Introducing C# 9: Improved pattern matching

2020-06-23 by **anthonygiretti**

Introduction

Pattern matching has been introduced in **C# 6** and has well evolved since. The latest improvement was pretty interesting on **C# 8 C# 8Miguel Bernard** (): . If you have missed all new features in **C# 8.**

In this article I will show you all the great new features of pattern matching.

Relational patterns

C# 9 allows you to use relational pattern which enables the use of **<, >, <=** and **>=** in patterns like this:

```csharp
using CSharp9Demo.Models;
using System;

namespace CSharp9Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            var product = new Product { Name = "Food", CategoryId = 4 };
            GetTax(product); // Returns 5
        }

        // Relational pattern
        private static int GetTax (Product p) => p.CategoryId switch
        {
            1 => 0,
            < 5 => 5,
            > 20 => 15,
            _ => 10
        };
    }

    public class Product
    {
        public string Name { get; set; }
        public int CategoryId { get; set; }
    }

    public class Book : Product
    {
        public string ISBN { get; get; }
    }

    public class ElectronicProduct : Product
    {
        public bool HasBluetooth { get; set; }
    }
}
```

Logical patterns

C# 9 lets you use logical operators like **'and'**, **'or'** and **'not'** , they can even be combined with relational patterns like this:

```csharp
using CSharp9Demo.Models;
using System;

namespace CSharp9Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            var product = new Product { Name = "Food", CategoryId = 4 };
            GetTax(product); // Returns 5
        }

        // Relational pattern combined with logical patterns
        private static int GetTax(Product p) => p.CategoryId switch
        {
            0 or 1 => 0,
            > 1 and < 5 => 5,
            > 20 => 15,
            _ => 10
        };
    }

    public class Product
    {
        public string Name { get; set; }
        public int CategoryId { get; set; }
    }

    public class Book : Product
    {
        public string ISBN { get; get; }
    }

    public class ElectronicProduct : Product
    {
        public bool HasBluetooth { get; set; }
    }
}
```

Not patterns

'not' logical operator can also be used in a **if** statement (it works also with a **ternary** statement), like this:

```csharp
using CSharp9Demo.Models;
using System;

namespace CSharp9Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            var product = new Product { Name = "Food", CategoryId = 4 };
            GetDiscount(product); // Returns 25
            GetDiscount2(product); // Returns 25
        }

        // Not pattern
        private static int GetDiscount (Product p)
        {
             if (p is not ElectronicProduct)
                return 25;

              return 0;
        }

        private static int GetDiscount2 (Product p) => p is not ElectronicProduct ? 25 : 0;
    }

    public class Product
    {
        public string Name { get; set; }
        public int CategoryId { get; set; }
    }

    public class Book : Product
    {
        public string ISBN { get; get; }
    }

    public class ElectronicProduct : Product
    {
        public bool HasBluetooth { get; set; }
    }
}
```

Simple type pattern

Another nice improvement of **C# 9**: Simple type pattern. When a type matches, the **underscore** symbol _ (commonly named **discard parameter**) can be omitted, thats makes the syntax lighter:

```csharp
using CSharp9Demo.Models;
using System;

namespace CSharp9Demo
{
    class Program
    {
        static void Main(string[] args)
        {
            var product = new Product { Name = "Food", CategoryId = 4 };
            GetDiscount(product); // Returns 25
        }

        // Simple type pattern
        private static int GetDiscount(Product p) => p switch
        {
            ElectronicProduct => 0, // ElectronicProduct _ => 0 before C# 9
            Book b => 75, // Book b _ => 75 before C# 9
            _ => 25
        };
    }

    public class Product
    {
        public string Name { get; set; }
        public int CategoryId { get; set; }
    }

    public class Book : Product
    {
        public string ISBN { get; get; }
    }

    public class ElectronicProduct : Product
    {
        public bool HasBluetooth { get; set; }
    }
}
```