

Министерство образования и науки Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Магнитогорский государственный технический университет им. Г.И. Носова»  
Многопрофильный колледж



**МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ  
ПРАКТИЧЕСКИХ ЗАНЯТИЙ**

**ПМ. 03. УЧАСТИЕ В ИНТЕГРАЦИИ ПРОГРАММНЫХ МОДУЛЕЙ**

по специальности 09.02.03 Программирование в компьютерных системах  
базовой подготовки

Магнитогорск, 2017

## **ОДОБРЕНО:**

Предметно - цикловой комиссией Информатика и вычислительная техника

Председатель И.Г.Зорина

Протокол № 7 от 14 марта 2017

Методической комиссией МпК

Протокол №4 от «23» марта 2017г

## **Разработчики:**

преподаватель МпК ФГБОУ ВО «МГТУ» Л.А. Фетисова

преподаватель МпК ФГБОУ ВО «МГТУ» И.Г. Зорина

преподаватель МпК ФГБОУ ВО «МГТУ» В.Д.Тугарова

преподаватель МпК ФГБОУ ВО «МГТУ» Ю.Ю.Дорохина

Методические указания по выполнению практических занятий разработаны на основе рабочей программы ПМ. 03. Участие в интеграции программных модулей. Содержание практических работ ориентировано на формирование общих и профессиональных компетенций по программе подготовки специалистов среднего звена по специальности 09.02.03 Программирование в компьютерных системах углубленной подготовки:

МДК03.01. Технология разработки программного обеспечения,

МДК03.02. Инструментальные средства разработки программного обеспечения,

МДК03.03. Документирование и сертификация.

## СОДЕРЖАНИЕ

1 Введение	4
2 Методические указания	6
МДК03.01	
Практическое занятие 1,2,3,4,5	6
Практическое занятие 6,7,8,9,10,11,12	12
Практическое занятие 13,14,15,16,17,18,19,20,21,22,23,24,25,26, 27, 28, 29, 30	17
Практическое занятие 31,32,33,34,35,36,37,38,39	19
МДК03.02	
Практическое занятие 1,2	25
Практическое занятие 3	27
Практическое занятие 4	31
Практическое занятие 5,6	35
Практическое занятие 7	47
Практическое занятие 8	51
Практическое занятие 9,10,11	53
Практическое занятие 12,13	79
Практическое занятие 14,15	88
Практическое занятие 16, 17,18,19	96
Практическое занятие 20	122
Практическое занятие 21,22,23,24	128
Практическое занятие 25,26,27	144
МДК03.03	
Практическое занятие 1,2,3	151
Практическое занятие 4,5,6,7,8	152
Практическое занятие 9,10,11	153

## 1 ВВЕДЕНИЕ

Важную часть теоретической и профессиональной практической подготовки студентов составляют практические занятия.

Состав и содержание практических работ направлены на реализацию действующего федерального государственного образовательного стандарта среднего профессионального образования по специальности 09.02.03 Программирование в компьютерных системах углубленной подготовки.

Ведущей дидактической целью практических занятий является формирование практических умений - профессиональных (умений выполнять определенные действия, операции, необходимые в последующем в профессиональной деятельности) и/или учебных (умений решать задачи по математике, физике, химии, информатике и др.), необходимых в последующей учебной деятельности по профессиональным модулям.

В соответствии с рабочей программой ПМ. 03. Участие в интеграции программных модулей, МДК03.01. Технология разработки программного обеспечения, МДК03.02. Инструментальные средства разработки программного обеспечения, МДК03.03. Документирование и сертификация.

В результате их выполнения, обучающийся должен:

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

Содержание практических занятий ориентировано на формирование общих компетенций по профессиональному модулю основной профессиональной образовательной программы по специальности:

- ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес.
- ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.
- ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность.
- ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.
- ОК 5. Использовать информационно-коммуникационные техно-

- логии в профессиональной деятельности.
- ОК 6. Работать в коллективе и в команде, эффективно общаться с коллегами, руководством, потребителями.
  - ОК 7. Брать на себя ответственность за работу членов команды (подчиненных), за результат выполнения заданий.
  - ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.
  - ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

И овладению профессиональными компетенциями:

ПК3.1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.

ПК3.2. Выполнять интеграцию модулей в программную систему.

ПК3.3. Выполнять отладку программного продукта с использованием специализированных программных средств.

ПК3.4. Осуществлять разработку тестовых наборов и тестовых сценариев.

ПК3.5. Производить инспектирование компонент программного продукта на предмет соответствия стандартам кодирования.

ПК3.6. Разрабатывать технологическую документацию.

Выполнение студентами практических работ по ПМ. 03. Участие в интеграции программных модулей, МДК 03.01. Технология разработки программного обеспечения, МДК03.02. Инструментальные средства разработки программного обеспечения, МДК03.03. Документирование и сертификация.

направлено на:

- обобщение, систематизацию, углубление, закрепление, развитие и детализацию полученных теоретических знаний по конкретным темам междисциплинарных курсов;

- формирование умений применять полученные знания на практике, реализацию единства интеллектуальной и практической деятельности;

- формирование и развитие умений: наблюдать, сравнивать, сопоставлять, анализировать, делать выводы и обобщения;

- развитие интеллектуальных умений у будущих специалистов: аналитических, проектировочных, конструктивных и др.;

- выработку при решении поставленных задач профессионально значимых качеств, таких как самостоятельность, ответственность, точность, творческая инициатива.

Продолжительность выполнения практической работы составляет не менее двух академических часов и проводится после соответствующего занятия, которое обеспечивает наличие знаний, необходимых для ее выполнения.

## 2 МЕТОДИЧЕСКИЕ УКАЗАНИЯ

### РАЗДЕЛ 1. МДК.03.01. ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

#### Тема 03.01.03 Основные подходы к интегрированию программных модулей

##### Практическое занятие № 1,2,3,4,5

Разработка технического задания

- Структура жизненного цикла программы
- Модели жизненного цикла
- Надежность программных продуктов
- Приемы надежного программирования

**Цель работы:** Составить набор тестов для проверки программы, с помощью которой решается предлагаемая задача.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

##### Материальное обеспечение

- персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

*Программное обеспечение:*

- С++ Builder, Borland Delphi.

##### Задание:

Составить тесты для проверки программы, используя методы покрытия операторов и покрытия условий. Сформировать список ошибок, которые могут быть выявлены этими тестами.

Задача: вычислить значение функции:

$$Y = \begin{cases} x^3, & \text{при } x < 0; \\ \sqrt{2x+1}, & \text{при } 0 \leq x \leq 0; \\ \frac{3x-1}{\cos x}, & \text{при } x > 2; \end{cases}$$

Программа должна быть составлена с применением пользовательских процедур и функций.

Спецификация программы:

- Название задачи: функция
- Входные данные:  
    Действительное число – аргумент функции
- Выходные данные:  
    Вычисленное значение функции.

### Порядок выполнения

Нужно составить программу, которая бы в зависимости от введенного с клавиатуры значения аргумента вычисляла бы значение функции. Необходимо предусмотреть контроль ввода значений. Также, необходимо, используя метод покрытия операторов и метод покрытия условий, составить набор тестов, направленных на обнаружение возможных ошибок.

Спецификация переменных:

Имя переменной в программе	Назначение переменной в программе	Тип переменной	Диапазон типа
x	Значение аргумента функции	Real	0..1
Good	Булева переменная, сигнализирующая об успешном преобразовании введенного с клавиатуры значения	Boolean	2.9e-39..1.7e38

Текст программы:

```

program func1;
Uses Crt;
var
  x : real;
  Good : boolean;
procedure Header;
begin
  WriteLn ('Вычислим значение функции');
  WriteLn (' Y = x^3, при x < 0, затем x = x + 1');
  WriteLn (' Y = sqrt(2x+1), при 0 <= x <= 2');
  WriteLn (' 3x-1');
  WriteLn (' Y = _____, при x > 2');
  WriteLn (' Cos x');
  WriteLn;
end; {procedure Header}
procedure ClrPlace;
var
  i : byte;

```



```

begin
  WriteLn;
  WriteLn ('Нажмите для выхода любую клавишу...');
  ReadKey;
  GotoXY(1,8);
  for i := 1 to 5 do
  begin
    ClrEol;
    Writeln;
  end;
  GotoXY(1,8);
end; {procedure ClrPlace}

```

```

procedure Entering;
var
  XStr : string;
  Code : integer;
begin
  Write ('Введите значение X: ');
  Read (XStr);
  Val (XStr,x,Code);
  if Code <> 0 then
  begin
    Good := false;
    WriteLn ('Ошибка в ',Code,' позиции');
    ClrPlace;
  end
  else Good := true;
end; {procedure Entering}

```

```

procedure OutCount(x:real);
var
  i : byte;
begin
  i := 0;
  if x < 0 then
  begin {x < 0}
    WriteLn ('X = ',x);
    WriteLn ('Y = ',x*x*x+1);
  end
  else
    if ((x >= 0) AND (x <= 2)) then

```

```

begin {0<=x<2}
  WriteLn ('X = ',x);
  WriteLn ('Y = ',SQRT(2*x+1));
end
else if x > 2 then
begin {x > 2}
  WriteLn ('X = ',x);
  if Cos(x)=0 then WriteLn ('Ф-ция в этой точке не определена')
  else WriteLn ('Y = ',(3*x-1)/Cos(x));
end;
ReadKey;
end;{procedure OutCount}

```

```

begin
  ClrScr;
  Header;
  Entering;
  if Good = true then OutCount(x);
end.

```

Набор тестов:

*Метод покрытия операторов*

Тест 1

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m > n$
- Входные данные:  $m=3; n=2$

Тест 2

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m = n$
- Входные данные:  $m=3; n=3$

Тест 3

- Определяет, будут ли производиться вычисления, когда входные данные содержат хотя бы одно отрицательное число.
- Входные данные:  $m=-1; n=5$

Тест 4

- Определяет, будут ли производиться вычисления, когда входные данные содержат хотя бы один ноль.
- Входные данные:  $m=0; n=2$

Тест 5

- Определяет, будет ли выполнен оператор `Factor := 1` в функции `Factor`, если во входных данных содержится хотя бы одна единица.

- Входные данные:  $m=1$ ;  $n=5$

#### Тест 6

- Определяет, будет ли выполнен оператор  $\text{Factor} := a * \text{Factor}(a-1)$  в функции  $\text{Factor}$ , если во входных данных содержится хотя бы одно число, удовлетворяющее условию  $a > 1$
- Входные данные:  $m=2$ ;  $n=4$

*Метод покрытия решений*

#### Тест 7

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m > n$
- Входные данные:  $m=3$ ;  $n=2$

#### Тест 8

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m < n$
- Входные данные:  $m=2$ ;  $n=3$

#### Тест 9

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m = n$
- Входные данные:  $m=3$ ;  $n=3$

#### Тест 10

- Определяет, будет ли выполнен оператор вывода на экран сообщения «М должно быть меньше N», когда  $m \neq n$
- Входные данные:  $m=3$ ;  $n=2$

#### Тест 11

- Определяет, будут ли производиться вычисления, когда входные условия удовлетворяют условию  $m < n$
- Входные данные:  $m=1$ ;  $n=5$

#### Тест 12

- Определяет, будут ли производиться вычисления, когда входные условия удовлетворяют условию  $m \geq n$
- Входные данные:  $m=3$ ;  $n=3$

#### Тест 13

- Определяет, будет ли выполнен оператор  $\text{Factor} := 1$  в функции  $\text{Factor}$ , если во входных данных содержится хотя бы одна единица.
- Входные данные:  $m=1$ ;  $n=5$

#### Тест 14

- Определяет, будет ли выполнен оператор  $\text{Factor} := a * \text{Factor}(a-1)$  в функции  $\text{Factor}$ , если во входных данных содержится хотя бы одно число, удовлетворяющее условию  $a > 1$

- Входные данные:  $m=2$ ;  $n=4$

### *Ошибки:*

Тест 1: выявляет ошибку, когда при введенных значениях  $m > n$  вычисления производятся далее.

Тест 2: выявляет ошибку, когда при введенных значениях  $m = n$  вычисления производятся далее.

Тест 3: выявляет ошибку, когда во входных данных находится отрицательное число, а вычисления производятся далее.

Тест 4: выявляет ошибку, когда во входных данных находится ноль, а вычисления производятся далее.

Тест 5: выявляет ошибку, когда находим факториал от единицы, а программа либо заикливается, либо выводит неверный результат.

Тест 6: выявляет ошибку, когда находим факториал числа, большего единицы, а в результате получаем единицу.

Тест 7: выявляет ошибку, когда при введенных значениях  $m > n$  вычисления производятся далее.

Тест 8: выявляет ошибку, когда входные данные удовлетворяют условию  $m < n$ , а вычисления далее не производятся.

Тест 9: выявляет ошибку, когда при введенных значениях  $m = n$  вычисления производятся далее.

Тест 10: выявляет ошибку, когда входные данные удовлетворяют условию  $m \neq n$ , а вычисления далее не производятся.

Тест 11: выявляет ошибку, когда входные данные удовлетворяют условию  $m < n$ , а вычисления далее не производятся.

Тест 12: выявляет ошибку, когда при введенных значениях  $m > n$  вычисления производятся далее.

Тест 13: выявляет ошибку, когда находим факториал от единицы, а программа либо заикливается, либо выводит неверный результат.

Тест 14: выявляет ошибку, когда находим факториал числа, большего единицы, а в результате получаем единицу.

### **Форма представления результата:**

- Алгоритм программы
- Программный код, составленный на языке программирования

### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.01.04. Основные методы и средства эффективной разработки** **Практическое занятие № 6,7,8,9,10,11,12**

Разработка алгоритмов и блок-схем

- Структурное программирование
- Метод восходящей разработки
- Модульное программирование
- Метод нисходящей разработки

**Цель работы:** Изучить основные методы и средства эффективной разработки программных модулей.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

#### **Материальное обеспечение**

- персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением
- Программное обеспечение:*
- С++ Builder, Borland Delphi.

**Задание:**

С использованием собственных, пользовательских процедур и функций, составить программу, в которой вычисляется

$$\sum_{k=2}^N \prod_{i=1}^{k-1} \sin(k)$$

Составить аналитический профиль программы.

Спецификация программы:

- Название задачи: вычисление значения числового ряда
- Название программы: lab5
- Система программирования: Turbo Pascal 7.0
- Системные требования: IBM PC/AT 286 и выше
- Входные данные:  
k – целое число
- Выходные данные:  
result – число типа real, результат вычислений.

**Порядок выполнения работы:**

Нужно составить программу, которая вычисляла бы значение числового ряда при заданном количестве членов (k). Удобнее сделать это, разделив процесс подсчета на два основных этапа: подсчет произведений и подсчет сумм произведений. Для этого необходимо создать соответствующие процедуру и функцию.

Спецификация переменных:

Имя переменной в программе	Назначение переменной в программе	Тип переменной	Диапазон типа
k	Число, определяющее соответствующий член ряда	integer	- 32768..32767
result	Число, накапливающее результат	real	2.9e-39..1.7e38
i	Вспомогательная локальная переменная-счетчик	integer	- 32768..32767
multi	Вспомогательная локальная переменная, накапливающая результат.	real	2.9e-39..1.7e38

Аналитический профиль программы.

№ строки	профиль	аналитический профиль	Текст программы
1.			program lab5;
2.			>
3.			uses Crt;
4.			>
5.			var
6.			k : integer;
7.			result : real;
8.			>
9.			function Add(multi:real):real;
10	1	1	begin
11	1	1	result := result + multi;
12	1	1	end; {function Add}
13			procedure Multiple (k : integer);
14			var
15			i : integer;

16			multi : real;
17	1	1	begin
18	1	1	multi := 1;
19	5	k	for i := 1 to k-1 do
20	4	k-1	begin
21	4	k-1	multi := multi * sin(i);
22	4	k-1	Add(multi);
23	4	k-1	end;
24	1	1	end; {procedure Multiple }
25	1	1	begin
26	1	1	ClrScr;
27	1	1	Write ('Введите k ');
28	1	1	Readln(k);
29	1	1	result := 0;
30	1	1	Multiple(k);
31	1	1	Writeln ('Результат вычислений: ',result);
32	1	1	end.

Профиль программы дан для k = 5

Текст программы:

```
program lab5;
```

```
uses Crt;
```

```
var
```

```
  k   : integer;
```

```
  result : real;
```

```
function Add(multi:real):real;
```

```
begin
```

```
  result := result + multi;
```

```
end; {function Add}
```

```
procedure Multiple (k : integer);
```

```
var
```

```
  i   : integer;
```

```
  multi : real;
```

```
begin
```

```
  multi := 1;
```

```
  for i := 1 to k-1 do
```

```
  begin
```

```
    multi := multi * sin(i);
```

```
    Add(multi);
```

```
end;  
end; {procedure Multiple}  
  
begin  
  ClrScr;  
  Write ('Введите k ');  
  Readln(k);  
  result := 0;  
  Multiple(k);  
  Writeln ('Результат вычислений: ',result);  
end.
```

**Форма представления результата:**

- алгоритм программы;
- программный код, составленный на языке программирования.

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.01.07 Стандарты качества программного обеспечения**

**Практическое занятие № 13, 14, 15, 16 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30**

Корректировка блок – схемы

- назначение и процессы управления разработкой программного средства;
- структура управления разработкой программных средств.

Корректировка и отладка программного кода

- критерии оценки качества программного обеспечения;
- процесс контроля качества.

**Цель работы:** корректировка блок-схемы для создания модулей, процедур и функций. Корректировка и отладка программного кода.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;



- использовать методы для получения кода с заданной функциональностью и степенью качества;

### **Материальное обеспечение**

- персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением

*Программное обеспечение:*

- С++ Builder, Borland Delphi.

### **Задание:**

Составить программу, в которой экран случайным образом заполняется символом \*, выход из программы происходит по нажатию любой клавиши. Составить спецификацию. Использовать стандартные библиотечные функции, например – функцию очистки экрана и др.

Спецификация программы:

- Название задачи: вывод символа в произвольном порядке
- Входные данные: нет
- Выходные данные: двумерный массив, элементы строк которого заполнены символом \*.

### **Порядок выполнения:**

Нужно составить программу, которая создавала бы в области, зарезервированной для видеопамати двумерный массив [1..2000,1..2], значение элементов строк которого соответствовало бы ASCII коду соответствующего символа экрана. Затем создать цикл, в котором происходит проверка ASCII кода символа в случайном месте экрана и, в случае, если это не \*, последующая запись в видеопамать символа \*. Перед началом цикла происходит очистка экрана с помощью стандартной процедуры ClrScr. Программа завершается при нажатии любой клавиши.

Спецификация переменных:

Имя переменной в программе	Назначение переменной в программе	Тип переменной	Диапазон типа
Screen	Массив, содержащий адреса видеопамати	Array of byte	0..255 (каждый элемент)
i	Вспомогательная переменная, содержащая значение индекса строки элемента массива	Word	0..65535

Текст программы:

```
program Lab4;
uses Crt;
var
```

```

Screen : array [1..2000,1..2] of byte absolute $B800:0;
i      : Word;
begin
  ClrScr;
  Randomize;
  Repeat
  begin
    i := round(random(2001));
    if Screen [i,1] <> 42 then
      begin
        Screen [i,1] := 42;
        Delay (100);
      end;
    end;
  until KeyPressed;
end.

```

#### **Форма представления результата:**

- алгоритм программы;
- программный код, составленный на языке программирования.

#### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

#### **Тема 03.01.08 Методы и средства разработки программной документации**

##### **Практическое занятие № 31,32,33,34,35,36,37,38,39**

- Оформление алгоритмов согласно стандартам; Начертание блок-схем на ПК;
- Разработка документации по сопровождению программы.

**Цель работы:** Оформление алгоритмов согласно стандартам. Начертание блок-схем на ПК. Разработка документации по сопровождению программы

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

### **Материальное обеспечение**

- персональные компьютеры в локальной сети с доступом к сети Internet – 11ед. с лицензионным программным и сетевым обеспечением  
*Программное обеспечение:*
- С++ Builder, Borland Delphi.

### **Задание:**

Написать программу, в которой создается случайным образом двумерный С использованием собственных, пользовательских процедур и функций, составить программу, в которой с клавиатуры вводится строка, подсчитывается количество вводимых символов «а» или «А», а также выдается код этих символов по таблице кодов ASCII. На экран выводится исходная строка и результат. Составить спецификацию и аналитический профиль программы.

Спецификация программы:

- Название задачи: подсчет количества символов
- Входные данные: строка символов - SInp
- Выходные данные:

Исходная строка;

Общее количество встречающихся символов «А» или «а»;

ALatBig, aLatSmall, ACyrBig, aCyrSmall – количество во введенной строке символов “А”, “а” (латиницей) и «А», «а» (кириллических) соответственно.

### **Порядок выполнения:**

Нужно составить программу, которая выдавала бы пользователю запрос на ввод с клавиатуры произвольной фразы, подсчитывала бы в ней количество символов «А» и «а» и выдавала количество и ASCII код этих символов и саму исходную фразу на экран.

Спецификация переменных:

Имя переменной в программе	Назначение переменной в программе	Тип переменной	Диапазон типа
SInp	Хранение введенной с клавиатуры строки	String	0..255 (каждый элемент)
ALatBig	Подсчет больших букв «А» латиницей	integer	- 32768..32767
aLatSmall	Подсчет малых букв «а»	integer	-

	латиницей		32768..32767
ACyrBig	Подсчет больших букв «А» кириллицей	integer	- 32768..32767
aCyrSmall	Подсчет малых букв «а» кириллицей	integer	- 32768..32767
i	Счетчик	integer	- 32768..32767

Аналитический профиль программы:

№ строки	Профиль	Аналитический профиль	Текст программы
3.			program LabWork6;
4.			
5.			Uses Crt;
6.			
7.			var
8.			SInp : string;
9.			
10.			Function Code(ch:char):byte;
11.			begin
12.			Code := ord(ch);
13.			end; {Function Code}
14.			Procedure onScreen(ALatBig, aLatSmall, ACyrBig, aCyrSmall : integer);
15.			begin
16.			if ALatBig+aLatSmall+ACyrBig+aCyrSmall <> 0 then
17.			begin
18.			Write ('Символы "А" и "а" встречаются в строке ');
19.			Write (ALatBig+aLatSmall+ACyrBig+aCyrSmall,' раз, ');
20.			WriteLn ('причем:');
21.			WriteLn;
22.			if ALatBig <> 0 then
23.			begin
24.			Write ('Большая латинская "А" (код ASCII ',Code('А'),'));
25.			WriteLn (' встречается ',ALatBig,' раз(a)');

26.			end
27.			else
28.			WriteLn ('Большая латинская "А" не встречается');
29.			if ALatSmall <> 0 then
30.			begin
31.			Write ('Малая латинская "а" (код ASCII ' ,Code('a'),' ');
32.			WriteLn (' встречается ',ALatSmall,' раз(a));
33.			end
34.			else
35.			WriteLn ('Малая латинская "а" не встречается');
36.			if ACyrBig <> 0 then
37.			begin
38.			Write ('Большая русская "А" (код ASCII ' ,Code('A'),' ');
39.			WriteLn (' встречается ',ACyrBig,' раз(a));
40.			end
41.			else
42.			WriteLn ('Большая русская "А" не встречается');
43.			if ACyrSmall <> 0 then
44.			begin
45.			Write ('Малая русская "а" (код ASCII ' ,Code('a'),' ');
46.			WriteLn (' встречается ',ACyrSmall,' раз(a));
47.			end
48.			else
49.			WriteLn ('Малая русская "а" не встречается');
50.			end
51.			else
52.			begin
53.			Write ('Во введенной Вами строке символы ');
54.			WriteLn ('"А" и "а" не встречается.');
55.			end;
56.			end; {Procedure onScreen}

57.			Procedure CharCount(Sinp:string);
58.			var
59.			i, ALatBig, aLatSmall, ACyrBig, aCyrSmall : integer;
60.			begin
61.			ALatBig := 0;
62.			aLatSmall := 0;
63.			ACyrBig := 0;
64.			aCyrSmall := 0;
65.			for i := 1 to Length(Sinp) do
66.			Case SInp[i] of
67.			'A': inc(ALatBig);
68.			'a': inc(aLatSmall);
69.			'А': inc(ACyrBig);
70.			'а': inc(aCyrSmall);
71.			end;
72.			onScreen(ALatBig, aLatSmall, ACyrBig, aCyrSmall);
73.			end; {Procedure CharCount}
74.			begin
75.			ClrScr;
76.			Write ('Введите строку: ');
77.			Read (SInp);
78.			WriteLn('Введена строка: ',SInp);
79.			CharCount(SInp);
80.			ReadKey;
81.			end.

**Форма представления результата:**

Алгоритм программы

Программный код, составленный на языке программирования

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

## РАЗДЕЛ 2. МДК.03.02. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

### Тема 03.02.01 Концепция и реализация программных процессов

**Практическое занятие № 1,2.** Разработка графических и текстовых средств документирования

**Цель работы:** Углубление знаний по использованию графических и текстовых средств документирования.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

#### **Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.

#### **Задание**

Выполнить анализ и классификацию графических и текстовых средств документирования.

#### **Порядок выполнения работы:**

Графические средства моделирования предметной области позволяют разработчикам в наглядном виде изучать существующую ИС, перестраивать ее в соответствии с поставленными целями и имеющимися ограничениями.

CASE-средства обладают следующими основными особенностями:

1. имеют мощные графические средства для описания и документирования ИС, обеспечивающие удобный интерфейс с разработчиком и развивающие его творческие возможности;
2. осуществляют интеграцию отдельных компонент CASE-средств, обеспечивающую управляемость процессом разработки систем;
3. используют специальным образом организованное хранилище проектных метаданных (репозитория).

Интегрированное CASE-средство должно содержать следующие компоненты:

1. репозиторий, являющийся основой CASE-средства. Он должен обеспечивать хранение версий проекта и его отдельных компонентов, синхронизацию поступления информации от различных разработчиков при групповой разработке, контроль метаданных на полноту и непротиворечивость;

2. графические средства анализа и проектирования, обеспечивающие создание и редактирование иерархически связанных диаграмм (DFD, ERD и др.), образующих модели ИС;

3. средства разработки приложений, включая языки 4GL и генераторы кодов;

4. средства конфигурационного управления;

5. средства документирования;

6. средства тестирования;

7. средства управления проектом;

8. средства реинжиниринга.

Современный рынок программных средств насчитывает около 300 различных CASE-средств, наиболее мощные из которых используются практически всеми ведущими западными фирмами.

Все современные CASE-средства могут быть классифицированы в основном по типам и категориям. Классификация по типам отражает функциональную их ориентацию на те или иные процессы ЖЦ.

Классификация по категориям определяет степень интегрированности по выполняемым функциям и включает следующее:

1. отдельные локальные средства, решающие небольшие автономные задачи (tools);

2. набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла систем (toolkit);

3. полностью интегрированные средства, поддерживающие весь ЖЦ систем и связанные общим репозиторием.

Помимо этого, CASE-средства можно классифицировать по следующим признакам:

1. применяемым методологиям и моделям систем и БД;

2. степени интегрированности с СУБД;

3. доступным платформам.

Классификация по типам в основном совпадает с компонентным составом CASE-средств.

На сегодняшний день российский рынок программного обеспечения располагает следующими наиболее развитыми CASE-средствами: Vantage Team Builder (Westmount I-CASE), Designer/2000, Silverrun, ERwin+Vpwin, S-Designer, CASE-Аналитик, CASE /4/0, PRO-IV, System Architect, Visible Analyst Workbench, EasyCASE; VIS; RATIONAL ROSE.



### **Форма представления результата:**

Классификация графических и текстовых средств документирования в виде схем.

#### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.02.02 Принципы построения ПО**

#### **Практическое занятие № 3. Создание диаграмм вариантов использования**

**Цель работы:** изучить методику построения диаграмм вариантов использования.

#### **Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

#### **Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, Microsoft Visio, браузер Microsoft Internet Explorer.

#### **Задание:**

По указанию преподавателя выберите индивидуальное задание.

1. Разработка программного комплекса «Управление кредитами в банке».
2. Разработка программного комплекса «Отделение колледжа».
3. Разработка программного комплекса «Обслуживание банкомата».
4. Разработка программного комплекса «Управление гостиницей».
5. Разработка программного комплекса «Магазин по продаже автозапчастей».

6. Разработка программного комплекса «Строительная фирма».
7. Разработка программного комплекса «Управление библиотечным фондом».
8. Разработка программного комплекса «АРМ работника склада»
9. Разработка программного комплекса «АРМ администратора ателье по ремонту оргтехники»
10. Разработка программного комплекса «АРМ администратора автосалона».
11. Разработка программного комплекса «АРМ администратора ресторана».
12. Разработка программного комплекса «АРМ администратора фитнес-клуба».
13. Разработка программного комплекса «АРМ администратора аэропорта».
14. Разработка программного комплекса «АРМ работника отдела кадров».
15. Разработка программного комплекса «АРМ администратора спорткомплекса».

Выполните реализацию вариантов использования и последовательности в соответствии с заданием.

### **Краткие теоретические сведения:**

#### *Диаграммы вариантов использования*

Понятие варианта использования (use case) впервые ввел Ивар Яacobson и придал ему такую значимость, что в настоящее время вариант использования превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать. На языке UML вариант использования изображают следующим образом:

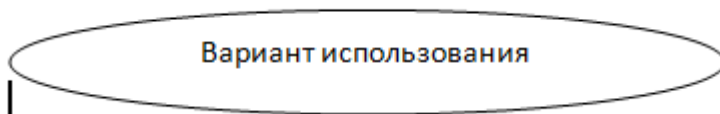


Рис.1. Вариант использования

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а

не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования. На языке UML действующие лица представляют в виде фигур:



Рис.2. Действующее лицо (актер)

Действующие лица делятся на три основных типа:

- пользователи;
- системы;
- другие системы, взаимодействующие с данной;
- время.

Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

*Связи между вариантами использования и действующими лицами*

В языке UML на диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации – это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).

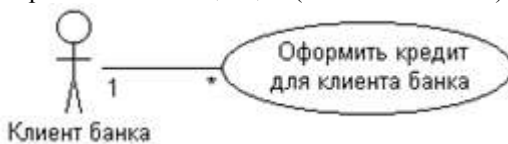


Рис.3. Пример связи коммуникации

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность.

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования

только при необходимости использовать функциональные возможности другого.

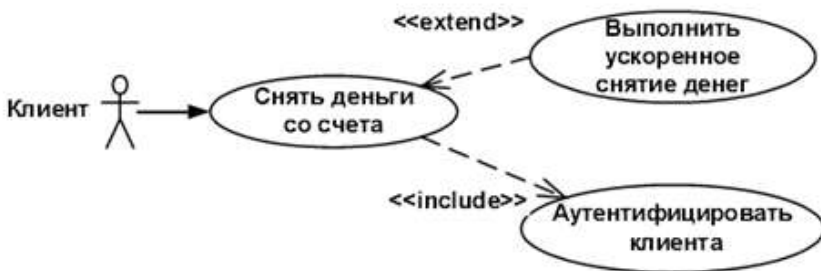


Рис.4. Пример связи включения и расширения

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты.



Рис.5. Пример связи обобщения

### Порядок выполнения

Изучить предлагаемый теоретический материал.

Постройте диаграмму вариантов использования для выбранной информационной системы.

Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

### Форма представления результата:

Диаграмма вариантов использования.

### Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.02.02 Принципы построения ПО**

#### **Практическое занятие № 4. Создание диаграмм взаимодействия**

**Цель работы:** изучить методику построения диаграмм вариантов взаимодействия.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

#### **Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, Microsoft Visio, браузер Microsoft Internet Explorer.

#### **Задание.**

Выполните реализацию диаграммы взаимодействия в соответствии с заданием, представленном в практической работе №3.

#### **Порядок выполнения работы:**

Из термина «взаимодействие» ясно, что диаграмма используется для описания некоторого типа взаимодействий между различными элементами в модели. Это взаимодействие является частью динамического поведения системы.

Это интерактивное поведение представлено в UML двумя диаграммами, известными как **диаграмма последовательности** и **диаграмма сотрудничества**. Основное назначение обеих диаграмм схожи.

Диаграмма последовательности подчеркивает временную последовательность сообщений, а диаграмма сотрудничества подчеркивает структурную организацию объектов, которые отправляют и получают сообщения.

### ***Назначение диаграмм взаимодействия***

Целью диаграмм взаимодействия является визуализация интерактивного поведения системы. Визуализация взаимодействия — сложная задача. Следовательно, решение состоит в том, чтобы использовать различные типы моделей, чтобы охватить различные аспекты взаимодействия.

Диаграммы последовательности и сотрудничества используются для отображения динамической природы, но под другим углом.

Целью диаграммы взаимодействия является —

- чтобы зафиксировать динамическое поведение системы.
- для описания потока сообщений в системе.
- для описания структурной организации объектов.
- для описания взаимодействия между объектами.

### ***Как нарисовать диаграмму взаимодействия?***

Как мы уже обсуждали, цель диаграмм взаимодействия состоит в том, чтобы охватить динамический аспект системы. Таким образом, чтобы охватить динамический аспект, нам нужно понять, что такое динамический аспект и как он визуализируется. Динамический аспект может быть определен как снимок работающей системы в определенный момент

У нас есть два типа диаграмм взаимодействия в UML. Одна — это диаграмма последовательности, а другая — диаграмма сотрудничества. Диаграмма последовательности фиксирует временную последовательность потока сообщений от одного объекта к другому, а диаграмма сотрудничества описывает организацию объектов в системе, участвующих в потоке сообщений.

Следующие вещи должны быть четко определены, прежде чем рисовать диаграмму взаимодействия

- объекты, принимающие участие во взаимодействии.
- потоки сообщений среди объектов.
- последовательность, в которой сообщения передаются.
- организация объекта.

Объекты, принимающие участие во взаимодействии.

Потоки сообщений среди объектов.

Последовательность, в которой сообщения передаются.

Организация объекта.

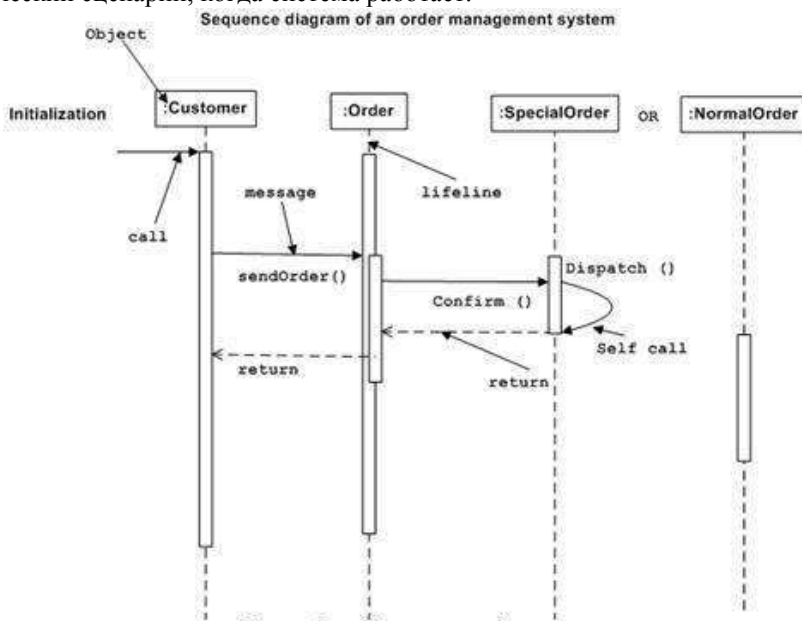
Ниже приведены две диаграммы взаимодействия, моделирующие систему управления заказами. Первая диаграмма — это диаграмма последовательности, а вторая — диаграмма сотрудничества.

### ***Диаграмма последовательности***

Диаграмма последовательности имеет четыре объекта (Customer, Order, SpecialOrder и NormalOrder).

На следующем рисунке показана последовательность сообщений для объекта *SpecialOrder*, и то же самое можно использовать в случае объекта *NormalOrder*. Важно понимать временную последовательность потоков сообщений. Поток сообщений — это не что иное, как вызов метода объекта.

Первый вызов — *sendOrder()*, который является методом объекта *Order*. Следующий вызов — метод *verify()*, который является методом объекта *SpecialOrder*, а последний вызов — *Dispatch()*, который является методом объекта *SpecialOrder*. Следующая диаграмма в основном описывает вызовы методов от одного объекта к другому, и это также фактический сценарий, когда система работает.



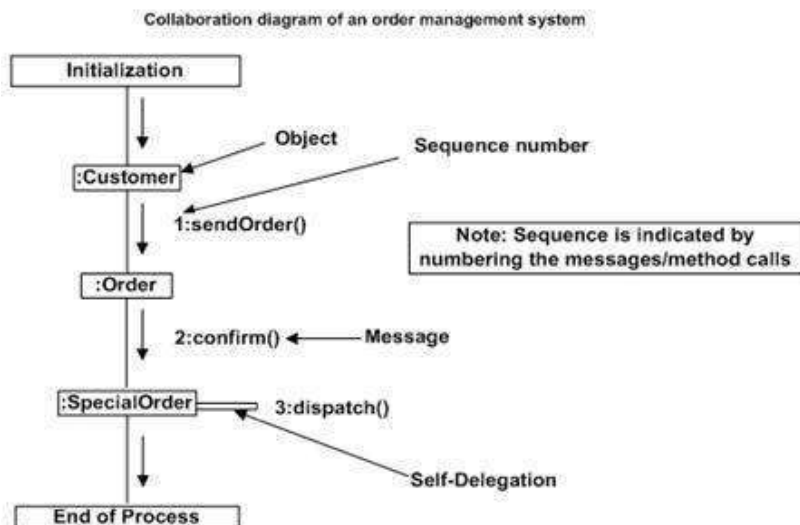
### Диаграмма сотрудничества

Вторая диаграмма взаимодействия — это диаграмма сотрудничества. Он показывает организацию объекта, как показано на следующей диаграмме. На диаграмме сотрудничества последовательность вызова метода указана с помощью некоторой техники нумерации. Число указывает, как методы вызываются один за другим. Мы взяли ту же систему управления заказами, чтобы описать диаграмму сотрудничества.

Вызовы методов аналогичны вызовам диаграмм последовательности. Однако различие в том, что диаграмма

последовательности не описывает организацию объекта, тогда как диаграмма сотрудничества показывает организацию объекта.

Чтобы выбрать между этими двумя диаграммами, акцент делается на тип требования. Если временная последовательность важна, тогда используется диаграмма последовательности. Если требуется организация, используется диаграмма сотрудничества.



Где использовать диаграммы взаимодействия?

Мы уже обсуждали, что диаграммы взаимодействия используются для описания динамической природы системы. Теперь мы рассмотрим практические сценарии, в которых используются эти диаграммы. Чтобы понять практическое применение, нам нужно понять основную природу последовательности и диаграмму сотрудничества.

Основное назначение обеих диаграмм схожи, поскольку они используются для захвата динамического поведения системы. Тем не менее, конкретная цель важнее уточнить и понять.

Диаграммы последовательности используются для определения порядка сообщений, передаваемых от одного объекта к другому. Диаграммы взаимодействия используются для описания структурной организации объектов, участвующих во взаимодействии. Одной диаграммы недостаточно для описания динамического аспекта всей системы, поэтому для ее представления в целом используется набор диаграмм.

Диаграммы взаимодействия используются, когда мы хотим понять поток сообщений и структурную организацию. Поток сообщений



означает последовательность потоков управления от одного объекта к другому. Структурная организация означает визуальную организацию элементов в системе.

Диаграммы взаимодействия могут быть использованы для моделирования потока управления по временной последовательности.

Моделировать поток управления структурными организациями.

- для форвард инжиниринга.
- для реверс-инжиниринга.

### **Порядок выполнения**

Изучить предлагаемый теоретический материал.

Постройте диаграмму взаимодействия для выбранной информационной системы.

### **Форма представления результата:**

Диаграмма взаимодействия.

### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

## **Тема 03.02.02 Принципы построения программного обеспечения Практическое занятие №5,6 «Построение диаграммы классов»**

### **Формируемые компетенции:**

ПК1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.

**Цель работы:** изучить основные принципы построения диаграммы классов.

### **Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

### **Материальное обеспечение**

- ПЭВМ

- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, Microsoft Visio, браузер Microsoft Internet Explorer.

### **Задание:**

Выполните реализацию диаграммы классов в соответствии с заданием, представленном в практической работе №3.

#### **Диаграммы классов**

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов UML - это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).

На диаграмме классов изображаются следующие элементы:

Пакет (package) - набор элементов модели, логически связанных между собой;

Класс (class) - описание общих свойств группы сходных объектов;

Интерфейс (interface) - абстрактный класс, задающий набор операций, которые объект произвольного класса, связанного с данным интерфейсом, предоставляет другим объектам.

#### **Класс**

Класс - это группа сущностей (объектов), обладающих сходными свойствами, а именно, данными и поведением. Отдельный представитель некоторого класса называется объектом класса или просто объектом.

Под поведением объекта в UML понимаются любые правила взаимодействия объекта с внешним миром и с данными самого объекта.

На диаграммах класс изображается в виде прямоугольника со сплошной границей, разделенного горизонтальными линиями на 3 секции:

Верхняя секция (секция имени) содержит имя класса и другие общие свойства (в частности, стереотип).

В средней секции содержится список атрибутов

В нижней - список операций класса, отражающих его поведение (действия, выполняемые классом).

Любая из секций атрибутов и операций может не изображаться (а также обе сразу). Для отсутствующей секции не нужно рисовать разделительную линию и как-либо указывать на наличие или отсутствие элементов в ней.

На усмотрение конкретной реализации могут быть введены дополнительные секции, например, исключения (Exceptions).

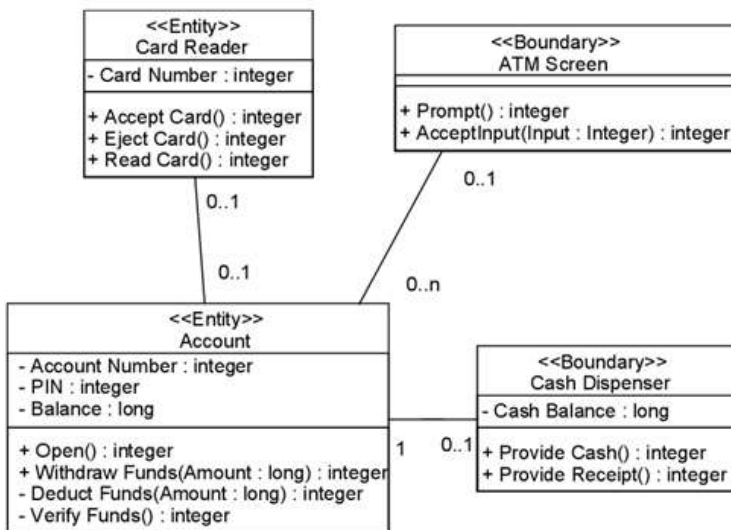


Рис. 2. Пример диаграммы классов

### Стереотипы классов

Стереотипы классов – это механизм, позволяющий разделять классы на категории.

В языке UML определены три основных стереотипа классов:

Boundary (граница);

Entity (сущность);

Control (управление).

### Граничные классы

Граничными классами (boundary classes) называются такие классы, которые расположены на границе системы и всей окружающей среды. Это экранные формы, отчеты, интерфейсы с аппаратурой (такой как принтеры или сканеры) и интерфейсы с другими системами.

Чтобы найти граничные классы, надо исследовать диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать, по крайней мере, один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

## **Классы-сущности**

Классы-сущности (entity classes) содержат хранимую информацию. Они имеют наибольшее значение для пользователя, и потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.

## **Управляющие классы**

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например, может быть класс SecurityManager (менеджер безопасности), отвечающий за контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими как разделение ресурсов, распределенная обработка данных или обработка ошибок.

Помимо упомянутых выше стереотипов можно создавать и свои собственные.

## **Атрибуты**

Атрибут – это элемент информации, связанный с классом. Атрибуты хранят инкапсулированные данные класса.

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attribute visibility).

У атрибута можно определить четыре возможных значения этого параметра:

Public (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В соответствии с нотацией UML общему атрибуту предшествует знак « + ».

Private (закрытый, секретный). Соответствующий атрибут не виден никаким другим классом. Закрытый атрибут обозначается знаком « - » в соответствии с нотацией UML.

Protected (защищенный). Такой атрибут доступен только самому классу и его потомкам. Нотация UML для защищенного атрибута – это знак « # ».

Package or Implementation (пакетный). Предполагает, что данный атрибут является общим, но только в пределах его пакета. Этот тип видимости не обозначается никаким специальным значком.

В общем случае, атрибуты рекомендуется делать закрытыми или защищенными. Это позволяет лучше контролировать сам атрибут и код.

С помощью закрытости или защищенности удастся избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут. Задаваемые параметры видимости повлияют на генерируемый код.

### **Операции**

Операции реализуют связанное с классом поведение. Операция включает три части – имя, параметры и тип возвращаемого значения.

Параметры – это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать, как имена операций, так и имена операций вместе с их параметрами и типом возвращаемого значения. Чтобы уменьшить загруженность диаграммы, полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент: тип данных аргумента, аргумент2:тип данных аргумента2,...): тип возвращаемого значения

Следует рассмотреть четыре различных типа операций:

- Операции реализации;
- Операции управления;
- Операции доступа;
- Вспомогательные операции.

### **Операции реализации**

Операции реализации (implementor operations) реализуют некоторые бизнес-функции. Такие операции можно найти, исследуя диаграммы взаимодействия. Диаграммы этого типа фокусируются на бизнес-функциях, и каждое сообщение диаграммы, скорее всего, можно соотнести с операцией реализации.

Каждая операция реализации должна быть легко прослеживаема до соответствующего требования. Это достигается на различных этапах моделирования. Операция выводится из сообщения на диаграмме взаимодействия, сообщения исходят из подробного описания потока событий, который создается на основе варианта использования, а последний –

на основе требований. Возможность проследить всю эту цепочку позволяет гарантировать, что каждое требование будет реализовано в коде, а каждый фрагмент кода реализует какое-то требование.

### **Операции управления**

Операции управления (manager operations) управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.

### **Операции доступа**

Атрибуты обычно бывают закрытыми или защищенными. Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют операции доступа (access operations). Такой подход дает возможность безопасно инкапсулировать атрибуты внутри класса, защитив их от других классов, но все же позволяет осуществить к ним контролируемый доступ. Создание операций Get и Set (получения и изменения значения) для каждого атрибута класса является стандартом.

### **Вспомогательные операции**

Вспомогательными (helper operations) называются такие операции класса, которые необходимы ему для выполнения его ответственностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Чтобы идентифицировать операции, выполните следующие действия:

Изучите диаграммы последовательности и кооперативные диаграммы. Большая часть сообщений на этих диаграммах является операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.

Рассмотрите управляющие операции. Может потребоваться добавить конструкторы и деструкторы.

Рассмотрите операции доступа. Для каждого атрибута класса, с которым должны будут работать другие классы, надо создать операции Get и Set.

### **Связи**

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме последовательности или кооперативной диаграмме, между ними должна существовать связь.

Существуют четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

### **Ассоциации**

Ассоциация (association) – это семантическая связь между классами. Их рисуют на диаграмме классов в виде обыкновенной линии.



Рис. 3. Связь ассоциация

Ассоциации могут быть двунаправленными, как в примере, или однонаправленными. На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление.

Направление ассоциации можно определить, изучая диаграммы последовательности и кооперативные диаграммы. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

### **Зависимости**

Связи зависимости (dependency) также отражают связь между классами, но они всегда однонаправлены и показывают, что один класс зависит от определений, сделанных в другом. Например, класс А использует методы класса В. Тогда при изменении класса В необходимо произвести соответствующие изменения в классе А.

Зависимость изображается пунктирной линией, проведенной между двумя элементами диаграммы, и считается, что элемент, привязанный к концу стрелки, зависит от элемента, привязанного к началу этой стрелки.

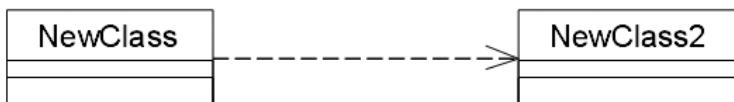


Рис. 4. Связь зависимость

При генерации кода для этих классов к ним не будут добавляться новые атрибуты. Однако, будут созданы специфические для языка операторы, необходимые для поддержки связи.

### **Агрегации**

Агрегации (aggregations) представляют собой более тесную форму ассоциации. Агрегация – это связь между целым и его частью. Например, у вас может быть класс Автомобиль, а также классы Двигатель, Покрышки и классы для других частей автомобиля. В результате объект класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышек и т. д. Агрегации визуализируют в виде линии с ромбиком у класса, являющегося целым:



Рис. 5. Связь агрегация

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции, объект-часть может принадлежать только единственному целому, и, кроме того, как правило, жизненный цикл частей совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Такое каскадное удаление нередко рассматривается как часть определения агрегации, однако оно всегда подразумевается в том случае, когда множественность роли составляет 1..1; например, если необходимо удалить Клиента, то это удаление должно распространиться и на Заказы (и, в свою очередь, на Строки заказа).

### **Обобщения (Наследование)**

Обобщение (наследование) - это отношение типа общее-частное между элементами модели. С помощью обобщений (generalization) показывают связи наследования между двумя классами. Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. Наследование пакетов означает, что в пакете-наследнике все сущности пакета-предка будут видны под своими собственными именами (т.е. пространства имен объединяются). Наследование показывается сплошной линией, идущей от класса-потомка к классу-предку (в терминологии ООП - от потомка к предку, от сына к отцу, или от подкласса к суперклассу). Со стороны более общего элемента рисуется большой полый треугольник.



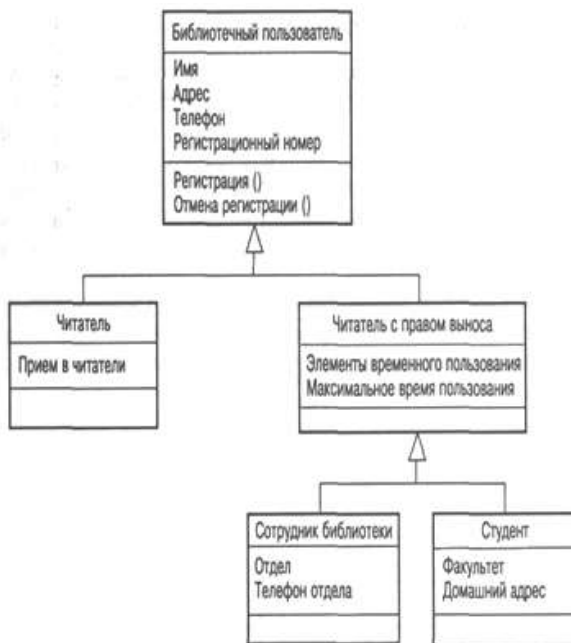


Рис. 6. Пример связи наследование

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

### Множественность

Множественность (multiplicity) показывает, сколько экземпляров одного класса взаимодействуют с помощью этой связи с одним экземпляром другого класса в данный момент времени.

Например, при разработке системы регистрации курсов в университете можно определить классы Course (курс) и Student (студент). Между ними установлена связь: у курсов могут быть студенты, а у студентов – курсы. Вопросы, на который должен ответить параметр множественности: «Сколько курсов студент может посещать в данный момент? Сколько студентов может за раз посещать один курс?»

Так как множественность дает ответ на оба эти вопроса, её индикаторы устанавливаются на обоих концах линии связи. В примере регистрации курсов мы решили, что один студент может посещать от нуля до четырех курсов, а один курс могут слушать от 0 до 20 студентов.

В языке UML приняты определенные нотации для обозначения множественности.

Таблица 1 - Обозначения множественности связей в UML

Множественность	Значение
0..*	Ноль или больше
1..*	Один или больше
0..1	Ноль или один
1..1 (сокращенная запись: 1)	Ровно один

### Имена связей

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи – это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Можно задать в связи с этим вопрос, является ли объект класса Person клиентом компании, её сотрудником или владельцем? Чтобы определить это, ассоциацию можно назвать «employs» (нанимает):

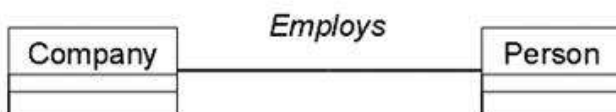


Рис. 7. Пример имен связей

### Роли

Ролевые имена применяют в связях ассоциации или агрегации вместо имен для описания того, зачем эти связи нужны. Возвращаясь к примеру с классами Person и Company, можно сказать, что класс Person играет роль сотрудника класса Company. Ролевые имена – это обычно имена существительные или основанные на них фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или ролевым именем, или именем связи, но не обоими сразу. Как и имена связей, ролевые имена не обязательны, их дают, только если цель связи не очевидна. Пример ролей приводится ниже:

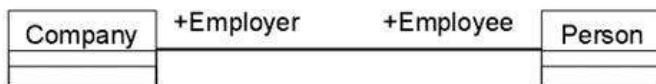


Рис. 8. Пример ролей связей

### Пакет. Механизм пакетов

В контексте диаграмм классов, пакет - это вместилище для некоторого набора классов и других пакетов. Пакет является самостоятельным пространством имен.



Рис. 9. Обозначение пакета в UML

В UML нет каких-либо ограничений на правила, по которым разработчики могут или должны группировать классы в пакеты. Но есть некоторые стандартные случаи, когда такая группировка уместна, например, тесно взаимодействующие классы, или более общий случай - разбиение системы на подсистемы.

Пакет физически содержит сущности, определенные в нем (говорят, что "сущности принадлежат пакету"). Это означает, что если будет уничтожен пакет, то будет уничтожено и все его содержимое.

Существует несколько наиболее распространенных подходов к группировке.

Во-первых, можно группировать их по стереотипу. В таком случае получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход может быть полезен с точки зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах пограничные классы уже оказываются в одном пакете.

Другой подход заключается в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам. Если для группировки классов не использовать некоторые эвристики, то она становится произвольной. Одна из них, которая в основном используется в UML, – это зависимость. Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость.

Таким образом, диаграмма пакетов представляет собой диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, то есть диаграмма пакетов – это форма диаграммы классов.

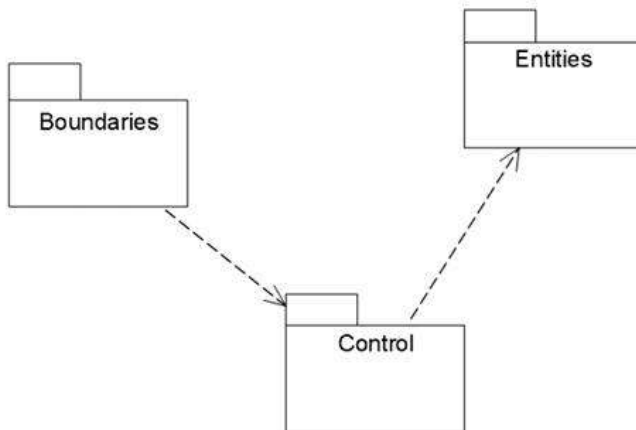


Рис. 10. Пример диаграммы пакетов

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными:

- один класс посылает сообщение другому;
- один класс включает часть данных другого класса;
- один класс использует другой в качестве параметра операции.

Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в вашей системе, однако они помогают выделить эти зависимости, а после того, как они все окажутся на виду, остается только поработать над снижением их количества. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится нечитаемой.

#### **Порядок выполнения**

Изучить предлагаемый теоретический материал.

Постройте диаграммы классов для выбранной информационной системы.

Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

**Форма представления результата:**

Диаграмма классов.

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.02.02 Принципы построения программного обеспечения  
Практическое занятие №7. «Построение диаграммы состояний»**

**Цель:** изучить основные принципы построения диаграммы состояний.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

**Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, Microsoft Visio, браузер Microsoft Internet Explorer.

**Задание:**

Выполните реализацию диаграммы состояний в соответствии с заданием, представленном в практической работе №3.

**Краткие теоретические сведения:**

**Диаграммы состояний**

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния.

### **Деятельность**

Деятельностью (activity) называется поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность – это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

### **Входное действие**

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности, входное действие рассматривается как непрерываемое. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

### **Выходное действие**

Выходное действие (exit action) подобно входному. Однако, оно осуществляется как составная часть процесса выхода из данного состояния. Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. В этом случае описанию деятельности, входного действия или выходного действия предшествует знак « ^ ».

Соответствующая строка на диаграмме выглядит как

До: ^Цель.Событие (Аргументы)

Здесь Цель – это объект, получающий событие, Событие – это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

Деятельность может также выполняться в результате получения объектом некоторого события. При получении некоторого события выполняется определенная деятельность.

Переходом (Transition) называется перемещение из одного состояния в другое. Совокупность переходов диаграммы показывает, как объект может перемещаться между своими состояниями. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события.

### **События**

Событие (event) – это то, что вызывает переход из одного состояния в другое. События размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

### **Ограждающие условия**

Ограждающие условия (guard conditions) определяют, когда переход может, а когда не может осуществиться. В противном случае переход не осуществится.

Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

### **Действие**

Действием (action), как уже говорилось, является непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий, однако, изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния.

Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

Событие или действие могут быть поведением внутри объекта, а могут представлять собой сообщение, посылаемое другому объекту. Если событие или действие посылается другому объекту, перед ним на диаграмме помещают знак « ^ ».

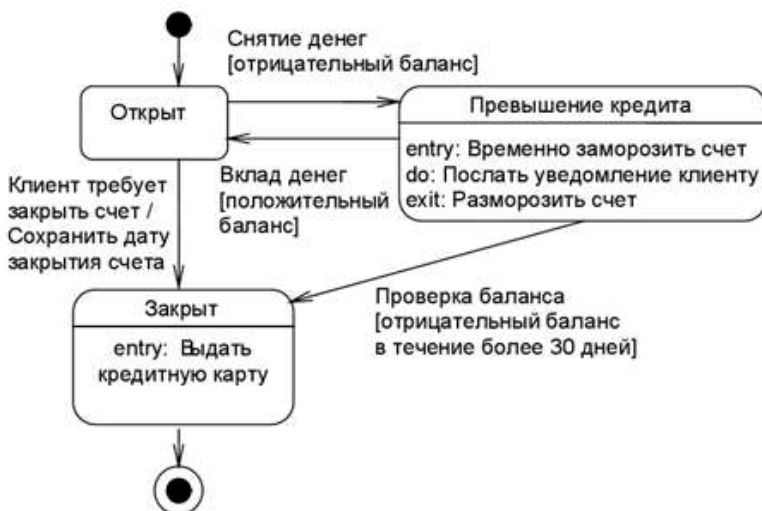


Рис. 1. Пример диаграммы состояний

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

### Форма представления результата:

Диаграмма состояний.

### Критерии оценки:

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».



Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.02.02 Принципы построения программного обеспечения Практическое занятие №8. «Построение диаграммы компонент»**

#### **Формируемые компетенции:**

ПК1. Анализировать проектную и техническую документацию на уровне взаимодействия компонент программного обеспечения.

**Цель:** изучить основные принципы построения диаграммы компонент.

#### **Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;

Оборудование, инструменты, материалы, таблицы, схемы, справочники, и др.)

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, Microsoft Visio, браузер Microsoft Internet Explorer.

#### **Задание:**

Выполните реализацию диаграммы компонент в соответствии с заданием, представленном в практической работе №3.

#### **Краткие теоретические сведения:**

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимо-

сти, соответствующие зависимостям на этапе компиляции или выполнения программы.

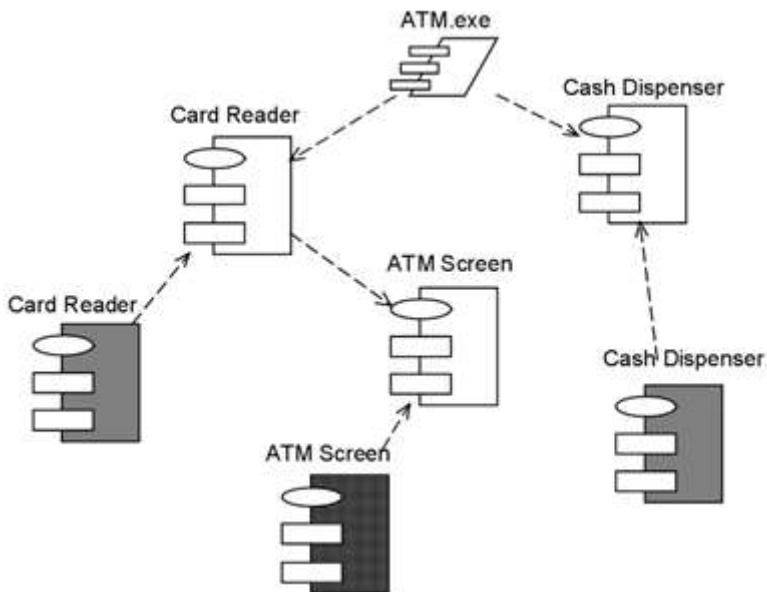


Рис. 1. Пример диаграммы компонентов

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

#### Объединение диаграмм компонентов и развертывания

В некоторых случаях допускается размещать диаграмму компонентов на диаграмме развертывания. Это позволяет показать какие компоненты выполняются и на каких узлах.

#### Порядок выполнения

Изучить предлагаемый теоретический материал.

Постройте диаграмму вариантов использования для выбранной информационной системы.

Выполните реализацию диаграммы компонент.

Построить отчет, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

### **Форма представления результата:**

Диаграмма компонентов.

### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

## **Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

### **Практическое занятие №9, 10, 11. Создание окна приложения в среде С#**

**Цель работы:** Изучить основные элементы среды разработки *Visual Studio Integrated Development Environment (IDE* - интегрированная среда разработки) С# при создании на языке С# приложений с графически интерфейсом.

### **Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

### **Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования С#

### **Задание 1**

1. Изучить теоретический материал.
2. Создать Windows форму.
3. На Windows форме создать кнопку "Приветствие".
4. Протестировать работу приложения

5. Добавить в форму две кнопки (1 и 2), для которых задать различные цвета (свойство BackColor).
6. Написать для кнопок 1 и 2 обработчики, которые изменяют цвета кнопок: при неоднократном нажатии любой кнопки цвета кнопок меняются (цвет кнопки 1 меняется на цвет кнопки 2 и наоборот).
7. Добавьте кнопку "Выход". Закрытие приложения обеспечивает метод Exit() класса Application.

Протестировать работу приложения.

### **Основные сведения**

Среда разработки Visual Studio Integrated Development Environment (IDE) - интегрированная среда разработки включает набор инструментов и не зависит от используемых языков программирования, представленных в Visual Studio. Visual Studio можно использовать для создания кода и на различных языках программирования: управляемый C++ - Managed C++, Visual Basic.NET, Java.NET, C#.

В практической работе проводится изучение среды разработки на языке программирования C# и следующих средств проектирования Windows - приложений:

- основные окна среды разработки C#;
- построение базовой инфраструктуры с помощью Application Wizard (мастер создания приложений);
- использование дизайнера форм Dialog Painter (программа для рисования диалоговых окон) для оформления диалоговых окон;
- добавление новых функциональных возможностей в приложение с использованием вкладки Properties (свойства).

### **Обзор среды разработки C#**

Для начала работы с Visual Studio.NET необходимо из главного меню выбрать пункт "Microsoft Visual Studio.NET" (VS). При этом на компьютере загрузится Developer Studio (визуальная среда разработки Microsoft Visual) на экране компьютера будет выведено окно, изображенное на рисунке 1.1.

### **Проектирование приложения**

В качестве приложения разработаем простое приложение, пользовательский интерфейс которого будет содержать только главное окно. Для этого необходимо выполнить следующие шаги:

1. Создать рабочую область, называемую также рабочей средой (проектирования), рабочим пространством и рабочей обстановкой нового проекта.

Для создания каркаса приложения можно использовать мастер создания приложений - Application Wizard.

- Изменить внешний вид автоматически создаваемых мастером окон до желаемого вида.

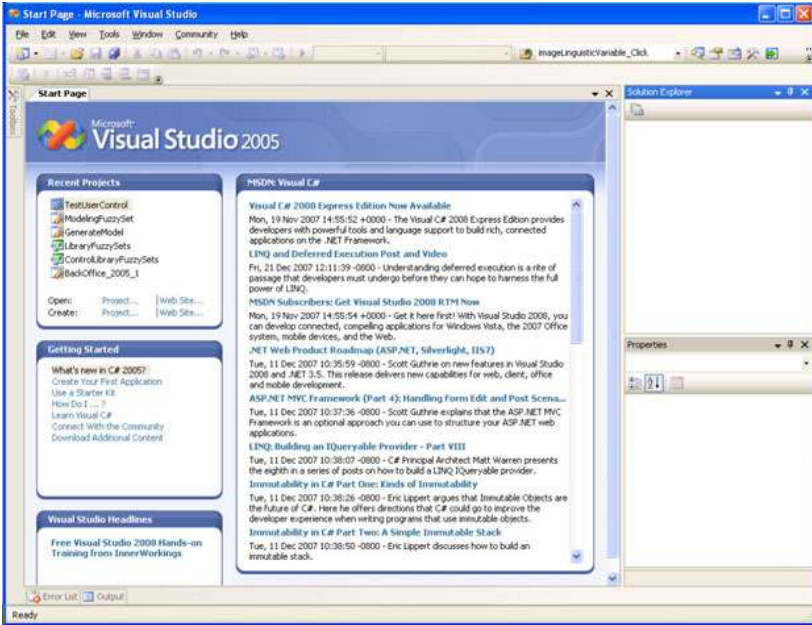


Рис. 1.1. Стартовое диалоговое окно IDE

Добавить код С#, который будет вызывать отображение приветствия.

### ***Создание рабочей области проекта***

В VS каждому разрабатываемому приложению нужна рабочая среда. Рабочая среда проекта состоит из папок, в которых хранятся файлы исходного кода, а также из папок, в которых хранятся различные конфигурационные файлы. Создание рабочей среды нового проекта производится следующим образом:

Щелкните на ссылке Project (Создать новый проект) метки Create на начальной странице (Start Page) VS.NET. При этом откроется окно создания нового проекта New Project (рисунок 1.2).

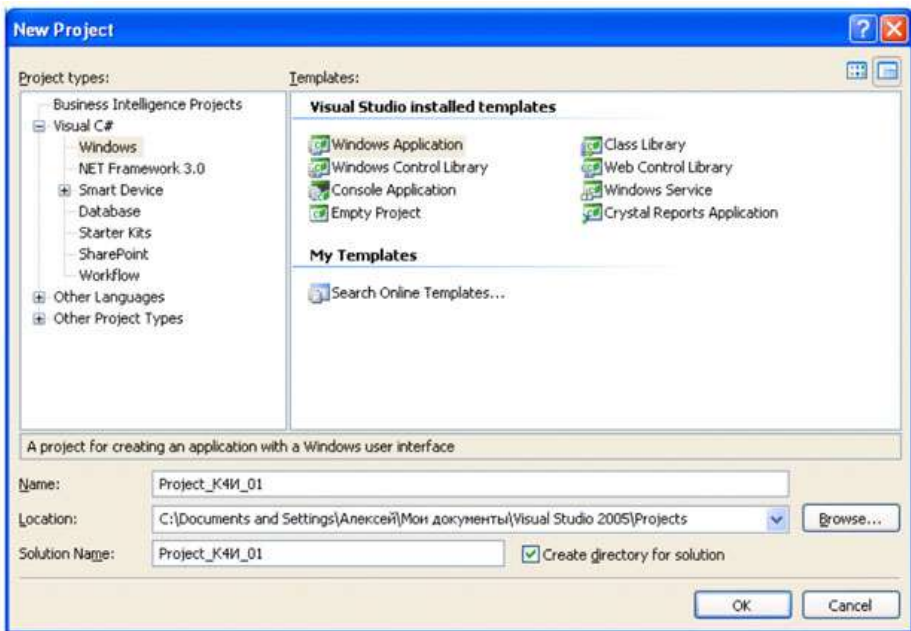


Рис. 1.2. Мастер создания нового проекта (New Project Wizard)

В дереве, отображаемом в подокне Project Type (Типы проектов) выберите "Visual C# /Windows". В подокне Templates (Шаблоны) выберите Windows Application (Приложение Windows).

В поле Name (Название проекта) наберите имя проекта - Project\_K4И\_01 (имя проекта присваивается в соответствии со следующим синтаксисом: Project\_"номер группы"\_"номер бригады в группе").

Щелкните на кнопке ОК. (Да). Мастер создания нового проекта создаст новый класс Form1, производный от System.Windows.Forms.Form с правильно настроенным методом Main(). В свойствах проекта автоматически будут созданы ссылки на необходимые сборки библиотеки базовых классов. На экране появится графический шаблон среды разработки (рисунок 1.3).

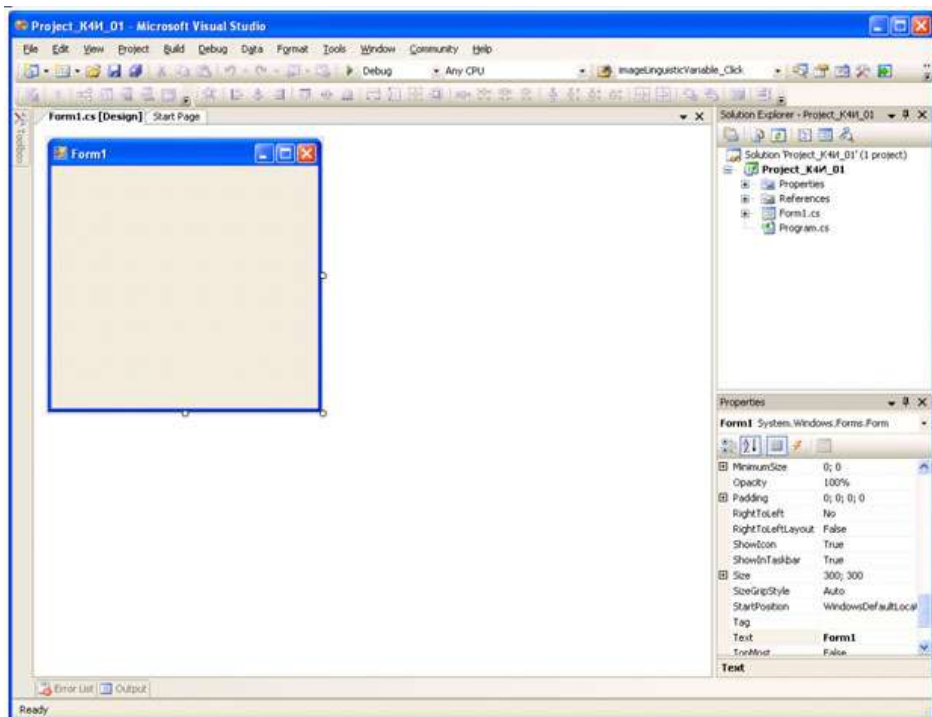


Рис. 1.3. Графический шаблон главного окна приложения

При помощи дизайнера графических форм можно добавлять в приложение любые элементы управления и он будет автоматически генерировать код для этих элементов (по умолчанию файл с главной формой приложения называется Form1.cs).

Для просмотра кода сгенерированного приложения можно в окне Solution Explorer щелкнуть правой кнопкой мыши на файле Form1.cs и в контекстном меню выбрать View Code (рисунок 1.4).

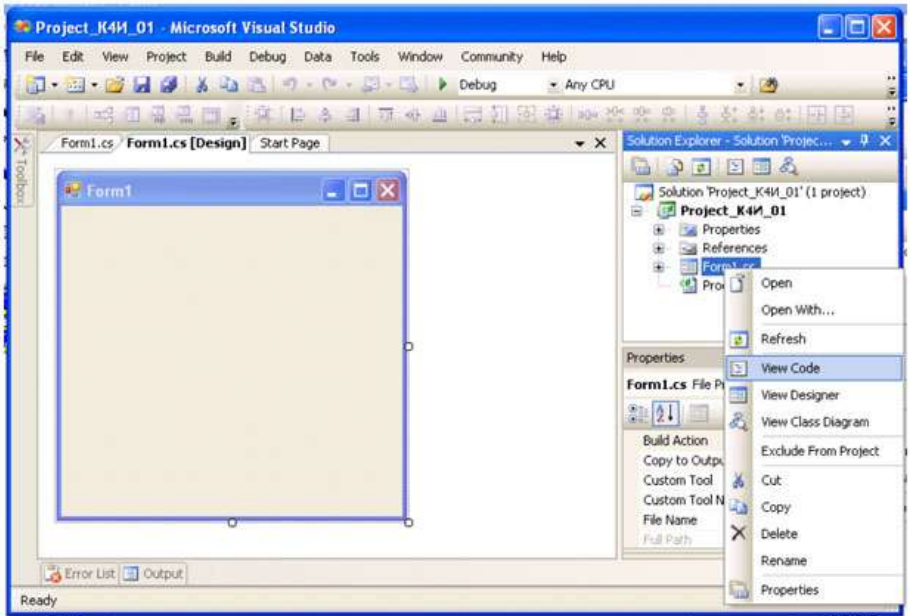


Рис. 1.4. Выбор режима View Code в контекстном меню

В результате будет выведен на экран следующий листинг кода приложения

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace Project_K4И_01
{
    public partial class Form1 : Form
    {
        public Form1()
        { InitializeComponent(); }
    }
}

```

Инициализация компонент реализуется кодом, который можно отобразить, в окне *Solution Explorer* щелкнуть на пункте *Form.Designer.cs* (рисунок 1.5).



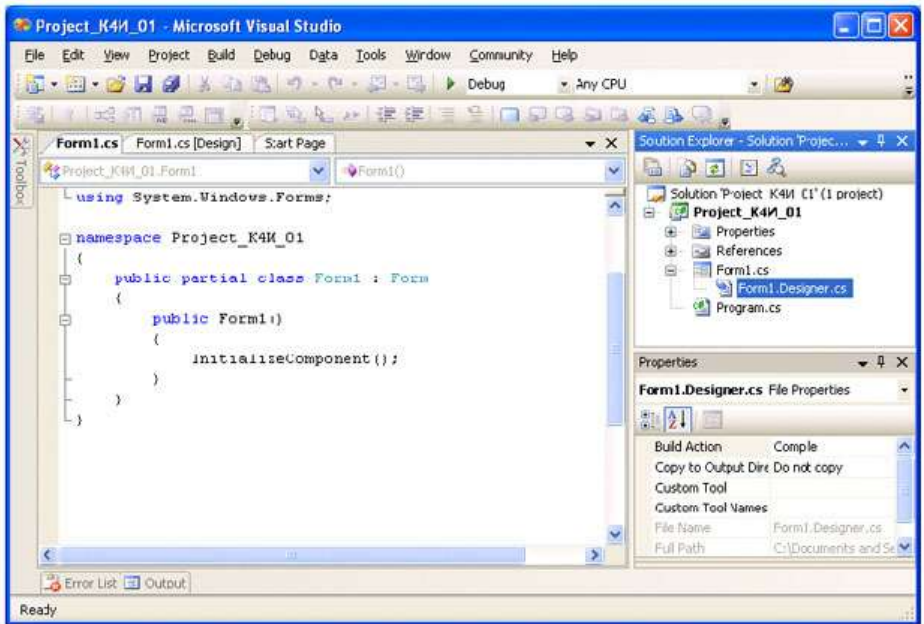


Рис. 1.5. Выбор режима Form.Designer.cs

```

namespace Project_K4M_01
{
    partial class Form1
    {
        private System.ComponentModel.IContainer components = null;
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }
        #region Windows Form Designer generated code
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.Text = "Form1";
        }
        #endregion
    }
}

```

```
}  
}
```

В определении класса Form1 используется ключевое слово *partial*, которое позволяет определять класс, структуру или интерфейс, распределенные по нескольким файлам. В Visual Studio 2005 классы Windows -форм формируются в двух файлах: Form1.cs и Form1.Designer.cs. В файле Form1.Designer.cs присутствует код, сгенерированный дизайнером Windows -формы, а файле Form1.cs - присутствует код инициализации класса и пользовательские члены класса (поля, свойства, методы, события, делегаты)

Код приложения (Program.cs) имеет следующий вид:

```
using System;  
using System.Collections.Generic;  
using System.Windows.Forms;  
namespace Project_K4II_01  
{  
    static class Program  
    {  
        [STAThread]  
        static void Main()  
        {  
            Application.EnableVisualStyles();  
            Application.SetCompatibleTextRenderingDefault(false);  
            Application.Run(new Form1());  
        }  
    }  
}
```

Метод Main является точкой входа для приложения и вызывает Application.Run, который создает класс Form1.

### **Класс System.Windows.Forms.Application**

Класс Application можно рассматривать как "класс низшего уровня", позволяющий нам управлять поведением приложения *Windows Forms*. Кроме того, этот класс определяет набор событий уровня всего приложения, например закрытие приложения или простой центрального процессора.

Наиболее важные методы этого класса (все они являются статическими) перечислены в таблице 1.1.

Таблица 1.1. Наиболее важные методы типа Application

Метод класса	Назначение Application
AddMessageFilter()	Эти методы позволяют приложению перехватывать

	сообщения <code>RemoveMessageFilter()</code> и выполнять с этими сообщениями необходимые предварительные действия. Для того чтобы добавить фильтр сообщений, необходимо указать класс, реализующий интерфейс <code>IMessageFilter</code>
<code>DoEvents()</code>	Обеспечивает способность приложения обрабатывать сообщения из очереди сообщений во время выполнения какой-либо длительной операции. Можно сказать, что <code>DoEvents()</code> - это "быстрый и грязный" заменитель нормальной многопоточности
<code>Exit()</code>	Завершает работу приложения
<code>ExitThred()</code>	Прекращает обработку сообщений для текущего потока и закрывает все окна, владельцем которых является этот поток
<code>OLERequired()</code>	Инициализирует библиотеки OLE. Можете считать этот метод эквивалентом <code>.NET</code> для вызываемого вручную метода <code>OleInitialize()</code>
<code>Run()</code>	Запускает стандартный цикл работы с сообщениями для текущего потока

Класс `Application` определяет множество статических свойств (таблица 1.2), большинство из которых доступны только для чтения.

Таблица 1.2. Наиболее важные свойства типа `Application`

Свойство	Назначение
<code>CommonAppDataRegistry</code>	Возвращает параметр системного реестра, который хранит общую для всех пользователей информацию о приложении
<code>CompanyName</code>	Возвращает имя компании
<code>CurrentCulture</code>	Позволяет задать или получить информацию о естественном языке, для работы с которым предназначен текущий поток
<code>CurrentInputLanguage</code>	Позволяет задать или получить информацию о естественном языке для ввода информации, получаемой текущим потоком
<code>ProductName</code>	Для получения имени программного продук-

	та, которое ассоциировано с данным приложением
ProductVersion	Позволяет получить номер версии программного продукта
StartupPath	Позволяет определить имя выполняемого файла для работающего приложения и путь к нему в операционной системе

Многие из этих свойств предназначены для получения общей информации о приложении, такой как название компании, номер версии и т.п.

Таким образом, при помощи многих свойств (например, `CompanyName` или `ProductName`) можно очень просто получить метаданные уровня сборки. В сборке можно использовать любое количество встроенных и пользовательских атрибутов. В результате можно получить значение атрибута `[assembly:AssemblyCompany(" ")]` при помощи свойства `Application.CompanyName` без необходимости прибегать к использованию типов, определенных в пространстве имен `System.Reflection`.

#### Проектирование окна приложения

Для разметки окон приложения в соответствии с требованиями пользователя необходимо изменить свойства класса `Forms1`. Это можно сделать с помощью дизайнера окон (`Form Designer`), путем изменения свойств в окне Свойства (`Properties`) или в коде программы.

Размеры окна можно изменить непосредственно в `Form Designer` с помощью мыши захватывая и, растягивая/сжимая границы окна.

Для изменения других свойств окна необходимо окно свойств `Properties`.

На вкладке `Properties` измените значение в поле `Text` (Заголовок) на Проект К4И. При этом на форме изменится заголовок окна (рисунок 1.6).

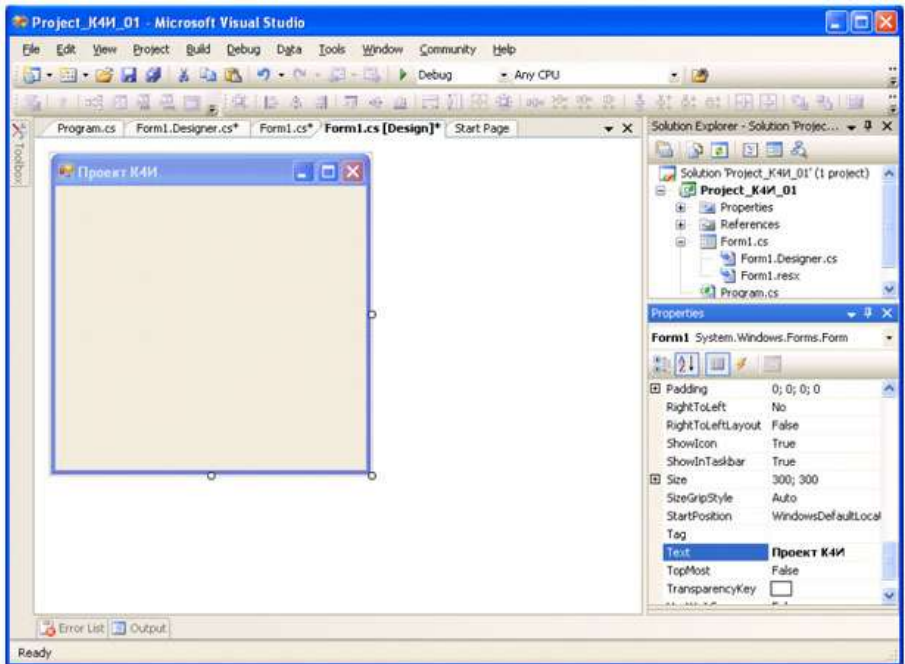


Рис. 1.6. Изменение значения в поле Text на вкладке Properties  
Откомпилируйте приложение, выбрав из главного меню команду Build Project\_K4I\_01 (рисунок 1.7)

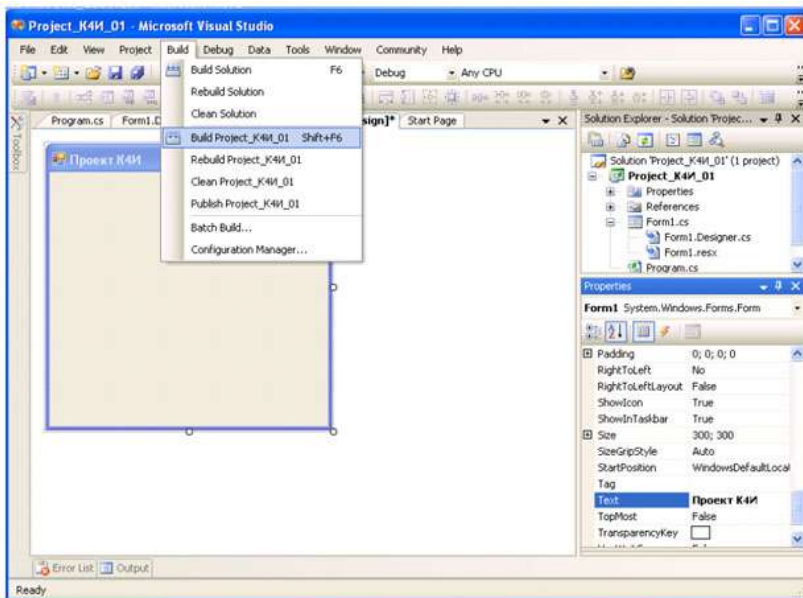


Рис. 1.7. Выбор из главного меню команды Build


В строке состояний должно появиться сообщение:

*Build succeeded*

Для запуска приложения выберите из главного меню команду Debug/Start (F5). Приложение запустится в отладочном режиме и на экране появится разработанное окно (рисунок 1.8.).



Рис. 1.8. Окно приложения Project\_K4И\_01

Для закрытия окна щелкните мышью на кнопке  *Добавление нового кода в приложение*

Добавим в главную форму элемент контроля - кнопку. Для этого откроем вкладку *ToolBox* (рисунок 1.9) и сначала щелкнем мышью на элементе *Button* вкладки, а затем щелкнем мышью на форме.

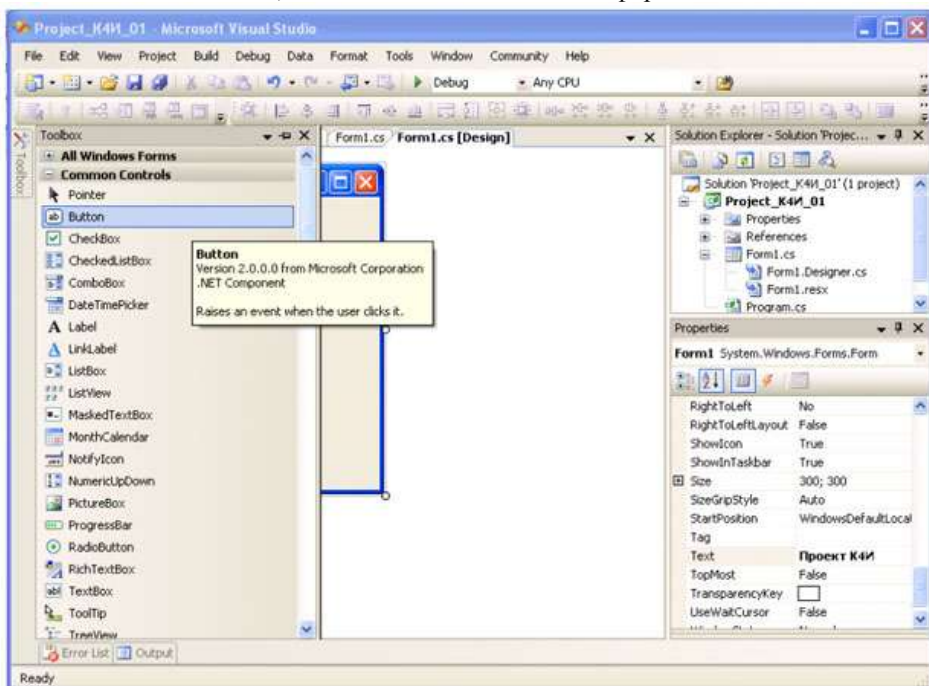


Рис. 1.9. Вкладка *ToolBox*

В результате получим форму с кнопкой (рисунок 1.10).

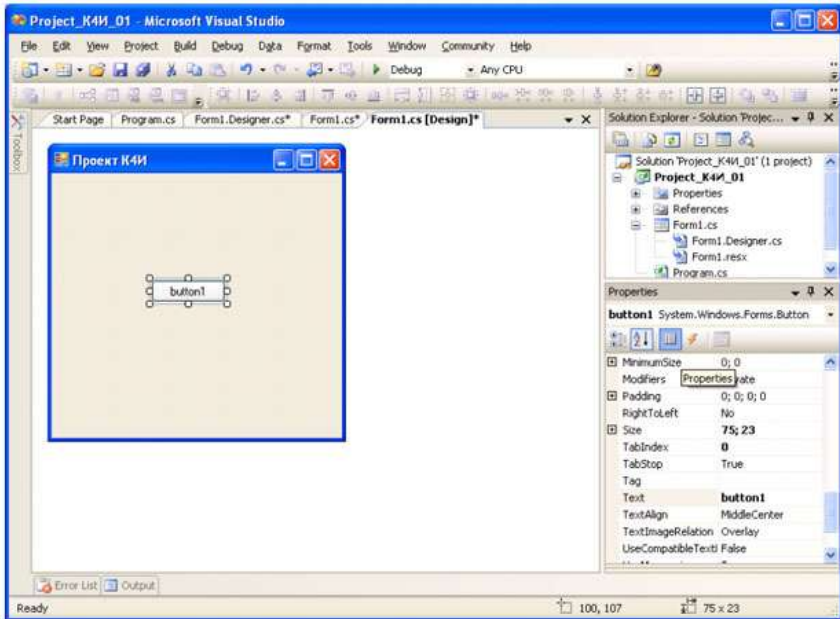


Рис. 1.10. Форма с установленной кнопкой

Установите кнопку в требуемое место на форме с помощью мыши.

Для задания текста на кнопке выделите ее на форме и откройте вкладку *Свойства* и измените свойство *Text* на "Приветствие". В результате название кнопки изменится (рисунок 1.11).



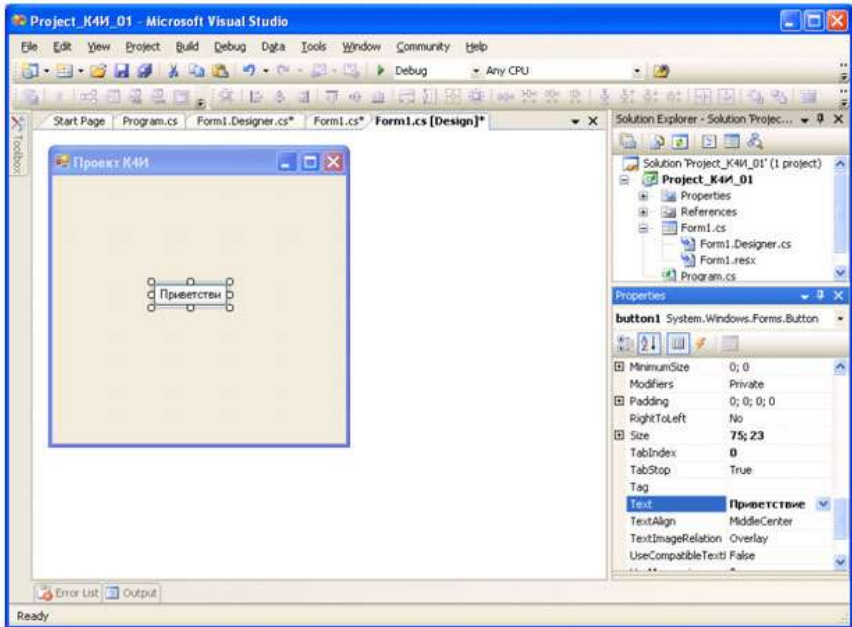


Рис. 1.11. Форма с измененным свойством Text кнопки

Для связывания функций кнопки с диалоговым окном необходимо создать обработчик события на нажатие кнопки. Для этого сделайте двойной щелчок на кнопке. В результате в коде приложения сформируется шаблон функции обработчика события Click для кнопки.

```
private void button1_Click(object sender, EventArgs e)
{
}

```

В полученный шаблон добавим функцию вывода диалогового окна с сообщением.

```
private void button1_Click(object sender, EventArgs e)
{
    // Сообщение
    MessageBox.Show("Поздравляю с первым проектом на C#");
}

```

После компиляции и запуска приложения получим следующее окно приложения (рисунок 1.12), а при нажатии кнопки будет выведено сообщение (рисунок 1.13).

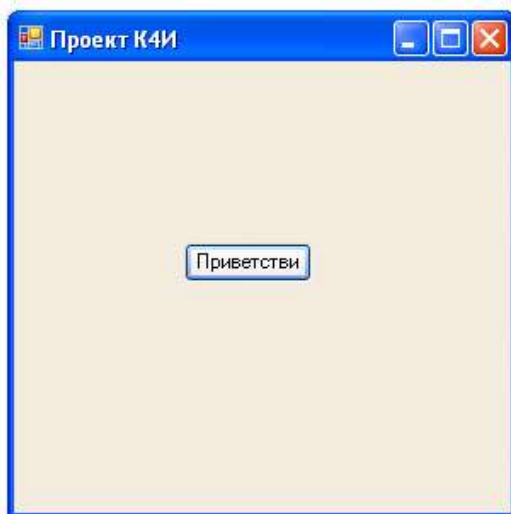


Рис. 1.12. Результат выполнения приложения

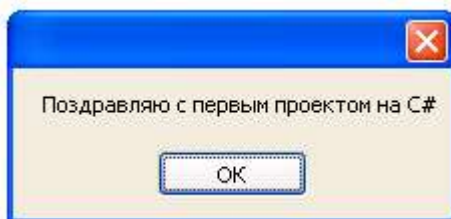


Рис. 1.13. Вывод сообщения

Структура и синтаксис функции

Рассмотрим код листинга функции:

```
private void button1_Click(object sender, EventArgs e)
{
    // Сообщение
    MessageBox.Show("Поздравляю с первым проектом на C#");
}
```

Первая строка является частью оболочки функции, сгенерированной *Developer Studio* на языке C#.

Первое слово в строке, `private` определяет видимость функции как внутреннюю, т.е. видимую только для членов класса `Form1`. Второе слово `void`, определяет тип данных возвращаемого значения (результата). Ключевое слово `void`, - перед именем функции, или в качестве аргумента функции (в скобках в конце строки), означает отсутствие соответствующего элемента. Третье слово в строке, `button1_Click`, обозначает имя

функции. За именем функции следует список передаваемых ей аргументов, заключенный в круглые скобки. Круглые скобки нужно использовать всегда, даже когда у функции нет параметров.

Правило 1. В C# при вызове функции за ее именем должны стоять круглые скобки, даже если данной функции не передается ни один параметр.

В следующей строке листинга открывающая фигурная скобка ( { ) отмечает начало тела функции. В конце тела функции ставится закрывающая фигурная скобка ( } ).

Правило 2. Тело функции всегда заключается в фигурные скобки { }.

Следующая строка начинается с двух косых черт, или слешей ( // ). Все, что следует до конца строки после двух идущих подряд косых черт (без пробелов, табуляций и т.п. между ними) рассматривается компилятором как комментарий и игнорируется. Исключением являются строки, в которых косая черта является частью текстовой, или литерной строки (строки букв). Это один из способов комментирования кода. Второй способ чаще используется при добавлении в код нескольких строк комментариев. В этом случае начало комментария обозначается идущими подряд косой чертой и звездочкой ( /\* ), а конец комментария завершается таким же набором символов, но переставленных в обратном порядке ( \*/ ).

Последняя строка в добавленном нами коде (в тексте это две строки):

```
MessageBox.Show("Поздравляю с первым проектом на C#");
```

Во-первых, C# чувствителен к регистру. В именах функций и переменных заглавные (прописные) буквы должны использоваться точно так же, как в их объявлениях. Это означает, что компилятор распознает следующие имена функций как имена трех различных функций:

```
MessageBox.Show messageBox.Show messagebox.Show
```

Правило 3. Язык C# чувствителен к регистру. При вводе программ, написанных на языке C#, учитывайте регистр. В частности, все идентификаторы вводите с учетом регистра.

За именем функции следуют аргументы функции, заключенные в круглые скобки, а после скобок стоит точка с запятой. Аргументы разделяются запятыми.

## Создание главного меню приложения

**Цель работы:** Изучить основные способы разработки главного меню приложения. Получить практические навыки в создании главного меню приложения.

### Задание 2

1. Изучить теоретический материал.

2. Создать главное меню, включающее следующие пункты: "Объект", "Справочник", "Справка".
3. Для пункта "Объект" создать следующие подпункты: "Сотрудник", "Клиент", "Договор", "Поручение", "Сделка", "Выход".
4. Для пункта "Справочник" создать следующие подпункты: "Должность", "Страна", "Регион", "Город", "ИМНС".
5. Для пункта "Справка" создать подпункт - "О программе"
6. Протестировать работу приложения.

### ***Указания по использованию .NET***

В любом языке программирования существуют традиционные стили программирования. Эти стили являются не частью самого языка, а соглашениями, скажем, по именованию переменных или использованию определенных классов, методов или функций. Если большинство разработчиков будут следовать одинаковым соглашениям, то им будет проще понять код друг друга, что, в свою очередь, облегчает поддержку программы. Так, общим соглашением в Visual Basic 6 было то, что строковые переменные должны иметь имена, начинающиеся с s или str, например, String sResult или String strMessage. Однако соглашения зависят от языка и среды разработки. Программисты на C++ для платформы Windows традиционно используют префикс psz или lpsz для обозначения строк: char \*pszResult; char \*lpszMessage;. Но на Unix-машинах такие префиксы не применяются: char \*Result; char \*Message;.

В соответствии с соглашениями в C# имена переменных не должны иметь префиксов: string Result; string Message;.

Соглашение, согласно которому имена переменных содержат префикс, указывающий тип данных, известно, как "венгерский" стиль именования объектов. При чтении такого кода разработчики могут сразу же сказать по имени переменной, какой тип данных она представляет.

В то время как для многих языков соглашения по именованию вырабатывались одновременно с развитием языка, для C# и платформы .NET Microsoft написала подробные рекомендации по использованию, которые приведены в документации MSDN для .NET/C#. Следовательно, с самого начала программы .NET будут иметь более высокий уровень совместимости по части понимания кода другими разработчиками. Эти рекомендации были разработаны с учетом опыта, полученного на протяжении более двадцати лет объектно-ориентированного программирования, и в результате являются тщательно продуманными и хорошо приняты сообществом разработчиков.

Однако необходимо отметить, что рекомендации не то же самое, что спецификации языка. Рекомендаций следует придерживаться по мере возможности. Если имеется веская причина для их несоблюдения, это не будет проблемой. Отклонение от рекомендаций должно быть вызвано

реальными причинами, а не простым нежеланием.

Одним из важных моментов является выбор имен для элементов программы: переменных, методов, классов, перечислений и пространств имен.

Очевидно, что названия обязаны отражать назначение элемента и не должны конфликтовать с другими именами.

Общая философия платформы .NET состоит в том, что имя переменной должно отражать назначение экземпляра переменной, а не тип данных.

Например, Height - хорошее название, а IntegerValue - нет. Однако этот принцип является труднодостижимым идеалом. В частности, при работе с элементами управления в большинстве случаев вам будет удобнее использовать имена переменных, подобные ConfirmationDialog и ChooseEmployeeListBox.

Конкретные рекомендации по именованию включают в себя следующие разделы.

Практически во всех случаях для имен следует использовать стиль Pascal, при котором первая буква каждого слова в названии является прописной

Например: EmployeeSalary, ConfirmationDialog, PlainTextEncoding.

Соединение слов с помощью знака подчеркивания не приветствуется, поэтому не придумывайте такие имена, как employee\_salary. В других языках часто используют все прописные буквы в названиях констант. Это не рекомендуется в C#, поскольку такие имена трудно читать, лучше применять паскалевский стиль:

```
const int MaximumLength;
```

Еще одна рекомендуемая схема - именование в стиле camel. Именование camel аналогично паскалевскому стилю, за исключением того, что первая буква первого слова не является прописной: employeeSalary, confirmationDialog, plainTextEncoding.

Существуют две ситуации, в которых лучше применять такое именование. Имена всех параметров, передаваемых в методы, должны записываться в стиле camel:

```
public void RecordSale (string salesmanName,int guanuity);
```

Также можно использовать camel -соглашение для того, чтобы отличить два элемента, которые в противном случае имели бы одинаковые имена. Наиболее общий случай, когда свойство является оболочкой для поля.

```
private string employeeName;  
public string EmployeeName  
{ get  
    { return employeeName; }  
}
```

Приведенный код является совершенно корректным с точки зрения рекомендаций. Отметим, однако, что в этом случае следует применять соглашение camel для закрытых членов и соглашение Pascal для открытых или защищенных членов, чтобы другие классы, использующие ваш код, видели только имена в стиле Pascal (за исключением имен параметров).

В большинстве случаев следует применять соглашения Pascal. Тем не менее, соглашение camel рекомендуется для закрытых переменных, которые не видны вне класса, где две переменные имеют одинаковое назначение. Например, если есть public свойство, которое инкапсулирует private поле с тем же именем, то можно использовать соглашение camel для поля и соглашение Pascal для свойства, как в приведенном выше примере EmployeeName.

Также необходимо обращать внимание на чувствительность к регистру. C# чувствителен к регистру, поэтому синтаксически в C# допустимо, чтобы имена различались только регистром. Однако нужно помнить, что ваши сборки могут быть вызваны из приложений VB.NET, а VB.NET не является чувствительным к регистру. Поэтому использовать имена, отличающиеся только регистром, можно лишь в том случае, если они никогда не будут видны вне сборки. В противном случае код, написанный в VB.NET, не сможет корректно использовать вашу сборку.

Необходимо по возможности делать так, чтобы стиль всех имен совпадал. Например, если один из методов в классе называется ShowConfirmationDialog, то другому методу не следует давать имя ShowDialogWarning или WarningDialogShow. Он должен называться ShowWarningDialog.

Имена пространств имен следует выбирать особенно тщательно для того, чтобы избежать использования такого же имени, которое применяется где-то еще. Необходимо помнить, что .NET различает имена объектов в разделяемых сборках только по именам пространств имен. Если использовать для двух пакетов программного обеспечения одно и то же имя пространства имен и установить оба пакета на один компьютер, возникнут проблемы. Рекомендуется создавать пространство имен верхнего уровня с именем вашей компании, а затем вкладывать пространства имен, постепенно сужая их названия до технологии, группы или отдела, где вы работаете, или до названия пакета, для которого предназначены ваши классы. Microsoft рекомендует имена пространств имен, которые начинаются с <НазваниеКомпании>.<НазваниеТехнологии>, например,

WeaponsOfDestructionCorp.RayGunControllers

или

WeaponsOfDestructionCorp.Viruses.

Имена не должны конфликтовать с ключевыми словами. Если попытаться в программе назвать элемент по имени одного из ключевых слов C#, это практически всегда вызовет синтаксическую ошибку., так как компилятор предположит, что имя соответствует оператору.

## Создание меню

В пространстве имен System.Windows.Forms предусмотрено большое количество типов для организации ниспадающих главных меню (расположенных в верхней части формы) и контекстных меню, открывающихся по щелчку правой кнопки мыши.

Элемент управления ToolStrip представляет собой контейнер, используемый для создания структур меню, панелей инструментов и строк состояний.

Элемент управления MenuStrip - это контейнер для структур меню в приложении. Этот элемент управления наследуется от ToolStrip. Система меню строится добавлением объектов ToolStripMenuItem к menuStrip.

Класс ToolStripMenuItem служит для построения структур меню. Каждый объект ToolStripMenuItem представляет отдельный пункт в системе меню.

Начнем с создания стандартного ниспадающего меню, которое позволит пользователю выйти из приложения, выбрав пункт Объект > Выход. Для этого необходимо перетащить элемент управления MenuStrip (рисунок 2.1) на форму в конструкторе.

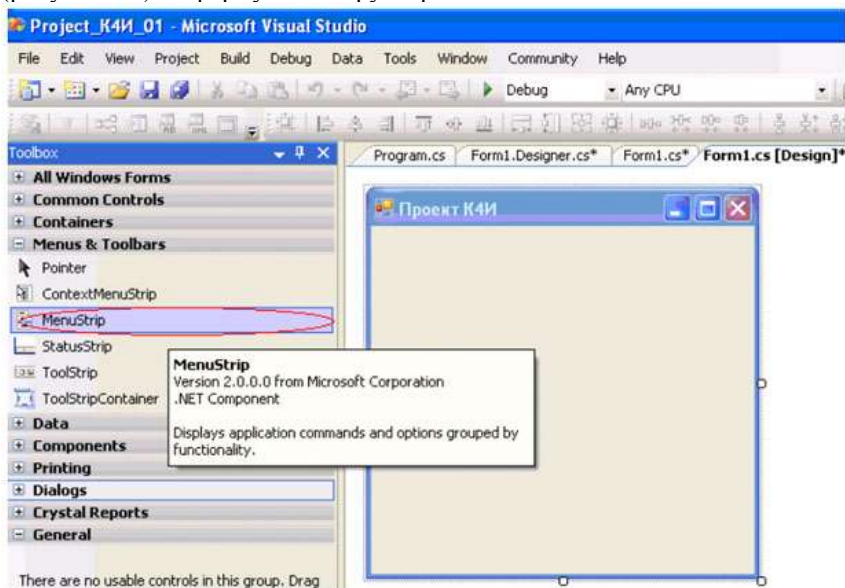


Рис. 2.1. Элемент управления MenuStrip

Элемент управления MenuStrip позволит вводить текст меню непосредственно в элементы меню. То, что должно получиться, представлено на рисунке 2.2.




Рис. 2.2. Простое меню на форме

При помощи графических средств можно настроить свойства любого элемента меню. Для пункта меню "Объект" зададим свойство Name равным `objektToolStripMenuItem`, для пункта меню "Выход") - `exitToolStripMenuItem`, а для пункта меню "Справка" - `HelpToolStripMenuItem`.

При двойном щелчке на пункте меню "Выход" (объект `exitToolStripMenuItem`) Visual Studio автоматически сгенерирует оболочку для обработчика события Click и перейдет в окно кода, в котором нам будет предложено создать логику метода (в нашем случае `exitToolStripMenuItem_Click`):

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    // Здесь мы определяем реакцию на выбор пользователем
    // пункта меню
}
```

На вкладке Свойства (Properties) при выводе окна событий, нажать кнопку  событию Click будет соответствовать метод `menuItemExit_Click` (рисунок 2.3).

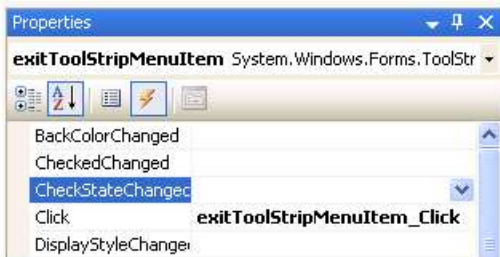


Рис. 2.3. Событие Click и обработчик события `exitToolStripMenuItem_Click`



Для корректного завершения приложения написать код для обработчика `exitToolStripMenuItem_Click`. Это можно сделать с помощью метода `Exit` класса `Application`:

```
private void exitToolStripMenuItem_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

Для тестирования созданного меню создадим обработчик для пункта меню "Объект", который будет сообщать, что выбран именно этот пункт меню.

```
private void objektToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("Пункт меню Объект");
}
```

При создании меню графическими средствами Visual Studio автоматически внесет необходимые изменения в служебный метод `InitializeComponent` и добавит переменные-члены, представляющие созданные элементы меню.

### **Создание многооконного приложения**

**Цель работы:** Изучить основные способы разработки многооконных приложений. Получить практические навыки в создании многооконных приложений.

#### **Задание 3**

1. Изучить теоретический материал.
2. Создать дочернее окно.
3. В дочернее окно добавить пункты меню.
4. Написать обработчик для вызова из главного меню дочернего окна.
5. Создать коды методов-заглушек для функций приложения.
6. Создать обработчики для вызова пунктов меню.

Протестировать работу приложения.

#### **Создание дочерней формы**

Основа Интерфейса (MDI) приложения - MDI родительская форма. Это - форма, которая содержит MDI дочерние окна. Дочерние окна являются "подокнами", с которыми пользователь взаимодействует в MDI приложении. Создание MDI родительской формы описано в "Создание главного меню приложения".

Для определения главного окна (Form1), как родительской формы в окне Свойств, установите `IsMdiContainer` свойство - `true`. Это определяет форму как MDI контейнер для дочерних форм. Для того чтобы родительское окно занимало весь экран необходимо свойству `WindowState` установить значение `Maximized`.

Создайте еще одно окно, которое будет дочерним (FormEmployee). Для этого выберите пункт меню Project/Add Windows Form.

Это окно должно вызываться из пункта главного меню "Сотрудник". Вставьте код, подобный следующему, чтобы создать новую MDI дочернюю форму, когда пользователь щелкает на пункте меню, например "Сотрудник" - имя объекта - employeeToolStripMenuItem (В примере ниже, указатель события обращается к событию Click для employeeToolStripMenuItem\_Click).

```
private void menuItemEmployee_Click(object sender,
System.EventArgs e)
{ // Создать объект FEmployee класса FormEmployee
FormEmployee FEmployee = new FormEmployee();
  // Установить родительское окно для дочернего
  FEmployee.MdiParent = this;
  // Вывести на экран дочернее окно
  FEmployee.Show();
}
```

Данный обработчик приведет к выводу на экран дочернего окна.

### **Создание меню в дочерней форме**

Добавьте в дочернее окно пункт меню "Действие" (actionToolStripMenuItem) с подпунктами "Отменить" (undoToolStripMenuItem), "Создать" (createToolStripMenuItem), "Редактировать" (editToolStripMenuItem), "Сохранить" (saveToolStripMenuItem) и "Удалить" (removeToolStripMenuItem). Перед пунктом удалить вставьте разделитель (Separator - name = toolStripSeparator1).

Добавьте в дочернее окно еще один пункт меню "Отчет" (reportToolStripMenuItem) с подпунктами "По сотруднику" (reportToolStripMenuItem1), "По всем сотрудникам" (reportToolStripMenuItem2). Дочернее окно будет иметь вид, представленный на рисунке 3.1

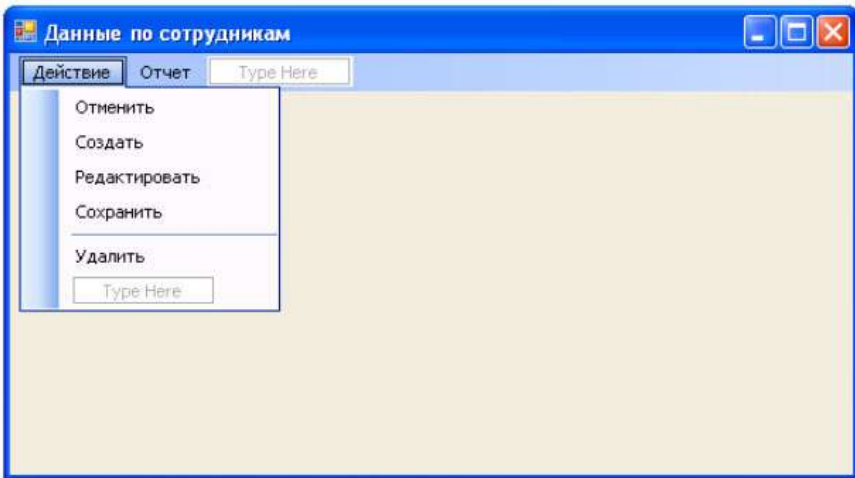


Рис. 3.1. Дочернее окно с меню

В главном меню родительской формы (Form1) имеются пункты "Объект", "Справочник" и "Справка". В дочерней форме (FormEmployee) сформированы пункты меню "Действие" и "Отчет". При загрузке дочерней формы меню родительской и дочерних форм должны были объединены и составлять следующую последовательность: "Объект", "Действие", "Отчет", "Справочник" и "Справка". Объединение пунктов меню производится с помощью задания значений свойств MergeAction и MergeIndex для объектов ToolStripMenuItem.

Проверьте, чтобы в меню главного окна для объекта objectToolStripMenuItem свойство MergeAction было установлено Append, а MergeIndex было равно 0, а для объектов dictionaryToolStripMenuItem и helpToolStripMenuItem - соответственно 1 и 2. С учетом этого, в окне "Сотрудник" для объектов actionToolStripMenuItem (Действие) и "Отчет" (reportToolStripMenuItem) свойству MergeAction необходимо задать значение Insert, а свойству MergeIndex задаем порядковый номер который определяет позицию данного пункта меню обновленном главном меню, т.е. 1 (после объекта objectToolStripMenuItem).

После компиляции программы, запуска ее на выполнение и вызова пункта меню "Сотрудник" экран должен иметь вид, представленный на рисунке 3.2.

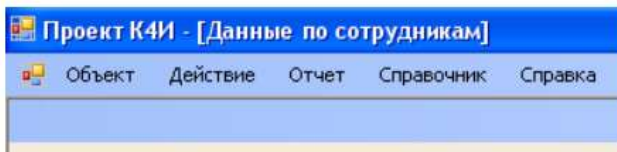


Рис. 3.2. Дочернее окно с подключенным меню

### **Создание обработчиков для меню дочерней формы**

Созданные пункты меню для дочернего окна должны инициировать выполнение соответствующих функций (Отменить, Создать, Редактировать, Сохранить и Удалить) приложения в отношении объектов конкретного дочернего окна. Для дочернего окна "Данные по сотруднику" эти функции должны выполнять соответственно отмену редактирования данных по сотруднику (функция "Отменить"), создавать новые данные по сотруднику (функция "Создать"), редактировать данные по сотруднику (функция "Редактировать"), сохранять созданные вновь или отредактированные функция по сотруднику (функция "Сохранить") и удалять данные по сотруднику (функция "Удалить").

Описанную функциональность целесообразно реализовать в программе в виде методов класса созданного FormEmployee. В приложении необходимо создать следующие методы:

- Undo - отменить;
- New - создать;
- Edit - редактировать;
- Save - сохранить;
- Remove - удалить.

На начальных этапах проектирования, как правило, неясна реализация каждого метода, поэтому целесообразно их выполнять в виде методов-заглушек, которые только сообщают пользователю о своем вызове, а в дальнейшем необходимо написать реальный код.

Для создания метода Undo в коде файла FormEmployee.cs добавьте следующий метод:

```
private void Undo()  
{ MessageBox.Show("метод Undo"); }
```

Далее создаем обработчик события вызова пункта меню "Отменить". Для этого в дизайнера формы класса FormEmployee делаем двойной щелчок на пункте меню "Отменить". Инструментальная среда VS сгенерирует следующий код:

```
private void undoToolStripMenuItem_Click(object sender, EventArgs e)  
{  
}
```

В код обработчика undoToolStripMenuItem\_Click добавим вызов метода Undo:

```
private void undoToolStripMenuItem_Click(object sender, EventArgs e)  
{  
    Undo();  
}
```

Откомпилируем приложение и протестируем вызов метода Undo. В результате выбора пункта меню "Отменить" должно быть выведено диалоговое окно с сообщением, приведенным на рисунке 3.3.

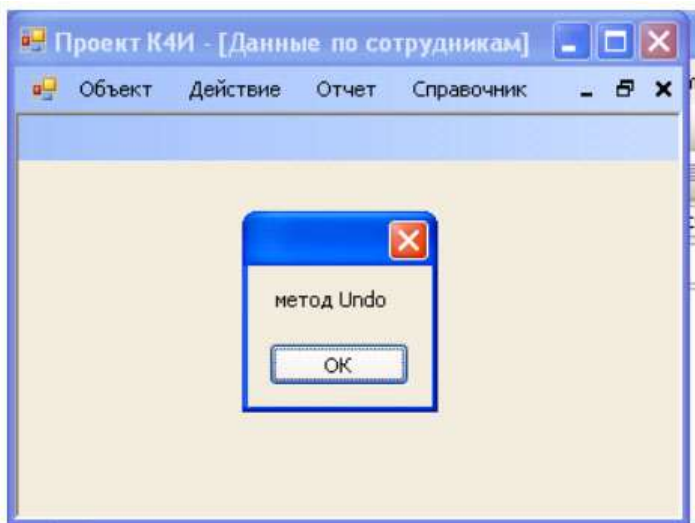


Рис. 3.3. Дочернее окно с подключенным меню  
Аналогичным образом создайте методы-заглушки для функций "Создать", "Редактировать", "Сохранить" и "Удалить".

**Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

**Форма представления результата:**

- Программный код, составленный на языке программирования

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

**Практическое занятие №12, 13.** Создание пользовательских диалоговых окон в среде C#

**Цель работы:** Изучить основные способы построения диалоговых окон, их параметры и получить практические навыки в разработке

## **Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

## **Материальное обеспечение:**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

## **Задание**

1. Изучить теоретический материал.
2. Создать модальное диалоговое окно с помощью класса MessageBox.
3. Создать пользовательское модальное диалоговое окно для пункта меню "О программе".
4. Написать обработчики для вызова модальных окон.
5. Протестировать работу приложения.

## **Основные сведения**

Диалоговое окно - это форма, обладающая некоторыми специальными характеристиками. Первая отличительная черта большинства диалоговых окон - то, что их размер изменять нельзя. Кроме того, в диалоговых окнах обычно не используются элементы управления, помещаемые в верхнюю часть обычных форм: ControlBox, MinimizeBox и MaximizeBox. Для пользователя диалоговое окно в противоположность обычному является практически неизменяемым.

Диалоговые окна бывают модальные и немодальные. Если приложение открывает модальное окно, то работа приложения блокируется до тех пор, пока не будет закрыто модальное окно. Немодальные окна могут работать одновременно с породившим их главным окном приложения. Такие окна часто используются для "плавающих" инструментальных панелей, настройки различных параметров приложения, причем отсутствие модальности позволяет использовать в приложении измененные параметры, не закрывая окна настройки этих параметров.

Простейшее модальное диалоговое окно можно создать на базе

класса `MessageBox`, входящего в библиотеку `Microsoft .NET Framework`. В практической работе 3 иллюстрировалось применение простейшего модального диалогового окна для вывода сообщения об активизации метода `Undo`. Для отображения диалогового окна использовался метод `Show`, передав ему через параметр текст сообщения "метод `Undo`". Прототип использованного метода `Show` следующий:

```
public static DialogResult Show(string message);
```

Когда пользователь щелкает кнопку `OK`, метод `Show` возвращает значение, равное `DialogResult.OK`

Существует множество перегруженных вариантов метода `MessageBox.Show`, позволяющих задать необходимый внешний вид диалоговой панели, а также количество и тип расположенных на ней кнопок.

Прототип наиболее общего варианта метода `MessageBox.Show`, позволяющий реализовать практически все возможности диалогового окна `MessageBox`, приведен ниже

```
public static DialogResult Show
```

```
{
```

```
string message, // текст сообщения
```

```
string caption, // заголовок окна
```

```
MessageBoxButtons btns, // кнопки, расположенные в окне
```

```
MessageBoxIcon icon, // значок, расположенный в окне
```

```
MessageBoxDefaultButton defButton, // кнопка по умолчанию
```

```
MessageBoxOptions opt // дополнительные параметры
```

```
};
```

Параметр `caption` позволяет задать текст заголовка диалогового окна `MessageBox`. С помощью параметра `btns` можно указать, какие кнопки необходимо расположить в окне диалогового окна. Этот параметр задается константами из перечисления `MessageBoxButtons` (таблица 4.1)





Таблица 4.1. Перечисление `MessageBoxButtons`

Константа	Кнопки, отображаемые в окне <code>MessageBox</code>
<code>OK</code>	<code>OK</code>
<code>OKCancel</code>	<code>OK, Cancel</code>
<code>YesNo</code>	<code>Yes, No</code>
<code>YesNoCancel</code>	<code>Yes, No, Cancel</code>
<code>RetryCancel</code>	<code>Retry, Cancel</code>
<code>AbortRetryIgnore</code>	<code>Abort, Retry, Ignore</code>

Параметр `icon` метода `MessageBox.Show` позволяет выбрать один из нескольких значков для расположения в левой части диалогового окна.

Он задается в виде константы перечисления `MessageBoxIcon` (таблица 4.2).

Таблица 4.2. Перечисление `MessageBoxIcon`

Константа	Значок
<code>Asterisk, Information</code>	
<code>Error, Stop</code>	
<code>Exclamation, Warning</code>	
<code>Question</code>	
<code>None</code>	Значок не отображается

Параметр `defButton` метода `MessageBox.Show` предназначен для выбора кнопки, которая получит фокус сразу после отображения диалогового окна. Этот параметр должен иметь одно из значений перечисления `MessageBoxDefaultButton` (таблица 4.3).

Таблица 4.3. Перечисление `MessageBoxDefaultButton`

Константа	Номер кнопки, получающей фокус ввода по умолчанию
<code>Button 1</code>	1
<code>Button 2</code>	2
<code>Button 3</code>	3

Если в диалоговом окне отображаются кнопки *Yes*, *No* и *Cancel*, то они будут пронумерованы последовательно: кнопка *Yes* получит номер 1 (константа `Button1`), кнопка *No* - номер 2 (константа `Button2`), а кнопка *Cancel* - номер 3 (константа `Button3`).

Параметр `opt` метода `MessageBox.Show` позволяет задать дополнительные параметры. Эти параметры должны иметь значения из перечисления `MessageBoxOptions` (таблица 4.4).

Таблица 4.4. Перечисление `MessageBoxOptions`



Константа	Описание
DefaultDesktopOnly	Окно с сообщением отображается только на рабочем столе, выбранном по умолчанию
RightAlign	Текст сообщения выравнивается по правому краю диалогового окна
RtlReading	Текст отображается справа налево
ServiceNotification	Окно отображается на активном рабочем столе, даже если к системе не подключился ни один пользователь. Данный параметр предназначен для приложений, работающих как сервисы ОС Microsoft Windows

Если в окне диалогового окна *MessageBox* имеется несколько кнопок, то для того, что бы определить, какую кнопку щелкнул пользователь, программа должна проанализировать значение, возвращенное методом `MessageBox.Show`.

Метод `MessageBox.Show` может вернуть одно из нескольких значений перечисления `DialogResult` (таблица 4.5).

Таблица 4.5. Перечисление `DialogResult`

Константа	Кнопка, при щелчке которой возвращается эта константа
Abort	Abort
Cancel	Cancel
Ignore	Ignore
No	No
None	Модальное диалоговое окно продолжает работать
OK	OK
Retry	Retry
Yes	Yes

Изменим метод `Remove`, добавив в него предупреждение перед удалением данных по сотруднику. Текст кода метода `Remove` должен иметь следующий вид:

```
private void Remove()
{
```

```

    DialogResult result = MessageBox.Show(" Удалить данные \n по со-
труднику? ",
    "Предупреждение", MessageBoxButtons.YesNo,
    MessageBoxIcon.Warning,
    MessageBoxDefaultButton.Button2);
    switch (result)
    {
        case DialogResult.Yes:
            {
//выполнить действия по удалению данных по сотруднику
                MessageBox.Show("Удаление данных");
                break;
            }
        case DialogResult.No:
            {
//отмена удаления данных по сотруднику
                MessageBox.Show("Отмена удаления данных");
                break;
            }
    }
}

```

В результате исполнения кода приложения и выбора пункта меню "Удалить" будет выводиться предупреждение, приведенное на рисунке 4.1 .

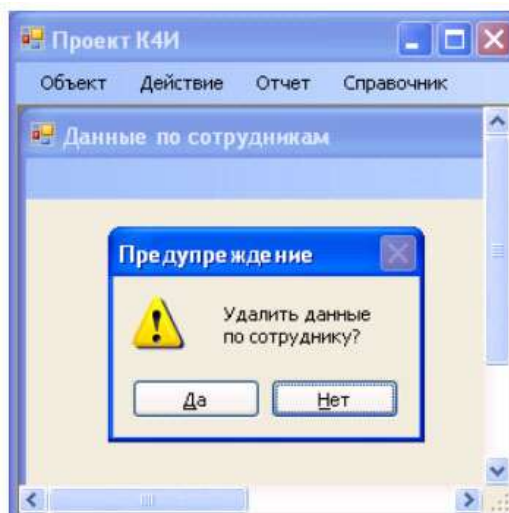


Рис. 4.1. Модальное диалоговое окно предупреждения

Диалоговое окно можно создать не только на основе класса MessageBox, но и с использованием Windows - формы.

Создадим новую форму *FormAbout* для вывода справочной информации о разрабатываемом приложении, которое должно иметь вид представленный на рисунке 4.2 .

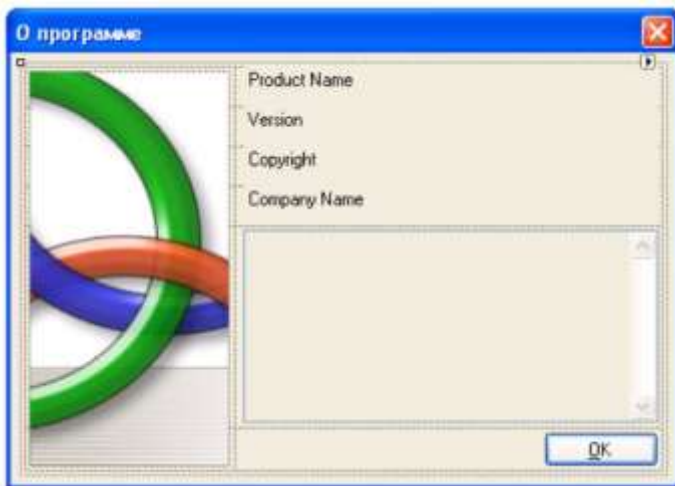


Рис. 4.2. Общий вид Windows - формы FormAbout

Для этого добавим в проект новый компонент (рисунок 4.3), выбрав из списка AboutBox (рис. 4.4).

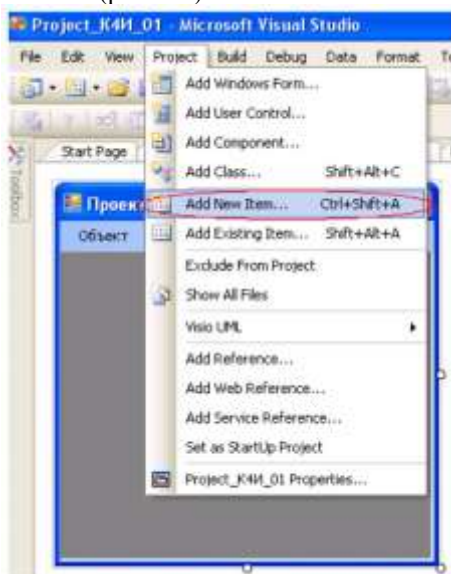


Рис. 4.3. Выбор режима добавления нового компонента в проект

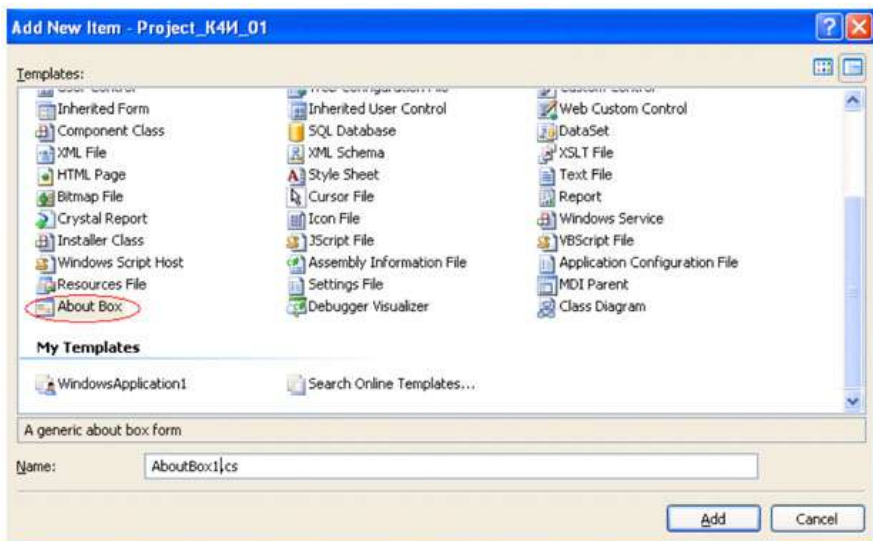


Рис. 4.4. Добавление нового компонента в проект

Для класса AboutBox можно задать логотип и дополнительную информацию. По умолчанию данный класс берет дополнительную информацию из метаданных сборки. Проверьте это.

Мы введем собственную информацию. Для этого изменим фрагмент кода конструктора класса AboutBox1 следующим образом.

```
public AboutBox1()
{
    InitializeComponent();
    this.Text = String.Format("О программе {0}", AssemblyTitle);
    this.labelProductName.Text = AssemblyProduct;
    this.labelVersion.Text = String.Format("Version {0}", AssemblyVersion);
    this.labelCopyright.Text = "@ПГЭУ, 2008";
    this.labelCompanyName.Text = "Долженко А.И.";
    this.textBoxDescription.Text = "Дисциплина Современные технологии
программирования. Студенческий проект";
}
```

Для открытия пользовательского модального диалогового окна используется метод ShowDialog. В практической работе диалоговое окно должно открываться при щелчке пользователем на пункте в меню "Справка/О программе". Код для открытия диалогового окна может выглядеть следующим образом:

```
// Открываем модальное диалоговое окно
private void aboutToolStripMenuItem_Click(object sender, EventArgs e)
{
```

```

AboutBox1 aboutBox = new AboutBox1();
aboutBox.ShowDialog(this);
}

```

Модальность формы определяет именно метод ShowDialog: при использовании кода ход выполнения программы будет приостановлен вплоть до того момента, пока метод ShowDialog не вернет соответствующее значение. Для пользователя это значит, что ему придется закрыть диалоговое окно, прежде чем он сможет выполнить какие-либо операции на главной форме.

После компиляции и загрузки приложения, вызвав пункт меню "Справка/О программе" на дисплее будет выведено диалоговое окно, приведенное на рисунке 4.5.

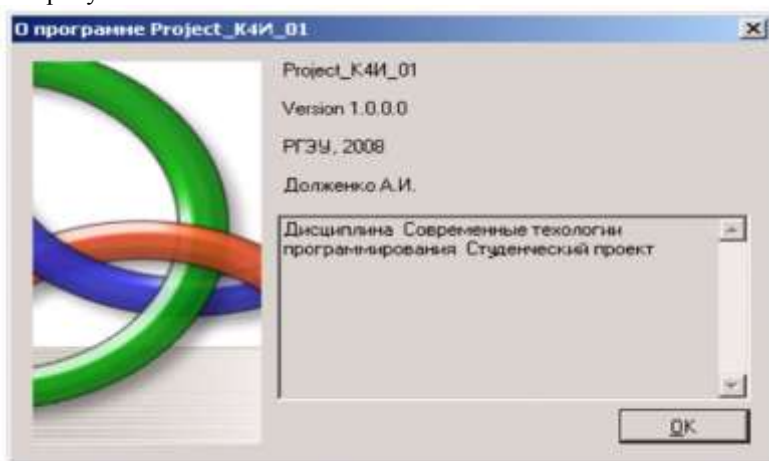


Рис. 4.5. Вызов модального окна

При нажатии на кнопку *OK* диалоговое окно будет автоматически закрыто и в ходе дальнейшего выполнения программы можно выяснить значение свойства DialogResult.

#### **Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

#### **Форма представления результата:**

- Программный код, составленный на языке программирования

#### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

**Практическое занятие №14, 15.** Создание панели инструментов и контекстного меню в среде C#

**Цель работы:** Изучить основные способы построения панели инструментов и контекстного меню, их параметры и получить практические навыки в разработке.

**Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

**Задание**

1. Изучить теоретический материал.
2. Создать панель инструментов.
3. Создать контекстное меню.
4. Написать обработчики для панели инструментов и контекстного меню.
5. Протестировать работу приложения

**Порядок выполнения**

В приложении для повышения качества интерфейса пользователя целесообразно предусматривать различные способы активизации функций системы. При выполнении практической работы 3 для приложения было создано меню, которое позволяет активизировать функции "Отменить", "Создать", "Редактировать", "Сохранить" и "Удалить". Данные

функции могут быть активизированы с помощью кнопок панели инструментов и контекстного меню.

### **Разработка панели инструментов**

Элемент управления *ToolStrip* используется непосредственно для построения панелей инструментов. Данный элемент использует набор элементов управления, происходящих от класса *ToolStripItem*.

В Visual Studio.NET предусмотрены средства, которые позволяют добавить панель инструментов при помощи графических средств. Для этого необходимо открыть панель *Toolbox* и добавьте элемент управления *ToolStrip* (рисунок 5.1) на разрабатываемую форму *FormEmployee*.

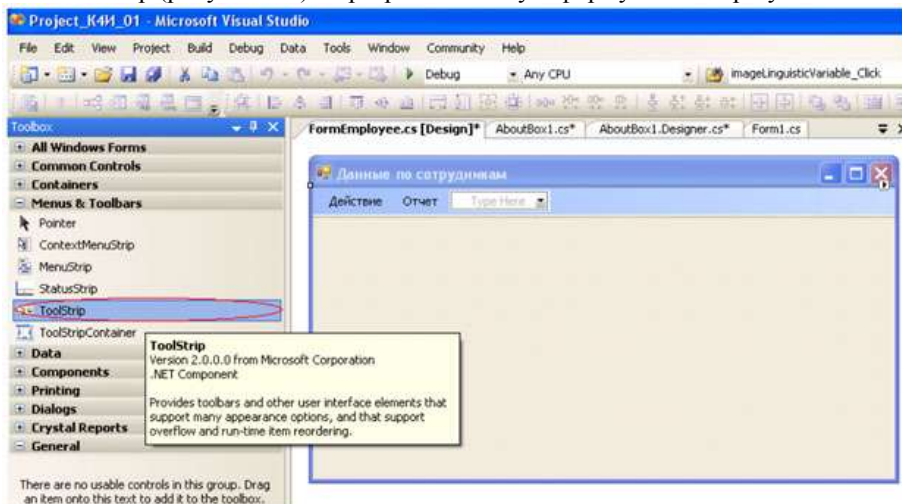


Рис. 5.1. Окно свойств панели инструментов

В выпадающем меню элемента управления *ToolStrip* на форме *FormEmployee* необходимо выбрать элемент управления *button* - кнопка (рисунок 5.2). При этом в панели инструментов добавится кнопка.

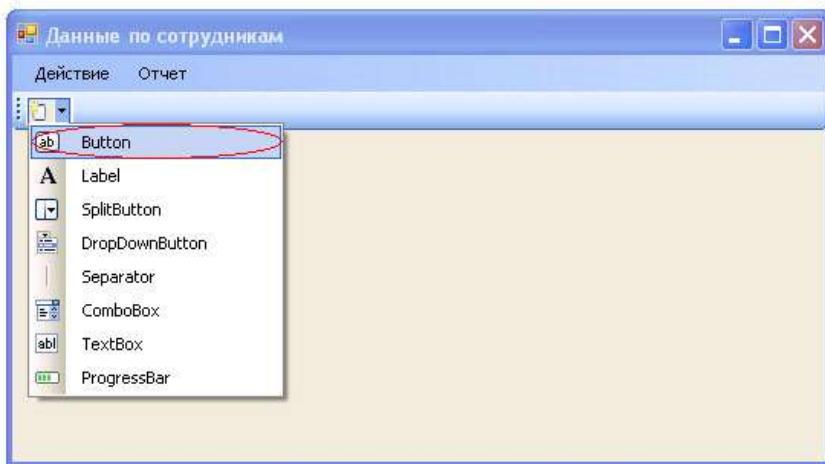


Рис. 5.2. Окно свойств панели инструментов

Добавьте на панель инструментов кнопки с именами *toolStripButtonUndo*, *toolStripButtonNew*, *toolStripButtonEdit*, *toolStripButtonSave*, *toolStripButtonRemove*. В результате должна быть сформирована панель инструментов с кнопками (рисунок 5.3).

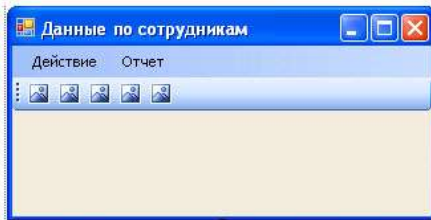



Рис. 5.3. Форма FormEmployee с панелью инструментов

Для кнопок панели инструментов сформируем графическое представление. Это можно сделать путем задания свойства *Image* соответствующей кнопке (рисунок 5.4).

При открытии коллекции свойства *Image* соответствующей кнопки, нажатии кнопки  открывается окно мастера выбора графического ресурса (рисунок 5.5).



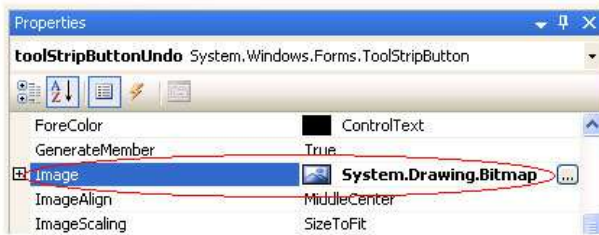


Рис. 5.4. Свойство Image для кнопки панели инструментов

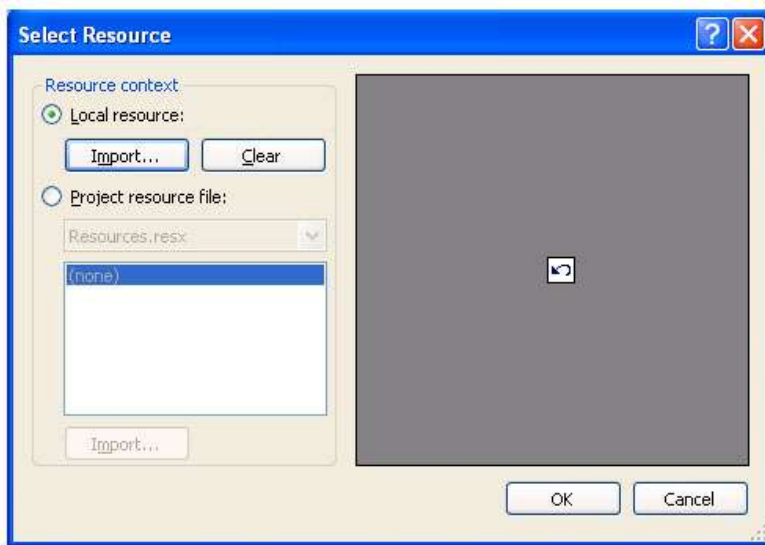


Рис. 5.5. Добавление изображения в ImageList

С помощью кнопки *Import* в локальный ресурс добавляют ссылки на необходимые графические файлы, для формирования изображения кнопок. Результаты формирования графического представления кнопок панель инструментов приведены на рисунке 5.6. Графические файлы расположены в папке Visual Studio 2005\VS2005ImageLibrary\bitmaps\commands\16color.

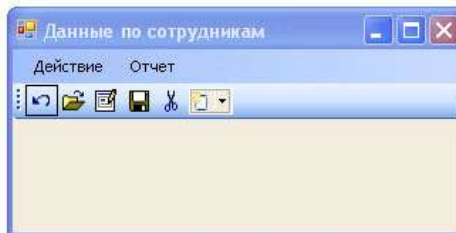


Рис. 5.6. Форма с панелью инструментов

Каждая кнопка панели инструментов, которая является объектом класса `toolStripButton`, может содержать текст, или изображение, или и то и другое.

Созданная панель инструментов содержит пять кнопок. По функциональности каждой из этих кнопок будут соответствовать пункты меню "Отменить", "Создать", "Редактировать", "Сохранить" и "Удалить".

Для удобства пользователя целесообразно снабдить кнопки панели инструментов всплывающими подсказками при фокусировке курсора на данной кнопке. Это можно сделать, если свойству `ToolTipText` класса `toolStripButton` задать значение текстовой строки с содержанием подсказки. На рисунке 5.7 для кнопки "Отменить" (`toolStripButtonUndo`) строка подсказки `ToolTipText` соответствует строке "Отменить".

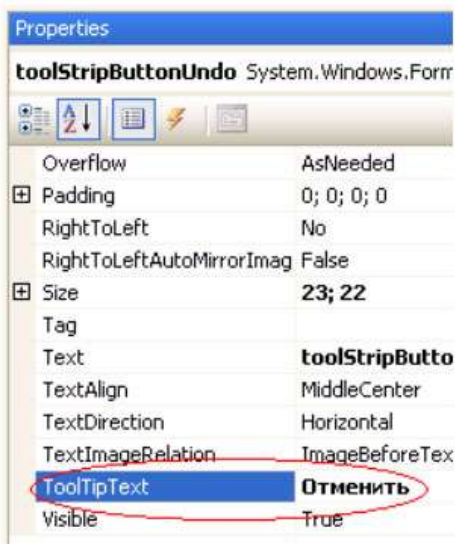


Рис. 5.7. Формирование подсказки для кнопки

На рисунке 5.8 показан вывод подсказки при фокусировке курсора на кнопке панели инструментов.

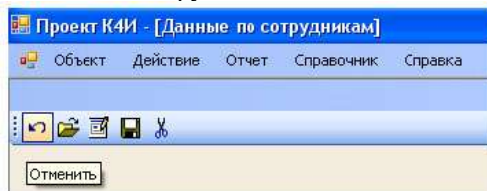


Рис. 5.8. Вывод подсказки для кнопки

Для распознавания реакции приложения при нажатии кнопок панели инструментов необходимо создать обработчик события для кнопок. При двойном нажатии на кнопку панели инструментов генерируется обработчик, в который нужно добавить вызов метода Undo. В этом случае обработчик нажатия кнопки панели инструментов будет иметь следующий вид:

```
private void toolStripButtonUndo_Click(object sender, EventArgs e)
{
    Undo();
}
```

### Контекстное меню

Класс ContextMenuStrip применяется для показа контекстного меню, или меню, отображаемого по нажатию правой кнопки мыши. Для создания объекта класса ContextMenuStrip необходимо открыть панель Toolbox и добавить элемент управления contextMenuStrip на форму FormEmployee (рисунок 5.9).

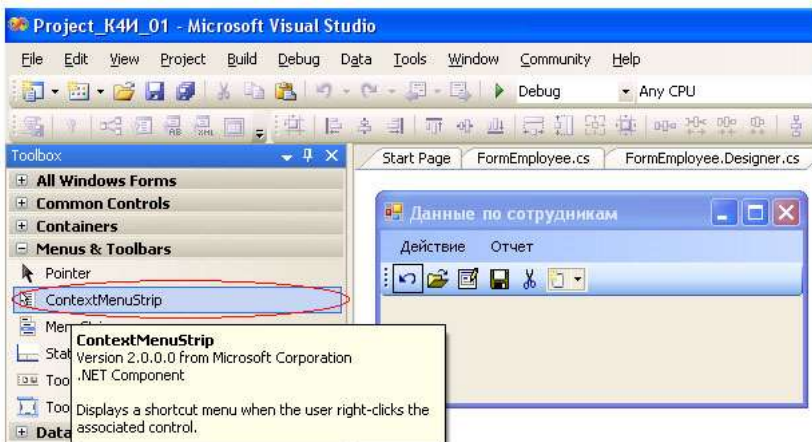


Рис. 5.9. Формирование на форме контекстного меню

В результате получаем форму FormEmployee с контекстным меню (рисунок 5.10)

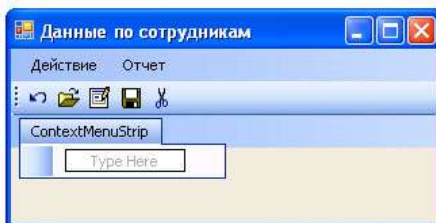


Рис. 5.10. Форма с контекстным меню

Формирование пунктов контекстного меню производится аналогично формированию пунктов главного меню (смотри "Создание главного меню приложения"). Сформированное контекстное меню приведено на рисунке 5.11.

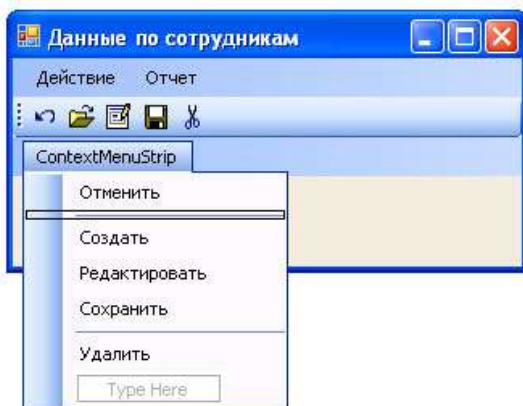


Рис. 5.11. Вид контекстного меню

После формирования пунктов контекстного меню необходимо его подключить к форме *FormEmployee*. Для этого на вкладке Свойства (*Properties*) формы *FormEmployee* строке, соответствующей свойству *ContextMenuStrip* нужно установить значение созданного объекта *contextMenuStrip1* (рис. 5.12)

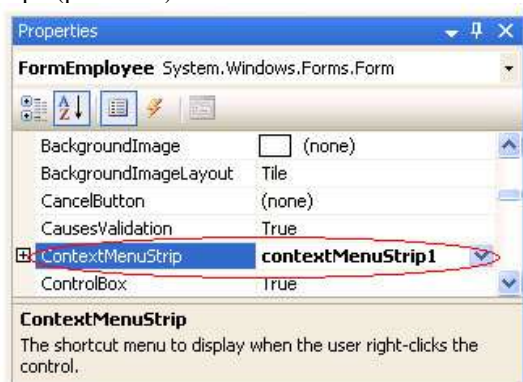


Рис. 5.12. Подключение контекстного меню к форме

После компиляции проекта и запуска приложения на выполнение можно проверить режим активизации контекстного меню. Для этого необходимо выбрать из главного меню пункт "Сотрудник" и на появившейся форме в любом месте щелкнуть правой кнопкой мыши. Результат всплывающего на форме контекстного меню показан на рисунке 5.13.

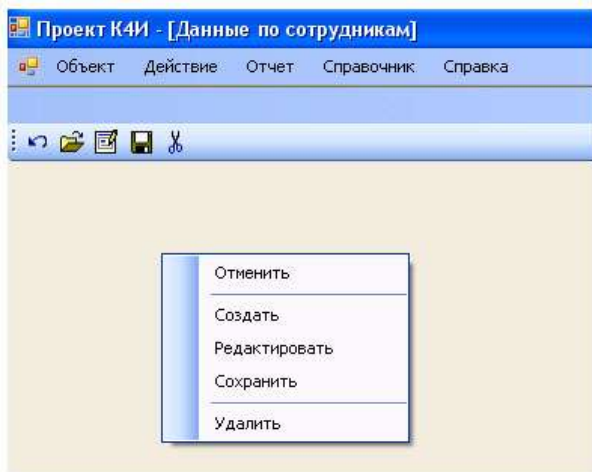


Рис. 5.13. Активизация контекстного меню

Привязка пунктов контекстного меню к конкретным функциям осуществляется путем создания кода обработчика событий для каждого пункта меню. Для формирования обработчика необходимо перейти в окно дизайнера формы `FormEmployee`, выделить на форме класс `ContextMenuStrip` и сделать двойной щелчок на соответствующем пункте меню, например "Отменить". В сгенерированном обработчике необходимо добавить вызов метода, для функции "Отменить" - метод `Undo()`. Листинг обработчика метода приведен ниже.

```
private void undoToolStripMenuItem_Click(object sender, EventArgs e)
{
    Undo();
}
```

#### **Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

#### **Форма представления результата:**

- Программный код, составленный на языке программирования

#### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

### **Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

**Практическое занятие №16, 17, 18, 19.** Создание ленточных и табличных форм для работы с данными в среде C#

**Цель работы:** Изучить основные элементы управления *Windows* -форм, их свойства и методы, а также получить практические навыки в разработке *Windows* -форм с элементами контроля

#### **Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

#### **Материальное обеспечение**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

#### **Задание**

1. Изучить теоретический материал.
2. Для формы *FormEmployee* создать требуемые элементы контроля.
3. Разработать методы для задания режимов "Просмотр", "Редактирование" для элементов контроля.
4. Разработать методы для задания режимов "Просмотр", "Редактирование" для управления активностью пунктов главного меню формы, контекстного меню и кнопок панели инструментов.
5. Сформировать обработчик события Load.
6. Протестировать программу.

#### **Общие сведения**

Классы, представляющие графические элементы управления, находятся в пространстве имен System.Windows.Forms. С их помощью обеспечивается реакция на действия пользователя в приложении Windows Forms. Классы элементов управления связаны между собой достаточно сложными отношениями наследования. Общая схема таких отношений представлена на рисунке 7.1.

Класс Control, как общий предок, обеспечивает все производные классы общим набором важнейших возможностей. В числе этих возможностей можно перечислить события мыши и клавиатуры, физические размеры и местонахождение элемента управления (свойства Height, Width, Left, Right, Location), установку цвета фона и цвета переднего плана, выбор шрифта и т.п.

При создании приложения можно добавить элементы управления на форму при помощи графических средств Visual Studio. Обычно достаточно выбрать нужный элемент управления в окне ToolBox и поместить его на форму. Visual Studio автоматически сгенерирует нужный код для формы. После этого можно изменить название элемента управления на более содержательное (например, вместо button1, предлагаемого по умолчанию, - buttonPrimer) Visual Studio позволяет не только размещать на форме элементы управления, но и настраивать их свойства. Для этого достаточно щелкнуть на элементе управления правой кнопкой мыши и в контекстном меню выбрать Properties (Свойства).

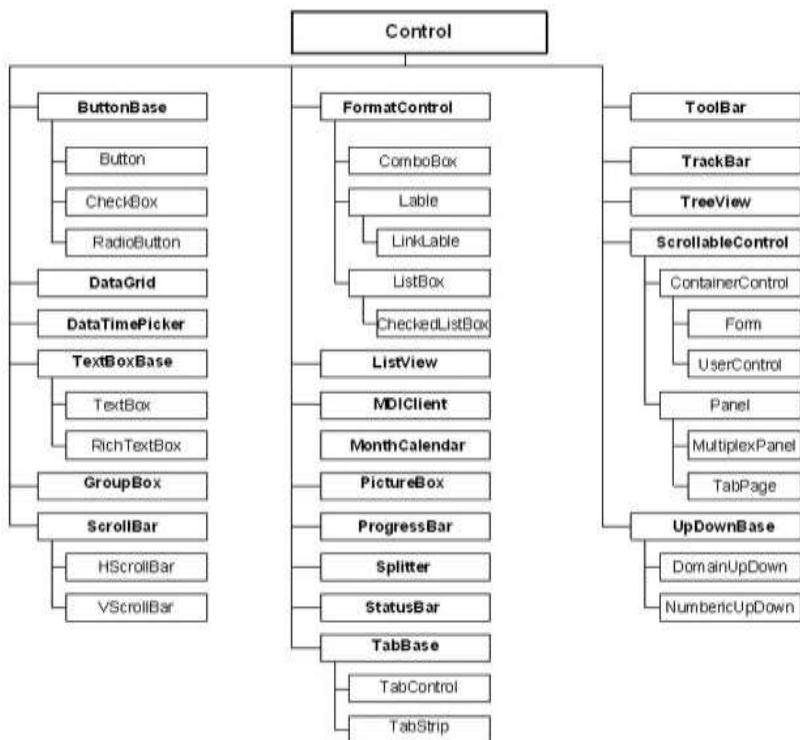


Рис. 7.1. Иерархия элементов управления Windows Forms  
 Все изменения, которые необходимо произвести в открывшемся окне (рисунок 7.2), будут добавлены в код метода InitializeComponents().

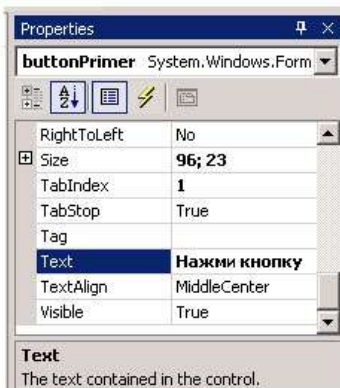



Рис. 7.2. Настройка элементов управления средствами Visual Studio  
 То же самое окно позволяет настроить не только свойства данно-



го элемента управления, но и обработку событий этого элемента. Перейти в список событий можно при помощи кнопки  в закладке Properties (рисунок 7.3). Можно выбрать в списке нужное событие и рядом с ним сделать двойной щелчок или ввести имя метода или выбрать метод из списка.

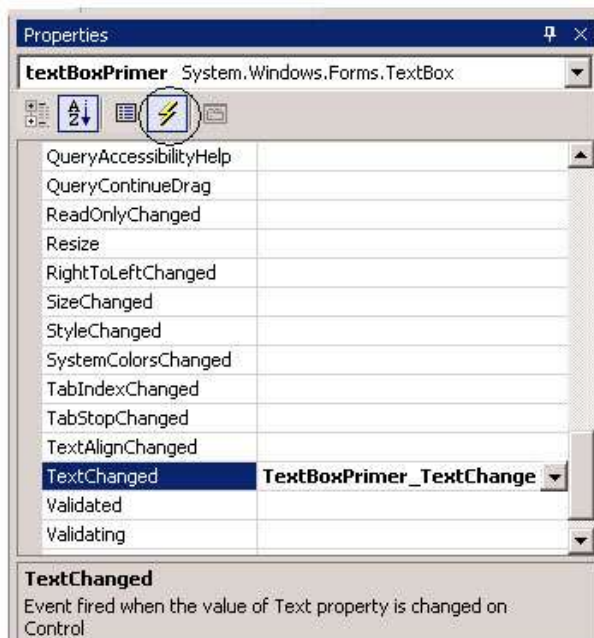


Рис. 7.3. Настройка обработчиков событий

После задания имени метода или двойного щелчка Visual Studio сгенерирует заготовку для обработчика события.

Рассмотрим основные элементы управления Windows -форм.

Элемент управления TextBox

Элемент управления TextBox (текстовое окно) предназначен для хранения текста (одной или нескольких строк). При желании текст в TextBox может быть настроен как "только для чтения", а в правой и нижней части можно поместить полосы прокрутки.

Класс TextBox происходит непосредственно от класса TextBoxBase, обеспечивает общими возможностями как TextBox, так и RichTextBox. Свойства, определенные в TextBoxBase, представлены в таблице 7.1.

Таблица 7.1. Свойства TextBoxBase

Свойство	Назначение
AcceptsTab	Определяет, что будет производиться при нажатии на клавишу Tab: вставка символа табуляции в само поле или переход к другому элементу управления
AutoSize	Определяет, будет ли элемент управления автоматически изменять размер при изменении шрифта на нем
BackColor, ForeColor	Позволяют получить или установить значение цвета фона и переднего плана
HideSelection	Позволяет получить или установить значение, определяющее, будет ли текст в TextBox оставаться выделенным после того, как этот элемент управления будет выведен из фокуса
MaxLength	Определяет максимальное количество символов, которое можно будет ввести в TextBox
Modified	Позволяет получить или установить значение, определяющее, был ли текст в TextBox изменен пользователем
Multiline	Указывает, может ли TextBox содержать несколько строк текста
ReadOnly	Помечает TextBox как "только для чтения"
SelectedText, SelectionLength	Содержат выделенный текст (или определенное количество символов) в TextBox
SelectionStart	Позволяет получить начало выделенного текста в TextBox
Wordwrap	Определяет, будет ли текст в TextBox автоматически переноситься на новую строку при достижении предельной длины строки

В TextBoxBase также определено множество методов: для работы с буфером обмена (Cut, Copy и Paste), отменой ввода (Undo) и прочими возможностями редактирования (Clear, AppendText и т. п.).

Из всех событий, определенных в TextBoxBase, наибольший интерес представляет событие TextChange. Это событие происходит при изменении текста в объекте класса, производном от TextBoxBase. Например, его можно использовать для проверки допустимости вводимых

пользователем символов (например, предположим, что пользователь должен вводить в поле только цифры или, наоборот, только буквы).

Свойства, унаследованные от Control и от TextBoxBase, определяют большую часть возможностей TextBox. Свойств, определенных непосредственно в классе TextBox, не так уж и много. Они представлены в таблице 7.2.

Таблица 7.2. Свойства, определенные в классе TextBox

Свойство	Назначение
AcceptsReturn	Позволяет определить, что происходит, когда пользователь при вводе текста нажал на Enter. Варианта два: либо в TextBox начинается новая строка текста, либо активизируется кнопка по умолчанию на форме
CharacterCasing	Позволяет получить или установить значение, определяющее, будет ли изменяться регистр вводимых пользователем символов
PasswordChar	Позволяет выбрать символ, используемый для отображения вводимых пользователем данных (в поле для ввода пароля)
ScrollBars	Позволяет получить или установить значение, определяющее, будут ли в TextBox с несколькими строками присутствовать полосы прокрутки
TextAlign	Позволяет определить выравнивание текста в TextBox (используются значения из перечисления HorizontalAlignment)

Значения перечисления HorizontalAlignment представлены в таблице 7.3.

Таблица 7.3. Значения перечисления HorizontalAlignment

Значение	Описание
Center	Выравнивание по центру
Left	Выравнивание по левому краю
Right	Выравнивание по правому краю

### **Класс Button**

Кнопка (button) - это самый простой из всех элементов управления и при этом наиболее часто используемый. Можно сказать, что кнопка - это возможность принять ввод (щелчок кнопкой мыши или набор на

клавиатуре) наиболее простым способом. Непосредственный предок класса `System.Windows.FormButton` в иерархии классов .NET - это класс `ButtonBase`, обеспечивающий общие возможности для целой группы производных от него элементов управления (таких как `Button`, `CheckBox` и `RadioButton`). Некоторые свойства `ButtonBase` представлены в таблице 7.4.

Таблица 7.4. Свойства `ButtonBase`

Свойство	Назначение
<code>FlatStyle</code>	Позволяет настроить "рельефность" кнопки. Используются значения из перечисления <code>FlatStyle</code>
<code>Image</code>	Позволяет задать изображение, которое будет выводиться на кнопке (при этом можно указать точное местонахождение изображения). Фоновый рисунок лучше настраивать при помощи свойства <code>BackgroundImage</code> , определенного в базовом классе <code>Control</code>
<code>ImageAlign</code>	Позволяет определить выравнивание изображения, размещенного на кнопке. Используются значения из перечисления <code>ContentAlignment</code>
<code>ImageIndex</code> , <code>ImageList</code>	Эти свойства используются для работы с набором изображений (объектом <code>ImageList</code> ), выводимых на кнопке
<code>IsDefault</code>	Определяет, будет ли эта кнопка являться кнопкой по умолчанию (то есть срабатывать при нажатии на <code>Enter</code> )
<code>TextAlign</code>	Позволяет получить или установить выравнивание текста на кнопке. Также используются значения из перечисления <code>ContentAlignment</code>

Сам класс `Button` не определяет каких-либо дополнительных возможностей помимо унаследованных от `ButtonBase`, за единственным, но существенным исключением свойства `DialogResult`. Это свойство позволяет возвращать значение при закрытии диалогового окна, например, при нажатии кнопок `OK` или `Cancel` (Отменить).

В подавляющем большинстве случаев выравнивание текста, размещенного на кнопке, производится по центру, так что текст будет размещен строго посередине кнопки. Однако если нам по каким-то причинам необходимо использовать другой стиль выравнивания, в нашем распоряжении - свойство `TextAlign`, определенное в классе `ButtonBase`. Для `TextAlign` используются значения из перечисления `ContentAlignment` (таблица 7.5). Значения из того же перечисления используются и для определения положения изображения на кнопке.

Таблица 7.5. Значения перечисления ContentAlignment

Значение	Описание (выравнивание)
BottomCenter	По нижнему краю кнопки, относительно боковых краев - посередине
BottomLeft	По нижнему краю кнопки, слева
BottomRight	По нижнему краю кнопки, справа
MiddleCenter	По центру кнопки
MiddleLeft	Относительно верхнего и нижнего краев - по центру, относительно боковых краев - слева
MiddleRight	Относительно верхнего и нижнего краев - по центру, относительно боковых краев - справа
TopCenter	По верхнему краю кнопки, относительно боковых краев - посередине
TopLeft	По верхнему краю кнопки, слева
TopRight	По верхнему краю кнопки, справа

### **Флажки**

Для флажка (тип CheckBox) предусмотрено три возможных состояния. Как и тип Button, CheckBox наследует большую часть своих возможностей от базовых классов Control и ButtonBase. Однако в этом классе существуют и свои собственные члены, обеспечивающие дополнительные уникальные возможности. Наиболее важные свойства CheckBox представлены в таблице 7.6.

Таблица 7.6. Свойства класса CheckBox

Свойство	Назначение
Appearance	Настраивает вид флажка. Для этого свойства используются значения из перечисления Appearance
AutoCheck	Позволяет получить или установить значение, определяющее, будут ли значения Checked и CheckState, а также внешний вид флажка автоматически изменяться при щелчке на нем
CheckAlign	Позволяет установить горизонтальное и вертикальное выравнивание собственно флажка (квадратика) в элементе

управления CheckBox. Используются значения из перечисления ContentAlignment

Checked	Возвращает значение типа bool, представляющее текущее состояние флажка (выбран или не выбран) Если для свойства ThreeState установлено значение true, то свойство Checked будет возвращать true как для явно выбранного флажка, так и для того флажка, для которого установлено значение "не определено" (indeterminate)
CheckState	Позволяет получить или установить значение флажка (установлен - не установлен - не определено), используя не true и false, как в Checked, а три значения из перечисления CheckState. Обычно используется, если свойство ThreeState для флажка имеет значение true (то есть он допускает три значения).
ThreeState	Определяет, будут ли для флажка использоваться три значения (из перечисления CheckState) или только два

Возможные состояния флажка (Indeterminate можно использовать только тогда, когда для свойства ThreeState установлено значение true) представлены в таблице 7.7.

Таблица 7.7. Значения перечисления CheckState

Значение	Описание
Checked	Флажок установлен
Indeterminate	Значение не определено (обычно флажок выглядит как "серый", затененный)
Unchecked	Флажок снят

Состояние "значение не определено" (indeterminate) может быть установлено, например, для верхнего элемента иерархии, в которой для одной части подчиненных элементов флажок установлен, а для другой - снят.

Переключатели и группирующие рамки

Тип RadioButton (переключатель) можно воспринимать, как несколько видоизмененный флажок при этом сходство между этими типами подчеркивается почти полным совпадением наборов членов. Между типами RadioButton и CheckBox существуют лишь два важных различия: в RadioButton предусмотрено событие CheckedChanged (возникающее при изменении значения Checked), а кроме того, RadioButton не поддерживает свойство ThreeState и не может принимать состояние

Indeterminate (не определено).

Переключатели всегда используются в группах, которые рассматриваются как некое единое целое. Внутри группы переключателей одновременно может быть выбран только один переключатель. Для группировки переключателей в группы используется тип `GroupBox`.

И флажок (`CheckBox`), и переключатель (`RadioButton`) поддерживают свойство `Checked`, при помощи которого очень удобно получать информацию о состоянии соответственно флажка и переключателя. Однако если есть необходимость задействовать дополнительное третье состояние флажка (не определено - `Indeterminate`), то придется вместо `Checked` использовать свойство `CheckState` и значения из одноименного перечисления `CheckState`.

Элемент управления `CheckedListBox`

Типы `Button`, `CheckBox` и `RadioButton` являются производными от `ButtonBase`, и их можно определить как некие разновидности кнопок. К членам семейства списков относятся `CheckedListBox` (список с флажками), `ListBox` (список) и `ComboBox` (комбинированный список).

Элемент управления `CheckedListBox` (список с флажками) позволяет помещать обычные флажки внутри поля с полосами прокрутки.

Кроме того, в элементе управления `CheckedListBox` предусмотрена возможность использования нескольких столбцов. Для этого достаточно установить значение `true` для свойства `MultiColumn`.

`CheckedListBox` наследует большинство своих возможностей от типа `ListBox`. То же самое справедливо и в отношении класса `ComboBox`. Наиболее важные свойства `System.Windows.Forms.ListBox` представлены в таблице 7.8.

Таблица 7.8. Свойства класса `ListBox`

Свойство	Назначение
<code>ScrollAlwaysVisible</code>	Определяет, будет ли полоса прокрутки выводиться всегда
<code>SelectedIndex</code>	Индекс выделенного в настоящий момент элемента в списке (если такой имеется). Если ни один элемент не выделен, то возвращается значение -1
<code>SelectedIndices</code>	Набор индексов выделенных в настоящий момент элементов в списке. Если не выделен ни один элемент, то возвращается пустой набор
<code>SelectedItem</code>	Значение выделенного в настоящий момент элемента. Если ни один из элементов не выделен, то возвращается <code>null</code>

SelectedItems	Возвращает коллекцию значений выделенных элементов (для списков, в которых допускается выбор нескольких значений)
SelectionMode	Определяет число элементов, которые возможно выбрать в списке одновременно. Для этого свойства используются значения из перечисления SelectionMode
Sorted	Определяет, будут ли элементы в списке упорядочены (по алфавиту) или нет
TopIndex	Возвращает индекс первого видимого элемента в списке

Помимо свойств в классе `ListBox` определены также многочисленные методы. Подавляющее большинство этих методов дублирует возможности, предоставляемые в наше распоряжение свойствами, поэтому мы их рассматривать не будем.

#### Комбинированные списки

Как и списки (объекты `ListBox`), комбинированные списки (объекты `ComboBox`) позволяют пользователю производить выбор из списка заранее определенных элементов. Однако у комбинированных списков есть одно существенное отличие от обычных: пользователь может не только выбрать готовое значение из списка, но и ввести свое собственное. Класс `ComboBox` наследует большинство своих возможностей от класса `ListBox` (который, в свою очередь, является производным от `Control`), однако в нем предусмотрены и собственные важные свойства, представленные в таблице 7.9.

Таблица 7.9. Свойства `ComboBox`

Свойство	Назначение
DroppedDown	"Раскрывающийся вниз": определяет, будет ли список ниспадающим
MaxDropDownItems	Определяет максимальное количество элементов, которое будет показано в нижней части ниспадающего списка. Допустимые значения - от 1 до 100
MaxLength	Определяет максимальную длину текста, который пользователь может ввести в <code>ComboBox</code>
SelectedIndex	Определяет индекс выделенного элемента <code>ComboBox</code> . Если ни один элемент не выделен, воз-



	вращается значение -1
SelectedItem	Возвращает ссылку на объект выделенного элемента ComboBox
SelectedText	Возвращает выделенный текст в поле редактирования ComboBox
SelectionLength	Определяет длину (в символах) выделенного текста в поле редактирования ComboBox
Style	Позволяет получить или установить стиль ComboBox. Для этого свойства используются значения из перечисления ComboBoxStyle
Text	Позволяет получить доступ к тексту в поле редактирования. При работе с ComboBox это унаследованное свойство используется чаще всех остальных

Стиль для ComboBox можно настроить при помощи свойства Style, для которого используются значения из перечисления ComboBoxStyle (таблица 7.10).

Таблица 7.10. Значения перечисления ComboBoxStyle

Значение	Описание
DropDown	Пользователь может вводить значения в поле редактирования. Для отображения списка пользователь должен нажать на кнопку со стрелкой, направленной вниз (Arrow Button)
DropDownList	Пользователь не может вводить значения в поле редактирования. Для отображения списка пользователь должен нажать на кнопку со стрелкой, направленной вниз (Arrow Button)
Simple	Пользователь может вводить значения в поле редактирования. Список значений виден всегда

### ***Порядок перехода по Tab***

Если на форме размещено несколько элементов управления, то пользователи обычно ожидают, что между ними можно будет перемещаться с помощью клавиши Tab. Часто бывает необходимо после размещения элементов управления настроить порядок перехода между ними. Для этого используются два свойства (унаследованные от базового класса Control и поэтому общие для всех элементов управления): TabStop и

TabIndex. Для свойства TabStop используются только два значения: true и false. Если для TabStop установлено значение true, то к этому элементу управления можно будет добраться с помощью клавиши Tab. Если же установлено значение false, то участвовать в переходах по Tab этот элемент управления не будет. Если элемент управления TabStop имеет значение true, то очередность перехода можно настроить с помощью свойства TabIndex:

В Visual Studio.NET предусмотрено средство, при помощи которого можно быстро настроить порядок перехода для элементов управления на форме. Это средство называется Tab Order Wizard и оно доступно из меню View (View > Tab Order). Чтобы изменить значения TabIndex для каждого элемента управления, достаточно просто щелкать мышью на элементах управления в выбранном нами порядке перехода. Для элементов управления, помещенных в группирующую рамку, Tab Order Wizard создает отдельную последовательность перехода.

#### **Элемент управления MonthCalendar**

В пространстве имен System.Windows.Forms предусмотрен элемент управления, при помощи которого пользователь может выбрать дату или диапазон дат, используя дружественный и удобный интерфейс. Это элемент управления MonthCalendar.

Наиболее важные свойства MonthCalendar представлены в табл. 7.11.

Таблица 7.11. Свойства MonthCalendar

Свойство	Назначение
BoldedDates	Массив объектов DateTime, выделенных подсветкой
CalendarDimensions	Определяет количество выводимых строк и столбцов
FirstDayOfWeek	Определяет, с какого дня будет начинаться неделя в MonthCalendar
MaxDate	Самая поздняя дата, которую разрешается выбрать пользователю (по умолчанию ограничений нет)
MaxSelectionCount	Максимальное количество дат, которое одновременно может выбрать пользователь
MinDate	Самая ранняя дата, которую разрешается выбрать пользователю (по умолчанию ограничений нет)
MonthlyBoldedDates	Массив выделенных подсветкой объектов DateTime для месяца

SelectionRange	Диапазон выделенных объектов
SelectionEnd	Самая поздняя дата в диапазоне выделенных объектов
SelectionStart	Самая ранняя дата в диапазоне выделенных объектов
ShowToday	Определяет, будет ли MonthCalendar выводить информацию о текущей дате
ShowTodayCircle	Определяет, будет ли MonthCalendar выводить информацию о текущей дате в нижней части и выделять ее в календаре обводкой
ShowWeekNumbers	Определяет, будет ли MonthCalendar отображать номера недель справа от каждой строки
TodayDate	Дата, которая будет считаться MonthCalendar сегодняшней. По умолчанию TodayDate - это системная дата на момент создания объекта MonthCalendar
TodayDateSet	Определяет, можно ли пользователю по своему усмотрению выбирать сегодняшнюю дату. Если для этого свойства установлено значение true, пользователь может выбрать в качестве сегодняшней (TodayDate) любое число

По умолчанию всегда выделяется (и подсветкой, и обводкой) текущая дата. Пользователь может выбрать другую дату - в этом и есть смысл графического интерфейса MonthCalendar.

Можно получить дату, выбранную пользователем в MonthCalendar, при помощи свойства SelectionStart. Это свойство возвращает ссылку на объект DateTime, которая хранится в специальной переменной (d) При помощи набора свойств типа DateTime можно извлечь всю необходимую информацию в нужном нам формате.

При помощи свойств Month, Day и Year можно извлечь из объектов DateTime нужную информацию и сформировали текстовые строки. Это вполне допустимый подход. Дело в том, что дату в необходимом текстовом формате проще получить из DateTime при помощи специальных "форматирующих" свойств самих объектов DateTime. Набор таких свойств (и некоторые методы) представлен в таблице 7.12.

Таблица 7.12. Члены класса DateTime

Член	Назначение
------	------------

Date	Позволяет получить информацию о дате (дата всегда отсчитывается от полуночи)
Day, Month, Year	Позволяют получить соответственно день, месяц и число из текущего объекта DateTime
DayOfWeek	Возвращает день недели для объекта DateTime
DayOfYear	Возвращает номер дня в году для объекта DateTime
Hour, Minute, Second, Millisecond	Возвращают информацию о часе, минутах, секундах и миллисекундах для объекта DateTime
MaxValue, MinValue	Возвращают минимальное и максимальное значения для DateTime
Now, Today	Эти два статических свойства типа DateTime позволяют получить информацию о текущей дате и времени (Now) или только о текущей дате (Today)
Ticks	Позволяет получить счетчик "тиков" (с интервалом в 100 наносекунд) для объекта DateTime
ToLongDateString(), ToLongTimeString(), ToShortDateString(), ToShortTimeString()	Преобразуют текущее значение объекта DateTime в разные виды текстового представления

При помощи вышеперечисленных членов можно значительно упростить вывод текстовой информации о дате.

### ***Элемент управления Panel***

Назначение элемента управления Panel (панель) - с его помощью можно объединить прочие элементы управления на форме. Panel происходит от базового класса ScrollableControl и поддерживает полосы прокрутки.

Элементы управления Panel обычно используются для экономии пространства на форме. Например, если элементы управления, которые планируем разместить на форме, на ней не умещаются, то можно поместить их внутрь Panel и установить для свойства AutoScroll объекта Panel зна-

чение true. В результате пользователь получит возможность доступа к "невмещающимся" элементам управления с помощью полос прокрутки.

### **Всплывающие подсказки (ToolTips)**

Большинство приложений с современным пользовательским интерфейсом поддерживают всплывающие подсказки. В приложениях .NET эта возможность реализуется при помощи типа System.Windows.Forms.ToolTip. ToolTip (всплывающие подсказки) - это небольшие окна с текстом, появляющиеся при наведении указателя мыши на элемент управления на форме. Наиболее важные члены класса ToolTip представлены в таблице 7.13.

Таблица 7.13. Члены класса ToolTip

Член	Назначение
Active	Определяет, будет ли всплывающая подсказка активной. Возможность отключить всплывающие подсказки может быть полезной, например, если в приложении предусмотрено два варианта интерфейса: для обычных и для опытных пользователей
AutomaticDelay	Позволяет получить или установить время задержки (в миллисекундах) при появлении подсказки
AutoPopDelay	Время (в миллисекундах), в течение которого подсказка остается видимой, если указатель мыши неподвижен и находится в области, занимаемой соответствующим элементом управления. По умолчанию это значение равно 10 значениям AutomaticDelay
GetTooltip()	Возвращает текст подсказки
InitialDelay	Время (в миллисекундах), в течение которого указатель должен оставаться неподвижным в соответствующей области для появления подсказки. Значение по умолчанию равно значению AutomaticDelay
ReshowDelay	Время (в миллисекундах), в течение которого появится другая подсказка при перемещении указателя мыши от одного элемента управления к другому. По умолчанию это значение равно 1/5 от значения AutomaticDelay
SetToolTip()	Ассоциирует подсказку с элементом управления

Для того чтобы настроить использование всплывающих подсказок для элементов управления можно сделать это с помощью графических средств Visual Studio.

Первое, что необходимо сделать - добавить на форму объект ToolTip, выбрав его в ToolBox. Затем можно указать текст всплывающей подсказки для любого элемента управления на форме (в том числе и для самой формы) из окна свойств данного элемента.

Проектирование элементов управления формы FormEmployee

Для разработки проекта приложения форма FormEmployee должна содержать элементы управления, в соответствии с видом, приведенном на рисунке 7.4.

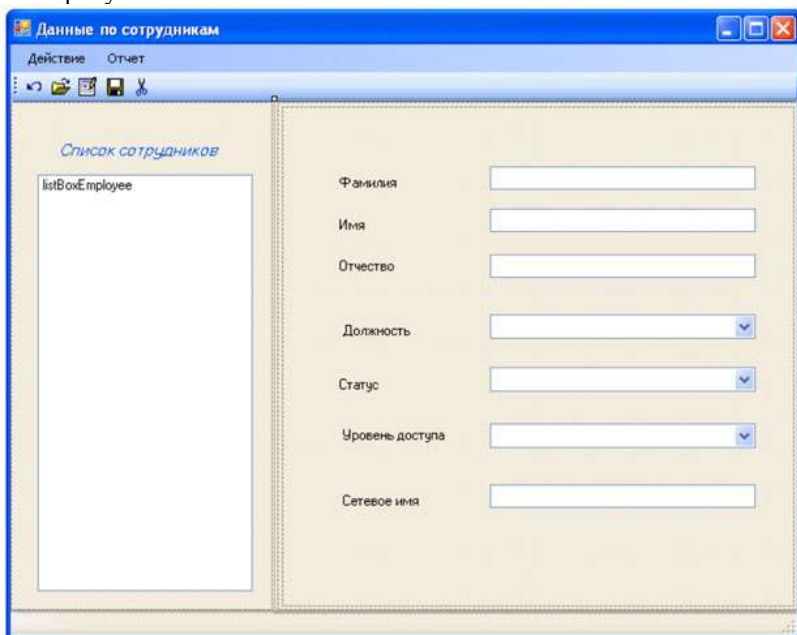


Рис. 7.4. Вид экранной формы FormEmployee

На данной форме необходимо сформировать элементы управления, приведенные в таблице 7.14.

Таблица 7.14. Элементы управления формы FormEmployee

Элемент контроля	Имя	Свойство Text/Items	Назначение
SplitContainer	splitContainerEmployee		Две панели с разделителем
Label	labelListEmployee	Список сотрудников	Надпись
listBox	listBoxEmployee		Список со-

			трудников
Label	labelSurname	Фамилия	Надпись
Label	labelName	Имя	Надпись
Label	labelPatronymic	Отчество	Надпись
Label	labelJobRole	Должность	Надпись
Label	labelStatus	Статус	Надпись
Label	labelAccess	Уровень доступа	Надпись
label	labelNetName	Сетевое имя	Надпись
textBox	textBoxNetName		Сетевое имя
textBox	textBoxSurname		Фамилия
textBox	textBoxName		Имя
textBox	textBoxPatronymic		Отчество
comboBox	comboBoxJobRole		Должность
comboBox	comboBoxStatus	Активен, выходящий, в отпуске, болеет, не работает, помечен как удаленный	Статус
comboBox	comboBoxAccess	Оператор, старший оператор, начальник смены, администратор, аналитик	Уровень доступа
menuItem	menuItemAction	Действие	Пункт меню "Редактировать"
menuItem	menuItemUndo	Отменить	Подпункт меню "Отменить"

menuItem	menuItemNew	Создать	Подпункт меню "Новый"
menuItem	menuItemEdit	Изменить	Подпункт меню "Изменить"
menuItem	menuItemSave	Сохранить	Подпункт меню "Сохранить"
menuItem	menuItem	Удалить	Подпункт меню "Удалить"
menuItem	menuItemReport	Отчет	Пункт меню "Отчет"
menuItem	menuItemReport1	По сотруднику	Подпункт меню "Отчет по сотруднику"
menuItem	menuItemReport2	По всем сотрудникам	Подпункт меню "Отчет по всем сотрудникам"

Вначале создайте на форме элемент SplitContainer (рисунок 7.5). На панели 1 создайте элементы управления labelListEmployee и listBoxEmployee (рисунок 7.6), а остальные элементы управления, приведенные в таблице 7.14, - на панели 2 (рисунок 7.4).

После создания на форме FormEmployee элементов управления в соответствии с таблицей 7.14 необходимо настроить порядок перехода между ними при нажатии клавиши Tab.





Рис. 7.5. Создание панелей на форме FormEmployee

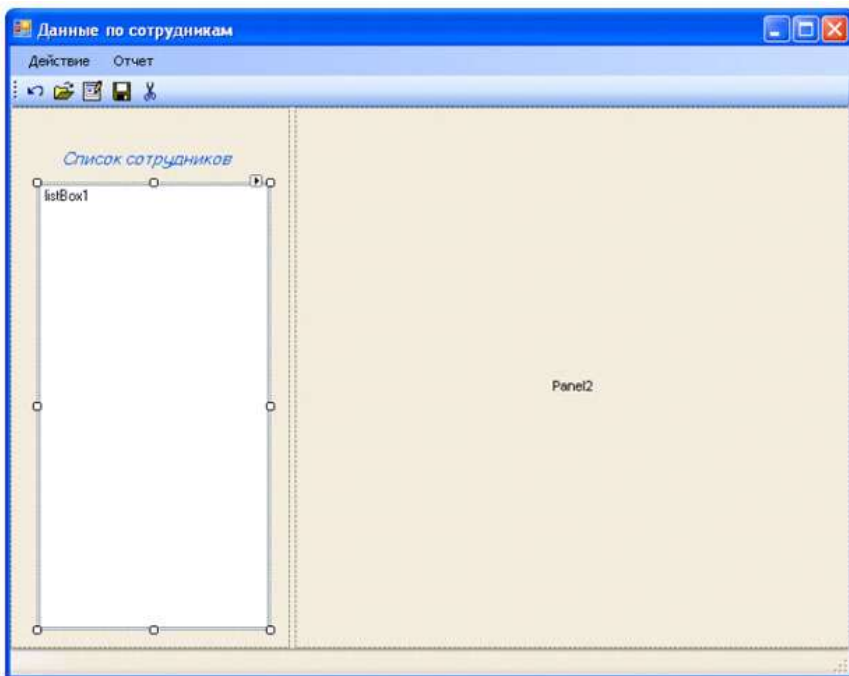


Рис. 7.6. Формирование элементов управления на панели 1 формы FormEmployee

Для этого необходимо задать последовательные номера свойству `TabIndex` элементов управления (в разрабатываемой форме это необходимо сделать для элементов управления `TextBox` и `ComboBox`) из окна *Properties* (рисунок 7.7) или вызвать мастер *Tab Order Wizard* из меню *View/Tab Order* (рисунок 7.8). Задание последовательности значений свойству `TabIndex` производится щелчком мыши на элементах управления в заданной последовательности.

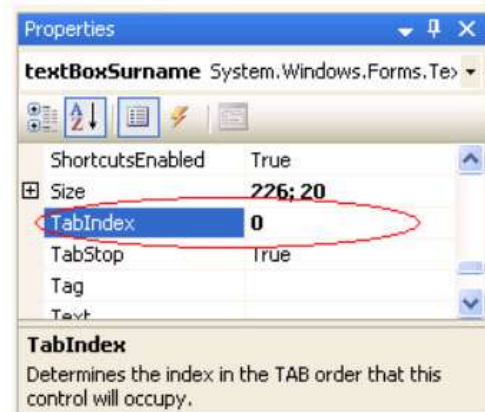


Рис. 7.7. Задание свойства TabIndex для элемента контроля

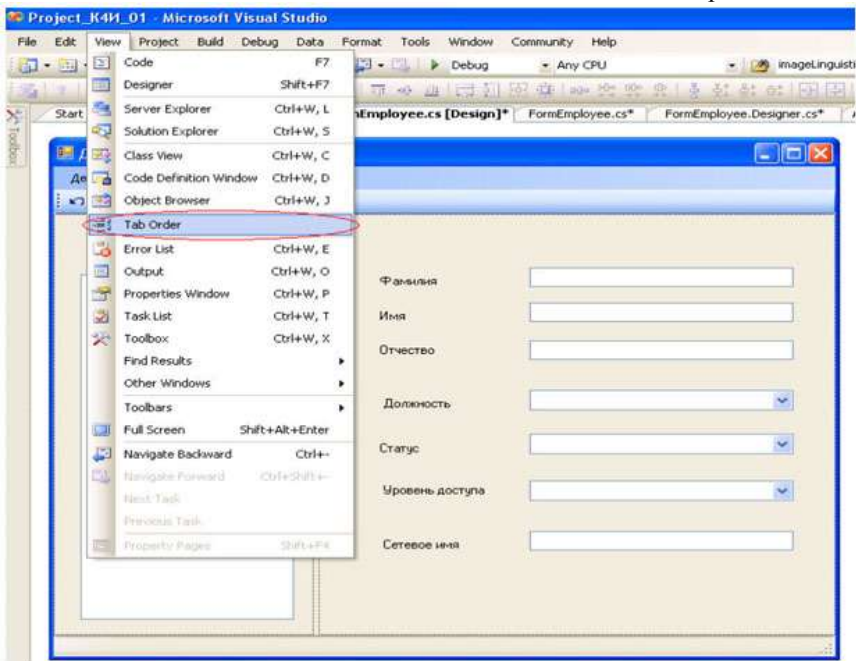


Рис. 7.8. Настройка перехода по элементам управления

Результат настройки порядка перехода между элементами управления при нажатии клавиши *Tab* приведен на рисунке 7.9.

Для работы с формой необходимо создать методы, которые разрешают только просматривать форму (режим просмотра) и редактировать форму (режим редактирования).

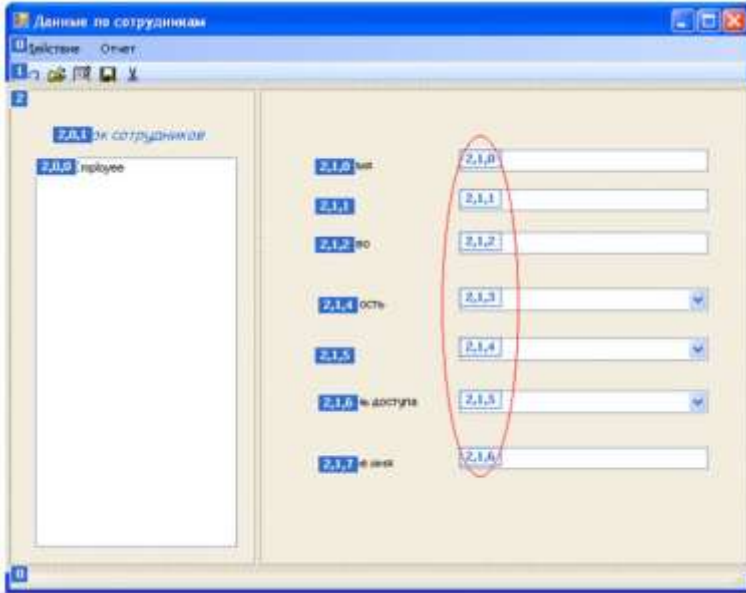


Рис. 7.9. Результат работы мастера Tab Order Wizard

Создадим метод для задания режима просмотра формы DisplayReadOnly. Метод DisplayReadOnly должен быть общедоступным, ничего не должен возвращать и не иметь параметров. Для задания режима просмотра (только для чтения) объекту класса TextBox необходимо свойству ReadOnly присвоить значение true, а для объекта класса comboBox - свойству Enabled значение false. Код метода DisplayReadOnly представлен далее:

```

public void DisplayReadOnly()
{
    this.textBoxSurname.ReadOnly = true;
    this.textBoxName.ReadOnly = true;
    this.textBoxPatronymic.ReadOnly = true;
    this.textBoxNetName.ReadOnly = true;
    this.comboBoxJobRole.Enabled = false;
    this.comboBoxStatus.Enabled = false;
    this.comboBoxAccess.Enabled = false;
}

```

Аналогичным образом сформируем метод DisplayEdit, который задает режим редактирования формы:

```

/// Задание режима редактирования
public void DisplayEdit()
{

```

```
this.textBoxSurname.ReadOnly = false;  
this.textBoxName.ReadOnly = false;  
this.textBoxPatronymic.ReadOnly = false;  
this.textBoxNetName.ReadOnly = false;  
this.comboBoxJobRole.Enabled = true;  
this.comboBoxStatus.Enabled = true;  
this.comboBoxAccess.Enabled = true;  
}
```

Для управления режимом доступности (только для чтения/редактирование) формы FormEmployee необходимо метод DisplayReadOnly вызывать при первоначальной загрузке формы (событие Load), при создании новых данных по сотруднику и при редактировании данных по сотруднику, а метод DisplayEdit - при сохранении данных по сотруднику и при отмене режима редактирования данных.

Проверьте правильность режима управления доступностью элементов управления формы FormEmployee.

Анализ кодов методов DisplayReadOnly() и DisplayEdit() показывает, что они могут быть объединены в один метод с параметром. Необходимо самостоятельно написать объединенный метод, получив в результате метод DisplayReadOnly(bool readOnly), в котором параметр readOnly определяет режим редактирования: если readOnly равен true, то режим только для просмотра, если равен false, то - редактирование.

В процессе работы приложения необходимо управлять доступом к пунктам меню в соответствии с диаграммой состояний для пунктов меню формы FormEmployee, приведенной на рисунке 7.10.

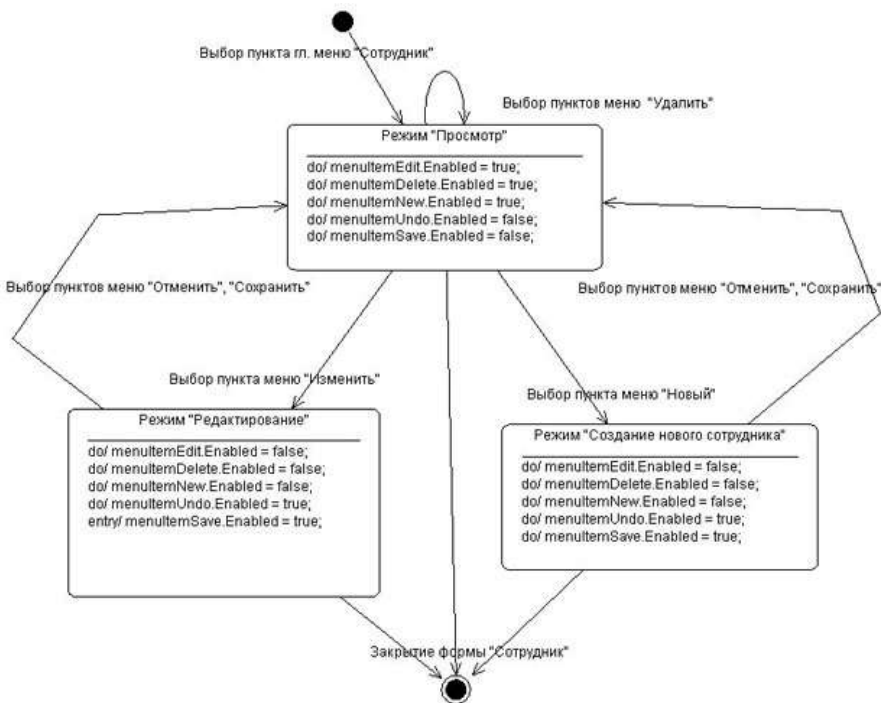


Рис. 7.10. Диаграмма состояний для активности подпунктов меню "Действие"

Диаграмма отображает возможные переходы между тремя режимами: "Просмотр", "Редактирование" и "Создание нового сотрудника".

При выборе в главном меню приложения пункта "Сотрудник" Windows-форма FormEmployee должна перейти в режим "Просмотр", что определяет доступ к пунктам меню "Создать", "Редактировать", "Удалить" и запрет доступа к подпунктам меню "Отменить", "Сохранить".

Если в режиме просмотр выбирается подпункт меню "Удалить", то в результате выполнения данной функции режим Windows-формы FormEmployee не должен измениться, т.е. форма должна остаться в режиме "Просмотр".

Если в режиме просмотр выбирается подпункт меню "Изменить", то Windows-формы FormEmployee должна перейти в режим "Редактирование". Данный режим предполагает, что разрешается доступ к подпунктам меню "Отменить", "Сохранить" и запрещается доступа к подпунктам меню "Создать", "Редактировать", "Удалить".

Аналогичным образом интерпретируются переходы формы FormEmployee из одного режима в другой.

На рисунке 7.10 представлены режимы и переходы для подпунк-

тов главного меню. Аналогичные режимы необходимо соблюдать для контекстного меню и кнопок панели инструментов.

Для управления доступом к пунктам главного меню создайте методы `MenuItemEnabled(bool itemEnabled)`, для контекстного меню - `MenuItemContextEnabled(bool itemEnabled)` и для кнопок панели управления - `StripButtonEnabled(bool itemEnabled)`. Управление доступностью пунктов главного и контекстного меню осуществляется через свойство `Enabled` класса `ToolStripMenuItem`, а кнопок панели управления - через свойство `Enabled` класса `ToolStripButton`.

Проверьте правильность режима управления пунктов главного и контекстного меню, а также кнопок панели управления формы `FormEmployee`.

С учетом того, что установка режимов просмотра и редактирования экранной формы, а также управление доступом к пунктам меню должно выполняться при реализации нескольких функций программы целесообразно для избежания дублирования кода все методы управления режимами объединить в один метод `DisplayForm`.

```
private void DisplayForm(bool mode)
```

```
{  
    DisplayReadOnly(mode);  
    MenuItemEnabled(mode);  
    MenuItemContextEnabled(mode);  
    StripButtonEnabled(mode);  
}
```

Первоначальная установка режима "*Просмотр*" должна проводиться при первоначальной загрузке формы `FormEmployee`.

#### **Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

#### **Форма представления результата:**

- Программный код, составленный на языке программирования

#### **Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**  
**Практическое занятие №20.** Создание строки состояния в среде C#

**Цель работы:** Изучить основные способы построения строки состояния и получить практические навыки в разработке

**Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

**Задание**

1. Изучить теоретический материал.
2. Создать строку состояний для главной и дочерней форм.
3. Написать обработчики для формирования строки состояний, отображающих информацию о пунктах меню.
4. Протестировать работу приложения.

**Основные сведения**

***Создание строки состояния***

На многих формах в реальных приложениях имеется элемент интерфейса, называемый строкой состояния (StatusStrip). Обычно в строке состояния выводится некоторая текстовая или графическая информация, относящаяся к работе приложения. Строка состояния может быть разделена на несколько "панелей" (panel) - отдельных частей окна. В каждой из этих панелей информация выводится отдельно.

Создадим строку состояния, в которой будут выводиться текстовые сообщения, относящиеся к пунктам меню.

В окне Toolbox выделим пункт StatusStrip и перетащим его на форму (рисунок 6.1).



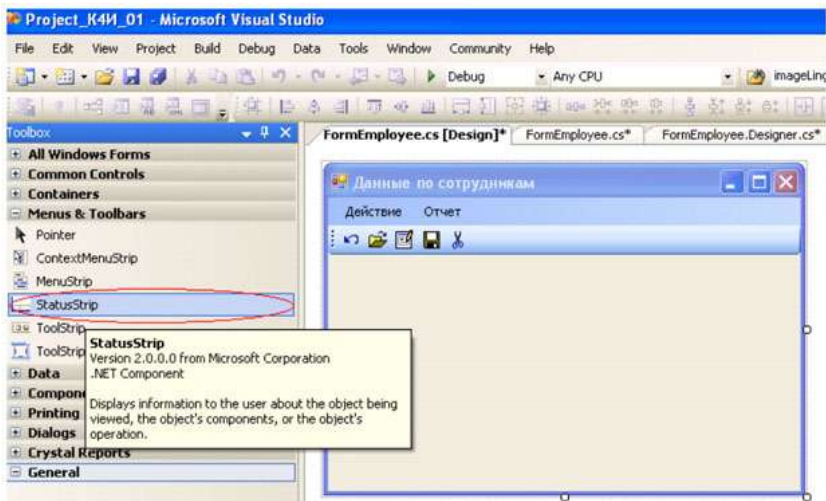


Рис. 6.1. Добавляем на форму строку состояния  
Объекту класса StatusStrip присвоим имя statusStripEmployee.  
Откроем выпадающий список объекта класса statusBarEmployee и выберем объект StatusLabel (рисунок 6.2). Присвоим ему имя toolStripStatusLabelEmployee.

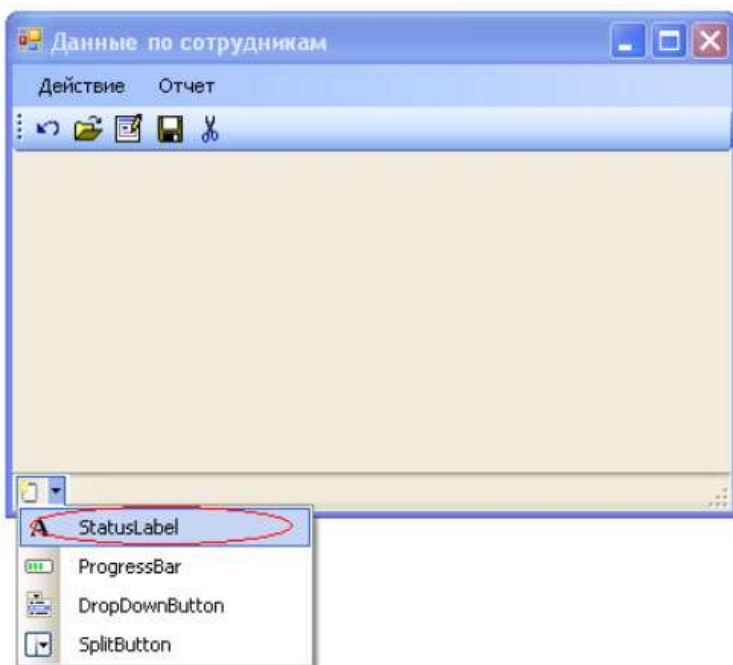


Рис. 6.2. Добавляем метку в строку состояния

При компиляции, запуске приложения и выборе пункта меню "Сотрудник" экранная форма будет иметь вид, представленный на рисунке 6.3.

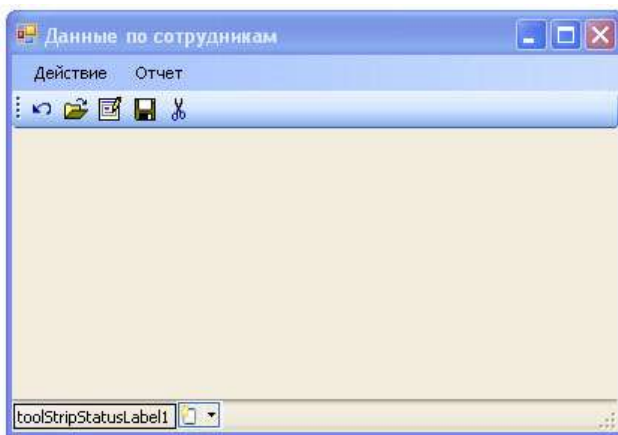


Рис. 6.3. Окно приложения со строкой состояния

Для управления текстом строки состояния необходимо разрабо-

тать обработчик события для соответствующих объектов.

Для формы FormEmployee в строке состояний необходимо вывести информацию при наведении курсора мыши на пунктах меню "Действие". Первоначально в дизайнера формы необходимо выделить пункт меню "Действие", перейти на вкладку Properties и открыть окно событий,



нажав кнопку. На данной вкладке необходимо выделить событие MouseEnter и в поле ввода сделать двойной щелчок. (рисунок 6.4).

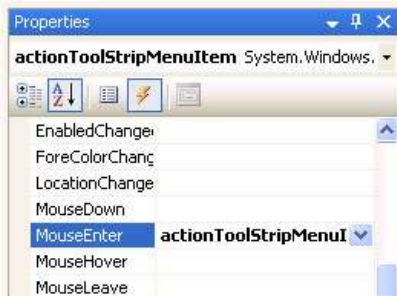


Рис. 6.4. Окно событий

Система сгенерирует код обработчика, приведенного ниже.

```
private void actionToolStripMenuItem_MouseEnter(object sender, EventArgs e)
{ }
```

Добавим в обработчик следующий код:

```
private void actionToolStripMenuItem_MouseEnter(object sender, EventArgs e)
{
    toolStripStatusLabelEmployee.Text =
        "Выбор действий по сотрудникам";
}
```

Если откомпилировать программу, запустить её, выбрать пункт меню "Сотрудник" и навести указатель мыши на пункт "Действие", то сгенерируется событие "MouseEnter" и в строке состояния выведется текстовое сообщение "Выбор действий по сотрудникам" (рисунок 6.5).

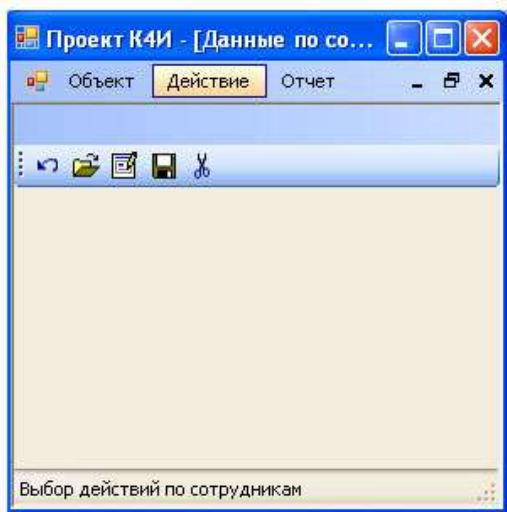


Рис. 6.5. Вывод сообщения в строке состояния

Если теперь переместить указатель мыши с пункта меню "Действие", то текст в строке состояния не изменится. Такой режим работы программы является неправильным, так как если указатель мыши перемещается с пункта меню "Действие", то строка состояния должна становиться пустой. Для обеспечения правильной работы программы воспользуемся ещё одним событием "MouseLeave", которое генерируется, когда мышь перемещается (покидает) с пункта меню "Действие". Обработчик данного события имеет следующий вид:

```
private void actionToolStripMenuItem_MouseLeave(object sender, EventArgs e)
{
    toolStripStatusLabelEmployee.Text = "";
}
```

Вышеприведенные обработчики будут вызываться только тогда, когда пользователь наведет указатель мыши на пункт меню "Действие". Для того чтобы обработчики реагировали на все строки пунктов главного меню "Действие" и "Отчет" формы FormEmployee необходимо сформировать соответствующие события MouseEnter и MouseLeave для всех подпунктов меню и создать для них обработчики.

Результаты компиляции и выполнения приложения приведены на рисунке 6.6.

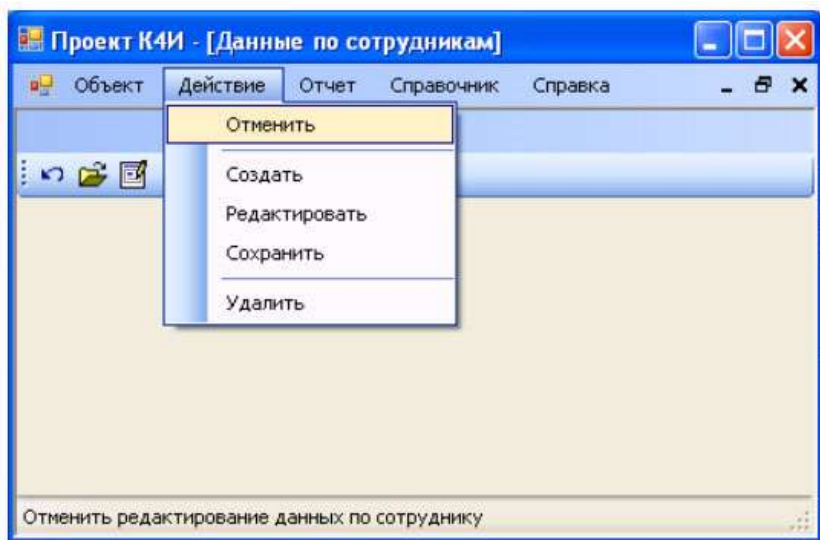


Рис. 6.6. Экранная форма со строкой состояния

**Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

**Форма представления результата:**

- Программный код, составленный на языке программирования

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

**Практическое занятие №21, 22, 23, 24.** Создание приложения базы данных визуальными средствами IDE. Создание проекта. Подключение файла данных к проекту

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

**Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

**Задание**

1. Изучите теоретический материал.
2. Создайте класс *DataSetEmployee*.
3. Для разрабатываемого приложения создайте объекты *dsEmployee*, *daJobTitle* и *daEmployee*.
4. Проведите компиляцию проекта и убедитесь в отсутствии ошибок трансляции.
5. Разработайте метод *Fill* для заполнения таблиц *DataSet*.
6. Протестировать работу приложения.

**Общие сведения**

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 базы данных, 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, решая 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).

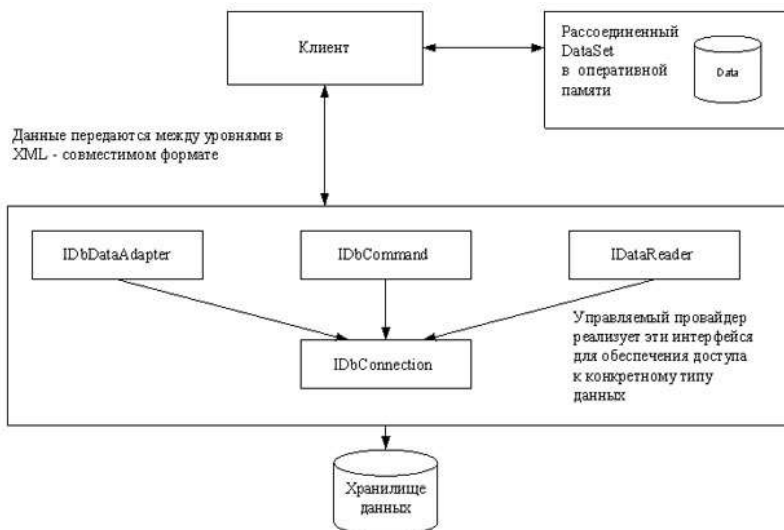


Рис. 8.1. Взаимодействие клиента с управляемыми провайдерами

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.SqlClient** 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, DataRow	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, DataRelation, DataTableCollection, DataTable	Тип DataRelationCollection представляет набор всех 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.

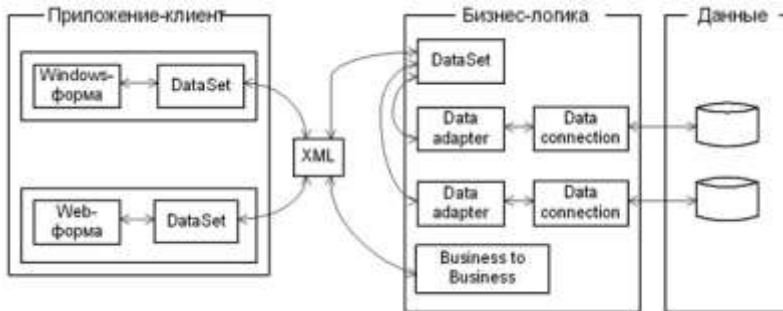


Рис. 8.2. Модель доступа к данным в ADO.NET

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 (Интранет), 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 содержится 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 параметризованные 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.

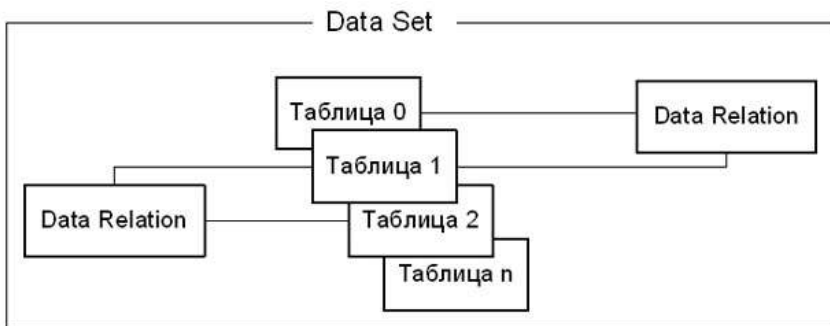


Рис. 8.3. Структура объекта DataSet

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.

Информация о базе данных

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

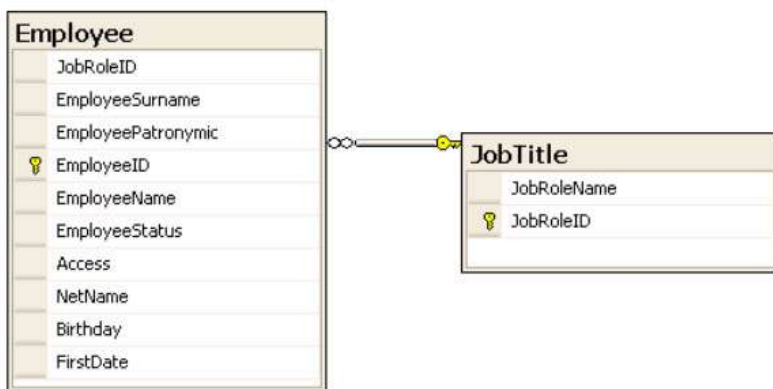


Рис. 8.4. Структура базы данных по сотрудникам компании  
База данных включает две таблицы:

- 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)

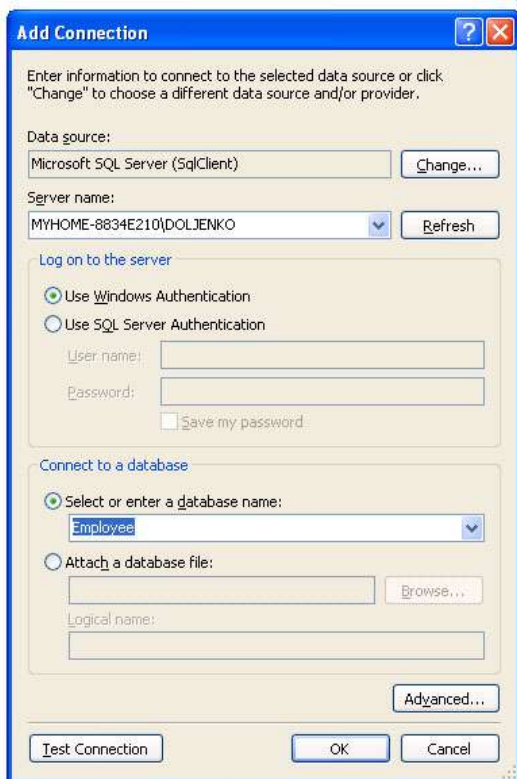


Рис. 8.5. Окно AddConnection

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 появится следующее сообщение (рисунок 8.6).



Рис. 8.6. Окно Microsoft Data Link

При нормальном соединении с базой данных можно открыть на-



вигатор *Server Explorer* из меню *View/ Server Explorer* или сочетанием клавиш *ALT+CTRL+S* (рисунок 8.7).

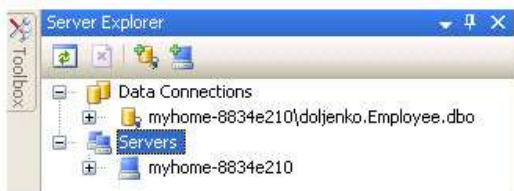


Рис. 8.7. Окно навигатора *Server Explorer*

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

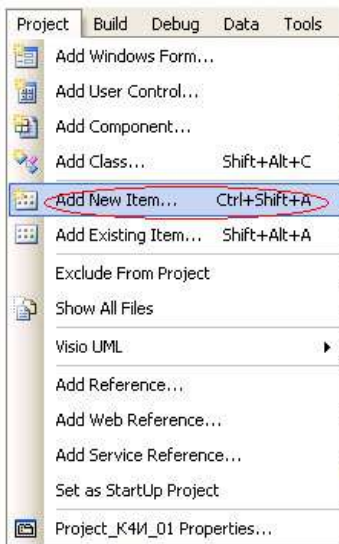


Рис. 8.8. Добавление в проект нового компонента

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

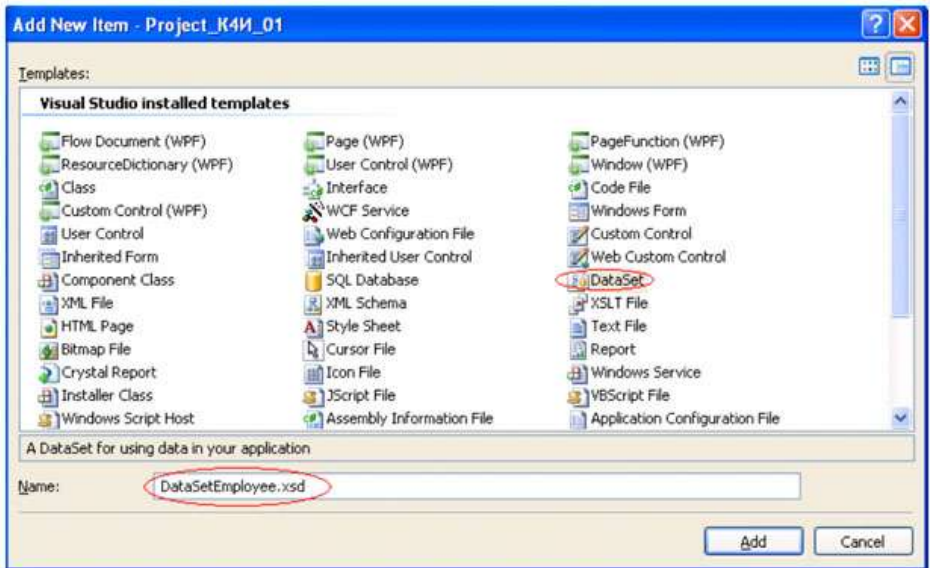


Рис. 8.9. Выбор нового компонента - DataSet

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

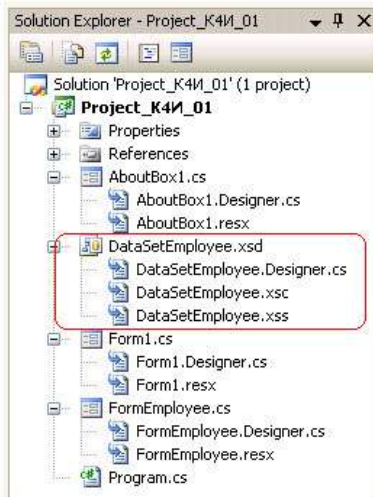


Рис. 8.10. Окно решения проекта с новым компонентом DataSet

Для добавления таблиц *Employee* и *JobTitle* к DataSet необходимо перетащить их из окна *Server Explorer* на поле графического дизайнера

(рисунок 8.11).

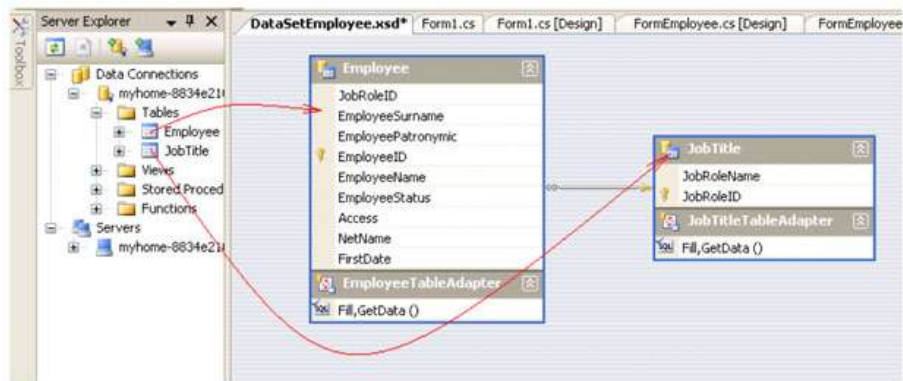


Рис. 8.11. Добавление таблиц к DataSet

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.

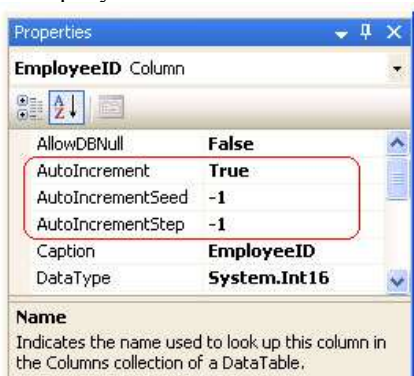


Рис. 8.12. Установка свойств для колонки `EmployeeID`

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_K4И_01.DataSetEmployeeTableAdapters.  
EmployeeTableAdapter();  
DataSetEmployeeTableAdapters.JobTitleTableAdapter daJobTitle =  
    new Project_K4И_01.DataSetEmployeeTableAdapters.  
JobTitleTableAdapter();
```

После того, как созданы объекты адаптеров данных `daEmployee` и `daJobTitle`, а также объект класса `DataSetEmployee` - `dsEmployee` необходимо создать метод для заполнения объекта `dsEmployee` из базы данных (в рассматриваемом примере база данных `Employee`, созданная в СУБД MS SQL Server 2005). 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 обработан");}
```

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".

**Порядок выполнения работы:**

- Написание программного кода на языке программирования
- Отладка программы на ПК

**Форма представления результата:**

- Программный код, составленный на языке программирования

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

**Тема 03.02.03 Структура и приемы работы с инструментальными средствами, поддерживающими создание ПО**

**Практическое занятие №25, 26, 27.** Отображение данных на экранной форме

**Цель работы:** Изучить основные приемы и способы отображения и связывания данных объекта DataSet и элементов управления *Windows* - формы.

**Выполнив работу, Вы будете:**

*уметь:*

- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ
- Учебная аудитория, оснащенная мультимедийным оборудованием

- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- Среда разработки Visual Studio, язык программирования C#

### **Задание**

1. Изучите теоретический материал.
2. Осуществите привязку источника данных к элементам управления экранной формы.
3. Разработайте необходимые методы для вывода информации из базы данных на экранную форму.
4. Протестируйте приложение

### **Основные сведения**

Для отображения информации на элементах управления Windows Forms необходимо осуществить их заполнение из соответствующих столбцов и строк таблиц DataSet. Это можно сделать путем написания специального кода, который предназначен для заполнения соответствующих элементов управления или использовать механизм привязки данных к пользовательским интерфейсам .NET.

Windows Forms позволяют отображать данные путем привязки их элементов управления к источникам данных. Привязка данных обычно используется для отображения результатов поиска, детализации сводок, генерации отчетов и ввода данных. Существует два типа привязки данных: простая и сложная. В случае простой привязки данных элемент управления привязывается к отдельному элементу данных. Простая привязка, в основном, используется с такими элементами управления, как поле ввода (TextBox) и надпись (Label). При сложной привязке данных элемент управления привязывается более чем к одному элементу данных - обычно к одному или нескольким столбцам в нескольких строках результирующего набора строк. К элементам управления, поддерживающим сложную привязку данных, относятся список (ListBox), выпадающий список (ComboBox), список данных (DataList) и табличный элемент (DataGrid).

В технологии привязки элементов управления Windows Forms используется абстрактный класс BindingManagerBase. Данный класс синхронизирует все элементы управления Windows Forms (т. е. объекты Binding), привязанные к одному источнику данных, позволяя отображать информацию об объекте в источнике данных - например, о строке в локальной таблице.

Класс BindingContext используется для создания экземпляра объекта BindingManagerBase, и возвращается при этом объект

CurrencyManager или PropertyManager в зависимости от типа источника данных:

Класс CurrencyManager является потомком класса BindingManagerBase и поддерживает указатель на текущий элемент в источнике данных, который реализует интерфейсы IList, IListSource или IBindingList. Источники данных не обязательно поддерживают указатель на текущий элемент. CurrencyManager уведомляет все привязанные элементы управления об изменении текущего элемента, чтобы они могли обновить свои данные. Класс CurrencyManager поддерживает текущее свойство объекта, а не сам объект в списке.

Свойство Position - это индекс с отсчетом от нуля, позволяющий узнать или задать текущую позицию в источнике данных. Свойство Count возвращает количество элементов данных в списке. Свойство Current возвращает текущий объект в списке, который необходимо привести к типу объекта в соответствующем источнике данных, прежде чем с ним можно будет работать.

При связывании элемента контроля ListBox устанавливается множественная связь с источником данных. Для связи используются свойства списка DataSource и DisplayMember. Свяжем элемент контроля listBoxEmployee, в котором должен отображаться список фамилий сотрудников, со столбцом EmployeeSurname таблицы Employee. Это можно сделать, добавив следующие строки кода в метод загрузки формы FormEmployee\_Load.

```
this.listBoxEmployee.DataSource = this.dsEmployee;
```

```
this.listBoxEmployee.DisplayMember =
```

```
"Employee.EmployeeSurname";
```

Протестируйте добавленный в программу код.

В разрабатываемом приложении на Windows -форме FormEmployee имеются четыре текстовых поля: textBoxSurname, textBoxName, textBoxPatronymic и textBoxNetName. Эти текстовые поля предназначены для отображения информации из одной записи таблицы Employee набора данных dsEmployee. Для того чтобы содержимое текстовых полей автоматически обновлялось при смене записи, их необходимо связать с соответствующими колонками набора данных dsEmployee. Связывание можно осуществить, используя свойство DataBindings элемента управления TextBox. Например, для элемента управления textBoxSurname можно осуществить связь с источником данных, добавив следующую строку кода в метод загрузки формы FormEmployee\_Load.

```
textBoxSurname.DataBindings.Add("Text", dsEmployee,
```

```
"Employee.EmployeeSurname");
```

Протестируйте добавленный в программу код.

Аналогично свяжите текстовые поля textBoxName, textBoxPatronymic и textBoxNetName с источником данных.

Протестируйте добавленный в программу код.

В таблице Employee значения для атрибута Access (доступ) задается в виде символической строки, значение которой выбирается из списка элемента управления comboBoxAccess. Коллекцию выпадающего списка элемента управления comboBoxAccess можно задать следующей строкой кода:

```
this.comboBoxAccess.Items.AddRange(new object[] { "не задан",  
"администратор",  
"начальник смены", "старший оператор", "оператор",  
"аналитик"});
```

Для заданной записи источника данных (таблица Employee) значение столбца Access необходимо отобразить в элементе контроля comboBoxAccess. Это можно сделать аналогично тому, как это делалось для элементов управления TextBox, задавая свойство DataBindings

Протестируйте добавленный в программу код.

Для элемента управления comboBoxStatus необходимо сформировать коллекцию выпадающего списка: не задан, активен, выходной, в отпуске, болеет, не работает, помечен как удаленный.

В таблице Employee значения для атрибута EmployeeStatus (статус) задается в виде целого числа (0, 1, 2, 3, 4, 6), однако статус сотрудника должен отображаться в элементе управления comboBoxStatus в виде строковых значений в соответствии со значениями его коллекции. В программе необходимо реализовать отображение целочисленных данных из DataSet в текстовые значения в элементе контроля comboBoxStatus. Для этого необходимо отслеживать изменение позиции в таблице источника данных dsEmployee и в соответствии со значением столбца EmployeeStatus активизировать требуемый элемент (Item) списка comboBoxStatus.

Объявим объект bmEmployee класса BindingManagerBase в форме FormEmployee:

```
BindingManagerBase bmEmployee;
```

В конструкторе класса FormEmployee, создадим объект bmEmployee применяя индекатор контента BindingContext включив в него связывание с таблицей Employee и добавим делегат для события, которое формируется при изменении позиции в данной таблице:

```
public FormEmployee()  
{  
    InitializeComponent();  
    bmEmployee = this.BindingContext[dsEmployee, "Employee"];  
    // Добавляем делегата PositionChanged для события - изменение  
    // позиции в таблице Employee DataSet dsEmployee  
    bmEmployee.PositionChanged += new  
    EventHandler(BindingManagerBase_PositionChanged);  
}
```



Кроме того, необходимо создать обработчик для сформированного события, который на основе выбранной строки (pos) таблицы *Employee* будет задавать свойству *Text* списка *comboBoxStatus* значение из коллекции *Items* по индексу (sel), полученному из столбца *EmployeeStatus Employee*.

```
private void BindingManagerBase_PositionChanged(object sender, EventArgs e)
{
    int pos = ((BindingManagerBase)sender).Position;
    int sel = (int)dsEmployee.Employee[pos].EmployeeStatus;
    this.comboBoxStatus.Text =
this.comboBoxStatus.Items[sel].ToString();
}
```

Протестируйте добавленный в программу код.

Для задания в элементе контроля *comboBoxJobRole* должности сотрудника необходимо получить данные из родительской таблицы *JobTitle*, с которой таблица *Employee* связана внешним ключом *JobRoleID*. Фактически необходимо осуществить вывод данных из справочника (таблица *JobTitle*) по данным в основной таблице *Employee*.

В *Windows Forms* элемент управления *ComboBox* имеет три свойства, управляющих привязкой к источникам данных. Эти свойства описаны в таблице 9.1.

Таблица 9.1. Свойства элемента управления *ComboBox*, управляющие привязкой к источникам данных

Свойство	Описание
Data Source	Позволяет узнать или задать источник данных для элемента управления. Это может быть объект <i>Data Table</i> , <i>DataView</i> или вообще любой объект, реализующий интерфейс <i>IList</i>
DisplayMember	Позволяет узнать или задать свойство источника данных, которое отображается в элементе управления. В случае <i>DataTable</i> и <i>DataView</i> это имя столбца
ValueMember	Позволяет узнать или задать свойство источника данных, которое предоставляет значения для элемента управления. В случае <i>DataTable</i> и <i>DataView</i> это имя столбца. По умолчанию задана пустая строка

Свойство *DataBindings* объекта *ComboBox* предоставляет доступ к коллекции *ControlBindingsCollection*. Метод *Add* этой коллекции добавляет в неё привязку. Перегруженный вариант метода *Add* принимает три аргумента:

PropertyName - имя свойства элемента управления, к которому осуществляется привязка;

DataSource - имя привязываемого источника данных;

DataMember - имя свойства привязываемого источника данных.

Вначале связываем элемент контроля comboBoxJobRole с набором данных JobTitle (родительская таблица - справочник), в соответствии с кодом, приведенным ниже.

```
comboBoxJobRole.DataSource = this.dsEmployee.JobTitle;
```

```
comboBoxJobRole.DisplayMember = "JobRoleName";
```

```
comboBoxJobRole.ValueMember = "JobRoleID";
```

После этого необходимо связать comboBoxJobRole с полем JobRoleID основной таблицы Employee (дочерняя таблица), в соответствии со следующим кодом

```
comboBoxJobRole.DataBindings.Add("SelectedValue",
```

```
dsEmployee, "Employee.JobRoleID");
```

После компиляции и запуска приложения экранная форма будет иметь вид, приведенный на рисунке 9.1.

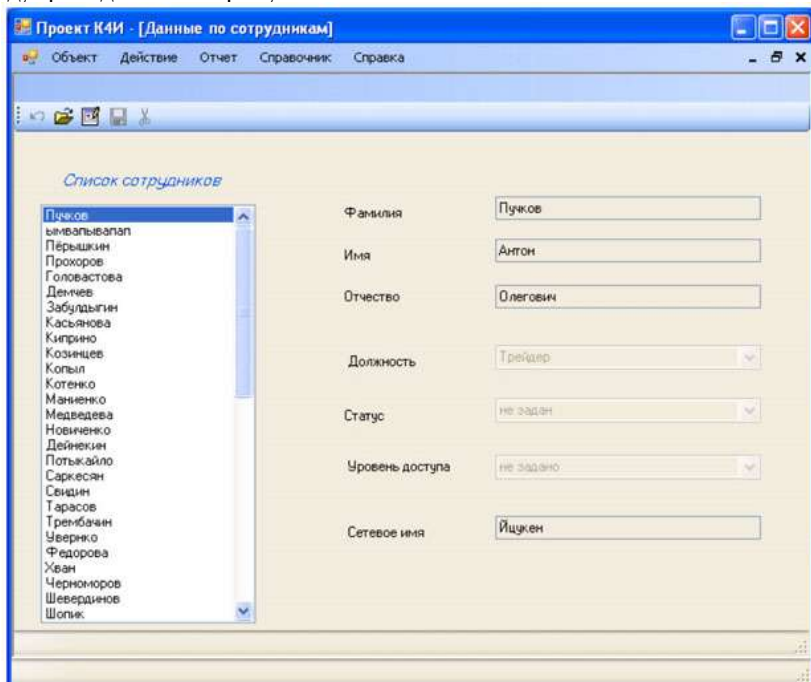


Рис. 9.1. Экранная форма в режиме просмотра (только для чтения)

## Порядок выполнения работы:

- Написание программного кода на языке программирования
- Отладка программы на ПК

**Форма представления результата:**

- Программный код, составленный на языке программирования

**Критерии оценки:**

Работа выполнена полностью и не содержит ошибок, студент грамотно представил отчет – оценка «отлично».

Работа выполнена полностью, но содержит не более двух ошибок, студент грамотно представил отчет – оценка «хорошо».

Работа выполнена с ошибками, студент представил краткий отчет – оценка «удовлетворительно».

Работа выполнена с грубыми ошибками, отчет составлен неграмотно – оценка «неудовлетворительно».

## МДК.03.03. Документирование и сертификация

### Тема 03.03.01 Составление программной документации.

#### Практическая работа № 1 Оформление программной документации.

**Цель работы:** ознакомление с основными государственными стандартами по разработке ПО, ознакомление с составом ЕСПД, ознакомление с содержанием основных программных документов и этапами их разработки.

#### Выполнив работу, Вы будете:

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

#### Материальное обеспечение:

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC

#### Задание

- Ознакомиться с системой национальных стандартов ЕСПД
- Ознакомиться с видами программных документов и основными правилами оформления программной документации
- Ознакомиться с содержанием и этапами разработки основных программных документов.
- Заполнить сводную таблицу по стадиям разработки программных документов.

#### Порядок выполнения работы:

- Изучите теоретический материал по теме.
- На основе изученного материала составьте сводную таблицу:

Стадии разработки	Этапы работ	Содержание работ
Техническое задание	а б ....	...

### **Форма представления результата:**

- Опрос по выполненному заданию.
- Составленная таблица.

### **Критерии оценки:**

Таблица заполнена полностью без ошибок, студент правильно отвечает на вопросы по заданию – оценка «отлично».

Таблица заполнена не до конца или содержит незначительные ошибки, студент в целом правильно отвечает на вопросы по заданию – оценка «хорошо».

Таблица заполнена частично или содержит множество ошибок, студент при ответе на вопросы допускает много ошибок – оценка «удовлетворительно».

Таблица не заполнена, студент не может ответить на большинство вопросов – оценка «неудовлетворительно».

## **Практическая работа № 2 Оформление программной документации.**

**Цель работы:** ознакомление с основными государственными стандартами по разработке ПО, ознакомление со структурой и содержанием ТЗ на ПО.

### **Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки документации программного обеспечения;

### **Материальное обеспечение:**

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC

### **Задание**

- Ознакомиться с теоретическим материалом
- Ознакомиться со стандартами для написания ТЗ на ПО: ГОСТ 19.106-78 и ГОСТ 34.602
- Разработать ТЗ на ПО по индивидуальному варианту.

### **Основные сведения:**

Техническое задание (ТЗ, техзадание) — исходный документ для проектирования сооружения или промышленного комплекса, конструирования технического устройства (прибора, машины, системы управления и т. д.), **разработки информационных систем**, стандартов либо проведения научно-исследовательских работ (НИР).

ТЗ содержит технико-экономическое обоснование разработки, основные технические требования, предъявляемые к сооружению или изделию, и исходные данные для разработки; в ТЗ указываются назначение объекта, область его применения, стадии разработки конструкторской (проектной, технологической, программной и т.п.) документации, ее состав, сроки исполнения и т. д., а также особые требования, обусловленные спецификой самого объекта либо условиями его эксплуатации.

Как правило, ТЗ составляют на основе анализа результатов предварительных исследований, расчетов и моделирования.

В связке заказчик-исполнитель, ТЗ позволяет:

- обеим сторонам о представить готовый продукт
- выполнить пунктную проверку готового продукта
- уменьшить число ошибок, связанных с изменением требований в результате их неполноты или ошибочности (на всех стадиях и этапах создания, за исключением испытаний)

- заказчику о осознать, что именно ему нужно
- требовать от исполнителя соответствия продукта всем условиям, оговоренным в ТЗ

- исполнителю о понять суть задачи, показать заказчику «технический облик» будущего изделия, программного изделия или автоматизированной системы о спланировать выполнение проекта и работать по намеченному плану о отказаться от выполнения работ, не указанных в ТЗ

### **Порядок выполнения работы:**

- Изучите теоретический материал по теме.
- На основе изученного материала по требованиям соответствующего стандарта составьте ТЗ на ПО по своему варианту.

### **Форма представления результата:**

- Разработанное по индивидуальному варианту ТЗ на ПО.

### **Критерии оценки:**

- Разработанный документ, соответствует указанным стандартам, составлен и оформлен согласно требованиям на ТЗ, содержание документа соответствует поставленной задаче – оценка «отлично».
- Разработанный документ, не совсем точно соответствует указанным стандартам, составлен и оформлен согласно требованиям на ТЗ, содержание документа соответствует поставленной задаче – оценка

- «хорошо».
- Разработанный документ, не соответствует указанным стандартам, составлен и оформлен не в соответствии с требованиями на ТЗ, содержание документа не совсем соответствует поставленной задаче – оценка «удовлетворительно».
  - Документ не разработан или содержит большое количество ошибок – оценка «неудовлетворительно».

### **Тема 03.03.02 Документирование программных средств** **Практическая работа № 3 Описание пользовательского интерфейса программы**

**Цель работы:** ознакомление со стандартами создания и оформления пользовательской документации ПО.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC

**Задание**

- Изучить виды пользовательской документации, стандарты создания и оформления пользовательской документации.

**Порядок выполнения работы:**

- Изучите теоретический материал и нормативные документы Российской системы стандартизации ГОСТ Р ИСО 9241-161-2016 и ГОСТ Р ИСО 9241-151-2014
- Руководствуясь положениями стандарта составить описание пользовательского интерфейса программы по индивидуальному заданию.

**Форма представления результата:**

- Разработанное описание пользовательского интерфейса.

**Критерии оценки:**

- Разработанный документ, соответствует указанным стандартам, составлен и оформлен согласно требованиям на пользовательскую документацию ПО, содержание документа соответствует поставленной задаче – оценка «отлично».
- Разработанный документ, не совсем точно соответствует указанным стандартам, составлен и оформлен согласно требованиям на пользовательскую документацию ПО, содержание документа соответствует поставленной задаче – оценка «хорошо».
- Разработанный документ, не соответствует указанным стандартам, составлен и оформлен не в соответствии с требованиями на пользовательскую документацию ПО, содержание документа не совсем соответствует поставленной задаче – оценка «удовлетворительно».
- Документ не разработан или содержит большое количество ошибок. – оценка «неудовлетворительно».

**Тема 03.03.03 Схемы алгоритмов, данных и систем****Практическая работа № 4,5 Правила применения символов и выполнения схем**

**Цель работы:** применять стандарты при создании сопроводительной документации ПО, осуществлять контроль за соблюдением требований технических регламентов.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

**Материальное обеспечение:**

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC



### **Задание**

-Ознакомиться с содержанием стандартов на оформление и составление программной документации: ГОСТ 19.701-90

- Проанализировать структуру стандартов разных видов на соответствие требованиям ГОСТ 19.701-90

- Составить по индивидуальному заданию схемы: данных, программ и взаимодействия программ.

### **Порядок выполнения работы:**

-Ознакомьтесь с содержанием стандартов на оформление и составление программной документации: ГОСТ 19.701-90

- Проанализируйте структуру стандартов заполнения отчетов по практике, курсовых проектов и выпускных квалификационных работ на соответствие требованиям ГОСТ 19.701-90

- Составьте по индивидуальному заданию схемы: данных, программ и взаимодействия программ.

### **Форма представления результата:**

Выполненные схемы, ответы на вопросы по изученному материалу.

### **Критерии оценки:**

- Схемы разработаны и оформлены в соответствии с указанными стандартами и поставленной задачей, даны правильные ответы на поставленные вопросы – оценка «отлично».
- Схемы разработаны и оформлены в соответствии с указанными стандартами и поставленной задачей с небольшими неточностями, в целом, даны правильные ответы на поставленные вопросы – оценка «хорошо».
- Схемы разработаны и оформлены без учета стандартов и не точно соответствуют поставленной задаче, в ответах на поставленные вопросы допущены ошибки – оценка «удовлетворительно».
- Схемы содержат большое количество ошибок – оценка «неудовлетворительно».

## **Тема 03.03.04.Сертификация программного обеспечения**

### **Практическая работа № 6 Составление заявки на проведение сертификационных работ**

**Цель работы:** ознакомление с процедурой разработки и оформления документов сертификации программного продукта.

**Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

#### **Материальное обеспечение:**

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC

#### **Задание**

- Изучение порядка проведения сертификации средств информатизации и правил заполнения бланков сертификата

#### **Порядок выполнения работы:**

- Изучите теоретический материал и нормативные документы Российской системы сертификации.
- Выпишите основные требования и положения по проведению сертификации с указанием юридического обоснования (приказ, закон, перечень и т.д.).
- Оформите документ по сертификации программного продукта.

#### **Форма представления результата:**

- Документ с перечнем требований и правил проведения сертификации ПС.
- оформленная заявка на проведение сертификации продукции в Системе добровольной сертификации

#### **Критерии оценки:**

- Составленный перечень требований полный, заявка соответствует стандартам оформления, заполнена правильно – оценка «отлично».
- Составленный перечень требований полный, заявка не совсем соответствует стандартам оформления, заполнена правильно – оценка «хорошо».
- Составленный перечень требований не полный, заявка не совсем соответствует стандартам оформления, заполнена правильно – оценка «удовлетворительно».
- Перечень составлен не точно, заявка не соответствует стандартам заполнения и форматирования – оценка «неудовлетворительно».

### **Тема 03.03.05. Государственная регистрация программы для ЭВМ и базы данных**

#### **Практическая работа № 7, 8, 9 Подготовка пакета документов для государственной регистрации программы для ЭВМ**

**Цель работы:** ознакомление с процедурой разработки и оформления пакета документов для государственной регистрации программы для ЭВМ и базы данных

#### **Выполнив работу, Вы будете:**

*уметь:*

- владеть основными методологиями процессов разработки программного обеспечения;
- использовать методы для получения кода с заданной функциональностью и степенью качества;

#### **Материальное обеспечение:**

- ПЭВМ.
- Компьютерный класс.
- Локальная сеть, сетевое программное обеспечение.
- Пакет прикладных программ «Microsoft Office»: текстовый процессор Microsoft Word, браузер Microsoft Internet Explorer.
- система электронного документооборота с открытым кодом NauDOC

#### **Задание**

- Изучение порядка проведения государственной регистрации программы для ЭВМ и базы данных

#### **Порядок выполнения работы:**

- Изучите теоретический материал и нормативные документы Российской системы сертификации: ГК РФ Статья 1262. Государственная регистрация программ для ЭВМ и баз данных
- Изучите правила оформления заявки на государственную регистрацию программы для электронных вычислительных машин или базы данных.
- Изучите правила составления документов, являющихся основанием для осуществления юридически значимых действий по государственной регистрации программы для электронных вычислительных машин или базы данных, и их формы

- Оформите заявку на Государственную регистрацию программ для ЭВМ и баз данных
- Определите и составьте документ об отказе в государственной регистрации с указанием перечня причин такого отказа в государственной регистрации.

**Форма представления результата:**

- Оформленные документы для проведения государственной регистрации программы для ЭВМ и базы данных, документ об отказе в регистрации.
- Ответы на вопросы по изученному материалу

**Критерии оценки:**

- Заявка составлена и оформлена согласно нормативным документам, составлен полный перечень причин отказа в регистрации, сформулированы правильные ответы на вопросы – оценка «отлично».
- Заявка составлена и оформлена согласно нормативным документам, с небольшими неточностями, составлен полный перечень причин отказа в регистрации, в целом, сформулированы правильные ответы на вопросы – оценка «хорошо».
- Заявка составлена и оформлена со значительными отступлениями от нормативных документов, в перечне причин отказа есть ошибки или неточности, в ответах на вопросы много ошибок – оценка «удовлетворительно».
- Заявка составлена и оформлена с большим количеством ошибок, в перечне причин отказа много ошибок, в ответах на вопросы много ошибок – оценка «неудовлетворительно».