Linkedin Article

> الـ **Value Type**: بتتخزن في **Stack Memory**
> الـ **Reference Type**: الـ **Variable نفسه** بيتخزن في **Stack** وبيشاور على عنوان في الـ Heap

Stack Memory

- ✧ هو زي Container كده بيشتغل بنظام (Last In, First Out) **LIFO** - آخر حاجة دخلت جوا ال Container هي أول حاجة هتطلع منه
- ✧ **سريع جداً** مقارنة بال Heap
- ✧ **محدود الحجم** (حوالي 1 ميجا عادة)
- ✧ بيتمسح **بشكل تلقائي** لما الـ Method تخلص

✧

إيه اللي بيتخزن في الـ Stack؟

1. **Local Variables** من النوع Value Type
2. **Method Parameters**
3. **Return Addresses**
4. **References** للـ objects اللي في الـ Heap

Heap Memory

- ✧ **أبطأ نسبياً** في allocation & deallocation
- ✧ حجمها كبير ومحدود فنفس الوقت بـ RAM المتاح
- ✧ فيها **Random Access** علي عكس Stack
- ✧ **بتحتاج Garbage Collection** عشان ت Clean ال Memory
- ✧ **مشترك** بين كل الـ threads

إيه اللي بيتخزن في الـ Heap؟

1. **Objects** (instances of classes)
2. **Arrays** (حتى لو كانت من value types)
3. **Strings**
4. **Boxing** للـ value types

Compilation vs. Interpretation

Compilation

- ✧ الكود **كله يتترجم مرة واحدة** قبل التشغيل
- ✧ ينتج عنه **ملف منفصل** (executable)
- ✧ **أسرع في التشغيل** لأن الترجمة بتكون خلصت
- ✧ **أبطأ في التطوير** لأن لازم ت compile كل مرة

Interpretation

- ✧ الكود **يتترجم سطر بسطر** وقت التشغيل
- ✧ **مفيش ملف منفصل** - بيقرأ الكود الأصلي
- ✧ **أبطأ في التشغيل** لأن بيترجم كل مرة
- ✧ **أسرع في التطوير** - غيرت وشغلت علطول

Examples of programming languages:

```python
# Python - Pure Interpretation
print("Hello World")
```

```cpp
// C++ - Pure Compilation
#include <iostream>
```

```cpp
int main() {
    cout << "Hello World";
    return 0;
}
```

Comparison table:

METHOD	FROM	TO	USE
Implicit	Smaller Value Types	Larger Value Types	Automatic
Explicit	any Type	any Type	Manual
Convert	any Type	Value Types	Methods
Parse	String only	Value Types	Methods

C# is Managed Code:

Managed Code means the code runs under the control of a runtime environment (CLR - Common Language Runtime) that provides:

- ✧ **Automatic Memory Management**: Garbage Collector handles memory allocation/deallocation
- ✧ **Safety**: Runtime checks prevent invalid operations
- ✧ **Exception Handling**: Structured error handling
- ✧ **Security**: Code access security and verification
- ✧ **Cross-Language Integration**: *Can work with other .NET languages*

vs Unmanaged Code (like C/C++): You manually manage memory, no runtime protection.

Struct is Considered Like Class:

Because of **Similarities:** between them

- ✧ Can have fields, properties, methods, constructors
- ✧ Can implement interfaces
- ✧ Support access modifiers (public, private, etc.)
- ✧ Can have static members

Differences:

- ✧ **Struct**: Value type (stack), copied by value, no inheritance
- ✧ **Class**: Reference type (heap), copied by reference, supports inheritance

Example:

```cpp
struct Point        // Like a class but value type
{
    public int X;
    public int Y;
    public void Display() { Console.WriteLine($"({X}, {Y})"); }
}

class Person        // Reference type
{
    public string Name;
    public void Display() { Console.WriteLine(Name); }
}
```