NEW FEATURES IN C# 14

The latest versions of .NET and C# were released last November.

C#14 brings some practical new features for everyday use. Here are the top 5!

1. Extension members and extension blocks

Since C#3, it has been possible to define object extension methods in the form of a static method within a static class, using the keyword `this :`

```
public class Pizza
{
      public string Name { get; set; } = "";
      public decimal Price { get; set; }
      public bool IsVegetarian { get; set; }
}

public static class PizzaExtensions
{
      public static bool IsExpensive(this Pizza pizza, decimal threshold)
            => pizza.Price > threshold;
}
```

```
// Utilisation
var pizza = new Pizza { Name = "Reine", Price = 14 };
bool expensive = pizza.IsExpensive(12);
```

C#14 allows you to do the same thing on properties. The new version also introduces the ability to create extensions on the type itself (as opposed to the instance). They are called as a static member. They can be a method, a property, or an operator.

The concept of extension blocks appears:

```
public static class PizzaExtensions
{
    // Extension d'une instance de Pizza
    extension(Pizza source)
    {
        // Méthode d'extension
        public bool IsExpensive(decimal threshold)
            => source.Price > threshold;

        // Propriété d'extension
        public string Category
            => source.IsVegetarian ? "Végétarienne" : "Classique";
    }
```

Objectware

```
// Extension statique du type Pizza
extension(Pizza)
{
    // Méthode statique d'extension du type Pizza
    public static Pizza GetPizzaDuChef()
        => new Pizza
        {
            Name = "Pizza du chef",
            Price = 12,
            IsVegetarian = false
        };

    // Propriété statique d'extension
    public static Pizza Default
        => new Pizza
        {
            Name = "Pizza standard",
            Price = 11,
            IsVegetarian = false
        };

    // Opérateur
    public static Pizza operator +(Pizza left, Pizza right)
        => new Pizza
        {
            Name = $"{left.Name} + {right.Name}",
            Price = left.Price + right.Price,
            IsVegetarian = left.IsVegetarian && right.IsVegetarian
        };
}
```

```
// Utilisation
var pizza = new Pizza { Name = "Margherita", IsVegetarian = true };
var category = pizza.Category; // "Végétarienne"

var pizzaDuChef = Pizza.GetPizzaDuChef();
var pizzaStandard = Pizza.Default;

// Opérateur
var margherita = new Pizza
{
    Name = "Margherita",
    Price = 10,
    IsVegetarian = true
};
var pepperoni = new Pizza
{
    Name = "Pepperoni",
    Price = 12,
    IsVegetarian = false
};

Pizza combo = margherita + pepperoni;
// combo.Name = "Margherita + Pepperoni"
// combo.Price = 22
// combo.IsVegetarian = false
```

2. The keyword field in the properties

With C#13, the code for a property with a private field remained very verbose:

```
public class Pizza
{
    private decimal _price;

    public decimal Price
    {
        get => _price;
        set => _price = Math.Round(Math.Max(0, value), 2);
    }
}
```

C#14 introduces the keyword, `field` which significantly reduces the amount of code required:

```
public class Pizza
{
     public decimal Price
    {
       get => field;
       set => field = Math.Round(Math.Max(0, value), 2);
    }
}
```

3. Null conditional assignment

C#14 allows the operator to be use`?.`directly on the left side of an assignment, thus avoiding the need to write explicit null checks before assigning.

C#13:

```
if (pizza != null)
{
     pizza.Name = "Margherita";
}
```

C#14 :

```
pizza?.Name = "Margherita";
```

4. Implicit conversions to Span<T> / ReadOnlySpan<T>

C# 14 improves type `Span<T> et ReadOnlySpan<T>` support by adding implicit conversions between them and arrays (`T[]`).

```
static decimal TotalPrice(ReadOnlySpan<Pizza> pizzas)
{
    decimal total = 0;
    foreach (var pizza in pizzas)
        total += pizza.Price; return total;
}

Pizza[] order = {
    new Pizza { Name = "Margherita", Price = 10 },
    new Pizza { Name = "Reine", Price = 12 },
    new Pizza { Name = "Pepperoni", Price = 13 }
};

// NOUVEAUTÉ C# 14 : conversion implicite Pizza[] → ReadOnlySpan<Pizza>
decimal total = TotalPrice(order); //35
```

5. Modifiers in lambda parameters (out, ref, etc.)

Previously, the use of modifiers was contingent on explicitly providing the parameter type:

```
delegate bool TryParse<T>(string name, string price, out T result);

// On doit fournir tous les types
TryParse<Pizza> converter = (string name, string priceText, out Pizza
result)
    => {
            var ok = decimal.TryParse(priceText, out var price);
            result = ok
                ? new Pizza { Name = name, Price = price }
                : default!;
            return ok;
        };
```

C#14 makes it possible to do without it:

```
// Nouvelle lambda C# 14 avec `out` sans types
TryParse<Pizza> converter = (name, priceText, out result)
    => {
        var ok = decimal.TryParse(priceText, out var price);
        result = ok
            ? new Pizza { Name = name, Price = price }
            : default!;
        return ok;
    };
```

Note that the modifier `params` is not affected.

In short, other new features

- `nameof` is now usable with unconstrained generic types: `nameof(List<>)`
- `partial` can be used to split the declaration and implementation of constructors and events in a partial class.
- C#14 allows defining compound assignment operators (`+=` , `-=` , etc.) in our types.

Article written by Carole CHEVALIER

Objectware