Programming Assignment: Object-Oriented Programming in C++

Instructions:

- Implement each program in C++.
- Follow proper object-oriented design principles.
- Use meaningful variable names and comments for better understanding.
- Test each program with multiple cases.

Section 1: Inheritance and Types of Inheritance

1. Single Inheritance

A. Create a base class `Employee` with attributes `name`, `id`, and `salary`.

- Derive a class `Manager` that adds an attribute `bonus`.
- Write a function to display the total salary (`salary + bonus`).
- Demonstrate creating an object of `Manager` and displaying details.

B. Create a base class `Person` with attributes `name` and `age`.

- Derive a class `Student` that adds `studentID` and `course`.
- Derive another class `Teacher` that adds `teacherID` and `subject`.
- Demonstrate inheritance by creating objects of `Student` and `Teacher` and displaying details.

C. Create a base class `BankAccount` with protected attributes `accountNumber` and `balance`.

- Derive a class `SavingsAccount` that adds a method `calculateInterest()`.
- Implement a method to deposit and withdraw money while maintaining balance constraints.
- Demonstrate object creation and transactions.

---

2. Multilevel Inheritance

A. Create a base class `Person` with attributes `name` and `age`.

- Derive a class `Student` with `rollNumber` and `course`.
- Further derive a class `GraduateStudent` with `thesisTitle`.
- Write functions to display details at each level.

B. Create a base class `Animal` with a method `eat()`.

- Derive a class `Mammal` that adds a method `walk()`.
- Derive a class `Dog` from `Mammal` that adds a method `bark()`.
- Demonstrate calling functions using a `Dog` object.

---

3. Multiple Inheritance

A. Create two base classes:

- `Sports` with attributes `sportName` and `score`.
- `Academics` with attributes `subject` and `marks`.
- Derive a class `StudentPerformance` that inherits both classes and calculates total performance.
- Demonstrate creating an object and displaying results.

B. Create two base classes:

1. `Employee` with `employeeID` and `salary`.
2. `Person` with `name` and `age`.

Derive a class `Manager` from both classes that adds `department`.

- Show how multiple inheritance works by creating a `Manager` object and displaying details.

---

4. Hierarchical Inheritance

Create a base class `Vehicle` with attributes `brand` and `year`.

- Derive two classes:
  - `Car` (adds `fuelType`).
  - `Bike` (adds `engineCC`).
- Demonstrate creating objects of both derived classes and displaying details.

---

5. Hybrid Inheritance

A. Create the following structure:

- Base class `Vehicle` with `brand` and `speed`.
- Class `Car` derived from `Vehicle` that adds `numDoors`.
- Class `Bike` derived from `Vehicle` that adds `hasGear`.
- Class `SportsCar` derived from both `Car` and `Bike` that adds `turboMode()`.

Demonstrate how hybrid inheritance works.

B. Create a class `Person` with attributes `name` and `age`.

- Derive `Student` and `Teacher` from `Person`.
- Create a derived class `TeachingAssistant` that inherits from both `Student` and `Teacher`.
- Demonstrate resolving ambiguity using virtual base classes.

Section 2: Dynamic Polymorphism and Virtual Functions

6. Virtual Function for Method Overriding

Create a base class `Shape` with a virtual function `area()`.

- Derive two classes:
  - `Circle` (calculates area using πr^2).
  - `Rectangle` (calculates area using `length × breadth`).
- Demonstrate calling `area()` dynamically.

---

7. Pure Virtual Function & Abstract Class

Create an abstract class `Animal` with a pure virtual function `makeSound()`.

- Derive `Dog` and `Cat` classes that implement `makeSound()`.
- Demonstrate polymorphism using pointers to the base class.

---

8. Dynamic Method Dispatch Using Virtual Functions

Create a base class `BankAccount` with a virtual function `calculateInterest()`.

- Derive classes:
  - `SavingsAccount` (fixed interest rate).
  - `CurrentAccount` (no interest).
- Demonstrate calling `calculateInterest()` dynamically.

---

9. Virtual Destructor

Create a base class `Base` with a destructor that prints a message.

- Derive a class `Derived` with its own destructor.
- Demonstrate deleting an object using a base class pointer and observe destructor behavior.

---

10. Abstract Class with Multiple Derived Classes

Create an abstract class `Employee` with a pure virtual function `calculateSalary()`.

- Derive two classes:

    - `FullTimeEmployee` (fixed monthly salary).
    - `PartTimeEmployee` (hourly wage).
- Demonstrate polymorphic behavior with different objects.

Section 3: Exception Handling

11. Exception Handling: Division by Zero

Write a function `divide(int a, int b)` that performs division.

- Throw an exception if `b == 0` and handle it gracefully in `main()`.

---

12. Exception Handling with Multiple Catch Blocks

Create a program that:

- Takes an integer input.
- Throws different exceptions for negative numbers, zero, and large numbers (>1000).
- Catches and handles these exceptions accordingly.

---

13. Exception Handling in Class Methods

Create a class `Student` with attributes `name` and `marks`.

- Write a function `setMarks(int m)` that:
  - Throws an exception if `m < 0` or `m > 100`.
  - Handles the exception in `main()`.

---

14. Exception Handling in Constructors

Create a class `BankAccount` with attributes `accountNumber` and `balance`.

- Throw an exception in the constructor if the balance is negative.
- Handle the exception in `main()`.

---

15. User-Defined Exception Class

Create a custom exception class `InvalidAgeException`.

- Write a function `checkAge(int age)` that throws `InvalidAgeException` if age is less than 18.
- Demonstrate handling this exception in `main()`.