Q2: What we mean by coding against interface rather than class?
Coding against interface rather than class means designing code to depend on an interface (a contract of methods and properties) instead of relying on a concrete class. This allows greater flexibility and easier testing because any class that implements the interface can be used interchangeably.

What we mean by code against abstraction not concreteness?
This means we should write code that relies on abstractions (interfaces or abstract classes) rather than specific implementations. Abstractions define *what* should be done, while concrete classes define *how* it is done. Depending on abstractions reduces coupling and increases maintainability and scalability.

Q3: What is abstraction as a guideline and how we can implement this through what we have studied?
Abstraction as a guideline means focusing on exposing only the necessary details of an object while hiding the implementation complexity. It allows developers to think about *what* an object does instead of *how* it performs the task.
We can implement abstraction through:

- **Interfaces** → defining contracts without implementation.
- **Abstract classes** → providing partial implementation and enforcing rules for derived classes.
- **Encapsulation** → exposing only relevant members and hiding internal details.

This ensures a cleaner, more modular, and more maintainable system design.