Using Constructors (C# Programming Guide)

07/20/2015 • 4 minutes to read • +11

In this article

C# Language Specification

See also

When a class or struct is created, its constructor is called. Constructors have the same name as the class or struct, and they usually initialize the data members of the new object.

In the following example, a class named `Taxi` is defined by using a simple constructor. This class is then instantiated with the new operator. The `Taxi` constructor is invoked by the `new` operator immediately after memory is allocated for the new object.

C# Copy

```
public class Taxi
{
    public bool IsInitialized;
    public Taxi()
    {
        IsInitialized = true;
    }
}

class TestTaxi
{
    static void Main()
    {
        Taxi t = new Taxi();
        Console.WriteLine(t.IsInitialized);
```

```
    }
}
```

A constructor that takes no parameters is called a *parameterless constructor*. Parameterless constructors are invoked whenever an object is instantiated by using the `new` operator and no arguments are provided to `new`. For more information, see Instance Constructors.

Unless the class is static, classes without constructors are given a public parameterless constructor by the C# compiler in order to enable class instantiation. For more information, see Static Classes and Static Class Members.

You can prevent a class from being instantiated by making the constructor private, as follows:

C# Copy

```
class NLog
{
    // Private Constructor:
    private NLog() { }

    public static double e = Math.E;  //2.71828...
}
```

For more information, see Private Constructors.

Constructors for struct types resemble class constructors, but `structs` cannot contain an explicit parameterless constructor because one is provided automatically by the compiler. This constructor initializes each field in the `struct` to the default value. However, this parameterless constructor is only invoked if the `struct` is instantiated with `new`. For example, this code uses the parameterless constructor for Int32, so that you are assured that the integer is initialized:

C# Copy

```
int i = new int();
Console.WriteLine(i);
```

The following code, however, causes a compiler error because it does not use `new`, and because it tries to use an object that has not been initialized:

C# Copy

```
int i;
Console.WriteLine(i);
```

Alternatively, objects based on `structs` (including all built-in numeric types) can be initialized or assigned and then used as in the following example:

C# Copy

```
int a = 44;  // Initialize the value type...
int b;
b = 33;      // Or assign it before using it.
Console.WriteLine("{0}, {1}", a, b);
```

So calling the parameterless constructor for a value type is not required.

Both classes and `structs` can define constructors that take parameters. Constructors that take parameters must be called through a `new` statement or a base statement. Classes and `structs` can also define multiple constructors, and neither is required to define a parameterless constructor. For example:

C# Copy

```
public class Employee
{
    public int Salary;

    public Employee(int annualSalary)
    {
```

```
        Salary = annualSalary;
    }

    public Employee(int weeklySalary, int numberOfWeeks)
    {
        Salary = weeklySalary * numberOfWeeks;
    }
}
```

This class can be created by using either of the following statements:

C# Copy

```
Employee e1 = new Employee(30000);
Employee e2 = new Employee(500, 52);
```

A constructor can use the `base` keyword to call the constructor of a base class. For example:

C# Copy

```
public class Manager : Employee
{
    public Manager(int annualSalary)
        : base(annualSalary)
    {
        //Add further instructions here.
    }
}
```

In this example, the constructor for the base class is called before the block for the constructor is executed. The `base` keyword can be used with or without parameters. Any parameters to the constructor can be used as parameters to `base`, or as part of an expression. For more information, see base.

In a derived class, if a base-class constructor is not called explicitly by using the `base` keyword, the parameterless constructor, if there is one, is called implicitly. This means that the following constructor declarations are effectively the same:

C# Copy

```
public Manager(int initialData)
{
    //Add further instructions here.
}
```

C# Copy

```
public Manager(int initialData)
    : base()
{
    //Add further instructions here.
}
```

If a base class does not offer a parameterless constructor, the derived class must make an explicit call to a base constructor by using `base`.

A constructor can invoke another constructor in the same object by using the this keyword. Like `base`, `this` can be used with or without parameters, and any parameters in the constructor are available as parameters to `this`, or as part of an expression. For example, the second constructor in the previous example can be rewritten using `this`:

C# Copy

```
public Employee(int weeklySalary, int numberOfWeeks)
    : this(weeklySalary * numberOfWeeks)
{
}
```

The use of the `this` keyword in the previous example causes this constructor to be called:

C# Copy

```
public Employee(int annualSalary)
{
    Salary = annualSalary;
}
```

Constructors can be marked as public, private, protected, internal, protected internal or private protected. These access modifiers define how users of the class can construct the class. For more information, see Access Modifiers.

A constructor can be declared static by using the static keyword. Static constructors are called automatically, immediately before any static fields are accessed, and are generally used to initialize static class members. For more information, see Static Constructors.

C# Language Specification

For more information, see Instance constructors and Static constructors in the C# Language Specification. The language specification is the definitive source for C# syntax and usage.

See also

- C# Programming Guide
- Classes and Structs
- Constructors
- Finalizers

Is this page helpful?

Yes No