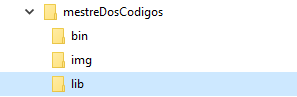
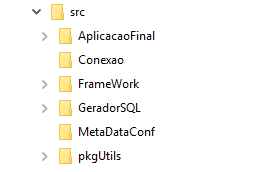
Estrutura de arquivos:

bin: Arquivos executáveis do projeto. Já existem alguns arquivos de configuração presentes lá. Fique atendo apenas nos arquivos .ini, que contém configurações de acesso à dados.

img: arquivos que estamos utilizando nas aplicações

lib: aquivos compilados de todos os projetos. Em caso de erro de compilação por falta de arquivos, verifique se o library path aponta para esta pasta.

src: É a pasta que contém todos os arquivos de código

AplicaçãoFinal: Esta pasta contém a aplicação final.

Conexao: Units para abstração da criação de conexão;

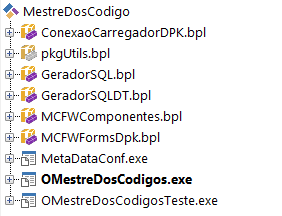
Framework: Componentes que vão auxiliar na criação da aplicação

GeradorSQL: Componente de geração de SQL;

MetaDataConf: Aplicação de configuração de metadados;

pkgUtils: arquivos comuns a todo o projeto;

Ordem de compilação

A aplicação deve ser compilada nesta ordem. Observe que todos os pacotes estão instalados (nesta ordem), exceto a pkgUtils.

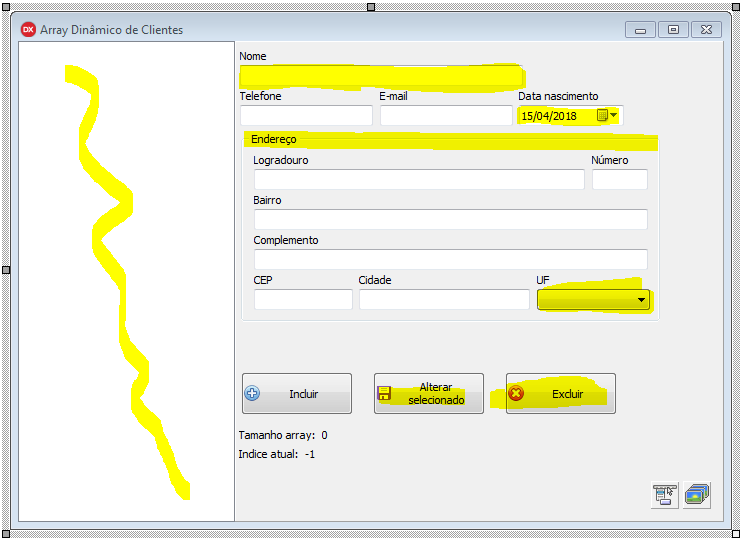
1) Criar uma aplicação que trabalhe com array dinâmico. O mesmo deve armazenar dados do cliente. Devem ser criados mecanismos para incluir, alterar e excluir um determinado valor do array.

OmestreDosCodigos.Impl.Cliente.ListaArrayDinamico

Optei por criar um array dinâmico usando Generics. Assim a classe que o manipula, pode lidar com qualquer tipo de dados. Seja um tipo básico, um objeto ou uma interface.

2) Criar um formulário que utilize ao menos 4 elementos de interface distintos como Edits, Buttons, Labels, Groupbox, Checkbox entre outros. Implementar validação de data, intervalos, número, obrigatoriedade.

OMestreDosCodigos.UI.frmArrayClientes

Esta tela está sendo utilizada para:

* Preencher os dados na classe Cliente;
* Manipular o array dinâmico do exercício um.

As validações requeridas estão sendo implementadas nos objetos de valor

* pkgUtils.Impl.EMail
* pkgUtils.Impl.Endereco
* pkgUtils.Impl.Telefone
* pkgUtils.Impl.Validavel

3) Crie uma aplicação que utilize Orientação a Objetos. As classes devem ter baixo acoplamento e alta coesão, também devem ser aplicados os conceitos de herança, polimorfismo e interfaces e utilizando ao menos 3 padrões de projeto explicando o motivo de sua utilização.

Builder

OMestreDosCodigos.Impl.Cliente.Builder

Utilizado para facilitar a construção de clientes à partir da interface com o usuário.

Local onde é utilizado: OmestreDosCodigos.UI.frmArrayClientes

Builder + Director

SQL.Impl.Coluna.Builder

SQL.Impl.Coluna.Director

Tanto Builder quanto Director garantem a construção correta dos objetos, instanciando corretamente as classes do tipo em comum.

Factory

Conexao.Intf.DataFabrica

Conexao.Impl.Fabrica.FireDac

Utilizada para abstrair a criação de componentes de conexão. “.FireDac” é uma implementação dessa fábrica que gera objetos para FireDac.

SQL.Impl.Fabrica

SQL.Intf.Fabrica

Permite criar os objetos SQL apropriados ao tipo de banco que os utilizarão. Assim garantimos um tipo de implementação em comum.

Strategy

As interfaces

* SQL.Intf.Coluna
* SQL.Intf.Condicao
* SQL.Intf.Juncao
* SQL.Intf.Select
* SQL.Intf.Tabela

São abstrações de tipos para que podem ter várias implementações. A ideia é que possam ser implementadas classes especializadas para cada banco.

Facade

GeradorSQL.Comp.Select

TMCSelect, além de ser o componente de geração de SQL's, também funciona como facade, escondendo atrás de si toda a complexidade envolvendo a criação dos objetos SQL para cada banco.

4) Utilizando Regex, deve ser criada uma rotina para validar pelo menos três tipos diferentes de entrada do usuário.

As validações estão presentes nos objetos de valor:

* pkgUtils.Impl.EMail
* pkgUtils.Impl.Endereco
* pkgUtils.Telefone

5) Criar uma aplicação que armazena os dados em um banco de dados relacional. Deve possuir tabelas pai-filho. Devem ser criadas rotinas para inclusão, alteração, busca e exclusão dos dados do usuário e um relatório. descrever no código as precauções tomadas em relação a qualidade, usabilidade e performance.

MetaDataConf.UI.ffwConfigEntidade

Esta tela irá gerenciar a manutenção dos metadados configuráveis do banco de dados. Ela alimenta as tabelas Entidade e Ent\_Propriedades do banco de dados.

6) Criar um componente que contenha as propriedades: Colunas, Tabelas e Condições permitindo gerar comandos SQL. Criar uma aplicação que utilize esse componente realizando a consulta e exibindo os dados retornados . Esse componente deve possuir um ícone personalizado.

GeradorSQL.Comp.Select.pas (TMCSelect)

Componente que gera consultas SQL otimizadas para cada banco de dados.

10) Cobrir as aplicações criadas nos itens 3 e 6 com pelo menos 80% de cobertura de testes unitários utilizando mocks e stubs.

Aplicação final (OmesteDosCodigosTeste) – 81% de cobertura

GeradorSQL.\* - 84% de cobertura