Overloading and Extension Methods

Filip Ekberg
Principal Consultant & CEO

@fekberg | fekberg.com

How would you add methods to a class you do not own?

Extension methods!

Overloading

Different from overriding!

Overriding is when you override the method that exists in the class you inherit from

Overloading is when you share a method name

You may have completely different parameters or return type

Which Overload Is Invoked?

```
void Process(Order order)  { ... }
void Process(object order) { ... }


Process(new Order());
```

Which Overload Is Invoked?

```
void Process(Order order)  { ... }
void Process(object order) { ... }


Process(new Order());
```

Which Overload Is Invoked?

```
void Process(Order order)  { ... }
void Process(object order) { ... }


Process(new Order() as object);
```

Which Overload Is Invoked?

```
void Process(Order order)  { ... }
void Process(object order) { ... }

Process(new Order() as object);
```

Method overloading is used all the time

How to properly work with dates and times

Dates and Times in .NET

Filip Ekberg

Example of Operator Overloading

Example of Operator Overloading

```
var future = new DateTime(2025, 02, 14);
var today  = new DateTime(2025, 01, 01);
```

Example of Operator Overloading

```
var future = new DateTime(2025, 02, 14);
var today  = new DateTime(2025, 01, 01);

var result = future - today;
```

Example of Operator Overloading

```
var future = new DateTime(2025, 02, 14);
var today  = new DateTime(2025, 01, 01);

TimeSpan result = future - today;
```

Example of Operator Overloading

```
var future = new DateTime(2025, 02, 14);
var today  = new DateTime(2025, 01, 01);

TimeSpan result = future - today;
```

How many **days**, **minutes**, **seconds**
between the **dates**?

Example of Operator Overloading

```
var future = new DateTime(2025, 02, 14);
var today  = new DateTime(2025, 01, 01);

TimeSpan result = future - today;
```

DateTime has an **operator overload** for **-**

If you overload ==
you also **have to** overload !=

Filter by title
nameof expression
new operator
sizeof operator
stackalloc expression
switch expression
true and false operators
with expression
Operator overloading
Statements
Special characters
Attributes read by the compiler
Unsafe code and pointers
Preprocessor directives
Compiler options
XML documentation comments
Compiler messages
Specifications
Download PDF

Overloadable operators

The following table shows the operators that can be overloaded:

Expand table

Operators	Notes
+x, -x, !x, ~x, ++, --, true, false	The `true` and `false` operators must be overloaded together.
x + y, x - y, x * y, x / y, x % y, x & y, x | y, x ^ y, x << y, x >> y, x >>> y	
x == y, x != y, x < y, x > y, x <= y, x >= y	Must be overloaded in pairs as follows: `==` and `!=`, `<` and `>`, `<=` and `>=`.

Non overloadable operators

The following table shows the operators that can't be overloaded:

Expand table

Operators	Alternatives

Adding an Operator Overload

Adding an Operator Overload

```
public
```

Adding an Operator Overload

```
public static
```

Adding an Operator Overload

```
public static Order
```

Adding an Operator Overload

```
public static Order operator
```

Adding an Operator Overload

```
public static Order operator +(Order first, Order second)
```

Adding an Operator Overload

```
public static Order operator +(Order first, Order second)
{
    var items = new List<Item>(first.LineItems);
    items.AddRange(second.LineItems);

    return new()
    {
        LineItems             = items,
        IsReadyForShipment    = first.IsReadyForShipment,
        ShippingProvider      = first.ShippingProvider,
        Total                 = first.Total + second.Total
    };
}
```

Don’t overuse operator overloading

It may result in unnecessary complexity!

When to Use Operator Overloading

When using the types with an operator is natural

Custom, or complex mathematical representations

Comparison

=, !-, <, <=, >, <=

Looking for value based equality?

Wait until you learn about record types!

Could This Be Converted into the Given Type?

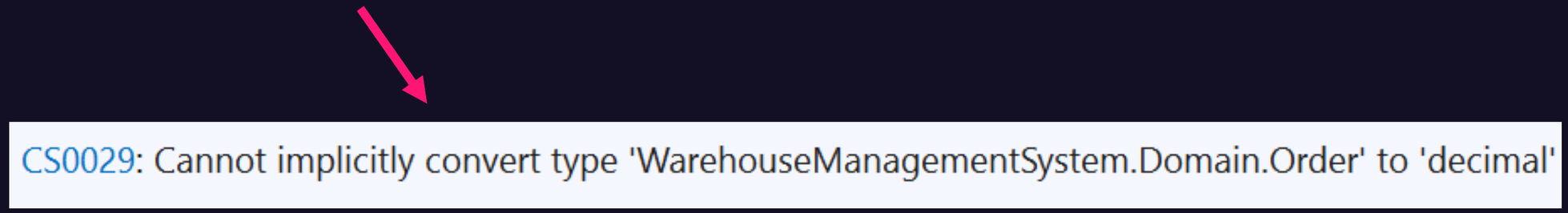
```
Order order = new();

decimal total = order; // Compiler error!
```

Could This Be Converted into the Given Type?

```
Order order = new();

decimal total = order; // Compiler error!
```



Conversion Operators

Implicit

A safe conversion! The compiler can implicitly determine what type to convert to

Explicit

A conversion that potentially loses data and conversion could throw an exception

Adding Conversion Operators

Adding Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}
```

Adding Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}

public static explicit operator Order(List<Item> items)
{
    return new() { LineItems = items.ToArray() };
}
```

Using the Conversion Operators

```
Order order = new();
decimal total = order;        // Uses the implicit conversion


List<Item> items = GetItems();
Order order = (Order)items;  // Uses the explicit conversion
```

Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}

public static explicit operator Order(List<Item> items)
{
    return new() { LineItems = items.ToArray() };
}
```

The resulting type of the conversion

Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}

public static explicit operator Order(List<Item> items)
{
    return new() { LineItems = items.ToArray() };
}
```

The type you convert from

Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}
```

Conversion Operators

```
public static implicit operator decimal(Order order)
{
    return order.Total;
}


Order order = new();

decimal total = order;
```

You are not required to implement a conversion both ways!

In many cases it may make sense to do so

Subclassing might not be allowed

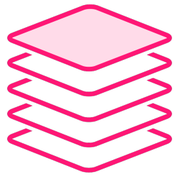
Extension Methods

”Enable you to "add" methods to existing types without creating a new derived type, recompiling, or otherwise modifying the original type”

docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/extension-methods

Extension Methods

Looks like an instance method
Defined in a completely different library than the original type



Lets you add method overloads
You can create methods with the same names as the type you extend as long as they have different parameters

Only for types you can’t control
Don’t introduce extension methods for types in your project, or types that you could otherwise change

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{




}
```

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {



    }
}
```

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {


    }
}
```

Imagine that Order comes from a completely different project that you do not control

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(                    )



    }
}
```

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(this Order order)



    }
}
```

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(this Order order)


    }
}
```

This indicates that it is an **extension method** for the type **Order**

Creating an Extension Method

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(this Order order)
        {
            return $"This is the report for {order.OrderNumber}";
        }
    }
}
```

Using an Extension Method

```
using namespace WarehouseManagementSystem.Domain.Extensions;

Order order = new();

var report = order.GenerateReport();
```

Using an Extension Method

```
using namespace WarehouseManagementSystem.Domain.Extensions;

Order order = new();

var report = order.GenerateReport();

var report = OrderExtension.GenerateReport(order);
```

Extension methods can **only access public properties** and **methods** on the **instance**

Extension Methods with Parameters

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(this Order order,
                                             string recipient) { ... }
    }
}
```

Extension Methods with Parameters

```
namespace WarehouseManagementSystem.Domain.Extensions
{
    public static class OrderExtensions
    {
        public static string GenerateReport(this Order order,
                                             string recipient) { ... }
    }
}

Order order = new();

var report = order.GenerateReport("Filip Ekberg");
```

Extension methods have the lowest priority!

The compiler will never choose an extension method if a better alternative exists on the type

Using an Extension Method

```
using namespace WarehouseManagementSystem.Domain.Extensions;

Order order = new();

var report = OrderExtensions.GenerateReport(order);
```

How would you **build** a reusable **(generic) extension method**?

Consider that the **domain library** in the solution is **installed through** the **company NuGet server**

Only build **extension** methods **for types outside** of your **control**

What Makes It an Extension Method

It is defined in a static class

It is a static method

The first parameter uses the this keyword

You can share your library as a NuGet package

Introducing System.Linq

Powerful set of extension methods

Extends for example IEnumerable<T> and IQueryable<T>

LINQ

“Language-Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language.”

Example:

```
order.LineItems.Where(item => item.Price > 60)
```

docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

Example of Intersect

```
var first   = new[] { 1, 2, 3, 4 };
var second  = new[] { 2, 3 };

var common = first.Intersect(second);
```

Extension methods can be a **powerful** addition and help **others** that have **similar needs**

Example of a Fluent Chain

Example of a Fluent Chain

```
var cheapestItems = order.LineItems
```

Example of a Fluent Chain

```
var cheapestItems = order.LineItems
                        .Where(item => item.Price > 60)
```

Example of a Fluent Chain

```
var cheapestItems = order.LineItems
                        .Where(item => item.Price > 60)
                        .OrderBy(item => item.Price)
```

Example of a Fluent Chain

```
var cheapestItems = order.LineItems
                        .Where(item => item.Price > 60)
                        .OrderBy(item => item.Price)
                        .Take(5);
```

Overloading

Overloading is when you share a method name

Requires different parameters and could have a different return type

Can be an extension method

A method overload can be defined as an extension method but the compiler will always try to use methods on the type

Example of Operator Overloading

```
public static Order operator +(Order first, Order second) { ... }
```

Example of Operator Overloading

```
public static
```

Example of Operator Overloading

```
public static Order
```

Example of Operator Overloading

```
public static Order operator
```

Example of Operator Overloading

```
public static Order operator +(Order first, Order second) { ... }
```

Check the **documentation** to follow **exactly** how to **overload** a **specific operator**

Use **explicit conversion** if your conversion can **throw** an exception or **lose** data

Example Extension Methods

Example Extension Methods

```
public static class IEnumerableExtensions
{
    public static IEnumerable<T> Find<T>(this IEnumerable<T> source,
                                         Func<T, bool> isMatch)
    { ... }
}
```

Example Extension Methods

```
public static class IEnumerableExtensions
{
    public static IEnumerable<T> Find<T>(this IEnumerable<T> source,
                                         Func<T, bool> isMatch)
    { ... }
}

public static class OrderExtensions
{
    public static string GenerateReport(this Order order)
    { ... }
}
```

Up Next:

Anonymous Types