ЛЕКЦИЯ №9 СОЗДАНИЕ DLL, РАБОТА С БАЗОЙ ДАННЫХ

Москва, 2019

Создание библиотеки

1. Недостатки разделяемого код двумя приложениями
2. Решение - компиляция библиотек кода в сборки с расширением dll.
3. Чтобы внести изменения в код библиотеки, которые не затрагивают интерфейсы с внешними сборками, то достаточно будет только перекомпилировать библиотеку и обновить файл сборки библиотеки. Исполняемую сборку перекомпилировать не нужно.
4. Dll может на любом языке NET.

Операции LINQ

Contains	Возвращает true, если входная последовательность содержит заданный элемент
Any	Возвращает true, если хотя бы один элемент последовательности удовлетворяет заданному условию
All	Возвращает true, если все элементы входной последовательности соответствуют заданному условию
SequenceEqual	Возвращает true, если все элементы входной последовательности соответствуют элементам другой последовательности

Определение

Концепция разделяемых классов и методов позволяет разделить определение классов и методов по разным файлам. Во время компиляции проекта они собираются воедино

Частичные методы

Определение

Метод расширения — метод, добавляемый в существующий класс путем специального определения в другом модуле/проекте и классе

Добавив в классе А метод расширения.

LINQ - технология

```
using System;

namespace ExtMethods.Module1
{
    public class Original
    {
        public int GetA()
        {
            Random rand = new Random(DateTime.Now.Millisecond);
            return rand.Next();
        }

        public int GetB()
        {
            Random rand = new Random(DateTime.Now.Millisecond * 553);
            return rand.Next();
        }
    }
}
```

```
using ExtMethods.Module1;

namespace ExtMethods.Module2
{
    public static class Extension
    {
        public static int GetC(this Original orig, int multi
        {
            return orig.GetA() * orig.GetB() * multiplier;
        }
    }
}
```

Расширения

Допустим необходимо подсчитать количество согласных букв, менять класс String не имеем права

```
using System;

namespace ExtMethods
{

    public static class StringExtensions
    {
        public static readonly string engConsonants = "wrtpsdfghklzxcvbnmy";
    }
}
```

Расширения

Допустим необходимо подсчитать количество согласных букв, менять класс String не имеем права

```
using System;
using System.Linq;

namespace ExtMethods
{
    public static class StringExtensions
    {
        public static readonly string engConsonants = "wrtpsdfghklzxcvbnmy";
        public static int CountConsonants(this String original)
        {
            return original.Where(chr => engConsonants.Contains(chr.ToString().ToLower())).Count();
        }
    }
}
```

Расширения

Пример использования класса String

```
namespace ExtMethods
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello!");

            string first = "Hello!";
            string second = "World";

            Console.WriteLine("Consosnants in " + first + ": " + first.CountConsonants());
            Console.WriteLine("Consosnants in " + second + ": " + second.CountConsonants());

            Console.ReadKey();
        }
    }
}
```

Работа с базами данных

После выбора поставщика данных важно работать с определенной базой данных. Вы должны установить
соединение с вашей базой данных для дальнейшего взаимодействия.
Класс DbConnection - это базовый класс, который обеспечивает функциональность, связанную с подключением.
SqlConnection является
наследуется от класса DbConnection и может использоваться для соединения с базой данных SQL Server.
Сервер поддреживает определенное количество подключений

Работа с базами данных

Listing 12-4. ExecuteNonQuery on Insert Command

```
string connectionString = "YOUR CONNECTION STRING HERE";
SqlConnection con = new SqlConnection(connectionString);
con.Open();

string command = "Insert into Student values(1,'Hamza Ali')";
SqlCommand cmd = new SqlCommand(command, con);
int result = cmd.ExecuteNonQuery();
con.Close();
if (result > 0)
  Console.WriteLine("Data is Inserted");
else
  Console.WriteLine("Error while inserting");
```

Работа с базами данных

```
string con = "YOUR CONNECTION STRING HERE";

string command = "select * from Student";
SqlCommand cmd = new SqlCommand(command, con);

SqlDataReader reader = cmd.ExecuteReader();

int StudentID = 0;
string StudentName = null;
if (reader.HasRows)
{
  while (reader.Read())
  {
  StudentID = int.Parse(reader[0].ToString());//0 index means first clm in the table which
  is StudentID
  StudentName = reader["StudentName"].ToString();//it will fetch the value of provided clm
  name
  }
}
reader.Close();
con.Close();

Console.WriteLine("ID is: " + StudentID);
Console.WriteLine("Name is: " + StudentName);
```

Работа с базами данных

```
string con = "YOUR CONNECTION STRING HERE";

string command = "select * from Student";
SqlDataAdapter ad = new SqlDataAdapter(command, con);

DataTable tbl = new DataTable();
ad.Fill(tbl);//Now the data in DataTable (memory)
con.Close();//connection closed

foreach (DataRow item in tbl.Rows)
{
  Console.WriteLine("ID is: " + item[0]);
  Console.WriteLine("Name is: " + item[1]);
}
```

Работа с базами данных

```
string con = "YOUR CONNECTION STRING HERE";

string command = "select * from Student";
SqlDataAdapter ad = new SqlDataAdapter(command, con);

DataTable tbl = new DataTable();
ad.Fill(tbl);//Now the data in DataTable (memory)
con.Close();//connection closed

foreach (DataRow item in tbl.Rows)
{
  Console.WriteLine("ID is: " + item[0]);
  Console.WriteLine("Name is: " + item[1]);
}
```

Работа с базами данных

Вы также можете использовать DataSet вместо DataTable при ожидании нескольких наборов результатов. Работа это
то же самое, за исключением того, что он может вернуть несколько таблиц. DataSet имеет свойство Table, с помощью которого вы можете перебирать
конкретные данные таблицы.
Как уже говорилось, вы можете выполнять дальнейшие операции над DataTable или DataSet, такие как вставка, удаление и т. Д.

Работа с базами данных

Listing 12-8. Insertion of data (disconnected layer)

```
string connectionString = "YOUR CONNECTION STRING HERE";
SqlConnection con = new SqlConnection(connectionString);
con.Open();

string command = "select * from Student";//Currently has One Row(for example)
SqlDataAdapter ad = new SqlDataAdapter(command, con);

DataTable tbl = new DataTable();
ad.Fill(tbl);//Now the data in DataTable (memory)

//Data in Memory (One Row)
foreach (DataRow item in tbl.Rows)
{
  Console.WriteLine("ID is: " + item[0]);
  Console.WriteLine("Name is: " + item[1]);
}

//New Record to add in DataTable
DataRow newRow = tbl.NewRow();
newRow["StudentID"] = 2;
newRow["StudentName"] = "Ali Asad";
tbl.Rows.Add(newRow);
```

Работа с базами данных

```
//Two Rows(As new row added to DataTable)
foreach (DataRow item in tbl.Rows)
{
  Console.WriteLine("ID is: " + item[0]);
  Console.WriteLine("Name is: " + item[1]);
}

//Now newRow has to add in Database(Pass newRow Parameters to this insert query)
string newCommand = @"Insert into Student(StudentID,StudentName)
                                Values(@StudentID,@StudentName)";

   SqlCommand insertCommand = new SqlCommand(newCommand, con);

   //Create the parameters
   insertCommand.Parameters.Add(new SqlParameter("@StudentID", SqlDbType.Int, Int32.
   MaxValue,"StudentID"));
   insertCommand.Parameters.Add(new SqlParameter("@StudentName",  SqlDbType.VarChar,
   40,"StudentName"));

   //Associate Insert Command to DataAdapter so that it could add into Database
   ad.InsertCommand = insertCommand;

   ad.Update(tbl);

   con.Close();
```

Entity Framework

Подключенные и отключенные слои вынуждают вас обрабатывать данные в виде физической схемы база данных.
Эти слои тесно связаны с реляционной базой данных, так как пользователь должен использовать SQL для
выполнения запросов и иметь в виду соединение, команду DataReader, DataSet, DataAdapter и т. д.
В отличие от этих слоев, Entity Framework предоставляет вам объектно-ориентированный способ взаимодействия с базой данных.
Не нужно беспокоиться о соединениях или объектах, подобных командам.
Подобные вещи автоматически обрабатываются Entity Framework.
Object Relational Mapper (ORM). Это предпочтительный подход к взаимодействию с базой данных для тех, кто / не имеет слабых знаний SQL, потому что он дает объектно-ориентированный взаимодействие с базой данных, поскольку она отображает схему базы данных в классы C #.

Entity Framework

LINQ используется вместо SQL, и вы можете использовать один из типов LINQ с вашим источником данных, предоставленным ADO.NET (база данных). Когда вы выполняете запросы LINQ к своему источнику данных ADO.NET, среда выполнения генерирует правильный оператор SQL от вашего имени. Но это замедляет производительность по сравнению с подключенными и отключенными слоями.

Entity Framework (EF) обычно имеет четыре подхода к использованию:

1. EF Designer из базы данных
2. Пустая модель EF Designer
3. Пустой код Первая модель
4. Код сначала из базы данных

Мы будем использовать подход «EF Designer from database» с базой данных.

Чтобы использовать подход Entity Framework (EF Designer из базы данных) для взаимодействия с базой данных, используя Мастер Entity Data Model Wizard:

Entity Framework

Это возможности Entity Framework. Вам не нужно ничего, кроме как работать с
только относительная вещь. u0831361_baseEntities - это имя, которое представляет вашу базу данных. Оно содержит все вещи внутри вашей база данных. Вы можете использовать его объект для доступа к базам данных или их использования, например, для доступа к таблицам и т. д.

«u0831361_baseEntities» является объектом базы данных, и вы можете получить доступ к ее таблицам и после сохранения базы данных будет добавлен новый заказ в вашу базу данных.
Вы можете выполнять другие операции, такие как обновление, удаление и поиск, используя LINQ.

Using

Ключевое слово `using` имеет три основных применения:

- Инструкция using определяет область, по завершении которой объект удаляется.
- Директива using создает псевдоним для пространства имен или импортирует типы, определенные в других пространствах имен.
- Директива using static импортирует элементы из одного класса.

Using

Ключевое слово Using упрощает работу с объектами которые реализуют интерфейс IDisposable.

Интерфейс IDisposable содержит один метод .Dispose(), который используется для освобождения ресурсов, которые захватил объект. При использовании Using не обязательно явно вызывать .Dispose() для объекта.

```
using (SqlConnection conn = new SqlConnection()) {
// какая-нибудь SQL операция
}
```

При этом компилятор генерирует следующий код:

```
SqlConnection conn = new SqlConnection();
try {

} finally {
    // здесь для conn вызывается .Dispose()
}
```

Using блоки делают код более читабельным и компактным.

Using

```
using (var font1 = new Font("Arial", 10.0f))
{
    byte charset = font1.GdiCharSet;
}
```

```
{
  var font1 = new Font("Arial", 10.0f);
  try
  {
    byte charset = font1.GdiCharSet;
  }
  finally
  {
    if (font1 != null)
      ((IDisposable)font1).Dispose();
  }
}
```

Вопрос

Допусти требуется проверка равенства переменной S нулю. (S – переменная с плавающей точкой)

If (S==0)
Eps = 1E-7 = 0.00000001; // 1* 10^(-7)

If (abs(S)<eps)

10^55 + 123 – 10^55 = 123

Возм. компьютерный результат: 0,006856?