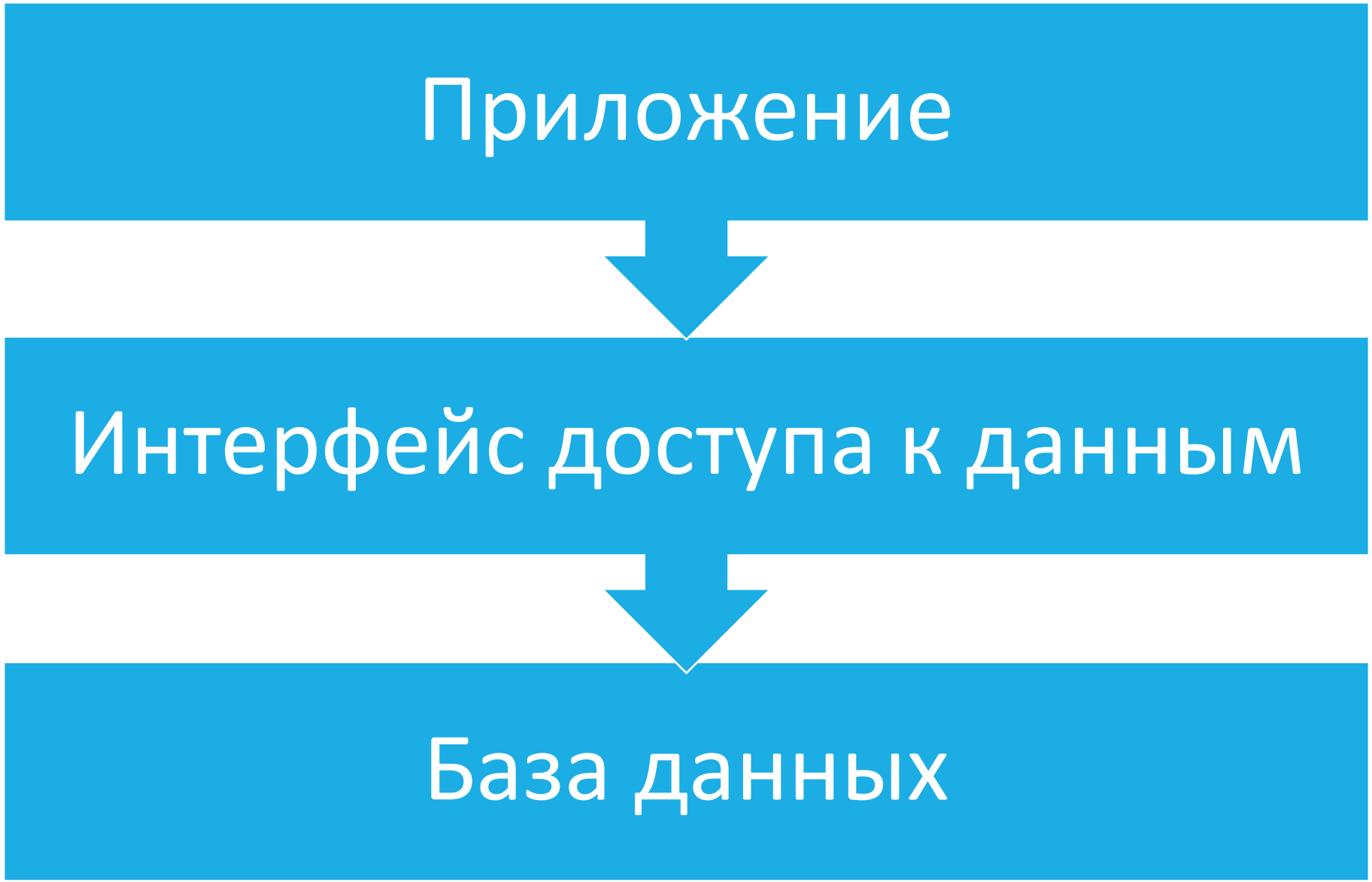
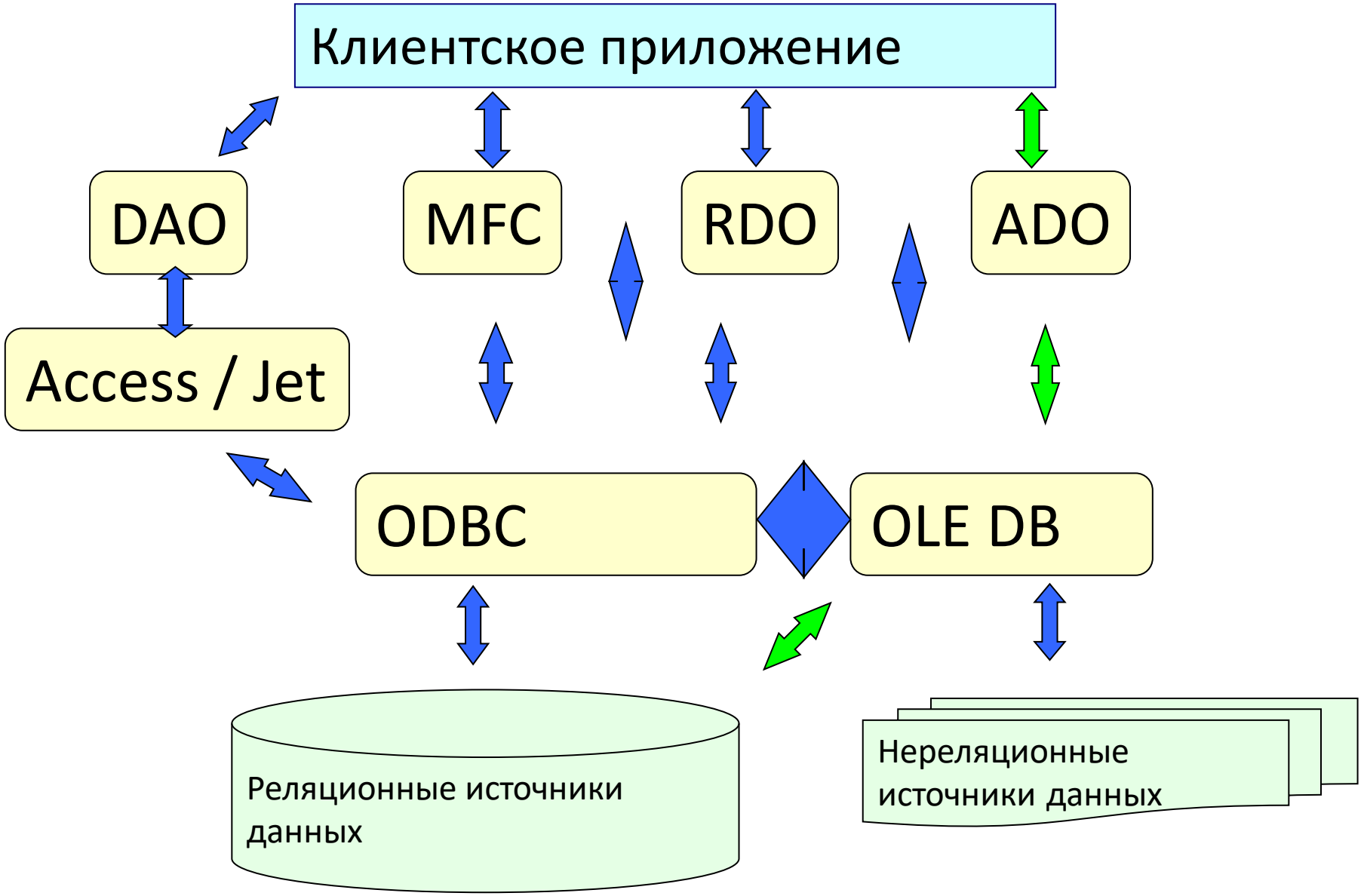
Иллюстрации к курсу лекций по дисциплине «Программирование на C#»

Программные интерфейсы для доступа к данным. Основы ADO.Net для работы с реляционной СУБД.

Доступ к источникам данных



Программные интерфейсы доступа к данным (история)



Сравнение интерфейсов доступа к данным (история)

	ODBC	MFC ODBC	DAO	RDO	OLE DB	ADO
Объектная модель	-	+	+	+	+	++
Нереляционные источники данных	-	-	-	-	+	+
Низкоуровневый контроль	+		-	-	+	
Производительность	+		-		++	
Соотношение функциональность / объем кода	-		+		-	+

ADO (ActiveX Data Objects)

API для доступа к данным, разработанный компанией Microsoft (MS Access, MS SQL Server) и основанный на технологии ActiveX. ADO позволяет представлять данные из разнообразных источников (реляционных баз данных, текстовых файлов и т. д.) в объектно-ориентированном виде.

Основной недостаток ADO

Неспособность работы в т.н. “отключенном” режиме.

ADO .NET (ActiveX Data Object для .NET)

- Подключенный режим (connected) :
  - ✓Явное подключение к источнику данных
- Автономный режим (disconnected):
  - ✓Работа с копией данных из источника. Подключение сначала открывается для чтения данных, потом закрывается и открывается снова только для изменения данных (возможно, пакетного).
- Entity Framework:
  - ✓Сокрытие низкоуровневых деталей работы с источником данных.

Подключенный режим

На данном уровне работа с данными ведётся через активное соединение с источником данных через специализированных поставщиков данных, предназначенного для конкретного источника данных (СУБД).

Для получения данных выполняются следующие шаги:

- Создание, настройка и открытие объекта подключения.
- Создание и настройка объекта команды, указывающего на объект подключения в конструкторе или через свойство Connection.
- Вызов метода ExecuteReader() настроенного объекта команды.
- Обработка каждой записи с помощью метода Read() объекта чтения данных.
- Объекты чтения данных предоставляют поток данных, для чтения в прямом направлении. Чтение происходит каждый раз по одной записи. Следовательно объекты чтения обрабатывают только SELECT-запросы. Открытие и закрытие подключения к БД полностью возлагается на программиста.

СУБД – система управления базами данных

Автономный режим

Автономный уровень ADO.NET позволяет отображать реляционные данные с помощью модели объектов в память.

Типы данных из System.Data воспроизводят не только отображение строк и столбцов, а также отношения между таблицами, первичные ключи и т. д.

Так как отображение данных происходит в память, постоянное подключение к серверу СУБД не требуется. Подключение может устанавливаться вновь только при изменении данных (добавление, удаление, обновление).

Т.е. автономный уровень бережнее относится к ресурсам СУБД.
Но есть недостаток – повышенные требования к клиенту. Например, при чтении большого набора данных, требуется сразу много памяти для загрузки данных (в отличие от подключаемого режима, где все читается последовательно).

Entity Framework

Entity Framework выводит абстракцию на новый уровень – уровень объектной модели приложения.

Т.е. данные из БД отображаются на бизнес-объекты приложения, что позволяет работать с данными как с обычными объектами языка. Сущности (entities) — это концептуальная модель физической базы данных, которая отображается на предметную область. Формально говоря, эта модель называется моделью сущностных данных (Entity Data Model — EDM). Модель EDM представляет собой набор клиентских классов, которые отображаются на физическую БД.

Однако, сущности могут и не напрямую отображаться на схему БД. Сущностные классы можно реструктурировать для соответствия существующим потребностям, и исполняющая среда EF отобразит эти уникальные имена на корректную схему базы данных.

Поставщики .Net для работы с данными

- **SQL Server** - предоставляет оптимизированный доступ к базам данных SQL Server (версии 7.0 и выше);
- **OLE DB** - предоставляет доступ к любому источнику данных, который имеет драйвер OLE DB. Это включает базы данных SQL Server версий, предшествующих 7.0;
- **Oracle** – устарел. Используйте DP.NET (Oracle Data Provider для .NET) производства Oracle, который доступен на веб-сайте http://www.oracle.com;
- **ODBC** - предоставляет доступ к любому источнику данных, имеющему драйвер ODBC.

Поскольку каждый поставщик использует одни и те же интерфейсы и базовые классы, легко писать обобщенный код доступа к данным работая с интерфейсами.

Классы для работы с данными

- **Connection** – обеспечивает подключение к БД;
- **Command** – для управления БД; позволяет выполнять команды SQL или хранимые процедуры;
- **DataReader** – предоставляет доступный только для однонаправленного чтения набор записей (подключенный режим работы к БД);
- **DataAdapter** – заполняет отсоединенный объект **DataSet** или **DataTable** и обновляет его содержимое.
- **Parameter** – именованный параметр в параметризованном запросе;
- **Transaction** – транзакция.

Основные классы в System.Data

Ключевые классы контейнеров данных, которые моделируют столбцы, отношения, таблицы, наборы данных, строки, представления, ограничения и др.

Дополнительно содержит ключевые интерфейсы, которые реализованы объектами данных, основанными на соединениях.

Основные классы:
DataRow, DataColumn, DataTable, DataSet, DataView, Constraint, DataRelation, *Exception.

Подключение к SQL Server

```
// Создание подключения
using (SqlConnection cn = new SqlConnection()
{
    cn.ConnectionString = @"Data
Source=(local)\SQLEXPRESS;Initial
Catalog=AdventureWorksLT;Integrated Security=SSPI;";
    cn.Open();
    // Работа с базой данных
    cn.Close();
 }
```

Создание строки подключения

```
SqlConnectionStringBuilder connect = new
SqlConnectionStringBuilder();
connect.InitialCatalog = "AdventureWorksLT";
connect.DataSource = @"(local)\SQLEXPRESS";
connect.ConnectTimeout = 30;
connect.IntegratedSecurity = true;
```

Хранение строки подключения (*.config)

Строку подключения хранят в конфигурационных файлах (например, web.config):

```
<connectionStrings>
    <add name="DBcon"
connectionString="Data Source=127.0.0.1\SQLEXPRESS;
Initial Catalog=SomeDB;Integrated Security=True"/>
</connectionStrings>
```

```
 // Получение строки подключения из *.config
string cnStr =
ConfigurationManager.ConnectionStrings["DBcon"].Co
nnectionString;
```

Хранение строки подключения (*.config)

Строку подключения хранят в конфигурационных файлах (например, web.config):

```
<connectionStrings>
    <add name="DBcon"
connectionString="Data Source=127.0.0.1\SQLEXPRESS;
Initial Catalog=SomeDB;Integrated Security=True"/>
</connectionStrings>
```

```
 // Получение строки подключения из *.config
string cnStr =
ConfigurationManager.ConnectionStrings["DBcon"].Co
nnectionString;
```

Класс Command

Класс Command позволяет выполнить запросы к базе данных (выборку, обновление, дополнение, удаление).

Свойства:

- **CommandType**:
  - CommandType.Text (по умолчанию)- операторы SQL ;
  - CommandType.TableDirect – работа с конкретной таблицей;
  - CommandType.StoredProcedure – вызов хранимой в БД процедуры.

- **CommandText**:
  - текст оператора SQL (для типа CommandType.Text);
  - имя таблицы (для CommandType.TableDirect);
  - имя хранимой процедуры с параметрами (для CommandType.StoredProcedure).

- **Connection** – ссылка на открытое соединение (объект Connection);
- **Parameters** – коллекция параметров запроса.

Работа с объектом SqlCommand

```
// Создание объекта команды с помощью конструктора
string strSQL = "Select * From SalesLT.Product";
SqlCommand myCommand = new SqlCommand(strSQL, cn);

// Создание еще одного объекта команды с помощью свойств
SqlCommand testCommand = new SqlCommand();
testCommand.Connection = cn;
testCommand.CommandText = strSQL;

// Открыть подключение
cn.Open();

// ТУТ делаем что-то с БД

// Закрыть подключение
cn.Close();
```

Основные методы SqlCommand

- ✓ **ExecuteReader()** – выполняет оператор SELECT, создает и возвращает ссылку на объект **DataReader**, который содержит результат выполнения запроса.

- ✓ **ExecuteNonQuery()** – выполняет операторы (INSERT, DELETE, UPDATE) языка SQL и возвращает количество обработанных записей.

- ✓ **ExecuteScalar()** – возвращает значение из первой строки первого столбца в результирующем наборе.

```
string strSQL = "UPDATE SalesLT.Customer SET
LastName = 'Johnson' WHERE LastName = 'Walton'";
SqlCommand myCommand = new SqlCommand(strSQL, cn);
int i = myCommand.ExecuteNonQuery();
```

Параметризация SqlCommand

(или основа борьбы с атакой SQLInjection)

В SQL-запросе в Command.Text можно задавать переменные – параметры (@variable).

Параметры позволяют менять (параметризовать) SQL-запрос без переписывания его текста.

Параметры используются при вызове хранимой процедуры для передачи входных данных и получения результатов.

Параметризация SqlCommand на примере

(или основа борьбы с атакой SQLInjection)

```
string strSQL = string.Format("Insert Into Inventory" +
"(CarID, Make, Color, PetName) Values(@CarId, @Make, @Color, @PetName)");

SqlCommand testCommand = new SqlCommand(strSQL, cn);
SqlParameter param = new SqlParameter();
param.ParameterName = "@CarID";
param.Value = id;
param.SqlDbType = SqlDbType.Int;
testCommand.Parameters.Add(param);
…
```