OBJECT ORIENTED PROGRAMMING

Lab Assignment: Inheritance, Method Overriding, and Constructor Chaining

17 September 2026

---

1. **Single Inheritance (Account and SavingsAccount)**
   Design a program containing a base class `Account` and a derived class `SavingsAccount`. The base class must store `accountNumber` and `balance`, initialized via a parameterized constructor. The derived class should add an `interestRate` attribute and implement a member function `calculateInterest()` that computes the annual interest earned based on the current balance. Create an instance of `SavingsAccount` and display the complete account summary.

2. **Multilevel Inheritance: 3-Tier Academic Hierarchy**
   Implement a three-tier multilevel inheritance hierarchy consisting of the classes `Person`, `Student`, and `GraduateStudent`:

   (a) `Person` contains `name` and `age`.

   (b) `Student` inherits from `Person` and introduces `rollNumber` and `branch`.

   (c) `GraduateStudent` inherits from `Student` and adds `researchArea` and `guideName`.

   Ensure each derived class forwards initialization values to its base class constructor. Implement a function `displayProfile()` in `GraduateStudent` to output all inherited and local attributes.

3. **Multilevel State Propagation (Vehicle, Car, ElectricCar)**
   Implement a multilevel inheritance structure with the classes `Vehicle`, `Car`, and `ElectricCar`:

   (a) `Vehicle` stores `brand` and `model`.

   (b) `Car` inherits from `Vehicle` and adds `seatingCapacity`.

   (c) `ElectricCar` inherits from `Car` and adds `batteryCapacity` (in kWh).

   Instantiate an `ElectricCar` object and verify through a display function that data members across all ancestral levels are properly initialized and accessible.

4. **Hierarchical Inheritance: Faculty Payroll System**
   Create a hierarchical inheritance model with a base class `Faculty` and two derived classes, `Professor` and `AssistantProfessor`. The base class stores `employeeID`, `name`, and `basicSalary`. `Professor` adds `publicationsCount`, while `AssistantProfessor` adds `labHoursAssigned`. Implement `printPayrollSlip()` in both derived classes to calculate and output the gross salary with their respective allowances (20% for Professor, 10% for Assistant Professor).

5. **Hierarchical Inheritance: Geometric Shapes**
   Implement a shape classification program using hierarchical inheritance. Define a base class `Shape` that stores a `color` attribute. Derive two independent subclasses, `Circle` and `Rectangle`:

   (a) `Circle` introduces `radius`.

   (b) `Rectangle` introduces `length` and `breadth`.

   Implement functions in each derived class to compute the area and print the result alongside the inherited color.

6. **Multiple Inheritance: Office Machine Integration**
   Implement multiple inheritance to model a composite office machine. Define two independent parent classes:

   (a) `Printer` containing a member function `printDocument()`.

(b) `Scanner` containing a member function `scanDocument()`.

Derive a class `AllInOneMachine` that inherits from both `Printer` and `Scanner`. Instantiate an object of `AllInOneMachine` and execute both operational functions sequentially. *(Note: In languages without direct multiple class inheritance, implement using interfaces.)*

7. **Multiple Inheritance: Student Scholarship Evaluation**
Design a program demonstrating multiple inheritance for student evaluation. Define a class `AcademicPerformance` storing `theoryMarks` and a class `SportsPerformance` storing `sportsGrade`. Derive a class `StudentReport` inheriting from both classes. Implement a member function `generateSummary()` that evaluates both academic and athletic performance to determine scholarship eligibility.

8. **Method Overriding and Dynamic Dispatch**
Write a program demonstrating method overriding. Define a base class `Employee` containing a function `calculateBonus()` that computes a 5% bonus on a base salary. Derive two classes, `Manager` and `Intern`. Override `calculateBonus()` in `Manager` to return 15% and in `Intern` to return a flat constant amount. Instantiate objects and demonstrate polymorphic invocation of the overridden functions.

9. **Constructor Chaining across Inheritance Boundaries**
Implement explicit constructor chaining across an inheritance hierarchy. Create a class `Box` that stores `width`, `height`, and `depth`, initialized via a parameterized constructor. Derive a subclass `HeavyBox` that introduces `weight`. The constructor of `HeavyBox` must pass dimensional arguments directly to the base constructor without manual reassignment. Display all dimensions and weight.

10. **Targeted Base Method Invocation from Overridden Function**
Write a program demonstrating targeted base method invocation from an overridden function. Create a base class `BaseLogger` containing a method `log()` that prints a message prefixed with an informational tag. Derive a class `TimestampLogger` that overrides `log()`. The overridden function must output a current timestamp and explicitly call the base class `log()` implementation to output the original message body.

11. **Multilevel Chaining and Progressive Overriding: Order Management**
Implement an order processing hierarchy demonstrating multilevel constructor chaining and progressive method overriding:

    (a) Class `Order` contains `orderId`, `orderDate`, and `baseAmount`, with a function `calculateTotal()` returning `baseAmount`.

    (b) Class `ShippedOrder` inherits from `Order`, adds `trackingNumber` and `shippingFee`, and overrides `calculateTotal()` to return the base total plus `shippingFee`.

    (c) Class `DiscountedShippedOrder` inherits from `ShippedOrder`, adds `discountPercentage`, and overrides `calculateTotal()` to apply the discount on the amount calculated by `ShippedOrder`.

    Create an instance of `DiscountedShippedOrder` and print a detailed breakdown of all charges and the final payable total.