**Stack and Heap in C#**

In C#, **Stack** and **Heap** are two types of memory used for storing data. Understanding them is crucial for performance and memory management.

**1. Stack (LIFO - Last In, First Out)**

📌 **What is it?**

* Stack is a **fast** memory area used for storing **method execution context**, local variables, and function calls.
* It follows the **LIFO (Last In, First Out)** principle.
* Memory is **automatically managed** (popped out when a method completes).

📌 **What goes in the Stack?**  
 **Value types** (like int, char, bool, struct)  
 **Local variables** (inside a method)  
 **Method call information** (like return address)

📌 **Example of Stack Usage**

csharp

CopyEdit

void StackExample()

{

int x = 10; // Stored in Stack

int y = 20; // Stored in Stack

}

Here, x and y are **value types** and stored on the **Stack**.  
Once StackExample() completes, x and y are **removed** from memory.

**2. Heap (Dynamic Memory)**

📌 **What is it?**

* Heap is a **larger, slower** memory area used for storing **objects and reference types**.
* Unlike Stack, memory is **not automatically freed** when a function exits.
* The **Garbage Collector (GC)** reclaims unused memory when needed.

📌 **What goes in the Heap?**  
**Reference types** (class, interface, delegate, array, string)  
**Objects created using new keyword**

📌 **Example of Heap Usage**

csharp

CopyEdit

class Person

{

public string Name;

}

void HeapExample()

{

Person p1 = new Person(); // Object stored in Heap, reference in Stack

p1.Name = "Alice"; // Name is stored in Heap

}

💡 p1 (a **reference**) is stored on the **Stack**, but the Person object is in the **Heap**.  
💡 The **Garbage Collector** removes p1 when it's no longer used.

**3. Stack vs. Heap: Key Differences**

| **Feature** | **Stack** | **Heap** |
| --- | --- | --- |
| **Speed** | ✅ Fast | 🐢 Slow |
| **Memory Size** | Small | Large |
| **Data Type Stored** | Value types, local variables, method calls | Objects, reference types |
| **Allocation Type** | Automatic | Dynamic (manual with new) |
| **Memory Management** | Managed automatically | Requires Garbage Collector |
| **Access** | Sequential (LIFO) | Random access |

**4. Example with Both Stack & Heap**

csharp

CopyEdit

class Person

{

public string Name;

}

void Example()

{

int a = 10; // Stored in Stack

Person p = new Person(); // p (reference) in Stack, Object in Heap

p.Name = "Alice"; // "Alice" stored in Heap

}

* a is a **value type** → Stored in **Stack**.
* p is a **reference variable** → Stored in **Stack**, but points to an **object in Heap**.
* "Alice" (a string) → Stored in **Heap**.

**5. When to Use What?**

✅ Use **Stack** for:

* Small, short-lived variables (like int, char, bool).
* Local function variables.

✅ Use **Heap** for:

* Large, long-lived objects (like class instances, arrays).
* Objects that need to be shared across different scopes.

**6. Common Issues**

🔴 **Stack Overflow**  
Occurs when too many nested function calls or large local variables exceed Stack size.

csharp

CopyEdit

void InfiniteRecursion()

{

InfiniteRecursion(); // Stack overflow error!

}

🔴 **Memory Leak (Heap)**  
If references to Heap objects are **never released**, they consume memory indefinitely.