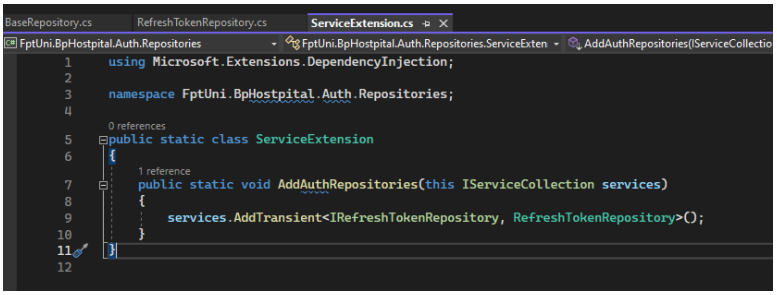
Các kĩ thuật trong asp.net

▼ Map Extension Method in ASP.NET Core Application

- Cách làm này giúp configure ko bắt buộc phải trong Program.cs



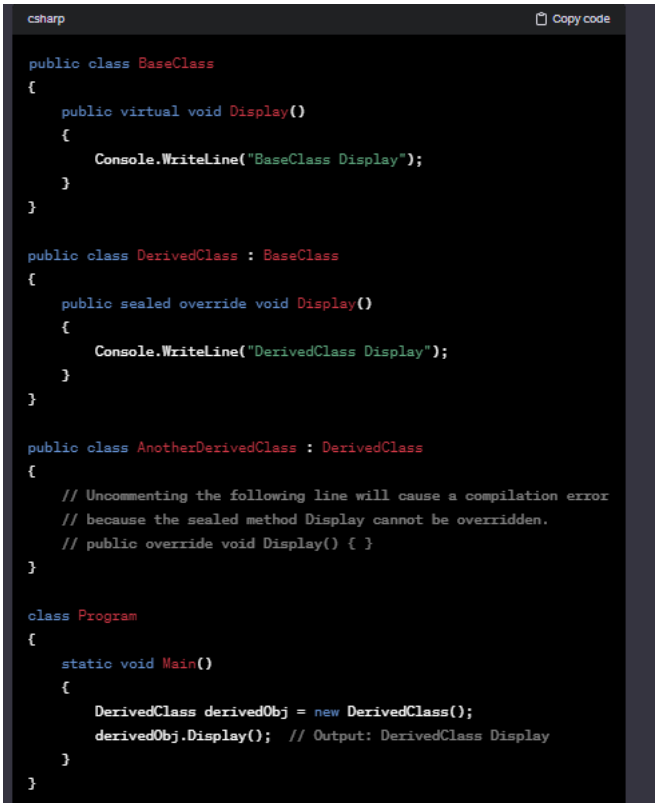
▼ Base trong ASP.NET

Base ở đây được gọi tới constructure cha . Base trong class Mammal được gọi tới constructure Animal

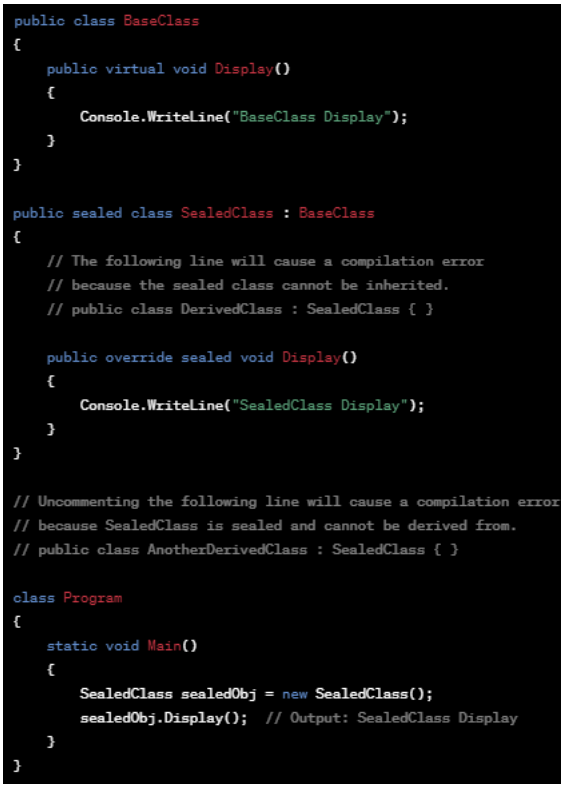
Kĩ Thuật Enum mới (SMART ENUM) vì nó sử dụng luôn cả OOP

▼ `sealed` trong C#

Có cách dùng là để → ngăn chặn **kế thừa hoặc ghi đè bởi các lớp khác**



```csharp
public class BaseClass
{
    public virtual void Display()
    {
        Console.WriteLine("BaseClass Display");
    }
}

public class DerivedClass : BaseClass
{
    public sealed override void Display()
    {
        Console.WriteLine("DerivedClass Display");
    }
}

public class AnotherDerivedClass : DerivedClass
{
    // Uncommenting the following line will cause a compilation error
    // because the sealed method Display cannot be overridden.
    // public override void Display() { }
}

class Program
{
    static void Main()
    {
        DerivedClass derivedObj = new DerivedClass();
        derivedObj.Display();  // Output: DerivedClass Display
    }
}
```



```
public class BaseClass
{
    public virtual void Display()
    {
        Console.WriteLine("BaseClass Display");
    }
}

public sealed class SealedClass : BaseClass
{
    // The following line will cause a compilation error
    // because the sealed class cannot be inherited.
    // public class DerivedClass : SealedClass { }

    public override sealed void Display()
    {
        Console.WriteLine("SealedClass Display");
    }
}

// Uncommenting the following line will cause a compilation error
// because SealedClass is sealed and cannot be derived from.
// public class AnotherDerivedClass : SealedClass { }

class Program
{
    static void Main()
    {
        SealedClass sealedObj = new SealedClass();
        sealedObj.Display();  // Output: SealedClass Display
    }
}
```

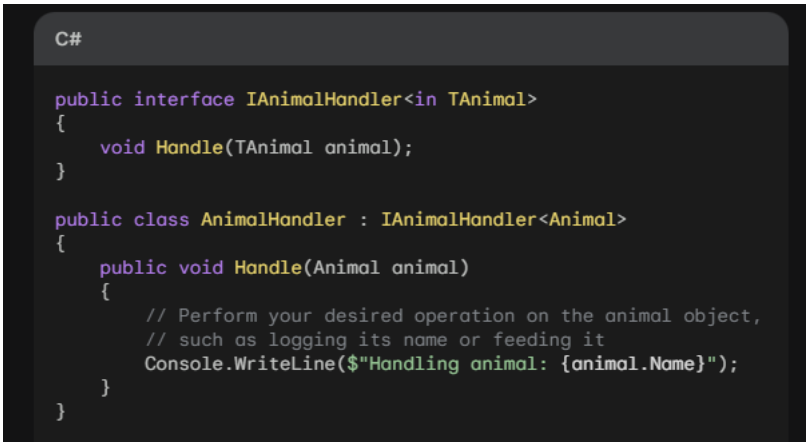
▼ `out` - Covariance (Sự Biến Thiên):

- tôi sử dụng **Out** ở đây là khi trong interface tôi chỉ muốn **sử dụng cho hàm trả về**

```
public interface IAnimalCreator<out TAnimal>
{
    TAnimal CreateAnimal(); // Creates and returns a new anima
}

public class AnimalHandler : IAnimalCreator<Animal>
{
    public Animal CreateAnimal()
    {
        // Create and return a new Animal object
        return new Animal { Name = "Lion Cub" };
    }
}
```

▼ In Contravariance (Sự Đối Ngược):

tôi sử dụng **In** ở đây là khi trong interface tôi chỉ muốn **sử dụng cho hàm ko trả về**



vậy cách giải quyết ở đây sẽ là dùng **Generics**

▼ `while` khi : kiểm tra điều kiện trước khi thực hiện lặp

while (condition)
{
// Code sẽ thực hiện miễn là điều kiện là đúng.
}

▼ Sử dụng vòng lặp `do-while` khi

Bạn muốn thực hiện lặp ít nhất một lần.

▼ Thế nào là Value type và Ref Type

—→ Value type là giá trị lưu trong địa chỉ

—→ Ref type là chứa references (địa chỉ)

- Type change `delegate`
- Type variables `interface`
- Type change `dynamic`
- Type change `object`
- Type change `string`

Thế nào là Anonymous type (kiểu vô dang) và Dynamic types (Kiểu động)

▼ Anonymous type (kiểu vô dang) :

- Ko có tên
- Read Only

```
var myProfile = new {
    name = "XuanThuLab",
    age = 20,
    skill = "ABC"
};
```

▼ Dynamic types (Kiểu động):

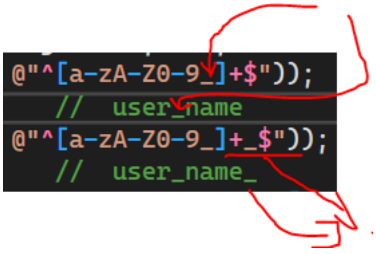
Kiểu động (Dynamic types) trong ASP.NET và C# nói chung cho phép bạn làm việc với biến mà không cần biết chính xác kiểu dữ liệu của nó cho đến khi thực thi chương trình. Điều này làm cho C# trở nên linh hoạt hơn và có thể tương tác tốt hơn với các ngôn ngữ khác.

Ví dụ:

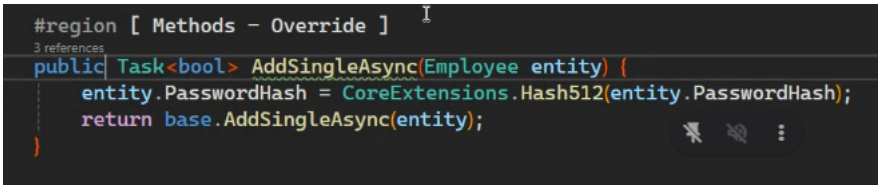
```
dynamic x = 20;
Console.WriteLine(x); // Output: 20

x = "Hello World!";
Console.WriteLine(x); // Output: Hello World!

x = DateTime.Now;
Console.WriteLine(x); // Output: The current date and time
```

Trong ví dụ trên, biến kiểu động `x` ban đầu được gán giá trị là số nguyên, sau đó là một chuỗi, và cuối cùng là một đối tượng DateTime. Mỗi lần gán, biến `x` thay đổi kiểu dữ liệu của mình mà không gặp bất kì lỗi nào.

Một số điều chú ý khi dùng Regrex



Virtual cho phép override



▼ Cách dùng out bool _

`_` chỉ là một biến đại diện không quan trọng

`out` keyword trong các phương thức như này thường được gọi là sử dụng "discard", nghĩa là bỏ qua giá trị được trả về.

AsNoTracking và Tracking

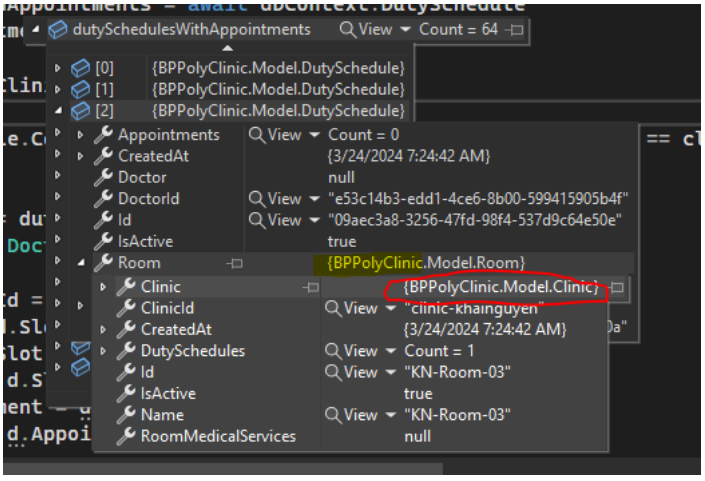
SQL SERVER

dotnet ef dbcontext scaffold "Persist Security Info=True;User ID=sa;Password=1234567890;Initial Catalog=TicketTravel;Data Source=DESKTOP-GRU7MA8\SQLEXPRESS;Connection Timeout=100000" Microsoft.EntityFrameworkCore.SqlServer -o Database

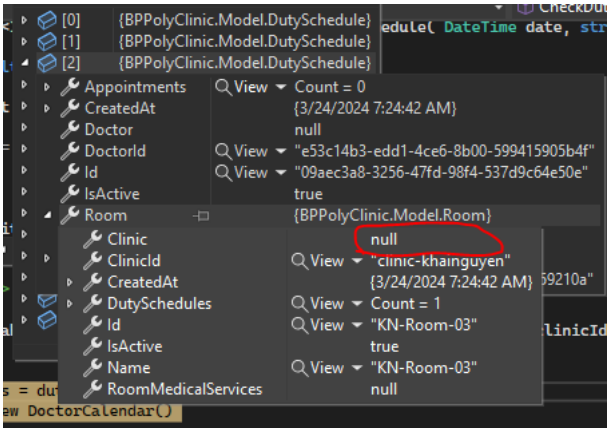
Server=DESKTOP-GRU7MA8\SQLEXPRESS;Initial Catalog=;Persist Security Info=False;User ID=sa;Password=1234567890;MultipleActiveResultSets=False;Encrypt=True;TrustServerCertificate=true;Connection Timeout=30;

```
  var dutySchedulesWithAppointments = await dbContext.DutySchedule
.Include(x => x.Appointments.Where(x=>x.IsActive))
.Include(y=> y.Room)
  .ThenInclude(a => a.Clinic)
.Include(z=> z.Slot)
.Where(x=> slotAvailable.Contains(x.SlotId)&& x.IsActive && x.Room.ClinicId == clinicId)
.ToListAsync();
```

TRONG code do có thenInclude nên Clinic ở đây có giá trị



Nếu như ko có thenInclude thì Clinic ở đây sẽ trả về Null



Global Exception

https://code-maze.com/dotnet-use-iexceptionhandler-to-handle-exceptions/?fbclid=IwZXh0bgNhZW0CMTAAAR3vBcARVSih4BOAI9lkEs_3qrdEWZgQ6hVpHfoXJ2QRmgprKH7s5a6Zt08_aem_Acd0S