1. Introduction

Define Object-Oriented Programming and its significance:

Object-Oriented Programming is a programming paradigm based on concept of objects, instances of classes, which help in organizing code into reusable structures.

Significances:

1. Code Reusability – Through inheritance,
2. Modularity and Maintainability – Code is organized into separate classes making it easier to debug and modify,
3. Data security – Through encapsulation hence restricting direct access,
4. Scalability – Structuring code in a way that supports extension and growth,
5. Abstraction – Unnecessary details are hidden from the user allowing them to focus on essential functionalities,

Explain the key OOP principles (Encapsulation, Inheritance, Polymorphism, and Abstraction) with examples.

1. Encapsulation:

Bundling data and methods that operate on that data in a single unit or class. Hides internal state of an object.

Example: Consider a class BankAccount that has private variables like balance and accountNumber. The class provides public methods like deposit and withdraw to manipulate the balance, preventing unauthorized direct access.

2. Inheritance:

Allows a new(derived) class to acquire properties from an existing(base) class. This promotes code reuse and hierarchical classification.
Example: A base class Animal may have attributes like movement and sound. A derived class Cat inherits these properties and may add specific features as claws.

3. **Polymorphism:**
   Enables a single interface to represent different data types or functions, it can be achieved through function overloading, operator overloading, and dynamic(runtime) polymorphism using virtual functions.
   Example: A function move() might behave differently depending on whether its applied to an object of class Bird or Fish.

4. **Abstraction:**
   Hiding implementation details from the user and exposing only the essential features. Making code reusable.
   Example: Consider a Bank Account system where users deposit and withdraw money but hide the internal implementation of how transactions are processed.

2. Analysis of the Case Scenario

Identify the key functional requirements of the employee management system.

1. Employee Record Creation – Adding new employee: name, employee ID, and salary.
2. Employee Record Retrieval – Displaying all employees. Search for an employee by their ID and display their details.
3. Employee Record Modification – Update details such as name and salary.
4. Employee Record Deletion
5. Secure Data Handling – Employee attributes (name, ID, salary) should be private with public getter and setter methods hence Encapsulation
6. Extension for different employee types – Design should allow for creating subclasses (Manager, Intern) that inherit from base Employee class
7. Polymorphic behaviour – Functions such as calculating bonuses or allowances can be overridden in derived classes.

Discuss how OOP principles can be applied to design the system effectively.

1. Encapsulation:

Employee Class has attributes: Name, Employee ID, Salary. It also has methods: Input employee details, Display employee details.
EmployeeManagementSystem class methods: Add Employee, Display all Employees, Search Employee by ID.

2. Inheritance:

Base class Employee: Attributes: Name, Employee ID, Salary,
Methods: Input employee details, Display employee details.
Derived classes:

1. Manager (inherits from Employee):
   Additional Attributes: Department, Bonus,
   Additional Methods: Display manager details including department and bonus
2. Engineer (inherits from Employee):
   Additional Attributes: Department, Bonus,
   Additional Methods: Display manager details including department and bonus.

3. Polymorphism:

Employee (Base) class: methods: Display employee details
Manager (Derived) class: methods: Display manager details including department and bonus
Engineer (Derived) class: methods: Display engineer details including specialization and project

4. Abstraction:

Employee class: methods: Input employee details
EmployeeManagementSystem class: methods: Add Employee
The user will not know what is involved in the database but just input details

4.Conclusion and Future Recommendations

Summarize the importance of OOP in software development

1. Code reusability through inheritance – Manager and Engineer classes are inheriting methods and attributes from Employee class hence no need to write twice.
2. Flexibility through polymorphism – Manager class displays its department and bonus as a unique feature of the display employee details from Employee class.
3. Modularity and encapsulation – organize methods and attributes into individual classes and makes it easier to debug and modify, for example adding display for salary in Engineer class at Engineer method to display
4. Abstraction for Simplification – EmploymentManagementSystem class hides the complex implementations to add employees by providing a method Add Employee
5. Enhanced Maintainability and Scalability – Since code is organized in classes, to scale it further you can easily add another class and to modify you deal with an individual class.

Provide recommendations on how the employee management system can be further improved using advanced OOP concepts.

1. Multiple Inheritance and Mixins:
   Consider mixin classes for shared functionality that does not fit neatly into primary inheritance hierarchy, such as auditing employee expenses and such
2. Layered architecture:
   Separate concerns by dividing the system into layers e.g., engineering dept, finance, data access, presentation.
3. Resource Acquisition Is Initialization (RAII):
   Use RAII to manage resources and ensure that they are released properly even in the event of an exception.
4. Single Responsibility Principle:

Ensure each class has one responsibility like separating the data model (employee details) from business logic (operating on the data).

5. Using Smart Pointers:

   Replace raw pointers with smart pointers (std::unique_ptr) to manage memory safely and prevent memory leaks.