**Snippet 09 — Interfaces**

* **Code Recap**

public interface IMyInterface

{

    void MyMethod();

}

public class MyClass : IMyInterface

{

    public void MyMethod()

    {

        Console.WriteLine("Hello");

    }

}

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

**interface**

* Defines a **contract**: a list of methods/properties/events that implementing classes must provide.
* Cannot contain implementation (until C# 8 added default interface methods).
* Cannot have fields or constructors.
* All members are **public** by default.

**Implementing an Interface**

public class MyClass : IMyInterface

{

public void MyMethod() // must match the interface signature

{

Console.WriteLine("Hello");

}

}

* A class that claims to implement an interface must provide **all required members**.
* Multiple interfaces can be implemented by one class.
* public class Car : IDriveable, IRepairable { ... }

**Why Interfaces?**

* Promote **loose coupling** (you depend on contracts, not concrete classes).
* Enable **multiple inheritance of behavior** (since C# doesn’t allow multiple base classes).
* Essential for **dependency injection & unit testing**.

💡 Example:  
Instead of coding against a concrete class:

Printer printer = new Printer();

You code against an interface:

IPrinter printer = new ConsolePrinter();

This allows you to swap implementations easily (e.g., MockPrinter for tests).

**🛠️ Practical Example**

public interface IShape

{

void Draw();

}

public class Circle : IShape

{

public void Draw() => Console.WriteLine("Drawing a Circle");

}

public class Square : IShape

{

public void Draw() => Console.WriteLine("Drawing a Square");

}

class Program

{

static void Main()

{

IShape shape1 = new Circle();

IShape shape2 = new Square();

shape1.Draw(); // Drawing a Circle

shape2.Draw(); // Drawing a Square

}

}

**Output**

Drawing a Circle

Drawing a Square

**🔧 Extras**

* **Naming convention**: Interfaces start with I (IEnumerable, IDisposable).
* **Multiple inheritance**: A class can only inherit from **one base class**, but can implement **many interfaces**.
* **Default interface methods (C# 8+)**: You can provide method bodies in interfaces, but only in rare cases.
* **Real-world analogy**: An interface is like a **job contract** — it lists the tasks you must do, but doesn’t tell you *how* to do them.

✅ snippet 09 explained: now you understand **interfaces, how to implement them, and why they’re important**.

👉 do you want me to also show you a snippet where a **class inherits from an abstract class AND implements an interface at the same time** (to see how they work together)?