Nightmarish Report

Igor Nolasco

Nightmarish Index

[**Microsoft docs** 1](#_Toc98770131)

[**Code** 4](#_Toc98770132)

[**Readme Questions and answers** 5](#_Toc98770133)

# **Microsoft docs**

Enumeration

Is a way of associating a series of instances to an integer (by default). The associated integer depends on the position of the instance in the string, meaning you do not have to call all of them, just the first (if you want them ordered in this way). In the nightmarish example, for instance, when the CREATE = 1 is declared, it is assumed that the value associated with the next instances is, by default, 2,3,4,5,6, etc. When we create a “public enum Operation” we are in fact using the base class for enumerations in .NET.

Attributes

Attributes are assertive declarations that associates information with code. They are found inside a square bracket [] and placed directly above the element they apply for. Attributes can have two types of parameters: the positional ones correspond to the parameters of the attribute constructor and must be specified first; the named parameters are related to properties of the attributes.

In the Nightmarish project, one example of the Attribute usage is the AttributeUsage applied to the in the PathAttribute class.

[AttributeUsage(AttributeTargets.Class, AllowMultiple = false)]

AttributeUsage is one of the 3 predefined attributes of the .NET framework. He is used to declare how an attribute class can be used. In our case, it can only target classes and it does not allow multiple instances of it.

**Dictionary Class**

Dictionary in .NET is a way of storing values paired with a key value. It is displayed in this way:  [IDictionary<TKey, TValue>](https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.idictionary-2?view=netframework-4.8" \t "_blank) . The keys have to be unique, the values have not. To create a dictionary you have to define the type of key and type of values inside. In our nightmarish example, the dictionary uses integer keys and Icrud<Entity> values.

**Nullable Contexts**

This is a new feature designed to avoid generating Nullreference exception errors. It has to be declared and activated like this: #nullable enable. After activating it, we can declare reference types with “?” If they can be nullable. The “!” suppresses the possible warnings for assigning null. We don’t need the “!” operator in integers because, by default, they should not be null, they are “0” instead. On strings, we do need them if we want to make it possible to assign null values.

This is the case on the 2nd version of the nightmarish project, on the employees model (for instance):

public string LastName { get; set; } = null!; the = null! Is used on this string and not on this integer public int EmployeeId { get; set; }.

**Asynchronous programming**

Asynchronous programming is a way of performing certain actions in a separate thread and then informing the main thread about the status of this action (if it was successful or not). It is very useful, as stated in the Microsoft docs, in games or whenever we need to multitask without overcharging the main thread.

**Reflection**

Reflection, as I understood, can be used to create the dynamic instantiation of a type or to get the type of an object and invoke its methods and access his properties. In the program operations section we have several examples of reflection :

Entity o = selected.CreateNewOfT();  
PropertyInfo[] propertyInfos = o.GetType().GetProperties();

We are first creating a new instance of the entity and then creating the array “propertyInfos” and assigning it to all the types of properties of the selected entity.

This was what I understood, at least.

**Delegates**

A delegate can be seen as a reference to a set of methods, all sharing the same return type(ex. Void) and parameters (ex. No parameters). This allows us to safeguard and make sure the methods used have a specific signature.

In our project, the delegate function is called on the OperationToExecute() method,and then, through it, we can call all other methods that return void and have no parameters, for instance:

OperationsDict.Add(1, new OperationToExecute(Operations.Create));

The delegated method calls on the create method inside this dictionary.

**Generic classes and methods**

Generic classes and methods allow us to save time, disk space and computer power, it is a new level of abstraction that allows us, through the generic type parameter <T>, to apply specific methods to many different classes. It is important to remember that the T parameter is used where a concrete one would be in the method. The method can later be invoked, and the T replaced by the type desired.

**IEnumerable**

The IEnumerable interface is used to allow the foreach to properly browse through the collection – it, as the name says – ENUMERATES the collection, allowing it to be consulted.

**?? and ??= operators (C# reference)**

On C# 8 or more, the ??= operator is used to make a variable (ONLY IF IT IS NULL) equal to the assigned expression. Variable ??= expression.

# **Code**

-The first step to run the program was to install the .NET 3.1 version. After that, the program ran.

On my second glance, I noted two mistakes: not all entities inside the program are available for consultation (only two of them) and the “read” function does not work

-I replaced the Read() method for the ToString() one and the READ problem persisted, the problem is, apparently, in the overrite of the ToString() method.

-It was all about the path directory afterall!! After Paulo showed us, I corrected the db/files.

-added the Employee model and the Manager<Employee> entry option on the RoadToDBConsole page.

-The employees can’t be read, working on it.

-after a lot of attempts, correcting json, jsonignores, etc ,etc it turned out the problem was………ON FUCKING COPY ALWAYS!! Hahahhaha the debugging tool showed that the contents list on the employee context was empty so that was a big clue.

-added all the corresponding classes that were missing. -Also added the new options for them in the roadToDBConsole. I tested them out, they all READ what I want them to read.

-Find method does not work, I want it to show the object it grabs, testing it now. Debugging to see if the contents list is empty or not when it tries to find..

-resolved the find method.

-went back to the json files, created an “address” class and added an “if” to cover the addition of objects Adress

-did the same with the shipAdress

-it is creating and deleting as it should.

-some errors regarding specificities of the Json file.

-the starting order does not make sense: one should not have to open a category to close everything, the same with saveall

-well everything kinda works, there is lots more to change and that I was planning to change but I think its better to study the newer version…

Version 2.0

-It does not read – it is not loading the json files. I changed the path in appsettings.json and it works just fine.

-there are some bugs in the creation and in the find method. Resolved, it was just a path thing, changing from my PC to my notebook lol

-Nullables problem: finding testing solutions

-Paulo showed today his method, I will try to implement and understand it

# **Readme Questions and answers**

1. Does it work? What does it do?

R: “Work” is subjective, At first, the first version did not work at all. First we had to install the .NET 3.1 framework. And then, it kind of worked. After that, there was a problem with loading the json files, the path directory was wrong. After changing that, it started to work a little bit more. There were also missing classes, json properties issues, inconsistencies and redundancies, that were greatly solved by the 2.0 version. In that version, we also had to change the appsettingjson directory.

1. Study the solution and projects. Identify what does what! Comment the code at your will.

R: Both the 1.0 and the 2.0 versions are commented!

1. Consider the other six db models. What do you need to integrate them in the solution? Check if you can get it done.

R: Well, all I needed was to create the corresponding classes, also derivates from the Entity, with the properties used in the json file.

1. Add a Find option in the menu that receives an ID and shows the selected entity only.

I have added the find option on the 1.0 version and it works very well. I will add it to the 2.0 version now.

1. Find any bugs and document them. If you can, correct them and document the solution.

R: I believe that I commented them above, on the “CODE” section.

1. Consider some topics highlighted in the comments, study and summarize them (hint: Reflection, Enumeration).

R: I studied and wrote about them all with my own words and I tried to bring up the context that we used them also.

1. If you needed to change the JSON files location, how many times would you modify the path? Suggest a correction for this issue.

In the 2.0 version we only need to change it once! This is done through the combination of the appsettings.json(with the path location) and the Iconfigurationroot interface.

7. Questions:

a) Console: How do we write in color?

A good way to do so is just as we use in the project: Console.ForegroundColor = ConsoleColor.NameOfColor;

b) How to map db models to json files? Explain how it is done.

This process is achieved through serialization, converting objects into strings. Deserialization does the exact opposite.

c) To integrate other db models, do you need to modify the manager?

I am already in the 2.0 version as I am answering this, but even in the first version, we don’t need to modify the manager to add more models, that is the purpose of generics, this abstraction allow us to grow the project easly.

d) What is the purpose of the ICrud interface and IHasPrimaryKey interface? How are the used?

Icrud interface sets the main methods that are applied to the models. We can also see that can only be applied to Ts that are entities and have Ihasprimarykeys applied. IhasPrimaryKey interface is an abstract way of each model applying its own primary key, since they all must have it, it makes sense to make it into an interface.

e) Is there a difference between "Operation" and "Entity"? What do those concepts mean?

Ofc there is! Operation are the possible actions that can be performed towards the entities. Entity is an abstract class that engulfs all the models, restricting them and obliging them to have certain things like the getprimarykey() method and the header. (also ihasprimarykey interface).

f) Why do we need "newOfT" in the Manager?

Even though its generic it does not mean It doesnt have to be instantiated! The constructor doesn’t have but the list has!

g) What is the purpose of the "new()" restriction on the top?

The new() restriction forces the generic class to have a public constructor without parameters – also the type in question cant be abstract.

h) Why do we need IEnumerable<T> implementation in the Manager?

The IEnumerable allows us to use the foreach to properly browse through the collection – it, as the name says – ENUMERATES the collection, allowing it to be consulted.

1. Is "Entity" really needed? What was its purpose?

Entity is an abstract class that engulfs all the models, restricting them and obliging them to have certain things like the getprimarykey() method and the header. (also ihasprimarykey interface). It is needed to restrict and safeguard our objects.

j) What is the usefulness of singleton design pattern in the Manager?

The singleton design pattern is efficient because it only creates one instance, we only have to deal with a single point of entry to the system, per say.

k) Is the path attribute the only way of solving the path issue? Suggest an alternative.

We do not need to define an attribute to show the file path, A static method that returns the file location would be enough – adding an abstract method on the father (entity) and then each son can define its own path overriding it.