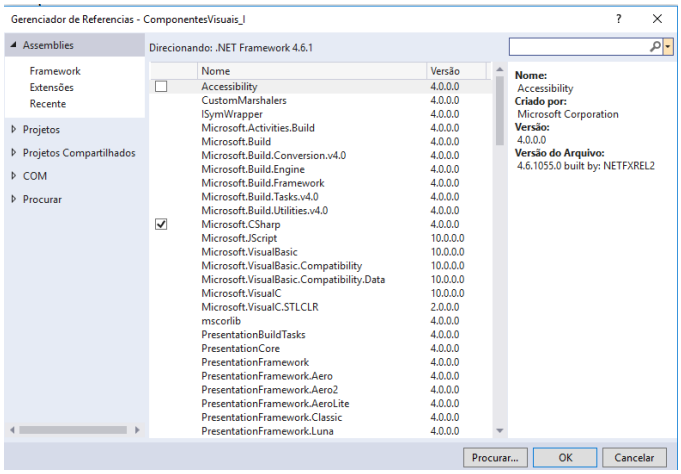
VISUAL STUDIO – PROJETO CRUD – PARTE III

Crie um projeto com o nome SistemaCRUD. Para isso utilize botão direito sobre o nome da solução. Selecione Adicionar / Aplicação do Windows Forms. Confirme no botão Próximo. Informe o nome do projeto. Confirme no botão Criar.

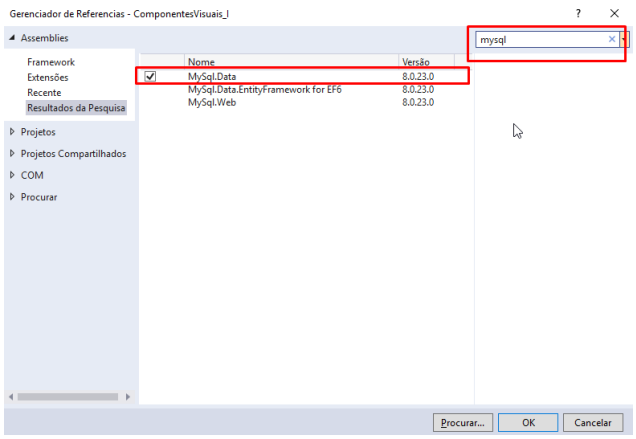
Defina o projeto como projeto de inicialização. Para isso, use o botão direito sobre o nome da solução. Selecione definir como projeto de inicialização.

Como será utilizado o MySQL, então é necessário importar o conector do MySQL para o Visual Studio. Acesse o link https://dev.mysql.com/downloads/connector/net/ e Instale o conector do MySQL.

Em seguida, deve-se adicionar a referência para o MySQL. Para isso dê um duplo clique sobre o nome do projeto. Selecione Adicionar e depois selecione Referência. Será apresentada uma janela semelhante à indicada a seguir:



Informe MySQL na caixa de pesquisa e marque o checkbox de MySqlData. Confirme em OK.



Dessa forma, o conector MySQL passa a ser reconhecido pelo projeto.

Se você não possui o XAMPP, instale-o para que se tenha um sevidor MySQL no computador.

Em seguida, crie o banco de dados **sistema_clientes.**

Crie a tabela como ilustrado a seguir:

```
CREATE TABLE nacionalidade (
 id INT(11) NOT NULL AUTO_INCREMENT,
 sigla VARCHAR(2) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
 pais VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_general_ci',
 PRIMARY KEY (id) USING BTREE
)
COLLATE='utf8mb4_general_ci'
ENGINE=InnoDB
;
```

```
CREATE TABLE clientes (
  id INT(11) NOT NULL AUTO_INCREMENT,
  nome VARCHAR(50) NOT NULL COLLATE 'utf8_general_ci',
  sexo VARCHAR(20) NOT NULL COLLATE 'utf8_general_ci',
  nascimento DATE NOT NULL,
  idnacionalidade INT(11) NULL DEFAULT NULL,
  PRIMARY KEY (id),
  INDEX cliente_nacionalidade_01 (idnacionalidade) USING BTREE,
  CONSTRAINT cliente_nacionalidade_01 FOREIGN KEY (idnacionalidade)
      REFERENCES sistema_clientes.nacionalidade (id) ON UPDATE RESTRICT ON DELETE RESTRICT
)
COLLATE='utf8_general_ci'
ENGINE=InnoDB
;
```

Note que a tabela cadastro possui uma chave estrangeira correspondente ao id da tabela nacionalidade.

Em seguida, crie uma nova pasta dentro do projeto usando o botão direito sobre o nome do projeto. Selecione Adicionar e, depois, nova pasta. Informe **ENTIDADES**.

Com o botão direito sobre a pasta Entidades, selecione Adicionar e, em seguida, selecione classes. Informe o nome **Cliente**.

Será criada essa nova classe:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace SistemaCRUD.ENTIDADES
8   {
        0 referências
9       public class Cliente
10      {
11      }
12  }
13
```

Ajuste a classe para o indicado a seguir:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaCRUD.ENTIDADES
{
    0 referências
    public class Cliente
    {
        int id;
        String nome, sexo;
        DateTime nascimento;

        0 referências
        public int Id { get => id; set => id = value; }
        0 referências
        public string Nome { get => nome; set => nome = value; }
        0 referências
        public string Sexo { get => sexo; set => sexo = value; }
        0 referências
        public DateTime Nascimento { get => nascimento; set => nascimento = value; }
    }
}
```

Em seguida, crie a classe Conexao dentro da raiz do projeto. Ou seja: Dê um clique com o botão direito sobre o nome do projeto. Selecione Adicionar e depois selecione classe. No entanto, fica a seu critério em criar ou não essa classe dentro de uma pasta específica. Essa classe é responsável pela conexão ao banco de dados MySQL.

Codifique a classe Conexao com o trecho de código a seguir:

```
using MySql.Data.MySqlClient;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SistemaCRUD
{
    1 referência
    class Conexao
    {
        String conexao = "SERVER = localhost; DATABASE=sistema_clientes; UID=vicente; PWD=vicente;";
        public MySqlConnection con = null;
        5 referências
        public void AbrirConexao()
        {
            try
            {
                con = new MySqlConnection(conexao);
                con.Open();

            }
            catch (MySqlException ex)
            {
                throw (ex);
            }

        }
        3 referências
        public void FecharConexao()
        {
            try
            {
                con = new MySqlConnection(conexao);
                con.Close();
            }
            catch (MySqlException ex)
            {
                throw (ex);
            }

        }
    }
}
```

A linha 12 cria a String de conexao. A linha 13 inicializa a variável **con** de conexão (do tipo MySqlConnection). Deve-se importar a biblioteca usando o comando indicado na linha 1.

As linhas 14 a 31 destacam o método AbrirConexao(). As linhas 16 a 25 engloba o trecho try / catch. A linha 18 define a variável de conexão **con** com a string **conexao**. A linha 19 abre a conexão com o banco de dados. Caso ocorra algum erro, a linha 24 (do catch) será executada.

As linhas 28 a 40 delimitam o método FecharConexao(). A linha 32 liga a variável **con** com a string de conexão **conexao** e a linha 33 fecha a conexão. Caso ocorra algum erro, a linha 37 será executada.

Em seguida, crie a pasta **DAO**. Depois, crie a classe ClienteDAO dentro dessa pasta, com a seguinte estrutura:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Data;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7   using MySql.Data.MySqlClient;
8   using SistemaCRUD.ENTIDADES;
9
10  namespace SistemaCRUD.DAO
11  {
        1 referência
12      public class ClienteDAO
13      {
14          MySqlCommand sql;
15          String sqlStr = "";
16          Conexao conexao = new Conexao();
17
            1 referência
18          public DataTable Listar()
19          {
20              try
21              {
22                  conexao.AbrirConexao();
23                  sqlStr = "SELECT * FROM clientes";
24                  sql = new MySqlCommand(sqlStr, conexao.con);
25                  MySqlDataAdapter da = new MySqlDataAdapter();
26                  da.SelectCommand = sql;
27                  DataTable dt = new DataTable();
28                  da.Fill(dt);
29                  return dt;
30              }
31              catch (Exception)
32              {
33                  throw;
34              }
35          }
36
37          public void Salvar(Cliente objeto)
38          {
39              try
40              {
41                  conexao.AbrirConexao();
42                  sqlStr = "INSERT INTO clientes (nome, sexo, nascimento) VALUES (@NOME, @SEXO, @NASCIMENTO)";
43                  sql = new MySqlCommand(sqlStr, conexao.con);
44                  sql.Parameters.AddWithValue("@NOME", objeto.Nome);
45                  sql.Parameters.AddWithValue("@SEXO", objeto.Sexo);
46                  sql.Parameters.AddWithValue("@NASCIMENTO", objeto.Nascimento);
47                  sql.ExecuteNonQuery();
48                  conexao.FecharConexao();
49
50
51
52              }
53              catch (Exception)
54              {
55                  throw;
56              }
57          }
58
            1 referência
59          public void Editar(Cliente objeto)
60          {
61              try
62              {
63                  conexao.AbrirConexao();
64                  sqlStr = "UPDATE clientes SET nome=@NOME, sexo=@SEXO, nascimento=@NASCIMENTO WHERE id=@ID";
65                  sql = new MySqlCommand(sqlStr, conexao.con);
66                  sql.Parameters.AddWithValue("@NOME", objeto.Nome);
67                  sql.Parameters.AddWithValue("@SEXO", objeto.Sexo);
68                  sql.Parameters.AddWithValue("@NASCIMENTO", objeto.Nascimento);
69                  sql.Parameters.AddWithValue("@ID", objeto.id);
70                  sql.ExecuteNonQuery();
71                  conexao.FecharConexao();
72
73
74
75              }
76              catch (Exception)
77              {
78                  throw;
79              }
80          }
81
```

```
82      public void Excluir(Cliente objeto)
83      {
84          try
85          {
86              conexao.AbrirConexao();
87              sqlStr = "DELETE FROM clientes WHERE id=@ID";
88              sql = new MySqlCommand(sqlStr, conexao.con);
89              sql.Parameters.AddWithValue("@ID", objeto.id);
90              sql.ExecuteNonQuery();
91              conexao.FecharConexao();
92          }
93          catch (Exception)
94          {
95              throw;
96          }
97      }
98
99
            1 referência
100     public DataTable Buscar(Cliente objeto)
101     {
102         try
103         {
104             conexao.AbrirConexao();
105             sqlStr = "SELECT * FROM clientes WHERE nome LIKE @NOME";
106             sql = new MySqlCommand(sqlStr, conexao.con);
107             sql.Parameters.AddWithValue("@NOME", objeto.nome + "%");
108             MySqlDataAdapter da = new MySqlDataAdapter();
109             da.SelectCommand = sql;
110             DataTable dt = new DataTable();
111             da.Fill(dt);
112             return dt;
113         }
114         catch (Exception)
115         {
116             throw;
117         }
118     }
119
120
121 }
122 }
```

A utilização de System.Data (na linha 3) é importante devido ao fato dessa classe referenciar a classe DataTable. Essa classe permite armazenar dados lidos de um banco de dados e deixá-los disponíveis na memória, os quais podem ser utilizados por algum componente de dados como, por exemplo, DataGridView.

A linha 7 faz referência ao conector MySQL. A linha 8 faz referência às classes que estão na pasta ENTIDADES, pois utiliza objetos da classe Cliente.

As linhas 14 a 16 define os atributos da classe: **sql** do tipo MysqCommand, que é o comando que permite executar um script SQL; a string **sqlStr** que armazena o script SQL e **conexao** (objeto da classe Conexao).

As linhas 18 a 35 delimitam o método Listar(). Esse método permite acessar todos os clientes da tabela **clientes**. Para isso, abre a conexão na linha 22. Define o script SQL (com Select) na lina 23. Faz a ligação entre o script SQL e a conexão **con** para o objeto MySqlCommand **sql**, na linha 24**.** Na linha 25 cria o objeto **da** (DataAdapter). Esse objeto pega os dados do banco de dados (executados por MySqlCommand). A linha 27 cria o objeto **dt** do tipo DataTable. Um DataTable armazena os dados do banco de dados, sendo uma coleção de linhas e colunas. Assim, o DataAdapter adapta os dados que vem do banco de dados e os adapta para ser armazenados no DataTable. É o que faz a linha 28: preenche (fill) os dados do DataAdapter no DataTable (dt). Como o método Listar é do tipo DataTable, ele retorna essa coleção de registros na linha 29.

As linhas 37 a 57 delimitam o método Salvar(Cliente objeto). Esse método faz a inserção de dados na tabela clientes. Ele possui o parâmetro **objeto** que é do tipo Cliente (linha 37). As linhas 39 a 56 delimitam o bloco try/catch. A linha 41 abre a conexão. A linha 42 cria o script SQL. Esse script possui três parâmetros cujos nomes se iniciam com @, onde cada parâmetro corresponde ao conteúdo de um campo do comando INSERT. A linha 43 liga o script SQL e a conexão no MySqlCommand **sql**. As linhas 44 a 46 relaciona os parâmetros com os respectivos conteúdos que serão inseridos na tabela **clientes**. A linha 47 executa a instrução SQL e, na linha 48, fecha a conexão chamando o método FecharConexao(). Qualquer detalhe de erro que ocorra no trecho try, o catch será executado.

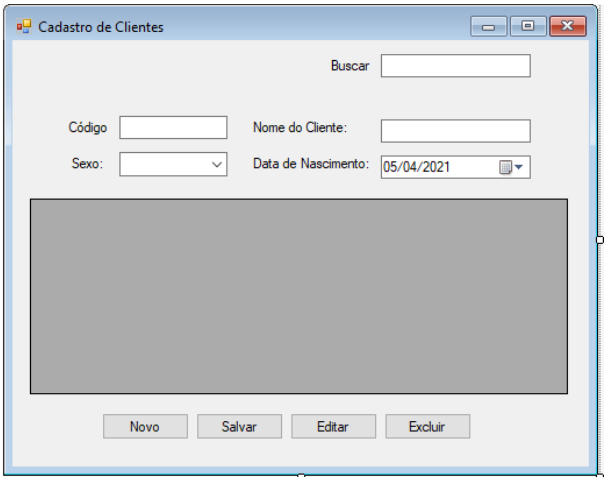
As linhas 59 a 80 delimitam o bloco de código do método Editar(Cliente objeto). Esse método permite efetuar a alteração de dados na tabela clientes e possui o parâmetro objeto, que do tipo Cliente. A linha 63 abre a conexão. A linha 64 cria a string SQL com os respectivos parâmetros, inclusive do código da chave primária id. A linha 65 estabelece a ligação entre o script SQL e a

conexão para o MySqlCommand. As linhas 66 a 69 passa os parâmetros para o MySqlCommand. A linha 70 executa o script SQL e a linha 71 fecha a conexão. Caso ocorra algum erro no try, o trecho catch será executado.

O trecho das linhas 82 a 97 delimita o método Excluir(Cliente objeto). Esse método permite excluir um cliente de acordo com sua chave primária id. Ele recebe a instância objeto da classe Cliente. A linha 86 abre a conexão. A linha 87 cria o script SQL. A linha 88 relaciona o script SQL e a conexão ao MySqlCommand. A linha 89 passa os parâmetros para MySqlCommand. A linha 90 executa o script SQL. A linha 91 fecha a conexão. Caso ocorra algum erro no trecho do try, o trecho catch é executado.

As linhs 100 a 118 delimita o método Buscar(Cliente objeto). Esse método permite efetuar a busca de um trecho da parte de um nome informado pelo usuário, cujo conteúdo está no nome do objeto da classe Cliente. A linha 104 abre a conexão. A linha 105 cria o script SQL para a busca usando o LIKE. A linha 106 relaciona o script SQL e a conexão no MySqlCommand. A linha 107 inclui o parâmetro para a correta execução do script SQL. A linha 108 cria o objeto **da** do tipo DataApdater. A linha 109 atribui o resultado da variável **sql** (MySqlCommand) ao DataAdapter. A linha 110 cria o objeto **dt** (DataTable). A linha 111 atribui os registros do DataAdapter ao DataTable. A linha 112 retorna o DataTable. O trecho catch é executado caso ocorra algum erro dentro do trecho try.

Em seguida, crie a pasta Views e dentro dela crie o formulário FrmClientes. Insira nesse formulário os seguintes componentes: três TextBox , um ComboBox, um DateTimePicker, cinco componentes Label (um para cada um dos componentes anteriores),um DataGridView e quatro botões. Observe o designer da janela a seguir:

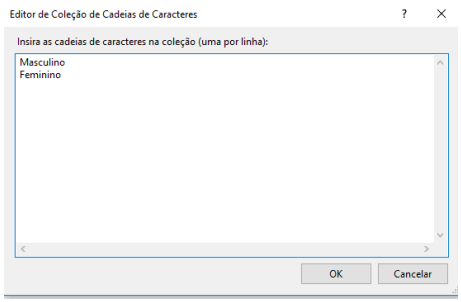


O DataGridView é o componente que ocupa uma boa área da parte inferior da tela na cor cinza escuro. Esse componente permitirá listar os registros dos clientes.

Acesse esse componente e altere a sua propriedade SelectionMode para FullRowSelect, para que a seleção de um registro ocorra com a seleção de uma linha (row).

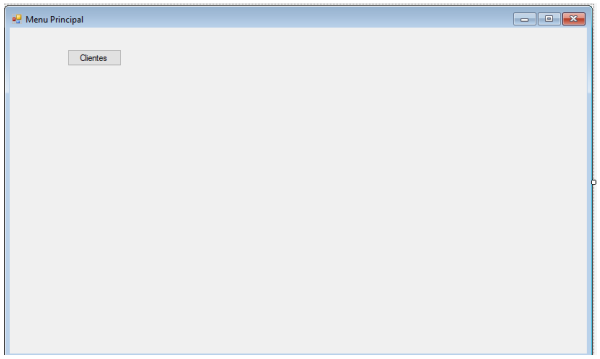
Observando a ordem em que aparacem na janela, informe a propriedade Name para cada componente, como se segue: txtPesquisa, txtCodigo, txtNomeCliente, cbxSexo, dtpDataNascimento, dgvDados, btnNovo, btnSalvar, btnEditar e btnExcluir.

Ajuste a propriedade Items do componente cbxSexo. Acesse essa propriedade e clique no botão de reticências. Na janela que se abre, informe como ilustrado:



Confirme em OK. São valores que são constantes e que serão opções para o usuário selecionar em tempo de execução.

Em seguida, renomeie o formulário Form1 para frmPrincipal. Altere o título do formulário para Menu Principal e, em seguida, insira um Button. Altere o seu Name para btnClientes. Altere a propriedade WindowState para Maximized. Observe o layout indicado:

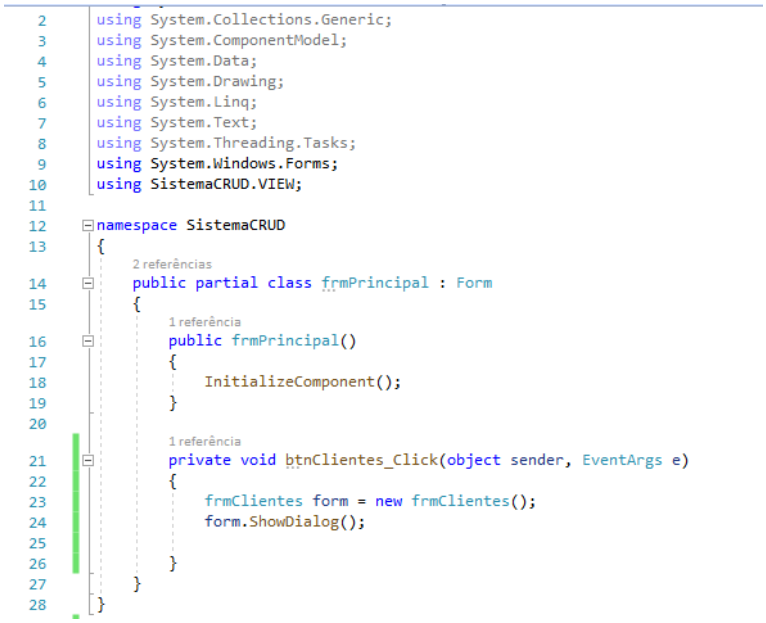


Dê um duplo clique no botão btnClientes e codifique o seguinte:

```
21      private void btnClientes_Click(object sender, EventArgs e)
22      {
23          frmClientes form = new frmClientes();
24          form.ShowDialog();
25
26      }
```

A linha 23 cria uma instância do formulário frmClientes. A linha 24 abre essa instância usando o método ShowDialog() que permite abrir uma janela no formato modal. Esse formato impede que se acesse outras janelas enquanto ela estiver aberta.

O código completo do formulário frmPrincipal:



```
2   using System.Collections.Generic;
3   using System.ComponentModel;
4   using System.Data;
5   using System.Drawing;
6   using System.Linq;
7   using System.Text;
8   using System.Threading.Tasks;
9   using System.Windows.Forms;
10  using SistemaCRUD.VIEW;
11
12  namespace SistemaCRUD
13  {
        2 referências
14      public partial class frmPrincipal : Form
15      {
            1 referência
16          public frmPrincipal()
17          {
18              InitializeComponent();
19          }
20
            1 referência
21          private void btnClientes_Click(object sender, EventArgs e)
22          {
23              frmClientes form = new frmClientes();
24              form.ShowDialog();
25
26          }
27      }
28  }
```

As linhas 9 e 10 são importantes para que se possa referenciar as respectivas bibliotecas dentro dessa classe.

Ative o evento Click desse botão, efetuando um duplo clique sobre ele. Digite o seguinte trecho de código:

Sem a linha 10, a linha correspondente à linha 23 deve ser codificaca com VIEW.frmClientes form = new VIEW.frmClientes().

Crie a pasta Model e, em seguida, crie a classe ClienteModel. Essa classe irá realizar a ligação da view com a parte dao. Essa classe faz o papel de um controle específico da view.

Com a classe ClienteModel aberta, ajuste seu código como ilustrado a seguir:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Data;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7  using SistemaCRUD.DAO;
8  using SistemaCRUD.ENTIDADES;
9
10 namespace SistemaCRUD.Model
11 {
       1 referência
12     public class ClienteModel
13     {
14         ClienteDAO clienteDAO = new ClienteDAO();
           1 referência
15         public DataTable Listar()
16         {
17             try
18             {
19                 DataTable dt = new DataTable();
20                 dt = clienteDAO.Listar();
21                 return dt;
22
23             }
24             catch (Exception ex)
25             {
26                 throw ex;
27             }
28         }
29
           1 referência
30         public void Salvar(Cliente objeto)
31         {
32             try
33             {
34                 clienteDAO.Salvar(objeto);
35             }
36             catch (Exception ex)
37             {
38                 throw ex;
39             }
40         }
41
           1 referência
42         public void Editar(Cliente objeto)
43         {
44             try
45             {
46                 clienteDAO.Editar(objeto);
47             }
48             catch (Exception ex)
49             {
50                 throw ex;
51             }
52         }
53
           1 referência
54         public void Excluir(Cliente objeto)
55         {
56             try
57             {
58                 clienteDAO.Excluir(objeto);
59             }
60             catch (Exception ex)
61             {
62                 throw ex;
63             }
64         }
65
           1 referência
66         public DataTable buscar(Cliente objeto)
67         {
68             try
69             {
70                 DataTable dt = new DataTable();
71                 dt = clienteDAO.Buscar(objeto);
72                 return dt;
73
74             }
75             catch (Exception ex)
76             {
77                 throw ex;
78             }
79         }
80     }
81 }
```

É criada a instância da classe ClienteDAO (linha 14), a qual possui as instruções SQL e a execução dessas instruções no banco de dados. As linhas 15 a 28 estabelece o método Listar(), o qual cria um objeto DataTable na linha 19. Executa o método Listar da classe ClienteDAO e retorna esses dados,

armazenando-os no DataTable (dt), linha 20. Retorna o DataTable na linha 21, visto que esse método é do tipo DataTable. O trecho 24 a 27 (catch) será executado caso ocorra algum erro no try.

O trecho de linhas 30 a 40 delimita a ação do método Salvar(Cliente objeto). Ele simplesmente executa o método Salvar(objeto) da classe ClienteDAO na linha 34. Caso ocorra algum erro (nesse trecho try), o trecho catch é chamado.

Os métodos Editar (42 a 52) e Excluir (54 a 64) efetuam ações semelhantes, chamando os respectivos métodos na classe ClienteDAO, respectivamente, linhas 46 e 58.

As linhas 66 a 79 estabelecem o método Buscar(Cliente objeto), cujo trecho try cria um objeto DataTable (linha 70). Atribui o conteúdo da busca de ClienteDAO ao DataTable (linha 71) e retorna o DataTable (linha 72).

Salve todos os arquivos do projeto.

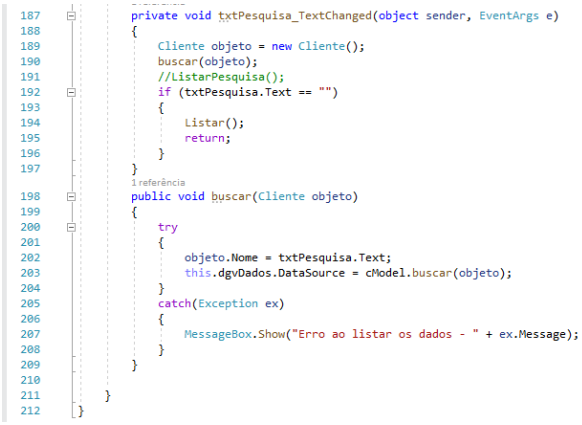
Abra o formulário frmClientes e pressione F7 para abrir a sua área de codificação. Ajuste o código dessa classe como ilustrado a seguir:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using SistemaCRUD.Model;
using SistemaCRUD.ENTIDADES;

namespace SistemaCRUD.VIEW
{
    3 referências
    public partial class frmClientes : Form
    {
        ClienteModel cModel = new ClienteModel();
        1 referência
        public frmClientes()
        {
            InitializeComponent();
        }


        1 referência
        private void frmClientes_Load(object sender, EventArgs e)
        {
            Listar();
        }

        5 referências
        public void Listar()
        {
            try
            {
                this.dgvDados.DataSource = cModel.Listar();
            }
            catch(Exception ex)
            {
                MessageBox.Show("Erro ao listar os dados - " + ex.Message);
            }
        }


        1 referência
        private void btnNovo_Click(object sender, EventArgs e)
        {
            habilitarCampos();
            limparCampos();
        }

        2 referências
        public void habilitarCampos()
        {
            txtNome.Enabled = true;
            cbxSexo.Enabled = true;
            dtpDataNascimento.Enabled = true;

        }
        3 referências
        public void desabilitarCampos()
        {
            txtNome.Enabled = false;
            cbxSexo.Enabled = false;
            dtpDataNascimento.Enabled = false;

        }

        4 referências
        public void limparCampos()
        {
            txtNome.Text = "";
            txtCodigo.Text = "";
            cbxSexo.Text = "";
        }

```

```
1 referência
public void Salvar(Cliente objeto)
{
    try
    {
        objeto.Nascimento = Convert.ToDateTime(dtpDataNascimento.Text);
        objeto.Nome = txtNome.Text;
        objeto.Sexo = cbxSexo.Text;

        cModel.Salvar(objeto);
        MessageBox.Show("Salvo com sucesso");




    }
    catch(Exception ex)
    {
        MessageBox.Show("Erro ao Salvar "+ex.Message);
    }
}

1 referência
private void btnSalvar_Click(object sender, EventArgs e)
{
    Cliente objeto = new Cliente();
    Salvar(objeto);
    Listar();
    limparCampos();
    desabilitarCampos();

}

1 referência
private void dgvDados_CellClick(object sender, DataGridViewCellEventArgs e)
{
    txtCodigo.Text = dgvDados.CurrentRow.Cells[0].Value.ToString();
    txtNome.Text = dgvDados.CurrentRow.Cells[1].Value.ToString();
    cbxSexo.Text = dgvDados.CurrentRow.Cells[2].Value.ToString();
    dtpDataNascimento.Text = dgvDados.CurrentRow.Cells[3].Value.ToString();
    habilitarCampos();
}
```

```
private void btnEditar_Click(object sender, EventArgs e)
{
    if (txtCodigo.Text == "")
    {
        MessageBox.Show("Selecione um registro para edição");
        return;
    }
    Cliente objeto = new Cliente();
    Editar(objeto);
    Listar();
    limparCampos();
    desabilitarCampos();

}

1 referência
public void Editar(Cliente objeto)
{
    try
    {
        objeto.id = Convert.ToInt32(txtCodigo.Text);
        objeto.Nascimento = Convert.ToDateTime(dtpDataNascimento.Text);
        objeto.Nome = txtNome.Text;
        objeto.Sexo = cbxSexo.Text;

        cModel.Editar(objeto);
        MessageBox.Show("Editado com sucesso");




    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro ao Editar " + ex.Message);
    }
}

1 referência
```

```
private void btnExcluir_Click(object sender, EventArgs e)
{
    if (txtCodigo.Text == "")
    {
        MessageBox.Show("Selecione um registro para excluir");
        return;
    }
    if (MessageBox.Show("Deseja excluir o registro selecionado?","Alerta", MessageBoxButtons.YesNo,
        MessageBoxIcon.Question,MessageBoxDefaultButton.Button2) == DialogResult.No)
    {
        return;
    }
    Cliente objeto = new Cliente();
    Excluir(objeto);
    Listar();
    limparCampos();
    desabilitarCampos();


}

1 referência
public void Excluir(Cliente objeto)
{
    try
    {
        objeto.id = Convert.ToInt32(txtCodigo.Text);

        cModel.Excluir(objeto);
        MessageBox.Show("Excluído com sucesso");




    }
    catch (Exception ex)
    {
        MessageBox.Show("Erro ao Editar " + ex.Message);
    }
}
```



```
187         private void txtPesquisa_TextChanged(object sender, EventArgs e)
188         {
189             Cliente objeto = new Cliente();
190             buscar(objeto);
191             //ListarPesquisa();
192             if (txtPesquisa.Text == "")
193             {
194                 Listar();
195                 return;
196             }
197         }
            1 referência
198         public void buscar(Cliente objeto)
199         {
200             try
201             {
202                 objeto.Nome = txtPesquisa.Text;
203                 this.dgvDados.DataSource = cModel.buscar(objeto);
204             }
205             catch(Exception ex)
206             {
207                 MessageBox.Show("Erro ao listar os dados - " + ex.Message);
208             }
209         }
210
211     }
212 }
```

Esse trecho de código importa as pastas Model e ENTIDADES (linhas 10 e 11). Cria a instância de ClienteModel (linha 15). O método Load do formulário, na linha 27, chama o método Listar() para carrregar os dados na grid.

O método Listar(), linhas 30 a 40, obtem os dados do método Listar() de cModel e os armazena no DataSource da grid (linha 34).

O click do botão novo possui o seu método nas linhas 42 a 46, que chama os métodos HabilitarCampos() e LimparCampos().

O método HabilitarCampos() (linhas 48 a 54), ativa com true na propriedade Enabled de todas as entradas do formulário.

O método LimparCampos() (linhas 63 a 68) atribui string vazia aos principais elementos de entrada do formulário.

O método DesabilitarCampos() (linhas 55 a 61) atribui false na propriedade Enabled das principais entradas de dados do formulário.

As linhas 71 a 90 delimita a ação do método Salvar(Cliente objeto), que atribui os valores informados nas entradas do formulário ao respectivo atributo da instância **objeto** de Cliente. Na linha 75 converte string para Data (que é o tipo do atributo nascimento). Nas linhas 76 e 77 há atribuição conteúdo para os dois atributos do tipo string do **objeto**. Em seguida, faz a chamada ao método Salvar do objeto cModel (ClienteModel), na linha 79. Em seguida, se não ocorrer nenhum erro, apresenta, na linha 80, mensagem de sucesso da inclusão dos dados.

As linhas 92 a 100 estabelecem o método do evento click do botão salvar, criando o objeto da classe cliente (linha 94). Efetua chamada ao método Salvar(objeto) do formulário (linha 95). Após salvar necessita atualizar a grid e isso é realizado com a chamada ao método Listar() (linha 96). Em seguida, limpa os campos e os desabilita (linhas 97 e 98).

De 102 a 109 tem-se as linhas do método correspondente ao click sobre uma linha da grid. Assim, quando uma linha receber um click, os campos do formulário devem ser preenchidos com os dados da grid ou com dados que devem ser acessados no banco de dados conforme alguma coluna da grid. Neste caso, obtem-se os conteúdos de cada coluna da grid. Cada conteúdo é acessando informando-se o nome do componente grid, seguido por CurrentRow (linha atual), seguido por Cells[indice].Value.ToString(). Como exemplo, tem-se: txtCodigo.Text=dgvDados.CurrentRow.Cells[0].Value.ToString();. Cada coluna da grid possui um índice a partir do primeiro, que é zero (0). Na ordem em que são visualizados na grid.

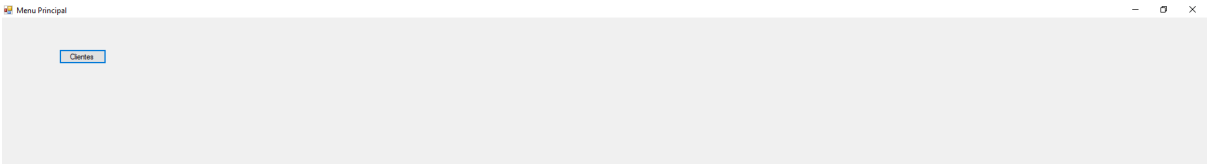
As linhas 111 a 124 destacam a delimitação do método correspondente ao click sobre o botão Editar. Neste caso, verifica se a caixa de texto do código é vazia ou não. Se for vazia não há condições de efetuar edição e indica que nenhum registro foi selecionado na grid com a mensagem da linha 115. Caso contrário chama o método Editar dessa classe enviando o objeto Cliente(linha 119). Após atualizar os dados lista os campos na grid novamente (linha 120). Em seguida, limpa os campos de entrada e desabilita-os (linhas 121 e 122).

O método Editar(Cliente objeto), linhas 126 a 146, faz com que os atributos do objeto Cliente sejam atualizados conforme os conteúdos informados nas entradas do formulário (linhas 130 a 133). Em seguida, chama o método Editar do objeto cModel (linha 135) e emite mensagem informando sobre o sucesso da edição (linha 136).

O método do click do botão Excluir está delimitado pelas linhas 148 a 166. Aqui também se verifica o conteúdo de txtCodigo. Se for vazio, inibe alguma exclusão, emitindo mensagem (linhas 150 a 154). Caso contrário, as linhas 155 a 159 estabelecem a verificação da seleção do botão No pelo usuário quando uma janela de diálogo é apresentada (linha 155). Caso esse botão for pressionado, o método é cancelado com return. Se o usuário selecionou outro botão, confirmando a exclusão, então, efetua a chamada ao método Excluir(objeto) dessa classe (linha 161). Atualiza a grid de dados chamando o método Listar() (linha 162). Em seguida, limpa e desabilita os campos de entrada do formulário (linhas 163 e 164).

O método Excluir(Cliente objeto), linhas 168 a 185, que é chamado pelo método explicado anteriormente, faz com que o atributo id do objeto receba o conteúdo do id da caixa de entrada txtCodigo (linha 172), convertido de string para inteiro. Em seguida, chama o método Excluir(objeto) da classe ClienteModel (linha 174). Em seguida, apresenta mensagem informando sobre o sucesso dessa ação.

Salve o projeto e execute-o. A janela principal é aberta.

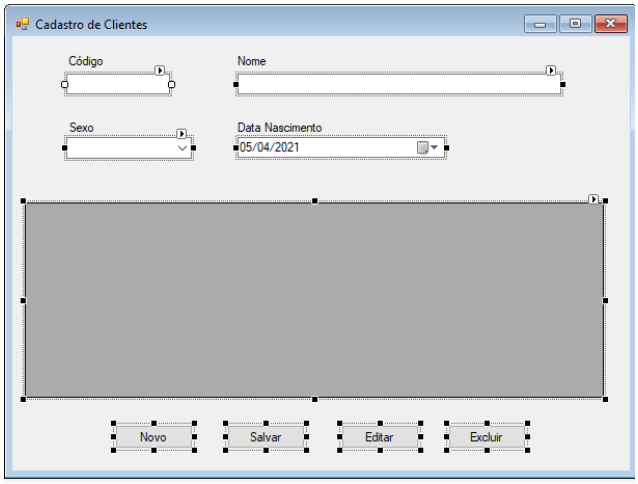


Clique no botão Clientes para abrir a janela frmClientes:

Observe que a janela de cadastro de clientes foi aberta com a grade (ou grid) de dados populada com as informações do banco de dados.

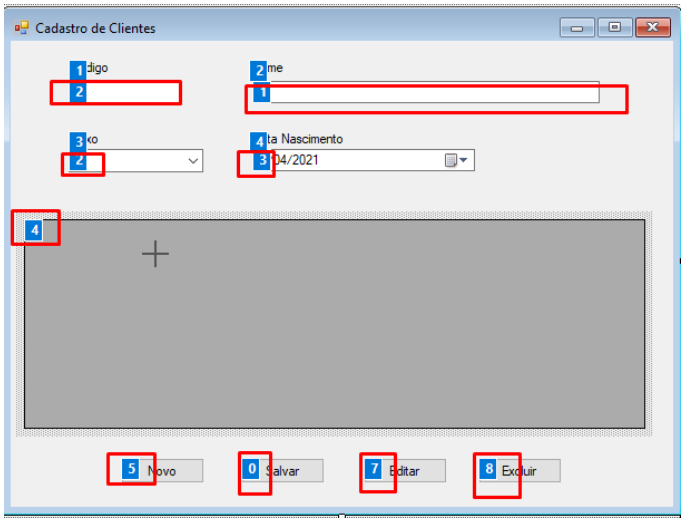
Pare a execução do projeto.

Abra a janela frmClientes. Selecione todo os componentes da janela pressionando a tecla CTRL:

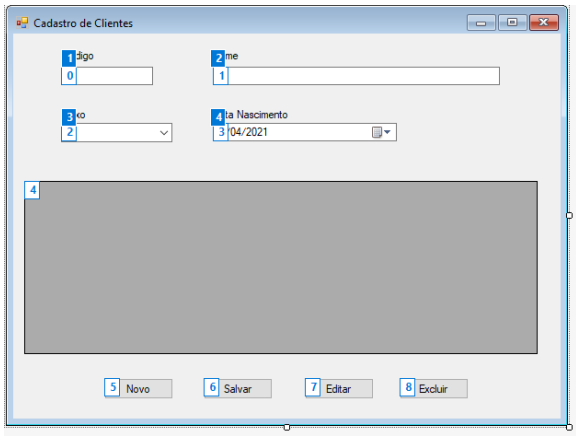


Acesse a opção View / Ordem de tabulação.

Irão aparecer retângulos em azul, como ilustrado:



Despreze os componentes Label e vá clicando em cada um dos seguintes retângulos, na sequência: código, nome, combobox sexo, data de nascimento, grid view, botão novo, botão salvar, botão editar e botão excluir. Observe como fica a sequência após as devidas seleções:



A sequência estabelelcida na figura anterior informa que, em tempo de execução, com o uso da tecla tab, o foco do cursor se movimentará para o controle seguinte, conforme a ordem indicada. Salve os arquivos.

Quando esse formulário for aberto os campos de edição devem estar desabilitados. Assim, selecione os quatro primeiros controles com CTRL e altere a propriedade Enabled para false.

Com esse formulário aberto, pressione F7 para abrir o seu respectivo código. Codifique os seguintes métodos no final da classe, como ilustrado:

```
public void habilitarCampos()
{
    txtNome.Enabled = true;
    cbxSexo.Enabled = true;
    dtpDataNascimento.Enabled = true;

}
0 referências
public void desabilitarCampos()
{
    txtNome.Enabled = false;
    cbxSexo.Enabled = false;
    dtpDataNascimento.Enabled = false;

}

1 referência
public void limparCampos()
{
    txtNome.Text = "";
    txtCodigo.Text = "";
    cbxSexo.Text = "";
}
```

O método habilitarCampos acessará cada componente de edição, exceto o correspondente ao código, e faz com que a propriedade Enabled de cada um seja true. O método desabilitarCampos faz o contrário, ou seja, desabilita a propriedade Enabled, deixando-a como false. O método limparCampos faz com que a propriedade Text de cada um das entradas de dados fiquem com o valor vazio.

Em seguida, crie o evento correspondente ao click sobre o botão Novo e codifique o seguinte:

```
private void btnNovo_Click(object sender, EventArgs e)
{
    habilitarCampos();
    limparCampos();
}
```

Assim, quando esse botão for clicado, os campos serão habilitados e seus conteúdos serão limpados. Salve o projeto e clique nesse botão. Em seguida, informe alguns conteúdos em cada um dos campos e pressione o botão novamente. Os campos serão limpos.

Selecione o componente DataGridView e ative o evento CellClick que será ativado quando uma célula (ou coluna) de uma determinada linha for clicada. Codifique o seguinte evento:

```
99      1 referencia
        private void dgvDados_CellClick(object sender, DataGridViewCellEventArgs e)
100     {
101         txtCodigo.Text = dgvDados.CurrentRow.Cells[0].Value.ToString();
102         txtNome.Text = dgvDados.CurrentRow.Cells[1].Value.ToString();
103         cbxSexo.Text = dgvDados.CurrentRow.Cells[2].Value.ToString();
104         dtpDataNascimento.Text = dgvDados.CurrentRow.Cells[3].Value.ToString();
105
106     }
```

Altere a propriedade WindowState do formulário frmPrincipal para Maximized para que possa ser aberto usando toda a janela do dispositivo.

Salve o projeto e execute-o.

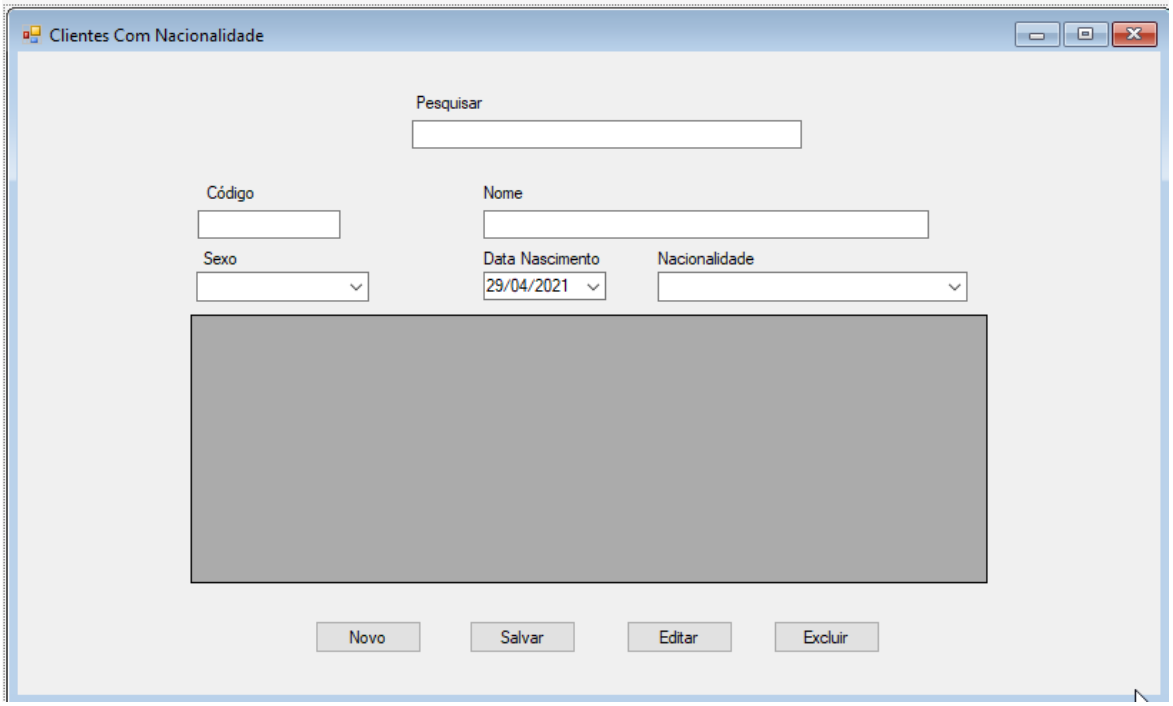
Em seguida, crie um novo formulário na pasta VIEW com o nome frmClientesCbx.

Abra o formulário frmClientes e selecione todos os seus componentes do seu Design. Dê um CTRL+C e copie esses componentes para frmClientesCbx.

Altere o título desse novo formulário para Clientes Com Nacionalidade.

Em seguida, copie todo o código de frmClientes para frmClientesCbx. Remova os erros existentes no código desse novo formulário.

Ajuste o design de frmClientesCbx e insira um componente ComboBox com uma label, como ilustrado:



Altere o name do combobox para cbxNacionalidade e altere a propriedade Text do seu Label para Nacionalidade.

Crie a classe Nacionalidade na pasta ENTIDADES, com a seguinte estrutura de código:

```
1   using System;
2   using System.Collections.Generic;
3   using System.Linq;
4   using System.Text;
5   using System.Threading.Tasks;
6
7   namespace SistemaCRUD.ENTIDADES
8   {
        0 referências
9       class Nacionalidade
10      {
11          private int id;
12          private String sigla;
13          private String pais;
14
            0 referências
15          public int Id { get => id; set => id = value; }
            0 referências
16          public string Sigla { get => sigla; set => sigla = value; }
            0 referências
17          public string Pais { get => pais; set => pais = value; }
18      }
19  }
```