Proof of Concept: Class vs Struct in C#

This document explains key differences between **Classes** and **Structures** in C# using a practical example. The code demonstrates various concepts to show how classes and structs behave differently in memory allocation, inheritance, method support, and more.

Point 1: Memory Allocation

- **Classes** are **reference types** and are stored on the **heap**.
- **Structs** are **value types** and are stored on the **stack**.
- Classes are **passed by reference**, while structs are **passed by value**.

Code Explanation: We initialize objects for both ClassObject and StructObject. Refer Point 1 in Program.cs

- When we manipulate a class, we work with a reference to its memory location.
- When we manipulate a struct, we are working with a copy of the data.

Result : This highlights that changes in the class affect the original object, while changes in a struct only affect the copy.

Point 2: Parameterless Constructor

- In **classes**, you can define a **parameterless constructor**.
- In **structs**, parameterless constructors were **not allowed** before C# 10.

Code Explanation: Refer Program.cs Point 2 Which demonstrates that structs cannot have a parameterless constructor

Result: This point Demonstrate we cant use parameterless constructor

Point 3: Inheritance

- **Classes** support **inheritance**, allowing you to create a new class based on an existing one.
- **Structs** do **not support inheritance**. You cannot create a child struct from a parent struct.

Code Explanation: Refer Point 3 in Program.cs , a child class (ChildClass) is derived from a parent class (ClassObject), but the same is not possible with structs.

Result: This highlights that structs are limited in terms of inheritance, unlike classes which can form a hierarchy.

Point 4: Virtual and Abstract Functions

- **Classes** can have **virtual** and **abstract** methods, enabling polymorphism (where methods can be overridden in derived classes).
- **Structs** do not support **virtual** or **abstract** methods.

Code Explanation: In the code (Program.cs Point 4), the parent class has a virtual method that can be overridden in the child class. However Struct cannot use this feature

Result: This demonstrates that classes provide more flexibility with method behavior, which is useful in polymorphic scenarios.

Structs are suitable for small, simple, and immutable data, while classes offer more flexibility and are suitable for complex, mutable objects. For More Points which I have learn from GeeksforGeeks which I have mentioned as follow :

https://www.geeksforgeeks.org/difference-between-class-and-structure-in-c-sharp/

Difference between Class and Structure

Class	**Structure**
Classes are of reference types.	Structs are of value types.
All the reference types are allocated on heap memory.	All the value types are allocated on stack memory.
Allocation of large reference type is cheaper than allocation of large value type.	Allocation and de-allocation is cheaper in value type as compare to reference type.
Class has limitless features.	Struct has limited features.
Class is generally used in large programs.	Struct are used in small programs.
Classes can contain constructor or destructor.	Structure does not contain parameter less constructor or destructor, but can contain Parameterized constructor or static constructor.
Classes used new keyword for creating instances.	Struct can create an instance, with or without new keyword.
A Class can inherit from another class.	A Struct is not allowed to inherit from another struct or class.
The data member of a class can be protected.	The data member of struct can't be protected.
Function member of the class can be virtual or abstract.	Function member of the struct cannot be virtual or abstract.

Class	Structure
Two variable of class can contain the reference of the same object and any operation on one variable can affect another variable.	Each variable in struct contains its own copy of data(except in ref and out parameter variable) and any operation on one variable can not effect another variable.