Part01

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

A: Because it gives more flexibility and allows changing the implementation without changing the main code.

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

A: When you have common code that should be shared between different classes.

3. How does implementing IComparable improve flexibility in sorting?

A: It allows us to define our own way of sorting, like by price or by name.

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

A: To create a new independent copy of an object so changes don’t affect the original one.

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

A: It allows writing separate implementations for methods with the same name from different interfaces.

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

A: Both provide encapsulation, but struct is a Value Type while class is a Reference Type.

7. What is abstraction as a guideline, what’s its relation with encapsulation?

A: Abstraction means focusing on what the object does, not how it does it. Encapsulation means hiding the details and exposing only what is needed. They work together.

8. How do default interface implementations affect backward compatibility in C#?

A: They allow adding new methods to an interface without breaking old code, because a default implementation is provided.

9. How does constructor overloading improve class usability?

A: It makes creating objects easier by providing multiple ways depending on the available data.

---

Part02

1. What we mean by coding against interface rather than class? And what we mean by code against abstraction not concreteness?

A: It means writing code that depends on an interface or an abstract class instead of a concrete class. This gives more flexibility and makes it easy to change the implementation.

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

A: Abstraction as a guideline means focusing on the essential behavior and hiding the details. We can implement this using abstract classes and interfaces.

---

Bonus

2. What is operator overloading?

A: Operator overloading means giving operators like (+, -, \*, ==) a custom behavior for our own classes or structs. It makes the code easier to read, for example using c1 + c2 instead of c1.Add(c2).