Constructors (C# programming guide)

Article • 04/10/2023

Whenever an instance of a class or a struct is created, its constructor is called. A class or struct may have multiple constructors that take different arguments. Constructors enable the programmer to set default values, limit instantiation, and write code that is flexible and easy to read. For more information and examples, see Instance constructors and Using constructors.

There are several actions that are part of initializing a new instance. Those actions take place in the following order:

1. *Instance fields are set to 0*. This is typically done by the runtime.
2. *Field initializers run*. The field initializers in the most derived type run.
3. *Base type field initializers run*. Field initializers starting with the direct base through each base type to System.Object.
4. *Base instance constructors run*. Any instance constructors, starting with Object.Object through each base class to the direct base class.
5. *The instance constructor runs*. The instance constructor for the type runs.
6. *Object initializers run*. If the expression includes any object initializers, those run after the instance constructor runs. Object initializers run in the textual order.

The preceding actions take place when a new instance is initialized. If a new instance of a `struct` is set to its `default` value, all instance fields are set to 0.

If the static constructor hasn't run, the static constructor runs before any of the instance constructor actions take place.

Constructor syntax

A constructor is a method whose name is the same as the name of its type. Its method signature includes only an optional access modifier, the method name and its parameter list; it does not include a return type. The following example shows the constructor for a class named `Person`.

C#

```csharp
public class Person
{
   private string last;
   private string first;

   public Person(string lastName, string firstName)
   {
      last = lastName;
      first = firstName;
   }

   // Remaining implementation of Person class.
}
```

If a constructor can be implemented as a single statement, you can use an expression body definition. The following example defines a `Location` class whose constructor has a single string parameter named *name*. The expression body definition assigns the argument to the `locationName` field.

C#

```
public class Location
{
    private string locationName;

    public Location(string name) => Name = name;

    public string Name
    {
        get => locationName;
        set => locationName = value;
    }
}
```

Static constructors

The previous examples have all shown instance constructors, which create a new object. A class or struct can also have a static constructor, which initializes static members of the type. Static constructors are parameterless. If you don't provide a static constructor to initialize static fields, the C# compiler initializes static fields to their default value as listed in the Default values of C# types article.

The following example uses a static constructor to initialize a static field.

C#

```csharp
public class Adult : Person
{
   private static int minimumAge;

   public Adult(string lastName, string firstName) : base(lastName, firstName)
   { }

   static Adult()
   {
      minimumAge = 18;
   }

   // Remaining implementation of Adult class.
}
```

You can also define a static constructor with an expression body definition, as the following example shows.

C#

```csharp
public class Child : Person
{
   private static int maximumAge;

   public Child(string lastName, string firstName) : base(lastName, firstName)
   { }

   static Child() => maximumAge = 18;

   // Remaining implementation of Child class.
}
```

For more information and examples, see Static Constructors.

In This Section

- Using constructors
- Instance constructors
- Private constructors
- Static constructors
- How to write a copy constructor

See also

- The C# type system
- Finalizers
- static
- Why Do Initializers Run In The Opposite Order As Constructors? Part One