address = address

    def get_customer_info(self):
        """Returns a string with customer information."""
        return f"Customer ID: {self._customer_id}, Name: {self._name}, Email: {self._email}, Address: {self._address}"
```

3. **Cart Class Applying Abstraction**

The `Cart` class abstracts complex cart operations, such as adding products and calculating totals, providing users with a straightforward interface to manage their cart.

```python
# Cart Class
class Cart:
    """Represents a shopping cart containing multiple products and their quantities."""

    def __init__(self, customer):
        # Encapsulated attributes for customer and cart items
        self._customer = customer
        self._items = {}

    def add_product(self, product, quantity):
        """Adds a product to the cart with a specified quantity."""
        if product._product_id in self._items:
            self._items[product._product_id]['quantity'] += quantity
        else:
            self._items[product._product_id] = {'product': product, 'quantity': quantity}
        product.update_stock(quantity)

    def remove_product(self, product_id):
        """Removes a product from the cart by product ID."""
        if product_id in self._items:
            del self._items[product_id]
```

Appendix B: Sample System Output

The following output examples provide a glimpse into the functionality and user interaction within the system.

1. **Cart View and Total Calculation**

```
********** CART DETAILS **********
Laptop - Quantity: 2 - Price: 1000
Smartphone - Quantity: 3 - Price: 500
```

2. Order Placement

```
***** ORDER DETAILS *****
Order ID: 1
Customer ID: 1, Name: Mustafa, Email: mustafa@gmail.com, Address: Buguju
Total Amount: 3500
Payment Status: Success, Amount: 3500
```

3. **Payment Processing Output (Success and Failure)**

Successful Payment:

```
Payment Status: Success, Amount: 3500
***** THANK YOU FOR SHOPPING WITH US *****
```

Failed Payment:

```
Payment Status: Failed, Amount: 3000
PayPal payment failed. Insufficient funds.
```