**Snippet 01 — SayHello method**

* **Code Recap**

// This is a Comment

/\* This is also a comment \*/

static void SayHello(string name)

{

    Console.WriteLine($"Hello {name}!");

}

SayHello("Anas");

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

**// This is a comment**

* **Meaning:** Everything after // on the same line is ignored by the compiler.
* **Why:** Used to explain the code for humans (documentation, notes, reasoning).
* **Variants:**
  + // → single-line comment
  + /\* ... \*/ → multi-line block comment
  + /// → XML documentation comment (used for auto-generated docs)

1. **Static**

* **Meaning:** Belongs to the *type itself* (the class/program) rather than an instance/object.
* **Here:** The method SayHello can be called directly without creating an object.
* **Use cases:**
  + Utility/helper methods (e.g., Math.Sqrt()).
  + Entry points (static void Main).
  + Shared data across all objects (static int counter).
* **Pitfall:** Overusing static can lead to tightly coupled, hard-to-test code. Prefer instance methods for behavior tied to an object’s state.

**Let’s go deep dive in “static” keyword in C#**

**1. What does static mean?**

The word **static** in C# means that something (a **field**, **method**, or **class**) belongs to the **type itself**, not to an object created from that type.

* Normal classes/objects → you need to new them to use their members.
* Static → you can use them **directly** without creating an object.

**2. Static Class**

A **static class** is like a normal class, but it has special restrictions:

* ✅ **Only contains static members** (fields, properties, methods).
* ✅ **Cannot be instantiated** → you cannot do new MyClass().
* ✅ **Sealed automatically** → you cannot inherit from it.
* ✅ **Cannot have instance constructors** (only a static constructor is allowed, and it runs once for the whole class).

**Example: ( Practical from scratch )**

1. **Create project**

* dotnet new console -n Snippet01Demo
* cd StaticDemo

1. **Replace Program.cs with:**

using System;

static class MathHelper

{

    public static double Pi = 3.14159;

    public static int Square(int x) => x \* x;

}

class Program

{

    static void Main()

    {

        // Usage:

        Console.WriteLine(MathHelper.Pi);          // 3.14159

        Console.WriteLine(MathHelper.Square(5));   // 25

    }

}

1. **Run**

* dotnet run

Here, MathHelper is never instantiated. You just call its methods directly.

**3. Static Method**

A **static method** belongs to the class, not to any instance.

* You call it without creating an object.
* Cannot access **instance members** (because no this exists).

**Example:**

class Greeter

{

    // Static Method.

    public static void SayHello(string name) => Console.WriteLine($"Hello {name}!");

}

// For Class Greeter.

Greeter.SayHello("Anas");

**4. Static Field / Property**

* Static fields keep **one shared value** for the entire class.
* All objects of that class share the same static field.

**Example:**

class Counter

{

    // Static Member

    public static int Count = 0;

    // Constructer.

    public Counter()

    {

        Count++; // Increase for all Instance.

    }

}

// Usage.

// For class Counter

        new Counter();

        new Counter();

        new Counter();

        new Counter();

        Console.WriteLine(Counter.Count);  // 4 (shared, not per object)

**5. Static Constructor**

* Runs **only once**, automatically, before the first static member is accessed.
* Used for initializing static data.
* No parameters allowed.

**Example:**

class Config

{

    public static string Setting;

    static Config()

    {

        Setting = "Initialized only once!";

        Console.WriteLine("Static Constructor Called.");

    }

}

// Usage

        // For Class Config

        Console.WriteLine(Config.Setting);

**6. Points to Remember**

* C# static class contains only static members.
* C# static class cannot be instantiated (new is not allowed).
* C# static class is sealed (no inheritance).
* C# static class cannot contain instance constructors.
* Static methods can only access other static members.
* Static members are shared across all instances (global to the type).
* Static constructor runs once automatically.
* Useful for **utility/helper classes** (e.g., Math, Console, DateTime).

**7. Real-World Examples**

* Console.WriteLine(...) → you never new Console().
* Math.Sqrt(25) → Math is a static class.
* DateTime.Now → Now is a static property.

⚡ **Quick Tip for memory:**  
Think of **static** as "**one copy for the whole program**," not per object.

1. **Void**

* **Meaning:** The method does **not return any value**.
* **Here:** SayHello only prints to the console, no return.
* **Alternatives:**
  + int, string, bool, custom types → method returns something.
  + Task or Task<T> → async methods.

1. **SayHello**

* **This is the method’s *name*.**
* Must follow C# naming conventions (PascalCase for methods).
* You choose descriptive names so the code is self-explanatory.
* Later, we can **call** this method by writing SayHello("James");.

1. **(string name)**

* **string**: parameter type. Only text input is accepted.
* **name**: parameter name. Acts like a variable inside the method.
* **Parameter use:** Gives flexibility. Instead of hardcoding "James", we can greet any name.
* **Variations:** You can add multiple parameters:
* static void Greet(string name, int age) { ... }

1. **{ ... } (curly braces)**

* **Block scope**: Groups statements into a method body.
* Defines where variables live and when they are destroyed.

1. **Console.WriteLine(...)**

* **Meaning:** Prints text (or other data) to the console and adds a newline.
* **Console** is a built-in class in the **System** namespace.
* **Other options:**
  + Console.Write(...) → without newline.
  + Console.ReadLine() → waits for user input.

1. **$"Hello {name}"**

* **This is an interpolated string.**
* “$” before the string allows {} placeholders inside.
* {name} → replaced by the variable’s value.
* Example:

string fruit = "Apple";

Console.WriteLine($"I like {fruit}");

// Output: I like Apple

* Without $, you’d have to write: "Hello " + name

1. **SayHello("James");**

* **Method call**: Executes the code in SayHello with argument "James".
* "James" is passed to string name, so the output is Hello James.

**🛠️ Practical (from scratch)**

1. **Create project**

* dotnet new console -n Snippet01Demo
* cd Snippet01Demo

1. **Replace Program.cs with:**

// This is a Comment

/\* This is also a comment \*/

static void SayHello(string name)

{

    Console.WriteLine($"Hello {name}!");

}

SayHello("Anas");

SayHello("Saad");

1. **Run**

* dotnet run

1. **Expected output**

* Hello Anas
* Hello Saad

1. **Variation to try**

// Taking Input from Console

static void SayHelloByConsole(string name)

{

    Console.WriteLine($"Hello {name}!");

}

SayHelloByConsole(Console.ReadLine());

→ program now asks the user to type a name at runtime.

**🔧 Extras**

* **When not to use static:** If you want each person to have their own state, you’d use a *class instance*:

using System;

class Greeter

{

    public void SayHello(string name)

    {

        Console.WriteLine($"Hello {name}!");

    }

}

class Program

{

    static void Main()

    {

        var g = new Greeter();

        g.SayHello("Anas");

        g.SayHello("Saad");

    }

}

* **Debug tip:** Set a breakpoint on Console.WriteLine(...) and watch the value of name each time.
* **Official docs:**
  + [Methods in C#](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/methods)
  + [Static classes and methods](https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-classes-and-static-class-members)

✅ That’s your first snippet fully unpacked: all keywords explained, practical example built, and extras added.