Generate a constructor in Visual Studio

01/26/2018 • 3 minutes to read •

In this article

This code generation applies to:

- C#
- Visual Basic

What: Lets you immediately generate the code for a new constructor on a class.

When: You introduce a new constructor and want to properly declare it automatically, or you modify an existing constructor.

Why: You could declare the constructor before using it, however this feature will generate it, with the proper parameters, automatically. Furthermore, modifying an existing constructor requires updating all the callsites unless you use this feature to update them automatically.

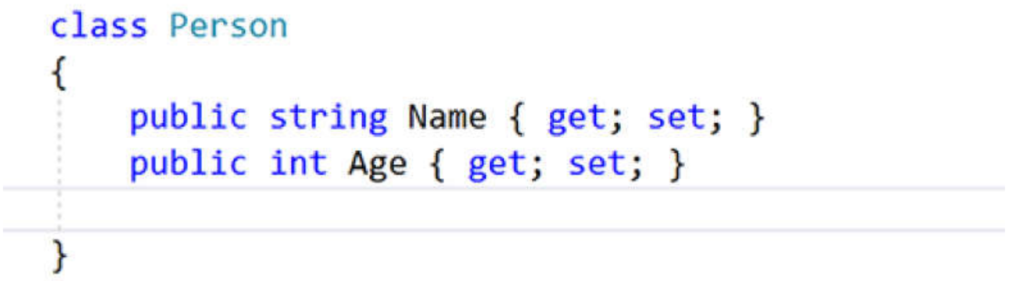
How: There are several ways to generate a constructor:

- Generate constructor and pick members
- Generate constructor from selected fields
- Generate constructor from new usage

- Add parameter to existing constructor
- Create and initialize field/property from a constructor parameter

Generate constructor and pick members (C# only)

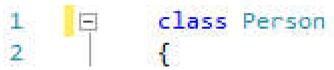
1. Place your cursor in any empty line in a class:



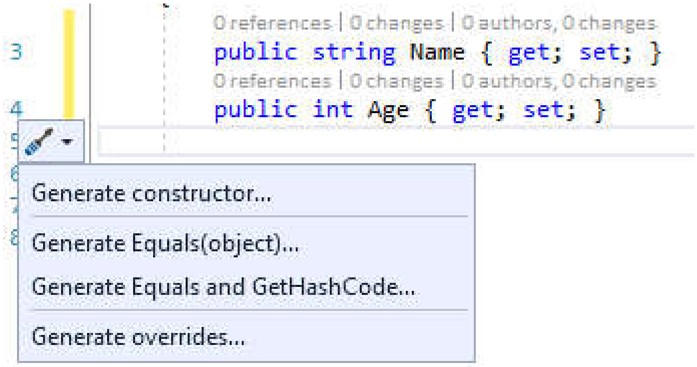
```
class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

}
```

2. Next, do one of the following:

   - **Keyboard**
     - Press **Ctrl**+**.** to trigger the **Quick Actions and Refactorings** menu.
   - **Mouse**
     - Right-click and select the **Quick Actions and Refactorings** menu.
     - Click the 🖌 icon that appears in the left margin if the text cursor is already on the empty line in the class.



```
class Person
{
```



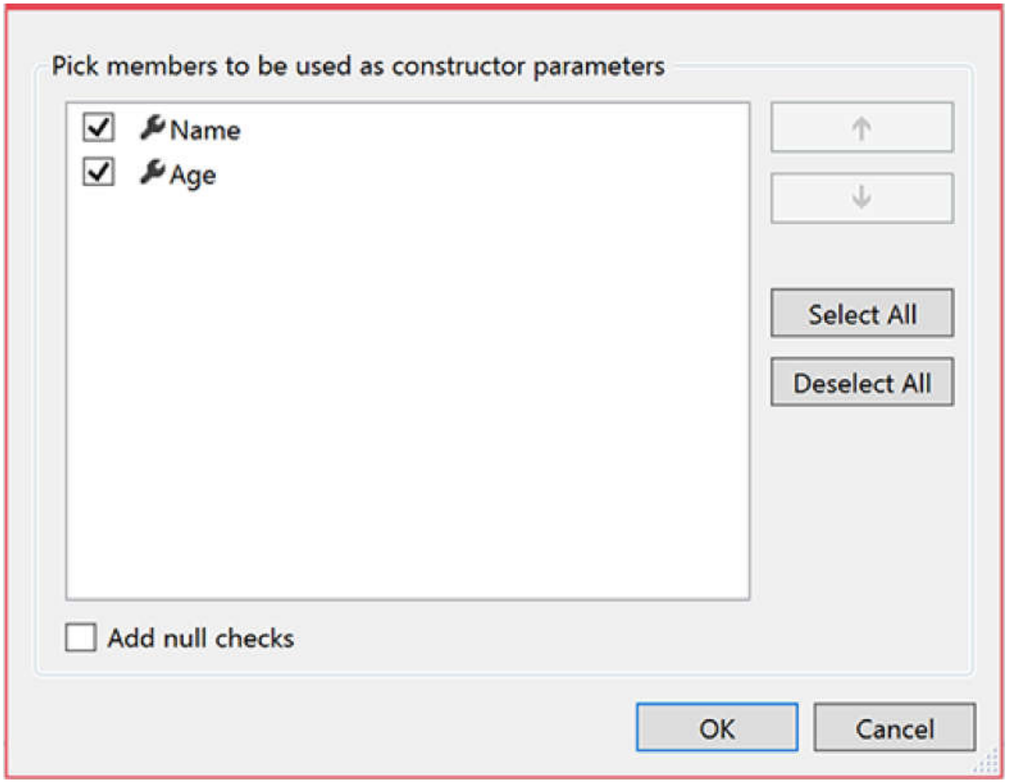
```
public string Name { get; set; }
public int Age { get; set; }
```

0 references | 0 changes | 0 authors, 0 changes
0 references | 0 changes | 0 authors, 0 changes
Generate constructor...
Generate Equals(object)...
Generate Equals and GetHashCode...
Generate overrides...

3. Select **Generate constructor** from the drop-down menu.

   The **Pick members** dialog box opens.

4. Pick the members you want to include as constructor parameters. You can order them using the up and down arrows. Choose **OK**.



Pick members to be used as constructor parameters

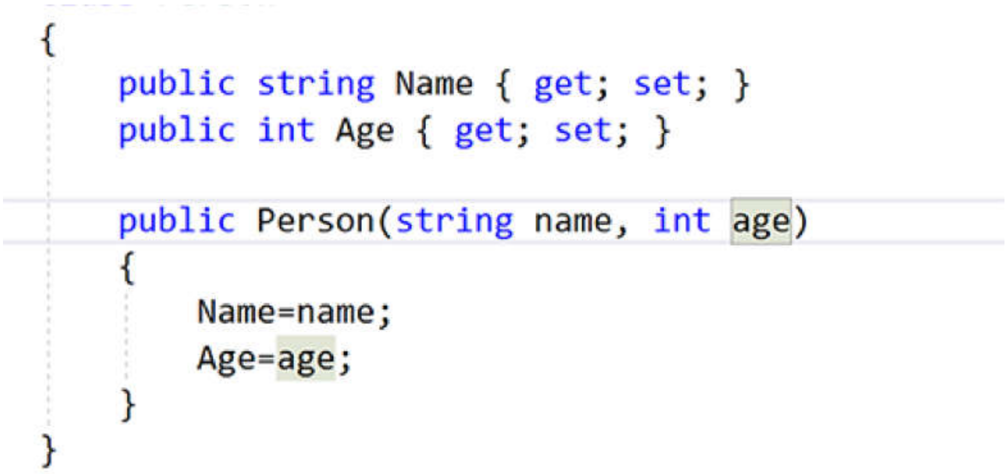
☑ Name
☑ Age

□ Add null checks

> 💡 **Tip**
>
> You can check the **Add null checks** checkbox to automatically generate null checks for your constructor parameters.

The constructor is created with the specified parameters.

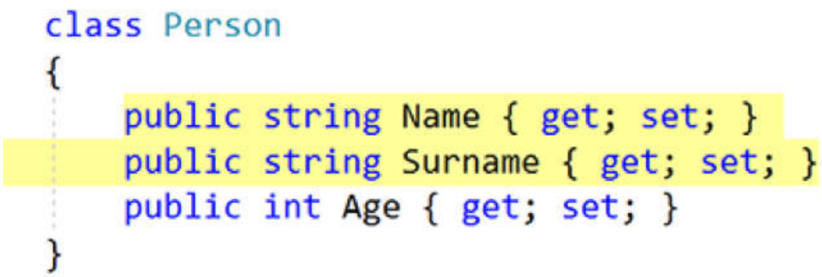
```
class Person
```



```
{
    public string Name { get; set; }
    public int Age { get; set; }

    public Person(string name, int age)
    {
        Name=name;
        Age=age;
    }
}
```

Generate constructor from selected fields (C# only)

1. Highlight the members you wish to have in your generated constructor:

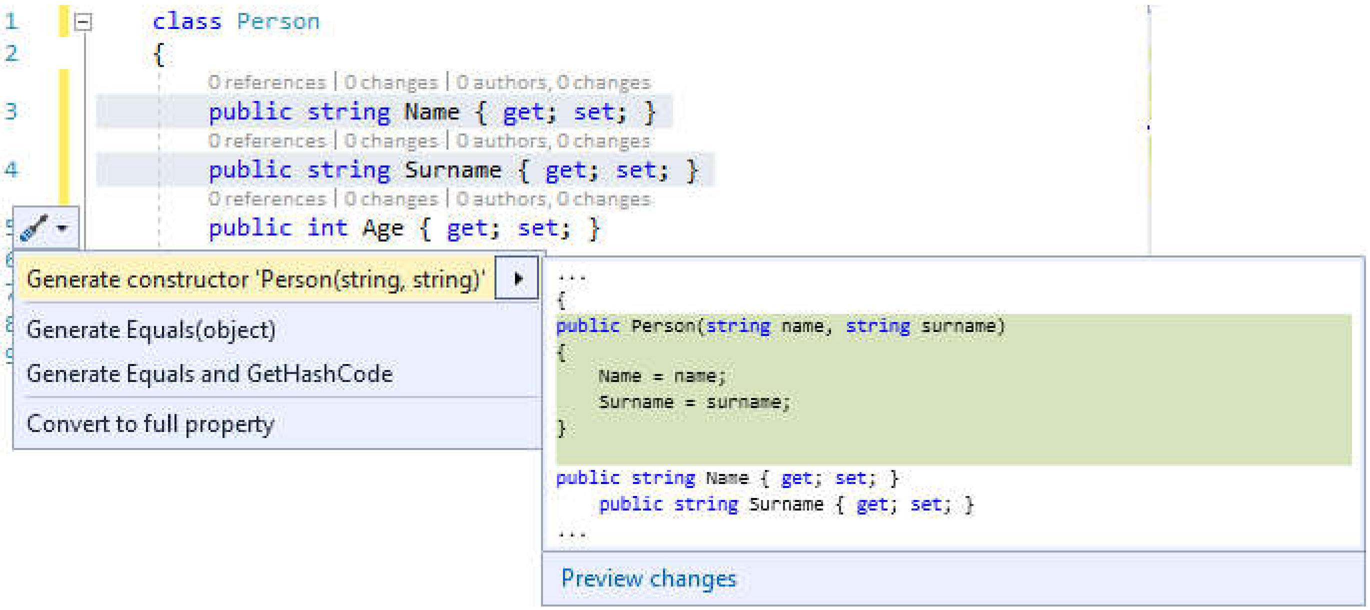


```
class Person
{
    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}
```

2. Next, do one of the following:

- **Keyboard**
  - Press **Ctrl**+**.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
  - Right-click and select the **Quick Actions and Refactorings** menu.

- Click the 🖌 icon that appears in the left margin if the text cursor is already on the line with the selection.



```
class Person
{
    0 references | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public string Surname { get; set; }
    0 references | 0 changes | 0 authors, 0 changes
    public int Age { get; set; }
```

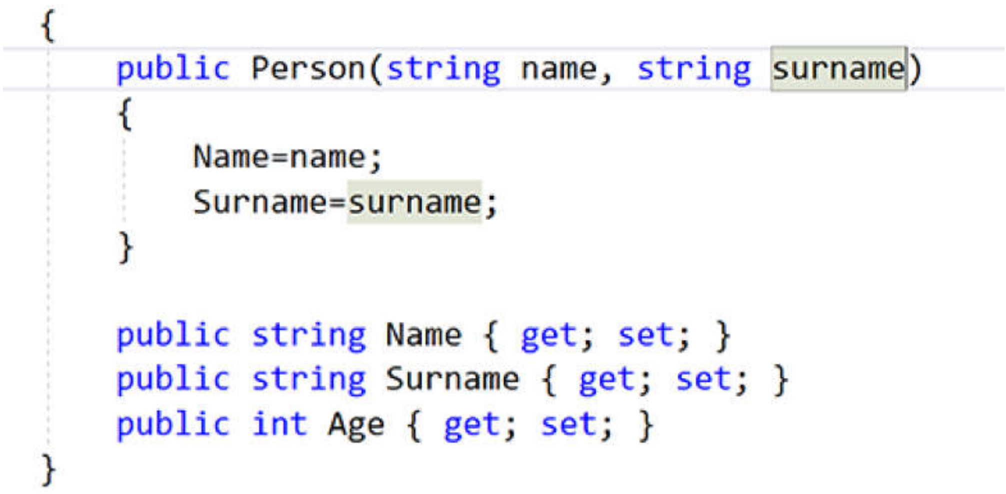
```
...
{
public Person(string name, string surname)
{
    Name = name;
    Surname = surname;
}

public string Name { get; set; }
    public string Surname { get; set; }
...
```

Generate constructor 'Person(string, string)'
Generate Equals(object)
Generate Equals and GetHashCode
Convert to full property
Preview changes

3. Select **Generate constructor 'TypeName(...)'** from the drop-down menu.

   The constructor is created with the selected parameters.

```
class Person
```

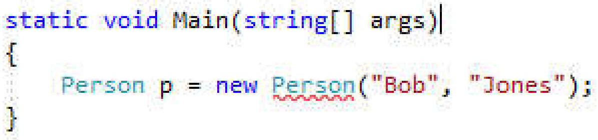


```
{
    public Person(string name, string surname)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; set; }
}
```

Generate constructor from new usage (C# and Visual Basic)

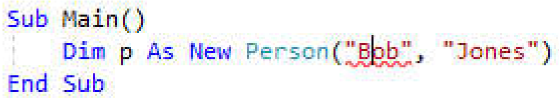
1. Place your cursor on the line where there is a red squiggle. The red squiggle indicates a call to a constructor that doesn't yet exist.

    - C#:



```
static void Main(string[] args)
{
    Person p = new Person("Bob", "Jones");
}
```

    - Visual Basic:



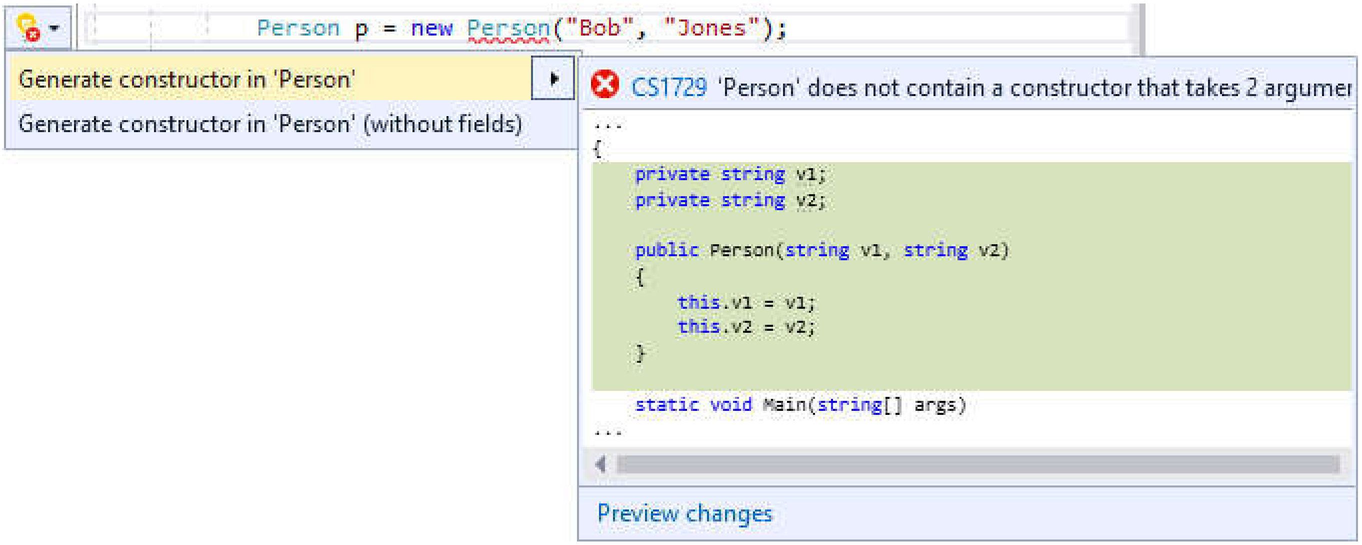
```
Sub Main()
    Dim p As New Person("Bob", "Jones")
End Sub
```

2. Next, do one of the following:

    - **Keyboard**

  - Press **Ctrl**+**.** to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**
  - Right-click and select the **Quick Actions and Refactorings** menu.
  - Hover over the red squiggle and click the 💡 icon that appears.
  - Click the 💡 icon that appears in the left margin if the text cursor is already on the line with the red squiggle.


Generate constructor in 'Person'
Generate constructor in 'Person' (without fields)
Preview changes

```
Person p = new Person("Bob", "Jones");
```

CS1729 'Person' does not contain a constructor that takes 2 argumer

```
...
{
    private string v1;
    private string v2;

    public Person(string v1, string v2)
    {
        this.v1 = v1;
        this.v2 = v2;
    }

    static void Main(string[] args)
...
```

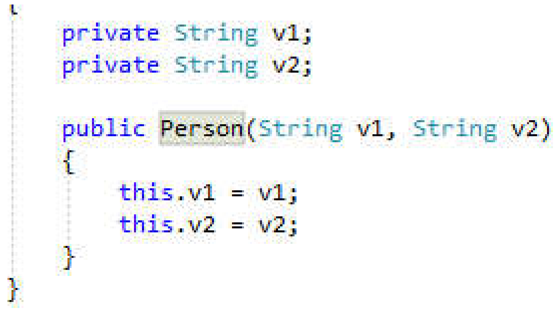
3. Select **Generate constructor in '*TypeName*'** from the drop-down menu.

> **Tip**
>
> Use the **Preview changes** link at the bottom of the preview window to see all of the changes that will be made before making your selection.

The constructor is created, with any parameters inferred from its usage.

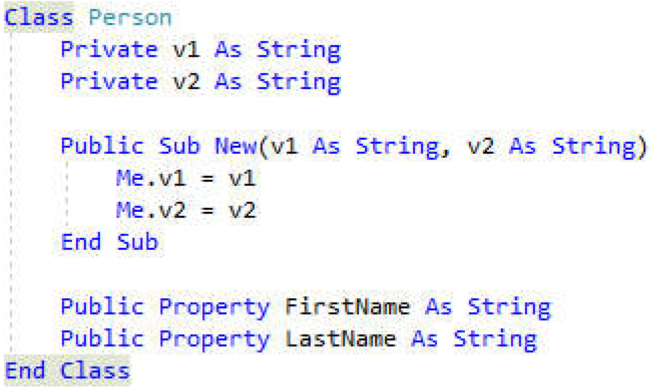
- C#:

```
class Person
{
```



```
{
    private String v1;
    private String v2;

    public Person(String v1, String v2)
    {
        this.v1 = v1;
        this.v2 = v2;
    }
}
```

- Visual Basic:

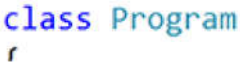


```
Class Person
    Private v1 As String
    Private v2 As String

    Public Sub New(v1 As String, v2 As String)
        Me.v1 = v1
        Me.v2 = v2
    End Sub

    Public Property FirstName As String
    Public Property LastName As String
End Class
```

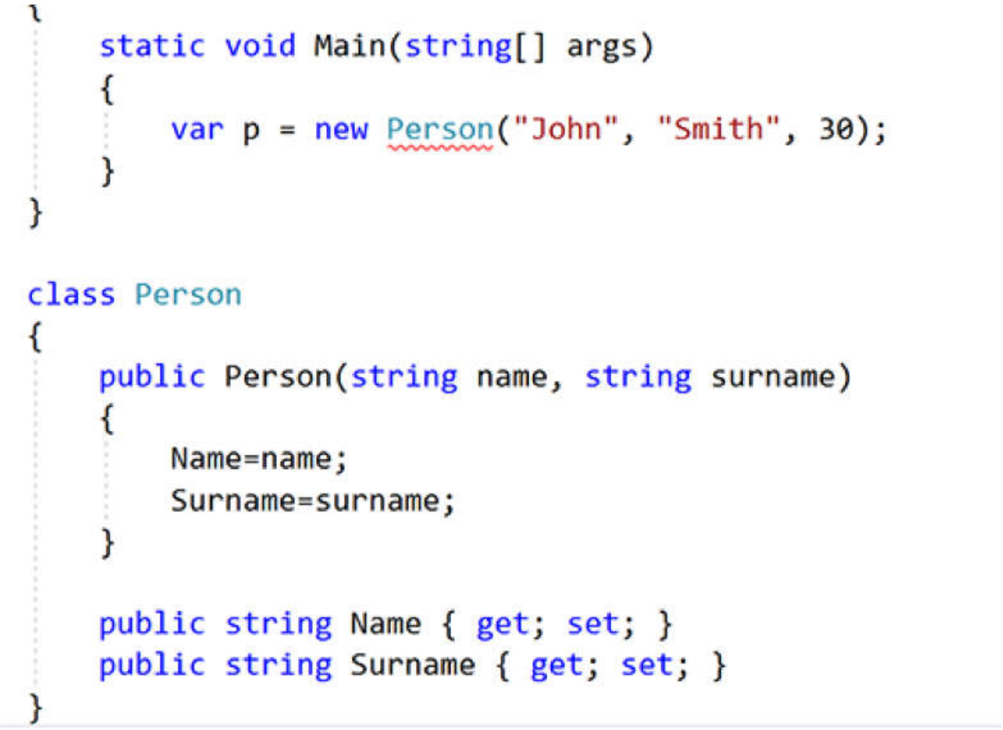
Add parameter to existing constructor (C# only)

1. Add a parameter to an existing constructor call.

2. Place your cursor on the line where there is a red squiggle indicating you've used a constructor that doesn't yet exist.



```
class Program
{
```



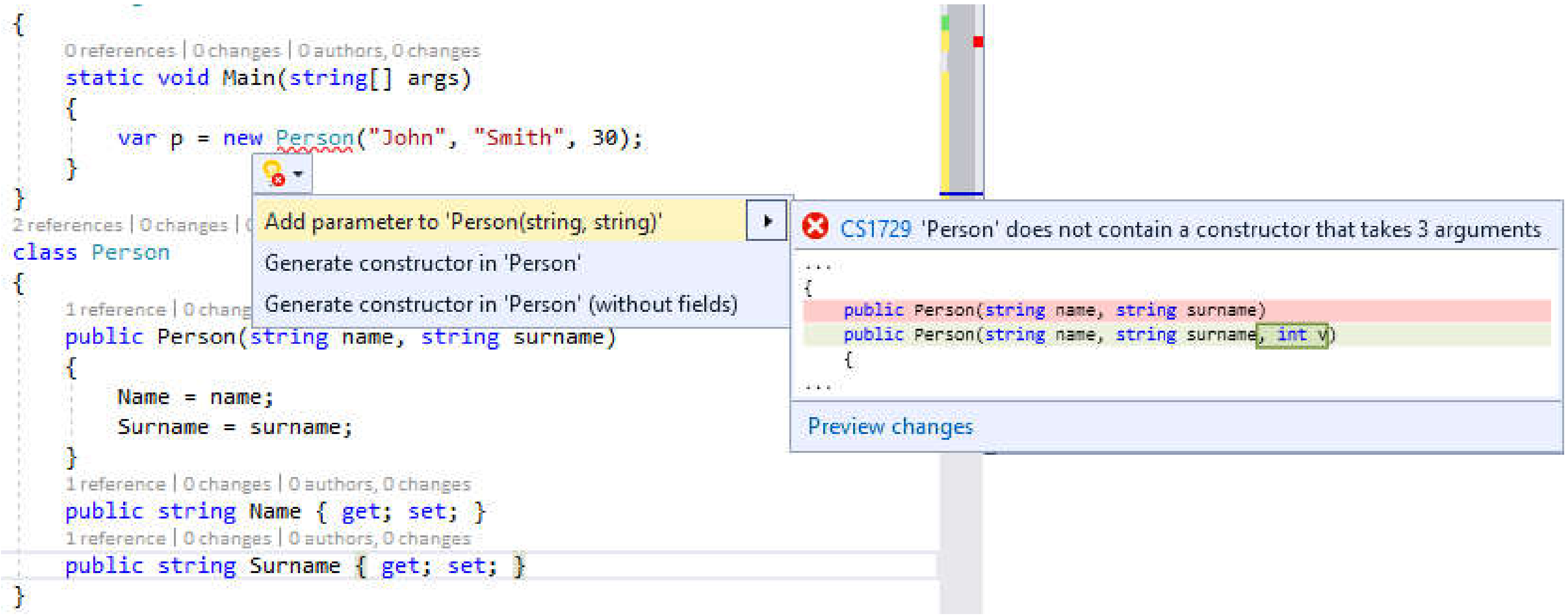
```
{
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}

class Person
{
    public Person(string name, string surname)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}
```

3. Next, do one of the following:

- **Keyboard**
  - Press **Ctrl**+. to trigger the **Quick Actions and Refactorings** menu.

- **Mouse**
  - Right-click and select the **Quick Actions and Refactorings** menu.
  - Hover over the red squiggle and click the 💡 icon that appears.
  - Click the 💡 icon that appears in the left margin if the text cursor is already on the line with the red squiggle.

```
0 references | 0 changes | 0 authors, 0 changes
class Program
```



```
{
    0 references | 0 changes | 0 authors, 0 changes
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}
2 references | 0 changes | (
class Person
{
    1 reference | 0 chang
    public Person(string name, string surname)
    {
        Name = name;
        Surname = surname;
    }
    1 reference | 0 changes | 0 authors, 0 changes
    public string Name { get; set; }
    1 reference | 0 changes | 0 authors, 0 changes
    public string Surname { get; set; }
}
```

Add parameter to 'Person(string, string)'
Generate constructor in 'Person'
Generate constructor in 'Person' (without fields)

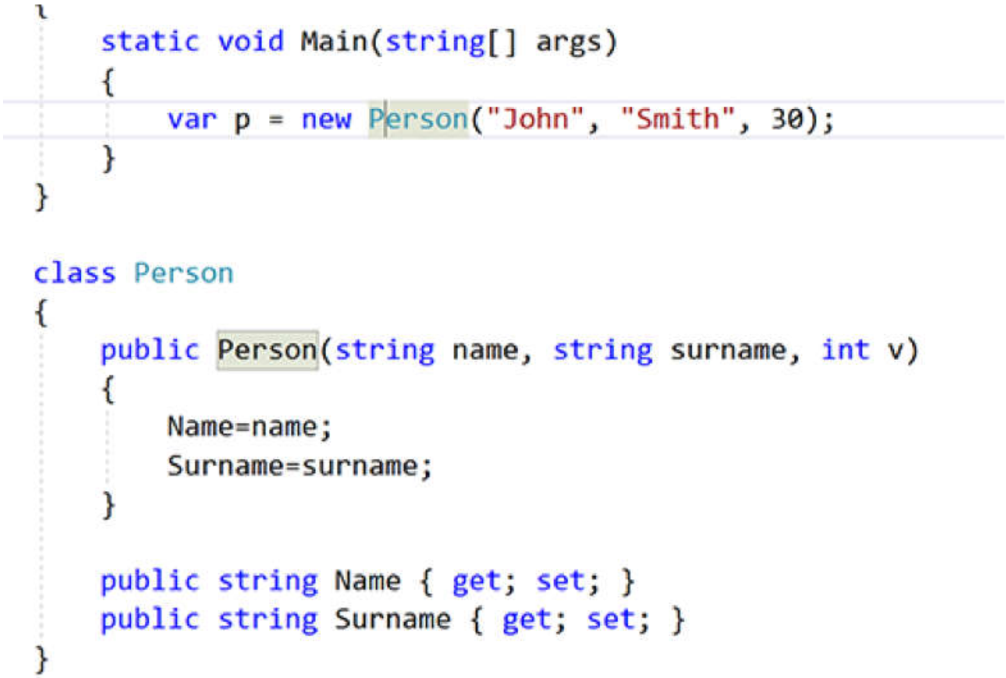
CS1729 'Person' does not contain a constructor that takes 3 arguments

```
...
{
    public Person(string name, string surname)
    public Person(string name, string surname, int v)
    {
...
```

4. Select **Add parameter to 'TypeName(...)'** from the drop-down menu.

   The parameter is added to the constructor, with its type inferred from its usage.

```
class Program
{
```



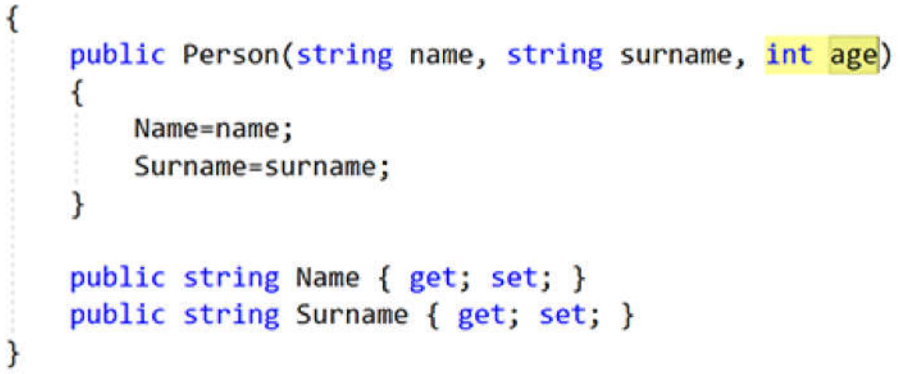
```
{
    static void Main(string[] args)
    {
        var p = new Person("John", "Smith", 30);
    }
}

class Person
{
    public Person(string name, string surname, int v)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}
```

You can also add a parameter to an existing method. For more information, see [Add parameter to a method](#).

Create and initialize a field or property from a constructor parameter (C# only)

1. Find an existing constructor, and add a parameter:

```
class Person
```

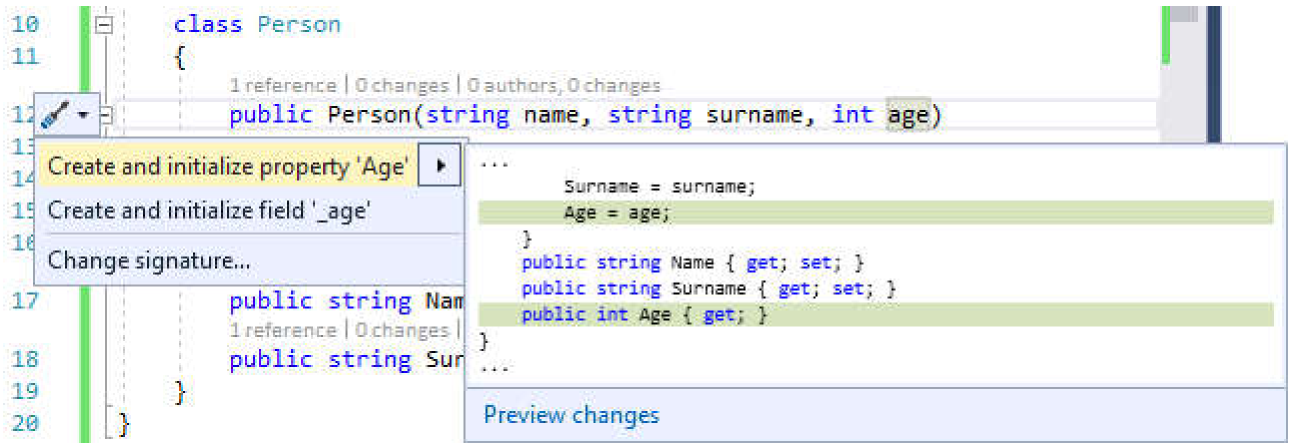


```
{
    public Person(string name, string surname, int age)
    {
        Name=name;
        Surname=surname;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
}
```

2. Place your cursor inside the newly added parameter.

3. Next, do one of the following:

- **Keyboard**
  - Press **Ctrl**+**.** to trigger the **Quick Actions and Refactorings** menu.
- **Mouse**
  - Right-click and select the **Quick Actions and Refactorings** menu.
  - Click the 🖌 icon that appears in the left margin if the text cursor is already on the line with the added parameter.



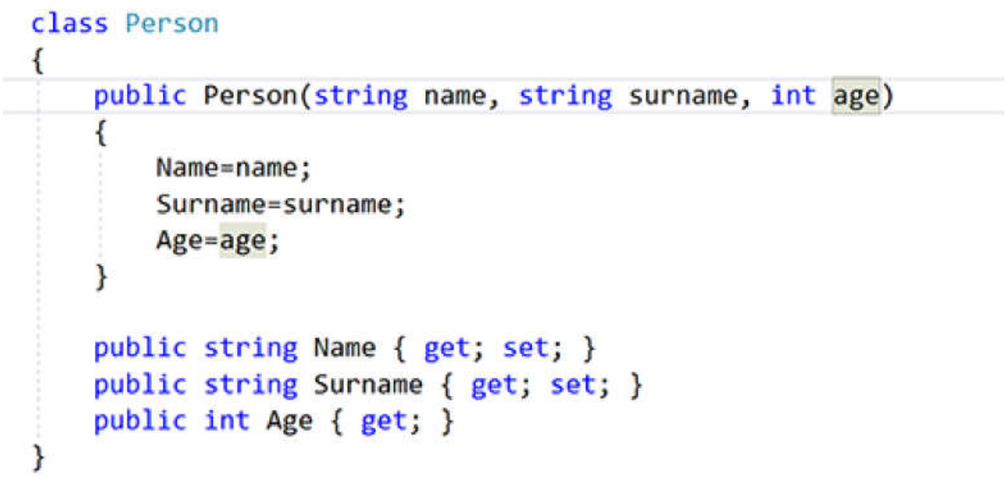
```
class Person
{
    1 reference | 0 changes | 0 authors, 0 changes
    public Person(string name, string surname, int age)

    public string Nam
    1 reference | 0 changes |
    public string Sur
    }
}
```

```
...
        Surname = surname;
        Age = age;
    }
    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; }
}
...
```

Create and initialize property 'Age'
Create and initialize field '_age'
Change signature...
Preview changes

4. Select **Create and initialize property** or **Create and initialize field** from the drop-down menu.

The field or property is declared and automatically named to match your types. A line of code is also added to initialize the field or property in the constructor body.



```
class Person
{
    public Person(string name, string surname, int age)
    {
        Name=name;
        Surname=surname;
        Age=age;
    }

    public string Name { get; set; }
    public string Surname { get; set; }
    public int Age { get; }
}
```

See also

- Code generation
- Preview changes