1.

Object-Oriented Programming (OOP):

Definition: Programming using "objects" (data + actions) and "classes" (blueprints for objects).

Significance: Organizes code, promotes reuse, flexibility, maintainability, and real-world modeling for easier software development.

Key OOP Principles:

Encapsulation:

Definition: Bundling data & methods in objects; hiding internal data for controlled access.

Example: Car object: Access speed only via methods like accelerate(), not directly.

Inheritance:

Definition: Derived classes inherit from base classes, reusing code and creating "is-a" relationships.

Example: Dog and Cat inherit from Animal, getting common traits like name and eat().

Polymorphism:

Definition: Objects of different classes respond to the same method call uniquely. "Many forms."

Example: Dog and Cat both eat(), but each implements it their own way (overriding the eat() method).

Abstraction:

Definition: Simplifying complexity by showing essential features and hiding implementation details.

Example: TV remote: You use abstract functions like "change channel" without knowing TV's internal workings. Shape class with calculateArea() method, implemented differently by Circle and Rectangle.

2.

Functional Requirements of Employee Management System:

Input Employee Data: Name, ID, Salary, (Department/Bonus for Manager, Specialization/Project for Engineer).

Display Employee Data: Show all details, including type-specific info.

Store Employee Records.

Search Employee by ID.

Display All Employees.

Handle Different Employee Types: Employee, Manager, Engineer.

OOP Principles Applied to Design:

Encapsulation:

Benefit: Data protection, controlled access, simplified interaction.

Application: private attributes in Employee, Manager, Engineer; public getters/setters for controlled data access.

Inheritance:

Benefit: Code reuse, clear hierarchy, reduces redundancy.

Application: Manager, Engineer inherit from Employee, inheriting common attributes and methods; extending with specific ones.

Polymorphism:

Benefit: Uniform handling of different employee types, flexible display.

Application: virtual displayDetails() in Employee, overridden in Manager, Engineer. EmployeeManagementSystem can call displayDetails() on any employee type and get the correct output.

Abstraction:

Benefit: Simplified system view, focus on essential actions.

Application: Employee class as abstract representation of employees; users interact with employee objects without needing to know low-level details.