# Automapper

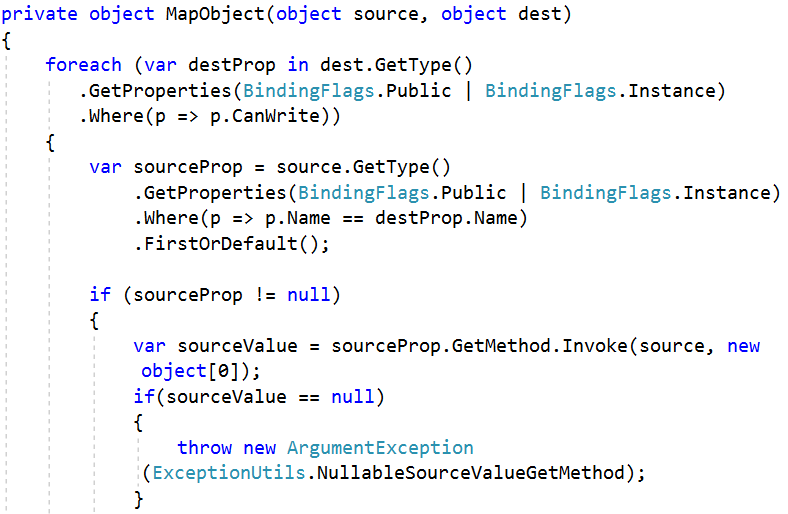


In this workshop we are going to implement our own custom auto mapper. We can use the mapper in every project we want only with adding reference to the library with the auto mapper. You already know reflection, and this will be very important in this exercise.

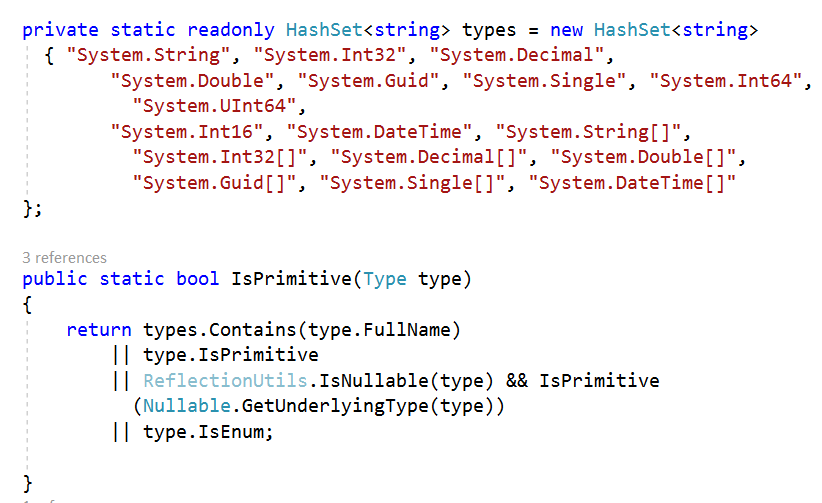
Create new **library .net core** project and create **Mapper**, **MapperConfiguration, ExceptionUtils** and **ReflectionUtils** classes.

## Mapper class

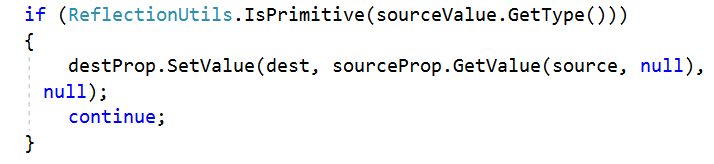
In this class the first thing we need is a private method which returns object. The returned object will be the destination object which we want to map from the source one. The method accepts object source and object destination parameters. You can call this method MapObject(), but you can choose another name if you prefer so. The first thing to do is to go through each destination properties and after that to find a property in the source object with the same name as the current destination property. If we find such we must get its value and check for its type. If the source value is null we have to throw ArgumentException with message “The get method of the source value provided is null” you can create a constant value in the ExceptionUtils class with this message.



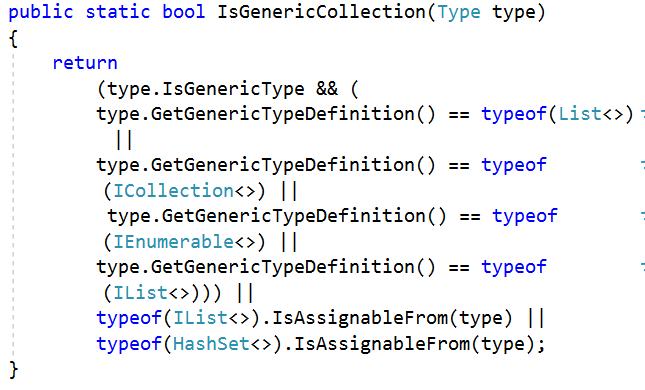
If the source value is different from null, we have to go to our ReflectionUtils class and create a method called IsPrimitive which will return Boolean result and will accept source type as parameter. All this method does is to check if the type is primitive or collection of primitives (int, string, double, float, decimal, int[], string[]…etc).



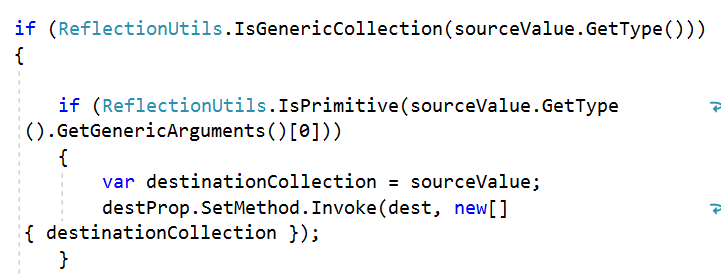
If the source value is primitive, we just set the destination property value to the one from the source.



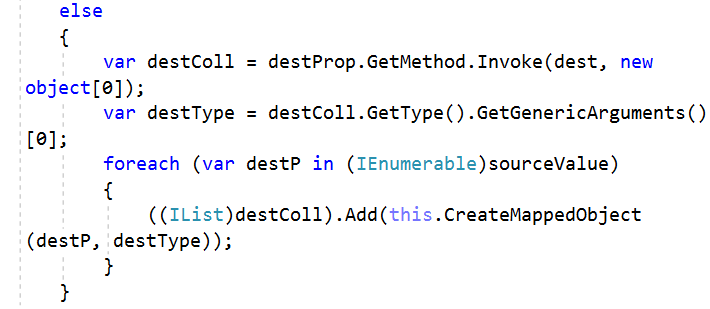
If the source value type is generic collection (go to ReflectionUtils class and create Boolean method which returns true if the collection provided is generic type)



and if its generic arguments are primitive types we have to create new destination collection and set it to the source value. After that we must set the destination property to this collection just by invoking its set method.

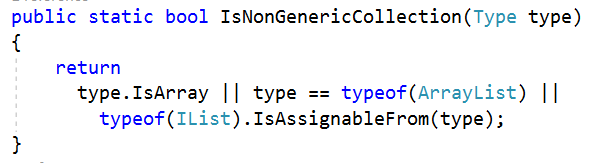


If the generic collection is not primitive it might be of type object and we have to go through each properties in this object (class Product for example with two properties for name and price) and map them.

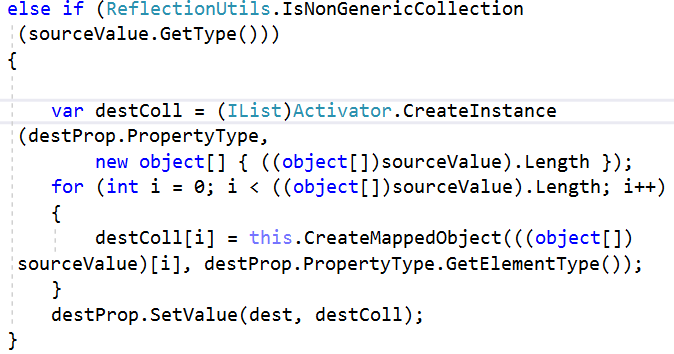


As you can see is the photo above we called a method which does not exist yet. For now, you should know that this method returns a destination object (we will implement it soon)

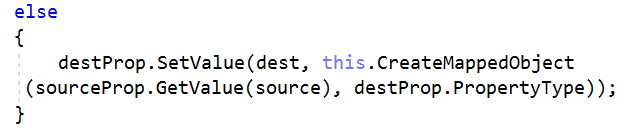
The last thing to check in this method is if the source value is of non-generic type (for example it can be list or an array). You can create a new Boolean method in the ReflectionUtils class which checks for this.



In this case we have to cast the source value to object[] and go through that collection and again we have to call the same CreateMappedObject method. And when we are finish with adding elements from the source value to the new destination collection we must set the destination property to that collection.



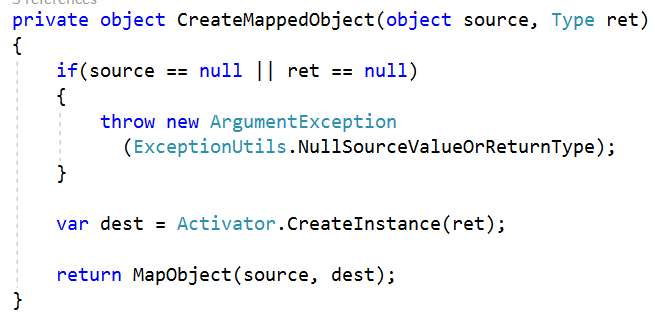
If the source value type is not one from the above ones we just set the destination property value to the source one



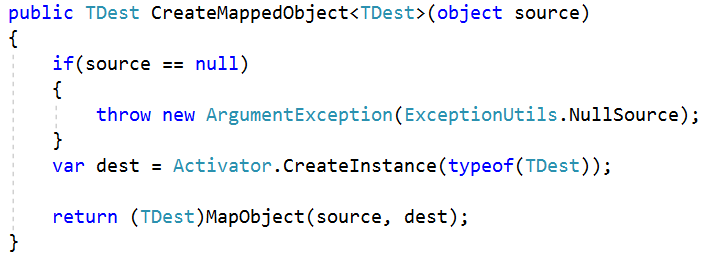
When everything is fine our method just returns the destination object.



Now its time to implement a private method called CreateMappedObject which accepts object source and the return type of the destination object. If the source provided or the return type are null we just throw an ArgumentException with message “The source or the return type is null”. Otherwise create an instance of the return type and call recursively the MapObject method with the source and the instance of the return type (I’m sure you already heard about recursion, and what is its purpose. If you don’t, after we are ready with the project you can test it and debug it to find out what the recursion does).

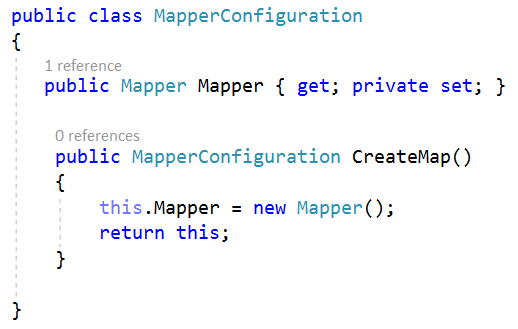


The last method to implement is the same as the above one but the only one difference is that it works with generic destination object and accepts only the source object as parameters. This method must be public because when we want to create mapping between two objects we will use this method. If the source provided is null we have to throw ArgumentException with message “The source provided is null”. Finally create an instance for the destination type and map the source object and the instance of the destination.



## MapperConfiguration class

This is the class where we will create an instance of the Mapper. To do that create a property of type Mapper and lastly create a method CreateMap() which instantiates the Mapper property and returns the MapperConfiguration.



*Finally, we are done with our custom auto mapper. You can test your work. Have fun!*