#### **Class and Object**

#### Tasks:

1. Create a class Student with attributes name and age, and methods get\_details() that prints out the student details.
2. Define a class Book with attributes title, author, and price. Create a method show\_info() to print the book's information.
3. Create an object-oriented program to model a BankAccount class with attributes account\_number, balance, and methods like deposit() and withdraw().

**Inheritance**

**Tasks:**

1. Create a base class Vehicle with a method start(). Create two subclasses Car and Bike, both overriding the start() method with their specific functionality.
2. Define a base class Employee with attributes name and salary. Create a subclass Manager that inherits Employee and has an additional method get\_manager\_info().
3. Create a class Shape with a method area(). Then, create two subclasses Circle and Rectangle to calculate the area of different shapes.

**Polymorphism**

**Tasks:**

1. Create a parent class Person with a method speak(). Create two subclasses Teacher and Student, each overriding the speak() method differently. Demonstrate polymorphism.
2. Create a function calculate\_area() that takes objects of different shape classes (e.g., Square, Circle) and calculates their area using polymorphism.

3. Implement method overloading (using default arguments) for a multiply() method in a class MathOperations that can multiply either two or three numbers.

**Encapsulation**

**Tasks:**

1. Create a class PasswordManager with a private attribute \_\_password. Implement methods set\_password() and get\_password() to set and retrieve the password in a secure way.
2. Implement a Car class with private attributes like \_\_speed. Add methods increase\_speed() and show\_speed() to modify and display the speed.
3. Create a Student class with private attributes \_\_marks. Add methods to safely update and retrieve the student's marks while hiding the attribute from direct access.

**CRUD App with PyMySQL**

**Setup and Database Creation**

* **Task 1.1:** Install PyMySQL and ensure MySQL is running on your machine.
* **Task 1.2:** Create a MySQL database named crud\_app\_db using PyMySQL from your Python script.
* **Task 1.3:** Create a table named employees with columns for id, name, age, department, and salary.

**Columns Structure:**

* + id (Primary Key, Auto Increment)
  + name (VARCHAR)
  + age (INT)
  + department (VARCHAR)
  + salary (DECIMAL)

**Create Functionality**

* **Task 2.1:** Write a Python function that connects to the MySQL database.
* **Task 2.2:** Take employee details from the user (e.g., name, age, department, salary) and insert them into the employees table.
* **Task 2.3:** Ensure the program handles errors and confirms successful insertion (use try-except blocks).

**Read (Retrieve) Functionality**

* **Task 3.1:** Write a function to fetch all employee records from the employees table and display them in a user-friendly format (not just tuples).
* **Task 3.2:** Add functionality to retrieve a single employee's details based on their id (input from the user).
* **Task 3.3:** Handle cases where the employee with the specified id does not exist, and show an appropriate message.

**Update Functionality**

* **Task 4.1:** Write a function that allows the user to update an employee’s data (e.g., change their department or salary) based on their id.
* **Task 4.2:** Ensure the user can update only specific fields without modifying other fields.
* **Task 4.3:** Show a message after a successful update, and handle cases where the employee does not exist.

**Delete Functionality**

* **Task 5.1:** Write a function to delete an employee’s record based on their id.
* **Task 5.2:** Prompt the user for confirmation before performing the deletion (e.g., "Are you sure you want to delete this record? [y/n]").
* **Task 5.3:** Ensure the system correctly handles cases where the employee to be deleted does not exist, showing an appropriate message.

**API Integration Using Requests Library**

**Setup and Environment**

* **Task 1.1:** Install the requests library if not already installed (pip install requests).
* **Task 1.2:** Identify a public or mock API that supports CRUD operations (e.g., JSONPlaceholder, MockAPI).

**Create (POST) Operation**

* **Task 2.1:** Create a Python function to **create a new resource** (e.g., add a new employee, user, post, etc.) by sending a POST request to the API.
* **Task 2.2:** Gather the necessary data from the user (e.g., name, age, department) and send it as JSON in the request body.
* **Task 2.3:** Handle the response from the API, displaying whether the creation was successful and showing the returned data (e.g., new record ID).

**Read (GET) Operation**

* **Task 3.1:** Write a function to **fetch all resources** (e.g., list all employees) by sending a GET request to the API.
* **Task 3.2:** Display the data in a user-friendly format, making sure to handle cases where no records are found or the server returns an error.
* **Task 3.3:** Add functionality to fetch a **specific resource by ID** using a GET request and displaying detailed information about that resource.

**Update (PUT) Operation**

* **Task 4.1:** Write a function to **update an existing resource** (e.g., modify an employee's details) by sending a PUT request to the API.
* **Task 4.2:** Take inputs from the user for the id of the resource they want to update, and ask for the new data to be updated (e.g., new salary or department).
* **Task 4.3:** Handle the API's response to show a success message or error in case the update fails (e.g., resource not found).

**Delete (DELETE) Operation**

* **Task 5.1:** Write a function to **delete a resource** (e.g., remove an employee) by sending a DELETE request to the API.
* **Task 5.2:** Ask the user to provide the id of the resource they want to delete and confirm the action (e.g., "Are you sure?").
* **Task 5.3:** Display appropriate messages based on whether the deletion was successful or if the resource did not exist.