ЛАБОРАТОРНАЯ РАБОТА №5 “РАЗРАБОТКА КОНСОЛЬНОГО ПИЛОЖЕНИЯ НА ЯЗЫКЕ C#”

1. Теоретическая часть

Платформа разработки **.NET** Framework позволяет создавать Web-сервисы XML, приложения Web Forms, Win32 GUI, Win32 GUI (с консольным интерфейсом пользователя), службы (управляемые Service Control Manager), утилиты и автономные компоненты.

NET Framework — это управляемая среда для разработки и исполнения приложений, **обеспечивающая контроль типов**.

Эта среда управляет выполнением программы она:

- выделяет память под данные и команды;
- назначает разрешения программе или отказывает в их предоставлении;
- начинает исполнение приложения и управляет его ходом;
- отвечает за освобождение и повторное использование памяти, занятой ресурсами, более ненужными программе.

.NET Framework состоит из двух основных компонентов: общеязыковой исполняющей среды (CLR) и библиотеки классов Framework Class Library (FCL).

CLR можно рассматривать как среду, управляющую исполнением кода и предоставляющую ключевые функции, такие, как компиляция кода, выделение памяти, управление потоками и сбор мусора. Благодаря использованию **общей системы типов (common type system, CTS)** CLR выполняет строгую проверку типов, а защита по правам доступа к коду позволяет гарантировать исполнение кода в защищенном окружении.

Библиотека классов **.NET Framework содержит набор полезных типов, разработанных специально для CLR** и доступных для многократного использования. Типы, поддерживаемые .NET Framework, являются объектно-ориентированными, полностью расширяемыми и обеспечивают эффективную интеграцию приложений с .NET Framework.

При компиляции кода для .NET Framework компилятор генерирует код на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд.

Рис. 3. Компиляция исходных файлов

При исполнении CLR транслирует CIL в команды процессора (отличие от JAVA**)**.

Общеязыковая спецификация (Common Language Specification, CLS)-стандарт, который определяет минимальный набор правила, для разработчиков компиляторов, чтобы их языки интегрировались с другими. Microsoft разработала ряд таких языков: C++ с управляемыми расширениями, C #, Visual Basic .NET (сюда относятся и Visual Basic Scripting Edition или VBScript и Visual Basic for Applications или VBA), а также JScript и др.(см. Рис.4).

Все эти механизмы позволяют создавать собственные классы, упрощают установку приложений, предоставляют возможность работы на нескольких платформах, предоставляют возможность интеграции языков программирования и , соответственно, сервис сторонним приложениям, упрощается повторное использование кода и создается большой рынок готовых компонентов, реализуется автоматическое управление памятью (сбор мусора), гарантируется корректное обращение к существующим типам, единая структура обработки исключений для восстановления основного алгоритма, обеспечивается взаимодействие с существующим кодом.

Благодаря использованию общеязыковой исполняющей среды, межъязыковая совместимость позволяет компонентам, реализованным с применением .NET Framework, взаимодействовать друг с другом независимо от языка, на котором они написаны. Так, приложение на Visual Basic .NET может обращаться к DLL, написанной на C#, а та, в свою очередь, способна вызвать ресурсы, созданные на управляемом C++ или любом другом .NET-совместимом языке. Межъязыковая совместимость поддерживается и для наследования в ООП (объектно-ориентированном программировании), например, на основе C#-класса можно объявлять классы в программах на Visual Basic .NET и наоборот.

Основные отличия и преимущества платформы .NET

• **Единая программная модель (**представляется единая общая объектно-ориентированная программная модель подключения функций ОС.)

• **Упрощенная модель программирования (**разработчику не нужно разбираться с реестром, глобально-уникальными идентификаторами (QUID), lUnknown, A d d R e f , Release, HRESUIT и т. д. Но, в случае, если пишется приложение .NET

Framework, которое взаимодействует с существующим не-.NET кодом, нужно разбираться во всех этих концепциях.

• **Отсутствие проблем с версиями** Архитектура .NET Framework позволяет изолировать прикладные компоненты, так что приложение всегда загружает версии компонент, с которыми оно строилось и тестировалось. Если приложение работает после начальной установки, оно будет работать всегда. (в отличии от сложностей с версиями DLL.

• **Упрощенная разработка** Компоненты .NET Framework (их называют просто типами) теперь не связаны с реестром. По сути установка приложений .NET Framework сводится лишь к копированию файлов в нужные каталоги и установку ярлыков в меню Start, на рабочем столе или на панели быстрого запуска задач. Удаление же приложений сводится к удалению файлов.

• **Работа на нескольких платформах** При компиляции кода для .NET Framework компилятор генерируеткод на общем промежуточном языке (common intermediate language, CIL), а не традиционный код, состоящий из процессорных команд конкретного процессора. Это значит, что можете развертывать приложение для .NET Framework на любой машине, где работает версия CLR и FCL.

•**Интеграция языков программирования** COM позволяет разным языкам взаимодействовать. .NET Framework позволяет разным языкам интегрироваться, т. е. одному языку использовать типы, созданные на других языках. На основе предоставляемой CLR общей системы типов (Common Type System, CTS).

•**Упрощенное повторное использование кода** .NET Framework позволяет создавать собственные классы, предоставляющие сервис сторонним приложениям для многократного использования кода.

•**Автоматическое управление памятью (сбор мусора)** CLR автоматически отслеживает использование ресурсов, гарантируя, что не произойдет их утечки. По сути она исключает возможность явного «освобождения» памяти используя сборщик мусора.

• **Проверка безопасности типов** CLR может проверять безопасность использования типов в коде, что гарантирует корректное обращение к существующим типам. Если входной параметр метода объявлен как 4-байтное значение, CLR обнаружит и предотвратит применение 8-байтного значения для этого параметра. Безопасность типов также означает, что управление может

передаваться только в определенные точки (точки входа методов). Невозможно указать произвольный адрес и заставить программу исполняться, начиная с этого адреса. Совокупность всех этих защитных мер избавляет от многих распространенных программных ошибок.

• **Развитая поддержка отладки** Поскольку CLR используется для многих языков, можно написать отдельный фрагмент программы на языке, наиболее подходящем для конкретной задачи, — CLR полностью поддерживает отладку многоязыковых приложений.

• **Единый принцип обработки сбоев** В CLR обо всех сбоях сообщается через исключения, которые позволяют отделить код, необходимый для восстановления после сбоя, от основного алгоритма. Такое разделение облегчает написание, чтение и сопровождение программ. Кроме того, исключения работают в многомодульных и многоязыковых приложениях. CLR также предоставляет встроенные средства анализа стека, заметно упрощающие поиск фрагментов, вызывающих сбои.

• **Безопасность** Используется «кодоцентрический» способ контроля за поведением приложений. Такой подход реализован в модели безопасности доступа к коду.

• **Взаимодействие с существующим кодом** В .NET Framework реализована полная поддержка доступа к COM-компонентам и Win32-функциям в существующих DLL. Конечные пользователи не смогут непосредственно оценить CLR и ее возможности, но они обязательно заметят качество и возможности приложений, построенных для CLR. CLR позволяет разрабатывать и развертывать приложения быстрее и с меньшими накладными расходами, чем это было в прошлом.

2. Практическая часть

Цель лабораторной работы – приобрести навыки разработки консольных приложений на языке C# и на примере данной лабораторной работы ознакомиться с основными принципами разработки приложений для платформы .NET.

В данной лабораторной работе мы познакомимся с основными базовыми принципами разработки приложений в на платформе .NET. В данной работе будут рассмотрены базовые операции языка CSharp и работа с файлами на данной платформе.
При разработке приложений на языке CSharp имеется два пути разработки соответствующих приложений:

• Использование для написания исходных кодов программ любого текстового редактора и последующая компиляция исходных кодов программ с

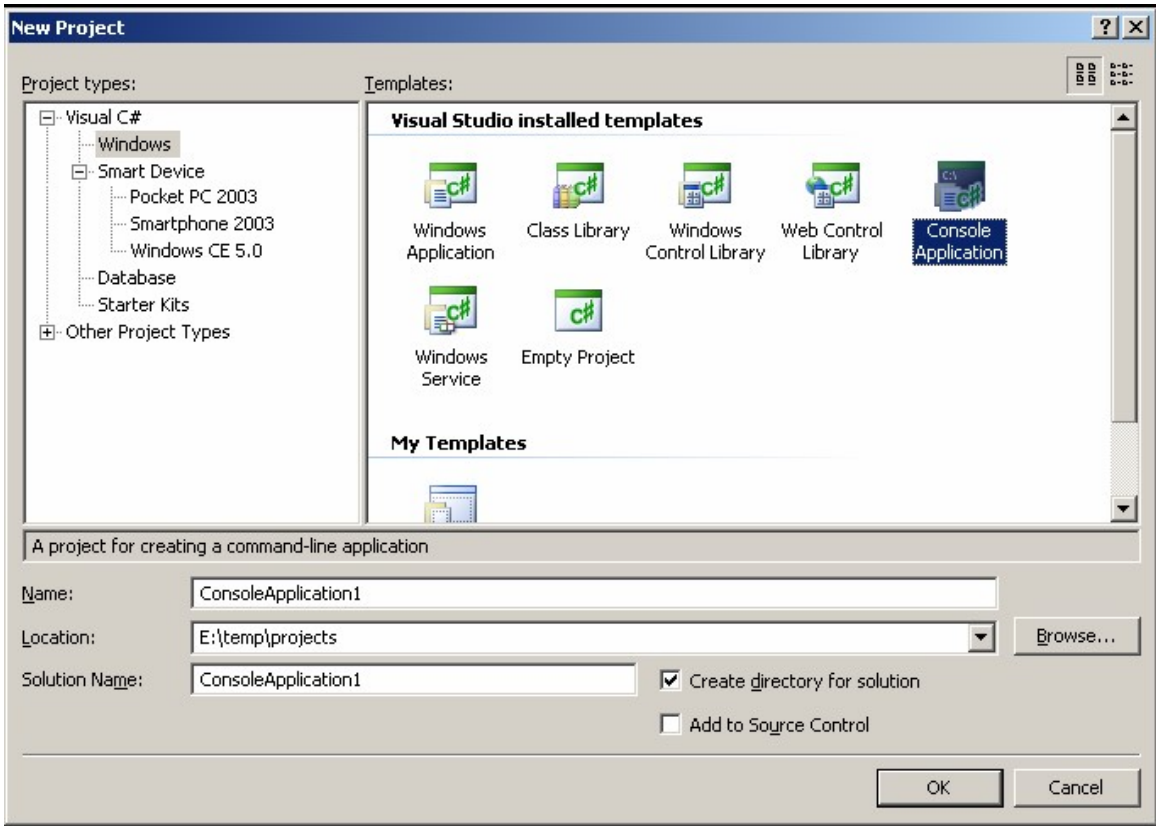
помощью компилятора **cvs.exe**, входящего в состав Microsoft .NET Framework SDK v2.0 (при этом способе написания приложений необходима лишь установка Microsoft .NET Framework SDK v2.0).

• Использование для написания исходных кодов программ среды разработки Microsoft Visual Studio 2003/2005. Данная среда разработки обладает мощными средствами разработки приложений, исчерпывающей помощью по всем компонентам, входящим в ее состав.

Последний способ более предпочтителен благодаря более высокой степени удобства использования, поэтому разработку наших работ будем производить с помощью Microsoft Visual Studio.

Приступим к созданию нашего первого приложения. По своему типу оно будет консольным, будет предусматривать работу с данными которые будут храниться в файлах.

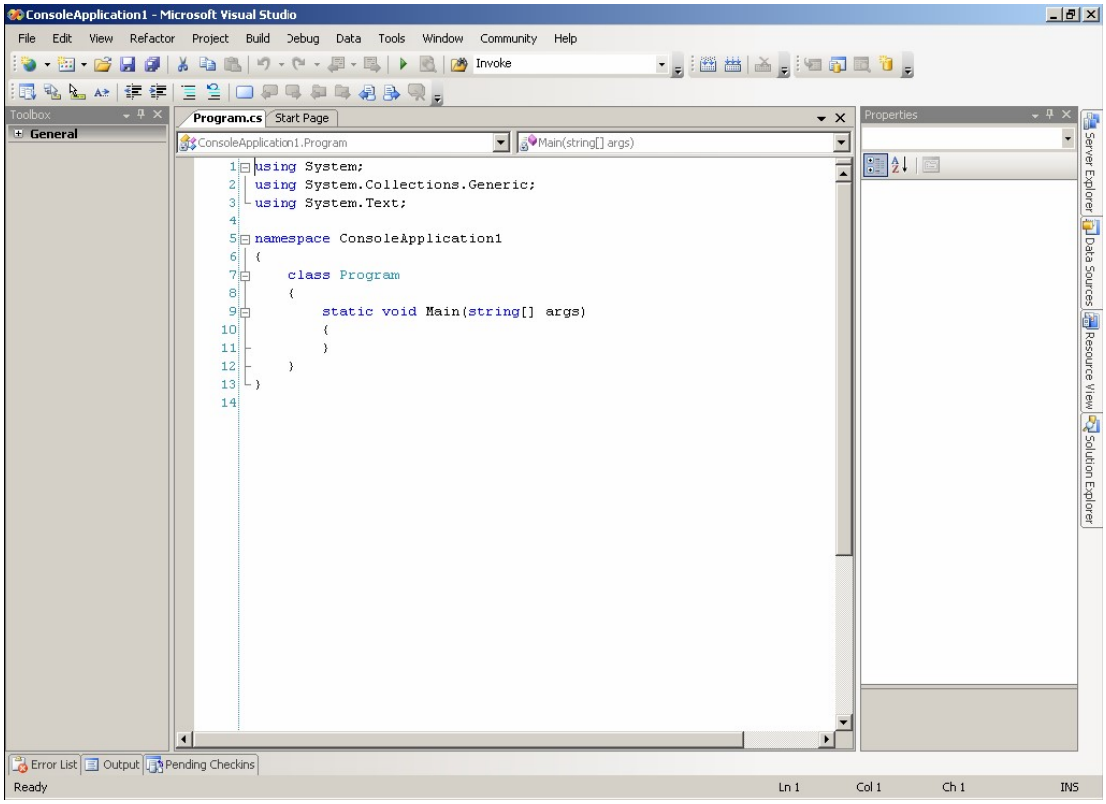
В данном окне мы видим список наших проектов. Для создания нового проекта необходимо выбрать команду меню **File->New Project…** после чего перед нами отобразится окно создания новых проектов:



В данном окне мы выбираем язык программирования (в нашем случае Visual C#), тип создаваемого приложения – **Console Application**, указываем название нашего проекта и его местоположение.

После того, как все параметры заданы завершаем создание нашего проекта нажатием на клавишу **OK**.

После этого перед пользователем отобразится главное окно разработки приложений:



Следует заметить, что после создания проекта система сама формирует начальный шаблон программы и пользователю остается наполнять его необходимым содержимым.

Как видно на рисунке, система сама добавила базовые необходимые библиотеки с помощью директивы `using`. Однако так как мы планируем вести работу с файлами нам также необходима дополнительная библиотека `System.IO`, поэтому добавим ее в список подключаемых библиотек.

После этого объявим необходимые нам переменные после объявления класса:

```
    ….
class Laba
{
private String region;
private String tkind;
private long quant;
    ….
```

Затем приступим к реализации бизнес-логики нашего приложения. Реализация описывается в функции `Main()` нашего приложения.

Рассмотрим подробнее формирование начального пользовательского меню с помощью данного фрагмента кода:

```
    ….
Console.WriteLine("Vyberite punkt:\n 1 - Vvesti novuju zapis\n 2 -
Posmotret' posadki derevjev v rajonax\n 3-Posmotret konkretnij
rajon\n"+" 4-Udalit zapis\n 5 - Redaktirovat' zapis\n 6 -
Sortirovat' po rajonu\n 7 - Exit");
     String choice=Console.ReadLine();
switch(choice)
{
….
```

В данном фрагменте кода с помощью функции `Console.WriteLine` пользователю отображается меню выбора функций.

После того, как пользователь делает свой выбор мы с помощью функции `Console.ReadLine()` присваиваем переменной `choice` выбор пользователя (т.е. номер выбранного пункта меню). После этого с помощью кнострукции `switch(choice)` мы обрабатываем пользовательские запросы.

Рассмотрим запрос пользователя на добавление новой записи в файл:

```
     case "1":
{
Console.WriteLine("Vvedite rajon, porodu dereva i kolichestvo:");

//чтение вводимых данных
String region=Console.ReadLine();
String tkind=Console.ReadLine();
long quant=System.Int64.Parse(Console.ReadLine());
region,tkind,quant);
Laba obj=new Laba(//открытие файла с данными на запись
FileStream fstr = new FileStream("e:\\file.txt",
FileMode.OpenOrCreate, FileAccess.Write);
fstr.Seek(0,SeekOrigin.End);
new StreamWriter(fstr);
StreamWriter sw=//запись в файл
sw.WriteLine(obj.region + " " +obj.tkind+ "
"+obj.quant.ToString()+" ");
sw.Close();
fstr.Close();
break;
}
```

Рассмотрим подробнее представленный выше фрагмент кода: здесь пользвателю программы предлагается ввести новые данные, которые будут помещены в файл. С помощью функциий `Console.ReadLine()` мы считываем пользовательский ввод и создаем экземпляр нашего класса с полученными данными.

После этого с помощью строки

```
FileStream fstr = new FileStream("e:\\file.txt", FileMode.OpenOrCreate, FileAccess.Write)
```

мы создаем объект класса `FileStream`, в качетсве параметров передавая ему путь к файлу, в котором хранятся данные, выбираем с помощью метода `FileMode` статус файла: в нашем случае это `OpenOrCreate` (что означает, что данный файл будет открыт, или в случае его отсуствия, будет создан соотвествующий файл), указывем с помощью метода `FileAccess` тип доступа к файлу: `Write`(так как мы плащнируем запись данных в файл). После того, как объект `FileStream` создан мы с помощью функции `Seek(0,SeekOrigin.End)` указываем, что запись данных будет производиться в конец файла.

Далее необходимо создать объект класса `StreamWriter` и в качестве параметра передать созданный нами ранее объект класса `FileStream`. После этого мы можем с помощью метода `WriteLine` класса `StreamWriter` внести необходимые нам данные в файл.

Все отсальные операции реализуются аналогичным образом, а с примером разработанного приложения можно ознакомиться в Приложении 1.

Индивидуальные задания

1. Разработать приложение, реализующее учет продажи телевизоров.
2. Разработать приложение, реализующее учет закупки компьютеров университетом.
3. Разработать приложение, реализующее учет посещаемости студентами лекций.
4. Разработать приложение, реализующее учет покупок мороженного в магазине.
5. Разработать приложение, реализующее учет количества сотрудников в подразделениях предприятия.

Общее требование ко всем лабораторным работам: приложение должно предусматривать хранение исходных данных в файле, должно предусматривать возможность добавления, изменения, удаления просмотра, поиска и сортировки информации.

Вопросы для самопроверки

1. Чем обусловлена упрощенная модель программирования в среде .NET

2. За счет чего обеспечивается отсутствие проблем с версиями программного обеспечения, используемого при разработке приложений на платформе .NET?
3. Каким образом реализуется взаимодействие с существующим кодом?
4. Чем обусловлена мультиплатформенность приложений, создаваемых с помощью платформы .NET?
5. Что такое общеязыковая спецификация?
6. Какие приложения можно создавать с помощью платформы .NET и какие объектно-ориентированные языки при этом используются?
7. Каковы основные компоненты платформы .NET?
8. Какие функции несет в себе .NET Framework?
9. Что такое CLR?
10. Что такое общеязыковая спецификация?