

O que aprendemos?

Transcrição

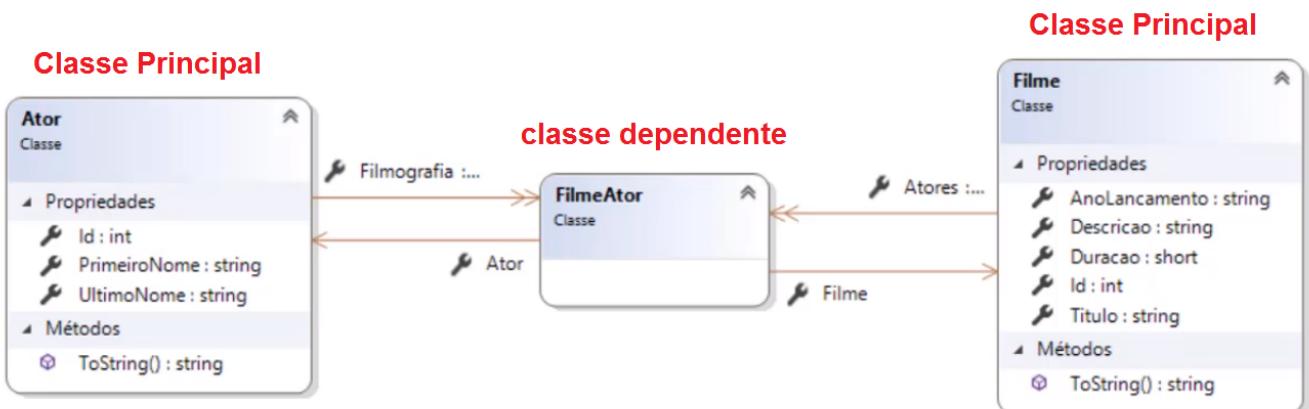
Fizemos o relacionamento de filmes com atores, ou seja, um relacionamento de cardinalidade **N:N** (muitos para muitos). Um filme possui um elenco de vários atores e um ator pode atuar em vários filmes. Contudo, o Entity Framework Core não suporta esse tipo de cardinalidade e o desenvolvedor deve criar uma classe que será responsável pelo relacionamento entre outras duas classes.

Devido a isso, acabamos por ter dois relacionamentos **1:N**: um para `FilmeAtor` e `Autor`, e outro entre `FilmeAtor` e `Filme`.

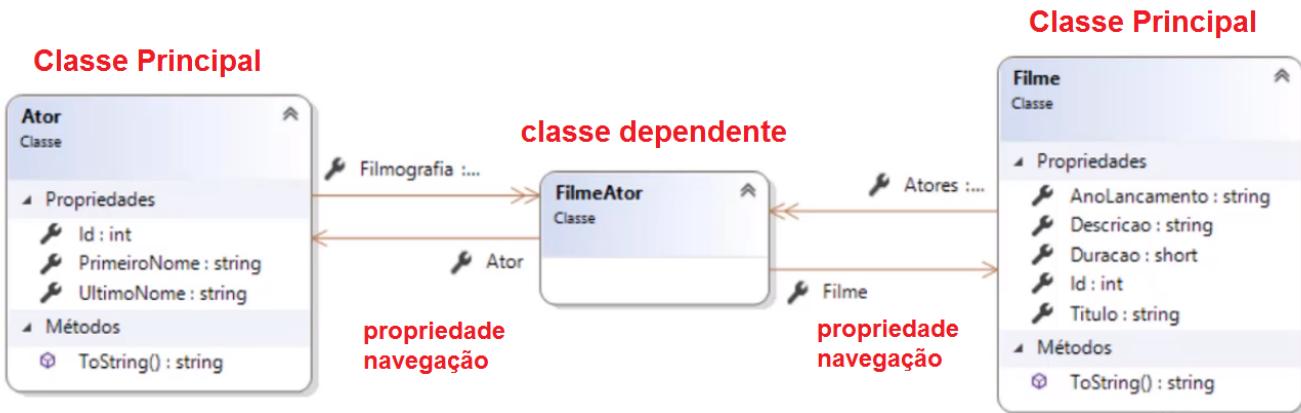
Existem alguns termos que são introduzidos pela tecnologia Entity que iremos citar nesta aula. Esses termos estão vinculados a cada entidade de relacionamento.

Classe Principal: considerada a classe independente da relação.

Classe Dependente: classe dependente da relação, ou "classe filha". É nela que reside a chave estrangeira.



Propriedades de Navegação: propriedades que referenciam uma ponta da relação. O Entity só irá descobrir a existência de um relacionamento via uma propriedade de navegação. Existem dois tipos de propriedades de navegação: por referência (aponta para apenas uma instância) ou por coleção (aponta para várias instâncias).



Chaves Estrangeiras: propriedades existentes na classe dependente que guardam o valor da classe principal, podem ser shadow properties.

No nosso exemplo não declaramos uma propriedade explicitamente no código da classe `FilmeAktor`, o que fizemos foi criar uma shadow property para a propriedade `actor_id`. Feito isso, configuramos o relacionamento utilizando a chave estrangeira `actor_id`. Fizemos essa configuração utilizando os métodos `HasOne()`, `WithMany()` e `HasForeignKey()`.

```

classDiagram
    class Ator {
        Id: int
        PrimeiroNome: string
        UltimoNome: string
        ToString(): string
    }
    class FilmeAktor {
        Actor
        Filme
    }
    class Filme {
        AnoLancamento: string
        Descricao: string
        Duracao: short
        Id: int
        Titulo: string
        ToString(): string
    }
    Ator "1" -- "*" FilmeAktor : Filmografia
    FilmeAktor "*" -- "1" Ator : Actor
    FilmeAktor -- "*" Filme : Filme
  
```

```

builder.ToTable("film_actor");

builder.Property<int>("film_id").IsRequired();
builder.Property<int>("actor_id").IsRequired();
builder.Property<DateTime>("last_update")
    .IsRequired()
    .HasColumnType("datetime")
    .HasDefaultValueSql("getdate()");

builder.HasKey("film_id", "actor_id");

builder
    .HasOne(fa => fa.Filme)
    .WithMany(f => f.Atores)
    .HasForeignKey("film_id");

builder
    .HasOne(fa => fa.Ator)
    .WithMany(a => a.Filmografia)
    .HasForeignKey("actor_id");
  
```

Aprendemos a convenção do Entity para **chave primária**. A chave primária em uma classe entidade é descoberta detecta a existência de uma propriedade chamada `Id` ou `<nome do tipo>Id`. Para configurarmos uma chave primária composta, podemos utilizar o método `HasKey()`.