Module 1 - Complete Revision: Data Structures and Algorithms

1. Introduction

This module introduces the foundations of data structures, covering abstraction, encapsulation, data types, interfaces, and abstract data types (ADTs). These concepts help in designing efficient and scalable systems.

2. Separation of Concerns

Ensures that a system is divided into distinct modules, each handling a specific responsibility.
Example: A web application has Frontend (UI/UX), Backend (Business Logic), and Database (Data Management).

3. Benefits of Separation of Concerns

- Easier Maintenance
- Scalability
- Improved Collaboration
- Better Code Reusability

4. Data Abstraction

Hiding the implementation details and showing only the relevant features.

Example: A car dashboard shows speed and fuel level but hides engine mechanics.

5. Data Encapsulation

Restricting direct access to data to protect it from unauthorized modification.

Example: A Bank Account class with private balance and public deposit/withdraw methods.

6. Difference: Abstraction vs Encapsulation

- Abstraction hides complexity (What the object does).
- Encapsulation protects data (How the object does it).

7. Data Types

- Integer (int) - Whole numbers
- Float (float, double) - Decimal numbers
- Character (char) - Single letter
- String (str) - Sequence of characters
- Boolean (bool) - True/False

8. Data Representation

- Integer 5 -> Binary: 101
- Character 'A' -> ASCII: 65
- Float 3.14 -> IEEE-754 representation

9. Interface vs Implementation

- Interface: Defines WHAT a system should do without specifying HOW.
- Implementation: Actual execution of the interface's functions.

Example: Remote Control (Interface) vs Circuit Board (Implementation).

10. Abstract Data Types (ADTs)

- ADTs define operations on data without specifying how they are implemented.
- Example: A List ADT provides add(), remove(), search(), but does not define whether it is implemented using an array or linked list.

11. Common ADTs

- List - Ordered collection of elements
- Stack - LIFO (Last In, First Out)
- Queue - FIFO (First In, First Out)
- Deque - Double-ended queue
- Set - Unique elements only
- Map (Dictionary) - Key-value pairs

12. Case Study: Implementing Abstraction in a Retail Inventory System

- Problem: ShopEase needs an inventory system.
- Solution: Use SoC, Encapsulation, and ADTs.
- Outcome: Secure, scalable, and efficient inventory management.

13. Summary

- SoC ensures modular and maintainable software.
- Abstraction hides implementation, while Encapsulation protects data.
- Data Types vs Representation impacts efficiency and memory usage.
- Interface vs Implementation differentiates structure from function.
- ADTs define reusable data structures for programming.

14. Self-Assessment Questions

1. What is Separation of Concerns, and why is it important?
2. How does Encapsulation improve software security?
3. Explain Data Abstraction with a real-world example.
4. What is the difference between Data Types and Data Representation?
5. Why do we need Abstract Data Types (ADTs) in software development?