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

Coding against an **interface** makes your code more flexible, decoupled, testable, and easier to extend. Coding against a **concrete class** creates rigid and tightly coupled code.

2-When should you prefer an abstract class over an interface?

- **Prefer abstract class** when you need **shared implementation, common state, or a single base class hierarchy**.
- **Prefer interface** when you need **multiple inheritance, flexibility, or just to define a contract without shared code**.

3-How does implementing IComparable improve flexibility in sorting?

1. It defines a **natural ordering** for the objects (e.g., products sorted by price).
2. Sorting methods (Array.Sort, List<T>.Sort) can use it **without extra logic**.
3. The comparison logic is **encapsulated inside the class**, so changes are centralized.
4. It allows the objects to be **easily reused in different contexts** where sorting is needed.

4-What is the primary purpose of a copy constructor in C#?

The primary purpose of a copy constructor in C# is to create a new object as a copy of an existing object, duplicating its values.

5-How does explicit interface implementation help in resolving naming conflicts?

Explicit interface implementation resolves naming conflicts by allowing a class to provide separate implementations for methods with the same name from different interfaces, ensuring each interface call is directed to the correct method.

6-What is the key difference between encapsulation in structs and classes?

Both structs and classes support encapsulation by using private fields and public properties. The key difference is that **classes** (reference types) support inheritance and polymorphism in addition to encapsulation, while **structs** (value types) only provide encapsulation without inheritance support.

7-what is abstraction as a guideline, what's its relation with encapsulation?

- **Abstraction** = focuses on *what an object does*, hiding implementation details (via abstract classes or interfaces).
- **Encapsulation** = focuses on *how data is hidden and protected*, grouping fields and methods together and controlling access (via private fields + properties).
- Relation: Abstraction hides *implementation*, Encapsulation hides *data*.

8-How does constructor overloading improve class usability?

Constructor overloading improves class usability by providing flexibility, ease of use, default values, and code reuse. It allows creating objects

with different levels of information, making the class more user-friendly and reducing errors.

9-What we mean by coding against interface rather than class ? and if u get it so What we mean by code against abstraction not concreteness ?

- **Coding against interface, not class** → depend on an interface (contract), not a specific implementation.
- **Coding against abstraction, not concreteness** → depend on abstract definitions (interfaces/abstract classes) instead of concrete classes.
- Benefit: more flexible, loosely coupled, and easier to extend/change.

10-What is abstraction as a guideline and how we can implement this through what we have studied ?

Abstraction as a guideline = focus on *what* an object does, not *how*.

Implemented in C# through:

1. **Abstract classes** (abstract methods + inheritance).
2. **Interfaces** (define contracts, multiple implementations).

Difference: Encapsulation hides *data*, Abstraction hides *implementation*.