МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
"Казанский (Приволжский) федеральный университет"
Елабужский институт (филиал) КФУ

УТВЕРЖДАЮ
Директор
Елабужского института КФУ
_____Е.Е. Мерзон
"22" 05 2024 г.

Программа дисциплины (модуля)
Разработка программных приложений

Направление подготовки/специальность: 15.03.06 Мехатроника и робототехника
Направленность (профиль) подготовки: Физические основы мехатроники и робототехники
Квалификация выпускника: бакалавр
Форма обучения: очно-заочная
Язык обучения: русский
Год начала обучения по образовательной программе: 2024

Содержание

Программу дисциплины разработал(а)(и) старший преподаватель Галиев Н.М. (Кафедра математики и прикладной информатики)

1. Перечень планируемых результатов обучения по дисциплине (модулю), соотнесенных с планируемыми результатами освоения ОПОП ВО

Обучающийся, освоивший дисциплину (модуль), должен обладать следующими компетенциями:

Шифр компетенции	Расшифровка приобретаемой компетенции
ПК-2	Способен разрабатывать, отлаживать, внедрять и сопровождать программное обеспечение мехатронных и робототехнических систем
ПК-2.1	Знать способы разработки, отладки и сопровождения программного обеспечения для мехатронных и робототехнических систем
ПК-2.2	Уметь разрабатывать, отлаживать и сопровождать программное обеспечение для мехатронных и робототехнических систем
ПК-2.3	Владеть навыками разработки, отладки и сопровождения программного обеспечения для мехатронных и робототехнических систем

Обучающийся, освоивший дисциплину (модуль):

Должен знать:
- способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем.

Должен уметь:
разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем.

Должен владеть:
навыками разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем.

2. Место дисциплины (модуля) в структуре ОПОП ВО

Данная дисциплина (модуль) включена в Блок 1 "Дисциплины (модули)" Б1.В.ДВ.03 основной профессиональной образовательной программы 15.03.06 «Мехатроника и робототехника» (Физические основы мехатроники и робототехники) и относится к дисциплинам по выбору, части, формируемой участниками образовательных отношений.
Осваивается на 4 курсе в 8 семестре.

3. Объем дисциплины (модуля) в зачетных единицах с указанием количества часов, выделенных на контактную работу обучающихся с преподавателем (по видам учебных занятий) и на самостоятельную работу обучающихся

Общая трудоемкость дисциплины составляет 4 зачетных(ые) единиц(ы) на 144 часа(ов).
Контактная работа - 30 часа(ов), в том числе лекции - 10 часа(ов), практические занятия - 0 часа(ов), лабораторные работы - 20 часа(ов), контроль самостоятельной работы - 0 часа(ов).
Самостоятельная работа - 78 часа(ов).
Контроль (зачёт / экзамен) - 36 часа(ов).
Форма промежуточного контроля дисциплины: экзамен в 8 семестре

4. Содержание дисциплины (модуля), структурированное по темам (разделам) с указанием отведенного на них количества академических часов и видов учебных занятий

4.1 Структура и тематический план контактной и самостоятельной работы по дисциплине (модулю)

N	Разделы дисциплины / модуля	Семестр	Виды и часы контактной работы, их трудоемкость (в часах)			Самостоятельная работа
			Лекции	Практические занятия	Лабораторные работы	
1.	Тема 1. Объектно-ориентированный подход к проектированию и разработке программ	8	2	0	6	14
2.	Тема 2. Визуальное программирование	8	4	0	6	12
3.	Тема 3. Разработка графического интерфейса пользователя	8	2	0	4	14
4.	Тема 4. Развитые элементы приложений	8	2	0	4	14
	Итого: 144 часа (из них 36 часов контроль)		10	0	20	78

4.2 Содержание дисциплины (модуля)

Тема 1. Объектно-ориентированный подход к проектированию и разработке программ

Интегрированная среда разработки (ИСР) Microsoft Office Access - структура, возможности. Проект, файлы, входящие в состав проекта. Программирование под Windows. Библиотека функций Windows API. Основные принципы объектно-ориентированного программирования. Особенности языка программирования Object Pascal. Классы и объекты, поля, свойства, методы, события. Конструкторы и деструкторы.

Тема 2. Визуальное программирование

Форма: свойства и методы формы. События, организация реакции на них. Визуальные компоненты, обзор, использование, библиотека VCL. Компоненты ввода и отображения текстовой информации. Компоненты - элементы управления. Компоненты - меню. Компоненты внешнего оформления. Организация диалогов. Компоненты - диалоги.

Тема 3. Разработка графического интерфейса пользователя

Требования к интерфейсу, многооконные приложения. События клавиатуры и "мыши". Технология Drag&Drop. "Продвинутые" компоненты для организации интерфейса пользователя. Исключительные ситуации (ИС) - классы, иерархия, обработка, вызов. Компоненты отображения графической информации. Канва, перо, кисть, их свойства и методы. Мультимедиа и анимация.

Тема 4. Развитые элементы приложений

Помощь пользователю: Help-система, подсказки, строка состояния. Разработка и добавление компонентов. Библиотеки DLL, назначение, структура, статический и динамический вызовы. Разработка приложений, использующих сети Интернет. Процессы, потоки. Порождение дочерних процессов. Управление окнами внешних программ. Сообщения Windows и их обработка. COM-технология программирования. Свойства, методы серверов MS Word и MS Excel.

5. Перечень учебно-методического обеспечения для самостоятельной работы обучающихся по дисциплине (модулю)

Самостоятельная работа обучающихся выполняется по заданию и при методическом руководстве преподавателя, но без его непосредственного участия. Самостоятельная работа подразделяется на самостоятельную работу на аудиторных занятиях и на внеаудиторную самостоятельную работу. Самостоятельная работа обучающихся включает как полностью самостоятельное освоение отдельных тем (разделов) дисциплины, так и

проработку тем (разделов), осваиваемых во время аудиторной работы. Во время самостоятельной работы обучающиеся читают и конспектируют учебную, научную и справочную литературу, выполняют задания, направленные на закрепление знаний и отработку умений и навыков, готовятся к текущему и промежуточному контролю по дисциплине.

Организация самостоятельной работы обучающихся регламентируется нормативными документами, учебно-методической литературой и электронными образовательными ресурсами, включая:

Порядок организации и осуществления образовательной деятельности по образовательным программам высшего образования – программам бакалавриата, программам специалитета, программам магистратуры (утвержденный приказом Министерства науки и высшего образования Российской Федерации от 6 апреля 2021 года № 245)

Порядок организации и осуществления образовательной деятельности по образовательным программам высшего образования - программам бакалавриата, программам специалитета, программам магистратуры (утвержден приказом Министерства образования и науки Российской Федерации от 5 апреля 2017 года №301)

Письмо Министерства образования Российской Федерации №14-55-996ин/15 от 27 ноября 2002 г. "Об активизации самостоятельной работы студентов высших учебных заведений"

Устав федерального государственного автономного образовательного учреждения "Казанский (Приволжский) федеральный университет"

Правила внутреннего распорядка федерального государственного автономного образовательного учреждения высшего профессионального образования "Казанский (Приволжский) федеральный университет"

Локальные нормативные акты Казанского (Приволжского) федерального университета

6. Фонд оценочных средств по дисциплине (модулю)

Фонд оценочных средств по дисциплине (модулю) включает оценочные материалы, направленные на проверку освоения компетенций, в том числе знаний, умений и навыков. Фонд оценочных средств включает оценочные средства текущего контроля и оценочные средства промежуточной аттестации.

В фонде оценочных средств содержится следующая информация:

- соответствие компетенций планируемым результатам обучения по дисциплине (модулю);
- критерии оценивания сформированности компетенций;
- механизм формирования оценки по дисциплине (модулю);
- описание порядка применения и процедуры оценивания для каждого оценочного средства;
- критерии оценивания для каждого оценочного средства;
- содержание оценочных средств, включая требования, предъявляемые к действиям обучающихся, демонстрируемым результатам, задания различных типов.

Фонд оценочных средств по дисциплине находится в Приложении 1 к программе дисциплины (модулю).

7. Перечень литературы, необходимой для освоения дисциплины (модуля)

Освоение дисциплины (модуля) предполагает изучение учебной литературы. Литература может быть доступна обучающимся в одном из двух вариантов (либо в обоих из них):

- в электронном виде - через электронные библиотечные системы на основании заключенных КФУ договоров с правообладателями;
- в печатном виде - в Научной библиотеке Елабужского института КФУ. Обучающиеся получают учебную литературу на абонементе по читательским билетам в соответствии с правилами пользования Научной библиотекой.

Электронные издания доступны дистанционно из любой точки при введении обучающимся своего логина и пароля от личного кабинета в системе "Электронный университет". При использовании печатных изданий библиотечный фонд должен быть укомплектован ими из расчета не менее 0,25 экземпляра на каждого обучающегося из числа лиц, одновременно осваивающих данную дисциплину

Перечень литературы, необходимой для освоения дисциплины (модуля), находится в Приложении 2 к рабочей программе дисциплины. Он подлежит обновлению при изменении условий договоров КФУ с правообладателями электронных изданий и при изменении комплектования фондов Научной библиотеки Елабужского института КФУ.

8. Перечень ресурсов информационно-телекоммуникационной сети "Интернет", необходимых для освоения дисциплины (модуля)

Основы программирования C# - https://intuit.ru/studies/courses/2247/18/info

Программирование на C# - https://learn.microsoft.com/ru-ru/dotnet/csharp/

Туториал по C# - https://www.tutorialspoint.com/csharp/index.htm

9. Методические указания для обучающихся по освоению дисциплины (модуля)

Вид работ	Методические рекомендации
лекции	Лекционные занятия проводятся с использованием интерактивных технологий и предполагают активное участие студентов. Для подготовки к занятиям рекомендуется выделять в материале проблемные вопросы, затрагиваемые преподавателем в лекции, и группировать информацию вокруг них. Желательно выделять в используемой литературе постановки вопросов, на которые разными авторам могут быть даны различные ответы. На основании постановки таких вопросов следует собирать аргументы в пользу различных вариантов решения поставленных проблем.
лабораторные работы	Лабораторные занятия — это одна из разновидностей практического занятия, являющаяся эффективной формой учебных занятий в организации высшего образования. Лабораторные занятия имеют выраженную специфику в зависимости от учебной дисциплины, углубляют и закрепляют теоретические знания. На этих занятиях студенты осваивают конкретные методы изучения дисциплины, обучаются экспериментальным способам анализа, умению работать с приборами и современным оборудованием. Лабораторные занятия дают наглядное представление об изучаемых явлениях и процессах, студенты осваивают постановку и ведение эксперимента, учатся умению наблюдать, оценивать полученные результаты, делать выводы и обобщения. Отчёт по итогам выполненных лабораторных работ выполняется на листах белой бумаги формата А4 в печатном или рукописном виде. При оформлении отчёта используется сквозная нумерация страниц, считая титульный лист первой страницей. Номер страницы на титульном листе не ставится. Номера страницы ставятся по центру вверху. При оформлении отчёта в печатном виде желательно соблюдать следующие требования. Для заголовков: полужирный шрифт, 14 пт, центрированный. Для основного текста: нежирный шрифт, 14 пт, выравнивание по ширине. Во всех случаях тип шрифта - Times New Roman, отступ абзаца 1.25 см, полуторный междустрочный интервал. Поля: левое - 3 см, правое - 1 см, верхнее и нижнее - 2 см. Отчет должен содержать следующие элементы: 1) Титульный лист с обязательным указанием варианта; 2) Цель работы; 3) Задание; 4) Основная часть; 5) Вывод
самостоя-тельная работа	Самостоятельная работа студентов по дидактической сути представляет собой комплекс условий обучения, организуемых преподавателем и направленных на самоподготовку учащихся. Учебная деятельность протекает без непосредственного участия преподавателя и заключается в проработке лекционного материала, подготовке к лабораторным занятиям; изучении учебной литературы из основного и дополнительного списка.
экзамен	Экзамен по курсу проводится по билетам. В каждом билете один теоретический вопрос и одна задача. После ответа студенту могут быть заданы дополнительные вопросы, как по материалам билета, так и по основным определениям курса в целом. При подготовке к экзамену необходимо опираться, прежде всего, на конспекты лекций и рекомендованные источники информации, весь объём работы рекомендуется распределять равномерно по дням, отведённым для подготовки к экзамену и контролировать каждый день выполнения работы.

10. Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем (при необходимости)

Перечень информационных технологий, используемых при осуществлении образовательного процесса по дисциплине (модулю), включая перечень программного обеспечения и информационных справочных систем, представлен в Приложении 3 к рабочей программе дисциплины (модуля).

11. Описание материально-технической базы, необходимой для осуществления образовательного процесса по дисциплине (модулю)

Материально-техническое обеспечение образовательного процесса по дисциплине (модулю) включает в себя следующие компоненты:

Учебная аудитория для проведения учебных занятий лекционного типа, семинарского типа, групповых и индивидуальных консультаций, текущего контроля и промежуточной аттестации № 61

Комплект мебели для преподавателя – 1 шт., посадочные места для обучающихся – 30 шт., одноместные столы – 12 шт., компьютерные столы – 18 шт., компьютеры – 19 шт., интерактивная панель – 1 шт., меловая доска настенная – 1 шт., выход в интернет, внутривузовская компьютерная сеть, доступ в электронную информационно-образовательную среду.

Помещение для самостоятельной работы № 10

Посадочные места для пользователей – 28 шт., металлические двусторонние стеллажи для книг – 11 шт., книжный шкаф открытый – 5 шт., проектор – 1 шт., ноутбуки для пользователей – 11 шт., шкаф каталожный – 8 шт., шкаф для одежды – 1 шт., ксерокс – 1 шт., рабочий стол библиотекаря – 1 шт., компьютер библиотекаря – 1 шт., вешалка для одежды – 1 шт., жалюзи рулонные «Омега» с фотопечатью – 4 шт., стенд настенный (бронированное стекло) – 4 шт., шкаф-витрина встроенный в арку – 2 шт., шкаф-витрина стеклянный – 2 шт., стеллаж трубчатый с деревянными полками – 2 шт., рабочий стол для инвалидов и лиц с ОВЗ – 2 шт., стол СИ-1 рабочий для инвалидов-колясочников – 1 шт., компьютер– 2 шт., наушники – 2 шт., устройство «Говорящая книга» (тифлоплеер) – 2 шт., видеоувеличитель – 2 шт., радиокласс – 1 шт., портативный тактильный дисплей - 1 шт., сканирующая читающая машина - 1 шт., сканер – 1 шт., веб-камера – 1 шт., выход в интернет, внутривузовская компьютерная сеть, доступ в электронную информационно-образовательную среду.

12. Средства адаптации преподавания дисциплины к потребностям обучающихся инвалидов и лиц с ограниченными возможностями здоровья

При необходимости в образовательном процессе применяются следующие методы и технологии, облегчающие восприятие информации обучающимися инвалидами и лицами с ограниченными возможностями здоровья:

- создание текстовой версии любого нетекстового контента для его возможного преобразования в альтернативные формы, удобные для различных пользователей;

- создание контента, который можно представить в различных видах без потери данных или структуры, предусмотреть возможность масштабирования текста и изображений без потери качества, предусмотреть доступность управления контентом с клавиатуры;

- создание возможностей для обучающихся воспринимать одну и ту же информацию из разных источников - например, так, чтобы лица с нарушениями слуха получали информацию визуально, с нарушениями зрения - аудиально;

- применение программных средств, обеспечивающих возможность освоения навыков и умений, формируемых дисциплиной, за счёт альтернативных способов, в том числе виртуальных лабораторий и симуляционных технологий;

- применение дистанционных образовательных технологий для передачи информации, организации различных форм интерактивной контактной работы обучающегося с преподавателем, в том числе вебинаров, которые могут быть использованы для проведения виртуальных лекций с возможностью взаимодействия всех участников дистанционного обучения, проведения семинаров, выступления с докладами и защиты выполненных работ, проведения тренингов, организации коллективной работы;

- применение дистанционных образовательных технологий для организации форм текущего и промежуточного контроля;

- увеличение продолжительности сдачи обучающимся инвалидом или лицом с ограниченными возможностями здоровья форм промежуточной аттестации по отношению к установленной продолжительности их сдачи:

- продолжительности сдачи зачёта или экзамена, проводимого в письменной форме, - не более чем на 90 минут;

- продолжительности подготовки обучающегося к ответу на зачёте или экзамене, проводимом в устной форме, - не более чем на 20 минут;

- продолжительности выступления обучающегося при защите курсовой работы - не более чем на 15 минут.

Программа составлена в соответствии с требованиями ФГОС ВО и учебным планом по направлению 15.03.06 «Мехатроника и робототехника» и профилю подготовки "Физические основы мехатроники и робототехники".

Приложение №1
к рабочей программе дисциплины (модуля)
Б1.В.ДВ.03.02 Разработка программных приложений

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
"Казанский (Приволжский) федеральный университет"
Елабужский институт (филиал) КФУ

Фонд оценочных средств по дисциплине (модулю)
Разработка программных приложений

Направление подготовки: 15.03.06 Мехатроника и робототехника
Профиль подготовки: Физические основы мехатроники и робототехники
Квалификация выпускника: бакалавр
Форма обучения: очно-заочная
Язык обучения: русский
Год начала обучения по образовательной программе: 2024

СОДЕРЖАНИЕ

1. Соответствие компетенций планируемым результатам обучения по дисциплине (модулю)

Код и наименование компетенции	Индикаторы достижения компетенций для данной дисциплины	Оценочные средства текущего контроля и промежуточной аттестации
ПК-2 Способен разрабатывать, отлаживать, внедрять и сопровождать программное обеспечение мехатронных и робототехнических систем	Знать способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем Уметь разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем Владеть навыками разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем	Тестирование по темам Тема 1. Объектно-ориентированный подход к проектированию и разработке программ Тема 2. Визуальное программирование Тема 3. Разработка графического интерфейса пользователя Тема 4. Развитые элементы приложений Лабораторные работы по темам Тема 1. Объектно-ориентированный подход к проектированию и разработке программ Тема 2. Визуальное программирование Тема 3. Разработка графического интерфейса пользователя Тема 4. Развитые элементы приложений Промежуточная аттестация: *экзамен*

2. Критерии оценивания сформированности компетенций

Компетенция	Зачтено			Не зачтено
	Высокий уровень (отлично) (86-100 баллов)	**Средний уровень (хорошо) (71-85 баллов)**	**Низкий уровень (удовлетворительно) (56-70 баллов)**	**Ниже порогового уровня (неудовлетворительно) (0-55 баллов)**
ПК-2	Знает способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем	Знает основные способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем. Допускает незначительные ошибки при ответе на вопрос или решении поставленной задачи	Знает некоторые способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем. Допускает типичные ошибки при ответе на вопрос или решении поставленной задачи	Не знает способы разработки, отладки и сопровождения программных приложений для мехатронных и робототехнических систем
	Умеет разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем	Умеет разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем. Допускает незначительные ошибки при ответе на вопрос или решении поставленной задачи	Умеет разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем. Допускает типичные ошибки при ответе на вопрос или решении поставленной задачи	Не умеет разрабатывать, отлаживать и сопровождать программные приложения для мехатронных и робототехнических систем
	Владеет навыками разработки, отладки и сопровождения программных приложений для	Владеет навыками разработки, отладки и сопровождения программных приложений для мехатронных и	Владеет навыками разработки, отладки и сопровождения программных приложений для	Не владеет навыками разработки, отладки и сопровождения

	мехатронных и робототехнических систем	робототехнических систем. Допускает незначительные ошибки при ответе на вопрос или решении поставленной задачи	мехатронных и робототехнических систем. Допускает типичные ошибки при ответе на вопрос или решении поставленной задачи	программных приложений для мехатронных и робототехнически х систем

3. Распределение оценок за формы текущего контроля и промежуточную аттестацию

Текущий контроль:
Лабораторные работы по темам
Тема 1. Объектно-ориентированный подход к проектированию и разработке программ
Тема 2. Визуальное программирование
Тема 3. Разработка графического интерфейса пользователя
Тема 4. Развитые элементы приложений
Максимальное количество баллов по БРС - 30.

Тестирование
Тема 1. Объектно-ориентированный подход к проектированию и разработке программ
Тема 2. Визуальное программирование
Тема 3. Разработка графического интерфейса пользователя
Тема 4. Развитые элементы приложений
Максимальное количество баллов по БРС - 20.

Итого 30+20=50 баллов

Промежуточная аттестация - экзамен – 50 баллов
Промежуточная аттестация проводится после завершения изучения дисциплины или ее части в форме, определяемой учебным планом образовательной программы с целью оценить работу обучающегося, степень усвоения теоретических знаний, уровень сформированности компетенций.
Преподаватель, принимающий экзамен, обеспечивает случайное распределение вариантов экзаменационных заданий между обучающимися с помощью билетов и/или с применением компьютерных технологий; вправе задавать обучающемуся дополнительные вопросы и давать дополнительные задания помимо тех, которые указаны в билете.

Устный или письменный ответ – 20 баллов.
Практическое задание – 30 баллов.
Итого 20+30=50 баллов.
Общее количество баллов по дисциплине за текущий контроль и промежуточную аттестацию: 50+50=100 баллов.

Соответствие баллов и оценок:
Для экзамена:
86-100 – отлично
71-85 – хорошо
56-70 – удовлетворительно
0-55 – неудовлетворительно

4. Оценочные средства, порядок их применения и критерии оценивания
4.1. Оценочные средства текущего контроля
4.1.1. Лабораторные работы
Тема 1. Объектно-ориентированный подход к проектированию и разработке программ
Тема 2. Визуальное программирование
Тема 3. Разработка графического интерфейса пользователя
Тема 4. Развитые элементы приложений
4.1.1.1. Порядок проведения.
Лабораторные работы проводятся в часы аудиторной работы.
Перед выполнением каждой работы студенты-бакалавры должны проработать соответствующий материал, используя конспекты теоретических занятий, периодические издания, учебно-методические пособия и учебники.
По окончании занятий студенты оформляют отчет по каждой работе, соблюдая следующую форму:

- Наименование темы;
- Цель работы;
- Задание и содержание выполненной работы,
- Письменные ответы на контрольные вопросы.
- Выводы по проделанной работе.
- Список использованных источников.

4.1.1.2 Критерии оценивания

27-30 баллов ставится, если обучающийся:
Правильно выполнил все задания. Продемонстрировал высокий уровень владения материалом. Проявлены превосходные способности применять знания и умения к выполнению конкретных заданий.
22-26 баллов ставится, если обучающийся:
Правильно выполнил большую часть заданий. Присутствуют незначительные ошибки. Продемонстрирован хороший уровень владения материалом. Проявлены средние способности применять знания и умения к выполнению конкретных заданий.
18-21 баллов ставится, если обучающийся:
Задания выполнил более чем наполовину. Присутствуют серьёзные ошибки. Продемонстрирован удовлетворительный уровень владения материалом. Проявлены низкие способности применять знания и умения к выполнению конкретных заданий.
0-17 баллов ставится, если обучающийся:
Задания выполнил менее чем наполовину. Продемонстрирован неудовлетворительный уровень владения материалом. Проявлены недостаточные способности применять знания и умения к выполнению конкретных заданий.

4.1.1.3. Содержание оценочного средства

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

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++ - Manadged C++, Visual Basic.NET, Java.NET, C#*.

В лабораторной работе проводится изучении 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.
2. Для создания каркаса приложения можно использовать мастер создания приложений - *Application Wizard*.

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

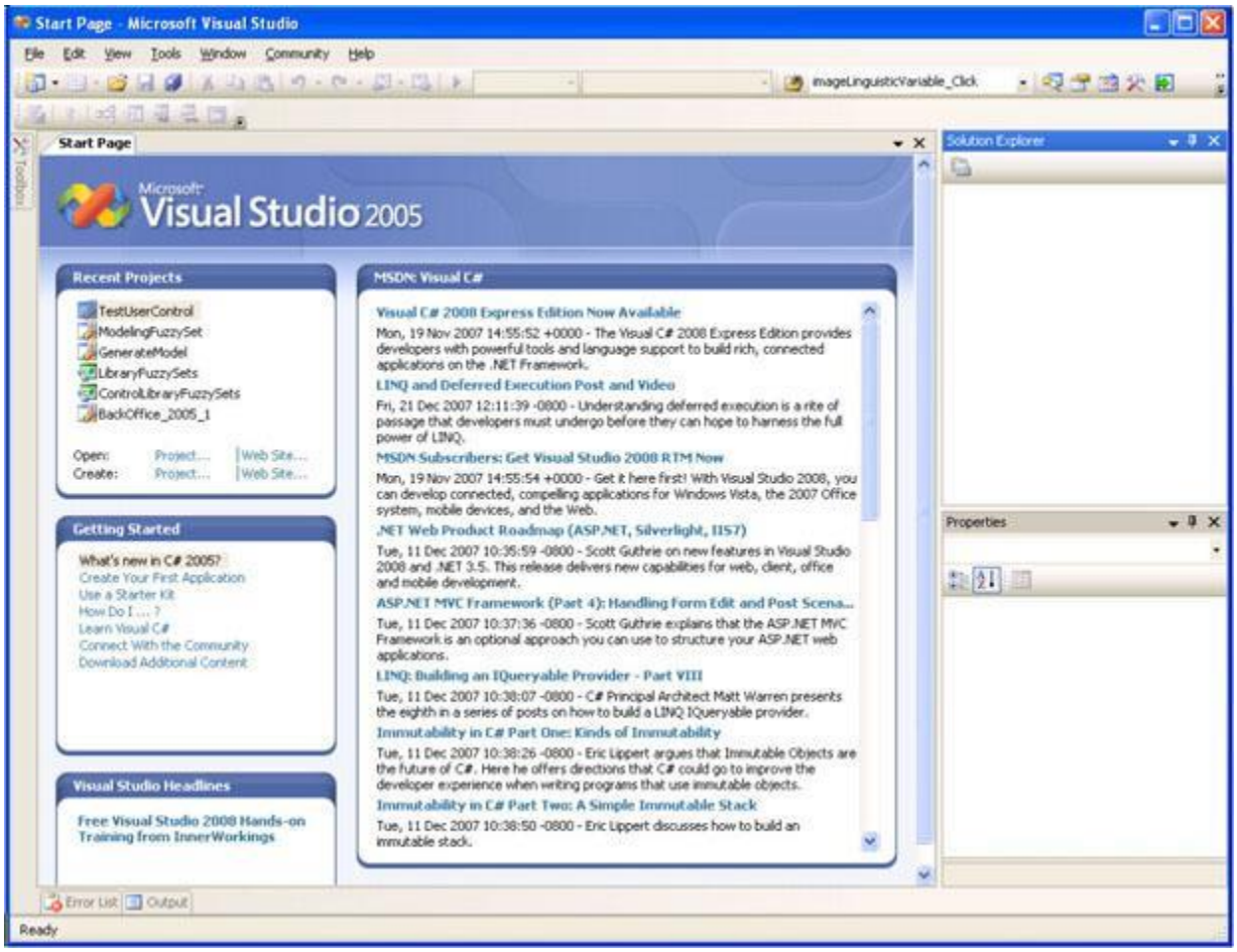

Рис. 1.1. Стартовое диалоговое окно IDE

4. Добавить код C#, 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:

1. 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).

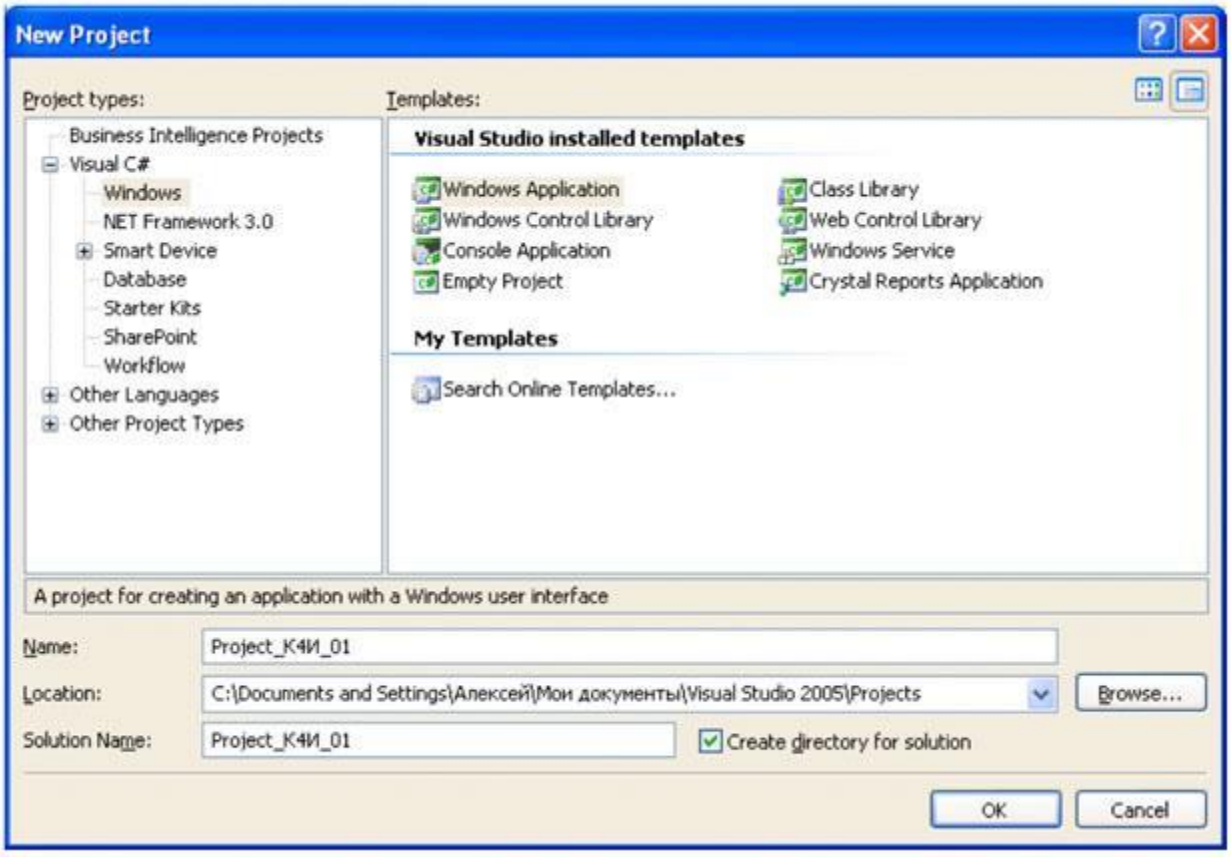

Рис. 1.2. Мастер создания нового проекта (New Project Wizard)

2. В дереве, отображаемом в подокне *Project Type* (Типы проектов) выберите *"Visual C# /Windows"*. В подокне *Templates* (Шаблоны) выберите *Windows Application* (Приложение *Windows*).
3. В поле *Name* (Название проекта) наберите имя проекта - Project_К4И_01 (имя проекта присваивается в соответствии со следующим синтаксисом: Project_"номер группы"_"номер бригады в группе").
4. Щелкните на кнопке *OK*. (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).

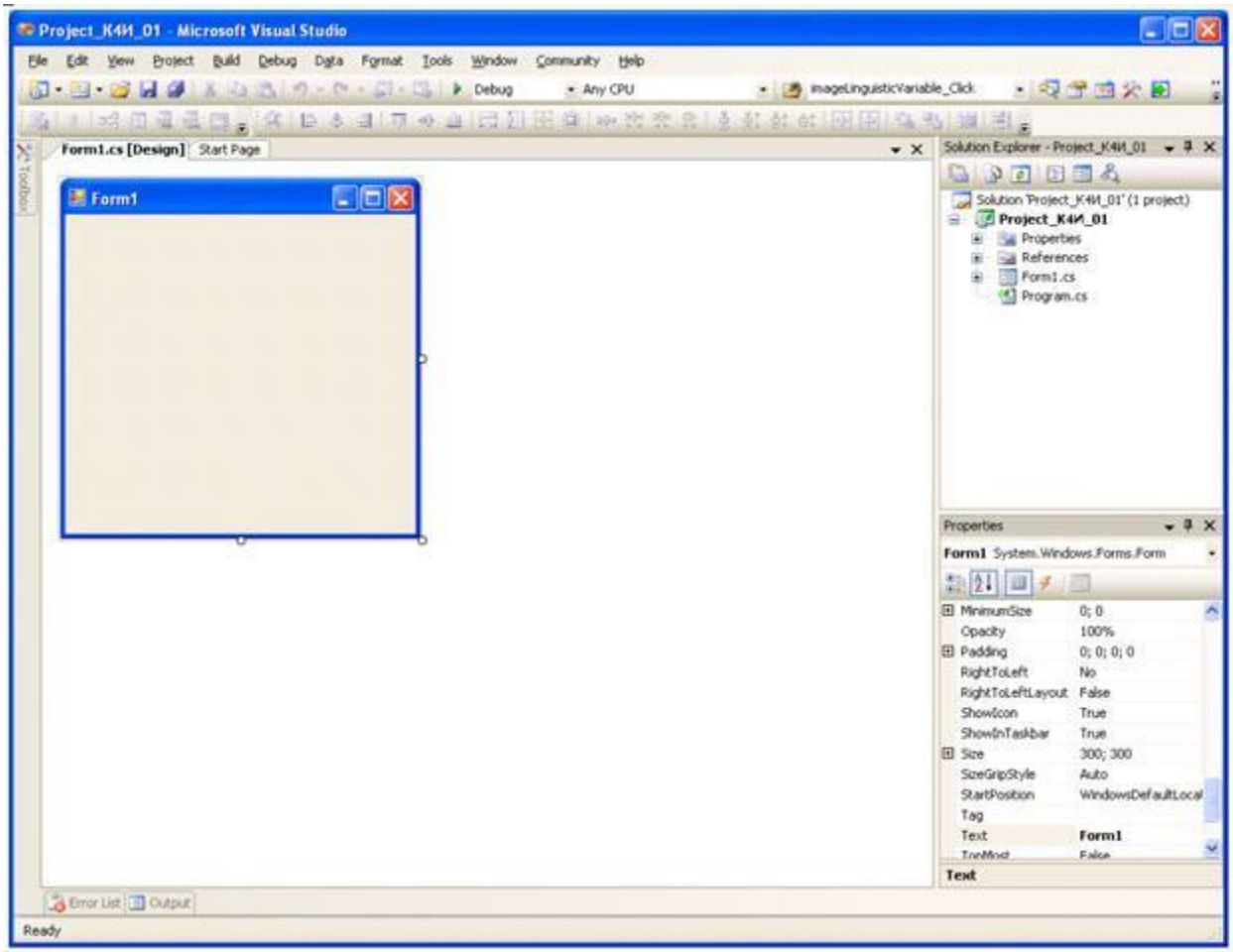

Рис. 1.3. Графический шаблон главного окна приложения

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).

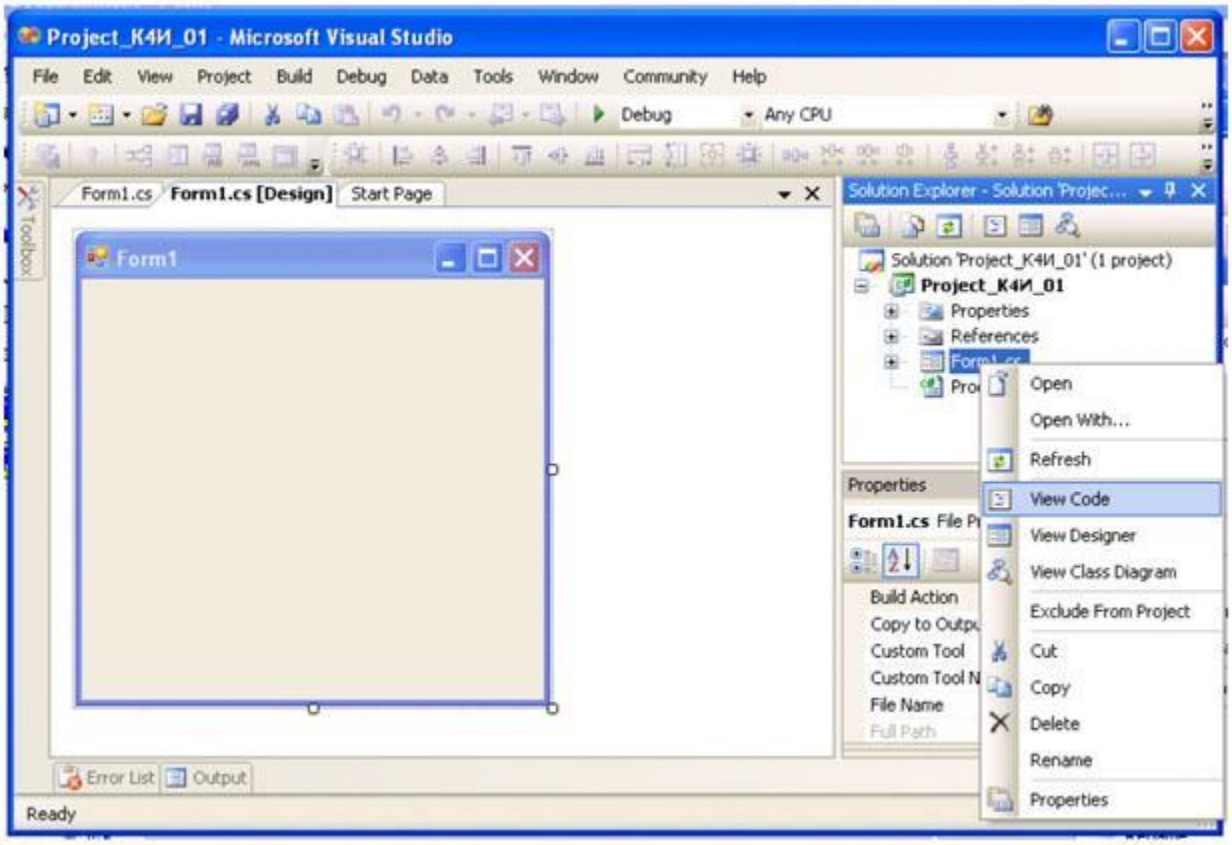

Рис. 1.4. Выбор режима View Code в контекстном меню

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_К4И_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).

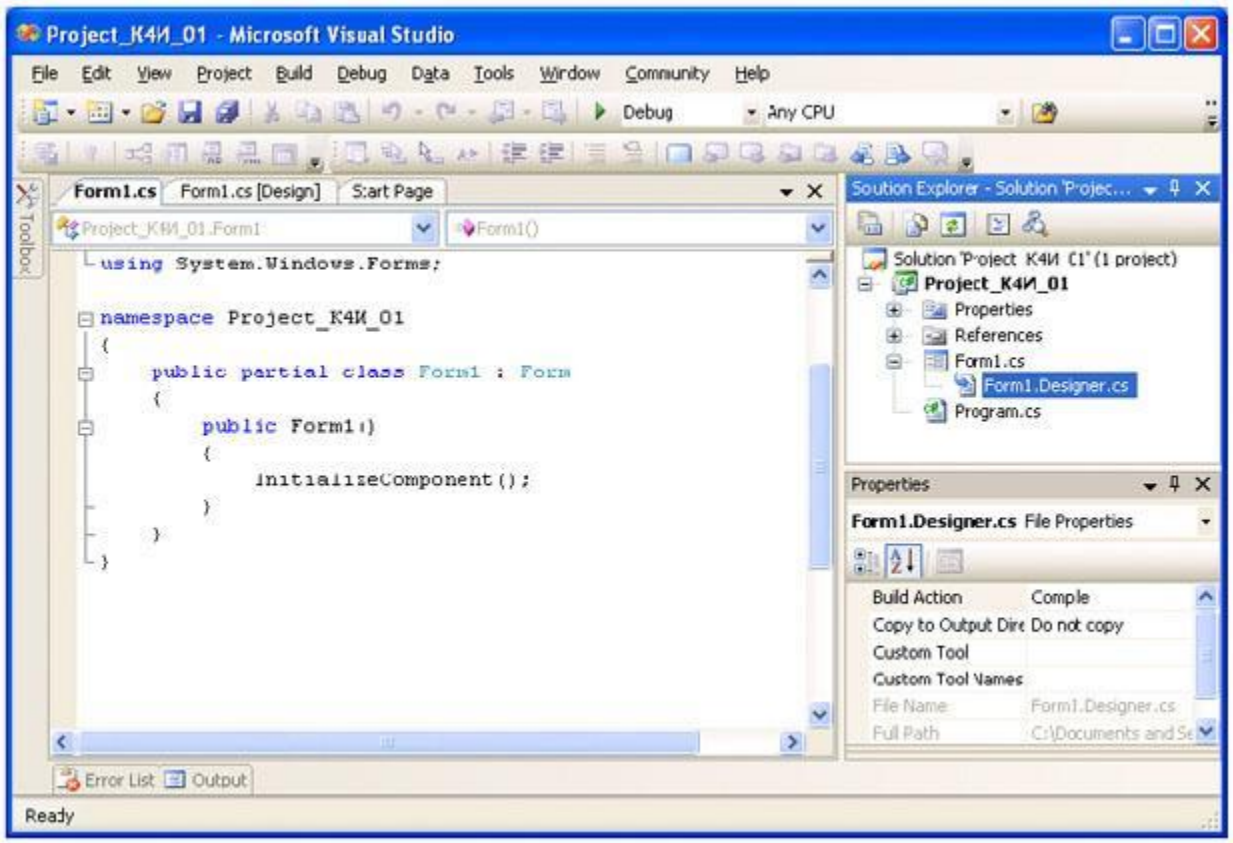

Рис. 1.5. Выбор режима Form.Designer.cs

```
namespace Project_К4И_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_К4И_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 RemoveMessageFilter( ) и выполнять с этими сообщениями необходимые предварительные действия. Для того чтобы добавить фильтр сообщений, необходимо указать класс, реализующий интерфейс IMessageFilter
DoEvents( )	Обеспечивает способность приложения обрабатывать сообщения из очереди сообщений во время выполнения какой-либо длительной операции. Можно сказать, что DoEvents() - это "быстрый и грязный" заменитель нормальной многопоточности
Exit( )	Завершает работу приложения
ExitThred( )	Прекращает обработку сообщений для текущего потока и закрывает все окна, владельцем которых является этот поток
OLERequired( )	Инициализирует библиотеки OLE. Можете считать этот метод эквивалентом *.NET* для вызываемого вручную метода Olelnitialize( )

Run( )	Запускает стандартный цикл работы с сообщениями для текущего потока

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

Таблица 1.2. Наиболее важные свойства типа Application

Свойство	Назначение
CommonAppDataRegistry	Возвращает параметр системного реестра, который хранит общую для всех пользователей информацию о приложении
CompanyName	Возвращает имя компании
CurrentCulture	Позволяет задать или получить информацию о естественном языке, для работы с которым предназначен текущий поток
CurrentlnputLanguage	Позволяет задать или получить информацию о естественном языке для ввода информации, получаемой текущим потоком
ProductName	Для получения имени программного продукта, 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).

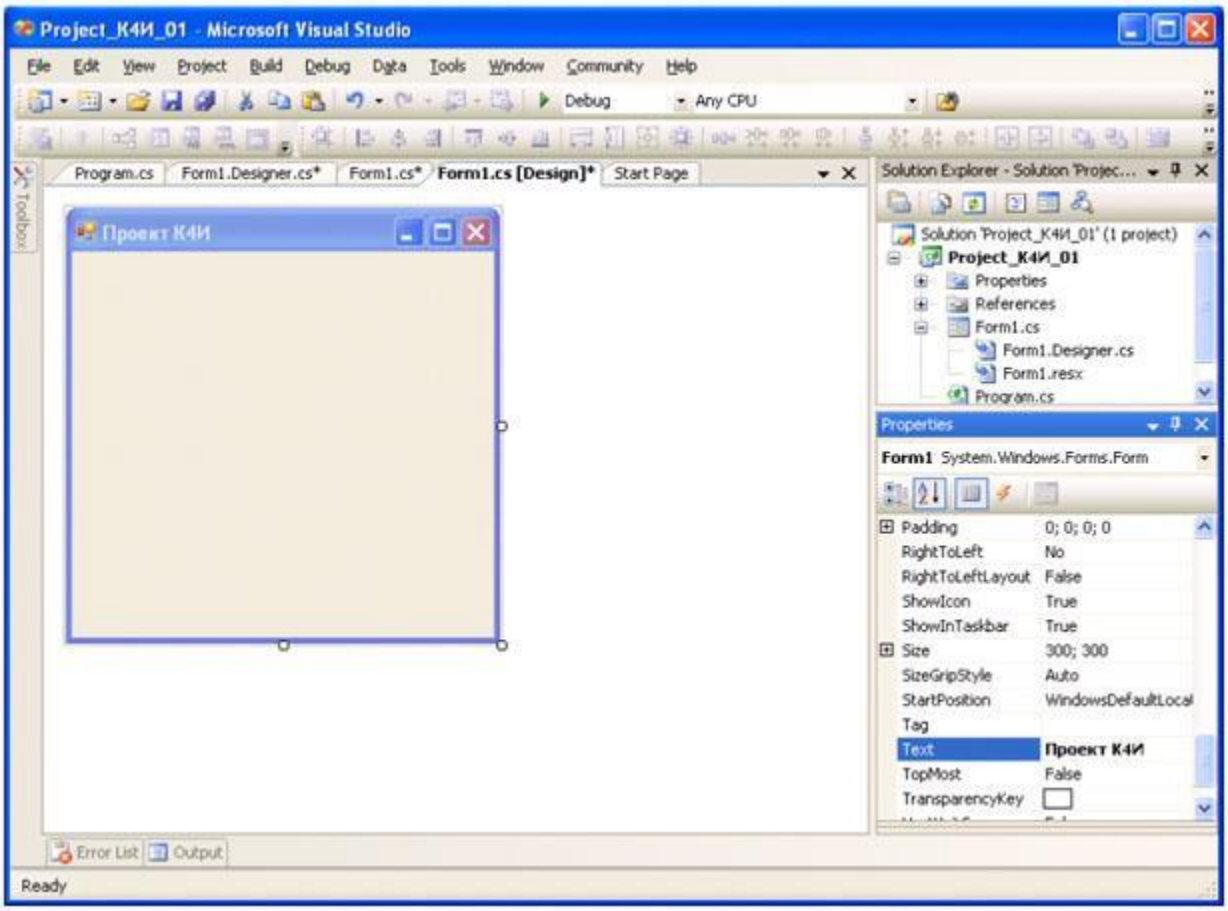

Рис. 1.6. Изменение значения в поле Text на вкладке Properties

Откомпилируйте *приложение*, выбрав из главного *меню* команду Build Project_К4И_01 (рисунок 1.7)

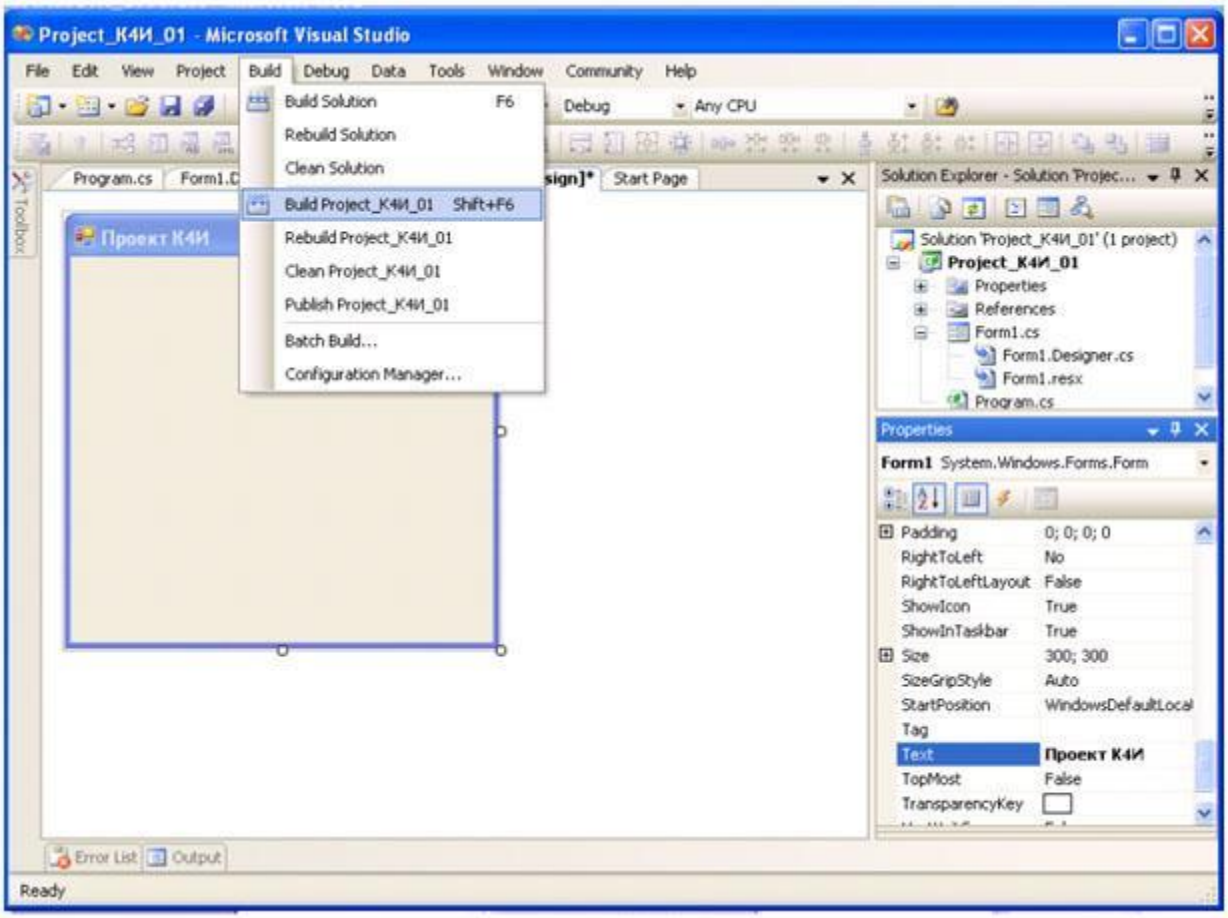

Рис. 1.7. Выбор из главного меню команды Build

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_К4И_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.

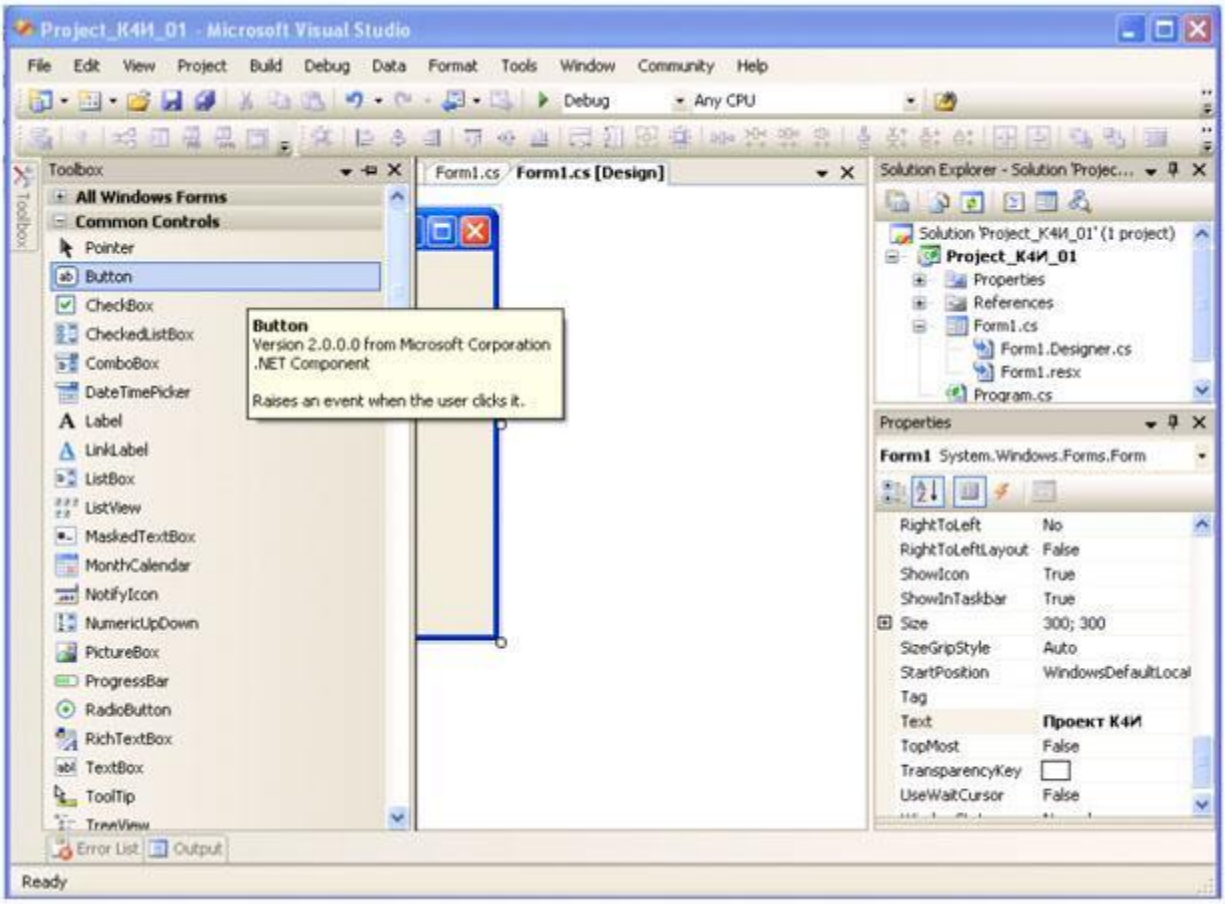

Рис. 1.9. Вкладка ToolBox

В результате получим форму с кнопкой (рисунок 1.10).

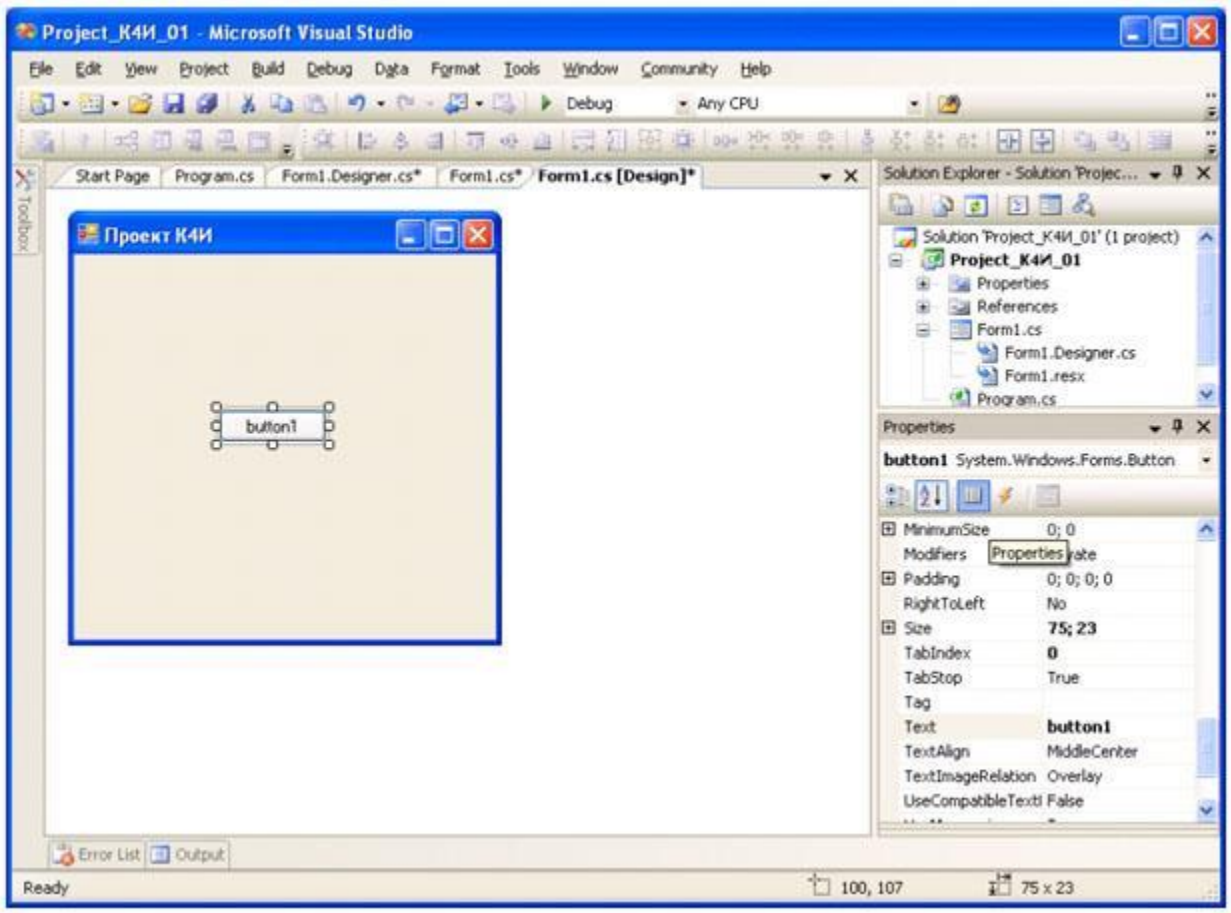

Рис. 1.10. Форма с установленной кнопкой

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).

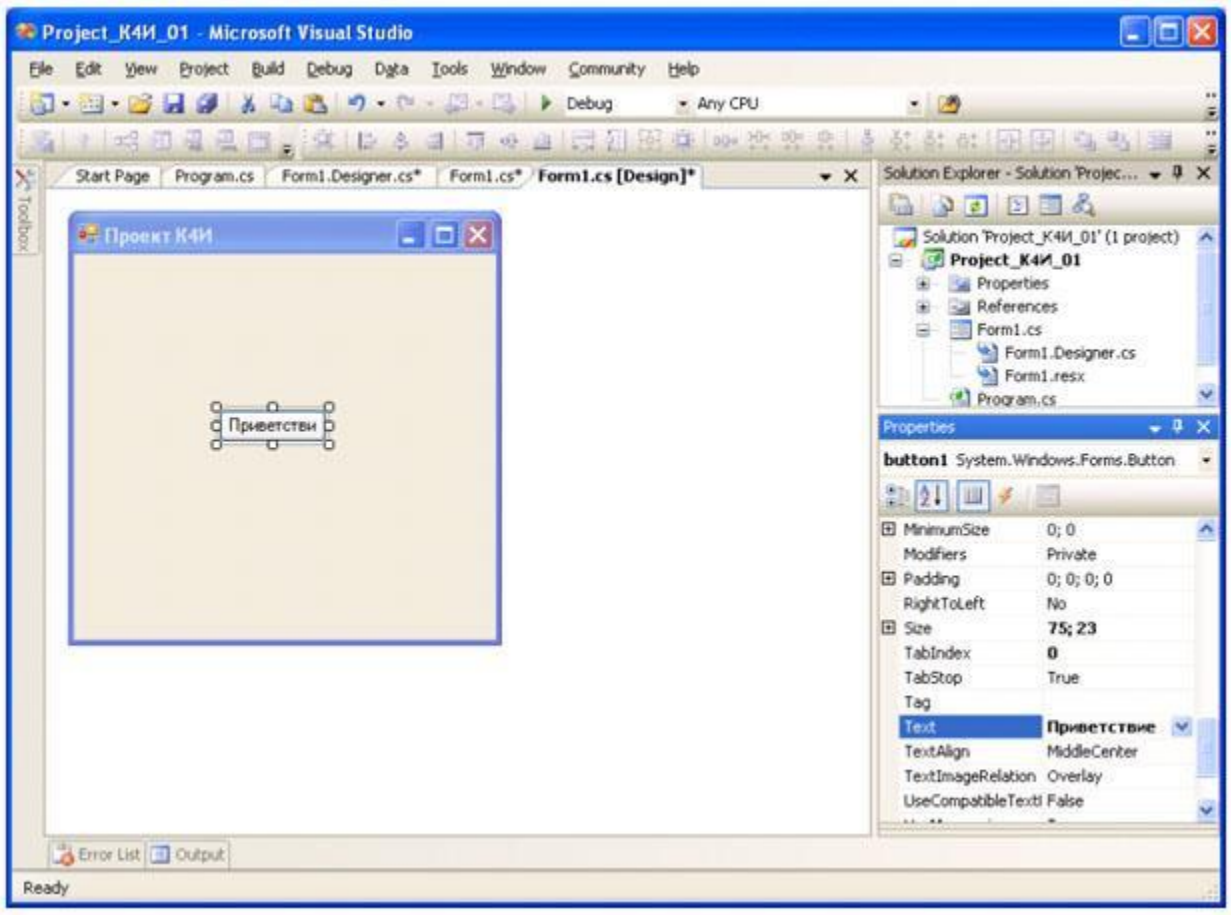

Рис. 1.11. Форма с измененным свойством Text кнопки

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).

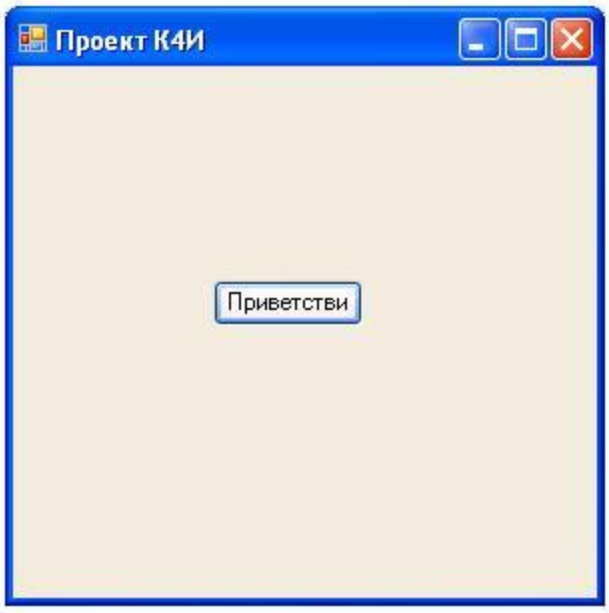

Рис. 1.12. Результат выполнения приложения

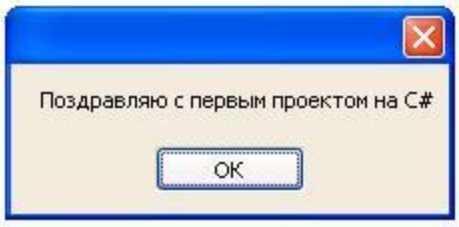

Рис. 1.13. Вывод сообщения

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.

Задание на лабораторную работу

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.
8. Протестировать работу приложения.

2. Лабораторная работа: Создание главного меню приложения
Цель работы: Изучить основные способы разработки главного *меню* приложении. Получить практические навыки в создании главного *меню* 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 восприняты 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 управлении наследуется от ToolStrip. Система *меню* строится добавлением объектов ToolStripMenu 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.

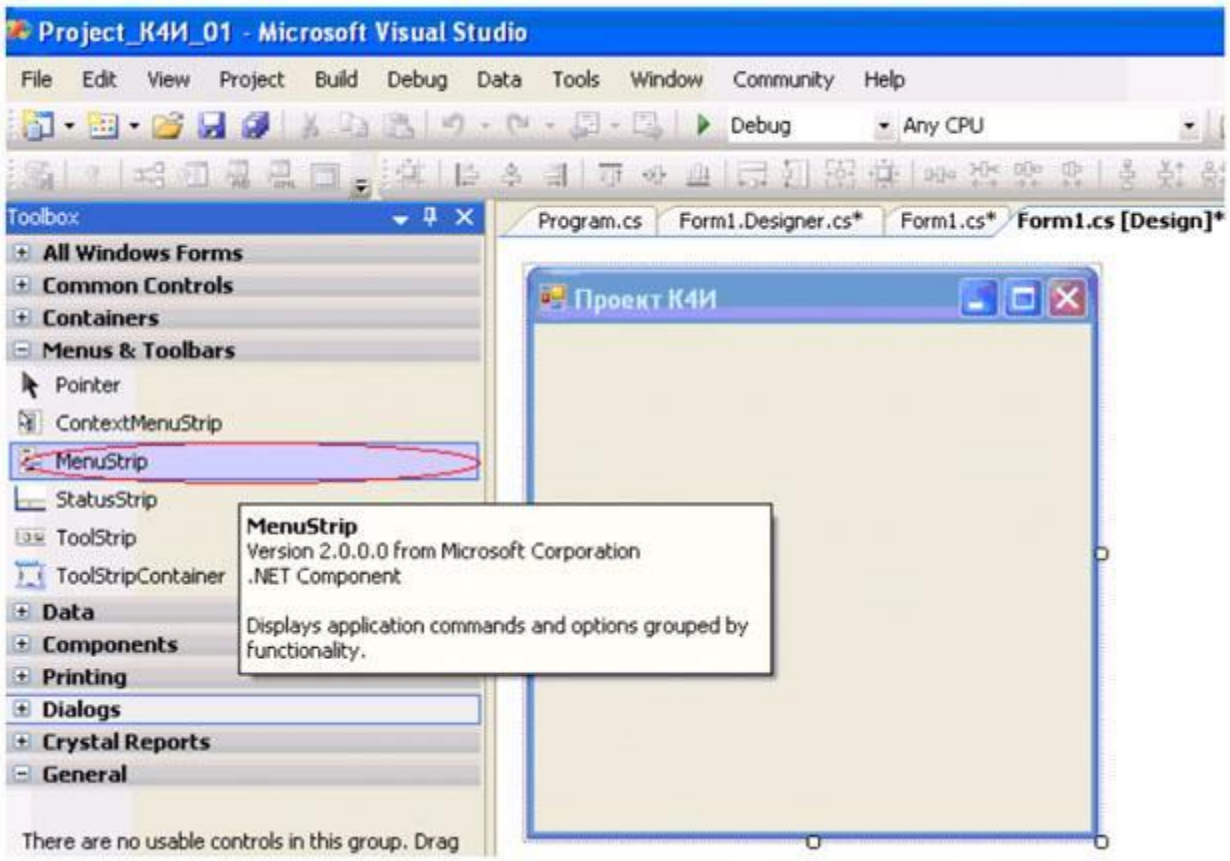

Рис. 2.1. Элемент управления MenuStrip

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

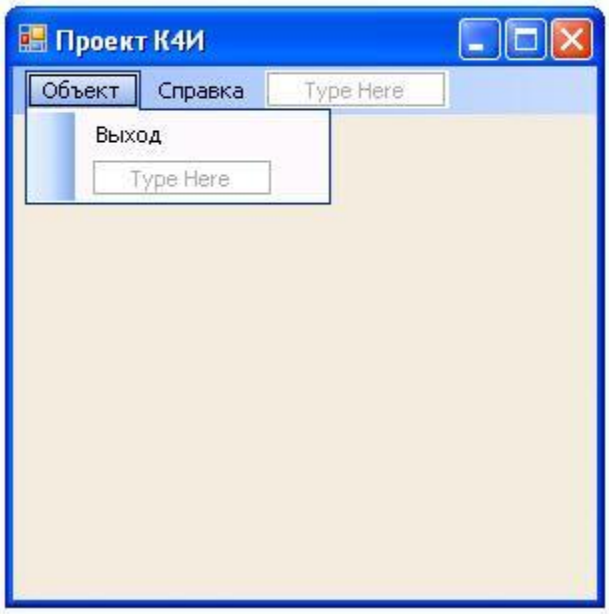

Рис. 2.2. Простое меню на форме

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).

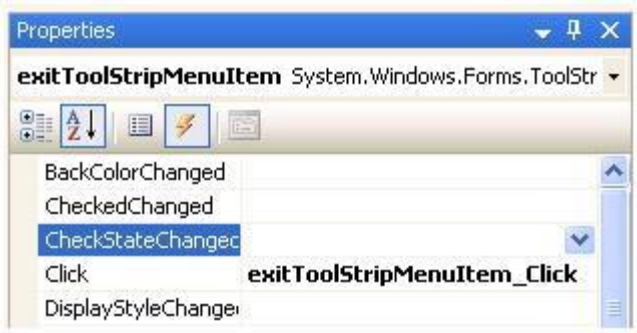

Рис. 2.3. Событие Click и обработчик события exitToolStripMenuItem_Click

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*.

Задание на лабораторную работу

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.

3. Лабораторная работа: Создание многооконного приложения

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

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 WindowsState 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 menuItemEmploye_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 дочерее 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

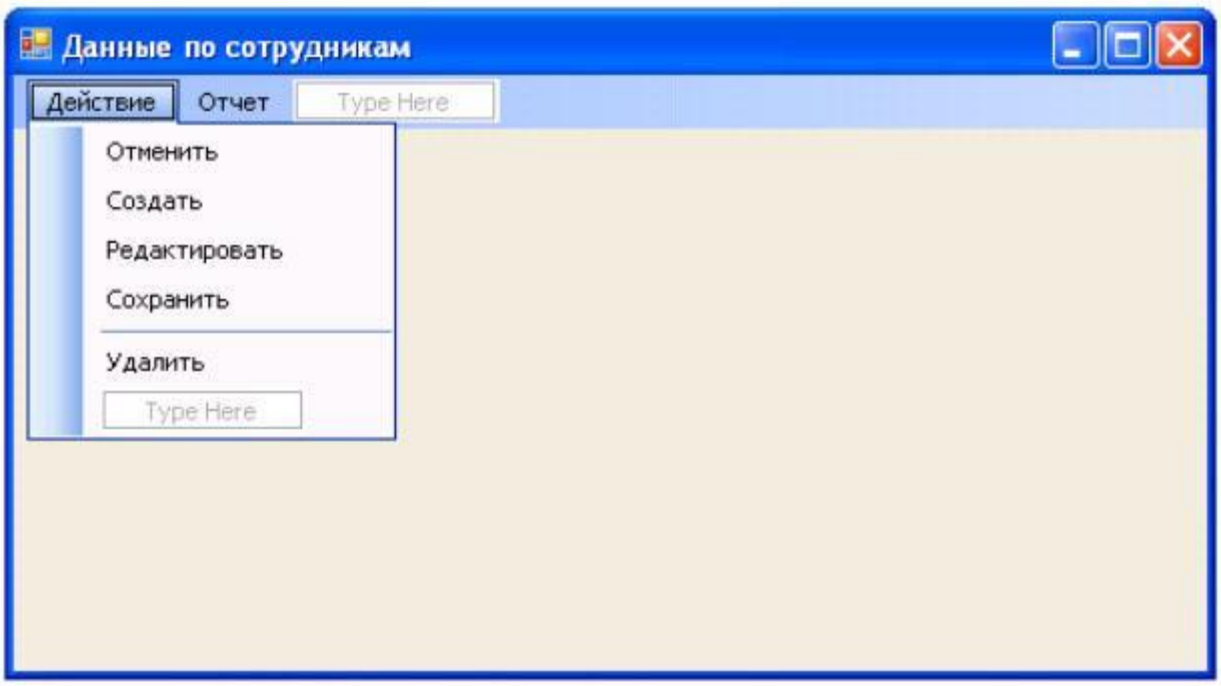

Рис. 3.1. Дочернее окно с меню

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.

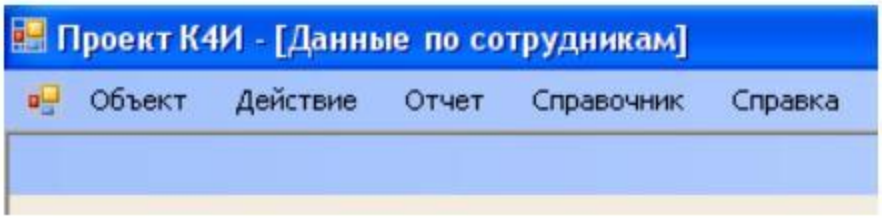

Рис. 3.2. Дочернее окно с подключенным меню

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 дизайнере 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.

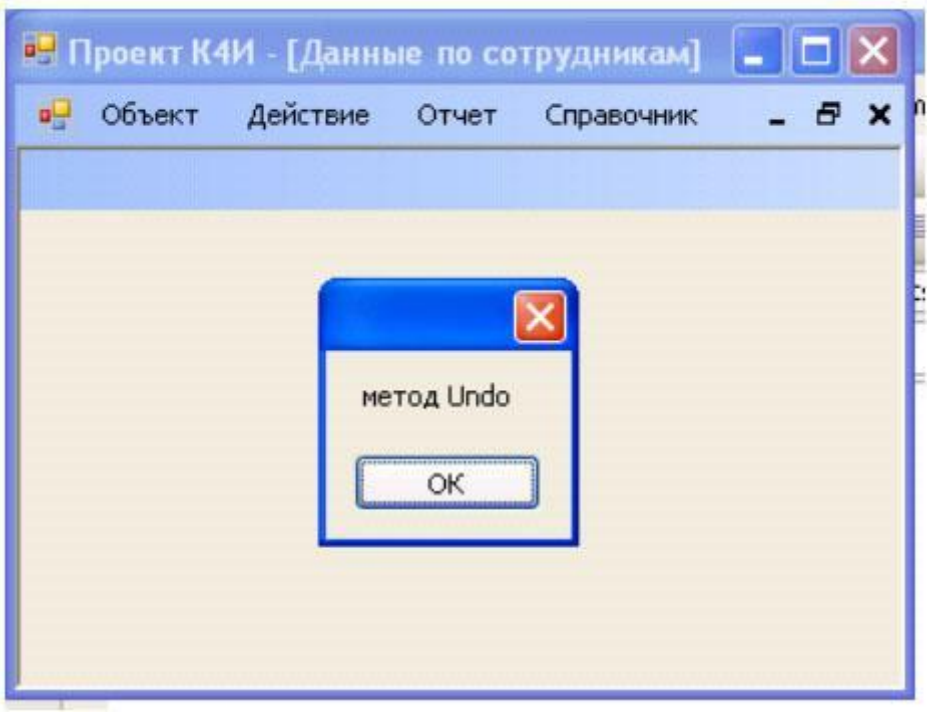

Рис. 3.3. Дочернее окно с подключенным меню

Аналогичным образом создайте методы-заглушки для функций "Создать", "Редактировать", "Сохранить" и "Удалить".

Задание на лабораторную работу

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.
7. Протестировать работу приложения.

4. Лабораторная работа: Создание пользовательских диалоговых окон

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

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 лабораторной 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 MessageBox**
OK	OK
OKCancel	OK, Cancel
YesNo	Yes, No
YesNoCancel	Yes, No, Cancel
RetryCancel	Retry, Cancel
AbortRetryIgnore	Abort, Retry, Ignore

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
Asterisk, Information	
Error, Stop	
Exclamation, Warning	
Question	
None	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
Button l	1
Button 2	2
Button 3	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 DialoqResult

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 сотруднику? ",
     "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 .

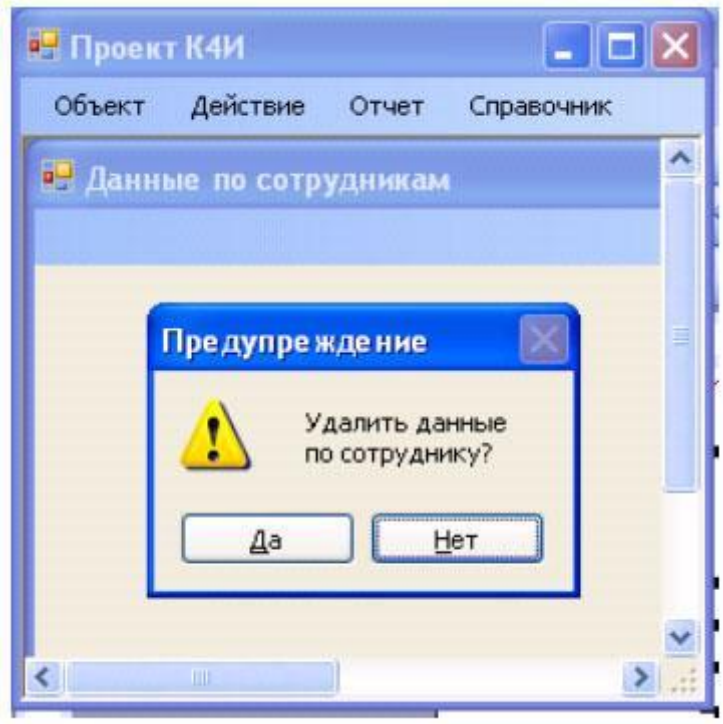

Рис. 4.1. Модальное диалоговое окно предупреждения

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 .

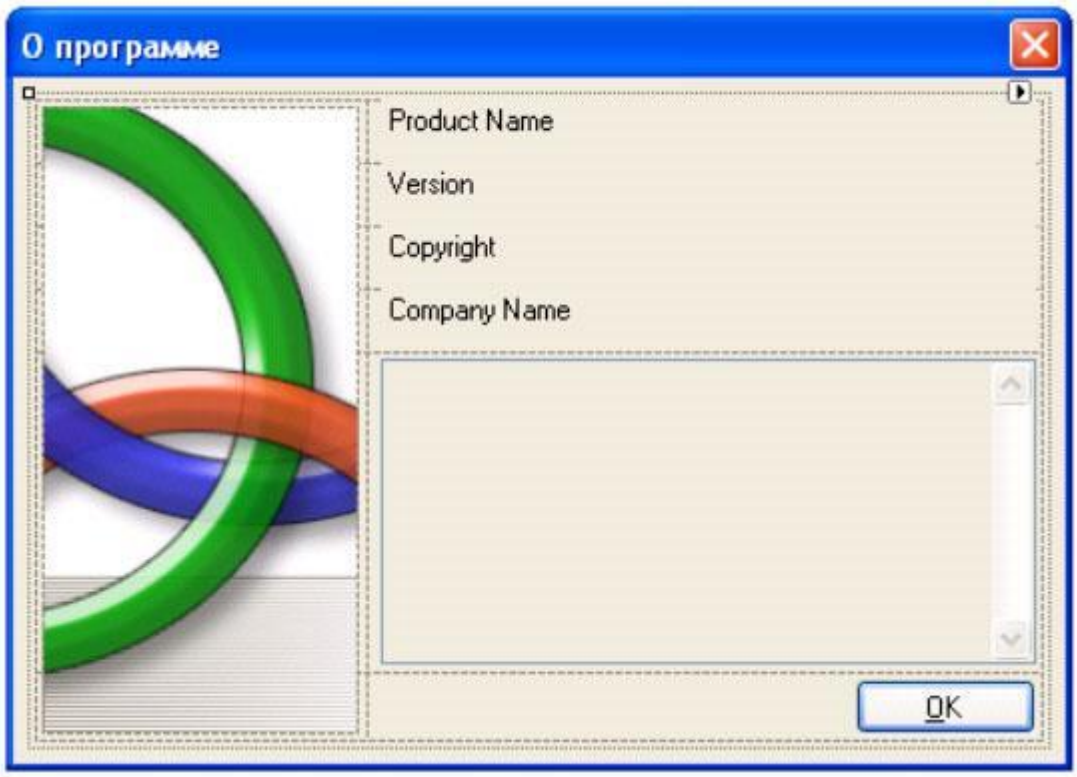

Рис. 4.2. Общий вид Windows - формы FormAbout

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

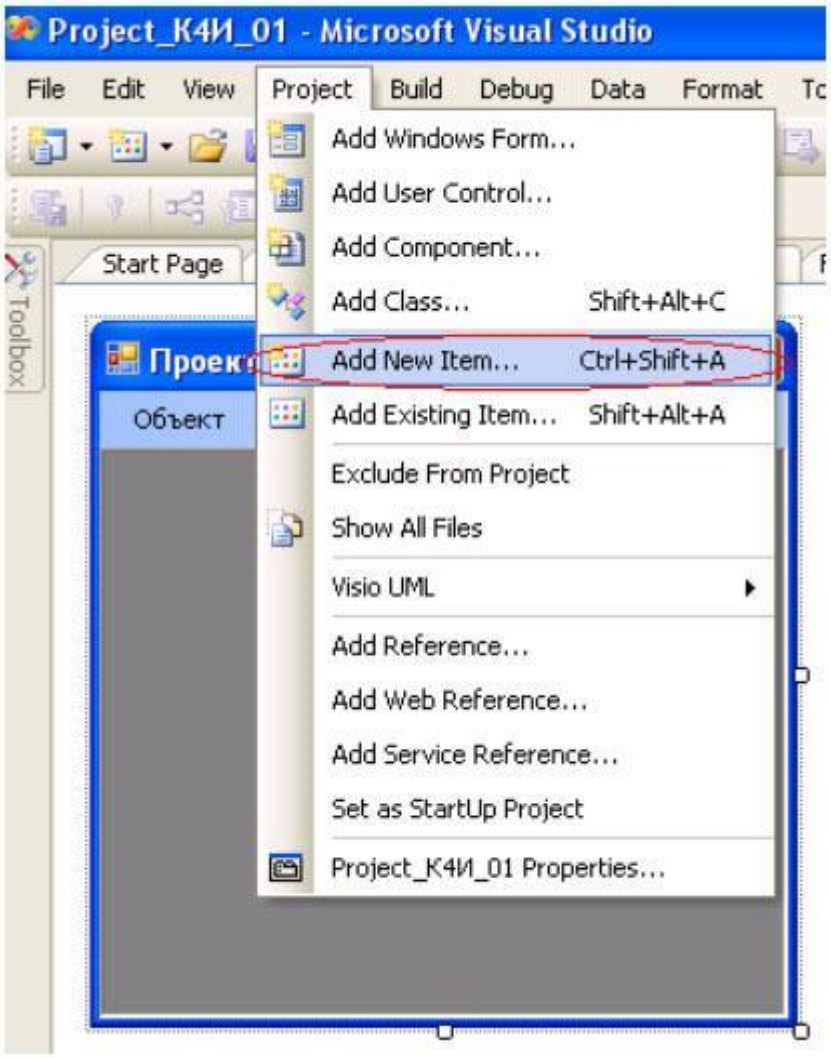

Рис. 4.3. Выбор режима добавления нового компонента в проект

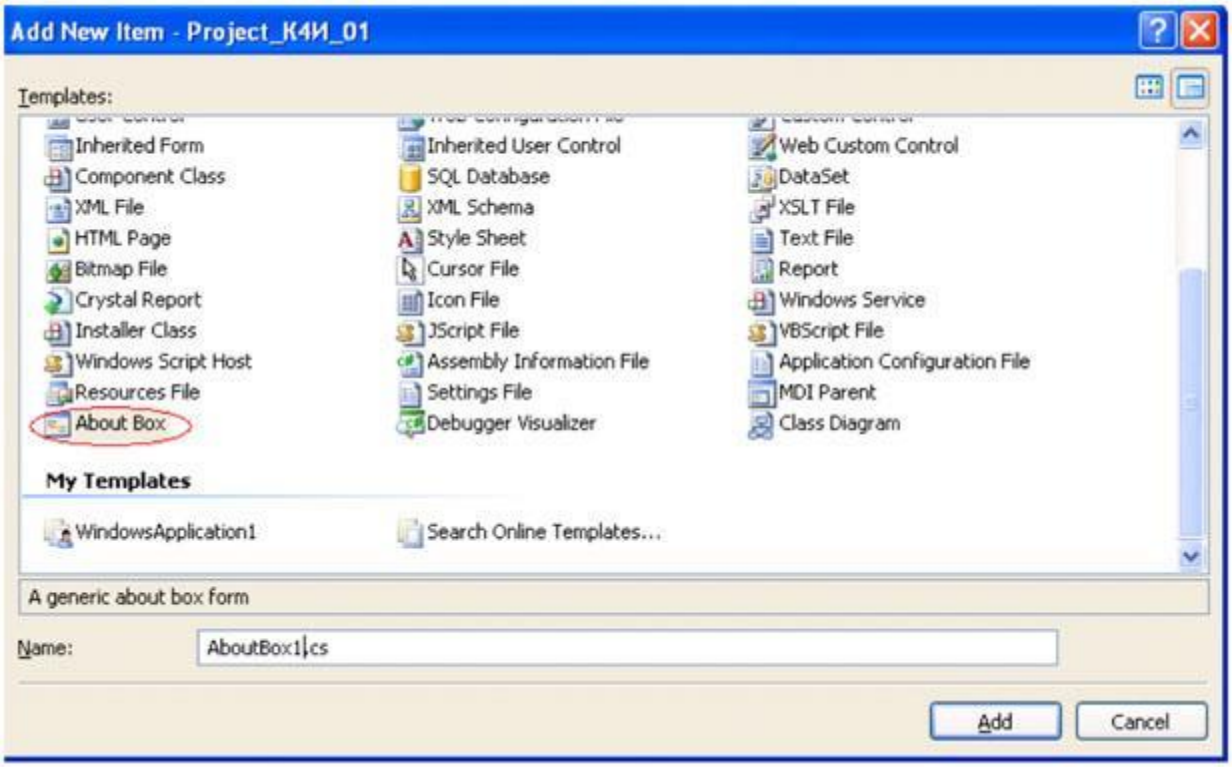

Рис. 4.4. Добавление нового компонента в проект

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 = "@РГЭУ, 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 лабораторной 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 дисплеи будет выведено *диалоговое окно*, приведенное на рисунке 4.5.

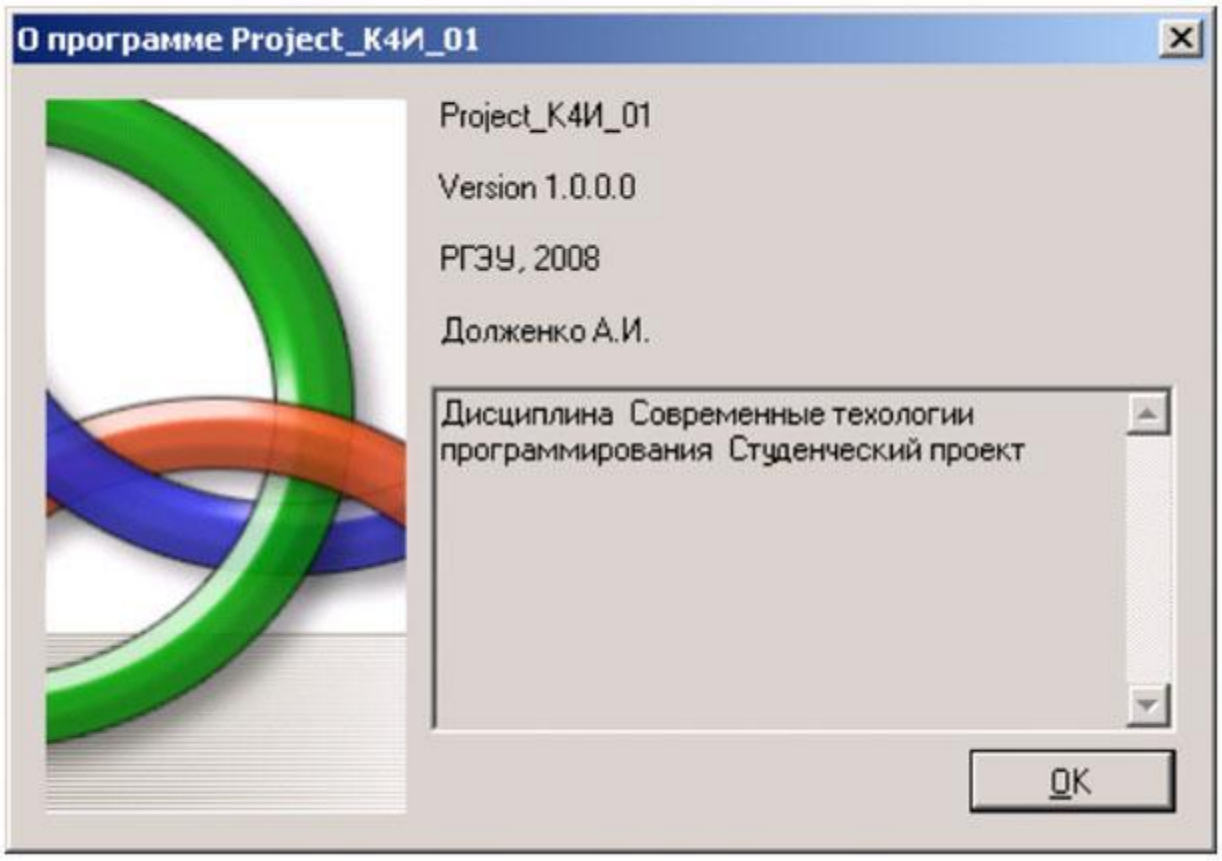

Рис. 4.5. Вызов модального окна

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

Задание на лабораторную работу

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.

5. Лабораторная работа: Создание панели инструментов и контекстного меню

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

В приложении для повышения качества интерфейса пользователя целесообразно предусматривать различные способы активизации функций системы. При выполнении лабораторной 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*.

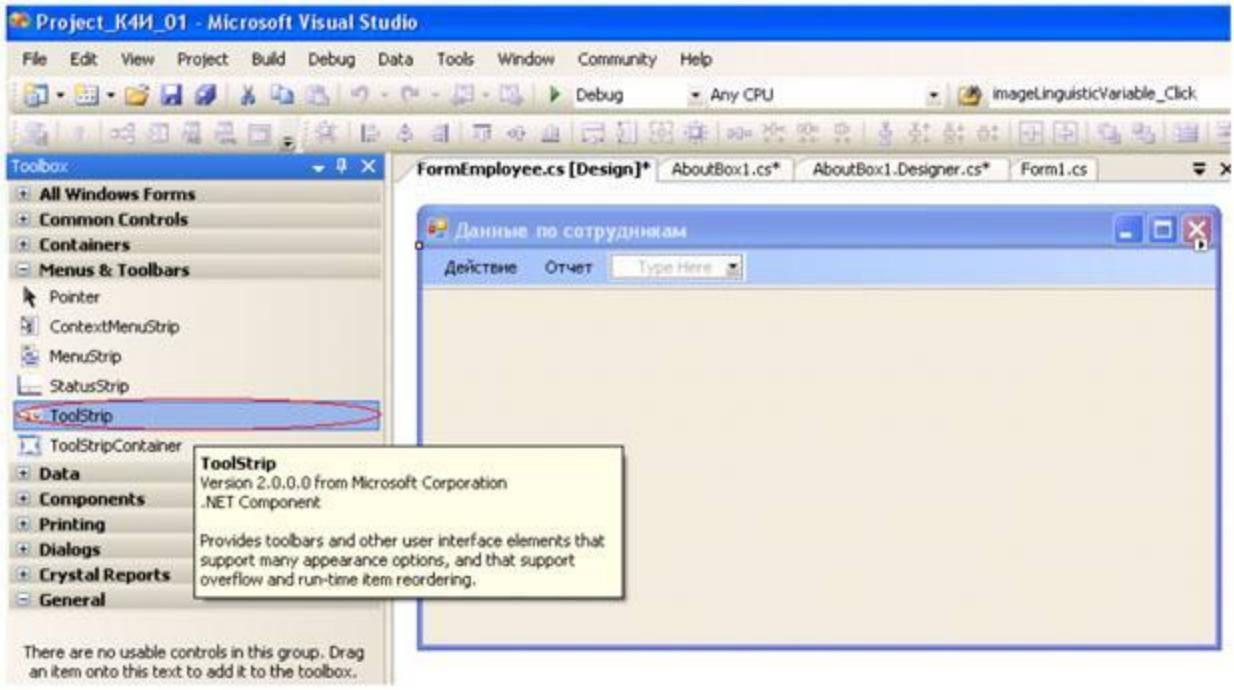

Рис. 5.1. Окно свойств панели инструментов

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.

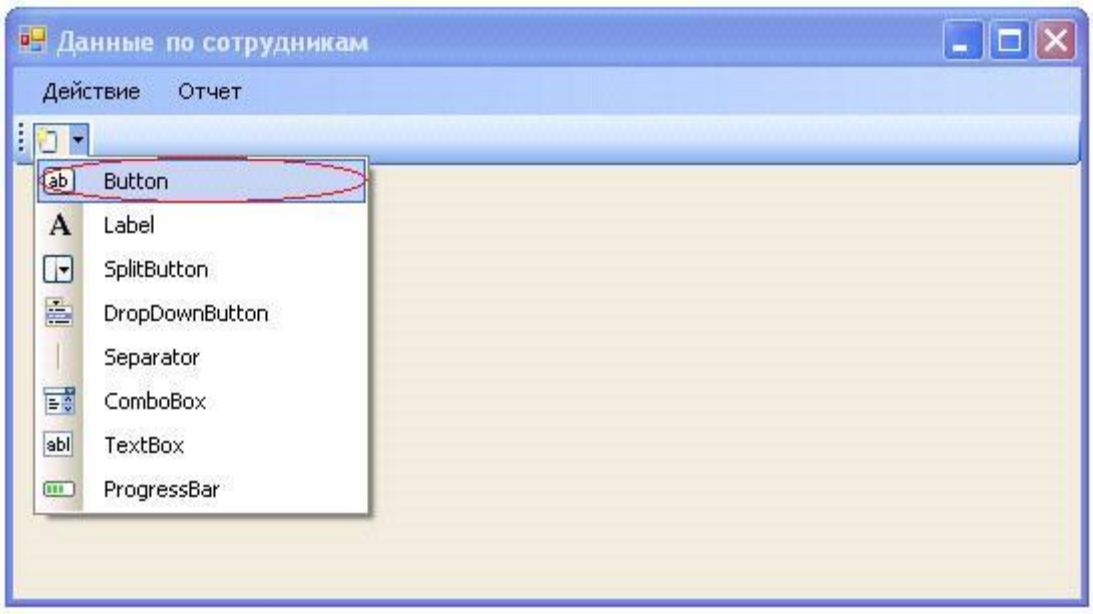

Рис. 5.2. Окно свойств панели инструментов

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

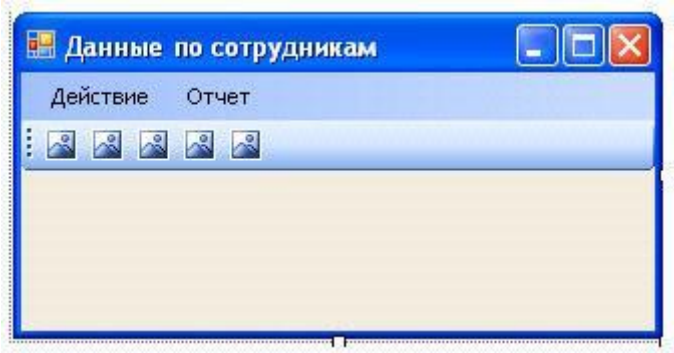

Рис. 5.3. Форма FormEmployee с панелью инструментов

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).

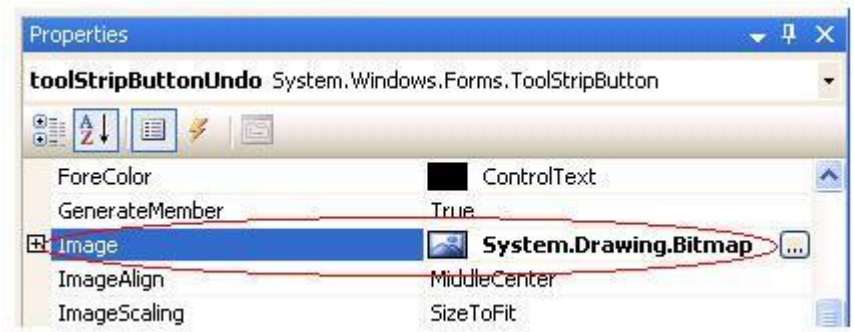

Рис. 5.4. Свойство Image для кнопки панели инструментов

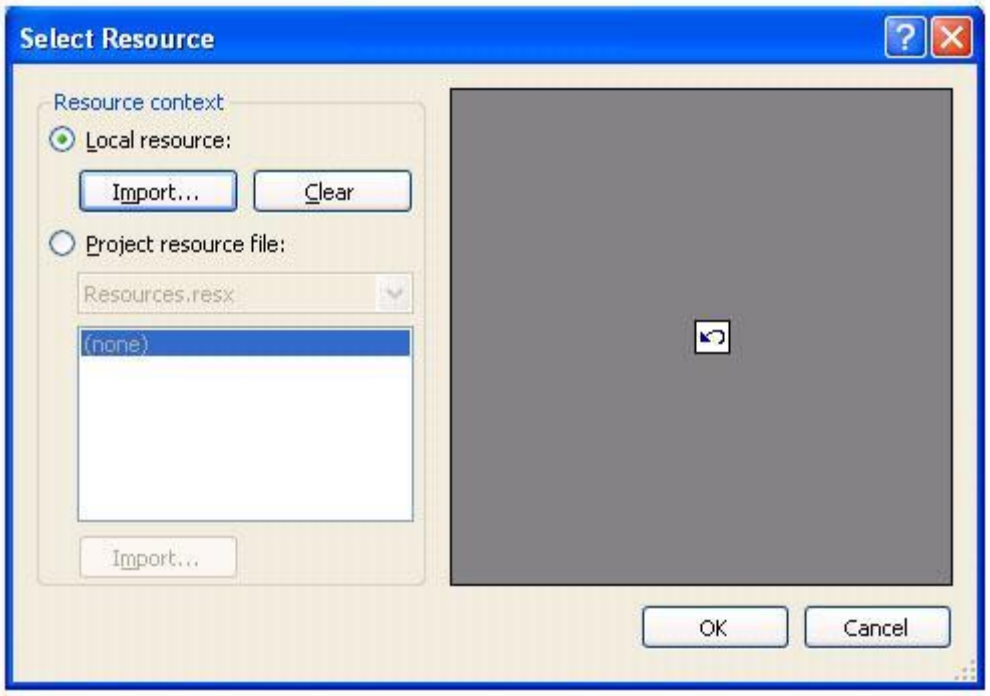

Рис. 5.5. Добавление изображения в ImageList

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 (для лабораторной работы графические файлы можно найти в папке Лабораторные работы).

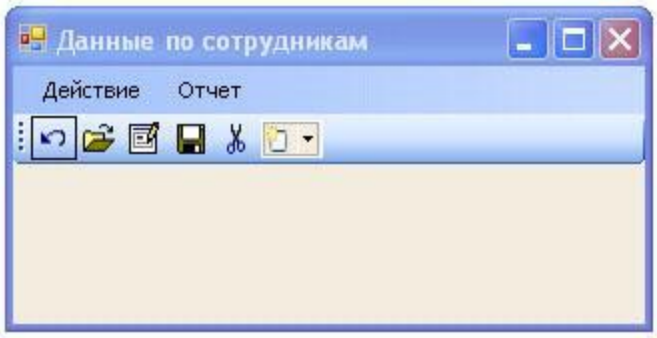

Рис. 5.6. Форма с панелью инструментов

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".

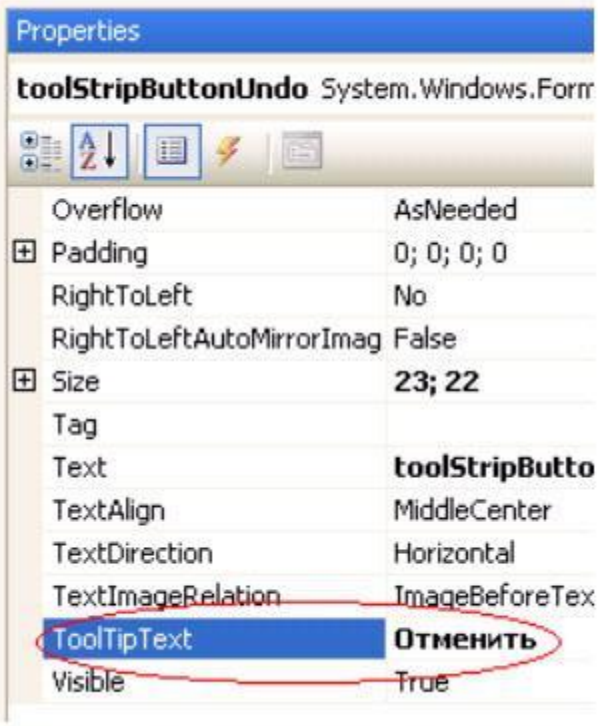

Рис. 5.7. Формирование подсказки для кнопки

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

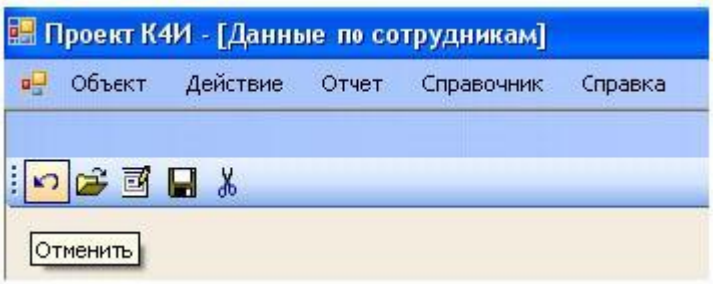

Рис. 5.8. Вывод подсказки для кнопки

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).

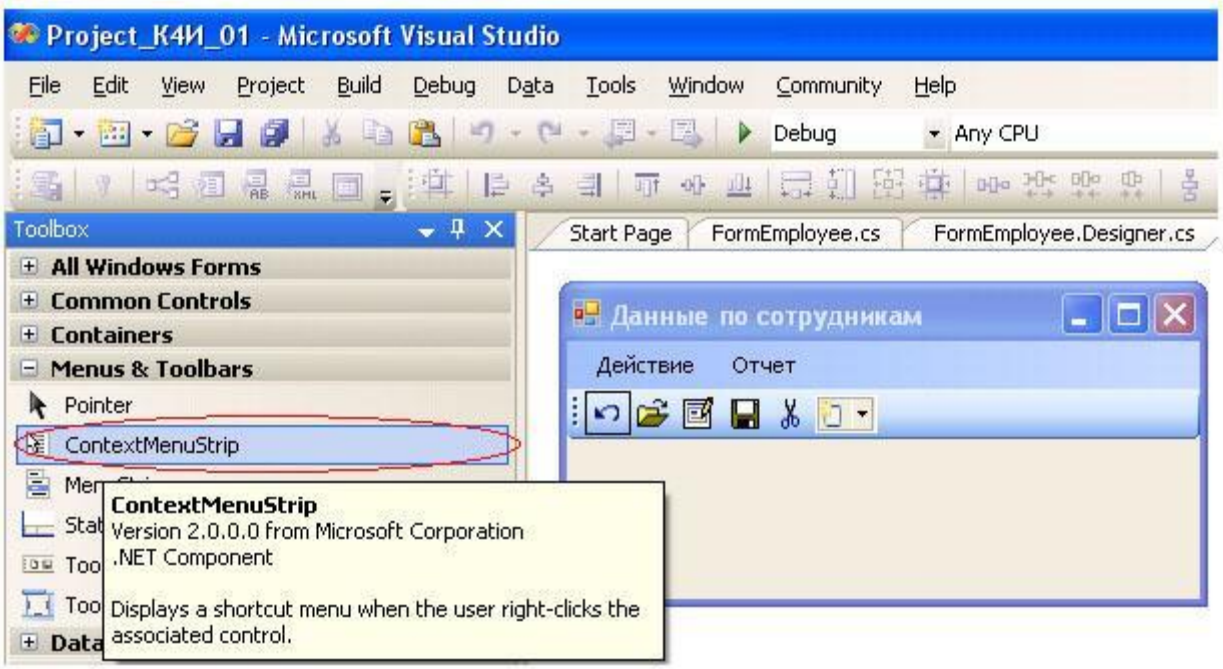

Рис. 5.9. Формирование на форме контекстного меню

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

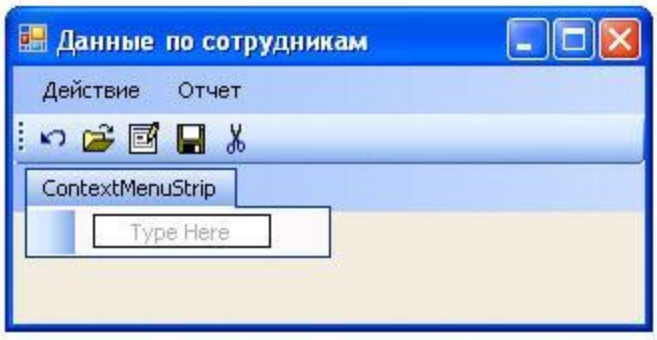

Рис. 5.10. Форма с контекстным меню

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

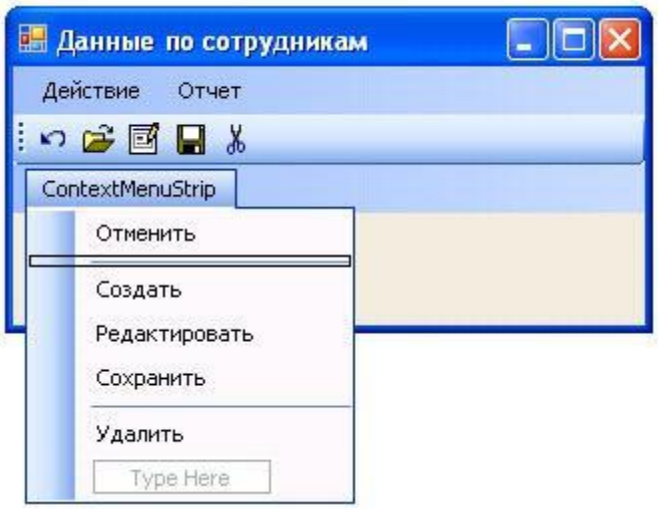

Рис. 5.11. Вид контекстного меню

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)

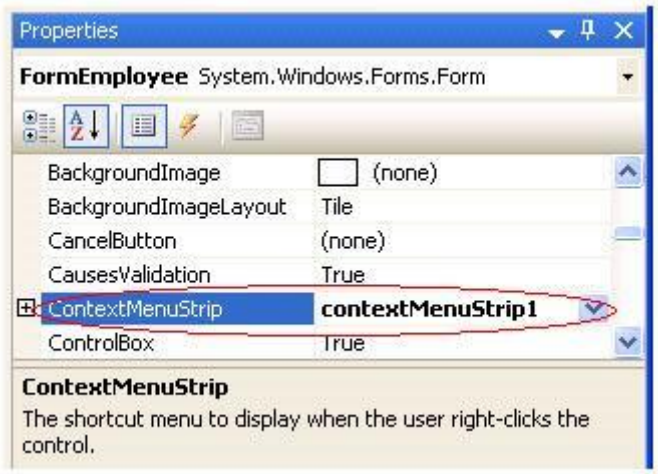

Рис. 5.12. Подключение контекстного меню к форме

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 всплытия на форме контекстного *меню* показан на рисунке 5.13.

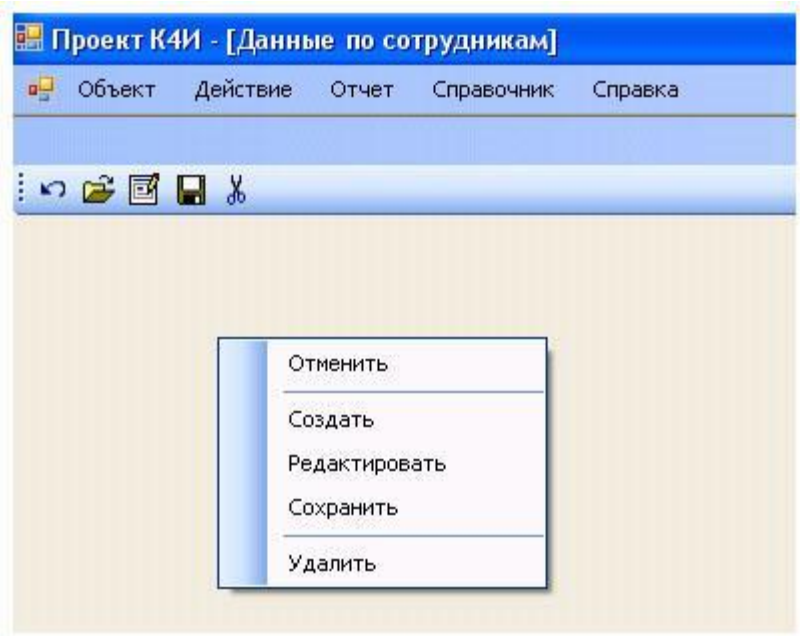

Рис. 5.13. Активизация контекстного меню

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 undo1ToolStripMenuItem_Click(object sender, EventArgs e)
{
   Undo();
}
```

Задание на лабораторную работу

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

6. Лабораторная работа: Создание строки состояния

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

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).

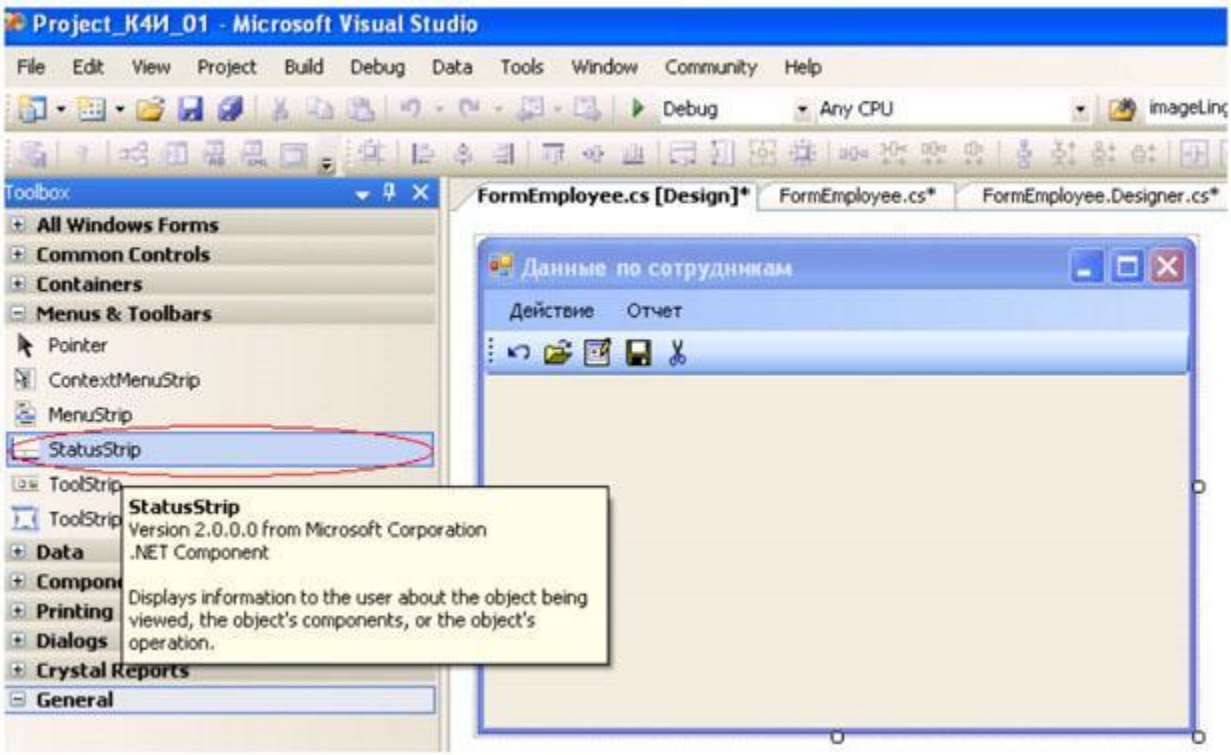

Рис. 6.1. Добавляем на форму строку состояния

Объекту класса StatusStrip присвоим имя statusStripEmployee. Откроем выпадающий *список* объекта класса statusBarEmployee и выберем *объект* StatusLable (рисунок 6.2). Присвоим ему имя toolStripStatusLabelEmployee.

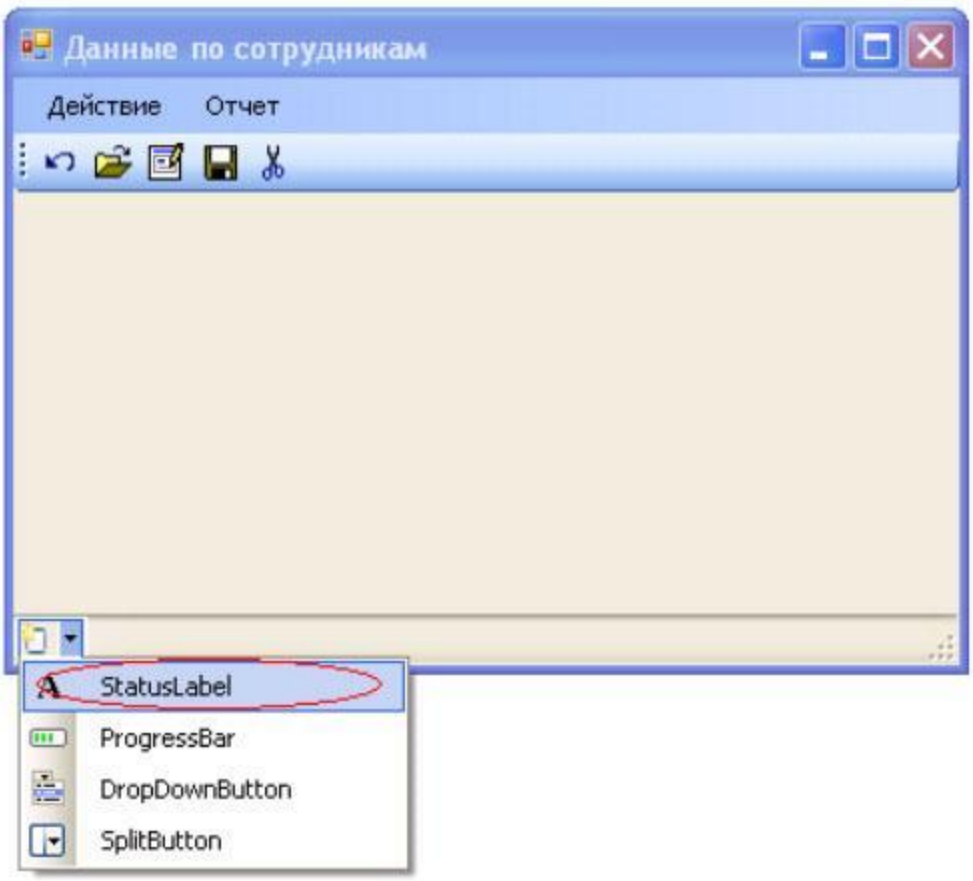

Рис. 6.2. Добавляем метку в строку состояния

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

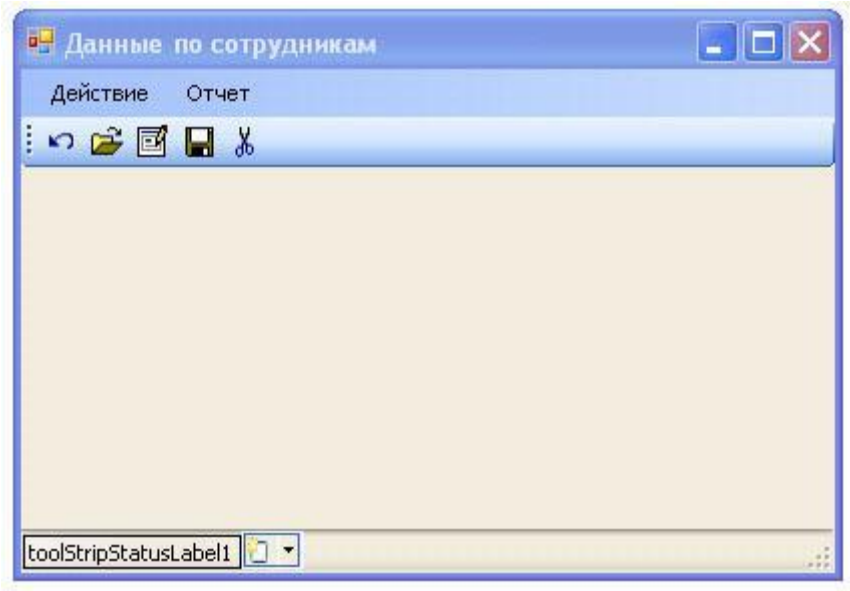

Рис. 6.3. Окно приложения со строкой состояния

Для управления текстом строки состояния необходимо разработать обработчик события для соответствующих объектов.

Для формы *FormEmployee* в строке состояний необходимо вывести информацию при наведении курсора мыши на пунктах *меню "Действие"*. Первоначально в дизайнере 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).

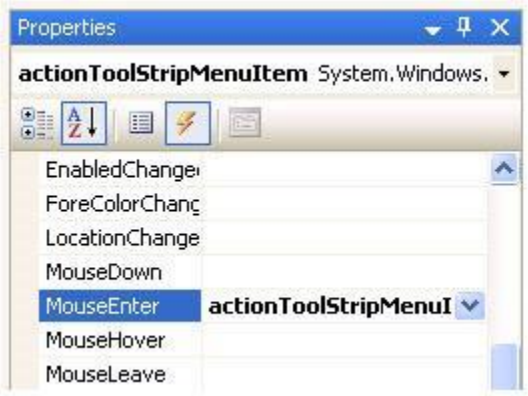

Рис. 6.4. Окно событий

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).

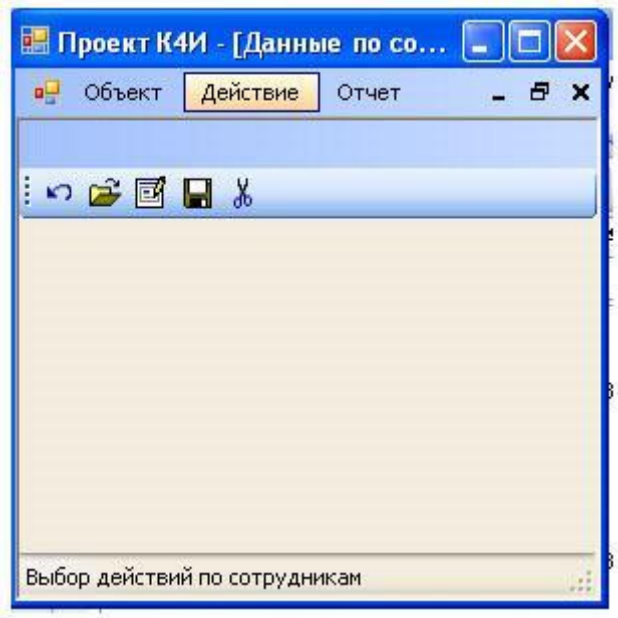

Рис. 6.5. Вывод сообщения в строке состояния

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.

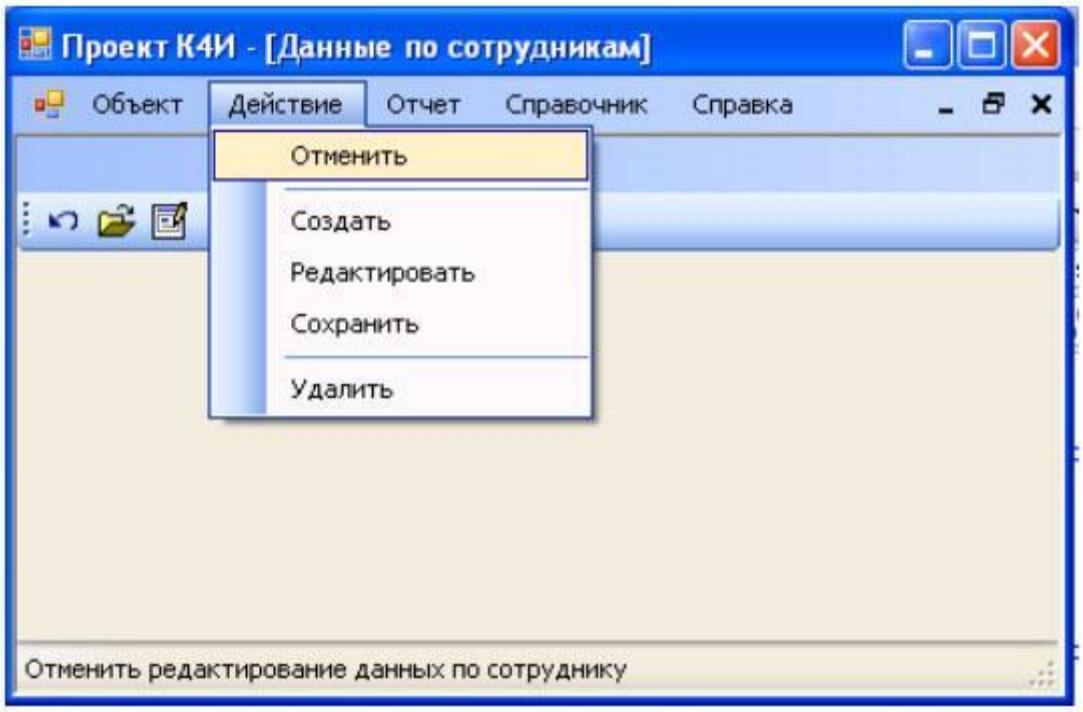

Рис. 6.6. Экранная форма со строкой состояния

Задание на лабораторную работу

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.

7. Лабораторная работа: Создание элементов управления

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

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 форме. *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).

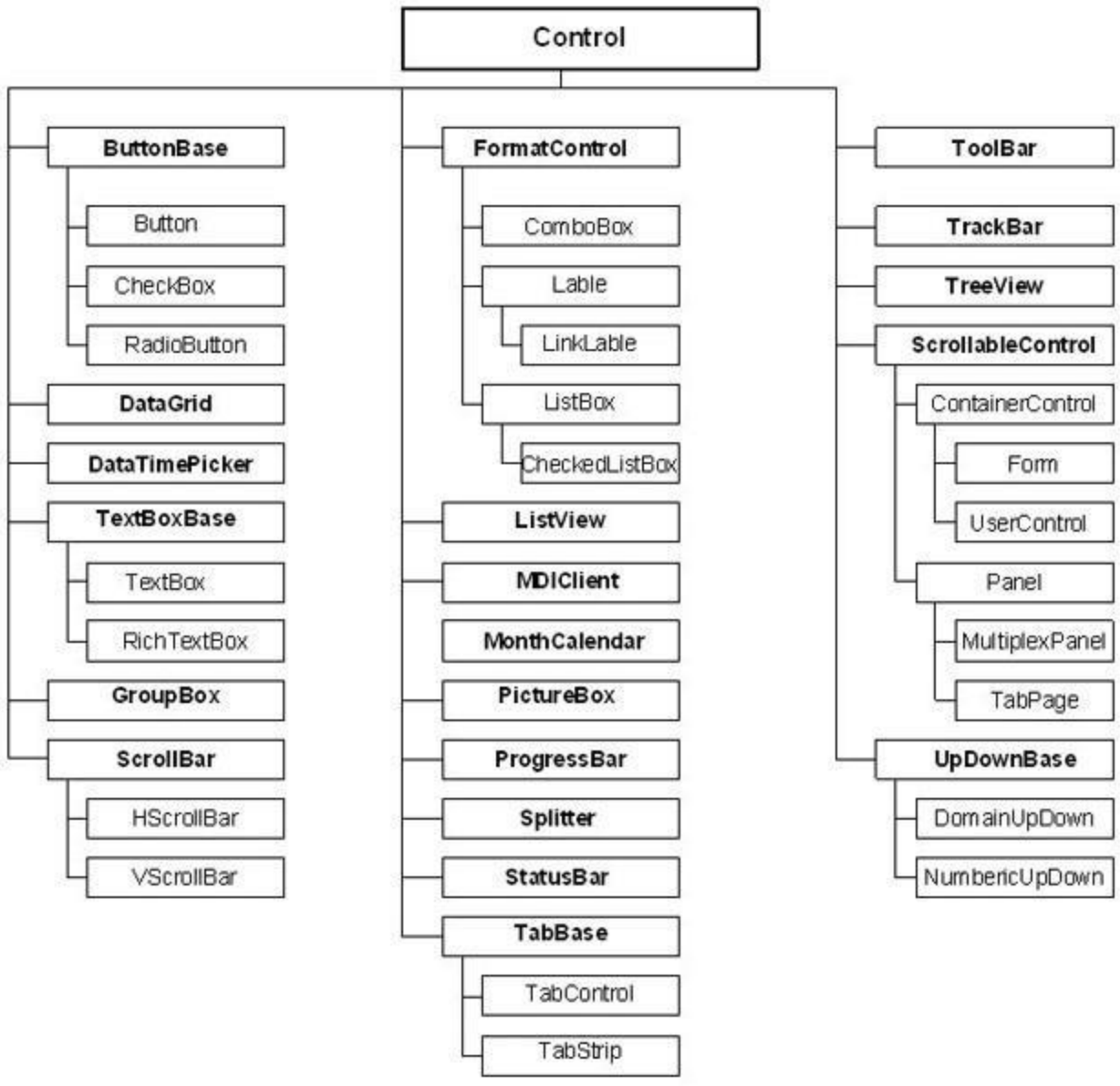

Рис. 7.1. Иерархия элементов управления Windows Forms

Все изменения, которые необходимо произвести в открывшемся окне (рисунок 7.2), будут добавлены в код метода InitializeComponents().

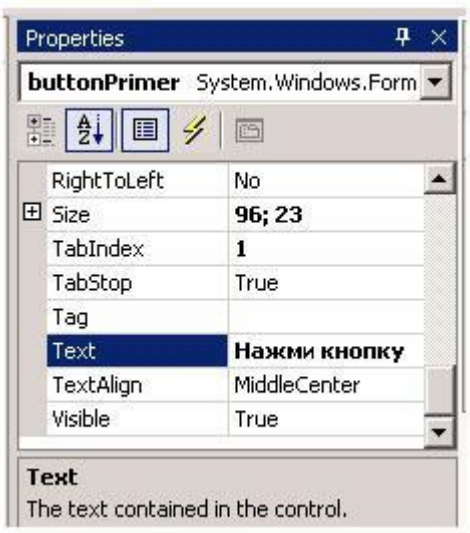

Рис. 7.2. Настройка элементов управления средствами Visual Studio

То же самое окно позволяет настроить не только свойства данного 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.

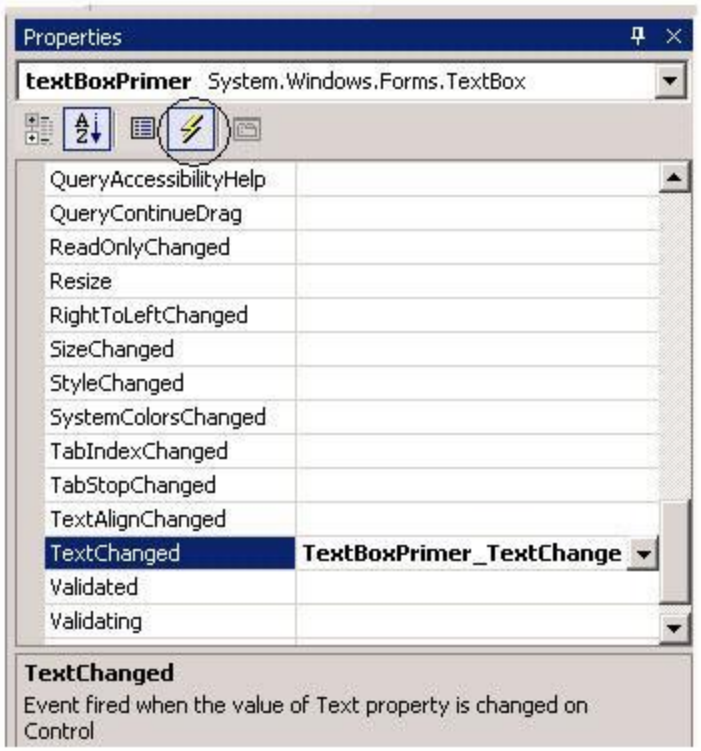

Рис. 7.3. Настройка обработчиков событий

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

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	Но 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
ScrollAlwaysVisible	Определяет, будет ли полоса прокрутки выводиться всегда
Selectedlndex	Индекс выделенного в настоящий момент элемента в списке (если такой имеется). Если ни один элемент не выделен, то возвращается значение -1
SelectedIndices	Набор индексов выделенных в настоящий момент элементов в списке. Если не выделен ни один элемент, то возвращается пустой набор
SelectedItem	Значение выделенного в настоящий момент элемента. Если ни один из элементов не выделен, то возвращается null
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
MaxDropDownltems	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 ComboBox
Selectedlndex	Определяет индекс выделенного элемента ComboBox. Если ни один элемент не выделен, возвращается значение -1
Selectedltem	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 Tablndex. 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 Tablndex:

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 AutoScrol1 объекта Panel значение true. В результате пользователь получит возможность доступа к "не вмещающимся" 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 AutomaicDelay
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 .

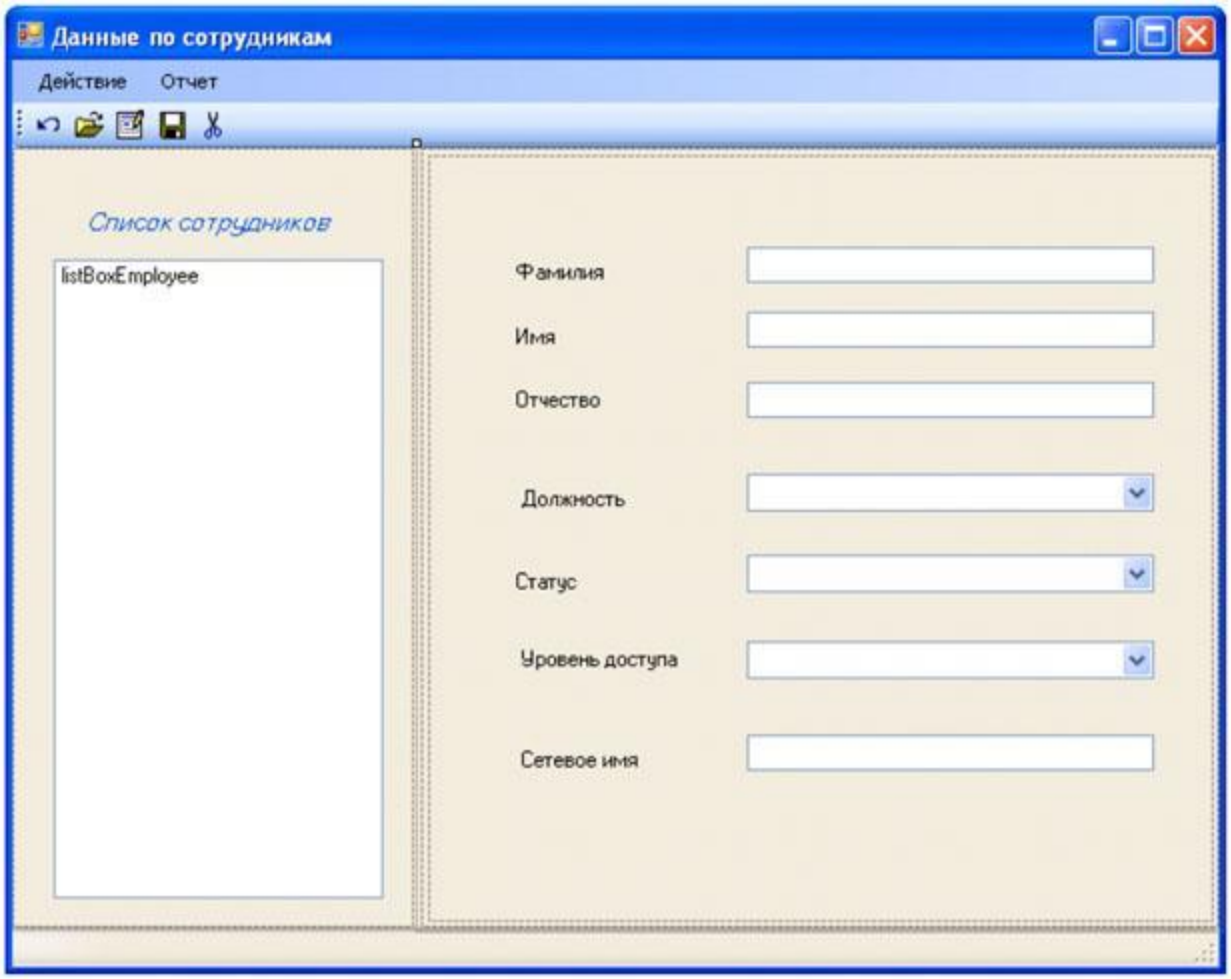

Рис. 7.4. Вид экранной формы FormEmployee

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 сотрудников
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, выходной, 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*.

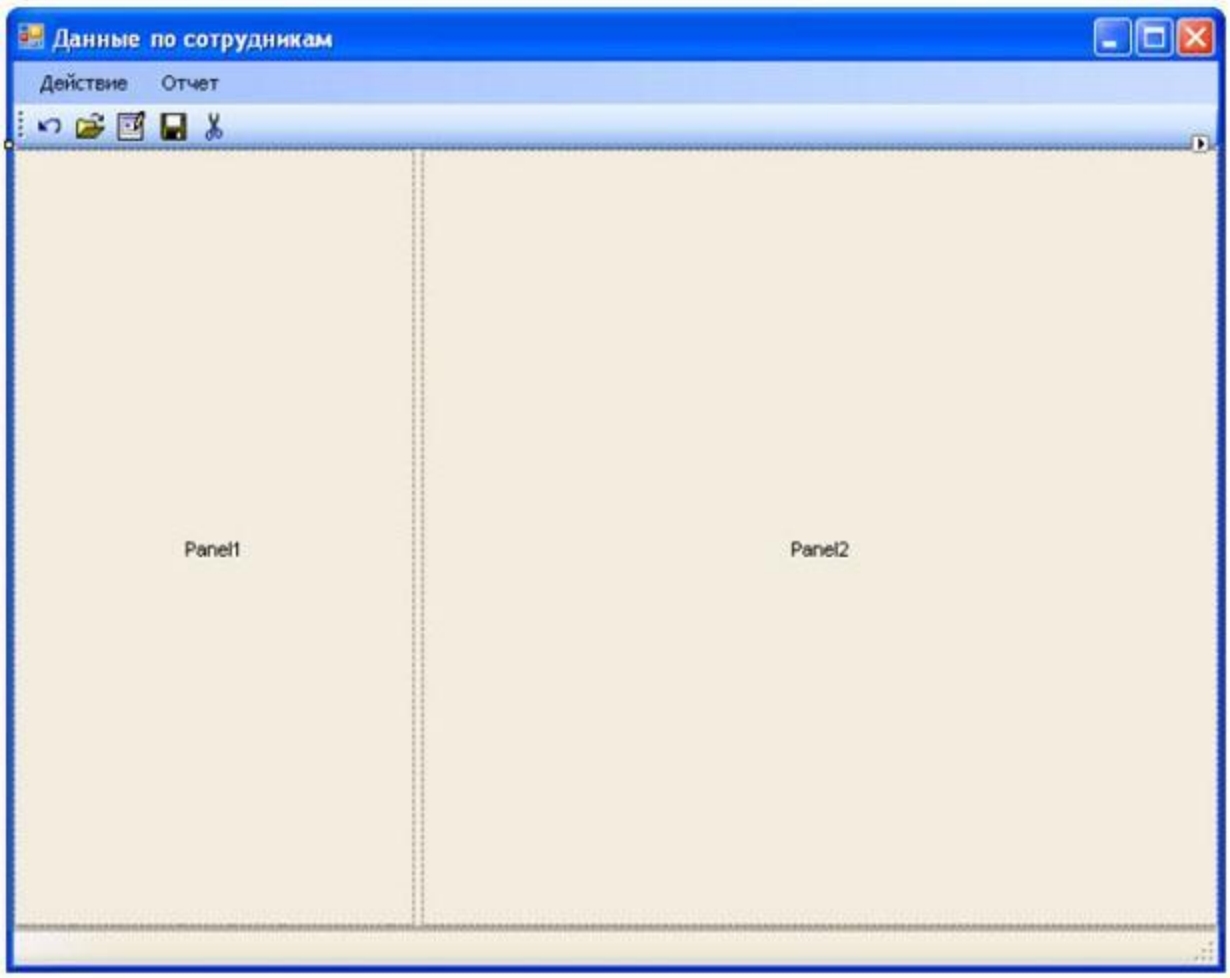

Рис. 7.5. Создание панелей на форме FormEmployee

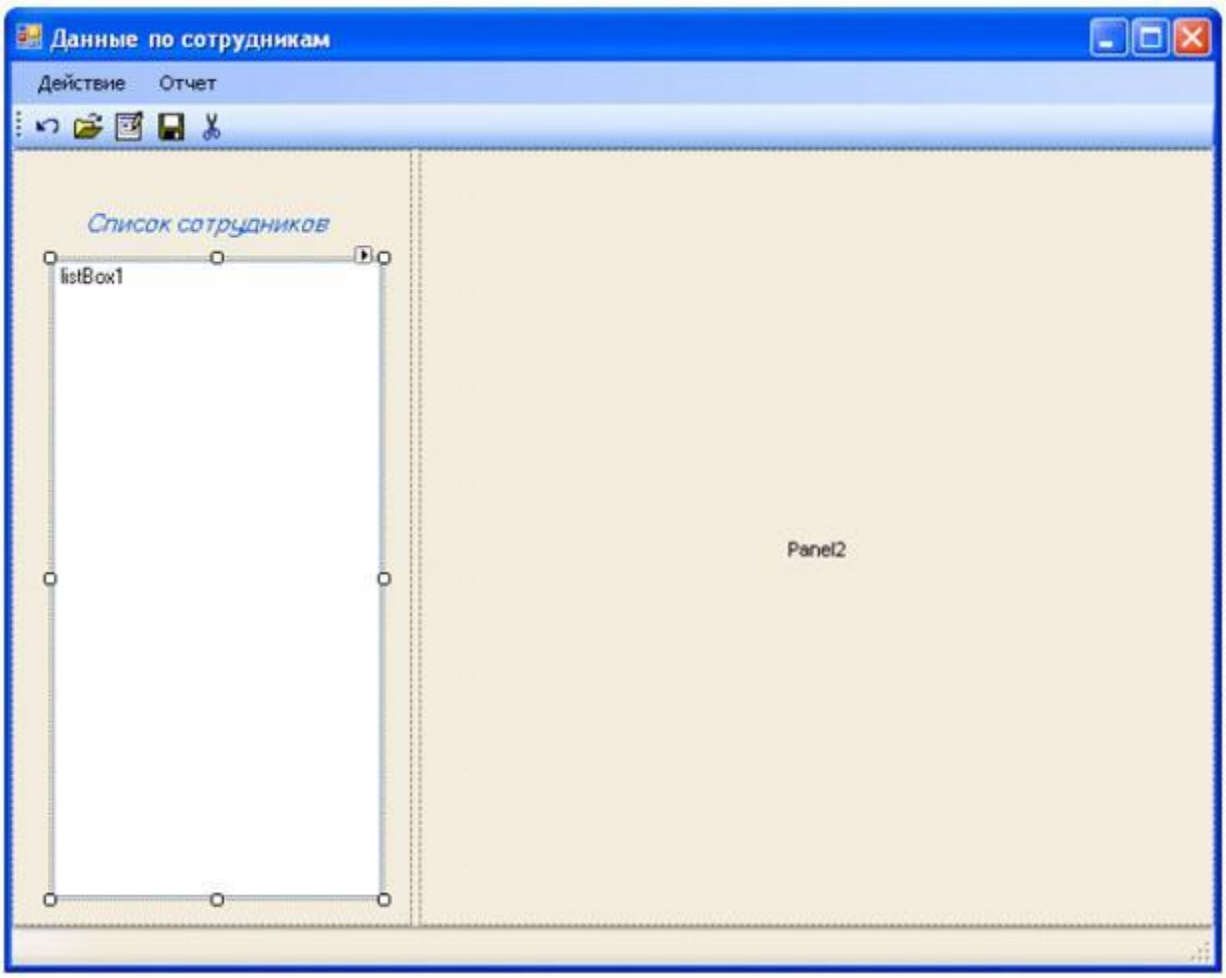

Рис. 7.6. Формирование элементов управления на панели 1 формы FormEmployee

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.

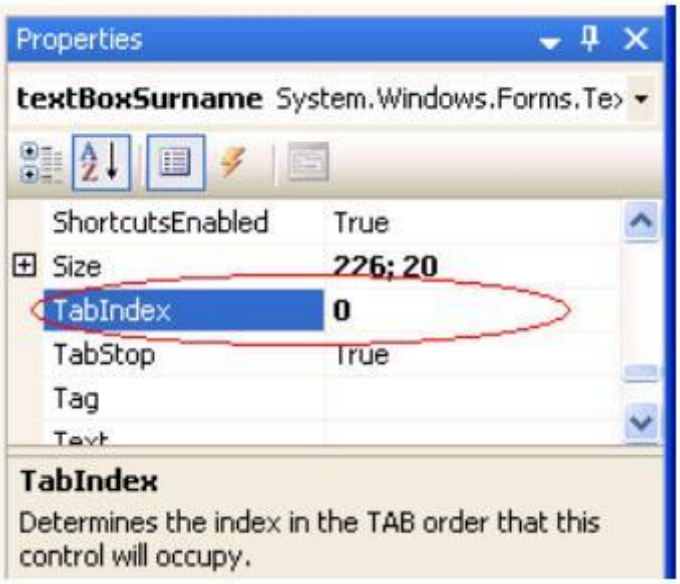

Рис. 7.7. Задание свойства TabIndex для элемента контроля

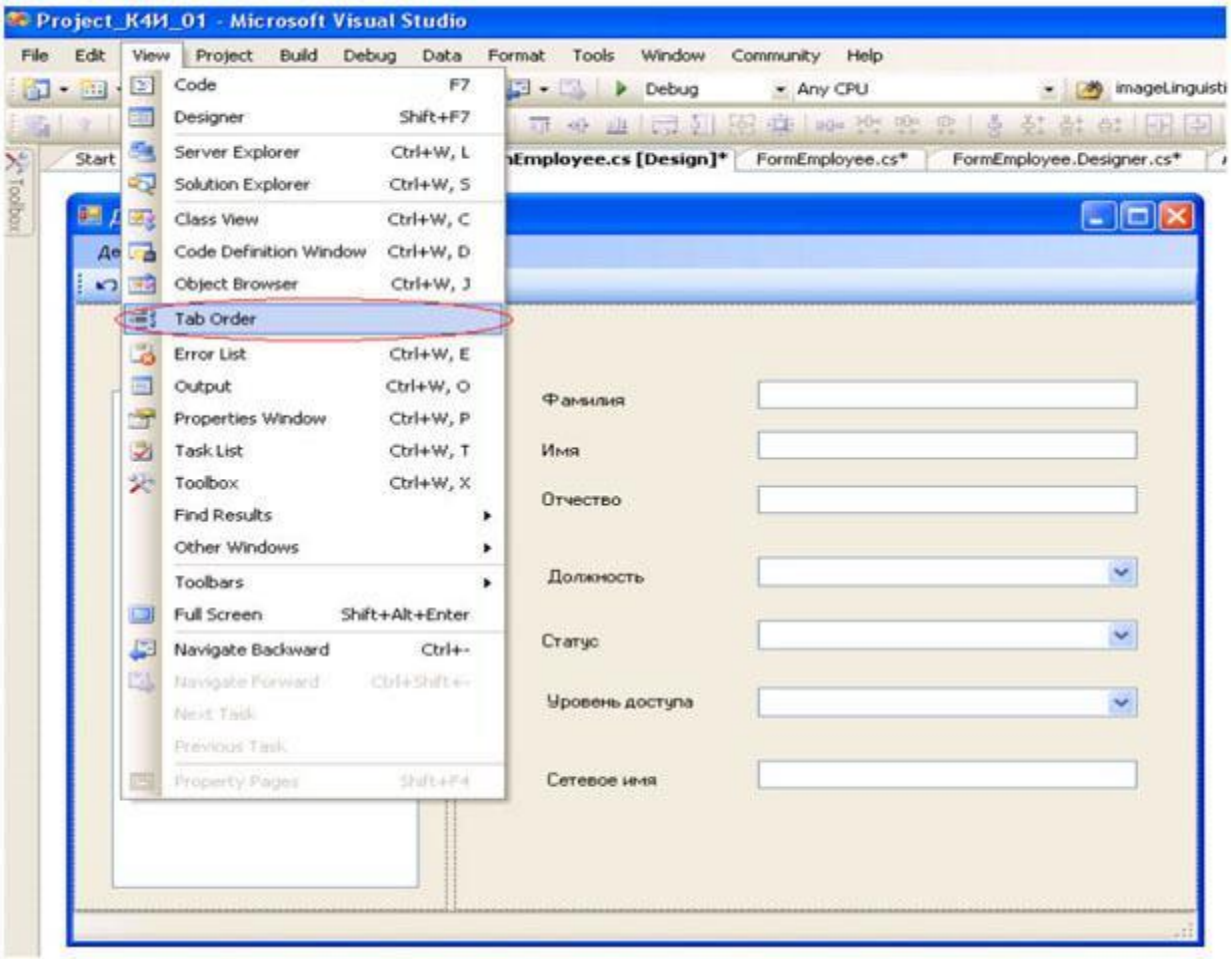

Рис. 7.8. Настройка перехода по элементам управления

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).

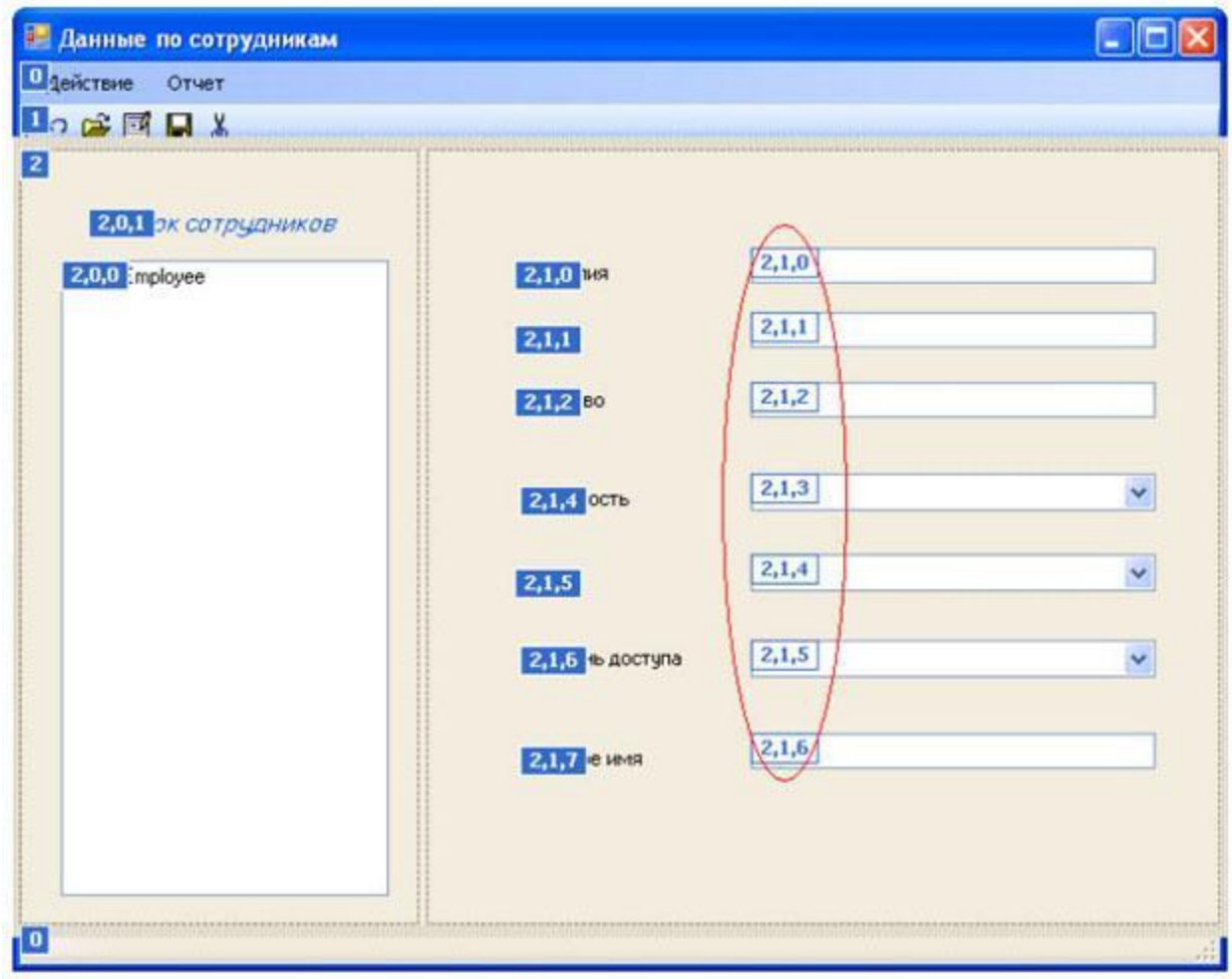

Рис. 7.9. Результат работы мастера Tab Order Wizard

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 параматр 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.

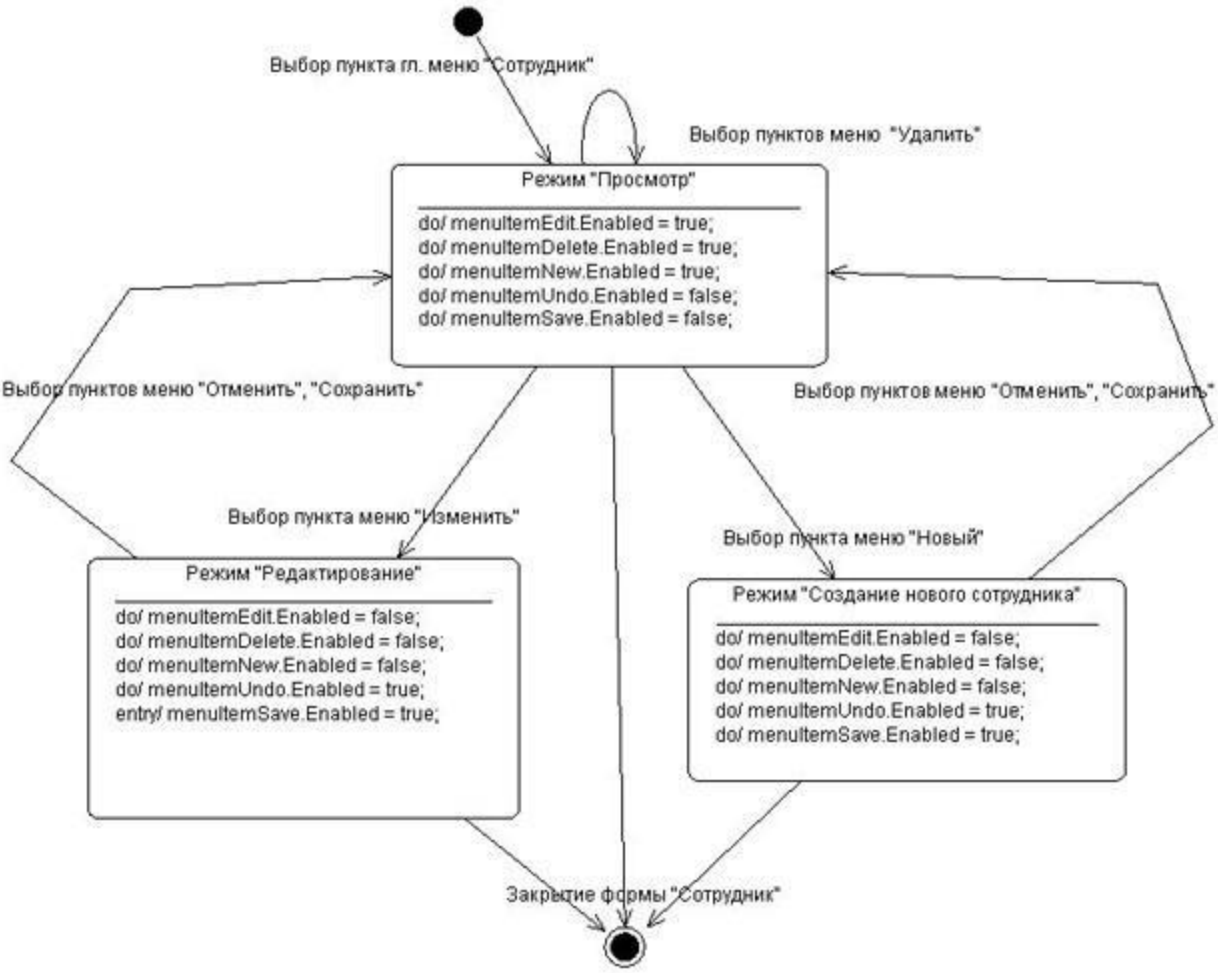

Рис. 7.10. Диаграмма состояний для активности подпунктов меню "Действие"

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 подпунктов 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*.

Задание на лабораторную работу

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.

8. Лабораторная работа: Подготовка ADO.NET к работе в приложении

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

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

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

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

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

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

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

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

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

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

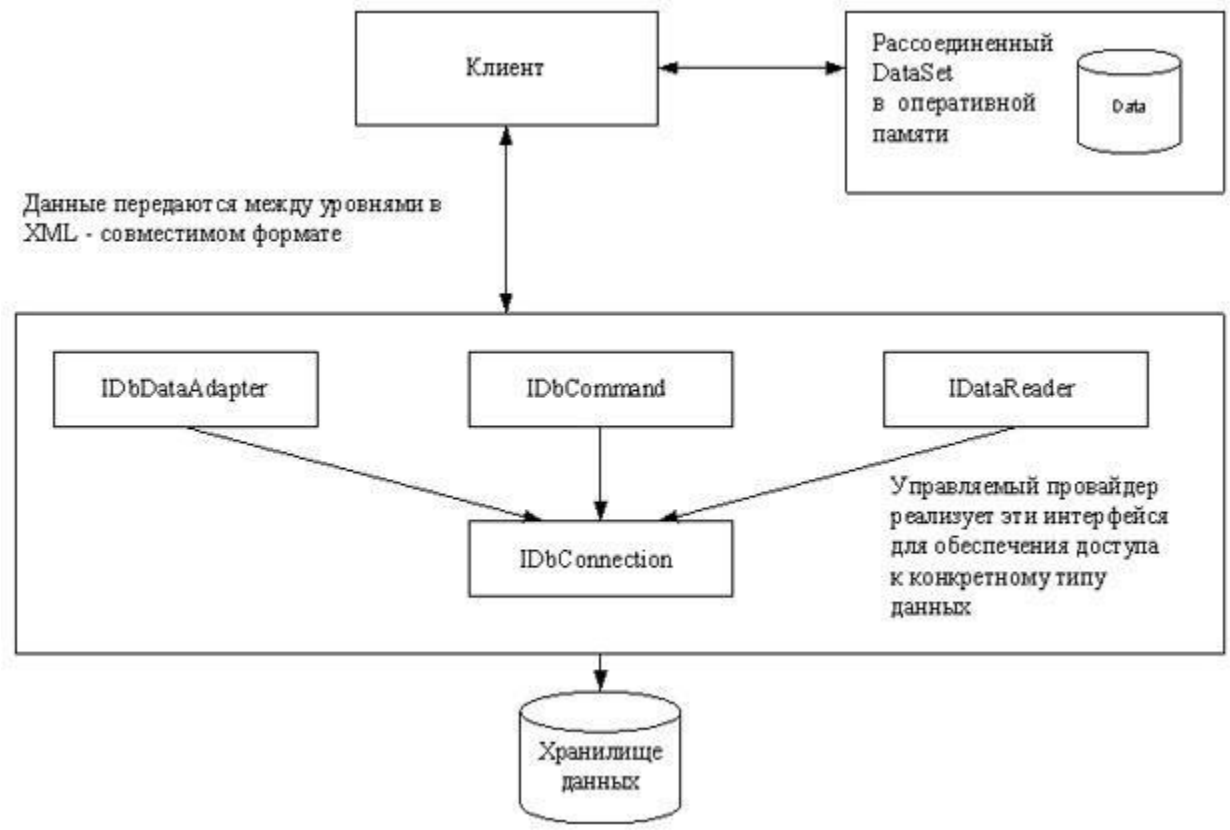

Рис. 8.1. Взаимодействие клиента с управляемыми провайдерами

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

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

Таблица 8.1. Пространства имен ADO.NET

Пространство имен	Описание
System.Data	Главное пространство имен ADO.NET. В нем определены типы, представляющие таблицы, столбцы, записи, ограничения и тип - DataSet.
System.Data.Common	Определены типы, общие для всех управляемых провайдеров. Многие из них выступают в качестве базовых классов для классов из пространств имен для провайдеров SQL и OleDb
System.Data.OleDb	В этом пространстве имен определены типы для установления соединений с OLE DB-совместимыми источниками данных, выполнения к ним SQL-запросов и заполнения

	данными объектов DataSet.
System.Data.SqlCIient	В этом пространстве имен определены типы, которые составляют управляемый провайдер SQL.
System.Data.SqlTypes	Представляют собой "родные" типы данных Microsoft SQL Server.

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

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

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

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

Таблица 8.2. Типы пространства имен System.Data

Тип	Назначение
DataColumnCollection, DataColumn	DataColumn представляет один столбец в объекте DataTable, DataColumnCollection - все столбцы
ConstraintCollection, Constraint	Constraint - объектно-ориентированная оболочка вокруг ограничения (например, внешнего ключа или уникальности), наложенного на один или несколько DataColumn, ConstraintCollection - все ограничения в объекте DataTable
DataRowCollection, Data Row	DataRow представляет единственную строку в DataTable, DataRowCollection - все строки в DataTable
DataRowView, Data View	DataRowView позволяет создавать настроенное представление единственной строки, DataView - созданное программным образом представление объекта DataTable, которое может быть использовано для сортировки, фильтрации, поиска, редактирования и перемещения
DataSet	Объект, создаваемый в оперативной памяти на клиентском компьютере. DataSet состоит из множества объектов DataTable и информации об отношениях между ними

ForeignKeyConstraint, UniqueConstraint	ForeignKeyConstraint представляет ограничение, налагаемое на набор столбцов в таблицах, связанных отношениями первичный - внешний ключ. UniqueConstraint - ограничение, при помощи которого гарантируется, что в столбце не будет повторяющихся записей
DataRelationCollection, 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.

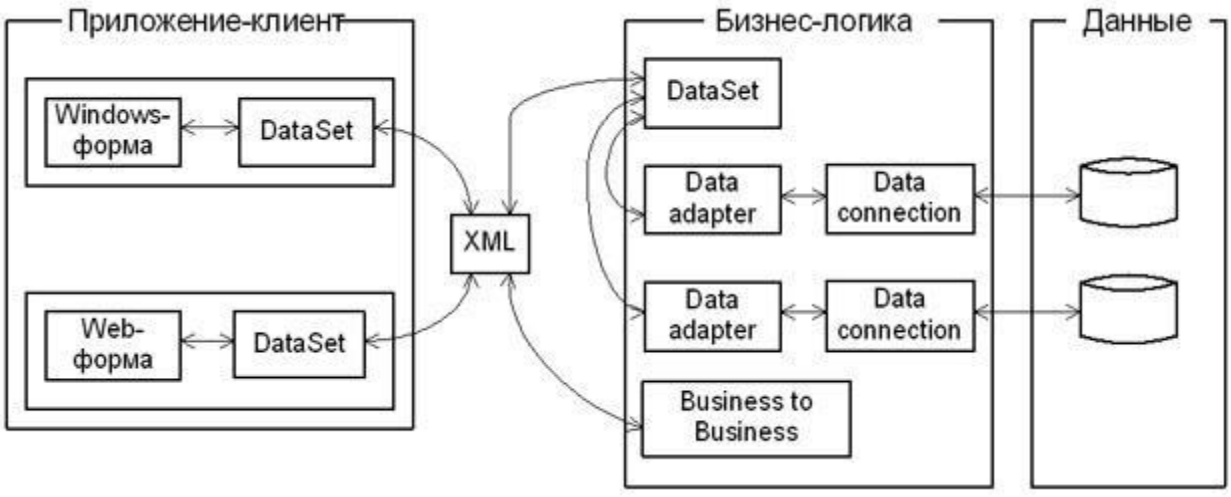

Рис. 8.2. Модель доступа к данным в ADO.NET

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 содержался *SQL -запрос* на выборку - SELECT, то *команда* может вернуть набор данных. Вы можете выбрать из *базы данных* только определенные строки и колонки, используя *объект* DataReader, который работает достаточно быстро, поскольку использует курсоры read-only, forward-only.

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

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

```
SELECT * FROM customers WHERE (customer_id=@customerid)
```

Значения параметров могут задаваться динамически, во *время выполнения* приложения.

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

Эту роль выполняет набор данных - DataSet, который представляет собой своеобразный *кэш* записей, извлеченных из базового источника. DataSet может состоять из одной или более таблиц, он имеет дело с копиями таблиц из *базы данных* источника. Кроме того, в данном объекте могут содержаться связи между таблицами и некоторые ограничения на выбираемые данные. Структура объекта DataSet приведена на рисунке 8.3.

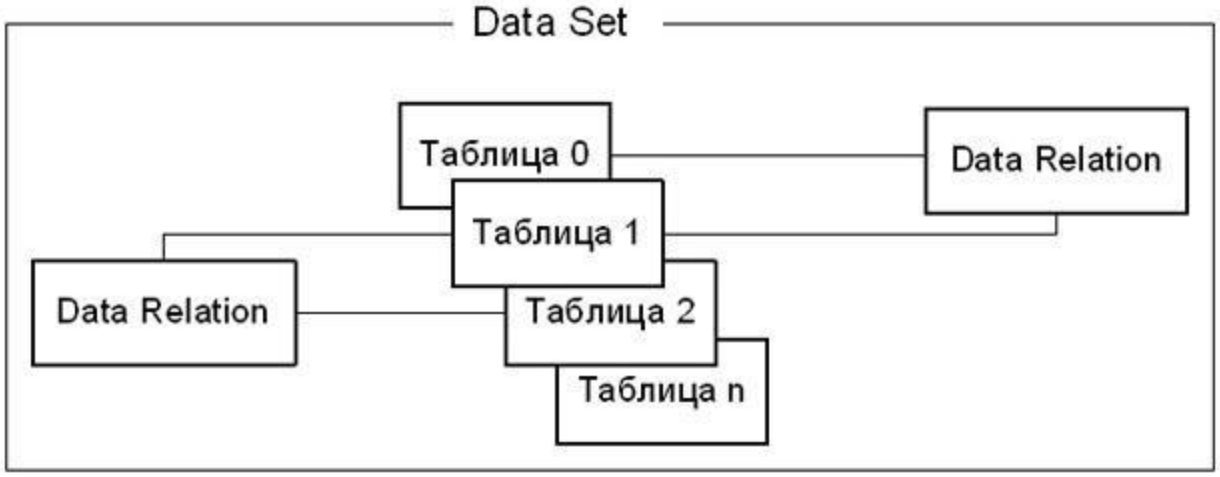

Рис. 8.3. Структура объекта DataSet

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

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

Важно отметить то, что DataSet - *пассивный контейнер* для данных, который обеспечивает только их хранение. Что же нужно поместить в этот *контейнер*, определяется в другом объекте - адаптере данных DataAdapter. В адаптере данных содержатся одна или более команд, которые определяют, какую информацию нужно поместить в таблицы объекта DataSet, *по* каким правилам нужно синхронизировать информацию в конкретной таблице DataSet и соответствующей таблицей основной *базы данных* и т. п. *Адаптер* данных обычно содержит четыре команды SELECT, INSERT, UPDATE, DELETE, для выборки, добавления, корректировки и удаления записей.

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

```
SELECT au_id, au_lname, au_fname FROM authors
```

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

Информация о базе данных

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

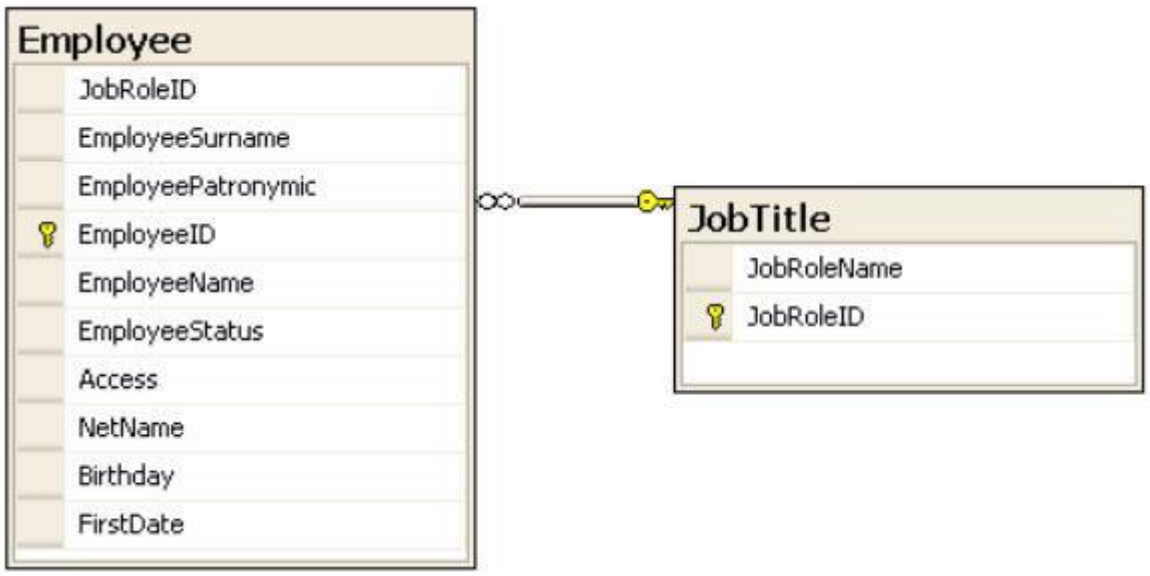

Рис. 8.4. Структура базы данных по сотрудникам компании

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 появляется следующее сообщение (рисунок 8.6).

Рис. 8.6. Окно Microsoft Data Link

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

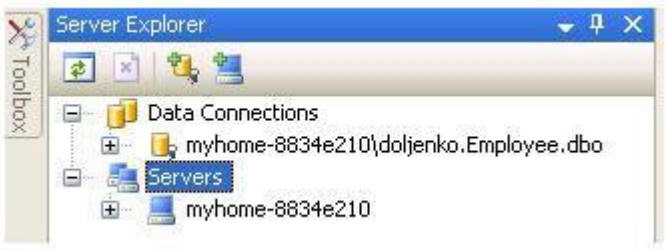

Рис. 8.7. Окно навигатора Server Explorer

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

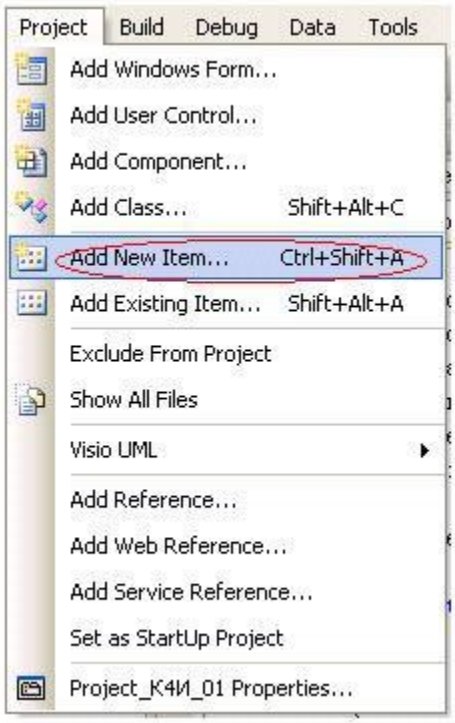

Рис. 8.8. Добавление в проект нового компонента

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

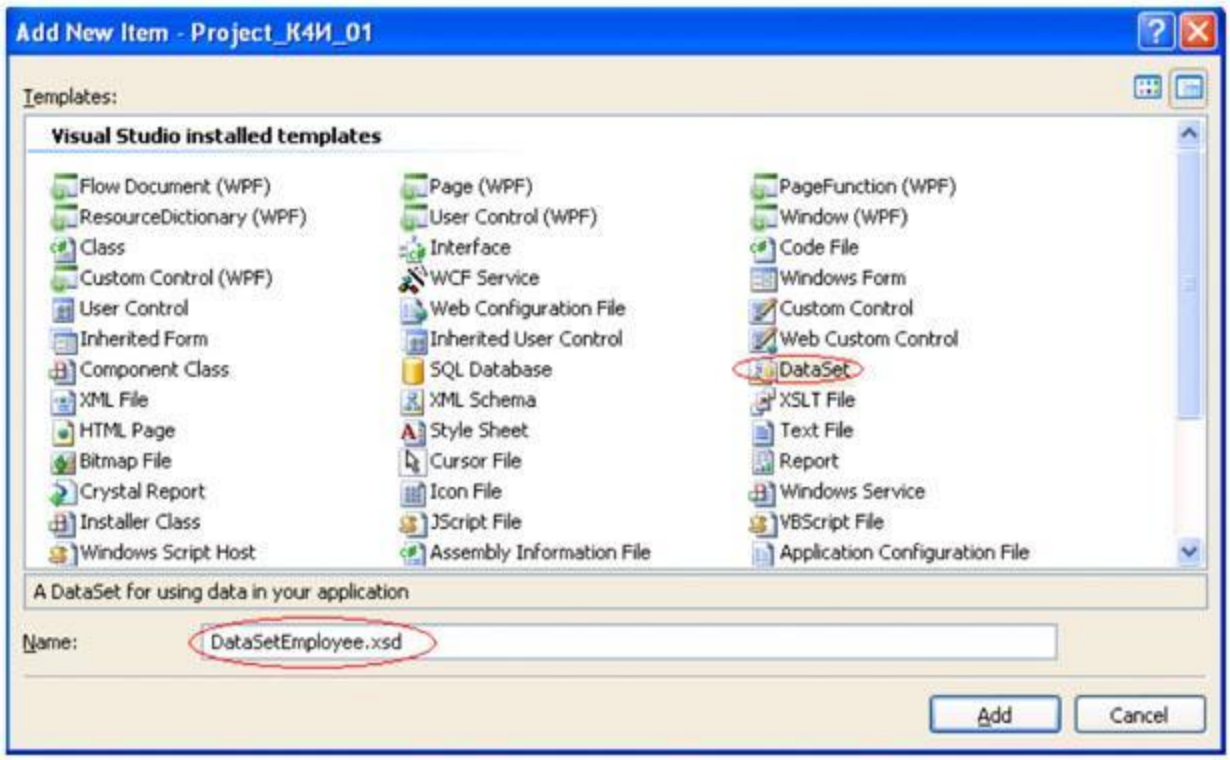

Рис. 8.9. Выбор нового компонента - DataSet

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

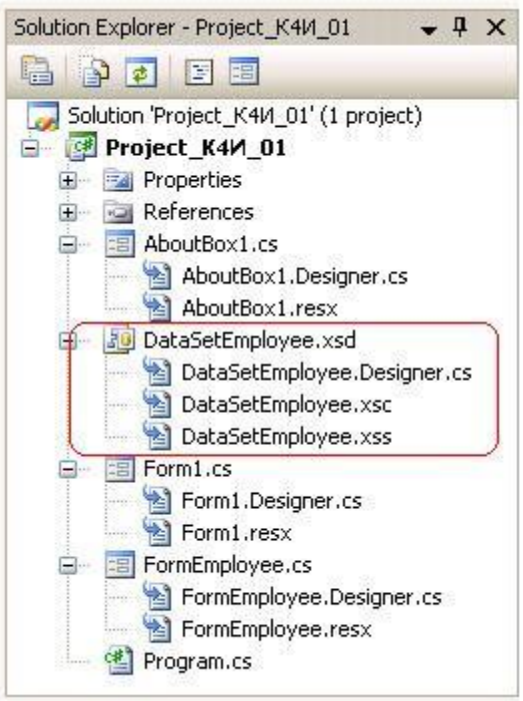

Рис. 8.10. Окно решения проекта с новым компонентом DataSet

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

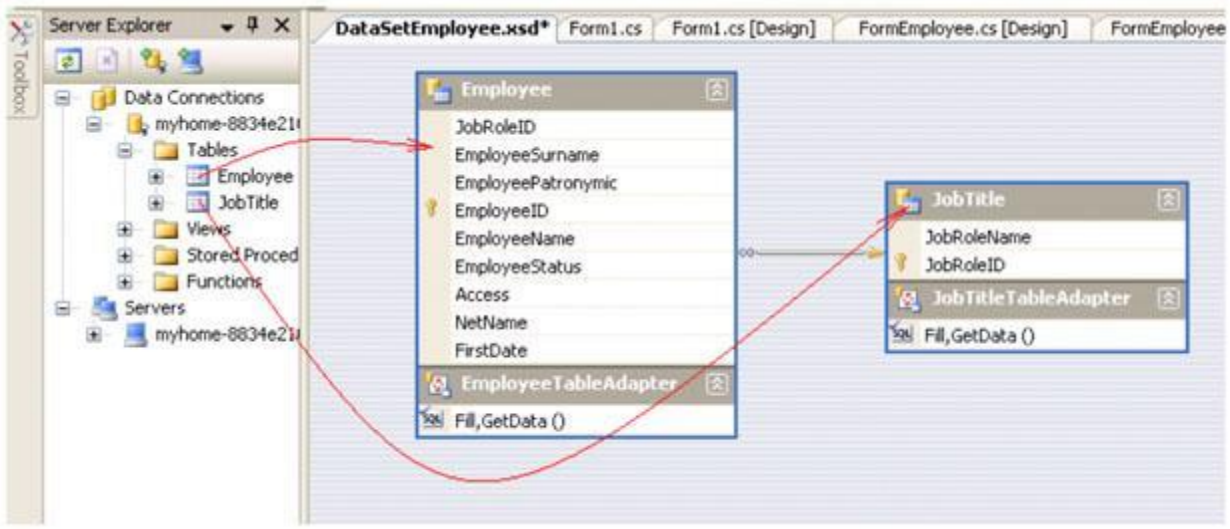

Рис. 8.11. Добавление таблиц к DataSet

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.

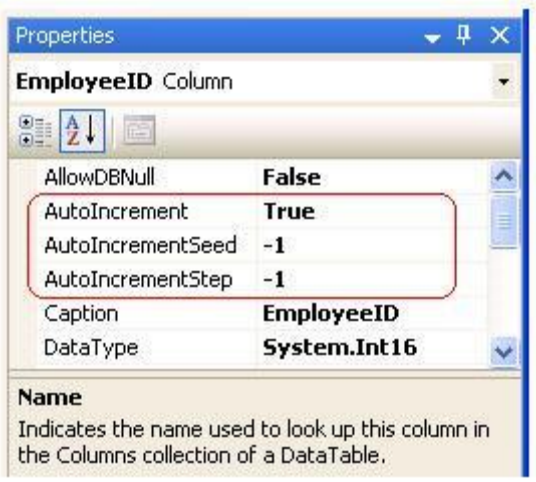

Рис. 8.12. Установка свойств для колонки EmployeeID

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_К4И_01.DataSetEmployeeTableAdapters.
EmployeeTableAdapter();
DataSetEmployeeTableAdapters.JobTitleTableAdapter daJobTitle =
 new Project_К4И_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 отработал");}
```

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

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

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

Задание на лабораторную работу

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 заполнение таблиц *DataSet*.
6. Протестировать работу приложения.
7.

4.1.2. Тестирование

Тема 1. Объектно-ориентированный подход к проектированию и разработке программ
Тема 2. Визуальное программирование
Тема 3. Разработка графического интерфейса пользователя
Тема 4. Развитые элементы приложений

4.1.2.1. Порядок проведения.

Тестирование проходит в письменной форме или с использованием компьютерных средств. Обучающийся получает определённое количество тестовых заданий. На выполнение выделяется фиксированное время в зависимости от количества заданий. Оценка выставляется в зависимости от процента правильно выполненных заданий. Тестирование проводится по вариантам. В каждом варианте – 10 тестовых заданий. За каждый правильный ответ начисляется 1 балл. Итого за тестирование студент может заработать до 10 баллов.
Ниже приведены примерные задания. Полный банк тестовых заданий хранится на кафедре.

4.1.2.2 Критерии оценивания

17-20 баллов ставится, если обучающийся набрал 86% правильных ответов и более.
14-16 баллов ставится, если обучающийся набрал от 71% до 85 % правильных ответов.
11-15 баллов ставится, если обучающийся набрал от 56% до 70% правильных ответов.
0--10 баллов ставится, если обучающийся набрал 55% правильных ответов и менее.

4.1.2.3. Содержание оценочного средства

1. Для чего осуществляется совершенствование процесса разработки ПО?
a) для улучшения качества создаваемых продуктов
b) для снижения цены разработки
c) для уменьшения сложности ПО
d) для уменьшения времени разработки
2. В каком случае приведен пример использования стратегии technology push?
(Ответ считается верным, если отмечены все правильные варианты ответов.)
a) внедрение стандартов качества ISO 9000 или CMMI
b) внедрение новых средств тестирования в ситуации, когда заказчик не удовлетворен качеством программной
системы
c) переход компании со средств структурной разработки на объектно-ориентрованные
3. При использовании какой стратегии изменения, вносимые в процесс, более глобальны?
(Отметьте один правильный вариант ответа.)
a) technology push
b) в обоих случаях изменения одинаковы
c) organization pull
4. Что такое вид деятельности?
(Отметьте один правильный вариант ответа.)
a) структура, согласно которой построена разработка ПО
b) определенный тип работы, выполняемый в процессе разработки ПО
c) определенный этап процесса, имеющий начало, конец и выходной результат
5. Какая стратегия нацелена на решение конкретных проблем компании?
a) technology push
b) обе стратегии
c) organization pull
6. К информатике относятся:
a. разработка встроенных систем реального времени
b. математическая логика
c. теория грамматик
d. методы построения компиляторов
7. Что может являться рабочим продуктом в процессе разработки ПО?
8. Что такое бизнес-реинжиниринг?
a. процесс модернизации программного обеспечения согласно требованиям заказчика

b. модернизация бизнеса в определенной компании
c. свод теоретических наук, основанных на математике и посвященных формальным основам вычислимости

9. В чем заключается согласованность ПО?
a) В том, что ПО должно быть согласовано с большим количеством интерфейсов
b) в том, что ПО основывается на объективных посылках
c) в согласованности заказчика и исполнителя
10. Какие утверждения верны для водопадной модели?
a) допускается возврат только на предыдущий шаг
b) не ограничена возможность возвратов на произвольный шаг назад
c) в рамках водопадной модели было введено прототипирование
Правильные ответы: 1-c, 2-a, 3-b, 4-d, 5-a, 6-a, 7-d, 8-b, 9-c, 10-d

1. Что такое фаза разработки?
(Отметьте один правильный вариант ответа.)
Вариант 1 структура, согласно которой построена разработка ПО
Вариант 2 определенный этап процесса, имеющий начало, конец и выходной результат
Вариант 3 выходной результат определенного этапа процесса
Вариант 4 определенный тип работы, выполняемый в процессе разработки ПО
2. Какие возвраты невозможны при разработке по водопадной модели?
(Ответ считается верным, если отмечены все правильные варианты ответов.)
Вариант 1 возврат от кодирования к разработке системных требований
Вариант 2 возврат от тестирования к кодированию
Вариант 3 возврат от тестирования к анализу
3. Какова цель создания прототипа при использовании водопадной модели?
(Ответ считается верным, если отмечены все правильные варианты ответов.)
Вариант 1 обоснованное принятие главных архитектурных решений
Вариант 2 уменьшение рисков разработки
Вариант 3 определение основных рисков
4. В какой модели каждый виток представляет собой фазу разработки?
(Отметьте один правильный вариант ответа.)
a) ни в одной из указанных моделей
b) в водопадной модели
c) в любой модели
d) в спиральной модели
5. К какому типу проектов относятся проекты по разработке ПО?
(Отметьте один правильный вариант ответа.)
Вариант 1 к творческим проектам
Вариант 2 к промышленным проектам
Вариант 3 и к творческим, и к промышленным проектам
6 Какое свойство зависит от размера программных объектов?
(Отметьте один правильный вариант ответа.)
a) изменяемость
b) нематериальность
c) согласованность
d) сложность
7. Какие виды деятельности включает в себя процесс создания ПО?
Вариант 1 только разработка программного кода
Вариант 2 разработка проектной документации
Вариант 3 разработка тестов
Вариант 4 разработка проектных планов
8. Отметьте верные утверждения:
(Ответ считается верным, если отмечены все правильные варианты ответов.)
a) один вид деятельности может выполняться на разных фазах
b) в рамках одной фазы может выполнятся несколько различных видов деятельности
c) одному виду деятельности соответствует одна фаза

9. Какие возвраты возможны при разработке по водопадной модели?

(Ответ считается верным, если отмечены все правильные варианты ответов.)

a) возврат от тестирования к анализу

b) возврат от кодирования к проектированию

c) возврат от тестирования к кодированию

10. Какова цель создания прототипа при использовании водопадной модели?

(Ответ считается верным, если отмечены все правильные варианты ответов.)

a) уменьшение рисков разработки

b) определение основных рисков

c) обоснованное принятие главных архитектурных решений

Правильные ответы: 1-c, 2-a, 3-b, 4-d, 5-a, 6-a, 7-d, 8-b, 9-c, 10-d

4.2. Оценочные средства промежуточной аттестации

По дисциплине предусмотрен экзамен. Он проходит по билетам. В каждом билете содержится один теоретический вопрос и одна задача. Экзамен проводится в устной, письменной или компьютерной форме. Оценивается владение материалом, его системное освоение, способность применять нужные знания, навыки и умения при анализе проблемных ситуаций и решении практических заданий.

4.2.1. Устный ответ

4.2.1.1. Порядок проведения.

Промежуточная аттестация нацелена на комплексную проверку освоения дисциплины. Обучающийся получает вопрос(ы)/задание(я) и время на подготовку. Промежуточная аттестация проводится в устной, письменной или компьютерной форме. Оценивается владение материалом, его системное освоение, способность применять нужные знания, навыки и умения при анализе проблемных ситуаций и решении практических заданий.

4.2.1.2. Критерии оценивания.

18-20 баллов ставится, если обучающийся:

В ответе качественно раскрыл содержание темы. Ответ хорошо структурирован. Прекрасно освоен понятийный аппарат. Продемонстрирован высокий уровень понимания материала. Превосходное умение формулировать свои мысли, обсуждать дискуссионные положения.

14-17 баллов ставится, если обучающийся:

Основные вопросы темы раскрыл. Структура ответа в целом адекватна теме. Хорошо освоен понятийный аппарат. Продемонстрирован хороший уровень понимания материала. Хорошее умение формулировать свои мысли, обсуждать дискуссионные положения.

11-13 баллов ставится, если обучающийся:

Тему частично раскрыл. Ответ слабо структурирован. Понятийный аппарат освоен частично. Понимание отдельных положений из материала по теме. Удовлетворительное умение формулировать свои мысли, обсуждать дискуссионные положения.

0--10 баллов ставится, если обучающийся:

Тему не раскрыл. Понятийный аппарат освоен неудовлетворительно. Понимание материала фрагментарное или отсутствует. Неумение формулировать свои мысли, обсуждать дискуссионные положения.

4.2.1.3. Оценочные средства.

Вопросы для устного или письменного ответа

1. Перечислите характеристики ПО по Бруксу и кратко характеризуйте каждую.
2. С какими иными видами человеческой деятельности соотносится создание ПО в данном разделе?
3. Что такое процесс создания ПО?
4. Расскажите о причинах отсутствия универсального процесса разработки ПО.
5. Почему возможно и целесообразно стандартизировать процесс на уровне компании?
6. Что такое стандартный и конкретный процессы и как они соотносятся?
7. Чем отличаются между собой текущий и конкретный процессы? Какие методологии разработки ПО поддерживают понятие конкретного процесса и какими средствами?
8. Дайте определение деятельности по совершенствованию процесса.
9. В чем главная трудность совершенствования процессов в компаниях?
10. Перечислите основные направления улучшения процесса.
11. Расскажите о стратегии organization pull к внедрению инноваций. Приведите примеры.
12. Расскажите о стратегии technology push к внедрению инноваций. Приведите примеры.
13. Расскажите о достоинствах, недостатках, а также возможных рисках этих стратегий.
14. Что такое модель процесса?
15. Что такое фаза процесса?
16. Что такое вид деятельности?
17. Почему нельзя отождествлять фазы и виды деятельности? Когда и по каким причинам это все таки происходит на практике?

18. В чем достоинства водопадной модели? В чем ее историческая роль? В чем ее недостатки?
19. Как в рамках водопадной модели предполагается работать с рисками?
20. Почему водопадная модель до сих пор используется? Объясните, почему эту модель удобно использовать в оффшорных проектах с почасовой оплатой?
21. Чем виток спиральной модели отличается от фазы в водопадной модели? Приведите пример последовательности витков спиральной модели. Опишите условия, при которых спираль завершается.
22. Расскажите про второе и третье измерение спиральной модели. Опишите различные секторы витка спирали.
23. В чем достоинства и недостатки спиральной модели? Каковы ограничения этой модели?
24. Как в рамках этой модели предполагается работать с рисками?
25. В чем трудность управления требованиями? При ответе на этот вопрос имейте в виду другие инженерные области и сферы бизнеса. Старайтесь отвечать на вопрос с наружи программной инженерии, а не изнутри.
26. Перечислите способы формализации требований. Под формализацией имеется в виду способ не промежуточной, а финальной фиксации.
27. Расскажите о способах и техниках "вытягивания" требований.
28. Перечислите разные виды документов, формализующих требования.
29. Расскажите об отличии функциональных и нефункциональных требований.
30. Расскажите о типовом цикле работы с требованиями.

4.2.2. Практическое задание

4.2.2.1. Порядок проведения.

Практические навыки проверяются путём выполнения обучающимися практических заданий в условиях, полностью или частично приближенных к условиям профессиональной деятельности. Проверяется знание теоретического материала, необходимое для правильного совершения необходимых действий, умение выстроить последовательность действий, практическое владение приёмами и методами решения профессиональных задач.

4.2.2.2. Критерии оценивания

27-30 баллов ставится, если обучающимся:
Задание выполнено полностью и правильно.
22-26 баллов ставится, если обучающимся:
Задание выполнено полностью, но нет достаточного обоснования. Или при верном решении допущена ошибка или недочет, не влияющий на правильную последовательность рассуждений.
18-21 баллов ставится, если обучающимся:
Задание выполнено частично или с фактическими ошибками.
0-17 баллов ставится, если обучающимся:
Задание не выполнено или выполнено с большим количеством фактических ошибок.

4.2.3.3. Содержание оценочного средства

1) Дана точка A(x,y). Разработать программу, определяющую, принадлежит ли она треугольнику с вершинами в точках (x1,y1), (x2,y2), (x3,y3).
2) Даны три положительных числа. Разработать программу, определяющую, можно ли построить треугольник со сторонами, длины которых равны этим числам. Если возможно, то ответить на вопрос, является ли он остроугольным.
3) Дано натуральное число n. Разработать программу, вычисляющую сумму первой и последней цифры этого числа.
4) Дано натуральное число n. Разработать программу, которая переставляет местами первую и последнюю цифры этого числа.
5) Дано натуральное число n. Разработать программу, проверяющую, есть ли в записи числа три одинаковые цифры.
6) Разработать программу, определяющую наибольшую и наименьшую цифры в записи данного натурального числа.
7) Дано натуральное число N. Разработать программу, находящую все числа в интервале от 0 до N-1, у которых произведение всех цифр совпадает с суммой цифр данного числа.
8) Дано натуральное число n. Разработать программу проверяющую, будут ли все цифры этого числа различными.
9) Дано натуральное число n. Разработать программу меняющую порядок следования цифр в этом числе на обратный, или сообщить, что это невозможно в силу переполнения.
10) Дано натуральное число N. Разработать программу, получающую новое число, которое образуется из числа N путем замены последней цифры на значение наименьшей цифры в записи числа N.
11) Дано натуральное число N. Разработать программу, переставляющую его цифры так, чтобы образовалось максимальное число, записанное теми же цифрами.
12) Разработать программу, которая называется палиндромом. (Палиндром – это число, которое читается одинаково в прямом и обратном направлениях).
13) В массив A[N] записаны натуральные числа. Разработать программу, которая определяет сумму тех элементов, которые кратны заданному числу K.

14) Дана последовательность целых чисел a1, a2, …, an. Разработать программу, которая определяет, какое число встречается раньше – положительное или отрицательное.
15) Дана последовательность целых чисел a1, a2, …, an. Разработать программу, которая определяет, будет ли эта последовательность возрастающей.
16) Дан массив действительных чисел, размерность которого N. Разработать программу, которая определяет количество отрицательных, положительных и нулевых элементов в нем.
17) Дана последовательность действительных чисел a1, a2, …, an. Разработать программу, которая меняет местами наибольший и наименьший элементы.
18) Разработать программу, которая в заданном одномерном массиве меняет местами соседние элементы, стоящие на четных местах с элементами, стоящими на нечетных местах.
19) Дана последовательность N целых чисел. Разработать программу, которая вычисляет сумму элементов массива, порядковые номера которых совпадают со значением этого элемента.
20) Дана последовательность целых положительных чисел. Разработать программу, которая определяет сумму только тех из них, которые больше заданного числа M. Если таких чисел нет, то выдать сообщение об этом.
21) Разработать программу, которая в массиве целых чисел с количеством n определяет наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
22) Дана последовательность n различных целых чисел. Разработать программу, определяющую сумму ее членов, расположенных между максимальным и минимальным значениями.
23) Дан массив A[N] целых чисел. Разработать программу, которая, не используя другой массив, переставляет элементы массива A[N] в обратном порядке.
24) Разработать программу, вычисляющую сумму положительных элементов матрицы A[N,N], находящихся над главной диагональю.
25) Задана квадратная матрица. Разработать программу, которая меняет сроку с максимальным элементом на главной диагонали со строкой с минимальным элементом на главной диагонали.

Приложение 2
к рабочей программе дисциплины (модуля)
Б1.В.ДВ.03.02 Разработка программных приложений

Перечень литературы, необходимой для освоения дисциплины (модуля)

Направление подготовки: 15.03.06 Мехатроника и робототехника
Профиль подготовки: Физические основы мехатроники и робототехники
Квалификация выпускника: бакалавр
Форма обучения: очно-заочная
Язык обучения: русский
Год начала обучения по образовательной программе: 2024

Основная литература:

1. Агафонов, Е. Д. Прикладное программирование : учебное пособие / Е. Д. Агафонов, Г. В. Ващенко. - Красноярск: СФУ, 2015. - 112 с. - ISBN 978-5-7638-3165-8. - Текст: электронный. - URL: https://znanium.com/catalog/product/550046 - Режим доступа: по подписке.

2. Медведев, М. А. Программирование на СИ#: Учебное пособие / Медведев М.А., Медведев А.Н., - 2-е изд., стер. - Москва :Флинта, Изд-во Урал. ун-та, 2017. - 64 с. ISBN 978-5-9765-3169-7. - Текст : электронный. - URL: https://znanium.com/catalog/product/948428 - Режим доступа: по подписке.

3. Калиногорский, Н. А. Основы практического применения интернет-технологий : учебное пособие / Н. А. Калиногорский. - 3-е изд., стер. - Москва : ФЛИНТА, 2020. - 182 с. - ISBN 978-5-9765-2302-9. - Текст : электронный. - URL: https://znanium.com/catalog/product/1142475 - Режим доступа: по подписке..

4. Царев, Р.Ю. Информатика и программирование [Электронный ресурс] : учеб. пособие / Р. Ю. Царев, А. Н. Пупков, В. В. Самарин, Е. В. Мыльникова. - Красноярск : Сиб. федер. ун-т, 2014. - 132 с. - ISBN 978-5-7638-3008-8. - Текст : электронный. - URL: https://znanium.com/catalog/product/506203 - Режим доступа: по подписке.

Приложение 3
к рабочей программе дисциплины (модуля)
Б1.В.ДВ.03.02 Разработка программных приложений

Перечень информационных технологий, используемых для освоения дисциплины (модуля), включая перечень программного обеспечения и информационных справочных систем

Направление подготовки: 15.03.06 Мехатроника и робототехника

Профиль подготовки: Физические основы мехатроники и робототехники

Квалификация выпускника: бакалавр

Форма обучения: очно-заочная

Язык обучения: русский

Год начала обучения по образовательной программе: 2024

Освоение дисциплины (модуля) предполагает использование следующего программного обеспечения и информационно-справочных систем:

Программное обеспечение: операционная система Windows, Microsoft office, PyCharm, Kaspersky Free для Windows

Электронная библиотечная система «ZNANIUM.COM»

Электронная библиотечная система Издательства «Лань»

Электронная библиотечная система «Консультант студента»