Constructors (C# Programming Guide)

05/05/2017 • 2 minutes to read • +7

In this article

Parameterless constructors
Constructor syntax
Static constructors
In This Section
See also

Whenever a class or 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 Using Constructors and Instance Constructors.

Parameterless constructors

If you don't provide a constructor for your class, C# creates one by default that instantiates the object and sets member variables to the default values as listed in the Default values of C# types article. If you don't provide a constructor for your struct, C# relies on an *implicit parameterless constructor* to automatically initialize each field to its default value. For more information and examples, see Instance constructors.

Constructor syntax

A constructor is a method whose name is the same as the name of its type. Its method signature includes only 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# Copy

```
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# Copy

```
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# Copy

```
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# Copy

```
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

- C# Programming Guide
- Classes and Structs
- Finalizers
- static
- Why Do Initializers Run In The Opposite Order As Constructors? Part One

Is this page helpful?

Yes No