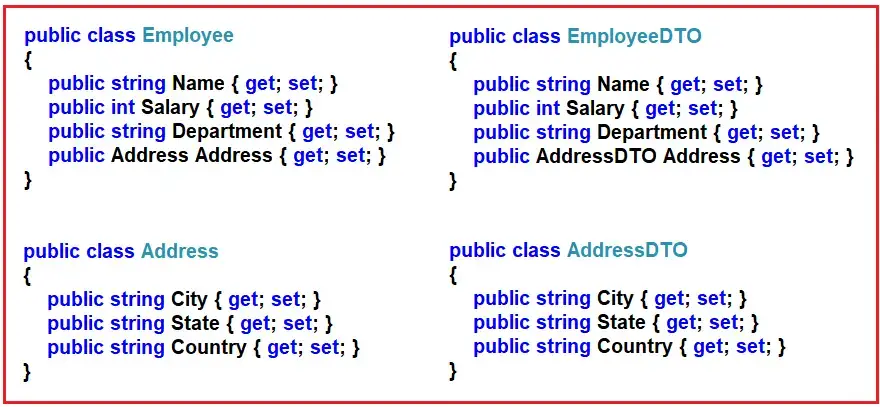
**AutoMapper Complex Mapping in C#**



Mapping

//Provide all the Mapping Configuration

var config = new MapperConfiguration(cfg => {

//Configuring Address and AddressDTO

cfg.CreateMap<Address, AddressDTO>();

//Configuring Employee and EmployeeDTO

cfg.CreateMap<Employee, EmployeeDTO>();

});

نکته ای که دارد این است که باید نوع complex را نیز به یک دیگر مپ کرده باشیم.

در مثال بالا نام property کلاس ادرس در هر دو یکی است و از قبل نیز مپ شده پس مشکلی نیست اما در حالتی که ما از قبل مپ کردیم اما نام property ها در 2 کلاس متفاوت باشد در این صورت به خطا می خوریم باید نام یک دو property را به یک دیگر بشناسانیم.

//Configuring Employee and EmployeeDTO

//Provide Mapping Information for AddressDTO and address

cfg.CreateMap<Employee, EmployeeDTO>()

.ForMember(dest => dest.AddressDTO, act => act.MapFrom(src => src.Address)); ;

از قبل ها این 2 کلاس به هم مپ شده اند پس بنابراین دیگر مشکلی نداریم.

در صورتی که نام اعضای کلاس پیچیده با هم متقاوت باشد نیاز است که آن 2 را نیز به هم مپ کنیم.

//Configuring Address and AddressDTO

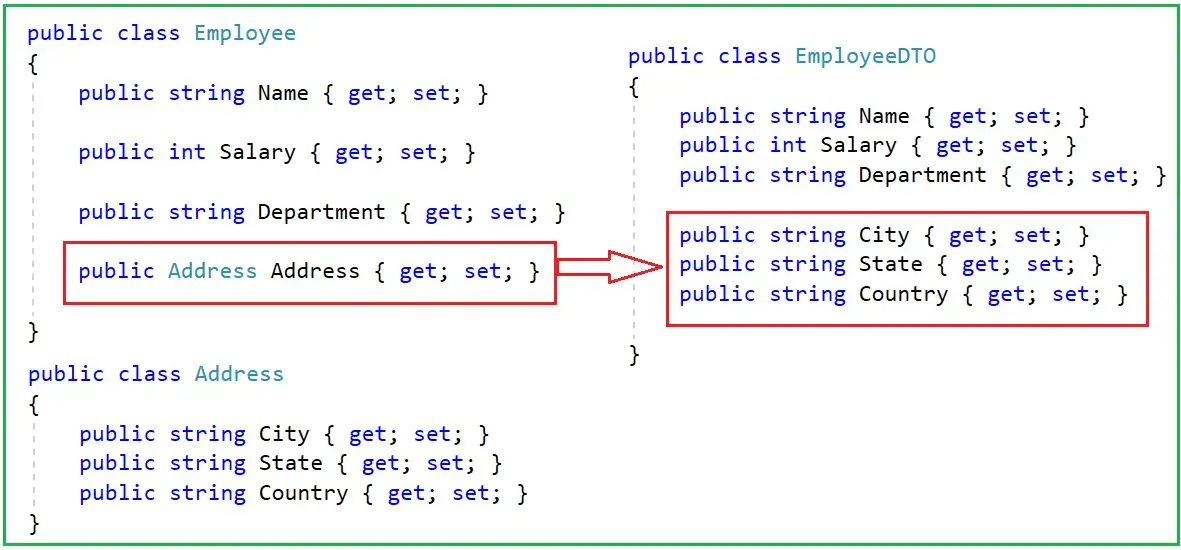
var config = new MapperConfiguration(cfg => {

cfg.CreateMap<Address, AddressDTO>()

.ForMember(dest => dest.EmpCity, act => act.MapFrom(src => src.City))

.ForMember(dest => dest.EmpState, act => act.MapFrom(src => src.State));

# How to Map Complex Type to Primitive Type using AutoMapper in C#



ره حل

//Provide all the Mapping Configuration

//Configuring Employee and EmployeeDTO

//Provide Mapping Information for City Property

//Provide Mapping Information for State Property

//Provide Mapping Information for Country Property

var config = new MapperConfiguration(cfg => {

cfg.CreateMap<Employee, EmployeeDTO>()

.ForMember(dest => dest.City, act => act.MapFrom(src => src.Address.City))

.ForMember(dest => dest.State, act => act.MapFrom(src => src.Address.State))

.ForMember(dest => dest.Country, act => act.MapFrom(src => src.Address.Country));

});

و برای حالت برعکس که بخواهیم primitive را به complex مپ کنیم از روش زیر استفاده می کنیم.

var config = new MapperConfiguration(cfg => {

cfg.CreateMap<Employee, EmployeeDTO>()

.ForMember(dest => dest.Address, act => act.MapFrom(src => new Address()

{

City = src.City,

State = src.State,

Country = src.Country

}));

});

## **AutoMapper Reverse Mapping in C# with Examples**

این کاری نیست جز two way mapping به این معنی که بخواهیم a را به b و b را به a مپ کنیم از این روش استفاده میکنیم.

فقط کافی است که در انتها از متد ReveseMap در مپینگ خود استفاده کنیم.

//Configuring AutoMapper

//Mapping Order with OrderDTO

//This will make the Mapping as Bi-Directional

//Now, we can also Map OrderDTO with Order Object

var config = new MapperConfiguration(cfg => {

cfg.CreateMap<Order, OrderDTO>()

.ForMember(dest => dest.OrderId, act => act.MapFrom(src => src.OrderNo))

.ForMember(dest => dest.Customer, act => act.MapFrom(src => new Customer()

{

CustomerID = src.CustomerId,

FullName = src.Name,

Postcode = src.Postcode,

ContactNo = src.MobileNo

})).ReverseMap();

});

نکته ای که باید در نظر گرفت این است که می توان در ادامه ی متد REVERSEMAP نیز مجدد مپ نوشت.

# AutoMapper in C#

فرض کنید یک mapping را به صورت سنتی انجام داده اید کلاس a به کلاس b حالا فرض کنید که چندین جا این کار انجام شده و اکنون نیاز است به کلاس A تعدادی property جدید اضافه کنید اکنون لازم است تمام قسمت های که این mapping انجام شده دستخوش تغییر شود ! راه حل ساده شده استفاده از auto mapper است.

یک کتابخانه open source است که برای مپ کردن object ها به یک دیگر کاربرد دارد مثال تبدیل dto به viewModel

مثال

**public** **static** Mapper InitializeAutomapper**()**

**{**

//Provide all the Mapping Configuration

var config = new MapperConfiguration**(**cfg =**>**

**{**

//Configuring Employee and EmployeeDTO

cfg.CreateMap**<**Employee, EmployeeDTO**>()**;

//Any Other Mapping Configuration ....

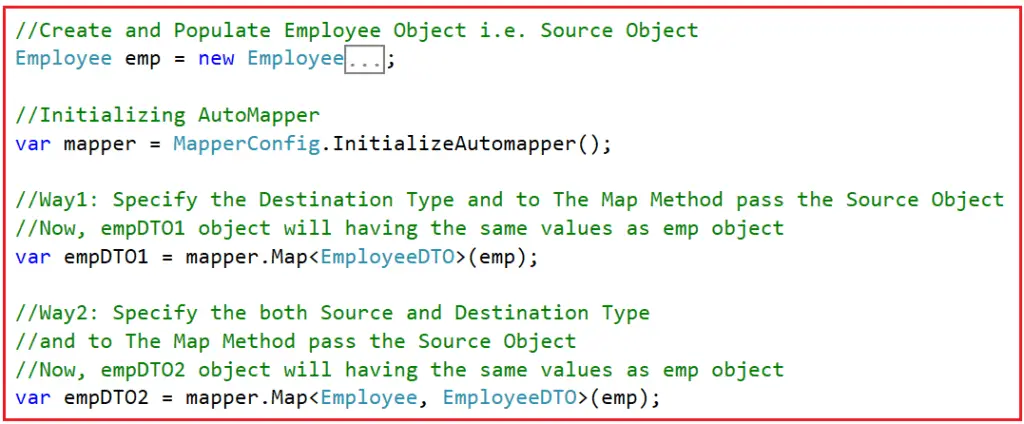
**})**;

//Create an Instance of Mapper and return that Instance

var mapper = new Mapper**(**config**)**;

**return** mapper;

**}**



نکته : به صورت پیش فرض مپینگ بین پروپرتی های صورت می گیرد که دارای نام یکسان هستند.

Mapping when names are different

cfg.CreateMap**<**Employee, EmployeeDTO**>()**

//Provide Mapping Configuration of FullName and Name Property

.ForMember**(**dest =**>** dest.FullName, act =**>** act.MapFrom**(**src =**>** src.Name**))**

//Provide Mapping Dept of FullName and Department Property

.ForMember**(**dest =**>** dest.Dept, act =**>** act.MapFrom**(**src =**>** src.Department**))**;

**Advantages and Disadvantages of using AutoMapper in C#**

کاهش تکرار کد – افزایش خوانایی و راحت تر شدن نگه داری – سازگاری در مپ نوع های داده پیچیده – استفاده آسان.

**Disadvantages**

می تواند دارای overhead یا سربار باشد به خصوص اگر درست کانفیگ نشده باشد – پیچیدگی در مپ کردن نوع داده های پیچیده – مناسب برای سناریو های روتین و آسان است و در سناریو های پیشرفته نیاز به مهارت بیشتری دارد – اگر قوانین نام گذاری یکسان رعایت نشده باشد نیاز است که به زمان جهت نوشتن مپینگ طولانی تر است –

**در حالت های ساده بهتر است مپینگ دستی استفاده شود که کارآمد تر و سر راست تر است.**

**When to use AutoMapper in C#?**

* **Large Projects with Complex Object Models: In large-scale applications with complex domain models and Data Transfer Objects (DTOs), AutoMapper can save significant time and effort by automating the mapping process.**
* **Projects with Repetitive Mapping Code: If your project involves a lot of repetitive and tedious mapping code (like copying properties from one object to another), AutoMapper can help reduce this boilerplate code, making your codebase cleaner and easier to maintain.**
* **When Consistency in Mapping Rules is Important: AutoMapper helps maintain consistent mapping rules across your application. This is particularly useful in team environments or large projects where different developers might otherwise implement mappings inconsistently.**
* **CRUD Operations in Layered Architecture: In applications with a clear separation of concerns, such as MVC applications, AutoMapper is valuable for transforming data between layers (e.g., from data access objects to business logic objects or business logic objects to view models).**
* **When You Need Customizable Mapping Logic: AutoMapper is not just for straightforward mappings; it also allows for complex and custom mapping logic. This is useful when you have specific rules for how certain properties should be mapped or transformed.**
* **In Applications Where Development Speed is a Priority: Rapid application development can benefit from AutoMapper, as it speeds up writing and updating code involving object mappings.**

**autoMapper may not be the best choice**

* **For Simple Projects:**If your project is simple, with few mappings, or if the mappings are straightforward, the overhead of introducing a third-party library might not be justified.
* **Performance-Critical Applications:**Be cautious in performance-sensitive applications, as AutoMapper can introduce some overhead, especially if not configured optimally.
* **When Explicit Control is Preferred**: In cases where you need or prefer explicit control over every aspect of the mapping, or if you want to avoid the abstraction

# AutoMapper Conditional Mapping in C#

برقراری شرطی جهت این که مپینگ در صورتی رخ دهد که نام به حرف a شروع شود یا در صورتی که مقدار amoun از عدد 100 بزرگتر باشد.

**Ternery operator**

**Condition mehtod**

cfg.CreateMap**<**Product, ProductDTO**>()**

//If the Name Start with A then Map the Name Value else Map the OptionalName value

.ForMember**(**dest =**>** dest.ItemName, act =**>** act.MapFrom**(**src =**>**

**(**src.Name.StartsWith**(**"A"**)** ? src.Name : src.OptionalName**)))**

//Map the quantity value if its greater than 0

.ForMember**(**dest =**>** dest.ItemQuantity, act =**>** act.**Condition(**src =**>** **(**src.Quantity **>** 0**)))**

//Map the amount value if its greater than 100

.ForMember**(**dest =**>** dest.Amount, act =**>** act.**Condition(**src =**>** **(**src.Amount **>** 100**)))**;

**})**;

**AutoMapper Ignore Method in C#**

مواقعی است که نمی خواهیم برخی از property های منبع به مقصد مپ شوند کتابخانه atuoMapper به صورت پیش فرض تمامی property های که درای نام یکسان هستند را به یک دیگر مپ می کند اما راه حل استفاده از متد Ignore() است.

//Ignoring the Address property of the destination type

.ForMember**(**dest =**>** dest.Address, act =**>** act.Ignore**())**;

**How to Ignore Multiple Properties**

در مواقعی که تعداد property های زیاد است از روش دیگری می توان برای ignore کردن مپ استفاده کرد به این صورت که یک attribute از نوع notMap ایجاد کرده و بر روی property های که نمی خواهیم مپ شوند می گذاریم سپس یک extenstion متد برای فعال کردن آن بر روی کلاس مپ شده ایجاد می کنیم.

**Attribute**

**public** **class** NoMapAttribute : System.Attribute

**{**

**}**

**Ex method**

**public** **static** **class** IgnoreNoMapExtensions

**{**

//Method is Generic and Hence we can use with any TSource and TDestination Type

**public** **static** IMappingExpression**<**TSource, TDestination**>** IgnoreNoMap**<**TSource, TDestination**>(**

this IMappingExpression**<**TSource, TDestination**>** expression**)**

**{**

//Fetching Type of the TSource

var sourceType = typeof**(**TSource**)**;

//Fetching All Properties of the Source Type using GetProperties() method

**foreach** **(**var property in sourceType.GetProperties**())**

**{**

//Get the Property Name

PropertyDescriptor descriptor = TypeDescriptor.GetProperties**(**sourceType**)[**property.Name**]**;

//Check if Property is Decorated with the NoMapAttribute

NoMapAttribute attribute = **(**NoMapAttribute**)**descriptor.Attributes**[**typeof**(**NoMapAttribute**)]**;

**if** **(**attribute != **null)**

**{**

//If Property is Decorated with NoMap Attribute, call the Ignore Method

expression.ForMember**(**property.Name, opt =**>** opt.Ignore**())**;

**}**

**}**

**return** expression;

**}**

**}**

**Mapping**

**//Configuring AutoMapper**

**var config = new MapperConfiguration(cfg =>**

**{**

**cfg.CreateMap<Employee, EmployeeDTO>()**

**.IgnoreNoMap();**

**});**

**Fixed and Dynamic Values in Destination Property in AutoMapper**

با استفاده از متد MapFrom می توانیم مقدار static ست کنیم.

با استفاده از این متد می توانیم خروجی یک متد را به property مپ کنیم.

cfg.CreateMap**<**PermanentAddress, TemporaryAddress**>()**

//Using MapFrom Method to Store Static or Hard-Coded Value in a Destination Property

.ForMember**(**dest =**>** dest.CreatedBy, act =**>** act.MapFrom**(**src =**>** "System"**))**

//Before AutoMapper 8.0, to Store Static Value use the UseValue() method

//.ForMember(dest => dest.CreatedBy, act => act.UseValue("System"))

//Using MapFrom Method to Store Dynamic Value in a Destination Property

//Here, we are calling GetCurrentDateTime method which will return a dynamic value

.ForMember**(**dest =**>** dest.CreatedOn, opt =**>** opt.MapFrom**(**src =**>** GetCurrentDateTime**()))**

////Before AutoMapper 8.0, To Store Dynamic value use ResolveUsing() method

//.ForMember(dest => dest.CreatedBy, act => act.ResolveUsing(src =>

//{

// return DateTime.Now;

//}))

.ReverseMap**()**;

**})**;

**Null Substitution in Automapper:**

این امکان را می دهد که یک گزینه برای مقدار مقصد داشته باشیم زمانی که منبع ما مقدار null جایگزینی برای آن در نظر گرفته شود. به جای مپ شدن مقدار null از منبع به مقصد مقدار در نظر گرفته شده توسط ما ست می شود.

در مثال زیر در صورتی که مقدار منبع null باشد مقدار N/A را ست می کند

.ForMember**(**dest =**>** dest.Address, act =**>** act.NullSubstitute**(**"N/A"**))**

PermanentAddress permAddress = new PermanentAddress**()**

**{**

Name = "Pranaya",

Address = **null**,

CreatedBy = "Dot Net Tutorials",

CreatedOn = DateTime.Now

**}**;

var TempAddress = mapper.Map**<**PermanentAddress, TemporaryAddress**>(**permAddress**)**;