**Snippet 08 — Abstract Classes and Methods**

* **Code Recap**

public abstract class BaseClass

{

    public abstract void MyMethod();

}

public class MyClass : BaseClass

{

    public override void MyMethod()

    {

        Console.WriteLine("Hello");

    }

}

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

**abstract class**

* Declared with the keyword abstract.
* Cannot be instantiated directly (❌ new BaseClass() is not allowed).
* Serves as a **blueprint** for subclasses.
* Can contain:
  + **abstract methods** (no body, must be implemented in derived classes).
  + **normal methods** (with body, optional to override).

**abstract method**

* Declared with abstract.
* No implementation (just signature).
* Must be overridden in derived classes.

Example:

public abstract class Animal

{

    public abstract void Speak(); // must be implemented

}

**override**

* Used in the derived class to **provide implementation** for an abstract (or virtual) method.
* Compiler ensures signatures match.

Example (from snippet):

public class MyClass : BaseClass

{

    public override void MyMethod()

    {

        Console.WriteLine("Hello");

    }

}

**✅ Why use Abstract Classes?**

* When you want a **common base definition** but you don’t want to allow creating objects of that base type.
* Example:
  + Shape (abstract) with abstract Draw().
  + Circle, Square, Triangle must each override Draw().

**Practical (from scratch)**

1. **Define abstract base: Create a file “Shape.cs” and write this code.**

public abstract class Shape

{

    public abstract void Draw();  // no body

}

1. **Concrete classes: Write this code in Program.cs.**

public class Circle : Shape

{

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

}

public class Square : Shape

{

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

}

1. **Program.cs:**

class Program

{

    static void Main()

    {

        Shape shape1 = new Circle();

        Shape shape2 = new Square();

        shape1.Draw();  // Drawing a circle

        shape2.Draw();  // Drawing a square

    }

}

**Output**

Drawing a circle

Drawing a square

**🔧 Extras**

* **abstract vs virtual**
  + abstract → **must** be overridden.
  + virtual → may be overridden (optional).
* **abstract vs interface**
  + Abstract class → can have fields, constructors, and both abstract + concrete methods.
  + Interface → only defines signatures, no implementation (until C# 8 default interface methods).
* **Polymorphism in action**
  + You can store different subclass objects in the same base type variable (Shape shape = new Circle()), and at runtime the correct method executes.

✅ snippet 08 explained: you now understand **abstract classes, abstract methods, and override** — one of the core pillars of OOP in C#.