Decorator Pattern

Using The Decorator Pattern At Labman

Arthur Clarkson

**5th September 2024

The Decorator is a structural design pattern that allows you to add new behaviours to an object by wrapping it inside a special wrapper object that contains these behaviours.

We use Salesforce (SF) as our Customer Relationship Manager (CRM), this service has an API which we pull data out of, to show on the intranet. We needed a way to easily select data from SF and have it be in the same shape as our data transfer objects (DTOs).

An issue we've come across is allowing for naming differences between the SF and our DTO fields, this is mainly down to SF's naming conventions for custom fields, forcing them to have `'__c'` appended to them. This is in conflict with our convention of using Pascal Case. To initially resolve this problem, in the SOQL generator we used a dictionary to map DTO fields to SF fieldnames, here is an example:

```
public static readonly Dictionary<PropertyInfo, string> DTO_FIELD_TO_SALESFORCE_FIELD = new()
{
          { typeof(Lead).GetProperty(nameof(Lead.Company)), "Company__c" }
};
```

Then in the `GenerateSelectClauseForType` we'd use the member variable name by default, but if the DTO field had a mapping in the dictionary above, it would use the string instead.

```
public string GenerateSelectClauseForType(Type typeNode)
{
        // Start of GenerateSelectClauseForType...

        var childNodes = typeNode.GetProperties(BINDING_FLAGS);
        List<string> fields = new();

        foreach(var node in childNodes)
        {
                string field = "";

                bool nodeInMapping = DTO_FIELD_TO_SALESFORCE_FIELD.ContainsKey(node);
                if(nodeInMapping)
                {
                        field = DTO_FIELD_TO_SALESFORCE_FIELD[node];
                }
                else
                {
                        field = node.Name;
                }

                fields.Add(field);
        }
```

```
        // Rest of GenerateSelectClauseForType...
}
```

There are many issues with the mapping approach, mainly being that you end up with one very big dictionary as the codebase grows, which is unmaintainable. After doing some research, I decided to update this method to use the Decorator pattern.

In the decorator pattern, you have a base class or interface and additional functionality is added by wrapping the object in "decorators". In our case, we need a decorator to wrap the `Lead.Company` field directly, to be able to add metadata to it, such as the salesforce fieldname, without actually changing the DTO class or field.

With this I made the `SelectFromSalesforce` decorator class:

```
/// <summary>
/// The atttribute will allow salesforce queries to use this field as part of the query. If applied to a
property, then the property will be serialised. If it is set on a class, then all properties will be
serialised.
/// </summary>
[AttributeUsage(AttributeTargets.Property | AttributeTargets.Class, Inherited = true, AllowMultiple =
true)]
public class SelectFromSalesforceAttribute : Attribute
{

        public string FieldName { get; set; }

        public SelectFromSalesforceAttribute()
        {

        }

}
```

To use this decorator you simply have to add it as an attribute to a member variable like so:

```
[SelectFromSalesforce(FieldName = "Company__c")]
public string Company { get; set; }
```

At this point we can delete the mapping dictionary and update the method, to check if the DTO field has the extra properties, provided by the decorator:

```
public string GenerateSelectClauseForType(Type typeNode)
{
        // Start of GenerateSelectClauseForType...

        var childNodes = typeNode.GetProperties(BINDING_FLAGS);
        List<string> fields = new();

        foreach(var node in childNodes)
        {
                string field = "";

                var selectFromSalesforceAttribute =
node.GetCustomAttributes(typeof(SelectFromSalesforceAttribute));

                bool nodeHasDecorator = selectFromSalesforceAttribute is not null;
                if(nodeHasDecorator)
```

```
                {
                        field = selectFromSalesforceAttribute.FieldName;
                }
                else
                {
                        field = node.Name;
                }

                fields.Add(field);
        }

        // Rest of GenerateSelectClauseForType...
}
```