C# for Beginners

The struct data type

Contents:

- struct
- stuct vs. class, differences and similarities
- Value types
- Initialization
- Nullable types
- When to use struct
- DateTime and TimeSpan structs

Farid Naisan, Senior Lecturer, Malmö University, farid.naisan@mau.se

struct (Structure)

- The type struct is one of the five core C# types, on which the whole C# language is based:
  - class, interface, delegate, struct, enum
- Structures are used as "mini-classes" to encapsulate data and related functionality.
- .NET provides hundreds (maybe more) ready-to-use structs.
  - **DateTime**, **Color** are two examples.
- The built-in, primitive types such as bool, int and double are all structures.
- A struct is a value type.

```
public enum Suit
{
    Spade,
    Club,
    Diamond,
    Heart
}
```

```
public struct PlayingCard
{
    //fields
    private Suit suit;
    private int value;  //1 to 14
    //constructor (No default ctor is allowed
    public PlayingCard(int value, Suit suit)
    {
        this.suit = suit;
        this.value = value;
    }
    //Property
    public Suit Suit
    {
        get { return suit; }
        set { suit = value; }
    }
}
```

Farid Naisan, Malmö University

Value types

- C# has two main categories of types namely value types and reference types.
- A variable of a reference type does not contain an instance of the type, i.e. it does not have the values of the object directly. The variable **loan** in the code below does not contain the instance. It contains the address of the instance in memory where it is created and where the values are stored.

```
BankLoan loan = new BankLoan();
```

- A variable of a value type differs from a reference type as it contains an instance of the type with values saved in its fields.

```
int number1 = 36;
```

- The variable **number1** contains its value directly.
- Values of value type variables are copied on assignment, passing an argument to a method, and returning a method result.

```
int number2 = number1; //number1 is copied to number2.
```


Structure limitations

- A structure has the same capabilities as a class but with the following exceptions:
  - Default constructor (a parameterless constructor) is **not** allowed. Every structure type already provides an implicit parameterless constructor that produces the default value of the type.
  - You can't initialize an instance field or property at its declaration. However, you can initialize a static or const field or a static property at its declaration.
  - A constructor of a structure type must initialize all instance fields of the type.
  - A structure type can't inherit from another class or structure type and it can't be the base of a class. However, a structure type can implement interfaces.
  - You can't declare a destructor (finalizer) within a structure type.

Differences: structures vs. classes

Structures	Classes
The default pubic constructor is reserved. Only constructors with parameters are accepted	Can have constructors with or without parameters.
Fields cannot be initialized	Fields can be initialized
Elements are public by default	Class variables and constants are private by default, other members are public by default
Must at least have one non-shared variable	A class can be completely empty
Cannot be inherited. The Protected modifier cannot be used with members.	Can be inherited and Protected can be used
Are value types	Are reference type
Deep copying (values are copied between variables)	Shallow copying (fields not copied, only reference is copied)

Similarities: Structures vs classes (advanced, you can skip this slide)

- Both may contain other types.
- Both have members,
  - constructors, fields, properties, methods, events and event handlers.
- Both can declare and raise events, and use event handlers.
- Both can declare delegates.
- The differences are outlined in the next slide.

Instantiation of a structure type

- In C#, you must initialize a declared variable before it can be used.
- Because a structure-type variable can't be null (unless it's a variable of a nullable value type), you must instantiate an instance of the corresponding type.
- You can instantiate a structure type by calling an appropriate constructor with the new operator.
- Every structure type has at least one constructor. That's an implicit parameterless constructor, which produces the default value of the type
- If all instance fields of a structure type are accessible, you can also instantiate it without the new operator. In that case you must initialize all instance fields before the first use of the instance. The example on the next slide shows how to do that:

struct- example

- struct Point3D with a parameterized constructor

```
public struct Point3D
{
    public Point3D(double x, double y, double z)
    {
        X = x;
        Y = y;
        Z = z;
    }

    public double X { get; set; }
    public double Y { get; set; }
    public double Z { get; set; }

    public override string ToString() { return $"({X}, {Y}, {Z})"; }
}
```

- Initialize a local struct variable

```
public void Test()
{
    Point3D p = new Point3D();//local variable
    p.X= 3;
    p.Y = 4;
    p.Z = 0;
    Console.WriteLine($"({p.X}, {p.Y})");  // output: (3, 4)
}
```

new used to initiate a local variable

- Initialize a struct variable as a field

```
Point3D p; //instance variable
public void Test2()
{
    p.X = 6;
    p.Y = 7;
    p.Z = 0;
    Console.WriteLine($"({p.X}, {p.Y})");  // output: (6, 7)
}
```

new not needed

Nullable types

- A nullable value type represents all values of its underlying value type T and an additional null value.
- A variable can be declared as nullable by adding a '?' after the type as in the following examples.

```
Point3D? p = null;

bool? ok = null;   //ok can still be assigned true/false

ok = true;
```

- You typically use a nullable value type when you need to represent the undefined value of an underlying value type. For example, a database field may contain true or false, or it may contain no value at all, that is, NULL. You can use the bool? type in that scenario.

Objects of a Structure type

- A structure is instantiated without using the keyword new. The following statements are exactly the same:

```
private Point3d startPoint;

private Point3d startPoint = new Point3d();
```

- Assigning null to a structure has no effect, although it is possible by declaring the structure as nullable (previous slide).
- Due to the fact that a struct is a value type, assigning `structure1 = structure2` results in a deep copying as illustrated in example in the next slide.

Deep-copying example

```
//NOTE:  structs are value types->creation with new not required
//New can be used, but it is still a value type
private Point3d startPoint;
private void Test()
{
    startPoint.X = 50;
    startPoint.Y = 100;

    Point3d endPoint = startPoint;
    endPoint.Y = 150;
}
```

Deep copying

```
public struct Point3d
{
    private double xCoord;
    private double yCoord;
    private double zCoord;

    public double X
    {
        get { return xCoord; }
        set { xCoord = value; }
    }
    public double Y
    {
        get { return yCoord; }
        set { yCoord = value; }
    }
    public double Z
    {
        get { return zCoord; }
        set { zCoord = value; }
    }

}
```


When to use structs (recommendations)

- Use structures if the following criteria is met:
  - The class needs not many methods.
  - The instance size is smaller than 16 bytes.
  - It will not have to be boxed frequently, i.e. you don't use the structure in places where objects are expected.
- **Boxing**: you can convert a **struct** (a value type) to an object by type casting:

  `(object)intVariable`
- **Unboxing** is the reverse mechanism.

  `(int)someObject`

```
void SaveObject (object obj)
{
   //code
}
```

Boxing of DateTime.Now from struct to object.

```
void Test()
{
   SaveObject(DateTime.Now);

   Passing a value type to the above
   method, causes an implicit boxing
   → 5 is converted to an object.

   SaveObject(5);

}
```

The DateTime struct

- The System.DateTime struct is very useful when you are working with dates and times.
- The DateTime value type represents dates and times with values ranging from 00:00:00 (midnight), January 1, 0001 Anno Domini (Common Era) through 11:59:59 P.M., December 31, 9999 A.D. (C.E.) in the Gregorian calendar.
- A DateTime value is always expressed based on an explicit or default calendar.
- You can assign an initial value to a new DateTime value in many different ways:
  - Calling a constructor, either one where you specify arguments for values, or use the implicit parameterless constructor.
  - Assigning a DateTime to the return value of a property or method.
  - Parsing a DateTime value from its string representation.

DateTime example

```
public void TestDates()
{
   DateTime date1 = new   DateTime(2021, 5, 1, 8, 30, 52);
   Console.WriteLine(date1);
}
```

Output: 2021-05-01 08:30:52

- To get the current date and time:

```
DateTime currentTime = DateTime.Now;

DateTime date1 = DateTime.Today;
```

- DateTime has many properties and methods that you can use to perform date and time operations: `int hour = currentTime.Hour;`

See: https://docs.microsoft.com/en-us/dotnet/api/system.datetime?view=net-5.0

TimeSpan

- Another .NET useful type is TimeSpan that several methods and properties to manipulated between dates and times:

```
void test()
{
    // Define two dates.
    DateTime date1 = new DateTime(2021, 1, 1, 8, 0, 15);
    DateTime date2 = new DateTime(2021, 5, 18, 13, 30, 30);

    // Calculate the interval between the two dates.
    TimeSpan interval = date2 - date1;
    Console.WriteLine("{0} - {1} = {2}", date2, date1, interval.ToString());
}
```

Output: 2021-05-18 13:30:30 - 2021-01-01 08:00:15 = 137.05:30:15 (days.hours.min.sec)

See: https://docs.microsoft.com/en-us/dotnet/api/system.timespan?view=net-5.0

Summary

- struct are one of the five types that the whole C# language is built on.
- A struct is a value type. Copying two struct variables copies all values.
- struct is used instead of class mostly as a container of or objects small and used a lot for data transfer.

```
public struct Point2D
{
    public double x;
    public double y;
}
```

- Most often you use a struct containing only data fields.
- Remember that classes are the normal way to encapsulate data and related functionality, although structs are faster and easier to use. Use structs only when you have simple forms of entities.
- For more info: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct