**Abstract Classes vs Interfaces in C#**

1. **Working Theory (keywords & concepts)**

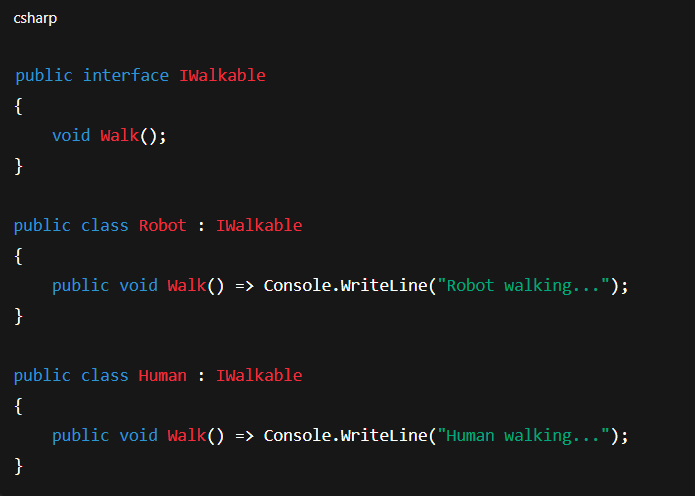
|  |  |  |
| --- | --- | --- |
| Feature | Abstract Class | Interface |
| Keyword | abstract class | interface |
| Instantiation | Cannot be instantiated directly (new not allowed). | Cannot be instantiated directly (new not allowed). |
| Inheritance | A class can inherit **only one abstract (or base) class**. | A class can implement **multiple interfaces**. |
| Members allowed | - Fields- Properties- Methods (abstract + normal)- Constructors- Access modifiers (public, protected, etc.) | - Only method/property/indexer/event signatures (until C# 8).- Since C# 8: **default implementations** allowed. |
| Implementation | Abstract methods **must be overridden** in derived classes. Normal methods can be used as-is. | All interface members **must be implemented** in the class (unless there’s a default implementation). |
| Access modifiers | Can use public, protected, internal, private. | All members are public by default (before C# 8). |
| State (fields/data) | ✅ Can contain fields (data members). | ❌ Cannot contain fields (no state). |
| Constructors | ✅ Can define constructors (called by derived classes). | ❌ Cannot define constructors. |
| Polymorphism | Used for **is-a** relationships (e.g., Dog is an Animal). | Used for **can-do** relationships (e.g., Dog can Swim, can Run). |
| Example use | When you want to provide **shared code + force subclasses to implement specific methods**. | When you want to define a **contract** that multiple classes (even unrelated ones) must follow. |

**Practical Examples:**

**Abstract Class**

****

**Interface**

****

**✅ Rule of Thumb**

* Use **abstract class** if:
  + You want to provide **shared logic** + force subclasses to implement some methods.
  + Your design fits an **is-a** relationship.
* Use **interface** if:
  + You want to define a **capability** or **contract** that many classes (possibly unrelated) can follow.
  + You need **multiple inheritance** of behaviors.

👉 example analogy:

* **Abstract class** = “Parent blueprint” (all Cars are Vehicles, they inherit wheels/engine).
* **Interface** = “Ability contract” (a Car **can drive**, a Drone **can fly**, a Fish **can swim** — but they may be totally unrelated).

✅ so that’s the abstract vs interface difference in depth.