**Interfaces vs Classes**  
Coding against interfaces promotes **loose coupling** and **polymorphism**, allowing for **swappable implementations** without affecting consumers.

**Abstract Classes vs Interfaces**

* Use **abstract classes** when sharing a common implementation.
* Use **interfaces** for defining contracts across **unrelated types**.

**IComparable Benefit**  
Enables **native sorting support** using standard library methods like Array.Sort().

**Copy Constructor Purpose**  
Ensures **deep copying** by creating new object instances for **reference-type fields**.

**Explicit Interfaces**  
Resolves **naming conflicts** when implementing multiple interfaces with **identical method names**.

**Struct vs Class Encapsulation**

* **Structs** are **value types** allocated on the **stack**.
* **Classes** are **reference types** allocated on the **heap**.

**Abstraction & Encapsulation**

* **Abstraction** exposes **essential features**.
* **Encapsulation** hides **implementation details**.
* They **complement each other** in software design.

**Default Interfaces**  
Allow adding new members to interfaces **without breaking existing implementations**.