07

Melhores práticas em configuração

Transcrição

Antes de começarmos com os requisitos de negócio da nossa aplicação, iremos discutir uma melhoria técnica. Reparem no código do nosso método `OnModelCrating()` :

```
namespace Alura.Filmes.App.Dados
{
    public class AluraFilmesContexto : DbContext
    {
        public DbSet<Ator> Atores { get; set; }
        public DbSet<Filme> Filmes { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=(localdb)mssqllocaldb;Database=AluraFilmes;Trusted_(
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.Entity<Ator>()
                .ToTable("actor");

            modelBuilder.Entity<Ator>()
                .Property(a => a.Id)
                .HasColumnName("actor_id");

            modelBuilder.Entity<Ator>()
                .Property(a => a.PrimieroNome)
                .HasCoumnName("first_name")
                .HasColumnType("varchar(45)")
                .IsRequired();

            modelBuilder.Entity<Ator>()
                .Property(a => a.UltimoNome)
                .HasColumnType("varchar(45)")
                .IsRequired();

            modelBuilder.Entity<Ator>()
                .Property<DateTime>("last_update");
                .HasColumnType ("datetime")
                .HasDefaultValueSql("getdate()")
                .IsRequired();

            modelBuilder.Entity<Filme>()
                .ToTable("film")

            modelBuilder.Entity<Filme>()
                .Property(f => f.id)

                .HasColumnName("film_id");

            modelBuilder.Entity<Filme>()
```

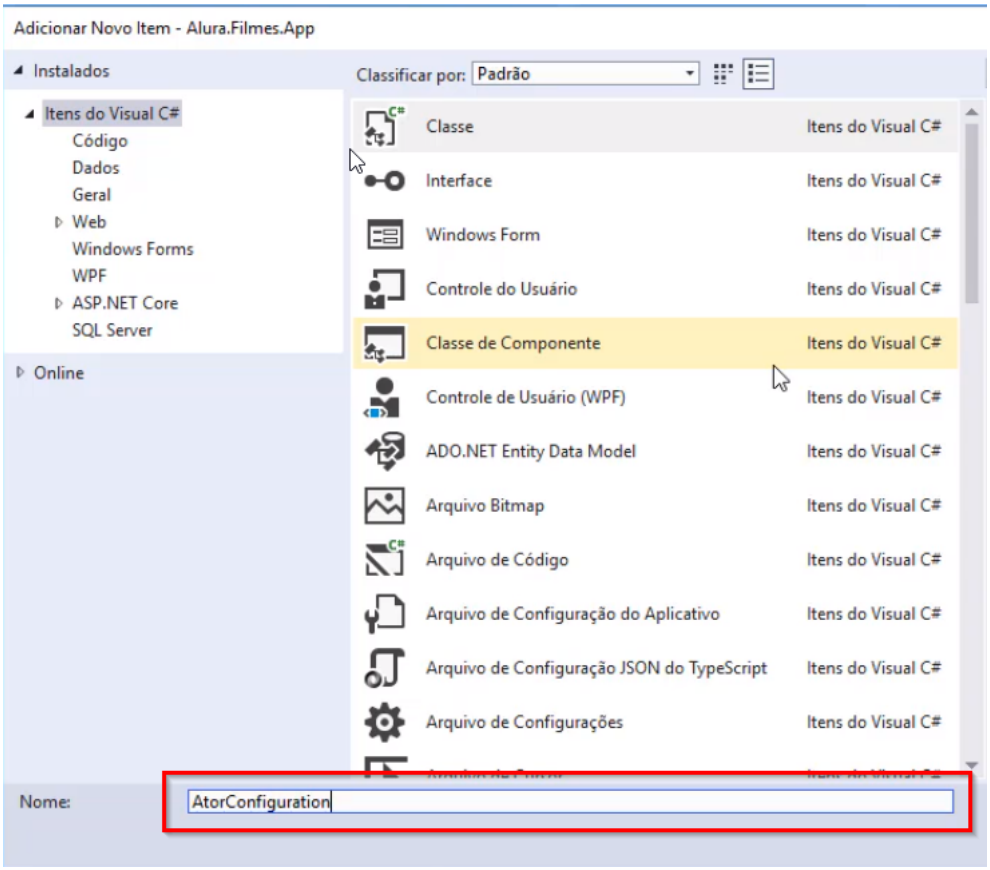
```
            .Property(f => f.Titulo)
            .HasColumnName("title")
            .HasColumnType("varchar(255)")
            .IsRequired();

        modelBuilder.Entity<Filme>()
            .Property(f => f.Descricao)
            .HasColumnName("description")
            .HasColumnType("text");

        modelBuilder.Entity<Filme>()
            .Property(f => f.AnoLancamento)
            .HasColoumnName("release_year")
            .HasColumnTypoe("varchar(4)");

        modelBuilder.Entity<Filme>()
            .Property(f => f.Duracao)
            .HasColumnName("length")

        modelBuilder.Entity<Filme>()
            .Property<DateTime>("last_update")
            .HasColumnType("datetime")
            .IsRequired();

    }
  }
}
```

O código está esteticamente desagradável e repetitivo, mas o problema não é meramente visual, e sim, na manutenção do código. Quando houver mais classes para configurarmos, algum ato de distração por parte do programador pode gerar uma mudança equivocada no código que ocasionará em problemas na execução do programa.

Suponhamos que a coluna `last_update` mude de nome e passe a se chamar `last_modified`. Teríamos de procura-la no método `OnModelCrating()` e modificar o nome. Mas poderemos realizar essa modificação na classe errada, e quanto mais classes disponíveis, maior a chance de erro. Veremos o que é possível ser feito para organizar melhor o nosso código de configuração.

A documentação (https://docs.microsoft.com/en-us/ef/core/what-is-new/) do Entity Fraework Core versão 2.0, no tópico **Self-contained type configuration for code first**, nos informa que foi resgatada uma funcionalidade do Entity Framework 6, que consiste em isolar o código de configuração de cada classe em uma outra classe. É justamente o que iremos fazer.

A classe que irá encapsular a configuração de cada classe, ficará armazenada dentro da pasta "Dados", localizada na área "Gerenciador de Soluções". Primeiramente, criaremos a classe de configuração para `Ator`, que chamaremos de `AtorConfiguration`.



removemos os *namespaces* desnecessários e aumentamos a da classe visibilidade. Teremos a `AtorConfiguration` pronta para ser trabalhada.

```
namespace Alura.Filmes.App.Dados
{
    public class AtorConfiguration
    {
    }
}
```

Iremos fazer com que essa classe implemente uma interface denominada `IEntityTypeConfiguration` para o tipo específico que queremos configurar. Essa interface possui um método `Configure()` para um `builder` da classe `Ator`. Teremos nessa classe apenas o código de configuração da classe `Ator`.

```
namespace Alura.Filmes.App.Dados
{
    public class AtorConfiguration : IEntityTypeConfiguration<Ator>
    {
        public void Configure(EntityTypeBuilder<Ator> builder)
        {

        }
    }
}
```

Moveremos todo o código de configuração da classe `Ator` para a nova classe criada `AtorConfiguration`. Não precisamos mais passar o `modelBuilder` no código de configuração, pois na linha do método `Configure()`, já existe a instância de `builder`. Iremos, também, importar o *system* para `DateTime`.

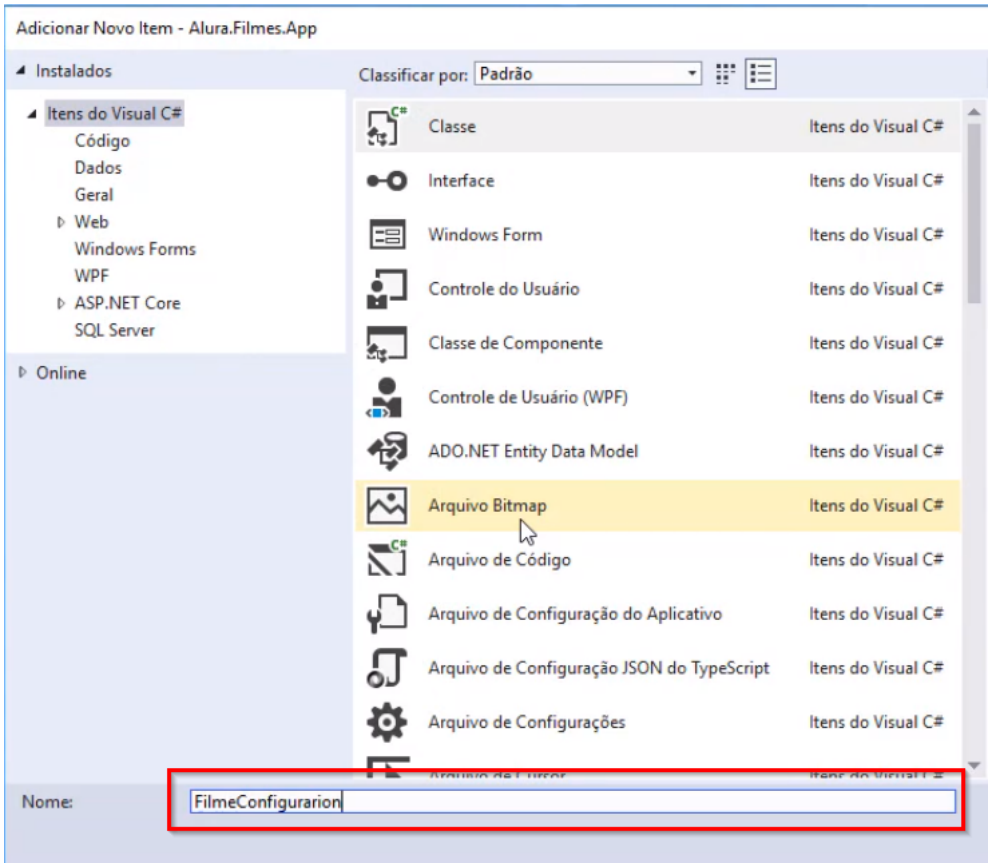
```
namespace Alura.Filmes.App.Dados
{
    public class AtorConfiguration : IEntityTypeConfiguration<Ator>
    {
        public void Configure(EntityTypeBuilder<Ator> builder)
        {
           builder
                .ToTable("actor");

           builder
                .Property(a => a.Id)
                .HasColumnName("actor_id");

           builder
                .Property(a => a.PrimieroNome)
                .HasCoumnName("first_name")
                .HasColumnType("varchar(45)")
                .IsRequired();

           builder
                .Property(a => a.UltimoNome)
                .HasColumnType("varchar(45)")
                .IsRequired();

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

        }
    }
}
```

Temos a configuração para a classe `Ator` isolada na classe `AtorConfiguration`. Para usarmos a classe que contém as configurações de `Ator` na classe de contexto, acionamos a instância do argumento de entrada `modelBuilder` e o método `ApplyConfiguration()` passando uma instância da configuração que queremos aplicar.

```
namespace Alura.Filmes.App.Dados
{
    public class AluraFilmesContexto : DbContext
    {
        public DbSet<Ator> Atores { get; set; }
        public DbSet<Filme> Filmes { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=(localdb)mssqllocaldb;Database=AluraFilmes;Trusted_(
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new AtorConfiguration());

            modelBuilder.Entity<Filme>()
                .ToTable("film")
```

```
            .ToTable("film");

        modelBuilder.Entity<Filme>()
            .Property(f => f.id)
            .HasColumnName("film_id");

        modelBuilder.Entity<Filme>()
            .Property(f => f.Titulo)
            .HasColumnName("title")
            .HasColumnType("varchar(255)")
            .IsRequired();

        modelBuilder.Entity<Filme>()
            .Property(f => f.Descricao)
            .HasColumnName("description")
            .HasColumnType("text");

        modelBuilder.Entity<Filme>()
            .Property(f => f.AnoLancamento)
            .HasColoumnName("release_year")
            .HasColumnTypoe("varchar(4)");

        modelBuilder.Entity<Filme>()
            .Property(f => f.Duracao)
            .HasColumnName("length")


        modelBuilder.Entity<Filme>()
            .Property<DateTime>("last_update")
            .HasColumnType("datetime")
            .IsRequired();

    }
  }
}
```

Faremos o mesmo procedimento para isolar o código de configuração da classe `Filme`. Na pasta "Dados", criaremos uma nova classe denominada `FilmeConfiguration`.



Na nova classe criada, iremos adicionar a interface `IEntityTypeConfiguration` e adicionar o método `Configure()`. Com isso, podemos trazer todo o código de configuração da classe `Filme` para a classe `FilmeConfiguration`.

```
namespace Alura.Filmes.App.Dados
{
    public class FilmeConfiguration : IEntityTypeConfiguration
    {
        public void Configure(EntityTypeBuilder<Filme>builder)
        {

            modelBuilder.Entity<Filme>()
                .ToTable("film")

            modelBuilder.Entity<Filme>()
                .Property(f => f.id)
                .HasColumnName("film_id");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Titulo)
                .HasColumnName("title")
                .HasColumnType("varchar(255)")
                .IsRequired();

            modelBuilder.Entity<Filme>()
                .Property(f => f.Descricao)
                .HasColumnName("description")
                .HasColumnType("text");

            modelBuilder.Entity<Filme>()
                .Property(f => f.AnoLancamento)
                .HasColoumnName("release_year")
```

```
                .HasColumnTypoe("varchar(4)");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Duracao)
                .HasColumnName("length")

            modelBuilder.Entity<Filme>()
                .Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .IsRequired();

        }
    }
}
```

Substiuiremos o `modelBuilder` por `builder` e importaremos o *system*.

```
namespace Alura.Filmes.App.Dados
{
    public class FilmeConfiguration : IEntityTypeConfiguration
    {
        public void Configure(EntityTypeBuilder<Filme>builder)
        {

            modelBuilder.Entity<Filme>()
                .ToTable("film")

            modelBuilder.Entity<Filme>()
                .Property(f => f.id)
                .HasColumnName("film_id");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Titulo)
                .HasColumnName("title")
                .HasColumnType("varchar(255)")
                .IsRequired();

            modelBuilder.Entity<Filme>()
                .Property(f => f.Descricao)
                .HasColumnName("description")
                .HasColumnType("text");

            modelBuilder.Entity<Filme>()
                .Property(f => f.AnoLancamento)
                .HasColoumnName("release_year")
                .HasColumnTypoe("varchar(4)");

            modelBuilder.Entity<Filme>()
                .Property(f => f.Duracao)
                .HasColumnName("length")

            modelBuilder.Entity<Filme>()
                .Property<DateTime>("last_update")
                .HasColumnType("datetime")
                .IsRequired();
```

```
        }
    }
}
```

O código de configuração de `Filme` também está isolado. Faremos o mesmo procedimento de aplicar o método `ApplyConfiguration()` na classe de contexto.

```
namespace Alura.Filmes.App.Dados
{
    public class AluraFilmesContexto : DbContext
    {
        public DbSet<Ator> Atores { get; set; }
        public DbSet<Filme> Filmes { get; set; }

        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            optionsBuilder.UseSqlServer("Server=(localdb)mssqllocaldb;Database=AluraFilmes;Trusted_(
        }

        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new AtorConfiguration());
            modelBuilder.ApplyConfiguration(new FilmeConfiguration();)

        }
    }
}
```

O programa será executado corretamente.