Part 01

1) Question: Why is it better to code against an interface rather than a concrete class?

Coding against an interface increases flexibility and reusability. It allows swapping implementations without modifying client code.

2) Question: When should you prefer an abstract class over an interface?

Use an abstract class when shared implementation is needed. Use it when you want to define non-abstract methods along with abstract ones.

3) Question: How does implementing IComparable improve flexibility in sorting?

It provides a standard way to define custom sorting logic. It simplifies sorting collections without additional comparators.

4) Question: What is the primary purpose of a copy constructor in C#?

A copy constructor creates a new object as a deep copy of an existing object. It ensures independent copies.

5) Question: How does explicit interface implementation help in resolving naming conflicts?

It allows separate implementations of methods from multiple interfaces with the same name.

6) Question: What is the key difference between encapsulation in structs and classes?

Structs are value types stored on the stack, while classes are reference types stored on the heap. Structs have less overhead.

7) Question: What is abstraction as a guideline, and what's its relation with encapsulation?

Abstraction hides implementation details and exposes essential features. Encapsulation enforces data hiding and integrates abstraction.

8) Question: How does constructor overloading improve class usability?

It allows creating objects with varying levels of initialization. It simplifies the creation process in different scenarios.

Part 02

2) Question: What do we mean by coding against an interface rather than a class?

Coding against an interface means relying on the abstraction (the interface) rather than the implementation (a class). It allows the code to work with any class that implements the interface, increasing flexibility and decoupling.

Question: What do we mean by coding against abstraction, not concreteness?

Coding against abstraction means designing your code to depend on abstract layers (like interfaces or abstract classes) instead of concrete implementations. This reduces dependencies and makes the system easier to extend and maintain.

3) Question: What is abstraction as a guideline, and how can we implement this through what we have studied?

Abstraction as a guideline means focusing on essential features while hiding implementation details. We implement this using interfaces and abstract classes, which allow defining contracts and shared behavior without exposing internal logic.