

Entity Relations

Customizing Entity Models



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://about.softuni.bg/>

- Object Composition
- Fluent API
- Attributes
- Table Relationships
 - One-to-One
 - One-to-Many
 - Many-to-Many



sli.do

#csharp-db

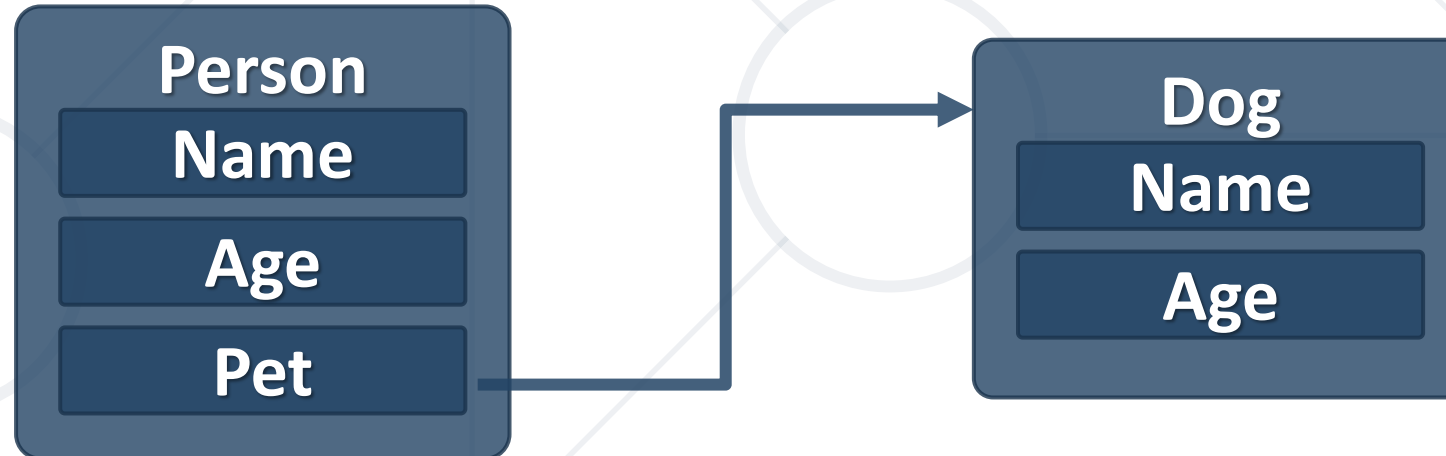


Object Composition

Describing Database Relationships

Object Composition

- Object composition denotes a "**has-a**" relationship
 - e.g., the **car** has an **engine**
- Defined in C# by one object having a property that is a reference to another



- Navigation properties create a **relationship** between entities
 - Either an **Entity Reference** (one-to-one-or-zero) or an **ICollection** (one-to-many or many-to-many)
- They provide **fast querying** of related records
- Can be **modified** by **directly** setting the reference



$f()$ \Rightarrow API

Fluent API

Working with Model Builder

- **Code First** maps your POCO (Plain Old CLR Objects) classes to tables using a **set of conventions**
 - e.g., property named "**Id**" maps to the **Primary Key**
- Can be customized using **annotations** and the **Fluent API**
- Fluent API (Model Builder) allows **full control** over DB mappings
 - Custom names of objects (columns, tables, etc.) in the DB
 - Validation and data types
 - Define complicated entity relationships

- Custom mappings are placed inside the **OnModelCreating** method of the DB context class

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Student>()
        .HasKey(s => s.StudentKey);
}
```

- Specifying Custom Table name

```
modelBuilder.Entity<Order>()  
    .ToTable("OrderRef", "Admin");
```

Optional schema
name

- Custom Column name/DB Type

```
modelBuilder.Entity<Student>()  
    .Property(s => s.Name)  
    .HasColumnName("StudentName")  
    .HasColumnType("varchar");
```

- Explicitly set Primary Key

```
modelBuilder  
    .Entity<Student>()  
    .HasKey("StudentKey");
```

- Other column attributes

```
modelBuilder.Entity<Person>()  
    .Property(p => p.FirstName)  
    .IsRequired()  
    .HasMaxLength(50)
```

```
modelBuilder.Entity<Post>()  
    .Property(p => p.LastUpdated)  
    .ValueGeneratedOnAddOrUpdate()
```

- Do not include property in DB (e.g., business logic properties)

```
modelBuilder  
    .Entity<Department>()  
    .Ignore(d => d.Budget);
```

- Disabling cascade delete

- If a FK property is non-nullable, cascade delete is **on by default**

```
modelBuilder.Entity<Course>()  
    .HasRequired(t => t.Department)  
    .WithMany(t => t.Courses)  
    .HasForeignKey(d => d.DepartmentID)  
    .OnDelete(DeleteBehavior.Restrict);
```

Throws exception on delete

- Mappings can be placed in entity-specific classes

```
public class StudentConfiguration
    : IEntityTypeConfiguration<Student>
{
    public void Configure(EntityTypeBuilder<Student> builder)
    {
        builder.HasKey(c => c.StudentKey);
    }
}
```

Specify target model

- Include in **OnModelCreating**

```
builder.ApplyConfiguration(new StudentConfiguration());
```



Attributes

Custom Entity Framework Behavior

- EF Code First provides a set of **DataAnnotation attributes**

- You can override default Entity Framework behavior

- Access nullability and size of fields

```
using System.ComponentModel.DataAnnotations;
```

- Access schema customizations

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

- For a full set of configuration options you need the **Fluent API**

- **[Key]** – explicitly specify **primary key**
 - When your PK column doesn't have an "**Id**" or "**<TypeName>Id**" suffix

```
[Key]  
public int StudentKey { get; set; }
```

- **Composite key** is defined using **Fluent API**

```
builder.Entity<Car>()  
    .HasKey(c => new { c.State, c.LicensePlate });
```

- **[PrimaryKey]** available **only** in EF7

- **[ForeignKey]** – explicitly **link** navigation property and foreign key property within the same class
- Works in **either direction** (FK to navigation property or navigation property to FK)
 - **[ForeignKey(NavigationPropertyName)]** – on the foreign key scalar property in the dependent entity
 - **[ForeignKey(ForeignKeyPropertyName)]** – on the related reference navigation property in the dependent entity
 - **[ForeignKey(ForeignKeyPropertyName)]** – on the navigation property in the principal entity

ForeignKey Attribute – Example

- `[ForeignKey(NavigationPropertyName)]`
- `[ForeignKey(ForeignKeyPropertyName)]`

```
public class Order
{
    ...
    [ForeignKey(nameof(Client))]
    public int ClientId { get; set; }
    public Client Client { get; set; }
}
```

```
public class Order
{
    ...
    public int ClientId { get; set; }
    [ForeignKey(nameof(ClientId))]
    public Client Client { get; set; }
}
```



```
public class Client
{
    ...
    public ICollection<Order> Orders { get; set; }
}
```



- **Table** – manually specify the name of the table in the DB

```
[Table("StudentMaster")]  
public class Student  
{  
    ...  
}
```

```
[Table("StudentMaster", Schema = "Admin")]  
public class Student  
{  
    ...  
}
```

- **Column** – manually specify the name of the column in the DB
 - You can also specify order and explicit data type

```
public class Student  
{
```

```
    ...  
    [Column("StudentName", Order = 2, TypeName="varchar(50)")]  
    public string Name { get; set; }  
}
```

Optional parameters

- **Required** – mark a nullable property as **NOT NULL** in the DB
 - Will throw an exception if not set to a value
 - Non-nullable types (e.g., **int**) will **not throw** an exception (will be set to language-specific default value)
- **MinLength** – specifies min length of a string (client validation)
- **MaxLength / StringLength** – specifies max length of a string (both client and DB validation)
- **Range** – set lower and/or upper limits of numeric property (client validation)

- **Index** – create index for column(s)
 - Primary key will always have an index

```
[Index(nameof(Url))]  
public class Student {  
    public string Url { get; set; }  
}
```

- **NotMapped** – property will not be mapped to a column
 - For business logic properties

```
[NotMapped]  
public string FullName => this.FirstName + this.LastName;
```



Table Relationships

Expressed As Properties and Attributes

- Expressed in SQL Server as a shared primary key
- Relationship direction must be explicitly specified with a **ForeignKey** attribute
- **ForeignKey** is placed above the key property and contains the **name** of the navigation property and vice versa



One-to-Zero-or-One: Implementation (1)

- Using the **ForeignKey** Attribute

```
public class Student
{
    [Key]
    public int Id { get; set; }
    public string Name { get; set; }
    [ForeignKey("Address")]
    public int AddressId { get; set; }
    public Address Address { get; set; }
}
```

Attributes

One-to-Zero-or-One: Implementation (2)

- Using the **ForeignKey** Attribute

```
public class Address
{
    public int Id { get; set; }
    public string Text { get; set; }
    [ForeignKey(nameof(Student))]
    public int StudentId { get; set; }
    public Student Student { get; set; }
}
```

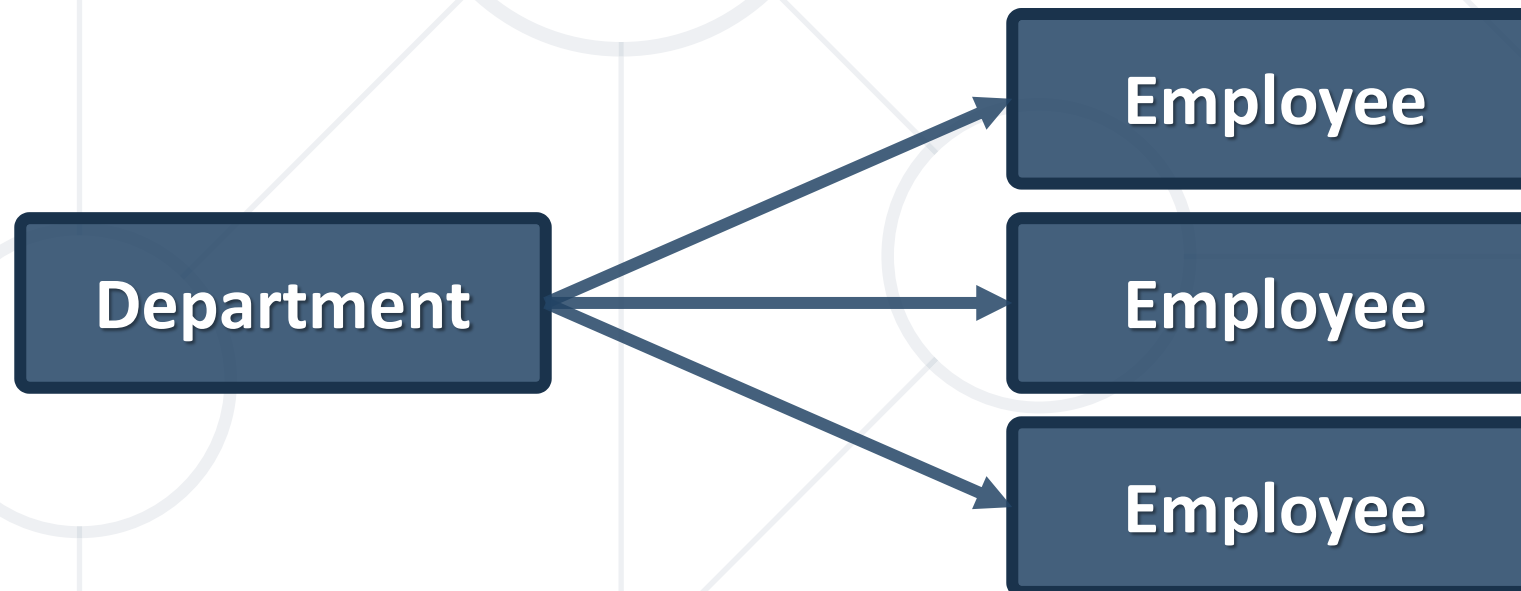
- **HasOne** → **WithOne**

```
modelBuilder.Entity<Address>()  
    .HasOne(a => a.Student)  
    .WithOne(s => s.Address)  
    .HasForeignKey<Student>(a => a.StudentId);
```

Address contains FK
to **Student**

- If **StudentId** property is **nullable** (**int?**), relation becomes **One-To-Zero-Or-One**

- Most common type of relationship
- Implemented with a **collection** inside the **parent entity**
 - The collection should be **initialized** in the **constructor**!



One-to-Many: Implementation (1)

- **One department** has **many employees**

```
public class Department
{
    public int Id { get; set; }
    public string Name { get; set; }

    public ICollection<Employee> Employees { get; set; }
}
```

One-to-Many: Implementation (2)

- Each employee has one department

```
public class Employee
{
    public int Id { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

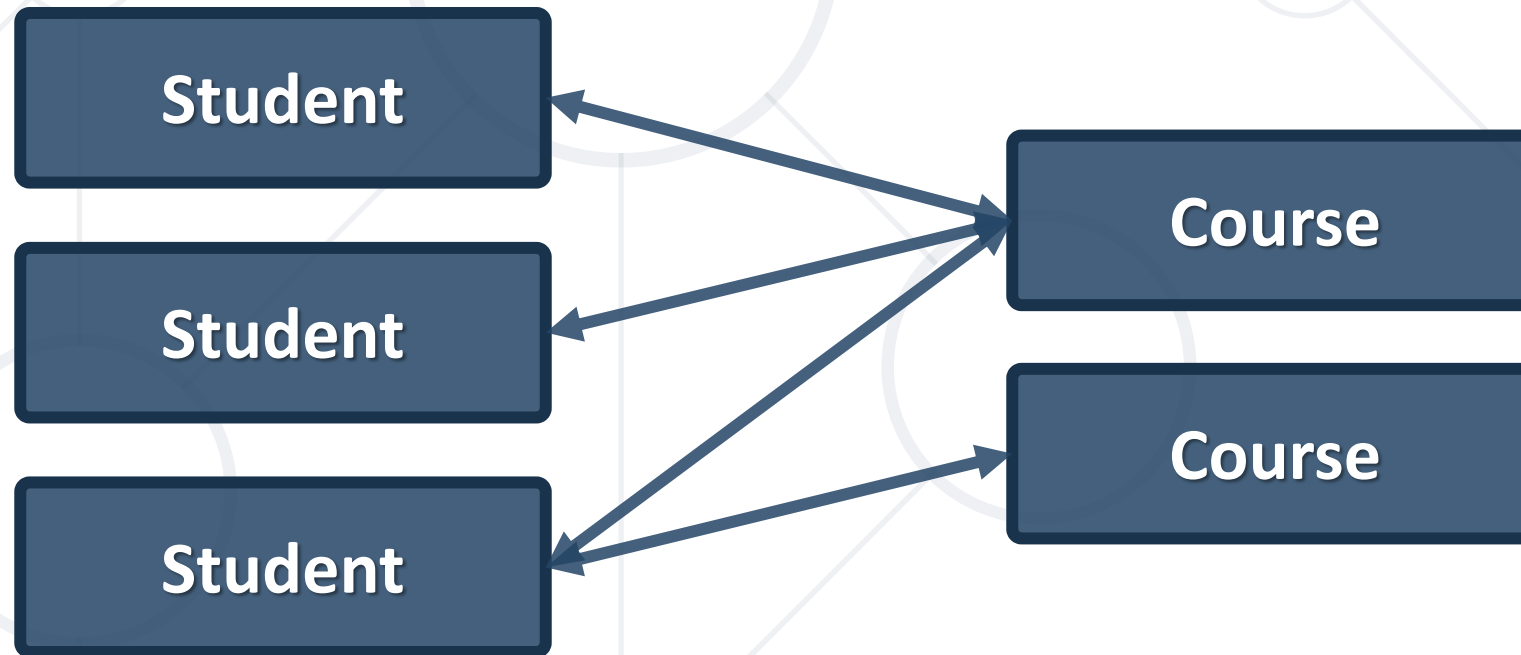
    public int DepartmentId { get; set; }
    public Department Department { get; set; }
}
```

- **HasMany** → **WithOne** / **HasOne** → **WithMany**

```
modelBuilder.Entity<Department>()  
    .HasMany(d => d.Employees)  
    .WithOne(e => e.Department)  
    .HasForeignKey(e => e.DepartmentId);
```

```
modelBuilder.Entity<Employee>()  
    .HasOne(e => e.Department)  
    .WithMany(d => d.Employees)  
    .HasForeignKey(e => e.DepartmentId);
```

- Requires a **collection navigation property on both sides**
 - Implemented with collections in each entity, referring the other



Many-to-Many Implementation (1)

```
public class Course
{
    public string Name { get; set; }
    public ICollection<Student> Students { get; set; }
}
```

```
public class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public ICollection<Course> Courses { get; set; }
}
```

- Mapping **both sides** of relationship

```
builder.Entity<Student>()
    .HasMany(s => s.Courses)
    .WithMany(s => s.Students)
    .UsingEntity<Dictionary<string, object>>(
        "StudentCourse",
        r => r
            .HasOne<Course>()
            .WithMany()
            .HasForeignKey("CourseId")
            .HasConstraintName("FK_StudentCourse_Courses_CourseId")
            .OnDelete(DeleteBehavior.Cascade),
        r => r
            .HasOne<Student>()
            .WithMany()
            .HasForeignKey("StudentId")
            .HasConstraintName("FK_StudentCourse_Students_StudentId")
            .OnDelete(DeleteBehaviour.Cascade),
        j =>
        {
            j.HasKey("StudentId", "CourseId");
            j.ToTable("StudentsCourses");
            j.IndexerProperty<int>("StudentId").HasColumnName("StudentId");
            j.IndexerProperty<int>("CourseId").HasColumnName("CourseId");
        }
    );
```

Composite Primary Key

Many-to-Many Implementation (2)


- You can optionally create a **join entity type**

```
public class Course
{
    public string Name { get; set; }
    public string Teacher { get; set; }

    public List<StudentCourse> StudentsCourses { get; set; }
}
```


```
public class Student
{
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public List<StudentCourse> StudentsCourses { get; set; }
}
```



```
public class StudentCourse
{
    public int CourseId { get; set; }
    public Course Course { get; set; }

    public int StudentId { get; set; }
    public Student Student { get; set; }
}
```



- **Indirect** many-to-many relationships

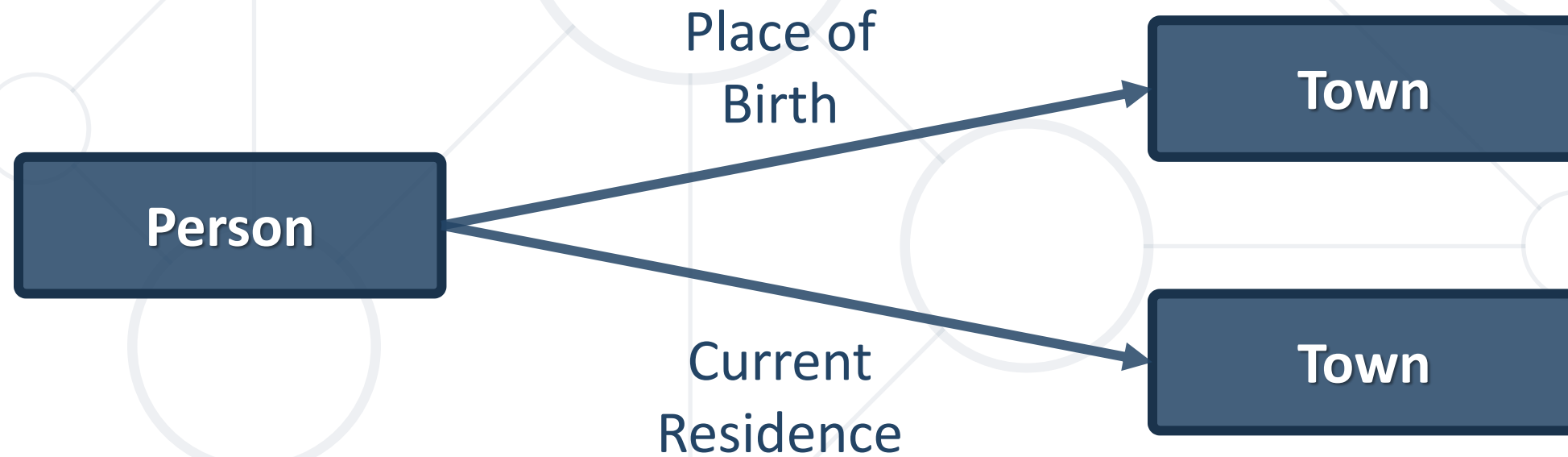
```
modelBuilder.Entity<StudentCourse>()  
    .HasKey(sc => new { sc.StudentId, sc.CourseId });
```

```
builder.Entity<StudentCourse>()  
    .HasOne(sc => sc.Student)  
    .WithMany(s => s.StudentCourses)  
    .HasForeignKey(sc => sc.StudentId);
```

```
builder.Entity<StudentCourse>()  
    .HasOne(sc => sc.Course)  
    .WithMany(s => s.StudentCourses)  
    .HasForeignKey(sc => sc.CourseId);
```

Composite
Primary Key

- When two entities are related by more than one key
- Entity Framework needs help from **Inverse Properties**



Multiple Relations Implementation (1)

- **Person** Domain Model – defined as usual

```
public class Person
{
    public int Id { get; set; }
    public string Name { get; set; }

    public Town PlaceOfBirth { get; set; }
    public Town CurrentResidence { get; set; }
}
```

Multiple Relations Implementation (2)

■ Town Domain Model

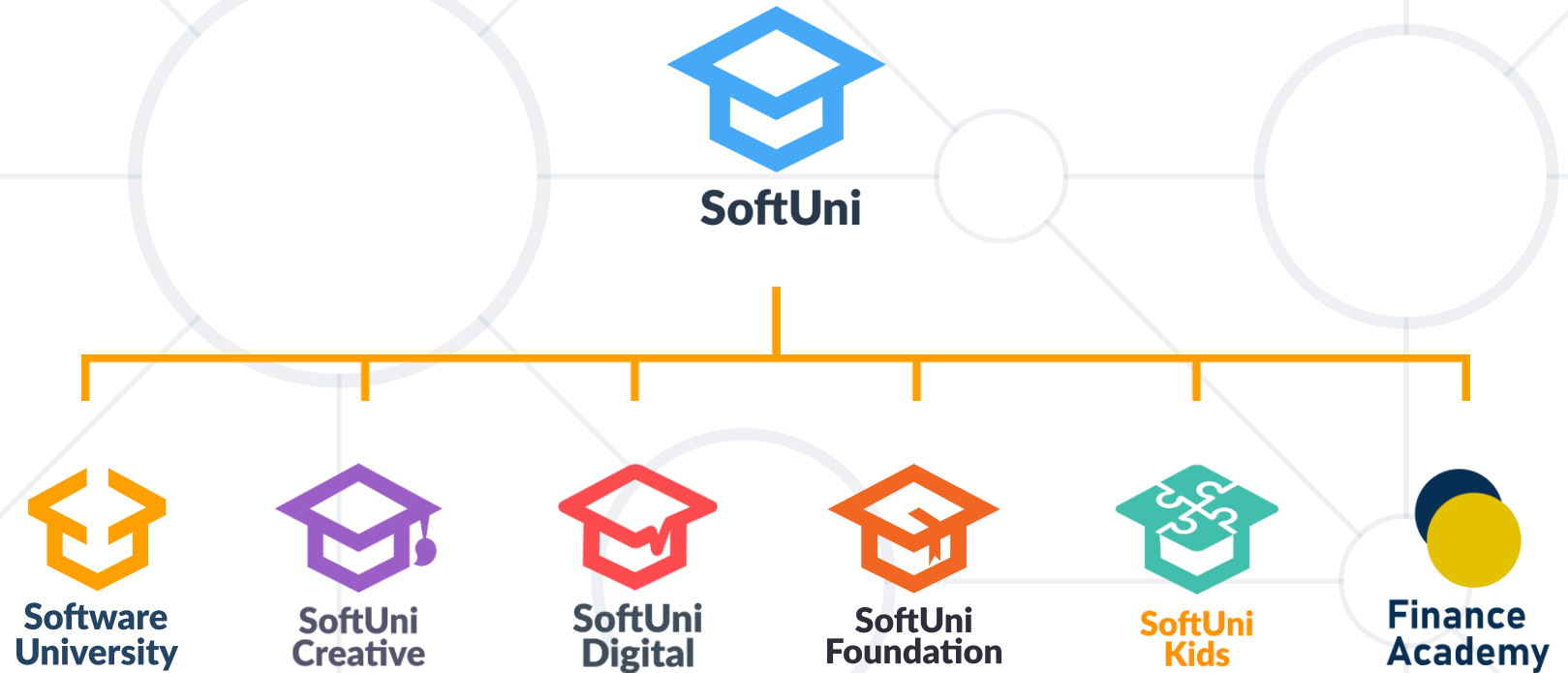
```
public class Town
{
    public int Id { get; set; }
    public string Name { get; set; }
    [InverseProperty("PlaceOfBirth")]
    public ICollection<Person> Natives { get; set; }
    [InverseProperty("CurrentResidence")]
    public ICollection<Person> Residents { get; set; }
}
```

Point towards
related property

- The **Fluent API** gives us full control over Entity Framework object mappings
- **Attributes** can be used to express special table relationships and to customize entity behaviour
- Objects can be composed from other objects to represent complex **relationships**



Questions?



SoftUni Diamond Partners

**SUPER
HOSTING
.BG**



**Coca-Cola HBC
Bulgaria**

 **Flutter**TM
International

INDEAVR
Serving the high achievers



AMBITIONED

 **DRAFT
KINGS**



BOSCH

 **Postbank**
Решения за твоето утре

 **PHAR
VISION**



SmartIT

DXC
TECHNOLOGY

createX

- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity

- Software University Forums

- forum.softuni.bg



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg>
- © Software University – <https://softuni.bg>

