

# Instance Constructors (C# Programming Guide)

07/20/2015 • 4 minutes to read •  +7

## In this article

[Example 1](#)

[Example 2](#)

[Example 3](#)

[See also](#)

Instance constructors are used to create and initialize any instance member variables when you use the [new](#) expression to create an object of a [class](#). To initialize a [static](#) class, or static variables in a non-static class, you define a static constructor. For more information, see [Static Constructors](#).

The following example shows an instance constructor:

C#

 Copy

```
class Coords
{
    public int x, y;


    // constructor
    public Coords()
    {
        x = 0;
        y = 0;
    }
}
```

### ⓘ Note

For clarity, this class contains public fields. The use of public fields is not a recommended programming practice because it allows any method anywhere in a program unrestricted and unverified access to an object's inner workings. Data members should generally be private, and should be accessed only through class methods and properties.

This instance constructor is called whenever an object based on the `Coords` class is created. A constructor like this one, which takes no arguments, is called a *parameterless constructor*. However, it is often useful to provide additional constructors. For example, we can add a constructor to the `Coords` class that allows us to specify the initial values for the data members:

C#

 Copy

```
// A constructor with two arguments.
public Coords(int x, int y)
{
    this.x = x;
    this.y = y;
}
```

This allows `Coords` objects to be created with default or specific initial values, like this:

C#

 Copy

```
var p1 = new Coords();
var p2 = new Coords(5, 3);
```

If a class does not have a constructor, a parameterless constructor is automatically generated and default values are used to initialize the object fields. For example, an [int](#) is initialized to 0. For information about the type default values, see [Default values of C# types](#). Therefore, because the `Coords` class parameterless constructor initializes all data members to zero, it can

be removed altogether without changing how the class works. A complete example using multiple constructors is provided in Example 1 later in this topic, and an example of an automatically generated constructor is provided in Example 2.

Instance constructors can also be used to call the instance constructors of base classes. The class constructor can invoke the constructor of the base class through the initializer, as follows:

C#

 Copy

```
class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }
}
```

In this example, the `Circle` class passes values representing radius and height to the constructor provided by `Shape` from which `Circle` is derived. A complete example using `Shape` and `Circle` appears in this topic as Example 3.

## Example 1

The following example demonstrates a class with two class constructors, one without arguments and one with two arguments.

C#

 Copy

```
class Coords
{
    public int x, y;

    // Default constructor.
    public Coords()
    {
    }
}
```

```
        x = 0;
        y = 0;
    }

    // A constructor with two arguments.
    public Coords(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    // Override the ToString method.
    public override string ToString()
    {
        return $"({x},{y})";
    }
}

class MainClass
{
    static void Main()
    {
        var p1 = new Coords();
        var p2 = new Coords(5, 3);


        // Display the results using the overridden ToString method.
        Console.WriteLine($"Coords #1 at {p1}");
        Console.WriteLine($"Coords #2 at {p2}");
        Console.ReadKey();
    }
}

/* Output:
   Coords #1 at (0,0)
   Coords #2 at (5,3)
*/
```

## Example 2

In this example, the class `Person` does not have any constructors, in which case, a parameterless constructor is automatically provided and the fields are initialized to their default values.

C#

 Copy

```
public class Person
{
    public int age;
    public string name;
}

class TestPerson
{
    static void Main()
    {
        var person = new Person();

        Console.WriteLine("Name: {person.name}, Age: {person.age}");
        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}

// Output: Name: , Age: 0
```

Notice that the default value of `age` is `0` and the default value of `name` is `null`.

## Example 3

The following example demonstrates using the base class initializer. The `Circle` class is derived from the general class `Shape`, and the `Cylinder` class is derived from the `Circle` class. The constructor on each derived class is using its base class

initializer.

C#

 Copy

```
abstract class Shape
{
    public const double pi = Math.PI;
    protected double x, y;

    public Shape(double x, double y)
    {
        this.x = x;
        this.y = y;
    }

    public abstract double Area();
}

class Circle : Shape
{
    public Circle(double radius)
        : base(radius, 0)
    {
    }
    public override double Area()
    {
        return pi * x * x;
    }
}

class Cylinder : Circle
{
    public Cylinder(double radius, double height)
        : base(radius)
    {
        y = height;
    }
}
```

```
public override double Area()
{
    return (2 * base.Area()) + (2 * pi * x * y);
}

class TestShapes
{
    static void Main()
    {
        double radius = 2.5;
        double height = 3.0;

        Circle ring = new Circle(radius);
        Cylinder tube = new Cylinder(radius, height);

        Console.WriteLine("Area of the circle = {0:F2}", ring.Area());
        Console.WriteLine("Area of the cylinder = {0:F2}", tube.Area());

        // Keep the console window open in debug mode.
        Console.WriteLine("Press any key to exit.");
        Console.ReadKey();
    }
}

/* Output:
   Area of the circle = 19.63
   Area of the cylinder = 86.39
*/
```

For more examples on invoking the base class constructors, see [virtual](#), [override](#), and [base](#).

## See also

- [C# Programming Guide](#)
- [Classes and Structs](#)

- [Constructors](#)
- [Finalizers](#)
- [static](#)

---

Is this page helpful?

 Yes  No

---