MethodOverload – Walkthrough

Method Overloading

C# enables several methods of the same name to be defined in the same class, as long as these methods have different sets of parameters (number of parameters, types of parameters or order of the parameters)

This is called method overloading

When an overloaded method is called, the C# compiler selects the proper method by examining the number, types and order of the call’s arguments

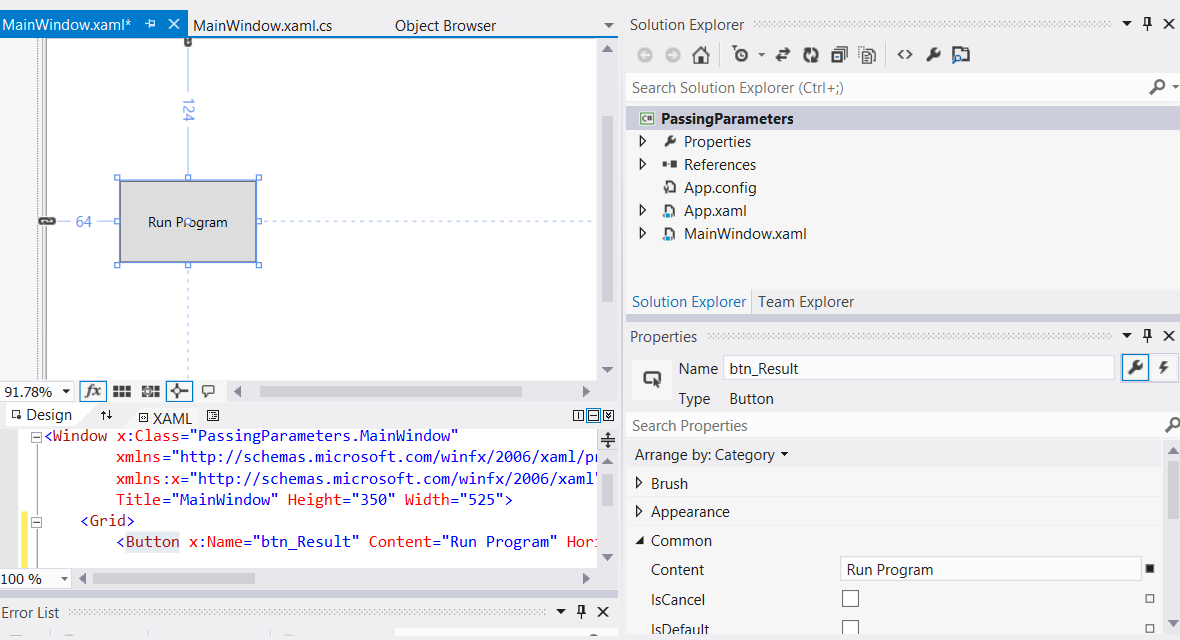
Method overloading commonly is used to create several methods with the same name that perform similar tasks, but on different data types

We are going to build a basic application which has a single button, which when clicked will:

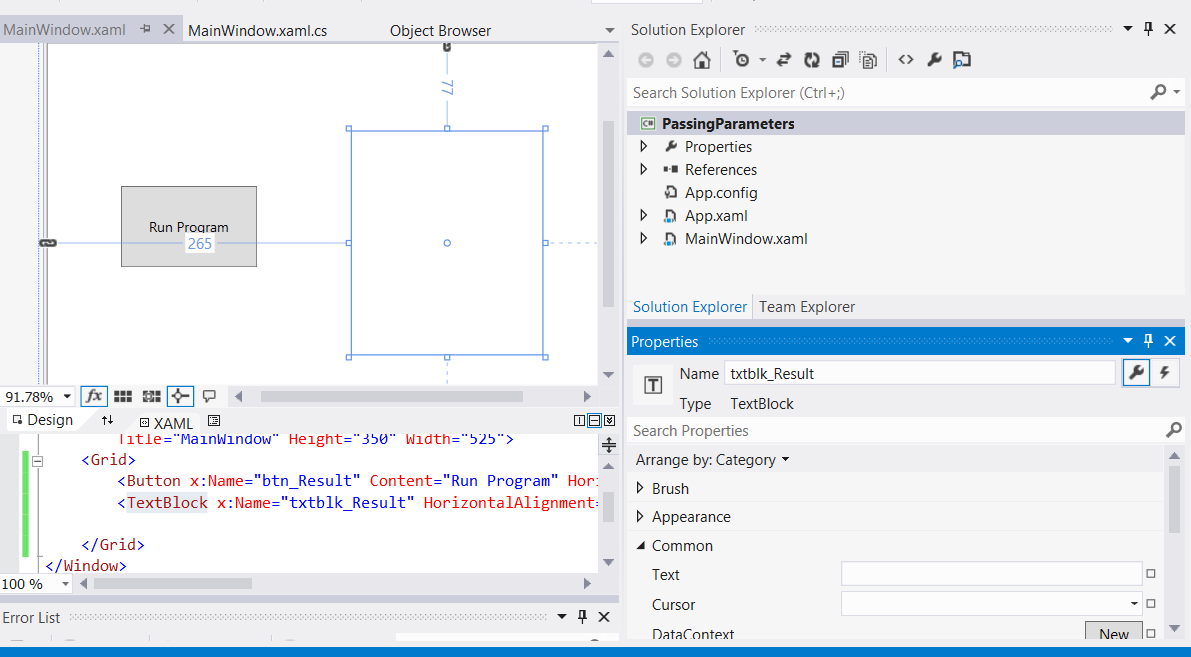
* use overloaded methods named “Square” to calculate the square of an int and a double

The aim of this tutorial is to show how to use overloaded methods typically to operate on different data types.

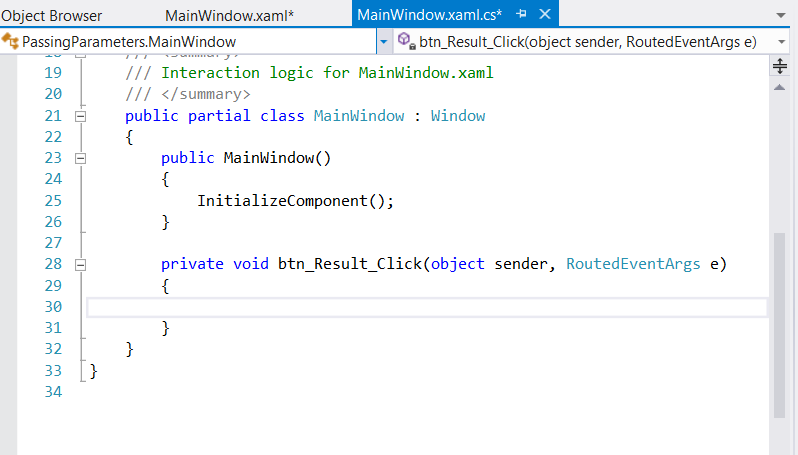
1. Create a new WPF project
2. Drag across a button from the toolbox
   1. set the “Name” property to btn\_Result
   2. set the “Content” property to “Run Program”



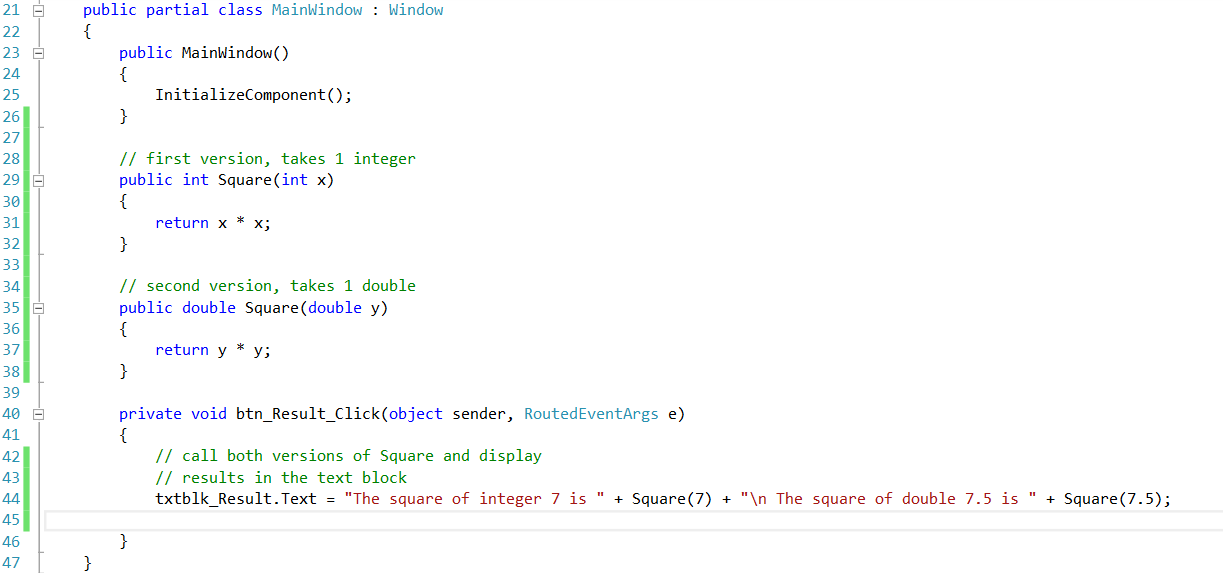
1. Drag across a text block and from the toolbox
   1. set the “Name” property to txtblk\_Result
   2. set the “Text” property to blank



1. Double-click on the “Run Program” button to automatically create an event handler in the MainWindow.xaml.cs file, which is a method called btn\_Result\_Click



1. We are now going to write the code for this event handler and additional “helper” overloaded methods called “Square”. The final code can be viewed in the screenshot below followed by a full explanation



The compiler distinguishes overloaded methods by their signatures

A method’s signature is a combination of the method’s name and parameter types

If the compiler looked only at method names during compilation, the code

above would be ambiguous - the compiler would not know how to distinguish the two Square methods

The compiler uses overload resolution to determine which method to call

This process first searches for all the methods that can be used in the context, based on the number and type of arguments that are present

It might seem that only one method would match, but recall that C# can convert variable values to other data types implicitly

Once all matching methods are found, the closest match is chosen

This match is based on a “best-fit” algorithm, which analyses the implicit conversions that will take place

Let us look at an example

In the code above, the compiler might use the logical name “Square of int” for the Square method that specifies an int parameter (line 29) and “Square of double” for the Square method that specifies a double parameter (line 35)

If a method Foo’s definition begins as

**void Foo( int a, float b )**

the compiler might use the logical name “Foo of int and float”

If the parameters are specified as

**void Foo( float a, int b )**

the compiler might use the logical name “Foo of float and int”

The order of the parameters is important to the compiler; it considers the preceding two Foo methods as distinct

So far, the logical names of methods that have been used by the compiler have not mentioned the methods’ return types

This is because method calls cannot be distinguished by return type

A syntax error is generated when two methods have the same signature and different return types

Overloaded methods with different parameter lists can have different return types

Overloaded methods need not have the same number of parameters