Int.TryParse(string str,out int result)

**Type conversion in c#**

1. Implicit Type conversion

Applicable only with value types

Directly done by the compiler

No data loss

Also called widening

Note : "high precision" often implies the ability to represent decimal numbers with great detail, it doesn't necessarily mean that the type can represent higher magnitudes or larger values. The precision and range of a numeric type are independent considerations. So, always the range is considered.

Order for implicit type conversion

Byte->short->char->int->long->decimal->float->double

Example:

byte a = 255;

int b = a;

Console.WriteLine(b+" Type after converting byte to int "+b.GetType());

long l = b;

Console.WriteLine(l + " Type after converting int to long " + l.GetType());

float f = b;

Console.WriteLine(f + " Type after converting int to float " + f.GetType());

decimal d = l;

Console.WriteLine(d + " Type after converting long to decimal " + d.GetType());

double db = f;

Console.WriteLine(db + " Type after converting float to double " + db.GetType());

Output:

255 Type after converting byte to int System.Int32

255 Type after converting int to long System.Int64

255 Type after converting int to float System.Single

255 Type after converting long to decimal System.Decimal

255 Type after converting float to double System.Double

1. Explicit Type Conversion

Performed explicitly by programmer.

May result in data loss

Narrowing

Explicit conversion operator: tells how an instance of a custom type can be explicitly converted to another type.

This conversion is explicitly used using (target source)sourceInstance

public static explicit operator TargetType(SourceType source)

{

// Define the conversion logic

}

1. Convert Class

Used for converting primitive data types.

bool result = Convert.ToBoolean("true");

byte result = Convert.ToByte("255");

char result = Convert.ToChar("A");

DateTime result = Convert.ToDateTime("2022-01-01");

decimal result = Convert.ToDecimal("123.45");

double result = Convert.ToDouble("123.45");

int result = Convert.ToInt32("42");

object result = Convert.ChangeType("123", typeof(int));

1. Parse method

Converting strings to other datatypes.

string numberString = "42";

int intValue = int.Parse(numberString);

string numberString = "42fsv";

int intValue = int.Parse(numberString);

The above two lines may result in exception.

So, it is advisable to use the TryParse method.

1. TryParse

bool b = int.TryParse("123f", out int res);

Console.WriteLine(res); // gives 0 as the parsing is not successful and default value in out parameter is 0

Console.WriteLine(b);

bool b1 = int.TryParse("123", out int res1);

Console.WriteLine(res1); // gives the converted value in the out parameter

Console.WriteLine(b1);

To avoid the issue in the parse method the tryParse is used to avoid the exceptions resulted there.

It will check if the conversion is possible or not and gives the result using the out parameter and the Boolean value is returned by executing the method.

If the parsing is successful, the returned Boolean value will be True, else it is True.

Note : the default value in the out parameter is always 0.

**IEneumerable vs IEnumerator**

IEneumerable is sugar coating of IEnumerator.

IEneumerable is implemented using the IEnumerator. It remembers state but IEnumerator doesn’t.

If we want to just loop through collection we use IEneumerable.

But, IEnumerator is a better fit when we want to remember the state and run from one function to other function.

**Extension methods**

New feature added in c# 3.0.

If we want to add a new method or feature to already existing class, then it would be violation to OCP principle in SOLID principles. So, to avoid this issue it would be one way to use the inheritance concept by declaring the child class and defining the methods there. Inheritance is a mechanism using which we can extend the functionality of a method.

There are few problems in inheritance:

1. If the parent class is sealed, then inheritance would be difficult.
2. If the original type is not a class and it is a struct then we cant apply the inheritance.
3. Even though we are extending the parent class functionality in the child class, but we are able to call the parent class and child class method with only the child class instance but not the parent class instance.

So, to avoid this issue we have Extension method concept where we can add new method to the existing class or structure without modifying the source code of the original type and the original type doesn’t require any recompilation and permissions to do so.

To create the extension methods:

Define a static class and define all the methods in the static class

Bind the static class to the class or structure.

Extension methods defined to be static , but when binded to the class or struct then they become non static.

If an extension method is defined with the same name in the original method and that method is called, then the extension method wont be imvoked, and the preference goes to original method only.

Extension method should have only one binding parameter(this Class obj).

If an extension method has n parameters then that means while calling the method only n-1 parameters are used.

This in parameters

String s=”10”;

s.ToInt(“10”);

Public static class StringExtension{

Public static int ToInt(this string ss,string s){

Return Convert.ToInt32(s); }

}

Clean coding 🡪 apply /// in xml

#region Properties //wraps them in a container

String s=10;

#endRegion

Sand castle : to do documentation automatically