07

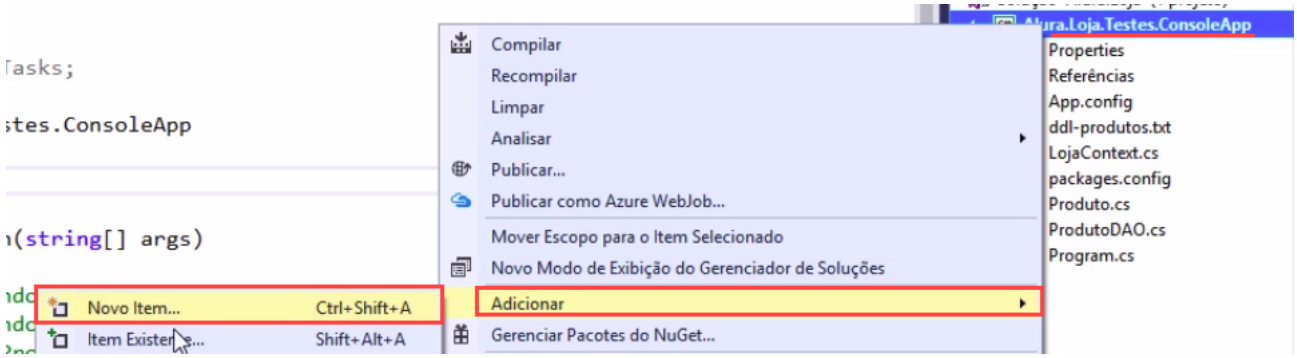
Organizando o código de acesso em um DAO

Transcrição

Aprendemos a trabalhar com todas as operações do *CRUD*, porém fizemos isso em uma classe de teste. Criaremos uma nova classe `ProdutoDAO`, mas dessa vez fazendo acesso aos dados usando o Entity.

Antes de implementarmos a `ProdutoDAO`, vamos utilizar um recurso de Orientação a Objetos que vai nos auxiliar na criação da classe. Como as classes *DAO*, tanto a que usa Entity e a que usa ADO.Net, possuem os mesmo métodos, podemos afirmar que elas assinam o mesmo contrato. Partindo dessa ideia, criaremos a interface `IProdutoDAO`.

Clicando com o botão direito em `Alura.Loja.Teste.ConsoleApp`, selecionaremos a opção "Adicionar > Novo Item".



Escolheremos a opção "Interface", colocando o nome `IProdutoDAO`. A interface irá conter as quatro operações do *CRUD*, adicionar, atualizar, remover, e listar.

```
namespace Alura.Loja.Testes.ConsoleApp
{
    interface IProdutoDAO
    {
        void Adicionar(Produto p);
        void Atualizar(Produto p);
        void Remover(Produto p);
        IList<Produto> Produtos();
    }
}
```

Faremos a classe `ProdutoDAO` que utiliza o ADO.Net implementar a interface. Além disso, é necessário mudar a visibilidade `internal` dos métodos para `public`.

```
internal class ProdutoDAO : IDisposable, IProdutoDAO
{

    //...

    public void Adicionar(Produto p)
    {
        //...
    }
```

```
        public void Atualizar(Produto p)
        {
            //...
        }

        public void Remover(Produto p)
        {
            //...
        }

        public IList<Produto> Produtos()
        {
            //...
        }
    }
```

Da mesma forma como criamos a interface, criaremos um nova classe chamada `ProdutoDAOEntity`. Faremos a nova classe implementar a interface `IDisposable` e `IProdutoDAO`.

```
class ProdutoDAOEntity : IProdutoDAO, IDisposable
{
    public void Adicionar(Produto p)
    {
        throw new NotImplementedException();
    }

    public void Atualizar(Produto p)
    {
        throw new NotImplementedException();
    }

    public void Dispose()
    {
       throw new NotImplementedException();
    }

    public IList<Produto> Produtos()
    {
        throw new NotImplementedException();
    }

    public void Remover(Produto p)
    {
        throw new NotImplementedException();
    }
}
```

Agora na classe `Program`, em vez de criarmos os objetos da classe `LojaContext`, criaremos de `ProdutoDAOEntity`. Dessa forma perderemos todos os métodos relacionados ao `LojaContext`, sendo necessário trocar pela chamada ao respectivo método de `ProdutoDAOEntity`.

```
static void Main(string[] args)
{
    //Chamada de Métodos...
}
```

```
    }

    private static void AtualizaProduto()
    {
        // inclui um produto

        // atualiza o produto
        using(var repo = new ProdutoDAOEntity())
        {
            Produto primeiro = repo.Produtos().First();
            primeiro.Nome = "Cassino Royale - Editado";

            repo.Atualizar(primeiro);
        }
        RecuperaProdutos();
    }

    private static void ExcluirProdutos()
    {
        using (var repo = new ProdutoDAOEntity())
        {
            IList<Produto> produtos = repo.Produtos();
            foreach (var item in produtos)
            {
                repo.Remover(item);
            }
        }
    }

    private static void RecuperarProdutos()
    {
        using (var repo = new ProdutoDAOEntity())
        {
            IList<Produto> produtos = repo.Produtos();
            Console.WriteLine("Foram encontrados {0} produto(s).", produtos.Count);
            foreach (var item in produtos)
            {
                Console.WriteLine(item.Nome);
            }
        }
    }

    private static void GravarUsandoEntity()
    {
        Produto p = new Produto();
        p.Nome = "Cassino Royale";
        p.Categoria = "Filmes";
        p.Preco = 19.89;

        using(var repo = new ProdutoDAOEntity())
        {
            repo.Adicionar(p);
        }
    }
```

Fizemos a troca, tiramos de `LojaContext` e passamos a usar `ProdutoDAOEntity`. Entretanto, a classe `ProdutoDAOEntity` ainda não está usando o Entity. Para utilizarmos qualquer método Entity, precisamos acessar a partir de um objeto de contexto na

aplicação. Por isso, criaremos um *atributo* privado do tipo `LojaContext` .

```
class ProdutoDAOEntity : IProdutoDAO, IDisposable
{
    private LojaContext contexto;

    //...
}
```

Para incluir um produto, dentro do método `Adicionar()` vamos apagar todo o conteúdo e invocar o método `Add()` da propriedade `contexto.Produto` . É necessário salvar as mudanças do contexto.

```
class ProdutoDAOEntity : IProdutoDAO, IDisposable
{
    private LojaContext contexto;

    public void Adicionar(Produto p)
    {
        contexto.Produtos.Add(p);
        contexto.SaveChanges();
    }

    // ...
}
```

Para atualizar, faremos da mesma forma que fizemos para adicionar um produto, mas chamaremos o método `Update()` .

```
class ProdutoDAOEntity : IProdutoDAO, IDisposable
{

    // ...

    public void Atualizar(Produto p)
    {
        contexto.Produtos.Update(p);
        contexto.SaveChanges();
    }

    // ...
}
```

Para remover, chamaremos o método `Remove()` .

```
class ProdutoDAOEntity : IProdutoDAO, IDisposable
{

    // ...

    public void Remover(Produto p)
    {
        contexto.Produtos.Remove(p);
        contexto.SaveChanges();
```

```
        }

        // ...
    }
```

Para listar os produtos basta retornarmos o método `ToList()`:

```
    class ProdutoDAOEntity : IProdutoDAO, IDisposable
    {

        // ...

        public IList<Produto> Produtos()
        {
            return contexto.Produtos.ToList();
        }

    }
```

Ainda falta criarmos uma instância de `LojaContext`. Faremos isso no construtor da classe. No método `Dispose()`, chamaremos o `contexto.Dispose()`

```
    class ProdutoDAOEntity : IProdutoDAO, IDisposable
    {
        private LojaContext contexto;

        public ProdutoDAOEntity()
        {
            this.contexto = new LojaContext();
        }

        public void Dispose()
        {
            contexto.Dispose();
        }

        //...
    }
```

Dessa forma, temos uma classe *DAO* que faz os acessos ao banco de dados de uma forma muito mais enxuta e limpa, com praticamente metade de linhas de código de uma classe com ADO.Net.