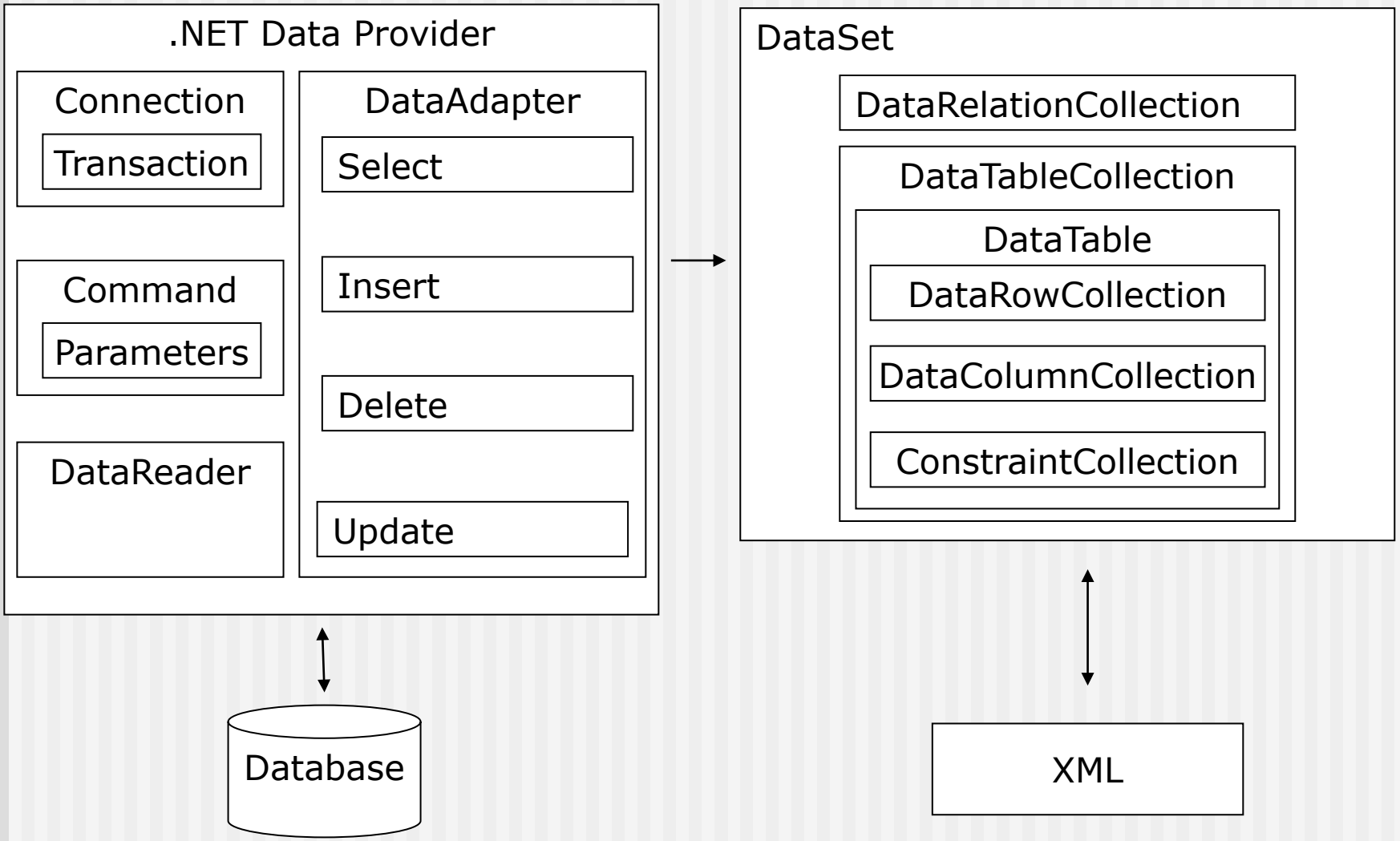
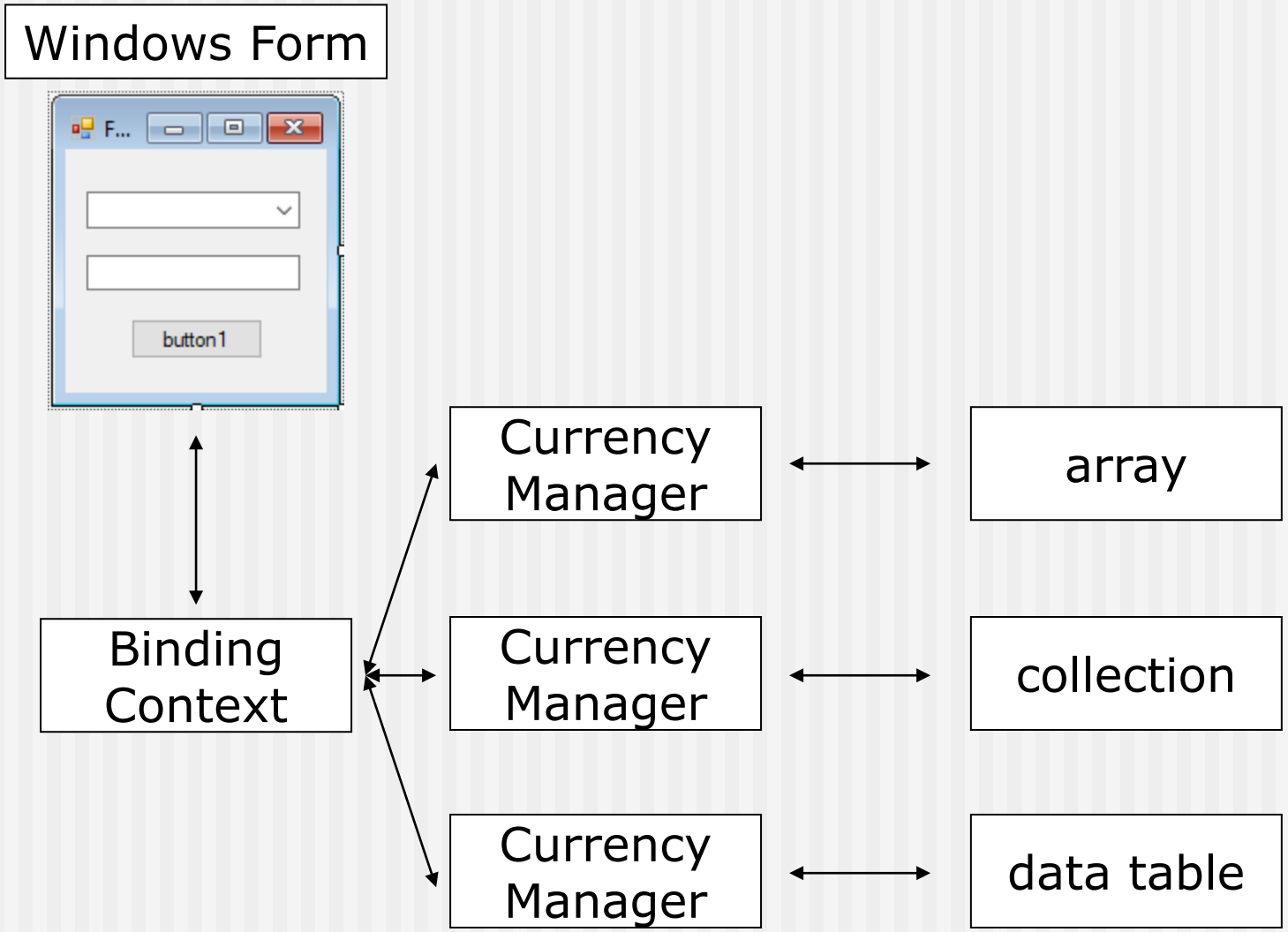
Seminar 2

ADO.NET
Data Binding

The ADO.NET Architecture



Data Binding – Windows Forms



Windows Forms – Structures to Bind To

- list-based binding – the object should support the *IList* interface
- ADO.NET provides data structures suitable for data binding
  - *DataColumn*
  - *DataTable* – with columns, rows, constraints
  - *DataView* – customized view of a single data table
  - *DataSet* – with tables, relationships
  - *DataViewManager* – customized view of a *DataSet*

CurrencyManager

- keeps data-bound controls synchronized with each other
- for each data source associated with a Windows Form, there is one *CurrencyManager* object
- *currency* = the current position within a data structure
- the *Position* property - determines the current position of all the controls using the same *CurrencyManager* (the position within the underlying list)

BindingContext

- manages the collection of *CurrencyManager* objects for any container control / form

The *DataBindings* Property and the *Binding* Class

- *DataBindings*
  - property of a control
  - retrieves the data bindings for the control
  - one can bind any property of a control to the property of an object
- *Binding* class
  - the simple binding between the property value of an object and the property value of a control
  - *Binding (String propertyName, Object dataSource, String dataMember)*

Dataset – *Fill, Update*

- a dataset doesn't contain data by default
- data tables are filled with data when executing *TableAdapter* / data adapter (*SqlDataAdapter*) queries / commands

```
aTableAdapter.Fill(aDataSet.TableName);
```

- saving data:

```
aTableAdapter.Update(aDataSet.TableName);
```

- when the *Update* method is called, the value of the *RowState* property is examined to determine which records must be saved and which command (*InsertCommand*, *UpdateCommand*, *DeleteCommand*) must be executed

Accessing Records

- every table exposes a collection of rows
- rows can be accessed through the collection's index or by using collection-specific statements in the host programming language
- typed dataset

```
TextBox1.Text = ds1.TableName[3].aField;
```

- untyped dataset

```
string val = (string)
ds1.Tables["TableName"].Rows[0]["aField"];
```

Related Tables, *DataRelation* Objects

- the data in a dataset's tables can be interrelated
- *DataRelation* objects can be created to describe the relationships among the dataset's tables
- a *DataRelation* object can be used to locate related records:
  - the *GetChildRows* method
    - called on a *DataRow* in the parent table
    - returns an array of related child records (*DataRow* objects)

Related Tables, *DataRelation* Objects

- a *DataRelation* object can be used to locate related records:
  - the *GetParentRow* method
    - called on a *DataRow* in the child table
    - returns a single *DataRow* from the parent table

Related Tables, *DataRelation* Objects

- return the child records for a parent record

```
string custID = "VINET";
NorthwindDataSet.OrdersRow[] orders;
orders = (NorthwindDataSet.OrdersRow[])
     northwindDataSet.Customers.
     FindByCustomerID(custID).GetChildRows
          ("FK_Orders_Customers");
MessageBox.Show(orders.Length.ToString());
```

Related Tables, *DataRelation* Objects

- return the parent record for a child record

```
int orderID = 10835;
NorthwindDataSet.CustomersRow customer;
customer = (NorthwindDataSet.CustomersRow)
     northwindDataSet.Orders.
     FindByOrderID(orderID).
     GetParentRow("FK_Orders_Customers");
MessageBox.Show(customer.CompanyName);
```

Creating an Application in Visual Studio

- create a new *Windows Forms* project
  - *File* -> New -> Project
  - select *Windows Forms App,* specify a name, choose a location
  - click on *OK*
  - the project is created and added to the *Solution Explorer*

Creating an Application in Visual Studio

- create a *Data Source*
  - start the *Data Source Configuration Wizard* (*Data Sources* window)
  - choose a Data Source Type (e.g., Database)
  - choose a Database Model (Dataset)
  - choose the data connection
  - select the database objects (e.g., the required tables)

Creating an Application in Visual Studio

- drag items (e.g., a particular table) from the Data Sources window onto the form to create *data-bound controls*

=> the following components are now visible in the component tray:

- *DataSet* - typed dataset that contains tables
- *BindingSource* - binds the controls on the form to the table in the dataset
- *BindingNavigator* – allows the user to navigate through the rows in the table
- *TableAdapter* – communication between the database and the dataset

Creating an Application in Visual Studio

- drag items (e.g., a particular table) from the Data Sources window onto the form to create *data-bound controls*

=> the following components are now visible in the component tray:

- *TableAdapterManager* - controls the order of individual inserts, updates, and deletes

Constraints

- two types of constraints: unique / foreign key
- unique constraint
  - all values in a set of columns must be unique
  - class *UniqueConstraint*
- foreign key constraint
  - defines rules on how to change related child records when a parent record is updated or deleted
  - class *ForeignKeyConstraint*

Constraints

- a foreign key constraint is automatically added when creating a *DataRelation* object in a dataset
- a table's constraints can be retrieved using the *Constraints* property
- the boolean property *EnforceConstraints* in the *Dataset* class indicates whether constraints are enforced or not (by default it's *true*)

Entity Framework

- set of technologies in ADO.NET that support the development of data-oriented applications
- developers can work with domain-specific objects and properties (as opposed to dealing with tables and columns)
- developers can query entities and relationships in the domain model; the Entity Framework translates such operations to data source–specific commands

Entity Framework