**Snippet 05 — Access Modifiers + Method Accessibility**

* **Code Recap**

public class MyClass

private

protected

internal

file

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

**What are Access Modifiers?**

They control **visibility / accessibility** of classes, methods, and members.  
Think of them as *doors* that decide who can enter and use something.

**1. public**

* Visible **everywhere** (any project, any assembly).
* Most open level of access.
* Used for APIs, libraries, methods meant to be used by other parts of the program.

Example:

// Example for Public Access Modifiers

public class Car

{

    public void Drive() => Console.WriteLine("Driving");

}

class Program

{

    static void Main()

    {

        var action = new Car();

        action.Drive();

    }

}

**2. private**

* Visible **only inside the same class**.
* Default if you don’t specify an access modifier on class members.
* Best for *internal details* (helpers, backing fields).

Example:

class Engine

{

    private void Ignite() => Console.WriteLine("Ignited");

    public void InClassCall()

    {

        // ✅ Allowed: private method can be called inside the same class

        Ignite();

    }

}

class Program

{

    static void Main()

    {

        Engine engine = new Engine();

        // engine.Ignite(); ❌ // Not allowed, because Ignite() is private

        // ✅ Allowed, calls private Ignite() internally

        engine.InClassCall();

    }

}

→ Cannot call Ignite() outside Engine.

**3. protected**

* Visible **inside the same class** and **derived (child) classes**.
* Perfect for creating **base classes** that hide implementation but allow extensions.

Example:

public class MyClass

{

    protected void MyMethod() => Console.WriteLine("Hello, I am from Protected Method.");

}

public class Child : MyClass

{

    public void Call() => MyMethod(); // ✅ allowed

}

class Program

{

    static void Main()

    {

        Child child = new Child();

        child.Call();

    }

}

→ MyMethod() is not visible outside MyClass, but Child can use it.

**4. internal**

* Visible **only inside the same assembly** (project).
* Not accessible from other projects unless you add InternalsVisibleTo.

💡 Example:

internal class Helper { }

→ Usable anywhere inside the current project, but invisible outside.

**5. file (C# 11+)**

* New modifier.
* Restricts visibility to the **same source file** only.
* Great for small helper types that should never “leak” outside the file.

💡 Example:

file class FileScopedHelper { }

→ Only usable inside that .cs file.

**✅ Summary Table**

|  |  |
| --- | --- |
| Modifier | Visibility Scope |
| public | Everywhere (any assembly/project). |
| private | Inside the same class only. |
| protected | Inside the same class + derived classes. |
| internal | Inside the same assembly (project). |
| file | Inside the same source file only (C# 11+). |

**🛠️ Practical (from scratch)**

1. **Create project**

* dotnet new console -n Snippet05Demo
* cd Snippet05Demo

1. **Program.cs**

using System;

public class MyClass

{

    public void PublicMethod() => Console.WriteLine("Public");

    private void PrivateMethod() => Console.WriteLine("Private");

    protected void ProtectedMethod() => Console.WriteLine("Protected");

    internal void InternalMethod() => Console.WriteLine("Internal");

    // Only work in C# (11+)

    // file class FileHelper { public void Print() => Console.WriteLine("File Scoped"); }

    public void InClassCallProtectedMethod()

    {

        ProtectedMethod();

    }

}

class Child : MyClass

{

    public void Test()

    {

        // this.PrivateMethod();   // ❌ not allowed

        this.ProtectedMethod();    // ✅ allowed in derived class

    }

}

class Program

{

    static void Main()

    {

        var obj = new MyClass();

        obj.PublicMethod();    // ✅ allowed

        obj.InternalMethod();  // ✅ allowed (same assembly)

        obj.InClassCallProtectedMethod(); // Calling from within class

        // obj.ProtectedMethod(); // ❌ not allowed

        // obj.PrivateMethod();   // ❌ not allowed

    }

}

1. **Expected Output**

Public

Internal

Protected

**🔧 Extras**

* **Default modifier**:
  + For classes → internal by default.
  + For class members → private by default.
* **Best practice**: Always use the **most restrictive** access modifier possible (start private, then open only if needed).
* file is mostly for **small, helper-only classes** to avoid polluting project scope.

✅ That’s snippets 05 explained: now you understand all the **access modifiers in C#**, how they differ, and when to use them.