

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки

Кафедра «Информатика»

Е. И. Гридина

СРЕДСТВА ВИЗУАЛЬНОГО ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ

ПРАКТИКУМ

для слушателей специальности переподготовки

1-40 01 73 «Программное обеспечение

информационных систем»

заочной формы обучения

Гомель 2019

УДК 004.65(075.8)
ББК 32.973я73
Г83

*Рекомендовано кафедрой «Профессиональная переподготовка» ГГТУ им. П. О. Сухого
(протокол № 3 от 30.11.2018 г.)*

Рецензент: зав. каф. «Информатика» ГГТУ им. П. О. Сухого
канд. техн. наук, доц. *Т. А. Трохова*

Гридина, Е. И.
Г83 Средства визуального программирования приложений : практикум для слушателей специальности переподготовки 1-40 01 73 «Программное обеспечение информационных систем» заоч. формы обучения / Е. И. Гридина. – Гомель : ГГТУ им. П. О. Сухого, 2019. – 66 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: <http://elib.gstu.by>. – Загл. с титул. экрана.

Практикум включает в себя шесть лабораторных работ, посвященных изучению различных элементов управления Windows Forms C#. Каждая лабораторная работа связана с изучением и практическим освоением различных средств визуального программирования приложений. Данный практикум написан в соответствии с требованиями, предъявленными к оформлению методических пособий, доступным языком и содержит множество примеров, позволяющих в кратчайшие сроки овладеть различными средствами визуального программирования приложений.

Издание адресовано слушателям ИПКиП специальности 1-40 01 73 «Программное обеспечение информационных систем» заочной формы обучения.

**УДК 004.65(075.8)
ББК 32.973я73**

© Учреждение образования «Гомельский
государственный технический университет
имени П. О. Сухого», 2019

СОДЕРЖАНИЕ

Лабораторная работа № 1	4
Лабораторная работа № 2	10
Лабораторная работа № 3	24
Лабораторная работа № 4	39
Лабораторная работа № 5	44
Лабораторная работа № 6	50
Список использованных источников.....	61
Приложение А	62

Лабораторная работа № 1

Тема: Технологии визуального проектирования

Цель: Ознакомиться с визуальными компонентами, применяемыми при создании приложений с пользовательским графическим интерфейсом на основе использования пространства имен System.Windows.Forms.

Теоретические сведения

Интерфейс программирования приложений, отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код – классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, C++, так и на VB.Net, J# и др.

Внешний вид приложения является нам преимущественно через формы. Формы являются основными строительными блоками. Они предоставляют контейнер для различных элементов управления. А механизм событий позволяет элементам формы отзываться на ввод пользователя, и, таким образом, взаимодействовать с пользователем.

При открытии проекта в Visual Studio в графическом редакторе мы можем увидеть визуальную часть формы - ту часть, которую мы видим после запуска приложения и куда мы переносим элементы с панели управления. Но на самом деле форма скрывает мощный функционал, состоящий из методов, свойств, событий и прочее. Рассмотрим основные свойства форм.

Большинство этих свойств оказывает влияние на визуальное отображение формы:

- Name: устанавливает имя формы – точнее имя класса, который наследуется от класса Form;
- BackColor: указывает на фоновый цвет формы. Щелкнув на это свойство, мы сможем выбрать тот цвет, который нам подходит из списка предложенных цветов или цветовой палитры;
- BackgroundImage: указывает на фоновое изображение формы;

- `BackgroundImageLayout`: определяет, как изображение, заданное в свойстве `BackgroundImage`, будет располагаться на форме;
- `ControlBox`: указывает, отображается ли меню формы. В данном случае под меню понимается меню самого верхнего уровня, где находятся иконка приложения, заголовок формы, а также кнопки минимизации формы и крестик. Если данное свойство имеет значение `false`, то мы не увидим ни иконку, ни крестика, с помощью которого обычно закрывается форма;
- `Cursor`: определяет тип курсора, который используется на форме;
- `Enabled`: если данное свойство имеет значение `false`, то она не сможет получать ввод от пользователя, то есть мы не сможем нажать на кнопки, ввести текст в текстовые поля и т.д.;
- `Font`: задает шрифт для всей формы и всех помещенных на нее элементов управления. Однако, задав у элементов формы свой шрифт, мы можем тем самым переопределить его;
- `ForeColor`: цвет шрифта на форме;
- `FormBorderStyle`: указывает, как будет отображаться граница формы и строка заголовка. Устанавливая данное свойство в `None` можно создавать внешний вид приложения произвольной формы;
- `HelpButton`: указывает, отображается ли кнопка справки формы;
- `Icon`: задает иконку формы;
- `Location`: определяет положение по отношению к верхнему левому углу экрана, если для свойства `StartPosition` установлено значение `Manual`;
- `MaximizeBox`: указывает, будет ли доступна кнопка максимизации окна в заголовке формы;
- `MinimizeBox`: указывает, будет ли доступна кнопка минимизации окна;
- `MaximumSize`: задает максимальный размер формы;
- `MinimumSize`: задает минимальный размер формы;
- `Opacity`: задает прозрачность формы;
- `Size`: определяет начальный размер формы;
- `StartPosition`: указывает на начальную позицию, с которой форма появляется на экране;
- `Text`: определяет заголовок формы;

- TopMost: если данное свойство имеет значение true, то форма всегда будет находиться поверх других окон;
- Visible: видима ли форма, если мы хотим скрыть форму от пользователя, то можем задать данному свойству значение false;
- WindowState: указывает, в каком состоянии форма будет находиться при запуске: в нормальном, максимизированном или минимизированном.

Пример

Для создания проекта Windows Forms, мы используем среду разработки Visual Studio (2013).

Создадим проект, для этого выберем пункт меню: Файл – Создать проект – Visual C# - Приложение Windows Forms. Введем внизу экранной формы – имя проекта и нажимаем кнопку ОК.

На экране будет отображен проект с первоначальной формой (1), обозревателем решений, в котором находятся файлы проекта (2), и свойствами формы (3), как показано на рисунке 1.1.

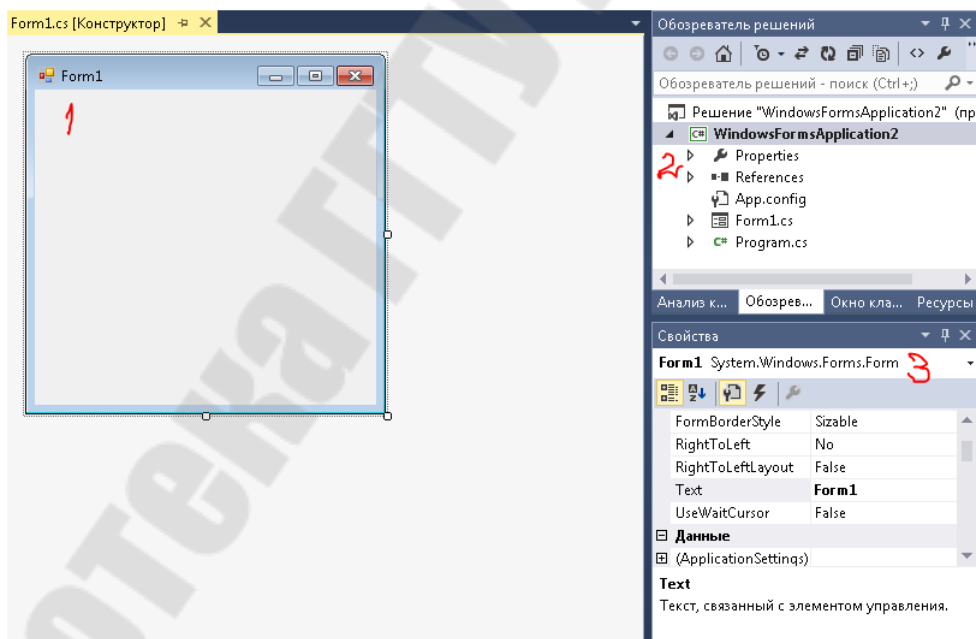


Рисунок 1.1 – Конструктор формы

Зададим для формы проекта, которая будет являться заставкой проекта, соответствующие свойства:

- Свойство Text – “Учет посещаемости”;

- Свойство `BackgroundImage` – указываем фоновое изображение в виде рисунка (локальный ресурс);
- Свойство `BackgroundImageLayout` – `Center`;
- Свойство `Cursor` – `Hand`;
- Свойство `Icon` – устанавливаем новую иконку;
- Свойство `Opacity` – 50%.

Добавим на кнопку два элемента управления `Button`. Доступ к панели элементов, на которой находятся различные элементы выполним, путем выбора пункта меню Вид – Панель Элементов. Слева от формы, на экране, будет расположена панель элементов, где различные элементы, расположены по различным тематическим группам.

Свойство `Text` для `Button1` – «Вход», свойство `Text` для `Button2` – «Выход».

Результат полученной формы представлен на рисунке 1.2.

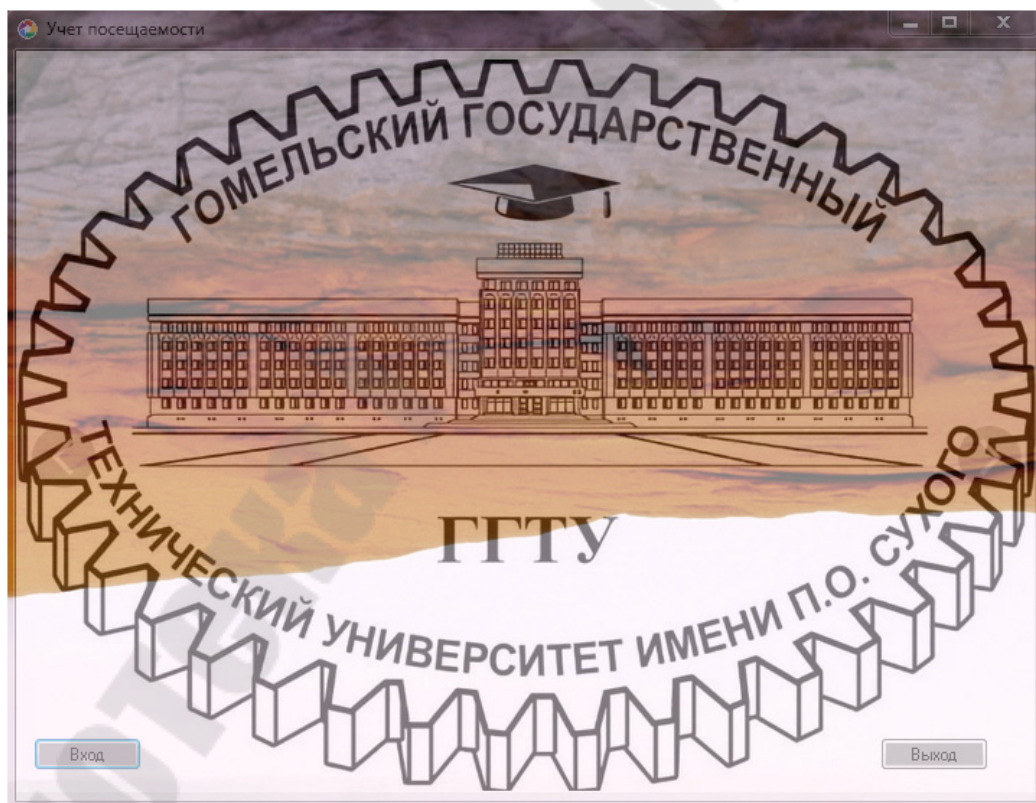


Рисунок 1.2 – Форма приложения

В конструкторе дважды нажмем на кнопку «Выход», чтобы открыть окно с кодом обработчика событий `Button Click`. Добавляем в этот метод код: `Application.Exit()`, который позволит закончить работу приложения.

Далее обработаем кнопку «Вход», для этого мы добавим еще одну форму в проект, чтобы передать управление данной форме при дальнейшей работе с программой.

Чтобы добавить еще одну форму в проект, нажмем на имя проекта в окне Обозреватель решений правой кнопкой мыши и выберем Добавить->Форма Windows.

Дадим новой форме какое-нибудь имя, например, Form2.cs:

Итак, у нас в проект была добавлена вторая форма. Теперь попробуем осуществить взаимодействие между двумя формами. Допустим, первая форма по нажатию на кнопку “Вход” будет вызывать вторую форму. Кликнув двойным щелчком по кнопке “Вход” перейдем в файл кода. Итак, мы попадем в обработчик события нажатия кнопки, который создается по умолчанию после двойного щелчка по кнопке: `Form2 newForm = new Form2(); newForm.Show();`

Перейдем ко второй форме и перейдем к ее коду – нажмем правой кнопкой мыши на форму и выберем View Code (Просмотр кода). Пока он пустой и содержит только конструктор. Поскольку C# поддерживает перегрузку методов, то мы можем создать несколько методов и конструкторов с разными параметрами и в зависимости от ситуации вызывать один из них.

При работе с несколькими формами надо учитывать, что одна из них является главной – которая запускается первой в файле Program.cs. Если у нас одновременно открыта куча форм, то при закрытии главной закрывается все приложение и вместе с ним все остальные формы.

Поэтому, для того, чтобы скрыть основную форму, установим свойство Visible в значение false.

Задание

Согласно варианту задания, представленного в приложении, разработать форму-“заставку”, реализовать завершение работы программы и передачу управления другим формам.

Вопросы к защите лабораторной работы

1. Что такое форма и ее назначение?
2. Что такое элементы управления? На какие группы они делятся?

3. Как установить элемент управления на форму и проинициализировать его?
4. Что такое событие? Как в .Net реализуются события?
5. Как добавить несколько кнопок на форму?
6. Как разместить на кнопке изображение?
7. Как добавить новую форму в приложение?
8. Как организовать переход к добавленной форме?
9. Что такое модальная и немодальная форма? Как они вызываются?

Лабораторная работа № 2

Тема: Библиотеки для создания приложений

Цель: Научиться использовать стандартные библиотеки для создания приложений

Теоретические сведения

Библиотека классов .NET Framework представляет собой библиотеку классов, интерфейсов и типов значений, которые обеспечивают доступ к функциональным возможностям системы. Она составляет основу для создания приложений, компонентов и элементов управления .NET Framework.

Пространство имен System.Windows.Forms содержит огромное количество типов для создания приложений с пользовательским графическим интерфейсом.

Элементом управления называется класс, порожденный от класса System.Windows.Forms.Control. Примеры элементов управления: кнопка (Button), главное меню (MainMenu), метка (Label).

В пространстве имен System.Windows.Forms определено четыре класса для работы с меню (рисунок 2.1).

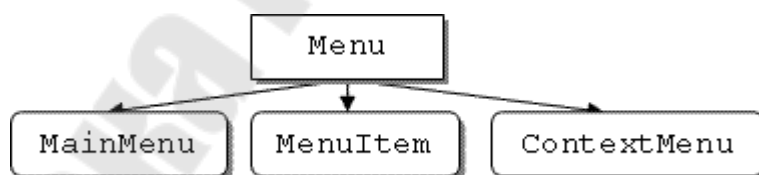


Рисунок 2.1 – Иерархия классов меню

Абстрактный класс Menu предоставляет базовую функциональность для контекстного и главного меню. Он содержит вложенный класс MenuItemCollection. Этот класс представляет коллекцию для хранения списка подменю данного меню. Коллекция содержит элементы класса MenuItem.

Класс MenuItem описывает строку меню, которая может содержать один или несколько подпунктов меню.

Класс MainMenu и класс ContextMenu описывают главное и контекстное меню соответственно. Они могут содержать один или несколько подпунктов меню типа MenuItem.

Класс Menu содержит поле класса MenuItemCollection. Для обращения к этому полю используется свойство public Menu.MenuItemCollection MenuItems {get; }.

Для обработки события нажатия на пункт меню используется событие Click класса MenuItem.

Главное меню в пространстве имен System.Windows.Forms представляется классом MainMenu. Контекстное меню в пространстве имен System.Windows.Forms представляется классом ContextMenuStrip.

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. 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.

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 создания окна необходимо выбрать пункт меню Project/Add Windows Form.

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`. Пример простейшего модального диалогового окна для вывода сообщения можно использовать метод `Show`, передав ему через параметр текст сообщения. Прототип использованного метода `Show` следующий: `public static DialogResult Show(string message);`.

Когда пользователь щелкает кнопку `ОК`, 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 на рисунке 2.2.

```
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
};
```

Рисунок 2.2 – Прототип `MessageBox.Show()`

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`: `ОК`, `ОКCancel`, `YesNo`, `YesNoCancel`, `RetryCancel`, `AbortRetryIgnore`.

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`: `Asterisk`, `Information`, `Error`, `Stop`, `Exclamation`, `Warning`, `Question`, `None`.

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`.

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`: `DefaultDesktopOnly`, `RightAlign`, `RtlReading`, `ServiceNotification`.

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`: `Abort`, `Cancel`, `Ignore`, `No`, `None`, `OK`, `Retry`, `Yes`.

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

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.

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

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.

Заккрытие формы - Application.Exit();

Пример

Все пользователи Windows-приложений хорошо знакомы с меню, которые представляют собой простой механизм выбора команд.

Добавим в приложение (созданное в лабораторной работе 1) простое меню.

1. Откройте панель инструментов Toolbox и перетащите управляющий элемент MenuStrip на форму приложения.

2. Для создания выпадающего меню, в окне Properties измените названия этих пунктов меню (пункты элемента меню Работа с таблицами, должна соответствовать названию таблиц в базе данных, рисунок 2.3).

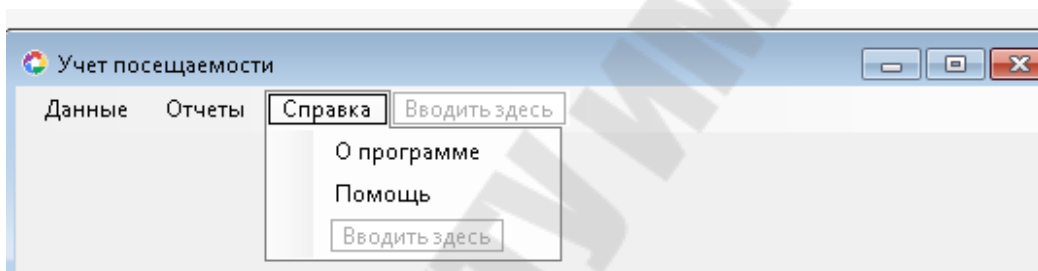


Рисунок 2.3 – Проектирование меню

В свойстве ShortcutKeys в окне Properties (рисунок 2.4) для пункта меню выбрать из появившегося окна нужное сочетание клавиш, для отображения этого сочетания рядом с названием пункта меню. Данный элемент будет активизироваться при нажатии соответствующей этой букве клавиши.

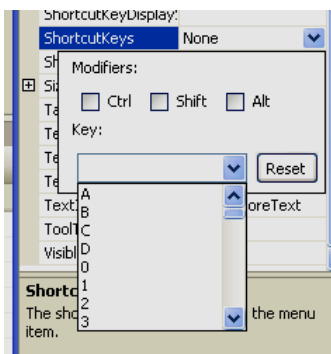
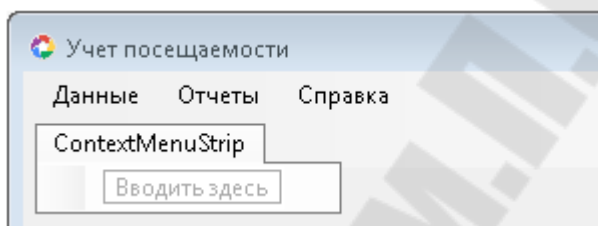


Рисунок 2.4 – Выбор сочетания клавиш для выбора пункта меню

Дополнительные возможности меню. В классе MenuStrip определены свойства, при помощи которых можно, к примеру, устанавливать флажок напротив пункта меню, прятать пункты меню, делать некоторые пункты меню недоступными и т. п. Вот перечень свойств, обеспечивающих подобные возможности.

Аналогичным образом создать всплывающее контекстное меню ContextMenuStrip (рисунок 2.5), дублирующее основные действия главного меню.

Рисунок 2.5 –
Контекстное меню



После
формирования

пунктов контекстного меню необходимо его подключить к форме Form2. Для этого на вкладке Свойства (Properties) формы Form2 строке, соответствующей свойству ContextMenuStrip нужно установить значение созданного объекта contextMenuStrip1 (рисунок 2.6).

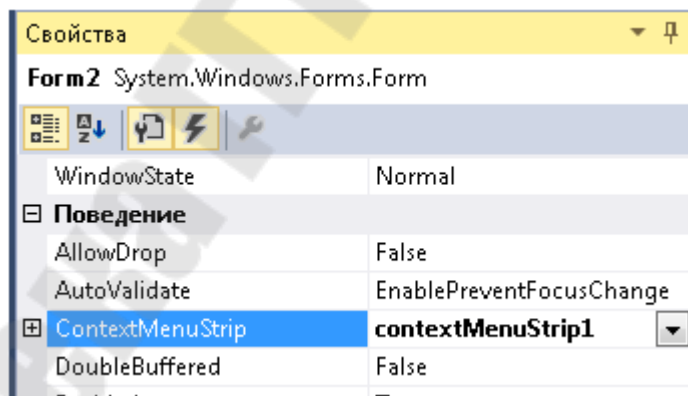


Рисунок 2.6 – Подключение контекстного меню

После компиляции проекта и запуска приложения на выполнение можно проверить режим активизации контекстного меню (рисунок 2.7).

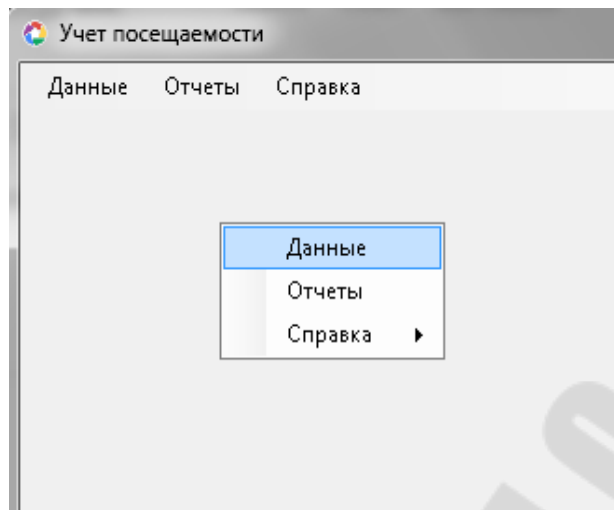


Рисунок 2.7 – Вызов контекстного меню

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 Form2, выделить на форме класс ContextMenuStrip и сделать двойной щелчок на соответствующем пункте меню

Создадим дочернюю форму. Для определения главного окна (Form2), 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 (Form3). Для этого выберите пункт меню Project/Add Windows Form. Это окно должно вызываться из пункта главного меню "Данные - Просмотр". Вставьте код (рисунок 2.8), чтобы создать новую MDI дочернюю форму, когда пользователь щелкает на пункте меню, например "Просмотр".

```
private void просмотрToolStripMenuItem_Click(object sender, EventArgs e)
{
    Form3 Dannie = new Form3();
    Dannie.MdiParent = this;
    Dannie.Show();
}
```

Рисунок 2.8 – Код передачи управление дочерней форме

Создадим новую форму FormAbout для вывода справочной информации о разрабатываемом приложении (рисунок 2.8). Для этого добавим в проект новый компонент, выбрав из списка AboutBox – Проект-Добавим новый элемент Ctrl+Shift+A – Окно о программе (About Box).

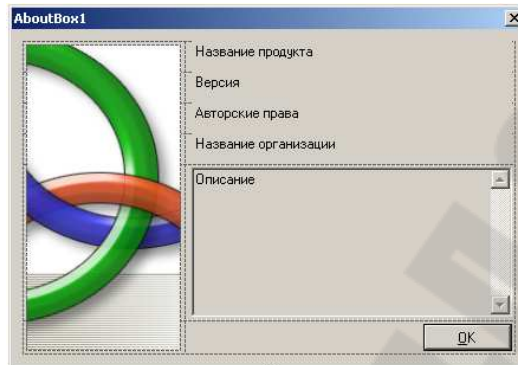


Рисунок 2.9 – Форма FormAbout

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 (рисунок 2.10).

```
public AboutBox1()
{
    InitializeComponent();
    this.Text = String.Format("О программе {0}", "Учет посещаемости");
    this.labelProductName.Text = AssemblyProduct;
    this.labelVersion.Text = String.Format("Версия {0}", AssemblyVersion);
    this.labelCopyright.Text = "@ИПКИП ГГТУ им.П.О.Сухого, 2018";
    this.labelCompanyName.Text = "Гридина Е.И.";
    this.textBoxDescription.Text = "Дисциплина Средства визуального программирования приложения";
}
```

Рисунок 2.10 – Конструктор класса AboutBox

Для открытия пользовательского модального диалогового окна используется метод ShowDialog. В лабораторной работе диалоговое окно должно открываться при щелчке пользователем на пункте в меню "Автор". Код для открытия диалогового окна может выглядеть, как показано на рисунке 2.11.

```

// Открываем модальное диалоговое окно
private void aboutToolStripMenuItem_Click(object sender,
EventArgs e)
{
    AboutBox1 aboutBox = new AboutBox1();
    aboutBox.ShowDialog(this);
}

```

Рисунок 2.11 – Метод aboutToolStripMenuItem_Click()

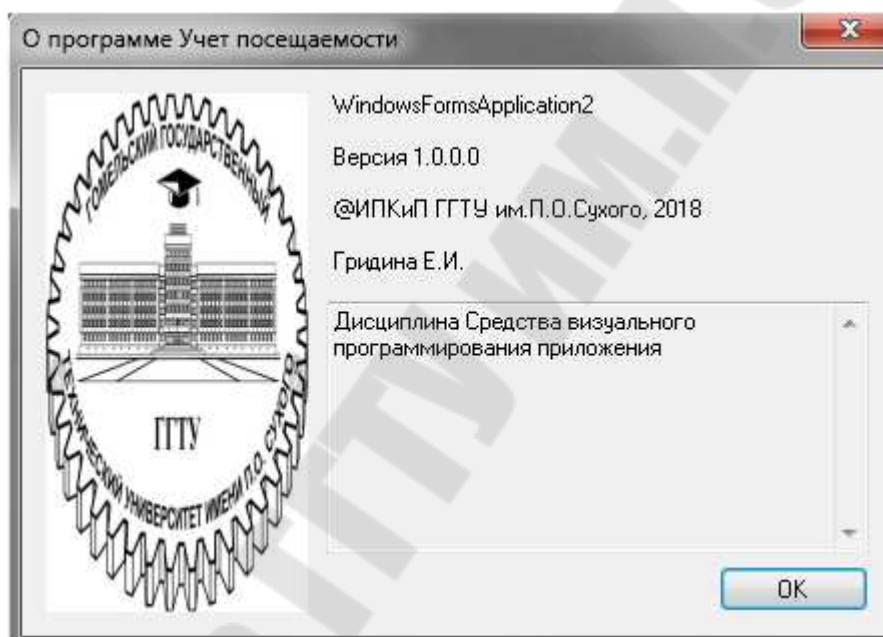


Рисунок 2.12 – Форма О программе

Пункт меню “Выход” определили командой Application.Exit());.

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.

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

Разработаем панель инструментов. Для этого необходимо открыть панель Toolbox и добавьте элемент управления ToolStrip на разрабатываемую форму Form2. В выпадающем меню элемента управления ToolStrip на форме Form2 необходимо выбрать элемент управления button – кнопка. При этом в панели инструментов добавится кнопка. Добавьте на панель инструментов кнопки с именами toolStripButtonUndo, toolStripButtonNew, toolStripButtonEdit, toolStripButtonSave, toolStripButtonRemove. В результате должна быть сформирована панель инструментов с кнопками (рисунок 2.13).

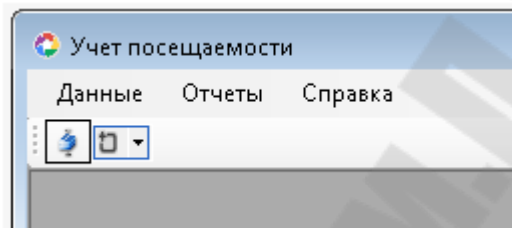


Рисунок 2.13 – Панель инструментов на форме

Для кнопок панели инструментов сформируем графическое представление. Это можно сделать путем задания свойства Image соответствующей кнопке.

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

Созданная панель инструментов содержит шесть кнопок. По функциональности каждой из этих кнопок будут соответствовать аналогичные пункты меню.

Аналогично, настроим свойство Image, для соответствующих пунктов меню.

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 для кнопки "Просмотр" (toolStripButtonUndo) строка подсказки ToolTipText соответствует строке "Просмотр данных".

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

В окне Toolbox выделим пункт StatusStrip и перетащим его на форму .

Откроем выпадающий список объекта класса statusBarEmployee и выберем объект StatusLabel (рисунок 2.14).

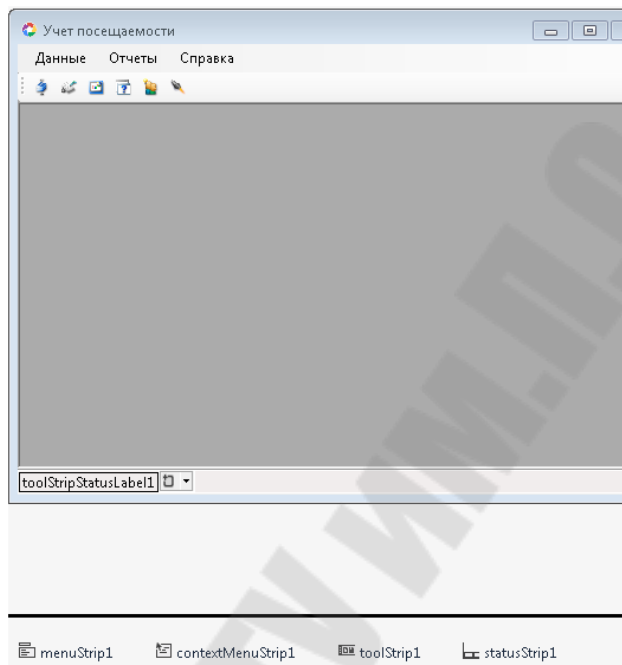



Рисунок 2.14 – Добавление элементов управления на форму

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

Для формы Form2 в строке состояний необходимо вывести информацию при наведении курсора мыши на пунктах меню "Данные". Первоначально в дизайнера формы необходимо выделить пункт меню "Действие", перейти на вкладку Properties и открыть окно событий, нажав кнопку . На данной вкладке необходимо выделить событие MouseEnter и в поле ввода сделать двойной щелчок (рисунок 2.15).

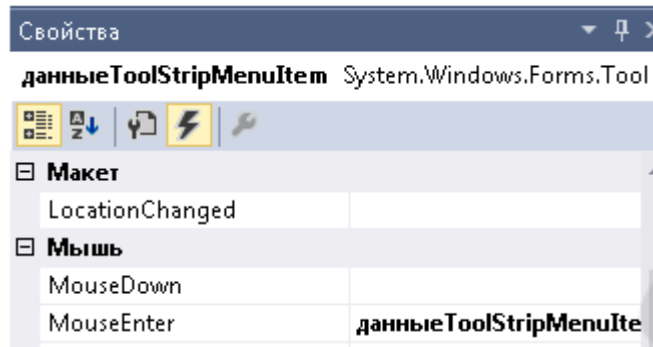


Рисунок 2.15 – События

Система сгенерирует код обработчика: `private void actionToolStripMenuItem_MouseEnter(object sender, EventArgs e) { }`
Добавим в обработчик код (рисунок 2.16).

```
private void данныеToolStripMenuItem_MouseEnter(object sender, EventArgs e)
{
    toolStripStatusLabel1.Text = "Данные";
}
```

Рисунок 2.16 – Метод обработки события MouseEnter

Если откомпилировать программу, запустить её, навести указатель мыши на пункт "Данные", то сгенерируется событие "MouseEnter" и в строке состояния выведется текстовое сообщение "Данные" (рисунок 2.17).

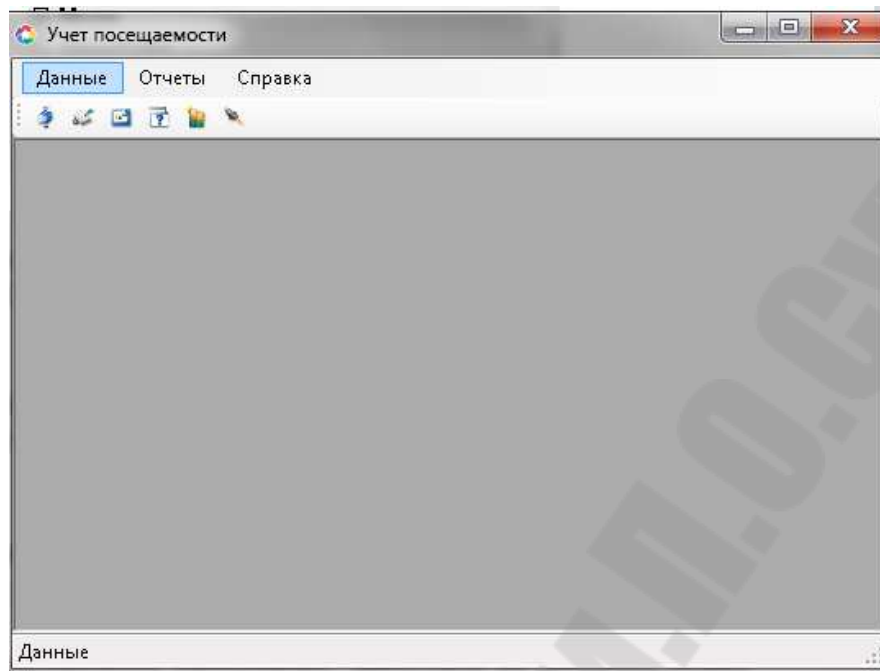


Рисунок 2.17 – Форма приложения

Если теперь переместить указатель мыши с пункта меню "Данные", 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, представлен на рисунке 2.17.

```
private void данныеToolStripMenuItem_MouseEnter(object sender, EventArgs e)
{
    toolStripStatusLabel1.Text = "Данные";
}

ссылка 1
private void данныеToolStripMenuItem_MouseLeave(object sender, EventArgs e)
{
    toolStripStatusLabel1.Text = "";
}
```

Рисунок 2.18 – Методы обработки событий "мыши"

Вышеприведенные обработчики будут вызываться только тогда, когда пользователь наведет указатель мыши на пункт меню "Данные".

Для того чтобы обработчики реагировали на все строки пунктов главного меню "Справка" и "Отчеты" формы Form2, необходимо сформировать соответствующие события MouseEnter и MouseLeave для всех пунктов и подпунктов меню и создать для них обработчики.

Задание

Для приложения, разработанного в лабораторной работе 1, реализовать основные элементы (главное меню, контекстное меню, таблица акселераторов, информацию о разработчике, панель инструментов, строка состояний и др.). Все кнопки панели инструментов должны быть снабжены всплывающими подсказками.

Вопросы к защите лабораторной работы

1. Что такое форма и ее назначение?
2. Что такое элементы управления? На какие группы они делятся?
3. Как установить элемент управления на форму и проинициализировать его?
4. Что такое событие? Как в .Net реализуются события?
5. Что такое обработчик события?
6. Как задаются обработчики событий для элементов управления?
7. Как создать верхнее меню?
8. Как добавить несколько пунктов меню?
9. Как установить определенному пункту меню сочетание клавиш?
10. Как создать панель инструментов?
11. Как добавить несколько кнопок на панель инструментов?
12. Как разместить на кнопке изображение?
13. Как добавить новую форму в приложение?
14. Как организовать переход к добавленной форме?
15. Что такое модальная и немодальная форма? Как они вызываются?

Лабораторная работа № 3

Тема: Элементы управления и их позиционирование

Цель: Овладение навыками создания и практического использования элементов управления

Теоретические 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.

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

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

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

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. Можно выбрать в списке нужное событие и рядом с ним сделать двойной щелчок или ввести имя метода или выбрать метод из списка.

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

Рассмотрим основные элементы управления Windows –форм.

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.

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.

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).

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 ОК 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 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.

Возможные состояния флажка (`Indeterminate` можно использовать только тогда, когда для свойства `ThreeState` установлено значение `true`).

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.

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`.

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`.

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.

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`).

Стиль для `ComboBox` можно настроить при помощи свойства `Style`, для которого используются значения из перечисления `ComboBoxStyle`.

Если на форме размещено несколько элементов управления, то пользователи обычно ожидают, что между ними можно будет перемещаться с помощью клавиши `Tab`. Часто бывает необходимо после размещения элементов управления настроить порядок перехода между ними. Для этого используются два свойства (унаследованные от базового класса `Control` и поэтому общие для всех элементов управления): `TabStop` и `TabIndex`. Для свойства `TabStop` используются только два значения: `true` и `false`. Если для `TabStop` установлено значение `true`, то к этому элементу управления можно будет добраться с помощью клавиши `Tab`. Если же установлено значение `false`, то участвовать в переходах по `Tab` этот элемент управления не будет. Если элемент управления `TabStop` имеет значение `true`, то очередность перехода можно настроить с помощью свойства `TabIndex`:

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

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

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.

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

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

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

AutoScroll объекта Panel значение true. В результате пользователь получит возможность доступа к "не вмещающимся" элементам управления с помощью полос прокрутки.

Пример

Необходимо создать приложение, которое будет способно добавлять и удалять элементы в дерево и в список. Главное окно программы будет содержать поле ввода данных, две кнопки: Добавить в список и Добавить в дерево, компонент ListView и компонент TreeView. В поле ввода пользователь может набрать любую строку. При нажатии на соответствующую кнопку содержимое поля переносится в ListView или TreeView.

Для этого создайте новое C# Windows-приложение под названием ViewApp. Добавьте на форму следующие элементы: ListView; TreeView; Button – два элемента; TextBox.

Примерное расположение элементов показано на рисунке 3.1. Вы можете построить интерфейс программы по своему усмотрению.

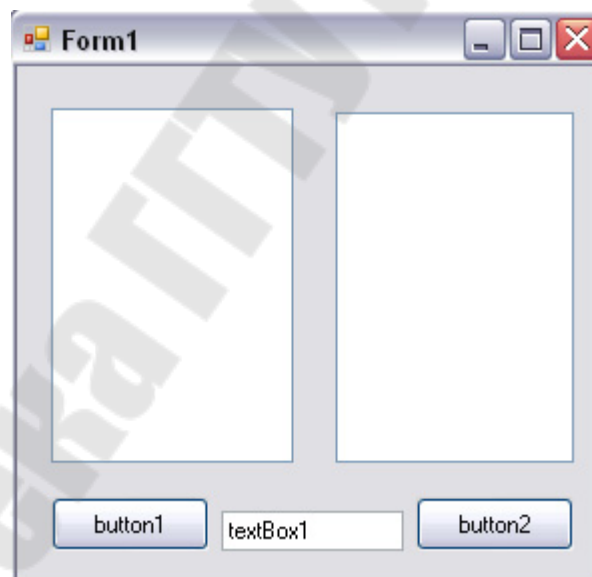


Рисунок 3.1 – Проектирование формы приложения

Измените свойства добавленных компонент: button1:Text — «добавить в список»; button2:Text — «добавить в дерево»; textBox1:Text — «».

Теперь необходимо наделить код функциональностью. В первую очередь следует добавить обработчик нажатия кнопки Добавить в список. Для этого щелкните два раза указателем мыши по кнопке на

форме. Перед вами откроется окно кода, в который будет добавлена функция `button1_Click`. Добавьте в функцию `button1_Click` код `listView1.Items.Add(textBox1.Text);`.

Откомпилируйте и запустите программу. Проверьте работоспособность программы добавляя в список различные значения.

Вторым объектом создаваемой программы является дерево.

Давайте рассмотрим работу с деревом.

Для начала необходимо добавить обработчик кнопки `Добавить` в дерево. Для этого щелкните два раза левой кнопкой мыши по кнопке на форме. При этом в программу добавится обработчик нажатия кнопки `button2` — функция `button2_Click`. Измените код функции `button2_Click` так, как представлено на рисунке 3.2.

```
private void button2_Click(object sender, System.EventArgs e)
{
    // получаем выделенный узел
    TreeNode node = treeView1.SelectedNode;
    // если выделенного узла нет
    if (node == null){// добавляем новый элемент
        // в корень основного дерева
        treeView1.Nodes.Add(textBox1.Text) ;}
    // если имеется выделенный узел
    else { // добавляем новый элемент
        // как вложенный в выделенный узел
        node.Nodes.Add(textBox1.Text);}
    }
}
```

Рисунок 3.2 – Код кнопки «Добавить»

Откомпилируйте и запустите программу. Результат работы представлен на рисунке 3.3.

Дерево дает более широкий спектр представления информации. Элементы дерева могут быть встроены в другие элементы. Каждый элемент дерева называется узлом. Если узел содержит другие узлы, то называется корнем дерева. Давайте рассмотрим код, который заставляет программу работать именно таким образом: `TreeNode node = treeView1.SelectedNode;`

Класс `TreeView` имеет свойство `SelectedNode`. Это свойство возвращает объект `TreeNode`, который выделен на текущий момент в дереве. Если в дереве не выделен ни один элемент, то свойство

SelectedNode возвращает null: `if(node==null){treeView1.Nodes.Add(textBox1.Text);}`

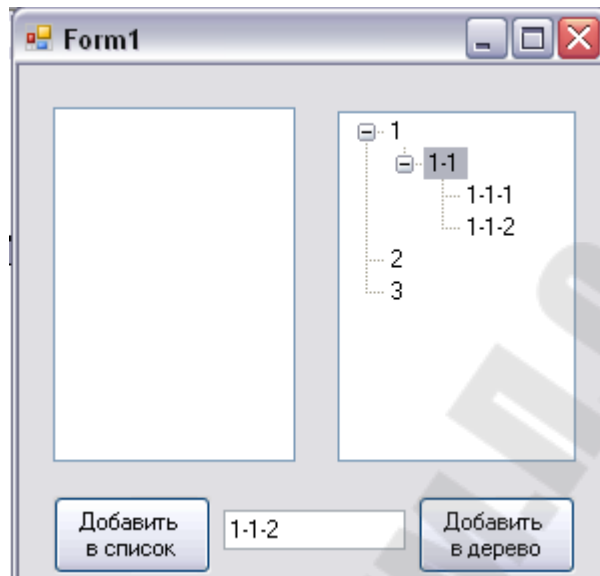


Рисунок 3.3 – Результаты работы с деревом

Если все-таки свойство SelectedNode вернет null, то элемент будет добавлен в список первого уровня. Так были добавлены «1-узел», «2-узел» и «3-узел». Свойство Nodes класса TreeView содержит коллекцию TreeNodeCollection всех узлов первого уровня. Функция Add класса TreeNodeCollection добавляет новый элемент в конец коллекции. `else {node.Nodes.Add(textBox1.Text);}`

Если же один из элементов дерева выделен, то переменная node будет содержать выделенный объект. Класс TreeNode имеет свойство Nodes, которое так же, как и свойство Nodes объекта TreeView, возвращает объект класса TreeNodeCollection. Каждый объект TreeNode содержит коллекцию узлов, для которых он является родительским.

Добавьте на форму компонент ImageList. Он отобразится не на форме, а в панели компонент ниже формы. Компонент ImageList имеет свойство Images.

Это свойство отвечает за коллекцию изображений, содержащихся в списке. Откройте окно редактирования Image Collection Editor, нажав кнопку с тремя точками в поле Collection окна свойств. Перед вами откроется окно Image Collection Editor (рис. 3.4).

Это окно позволяет добавить новое изображение в коллекцию или удалить уже существующее. Для того чтобы добавить новое изображение, вам необходимо нажать кнопку Add.

Список Members может содержать сколько угодно элементов. Для нашего примера будет достаточно одного элемента в списке. Закройте Image Collection Editor, нажав кнопку ОК.

Компонент ListView имеет свойство LargeImageList, которое хранит список изображений, закрепленный за ListView объектом. Установите в окне свойств объекта listView1 свойство LargeImageList как imageList1. Теперь за списком listView1 будет закреплен список изображений imageList1.

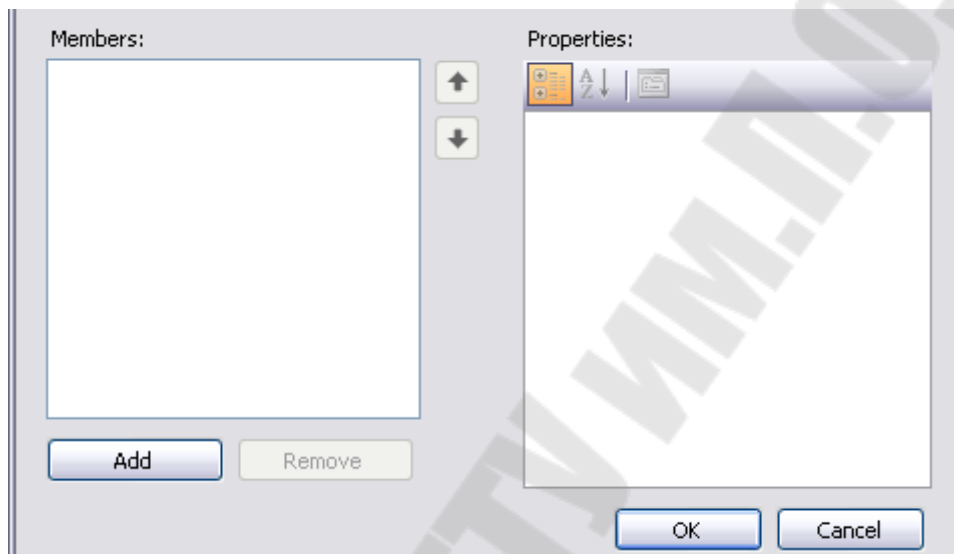


Рисунок 3.4 – Окно настройки коллекции изображений

Далее давайте изменим код нашей программы, для того чтобы список мог содержать не только текстовую информацию, но и графическую. Программа потребует минимальных изменений. Класс ListViewItemCollection содержит несколько вариантов функции Add. Мы использовали Add(string text). Другим вариантом функции является Add(string text, int imageIndex). Вторым параметром этой функции является индекс изображения в списке ImageList, которое было предварительно установлено для ListView.

Замените вызов функции listView1.Items.Add(textBox1.Text);, внутри функции button1_Click на listView1.Items.Add(textBox1.Text,0);

Добавился лишь второй параметр при вызове функции. Однако попробуйте вновь запустить программу. Добавьте в список элементы «один» и «два». На этот раз список будет содержать и текстовое представление элемента, и его пиктограмму.

Для всех элементов нашего списка пиктограмма будет одна и та же. Однако вы можете установить в качестве пиктограммы любой элемент списка изображений. Вторым параметром функции Add является индекс изображения из списка ImageList. Если ImageList содержит пять элементов, то для установки пиктограммы с индексом 3 вам необходимо вызвать функцию Add как: listView1.Items.Add (textBox1.Text,2);.

Так же, как пиктограммы в ListView компоненте, пиктограммы могут использоваться другими компонентами. Использование ImageList схоже в применении со всеми компонентами. Вот как выглядят ImageList вместе с TreeView. В окне свойств объекта treeView1 установите свойство ImageList в imageList1.

Для того чтобы добавить отображение пиктограммы в дерево, измените код функции button2_Click (рисунок 3.5).

```
// получаем выделенный узел
TreeNode node = treeView1.SelectedNode;
// если выделенного узла нет
if (node == null)
{
    // добавляем новый элемент
    // в корень основного дерева
    TreeNode newNode = new TreeNode();
    newNode.Text = textBox1.Text;
    newNode.ImageIndex = 0;
    treeView1.Nodes.Add(newNode);
}
// если имеется выделенный узел
else
{
    // добавляем новый элемент
    // как вложенный в выделенный узел
    TreeNode newNode = new TreeNode();
    newNode.Text = textBox1.Text;
    newNode.ImageIndex = 0;
    node.Nodes.Add(newNode);
}
```

Рисунок 3.5 – Код работы с «деревом»

Функция Add класса `TreeNodeCollection` так же перегружена, как и функция Add класса `ListViewItemsCollection`. Вторым вариантом функции Add является `Add(System.Windows.Forms.TreeNode node)`. Для того чтобы добавить текстовый элемент с пиктографическим изображением в дерево, необходимо: установить у `TreeView` свойство `ImageList`; создать элемент `TreeNode`; установить свойство `Text`; установить свойство `ImageIndex`; вызвать функцию `TreeNode.Nodes.Add()`.

Запустите программу с новым кодом. Попробуйте добавить элементы в дерево. Так же, как и в списке, все узлы дерева будут отображаться с пиктограммой (рис. 3.6.).

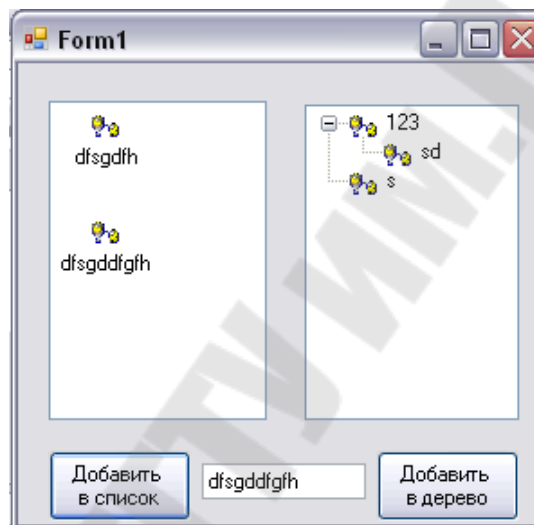


Рисунок 3.6 – Использование пиктограмм компонентом `TreeView`

Создайте новый Windows Forms проект с именем `ScrollApp`. Поместите на форму элемент `VScrollBar` и четыре элемента `Label`. Сделайте это так, как показано на рисунке 3.7.

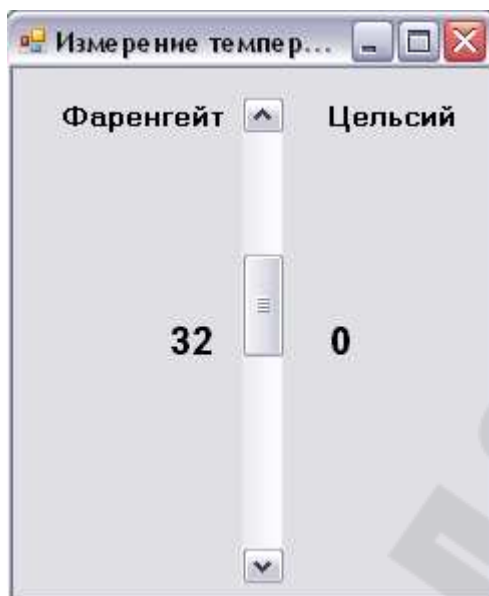


Рисунок 3.7 – Проектирование формы приложения

Установите для компонент свойства в соответствии с приведенным ниже списком: Form1: Text – Измерение температуры; vScrollBar1: LargeChange – 10, Maximum – 60, Minimum – 20, SmallChange – 1, Value – 32; label1:Text – Фаренгейт, Font Size – 10, Font Bold – True; label2: BackColor – White, Text – 32, Font Size – 14, Font Bold – True, Name – labelFarTemp; label3: Text – Цельсий, Font Size – 10, FontBold – True; label4: BackColor – White, Text – 0, Font Size – 14, Font Bold – True, Name – labelCTemp.

Компонент ScrollBar имеет событие Scroll. Давайте добавим обработчик этого события в код программы. Для этого в окне свойств на закладке событий для элемента vScrollBar1 щелкните два раза по событию Scroll. В код программы добавится обработчик события Scroll: `private void vScrollBar1_Scroll(object sender, System.Windows.Forms.ScrollEventArgs e){}`.

Добавьте в функцию vScrollBar1_Scroll код, представленный ниже: `labelFarTemp.Text = vScrollBar1.Value.ToString() ; labelCTemp.Text =Convert.ToString((int) ((double)vScrollBar1.Value - 32)/9*5));`

Этот код переводит значение Value полосы прокрутки во время изменения положения бегунка и определяет значение температуры по шкале Фаренгейта. Затем вычисляется значение температуры по Цельсию и отображаются оба значения. Откомпилируйте и запустите программу.

Задание

1. Выполните все примеры, приведенные в практической части лабораторной работы.

2. Измените первый пример следующим образом. Добавьте на форму, содержащую элементы Список и Дерево, кнопки для удаления выделенного (или выделенных) элементов из списка и дерева.

3. Создайте приложение, показанное на рисунке 3.8. Пользователю должна быть предоставлена возможность выбирать элементы в CheckedListBox, а затем посредством нажатия кнопки переносить выделенные элементы в обычный список ListBox.

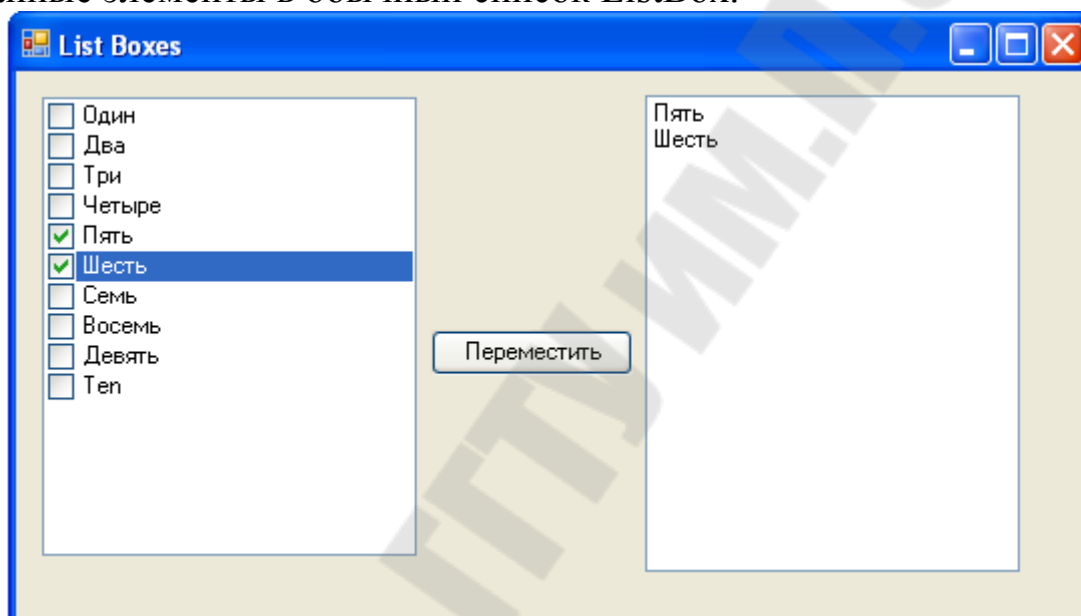


Рисунок 3.8 – Форма с двумя видами списков

4. Создайте приложение, на форме которого разместите три полосы прокрутки для управления цветом (красным, зеленым, синим), который используется как фон формы. Начальные положения бегунков полос прокруток должны соответствовать заданному в дизайнерах цвету фона формы.

5. Создайте форму (рисунок 3.9), которая позволит ввести имя, адрес, род занятий и возраст. На форме должны располагаться кнопки ОК и Help. Данные из формы считываются и заносятся в результирующее текстовое поле, расположенное на этой же форме. При нажатии на кнопку Help выводится краткое описание каждого текстового окна в результирующем текстовом поле (рисунок 3.10). В программе

должна быть реализована проверка вводимого текста по следующим критериям:

- а) поле с именем пользователя не должно быть пустым;
- б) возраст пользователя должен представлять собой число, большее или равное 0;
- в) род занятий пользователя должен описываться как программист или оставаться пустым (можно использовать флажок или текстовое поле);
- г) поле с адресом пользователя не может оставаться пустым;
- д) поле для вывода результирующей информации должно быть только для чтения.

Подсказка: Для проверки значения возраста необходимо обработать сообщение KeyPress. Нажатие кнопки ОК невозможно до введения всей необходимой информации.

Текстовые поля

Имя

Адрес

Программист

Пол

Женский Мужской

Возраст

Результат

Имя = Ваше имя
Адрес = Ваш адрес
Программист = Отметьте 'Программист' если вы являетесь программистом
Пол = Выберите ваш пол
Возраст = Ваш возраст

Рисунок 3.9 – Внешний вид формы при нажатии кнопки Help

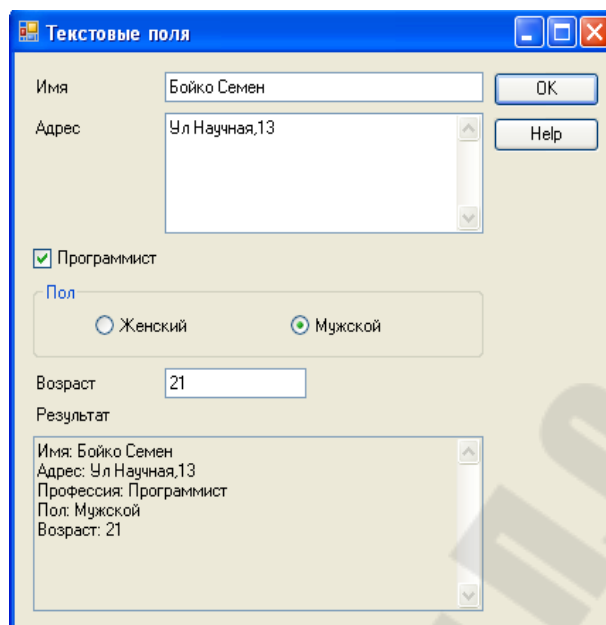


Рисунок 3.10 – Информация из текстовых полей и других элементов управления заносится в результирующее текстовое поле при нажатии кнопки ОК.

Вопросы к защите лабораторной работы

1. Что такое форма и ее назначение?
2. Что такое элементы управления? На какие группы они делятся?
3. Как установить элемент управления на форму и проинициализировать его?
4. Что такое событие? Как в .Net реализуются события?
5. Что такое обработчик события?
6. Как задаются обработчики событий для элементов управления?
7. Как создать верхнее меню?
8. Как добавить несколько пунктов меню?
9. Как установить определенному пункту меню сочетание клавиш?
10. Как создать панель инструментов?
11. Как добавить несколько кнопок на панель инструментов?
12. Как разместить на кнопке изображение?
13. Как добавить новую форму в приложение?
14. Как организовать переход к добавленной форме?
15. Что такое модальная и немодальная форма? Как они вызываются?

Лабораторная работа № 4

Тема: Элементы управления и привязка данных

Цель: Ознакомиться с элементами управления и выполнить привязку к данным.

Теоретические сведения

К стандартным элементам управления можно разрабатывать и собственные элементы управления.

Привязка данных – это логическая ассоциация между свойствами элементами управления и свойствами некоторого объекта.

При создании элементов управления, взаимодействующих с данными, иногда бывает нужно привязать элемент управления к типу, а не к объекту. Такая ситуация особенно часто возникает на этапе разработки, когда данные недоступны, но все равно нужно, чтобы элемент управления отображал данные из открытого интерфейса типа.

Кроме прямого добавления элементов в коллекцию Items компонентов ListBox и ComboBox мы также можем использовать механизм привязки данных.

Привязка данных в ListBox и ComboBox реализуется с помощью следующих свойств:

- DataSource: источник данных - какой-нибудь массив или коллекция объектов;
- DisplayMember: свойство объекта, которое будет использоваться для отображения в ListBox/ComboBox;
- ValueMember: свойство объекта, которое будет использоваться в качестве его значения.

Пример

Добавим в приложение (созданное в лабораторных работах 1-2) дополнительно новый пункт меню “Добавить” .

Добавим, в разрабатываем приложение, новую форму, на которой расположим элементы управления, в соответствии с рисунком, представленными на рисунке 4.1.

Добавив на форму элементы управления, оформив, как показано на рисунке 4.2.

Для поля, хранящего телефон, используем элемент MaskedTextBox, и используя свойство Mask, настраиваем соответствующие ограничения, позволяющие контролировать ввод пользователя и проверять его автоматически на наличие ошибок (рисунок 3.3).

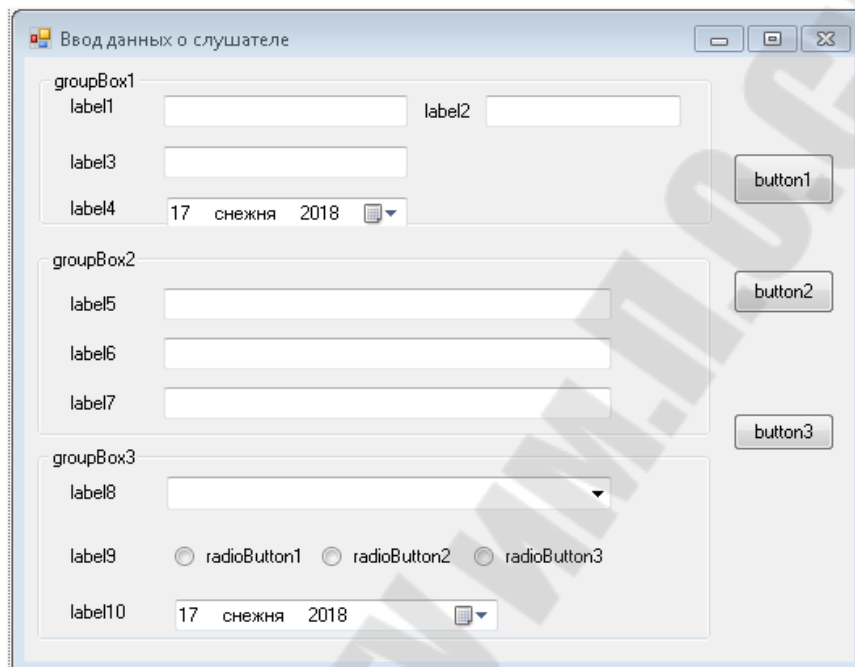


Рисунок 4.1 – Конструктор формы

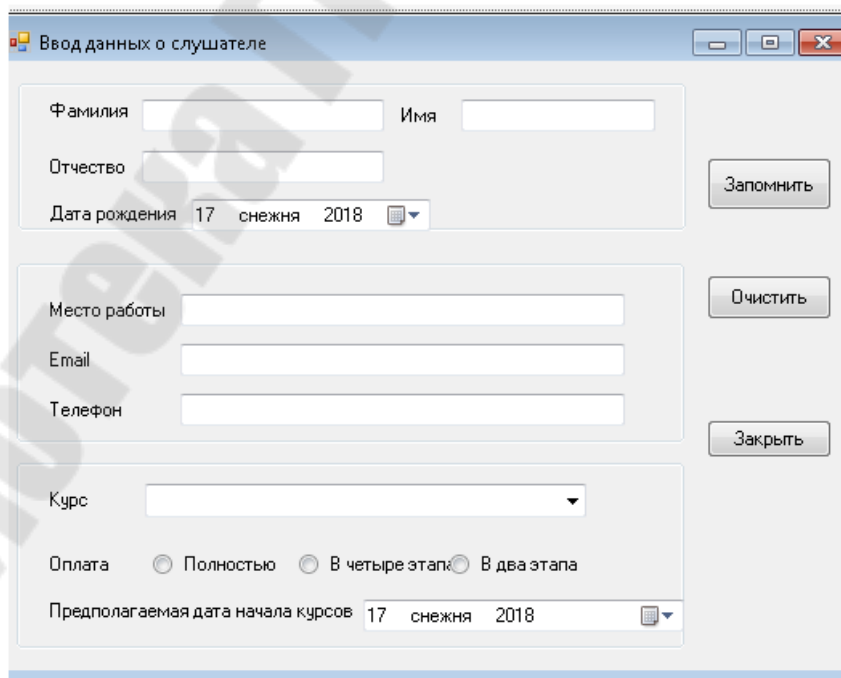


Рисунок 4.2 – Конструктор формы

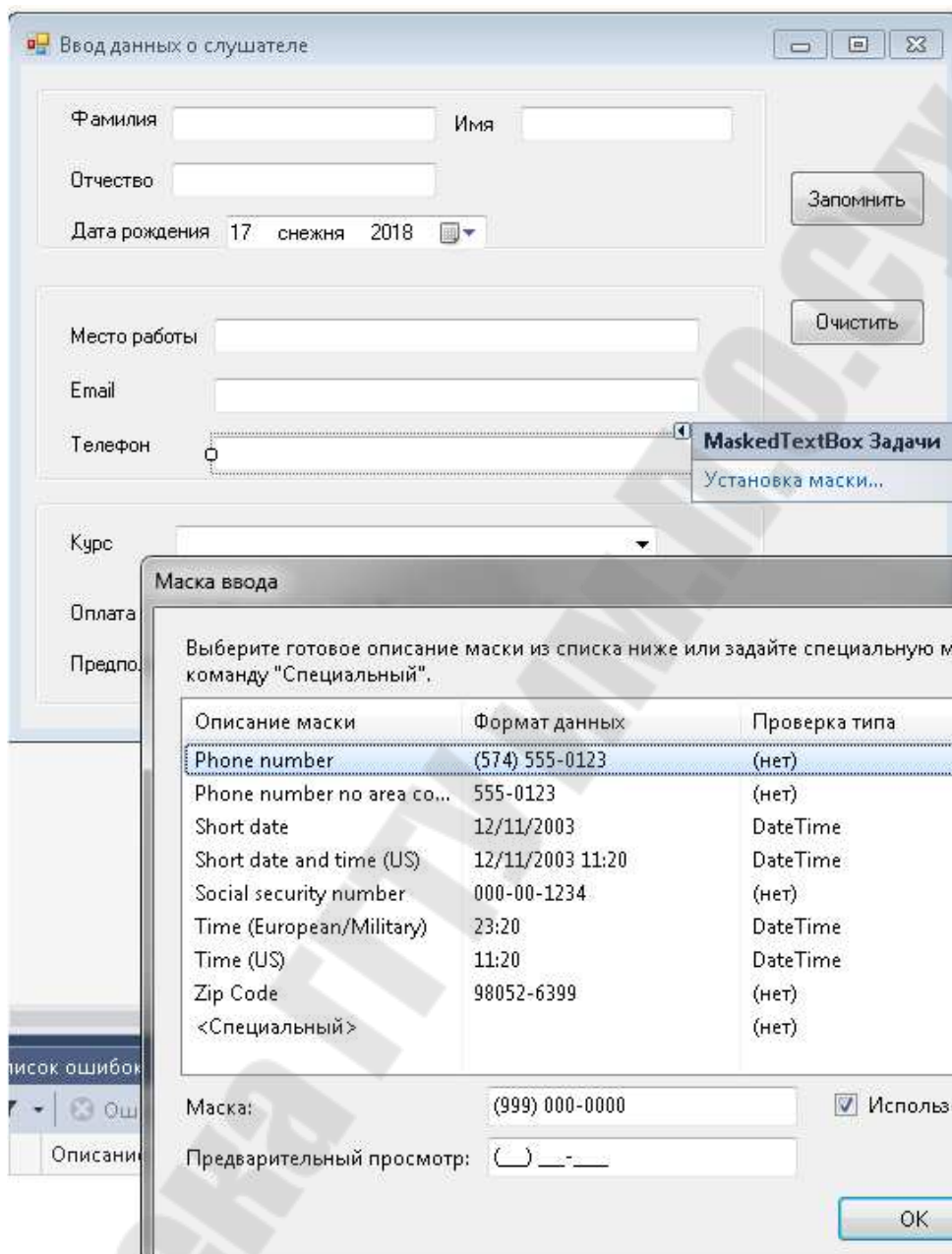


Рисунок 4.3 – Настройка маски

Для поля, хранящего курс, используем элемент ComboBox, и используя свойство Items, задаем элементы списка (рисунок 4.4).

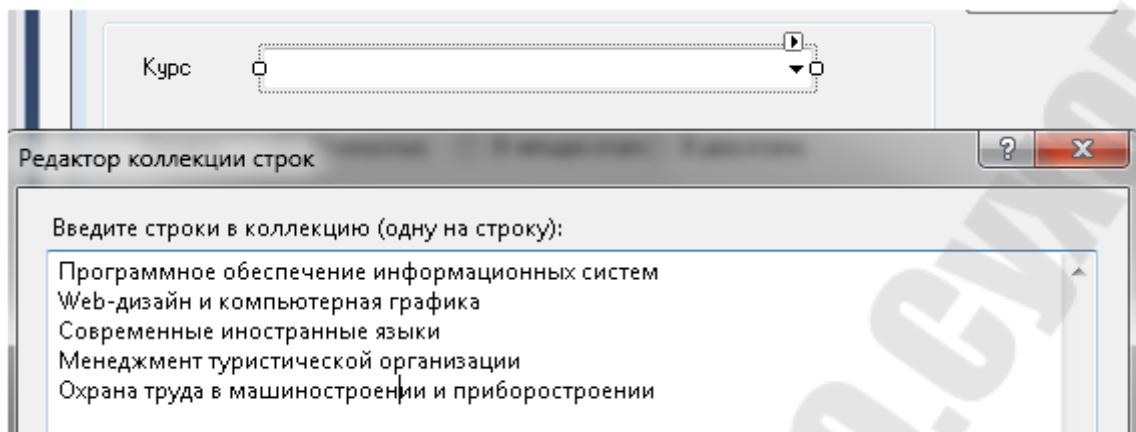


Рисунок 4.4 – Настройка элемента ComboBox

Для кнопки Закр^ыть определим код: `this.Close()`;
Для кнопки Очи^стить, код представлен на рисунке 4.5.

```
this.textBox1.Text = "";  
this.textBox2.Text = "";  
this.textBox3.Text = "";  
this.textBox4.Text = "";  
this.textBox5.Text = "";  
this.comboBox1.Text = "";  
this.maskedTextBox2.Text = "";
```

Рисунок 4.5 – Код кнопки “Очистить”

Задание

1. Выполните все примеры, приведенные в практической части лабораторной работы.
2. Измените первый пример следующим образом. Добавьте на форму, содержащую элементы Список и Дерево, кнопки для удаления выделенного (или выделенных) элементов из списка и дерева.
3. Для приложения, разработанного в лабораторных работах 1-2, реализовать форму добавления данных, через соответствующую форму и с использованием элементов управления.
4. Разработать дополнительные формы, необходимые для реализации приложения.

Вопросы к защите лабораторной работы

1. Что такое форма и ее назначение?
2. Что такое элементы управления? На какие группы они делятся?
3. Как установить элемент управления на форму и проинициализировать его?
4. Что такое событие? Как в .Net реализуются события?
5. Что такое обработчик события?
6. Как задаются обработчики событий для элементов управления?
7. Как создать верхнее меню?
8. Как добавить несколько пунктов меню?
9. Как установить определенному пункту меню сочетание клавиш?
10. Как создать панель инструментов?
11. Как добавить несколько кнопок на панель инструментов?
12. Как разместить на кнопке изображение?
13. Как добавить новую форму в приложение?
14. Как организовать переход к добавленной форме?
15. Что такое модальная и немодальная форма? Как они вызываются?

Лабораторная работа № 5

Тема: Доступ к источникам данных

Цель: Ознакомиться с компонентами доступа к источникам данных

Теоретические сведения

При создании базы данных данные сохраняются в таблицах – списках строк и столбцов, относящихся к конкретной области. Например, можно создать таблицу «Контакты» для сохранения имен, адресов и телефонных номеров или таблицу «Товары» для сохранения сведений об этих товарах. Определение структуры базы данных необходимо всегда начинать с создания ее таблиц. Таблицы создаются раньше любых других объектов базы данных.

Перед созданием таблиц необходимо тщательно проанализировать требования к базе данных и создать ее план, чтобы точно выяснить, какие таблицы нужны.

Таблица содержит данные по определенной теме, например, сведения о сотрудниках или товарах. Каждая запись в таблице включает данные об одном элементе, например о конкретном сотруднике. Запись состоит из полей и включает такие сведения, как имя, адрес и телефонный номер. Кроме того, запись обычно называется строкой, а поле – столбцом.

База данных может включать множество таблиц, в которых хранятся данные по различным темам. Каждая таблица может состоять из множества полей различного типа, включая текст, числа, даты и рисунки.

Компонент `BindingSource` выступает в качестве источника данных для некоторых или всех элементов управления в форме. В `Visual Studio` компонент `BindingSource` можно привязать к элементу управления с помощью свойства `DataBindings`, которое можно настроить в окне Свойства.

Можно привязать компонент `BindingSource` как к простым источникам данных, например к одному свойству объекта или базовой коллекции, как то `ArrayList`, так и к сложным источникам данных, например к таблице базы данных. Компонент `BindingSource` действует в качестве посредника, обеспечивающего привязку и

работу служб оперативного управления. Во время разработки или выполнения можно привязать компонент `BindingSource` к сложному источнику данных путем присвоения его свойствам `DataSource` и `DataMember` значений базы данных и таблицы соответственно. В следующем примере описывается использование компонента `BindingSource` в существующей архитектуре привязки данных.

Если начать добавление элементов к компоненту `BindingSource`, не указав сначала список для привязки, то компонент будет работать в качестве списочного источника данных и принимать добавляемые элементы.

Кроме того, можно написать код для реализации пользовательской функции "AddNew" с использованием события `AddingNew`, инициируемого при вызове метода `AddNew` перед добавлением элемента в список. Дополнительные сведения см. в разделе Архитектура компонента `BindingSource`.

Если необходимо перемещение по данным в форме, компонент `BindingNavigator` обеспечивает переход по данным и управление ими в координации с компонентом `BindingSource`.

Компонент `BindingSource` действует в качестве `CurrencyManager` для всех своих привязок, что позволяет ему обеспечить доступ к текущей информации и сведениям о позиции для данного источника данных.

Обычно пользователь работает с таким представлением источника данных, для которого можно выполнить упорядочение и фильтрацию.

Элемент управления `BindingNavigator` обеспечивает пользовательский интерфейс для перехода и управления элементами управления, привязанными к данным. Элемент управления `BindingNavigator` позволяет пользователям переходить по данным и управлять ими в элементе управления `Windows Form`.

Элемент управления `BindingNavigator` состоит из элемента `ToolStrip` с набором объектов `ToolStripItem` для большинства обычных действий с данными: добавления, удаления и перемещения по данным. По умолчанию элемент управления `BindingNavigator` содержит стандартные кнопки.

Каждому элементу управления данной коллекции соответствует член компонента `BindingSource`, обеспечивающий ту же функциональность программным путем. Например, кнопка `MoveFirstItem` соответствует методу `MoveFirst` компонента

BindingSource, кнопка DeleteItem соответствует методу RemoveCurrent и т.д.

Если кнопки по умолчанию не удовлетворяют требованиям приложения или если необходимо использовать дополнительные кнопки с иной функциональностью, существует возможность создания собственных кнопок ToolStrip.

Пример

1. Изучить свойства заданных компонент.
2. Открыть созданный ранее проект в л.р.№1. Добавить новый источник данных. (Проект – Добавить новый источник данных).
3. Добавить компонент bindingSource, указав в свойствах источник данных.
4. В приложении будет автоматически создан элемент DataSet (рисунок 5.1).

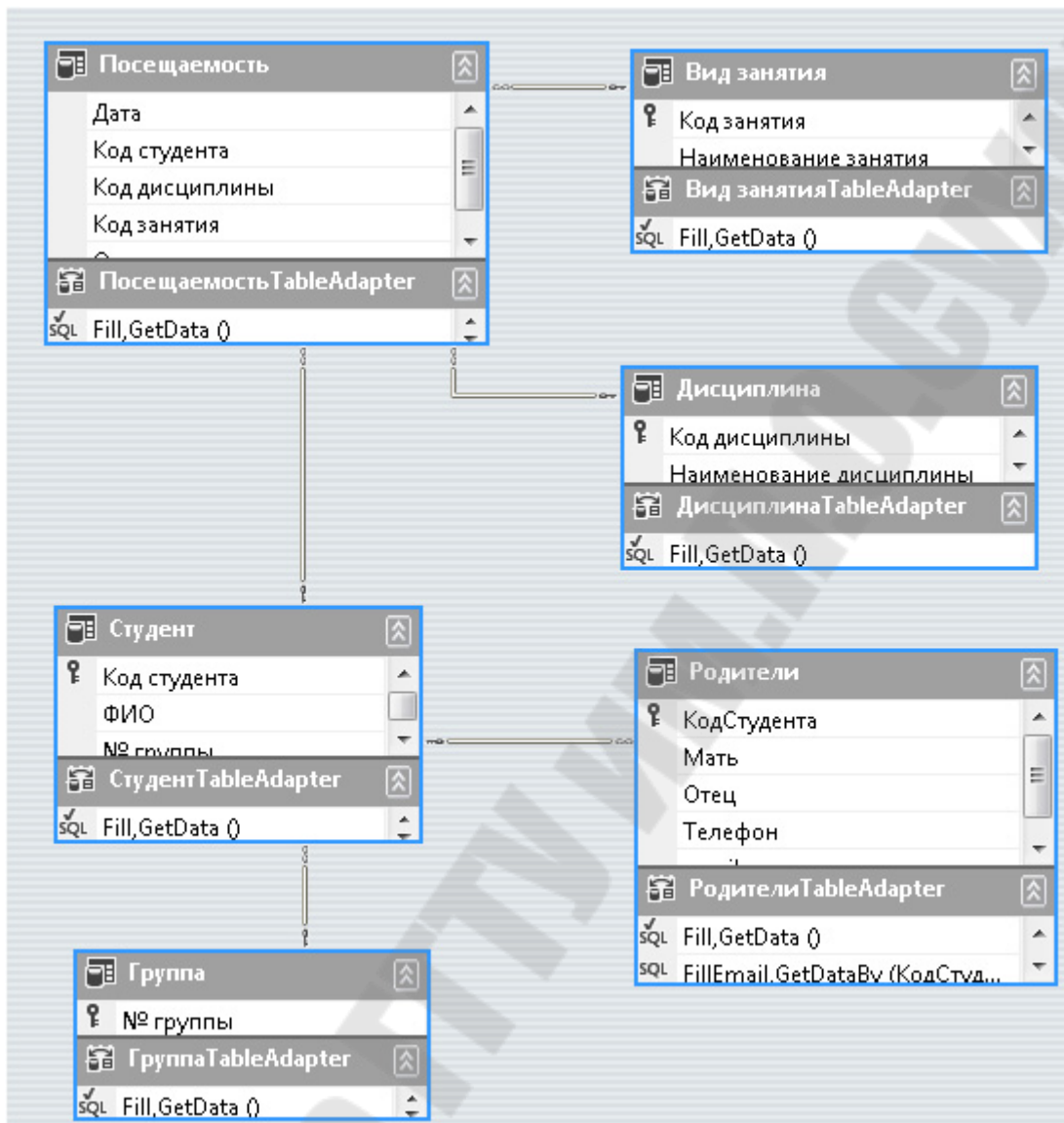


Рисунок 5.1 – Элемент DataSet

5. Добавить на форму компонент `dataGridView` и выполнить привязку к таблице БД (рисунок 5.2).

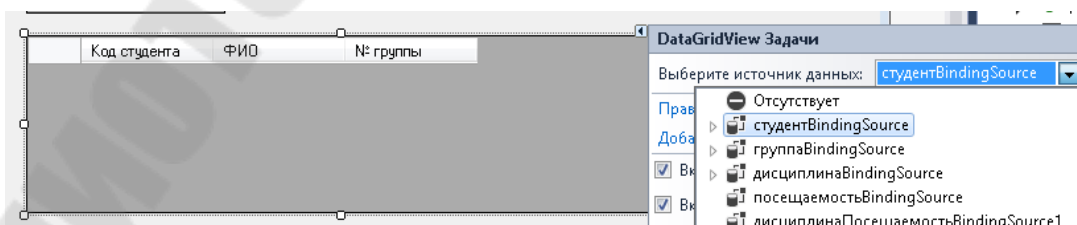


Рисунок 5.2 – Привязка таблицы базы данных к элементу `dataGridView`

Аналогично, можно добавить еще один компонент `dataGridView`, для отображения связи с второстепенной таблицей (рисунок 5.3).

6. Добавить на форму компонент `bindingNavigator` и установить свойство `BindingSource` и выполнить привязку к источнику данных.

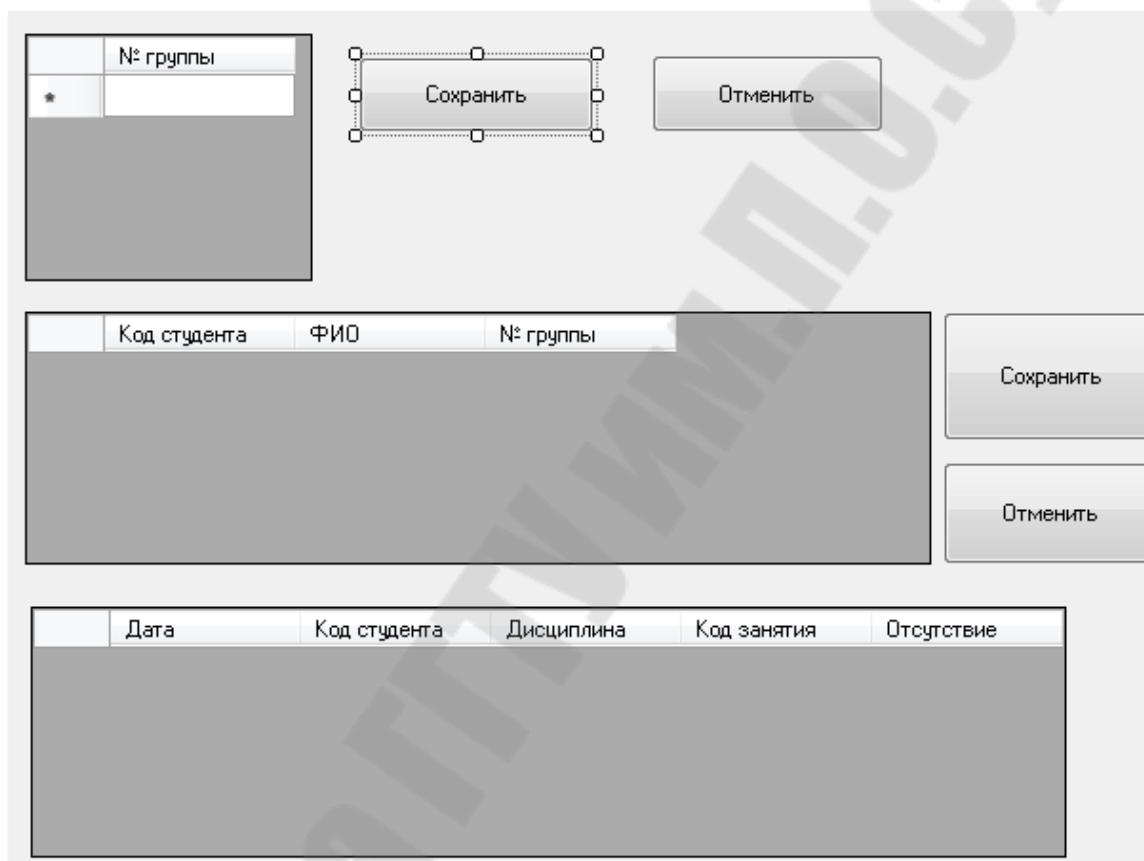


Рисунок 5.3 – Проектирование формы

Задание

1. Для заданной предметной области разработать структуру (полей) записей БД;
2. Создать файлы базы данных;
3. Заполнить начальными записями базу данных.
4. Разработать визуальное приложение, обеспечивающее взаимодействие набора данных с компонентами отображения данных. В качестве основного приложения взять наработки лабораторной работы № 1.

Вопросы к защите лабораторной работы

1. Понятие базы данных. Модели баз данных.
2. Архитектура СУБД.
3. Реляционные базы данных. Первичные ключи и индексы.
4. Избыточность данных.
5. Архитектура баз данных С#. Провайдер данных.
6. Типы данных поля, виды полей. Объекты для реальных полей
7. Проверка правильности введённого в поле значения.
8. Понятие и состояние набора данных.
9. Навигация по набору данных.
10. Компонент TDBNavigator.
11. Компонент TDBGrid. Создание объектов-столбцов.
12. Пустые столбцы. Формирование списка возможных значений столбца.
13. Управление отображением данных в TDBGrid.
14. Дополнительные возможности компонента TDBGrid.

Лабораторная работа № 6

Тема: Разработка приложения в визуальной среде

Цель: Разработать визуальное приложение, позволяющее добавлять, редактировать и удалять записи из БД. Научиться формировать вычисляемые поля, фильтры для отбора данных, организовывать поиск по значениям любого заданного столбца.

Теоретические сведения:

Основным объектом ADO является источник данных, представленный объектом DataSet. DataSet состоит из объектов типа DataTable и объектов DataRelation. В коде к ним можно обращаться как к свойствам объекта DataSet, то есть, используя точечную нотацию. Свойство Tables возвращает объект типа DataTableCollection, который содержит все объекты DataTable используемой базы данных.

Объекты DataTable используются для представления одной из таблиц базы данных в DataSet. В свою очередь, DataTable составляется из объектов DataColumn.

DataColumn – это блок для создания схемы DataTable. Каждый объект DataColumn имеет свойство DataType, которое определяет тип данных, содержащихся в каждом объекте DataColumn. Например, можно ограничить тип данных до целых, строковых и десятичных чисел. Поскольку данные, содержащиеся в DataTable, обычно переносятся обратно в исходный источник данных, необходимо согласовывать тип данных с источником.

Объект DataSet имеет также свойство Relations, возвращающее коллекцию DataRelationCollection, которая в свою очередь состоит из объектов DataRelation.

Каждый объект DataRelation выражает отношение между двумя таблицами (сами таблицы связаны по какому-либо полю (столбцу)). Следовательно, эта связь осуществляется через объект DataColumn.

Коллекция Rows объекта DataTable возвращает набор строк (записей) заданной таблицы. Эта коллекция используется для изучения результатов запроса к базе данных. Мы можем обращаться к записям таблицы как к элементам простого массива.

DataSet – это специализированный объект, содержащий образ базы данных. Для осуществления взаимодействия между DataSet и

собственно источником данных используется объект типа `DataAdapter`. Само название этого объекта – адаптер, преобразователь, – указывает на его природу. `DataAdapter` содержит метод `Fill()` для обновления данных из базы и заполнения `DataSet`.

Объект `DBConnection` осуществляет связь с источником данных. Эта связь может быть одновременно использована несколькими командными объектами.

Объект `DBCommand` позволяет послать базе данных команду (как правило, команду SQL или хранимую процедуру). Объекты `DBConnection` и `DBCommand` иногда создаются неявно в момент создания объекта `DataSet`, но их также можно создавать явным образом.

Когда в приложении пользователь работает с наборами данных `DataSet`, то фактически он оперирует с записями, оторванными от основной базы и находящимися в памяти компьютера. При такой модели доступа к данным возникает необходимость периодического обновления информации, содержащейся в основной базе данных. Когда пользователь вносит какие-либо изменения в записи таблицы объекта `DataSet`, то `DataSet` отслеживает и запоминает все эти изменения. По мере необходимости все сделанные изменения можно вернуть источнику данных. В течение процесса обновления источника данных происходит несколько событий, которые можно использовать для проверки правильности введенной пользователем информации.

Поскольку набор данных это фактически набор записей, оторванных от основной базы и помещенных в оперативную память компьютера, то процесс возврата изменений первоначальному источнику данных является изолированной процедурой и отделен от процесса модификации данных в `DataSet`.

Модификация источника данных через набор данных происходит в два этапа. На первом этапе модифицируется сам набор данных: пользователь добавляет новые записи, изменяет или удаляет существующие. Все эти процессы происходят в копии, оторванной от основного источника данных. На втором шаге необходимо послать эти изменения от оторванного набора данных к первоначальному источнику данных. То есть все изменения в наборе данных не передаются к основному источнику данных автоматически, этот процесс нужно активизировать явно. Это обычно достигается путем вызова метод `Update` адаптера данных.

Объекты типа DataAdapter позволяют проводить 4 основные операции в наборе данных соответствующих объектов БД.

TableAdapter позволяет работать с данными в наборе данных соответствующих таблиц БД через соответствующие свойства: SelectCommand — выбрать данные, Метод Fill — заполняет данными набор данных соответствующих таблиц БД.

Он реализует выборку данных их соответствующих таблиц БД.

Пример

Для реализации вывода таблиц, которые связаны между собой, добавим несколько элементов управления dataGridView, как показано на рисунке 6.1.

The screenshot displays a Windows application form with three data grids and control buttons. The top grid shows a list of group numbers, with 'ЗТМ-12' selected. The middle grid shows student details, with the first row containing '21', 'Иванов И.И.', and 'ЗТМ-12'. The bottom grid shows attendance records, with the first row containing '20.04.2018', '21', '1', '1', and an unchecked checkbox. Control buttons 'Сохранить' and 'Отменить' are present next to each grid.

	№ группы
▶	ЗТМ-12
	ЗТМ-11
	ЗО-11
	ЗЧА-11

	Код студента	ФИО	№ группы
▶	21	Иванов И.И.	ЗТМ-12
*			

	Дата	Код студента	Дисциплина	Код занятия	Отсутствие
▶	20.04.2018	21	1	1	<input type="checkbox"/>
*					<input type="checkbox"/>

Рисунок 6.1 – Форма приложения

При внесении изменений все новые данные остаются только на форме. В базе данных они не сохраняются, и при повторном вызове приложения, конечно же, будут отсутствовать. Это происходит потому, что данные были загружены в объект DataSet, который

представляет собой копию таблицы в памяти. Все действия выполняются с этой копией. Для того чтобы изменения отобразились в базе данных, необходимо выполнить метод Update класса TableAdapter.

Так как таблицы, отображенные на форме, связаны между собой, необходимо это учитывать при обновлении, изменении и удалении данных.

В случае, если мы учитываем вышеизложенные модификации без каскадного обновления, для этого реализуем код, представленный на рисунке 6.2 – 6.3.

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        this.посещаемостьTableAdapter.Update(db1DataSet.Посещаемость);
        this.студентTableAdapter.Update(db1DataSet.Студент);
        this.группаTableAdapter.Update(db1DataSet.Группа);
    }
    catch (Exception error)
    {
        MessageBox.Show(error.Message);
    }
}
```

Рисунок 6.2 – Код кнопки “Сохранить”

```
private void button2_Click(object sender, EventArgs e)
{
    this.db1DataSet.Группа.RejectChanges();
}
```

Рисунок 6.3 – Код кнопки “Отменить”

Однако, в данном случае, может возникнуть исключительная ситуация, которая не позволит обновить данные в базе данных.

Для этого необходимо, для каждого dataGridView, обработать событие CellEndEdit, которое позволит обновлять вводимые данные в dataGridView и сохранять в базе данных.

Но необходимо, при создании базы данных указать каскадное обновление данных в таблице.

Поэтому, учитывая каскадное изменение во всех взаимосвязанных таблицах, рассматриваем код, представленный на рисунках 6.4 – 6.5.

```
private void button3_Click(object sender, EventArgs e)
{
    //---если в наборе данных что-то менялось
    if (this.db1DataSet.HasChanges())
    {
        //---получаем удаленные записи в дочерней таблице
        //---и записываем в БД
        DataTable dtDelChild = db1DataSet.Студент.GetChanges(DataRowState.Deleted);
        if (dtDelChild != null)
        {
            this.студентTableAdapter.Update(db1DataSet.Студент);
            dtDelChild.Dispose();
        }
        //---записываем в БД изменения в родительской таблице
        DataTable dtAnyChangesInParent = db1DataSet.Группна.GetChanges();
        if (dtAnyChangesInParent != null)
        {
            this.группнаTableAdapter.Update(db1DataSet.Группна);
            dtAnyChangesInParent.Dispose();
        }
        //---получаем добавленные записи в дочерней таблице
        DataTable dtAddChild = db1DataSet.Студент.GetChanges(DataRowState.Added);
        if (dtAddChild != null)
        {
            this.студентTableAdapter.Update(db1DataSet.Студент);
            dtAddChild.Dispose();
        }
        //---получаем измененные записи в дочерней таблице
        DataTable dtEditChild = db1DataSet.Студент.GetChanges(DataRowState.Modified);
        if (dtEditChild != null)
        {
            this.студентTableAdapter.Update(db1DataSet.Студент);
            dtEditChild.Dispose();
        }
        //---подтверждаем изменения
        db1DataSet.AcceptChanges();
    }
}
```

Рисунок 6.4 – Код кнопки “Сохранить”

```

private void button7_Click(object sender, EventArgs e)
{
    //---если в наборе данных что-то менялось
    if (db1DataSet.HasChanges())
    {
        //---отключаем ограничения
        db1DataSet.EnforceConstraints = false;
        //---отменяем изменения в таблицах
        db1DataSet.Группа.RejectChanges();
        db1DataSet.Студент.RejectChanges();
        //---включаем ограничения
        db1DataSet.EnforceConstraints = true;
    }
}

```

Рисунок 6.5 – Код кнопки “Отменить”

Привязывать элементы пользовательского интерфейса можно отнюдь не исключительно к таблично представленным данным. Практически любая структура данных может выступить в роли их источника – обычные объекты, массивы, коллекции и т.д.

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

Windows Forms позволяют отображать данные путем привязки их элементов управления к источникам данных. Привязка данных обычно используется для отображения результатов поиска, детализации сводок, генерации отчетов и ввода данных.

BindingSource представляет собой промежуточный слой между источником данных и соответствующим элементом управления (controlom), к нему привязанным.

Допустим, у нас есть Label, TextBox и ComboBox, привязанные к таблице (DataTable). Предположим, что для обновления информации был создан и заполнен данными из БД новый DataTable. Встает несложная, в общем то, задача – сменить у всех трех controlov источник данных. Теперь можно изменить свойства DataSource/DataMember единственного BindingSource, а не у всех controlov.

Для получения значений групповых операций из базы данных, необходимо для этого предусмотреть поля, для вывода результатов (рисунок 6.6) и соответствующий запрос.

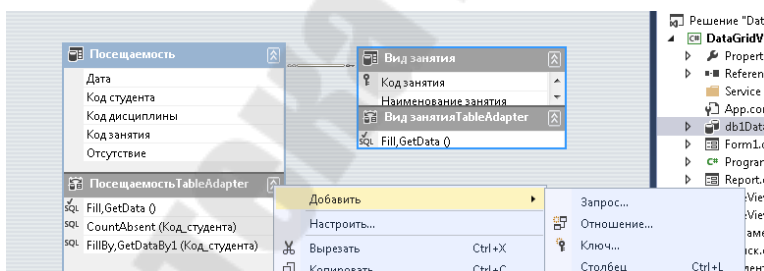
Дата	Код студента	Дисциплины	Занятие	Отсутствие
01.04.2009	1	математика	лекция	<input type="checkbox"/>
02.04.2009	1	математика	лекция	<input type="checkbox"/>
03.04.2009	1	физика	лабораторная р...	<input checked="" type="checkbox"/>

Рисунок 6.6 – Форма приложения

Для создание вычисляемого поля, добавили запрос, выполняющий подсчет количества пропущенных занятий (рисунки 6.7, 6.8).

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    try
    {
        label2.Text = Convert.ToString(посещаемостьTableAdapter.CountAbsent(Convert.ToInt32(код_студентаTextBox.Text)));
    }
    catch (Exception er) { MessageBox.Show(er.Message); }
}
```

Рисунок 6.7 – Код реализации работы запроса



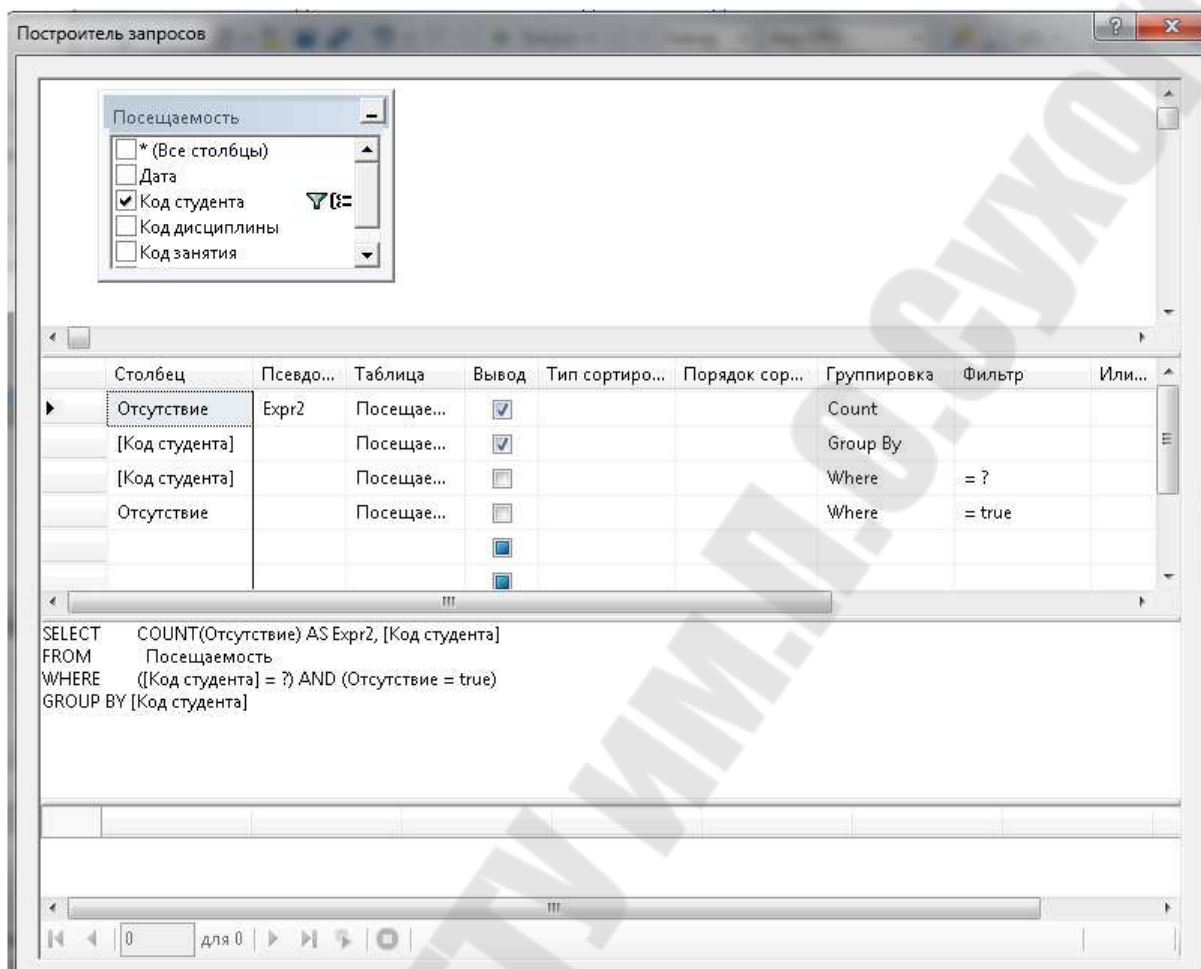


Рисунок 6.8 – Разработка запроса

Для реализации возможности формирования электронного сообщения, используем соответствующий код (рисунок 6.9).

```

private void button1_Click(object sender, EventArgs e)
{
    string email = this.emailTextBox.Text;
    MailMessage mail = new MailMessage();
    mail.From = new MailAddress("gridina_elenabk.ru");
    mail.To.Add(email);
    mail.Subject = "Информация о посещении занятий";
    mail.Body = "Свяжитесь с кафедрой";

    SmtplibClient SmtplibServer = new SmtplibClient("smtp.mail.ru");
    SmtplibServer.Port = 25;
    SmtplibServer.Credentials = new NetworkCredential("email@bk.ru", "password");
    SmtplibServer.EnableSsl = true;

    try
    {
        SmtplibServer.Send(mail);
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message);
    }
}

```

Рисунок 6.9 – Код, реализующий создание электронного письма

Для реализации поиска необходимой информации, реализуем соответствующий интерфейс (рисунок 6.10).

Код студента	ФИО	№ группы
1	Иванов И.И.	30-11
2	Петров П.П.	30-11
3	Семченко П.Р.	30-11
4	Бобр С.П.	30-11
5	Бондарчук А.П.	30-11

ПОИСК (выделение первой записи, удовлетворяющей требованиям)

Бобр

ФИЛЬТРАЦИЯ данных

Рисунок 6.10 – Форма “Поиск”

И разработан соответствующий код для реализации поиска (рисунок 6.11).

```
private void button1_Click(object sender, EventArgs e)
{
    for (int i = 0; i < студентDataGridView.RowCount; i++)
    {
        this.студентDataGridView.Rows[i].Selected = false;
        for (int j = 0; j < студентDataGridView.ColumnCount; j++)
            if (студентDataGridView.Rows[i].Cells[j].Value != null)
                if (студентDataGridView.Rows[i].Cells[j].Value.ToString().Contains(textBox1.Text))
                {
                    студентDataGridView.Rows[i].Selected = true;
                    break;
                }
    }
}
```

Рисунок 6.11 – Код формы “Поиск”

Так же не забываем про фильтрацию данных (рисунки 6.12 – 6.13).

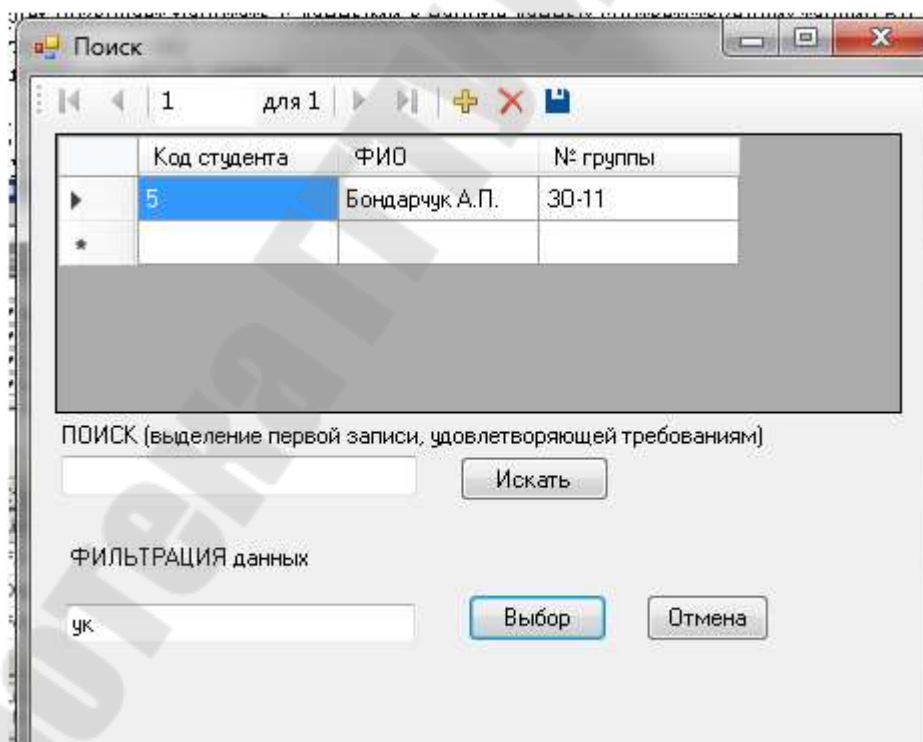


Рисунок 6.12 – Результат фильтрации данных

```
private void button2_Click(object sender, EventArgs e)
{
    студентBindingSource.Filter = String.Format("ФИО LIKE '{0}%", textBox2.Text);
}

```

ссылка 1

```
private void button3_Click(object sender, EventArgs e)
{
    студентBindingSource.Filter = "";
}

```

Рисунок 6.13 – Код реализации фильтрации данных

Задание

Добавить в приложение, разработанное в лабораторной работе №1-5, функционал, позволяющий добавлять, редактировать, удалять записи из БД, выполнять фильтрацию и поиск данных.

Вопросы к защите лабораторной работы

1. Для чего предназначен DataSet?
2. Какие коллекции DataSet Вам известны?
3. Как может быть поддержано изменение в наборе данных ?
4. Дайте характеристику объекту DataRow, опишите его свойства и методы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Артемов Д.В. Microsoft SQL Server 2000 / Д.В.Артемов. – М. : Русская Редакция, 2001. - 555с.
2. Биллиг В.А. Основы программирования на С#/ – М.: Изд-во «Интернет-университет информационных технологий – ИНТУ-ИТ.ру», 2006. – 488с.
3. Герман, О. В. Программирование на JAVA и С# для студента / О. В. Герман, Ю. О. Герман. - Санкт-Петербург : БХВ-Петербург, 2005. - 511 с.
4. Осипов Н.А. Разработка Windows приложений на С# – СПб: НИУ ИТМО, 2012. – 74 с.
5. Павловская Т.А. С#. Программирование на языке высокого уровня: Учебник для вузов. – СПб.: Питер, 2007. – 432с.
6. Рихтер, Дж. CLR via С#. Программирование на платформе Microsoft .NET Framework 4.5 на языке С# / Джеффри Рихтер ; [перевел с англ. Е. Матвеев]. - 4-е изд.. - Санкт-Петербург [и др.] : Питер, 2014. - 895 с.
7. Троелсен, Э. С# и платформа . NET [Текст]: библиотека программиста / Э. Троелсен. – Санкт-Петербург: ООО «Питер-Пресс», 2007. – 796 с.
8. Шилдт Г.С. Полный справочник по С#. – М.: Издательский дом «Вильямс», 2004. – 752с.
9. Шилдт Г.С. С#: Учебный курс. – СПб.:Питер, 2002. – 512с.
10. Каталог API (Microsoft) и справочных материалов [Электронный ресурс] – Режим доступа: <https://msdn.microsoft.com/library>

ПРИЛОЖЕНИЕ А
(справочное)

ВАРИАНТЫ ЗАДАНИЙ

1. Предметная область: **Страховое агентство**

Информационные объекты:

Штаты (Табельный номер сотрудника, ФИО, Должность, Образование, Стаж работы, Оклад)

Виды страхования (Код страховки, Название страховки, Примечание)

Список застрахованных (Код застрахованного, ФИО застрахованного, Адрес, Код страховки, Страховая сумма, Табельный номер сотрудника, Дата страховки)

Страховые взносы (Код застрахованного, Период страховки, Страховой взнос, Дата взноса)

Страховые возмещения (Код застрахованного, Код страховки, Сумма выплаты, Дата выплаты)

2. Предметная область: **Фирма комиссионной продажи автомобилей**

Информационные объекты:

Марка авто (Код марки, Название, Производитель)

Цвет авто (Код цвета, Название)

Продажа авто (Номер авто, Код марки, Код цвета, Номер владельца, Дата выпуска, Дата приема на продажу, Дата продажи, Цена, Код покупателя, Процент фирме)

Владельцы (Код владельца, ФИО, Адрес)

Покупатели (Код покупателя, ФИО, Адрес)

3. Предметная область: **Банк**

Информационные объекты:

Штаты (Код должности, Должность, Оклад, Количество должностных единиц)

Персонал (Табельный номер сотрудника, ФИО, Код должности, Адрес, Телефон)

Вклады (Вид вклада, Номер счета, Дата создания, Вид валюты)

Клиенты (Номер счета, ФИО, Адрес)

Операции (Код операции, Наименование)

Движение денег (Номер движения, Номер счета, Код операции, Дата, Табельный номер сотрудника, Сумма операции)

4. Предметная область: **Отдел сбыта предприятия**

Информационные объекты:

Изделие (Код изделия, Наименование, Цена, Срок реализации)

Склад (Код поступления, Код изделия, Дата изготовления, Количество)

Сотрудники отдела (Табельный номер сотрудника, ФИО, Код должности, Код категории)

Должности (Код должности, Название, Оклад)

Категория (Код категории, Название)

Покупатели (Код покупателя, Название, Адрес)

Продажа (Код продажи, Дата продажи, Код покупателя, Код изделия, Количество, Код сотрудника)

5. Предметная область: **Школа**

Информационные объекты:

Учитель (Код учителя, ФИО, Адрес, Семейное положение, Наличие детей, Стаж, Код категории, Код звания)

Категория (Код категории, Название)

Звание (Код звания, звание)

Оплата (Код оплаты, Код категории, ставка, премия)

Доплата (Код доплаты, Наименование, Размер)

Начисление (Код начисления, Код учителя, Код оплаты, Количество часов в неделю, Код доплаты)

6. Предметная область: **Киоск**

Информационные объекты:

Виды товаров (Код вида, Название вида)

Товары (Код товара, Код вида, Название, Цена)

Поступление товаров (Код поступления, Код товара, Дата поступления, Количество)

Продажа товаров (Код продажи, Код товара, Дата продажи, Количество)

7. Предметная область: **Научно-исследовательский институт (НИИ)**

Информационные объекты:

Отдел (Номер отдела, Название, Надбавка, Тема, Дата окончания работы)

Сотрудники (Табельный номер, ФИО, Номер отдела, Код должности, Код звания, Код степени, Стаж, Процент надбавки)

Должности (Код должности, Название, Оклад)

Звание (Код звания, Звание)

Степень (Код степени, степень)

8. Предметная область: **Книжный магазин**

Информационные объекты:

Область знаний (Код области знаний, Название)

Книги (Код области знаний, Индекс в каталоге, Авторы, Название, Год выпуска, Цена)

Поступление книг (Код поступления, Код области знаний, Индекс в каталоге, Количество, Дата поступления)

Продажа книг (Код продажи, Код области знаний, Индекс в каталоге, Количество, Дата продажи)

9. Предметная область: **Склад товаров**

Информационные объекты:

Товар (Код товара, Наименование, Единица измерения, Стоимость)

Поступление товаров (Код поступления, Код товара, Дата, Количество, Номер накладной)

Выдача товаров (Код выдачи, Код товара, Количество, Дата, Код МОЛ)

МОЛ (Код МОЛ, ФИО)

10. Предметная область: **Общежитие**

Информационные объекты:

Штаты (Код должности, Название, Количество штатных единиц, зарплата)

Персонал (Табельный номер сотрудника, ФИО, Код должности, Кабинет)

Студент (Код студента, ФИО, Номер комнаты, Домашний адрес, Код факультета, Курс, Группа)

Факультет (Код факультета, Название, Декан, Телефон деканата)

11. Предметная область: **Аптечный киоск**

Информационные объекты:

Готовые лекарства (Код лекарства, Название, Срок реализации)

Поступление лекарств (Код поступления, Код лекарства, Цена, Количество, Дата)

Продажа (Код продажи, Код лекарства, Количество, Дата)

12. Предметная область: **Кассы ж/д вокзала**

Информационные объекты:

Поезд (Номер операции, Номер поезда, дата отправления, количество проданных билетов, цена одного билета, тип вагона)

Пассажир (Код пассажира, ФИО пассажира, Номер поезда, пункт отправления, пункт назначения)

Багаж (Код пассажира, Количество багажа, Количество дополнительных мест для багажа)

13. Предметная область: **Гаражный кооператив**

Информационные объекты:

Кооператив (Код кооператива, Адрес, Количество гаражей, Председатель, Телефон)

Члены кооператива (Код кооператива, Номер гаража, ФИО владельца, Дата вступления в кооператив, Марка авто, Номер авто)

Электроэнергия (Код кооператива, Номер гаража, Предыдущее показание счетчика, Текущее показание, Дата уплаты, Стоимость за единицу)

14. Предметная область: **Фирма-продавец компьютерных комплектующих**

Информационные объекты:

Комплектующие (Код комплектующего, Название, Производитель, Дата изготовления, Гарантийный срок, Цена)

Продажа (Номер чека, Код комплектующего, Дата продажи, количество)

Покупатели (Номер чека, ФИО покупателя, Адрес)

Гридина Елена Ивановна

**СРЕДСТВА ВИЗУАЛЬНОГО
ПРОГРАММИРОВАНИЯ ПРИЛОЖЕНИЙ**

**Практикум
для слушателей специальности переподготовки
1-40 01 73 «Программное обеспечение
информационных систем»
заочной формы обучения**

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 18.02.19.

Пер. № 8Е.

<http://www.gstu.by>