DSA Practical Exam - 1 [80 Marks]

Instructions:

1. Execute the given programs in your preferred IDE.
2. Save all your files and submit In Github.
3. Marks will be awarded based on correctness, efficiency, and proper commenting.

Q1. Object-Oriented Programming (10 marks)

Input-Output Explanation:

- **Input:** The program will define classes and objects representing real-world entities (e.g., `Student` and `Teacher`). You will input data such as names, IDs, and other properties for these objects.
- **Output:** The program will demonstrate how principles like abstraction (hiding details), encapsulation (using private attributes and methods), inheritance (reusing code from parent classes), and polymorphism (overriding methods) work. It will also show the use of the `this` keyword to resolve naming conflicts.

Q2. Data Encapsulation (10 marks)

Input-Output Explanation:

- **Input:** Create a `BankAccount` object with attributes like account number and balance. Input new balance values using setter methods.
- **Output:** The program should display account details like account number and updated balance using getter methods. The private attributes should not be directly accessible outside the class, ensuring encapsulation.

Q3. Array of Objects and Static Members (10 marks)

Input-Output Explanation:

- **Input:** Enter details of 5 books (title, author, and price). Provide a threshold value to filter books based on price.
- **Output:** The program will display the details of books priced above the given threshold. Additionally, static members will count and display the total number of books created.

Q4. Constructor and Destructor (10 marks)

Input-Output Explanation:

- **Input:** Define a class and create objects using different constructors (default, parameterized, copy). Input data specific to constructors (e.g., for a parameterized constructor, input values for initializing attributes).
- **Output:** The program will display the initialized object attributes and demonstrate how destructors are invoked automatically when objects go out of scope.

Q5. Inheritance and Ambiguity Resolution (10 marks)

Input-Output Explanation:

- **Input:** Create a hierarchy with base and derived classes. Input data relevant to the classes (e.g., student details in one class and teacher details in another).
- **Output:** The program will show single and multiple inheritance in action. For multiple inheritance, ambiguity will be resolved using the membership label operator, ensuring correct attribute or method access.

Q6. Polymorphism (10 marks)

Input-Output Explanation:

- **Input:** Define a base class `Shape` with a method `calculateArea`. Create objects of derived classes `Circle` and `Rectangle` and input dimensions like radius, length, and width.
- **Output:** The program will calculate and display the area for each shape using method overriding (runtime polymorphism). It will also demonstrate how the base class pointer calls the derived class method at runtime.

Q7. Data Abstraction and Abstract Classes (20 marks)

1. Create an abstract base class `Employee` with the following:
   - A pure virtual function `calculateSalary()` and a normal function `displayDetails()`.
2. Create two derived classes `FullTimeEmployee` and `PartTimeEmployee` that override the `calculateSalary()`method.
   - Input and display employee details like name, hours worked, and salary.
3. Use different access modifiers (`public`, `private`, and `protected`) in inheritance to demonstrate access control in the derived classes.

Explanation:

Input:

- Define attributes for `Employee` (e.g., name, employee ID) and specific attributes for derived classes (e.g., `hoursWorked` for `PartTimeEmployee` or `monthlySalary` for `FullTimeEmployee`). Input these values during runtime.

Output:

- Display the name, ID, and calculated salary of each employee using the overridden `calculateSalary()` function.
- Demonstrate abstraction by preventing direct instantiation of the `Employee` class (abstract class).
- Show access control in action by attempting to access attributes/methods with different modifiers (`public`, `protected`, `private`) from derived and external classes.