**C++ Lab Assignments**

**Prepared By: M Shravan Date: 27-03-2025**

**=================================================================**

**1. Class and Object**

**Assignment: Create a Class for Managing Employee Data**

**Objective:** Learn how to define a class and create objects.

**Description:**

* Create a class Employee that contains:
  1. Private members: name, employeeId, salary.
  2. Public member functions to:
     1. Set employee details (name, ID, and salary).
     2. Display employee details.
* In the main() function:
  1. Create an object of the Employee class.
  2. Use member functions to set and display the employee's data.

**Requirements:**

* Define the class with both private and public members.
* Create an object in main() and access its members using functions.

**Objective:**  
Understand the basic usage of classes and objects, and how to use functions for setting and displaying object data.

**2. Abstraction**

**Assignment: Implement a Bank Account System with Abstraction**

* **Objective:** Understand the concept of abstraction by hiding implementation details and exposing only the relevant interface.

**Description:**

* Create a class BankAccount with the following attributes:
  + Private members: accountNumber, balance.
  + Public functions:
    - deposit(), withdraw(), displayBalance().
    - A private function validateTransaction() to check whether a withdrawal is valid.
* The class should only expose the relevant functionalities (deposit, withdraw, and balance check), while hiding the implementation details of the transaction validation.

**Requirements:**

* Implement functions that simulate bank account operations.
* Use abstraction to hide the validation logic from the user interface.

**Objective:**  
Learn how to hide unnecessary details from the user and only expose relevant information (abstraction).

**3. Encapsulation**

**Assignment: Design a Class for Car with Encapsulation**

* **Objective:** Understand how encapsulation works by restricting direct access to data members and providing controlled access via methods.

**Description:**

* Create a class Car with the following attributes:
  + Private members: make, model, year, and speed.
  + Public methods:
    - Setters and getters for all attributes (to access private data).
    - accelerate() to increase the car's speed.
    - brake() to decrease the car's speed.
* Ensure that all data members are private and provide proper getter and setter methods for each attribute.

**Requirements:**

* Use setter and getter methods to access private members, ensuring encapsulation.
* The car's speed should not be directly modified from outside the class; it should be controlled by methods.

**Objective:**  
Understand how to use encapsulation to restrict access to an object’s internal state and provide controlled access through public methods.

**4. Inheritance**

**Assignment: Create a Class Hierarchy for Animals (Inheritance)**

* **Objective:** Learn how inheritance works by creating a hierarchy of classes.

**Description:**

* Create a base class Animal with:
  + Private members: name, age.
  + Public methods: makeSound(), eat(), and a constructor to initialize the name and age.
* Create two derived classes Dog and Cat that inherit from Animal:
  + Add specific methods to the derived classes:
    - bark() for the Dog class.
    - meow() for the Cat class.
* In the main() function:
  + Create objects of both Dog and Cat.
  + Demonstrate polymorphism by calling methods on the base class (Animal), and show that the correct methods are invoked from the derived classes.

**Requirements:**

* Use inheritance to extend functionality in derived classes.
* Override the makeSound() method in both derived classes.

**Objective:**  
Understand how to use inheritance to extend the functionality of a base class and reuse code in derived classes.

**5. Combining Inheritance, Abstraction, and Encapsulation**

**Assignment: Create a Vehicle System Using Inheritance, Abstraction, and Encapsulation**

* **Objective:** Combine inheritance, abstraction, and encapsulation to create a real-world system.

**Description:**

* Create an abstract base class Vehicle with:
  + Private members: make, model, year.
  + A pure virtual function startEngine() (this makes Vehicle an abstract class).
* Create two derived classes Car and Motorcycle:
  + Both classes should implement the startEngine() function, displaying a message specific to each vehicle.
  + Include a method displayDetails() in each derived class to show the make, model, and year.
* Ensure that all attributes are encapsulated (private), and provide getter methods to access them.

**Requirements:**

* Use an abstract base class with a pure virtual function.
* Ensure that Car and Motorcycle inherit from Vehicle and provide their own implementation of startEngine().
* Demonstrate polymorphism by creating an array of Vehicle pointers and invoking startEngine() dynamically.

**Objective:**  
Understand how to combine multiple OOP principles (inheritance, abstraction, and encapsulation) to design a real-world system.

**6. Polymorphism (Overloading and Overriding)**

**Assignment: Shape Area Calculation Using Polymorphism**

* **Objective:** Learn how to achieve polymorphism using method overloading and overriding.

**Description:**

* Create a base class Shape with a function calculateArea():
  + The calculateArea() function will be overloaded to handle different types of shapes (e.g., Circle, Rectangle).
  + Overload calculateArea() to calculate the area of both circles (π \* r²) and rectangles (length \* width).
* Create derived classes Circle and Rectangle:
  + Each class should implement its own version of calculateArea().
* Demonstrate polymorphism by calling calculateArea() on objects of both Circle and Rectangle using a Shape pointer or reference.

**Requirements:**

* Use function overloading to calculate areas for different shapes.
* Use function overriding for derived classes to calculate areas of their specific shapes.

**Objective:**  
 Learn about polymorphism through both method overloading and overriding.

**7. Constructor and Destructor in Inheritance**

**Assignment: Constructor and Destructor in Inheritance (Constructor Chaining)**

* **Objective:** Understand constructor chaining and destructor behavior in an inheritance hierarchy.

**Description:**

* Create a base class Shape with a constructor that takes parameters for width and height (for rectangle-like shapes).
* Create a derived class Rectangle with a constructor that initializes the Shape base class and its own additional parameters (length and width).
* Add a destructor to both classes to print messages when an object is destroyed.
* In the main() function:
  + Create a Rectangle object, demonstrating constructor chaining (base class constructor being called first).
  + Observe the order of destructor calls when the object goes out of scope.

**Requirements:**

* Demonstrate constructor chaining and proper use of destructors.
* Observe the destruction order of base and derived class objects.

**Objective:**  
Understand constructor chaining in inheritance and how destructors work in class hierarchies.

**8**. **Function Overloading**

**Assignment: Create a Calculator Class with Function Overloading**

**Objective:** Demonstrate function overloading for mathematical operations.

**Description:**

* Create a class Calculator with overloaded functions to perform addition, subtraction, and multiplication for both integers and floating-point numbers.

**Real-Time Application:**  
This can be used in simple calculation-based applications like scientific calculators, financial apps, etc.

**Tasks:**

* Overload functions for addition, subtraction, and multiplication of integer and float types.
* Allow users to input two numbers and choose an operation to perform.

**9**. **Operator Overloading**

**Assignment: Create a Time Class with Operator Overloading**

**Objective:** Demonstrate operator overloading for the + operator.

**Description:**

* Create a Time class to represent time in hours, minutes, and seconds.
* Overload the + operator to add two Time objects together.

**Real-Time Application:**  
This can be applied in project management tools to calculate cumulative time, or in event scheduling systems.

**Tasks:**

* Define the + operator to add two Time objects.
* Create two Time objects, perform the addition, and display the result.

**10**. **Virtual Function**

**Assignment: Create a Payment System with Virtual Functions**

**Objective:** Demonstrate dynamic polymorphism using virtual functions.

**Description:**

* Create a base class Payment with a virtual function processPayment().
* Derive two classes CreditCard and Paypal, each overriding the processPayment() function.

**Real-Time Application:**  
This can be used in e-commerce applications where different payment methods need to be processed.

**Tasks:**

* Implement a virtual function in the base class.
* Override the processPayment() function in the derived classes.

**11. Pure Virtual Function**

**Assignment**: Create a Shape System with a Pure Virtual Function

**Objective**: Demonstrate pure virtual functions and abstract classes.

**Description**:

* Create an abstract base class Shape with a pure virtual function draw().
* Derive classes Circle and Rectangle that implement the draw() method.

**Real-Time Application**:  
This can be used in graphical applications where different shapes need to be drawn, and the drawing method needs to be defined specifically for each shape.

**Tasks**:

* Define an abstract class with a pure virtual function.
* Derive at least two classes and implement the pure virtual function.

**12**. **Inline Function**

**Assignment: Create a Discount Calculator with Inline Function**

**Objective:** Demonstrate the use of inline functions for efficiency.

**Description:**

* Create a class Product with a member function calculateDiscount().
* Use an inline function for calculating the discount price.

**Real-Time Application:**  
This is used in e-commerce websites or retail systems where discount calculations happen frequently.

**Tasks:**

* Implement the calculateDiscount() function as an inline function.

**13**. **Friend Function**

**Assignment: Create a Bank Account System with Friend Function**

**Objective:** Use a friend function to access private data in a class.

**Description:**

* Create a BankAccount class with private members for account balance.
* Implement a friend function transferMoney() that can access and modify private data.

**Real-Time Application:**  
This can be applied in banking systems where transferring money between accounts requires special permissions.

**Tasks:**

* Create a friend function that modifies private members of a class.

**14**. **Constructor Overloading**

**Assignment: Create a Library System with Constructor Overloading**

**Objective:** Demonstrate constructor overloading.

**Description:**

* Create a Book class with a constructor that can be overloaded to initialize different attributes of a book (title, author, and year).

**Real-Time Application:**  
 This can be used in a library management system to manage different book records.

**Tasks:**

* Implement constructor overloading to allow different ways to initialize a book.

**=======================\*\*\*\*\*\*\*\*\*\*===========================**