Лабораторная работа 1.

Интегрированная среда разработки Visual Studio.Net

Цель работы: изучение основных возможностей среды разработки на примере простейшей программы на языке Microsoft Visual C#.

Теоретическая часть

Интегрированная среда разработки (integrated development environment, IDE) Visual Studio.NET, теперь едина для всех языков программирования .NET от Microsoft. Таким образом, какой бы тип проекта вы ни создавали (ATL, MFC, C#, Visual Basic.NET, FoxPro, стандартный C++ и т. п.), вы все равно будете работать в одной и той же среде.

При запуске интегрированной среды разработки Visual Studio.NET на экране появляется ее окно (см. рис.1), в котором предлагается выбрать язык программирования. Необходимо выбрать Microsoft Visual C# (C- Шарп).

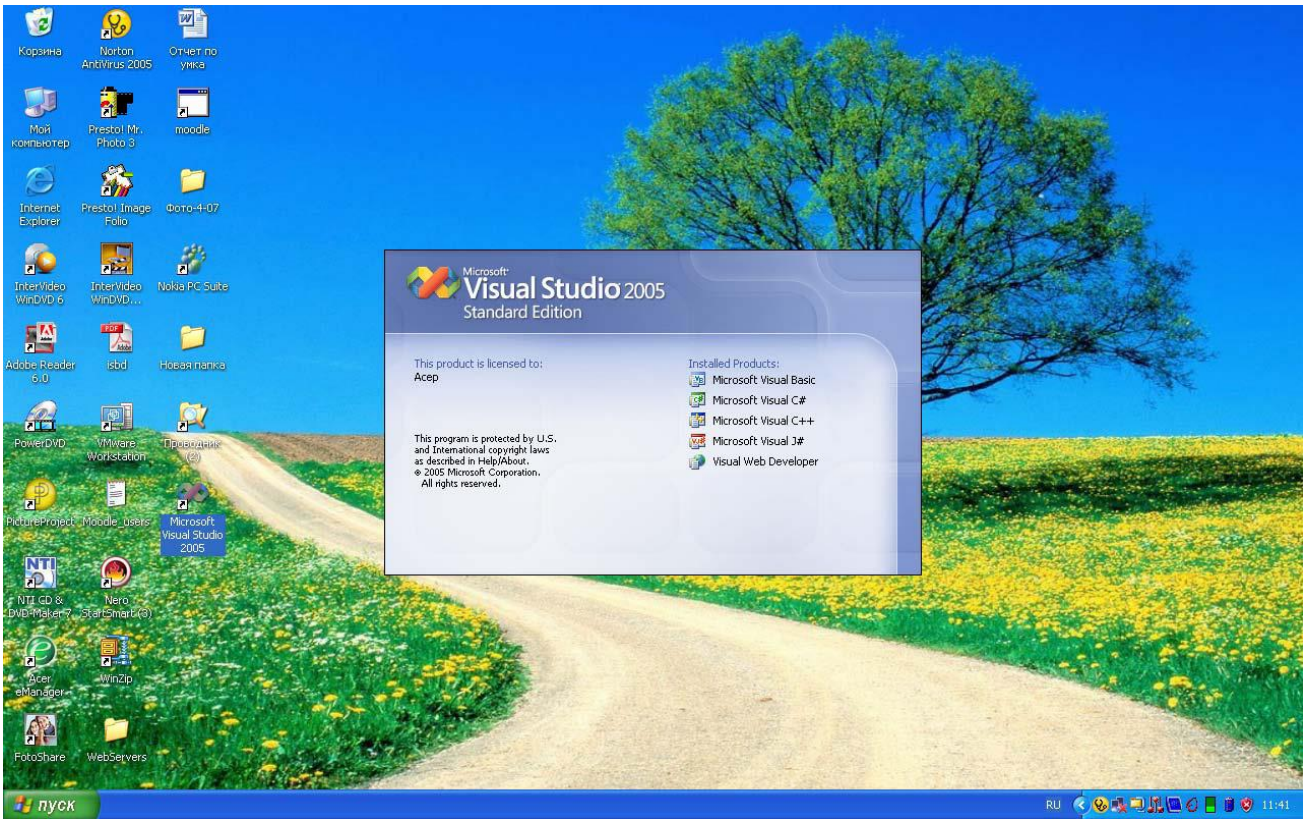


Рис1. Окно **Visual Studio.Net для** выбора языка программирования.

После этого на экране появляется главное меню (рис.2). Основными элементами этого меню являются:

строка заголовка содержит название программы и имя открытого файла, а также кнопки свертывания, восстановления и закрытия;

строка меню , которая представляет доступ ко всем функциям и командам программы;

палитра компонентов она состоит из нескольких закладок, на которых располагаются визуальные компоненты, используемые при создании программ;

инспектор свойств в этом окне производится настройка основных свойств визуальных компонентов;

дизайнер форм содержит название и кнопки управления окном;

проводник решений обеспечивает выбор решения, проекта, исходного файла, ссылок на внешние сборки и прочих ресурсов, которые и образуют приложение.

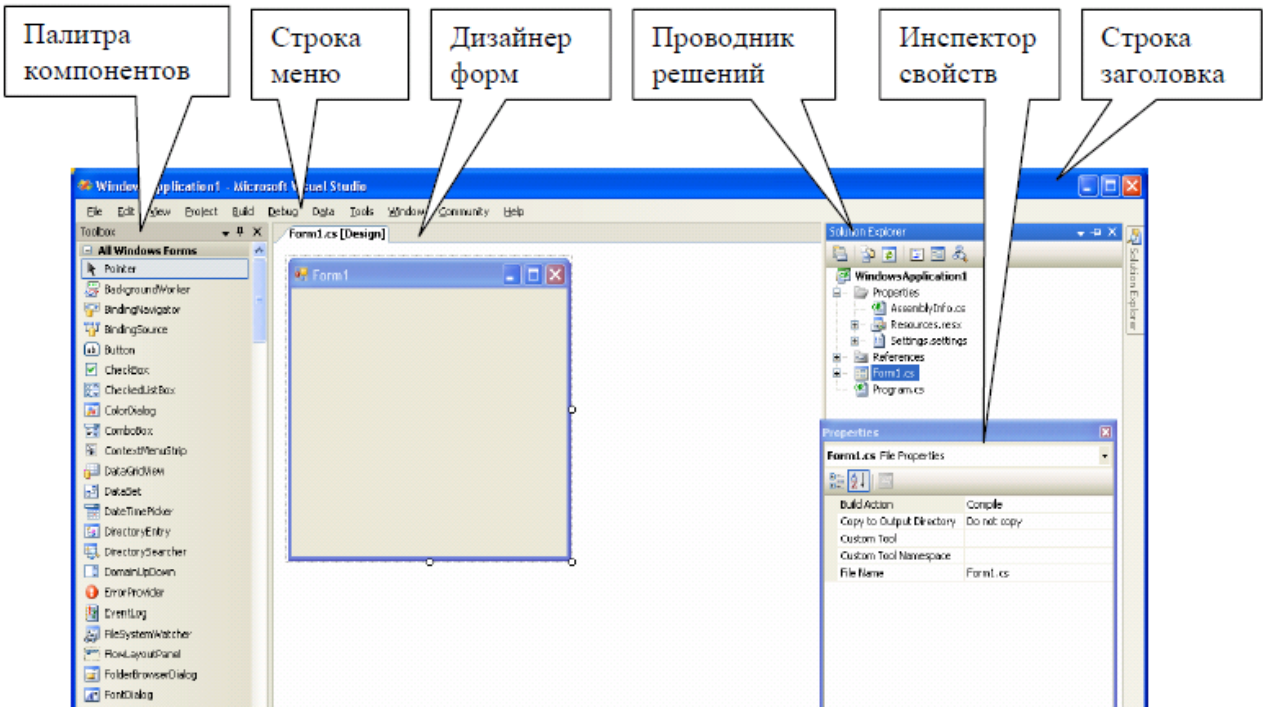


Рис. 2. Главное меню.

Рассмотрим методику создания приложения C# с использованием интегрированной среды разработки Visual Studio.NET на простейшем примере. Пример1. Разработать программу на C# , которая выводит на экран фразу:

« **Моя первая программа на языке C# . Студент Иванов(а) А.А.».**

Для начала необходимо запустить Visual Studio.NET. Это можно сделать с помощью ее ярлыка, расположенного на рабочем столе Вашего компьютера. В ***строке меню*** (рис.2) выбрать **File** и в выпадающем меню выбрать **New**, а затем **Project.** В появившимся окне **New project** (см. рис.2) выбираем необходимый тип проекта. Для нашей задачи выбираем **Console Application** (консольное приложение) в окне **Templates (**шаблоны)и здесь же в окне Name задаем имя создаваемой программы, допустим **Fistprog.**

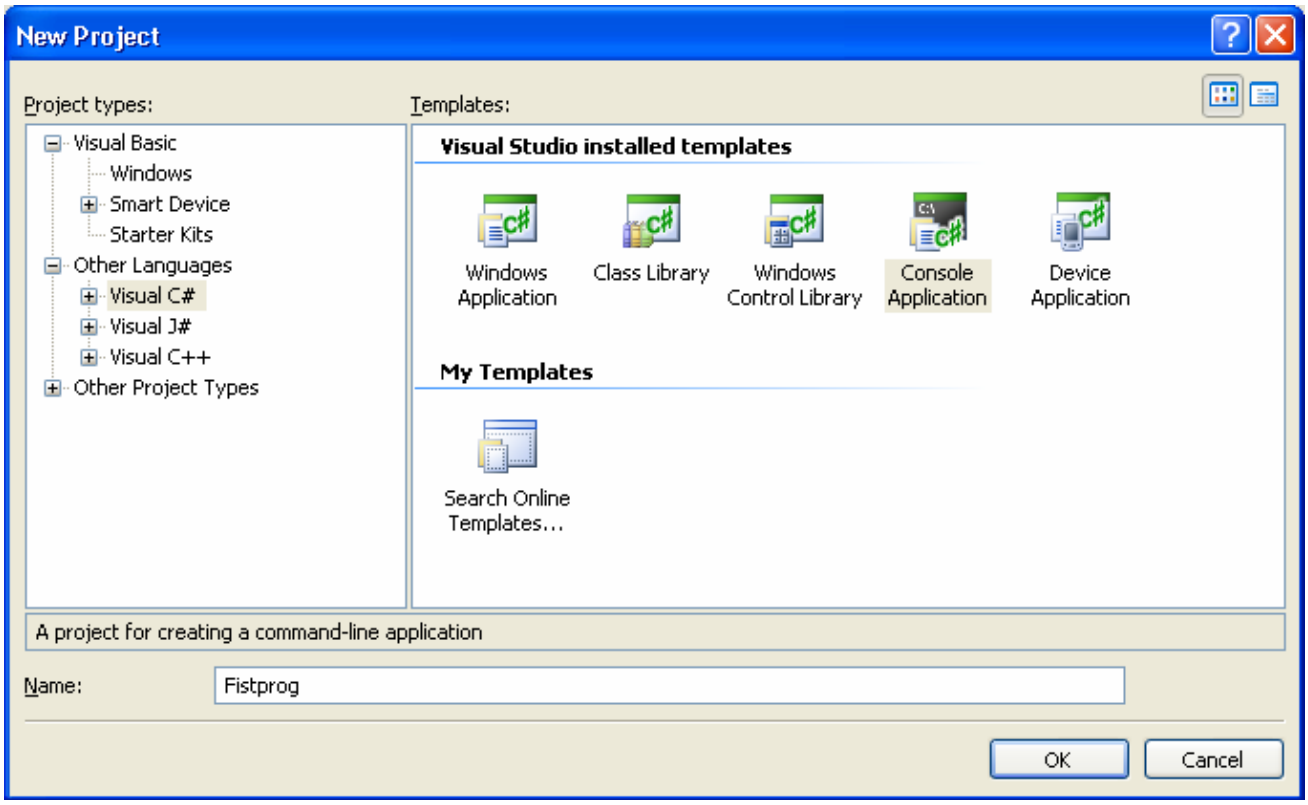


Рис. 3. Создание нового проекта Fistprog.

Нажав кнопку ОК мы увидим новое окно со скелетом программы (см. рис.3).

Обратите внимание, что Visual Studio .NET создает пространство имен, беря за основу название проекта **(Fistprog**), и автоматически добавляет в код оператор**using System** , поскольку почти каждой программе нужны типы из пространства имен **System.**

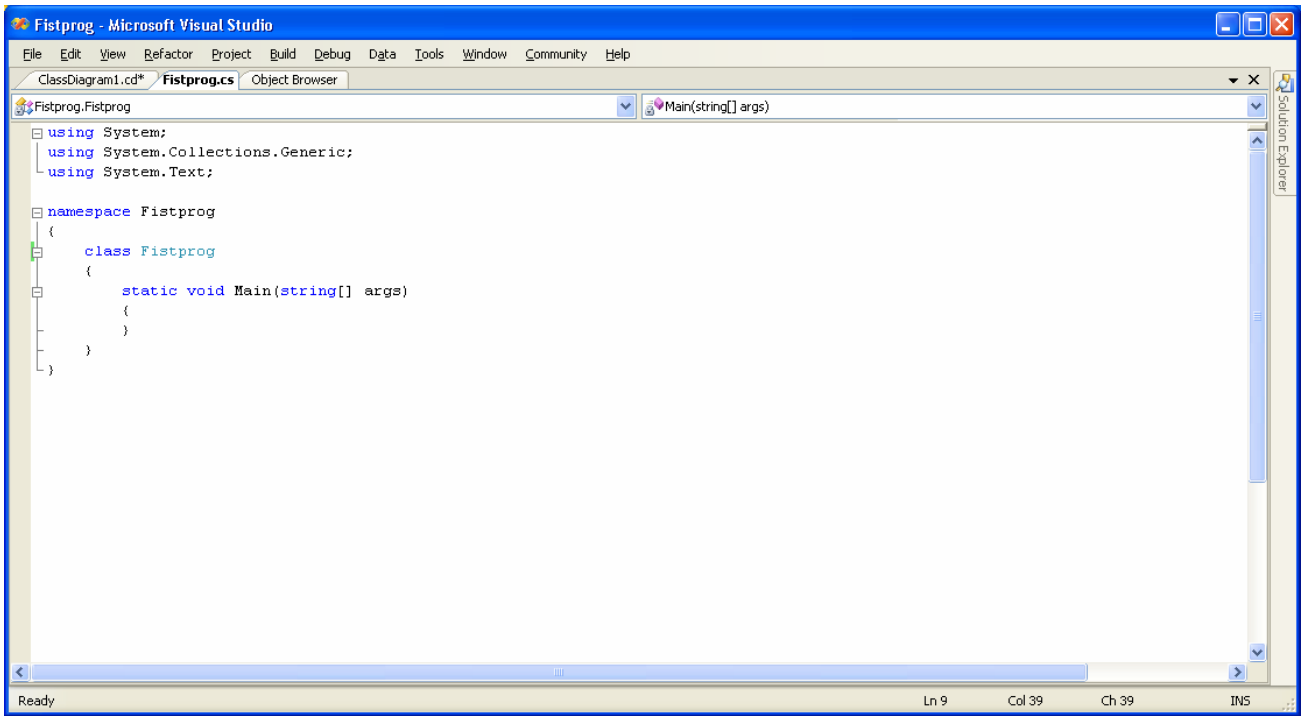


Рис. 4. Окно со скелетом программы.

Visual Studio .NET создает класс с именем ***Classl,*** но его можно переименовать. Дав классу другое имя, не забудьте переименовать и файл ***Classl.cs.*** Файл ***Classl.cs***, указанный в окне Solution Explorer, переименовать в ***Fistprog.cs***.

В интегрированной среде разработки Visual Studio.NET проекты логически организуются в ***решения*** (solutions). Каждое решение состоит из одного или нескольких проектов. В свою очередь, каждый проект может состоять из любого количества исходных файлов, ссылок на внешние сборки и прочих ресурсов, которые и образуют приложение. Вы сможете открыть любой из данных ресурсов с помощью окна Solution Explorer - проводника решений (рис. 4).

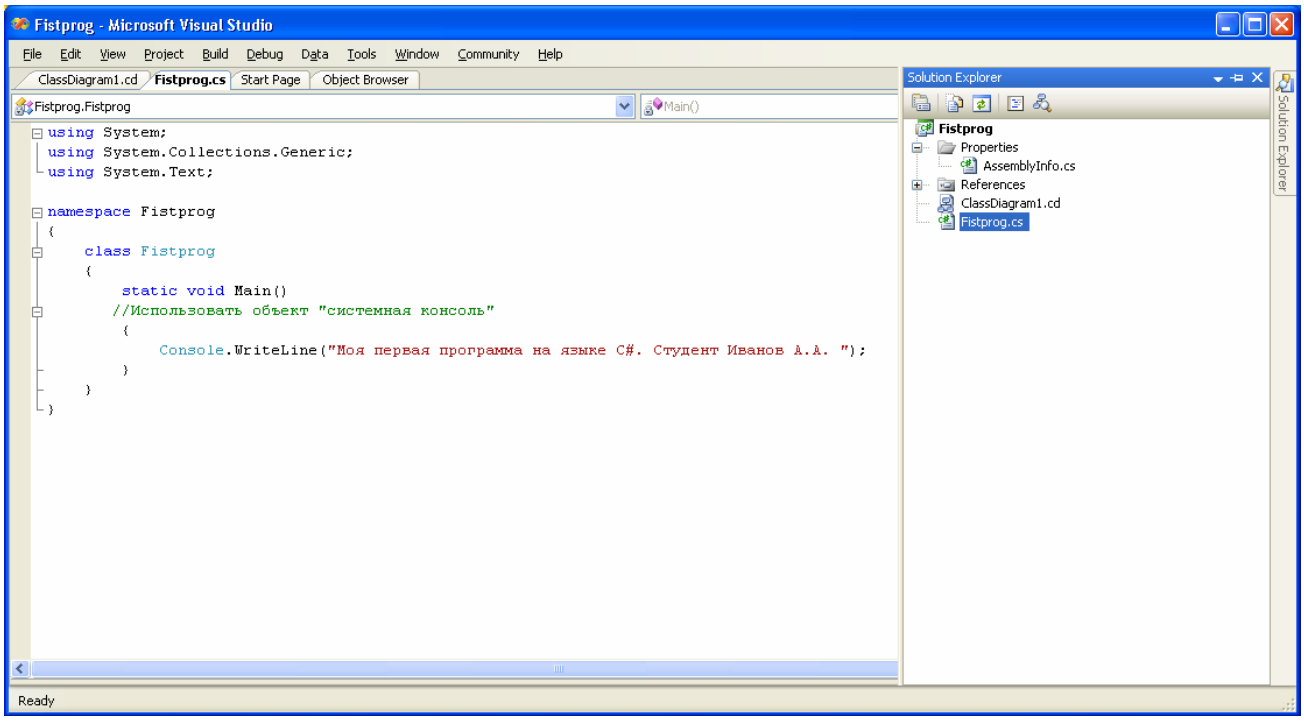


Рис.5. Окно Solution Explorer

При щелчке правой кнопкой мыши на любом элементе в окне Solution Explorer в вашем распоряжении будет контекстное меню (рис. 4), с помощью которого ВЫ сможете воспользоваться любым из множества CASE-средств, например, для добавления в ваш тип новых членов — методов, свойств, полей и т. д.

Другой важный элемент интегрированной среды разработки — это окно Properties (Свойства). В этом окне отображаются важнейшие характеристики выделенного в настоящий момент элемента. Этим элементом может быть и исходный файл, и элемент управления графического интерфейса пользователя, и проект в целом. Например, чтобы изменить имя исходного файла, выберите его в окне Solution Explorer и измените значение свойства FileName в окне Properties (рис. 5).

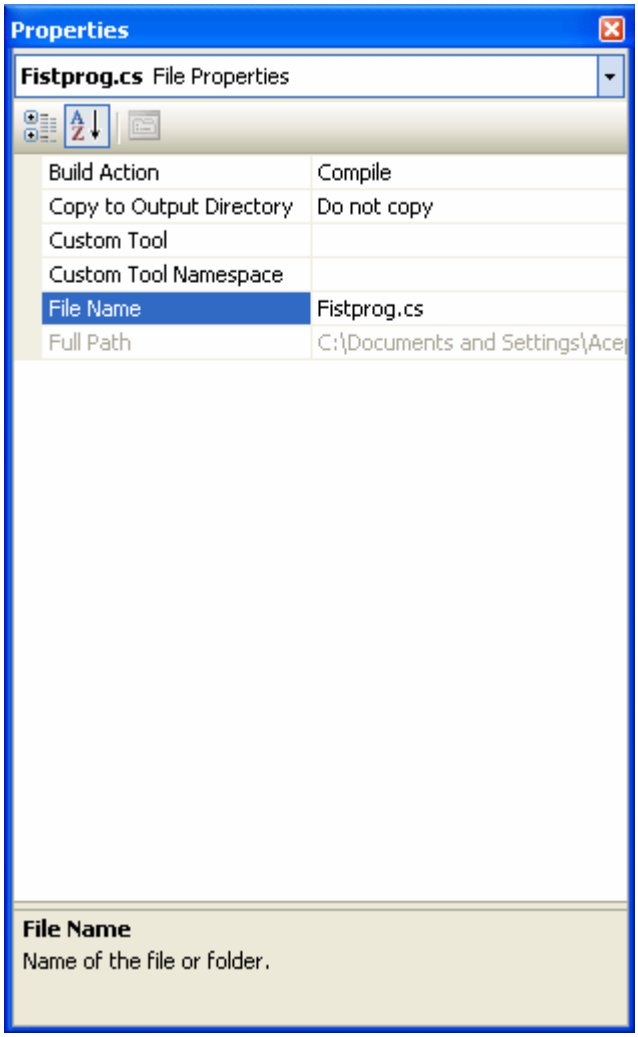


Рис.6. Изменение имени файла с помощью окна инспектора свойств.

Изменение имени класса производится точно так же — просто выберите

нужный класс на вкладке **Class View** и выполните необходимые действия в окне Properties (рис. 5). Обратите внимание, что изменения автоматически будут произведены во всем вашем коде.

Экономное размещение кода на экране - еще одна полезная особенность среды разработки Visual Studio.NET — это возможность при отображении сжимать и разжимать участки программного кода, как при работе с обычной иерархией из дерева и узлов (рис.4). Нажатие на значок «+> приводит к открытию участка программного кода для выбранного вами элемента, нажатие на значок «-» позволяет скрыть данный отрезок кода, освободив место на рабочем столе. При наведении курсора мыши на

многоточие, означающее скрытый для удобства код, данный код будет показан во всплывающем окне.

В среде разработки Visual Studio.NET предусмотрена полная поддержка

технологии**IntelliSense,** «подсказывающей» вам в то время, когда вы набираете код, и предлагающей закончить за вас начатую строку. Это позволяет добавлять проект ссылки не только на внешние сборки .NET, но и на двоичные файлы COM, и на другие проекты.

Написание программы заключается в добавлении к скелету программы необходимого программного кода. Предварительно удалите из скелета метода **Main()** аргументы **(string[ ] arcs)** и комментарий, затем вставьте тело метода следующие две строчки:

// Использовать объект «Системная консоль»

Console.WriteLine (Моя первая программа на языке C# . Студент Иванов А.А.).

Существует несколько способов откомпилировать и выполнить программу **Fistprog** из среды Visual Studio .NET. Обычно программист выполняет каждую задачу отдельно, выбирая команды в меню, щелкая кнопки или зачастую пользуясь комбинациями клавиш.

Например, чтобы откомпилировать программу **Fistprog**, можно нажать

комбинацию клавиш **<Ctrl>+<Shift>+<B>** или выбрать в меню пункт **BuiLd —**

Build Solution. В качестве альтернативы можно щелкнуть по кнопке **Build,** расположенной на одноименной панели инструментов (возможно, потребуется щелкнуть правой кнопкой по любой панели инструментов, чтобы добавить панель инструментов Build).

Чтобы выполнить программу **Fistprog** без отладчика, можно нажать комбинацию<Ctrl>+<F5> на клавиатуре, выбрать в меню пункт Debug —> Start Without Debugging или щелкнуть по кнопке Start Without Debugging на панели инструментов Build.

Для запуска своей программы нажмите одновременно <Ctrl>+<F5> на клавиатуре. Результат работы программы будет выведен в командном окне(см. рис.7)

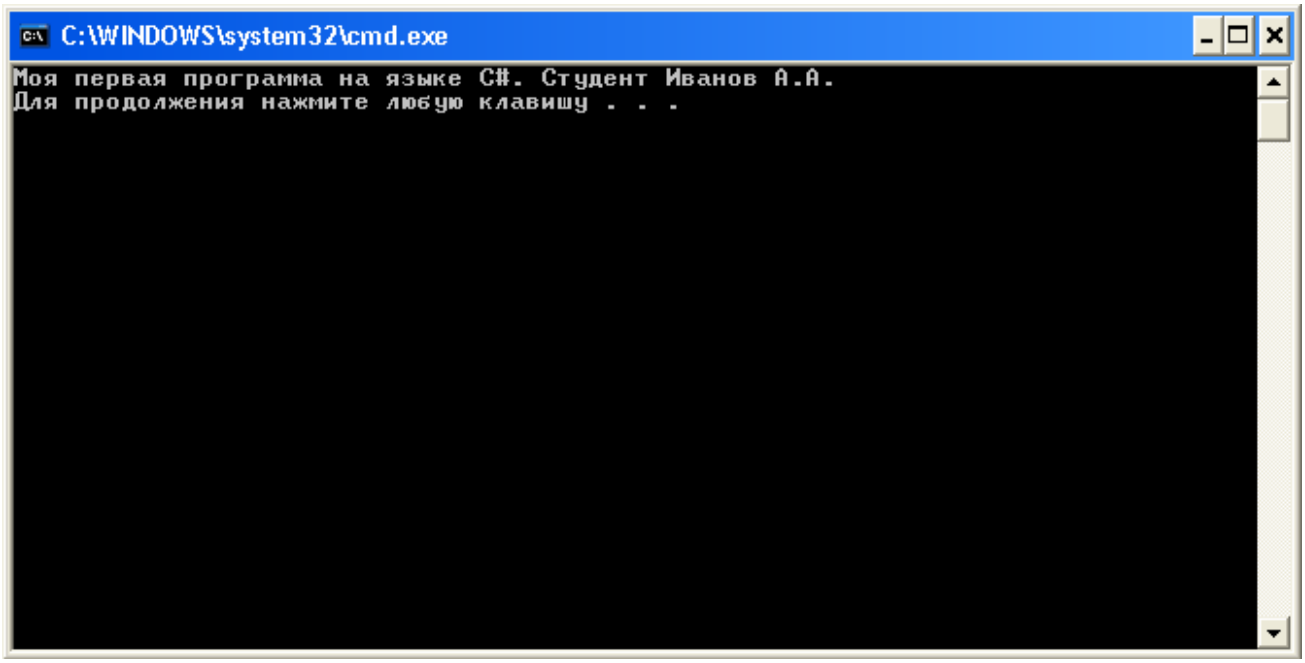


Рис.7. Командное окно.

Сохраните проект в своей папке.

Задания для самостоятельной работы:

Номер варианта	Текст	Имя программы
1	Здравствуй мир	Мир
2	Язык программирования C#	Программа
3	Интегрированная среда разработки	Среда
4	Приложение на языке C#	Приложение
5	Элементы управления Tools	Управление

6	Объектно-ориентированное программирование	Объект
7	Философия .Net	Философия
8	Библиотека базовых классов	Класс
9	Парадигма инкапсуляции	Инкапсуляция
10	Парадигма наследования	Наследования
11	Парадигма полиморфизма	Полиморфизм
12	Инспектор свойств	Инспектор
13	Лабораторная работа	Работа

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 2.

Разработка элементарных программ наязыке программирования C#

Цель работы: познакомиться с базовыми особенностями языка программирования **C#**и разработки программ с использованием среды **Visual Studio.Net.**

Теоретическая часть

Среда разработки **Visual Studio .Net** - это уже проверенный временем программный продукт, являющийся седьмой версией Студии. Но новинки этой версии, связанные с идеей .Net, позволяют считать ее принципиально новой разработкой, определяющей новый этап в создании программных продуктов. Выделю две важнейшие, на мой взгляд, идеи:

- **открытость**для языков программирования**;**
- принципиально новый подход к построению **каркаса среды - Framework.Net**.

Среда разработки теперь является открытой языковой средой. Это означает, что наряду с языками программирования, включенными в среду фирмой Microsoft - **Visual C++ .Net** (с управляемыми расширениями), **Visual C# .Net** , **.Net**, **Visual Basic .Net**, - в среду могут добавляться любые языки программирования, компиляторы которых создаются другими фирмами-
производителями. Таких расширений среды **Visual Studio** сделано уже
достаточно много, практически они существуют для всех известных языков - **Fortran**и **Cobol**, **RPG**и**Component Pascal**, **Oberon**и **SmallTalk**. Я у себя на компьютере включил в среду компилятор одного из лучших объектных
языков - языка **Eiffel**.

Открытостьсреды не означает полной свободы. Все разработчики

компиляторов при включении нового языка в среду разработки должны следовать определенным ограничениям. Главное ограничение, которое можно считать и главным достоинством, состоит в том, что все языки, включаемые в среду разработки *Visual Studio .Net* , должны использовать

единый *каркас - Framework .Net* . Благодаря этому достигаются многие

желательные свойства: легкость использования компонентов, разработанных на различных языках; возможность разработки нескольких частей одного приложения на разных языках; возможность бесшовной отладки такого приложения; возможность написать класс на одном языке, а его потомков - на других языках. Единый каркас приводит к сближению языков программирования, позволяя вместе с тем сохранять их индивидуальность и имеющиеся у них достоинства. Преодоление языкового барьера - одна из важнейших задач современного мира. Благодаря единому каркасу, *Vis*

ual

Studio .Net в определенной мере решает эту задачу в мире программистов.

Framework .Net - единый каркас среды разработки

В *каркасе Framework .Net* можно выделить два основных компонента:

- статический - **FCL (Framework Class Library)**- библиотеку классов каркаса;
- динамический - ***CLR (CommonLanguageRuntime)*** - общеязыковую исполнительную среду.

Библиотека классов FCL - статический компонент каркаса

Понятие каркаса приложений - **Framework Applications** - появилось достаточно давно; по крайней мере оно широко использовалось еще в

четвертой версии Visual Studio. Важна роль библиотеки классов **MFC**

(MicrosoftFoundationClasses)каккаркасаприложений**VisualC.**

Несмотря на то, что каркас, первоначально был представлен только статическим компонентом, уже тогда была очевидна его роль в построении приложений.

Уже в то время важнейшее значение в библиотеке классов **MFC** имели классы, задающие архитектуру строящихся приложений. Когда разработчик выбирал один из возможных типов приложения, например, архитектуру **Document-View**, то в его приложение автоматически встраивались класс **Document**, задающий структуру документа, и класс **View**, задающий его визуальное представление. Класс **Form**и классы, задающие элементы управления, обеспечивали единый интерфейс приложений. Выбирая тип приложения, разработчик изначально получал нужную ему функциональность, поддерживаемую классами каркаса. Библиотека классов поддерживала и более традиционные для программистов классы, задающие расширенную систему типов данных, в частности, динамические типы данных - списки, деревья, коллекции, шаблоны.

Основа языка C#

Объявление и инициализация переменных:

Тип_переменнойимя_переменной[=значение];

Примеры:

```
int x; //обьявление переменной x
x=100; //инициализация переменной x
long w,z=100; //обьявление переменных w и z и
//инициализация z
long q=100*z; //обьявление переменной с динамической
//инициализацией
```

C - язык со строгим контролем типов данных. Есть 2 основные категории встроенных типов данных в C# - простые типы и ссылочные типы. Основные простые типы данных в C#:

Тип	Описание	Бит

bool	Значение истина/ложь	1
byte	8-битовое беззнаковое целое	8
char	символ	16
decimal	Числовой тип для финансовых вычислений	128
double	Число двойной точности с плавающей точкой	64
float	Число с плавающей точкой	32
int	Знаковое целое	32
long	Длинное знаковое целое	64
Sbyte	8-битовое знаковое целое	8
short	Короткое целое	16
uint	Беззнаковое целое	32
ulong	Беззнаковое длинное целое	64
ushort	Беззнаковое короткое целое	16

Область видимости переменной в C# - блок кода (заключенный в фигурные скобки {}). Переменная создается при входе в область видимости и уничтожаются при выходе из нее.

Основные управляющие операторы:

Условный:

if (*условие*) *оператор* [**else***оператор*];

if (*условие1*) *оператор1;*

else if (*условие2*) *оператор2;*

else if (*условие3*) *оператор3;*

…

Выбора:

switch (*выражение*){

case*константа1*:

оператор1;

…

break;

case*константа2*:

операторX1;

…

break;

…

default :

операторZ1 ;

…

break;

}

Цикла:

for(*инициализация*, *условие_выхода*, *итерация*) *оператор*;

while (*условие_продолжения*) *оператор*;

do*оператор;***while** (*условие продолжения*);

Пространство имен:

Пространство имен определяет область объявления, что позволяет хранить каждый набор имен отдельно от других наборов. В C# имена, объявленные в одном пространстве имен, не конфликтуют с такими же именами, объявленными в другом пространстве имен. Библиотекой .NET Framework (библиотекой C#) используется пространство имен System.

Для того, чтобы сделать видимыми пространства имен без указания полного имени (через ‘.’) используется директива using

Синтаксис:

using*имя_пространства_имен* ;

также возможно использование псевдонимов для имен

using*псевдоним = имя*;

Пространство имен объявляется с помощью ключевого слова namespace.

Синтаксис:

namespace*имя* {

члены_пространства_имен}

Пример программы на C#

Программа ‘Hello, World’ на языке C# выглядит следующим образом:

```
using System;
class HelloWorld01
{
public staticvoid Main()
{
Console.Write("Hello, World!");
Console.ReadLine();
}
}
```

Задания для самостоятельной работы:

Написать C# программу, реализующую функцию согласно варианту задания. Исходные данные вводятся с клавиатуры.

1. Реализовать функцию вычисления суммы двух целых чисел
2. Реализовать функцию вычисления разности двух целых чисел
3. Реализовать функцию вычисления произведения двух целых чисел
4. Реализовать функцию вычисления частного двух целых чисел
5. Реализовать функцию вычисления суммы двух вещественных чисел
6. Реализовать функцию вычисления разности двух вещественных чисел
7. Реализовать функцию вычисления произведения двух вещественных чисел
8. Реализовать функцию вычисления частного двух вещественных

чисел

9. Реализовать функцию возведения целого числа в квадрат

10. Реализовать функцию возведения в квадрат суммы двух целых чисел

11. Реализовать функцию возведения в квадрат разности двух целых чисел

12. Реализовать функцию возведения в квадрат произведения двух целых чисел

13. Реализовать функцию возведения в квадрат частного двух целых чисел

14. Реализовать функцию возведения в квадрат суммы двух вещественных чисел

15. Реализовать функцию возведения в квадрат разности двух вещественных чисел

16. Реализовать функцию возведения в квадрат произведения двух вещественных чисел

17. Реализовать функцию возведения в квадрат частного двух вещественных чисел

18. Реализовать функцию возведения в куб целого числа

19. Реализовать функцию возведения в куб суммы двух целых чисел

20. Реализовать функцию возведения в куб разности двух целых чисел

21. Реализовать функцию возведения в куб произведения двух целых чисел

22. Реализовать функцию возведения в куб частного двух целых чисел

23. Реализовать функцию возведения в куб суммы двух вещественных чисел

24. Реализовать функцию возведения в куб разности двух

вещественных чисел

25. Реализовать функцию возведения в куб произведения двух вещественных чисел

26. Реализовать функцию возведения в куб частного двух вещественных чисел

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.

3. Ответов на контрольные вопросы лабораторной работы.

4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 3.
Массивы и операторы циклов for, foreach, while и do-whilев языке программирования C#

Цель работы: изучить способы задания и инициализации массивов и работу операторов циклов **for, foreach, while** и **do-while** вязыке программирования C#

Теоретическая часть

Массивы в C#

Массивы в C# несколько отличаются от других C-подобных языков. Начнем сразу с примеров.

Пример первый:

```
...
int[] k; //k - массив
newint [3]; //Определяем массив из 3-х целых
k[0]=-5; k[1]=4; k[2]=55; //Задаем элементы массива
//Выводим третий элемент массива
Console.WriteLine(k[2].ToString());
...
```

Смысл приведенного фрагмента ясен из комментариев. Обратите внимание на некоторые особенности. Во-первых, массив определяется именно как

```
int[] k;
```

а не как один из следующих вариантов:

```
int k[]; //Неверно!
int k[3]; //Неверно!
int[3] k; //Неверно!
```

Во-вторых, так как массив представляется как ссылочный объект, то

для создания массива необходима строка

```
newint [3];
```

Именно в ней мы и определяем размер массива. Хотя, вообще говоря, возможны конструкции вида

```
int[] k =   newint [3];
```

Элементы массива можно задавать сразу при объявлении. Вот пример:

```
int[] k = {-5, 4, 55};
```

Разумеется, приведенные конструкции применимы не только к типу **int** и не только к массиву размера 3.

В C#, как и в C/C++, нумерация элементов массива идет с нуля. Таким образом, в нашем примере начальный элемент массива - это k[0], а последний - k[2]. Элемента k[3], разумеется, нет.

Теперь переходим к многомерным массивам. Вот так задается двумерный массив:

```
int[,] k = newint [2,3];
```

Обратите внимание, что пара квадратных скобок только одна. Естественно, что в нашем примере у массива 6 (=2*3) элементов (k[0,0] - первый, k[1,2] - последний).

Аналогично мы можем задавать многомерные массивы. Вот пример трехмерного массива:

```
int[,,] k = newint [10,10,10];
```

А вот так можно сразу инициализировать многомерные массивы:

```
int[,] k = {{2,-2},{3,-22},{0,4}};
```

Приведенные выше примеры многомерных массивов называются прямо угольными. Если их представить в виде таблицы (в двумерном случае), то массив будет представляться в виде прямоугольника.

Наряду с прямоугольными массивами существуют так называемые ступенчатые. Вот пример:

```
//Объявляем 2-мерный ступенчатый массив
int[][] k = newint [2][];
```

```
//Объявляем 0-й элемент нашего ступенчатого массива
//Это опять массив и в нем 3 элемента
k[0]=new    int[3];
//Объявляем 1-й элемент нашего ступенчатого массива
//Это опять массив и в нем 4 элемента
k[1]=new    int[4];
k[1][3]=22; //записываем 22 в последний элемент массива
...
```

Обратите внимание, что у ступенчатых массивов мы задаем несколько пар квадратных скобок (столько, сколько размерность у массива). И точно так же мы что-нибудь делаем с элементами массива - записываем, читаем и т. п.

Самая важная и интересная возможность у ступенчатых массивов - это их "непрямоугольность". Так, в приведенном выше примере в первой "строке" массива **k** три целых числа, а во второй - четыре. Часто это оказывается очень к месту.

Операторы циклов C#

Оператор цикла **for**

for(*инициализация* , *условие_выхода*, *итерация*) *оператор*;

Начнем сразу с примера цикла for:

```
int k = Int32.Parse(Console.ReadLine());
int sum=0;
for(int i=1; i<=k; i++){
sum+=i;
}
Console.WriteLine(sum);
```

Этот пример подсчитывает сумму чисел от 1 до введенного пользователем числа **k**. Сумма записывается в переменную **sum** и выводится на экран.

Очень часто циклы используются для некоторых действий с

массивами. Так как нумерация элементов массива идет с нуля, то типичный цикл будет выглядеть так:

```
int[] a = {-5, 4, 55};
int sum=0;
for(int i=0; i&lt;3; i++){
sum+=a[i];
}
```

В этом примере начальное значение для счетчика цикла равно нулю, и в условии продолжения цикла мы пишем знак "меньше", после которого ставится количество элементов в массиве. Разумеется, если в цикле должен выполнится только один оператор, то фигурные скобки можно не писать. Тут все, как в других C/C++-подобных языках.

Циклaforeach:

```
int[] m = {-5, 4, 10};
int sum=0;
foreach(int i in m){
sum+=i;
}
```

В данном примере мы суммируем все элементы массива **m,** записывая сумму в **sum** .

В приведенном примере наш цикл перебирает все элементы массива **m**.

На это нам указывает строка

```
...
foreach(int i in m){
...
```

которая интерпретируется так: для каждого целого числа из массива **m** делаем что-то там. Если бы элементами массива были бы не целые, а, скажем, вещественные, то мы записали бы что-то вроде:

```
...
foreach(float i in m){
...
```

т. е. мы пишем именно тип элементов массива.

Цикл while

Циклы **while** бывают двух видов - собственно цикл **while** и **do-while**.

while (*условие_продолжения*) *оператор*;

do*оператор;***while** (*условие продолжения*);

Оба эти цикла используются, как правило, тогда, когда точно не известно, сколько раз цикл должен выполнится. Например, при вводе пользователем пароля или при подсчете чего-либо с определенной точностью. Оба эти цикла будут выполняться до тех пор, пока условие в круглых скобках после слова **while** будет истинно. Как только условие станет равным **false,** выполнение цикла прекращается. Самое важное отличие между **while** и **do-while** в том, что **while** может не выполниться ни одного раза, тогда как **do-while**по крайней мере один раз выполнится. Вот примеры их использования:

```
string password;
do{
password=Console.ReadLine();
}
while(password!="wi98zK");
int k=0; //Количество попыток
//заводим новую последовательность случайных чисел
Random rnd=  newRandom(112);  //Пишемлюбойпараметр
while(rnd.Next(1, 6)!=5)
{
k++;
};
```

Console.WriteLine("C "+(k+1).ToString()+"-горазавыпало 5");

В первом примере цикл будет вращаться до тех пор, пока пользователь не введет правильный пароль (wi98zK), во втором - пока некоторое случайное число не окажется равным 5. При этом если число с самого начала оказалось равным пяти, то цикл вообще выполняться не будет.

Задания для самостоятельной работы

1. Напишите программу вычисления корней квадратного уравнения (Y = 4X2 +2X +2).
2. Напишите программу подсчёта факториала числа 100.
3. Найти среднее арифметическое N чисел (23,12,11,21,32,27).
4. Вычислить Z = Xk , используя операции умножения, т.е. Z = X*X*X*X… (X=3, k=5).
5. Вычислить F = m*(m + 1)*(m + 2)*…*n, где m=12, n=15.
6. Задать два числа X и Y, переменной M присвоить значение
1. если X < Y, и ?1 , если X >=Y (X=12, Y=34).
7. Заданы два числа K и N, переменной присвоить значение 1, если K < N, 0, если K = N, и –1, если K > N (K=213.01, N=43.76).
8. Задайте число P, если P < 0,то значение Y вычислить по формуле Y = P^2, если P > 0, то Y = P^3 (P=-21).
9. Задайте числа P, Y, X, если P < 0, то T = X + Y, если P >= 0, то T = X – Y (P=0.9, Y=6.7, X=5.5)
10. Найти минимальное из N чисел (Получить N чисел с помощью **Random**).
11. Найти максимальное из N чисел (Получить N чисел с помощью **Random**).
12. Ввести N чисел и определить сколько среди них положительных, сколько отрицательных (N={1,2,3,3,4, 5,6,7, -8,-9,-6}).

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше,

оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 4.
Объекты и классы (наследование, конструкторы, деструкторы).

Цель работы: познакомиться с основой объектного подхода в языке C#, созданием объектов, классов и механизмом наследования.

Теоретическая часть

Классы и объекты

Класс является основой для создания объектов. В классе определяются данные и код, который работает с этими данными. Объекты являются экземплярами класса.

Методы и переменные, составляющие класс, называются членами класса. При определении класса объявляются данные, которые он содержит, и код, работающий с этими данными. Данные содержатся в переменных экземпляра, которые определены классом, а код содержится в методах. В C# определены несколько специфических разновидностей членов класса. Это — переменные экземпляра, статические переменные, константы, методы, конструкторы, деструкторы, индексаторы, события, операторы и свойства. Непосредственно инициализация переменных в объекте (переменных экземпляра) происходит в конструкторе. В классе могут быть определены несколько конструкторов.

Синтаксис класса:

class*имя_класса*{

тип_доступатипимя_переменной1;

тип_доступатипимя_переменной2;

…

тип_доступа возвращаемый_тип

имя_метода1(*список_параметров*)

тело_метода}

}

гдетип_доступаможетбытьpublic, private, protected, internal. Члены класса с

типом доступа public доступны везде за пределами данного класса, с типом доступа protected – внутри членов данного класса и производных, с типом доступа private - только для других членов данного класса. Тип доступа internal применяется для типов, доступных в пределах одной сборки.

Пример:

```
class Animal{
public string Name;
private int Weight; protected int Type;
public int Animal(int W, int T, string N){
Weight=W;
Type=T;
Name=N;
}
public int GetWeight(){return Weight;}
}
```

Создание обьекта

имя_класса имя_обьекта = **new***имя_класса*();

При создании обьекта класса происходит вызов соответствующего конструктора класса.

Конструктор и деструктор

Конструктор класса – метод для инициализации объекта при его создании. Он имеет то же имя, что и его класс. В конструкторах тип возвращаемого значения не указывается явно. Конструкторы используются для присваивания начальных значений переменным экземпляра, определенным классом, и для выполнения любых других процедур инициализации, необходимых для создания объекта.

Все классы имеют конструкторы независимо от того, определен он или нет. По умолчанию в C# предусмотрено наличие конструктора, который присваивает нулевые значения всем переменным экземпляра (для переменных обычных типов) и значения null (для переменных ссылочного

типа). Но если конструктор явно определен в классе, то конструктор по умолчанию использоваться не будет.

имя_класса (*список_параметров*) {*тело_конструктора* }

Деструктор – метод, вызывающийся автоматически при уничтожении объекта метка класса (непосредственно перед “сборкой мусора”). Деструктор не имеет параметров и возвращаемого значения.

~ *имя_класса*() {*тело_деструктора* }

Наследование

Наследование — это свойство, с помощью которого один объект может приобретать свойства другого. При этом поддерживается концепция иерархической классификации, имеющей направление сверху вниз. Используя наследование, объект должен определить только те качества, которые делают его уникальным в пределах своего класса. Он может наследовать общие атрибуты от своих родительских классов.

Синтаксис:

class*имя_класса* : *имя_родительского_класса* {*тело_класса*}

Пример:

```
class Predator:Animal{
private int Speed;
}
```

С помощью наследования создается иерархия классов (отношение ‘являться’). Кроме того, можно построить еще одну структуру – иерархию объектов (тогда, когда один объект является частью другого – отношение ‘часть-целое’).

Задания для самостоятельной работы:

Построить иерархию классов в соответствии с вариантом задания:

1. Студент, преподаватель, персона, заведующий кафедрой.
2. Служащий, персона, рабочий, инженер.
3. Рабочий, кадры, инженер, администрация.

4. Деталь, механизм, изделие, узел.
5. Организация, страховая компания, нефтегазовая компания, завод.
6. Журнал, книга, печатное издание, учебник.
7. Тест, экзамен, выпускной экзамен, испытание.
8. Место, область, город, мегаполис.
9. Игрушка, продукт, товар, молочный продукт.
10. Квитанция, накладная, документ, счет.
11. Автомобиль, поезд, транспортное средство, экспресс.
12. Двигатель, двигатель внутреннего сгорания, дизель, реактивный двигатель.
13. Республика, монархия, королевство, государство.
14. Млекопитающее, парнокопытное, птица, животное.
15. Корабль, пароход, парусник, кров

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно

содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 5.
Программирование полиморфных методов.

Цель работы: Познакомиться с программированием полиморфных методов при объектно-ориентированном подходе при использовании языка C#.

Теоретическая часть

Полиморфизм

Полиморфизм – одна из основных составляющих объектно-ориентированного программирования, позволяющая определять в наследуемом классе методы, которые будут общими для всех наследующих классов, при этом наследующий класс может определять специфическую реализацию некоторых или всех этих методов. Главный принцип полиморфизма: «один интерфейс, несколько методов». Благодаря ему, можно пользоваться методами, не обладая точными знаниями о типе объектов. Основным инструментом для реализации принципа полиморфизма является использование виртуальных методы и абстрактных классов.

Виртуальные методы

Метод, при определении которого в наследуемом классе было указано ключевое слово `virtual`, и который был переопределен в одном или более наследующих классах, называется виртуальным методом. Следовательно, каждый наследующий класс может иметь собственную версию виртуального метода.

Выбор версии виртуального метода, которую требуется вызвать, осуществляется в соответствии с типом объекта, на который ссылается ссылочная переменная, во время выполнения программы. Другими словами, именно тип объекта, на который указывает ссылка (а не тип ссылочной переменной), определяет вызываемую версию виртуального метода. Таким образом, если класс содержит виртуальный метод и от этого класса были наследованы другие классы, в которых определены свои версии метода, при ссылке переменной типа наследуемого класса на различные типы объектов вызываются различные версии виртуального метода.

При определении виртуального метода в составе наследуемого класса

перед типом возвращаемого значения указывается ключевое слово `virtual`, а при переопределении виртуального метода в наследующем классе используется модификатор `override`. Виртуальный метод не может быть определен с модификатором `static` или `abstract`.

Переопределять виртуальный метод не обязательно. Если наследующий класс не предоставляет собственную версию виртуального метода, то используется метод наследуемого класса.

Переопределение метода положено в основу концепции динамического выбора вызываемого метода - выбора вызываемого переопределенного метода осуществляется во время выполнения программы, а не во время компиляции.

Синтаксис:

virtual*типимя* (*список_параметров*){*тело_метода*};

Пример:

Абстрактные классы

В абстрактном классе определяются лишь общие предназначения методов, которые должны быть реализованы в наследующих классах, но сам по себе этот класс не реализует один, или несколько подобных методов, называемых абстрактными (для них определены только некоторые характеристики, такие как тип возвращаемого значения, имя и список параметров).

При объявлении абстрактного метода используется модификатор `abstract`. Абстрактный метод автоматически становится виртуальным, так что модификатор `virtual` при объявлении метода не используется.

Абстрактный класс предназначен только для создания иерархии классов, нельзя создать объект абстрактного класса.

Пример:

```
abstractclass Animal
{
publicstring Name;
protectedint Weight;
privateint Type;
```

```
abstractvoid Feed();
publicint Animal(int W, int T, string N)
{
Weight=W;
Type=T; Name=N;
}
publicint GetWeight(){return Weight;}
}
class Predator:Animal
{
privateint Speed;
overridevoid Feed(int Food){
Weight += Food;}
}
```

Задания для самостоятельной работы:

Расширить иерархию классов из лабораторной работы №2 с использованием виртуального класса в качестве основы иерархии. Показать пример использования полиморфизма методов.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом

обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 6.
Расширенные возможности языка программирования C#. Архитектура WindowsForm

Цель работы:познакомиться с расширенными возможностями языка программирования C#, такими, как интерфейсы и делегаты.

Теоретическая часть

Интерфейсы

В C# для полного отделения структуры класса от его реализации используется механизм интерфейсов.

Интерфейс является расширением идеи абстрактных классов и методов. Синтаксис интерфейсов подобен синтаксису абстрактных классов. Объявление интерфейсов выполняется с помощью ключевого слова interface. При этом методы в интерфейсе не поддерживают реализацию.

Членами интерфейса могут быть методы, свойства, индексаторы и события.

Интерфейс может реализовываться произвольным количеством классов. Один класс, в свою очередь, может реализовывать любое число интерфейсов. Каждый класс, включающий интерфейс, должен реализовывать его методы. В интерфейсе для методов неявным образом задается тип public. В этом случае также не допускается явный спецификатор доступа.

Синтаксис:

[атрибуты] [*модификаторы*] **interface**

Имя_интерфейса [**:***список_родительских_интерфейсов*] **{**

*обьявление_свойств_и_методов***}**

Пример:

```
interface Species
{
string Species();
void Feed();
}
class Cheetah:Animal,Species{
```

```
privatestring ScientificName;
publicstring Species()
   return ScientificName;}
publicvoid Feed()
{Weight++;}
}
```

Можно объявлять ссылочную переменную, имеющую интерфейсный тип. Подобная переменная может ссылаться на любой объект, который реализует ее интерфейс. При вызове метода объекта с помощью интерфейсной ссылки вызывается версия метода, реализуемого данным объектом.

Возможно наследование интерфейсов. В этом случае используется синтаксис, аналогичный наследованию классов. Если класс реализует интерфейс, который наследует другой интерфейс, должна обеспечиваться реализация для всех членов, определенных в составе цепи наследования интерфейсов.

Делегаты

Делегат — это объект, имеющий ссылку на метод. Делегат позволяет выбрать вызываемый метод во время выполнения программы. Фактически значение делегата – это адрес области памяти, где находится точка входа метода.

Важным свойством делегата является то, что он позволяет указать в коде программы вызов метода, но фактически вызываемый метод определяется во время работы программы, а не во время компилирования.

Делегат объявляется с помощью ключевого слова delegate, за которым указывается тип возвращаемого значения, имя делегата и список параметров вызываемых методов.

Синтаксис:

delegate *тип_возвращаемого_значения имя_делегата* (*список_параметров*);

Характерной особенностью делегата является возможность его использования для вызова любого метода, который соответствует подписи

делегата. Это дает возможность определить во время выполнения программы, какой из методов должен быть вызван. Вызываемый метод может быть методом экземпляра, ассоциированным с объектом, либо статическим методом, ассоциированным с классом. Метод можно вызвать только тогда, когда его подпись соответствует подписи делегата.

Многоадресность делегатов

Многоадресность — это способность делегата хранить несколько ссылок на различные методы, что позволяет при вызове делегата инициировать эту цепочку методов.

Для создания цепочки методов необходимо создать экземпляр делегата, и пользуясь операторами + или += добавлять методы к цепочке. Для удаления метода из цепочки используется оператор - или -=. Делегаты, хранящие несколько ссылок, должны иметь тип возвращаемого значения void.

Задания для самостоятельной работы

Реализовать для иерархии из лабораторной работы №5 механизм интерфейсов, при этом один из классов должен реализовывать как минимум 2 интерфейса. Использовать для проверки всех методов данного класса многоадресный делегат.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом

обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работая 7.
Работа с элементами WINDOWSFORMS.Модальные диалоговые окна.

Цель работы: Научиться создавать формы верхнего уровня и модальные диалоговые окна.

Теоретическая часть

Пространство имен System.Windows.Forms. Основные компоненты пространства имен System.Windows.Forms:

1. System.Windows.Forms компонуется из различных классов, структур, делегатов, интерфейсов и перечней. Сотни типов пространства имен System.Windows.Forms можно объединить в следующие большие категорий.
2. Базовая инфраструктура. Это типы, представляющие базовые операции программы .NET Forms (Form, Application и т.д.), а также различные типы, обеспечивающие совместимость с разработанными ранее элементами управления ActiveX.
3. Элементы управления. Все типы, используемые для создания пользовательскогоинтерфейса (Button, MenuStrip, ProgressBar, DataGridView ит.д.), являются производными базового класса Control.
4. Компоненты. Это типы, не являющиеся производными базового класса Control, но тоже предлагающие визуальные инструменты (ToolTip, ErrorProvider и т.д.) для программ .NET Forms, Многие компоненты (например, Timer) во время выполнения не видимы, но они могут конфигурироваться визуально в режиме проектирования.
5. Диалоговые окна общего вида. Среда Windows Forms предлагает целый ряд стандартных заготовок диалоговых окон для выполнения типичных действий (OpenFileDialog, PrintDialog и т.д.).

Общее число типов в System.Windows.Forms намного больше 100. В таблице 2.1 описаны наиболее важные из типов System.Windows.Forms, предлагаемых в .NET 2.0.

Таблица 2.1 – Типы пространства имен System.Windows.Forms

Классы	Описание
Application	Класс, инкапсулирующий средства поддержки WindowsForms, необходимые любому приложению.
Button, CheckBox, ComboBox, DateTimePicker, ListBox, LinkLabel, MaskedTextBox, MonthCalendar, PictureBox, TreeView	Классы, которые (вместе со многими другими классами) определяют различные GUI-элементы
FlowLayoutPanel, TableLayoutPanel	Платформа .NET 2.0 предлагает целый набор администраторов оформления, выполняющих автоматическую корректировку размещения элементов управления в форме при изменении ее размеров
Form	Тип, представляющий главное окно, диалоговое окно или дочернее окно MDI в приложении Windows Forms
ColorDialog, OpenFileDialog, SaveFileDialog, FontDialog, PrintPreviewDialog, FolderBrowserDialog	Представляют различные диалоговые окна, соответствующие стандартным операциям в рамках GUI
Menu, MainMenu, Menultem, ContextMenu, MenuStrip, ContextMenuStrip	Типы, используемые для построения оконных и контекстно-зависимых систем меню. Новые (появившиеся в .NET 2.0) элементы
StatusBar, Splitter, ToolBar, ScrollBar, StatusStrip, ToolStrip	Типы, используемые для добавления в форму стандартных элементов управления

Создание окон

Все окна являются объектами типа Form из пространства имен System.Windows.Forms. Поэтому создание диалогового или модального окна отличается только свойствами, устанавливаемыми для данного окна в окне Properties среды разработки (свойство FormBorderStyle) и методом, которым форма выводится на экран:

Form2.Show() – для немодальной формы;

Form2.ShowDialog() – для модального диалогового окна.

В случае использования диалоговых окон, доступны различные

варианты возвращения результатов функцией ShowDialog(). Результат имеет тип DialogResult.

Методика и порядок выполнения работы.

1. Создайте в среде разработки MS VS 2005 проект. Вкачестветипа проектавыбрать «WindowsApplication» (или «WindowsFormsApplication» в зависимости от версии .NET Framework)

2. После создания и сохранения проекта измените программу такимобразом, чтобы координаты курсора мыши выводились в заголовке главного окна приложения.

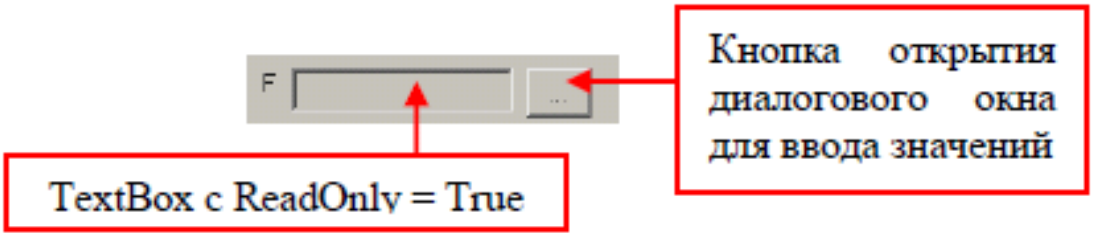
Для этого в обработчик события движения мыши над оконной формой(MouseMove) добавьте строку кода:

Text = string.Format("Координаты:{0}, {1}", e.X, e.Y);

3. Добавьте текстовое поле (TextBox) в режиме разработки (для этогонеобходимо использовать панель элементов управления «ToolBox»).

Дополните обработчик движения мыши таким образом, чтобы в текстовомполе отображалась сумма координат указателя мыши.

4.Измените программу такимобразом, чтобы все поля для ввода значений неизвестных переменных были доступны только для чтения (установить свойство ReadOnly). Рядом с полями TextBox разместите кнопки открытия диалогового окна, например:



5.Ввод значений в каждое окно должен производиться путем открытия диалогового окна с полем ввода значения и двумя кнопками OK и Cancel. (для обоих кнопок необходимо установить свойства DialogResult для последующего анализа возвращаемых значений методом ShowDialog).

6.Данное диалоговое окно открывается по нажатию специальной

кнопки, расположенной рядом с каждым текстовым окном для ввода значений. После указания значения и нажатия кнопки OK, диалоговое окно закрывается и введенное значение переносится в соответствующее поле.

7.После разработки всех визуальных компонент и форм, вычислите выражения в соответствии с вариантом индивидуального задания. Результат вывести в текстовое поле.

Задания для самостоятельной работы:

1. Вычислить значение выражения:

$$U = 1 - \frac{\cos^2(x \cdot t)}{2} + \frac{\sin^3(x \cdot z)}{3} - \frac{\cos^4(x \cdot t)}{4} + \frac{\sin^5(x \cdot z)}{5} -$$

… ,есликоличество слагаемых в правой части выражения пользователь вводит с клавиатуры.

2. Вычислить значение выражения:

$$U = 1 - \frac{\cos^2(x)}{2!} + \frac{\sin^3(x)}{3!} - \frac{\cos^4(x)}{4!} + \frac{\sin^5(x)}{5!} - \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

3. Вычислить значение выражения:

$$y = \frac{\sin(x) \cdot x^2}{2!} - \frac{x^2 \cdot \sin(x^3)}{3!} + \frac{\sin(x^3) \cdot x^4}{4!} - \frac{x^4 \cdot \sin(x^5)}{5!} + \ldots,$$

… ,если количество в правой части выражения пользователь вводит с клавиатуры.

4. Вычислить значение выражения:

$$h = \frac{x^2}{1\cdot 3} - \frac{\sin(x^4)}{3\cdot 5} + \frac{x^6}{5\cdot 7} - \frac{\sin(x^8)}{7\cdot 9} + \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

5. Вычислить значение выражения:

$$U = 1 - \frac{\sin^2(x)\cdot y}{2} + \frac{\sin^3(x)\cdot y^3}{3} - \frac{\sin^4(x)\cdot y^5}{4} + \frac{\sin^5(x)\cdot y^7}{5} - \ldots,$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

6. Вычислить значение выражения:

$$Z = 1 - \frac{\sin(x^2)}{2} + \frac{\cos(x^3)}{3} - \frac{\sin(x^4)}{4} + \frac{\cos(x^5)}{5} - \ldots$$

… ,если количество слагаемых в правой части выражения и целое *x*пользователь вводит с клавиатуры.

7. Вычислить значение выражения:

$$U = 1 - \frac{\sin^2(x)}{2} + \frac{\sin^3(x)}{3} - \frac{\sin^4(x)}{4} + \frac{\sin^5(x)}{5} - \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

8. Вычислить значение выражения:

$$V = \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \frac{x^5}{5!} + \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

9. Вычислить значение выражения:

$$s = -\frac{x^2}{2 \cdot 3} + \frac{x^3}{3 \cdot 4} - \frac{x^4}{4 \cdot 5} + \frac{x^5}{5 \cdot 6} - \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

10. Вычислить значение выражения:

$$h = \frac{y^3 \cdot x^2}{1 \cdot 3} - \frac{y^5 \cdot x^4}{3 \cdot 5} + \frac{y^7 \cdot x^6}{5 \cdot 7} - \frac{y^9 \cdot x^8}{7 \cdot 9} + \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

11.Вычислить значение выражения:

$$p = -\frac{x}{1 \cdot 2 \cdot 3} + \frac{x^3}{3 \cdot 4 \cdot 5} - \frac{x^5}{5 \cdot 6 \cdot 7} + \frac{x^7}{7 \cdot 8 \cdot 9} - \ldots$$

… ,если количество слагаемых в правой части выражения пользователь вводит с клавиатуры

12.Вычислить значение выражения: $T = -\frac{x}{1 \cdot 3} + \frac{x^3}{3 \cdot 5} - \frac{x^5}{5 \cdot 7} + \frac{x^7}{7 \cdot 9} - \ldots$, если количество слагаемых в правой части выражения пользователь вводит с клавиатуры.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 8.
Программирование графики с использованием GDI+

Цель работы: научиться использовать инструменты визуализации двумерной графики, предоставляемые пространством имен GDI+.

Теоретическая часть

В таблице 4.1 приводится описание базовых пространств имен GDI+.

Таблица 4.1 – Базовые пространства имен GDI+

Пространство имен	Описание
System.Drawing	Базовое пространство имен GDI+, определяющее множество типов для основных операций визуализации (шрифты, перья, кисти и т.д.), а также тип Graphics.
System.Drawing.Drawing2D	Содержит описание типов, используемых для более сложной двумерной/векторной графики(градиентные кисти, стили окончания линий,геометрические трансформации и т.д.)
System.Drawing.Imaging	Содержит описание типов, обеспечивающих обработку графических изображений(изменение палитры, извлечение метаданных изображения, работа с метафайлами и т.д.)
System.Drawing.Printing	Содержит описание типов, обеспечивающих отображение графики на печатной странице, непосредственное взаимодействие с принтером.
System.Drawing.Text	Дает возможность управлять коллекциями шрифтов.

При создании приложения Windows с использованием мастера, ссылка на System.Drawing.dll устанавливается автоматически.

Обзор пространства имен System.Drawing

Таблица 4.2 – Базовые типы пространства имен System.Drawing

Тип	Описание
Bitmap	Тип, инкапсулирующий данные изображения
Brush, Brushes, SolidBrush, SystemBrushes, TextureBrush	Объекты Brush используются для заполнения внутренних областей графических форм (прямоугольников, эллипсов и т.д.)
BufferedGraphics	Новый тип .NET 2.0, обеспечивающий графический буфер для двойной буферизации, которая используется для уменьшения или полного исключения эффекта мелькания, возникающего при перерисовке
Color SystemColors	Определяют ряд статических свойств, доступных только для чтения и используемых для получения нужного цвета при использовании перьев и кистей
Font, FontFamily	Тип Font инкапсулирует характеристики данного шрифта (название, плотность, размер и т.д.). FontFamily предлагает абстракцию для группы шрифтов, имеющих аналогичный дизайн, но определенные вариации стиля
Graphics	Представляет реальную поверхность нанесения изображения, а так же предлагает ряд методов для визуализации текста, изображений и графических шаблонов.
Pen Pens SystemPens	Pens – это объекты, используемые для построения линий, кривых. Тип Pen определяет ряд статических свойств, возвращающих новый объект Pen заданного цвета.
Point, PointF	Структуры, представляющие точку
Rectangle RectangleF	Структуры, представляющие прямоугольник
Size SizeF	Структуры, представляющие размер (в виде целого или вещественного числа)

Утилитарные типы System.Drawing

Многие из методов рисования, определенные объектом System.Drawing.Graphics, требуют указать позицию или область, в которой требуется отобразить данный элемент. Другие методы требуют указания размеров, координат, границ геометрического образа.

Тип Point(F) поддерживает ряд полезных членов:

- перегруженные операции: +, -, ==, != ;
- доступ к значениям (x,y): X, Y;
- поле IsEmpty возвращает значение true, если x и y установлены в 0.

Типы **Rectangle(F)**, как и Point, необходимы в большинстве GUI-приложений. Одним из наиболее полезных методов типа Rectangle является Contains(), который позволяет выяснить, находится ли данный тип Point или Rectangle в рамках границ некоторого другого объекта.

Класс Graphics

Класс System.Windows.Graphics – это интерфейс для функциональных возможностей GDI+. Этот класс не только предоставляет поверхность, на которой размещаются изображения (поверхность формы, поверхность элемента управления или область в памяти), но определяет также члены, которые позволяют отображать текст, изображения (пиктограммы, точечные рисунки и т.д.) и самые разные геометрические формы.

Класс Graphics не допускает непосредственного создания своего экземпляра с помощью ключевого слова new, поскольку класс не имеет открытых конструкторов.

Для получения доступа к объекту Graphics необходимо создатьобработчик события WM_PAINT, для этого для формы необходимо зарегистрировать обработчик события Paint. В качестве параметра метода-обработчика будет передан объект типа PaintEventArgs

Задания для самостоятельной работы.

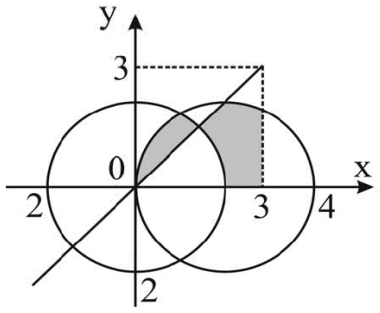
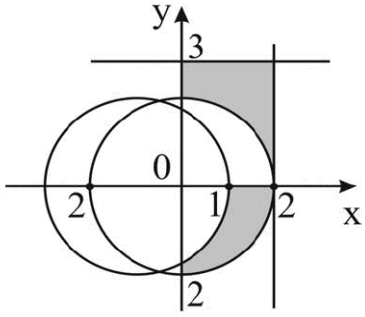
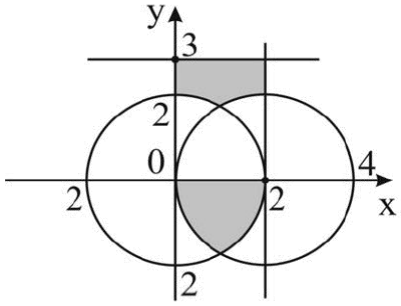
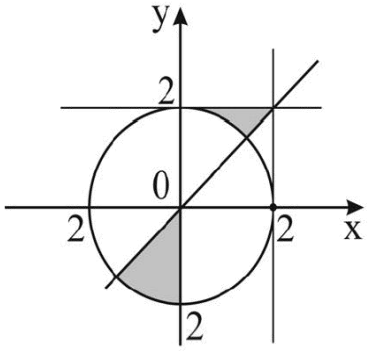
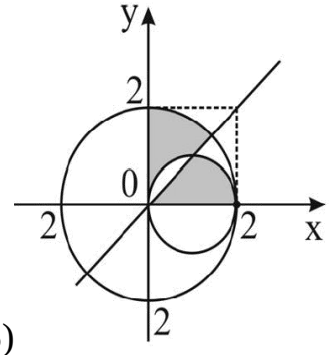
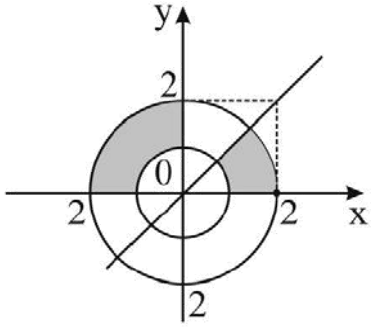
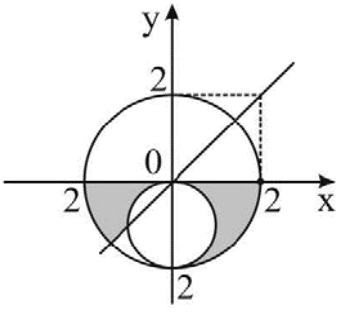
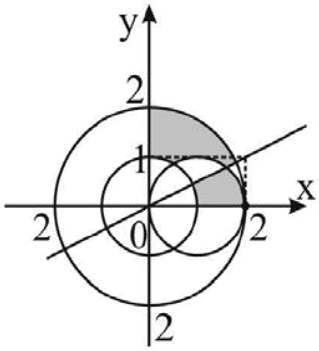
1.Создайте приложение Windows в среде Visual Studio 2005.

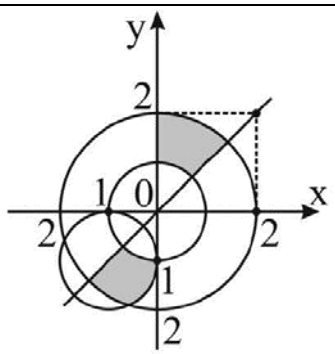
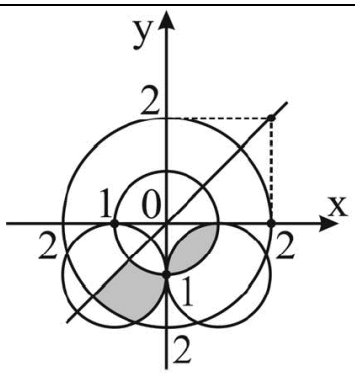
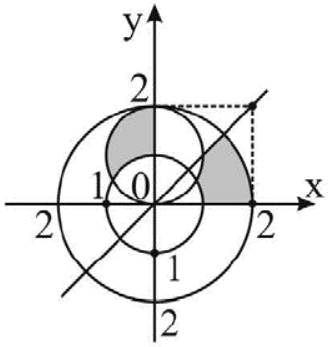
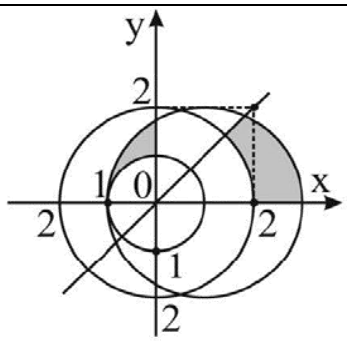
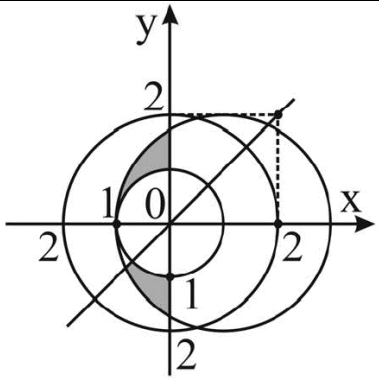
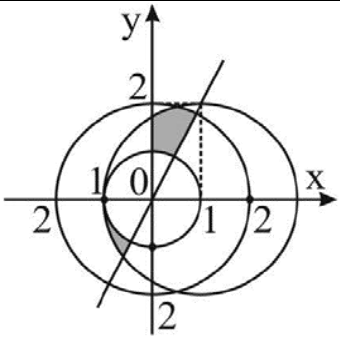
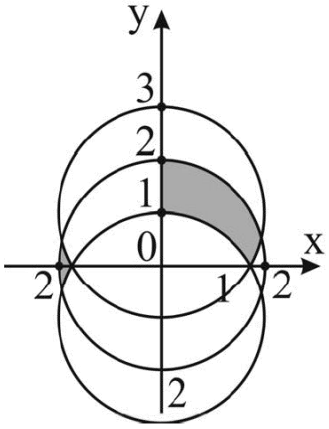
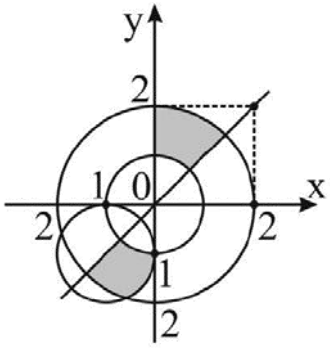
2.Выведите на поверхность формы рисунок, в соответствие с

индивидуальным заданием.

3.Создайте текстовые поля для ввода координат точки на плоскости.

4.При изменении координат точки в текстовых полях, должно выводится сообщение, принадлежит ли данная точка закрашенной области (в соответствие с индивидуальным заданием).

<table>
<tr>
<td>1)
 
</td>
<td>2)
 
</td>
</tr>
<tr>
<td>3)
 
</td>
<td>4)
 
</td>
</tr>
<tr>
<td>5)
 
</td>
<td>6)
 
</td>
</tr>
<tr>
<td>7)
 
</td>
<td>8)
 
</td>
</tr>
</table>

9) 	10) 
11) 	12) 
13) 	14) 
15) 	16) 

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа

SuperVGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 9.
Главное и контекстное меню приложений Windows

Цель работы: научиться использовать в программах контекстные меню, главное меню приложения.

Теоретическая часть

В рамках платформы .NET 2.0 элементом управления для создания системы меню является MenuStrip. Этот элемент управления позволяет создавать как пункты меню, представляющие собой любые подходящие элементы управления. Некоторые общие элементы интерфейса, которые могут содержаться в MenuStrip:

ToolStripMenuItem - традиционный пункт меню;

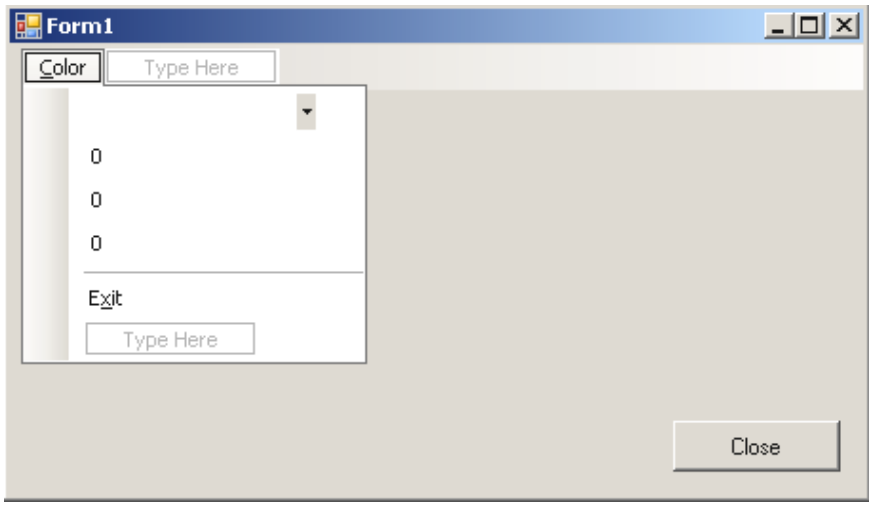
ToolStripComboBox - встроенный элемент ComboBox; ToolStripSeparator - простая линия, разделяющая содержимое. ToolStripTextBox - встроенный элемент TextBox.

MenuStrip содержит строго типизированную коллекцию ToolStripItemCollection. Подобно другим типам коллекции, этот объект поддерживает методы Add(), AddRange(), Remove() и свойство Count. Эта коллекция обычно заполняется не напрямую, а с помощью различных инструментов режима проектирования.

Методика и порядок выполнения работы.

1. Создайтеприложение Windows Application всреде MS Visual Studio 2005.
2. Поместите элемент MenuStrip в главную форму в окне проектирования.

3. Создайте меню уровня со следующей структурой:



3.1 Для этого свойстве Text пункта меню верхнего уровня установите равным «&Color».

3.2 Первым элементом меню выберите элемент типа ToolStripComboBox, для которого, в режиме проектирования, установите следующие свойства:

DropDownStyle = DropDownList

Items = список значений: белый, красный, черный, синий, желтый ToolTipText = «Готовые цвета»

3.3 Далее следуют три элемента типа ToolStripTextBox, для каждого задается свойство Text = 0. Дополнительно, для toolStripTextBox1 свойство ToolTipText устанавливается равным «Красный», для toolStripTextBox2 – «Зеленый», toolStripTextBox3 – «Синий».

3.4 Следующий пункт меню типа ToolStripSeparator.

3.5 Последний пункт меню типа ToolStripMenuItem, для которого устанавливаются свойство Text = “E&xit”.

4. После создания меню, необходимо инициализировать начальные значения пункта ToolStripComboBox. Для этого в обработчике события Load главной формы добавьте код.

5. Для обработки выбора пункта Exit, добавьте в обработчик события Click данного пункта код.

6. Запустите приложение, в случае необходимости исправьте ошибки. Обратите внимание на работу пункта меню Exit.

Следующим этапом является обработка выбора пунктов меню: Для обработки выбора цвета в первом пункте меню выполните следующие действия:

1 Создайте обработчик события SelectedIndexChanged.

2 Запустите приложение, протестируйте обработчик.

Для задания RGB-компонент цвета с использованием пунктов меню типа ToolStripTextBox выполните следующие действия:

1. Для первого текстового поля в меню создайте обработчик события TextChanged.

2. Для второго и третьего полей установите обработчики событий TextChanged. В качестве функции-обработчика события для обоих оставшихся диалоговых окон выбрать из списка обработчик toolStripTextBox1_TextChanged первого окна (которое отвечает за ввод компоненты красного цвета).

Создайте кнопку Close, установите для нее обработчик события Click такой же как у пункта меню Exit.

По аналогии с созданием главного меню приложения создайте контекстное приложения, открывающееся при щелчке правой кнопкой мыши на окне приложения. Для этого:

1. Спроектируйте контекстное меню на основе элемента управления ContextMenuStrip.

2. Укажите значение свойства ContextMenuStrip главной формы равным ContextMenuStrip1 (необходимо указать имя созданного вами контекстного меню).

3. Подключите обработчики событий, созданные для главного меню к пунктам контекстного меню.

Выполните задание в соответствие с индивидуальным заданием, дополнив главное и контекстное меню своими пунктами.

Задания для самостоятельной работы:

1.Создайте и синхронизируйте пункты главного и контекстного менюa*x+b*y+sin(z),где x,y,z - неизвестные,вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которыхвыбираются из пунктов типа ToolStripComboBox. Значение выражения

выводится в заголовке главного окна.

2. Создайте и синхронизируйте пункты главного и контекстного меню

для вычисления выраженияa^2*x+ (b-a)*y+sin(z) где *x* , *y* , *z* - неизвестные, вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значениевыражения выводится в заголовке главного окна.

3. Создайте и синхронизируйте пункты главного и контекстного меню

для вычисления выражения a*x^2+b*(x-y)+lg(a*z), где *x* , *y* , *z* - неизвестные, вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значениевыражения выводится в заголовке главного окна.

4. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения, г $\frac{a \cdot x}{b} + b \cdot \frac{y}{x} + \sin(z)$

вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выраженияявыводится в заголовке главного окна.

5. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot z + b \cdot y + \sin(z)}{a + b}$, где *x* , *y* , *z* – неизвестные,

вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

6. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения, г, $a \cdot \frac{x}{y} + b \cdot \frac{y}{z} + \sin(\frac{z}{x})$

неизвестные,вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

7. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{b - y} + \frac{b \cdot y + \sin(z)}{a \cdot x}$, где *x* , *y* , *z* - неизвестные,

вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

8. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot \sqrt{x} + b \cdot y + \cos(x) \cdot \sin(z)$,

неизвестные, вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

9. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выраженияг $\lg(a \cdot x) + \sin(z) - \cos(y)$,

неизвестные, вводимые в поля ToolStripTextBox; *a* , *b* - константы, значениякоторых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

10. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $a \cdot x + \frac{b \cdot y}{\lg(z)} + \frac{\sin(z)}{\cos(y)}$,

неизвестные,вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которыхвыбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

11. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражениягые, $\frac{a \cdot x}{\cos(z)} + \frac{b \cdot y + \sin(z)}{\sqrt{|x - y|}}$,

вводимые в поля ToolStripTextBox; *a* , *b* - константы, значения которых

выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

12. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{|z|} + b \cdot y + \frac{\sin(z)}{\sqrt{|x^2 - y|}}$, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

13. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\frac{a \cdot x}{y} + \frac{b \cdot y + \sin(z)}{|x - y^2|}$, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

14. Создайте и синхронизируйте пункты главного и контекстного меню для вычисления выражения $\sqrt{|a \cdot x + b \cdot y + \sin(z)|}$, вводимые в поля ToolStripTextBox; a , b - константы, значения которых выбираются из пунктов типа ToolStripComboBox. Значение выражения выводится в заголовке главного окна.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт

персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 10.
Несвязный уровень ADO.NET

Цель работы: изучить назначение и основные способы создания объектов ADO.NET при помощи Visual Studio IDE

Теоретическая часть

В платформе .NET определено множество типов (организованных в соответствующие пространства имен) для взаимодействия с локальными и удаленными хранилищами данных. Общее название пространств имен с этими типами - ADO.NET.

ADO.NET - это новая технология доступа к базам данных, специально оптимизированная для нужд построения рассоединенных (disconnected) систем на платформе .NET.

Технология ADO.NET ориентирована на приложения N-tier - архитектуру многоуровневых приложений, которая в настоящее время стала фактическим стандартом для создания распределенных систем.

Основные отличительные особенности ADO.NET:

- ADO расширяет концепцию объектов-наборов записей в базе данных новым типом DataSet, который представляет локальную копию сразу множества взаимосвязанных таблиц. При помощи объекта DataSet пользователь может локально производить различные операции с содержимым базыданных, будучи физически рассоединен с СУБД, и после завершения этих операций передавать внесенные изменения в базу данных при помощи соответствующего "адаптера данных" (data adapter);
- в ADO.NET реализована полная поддержка представления данных в XML -совместимых форматах. В ADO.NET сформированные для локальной обработки наборы данных представлены в формате XML (в этом же формате они и передаются с сервера баз данных). Данные в форматах XML очень удобно передавать при помощи обычного HTTP, решает многие проблемы с установлением соединений через брандмауэры;

- ADO.NET - это библиотека управляемого кода и взаимодействие с ней производится как с обычной сборкой .NET. Типы ADO.NET используют возможности управления памятью CLR и могут использоваться во многих .NET - совместимых языках. При этом обращение к типам ADO.NET (и их членам) производится практически одинаково вне зависимости от того, какой язык используется.

Все типы ADO.NET предназначены для выполнения единого набора задач:

- установить соединение с хранилищем данных;
- создать и заполнить данными объект DataSet ;
- отключиться от хранилища данных и вернуть изменения, внесенные в объект DataSet обратно в хранилище данных.

Объект DataSet - это тип данных, представляющий локальный набор таблиц и информацию об отношениях между ними.

DataSet - набор связанных таблиц. На практике можно создать на клиенте объект DataSet, который будет представлять полную копию удаленной базы данных.

После создания объекта DataSet и его заполнения данными можно программными средствами производить запросы к нему и перемещаться по таблицам, выполнять все операции, как при работе с обычными базами данных: добавлять в таблицы новые записи, удалять и изменять существующие, применять к ним фильтры и т.п. После того как клиент завершит внесение изменений, информация о них будет отправлена в хранилище данных для обработки.

Создание DataSet осуществляется при помощи управляемого провайдера (managed provider).

Управляемый провайдер - это набор классов, реализующих интерфейсы, определенные в пространстве имен System.Data.

Речь идет об интерфейсах IDbCommand, IDbDataAdapter, IDbConnection и IDataReader (рисунок 8.1).

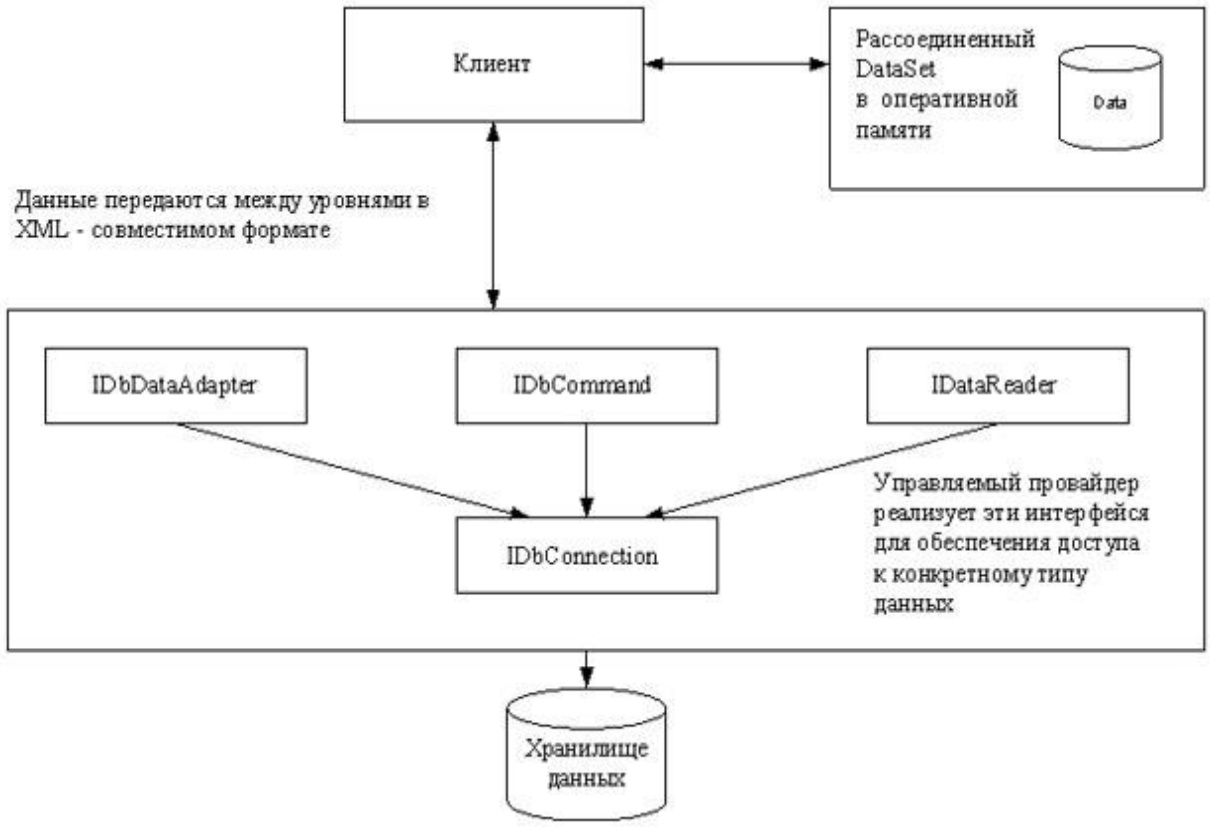


Рис. 8.1. Взаимодействие клиента с управляемыми провайдерами

В состав ADO.NET включены два управляемых провайдера: провайдер SQL и провайдер OleDb. Провайдер SQL специально оптимизирован под взаимодействие с Microsoft SQL Server версии 7.0 и последующих. Для других источников данных предлагается использовать провайдер OleDb, который можно использовать для обращения к любым хранилищам данных, поддерживающим протокол OLE DB. Следует отметить, что провайдер OleDb работает при помощи "родного" OLE DB и требует возможности взаимодействия при помощи COM.

Все возможности ADO.NET заключены в типах, определенных в соответствующих пространствах имен. Краткий обзор главных пространств имен ADO.NET представлен в таблице 8.1.

Таблица 8.1. Пространства имен ADO.NET

Пространство имен	**Описание**
System.Data	Главное пространство имен ADO.NET. В нем определены

	типы, представляющие таблицы, столбцы, записи, ограничения и тип - DataSet.
System.Data.Common	Определены типы, общие для всех управляемых провайдеров. Многие из них выступают в качестве базовых классов для классов из пространств имен для провайдеров SQL и OleDb
System.Data.OleDb	В этом пространстве имен определены типы для установления соединений с OLE DB-совместимыми источниками данных, выполнения к ним SQL-запросов и заполнения данными объектов DataSet.
System.Data.SqlCIient	В этом пространстве имен определены типы, которые составляют управляемый провайдер SQL.
System.Data.SqlTypes	Представляют собой "родные" типы данных Microsoft SQL Server.

Все пространства имен ADO.NET расположены в одной сборке - System.Data.dll. Это означает, что в любом проекте, использующем ADO.NET, мы должны добавить ссылку на эту сборку.

В любом приложении ADO.NET необходимо использовать, по крайней мере, одно пространство имен - System.Data. Кроме того, практически во всех ситуациях требуется использовать либо пространство имен System.Data.OleDb или System.Data.SqlClient - для установления соединения с источником данных.

Типы пространства имен System.Data предназначены для представления данных, полученных из источника (но не для установления соединения непосредственно с источником).

В основном эти типы представляют собой объектные представления примитивов для работы с базами данных - таблицами, строками, столбцами, ограничениями и т. п. Наиболее часто используемые типы System.Data представлены в таблице 8.2.

Таблица 8.2. Типы пространства имен System.Data	
Тип	**Назначение**
DataColumnCollection, DataColumn	DataColumn представляет один столбец в объекте DataTable, DataColumnCollection - все столбцы
ConstraintCollection, Constraint	Constraint - объектно-ориентированная оболочка вокруг ограничения (например, внешнего ключа или уникальности), наложенного на один или несколько DataColumn, ConstraintCollection - все ограничения в объекте DataTable
DataRowCollection, Data Row	DataRow представляет единственную строку в DataTable, DataRowCollection - все строки в DataTable
DataRowView, Data View	DataRowView позволяет создавать настроенное представление единственной строки, DataView - созданное программным образом представление объекта DataTable, которое может быть использовано для сортировки, фильтрации, поиска, редактирования и перемещения
DataSet	Объект, создаваемый в оперативной памяти на клиентском компьютере. DataSet состоит из множества объектов DataTable и информации об отношениях между ними
ForeignKeyConstraint, UniqueConstraint	ForeignKeyConstraint представляет ограничение, налагаемое на набор столбцов в таблицах, связанных отношениями первичный - внешний ключ. UniqueConstraint - ограничение, при помощи которого гарантируется, что в столбце не будет повторяющихся записей
DataRelationCollection,	Тип DataRelationCollection представляет набор всех

DataRelation, DataTableCollection, DataTable	отношений (то есть объектов DataRelation) между таблицами в DataSet. Тип DataTableCollection представляет набор всех таблиц (объектов DataTable) в DataSet

В традиционных системах клиент-сервер при запуске приложения пользователем автоматически устанавливается связь с базой данных, которая поддерживается в "активном" состоянии до тех пор, пока приложение не будет закрыто. Такой метод работы с данными становится непрактичным, поскольку подобные приложения трудно масштабируются. Например, такая прикладная система может работать достаточно быстро и эффективно при наличии 8-10 пользователей, но она может стать полностью неработоспособной, если с ней начнут работать 100, 200 и более пользователей. Каждое открываемое соединение с базой данных "потребляет" достаточно много системных ресурсов сервера, они становятся занятыми поддержкой и обслуживанием открытых соединений, их не остается на процессы непосредственной обработки данных.

При разработке прикладных систем в сети Интернет (Web -приложения) необходимо добиваться максимальной масштабируемости. Система должна работать одинаково эффективно как с малым, так и с большим числом пользователей.

По этой причине, в ADO.NET используется модель работы пользователя в отрыве от источника данных. Приложения подключаются к базе данных только на небольшой промежуток времени. Соединение устанавливается только тогда, когда клиент удаленного компьютера запрашивает на сервере данные. После того, как сервер подготовил необходимый набор данных, сформировал и отправил их клиенту в виде WEB -страницы, связь приложения с сервером сразу же обрывается, и клиент просматривает полученную информацию уже не в связи с сервером. При работе в сети Интернет нет необходимости поддерживать постоянную "жизнеспособность" открытых соединений, поскольку неизвестно, будет ли конкретный клиент вообще далее взаимодействовать с источником данных. В таком случае целесообразнее сразу освобождать занимаемые

серверные ресурсы, что обеспечит обслуживание большего количества пользователей. Модели доступа к данным представлена на рисунке 8.2.

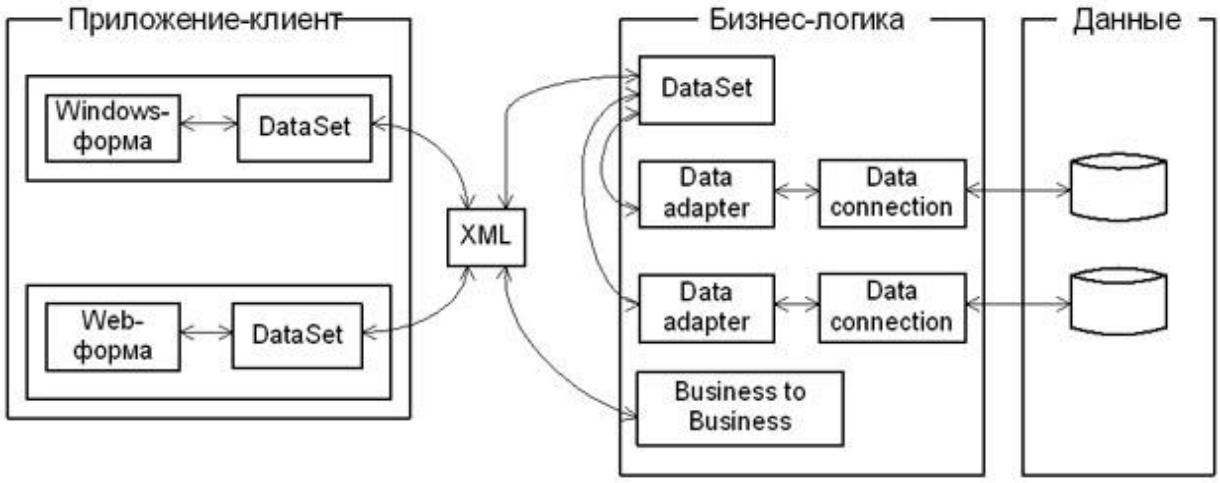


Рис. 8.2. Модель доступа к данным в ADO.NET

В объектной модели ADO.NET можно выделить несколько уровней.

Уровень данных. Это по сути дела базовый уровень, на котором располагаются сами данные (например, таблицы базы данных MS SQL Server). На данном уровне обеспечивается физическое хранение информации на магнитных носителях и манипуляция с данными на уровне исходных таблиц (выборка, сортировка, добавление, удаление, обновление и т. п.).

Уровень бизнес-логики. Это набор объектов, определяющих, с какой базой данных предстоит установить связь и какие действия необходимо будет выполнить с содержащейся в ней информацией. Для установления связи с базами данных используется объект DataConnection. Для хранения команд, выполняющих какие либо действия над данными, используется объект DataAdapter. И, наконец, если выполнялся процесс выборки информации из базы данных, для хранения результатов выборки используется объект DataSet. Объект DataSet представляет собой набор данных "вырезанных" из таблиц основного хранилища, который может быть передан любой программе-клиенту, способной либо отобразить эту информацию конечному пользователю, либо выполнить какие-либо манипуляции с полученными данными.

Уровень приложения. Это набор объектов, позволяющих хранить и отображать данные на компьютере конечного пользователя. Для хранения информации используется уже знакомый нам объект DataSet, а для отображения данных имеется довольно большой набор элементов управления (DataGrid, TextBox, ComboBox, Label и т. д.). В Visual Studio .Net можно вести разработку двух типов приложений. В первую очередь это традиционные Windows - приложения (на основе Windows -форм), которые реализованы в виде exe-файлов, запускаемых на компьютере пользователя. Ну и конечно, Web - приложения (на основе Web -форм), которые работают в оболочке браузера. Как видно из рисунка 8.2, для хранения данных на уровне обоих типов приложений используется объект DataSet.

Обмен данными между приложениями и уровнем бизнес-логики происходит с использованием формата XML, а средой передачи данных служат либо локальная сеть (Интернет), либо глобальная сеть (Интернет).

В ADO.NET для манипуляции с данными могут использоваться команды, реализованные в виде SQL -запросов или хранимых процедур (DataCommand). Например, если необходимо получить некий набор информации базы данных, вы формируете команду SELECT или вызываете хранимую процедуру по ее имени.

Когда требуется получить набор строк из базы данных, необходимо выполнить следующую последовательность действий:

- открыть соединение (connection) с базой данных;
- вызвать на исполнение метод или команду, указав ей в качестве параметра текст SQL -запроса или имя хранимой процедуры;
- закрыть соединение с базой данных.

Связь с базой данных остается активной только на достаточно короткий срок - на период выполнения запроса или хранимой процедуры.

Когда команда вызывается на исполнение, она возвращает либо данные, либо код ошибки. Если в команде содержался SQL -запрос на выборку - SELECT, то команда может вернуть набор данных. Вы можете выбрать из базы данных только определенные строки и колонки, используя объект DataReader,

который работает достаточно быстро, поскольку использует курсоры read-only, forward-only.

Если требуется выполнить более чем одну операцию с данными, например, получить некоторый набор данных, а затем скорректировать его, - то необходимо выполнить последовательность команд. Каждая команда выполняется отдельно, последовательно одна за другой. Между выполняемыми командами соединение с базой отсутствует. Например, чтобы получить данные из базы - открывается связь, выбираются данные, затем связь закрывается. Когда выполняется обновление базы после корректировки информации пользователем, снова открывается связь, выполняется обновление данных в исходных таблицах и связь снова закрывается.

Команды работы с данными могут содержать параметры, т. е. могут использоваться параметризированные запросы, как, например, следующий запрос:

SELECT * FROM customers WHERE (customer_id=@customerid)

Значения параметров могут задаваться динамически, во время выполнения приложения.

Как правило, в приложениях необходимо извлечь информацию из базы данных и выполнить с ней некоторые действия: показать пользователю на экране монитора, сделать нужные расчеты или послать данные в другой компонент. Очень часто, в приложении нужно обработать не одну запись, а их набор: список клиентов, перечень заказов, набор элементов заказа и т. п. Как правило, в приложениях требуется одновременная работа с более чем одной таблицей: клиенты и все их заказы; автор и все его книги, заказ и его элементы, т.е. с набором связанных данных. Причем для удобства пользователя данные требуется группировать и сортировать то по одному, то по другому признаку. При этом нерационально каждый раз возвращаться к исходной базе данных и заново считывать данные. Более практично работать с некой временной "вырезкой" информации, хранящейся в оперативной памяти компьютера.

Эту роль выполняет набор данных - DataSet, который представляет собой своеобразный кэш записей, извлеченных из базового источника. DataSet может

состоять из одной или более таблиц, он имеет дело с копиями таблиц из базы данных источника. Кроме того, в данном объекте могут содержаться связи между таблицами и некоторые ограничения на выбираемые данные. Структура объекта DataSet приведена на рисунке 8.3.

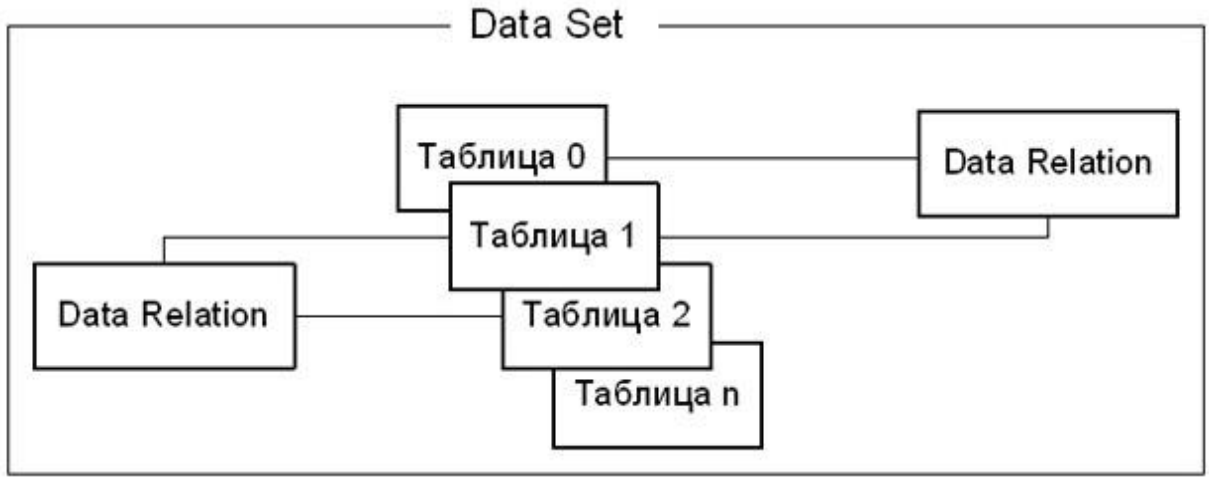


Рис. 8.3. Структура объекта DataSet

Данные в DataSet - это некий уменьшенный вариант основной базы данных. Тем не менее, вы можете работать с такой "вырезкой" точно так же, как и с реальной базой. Поскольку каждый пользователь манипулирует с полученной порцией информации, оставаясь отсоединенными от основной базы данных, последняя может в это время решать другие задачи.

Конечно, практически в любой задаче обработки данных требуется корректировать информацию в базе данных (хотя и не так часто, как извлекать данные из нее). Вы можете выполнить операции коррекции непосредственно в DataSet, а потом все внесенные изменения будут переданы в основную базу данных.

Важно отметить то, что DataSet - пассивный контейнер для данных, который обеспечивает только их хранение. Что же нужно поместить в этот контейнер, определяется в другом объекте - адаптере данных DataAdapter. В адаптере данных содержатся одна или более команд, которые определяют, какую информацию нужно поместить в таблицы объекта DataSet, по каким правилам нужно синхронизировать информацию в конкретной таблице DataSet

и соответствующей таблицей основной базы данных и т. п. Адаптер данных обычно содержит четыре команды SELECT, INSERT, UPDATE, DELETE, для выборки, добавления, корректировки и удаления записей.

Например, метод Fill объекта DataAdapter, заполняющего данными контейнер DataSet, может использовать в элементе SelectCommand следующий запрос:

SELECT au_id, au_lname, au_fname FROM authors

Набор данных DataSet - "независимая" копия фрагмента базы данных, расположенная на компьютере пользователя. Причем в этой копии могут быть не отражены те изменения, которые могли внести в основную базу данных другие пользователи. Если требуется увидеть самые последние изменения, сделанные другими пользователями, то необходимо "освежить" DataSet, повторно вызвав метод Fill адаптера данных.

Разрабатываемое приложение предназначено для работы с базой данных сотрудников компании. На рисунке 8.4 представлена структура базы данных.

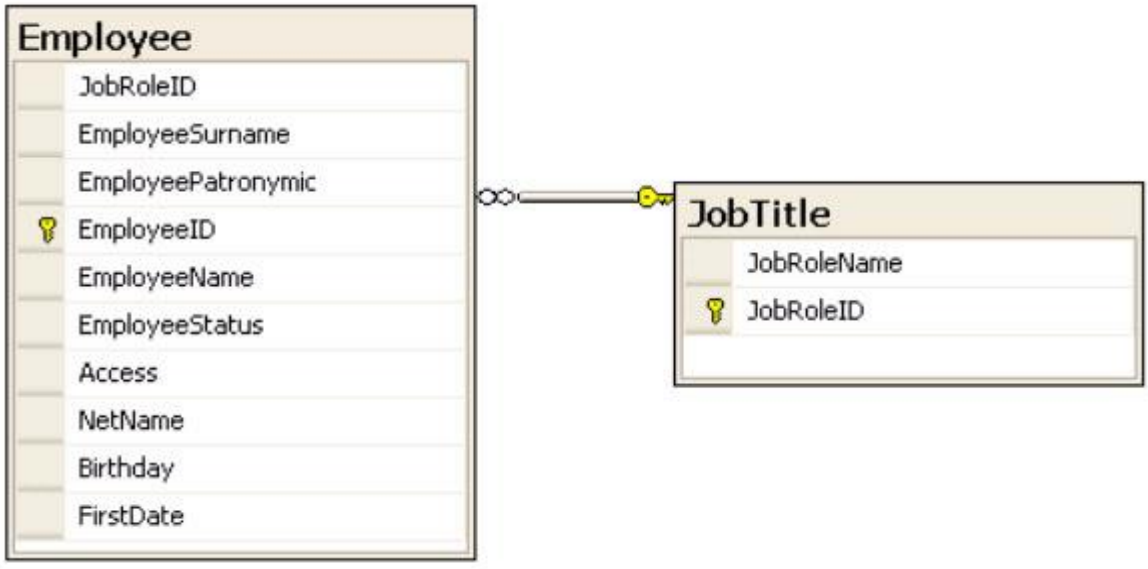


Рис. 8.4. Структура базы данных по сотрудникам компании

База данных включает две таблицы:

- сведения о сотрудниках - Employee ;
- справочник должностей - JobTitle.

Назначение атрибутов таблицы Employee приведены в таблице 8.4

Таблица 8.4. Атрибуты таблицы Employee

Имя атрибута	Назначение	Тип
EmployeeID	Суррогатный ключ	smallint
JobRoleID	Внешний ключ	smallint
EmployeeSurname	Фамилия	varchar(50)
EmployeeName	Имя	varchar(20)
EmployeePatronymic	Отчество	varchar(20)
EmployeeStatus	Статус	int
Access	Уровень доступа	varchar(20)
NetName	Сетевое имя	varchar(20)
Birthday	Дата рождения	Smalldatetime
FirstDate	Дата приема на работу	smalldatetime

Суррогатный ключ EmployeeID, как и все остальные суррогатные ключи базы данных, генерируется сервером базы данных автоматически, т.е. для него задано свойство IDENTITY для СУБД MS SQL Server или AutoNumber для MS Access. Атрибут JobRoleID является внешним ключом, с помощью которого осуществляется связь с таблицей JobTitle.

Назначение атрибутов таблицы JobTitle приведено в таблице 8.3.

Таблица 8.3. Атрибуты таблицы JobTitle

Имя атрибута	Назначение	Тип
JobRoleID	Суррогатный ключ	smallint
JobRoleName	Наименование должности	varchar(50)

В рассматриваемом приложении в качестве СУБД используется MS SQL Server 2005. Создаем соединение проекта с базой данных. Для этого выбираем пункт меню Tools/Connect to Database. Появляется окно AddConnection (рисунок 8.5)

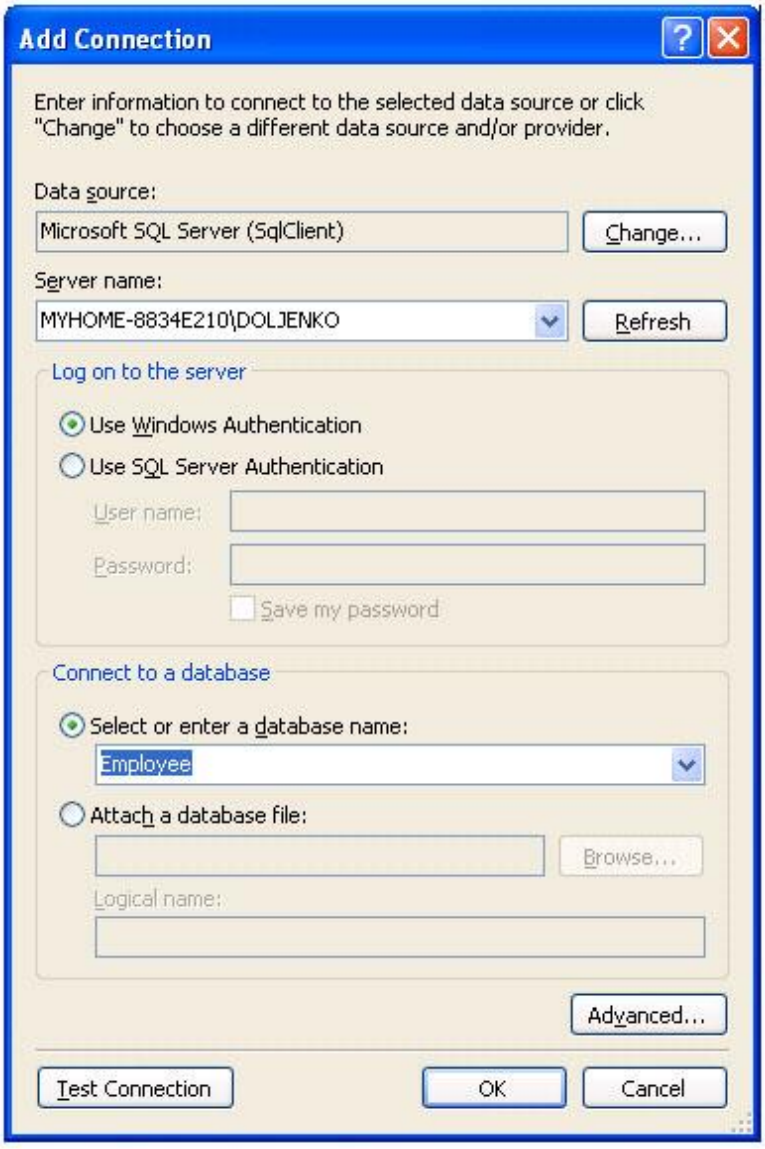


Рис. 8.5. Окно AddConnection

В пункте "Server name" задаем имя сервера, которое необходимо узнать у преподавателя (на рисунке 8.5 MYHOME-8834E210\DOLJENKO). В пункте Select or enter database name - имя базы данных, которое определит преподаватель (на рисунке 8.5 - Employee).

Для проверки правильности подключения к базе данных нажимаем клавишу "Test Connection". При правильном подключении появляется следующее сообщение (рисунок 8.6).

Рис. 8.6. Окно Microsoft Data Link

При нормальном соединении с базой данных можно открыть навигатор Server Explorer из меню View/ Server Explorer илисочетаниемклавиш ALT+CTRL+S (рисунок 8.7).

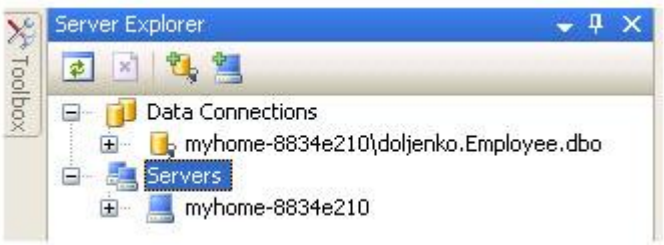


Рис. 8.7. Окно навигатора Server Explorer

Добавим в проект объект класса DataSet. Для этого выберем пункт меню Project/Add New Item. . . (рисунок 8.8).

В окне Add New Item (рисунок 8.9) выберем шаблон DataSet и присвоим ему имя DataSetEmployee.

После нажатия кнопки Add система генерирует класс DataSetEmployee, который добавляется в решение проекта (рисунок 8.10).

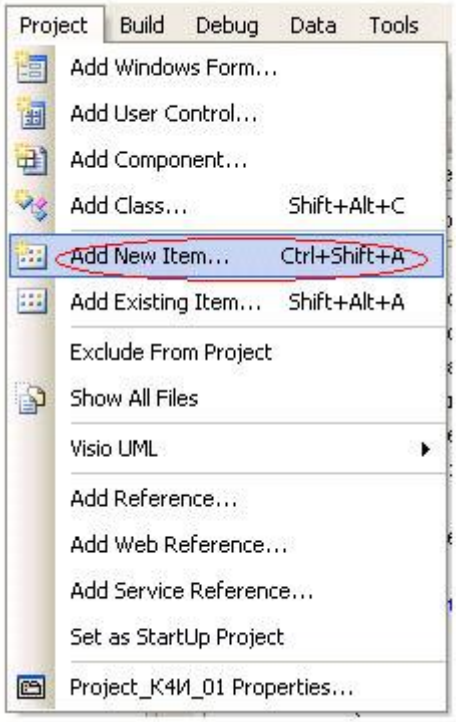


Рис. 8.8. Добавление в проект нового компонента

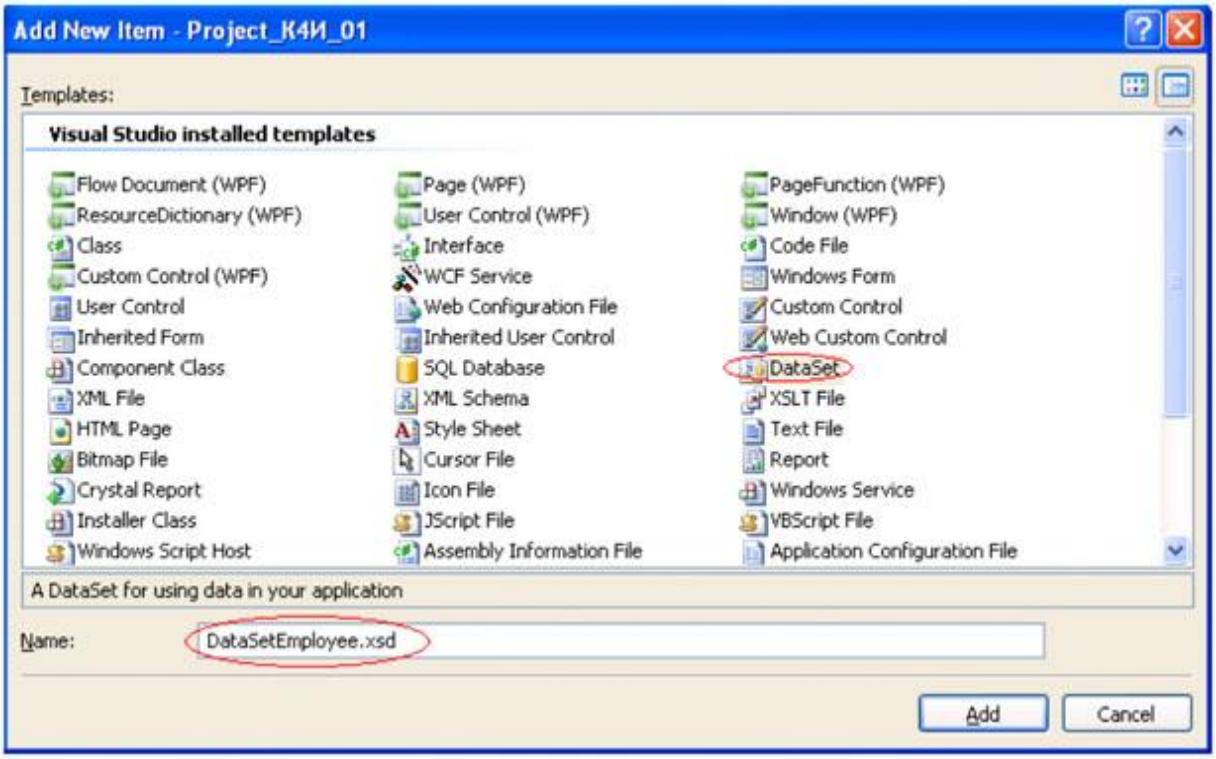


Рис. 8.9. Выбор нового компонента - DataSet

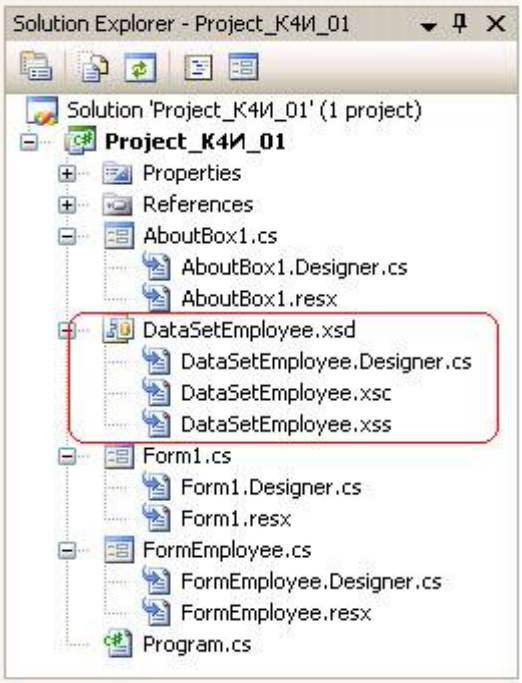


Рис. 8.10. Окно решения проекта с новым компонентом DataSet

Для добавления таблиц Employee и JobTitle к DataSet необходимо перетащить их из окна Server Explorer на поле графического дизайнера (рисунок 8.11).

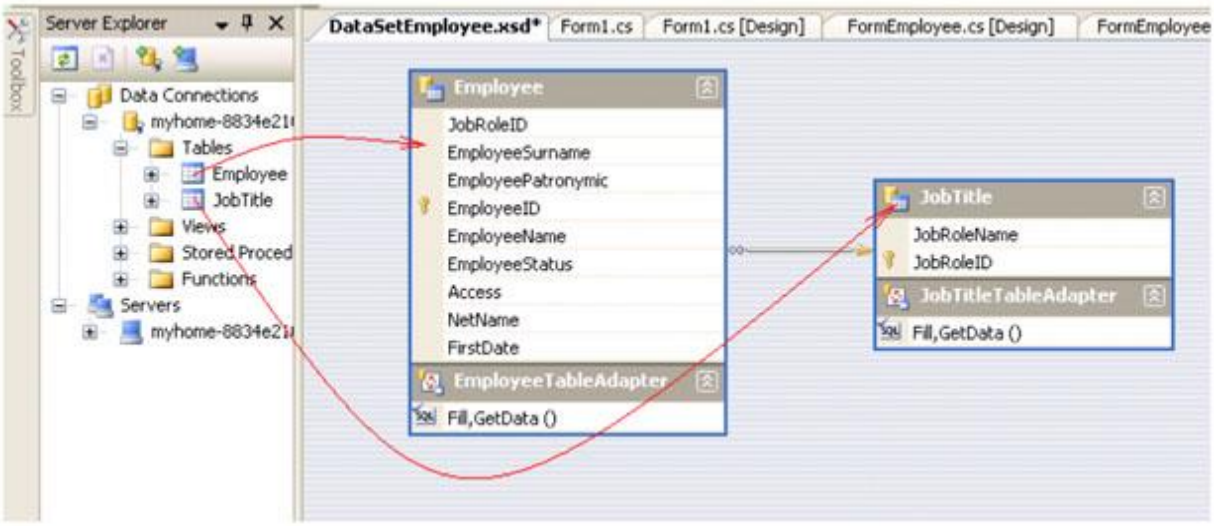


Рис. 8.11. Добавление таблиц к DataSet

В результате будут созданы классы таблиц, адаптеры и методы Fill и GetData.

При формировании класса DataSetEmployee необходимо учесть то, что первичные ключи таблиц Employee и JobTitle являются суррогатными и автоматически формируются (ключ со свойством автоинкремент) источником данных (например, MS SQL Server). При формировании новых записей в приложении необходимо обеспечить уникальность первичных ключей для таблиц объекта DataSetEmployee. Это можно обеспечить, задав для ключевых колонок таблиц Employee и JobTitle следующие свойства:

AutoIncrement = true;

AutoIncrementSeed = -1;

AutoIncrementStep = -1;

Столбец со свойством AutoIncrement равным true генерирует последовательность значений, начинающуюся со значения AutoIncrementSeed и имеющую шаг AutoIncrementStep. Это позволяет генерировать уникальные значения целочисленного столбца первичного ключа. В этом случае при добавлении новой записи в таблицу будет генерироваться новое значение первичного ключа, начиная с -1, -2, -3 и т.д., которое никогда не совпадет с первичным ключом источника данных, т.к. в базе данных генерируются положительные первичные ключи. Свойства AutoIncrementSeed и AutoIncrementStep устанавливаются равными -1, чтобы гарантировать, что когда набор данных будет синхронизироваться с источником данных, эти значения не будут конфликтовать со значениями первичного ключа в источнике данных. При синхронизации DataSet с источником данных, когда добавляют новую строку в таблицу MS SQL Server 2005 с первичным автоинкрементным ключом, значение, которое этот ключ имел в таблице DataSet, заменяется значением, сгенерированным СУБД.

Установка свойств AutoIncrement, AutoIncrementSeed и AutoIncrementStep для колонки первичного ключа EmployeeID таблицы Employee приведена на рисунке 8.12.

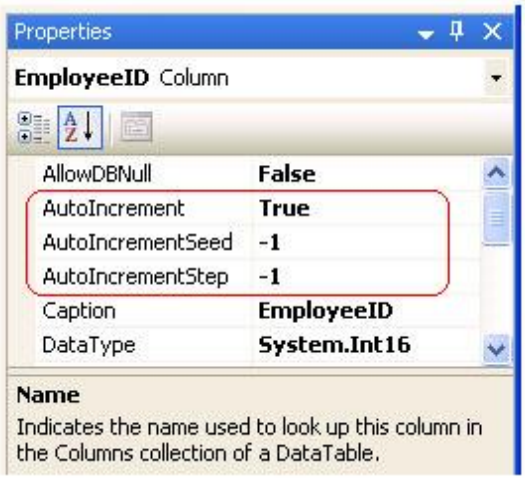


Рис. 8.12. Установка свойств для колонки EmployeeID

Аналогичные установки свойств AutoIncrement, AutoIncrementSeed и AutoIncrementStep необходимо сделать и для колонки JobTitleID таблицы JobTitle.

После создания класса DataSetEmployee и адаптера необходимо создать объекты этих классов, добавив следующий код к файлу FormEmployee.cs.

```
DataSetEmployee dsEmployee = new DataSetEmployee();
DataSetEmployeeTableAdapters.EmployeeTableAdapter daEmployee =
new Project_К4И_01.DataSetEmployeeTableAdapters.
EmployeeTableAdapter();
DataSetEmployeeTableAdapters.JobTitleTableAdapter daJobTitle =
new Project_К4И_01.DataSetEmployeeTableAdapters.
JobTitleTableAdapter();
```

Послетого, каксозданыобъектыадаптеровданныхdaEmployeeидаJobTitle, атакжеобъектклассаDataSetEmployee - dsEmployeeнеобходимосоздатьметоддлязаполненияобъектadsEmployeeизбазыданных (врассматриваемомпримеребазаданныхEmployee, созданнаявСУБДMSSQLServer2005). Для заполнения данными dsEmployee из базы данных Employee создадим метод EmployeeFill( ):

```
public void EmployeeFill()
{   daJobTitle.Fill(dsEmployee.JobTitle);
daEmployee.Fill(dsEmployee.Employee);
```

MessageBox.Show("Метод Fill отработал");}

В методе EmployeeFill( ) для объектов класса DataAdapter применяется метод Fill, который производит заполнение таблиц (JobTitle и Employee) объекта dsEmployee данными из базы данных. Метод Fill адаптера данных DataAdapter требует указания в качестве параметров задания соответствующей таблицы DataSet, то есть dsEmployee.JobTitle и dsEmployee.Employee.

Метод MessageBox.Show введен в метод EmployeeFill для первоначального тестирования, после которого его нужно убрать.

Вызов метода EmployeeFill необходимо добавить в обработчик события Load для формы FormEmployee, возникающего при нажатии на пункт меню "Сотрудник".

Задания для самостоятельной работы.

1.Изучите теоретический материал.

Создайте класс DataSetEmployee.

Для разрабатываемого приложения создайте объекты dsEmployee, daJobTitle и daEmployee.

Проведите компиляцию проекта и убедитесь в отсутствии ошибок трансляции.

Разработайте метод Fill для заполнение таблиц DataSet.

Протестировать работу приложения.

2.Редактирование таблицы через пользовательский интерфейс. Без использования средств Visual Studio по связыванию с БД.

Варианты заданий:

1)Таблица параметров компьютера.

2)Таблица успеваемости.

3)Таблица расписания самолётов.

4)Таблица товаров книжного магазина.

5)Таблица товаров продуктового магазина.

6)Таблица расписания автобусов.

7)таблица расписания поездов.

8)Таблица описания медецинских препоратов.

9)Таблица описания мебели.

10)Таблица описания марок автомобилей.

11)Таблица описания стран.

12)Таблица описания городов мира.

13)Таблица описания марок телефонов.

14)Таблица описания марок телевизоров.

Требование. Чтобы в таблице были данные различных типов: вещественные числа (или десятичные), строки, дата/время, целые (получается не менее 5 столбцов). Получение данных должно производится с использованием класса SqlCommand (или другого провайдера, если не Sql Server), изменения тоже.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 11.
Программирование с использованием ParallelExtensions

Цель работы: приобрести начальные знания о ParallelExtensions и научиться работать с ней.

Теоретическая часть

Parallel Extensions to the .NET Framework (другие названия - Parallel FX Library, PFX) - это библиотека, разработанная фирмой Microsoft, для использования в программах на базе управляемого (managed) кода. Она позволяет распараллеливать задачи, в которых могут использоваться специальные - координирующие (coordinating) - структуры данных. Тем самым, библиотека PFX упрощает написание параллельных программ, обеспечивая увеличение производительности при увеличении числа ядер или числа процессоров, исключая многие сложности современных моделей параллельного программирования. Первая версия библиотеки была представлена 29 ноября 2007 года, впоследствии выходили обновления в декабре 2007 и июне 2008 годов. На момент написания данных лекций, библиотека PFX вошла в состав .NET 4 CTP и Visual Studio 2008/2010.

Parallel Extensions обеспечивает несколько новых способов организации параллелизма:

- Параллелизм при декларативной обработке данных. Реализуется при помощи параллельного интегрированного языка запросов (PLINQ) - параллельной реализации LINQ. Отличие от LINQ заключается в том, что запросы выполняются параллельно, обеспечивая масштабируемость и загрузку доступных ядер и процессоров.
- Параллелизм при императивной обработке данных. Реализуется при помощи библиотечных реализаций параллельных вариантов основных итеративных операций над данными, таких как циклы for и foreach. Их выполнение автоматически распределяется на все доступные ядра/процессоры вычислительной системы.

- Параллелизм на уровне задач. Библиотека Parallel Extensions обеспечивает высокоуровневую работу с пулом рабочих потоков, позволяя явно структурировать параллельно исполняющийся код с помощью легковесных задач. Планировщик библиотеки Parallel Extensions выполняет диспетчеризацию и управление исполнением этих задач, а также единообразный механизм обработки исключительных ситуаций.

Требования

Parallel Extensions - это управляемая (managed) библиотека и для своей работы она требует установленный .NET Framework 3.5.

Лучший способ использования Parallel Extensions

Библиотека Parallel Extensions была разработана в целях обеспечения наибольшей производительности при использовании многоядерных процессоров или многопроцессорных машин. Вместо того чтобы принимать решение о верхней границе количества распараллеливаемых задач во время разработки, библиотека позволяет автоматически масштабировать выполняемые задачи, динамически вовлекая в работу все большее количество ядер, по мере того как они становятся доступными. Прирост производительности достигается при использовании Parallel Extensions на многоядерных процессорах или многопроцессорных машинах. Вместе с этим Parallel Extensions разработана так, чтобы минимизировать издержки и при выполнении на однопроцессорных машинах.

Как начать программировать с использованием Parallel Extensions

Для того чтобы начать разрабатывать программы с применением Parallel Extensions необходимо выполнить следующие шаги:

1. Установить Parallel Extensions to the .Net Framework
2. Запустить Visual Studio
3. Создать новый проект
4. Добавить в проект ссылку на библиотеку System.Threading.dll

Microsoft Parallel Extensions for .Net состоитиздвухкомпонент:

- TPL (Task Parallel Library);

- PLINQ (Parallel Language-Integrated Query).

А также содержит набор координирующих структур данных, используемых для синхронизации и координации выполнения параллельных задач.

Основная, базовая концепция Parallel Extensions - это задача (Task) - небольшой участок кода, представленный лямбда-функцией, который может выполняться независимо от остальных задач. И PLINQ, и TPL API предоставляют методы для создания задач - PLINQ разбивает запрос на небольшие задачи, а методы TPL API - Parallel.For, Parallel.ForEach и Parallel.Invoke - разбивают цикл или последовательность блоков кода на задачи.

Parallel Extensions содержит менеджер задач, который планирует выполнение задач. Другое название менеджера задач - планировщик. Планировщик управляет множеством рабочих потоков, на которых происходит выполнение задач. По умолчанию, создается число потоков равное числу процессоров (ядер). Каждый поток связан со своей очередью задач. По завершении выполнения очередной задачи, поток извлекает следующую задачу из своей локальной очереди. Если же она пуста, то он может взять для исполнения задачу, находящуюся в локальной очереди другого рабочего потока. Задачи выполняются независимо друг от друга. Поэтому при использовании ими разделяемых ресурсов требуется выполнять синхронизацию при помощи блокировок или других конструкций.

TPL (Task Parallel Library)

Перед использованием TPL в своем приложении, убедитесь, что ваш проект содержит ссылку на System.Threading.dll. Возможности TPL сосредоточены в двух пространствах имен этой библиотеки: System.Threading и System.Threading.Tasks. В них определены три простых типа:

- System.Threading.Parallel
- System.Threading.Tasks.Task
- System.Threading.Tasks.Future<T>

System.Threading.Parallel

Класс System.Threading.Parallel позволяет распараллеливать циклы и последовательность блоков кода в приложениях .Net. Эта функциональность реализована как набор статических методов, а именно методов For, ForEach и Invoke.

Parallel.For и Parallel.ForEach

Конструкции Parallel.For и Parallel.ForEach являются параллельными аналогами циклов for и foreach. Их использование корректно в случае независимого исполнения итераций цикла, то есть, если ни в одной итерации не используется результаты работы с предыдущих итераций. Тогда эти итерации могут быть выполнены параллельно.

Parallel.Invoke

Статический метод Invoke в параллельной реализации позволяет распараллелить исполнение блоков операторов. Часто в приложениях существуют такие последовательности операторов, для которых не имеет значения порядок выполнения операторов внутри них. В таких случаях, вместо последовательного выполнения операторов одного за другим, возможно их параллельное выполнение, позволяющее повысить производительность. Подобные ситуации часто возникают в рекурсивных алгоритмах и алгоритмах типа "разделяй и властвуй".

System.Threading.Tasks.Task

С помощью методов этого класса возможно организовать асинхронное выполнение нескольких методов, включая ожидание завершения и прерывание их работы. Более подробно работа с данным классом описана в Лекции 5.

System.Threading.Tasks.Future<T>

С помощью этого класса также возможно асинхронно выполнять несколько методов. При этом эти методы могут возвращать результат работы.

PLINQ (ParallelLanguage-IntegratedQuery)

PLINQ (ParallelLanguage-IntegratedQuery) - параллельный интегрированный язык запросов, т.е., это параллельная реализация LINQ.

Традиционно, запросы в языках программирования представлялись просто как значения строкового типа и, как следствие, не осуществлялось проверки

типов для них на этапе компиляции. LINQ сделал запросы языковой конструкцией самих языков C# и Visual Basic. Стало возможным разрабатывать запросы, не задумываясь над типом коллекции, используя ключевые слова языка и знакомые операторы.

Отличие PLINQ от LINQ состоит в том, что запросы выполняются параллельно, используя все доступные ядра и процессоры.

Более подробно работа с PLINQ описана в Лекции 7.

Координирующие структуры данных

Пространство имен System.Threading стандартной библиотеки классов .NET Framework 3.5 содержит множество низкоуровневых примитивов синхронизации, таких как мьютексы, мониторы и события. Библиотека PFX предлагает к использованию более высокоуровневые примитивы, а именно:

1. System.Threading.CountdownEvent
2. System.Threading.LazyInit<T>
3. System.Threading.ManualResetEventSlim
4. System.Threading.SemaphoreSlim
5. System.Threading.SpinLock
6. System.Threading.SpinWait
7. System.Threading.WriteOnce<T>
8. System.Threading.Collections.BlockingCollection<T>
9. System.Threading.Collections.ConcurrentQueue<T>
10. System.Threading.Collections.ConcurrentStack<T>

Наиболее распространенные причины низкой производительности параллельных программ

Параллельное программирование представляет собой написание программ, которые могут исполняться на нескольких ядрах/процессорах, что, в общем случае, должно вести к сокращению времени решения задач. Однако, на практике, в частности, при использовании библиотеки PFX, возможны ситуации когда с увеличением количества используемых процессоров не удается добиться такого уменьшения - в этом случае говорят о низкой производительности параллельной программы.

Причины таких ситуаций могут быть различны:

- задача, как таковая, плохо поддается распараллеливанию,
- неправильное использование библиотеки PFX для

распараллеливания приложения, и др.

Существует ряд общих ситуаций, которые приводят к низкой эффективности параллельных программ, в частности, использующих библиотеку PFX. Ниже приведено краткое описание некоторых из таких ситуаций.

Достаточный объем вычислительной работы

Одним из основных требований, которым должна удовлетворять программа для того, чтобы была возможность ее эффективного распараллеливания, является наличие достаточного объема работы, который она должна выполнить. Допустим, что только половина всех вычислений в программе может быть выполнена параллельно, тогда согласно закону Амдала множитель производительности будет не более двух. Данное правило вполне очевидно, если представить себе, что программа 90% времени исполнения находится в ожидании ответа на SQL-запрос - параллельное исполнение оставшихся 10% не принесет существенного эффекта. Вместе с тем стоит отметить, что в этом случае мы можем параллельно посылать SQL-запросы разным серверам и использовать асинхронное взаимодействие с серверами.

Одним словом, для того чтобы получить эффект от параллельного выполнения задач, нужно иметь достаточный объем работы для распараллеливания, как минимум перекрывающий издержки самого процесса распараллеливания.

Гранулярность задач (размер отдельных задач и их общее количество)

Параллельная программа, решающая общую задачу, обычно состоит из отдельных фрагментов - подзадач, которые определяет в коде сам программист. Если количество подзадач будет слишком велико, то большинство из них будут простаивать в очереди к ядрам процессора, а объем издержек на запуск подзадач будет расти. Если количество подзадач будет слишком малым, то некоторые

ядра процессора будут простаивать, снижая общую производительность системы.

Некоторые компоненты Parallel Extensions API, такие как, например, Parallel.For и PLINQ, способны сами определить подходящий для данной системы объем и количество параллельно исполняющихся задач, тогда как при работе с такими компонентами как TPL и Futures ответственность за принятие правильного решения лежит на самом программисте.

Балансировка нагрузки

Даже в том случае, когда гранулярность задач подобрана верно, возникает проблема распределения этих задач по ядрам. Эта проблема связана с тем, что потенциально разные задачи могут исполняться разное (зачастую неопределенное) время и, следовательно, в процессе исполнения программы без балансировки нагрузки ядер возможны простои некоторых ядер и простои задач в очередях к другим ядрам.

Так, например, если Parallel.For будет предполагать, что все итерации распараллеливаемого цикла исполняются одинаковое время, то мы можем просто разбить пространство итераций на блоки по числу доступных ядер и параллельно исполнить эти блоки. Однако в действительности длительность каждой итерации может быть различной, что ведет к снижению производительности в отсутствии балансировки. В действительности, компонент Parallel.For в PFX реализует автоматическую балансировку, которая во многих случаях решает эту проблему.

Выделение памяти и сборка мусора

Некоторые программы характеризуются интенсивным взаимодействием с памятью, что вызывает значительные временные затраты как на выделение памяти, так и на сборку мусора. К примеру, программа, которая обрабатывает текст, будет интенсивно использовать память, в особенности, если программист не обратил должного внимания на косвенные выделения памяти.

К сожалению, выделение памяти на современных вычислительных архитектурах это операция, которая может требовать синхронизации между вычислительными потоками, в которых выполняется эта операция. Как

минимум мы должны быть уверены, что области памяти, выделяемые параллельно, не перекрываются.

Более серьезные временные потери возникают при сборке мусора. Если сборщик мусора находится в постоянной активности при исполнении программы, то это может стать узким местом при ее распараллеливании. Разумеется, сборщик мусора в .NET может выполнять свою работу параллельно с остальными задачами программы, но для этого могут потребоваться дополнительные усилия со стороны программиста.

Кэш-промахи

Данная ситуация связана с принципами кэширования в современных компьютерах. Когда процессор производит выборку значения из основной памяти, копия этого значения попадает в кэш, что ускоряет доступ к этому значению, если, конечно, оно понадобится еще раз в последующих вычислениях. На самом деле, процессор кэширует не только одно выбранное значение, но и некоторую окрестность этого значения в памяти. Значения, перемещаемые из основной памяти в кэш и сохраняемые в кэш построчно, называются строками кэша, и типичный размер составляет 64 или 128 байт.

Одной из проблем, связанных с кэшированием на многоядерных компьютерах, является ситуация при которой одно из ядер записывает значение в определенную область памяти, находящуюся в кэше другого ядра. В этом случае другое ядро при обращении к соответствующей строке кэша получит ситуацию промаха и будет вынуждено обновить эту строку из основной памяти. С некоторой вероятностью может сложиться ситуация при которой одно ядро постоянно пишет в основную память, нарушая тем самым целостность кэша второго ядра, которое будет вынуждено постоянно обновлять свой кэш, что приведет к резкому падению производительности.

Более тонкая проблема возникает, когда ядра записывают значения в разные области памяти, которые, тем не менее, расположены так близко, что находятся в одной строке кэша. Несмотря на кажущуюся малую вероятность такой ситуации, она достаточно часто встречается на практике, поскольку,

например, поля объекта или промежуточные результаты обработки массивов объектов зачастую находятся в памяти относительно близко.

Есть несколько способов избежать подобных проблем. Можно, например, проектировать структуры данных специальным образом, снижая вероятность появления проблем кэширования, или выделять память в разных потоках.

Кроме того, проектируя процессы параллельной обработки необходимо уже на алгоритмическом уровне сохранять локальность обрабатываемых данных относительно потока. Так если, например, стоит задача обработать массив чисел, то очевидно, что локальность обработки каждым ядром своей половины массива будет гораздо выше, чем локальность обработки одним ядром элементов массива с четными индексами, а вторым - с нечетными. В последнем случае только половина элементов строки кэша ядра будет содержать нужные данному ядру элементы.

Задание для самостоятельной работы:

Составить описание пространства имён:

1.System.Threading.CountdownEvent

2.System.Threading.LazyInit<T>

3.System.Threading.ManualResetEventSlim

4.System.Threading.SemaphoreSlim

5.System.Threading.SpinLock

6.System.Threading.SpinWait

7.System.Threading.WriteOnce<T>

8.System.Threading.Collections.BlockingCollection<T>

9.System.Threading.Collections.ConcurrentQueue<T>

10.System.Threading.Collections.ConcurrentStack<T>

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное

обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования С.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторные работа 12.
Вариант Parallel.For с локальными состояниями

Цель работы: познакомиться с конструкциями Parallel.For и её видами, научиться работать с ними.

Теоретическая часть

Конструкции Parallel.For/ForEach имеют несколько перегруженных (overloaded) вариантов, с помощью которых можно организовать передачу информации между итерациями цикла, исполняющимися в одном потоке. Вспомним, что при выполнении Parallel.For/ForEach каждый рабочий поток выполняет, в общем случае, несколько итераций. Например, при общем количестве итераций 100, на 4-хядерной машине, обычно будет запущено 4 потока, каждый из которых выполнит 25 итераций. Обычно, передача информации (например, подсчет какого-либо значения) между итерациями, исполняющимися в одном потоке, нужна для вычисления некоторого сводного значения по всем итерациям цикла.

Одним из вариантов перегруженной конструкции Parallel.For для поддержки вычислений такого рода выглядит так:

```
public static void For<TLocal> (
int fromInclusive, int toExclusive,
    Func<TLocal> threadLocalInit,
    Action<int, ParallelState<TLocal>> body,
Action<TLocal> threadLocalFinally);
```

Здесь тип TLocal задает тип объекта, с которым будут работать итерации в рамках одного потока. Делегат threadLocalInit вызывается один раз для каждого потока, выделенного для исполнения группы итераций цикла, перед тем как этот поток начал исполнять свои итерации. Обычно с помощью этого делегата создается или инициализируется объект типа TLocal с которыми будут работать итерации.

Результат каждой итерации может быть записан в объект класса ParallelState<TLocal> который является одним из встроенных классов

библиотеки PFX), передаваемый в качестве параметра в делегат body. Делегат body может обновлять специальное поле (а точнее, свойство) ThreadLocalState, которое имеет объект ParallelState<TLocal>, и которое имеет тип TLocal, и значение этого поля будет доступно следующим итерациям, исполняющимся в данном потоке. После того, как поток завершил исполнение своих итераций, в рамках его же, будет вызван делегат threadLocalFinally, которому в качестве параметра будет передано значение ThreadLocalState.

Примером использования одного из перегруженных вариантов конструкции Parallel.For может служить следующий код:

```
Parallel.For(0, N, () => new NonThreadSafeData(), (i,loop)= >
{
UseData(loop.ThreadLocalState);
});
```

(Здесьнеиспользованделегат threadLocalFinallyизвышеприведеннойсигнатуры).

В данном примере для каждого нового потока в рамках Parallel.For будет создан свой объект класса NonThreadSafeData (т.е., в данном случае TLocal = NonThreadSafeData). Таким образом, объект NonThreadSafeData будет передаваться между итерациями одного потока, что позволит потоку безопасно использовать этот объект. При этом, таких объектов будет создано ровно столько, сколько потоков будет запущено. Отметим также, что локальная переменная loop в приведенном выше примере имеет тип ParallelState<TLocal> .

Полный пример вычисления некоторого сводного значения по совокупности данных с использованием объектов-состояний приведен ниже. Такое использование локальных по отношению к потокам объектов позволяет избежать применения блокировок при доступе к общему объекту на каждой итерации, вычислить промежуточные значения и затем объединить их с помощью делегата threadLocalFinally, используя блокирующую конструкцию:

```
int total = 0;
Parallel.ForEach(data, ()=>0, (elem,i,loop)= >
{
```

```
    loop.ThreadLocalState += Process(elem);
},
partial => Interlocked.Add(ref total, partial));
```

Отметим, что в данном случае локальная переменная partial имеет тип TLocal, который, в данном примере, есть тип int (см. делегат threadLocalInit).

Если требуется передавать между итерациями несколько общих переменных, то, как обычно делают в таких случаях, можно создать собственный класс из этих переменных, и уже его передавать как ThreadLocalState:

```
class MultipleValues { public int Total, Count; }

int total = 0, count = 0;
Parallel.ForEach(data,
()=>new MultipleValues { Total=0, Count=0 },
   (elem,i,loop)= >
{
   loop.ThreadLocalState.Total += Process(elem);
loop.ThreadLocalState.Count++;
},
partial => {
Interlocked.Add(ref total, partial.Total);
Interlocked.Add(ref count, partial.Count);
})
```

Задания для самостоятельной работы:

Реализуйте подсчет суммы элементов массива с использованием предлагаемого подхода, кроме того, подсчитайте, сколько потоков было запущено при исполнении Parallel.For/ForEach в вашей программе.

Изучите понятие "анонимный класс" языка C# и выясните можно ли использовать анонимные классы вместо явного определения класса MultipleValues в приведенном выше примере,и если нельзя, то почему.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 13.
Пользовательские расширения Parallel.For

Цель работы: закрепить знания о конструкции Parallel.For и познакомиться с её пользовательскими расширениями.

Теоретическая часть

Иногда в циклах, используемых в программах, итерации являются независимыми; т.е., ни в одной итерации не используются результаты работы предыдущих итераций. Посредством класса System.Threading.Parallel такие циклы могут быть преобразованы в циклы, в которых каждая итерация потенциально может быть выполнена параллельно при наличии достаточного количества доступных ядер (процессоров). Заметим, что параллельное исполнение циклов, итерации которых не являются независимыми, может привести к некорректным результатам. В этом случае необходимо либо пересмотреть структуру цикла, либо использовать примитивы синхронизации и\или потокобезопасные структуры данных.

В PFX существует несколько вариантов метода For, исполняющихся параллельно. Наиболее часто применяемым является For(Int32, Int32, Action<Int32>). Здесь первый параметр задает начальный индекс, второй параметр - конечный индекс, и третий параметр - делегат, определяющий действие, которое будет выполняться на каждой итерации.

Рассмотрим следующий последовательный цикл на C#:

```
for (int i = 0; i < N; i++)
{
results[i] = Compute(i);
}
```

СпомощьюPFX и класса System.Threading.Parallel можно распараллелить этот цикл следующим образом:

```
using System.Threading;
…
```

```
Parallel.For(0, N, delegate(int i)
{
results[i] = Compute(i);
});
```

Для упрощения кода можно использовать синтаксис лямбда-выражений, введенный в C# 3.0 (Visual Studio 2008):

```
using System.Threading;
…
Parallel.For(0, N, i =>
{
results[i] = Compute(i);
});
```

Теперь итерации этого цикла могут быть выполнены параллельно.

Распараллеливание циклов foreach выполняется аналогичным образом. РассмотримциклнаC#:

```
foreach(MyClass c in data)
{
  Compute(c);
}
```

С помощью PFX он распараллеливается следующим образом:

```
Parallel.Foreach(data, delegate(MyClass c))
{
  Compute(c);
}
```

Упрощенный код с использованием лямбда-выражений выглядит так:

```
Parallel.Foreach(data, c =>
{
  Compute(c);
});
```

Использование foreach, как правило, менее эффективно, чем for, поскольку несколько потоков должны иметь доступ к общей обрабатываемой структуре

данных, например, списку. Однако, реализация Parallel.ForEach достаточно интеллектуальна, и в ней доступ из нескольких потоков к общей структуре данных происходит достаточно эффективно.

В качестве примера использования Parallel.For рассмотрим задачу перемножения двух матриц. Последовательная реализация этой задачи может выглядеть так:

```
voidMultiplyMatrices(intsize, double[,] m1, double[,] m2, double[,] result)
{
for (int i = 0; i < size; i++)
  {
for (int j = 0; j < size; j++)
   {
result[i,j] = 0;
for (int k = 0; k < size; k++)
{
result[i,j] += m1[i,k] * m2[k,j];
}
}
  }
}
```

Распараллеливание заключается в простой замене внешнего цикла For циклом Parallel.For:

```
void  MultiplyMatrices(int size, double[,] m1, double[,] m2, double[,] result)
{
  Parallel.For (0; size; i =>
  {
for (int j = 0; j < size; j++)
   {
result[i,j] = 0;
for (int k = 0; k < size; k++)
```

```
{
result[i,j] += m1[i,k] * m2[k,j];
}
}
  });
}
```

Также можно выполнить распараллеливание внутреннего (по j) цикла, но только для матриц достаточно большого размера. Распараллеливание внешнего цикла дает достаточную степень параллельности, чтобы получить от нее эффект. Необдуманное использование Parallel.For может нанести ущерб производительности, поэтому внутренний цикл оставлен последовательным.

Еще один пример применения Parallel.ForEach - перечисление всех файлов изображений в директории и обработка каждого из них:

```
foreach(string imagePath in Directory.GetFiles(path, "*.jpg"))
{
  ProcessImage(imagePath);
}
```

Параллельный вариант выглядит следующим образом:

```
Parallel.ForEach(Directory.GetFiles(path, "*.jpg"), imagePath =>
{
  ProcessImage(imagePath);
});
```

Задания для самостоятельной работы:

Реализуйте подсчет суммы элементов массива с использованием предлагаемого подхода, кроме того, подсчитайте, сколько потоков было запущено при исполнении Parallel.For/ForEach в вашей программе.

Изучите понятие "анонимный класс" языка C# и выясните можно ли использовать анонимные классы вместо явного определения класса MultipleValues в приведенном выше примере,и если нельзя, то почему.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками:

процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 14.
Рекурсия и параллелизм

Цель работы: ознакомиться с понятиями рекурсии и параллелизма, их отличиями и путями решения проблем этих методов.

Теоретическая часть

При изучении применения рекурсии в программировании, одним из наиболее типичных примеров является обработка структур данных, представленных в виде дерева. При этом, при обработке деревьев используются различные способы прохождения по их вершинам. Однако, при попытках параллельной (многопоточной) обработки деревьев возникают проблемы, которые отсутствуют при обычной, последовательной обработке.

Рассмотрим простую структуру данных - бинарное дерево (Tree):

```
class Tree<T>
  {
  public Tree<T> Left, Right; // потомки
  public T Data; // данные для этого узла
  }
```

Пример 4.1.

Предположим, что нам необходимо обработать каждую вершину дерева с помощью действия Action<T>, не заботясь о порядке в котором обрабатываются вершины. В последовательном, рекурсивном варианте это сделать очень легко:

```
publicstaticvoidProcess<T> (Tree<T>tree, Action<T>action)
{
  if (tree == null) return;
// Обработать текущую вершину, затем левое поддерево,
// а потом правое

action(tree.Data);
Process(tree.Left, action);
```

```
Process(tree.Right, action);
}
```

Пример 4.2.

Задача 1.

Реализуйте класс Tree, представляющий дерево с произвольной степенью ветвления (т.е., не только со степенью 2). Переделайте соответственно процедуру Process.

Нерекурсивный вариант с явным использованием стека (объекта класса Stack) может выглядеть следующим образом:

```
public static void Process<T> (Tree<T> tree, Action<T> action)
{
if (tree == null) return;
var toExplore = new Stack<Tree<T>> ();

// Начало обработки с корневой вершины
   toExplore.Push(tree);
while (toExplore.Count > 0)
   {
   // Извлечь очередную вершину, обработать ее, поместить в
// стекеепотомков

var current = toExplore.Pop();
action(current.Data);
if (current.Left != null)
toExplore.Push(current.Left);
if (current.Right != null)
toExplore.Push(current.Right);
   }
}
```

Пример 4.3.

Задача 2.

Переписать приведенный выше метод Process с использованием класса Queue вместо Stack.

Для перехода к параллельному варианту обработки вершин дерева, предположим, что само действие является независимым, вычислительно сложным (т.е., параллельная обработка вершин дерева имеет смысл) и безопасным для использования в многопоточной среде.

С использованием средств .NET Framework, имеются различные возможности для реализации такого параллельного варианта. Ниже приведена реализация, следующая оригинальному рекурсивному алгоритму и использующая класс ThreadPool:

```
public static void Process<T> (Tree<T> tree, Action<T> action)
{
if (tree == null) return;

// Событие mre используется, чтобы реализовать возврат из
// данного метода только после того, как будет закончена
// обработкавершин-потомков

using (var mre = new ManualResetEvent(false))
   {
// Обработатьлевогопотомкаасинхронно

ThreadPool.QueueUserWorkItem(delegate
      {
Process(tree.Left, action);
mre.Set();
});

// Обработать текущую вершину и правого потомка синхронно
```

```
action(tree.Data);
Process(tree.Right, action);

// Ожидание окончания обработки левого потомка

      mre.WaitOne();
   }
}
```

Пример 4.4.

Задача 3.

Показать, как аналогичный параллельный вариант можно реализовать, используя класс Thread.

Недостаток предыдущего варианта заключался в том, что обработка правого потомка задерживалась до тех пор, пока не будут обработаны данные текущей вершины. Чтобы повысить степень параллелизма метода Process, можно его модифицировать следующим образом:

```
public static void Process<T> (Tree<T> tree, Action<T> action)
{
if (tree == null) return;

// Необходимо воспользоваться конструкцией события
// для ожидания завершения обработки потомков

using (var mre = new ManualResetEvent(false))
   {
int count = 2;

// Обработать левого потомка асинхронно

ThreadPool.QueueUserWorkItem(delegate
```

```
        {
Process(tree.Left, action);
if (Interlocked.Decrement(ref count) == 0)
mre.Set();
        });

// Обработать правого потомка асинхронно

ThreadPool.QueueUserWorkItem(delegate
        {
Process(tree.Right, action);
if (Interlocked.Decrement(ref count) == 0)
mre.Set();
        });

// Обработать текущую вершину синхронно

action(tree.Data);

// Ожидание завершения обработки потомков

        mre.WaitOne();
    }
}
```

Пример 4.5

Задача 4.

Переписать приведенный выше метод Process в предположении, что класс Tree определен так, как в Задаче 1. Подготовьте несколько объектов класса Tree с количеством вершин, соответственно, 100, 200, 500, 1000, и протестируйте метод Process на данных деревьях.

Задания для самостоятельной работы:

1. Напишите рекурсивную функцию summ(s,i, j), проверяющую, является ли симметричной часть строки s, начинающаяся i-м и заканчивающаяся j-м ее элементами.

2. Напишите рекурсивную функцию, "переворачивающую" заданное натуральное число.

3. Напишите рекурсивную функцию, которая по последовательности литер, представляющих двоичное число, определяет соответствующее десятичное число.

4. Напишите рекурсивную функцию, вычисляющую *n!!*.

5. Напишите рекурсивную функцию, определяющую количество единиц в двоичном представлении натурального числа.

6. Напишите рекурсивную функцию, которая по заданным натуральным числам *m* и *n* выводит все различные представления числа « в виде суммы *m* натуральных слагаемых. Представления, различающиеся лишь порядком слагаемых, считаются одинаковыми.

7. Напишите рекурсивную функцию, которая формирует в обратном порядке текст, задаваемый пользователем в текстовом поле формы.

8. Напишите рекурсивную функцию для вычисления биномиального коэффициента C.

9. Напишите рекурсивную функцию, проверяющую, является ли последовательность символов идентификатором. Напомним, что идентификатором будем считать последовательность букв и/или цифр, начинающуюся с буквы.

10. В последовательности символов могут встречаться только цифры и знаки + (плюс), при этом последовательность представляет собой формулу сложения однозначных чисел. Напишите рекурсивную функцию, определяющую значение формулы.

11. В последовательности символов могут встречаться только цифры и знаки + (плюс) или - (минус), при этом последовательность представляет собой

формулу арифметической суммы однозначных чисел. Напишите рекурсивную функцию, определяющую значение формулы.

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Лабораторная работа 15.
Асинхронная модель программирования и класс Future<T>

Цель работы: реализовать "асинхронную модель программирования" (АМП) с использованием объектов класса Future<T>, и создать объекты класса Future<T> с использованием имеющейся реализации АМП.

Теоретическая часть

Базовыми конструкциями, лежащими в основе АМП, являются

1. метод BeginXx, с помощью которого запускается асинхронная операция, и который возвращает объект типа IAsyncResult, и
2. метод EndXx, который принимает в качестве входного аргумента IAsyncResult и возвращает вычисленное значение.

Для конкретных задач, для которых асинхронные операции являются вычислительно сложными (т.е., выполнение этих операций требует большого количества вычислительных операций), эти операции могут быть реализованы с помощью объектов класса

System.Threading.Tasks.Future<T>,

поскольку класс Future<T> наследует класс

System.Threading.Tasks.Task

и реализует интерфейс IAsyncResult (отметим, что интерфейс IAsyncResult является в .NET, в действительности, объектом типа System.Runtime.Remoting.Messaging.AsyncResult).

Предположим, что мы имеем класс, предназначенный для вычисления числа - с произвольной точностью, задаваемой количеством требуемых десятичных знаков после запятой.

```
public class PI
  {
public string Calculate (int decimalPlaces )
   {
     . . .
   }
```

```
  }
```

Пример 6.1.

```
public class PI
  {
public string Calculate (int decimalPlaces )
{
     . . .
    }
  }
```

Реализовать АМП, в данном случае, означает ввести в класс PI методы BeginCalculate и EndCalculate, позволяющие использовать метод Calculate асинхронным образом. Используя Future<T>, это можно сделать, добавив лишь несколько строк кода:

```
public class PI
  { . . .
public IAsyncResult BeginCalculate (int decimalPlaces)
    {
return Future.Create ( () => Calculate(decimalPlaces) );
    }
public string EndCalculate ( IAsyncResult ar )
    {
var f = ar as Future<string>;
if ( f == null )
throw new ArgumentException ("ar" );
return f.Value;
    }
  }
```

Пример 6.2.

```
public class PI
  {  . . .
public IAsyncResult BeginCalculate (int decimalPlaces)
    {
return Future.Create ( () => Calculate(decimalPlaces) );
    }
public string EndCalculate ( IAsyncResult ar )
    {
var f = ar as Future<string>;
if ( f == null )
throw new ArgumentException ("ar" );
return f.Value;
    }
  }
```

На самом деле, в классической асинхронной модели программирования требуется также, чтобы метод BeginXx мог принимать еще 2 аргумента:

1. функцию "обратного вызова" (AsyncCallback), которая автоматически вызывается по завершении асинхронной операции, и
2. объект, задающий состояние программы пользователя в момент асинхронного вызова (user-defined state object).

Добавление функции AsyncCallback очень легко реализовать, используя возможности, предоставляемые классом Future<T>:

```
public IAsyncResult BeginCalculate (int decimalPlaces,
                                    AsyncCallback ac )
  {
var f = Future.Create ( () => Calculate (decimalPlaces) );
if ( ac != null )
    f.Completed += delegate { ac ( f ); };
return f;
  }
```

Пример 6.3.

```
public IAsyncResult BeginCalculate (int decimalPlaces,
                                    AsyncCallback ac )
  {
var f = Future.Create ( () => Calculate (decimalPlaces) );
if ( ac != null )
   f.Completed += delegate { ac ( f ); };
return f;
  }
```

Решение, реализованное в пример 6.3, состоит в регистрации для объекта класса Future<T> события Completed, при наступлении которого теперь будет вызываться функция AsyncCallback.

Задания для самостоятельной работы:

Задача 1.

Реализуйте полностью класс PI с методами BeginCalculate и EndCalculate, добавив к нему соответствующий метод Main для проверки всего решения.

Задача 2.

Реализуйте метод Main для данного класса, в котором асинхронным образом пингуется несколько хостов, имена которых заданы в виде списка или массива.

Задача 3.

Рассмотрите пример 6.3 и выясните может ли произойти такая ситуация, когда к моменту регистрации события Completed, выполнение функции, связанной с Future, уже закончится, а потому не будет произведен необходимый вызов функции AsyncCallback

Задача 4.

Написать метод Main, в котором попеременно асинхронно читаются 2 файла с использованием объектов Future<T>, аналогичных показанному в пример 6.10.

Задача 5.

Реализуйте метод Main для использования функции Create из пример 6.11, и распечатайте ID главного потока и потока в рамках которого будет выполняться присваивание

f.Value = endFunc(iar);

Аппаратура и материалы. Для выполнения лабораторной работы необходим персональный компьютер со следующими характеристиками: процессор Intel Pentium-совместимый с тактовой частотой 800 МГц и выше, оперативная память – не менее 64 Мбайт, свободное дисковое пространство – не менее 500 Мбайт, устройство для чтения компакт-дисков, монитор типа Super VGA (число цветов от 256) с диагональю не менее 15 дюймов. Программное обеспечение – операционная система Windows2000/XP и выше, интерпретатор языка программирования C#.

Указания по технике безопасности. Техника безопасности при выполнении лабораторной работы совпадает с общепринятой для пользователей персональных компьютеров, самостоятельно не производить ремонт персонального компьютера, установку и удаление программного обеспечения; в случае неисправности персонального компьютера сообщить об этом обслуживающему персоналу лаборатории (оператору, администратору); соблюдать правила техники безопасности при работе с электрооборудованием; не касаться электрических розеток металлическими предметами; рабочее место пользователя персонального компьютера должно содержаться в чистоте; не разрешается возле персонального компьютера принимать пищу, напитки.

Методика и порядок выполнения работы. Проработать примеры лабораторной работы выполнить индивидуальные задания.

Содержание отчета и его форма. Отчет по лабораторной работе должен состоять из:

1. Названия лабораторной работы.
2. Цели и содержания лабораторной работы.
3. Ответов на контрольные вопросы лабораторной работы.
4. Формулировки индивидуального задания и порядка его выполнения.

Отчет о выполнении лабораторной работы в письменном виде сдается преподавателю.

Список использованных источников

1. Методические указания к практическим занятиям по дисциплине "Программирование в корпоративных сетях" для студентов специальности 090105 "Комплексное обеспечение информационной безопасности" / сост. Р. А. Воронкин ; рец. Г. И. Линец. - Ставрополь : СевКавГТУ, 2006. - 131 с. : ил. - Библиогр.: с. 130(4 назв.)

2. Воронкин, Р. А. (СевКавГТУ). Средства обеспечения безопасности корпоративных информационных систем : учеб. пособие (Лабораторный практикум) / Р.А. Воронкин ; Сев.-Кав. гос. техн. ун-т, Ч. 1. - Ставрополь : Изд-во СевКавГТУ, 2010. - 308 с. : ил. - Библиогр.: с. 305

3. Сорокин, А. А. (СевКавГТУ). Проектирование корпоративных информационных систем : учеб. пособие (Курс лекций) / Ал. Ан. Сорокин, Ан. Ал. Сорокин ; ФГБОУ Сев.-Кав. гос. техн. ун-т. – Ставрополь : Издательство СевКавГТУ, 2012. – 439 с.