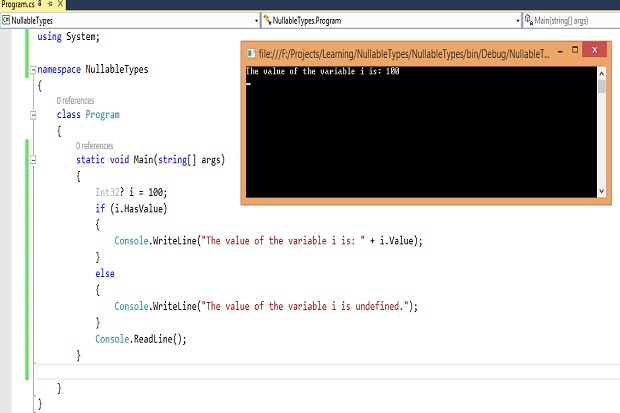
**How to work with nullable types in C#**

Take advantage of nullable types to assign 'no values' or 'null values' to value types when there is no value available



The C# language provides support for two types of data: value types and reference types. While a variable of type System.String is a reference type, a variable of type Int32 is a value type. Assigning a null value to a value type was a challenge for a long time until the concept of nullable types were introduced. You cannot assign a null value directly to a value type. You cannot assign a null value directly to a value type. You can assign a null value to a value type only by taking advantage of nullable types -- a feature added to the newer versions of .Net Framework.

Nullable types have been introduced in the C# programming language. These are instances of the struct named System.Nullable<T>. In using a nullable type, apart from the value within the permissible range for the value type, you can also have a null value. Hence, if you have a nullable Boolean variable, the possible values you can assign to the Boolean variable include true, false, or null. This feature comes in handy especially when you are working with data residing in the database and you want to assign values to value types that may or may not be null.

Note that you can only have value types as nullable -- you cannot have reference types that are nullable. Reference types cannot be nullable types because they have support for null -- that is, you can assign the value null to any reference type. A value type derives from System.ValueType and contains the data inside its own memory allocation. In other words, variables or objects or value types have their own copy of the data.

On the contrary, a reference type extends System.Object and points to a location in the memory that contains the actual data. It should be noted that any unary and binary operators that can be used on a value type can also be applied to its nullable counterpart. The following code snippet illustrates the syntax for defining a nullable type in C#.

System.Nullable<T> variable = null;

or

T? variable = null;

Here, T represents the data type of the variable. The following statement would not compile as you cannot assign a null value to a value type.

Int32 i = null;

To assign a null value to a value type, you need to take advantage of a nullable type as shown in the code snippet below.

Int32? i = null;

**The HasValue and Value properties**

There are two public read-only properties, HasValue and Value, in an instance of a nullable type. While the former is used to check if the nullable variable contains a value, the latter is used to retrieve the value contained inside the nullable variable. Note that HasValue has a default value of false. The following code listing illustrates how the HasValue and Value properties can be used.

  static void Main(string[] args)

        {

            Int32? i = 100;

            if (i.HasValue)

            {

                Console.WriteLine("The value of the variable i is: "+i.Value);

            }

            else

            {

                Console.WriteLine("The value of the variable i is undefined.");

            }

            Console.ReadLine();

        }

Note that you can also check if the value of a nullable variable is null as shown in the code snippet below.

Int32? i = 100;

if(i != null)

Console.Writeline("The value of the variable i is not null");

**Implicit and explicit conversions**

You can cast a nullable type to a non-nullable type either explicitly or by using the Value property. The following code snippet illustrates this.

Int32? i = null;

Int32 j = (Int32)i;

It should be noted that if you typecast a nullable type to a non-nullable type and the nullable type contains a null value, you would encounter InvalidOperationException.

The following code snippet illustrates how you can do an implicit cast when working with a nullable type.

Int32? i = null;

i = 100;

**The null coalescing operator (??)**

The null coalescing operator (represented as ??) is used to define the default value that would be returned when you assign a nullable type to a non-nullable type. Here is a code example that illustrates this.

Int32? i = null;

Int32 j = i ?? 100;

Console.WriteLine("The value of the variable j is: " + j);

The value of the variable i is assigned to j if i is not null. If the value of the variable i is null, an integer value 100 is assigned to the variable j. This is how the null coalescing operator works.