DataBinding

Formação para formadores CET em .NET

Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

Miguel Carvalho (mcarvalho@cc.isel.ipl.pt)

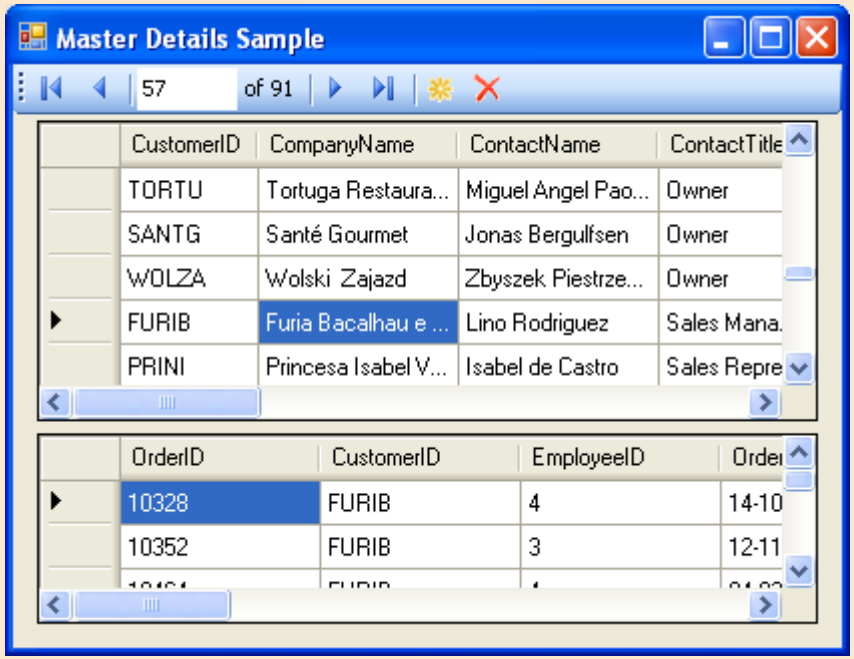
DataBinding

Typed Data Sets

Data Binding Enquadramento

O Data Binding encapsula todos os passos de apresentação e manipulação dos dados em componentes reutilizáveis.

Camada Aplicacional de Apresentação



Popular a UI com os dados

Trazer dados para a aplicação

Recolher as alterações dos dados

Guardar as alterações

Dados

Data Binding Enquadramento…

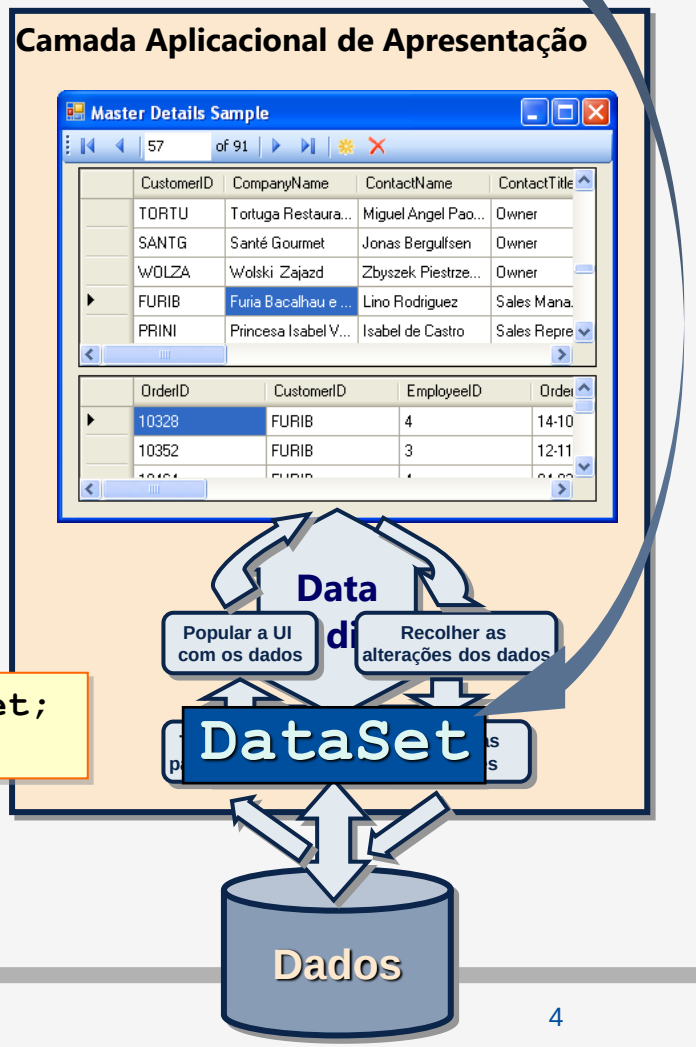
Para implementar *data binding* numa aplicação .Net é necessário um ***data source***:

- uma colecções de objectos;
- ou, uma única instância do próprio objecto.

Uma das formas mais comuns de implementar *data binding* é através da utilização de **`DataSet`**'s como *data source*.

`DataSet`:

- um contentor "complexo" para dados;




```
_ordersDataGridView.DataSource = _customersDataSet;
```

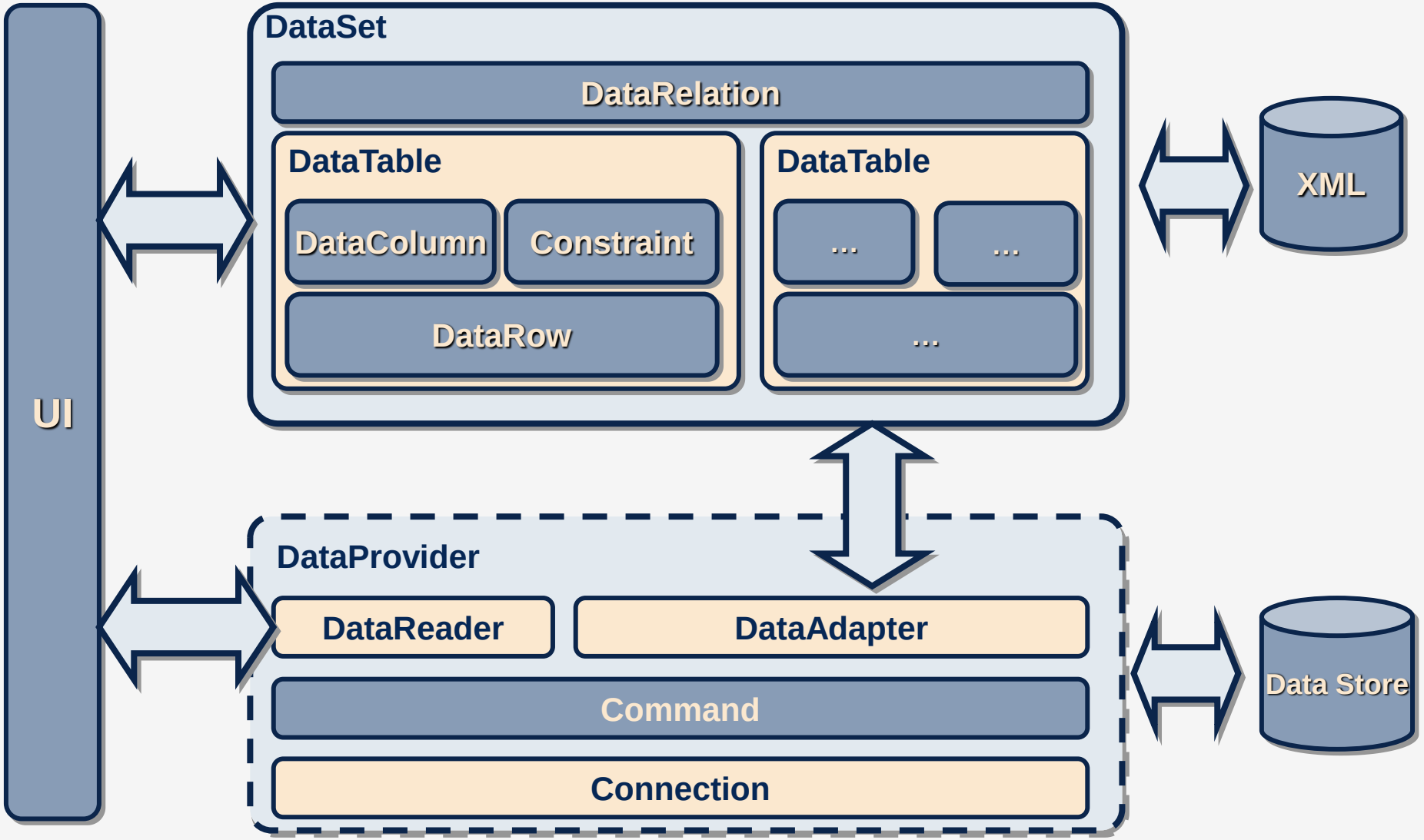
O conceito de *Binding* é dado à ligação entre um controlo e um *data source* (neste caso do tipo DataSet)

Demo 1

CustomersDataSet

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

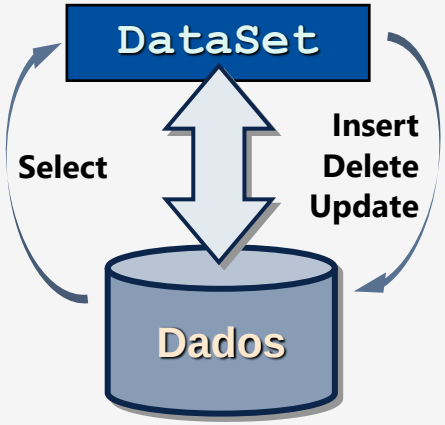
ADO.NET

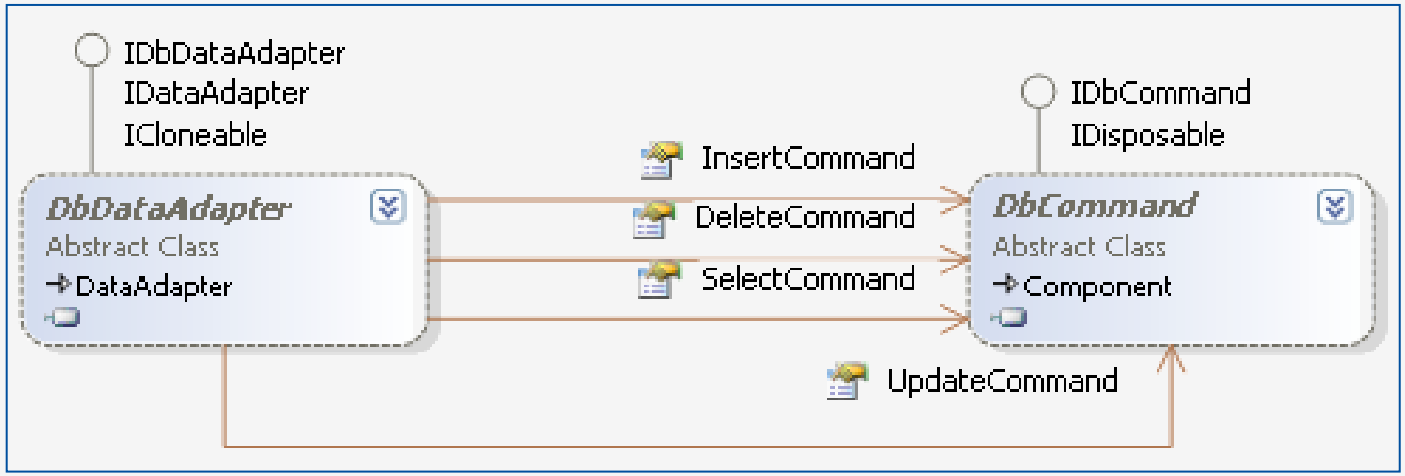

UI
DataSet
DataRelation
DataTable
DataColumn
Constraint
DataRow
DataTable
...
...
...
XML
DataProvider
DataReader
DataAdapter
Command
Connection
Data Store

DbDataAdpater

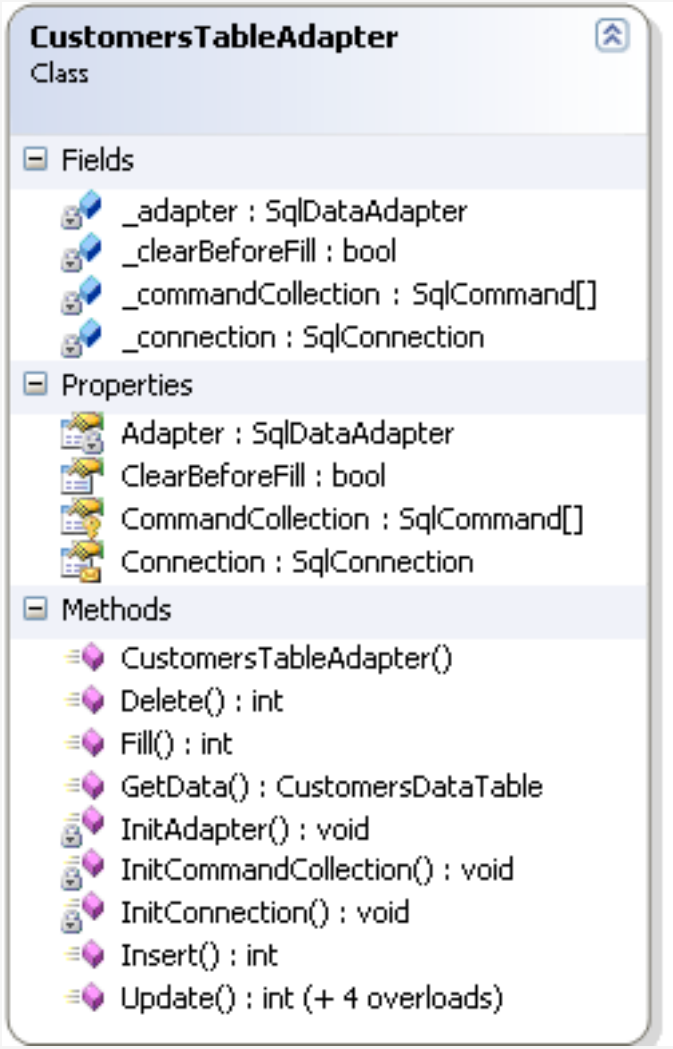
- Um `DataAdapter` serve para actualizar a informação entre um `DataSet` e a sua fonte de dados.

- **`DbDataAdapter`** representa um conjunto de comandos usados para preencher um `DataSet` e actualizar a base de dados





<datatablename>TableAdapter



"*strongly-typed wrapper* para um **`DbDataAdapter`**"

- Uma estratégia de uma instância de um subtipo de **`DbDataAdapter`** para cada **`DataTable`** existente no **`DataSet`**.
- As *queries* SELECT, INSERT, UPDATE e DELETE são geradas automaticamente e encapsuladas em objectos do tipo **`SqlCommand`** dentro do **`TableAdapter`**;
- São gerados um conjunto de métodos *strongly-typed* **`Fill`**, **`GetData`**, **`Update`**, **`Delete`** e **`Insert`**, que actuam directamente sobre o respectivo **`DataTable`**.

  (Um dos métodos **`Update`** recebe uma **`DataRow`** não tipificada, para facilitar a integração com o resultado do método **`Select`**).

<datatablename>TableAdapter

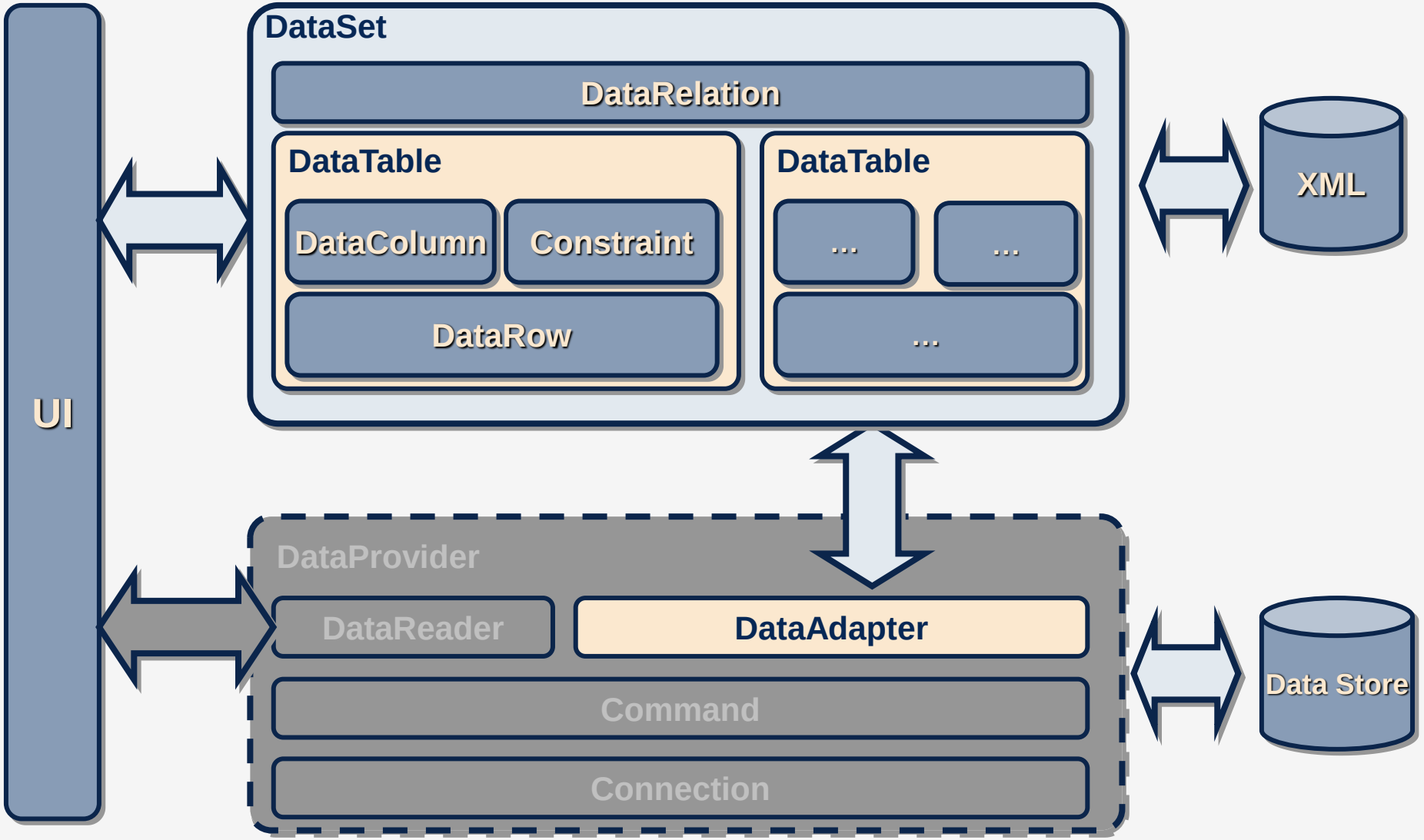
O *designer* do VS2005 gera automaticamente a chamada aos métodos `Fill` e `Update` do `<datatablename>TableAdapter` para actualização do `DataSet` e da base de dados respectivamente.

- **`Fill`**:
  usa a propriedade **`SelectCommand`** para preencher o **`DataSet`** recebido por argumento;
  Ex: **`_ordersTableAdapter.Fill(_customersDataSet.Orders);`**

- **`Update`**:
  actualiza a DB com os dados provenientes do **`DataSet`**.
  Ex: **`_ordersTableAdapter.Update(_customersDataSet.Orders);`**

  – Executa um comando em separado para cada linha;
  – Baseia-se na flag **`RowState`** (*unmodified*, *modified*, *added* ou *deleted*) de cada linha armazenada na **`DataTable`**;
  – ***Optimistic Concurrency***: a tabela mantém uma cópia do valor original de cada linha modificada. Assim é possível detectar se houve alguma alteração à DB desde que o **`DataSet`** foi preenchido até que seja reposto para a DB.
  ➔ Em caso de detecção de concorrência é lançada `DBConcurrencyException`.

ADO.NET


UI
DataSet
DataRelation
DataTable
DataColumn
Constraint
DataRow
DataTable
...
...
...
XML
DataProvider
DataReader
DataAdapter
Command
Connection
Data Store

strongly-typed **DataSet**

- *untyped* `DataSet` na versão 1.1

```
SqlDataAdpater adapter = SqlDataAdpater(query, connection);
DataSet data = DataSet();
adapter.Fill(data, "Customers");
```

Identificador do `DataTable`

- *strongly-typed* `DataSet` na versão 2
  Existe uma propriedade *strongly-typed* para cada um dos **`DataTable's`** contidos do **`DataSet`**.

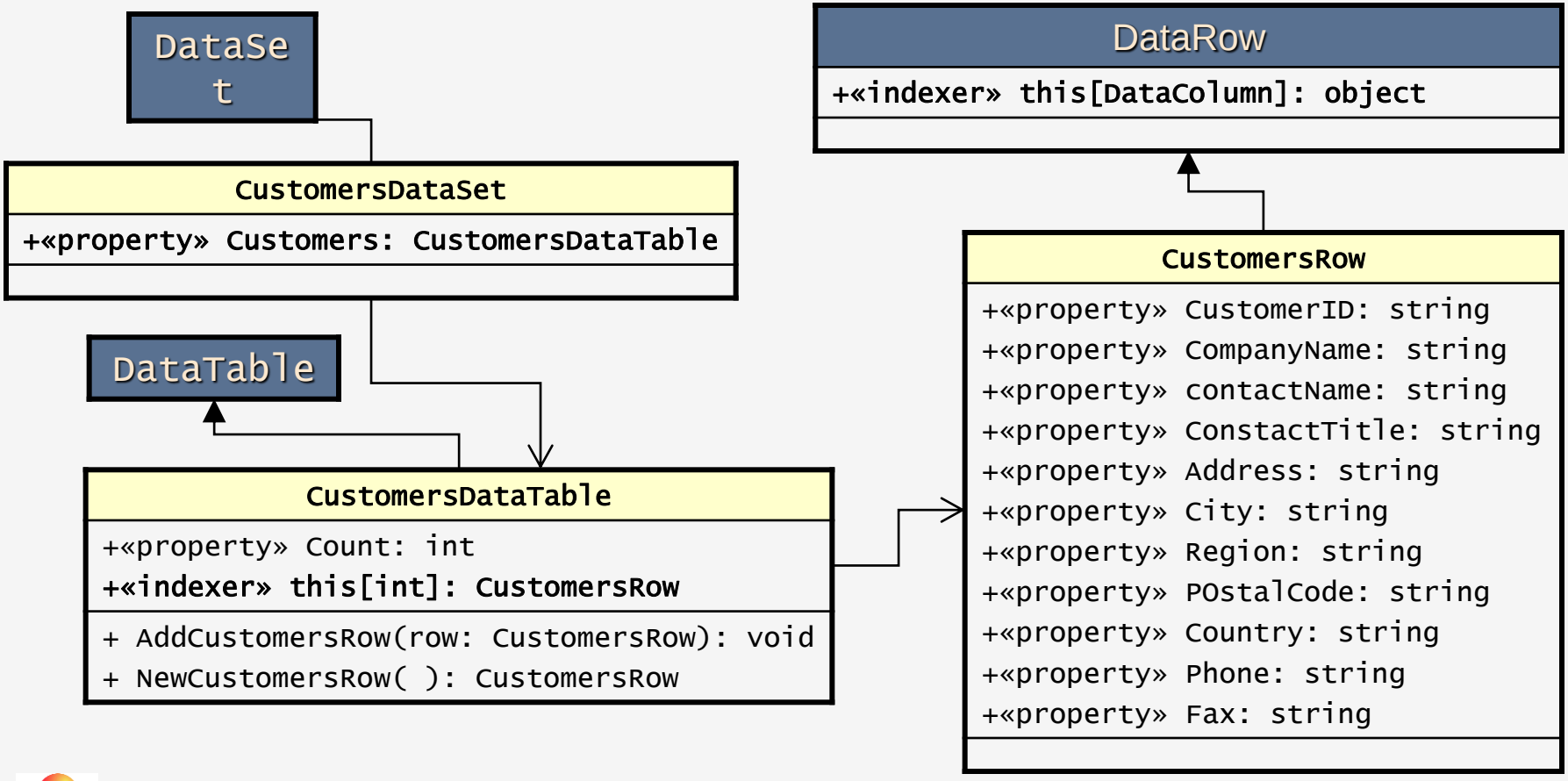
```
data = new CustomersDataSet();
adapter = new CustomersTableAdapter();
adapter.Fill(data.Customers);
```

Validação em tempo de compilação.
Não aceita outro tipo de `DataTable`, que não seja `CustomersDataTable`.

```
adapter.Update(data.Customers);
```

Typed DataSets

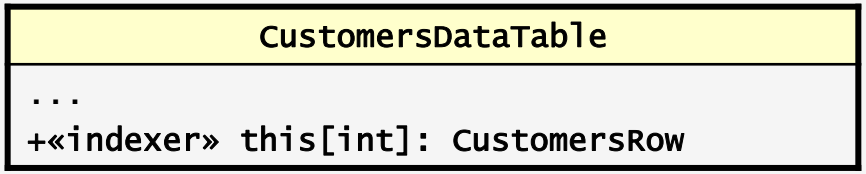
- Implementado com um conjunto de classes que derivam dos tipos base que compõem um **`DataSet`** (**`DataTable`**, **`DataRow`**, etc)



Typed DataTable

<typed>DataTable tem:

- propriedades <name>Column do tipo DataColumn para cada um das colunas na tabela;
- um *type-safe indexer* que permite percorrer as linhas de uma tabela como instâncias *type-safe* da classe <type>Row (estende DataRow):

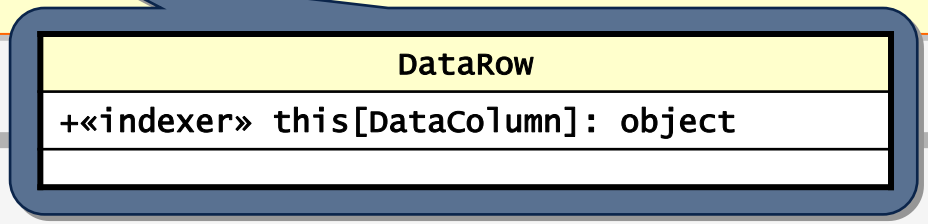


<typed>Row tem:

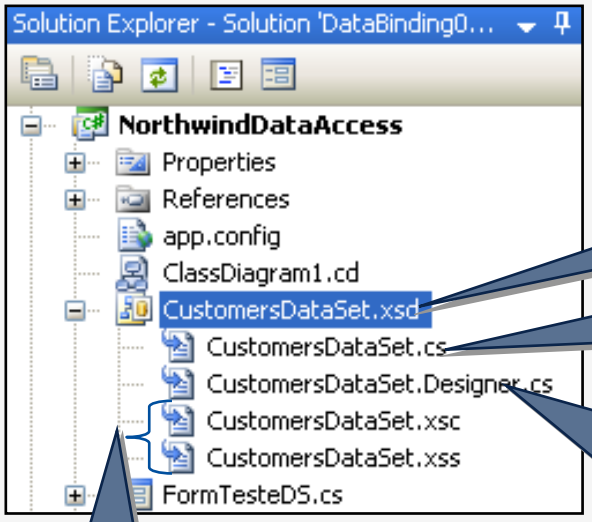
- propriedades tipificadas com valor de cada uma das colunas:

```
public string CustomerID {
  get {
    return ((string)(this[this.tableCustomers.CustomerIDColumn]));
 ...
```

Name != Source (DB column name)



Estrutura de Ficheiros




```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema id="CustomersDataSet"...>
...
```

```
namespace NorthwindDataAccess{
  partial class CustomersDataSet {
  }
}
```

```
namespace NorthwindDataAccess{
  partial class CustomersDataSet: DataSet {
    // Definição dos typed DataTables e DataRows
  }
}
namespace NorthwindDataAccess.CustomersDataSetTableAdapters{

  public partial class CustomersTableAdapter:Component {
    // Definição dos Table Adapters
  }
}
```

Especificação do layout do diagrama associado a `CustomersDataSet`. Gerado e mantido automaticamente pelo Designer do VS.

Demo

CustomersDataLayer

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DataView

- Uma instância de `DataView` não contem dados; consiste apenas numa vista para os dados armazenados num `DataTable`.
- Por omissão, uma instância de `DataTable` já tem um `DataView` acessível através da propriedade `DefaultView`.

```
_detailsView = _customersDataSet.Order_Details.DefaultView;
```

- Quando é feito o *binding* entre um controlo e um `DataTable`, na realidade o controlo liga-se ao seu `DefaultView` e não ao próprio `DataTable`.
  - Isto faz com que o *data item* seleccionado no controlo seja uma instância de `DataRowView` e não `DataRow`;

```
DataRowView rowView = (DataRowView) _ordersDataGridView.CurrentRow.DataBoundItem;
CustomersDataSet.OrdersRow row = (CustomersDataSet.OrdersRow) rowView.Row;
```

- Com uma instância de `DataView` é possível filtrar os dados apresentados no *bound control*, mantendo todos os dados nas tabelas do `DataSet`.

```
_detailsView.RowFilter = "OrderID = '" + row.OrderID + "'";
_detailsDataGridView.DataSource = _detailsView;
```

Demo 2

FilterDetails

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

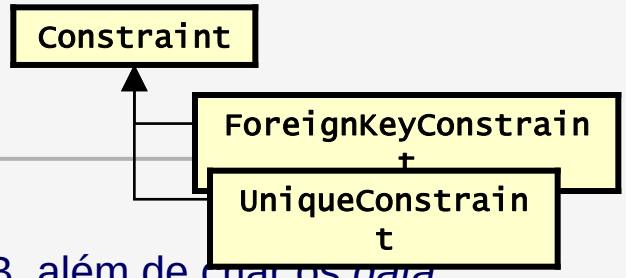
Typed `DataTable` ... Colunas Adicionais

- Colunas adicionais:

  Podem ser ainda adicionadas colunas a um *data table*, cujo valor seja o resultado computação do valor de outras colunas (através da propriedade **`Expression`**).

```
public partial class Order_DetailsDataTable : DataTable, IEnumerable {
  ...
  private void InitExpressions() {
    this.TotalColumn.Expression = "UnitPrice * Quantity * (1-Discount)";
  }
}
```

Typed DataSet ... Relations



- Quando o *designer* obtém a informação do *schema* da DB, além de criar os *data tables*, também cria:
  - *constraints* que serão adicionadas à definição de cada tabela. Ex:

```
this.Constraints.Add(
  new System.Data.UniqueConstraint(
          "Constraint1",
          new System.Data.DataColumn[]{this.columnCustID}, true));
```

  - relações entre tabelas. Ex:

```
this.relationFK_Order_Details_Orders = new DataRelation(
   "FK_Order_Details_Orders",
   new DataColumn[] {this.tableOrders.OrderIDColumn},
   new DataColumn[] {this.tableOrder_Details.OrderIDColumn},
   false);
```

- Obter linhas com base em relações:

```
DataRelation rel = _customersDataSet.Relations["FK_Order_Details_Orders"];
Order_DetailsRow[] rows = (Order_DetailsRow [])row.GetChildRows(rel);
double soma = 0;
foreach (Order_DetailsRow r in rows) {
  soma += r.Total;
}
```

demo

CENTRO DE CÁLCULO

Typed DataTable...

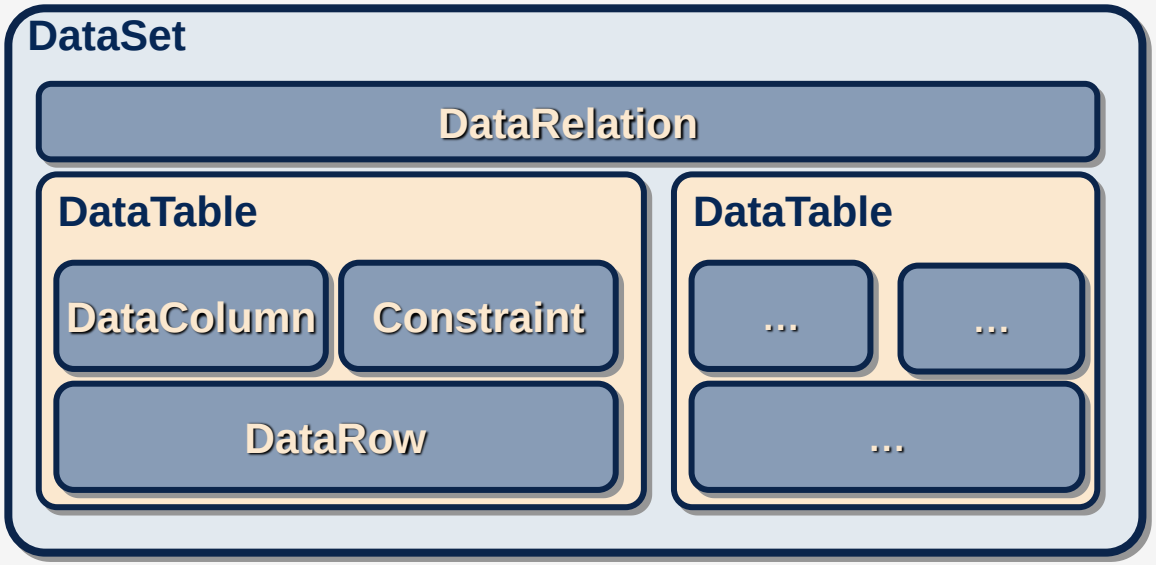
- Eventos fortemente tipificados que notificam a ocorrência de alterações, inserções ou remoções de linhas na tabela. Estes eventos aparecem aos pares:
  - **<type>RowChanging <type>RowChanged**
  - **<type>RowDeleting <type>RowDeleted**

  Eventos lançados antes e após a actualização ser commited, respectivamente.

- Métodos para criação e inserção de linhas:
  - **New<type>Row()**: cria uma nova instância do tipo **<type>Row**;
  - **Add<type>Row(…)**: adiciona uma nova linha (instância **<type>Row**).

- Método para obter um conjunto de linhas:
  - **DataRow[] Select (string *filterExpression*)**

Vantagens dos **Typed** `DataSets`



Vantagens na utilização de `DataSets` fortemente tipificados:

- Validação de tipos em tempo de compilação;
- Facilita a identificação de erros de cada vez que existe uma actualização do *schema* e consequente actualização da definição do `DataSet`;
- Permite que os nomes das propriedades da `<typed>DataRow` sejam distintos dos respectivos nomes na base de dados.

DataSet's <> XML

Os DataSet's disponibilizam dois métodos para leitura e escrita de dados em XML.

- ReadXml: por omissão e caso o XML não inclua um *schema*, este método pode inferir o *schema* apropriado para esses dados.
  - Opção 1:
    - O elemento *root* será tratado como o contentor para o DataSet;
    - Cada elemento que contenha outros elementos, será uma linha de uma determinada tabela;
    - Os elementos dessa linha que contenham apenas texto, serão as suas colunas.
  - Opção 2:
    - Igual à opção 1, mas as colunas poderão ser definidas como atributos do elemento que representa a linha.

- Excepções:
  - **ArgumentException**: tentativa de carregamento de XML cujo schema não representa um modelo relacional válido.

    Ex: se o mesmo tipo de elemento aparecer em diferentes níveis da hierarquia.
  - **XmlException**: se o XML estiver mal definido.

Demo 3

DataSetFromXml

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

DataBinding

Arquitectura

Designações

- ***bound control*** – controlo ligado a um *data source*.
- ***data source*** – uma única instância, uma colecção de instâncias ou ainda um contentor de colecções (Ex: `DataSet`);

Instância

- ***data member*** – caminho relativo para um "pedaço" de informação (*data item*).

  Exemplos:
  - Propriedade `CompanyName` de uma instância da classe `Customer`;
  - Coluna de um `DataTable` : `Customers.CompanyName`

Informação de Tipo

- ***data item*** – **elemento** de um colecção usada como fonte de dados de *databinding,* num determinado momento.

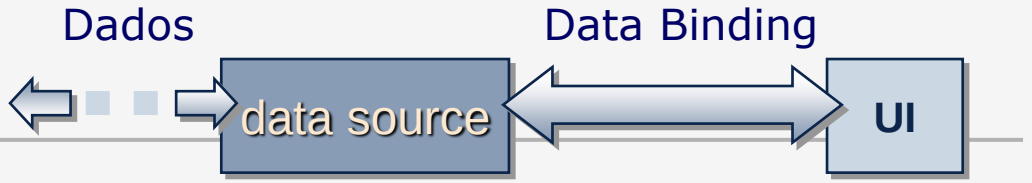
  Exemplos:
  - Instancia de `Customer` numa colecção de `BindingList<Customer>`;
  - Uma `DataRow` num `DataTable`.

Instância

- ***field*** – parte dos dados apresentada no controlo.

  Exemplos:
  - Propriedade pública de um objecto;
  - Uma coluna de um `DataRow`.

Conceitos…



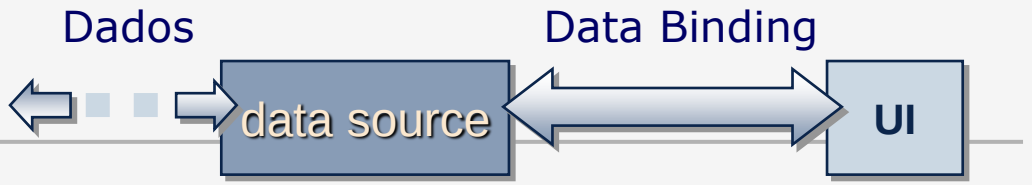
DataBinding é um mecanismo de ligação de dados a um **elemento da interface do utilizador** (***Databound Control***).

- **Direcção**:
  - Fluxo dos dados entre o d*ata source* e *databound controls*:

    one-way ou ***two-way.***

- **Ocorrência**:
  - Em que momento é que os dados flúem do *data source* para o *databound control*:
    - No estabelecimento da ligação *databinding*;
    - Chamada de um método de *refresh* no controlo, no *datasource* ou no mecanismo de *databinding*;
    - Chamada de um *handler* de um evento por alteração de dados no *datasource* ou no controlo.

Conceitos…



- Dois tipos de *DataBinding*:
  - **Simples**: ligação de uma propriedade de um controlo (ex: **`TextBox`**, **`Label`**, etc) a uma propriedade de um *data item*.

```
Binding bind = new Binding(
  "Text", _customersDataSet.Customers, "CompanyName", true);
_txtCompanyName.DataBindings.Add(bind);
```

  - **Complexo (*list-based binding*)**: ligação de um controlo (ex: **`ComboBox`**, **`ListBox`**, **`DataGrid`**, **`etc`**) a uma lista de dados (tipo **`IList`**)

```
_customersDataGridView.DataSource = _customersDataSet.Customers;
```

IBindableComponent... Binding

Suporte para *data binding* simples:

- A classe `Control` (da qual derivam todos os controlos *Windows Forms*) implementa `IBindableComponent`.

IBindableComponent
+«property» DataBindings: ControlBindingsCollection { get;} +«property» BindingContext: BindingContext { get; set;}

- `DataBindings` - colecção de objectos do tipo `Binding`.

  Ex: `_txtCustomerId.DataBindings.Add(bind);`

- `Binding` – associação entre um *data member* e a propriedade de um controlo.

  Ex:

```
Binding bind = new Binding(
  "Text",
  _customersDataSet.Customers,
  "CustomerID",
  true);
```

Propriedade no controlo a ligar

data source

data member

Formata automaticamente os valores apresentados

Demo 6

SyncSimpleDataBinding

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

IBindableComponent...BindingManagerBase

- `BindingContext` - gere uma colecção de objectos do tipo `BindingManagerBase`, acessíveis através de um *indexer* cuja chave é a referência para o *data source* correspondente.

  Ex:
```
BindingManagerBase mgr =
        _txtCustomerId.BindingContext[_customersDataSet.Customers];
```

BindingContext
+«indexer» this[dataSource: Object]: BindingManagerBase

- `BindingManagerBase` – mantém uma referência para o *data item* corrente, no *data source*.

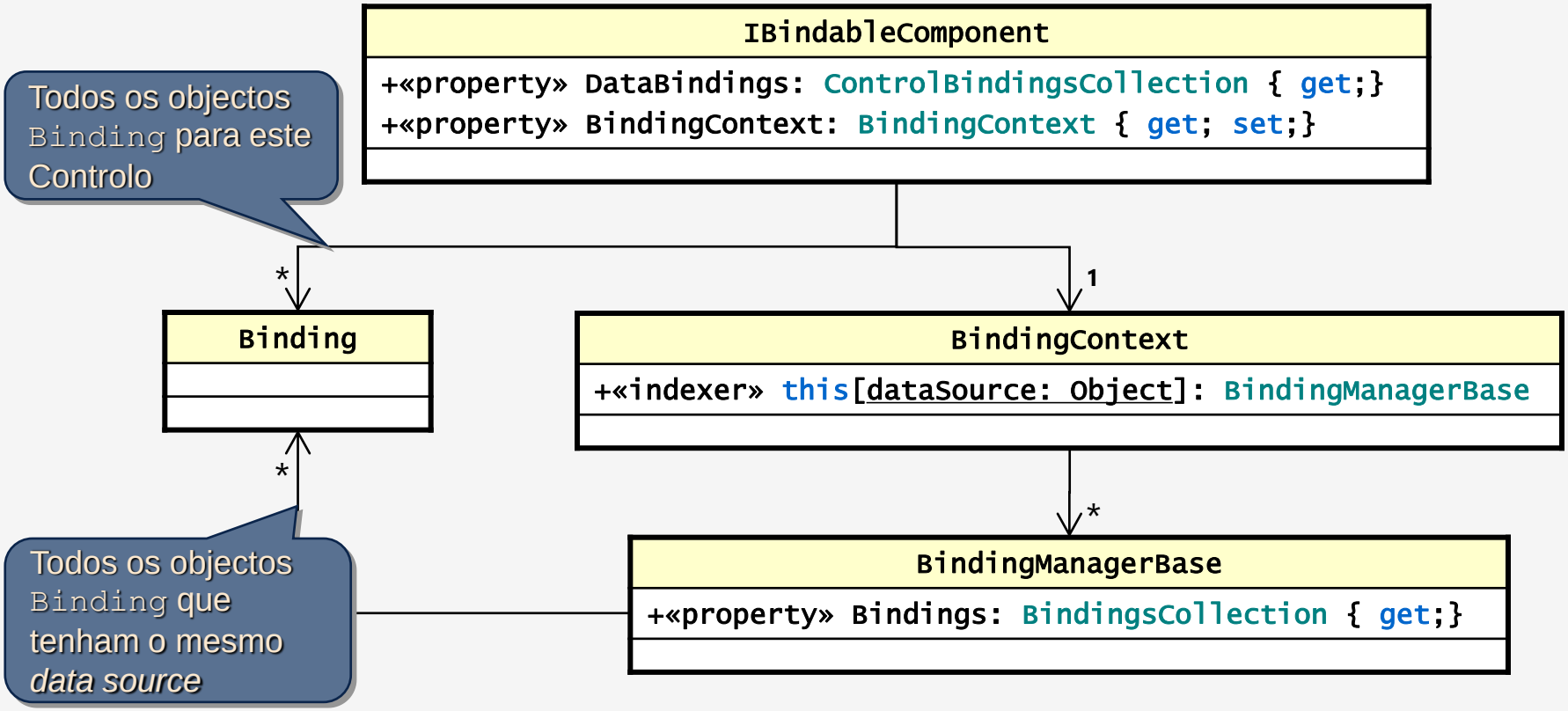
Classe abstracta, cuja classe concreta mais conhecida é o `CurrencyManager`

BindingManagerBase
... +«property» Count: int { get;} +«property» Current: Object { get;} +«property» Position: int { get; set; }

demo

IBindableComponent...BindingManagerBase

- O objecto `BindingContext` é partilhado por todos os controlos e pelo próprio `Form` que contém esses controlos.
- Cada instância de `BindingManagerBase` tem uma colecção com todos os objectos `Binding` que tenham o mesmo *data source*.



Sincronização entre controlos

O *data member* representa o caminho relativo para um objecto (*data item*) dentro do *data source*.

- Outra forma de definir um objecto `Binding`:

```
Binding bind = new Binding(
    "Text", _customersDataSet.Customers, "Customers.PostalCode", true);
```

demo

ATENÇÃO:

1. A chave para o `BindingManagerBase` passa a ser o *data source* + *data member*:

```
_mgr = _txtCustomerId.BindingContext[_customersDataSet.Customers, "Customers"];
```

2. Para que dois controlos apresentem dados da mesma fonte sincronamente, deverão ter o seu *datasource* definido de forma coerente:

```
_txtCompanyName.DataBindings.Add(
  new Binding("Text", _customersDataSet.Customers, "CompanyName", true));
_txtPostalCode.DataBindings.Add(
  new Binding("Text", _customersDataSet, "Customers.PostalCode", true));
```

data source incoerente com `_tableCustomers`

`CurrencyManager` e `PropertyManager`

- A classe abstracta `BindingManagerBase` tem dois subtipos concretos: `CurrencyManager` e `PropertyManager`.
- Quando é estabelecido o *data binding* entre um controlo e um *data source* é sempre associado um `BindingManagerBase` uma vez que o *data dource* não tem a noção do ***data item* corrente**.

- O `BindingManagerBase` será uma instância de `CurrencyManager` ou `PropertyManager`, consoante o *data source* seja uma colecção de objectos (Ex: `DataTable` ou `IList`) ou uma única instância, respectivamente.

- O `PropertyManager` apenas mantém a propriedade corrente de um determinado objecto. Nesta implementação as propriedades `Position` e `Count` não têm qualquer efeito.

Data Binding Complexo

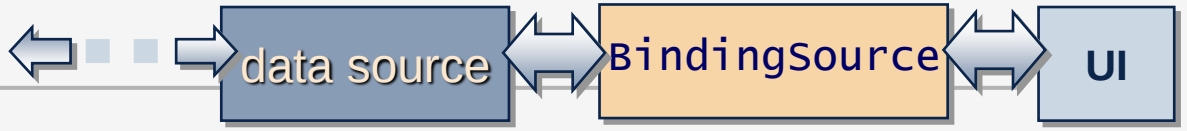
- Além das propriedades da interface `IBindableComponent`, os controlos que suportam *databinding* complexo expõem as seguintes propriedades:
  - `Datasource`
  - `DisplayMember`
  - `ValueMember` (valor acessível na propriedade `SelectedValue` em `ComboBox`)

```
_cbCompanyName.DisplayMember = "CompanyName";
_cbCompanyName.ValueMember = "CustomerId";
_cbCompanyName.DataSource = _customersDataSet.Customers;
```

DataBinding

BindingSource

BindingSource



- Actua como um *proxy* entre o *data source* e vários controlos.
- Cria um nível de abstracção entre os dados e a *user interface*, permitindo que o *data source* possa ser alterado (mesmo em *runtime)* para os vários controlos que estão ligados.
- Ponto único de sincronização entre o *datasource* e vários controlos. Garante que o mesmo *data item* seja ligado sincronamente a diferentes controlos.
- **Navegação** num `BindingSource`: `Position`, `Contains`, `IndexOf`, `MoveFirst`, `MoveLast`, `MoveNext` e `MovePrevious`.
- Propriedade **`Current`**: *data item* corrente.

  (Atenção: O *data item* de um `DataSource` do tipo `DataSet` é um `DataRowView` porque na realidade o *binding* é feito ao seu `DefaultDataView`).
- Facilita ***Master-Details*** *data binding*: ligação de `BindingSource`'s em cadeia.

Demo 7

MasterDetailsBindingSource

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Monitorar os Dados com Eventos

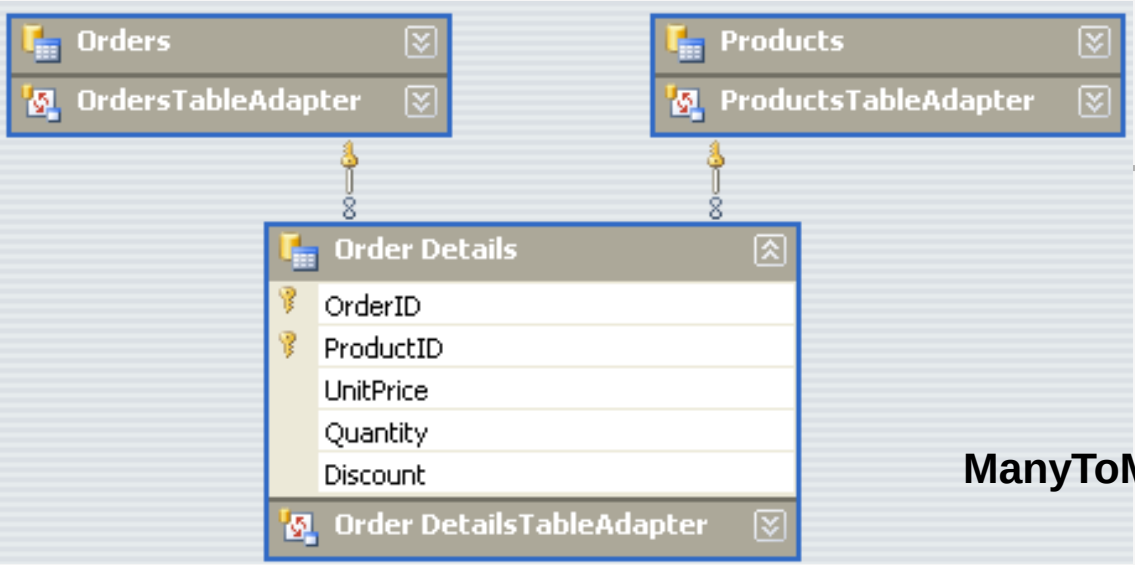
- `AddingNew`: resultante da invocação do método `AddNew()`.
- `BindingComplete`: O *bound control* acabou de ler o data item de `Current`.
- **`CurrentChanged`**: Alteração na lista por inserção ou remoção de *itens*, que provoca que o `Current` se refira a um novo item.
- `CurrentItemChanged`: Alteração num dos valores do item `Current`.

  Este evento pode ser lançado pelo evento `ListChanged` de `IBindingList`.
- `DataError`: Uma excepção com origem no `CurrencyManager` não apanhada pelo `BindingSource`.
- `DataMemberChanged`, `DataSourceChanged`.
- **`ListChanged`** (activado ou desactivado pela propriedade **`RaiseListChangedEvents`**):
  - Alterações à lista interna: adição, remoção ou edição de data itens;
  - Alterações nas propriedades da lista interna (Ex: `AllowEdit`);
  - Se a lista interna for substituída por uma nova lista de dados;
  - Pelo lançamento do evento `ListChanged` de `IBindingList`.

    (este evento também é lançado mesmo quando a manipulação de *data itens* é feita através do próprio `BindingSource`)
- `PositionChanged`: resultante da alteração da propriedade `Position`.

demo

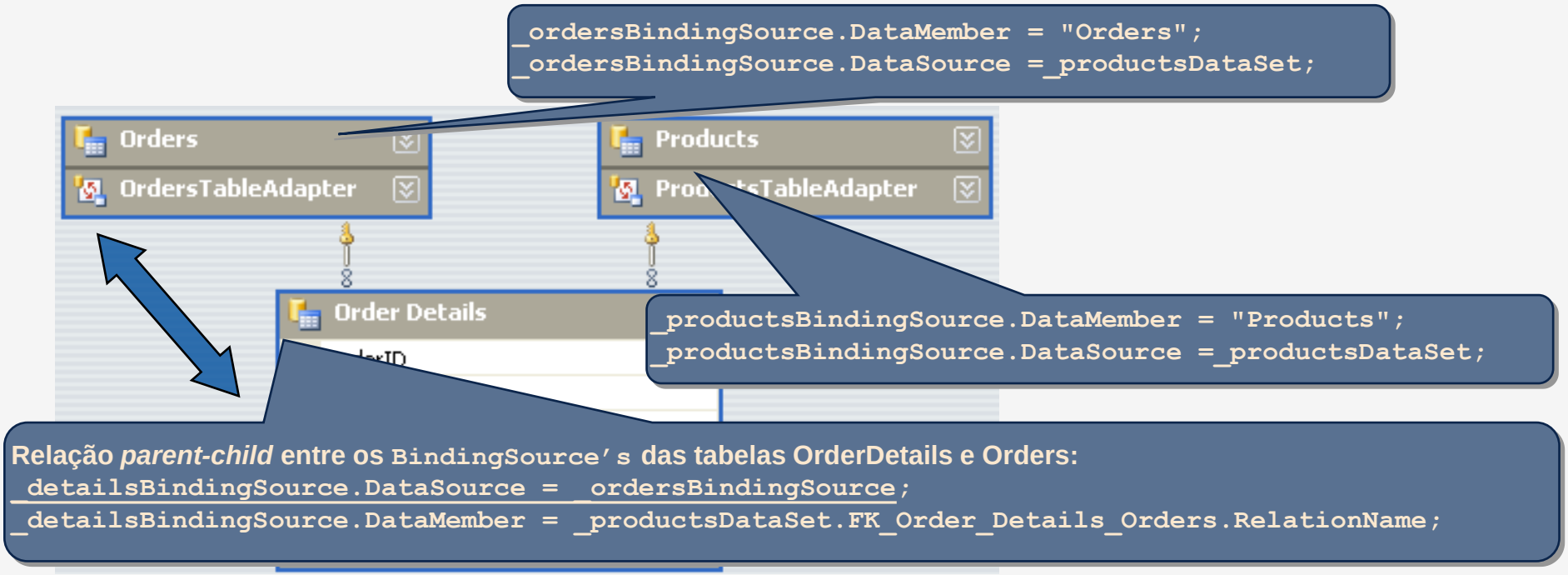
Demo 8



ManyToMany

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Sincronizar relações *Many-To-Many*



- A propriedade `Filter` do `BindingSource` depende da propriedade `Filter` do seu *data source* interno (exemplo `Filter` de `DataView`).

BindingSource

"The BindingSource component that encapsulates a currency manager and a data source."

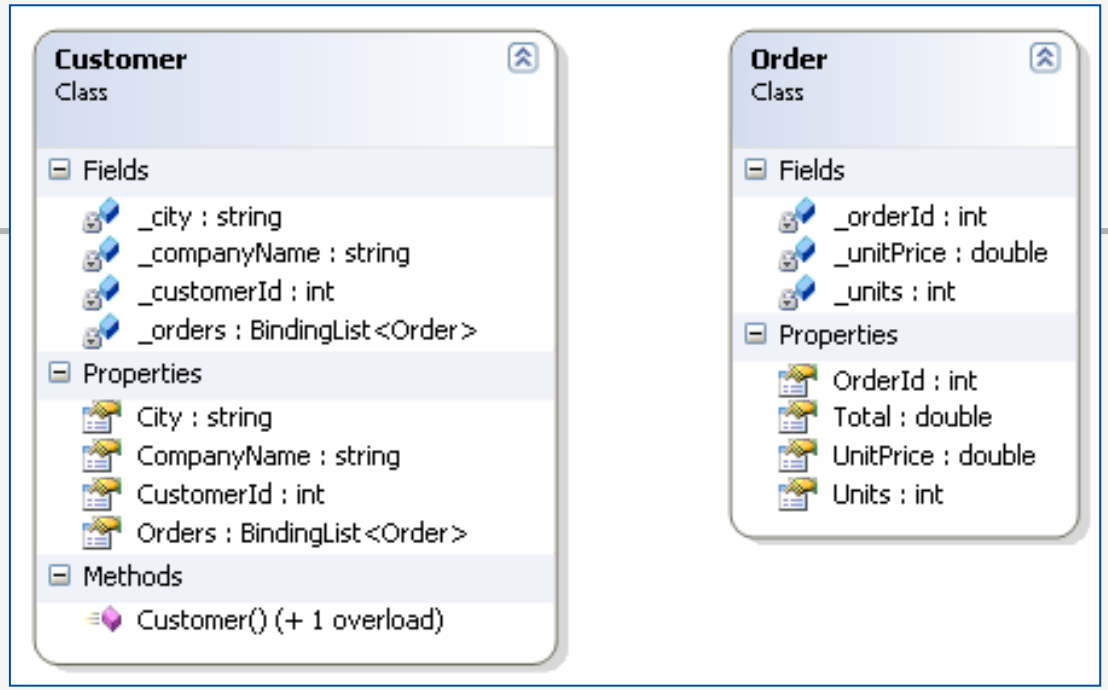
Expõe:

- Um conjunto de métodos e propriedades para manipulação de uma lista de dados interna;
- Um conjunto de eventos que permitem seguir as alterações sobre o *data source*, para actualização de controlos interligados.

demo

Demo 9

BusinessObjects



CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

`BindingSource` …Manipulação de d*ata Itens*

- Se um `BindingSource` não tem um *data source* definido o primeiro *data item* inserido determinará o tipo de elementos armazenados.
  - Tentativas de inserção de um tipo incompatível resulta em **`InvalidOperationException`**.

- `BindingSource` como contentor de dados: `Add` e `AddNew`.
  - Evento `AddingNew` lançado pela chamada ao método `AddNew;`
  - No argumento passado ao *handler* deste evento (do tipo `AddingNewEventArgs`) pode ser definido o novo objecto adicionado:

    Ex: `e.newObject = new MyCustomObject);`

- Propriedade `AutoGenerateColumns` do `DataGridView`:
  - Deve estar a `true` sempre que não forem definidas previamente as colunas;
  - Não se podem adicionar dados a um `BindingSource` de uma `DataGridView` que não tenha colunas definidas.

- Se a classe do *data item* não tiver um construtor sem parâmetros, o `BindingSource` impede o adicionar de novos elementos, através da propriedade:

```
bool AllowNew { get; set; }
```

- **Como são armazenados os *data itens*?**

IList

- Como são armazenados os *data itens*?
  - Numa lista interna referida pela propriedade `List`: `public IList List { get;}`
  - Não é possível alterar a propriedade `List` porque a lista é obtida a partir do *data source*.

- Num `BindingSource` com *data source* do tipo `DataSet`, a sua propriedade `List` refere a `DataView` dada pela propriedade `DefaultView` desse `DataSet`.
  - `DataView` implementa `IList`, logo cumpre o requisito mínimo para ser um *data source* com suporte a *data binding* complexo.

- Um *data source* pode ser uma colecção de objectos do tipo **`IList`**:
  - interface mínima requerida para suporte a *data biding* complexo;

- Ligação entre `BindingSources` para listas de objectos.

demo

`IList` *data source*

Um *data source* pode ser iniciado explicitamente com uma instância do tipo `IList`:

- Alterações aos dados do *bound control* são reflectidas sobre o *data source*.
  - Outros controlos ligados ao mesmo *data source* de forma coerente (Ex: através do mesmo `BindingSource`) serão actualizados com essas alterações. Isso acontece porque o *bound control*, notifica o `CurrencyManager`, que por sua vez responderá a todos os outros controlos que estejam em comunicação com o mesmo `CurrencyManager`.

- Actualizações por intermédio do `BindingSource`, são reflectidas nos *bound control*'s e no *data source*.
  - Pela mesma razão anterior, todos os controlos que partilhem o mesmo `CurrencyManager`, também serão actualizados com as mesmas alterações.

- Actualizações sobre o *data source* poderão não ser reflectidas nos *bound controls*?

Demo 11

IListDataSource

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

`List<T>` *data source*

`List<T>`:

- **Não** notifica sobre inserções, remoções ou actualizações sobre o *data source*;
- **Não** permite inserções de elementos de modo transaccional através da implementação da interface `ICancelNew`;
- **Não** tem suporte para ordenação dos elementos;
  - `BindingSource.SupportsSorting`, retorna `false` para um *data source* do tipo List<T>
- **Não** tem suporte para pesquisa de elementos;
  - `BindingSource.SupportsSearching`, retorna `false` para um *data source* do tipo List<T>

DataBinding

BindingList<T>

BindingList<T>

- `BindingList<T>` é o *data source* usado por omissão pelo `BindingSource` quando não é definido explicitamente o tipo de colecção para armazenamento dos *data itens*.

- `BindingList<T>` é o tipo de colecção indicado para suporte a *data binding* complexo com dois sentidos de actualização (*data source* ⇔ UI).

- Disponibiliza a infra-estrutura, mas não implementa a mesma funcionalidade que é oferecida num *data source* do tipo `DataView`, nomeadamente:
  - Ordenação;
  - Filtragem;
  - Pesquisa;
  - Edição transaccional de elementos;
  - Notificação de alteração das propriedades de um data item.

- `BindingList<T>` pode ser estendido para suporte ou alteração de comportamento das suas funcionalidades.

BindingList<T>

```
public class BindingList<T> : Collection<T>, IBindingList, IList,
    ICollection, IEnumerable, ICancelAddNew, IRaiseItemChangedEvents
```

- **ICancelAddNew**:
  - Disponibiliza o mecanismo de adição de elementos de modo transaccional. Isto é, permite remover um novo *data item* inserido pela operação `AddNew`, antes da operação ser finalizada.

- **IRaiseItemChangeEvents**:
  - Detecta actualizações sobre as propriedades dos *data itens*, sinalizando-as em eventos do tipo `ListChanged` (esta interface é usualmente implementada em conjunto com `IBindingList`).

- **IBindingList**:
  - Actualizações no sentido *data source* ➔ *bound control* (evento `ListChanged`), sempre que há inserção, remoção ou substituição de um *data item*.
  - Criação de novas instância do mesmo tipo do *data item* (método `AddNew`);
  - Suporte para ordenação e pesquisa dos elementos;

ICancelAddNew

```
public interface ICancelAddNew {
  void CancelNew(int itemIndex); // roll back the addition of a new data item
  void EndNew(int itemIndex); // to commit the transaction of adding a new data item
}
```

- Se após a chamada ao método `AddNew` for efectuada uma outra operação sobre a colecção (Ex: inserção, remoção, selecção de outro elemento, etc), a adição também será *commited*. Ou seja, não é obrigatório existir um fim explicito da transacção para que esta fique *commited*.

- A classe `BindingSource` também implementa `ICancelAddNew`, dando suporte aos *bound control*'s para adição transnacional de elementos, mesmo quando o *data source* não o dá.

  (`BindingSource` tem implementação explícita da interface `ICancelAddNew`)

Demo 12

BindingListDataSource

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

IEditableObject

```
public interface ICancelAddNew {
  void BeginEdit(); // Called when an editing operation is commenced against an instance
  void EndEdit();   // Pushes changes into the underlying object
  void CancelEdit();// Discard changes since the last BeginEdit or AddNew call
}
```

- O método `CancelEdit` não deve ser chamado após o `EndEdit` e antes do `BeginEdit` ter sido novamente chamado.
- A implementação da interface `IEditableObject` consiste em fazer *cache* das alterações ao objecto de modo a preservar os valores originais antes da operação de edição ter-se iniciado com a chamada ao método `BeginEdit`.
- Dependendo da forma como é usado o *data binding*, poderão existir vária chamadas ao método `BeginEdit` e só se quer preservar os valores originais quando a operação de edição começou realmente e não repetidamente de cada vez que o `BeginEdit` é invocado.
- O mecanismo de *data binding* para *data source's* do tipo `DataSet` obedece a este comportamento porque os *data itens* são instâncias do tipo `DataRowView` que implementam `IEditableObject`.

IRaiseItemChangeEvents

```
public interface IRaiseItemChangedEvents {
  bool RaisesItemChangedEvents { get; } // Gets a value indicating whether item property
                                        // value changes raise ListChanged events
}
```

- Verifica se o tipo `T` de `BindingList<T>` implementa `INotifyPropertyChanged`, para indicar que o evento `ListChanged` lança eventos do tipo `ItemChanged` quando os valores das propriedades dos *data itens* são alterados.

  (Este membro não pode ser redefinido nas classes derivadas)

  (A abordagem indicada por Bryan Noyes sobre a implementação desta interface contraria a especificação do MSDN)

```
public interface INotifyPropertyChanged {
  event PropertyChangedEventHandler PropertyChanged;// Occurs when a property value changes
}
```

- Na versão 1.1 este processo era suportado por intermédio de eventos de alteração de propriedades que obedeciam ao padrão `<propertyname>Changed`.

Demo 13

RaiseChangeEvents

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

IBindingList

```
public class IBindingList : IList, ICollection, IEnumerable {
  // Editing members
  bool AllowNew { get; } // The default is true
  bool AllowEdit { get; } // The default depends on the underlying type contained in the list.
  bool AllowRemove { get; } // The default is true
  Object AddNew(); // Adds and returns a new data item

  // Sorting members
  bool SupportsSorting { get; } // The default is false in BindingList
  bool IsSorted { get; } // Is the list data source sorted?
  PropertyDescriptor SortProperty { get; } // Current sort column
  ListSortDirection SortDirection { get; } // Current sort direction
  void ApplySort(PropertyDescriptor property, ListSortDirection direction);
  void RemoveSort(); // Revert to an unsorted state

  // Searching members
  bool SupportsSearching { get; } // The default is false in BindingList
  int Find(PropertyDescriptor property,  object key);

  // Indexing members
  void AddIndex(PropertyDescriptor property); // Add index to desired column
  void RemoveIndex(PropertyDescriptor property); // Remove index from desired column

  // Change notification members
  bool SupportsChangeNotification { get; } // Supported?
  event ListChangedEventHandler ListChanged; // Broadcast list change
```

PropertyDescriptor

- Classe definida no *namespace* `System.ComponentModel` tem um papel importante no mecanismo de *data binding* como forma de inspecção dinâmica da informação de tipo das propriedades individuais dos *data itens*.

- Propriedades relevantes para *data binding*:

Método	Descrição
`string Name { get; }`	Nome da propriedade
`Type PropertyType { get; }`	Tipo da propriedade
`Type ComponentType { get; }`	Tipo da classe em que a propriedade é definida.

- Métodos relevantes para *data binding*:

Método	Descrição
`object GetValue(object component)`	Recebe uma referência para o data item donde se pretende obter o valor da propriedade.
`void SetValue(object component, object value)`	Altera o valor da propriedade no *data item* recebido por parâmetro com o valor de `value`.

`BindingList<T>` e `IBindingList`

- **`BindingList<T>`** não implementa as funcionalidades de **ordenação e pesquisa.** Razões:
  - Dificuldade de implementar uma "**ordenação genérica genuína**".
  - Diferentes tipos têm diferentes propriedades, que podem não ser ordenáveis ou pesquisáveis.

- Para dar suporte a estas funcionalidades pode ser definida um novo tipo de lista que estende de **`BindingList<T>`** e redefine os métodos e propriedades apropriadas.

- Os métodos e propriedades de `IBindingList` implementados em `BindingList<T>` são não virtuais e a sua implementação inclui a chamada a métodos virtuais com o mesmo nome acrescido do sufixo `Core`, como por exemplo: `AddNewCore`, `ApplySortCore` e `IsSortedCore`.

  (aplicação do padrão *template method*)

Suporte para pesquisa em `IBindingList`

- Redefinir os métodos:

```
public class BindingList<T> : IBindingList, … {
  // Core Find members
  protected virtual void FindCore(
    PropertyDescriptor property, object key);
  protected virtual bool SupportsSearchingCore { get; }
}
```

Demo 14

BindingListView

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Suporte para ordenação em `IBindingList`

```
public class BindingList<T> : IBindingList, … {
  // Core Sort methods
  protected virtual void ApplySortCore(
    PropertyDescriptor property, ListSortDirection direction);
  protected virtual void RemoveSortCore();
  // Core Sort properties
  protected virtual bool SupportsSortingCore { get; }
  protected virtual bool IsSortedCore { get; }
  protected virtual PropertyDescriptor SortPropertyCore { get; }
  protected virtual ListSortDirection SortDirectionCore { get; }
```

```
public enum ListSortDirection {
  Ascending = 0, Descending = 1,
}
```

- Remover a ordenação (`RemoveSort`) significa repor os elementos segundo a ordem original antes da invocação do método `ApplySort`.
  - Uma solução passa por guardar uma cópia da colecção original. Contudo é introduzido um outro grau de complexidade derivado da possibilidade de haver inserção ou remoção de novos elementos, obrigando a intercepção dessas operações.

- BindindList para XML
- Parar o binding no BindingSource e bombar dados para o data source.

Referências

- **Chris Sells**

  Publisher: Addison Wesley Professional
  Copyright: 2004

- **Chris Sells** and **Michael Weinhardt**
  Publisher: Addison Wesley Professional
  Status: Not Yet Published
  Estimated Availability: 05/05/2006

- **Brian Noyes**

  Publisher: Addison Wesley Professional
  Copyright: 2006