
Q1: Why is it better to code against an interface rather than a concrete class?
It makes the code **flexible and loosely coupled**. You can swap implementations without changing the code that uses them.

Q2: When should you prefer an abstract class over an interface?
When you need to share **common behavior (shared code)** or maintain **state** across derived classes.
Interfaces define *what to do*, abstract classes can also provide *partial how to do it*.

Q3: How does implementing IComparable improve flexibility in sorting?
It allows you to define a **comparison rule** (e.g., by Price, by Name) inside the class itself.
Then, built-in methods like Array.Sort() or List.Sort() can sort your objects directly.

Q4: What is the primary purpose of a copy constructor in C#?
To create a **new object as a copy of an existing one** with the same values.
Useful when you need a separate clone instead of referencing the same object.

Q5: How does explicit interface implementation help in resolving naming conflicts?
It lets you implement two methods with the same name (from different interfaces) and control which one gets called **only when accessed through that specific interface reference**.

Q6: What is the key difference between encapsulation in structs and classes?
Structs are **value types** (copied by value).
Classes are **reference types** (copied by reference).
Both support encapsulation (hiding data via properties), but they differ in **memory management and copying behavior**.

Q7: What is abstraction as a guideline, what's its relation with encapsulation?
Abstraction = Hiding *complex details* and showing only the *essential features* (e.g., a "Print" button hides printer details).

Encapsulation = Hiding *internal data* and controlling access via properties/methods.
Relation: Abstraction = hiding *what happens*, Encapsulation = hiding *how data is stored*.

Q8: How do default interface implementations affect backward compatibility in C#?
Adding a new method to an interface doesn't break old classes, because they can fall back on the **default implementation**.
This prevents breaking existing code.

Q9: How does constructor overloading improve class usability?
It gives **flexibility** when creating objects, allowing you to pass different sets of data.
Example: You can create a Book with just a title, with title + author, or with no arguments at all.
