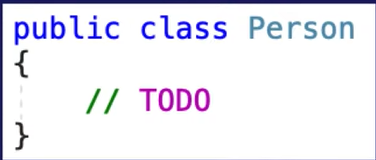
Person Class:

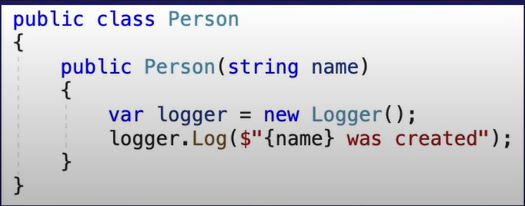


Logger Class:

Graphical user interface, text, application

Description automatically generated

Without Dependency Injection



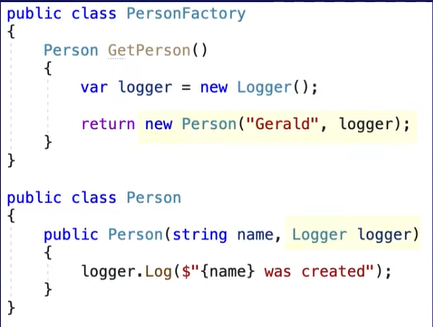
With Dependency Injection

Graphical user interface, text

Description automatically generated

Ways of doing dependency injections:

Constructor Injection:



Method Injection:

Graphical user interface, text, application, chat or text message

Description automatically generated

Property:

Graphical user interface, text, application, email

Description automatically generated

**Example 2:**

**TextLogger**:

Text

Description automatically generated

**DatabaseLogger**:

Text

Description automatically generated with medium confidence

**Program for TextLogger Use:**

Text

Description automatically generated

**Program for DatabaseLogger Use:**

Text

Description automatically generated

**Now With DI:**

Text

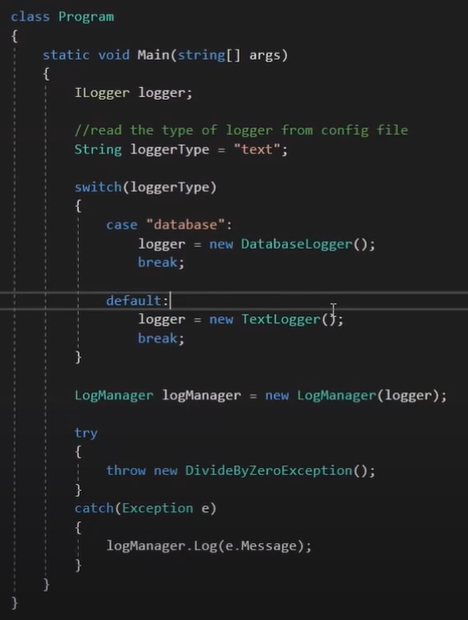
Description automatically generated

Text

Description automatically generated

Text

Description automatically generated



**Example 3:**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

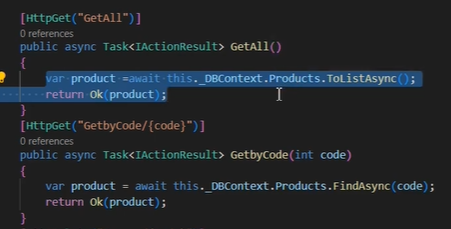
Graphical user interface, text, application, email

Description automatically generated

**Example 4:**

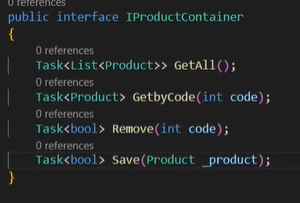
Let say we have many controllers which are using the same code, in case if the Model Class/Table name “Products” is changed then we have to change that in all of the controllers.

\_DBContext.Products.ToListAsync()

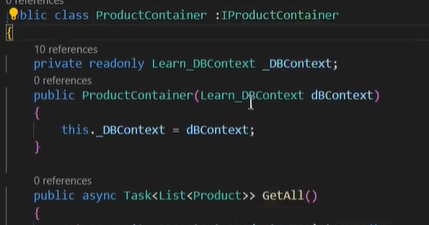


**Using DI:**

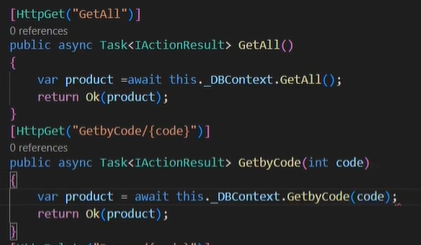
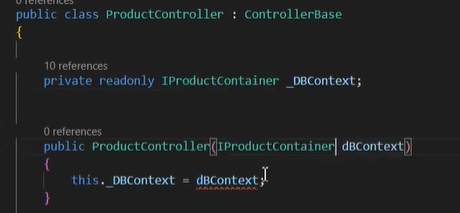
**IProductContainer Interface:**



**Interface Implementation:**



**Using It In Controller:**



\_DBContext.GetAll()

***Remember to add the service inside Program.cs File***:

builder.Services.AddScoped<IProductContainer, ProductContainer>();

**Service Scopes:**

AddSingleton / AddScoped / AddTransient <IProductContainer, ProductContainer>();

**Singleton** which creates a single instance throughout the application. It creates the instance for the first time and reuses the same object in the all calls.

**Scoped** lifetime services are created once per request within the scope. It is equivalent to a singleton in the current scope. For example, in MVC it creates one instance for each HTTP request, but it uses the same instance in the other calls within the same web request.

**Transient** lifetime services are created each time they are requested. This lifetime works best for lightweight, stateless servic