Scenarios Where EF Core Considers a Class as an Entity (Including Entities)

1. Defined as a DbSet in DbContext

When a class is defined as a DbSet<TEntity> in the DbContext, EF Core automatically treats it as an entity and includes it in migrations and database operations.

```
public class ApplicationDbContext : DbContext
{
  public DbSet<Customer> Customers { get; set; }
  public DbSet<Order> Orders { get; set; }
}
```

In this example, Customer and Order are entities because they are defined as DbSet properties in the DbContext.

2.Use Fluent API to configure a model

Even if a class is not defined as a DbSet, you can explicitly include it as an entity using the Fluent API in the OnModelCreating method.

You can override the OnModelCreating method in your derived context and use the fluent API to configure your model. This is the most powerful method of configuration and allows configuration to be specified without modifying your entity classes. Fluent API configuration has the highest precedence and will override conventions and data annotations. The configuration is applied in the order the methods are called and if there are any conflicts the latest call will override previously specified configuration.

```
using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.FluentAPI.Required;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    #region Required
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Blog>()
            .Property(b => b.Url)
            .IsRequired();
    }
    #endregion
}

public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }
}
```

Grouping configuration

To reduce the size of the `OnModelCreating` method all configuration for an entity type can be extracted to a separate class implementing IEntityTypeConfiguration<TEntity>.

```
public class BlogEntityTypeConfiguration : IEntityTypeConfiguration<Blog>
{
    public void Configure(EntityTypeBuilder<Blog> builder)
    {
        builder
            .Property(b => b.Url)
            .IsRequired();
    }
}
```

Then apply the configurations on `OnModelCreating`

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
   base.OnModelCreating(modelBuilder);
   modelBuilder.ApplyConfiguration(new BlogEntityTypeConfiguration();
}
```

3. Defined in a Relationship:

DbSet /navigation properties / modelBuilder.Entity<>

If a class is referenced in a relationship with another entity (e.g., through navigation properties), EF Core may automatically consider it an entity, even if it's not explicitly defined as a DbSet.

By convention, types that are exposed in DbSet properties on your context are included in the model as entities. Entity types that are specified in the OnModelCreating method are also included, as are any types that are found by recursively exploring the navigation properties of other discovered entity types.

In the code sample below, all types are included:

- Blog is included because it's exposed in a DbSet property on the context.
- Post is included because it's discovered via the Blog.Posts navigation property.
- AuditEntry because it is specified in OnModelCreating.

```
public class Blog
{
    public int BlogId { get; set; }
    public string Url { get; set; }

    public List<Post> Posts { get; set; }
}

public class Post
{
    public int PostId { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Blog Blog { get; set; }
}

public class AuditEntry
{
    public int AuditEntryId { get; set; }
    public string Username { get; set; }
    public string Action { get; set; }
}

using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<AuditEntry>();
    }
}
```

You can also apply certain attributes (known as *Data Annotations*) to your classes and properties. Data annotations will override conventions, but will be overridden by Fluent API configuration.

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using Microsoft.EntityFrameworkCore;

namespace EFModeling.EntityProperties.DataAnnotations.Annotations;

internal class MyContext : DbContext
{
    public DbSet<Blog> Blogs { get; set; }
}

[Table("Blogs")]
public class Blog
{
    public int BlogId { get; set; }

    [Required]
    public string Url { get; set; }
}
```

Lack of Direct Access: Without a DbSet property, you cannot directly perform LINQ queries or CRUD operations on the entity using the DbContext instance. This means you can't directly use methods like context.Entities.Add() or context.Entities.Where().

Accessing Entities Without DbSet:

You can use the **Set<TEntity>()** method of DbContext to access entities not defined as DbSet. This method provides a generic DbSet that can be used for querying and updates.

```
using (var context = new ApplicationDbContext())
{
  //Access Product entity using Set<Product>()
  var products = context.Set<Product>().ToList();

  //Perform CRUD operations
  context.Set<Product>().Add(new Product { Name = "New Product", Price = 10.99m });
  context.SaveChanges();
}
```