Threads in C#

🔹 What is a Thread?

- A thread is the smallest unit of execution within a process.
- It represents a line of execution in a program.
- Threads can be scheduled independently by the operating system.

🔹 Process vs. Thread

Feature	Process	Thread
Definition	Represents an application	Represents a module/unit of execution
Weight	Heavyweight	Lightweight
Memory	Each process has separate memory	Threads share common memory
Communication	Harder between processes	Easy between threads (shared space)

🔹 Thread Lifecycle in C#

State	Description
Unstarted	When a Thread instance is created but Start() is not yet called.
Runnable	When Start() is called and thread is ready to run.
Running	When the thread is actively executing. Only one thread per core executes at a time.
Not Runnable	When thread is sleeping (Sleep()), waiting (Wait()), or blocked by I/O.

State	Description
Dead (Terminated)	Thread has completed its task or exited.

🔹 Thread Class

- The Thread class is used to create and manage threads.
- It belongs to the System.Threading namespace.

csharp

```
using System.Threading;
```

🔸 Important Properties of Thread Class

Property	Description
CurrentThread	Returns the currently executing thread instance.
IsAlive	Checks if the thread is alive.
IsBackground	Gets/Sets whether a thread is background.
Name	Gets/Sets the thread name.
Priority	Gets/Sets the thread's priority.
ThreadState	Returns the current state of the thread.

🔸 Important Methods of Thread Class

Method	Description
Abort()	Terminates a thread (deprecated).
Join()	Blocks calling thread until this thread terminates.
Resume()	Resumes a suspended thread (Obsolete).

Method	Description
Sleep(int)	Suspends current thread for given milliseconds.
Start()	Starts thread execution.
Suspend()	Suspends thread (Obsolete).
Yield()	Yields execution to another thread.

🔹 Types of Threads in C#

1️⃣ Foreground Thread

- Keeps running until its work is finished, even if the Main thread exits.
- Does NOT depend on the main thread.
- Used for important/background-independent tasks.

2️⃣ Background Thread

- Terminates automatically when the Main thread ends.
- Depends on the life of the main thread.
- Used for low-priority or continuous background operations.

🔸 Code Example: Foreground vs Background Threads

Csharp

```
using System;
using System.Threading;

class Program
{
    static void Main()
    {
        Thread foregroundThread = new Thread(PrintMethod);
        foregroundThread.Name = "ForegroundThread";
        foregroundThread.IsBackground = false; // Explicitly setting foreground
        foregroundThread.Start();

        Thread backgroundThread = new Thread(PrintMethod);
        backgroundThread.Name = "BackgroundThread";
        backgroundThread.IsBackground = true; // Setting as background thread
        backgroundThread.Start();

        Console.WriteLine("Main thread ends.");
    }

    static void PrintMethod()
    {
        for (int i = 1; i <= 5; i++)
        {
            Console.WriteLine($"{Thread.CurrentThread.Name} prints {i}");
            Thread.Sleep(1000); // Sleep to simulate long task
        }
    }
}
```

◆ Output Behavior

- Foreground thread continues running even after main thread ends.
- Background thread may terminate early if main thread exits first.

🟡 What is the Default Type of a Thread?

✅ By default, every thread is a Foreground Thread unless explicitly set otherwise.

📌 Summary Table: Foreground vs Background

Feature	Foreground Thread	Background Thread
Lifetime	Continues even if Main thread exits	Ends when Main thread exits
Priority Use	Important tasks	Background tasks
Default?	✅ Yes	❌ No (must be set)
Set via Property	IsBackground = false	IsBackground = true