USER

Owais Mukhtar  Shakir ullah

**café Management System report**

# **Report on the Program**

The Café Management and Accounting System (CMAS) is a Python-based application designed to manage customers, orders, sales, and expenses while also generating reports and tracking business performance. It employs object-oriented programming (OOP) principles like **encapsulation**, **abstraction**, **inheritance**, and **polymorphism**, which ensure code modularity, reusability, and clarity. Here's a detailed breakdown of how these principles are applied:

## **1. Encapsulation**

Encapsulation is implemented by grouping related properties and methods into classes, ensuring that data is only accessed through defined interfaces.

* **Usage in Program**:
  + Each functional aspect of the system (e.g., customers, orders, expenses, sales, and reports) is encapsulated within its respective class (Customer, Order, Expense, Sales, and Report).
  + Sensitive operations, like generating unique IDs or accessing database records, are handled through private methods (e.g., \_generate\_id, \_get\_data) within the BaseModel class. These are not directly accessible from outside, ensuring controlled interaction with the data.
  + Example: The Customer class encapsulates customer-related operations such as adding a customer, ensuring that other parts of the system don’t directly manipulate customer data.

## **2. Abstraction**

Abstraction is achieved by hiding complex details and exposing only the necessary functionality to the user.

* **Usage in Program**:
  + The BaseModel class provides generic methods (\_generate\_id, \_get\_data) that abstract the complexity of interacting with the database. Child classes like Customer or Order use these methods without needing to know the internal workings.
  + The CMASApp class abstracts the user interface logic by presenting simple buttons for tasks like "Add Customer" or "Track Sales". The user doesn’t see the underlying code handling these actions.
  + Example: The generate\_income\_statement method in the Report class hides the detailed computations of sales and expenses, presenting only the final income statement.

## **3. Inheritance**

Inheritance is used to create a hierarchy of classes, where child classes inherit common properties and methods from a parent class.

* **Usage in Program**:
  + The BaseModel class serves as the parent class for Customer, Order, Expense, Sales, and Report. It provides shared functionality like ID generation and database access, which these subclasses reuse and extend.
  + Example: The Expense class inherits \_get\_data from BaseModel to fetch expense data without redefining this functionality.

## **4. Polymorphism**

Polymorphism allows different classes to implement methods in their unique way, even if they share the same method name.

* **Usage in Program**:
  + The BaseModel class defines common operations (e.g., \_generate\_id, \_get\_data) that are reused but can be customized or extended in subclasses.
  + Each subclass (e.g., Customer, Order, Sales) processes data differently based on its context. For instance:
    - The process\_order method in Order handles order creation uniquely by calculating totals and associating items with a customer.
    - The record\_sale method in Sales takes an order and processes it into a sales record with a specific format.

## **How the Program Works**

1. **Data Management**:
   * A Database class handles CRUD (Create, Read, Update, Delete) operations using in-memory storage (e.g., JSON-like structures).
   * Records for Menu, Customers, Orders, Expenses, and Sales are stored and accessed centrally through this class.
2. **Core Functionalities**:
   * **Adding Customers**: Users can add new customers by entering their name and phone number. The system generates a unique ID for each customer.
   * **Processing Orders**: Orders are created by selecting menu items and quantities. The system calculates the total cost and links the order to a customer.
   * **Recording Sales**: Each processed order is converted into a sales record, tracking the date and total amount.
   * **Managing Expenses**: Users can add expenses categorized by type (e.g., rent, utilities) and track them by month.
   * **Reporting**: The system generates an income statement summarizing total sales, total expenses, and profit/loss. It can also produce a graphical representation of monthly sales data.
3. **User Interface**:
   * Built with Tkinter, the GUI allows admins to log in and perform tasks using buttons and input dialogs. Visual elements like bar charts for sales data are rendered using matplotlib.

## **Conclusion**

The program successfully demonstrates OOP principles to create a modular, maintainable, and scalable system. By encapsulating functionality, abstracting complex logic, reusing common behaviors through inheritance, and enabling polymorphic behavior, the CMAS application provides a robust solution for managing and analyzing café operations.

# **Flow chart**



Income Statement

add\_customer

check\_expenses

process\_order

Successful

Error

**Admin login**