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: Food Delivery System

Submitted by

S/N	Reg Number	Name
1.	M23B38/005	Emmanuel Nsubuga
2.	M23B38/017	Ronald Tusiime
3.	M23B38/018	Agaba Itungo
4.	M23B23/001	Joy Abaho
5.	M23B23/047	M Rachel Isooba

Date Submitted:

13th November 2024

1.0 Abstract *(Half a page)*

This project aims to develop a food delivery system using Python, applying Object-Oriented Programming (OOP) principles. The system provides a user-friendly interface for restaurants, customers, and delivery personnel to manage food orders efficiently. Key functionalities include:

- **Menu Management:** Allowing restaurants to create, and display menu items with prices.
- **Order Placement and Tracking:** Enabling customers to browse restaurant menus, place orders, and track order status in real time.
- **Delivery Management:** Assigning orders to delivery personnel and tracking delivery progress.
- **Payment Processing:** Simulating payment transactions for orders, supporting multiple payment methods.

The project demonstrates the practical application of OOP principles, including inheritance, encapsulation, polymorphism, and abstraction, to develop a scalable and maintainable food delivery system.

2.0 Introduction, problem statement, and project objectives

2.1 Introduction

Food is a key part of every UCU (Uganda Christian University) student's life. Whether it be breakfast, lunch, or dinner, students flock to the various canteens and restaurants for food. Many of us understand the stress a day from 8 a.m. to 6 p.m. brings, with coursework and group discussions; often, there is neither time nor energy left to grab a bite. We have acknowledged the rapid growth of technological advancements in the food delivery industry and believe UCU shouldn't be an exception. Currently, UCU students use the 60 Second Delivery service, which is, for the most part, unreliable, costly, and inconsistent. Many students either fail to reach the delivery service, receive the wrong order, or don't receive their order at all, often after spending money on airtime just to contact them. Having seen this problem, we find it necessary to develop a project that aims to create a comprehensive food delivery system leveraging OOP principles to provide a flexible, efficient, and user-friendly platform. Our goal is to simplify food ordering from the comfort of a student's room, enabling them to order from anywhere, at any time, with minimal cost.

2.2 Problem Statement

Uganda Christian University (UCU) students lack access to a reliable, affordable, and user-friendly food delivery service that accommodates their busy schedules, that leads to frequent missed meals, frustration, and inefficient service usage.

2.3 Project Objectives

This project aims to address these challenges by developing a food delivery system with the following objectives:

1. **Build a Reliable Food Delivery Platform:** Create a dependable food delivery system that actually works for UCU students, providing consistent service that meets delivery expectations.

2. **Maximize User Convenience**: Allow students to order food from their rooms at any time, with a simple, user-friendly interface.

3. **Boost Service Accuracy:** Develop a system that minimizes wrong or missed orders, so students get exactly what they order, on time.

4. **Cut Airtime Costs for Ordering:** Provide an ordering system that doesn't rely on airtime, helping students save money on delivery calls.

5. **Payment Variety:** Support multiple payment methods, including credit card and mobile payment and simulates payment processing for a seamless transaction experience.

6. **Automate Order Placement and Tracking**: Develop an interface for customers to browse restaurant menus, place orders, and view order status.

7. **Use Object-Oriented Programming (OOP) Principles:** Apply OOP concepts to make the system flexible and modular, allowing easy maintenance and future updates based on what users need.

By achieving these objectives, this project will contribute to the development of a robust and efficient food delivery system that benefits restaurants, delivery personnel, and customers alike.

3.0 Methods, tools, and designs used for the project

3.1 Methods

The food delivery system was developed using Object-Oriented Programming (OOP), which supports a modular and reusable approach to software development. Key OOP concepts used are:

- **Classes and Objects**: We had the following classes: Restaurant, Customer, Order, DeliveryPerson, and Payment which were used to create instances for restaurants, customers, orders, delivery personnel and payments. These objects interact to manage different functionalities.
- **Encapsulation**: Sensitive data, such as customer payment details and order details, customer details, restaurant details and delivery details were encapsulated within classes to ensure security and data integrity. The use of setters and getters were used to access this protected information outside their individual classes
- **Abstraction**: Abstraction was used on the class Members, specifically on the method display, to hide its complexity. The concept was also used in other areas where data was exchanged between classes while still hide their complexity from each other. The application of modular code is another example of the use of abstraction in our code.
- **Inheritance**: The system uses inheritance to create different types of payment methods (e.g., CreditCardPayment, MobileMoneyPayment), which inherit from a base CashPayment class as well as different types of user classes (Restaurant, Customer, DeliveryPerson) that inherit from the abstract class Members
- **Polymorphism**: The system leverages polymorphism to handle various payment types, allowing for flexible code that can accommodate different payment methods. The type of polymorphism used is method overriding.

3.2 Tools

- **Programming Language**: Python
- **IDE and Development Tools**: VS Code
- **Libraries**: Datetime, abc and random
- **Others**: Canva, Whimsical, google docs

3.3 Design

This project uses a combination of procedural, functional and object-oriented programming to achieve this Food delivering system. This projected is composed of four python files. They are as follows:

- **Attempt1.py** : Contains all the object oriented code with a variety of classes and methods. This is the main structure of the application
- **Atempt2.py**: Contains mostly function based code. This file is the basic implementation of the Object-oriented code in Attempt 1. This helps provide a level of abstraction from the complexity of the code in Attempt1.py
- **Attempt3.py**: Contains function-based cone. This file implements basic information retrieval from the user and validation and verification of this information. It has basic input statements, try, error statements and data validation. This file helps hide the complexity of data collection, verification and validations from Atempt2.py
- **main.py**: The runnable file. This file used procedural programming to achieve or create user interface for the user, using all the functions and methods developed in the files above. This file abstracts the complexity of the entire program from the user.

Class Structure:

The program provides an interface for different groups, the restaurant, the customer and the deliverer, allowing them to interact with it each other through the use of t the following classes

- **OrderingSystem**: Manages the entire system, from the restaurants, customers, and delivery personnel. It stores all the restaurants, customers and delivery people and enables them to be used through out the program.
- **Members**: A base abstract class for shared attributes and methods among the different users in the system (e.g., Restaurant, Customer, DeliveryPerson). This class helps prevent code repetition
- **Restaurant**: Manages menu, order history, and updates order status.
- **Customer**: Allows customers to view restaurant menus, place orders and track their orders
- **Order**: Maintains order details, total price, delivery status, and assigned delivery person.
- **DeliveryPerson**: Tracks orders assigned for delivery and updates their status.
- **Payment classes (CashPayment, CreditCardPayment, MobileMoneyPayment)**: Supports different payment methods with unique fee structures and OTP verification for mobile payments.

The flow of the program

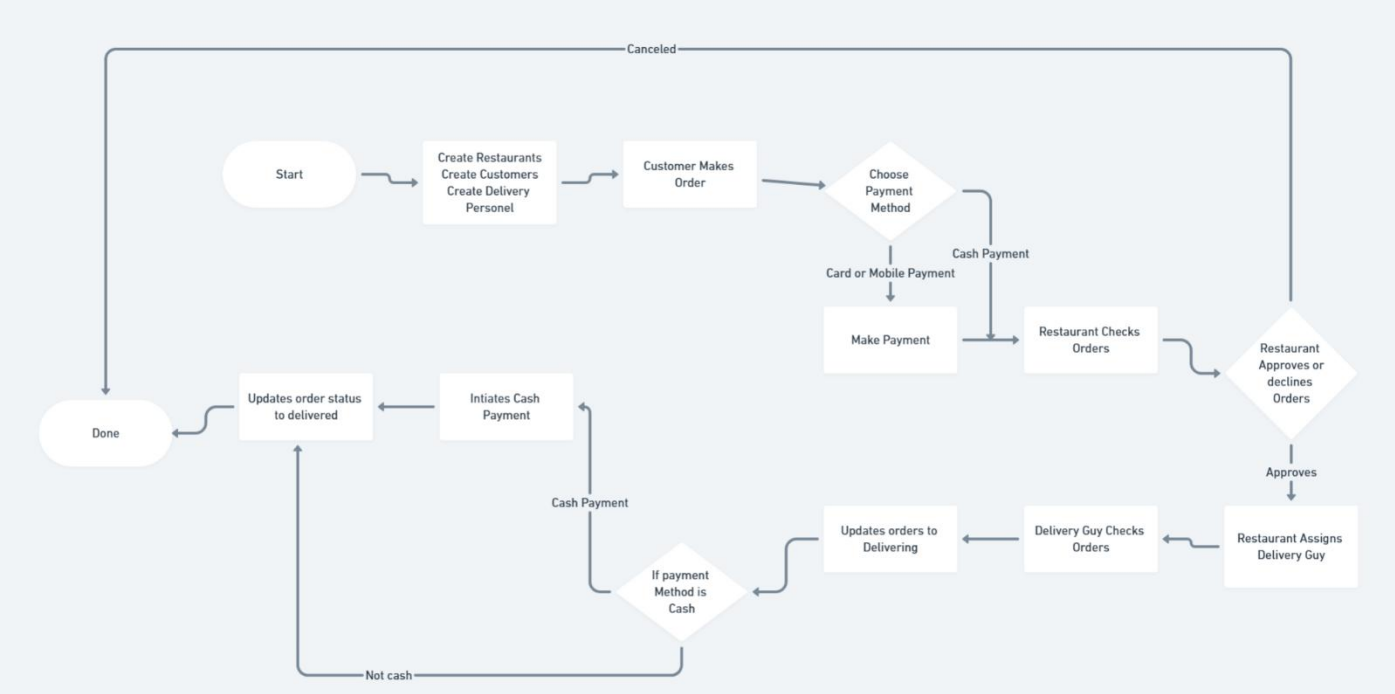

Figure 1 Flow of Program

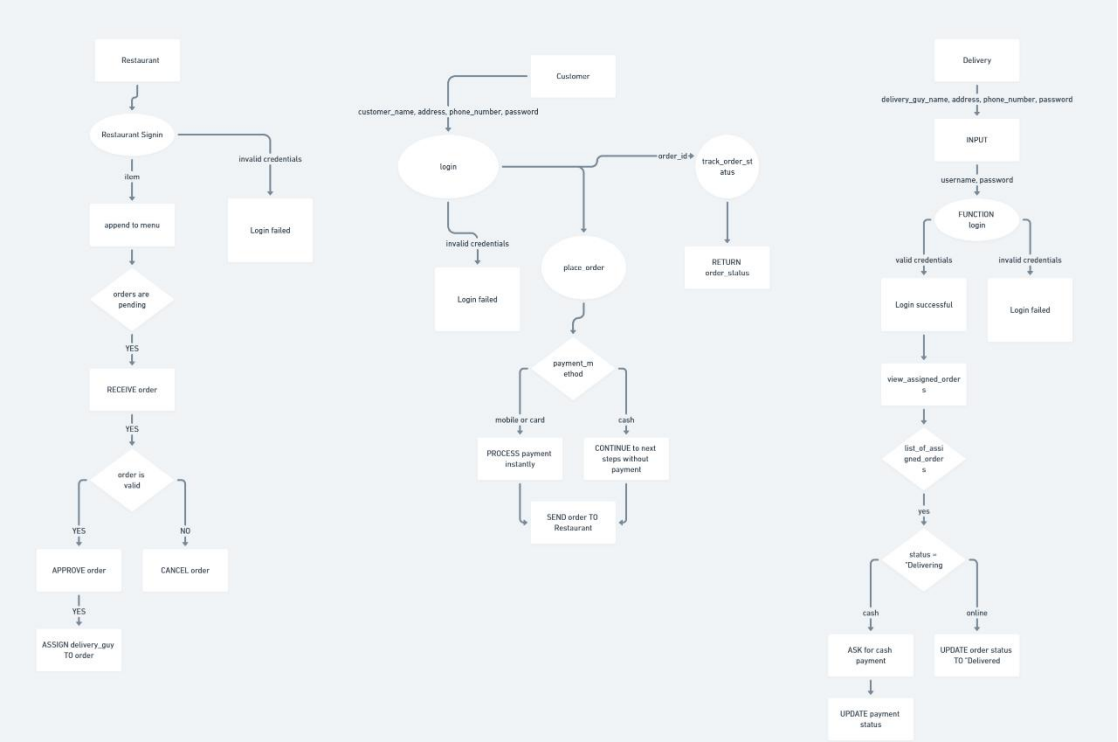

Figure 2 Flowchart for different Interfaces

4.0 Results

The project is a system that will be operated using the internet. This means that user only need an internet connection to access the delivery system. This means UCU students do not need to waste money on airtime, since we have eduroam and can therefore order from the comfort of their rooms.

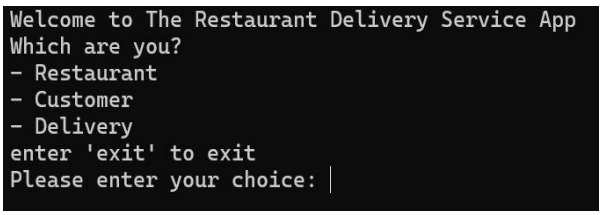

Figure 3 Main Menu

This project is a food delivery system that provide 3 user-interfaces for 3 different types of users; The restaurant, the students and the delivery person. This allows each user to interact with the order separately, ensuring a clean flow of information between them. This ensures the efficiency of the delivery service enabling the achievement of the objective creating a reliable food delivery platform.

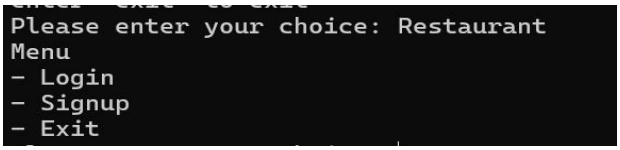

Figure 4 User Authentication Menu

Figure 2 shows that the system provides the ability for the different users, ensuring that only authorized individuals access they accounts and they can manage the information the system stores.

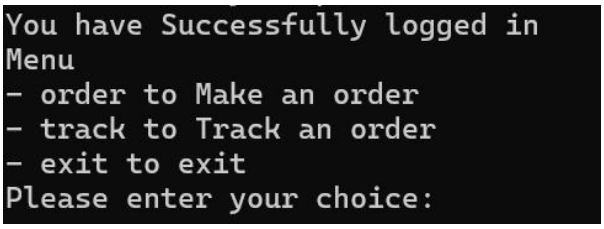

Figure 5 Customer Menu

As seen in Figure 3, this system enables users to make orders from specific restaurants, this reduces the chance of the customer getting a wrong order. This achieves the objective of boosting service accuracy. It allows the customer to track their order. This ensures that they are always able to know whether or not their order is being processed and how far long it will be. This achieves the objective of automated order placement and tracking.

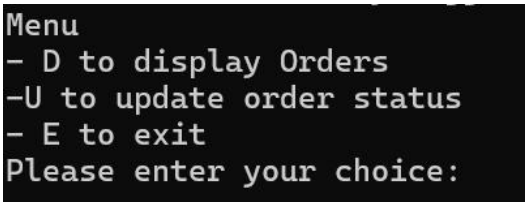

Figure 6 Restaurant Menu

Based on Figure 4 the system allows the restaurant to see all the orders that customers have made to it. It allows the Restaurant to update the order status. The restaurant can accept the order, or choose to cancel it if they don't have the request item available. This is means that both the restaurant and the user are kept informed about their orders, improving the automation of order placement and tracking

.

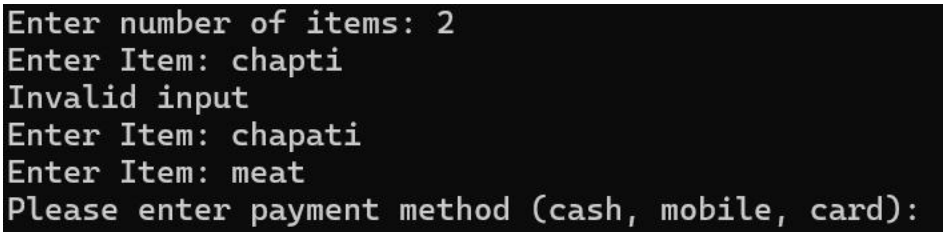

Figure 7 Order Menu

The project includes classes that manages different payment methods, including cash, credit card and mobile payments. The system allows the users to choose from these three modes of payments. This achieves the objective to establish a variety of payment methods. This make the delivery process more convenient for the user as they are not restricted to one form of payment

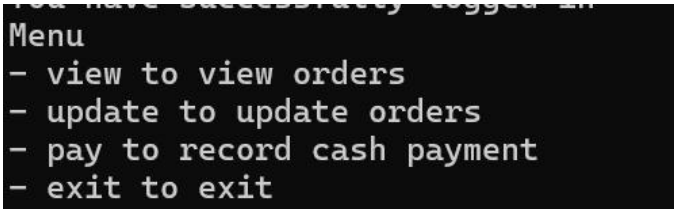

Figure 8 Delivery Personnel Menu

The system allows the delivery personnel to view the orders they have been assigned, update the status of the orders, and initiated cash payments. This allows the delivery person to know what orders he has to do deliver and from where. It also allows the customer to know at when the deliver person is bringing their order. Lastly it enables the delivery man to receive a cash as a payment method while keeping record of it. This ensures that the customer's order is not forgotten. End ensures the status of the order is tracked in real time.

Object-Oriented Design Implementation

Encapsulation:

```
class OrderingSystem:
    def __init__(self):
        self.__restaurants = []
        self.__customers = []
        self.__DeliveryPersons = []

    def set_deliver(self, instance):
        self.__DeliveryPersons.append(instance)

    def get_deliver(self):
        return self.__DeliveryPersons
```

Figure 9 OrderSystem Class

```
@property
def menu(self):
    return self.__menu

@menu.setter
def menu(self, noItems):
    for i in range (0,noItems):
        item =input(f"Please enter item {i+1}: ").lower(
        from Attempt3 import get_price
        price = get_price(item)
        self.__menu[item]=price
```

Figure 10 Restaurant Class Getter and Setter

Encapsulation is demonstrated in the code by using private attributes (e.g., __restaurants ,__orders, _amount, etc.), which are only accessible through getter and setter methods or specific functions within each class. In the OrderingSystem class, __restaurants,__customers, and __DeliveryPersons are encapsulated, and access to them is managed through the get_ and set_methods
In this project we used encapsulation to protect to the stored information from unauthorized access. This provides our users with a sense of security that their information won't be accessed by another member without the proper authentication.

Abstraction:

```
class Members(ABC):
    def __init__(self, name, address, number, password):
        self.__name = name
        self.__address = address
        self.__number = number
        self.__password = password

    @abstractmethod
    def display(self):
        pass
```

Figure 11 Abstract class

Abstraction is demonstrated with the Members class, which is an abstract class defining an interface for all types of members without implementation details. The display method is an abstract method in the Members class, forcing derived classes (like Customer, Restaurant, and DeliveryPerson) to implement this method based on specific details.

Inheritance:

```
class Restaurant(Members):
    def __init__(self, name, address,number, OH, CH, password):
        super().__init__(name, address, number, password)
        self.__openingHours = OH
        self.__closingHours = CH
        self.__menu = {}
        self.__orders = []
```

Figure 12 Inheritance Restaurat class

Inheritance is shown through the relationship between the Members superclass and its subclasses Customer, Restaurant, and DeliveryPerson. Here, Restaurant inherits from Members using super().__init__() to initialize common attributes, while adding specific attributes such as__openingHours and __closingHours. Inheritence has allowed us to reuse code, making our program less bulking and reducing the lines of code we have to write.

Polymorphism:

```
class CreditCardPayment(CashPayment):
    def __init__(self, amount, CVV, CardNo):
        super().__init__(amount)
        self.__CVV = CVV
        self.__cardno = CardNo

    def process_payment(self):
        # 2% fee for credit card payments
        fee = self._amount * 0.02
        self._amount += fee
        print(f"Credit Card processing fee applied.
        self._status = "Complete"
```

Figure 13 Polymorphism CreditCardPaymet class

```
class MobileMoneyPayment(CashPayment):
    def __init__(self, phone_number):
        self._phone_number = phone_number  # Encapsulated attribute for mobile money transactions
        self._otp = None  # Placeholder for OTP

    def process_payment(self,order):
        # Assume a fixed UGX 500 fee for mobile money transactions
        total = order.get_total()
        self._amount =total
        processing_fee = 500
        self._amount += processing_fee
        print(f"Mobile Money processing fee applied. Total amount: UGX {self._amount}")
```

Figure 14 Polymorphism MobileMoneyPayment Class

Polymorphism is illustrated by the process_payment method, which is implemented differently across various payment classes (CashPayment, CreditCardPayment, and MobileMoneyPayment), each processing payments with unique methods and fees.
Inheritance and polymorphism provide flexibility for adding new features. For example, the Payment classes are extended to handle new payment types

Challenges Faced

1. **Complexity in Implementing Real-Time Tracking**:
    - One of the main challenges was implementing a reliable mechanism for real-time order and delivery tracking. Creating a system that updates order status in real

time required designing efficient data flow between different classes (e.g., Order and DeliveryPerson), which involved complex interactions and regular testing.

2. **User Interface Design and Usability**:
    - Developing an intuitive and responsive user interface was challenging due to the need for compatibility with multiple roles (customer, restaurant staff, delivery personnel). Creating a simple yet functional GUI using Python's Tkinter library required significant adjustments to ensure ease of navigation and usability for all users and hence was not achieved.
3. **Managing Different Payment Methods**:
    - Integrating multiple payment methods while maintaining security and data integrity was complex. Each payment method had unique requirements, such as OTP verification for mobile money and processing fees for credit card transactions. Structuring the Payment class and its subclasses in a way that supports these differences while ensuring a cohesive payment process required careful planning and testing.
4. **Maintaining Data Consistency Across Classes**:
    - Ensuring data consistency across various interacting classes (e.g., Order, Customer, Restaurant) presented difficulties. For instance, changes in order status needed to be reflected across multiple instances and records. This was resolved by focusing on proper encapsulation and maintaining robust interaction patterns between objects.

Lessons Learned

1. **Importance of OOP Principles in Complex Systems**:
    - Applying OOP principles, particularly encapsulation and inheritance, demonstrated their value in building a structured and modular application. Encapsulation allowed us to manage sensitive data securely, while inheritance enabled code reuse across similar entities, reducing development time and enhancing maintainability.
2. **Significance of Clear Class Interactions**:
    - Defining clear responsibilities for each class and establishing well-defined interactions among them was crucial for maintaining data flow and avoiding redundancies. Establishing class roles early in the design process helped simplify complex operations, especially in areas such as order tracking and payment processing.
3. **Adaptability in Design for Future Scalability**:
    - Designing the system with extensibility in mind highlighted the importance of building adaptable code. By structuring the Payment class with polymorphism, for example, we laid the groundwork for easily integrating new payment methods without altering existing code, showcasing the benefits of scalable design.
4. **Value of Testing and Iteration**:
    - Frequent testing and iterative development were essential to identify and resolve issues early. As new features were added, consistent testing across different user roles and workflows ensured system reliability, particularly in the order placement and tracking functionalities.

5.0 Conclusion & Recommendation (1 Page)

Conclusion

This project successfully demonstrates how Object-Oriented Programming (OOP) principles can be applied to develop a food delivery system for the Uganda Christian University (UCU) students. We addressed challenges, such as unreliable service, high ordering costs, and limited payment options in our project and ensured the system provides a reliable and cost-effective solution. Through the use of encapsulation, inheritance, polymorphism, and abstraction, the project provides menu management, order tracking, delivery assignment, and payment processing, while ensuring secure data handling and code reusability.

The project achieves the following goals:

- **Streamlining Operations**: Automating tasks such as order placement, and delivery tracking to reduce manual effort and minimizes potential errors.
- **Enhancing Customer Experience**: Providing real-time order tracking, and efficient payment options

- **Supporting Business Growth**: Enabling restaurants to manage multiple orders efficiently while securely processing payments aids in scaling operations and improving profitability.

Recommendation

1. Create a GUI interface for the Application to make the project more user friendly
2. Do more validation and data verification to prevent User error.