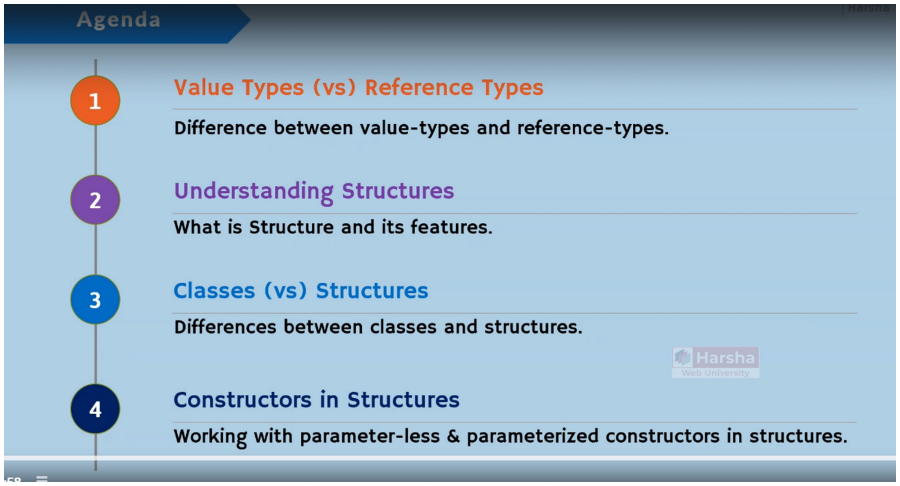
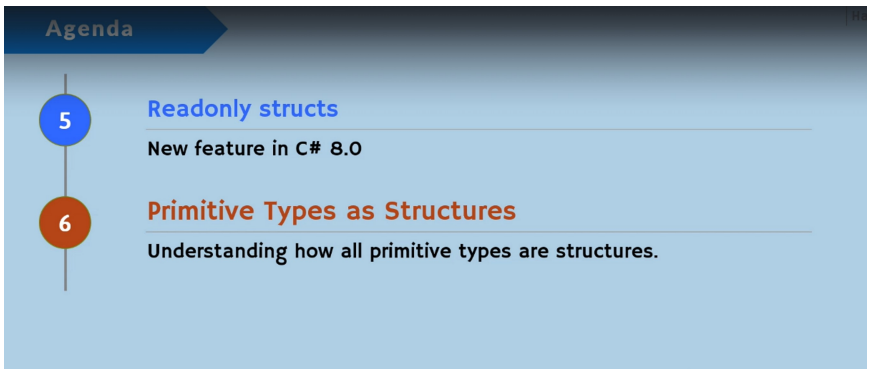
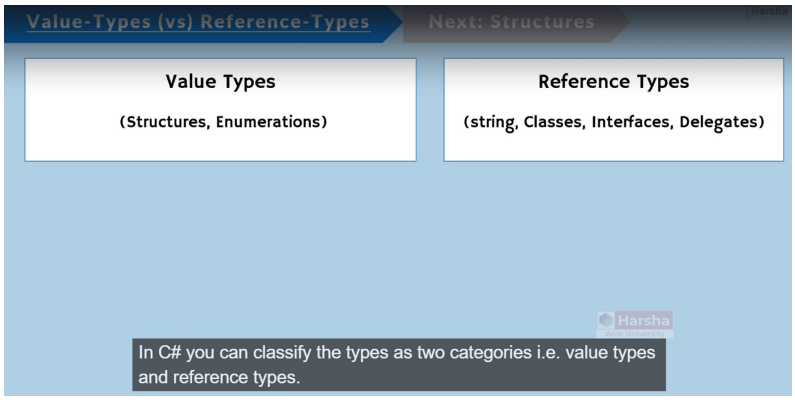
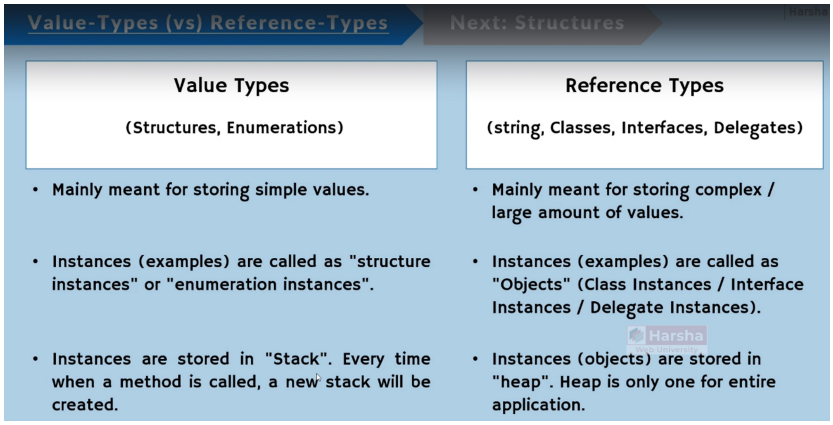
Section 14: Structures

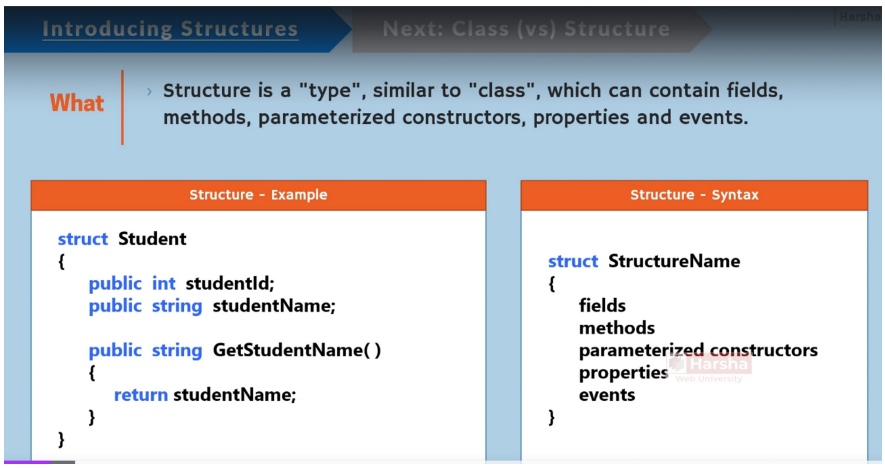
14 July 2024 13:51



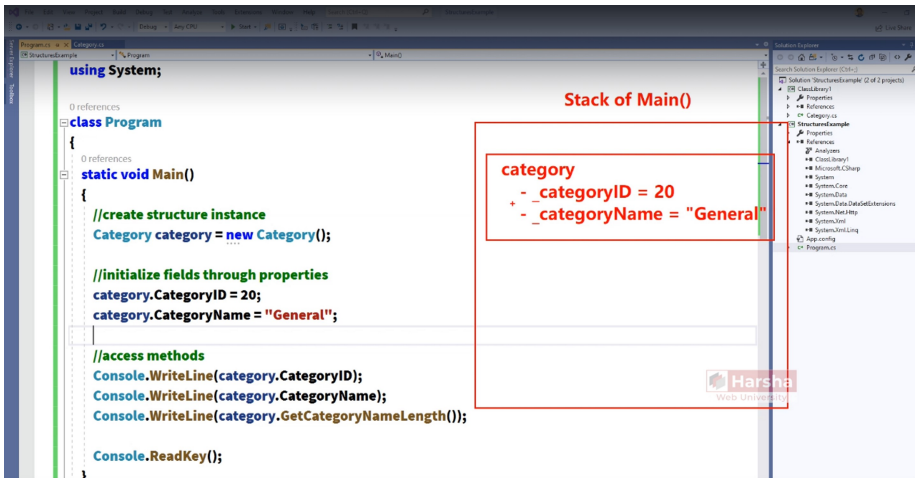

Structures are meant for limited amount of fields, may be 1 or 2. Where as classes are meant for storing complex amount of data.

When to use Class and Structure?

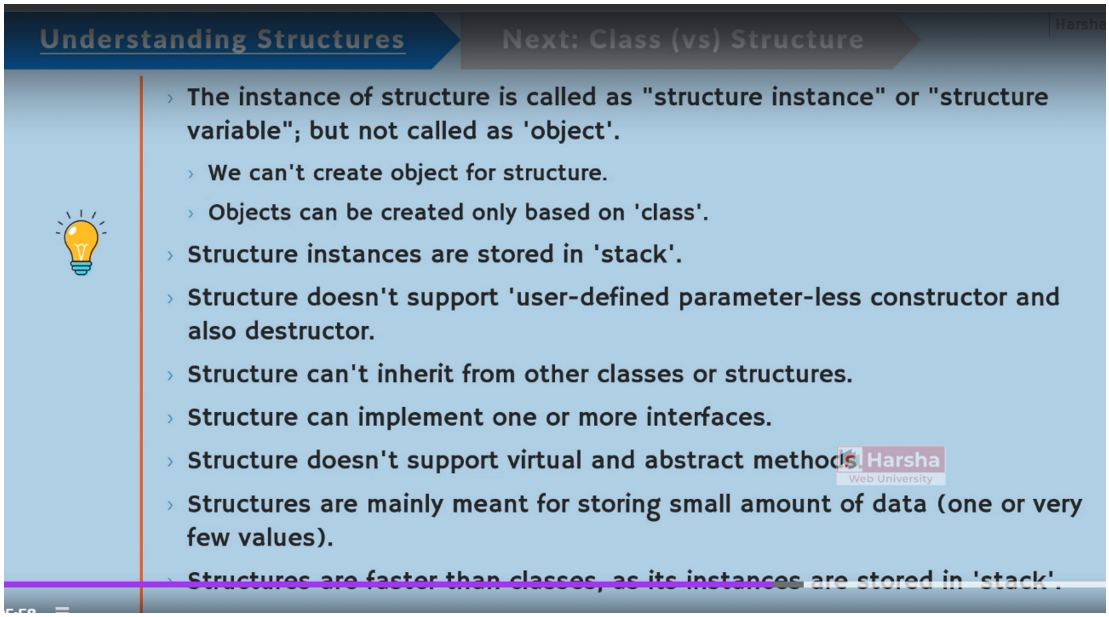
=> If you really want to store one or two fields, then you can go for structure.
because structures are little bit faster than classes. This is because; structure instances get stored in the stack. The data that is present in the stack can be accessible faster than heap.

On the other hand, if you want to really store more than two fields, then it is recommended to use classes.

Additionally, structures doesn’t support inheritance but classes support inheritance.




Structure does not 'user-defined parameter less constructor' and destructor.
However, we can write parametarized constructor and constructor overloading.




Structure does not support inheritance because it was meant for storing small amount of value.

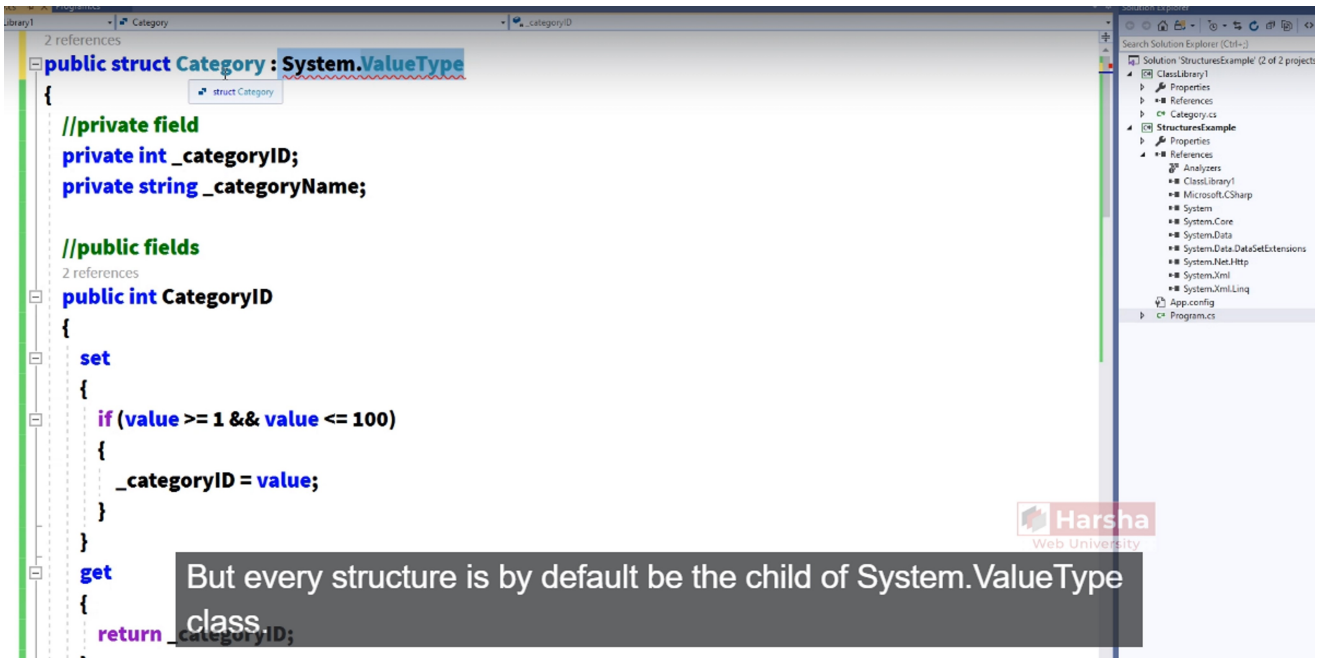
## Class (vs) Structure

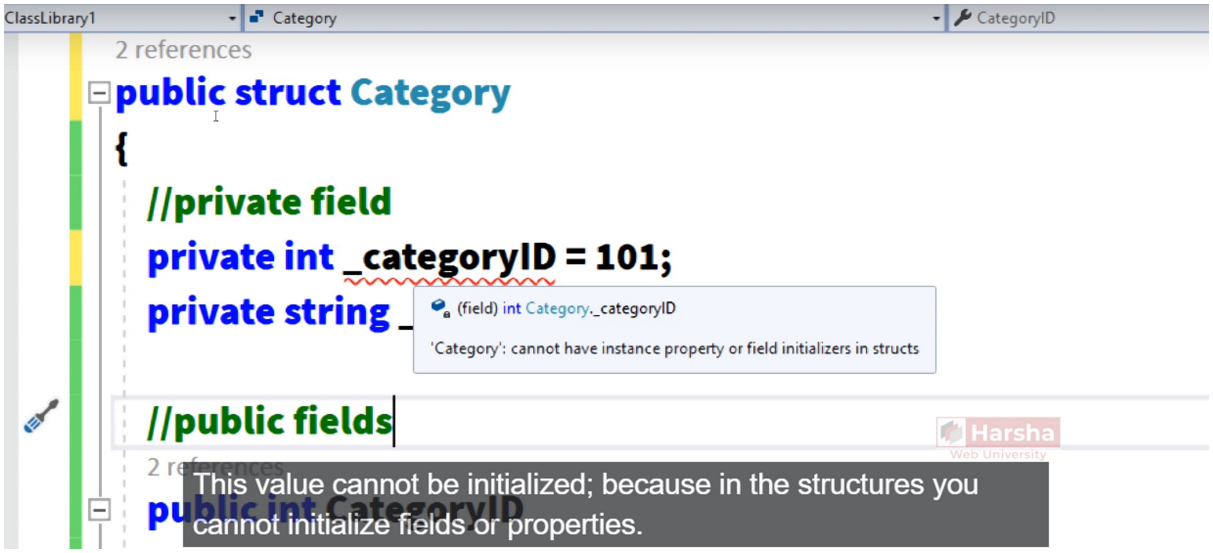
Next: Comparison Table

Harsha

| Structures | Classes |
|---|---|
| • Structures "value-types". | • Classes are "reference-types". |
| • Structure instances (includes fields) are stored in stack. Structures doesn't require Heap. | • Class instances (objects) are stored in Heap; Class reference variables are stored in stack. |
| • Suitable to store small data (only one or two values). | • Suitable to store large data (any no. of values) |
| • Memory allocation and de-allocation is faster, in case of one or two values. | • Memory allocation and de-allocation is a bit slower. |
| • Structures doesn't support Parameter-less Constructor. | • Classes support Parameter-less Constructor. |
| • Structures doesn't support inheritance (can't be parent or child). | • Classes support Inheritance. |
| • The "new" keyword just initializes all fields of the "structure instance". | • The "new" keyword creates a new object. |








## Class (vs) Structure (contd...)

Next: Comparison Table

| Structure | Classes |
|---|---|
| • Structures doesn't support abstract methods and virtual methods. | • Classes support abstract methods and virtual methods. |
| • Structures doesn't support destructors. | • Classes support destructors. |
| • Structures are internally derived from "System.ValueType".<br>System.Object → System.ValueType → Structures | • Classes are internally and directly derived from "System.Object".<br>System.Object → Classes |
| • Structures doesn't support to initialize "non-static fields", in declaration. | • Classes supports to initialize "non-static fields", in declaration. |
| • Structures doesn't support "protected" and "protected internal" access modifiers. | • Classes support "protected" and "protected internal" access modifiers. |
| • Structure instances doesn't support to assign "null". | • Class's reference variables support to assign "null". |

## Comparison Table: Structure (vs) Class

**Based on inheritance and object**

| Class Type | Can Inherit from Other Classes | Can Inherit from Other Interfaces | Can be Inherited | Can be Instantiated |
|---|---|---|---|---|
| Normal Class | Yes | Yes | Yes | Yes |
| Abstract Class | Yes | Yes | Yes | No |
| Interface | No | Yes | Yes | No |
| Sealed Class | Yes | Yes | No | Yes |
| Static Class | No | No | No | No |
| Structure | No | Yes | No | Yes |



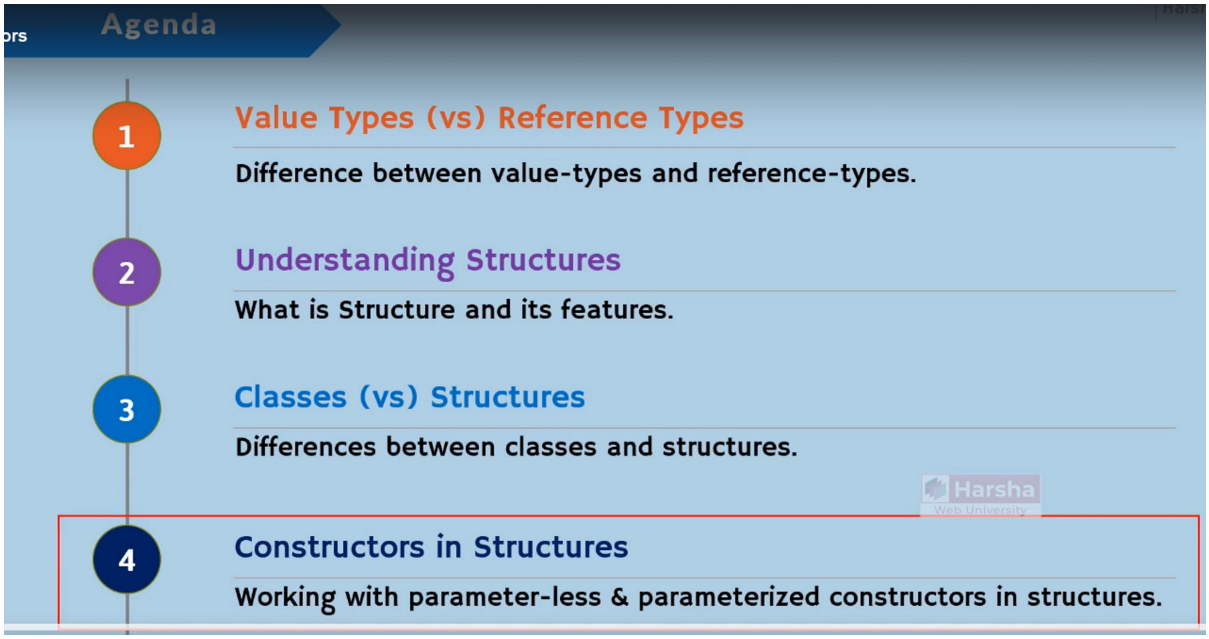
## Comparison Table: Structure (vs) Class

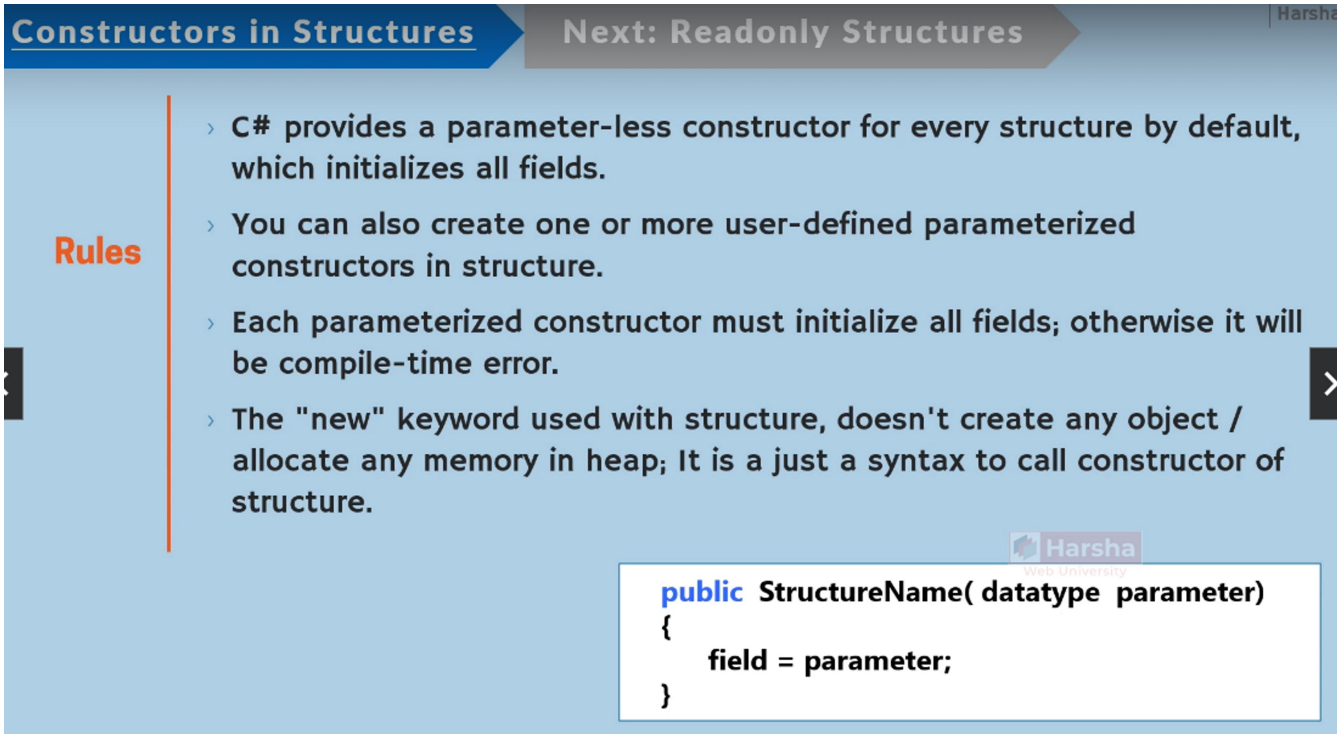


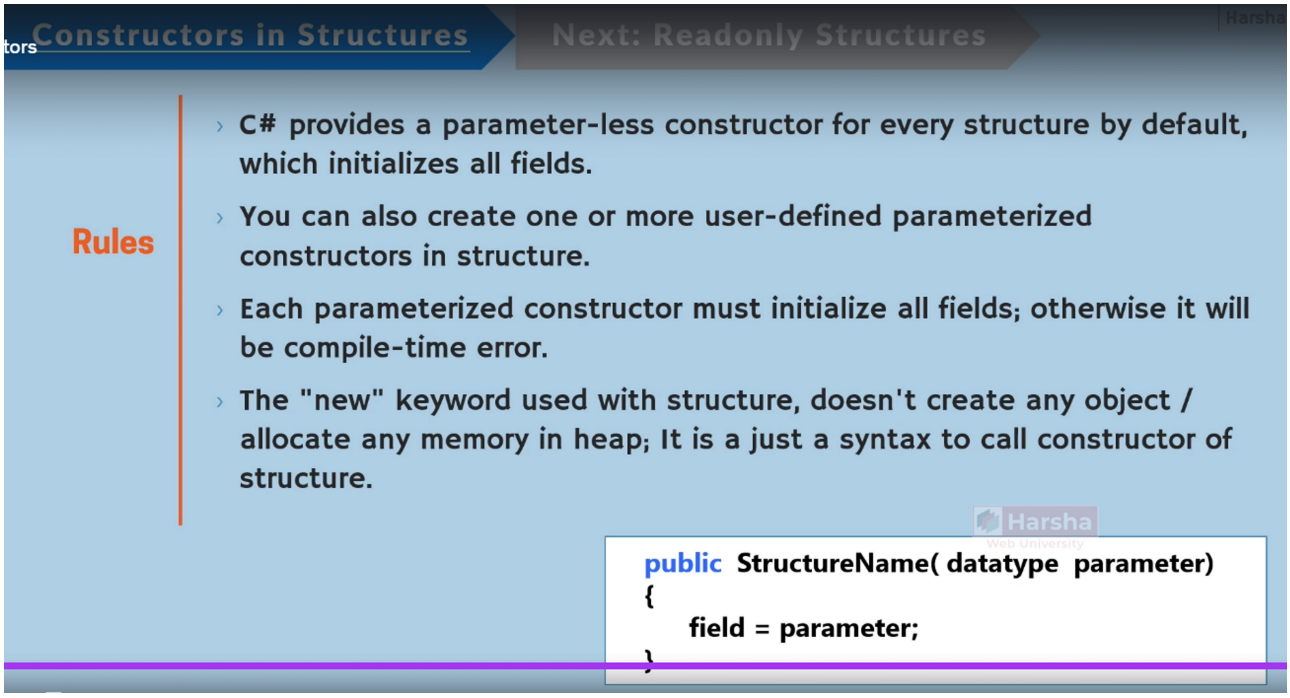
| Type | 1. Non-Static Fields | 2. Non-Static Methods | 3. Non-Static Constructors | 4. Non-Static Properties | 5. Non-Static Events | 6. Non-Static Destructors | 7. Constants |
|---|---|---|---|---|---|---|---|
| Normal Class | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Abstract Class | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Interface | No | No | No | No | Yes | No | No |
| Sealed Class | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Static Class | No | No | No | No | No | No | Yes |
| Structure | Yes | Yes | Yes | Yes | Yes | No | Yes |

## Comparison Table: Structure (vs) Class

| Type | 8. Static Fields | 9. Static Methods | 10. Static Constructors | 11. Static Properties | 12. Static Events | 13. Virtual Methods | 14. Abstract Methods | 15. Non-Static Auto-Impl Properties | 16. Non-Static Indexers |
|---|---|---|---|---|---|---|---|---|---|
| Normal Class | Yes | Yes | Yes | Yes | Yes | Yes | No | Yes | Yes |
| Abstract Class | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes |
| Interface | No | No | No | No | No | No | Yes | Yes | No |
| Sealed Class | Yes | Yes | Yes | Yes | Yes | No | No | Yes | Yes |
| Static Class | Yes | Yes | Yes | Yes | Yes | No | No | No | No |
| Structure | Yes | Yes | Yes | Yes | Yes | No | No | Yes | Yes |

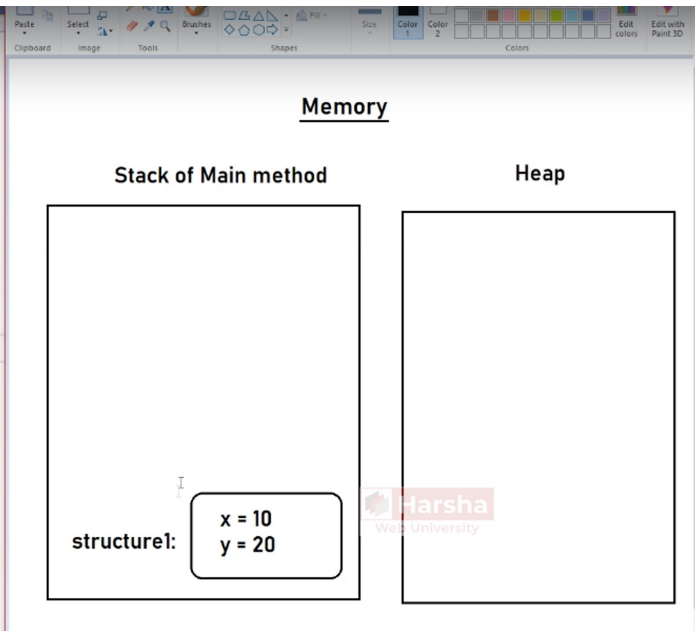

Agenda
1
Value Types (vs) Reference Types
Difference between value-types and reference-types.
2
Understanding Structures
What is Structure and its features.
3
Classes (vs) Structures
Differences between classes and structures.
Harsha
Web University
4
Constructors in Structures
Working with parameter-less & parameterized constructors in structures.


Constructors in Structures
Next: Readonly Structures
Rules
C# provides a parameter-less constructor for every structure by default, which initializes all fields.
You can also create one or more user-defined parameterized constructors in structure.
Each parameterized constructor must initialize all fields; otherwise it will be compile-time error.
The "new" keyword used with structure, doesn't create any object / allocate any memory in heap; It is a just a syntax to call constructor of structure.
Harsha
Web University
public StructureName( datatype parameter)
{
field = parameter;
}

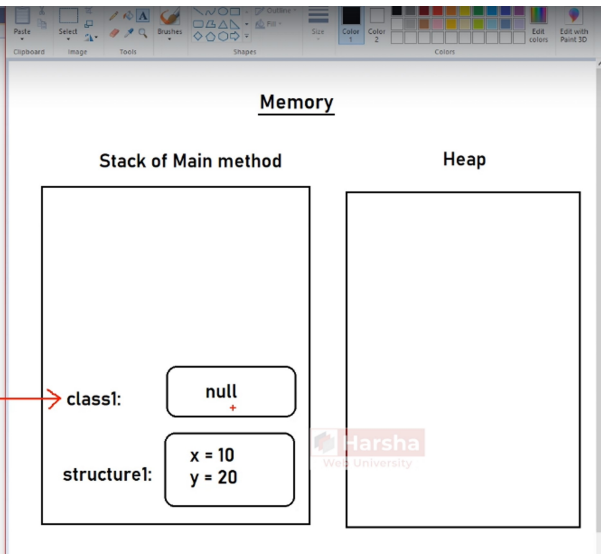

tors
Constructors in Structures
Next: Readonly Structures
Rules
C# provides a parameter-less constructor for every structure by default, which initializes all fields.
You can also create one or more user-defined parameterized constructors in structure.
Each parameterized constructor must initialize all fields; otherwise it will be compile-time error.
The "new" keyword used with structure, doesn't create any object / allocate any memory in heap; It is a just a syntax to call constructor of structure.
Harsha
Web University
public StructureName( datatype parameter)
{
field = parameter;
}


(vs) Class Practically
C#
Structures
Harsha Vardhan
UI Expert
.NET Expert
Module Lead
Corporate Instructor
Harsha
Web University

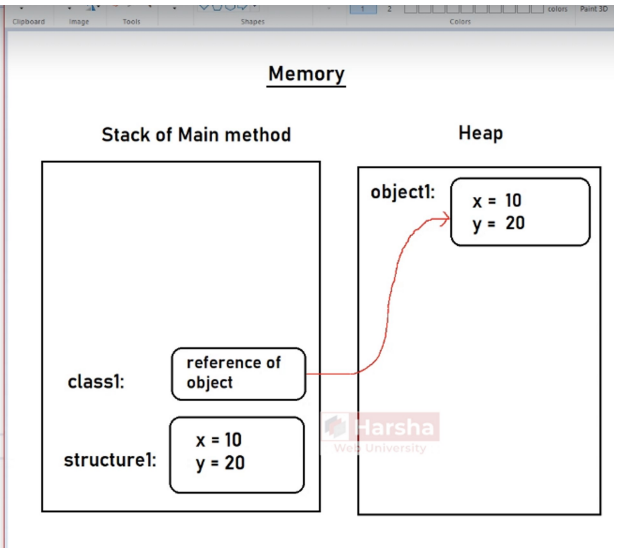
```
0 references
class Program
{
    0 references
    static void Main()
    {
        //create structure instance
        Structure1 structure1;
        structure1.x = 10;
        structure1.y = 20;


    }
}
```



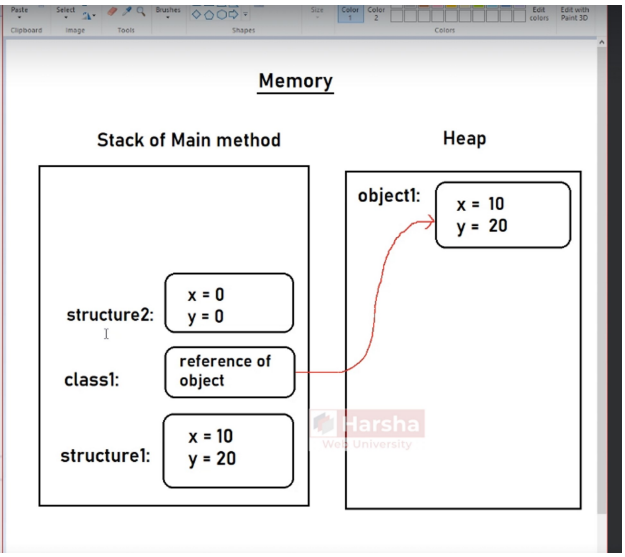

```
0 references
class Program
{
    0 references
    static void Main()
    {
        //create structure instance
        Structure1 structure1;
        structure1.x = 10;
        structure1.y = 20;

        //create object based on class
        Class1 class1;

    }
}
```



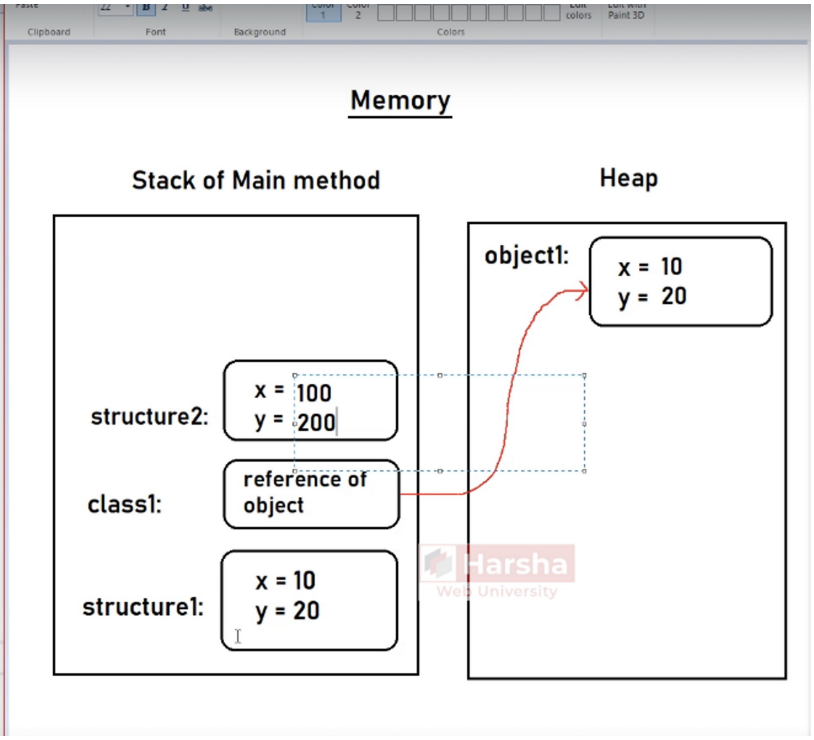

```
0 references
class Program
{
    0 references
    static void Main()
    {
        //create structure instance
        Structure1 structure1;
        structure1.x = 10;
        structure1.y = 20;

        //create object based on class
        Class1 class1;
        class1 = new Class1();
        class1.x = 10;
        class1.y = 20;


    }
}
```

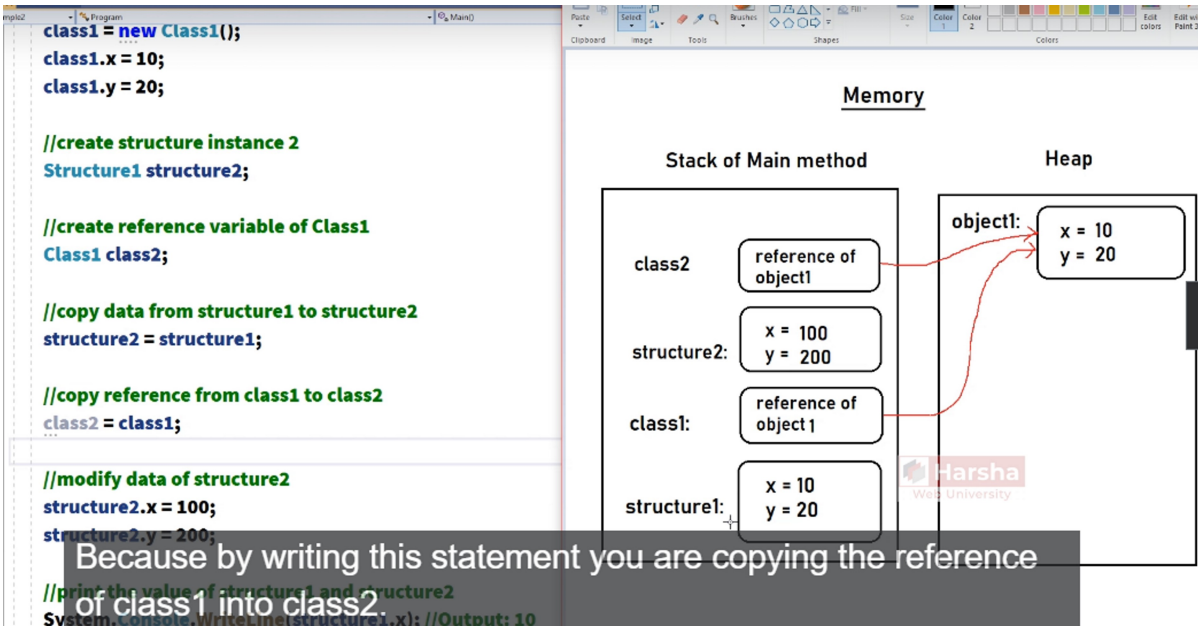


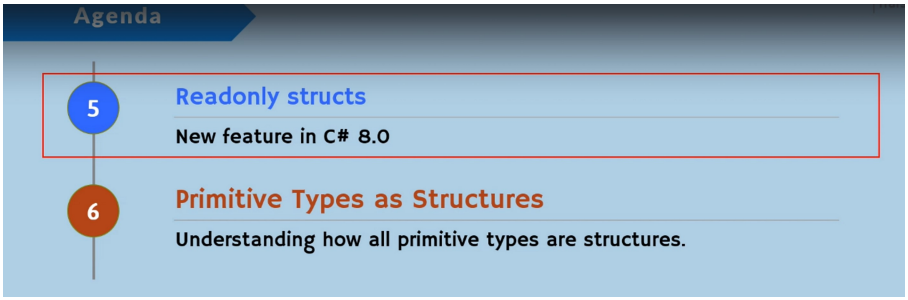
```
0 references
class Program
{
    0 references
    static void Main()
    {
        //create structure instance
        Structure1 structure1;
        structure1.x = 10;
        structure1.y = 20;

        //create object based on class
        Class1 class1;
        class1 = new Class1();
        class1.x = 10;
        class1.y = 20;

        //create structure instance 2
        Structure1 structure2 = new Structure1();

    }
}
```

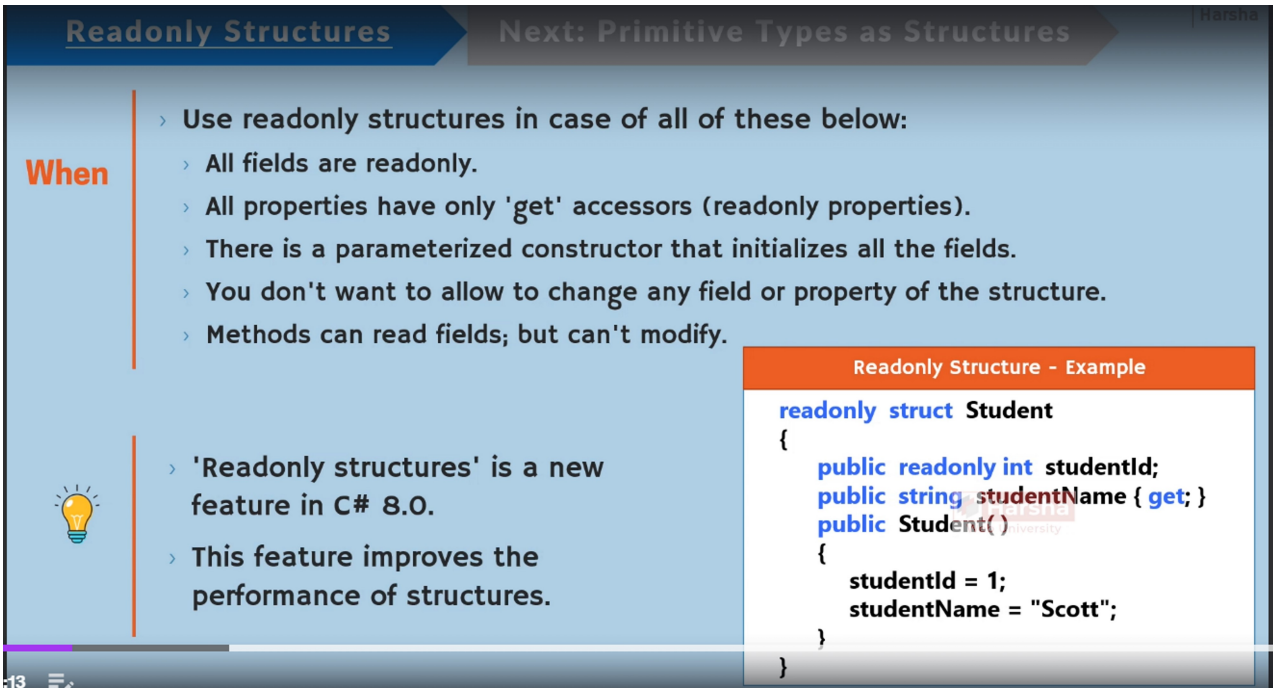



```
    Structure1 structure1;
    structure1.x = 10;
    structure1.y = 20;

    //create object based on class
    Class1 class1;
    class1 = new Class1();
    class1.x = 10;
    class1.y = 20;

    //create structure instance 2
    Structure1 structure2 = new Structure1();

    //copy data from structure1 to structure2
    structure2 = structure1;

    //modify data of structure2
    structure2.x = 100;
    structure2.y = 200;
  }
}
```

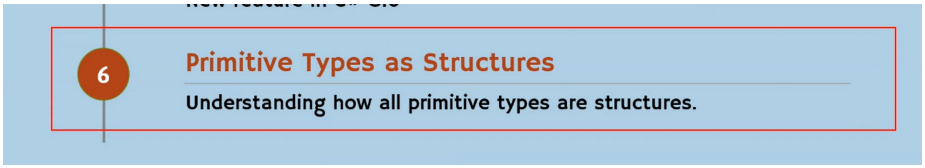




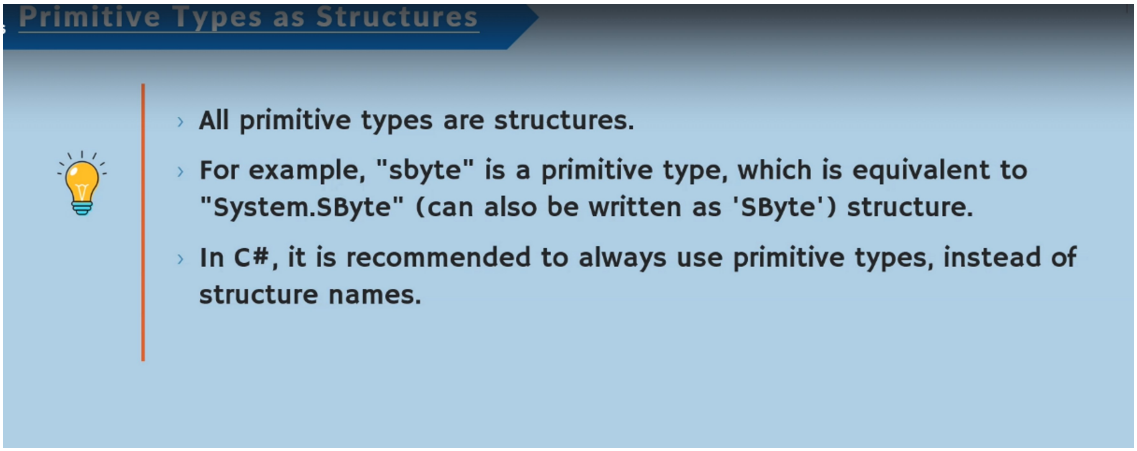



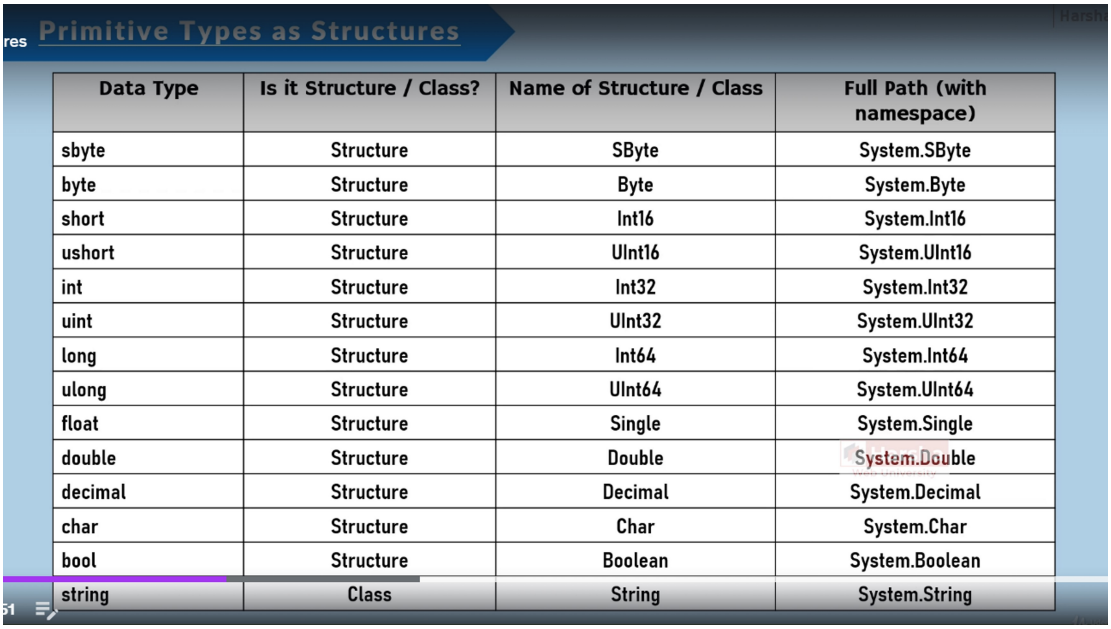




In real time projects, whenever you want to restrict yourself to create all members as readonly members. In that case better to make it as readonly structures.

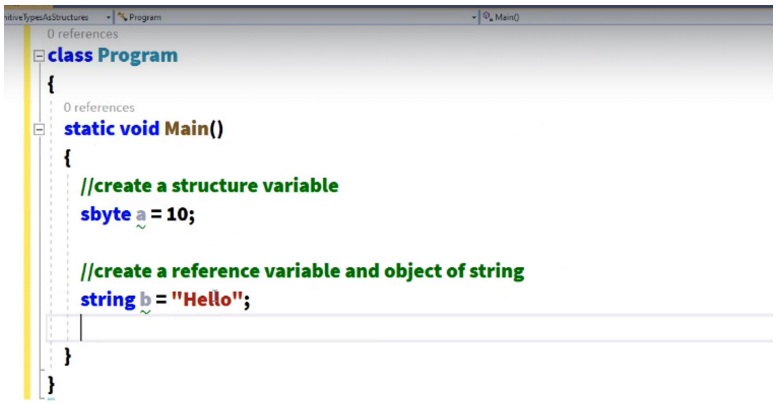


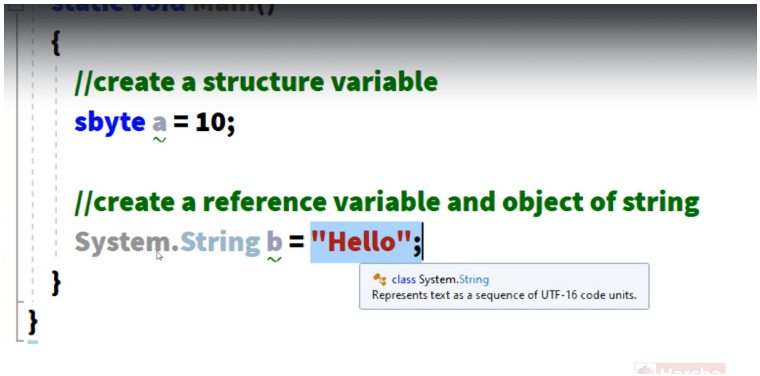

All the primitive types except string are structures. That includes sbyte, byte, short, ushort, int, uint, long, ulong, float, double, decimal, char and bool. All these are internally structures.




| Data Type | Is it Structure / Class? | Name of Structure / Class | Full Path (with namespace) |
|---|---|---|---|
| sbyte | Structure | SByte | System.SByte |
| byte | Structure | Byte | System.Byte |
| short | Structure | Int16 | System.Int16 |
| ushort | Structure | UInt16 | System.UInt16 |
| int | Structure | Int32 | System.Int32 |
| uint | Structure | UInt32 | System.UInt32 |
| long | Structure | Int64 | System.Int64 |
| ulong | Structure | UInt64 | System.UInt64 |
| float | Structure | Single | System.Single |
| double | Structure | Double | System.Double |
| decimal | Structure | Decimal | System.Decimal |
| char | Structure | Char | System.Char |
| bool | Structure | Boolean | System.Boolean |
| string | Class | String | System.String |


0 references
class Program
{
0 references
static void Main()
{
//create a structure variable
sbyte a = 10;
//create a reference variable and object of string
string b = "Hello";
}
}


{
//create a structure variable
sbyte a = 10;
//create a reference variable and object of string
System.String b = "Hello";
}
}
class System.String
Represents text as a sequence of UTF-16 code units.