МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Н.А. Осипов

Разработка Windows приложений на C#

Учебное пособие

Санкт-Петербург

2012

УДК 004.655, 004.657, 004.62

Н.А. Осипов

Разработка Windows приложений на C# - СПб: НИУ ИТМО, 2012. – 74 с.

В пособии излагаются основы разработки Windows приложений и методические указания к выполнению лабораторных работ по дисциплине «Технологии программирования».

Предназначено для студентов, обучающихся по всем профилям подготовки бакалавров направления: 210700 Инфокоммуникационные технологии и системы связи.

Рекомендовано к печати Ученым советом факультета Инфокоммуникационных технологий, протокол № 4 от 13 декабря 2011г.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Оглавление

Введение 5

Лабораторная работа 1. Введение в разработку форм 6
Упражнение 1. Настройка прямоугольной формы Windows 6
Упражнение 2. Создание непрямоугольной формы Windows 9
Упражнение 3. Создание наследуемой формы 9
Упражнение 4. Создание MDI-приложения 10

Лабораторная работа 2. Работа с элементами управления 14
Упражнение 1. Обработка событий Click и MouseMove 14
Упражнение 2. Работа со списками 15
Упражнение 3. Создание и использование элемента управления ToolStrip 17
Упражнение 4. Использование элемента управления StatusStrip 19
Упражнение 5. Работа с контейнерными элементами управления 19
Упражнение 6. Элементы с поддержкой отображения текста 22
Упражнение 7. Элементы с поддержкой редактирования текста 24
Упражнение 8. Добавление и удаление элементов управления в режиме работы приложения 25
Упражнение 9. Проверка вводимых значений. События KeyPress и Validating. Элемент управления ErrorProvider 26

Лабораторная работа 3. Создание элементов управления 28
Упражнение 1. Создание составного элемента управления 28
Упражнение 2. Создание специализированного элемента управления 30
Упражнение 3. Создание расширенных элементов управления 31

Лабораторная работа 4. Использование окон диалога в формах 32
Упражнение 1. Использование компонента SaveFileDialog 32
Упражнение 2. Использование компонента ColorDialog 33
Упражнение 3. Использование компонента FontDialog 34
Упражнение 4. Использование компонента OpenFileDialog 34

Лабораторная работа 5. Взаимодействие управляемого и неуправляемого кода 35
Упражнение 1. Использование COM-компонента для создания PDF-приложения 35
Упражнение 2. Вызов функции API 36

Лабораторная работа 6. Организация печати в формах windows 39
Упражнение 1. Использование диалоговых окон для печати 39
Упражнение 2. Создание документа печати 40
Упражнение 3. Создание специализированной формы предварительного просмотра 41

Лабораторная работа 7. Асинхронное программирование 43
Упражнение 1. Работа с компонентом BackgroundWorker 43

Упражнение 2. Использование делегатов 45
Упражнение 3. Асинхронный запуск произвольного метода 47

Лабораторная работа 8. Повышение удобства использования приложений 49
Упражнение 1. Создание контекстной справки 49
Упражнение 2. Использование справочного файла 50
Упражнение 3. Добавление всплывающих подсказок 51
Упражнение 4. Автоматический выбор языка при запуске приложения 51
Упражнение 5. Локализация приложения 52

Лабораторная работа 9. Развертывание windows приложений 55
Упражнение 1. Использование строго именованной сборки 55
Упражнение 2. Работа с глобальным кэшем сборок 56
Упражнение 3. Создание и использование файлов конфигурации приложения 58
Упражнение 4. Создание и использование Windows Installer Setup Project 59
Упражнение 5. Публикация приложения с помощью ClickOnce в сетевой папке 61

Лабораторная работа 10. Подключение к базе данных 62
Упражнение 1. Организация доступа к данным и работа с объектом DataReader 62
Упражнение 2. Извлечение и обновление данных с помощью объектов DataAdapter и DataSet 64
Упражнение 3. Использование объектов DataView 65
Упражнение 4. Связывание данных с элементами управления 66
Упражнение 5. Создание связанной с данными формы в мастере источников данных 68

Список литературы 69

Введение

В результате изучения дисциплины «Технологии программирования» студенты познакомятся:

- с базовыми концепциями и терминологией объектно-ориентированного программирования;
- с основами разработки форм Windows;
- с инструментальными средствами разработки программного обеспечения;

и приобретут навыки:

- разработки технических заданий на проектирование программного обеспечения;
- разработки алгоритма и реализации программного обеспечения на основе современных средств Microsoft Visual Studio;
- использования стандартных средств отладки программ.

По окончании обучения студенты смогут:

- понимать основные элементы .NET Framework и связь C# с элементами платформы .NET;
- работать в среде разработки Microsoft Visual Studio;
- создавать, отлаживать, компилировать и выполнять программы;
- создавать и использовать компоненты пользовательского интерфейса;
- использовать стандартные элементы управления и применять их для контроля данных;
- создавать документы, реализующие печать форм;
- создавать приложения, использующие ранее разработанные модули и функции API;
- знать базовые концепции и терминологию объектно-ориентированного программирования;
- создавать, приложения, реализующие многопоточную технологию программирования;
- использовать дополнительные возможности, повышающие удобство работы для пользователя при взаимодействии с приложением;
- развертывать созданные приложения и использовать стандартные и пользовательские атрибуты;
- реализовывать возможность подключения к базе данных;

В основном при выполнении упражнений приложения создаются обучаемыми с нуля. Для изучения наиболее сложных вопросов используются готовые стартовые проекты. Исходные файлы для выполнения таких заданий находятся в сетевой папке \\atec\student\Work\C#\Windows_App. Скопируйте их на локальную машину перед выполнением упражнений.

Лабораторная работа 1. Введение в разработку форм

Цель работы

Изучение методов построения форм Windows и получение навыков по настройке форм, созданию непрямоугольных и наследуемых (производных) форм.

Упражнение 1. Настройка прямоугольной формы Windows

Формы Windows — это основной компонент пользовательского интерфейса. Формы предоставляют контейнер, который содержит элементы управления, меню и позволяет отображать приложение в уже привычной и единообразной модели. Формы могут реагировать на события мыши и клавиатуры, поступающие от пользователя, и выводить на экран данные для пользователя с помощью элементов управления, которые содержатся в форме.

Формы Windows содержат множество свойств, позволяющих настраивать их внешний вид и поведение. Просматривать и изменять эти свойства можно в окне **Properties** конструктора при разработке, а также программно во время выполнения приложения.

В следующей таблице перечислены некоторые свойства форм Windows, отвечающие за внешний вид и поведение приложения:

Свойство	**Описание**
Name	Задает имя классу **Form**, показанному в конструкторе. Данное свойство задается исключительно во время разработки
BackColor	Указывает цвет фона формы
Enabled	Указывает, может ли форма принимать ввод от пользователя. Если свойству **Enabled** задано значение **False**, все элементы управления формы также блокируются
ForeColor	Указывает цвет переднего плана формы, то есть цвет выводимого текста. Если отдельно не указать значение свойства **ForeColor** элементов управления формы, они примут то же значение
FormBorderStyle	Указывает вид и поведение границы и строки заголовка формы Значения свойства:
	None - Форма не имеет границы, не может быть минимизирована или развернута до максимальных размеров и у нее нет экранной кнопки управления окном и кнопки справки
	FixedSingle - Форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. Форма

Свойство	Описание
	может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки или кнопку управления окном, что определяется остальными свойствами
	Fixed3D - Форма имеет объемную границу, и размеры формы нельзя изменить во время выполнения. Форма может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки или кнопку управления окном, что определяется остальными свойствами
	FixedDialog - Форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. У формы нет экранной кнопки управления окном, но может быть кнопка справки, что определяется остальными свойствами. Форму можно минимизировать и развернуть до максимальных размеров
	Sizable - Форма имеет настройки по умолчанию, но они могут изменяться пользователем. Форма может быть минимизирована, развернута до максимальных размеров, и иметь кнопку справки, что определяется остальными свойствами
	FixedToolWindow - Форма имеет тонкую границу, и размеры формы нельзя изменить во время выполнения. Форма содержит только кнопку закрытия
	SizableToolWindow - Форма имеет тонкую границу, и размеры формы могут быть изменены пользователем. Форма содержит только кнопку закрытия
Location	Когда свойству **StartPosition** задано значение **Manual**, это свойство указывает исходное положение формы относительно верхнего левого угла экрана
MaximizeBox	Указывает, есть ли у формы кнопка **MaximizeBox**
MaximumSize	Устанавливает максимальный размер формы. Если задать этому свойству размер 0; 0, у формы не будет верхнего ограничения размера
MinimizeBox	Указывает, есть ли у формы кнопка **MinimizeBox**
MinimumSize	Устанавливает минимальный размер формы, который пользователь может задать
Opacity	Устанавливает уровень непрозрачности или прозрачности формы от 0 до 100%. Форма, непрозрачность которой составляет 100%, полностью непрозрачна, а форма, имеющая 0 % непрозрачности, наоборот, полностью прозрачна

Свойство	Описание
Size	Принимает и устанавливает исходный размер формы
StartPosition	Указывает положение формы в момент ее первого выведения на экран
Text	Указывает заголовок формы
TopMost	Указывает, всегда ли форма отображается поверх всех остальных форм, свойству **TopMost** которых не задано значение **True**
Visible	Указывает, видима ли форма во время работы
WindowState	Указывает, является ли форма минимизированной, развернутой до максимальных размеров, или же при первом появлении ей задан размер, указанный в свойстве **Size**

Создание нового проекта

1. Откройте Visual Studio и создайте новый проект Windows Forms. Проект откроется с формой по умолчанию с именем **Form1** в конструкторе.

2. Выберите форму в конструкторе. Свойства формы отображаются в окне **Properties**.

3. В окне **Properties** задайте свойствам значения, как указано ниже:

Свойство	Значение
Text	Trey Research
FormBorderStyle	Fixed3D
StartPosition	Manual
Location	100; 200
Opacity	75%

4. Перетащите три кнопки из **Toolbox** в форму и разместите их так, как вам будет удобно.

5. Поочередно выберите каждую кнопку и в окне **Properties** задайте свойству кнопок **Text** значения **Border Style, Resize** и **Opacity.**

6. Для кнопки **Border Style** задайте свойство **Anchor – Top, Left.**

Реализация обработчиков событий

7. В конструкторе дважды щелкните кнопку **Border Style**, чтобы открыть окно с кодом обработчика события **Buttonl Click.** Добавьте в этот метод следующую строку кода:

```
this.FormBorderStyle = FormBorderStyle.Sizable;
```

8. Возвратитесь в окно конструктора, дважды щелкните кнопку Resize и добавьте следующую строку:

```
this.Size = new Size(300, 500);
```

9. Возвратитесь в окно конструктора, дважды щелкните кнопку Opacity и добавьте следующую строку:

```
this.Opacity = 1;
```

Запуск готового решения

10. Для построения решения выберите меню **Build (Построение)**, далее команду **Build Solution (Построить решение)**. При наличии ошибок исправьте их и снова постройте решение. В дальнейшем при необходимости выбора последовательности действий очередность команд будет описываться, например, так: **Build | Build Solution.**

11. Нажмите Ctrl + F5 или выберите **Debug (Отладка) | Start Without Debugging (Запуск без отладки)**, чтобы запустить приложение. Щелкайте каждую кнопку и наблюдайте, как изменяется вид формы.

12. Измените поочередно расположение левой и верхней границ формы и сравните поведение кнопок внутри формы. Обратите внимание, что расстояние до этих границ от кнопки **Border Style** остается постоянным. Почему?

Упражнение 2. Создание непрямоугольной формы Windows

В этом упражнении вы создадите треугольную форму Windows.

1. Откройте Visual Studio и создайте новый проект Windows Forms. Проект откроется с формой по умолчанию с именем **Form1** в конструкторе.

2. В окне Properties задайте свойству **FormBorderStyle** значение **None,** а свойству **BackColor** значение **Red.** В этом случае форму легче будет увидеть при тестировании приложения.

3. Перетащите кнопку из Toolbox в левый верхний угол формы. Задайте свойству **Text** кнопки значение **Close Form.**

4. Дважды щелкните кнопку Close Form и добавьте в обработчик события **Button1 Click** следующий код:

```
this.Close();
```

5. В конструкторе дважды щелкните форму, чтобы открыть обработчик события **Form1 Load.** Добавьте в этот метод следующий код (он задает области формы треугольную форму указанием многоугольника с тремя углами):

```
System.Drawing.Drawing2D.GraphicsPath myPath =
 new System.Drawing.Drawing2D.GraphicsPath();
myPath.AddPolygon(new Point[] { new Point(0, 0),
 new Point(0, this.Height),
 new Point(this.Width, 0) });
 Region myRegion = new Region(myPath);
 this.Region = myRegion;
```

6. Постройте и запустите приложение. Появится треугольная форма.

Упражнение 3. Создание наследуемой формы

Если у вас имеется уже готовая форма, которую вы собираетесь использовать в нескольких приложениях, удобно создать наследуемую (производную) форму. В этом упражнении вы создадите новую форму и унаследуете ее от существующей базовой формы, а затем измените производную форму, настроив ее для конкретной работы.

1. Откройте проект из предыдущего упражнения. Базовой формой для создания производной будет треугольная форма.
2. Для кнопки **Close Form** задайте свойство **Modifiers** как **protected**.
3. Добавьте производную форму: меню **Project (Проект) | Add Windows Form…(Добавить форму Windows)**, в окне **Categories (Категории)** укажите **Windows Form**, в окне **Templates (Шаблоны)** выберите **Inherited Form (Наследуемая форма).**
4. В окне **Add New Item** в поле **Name** укажите название формы: nForm.cs и нажмите **Add** для добавления формы.
5. В появившемся окне **Inheritance Picker**, в котором отображаются все формы текущего проекта, выберите базовую форму Form1 и нажмите OK.
6. Постройте проект.
7. Откройте форму nForm в режиме конструктора. Проверьте, что она имеет треугольную форму и свойства базовой формы и элемента управления наследованы.
8. Настройте свойства производной формы:
   a. для кнопки:
      i. свойство **Text** – Hello!!!
      ii. свойство **BackColor** – Brown
   b. для формы: свойство **BackColor** – Blue
9. Постройте проект.
10. Задайте производную форму в качестве стартовой, указав в функции **Main** следующий код:

```
Application.Run(new nForm());
```

11. Постройте и запустите приложение. Должна открыться производная форма со своими свойствами. Проверьте, наследуется ли закрытие формы кнопкой.

Упражнение 4. Создание MDI-приложения

В этом упражнении Вы создадите MDI-приложение с родительской формой, загружающей и организующей дочерние формы. Также Вы познакомитесь с элементом управления **MenuStrip**, который позволяет создать меню формы.

Создание нового проекта с базовой формой

1. Создайте новый проект Windows Forms, укажите имя MdiApplication.
2. Переименуйте файл Form1.cs на ParentForm.cs.
3. Для формы задайте следующие свойства:

Name	**ParentForm**
Size	420; 320
Text	Parent Form

4. Проверьте, что произошли изменения в функции Main так, чтобы форма ParentForm стала стартовой.

5. Откройте файл ParentForm.cs в режиме конструктора.
6. Для свойства формы **IsMdiContainer** задайте значение **True.** Таким способом эта форма будет определена как родительская форма MDI.

Создание меню для работы с формами

7. Создайте пункт меню **File:**
   a. Откройте ПИ **Toolbox**, добавьте на форму ЭУ **MenuStrip** и задайте для его свойства **Name** значение **MdiMenu**.
   b. Выделите меню в верхней части формы и задайте имя первого пункта меню **&File**.
   c. Для свойства **Name** пункта меню **File** задайте значение **FileMenuItem.**
   d. Раскройте меню **File**.
   e. Выделите элемент, появившейся под элементом **File,** и задайте его как **&New**.
   f. Для свойства **Name** пункта меню **New** задайте значение **NewMenuItem.**
   g. Выделите элемент, появившийся под элементом **New,** и задайте его как **&Exit.**
   h. Для свойства **Name** пункта меню **Exit** задайте значение **ExitMenuItem.**
   i. Дважды кликните левой кнопкой мыши по пункту меню **Exit** для создания обработчика события **Click**.
   j. В обработчик события **Click** для пункта меню **Exit** добавьте следующий код:

```
this.Close();
```

8. Создайте пункт меню **Window:**
   a. Переключитесь в режим конструктора.
   b. Выделите второй пункт меню справа от **File** и задайте его значением **&Window**.
   c. Для свойства **Name** пункта меню **Window** задайте значение **WindowMenuItem.**
   d. Раскройте меню **Window**.
   e. Выделите элемент, появившейся под элементом **Window,** и задайте для его свойства **Text** значение **&Cascade**.
   f. Для свойства **Name** пункта меню **Cascade** задайте значение **WindowCascadeMenuItem.**
   g. Выделите элемент, появившийся под элементом **Cascade,** и задайте для его свойства **Text** значение **&Tile.**
   h. Для свойства **Name** пункта меню **Tile** задайте значение **WindowTileMenuItem.**
   i. Дважды кликните левой кнопкой мыши по пункту меню **Cascade** для создания обработчика события **Click**:

```
this.LayoutMdi (System.Windows.Forms.MdiLayout.Cascade);
```

j. Вернитесь в режим конструктора и дважды кликните левой кнопкой мыши по пункту меню **Tile.**
k. В обработчик события **Click** для пункта меню **Tile** добавьте следующий код:

```
this.LayoutMdi(System.Windows.Forms.MdiLayout.TileHorizontal);
```

9. Реализуйте список открытых окон в меню **Window**:
   a. В конструкторе выберите компонент **Mdimenu**. Укажите в свойстве **MdiWindowListItem** имя пункта, созданного для этого – **WindowMenuItem.**

Создание дочерней формы

10. Создайте дочернюю форму:
    a. Выберите пункт меню **Project | Add Windows Form.**
    b. Задайте имя формы **ChildForm.cs**.
    c. Для свойства **Text** формы задайте значение **Child Form**.
    d. На ПИ **Toolbox** дважды кликните левой кнопкой мыши по ЭУ **RichTextBox** и задайте для его свойства **Name** значение **ChildTextBox**.
    e. Для свойства **Dock** ЭУ **RichTextBox** задайте значение **Fill**.
    f. Удалите существующий текст (если он есть) для свойства **Text** ЭУ **RichTextBox** и оставьте его пустым.
    g. На ПИ **Toolbox** дважды кликните левой кнопкой мыши по ЭУ **MenuStrip**.
    h. Для свойства **Name** ЭУ **MenuStrip** задайте значение **ChildWindowMenu**.
    i. Выделите меню в верхней части формы и наберите текст **F&ormat**.
    j. Для свойства **Name** пункта меню **Format** задайте значение **FormatMenuItem**, для свойства **MergeAction** установите значение **Insert**, а свойству **MergeIndex – 1**. В этом случае меню **Format** будет располагаться после **File** при объединении базового и дочерних меню.
    k. Выделите элемент, появившийся под элементом **Format,** и наберите текст **&Toggle Foreground**.
    l. Для свойства **Name** пункта меню **Toggle Foreground** задайте значение **ToggleMenuItem**.
    m. Дважды кликните левой кнопкой мыши по пункту меню **Toggle Foreground** и добавьте следующий код в обработчик события **Click**:

```
if (ToggleMenuItem.Checked)
{
 ToggleMenuItem.Checked = false;
 ChildTextBox.ForeColor = System.Drawing.Color.Black;
}
 else
{
      ToggleMenuItem.Checked = true;
```

```
        ChildTextBox.ForeColor = System.Drawing.Color.Blue;
      }
```

Отображение дочерней формы

11. Отобразите дочернюю форму в родительской форме:
    a. Откройте ParentForm.cs в режиме конструктора.
    b. Дважды кликните левой кнопкой мыши по кнопке **New** в меню **File** для создания обработчика события **Click.**
    c. Добавьте следующий код для обработчика события **Click** для пункта меню **New:**

```
ChildForm newChild = new ChildForm();
newChild.MdiParent = this;
newChild.Show();
```

Работа с приложением

12. Проверьте работу приложения:
    a. Постройте и запустите приложение.
    b. Когда появится родительская форма, выберите пункт меню **File | New**.
    c. В родительском окне появится новая дочерняя форма. Обратите внимание на то, дочернее меню сливается с родительским и пункты меню упорядочиваются в соответствие со свойством **MergeIndex**, установленным ранее.
    d. Наберите какой-нибудь текст в дочернем окне и воспользуйтесь пунктом меню **Format** для изменения цвета шрифта текста.
    e. Откройте еще несколько дочерних окон.
    f. Выберите пункт меню **Window | Tile**. Обратите внимание на то, что дочерние окна выстраиваются в упорядоченном порядке.
    g. Закройте все дочерние окна.
    h. Обратите внимание на то, что, когда закроется последнее дочернее окно, меню родительской формы изменится, и оттуда исчезнет пункт **Format**.
    i. Для закрытия приложения выберите пункт меню **File | Exit**.

13. Обратите внимание, что заголовок у дочерних окон одинаковый. При создании нескольких документов, например в Microsoft Word, они называются ДокументN, где N — номер документа. Реализуйте эту возможность:
    a. Откройте код родительской формы и в классе ParentForm объявите переменную openDocuments:

```
private int openDocuments = 0;
```

    b. К свойству **Text** дочерней формы добавьте счетчик числа открываемых документов (в коде обработчика события **Click** для пункта меню **New**):

```
newChild.Text = newChild.Text+" "+ ++openDocuments;
```

14. Запустите приложение. Теперь заголовки новых документов содержат порядковый номер.

Дополнительное упражнение

Для углубления знаний о добавлении и настройке форм Windows выполните следующие задания.

Задание 1. Создайте пользовательскую форму, которая во время выполнения будет иметь овальное очертание. Данная форма должна содержать функциональность, дающую возможность пользователю закрывать ее во время выполнения.

Рекомендация: при разработке формы в виде эллипса используйте следующий код:

```
// Добавление эллипса, вписанного в прямоугольную форму
// заданной ширины и высоты
myPath.AddEllipse(0, 0, this.Width, this.Height);
```

Задание 2. Создайте приложение с двумя формами и установите вторую форму как стартовую. Сделайте так, чтобы при запуске стартовая форма разворачивалась до максимальных размеров и содержала функциональность, дающую возможность пользователю открыть первую форму, отображающуюся в виде ромба зеленого цвета с кнопкой (в центре ромба) закрытия формы с надписью GREENPEACE.

Лабораторная работа 2. Работа с элементами управления

Цель работы

Изучение способов использования элементов управления и получение навыков по обработке событий.

Упражнение 1. Обработка событий Click и MouseMove

Элементы управления – это компоненты, объединяющие графический интерфейс с предварительно разработанной функциональностью. Элементы управления представляют собой многократно используемые блоки кода, предназначенные для выполнения определенных задач. Все элементы управления являются производными базового класса **Control**, а значит, тоже используют различные свойства, задающие размер, расположение и другие основные аспекты элементов управления.

Выполнив первое упражнение этого задания, вы создадите простое приложение, отслеживающее события мыши, которые происходят у конкретного элемента управления.

Размещение на форме элементов управления

1. Создайте новое Windows приложение. Назовите его WinQuestion.
2. Расположите на форме две кнопки **Button** и надпись **Label,** разместите их по-своему усмотрению.
3. Установите следующие свойства элементов управления и формы:

Объект	Свойство	Значение
Form1	FormBorderStyle	Fixed3D
	Size	350; 200
	Text	Насущный вопрос
label1	Text	Вы довольны своей зарплатой?
Button1	Name	btnyes
	Text	Да
Button2	Name	btnno
	Text	Нет

4. Щелкните дважды по кнопке "Да". В обработчике этой кнопки добавьте следующий код:

```
MessageBox.Show("Мы и не сомневались, что Вы так
думаете!");
```

5. Выделите кнопку "Нет". В окне **Properties** переключитесь в окно событий и дважды щелкните в поле MouseMove.

6. В обработчике этого события добавьте код для связывания движения мыши с координатами кнопки и указания координат, куда кнопка будет перемещаться:

```
btnno.Top -= e.Y;
btnno.Left += e.X;
if (btnno.Top < -10 || btnno.Top > 100)
btnno.Top = 60;
if (btnno.Left < -80 || btnno.Left > 250)
btnno.Left = 120;
```

7. Запустите приложение и нажмите на каждую из кнопок.

Упражнение 2. Работа со списками

Основными элементами управления списком являются **ListBox, ComboBox** и **CheckedListBox**. Несмотря на некоторые отличия во внешнем виде и разные функциональные возможности, они одинаково формируют и представляют списки данных и включают в себя коллекцию **Items,** которая систематизирует элементы, содержащие один из этих элементов управления.

ListBox — самый простой элемент управления списка. Он служит главным образом для отображения простого списка элементов в пользовательском интерфейсе, по которому легко перемещаться.

CheckedListBox – помечаемый список. Является разновидностью простого списка. Его дополнительное достоинство — в наличии чекбоксов рядом с каждым элементом списка. Пользователь имеет возможность отметить один или несколько элементов списка, выставив напротив его флажок.

ComboBox – выпадающий список. Постоянно на форме представлено только одно значение этого списка. При необходимости пользователь может раскрыть список и выбрать другое интересующее его значение или ввести собственное.

Создание приложения, использующее список

1. Создайте новый проект Windows Forms, укажите имя TestList.
2. Добавьте на форму следующие элементы управления:
    a. **GroupBox**,
    b. **CheckedListBox** (поместите в **GroupBox**)
    c. **ComboBox**
    d. три элемента **Button**.
3. Установите следующие свойства формы и элементов управления:

Объект	**Свойство**	**Значение**
Form1	FormBorderStyle	Fixed3D
	Text	Работа со списками
	Size	410;310
groupBox1	Text	Список участников
CheckedListBox	Name	memberList
ComboBox	Name	peopleList
	Text	
Button1	Name	buttonAdd
	Text	Добавить
Button2	Name	buttonDelete
	Text	Удалить
Button3	Name	buttonSort
	Text	Сортировать

4. Проинициализируйте элемент управления **ComboBox** списком предполагаемых участников. Для этого в окне свойств peopleList выберите свойство **Items**. Откройте окно **String Collection Editor,** нажав на кнопку с тремя точками в поле **Items**. Добавьте в окно Ф.И.О. нескольких участников. Нажмите OK.

5. Добавьте обработчики для кнопок **Добавить** и **Удалить,** два раза щелкнув левой кнопкой мыши по каждой из кнопок.

6. В тело обработчика события кнопки **Добавить** вставьте следующий код:

```
if (peopleList.Text.Length != 0)
 {
 memberList.Items.Add(peopleList.Text);
 }
 else MessageBox.Show("Выберите элемент из списка или
введите новый");
```

7. Для реализации удаления элементов из списка введите код в тело обработчика события кнопки **Удалить**:

```
while (memberList.CheckedIndices.Count > 0)
 memberList.Items.RemoveAt(memberList.CheckedIndices[0]);
```

8. Для реализации сортировки элементов введите код в тело обработчика события кнопки **Сортировать:**

```
memberList.Sorted = true;
```

9. Откомпилируйте и запустите приложение. Заполните список участников, выбирая их из элемента **ComboBox.** Запишите новые данные в этот элемент и добавьте их в список. Отсортируйте список участников.

Упражнение 3. Создание и использование элемента управления ToolStrip

ToolStrip – это элемент управления, разработанный с целью упрощения создания пользовательских панелей инструментов, которые выглядят и работают, как панели инструментов Microsoft Office и Microsoft Internet Explorer. Используя элемент управления **ToolStrip**, вы можете быстро разрабатывать легко настраиваемые панели инструментов профессионального вида.

Добавление на форму шаблона панели инструментов

1. Откройте проект MdiApplication.
2. Откройте форму **ParentForm** в режиме конструктора.
3. Добавьте на форму ЭУ **ToolStrip**.
4. На форме откройте выпадающий список ЭУ **ToolStrip** и выберите **button** – добавится элемент ***toolStripButton1***. В панели инструментов он представлен в виде кнопки с рисунком, обозначающим функцию, которую этот элемент содержит.
5. Снова откройте выпадающий список ЭУ **ToolStrip** и выберите **Separator** – добавится элемент, который отделяет одни элементы панели инструментов от других.
6. Справа от разделителя добавьте еще две кнопки – элементы ***toolStripButton2*** и ***toolStripButton3.***
7. В итоге вы должны получить три кнопки, отделенные одним разделителем.

Отображение рисунка на элементах панели управления

8. Выберите первую кнопку. Убедитесь, что в окне Properties свойству **DisplayStyle** задано значение **Image.**
9. В окне Properties выберите изображение элемента управления, щелкнув свойство **Image** и выбрав соответствующее изображение или путь к нему в диалоговом окне **Select Resource.** Если у Вас есть готовые файлы подходящих изображений, то выберите их, в противном случае укажите любой рисунок из папки *Мои рисунки*. Набор готовых изображений можно найти в графических файлах в папке \Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\ .
10. Повторите предыдущие два пункта для остальных кнопок.

Создание графических изображений кнопок панели инструментов

11. Если Вас не устраивает вид готовых рисунков, то можно для придания кнопкам графических изображений самостоятельно их создать. Для работы с изображениями в среде разработки существует **Image Editor**. Он позволяет создать изображения с использованием простейших инструментов.
12. Для создания файла с изображением выберите меню **File | New | File...** В появившемся окне **New File** выберите тип файла

“Файл точечного рисунка”, нажмите **Open**. Появится пустое изображение с дополнительной панелью управления. В основном меню появится новый пункт меню **Image.** Для отображения панели с палитрой компонент выберите в меню пункт **Image | Show Colors Window (Показать окно выбора цвета).**

13. Создайте по своему усмотрению изображение для кнопки **New**, например, в виде белого листа и сохраните изображение в файл с именем Icon_New.bmp в каталог с решением.

14. Повторите действия для создания иконок для других кнопок, например, в виде нарисованных распылителем букв ***C*** и ***T***. Сохраните изображения в каталог с решением в файлы с именами Icon_Windows_Cascade.bmp и Icon_Windows_Title.bmp соответственно.

15. Выполните действия для указания новых изображений этим кнопкам.

16. Сохраните и запустите проект. Проверьте вид и работоспособность кнопок.

Добавление обработчиков событий для кнопок

17. Добавьте обработчик события **Click** объекта ***toolStrip1***, щелкнув два раза указателем мыши по имени события **Click** на закладке событий в окне свойств. В программу добавится функция toolStrip1_ItemClicked как обработчик события, происходящего при нажатии кнопки на панели инструментов.

18. В окне Properties для ***toolStripButton1*** в свойстве **Tag** запишите **NewDoc.** Аналогично укажите для ***toolStripButton2*** и ***toolStripButton3*** для свойства **Tag** значения **Cascade** и **Title** соответственно.

19. Укажите для кнопок всплывающие подсказки в свойстве **ToolTipText**: Create new document, Windows cascade и Windows title.

20. В обработчике события **Click** объекта toolStrip1_ItemClicked добавьте код, который будет реализовывать различные действия в зависимости от нажимаемой кнопки:

```
        switch(e.ClickedItem.Tag.ToString())
              {
              case "NewDoc":
                    ChildForm newChild = new ChildForm();
                    newChild.MdiParent = this;
                    newChild.Show();
                    newChild.Text = newChild.Text+" "+
    ++openDocuments;
                    break;
              case "Cascade":
        this.LayoutMdi (System.Windows.Forms.MdiLayout.Cascade);
                    break;
              case "Title":
        this.LayoutMdi
    (System.Windows.Forms.MdiLayout.TileHorizontal);
              break;
        }
```

21. Откомпилируйте и запустите приложение. Проверьте работоспособность кнопок.

Упражнение 4. Использование элемента управления StatusStrip

Элемент управления **StatusStrip** применяется в программах для вывода информации в строку состояния — небольшую полоску, расположенную внизу приложения. В этом упражнении вы добавите к приложению MdiApplication строку состояния, на которой показывается вариант ориентации окон и выводится текущая дата.

1. Откройте проект MdiApplication.
2. Увеличьте размер формы **ParentForm** до значения (450;350).
3. Добавьте на форму **ParentForm** элемент управления **StatusStrip**.
4. Удалите содержимое поля свойства **Text**.
5. Щелкните на кнопку выпадающего списка панели и выберите **StatusLabel.** Добавится элемент ***toolStripStatusLabel1*** – первая панель для отображения.
6. Создайте еще одну панель аналогичным способом – ***toolStripStatusLabel2*** и установите им следующие свойства:

Объект	**Свойство**	**Значение**
Первая панель	Text	Status
	Name	spWin
Вторая панель	Text	Data
	Name	spData

7. Для отображения информации на первой панели вставьте в соответствующие обработчики команд меню и кнопок на панели инструментов следующую строку кода:
   a. Для каскадной ориентации:
   ```
   spWin.Text="Windows is cascade";
   ```
   b. Для горизонтальной ориентации:
   ```
   spWin.Text="Windows is horizontal";
   ```
8. Для отображения даты на второй панели в конструкторе формы **ParentForm** добавьте код:
```
public ParentForm()
{
InitializeComponent();
// Свойству Text панели spData устанавливается текущая дата
    spData.Text =
Convert.ToString(System.DateTime.Today.ToLongDateString());
    }
```
9. Откомпилируйте и запустите приложение. Проверьте работоспособность панели состояния.

Упражнение 5. Работа с контейнерными элементами управления

Контейнерные элементы управления — это специализированные элементы управления, выступающие в роли настраиваемого вместилища для других элементов управления. К контейнерным элементам управления относятся **Panel** и **GroupBox**. Они предоставляют форме логические и

физические подразделы, которые могут группировать другие элементы управления в единообразные подгруппы пользовательского интерфейса. Например, элемент управления **GroupBox** содержит в себе набор связанных элементов управления **RadioButton**. Контейнерные элементы управления помогут вам создать ощущение стиля или информационного потока в пользовательском интерфейсе и позволят согласованно управлять элементами управления, которые содержатся в них.

Выполнив это упражнение, вы научитесь создавать формы Windows с использованием различных контейнерных элементов управления.

Создание проекта с возможностью группировки элементов на вкладках

1. Откройте Visual Studio и создайте новый проект Windows Forms. Назовите его WinContainer.

2. Перетащите из Toolbox в форму элемент управления **TabControl.** В окне Properties задайте свойству **Dock** значение **Fill.**

3. В окне Properties выберите свойство **TabPages,** чтобы открыть TabPage Collection Editor. Добавьте вкладки так, чтобы их стало всего пять. Задайте свойствам **Text** этих пяти элементов управления **TabPage** значения **GroupBox**, **Panel, FlowLayoutPanel**, **TableLayoutPanel** и **SplitContainer**. Щелкните ОК.

Настройка контейнерного элемента GroupBox

4. В форме выберите вкладку GroupBox. Перетащите элемент управления **GroupBox** из Toolbox в элемент управления **TabPage.**

5. Перетащите в **GroupBox** два элемента управления **RadioButton.**

6. Добавьте на вкладку GroupBox вне элемента управления **GroupBox** кнопку (элемент управления **Button**). Для кнопки свойство **Text** сделайте пустым, а свойству **Name** укажите значение but.

7. Дважды кликните по кнопке и добавьте код обработчика события установки надписи на кнопке в зависимости от выбранного переключателя (**RadioButton**):

```
if (radioButton1.Checked == true)
      this.but.Text = "First";
           else if (radioButton2.Checked == true)
                this.but.Text = "Second";
```

Настройка элемента Panel

8. Выберите в форме вкладку Panel. Перетащите элемент управления **Panel** из Toolbox в элемент управления **TabPage.** Для элемента **Panel** задайте свойству **Dock** значение **Fill.**

9. Перетащите четыре элемента управления **Button** из Toolbox в элемент управления **Panel.**

10. Свойству **AutoScroll** установите значение **True**, в этом случае элемент управления **Panel** будет отображать полосы прокрутки, если элементы находятся за пределами видимых границ.

Настройка элемента FlowLayoutPanel

11. Выберите в форме вкладку FlowLayoutPanel. Перетащите элемент управления **FlowLayoutPanel** из Toolbox в элемент управления **TabPage**. Задайте значение **Fill** свойству **Dock** элемента управления **FlowLayoutPanel**.

12. Перетащите четыре элемента управления **Button** из Toolbox в элемент управления **Panel.** Обратите внимание на размещение добавляемых элементов: по умолчанию порядок следования элементов управления в *FlowLayoutPanel* — слева направо. Это значит, что элементы управления, расположенные в *FlowLayoutPanel,* будут находиться в левом верхнем углу и размещаться вправо до тех пор, пока не достигнут края панели. Такое поведение контролируется свойством *FlowDirection,* которому может быть задано четыре значения заливки в *FlowLayoutPanel: LeftToRight* — по умолчанию, *RightToLeft* — справа налево, *TopDown* — сверху вниз и *BottomUp* — снизу вверх.

13. Дважды щелкните кнопку **button5** и добавьте в обработчик события button5_Click следующий код:

```
flowLayoutPanel1.SetFlowBreak(button6, true);
```

Настройка элемента TableLayoutPanel

14. Выберите конструктор формы (если это необходимо). В форме выберите вкладку TableLayoutPanel. Перетащите элемент управления **TableLayoutPanel** из Toolbox в **TabPage**. Задайте свойству **CellBorderStyle** (определяет вид ячеек таблицы и их поведение) значение **Inset**, а свойству **AutoScroll — True**.

15. Перетащите элемент управления **Button** из Toolbox в левую верхнюю ячейку элемента управления **TableLayoutPanel**.

16. Дважды щелкните **Button9** и добавьте в обработчик события Button9_Click следующий код:

```
Button aButton = new Button();
tableLayoutPanel1.Controls.Add(aButton, 1, 1);
```

Настройка элемента SplitContainer

17. В конструкторе выберите вкладку SplitContainer. Перетащите элемент управления **SplitContainer** из Toolbox в **TabPage**. Задайте свойству **BorderStyle** значение **Fixed3D**.

18. Перетащите два элемента управления **Button** из Toolbox в Panel1 элемента управления **SplitContainer**. Задайте свойствам **Text** этих кнопок значения *Fix/Unfix Panel1* и *Fix/Unfix Splitter*. Измените размеры кнопок так, чтобы отображался текст.

19. Добавьте кнопку в Panel2 и задайте свойству **Text** значение *Collapse/Uncollapse Panel1*. Измените размеры кнопок так, чтобы отображался текст.

20. Дважды щелкните кнопку *Fix/Unfix Panel1* и добавьте в обработчик события *Click* следующий код:

```
if (splitContainer1.FixedPanel == FixedPanel.Panel1)
        splitContainer1.FixedPanel = FixedPanel.None;
```

```
            else
              splitContainer1.FixedPanel = FixedPanel.Panel1;
```

21. Дважды щелкните кнопку *Fix/Unfix Splitter* и добавьте в обработчик события *Click* следующий код

```
        splitContainer1.IsSplitterFixed =
 !(splitContainer1.IsSplitterFixed);
```

22. Дважды щелкните кнопку *Collapse/Uncollapse Panell* и добавьте в обработчик события *Click* следующий код:

```
        splitContainer1.Panel1Collapsed =
 !(splitContainer1.Panel1Collapsed);
```

23. Постройте и запустите приложение.

24. На вкладке GroupBox поочередно выбирайте переключатели следите за изменением надписи на кнопке.

25. На вкладке Panel измените размер формы с помощью мыши. Проверьте, появились ли полосы прокрутки.

26. На вкладке FlowLayoutPanel измените размер формы с помощью мыши. Просмотрите, что автоматически изменилась компоновка. Щелкните кнопку **button5** и проверьте, прервалась ли последовательность на элементе управления **button6** (это было реализовано вызовом метода *SetFlowBreak*).

27. На вкладке TableLayoutPanel щелкните кнопку **button9**, добавится новая кнопка.

28. На вкладке SplitContainer измените размеры формы, а также размеры каждой панели, передвинув *Splitter*. По очереди щелкайте каждую кнопку и смотрите, как это отражается на возможности элемента управления изменять свои размеры.

Упражнение 6. Элементы с поддержкой отображения текста

Выполнив это упражнение, вы научитесь использовать элемент управления **LinkLabel** в форме и настраивать его так, чтобы открывалось диалоговое окно, запрашивающее у пользователя имя.

Настройка элемента управления LinkLabel

1. Откройте Visual Studio и создайте новый проект Windows Forms. Назовите его WinLinkLabel.

2. Перетащите два элемента управления **LinkLabel** из Toolbox в форму.

3. В окне Properties для первого элемента задайте свойству **Text** значение **Open Form,** для второго – значение **Microsoft**.

Создание диалоговой формы

4. В меню Project выберите Add Windows Form и добавьте в свой проект новую форму Windows с именем **Form2.**

5. В конструкторе перетащите два элемента управления **Button** в форму **Form2.** Задайте свойству **Text** этих кнопок значения **Accept** и **Cancel.** Расположите их в правом нижнем углу формы.

6. Задайте свойству **DialogResult** кнопки Accept значение **ОК,** а свойству **DialogResult** кнопки Cancel — значение **Cancel.**

7. Перетащите два элемента управления **TextBox** из Toolbox в форму.

8. Задайте свойству **Modifiers** каждого элемента управления **TextBox** значение **Internal.** Свойство **Text** для них оставьте пустым.

9. Перетащите два элемента управления **Label** из Toolbox в форму и разместите их рядом с элементом управления **TextBox.**

10. Задайте свойствам **Text** элементов управления **Label** значения **&First Name** и **&Last Name.**

11. Проверьте, что свойству **UseMnemonic** всех надписей установлено значение **True.**

12. Установите свойство **TabIndex** в окне Properties, как показано ниже.

Элемент управления	Индекс закладки
label1	0
textBox1	1
label2	2
textBox2	3
button1	4
button2	5

Реализация обработчика события вызова диалогового окна

13. Выберите в конструкторе закладку для формы **Form1.** Дважды щелкните первый элемент управления **linkLabel1** для создания обработчика события **linkLabel1_LinkClicked.** Добавьте следующий код:

```
DialogResult aResult;
 Form2 aForm = new Form2();
 aResult = aForm.ShowDialog();
      if (aResult == System.Windows.Forms.DialogResult.OK)
      {
 MessageBox.Show("Your name is " + aForm.textBox1.Text + "
" + aForm.textBox2.Text);
      }
 linkLabel1.LinkVisited = true;
```

Реализация обработчика события вызова веб-страницы

14. Выберите в конструкторе закладку для формы **Form1.** Дважды щелкните второй элемент управления **linkLabel2** для создания обработчика события **linkLabel2_LinkClicked.** Добавьте следующий код:

```
System.Diagnostics.Process.Start("www.limtu.com");
 linkLabel2.LinkVisited = true;
```

15. Постройте и запустите приложение.

16. Щелкните элемент управления **linkLabel – Open Form,** чтобы открыть форму. Введите соответствующую информацию в поля ввода и проверьте кнопки Accept и Cancel.

17. Щелкните элемент управления **linkLabe2 – Microsoft,** чтобы открыть сайт известного учебного центра.

Упражнение 7. Элементы с поддержкой редактирования текста

TextBox — это основной элемент управления, с помощью которого можно принимать вводимый пользователем текст а также отображать текст для пользователя. Существует возможность создавать как поля, отображающие многострочный текст, так и поля отображающие знак пароля вместо реально введенного текста.

Элемент управления **MaskedTextBox** — это видоизмененный элемент управления **TextBox**, позволяющий задавать предварительно установленный шаблон для принятия пользовательского ввода или отказа от него. С помощью свойства **Mask** можно указать обязательные или необязательные символы либо тип вводимых символов (буквы или цифры) и применить форматирование для отображения строк.

1. Откройте выполненное вами в предыдущем упражнении решение WinLinkLabel.
2. Отобразите конструктор для формы *Form2*.
3. Добавьте элемент управления **TextBox** на форму под расположенными ранее элементами. Перетащите элемент управления **Label** в форму и разместите слева от этого элемента.
4. Задайте свойству **Text** элемента управления **Label** значение **Address**.
5. Для элемента управления **TextBox** задайте следующие свойства:

Свойства	**Значение**	**Комментарий**
Multiline	True	многострочный
WordWrap	False	переход слова с одной строки на другую
ScrollBars	Both	отображение полос прокрутки

6. Измените размеры элемента управления **TextBox** так, чтобы он вмещал адрес. При необходимости увеличьте размеры формы и переместите кнопки **Accept** и **Cancel**.
7. Перетащите элементы управления **MaskedTextBox** и **Label** из Toolbox на форму и разместите их под ранее введенные элементы.
8. Свойству **Text** элемента управления **Label** задайте значение **Phone Number**.
9. Задайте значение *(999)-000-0000* свойству **Mask** элемента управления **MaskedTextBox**.
10. Задайте значение **Internal** свойству **Modifiers** для последних элементов управления **TextBox** и **MaskedTextBox**.
11. Откройте окно кода формы *Form1*.
12. В обработчике события linkLabel1_LinkClicked добавьте в блок if, расположенный под кодом, который вы добавили в предыдущем упражнении, следующий код

```
MessageBox.Show("Your address is " + aForm.textBox3.Text);
MessageBox.Show("Your phone number is " +
aForm.maskedTextBox1.Text);
```

13. Постройте и запустите приложение. Введите в текстовое поле свой телефон. Проверьте, что номер отображается согласно требуемому формату.

Упражнение 8. Добавление и удаление элементов управления в режиме работы приложения

При размещении на форме элемента управления в режиме дизайна, среда создает код, описывающий этот элемент. Если назначить в обработчике заданного элемента управления генерацию аналогичного кода, то в запущенном приложении можно будет добавлять на форму или удалять элементы, активизируя этот обработчик.

Для работы с элементами управления используется объект **ControlsCollection**, содержащий ряд методов, основные из которых будут использованы в данном упражнении.

1. Создайте новое приложение и назовите его RegistrationForm.
2. Добавьте на форму три надписи, два текстовых поля, кнопку, элементы CheckBox и GroupBox
3. Установите следующие значения свойств формы и элементов управления:

Объект	**Свойство**	**Значение**
Form1	FormBorderStyle	Fixed3D
	Text	Регистрация
	Size	400;310
label1	Location	30;10
	Text	Выберите тип регистрации
label2	Location	16; 32
	Text	Name
label3	Location	16; 64
	Text	PIN
button1	Location	80; 248
	Text	Регистрация
textBox1	Location	96; 32
	Text	
	Size	184; 20
textBox2	Location	96; 64
	Size	184; 20
	Text	
checkBox1	Location	40; 40
	Size	232; 24
	Text	Расширенные возможности
groupBox1	Text	Введите регистрационные данные
	Location	16; 80
	Size	344; 144

4. Для реализации возможности добавления и удаления элементов в процессе выполнения программы реализуйте обработчик события CheckedChanged: щелкните дважды на элементе checkBox1 и добавьте следующий код:

```
if (checkBox1.Checked == true)
 {
      Label lbl = new Label();
      lbl.Location = new System.Drawing.Point(16, 96);
      lbl.Size = new System.Drawing.Size(32, 23);
      lbl.Name = "labelll";
      lbl.TabIndex = 2;
      lbl.Text = "PIN2";
      groupBox1.Controls.Add(lbl);
      TextBox txt = new TextBox();
      txt.Location = new System.Drawing.Point(96, 96);
      txt.Size = new System.Drawing.Size(184, 20);
      txt.Name = "textboxx";
      txt.TabIndex = 1;
      txt.Text = "";
      groupBox1.Controls.Add(txt);
 }
 else{
      }
```

5. Откомпилируйте и запустите приложение. Проверьте, что при установке флажка в ЭУ checkBox “Расширенные возможности” на форме появляется надпись и поле ввода для дополнительных данных.

6. Для удаления ЭУ с формы могут применяться методы: **Clear** (удаление всех элементов из коллекции), **Remove** (удаление элемента из коллекции) и **RemoveAt** (удаление элемента по заданному индексу). В тело оператора else добавьте код для удаления ЭУ по индексу:

```
int lcv;
lcv = groupBox1.Controls.Count;// определяется количество
 while (lcv > 4)
 {
      groupBox1.Controls.RemoveAt(lcv - 1);
      lcv -= 1;
 }
```

7. Запустите приложение. Убедитесь, что при включении “Расширенные возможности” дополнительные элементы появляются на форме, а при выключении – исчезают.

Упражнение 9. Проверка вводимых значений. События KeyPress и Validating. Элемент управления ErrorProvider

При внесении значений параметров пользователем во многих случаях требуется проверять вводимый текст по заданным критериям. Например, регистрационный номер, телефон не должны содержать букв, поле имени - цифр. В этом упражнении рассматриваются реализации проверок, которые можно осуществлять, используя встроенные события текстового поля.

Использование события KeyPress

1. Откройте приложение RegistrationForm.

2. Выделите поочередно текстовые поля **TextBox1** и **TextBox2,** в окне **Properties** создайте обработчики события **KeyPress**, возникающего при нажатии любой клавиши в поле.

3. В тело обработчика события **KeyPress** для текстового поля **TextBox1** укажите следующий код (для элемента **TextBox1** недопустимыми значениями будут цифры):

```
if (char.IsDigit(e.KeyChar) )
 {
      e.Handled = true;
      MessageBox.Show("Поле Name не может содержать цифры");
 }
```

4. Для элемента **TextBox2**, наоборот, недопустимыми значениями будут буквы, в обработчике события **KeyPress** для текстового поля **TextBox2** укажите код:

```
if (!char.IsDigit(e.KeyChar) )
{
 e.Handled = true;
 MessageBox.Show("Поле PIN не может содержать буквы");
 }
```

5. Откомпилируйте и запустите приложение. Попробуйте ввести в поле **Name** цифры, в поле **PIN** – буквы.

6. Для защиты текстового поля, появляющегося при установке галочки в чекбоксе "Расширенные возможности", необходимо вручную определить событие **KeyPress.** В обработчике события CheckedChanged для элемента **CheckBox1** укажите код:

```
txt.KeyPress+= new
System.Windows.Forms.KeyPressEventHandler(this.textBox2_KeyPress);
```

7. Запустите и протестируйте приложение.

Применение события Validating

Событие **KeyPress** блокирует часть клавиатуры. Другим способом проверки является событие **Validating**, позволяющее работать с клавиатурой, но блокирующее другие действия пользователя.

8. Закомментируйте обработчик элемента **TextBox2.**

9. В режиме дизайна формы в окне Properties элемента **TextBox2** создайте обработчик события **Validating** и запишите следующий код:

```
if(textBox2.Text =="")
 {
      e.Cancel=false;
 }
 else
 {
      try
      {
           double.Parse(textBox2.Text);
           e.Cancel = false;
      }
      catch
      {
 e.Cancel = true;
 MessageBox.Show("Поле PIN не может содержать буквы");
```

```
            }
        }
```

10. Запустите приложение. При переключении фокуса ввода или нажатии на кнопку "регистрация" происходит событие **Validating**.

Применение элемента управления ErrorProvider

Элемент управления **ErrorProvider** удобно применять, когда нужно выводить небольшую иконку в случае ошибки ввода.

11. В режиме дизайна из окна **ToolBox** перенесите на форму элемент управления **ErrorProvider**.

12. В коде формы в обработчике textBox1_KeyPress добавьте следующую строку:

```
errorProvider1.SetError(textBox1, "Must be letter");
```

13. Запустите приложение. При ошибке ввода появляется мигающая иконка уведомления, при наведении на нее всплывает поясняющее сообщение об ошибке.

Лабораторная работа 3. Создание элементов управления

Цель работы

Изучение способов разработки элементов управления и получение навыков по их настойке и применению в дальнейшей работе.

Упражнение 1. Создание составного элемента управления

В дополнение к уже существующим элементам управления можно разрабатывать собственные, чтобы обеспечить для своих приложений специализированную функциональность.

Существует три вида разрабатываемых пользователем элементов управления:

- **составные (composite)**, которые создаются при объединении других элементов управления Windows Forms;
- **специализированные (custom)**, создаваемые с нуля и предоставляющие собственный код для прорисовки;
- **расширенные (extended)**, которые добавляют функциональность к уже существующему элементу управления Windows Forms.

Составные элементы управления наследуются от класса **UserControl,** Он предоставляет базовый уровень функциональности, обеспечивающий добавление других элементов управления, а также свойств, методов и событий. Класс **UserControl** имеет собственный конструктор, позволяющий использовать в Visual Studio IDE перетаскивание дополнительных элементов управления из **Toolbox** на поверхность конструктора и настраивать их.

В этом упражнении вы создадите составной элемент управления, действующий как цифровые часы. В него вы добавите элемент управления **Label,** отображающий правильное время, и компонент **Timer,** каждую

секунду обновляющий **Label.** Предоставив свойство **Enabled** элемента управления **Timer,** вы дадите пользователям возможность включать и отключать часы.

Разработка составного элемента управления

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinTimer1**.

2. В меню **Project** выберите **Add User Control** и щелкните **Add** в диалоговом окне **Add New Item**. Укажите имя **UserControlTimer** и нажмите **Добавить**. К вашему проекту будет добавлен пустой пользовательский элемент управления, который откроется в конструкторе.

3. Из **Toolbox** перетащите **Label** в пользовательский элемент управления.

4. Удалите данные из свойства **Text** ЭУ **Label.**

5. Измените размеры пользовательского элемента управления так, чтобы он был приблизительно равен размеру элемента управления **Label.**

6. Из **Toolbox** перетащите в пользовательский элемент управления компонент **Timer.**

7. В окне **Properties** компонента **Timer** присвойте свойству **Interval** значение **1000** и свойству **Enabled** значение **True.**

8. Дважды щелкните компонент **Timer,** чтобы открыть в окне кода обработчик события **Timer.Tick** по умолчанию и добавьте следующую строку программы:

```
label1.Text = DateTime.Now.ToLongTimeString();
```

9. В окне кода добавьте следующее объявление свойства:

```
public bool TimeEnabled
 {
      get { return timer1.Enabled; }
      set { timer1.Enabled = value; }
}
```

10. В меню **File** выберите **Save All**, чтобы сохранить ваше решение.

11. В меню **Build** выберите **Build Solution**.

Применение составного элемента управления

12. Выберите вкладку конструктора **Forml.** Из **Toolbox** перетащите **UserControlTimer** в форму. К форме добавится экземпляр вашего пользовательского элемента управления и начнет отсчитывать каждую секунду. Обратите внимание, что вы можете приостановить это, присвоив свойству **TimeEnabled** значение **False** в окне **Properties**.

13. Постройте и запустите приложение. Обратите внимание, что пользовательский элемент управления во время выполнения действует так же, как в конструкторе.

Разработка библиотеки классов элементов управления

14. Чтобы разработать библиотеку классов элементов управления, выполните следующие действия:

   a. Создайте в Visual Studio новое приложение – **Библиотека элементов управления Windows**.

b. По умолчанию проект содержит составной элемент управления.
c. Добавьте элементы управления и код, как описано в упражнении 1.
d. Выберите элемент управления, который не должен изменяться при наследовании классов (в данном случае это и **label1**, и **timer1**), и задайте для свойства **Modifiers** (**Модификаторы)** этого элемента управления значение Private.
e. Создайте библиотеку DLL, построив проект.

Упражнение 2. Создание специализированного элемента управления

Для специализированных элементов управления не существует пользовательского интерфейса по умолчанию, они должны предоставлять весь код, необходимый для отображения своего графического представления. Элемент управления разрабатывается для того, чтобы получить точно такое визуальное представление, которого вам необходимо, и закодировать любую требуемую функциональность для взаимодействия с пользователем

Специализированные элементы управления наследуются от класса **Control.** Этот класс обеспечивает функциональность, требуемую для элемента управления. Он предоставляет базовую функциональность, необходимую для взаимодействия элемента управления с остальной частью приложения.

Ключевой задачей при разработке специализированного элемента управления является реализация видимого пользовательского интерфейса. Его можно создать реализацией метода **OnPaint,** вызываемого каждый раз, когда элемент управления отображается на экране.

Выполнив это упражнение, вы научитесь создавать специализированный элемент управления, также представляющий собой цифровые часы. Подобно элементу управления из предыдущего упражнения, они включают компонент **Timer** для обновления пользовательского интерфейса на регулярной основе. Однако теперь вы сами создадите отображение для этого элемента управления вместо того, чтобы использовать элемент управления **Label.**

Разработка специализированного элемента управления

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinTimer2.**

2. В меню **Project** выберите **Add New Item**. Выберите **Custom Control** в диалоговом окне **Add New Item** и щелкните **Add**. Укажите имя **UserControlTimer2** и нажмите **Add (Добавить).** К проекту будет добавлен новый специализированный элемент управления. Откройте и просмотрите его код.

3. Перейдите на вкладку конструктор. В конструкторе **UserControlTimer2** перетащите компонент **Timer** из **Toolbox** на поверхность конструктора.

4. В окне **Properties** присвойте свойству **Interval** компонента **Timer** значение **1000** и свойству **Enabled** значение **True.**

5. Дважды щелкните компонент **Timer** чтобы открыть окно кода в обработчике события **Timer.Tick** по умолчанию и добавьте следующую строку программы:

```
this.Refresh();
```

6. Переопределите метод **OnPaint** и укажите код для отображения прямоугольника, залитого синим цветом, заполняющего весь элемент управления:

```
protected override void OnPaint(PaintEventArgs pe)
 {
      base.OnPaint(pe);
      Graphics g = pe.Graphics;
      g.FillRectangle(Brushes.Blue, 0, 0, this.Width,
 this.Height);
}
```

7. Для отображения времени добавьте следующий код к методу **OnPaint** ниже определения прямоугольника:

```
pe.Graphics.DrawString(DateTime.Now.ToLongTimeString(),
this.Font, new SolidBrush(this.ForeColor), 0, 0);
```

8. В меню **File** выберите **Save Al**l, чтобы сохранить ваше решение.

9. В меню **Build** выберите **Build Solution**.

Применение специализированного элемента управления

10. Выберите вкладку конструктора **Forml.** Из **Toolbox** перетащите **UserControlTimer2** в форму. К ней будет добавлен экземпляр вашего специализированного элемента управления, который начнет отсчитывать каждую секунду.

11. Постройте и запустите приложение. Обратите внимание, что пользовательский элемент управления действует во время выполнения таким же способом, как в конструкторе.

Упражнение 3. Создание расширенных элементов управления

Расширенные элементы управления – это разработанные пользователем элементы управления, расширяющие уже существующий элемент управления .NET Framework. Сохраняется вся функциональность существующих элементов управления, но при этом добавляются свойства и методы, а при необходимости изменяется и внешнее представление элемента управления.

Разработка расширенного элемента управления

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinButNum.**

2. В меню **Project** выберите **Add Class**. Назовите этот класс **ClickButton** и щелкните **Add**.

3. Измените объявление класса, чтобы **ClickButton** наследовал класс **Button**:

```
public class ClickButton : System.Windows.Forms.Button
```

4. Добавьте следующее поле и свойство в окно кода с целью создания свойства **Clicks**:

```
int mClicks;
public int Clicks
{
 get { return mClicks; }
}
```

5. Переопределите метод **OnClick**, чтобы инкрементировать закрытую переменную **mClicks** каждый раз, когда щелкается кнопка:

```
protected override void OnClick(EventArgs e)
{
 mClicks++;
 base.OnClick(e);
}
```

6. Переопределите метод **OnPaint**, чтобы отобразить количество щелчков в правом нижнем углу элемента управления:

```
protected override void
OnPaint(System.Windows.Forms.PaintEventArgs pevent)
{
 base.OnPaint(pevent);
 System.Drawing.Graphics g = pevent.Graphics;
 System.Drawing.SizeF stringsize;
 stringsize = g.MeasureString(Clicks.ToString(),
this.Font, this.Width);
 g.DrawString(Clicks.ToString(), this.Font,
 System.Drawing.SystemBrushes.ControlText,
 this.Width - stringsize.Width - 3, this.Height -
 stringsize.Height - 3);
}
```

7. Сохраните и постройте решение.

Применение расширенного элемента управления

8. Выберите вкладку конструктора Forml.

9. Из **Toolbox** перетащите экземпляр **ClickButton** в форму и измените его размеры в сторону увеличения.

10. Постройте и запустите приложение.

11. В форме щелкайте **ClickButton1**. Обратите внимание, что количество щелчков отображается в правом нижнем углу.

Лабораторная работа 4. Использование окон диалога в формах

Цель работы

Изучение способов использования компонентов, представляющие диалоговые окна и получение навыков по работе с окнами диалога.

Упражнение 1. Использование компонента SaveFileDialog

Чтобы пользователи могли сохранять файлы, можно использовать встроенный компонент **SaveFileDialog**.

В этом упражнении Вы отобразите диалоговое окно, используя метод **ShowDialog**. Затем с помощью поля **DialogResult.OK** проверите, нажал ли пользователь кнопку **ОК**.

Для реализации отображения диалогового окна обозревателя папок выполните:

1. Создайте приложение Windows Forms, укажите имя **TestStandartDialog**.

2. Добавьте элемент **MenuItem**, задайте имя первого пункта меню **Файл** и команду **Сохранить как…**

3. Добавьте в форму элемент управления **richTextBox**, оставив имя по умолчанию **richTextBox1**. Свойству **Dock** установите **Fill**.

4. Добавьте в форму компонент **SaveFileDialog**. Проверьте, что в области компонентов появился компонент **saveFileDialog1**.

5. Дважды щелкните кнопку, чтобы добавить в редактор кода обработчик событий по умолчанию.

6. В обработчике событий добавьте следующий код для отображения диалогового окна **Сохранение файла**. Этот код сохраняет текст, введенный в элемент управления **richTextBox**, в текстовый файл в указанной папке.

```
saveFileDialog1.Filter = "txt files (*.txt)|*.txt";
     if(saveFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK
    && saveFileDialog1.FileName.Length > 0)
{
    richTextBox1.SaveFile(saveFileDialog1.FileName,
        RichTextBoxStreamType.PlainText);
}
```

7. Постройте и протестируйте приложение.

8. В открывшейся форме введите какой-либо текст в текстовое поле.

9. Выберите команду **Сохранить как…** и сохраните файл (имя и место для сохранения файла выберите по своему усмотрению).

10. Убедитесь, что текстовый файл находится в указанном месте.

Упражнение 2. Использование компонента ColorDialog

Для отображения диалогового окна цветовой палитры можно использовать встроенный компонент **ColorDialog** вместо того, чтобы создавать свое собственное диалоговое окно.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность выбрать цвет и применить его в форме Windows после указания соответствующей команды меню.

Для реализации отображения диалогового окна цветовой палитры выполните:

1. Для элемента MenuItem задайте имя второго пункта меню – **Формат** и команду – **Цвет фона**.

2. Добавьте в форму компонент **ColorDialog**.

3. Проверьте, что в области компонентов появился компонент **colorDialog1**.

4. Дважды щелкните кнопку **Цвет фона**, чтобы создать обработчик событий по умолчанию в редакторе кода.

5. В обработчике событий добавьте следующий код для отображения диалогового окна выбора цвета и изменения фонового цвета в соответствии с выбором пользователя:

```
if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
      richTextBox1.BackColor = colorDialog1.Color;
    }
```

6. Постройте и протестируйте приложение.

Упражнение 3. Использование компонента FontDialog

Для отображения диалогового окна выбора шрифтов можно использовать встроенный компонент **FontDialog** вместо того, чтобы создавать свое собственное диалоговое окно.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность выбрать шрифт в диалоговом окне и затем применить его к тексту.

Для реализации отображения диалогового окна выбора шрифтов выполните:

1. Задайте в меню **Формат** новую команду – **Шрифт**.

2. Перетащите в форму компонент **FontDialog**.

3. Проверьте, что в области компонентов появится компонент **fontDialog1**.

4. Дважды щелкните команду **Шрифт**, чтобы создать в редакторе кода обработчик событий по умолчанию.

5. В обработчик событий добавьте следующий код для отображения диалогового окна выбора шрифта текста в окне и изменения шрифта текста в соответствии с выбором пользователя:

```
if (fontDialog1.ShowDialog() == DialogResult.OK)
    {
      richTextBox1.Font = fontDialog1.Font;
    }
```

6. Постройте и протестируйте приложение.

Упражнение 4. Использование компонента OpenFileDialog

Чтобы пользователи могли выбрать текстовый файл и загрузить его в элемент управления **RichTextBox** в форме Windows Forms, можно использовать компонент **OpenFileDialog**.

В этом упражнении Вы дополните приложение **TestStandartDialog**, чтобы дать пользователям возможность открыть текстовый файл.

1. Задайте в меню **Файл** новую команду **Открыть…**

2. Перетащите в форму компонент **OpenFileDialog**.

В области компонентов появился компонент **openFileDialog1**.

3. Дважды щелкните команду **Открыть…**, чтобы создать в редакторе кода обработчик событий по умолчанию.

4. В обработчик событий добавьте следующий код для отображения диалогового окна открытия файла:

```
      Stream myStream = null;
       OpenFileDialog openFileDialog1 = new OpenFileDialog();
          openFileDialog1.InitialDirectory = @"c:\";
          openFileDialog1.Filter = "txt files (*.txt)|*.txt|All
files (*.*)|*.*";
          openFileDialog1.FilterIndex = 2;
           if (openFileDialog1.ShowDialog() == DialogResult.OK)
                 {
                     try
                     {
           if ((myStream = openFileDialog1.OpenFile()) != null)
                         {
                             using (myStream)
                             {

richTextBox1.LoadFile(openFileDialog1.FileName,
                     RichTextBoxStreamType.PlainText);
                             }
                         }
                     }
                     catch (Exception ex)
                     {
      MessageBox.Show("Error: Could not read file from disk: "
+ ex.Message);
                     }
                 }
```

5. Постройте и протестируйте приложение.

Лабораторная работа 5. Взаимодействие управляемого и неуправляемого кода

Цель работы

Изучение способов использования возможностей объединения старого кода с кодом, управляемым средой CLR и получение навыков по работе со службами взаимодействия управляемого кода с неуправляемым.

Упражнение 1. Использование COM-компонента для создания PDF-приложения

При создании приложений, использующих платформу Microsoft .NET, возникает задача применения в собственных проектах уже готовых библиотек кода, написанных на других языках. Сжатые сроки разработки и уже имеющиеся программные блоки не позволяют отказаться от готовых решений, поэтому их приходится использовать, встраивая в структуру собственных проектов.

Код, выполняющийся под управлением среды выполнения (в случае платформы .NET — среды Common Language Runtime), называется **управляемым**.

Код, запускаемый не под управлением среды, называется **неуправляемым**. Примером неуправляемого кода могут служить COM-компоненты, Microsoft ActiveX интерфейсы и функции API Win32.

Microsoft .NET Framework позволяет взаимодействовать с COM-компонентами, COM+-службами, внешними типами библиотек и разными службами операционной системы.

Платформа .NET Framework предлагает две службы взаимодействия управляемого кода с неуправляемым — Platform Invoke и COM interoperability, которые используют реальные возможности проекта.

1. Создайте новое Windows-приложение и назовите его **PDF Reader**.
2. Добавьте на форму элементы **OpenFileDialog** и **MenuStrip**.
3. Установите следующие свойства формы **Form1**:

Свойство	**Значение**
Text	Обозреватель документов в формате PDF
WindowState	Maximized

4. Свойство **FileName** элемента **OpenFileDialog** сделайте пустым.
5. Добавьте пункт меню верхнего уровня **File** с командами **Open** и **Exit.**
6. Выберите **Tools | Choose Toolbox Items…**
7. На вкладке **COM Components** выберите **Adobe Acrobat (PDF) Reader** (этот компонент появляется после установки программы Adobe Acrobat Reader) и нажмите **OK**. Убедитесь, что на панели Toolbox этот ЭУ действительно появился.
8. Перенесите **Adobe Acrobat Reader** на форму и установите свойству **Dock** значение **Fill**, а свойству **(Name) – axAcroPDF1** (от имени объекта будет вызываться нужные нам методы, в принципе имя может быть любым).
9. Добавьте обработчик пункта меню **Open**:

```
    private void openToolStripMenuItem_Click (object sender,
System.EventArgs e)
    {
     openFileDialog1.Filter = "Файлы pdf|*.pdf";
     openFileDialog1.ShowDialog();
     axAcroPDF1.LoadFile(openFileDialog1.FileName);
    }
```

10. Реализуйте обработчик события **Click** для пункта меню **Exit**.
11. Постройте и запустите приложение. При открытии документа в формате pdf происходит, по сути, встраивание в форму интерфейса программы Adobe Acrobat Reader.

Упражнение 2. Вызов функции API

Службы Platform Invoke позволяют управляемому коду запускать функции неуправляемого кода, которые находятся в файлах библиотек динамической компоновки (DLL).

Эти службы предоставляют механизмы обнаружения и запуска неуправляемых функций и преобразование типов данных входящих и исходящих аргументов функции.

Когда управляемый код запускает функцию неуправляемого кода, локализованную в DLL-файле, службы Platform Invoke находят этот DLL файл, загружают его в оперативную память и находят адрес функции в памяти. После этого службы передают входящие аргументы функции в стек, преобразовывают данные, которые необходимо перевести, эмулируют сборку мусора и передают управление по адресу неуправляемой функции в памяти.

Первым шагом в запуске неуправляемой функции является объявление функции. Функция должна быть статической (static) и внешней (extern). Далее следует импорт библиотеки, содержащей эту функцию.

Импортировать библиотеку нужно, используя атрибут **DllImport**, который находится в пространстве имен System.Runtime.InteropServices.

1. Создайте новое приложение и назовите его WinAnim.
2. Расположите на форме три кнопки и установите следующие свойства формы и кнопок:

Свойство	**Значение**
Form1	
Text	Анимация формы
Button1	
Name	btnAW_BLEND
Location	30; 62
Size	232; 23
Text	Проявление
Button2	
Name	btnHOR_AW_SLIDE
Location	30; 118
Size	232; 23
Text	Горизонтальное появление
Button3	
Name	btnCenter_AW_SLIDE
Location	30; 182
Size	232; 23
Text	Появление из центра

3. Добавьте класс WinAPIClass:

```
using System;
using System.Runtime.InteropServices;
using System.Windows.Forms;
namespace AnimatedWindow
{
      public class WinAPIClass
{
#region Анимация окна
```

```
    /// <summary>
    /// Тип анимации окна. Перечисление возвращает тип int
    /// после приведения. Каждому элементу присвоено
    /// свое значение типа int.
    /// </summary>
    [Flags]
    public enum AnimateWindowFlags:int
    {
    AW_HOR_POSITIVE = 1,
    AW_HOR_NEGATIVE = 2,
    AW_VER_POSITIVE = 4,
    AW_VER_NEGATIVE = 8,
    AW_CENTER = 16,
    AW_HIDE = 65536,
    AW_ACTIVATE = 131072,
    AW_SLIDE = 262144,
    AW_BLEND = 524288
    };
    /// <summary>
    /// Анимация окна.
    /// </summary>
    /// <param name="hwnd">Окно.</param>
    /// <param name="dwTime">Время.</param>
    /// <param name="dwFlags">Тип анимации. Если в
    /// неуправляемом коде используется перечисление, то его
    /// нужно конвертировать в тип данных int.
     </param>
    /// <returns></returns>
[DllImportAttribute("user32.dll",EntryPoint="AnimateWindow",Set
LastError=true)]
    public static extern bool AnimateWindow(IntPtr hwnd,int
dwTime,int dwFlags);
    /// <summary>
    /// Анимация окна.
    /// </summary>
    /// <param name="ctrl">Окно.</param>
    /// <param name="dwTime">Время.</param>
    /// <param name="Flags">Флаги.</param>
    /// <returns></returns>
    public static bool AnimateWindow(Control ctrl,int dwTime,
AnimateWindowFlags Flags)
    {
          return AnimateWindow(ctrl.Handle,dwTime,(int)Flags);
     }
     #endregion
          }
    }
```

4. Создайте обработчики кнопок:

```
    private void btnAW_BLEND_Click(object sender,
System.EventArgs e)
    {
    // Скрываем окно
    this.Hide();
    // Запускаем анимацию.
    // Второй параметр в скобках — время анимации в
    // миллисекундах.
```

```
        WinAPIClass.AnimateWindow(this, 3000,
    WinAPIClass.AnimateWindowFlags.AW_ACTIVATE|
        WinAPIClass.AnimateWindowFlags.AW_BLEND);
        // Отображаем кнопки после анимации
        this.btnAW_BLEND.Invalidate();
        this.btnHOR_AW_SLIDE.Invalidate();
        this.btnCenter_AW_SLIDE.Invalidate();
        }
        private void btnHOR_AW_SLIDE_Click(object sender,
    System.EventArgs e)
        {
         this.Hide();
         WinAPIClass.AnimateWindow(this, 3000,
    WinAPIClass.AnimateWindowFlags.AW_HOR_POSITIVE|
         WinAPIClass.AnimateWindowFlags.AW_SLIDE);
         this.btnAW_BLEND.Invalidate();
         this.btnHOR_AW_SLIDE.Invalidate();
         this.btnCenter_AW_SLIDE.Invalidate();
         }
        private void btnCenter_AW_SLIDE_Click(object sender,
    System.EventArgs e)
        {
         this.Hide();
         WinAPIClass.AnimateWindow(this, 3000,
    WinAPIClass.AnimateWindowFlags.AW_CENTER|
         WinAPIClass.AnimateWindowFlags.AW_SLIDE);
         this.btnAW_BLEND.Invalidate();
         this.btnHOR_AW_SLIDE.Invalidate();
         this.btnCenter_AW_SLIDE.Invalidate();
}
```

5. Постройте и запустите приложение. Протестируйте три вида анимации.

Лабораторная работа 6. Организация печати в формах windows

Цель работы

Изучение классов, реализующих задачу программирования печати и получение навыков по работе в программе с диалоговыми окнами.

Упражнение 1. Использование диалоговых окон для печати

При печати различных документов пользователям часто приходится изменять параметры печати. Обычно они ограничиваются заданием таких параметров, как ориентация страницы, ширина полей и размер бумаги.

.NET Framework содержит классы, которые предоставляют пользователям возможность осуществлять и более сложные настройки.

В ходе выполнения данного упражнения будет создано простое приложение, дающее пользователю возможность использовать **PageSetupDialog**, **PrintDialog** и **PrintPreviewDialog** для управления печатью пробного документа.

Добавление компонентов печати

1. Создайте новый проект Windows Forms. Назовите его WinPrint.

2. Из Toolbox перенесите на форму элемент **PrintDocument**.

3. С помощью окна Properties для компонента **printDocument1** добавьте обработчик события *PrintPage* и внутри него добавьте следующий код:

```
        Font myFont = new Font("Tahoma", 12, FontStyle.Regular,
GraphicsUnit.Pixel);
          string Hello = "Hello World!";
e.Graphics.DrawString(Hello, myFont, Brushes.Black, 20, 20);
```

4. Откройте **Form1** в режиме конструктора.

5. Из Toolbox перетащите три элемента **Button** на форму.

6. Укажите последовательно для кнопок значения свойства **Text**:
   - *Page Setup,*
   - *Print,*
   - *Print Preview.*

7. При необходимости увеличьте размеры кнопок.

8. Из Toolbox перетащите в форму компоненты **PageSetupDialog**, **PrintDialog** и **PrintPreviewDialog**.

9. В окне Properties укажите свойству **Document** каждого компонента диалогового окна значение **PrintDocumentl**.

10. Для элемента **PrintDialog** присвойте свойству **AllowSomePages** в значение **True**.

Реализация вызова диалоговых окон

11. В конструкторе дважды щелкните кнопку **Page Setup** и добавьте следующий код:

```
pageSetupDialog1.ShowDialog();
```

12. В конструкторе дважды щелкните **Print** и добавьте следующий код:

```
if (printDialog1.ShowDialog() == DialogResult.OK)
        printDocument1.Print();
```

13. В конструкторе дважды щелкните **Print Preview** и добавьте следующий код:

```
printPreviewDialog1.ShowDialog();
```

14. Постройте и запустите приложение. Выберите каждую из кнопок, чтобы проверить открытие различных диалоговых окон печати.

Упражнение 2. Создание документа печати

Выполнив это упражнение, вы дополните решение, разработанное в упражнении 1, и создадите приложение, разрешающее пользователю открывать текстовый файл и печатать его содержимое.

1. Откройте решение, выполненное в упражнении 1.

2. Из Toolbox перетащите элемент - диалоговое окно **OpenFileDialog** на форму.

3. В диалоговом окне Properties присвойте свойству **Filter** элемента **openFileDialog1** значение **Text Files | *.txt**, и очистите поле свойства **FileName**.

4. На форму добавьте новую кнопку с названием **Open File**.

5. Дважды щелкните **Open File**, чтобы открыть редактор кода в обработчике события Click и **выше** обработчика события добавьте код:

```
string s;
string[] strings;
int ArrayCounter = 0;
```

6. Внутри этого обработчика события добавьте код:

```
System.Windows.Forms.DialogResult aResult;
aResult = openFileDialog1.ShowDialog();
if (aResult == System.Windows.Forms.DialogResult.OK)
{
System.IO.StreamReader aReader =
 new System.IO.StreamReader(openFileDialog1.FileName);
    s = aReader.ReadToEnd();
    aReader.Close();
    strings = s.Split('\n');
```

7. В обработчике события printDocument1_PrintPage замените существующий код следующим:

```
float LeftMargin = e.MarginBounds.Left;
 float TopMargin = e.MarginBounds.Top;
  float MyLines = 0;
    float YPosition = 0;
     int Counter = 0;
      string CurrentLine;
MyLines = e.MarginBounds.Height /
this.Font.GetHeight(e.Graphics);
   while (Counter < MyLines && ArrayCounter <=
strings.Length - 1)
            {
CurrentLine = strings[ArrayCounter];
 YPosition = TopMargin + Counter *
this.Font.GetHeight(e.Graphics);
  e.Graphics.DrawString(CurrentLine, this.Font,
Brushes.Black,     LeftMargin, YPosition, new StringFormat());
Counter++;
 ArrayCounter++;
}
 if (!(ArrayCounter >= strings.GetLength(0) - 1))
  e.HasMorePages = true;
    else
        e.HasMorePages = false;
```

8. Постройте и запустите приложение. Выберите **Open File** и откройте текстовый файл на своем компьютере. Выберите **Print Preview** для просмотра файла в диалоговом окне **Print Preview**. Напечатайте файл (если принтер подключен). Попробуйте с другой страницей и параметрами печати.

Упражнение 3. Создание специализированной формы предварительного просмотра

Хотя компонент **PrintPreviewDialog** является простым, удобным в работе способом предоставить в ваших приложениях функциональность предварительного просмотра, его трудно настроить. Для приложений со специализированным предварительным просмотром можно для создания

специализированного компонента предварительного просмотра использовать элемент управления **PrintPreviewControl**.

Выполнив данное упражнение, вы создадите специализированную форму предварительного просмотра и добавите ее к решению, созданному в упражнении 2. Добавите к форме **PrintPreviewControl** элементы управления, дающие пользователю возможность указывать масштаб, количество строк и столбцов, а также переключать режим сглаживания.

1. Откройте решение, выполненное в упражнении 2.

2. Добавьте к проекту новую форму.

3. Из Toolbox перетащите **SplitContainer** в форму. В свойстве **Orientation** должна быть задана вертикаль.

4. Из Toolbox перетащите **PrintPreviewControl** в *Pane12* и присвойте свойству **Dock** значение **Fill**.

5. Для **printPreviewControl1** присвойте свойству **Modifiers** значение **Internal**.

6. Из Toolbox добавьте в *Panel1* три элемента управления **Label**, три **NumericUpDown** (надпись **Label** определяет назначение соответствующего элемента **NumericUpDown**), один **Checkbox** и один **Button**. Свяжите надписи с элементами управления **NumericUpDown** и установите свойства, как показано в следующей таблице:

Элемент управления	Свойство	Значение
Labell	Text	Rows
Label2	Text	Columns
Label3	Text	Magnification
NumericUpDown 1	Minimum	1
NumericUpDown2	Minimum	1
NumericUpDown3	Minimum	25
NumericUpDown 1	Maximum	8
NumericUpDown2	Maximum	8
NumericUpDown3	Maximum	500
NumericUpDown3	Increment	25
CheckBoxl	Text	AntiAlias
Buttonl	Text	Print

7. Дважды щелкните **NumericUpDown1** и добавьте к обработчику события numericUpDown1_ValueChanged следующий код:

```
printPreviewControl1.Rows = (int)numericUpDown1.Value;
```

8. В конструкторе дважды щелкните **NumericUpDown2** и добавьте к обработчику события numericUpDown2_ValueChanged следующий код:

```
printPreviewControl1.Columns = (int)numericUpDown2.Value;
```

Свойство **Columns** указывает количество отображаемых страниц по горизонтали, а свойство **Rows** – по вертикали.

9. В конструкторе дважды щелкните **NumericUpDown3** и добавьте к обработчику события numericUpDown3_ValueChanged следующий код

```
printPreviewControl1.Zoom = (double)numericUpDown3.Value / 100;
```

10. В конструкторе дважды щелкните **CheckBox1** и добавьте к обработчику события checkBox1_CheckedChanged следующий код:

```
printPreviewControl1.UseAntiAlias = checkBox1.Checked;
```

11. В конструкторе дважды щелкните **Button1** и добавьте к обработчику события button1_Click следующий код

```
this.DialogResult = System.Windows.Forms.DialogResult.OK;
```

12. В редакторе кода формы **Form1** закомментируйте код, существующий в обработчике события *PrintPreviewToolStripMenuItemClick,* и добавьте следующий:

```
Form2 aForm = new Form2();
System.Windows.Forms.DialogResult aResult;
 aForm.printPreviewControl1.Document = printDocument1;
  aResult = aForm.ShowDialog();
  if (aResult == System.Windows.Forms.DialogResult.OK)
    printDocument1.Print();
```

13. Постройте и выполните приложение. С помощью команды **Open** меню **File** откройте текстовый файл и затем щелкните **Print Preview** для проверки вашей новой формы предварительного просмотра.

14. Сохраните Ваше приложение и закройте Visual Studio .NET.

Лабораторная работа 7. Асинхронное программирование

Цель работы

Изучение возможностей, реализующих асинхронное программирование и получение навыков по работе в программе с потоками.

Упражнение 1. Работа с компонентом BackgroundWorker

Класс **BackgroundWorker** позволяет выполнить операцию в отдельном, выделенном потоке. Операции, требующие много времени, такие как загрузка и транзакции базы данных, могут создавать впечатление, что пользовательский интерфейс перестал отвечать на действия пользователя. Если необходимо обеспечить быстрое реагирование пользовательского интерфейса, а подобные операции приводят к длительным задержкам, эффективным решением может стать класс **BackgroundWorker**.

Чтобы запустить занимающую много времени операцию в фоновом режиме, следует создать экземпляр **BackgroundWorker** и отслеживать события, сообщающие о ходе выполнения операции и сигнализирующие о ее завершении. Можно создать объект **BackgroundWorker** программными средствами или перетащить его в форму из вкладки Компоненты Панели элементов. Класс **BackgroundWorker**, созданный в конструкторе Windows Forms, появляется в области компонентов, а его свойства отображаются в окне "Свойства".

Выполнив это упражнение, вы научитесь применять компонент **BackgroundWorker**. Вам предстоит добавить к приложению компонент **BackgroundWorker** и написать отнимающий продолжительное время

метод, который будет выполняться в отдельном потоке. Затем вы сообщите о продвижении потока и реализуете функциональность отмены фонового процесса.

1. Создайте новый проект Windows Forms. Назовите его WinBackgroundWorker.

2. Добавьте на форму элементы управления: два элемента **Label**, **TextBox**, **ProgressBar** и две кнопки **Button**.

3. Для элементов укажите свойства в соответствии с таблицей:

Элемент	**Свойство**	**Значение**
label1	Location	10;15
	Text	Second to sleep
label2	Location	10; 40
	Text	Progress
textBox1	Location	105; 13
	Size	80; 20
progressBar	Location	110; 40
	Size	240; 20
button1	Location	195; 12
	Text	Start
	Size	75; 25
button2	Location	270; 12
	Text	Cancel
	Size	75; 25
Form1	Size	370;110

4. Для элемента **textBox1** допустимыми значениями будут только цифры, поэтому в обработчике события **KeyPress** для текстового поля **textBox1** укажите код:

```
if (!char.IsDigit(e.KeyChar))
   {
     e.Handled = true;
      MessageBox.Show("Поле должно содержать цифры");
           }
```

5. Из Toolbox перетащите элемент **BackgroundWorker** в форму.

6. В окне Properties установите свойства **WorkerSupportsCancellation** и **WorkerReportsProgress** в *True* (для поддержки асинхронной отмены и возможности сообщения основному потоку информации о продвижении фонового процесса соответственно).

7. Дважды щелкните **BackgroundWorker**, чтобы открыть обработчик события backgroundWorker1_DoWork по умолчанию. Добавьте к этому обработчику события следующий код:

```
int i;
i = int.Parse(e.Argument.ToString());
 for (int j=1; j <= i; j++)
{
      if (backgroundWorker1.CancellationPending)
```

```
{
      e.Cancel = true;
    return;
}
System.Threading.Thread.Sleep(1000);
    backgroundWorker1.ReportProgress((int)(j *100/ i));
}
```

8. Для элемента **backgroundWorkerl** в окне Properties щелкните кнопку Events, затем дважды щелкните *ProgressChanged,* чтобы открыть окно кода обработчика события backgroundWorker1_ProgressChanged. Добавьте следующий код:

```
progressBar1.Value = e.ProgressPercentage;
```

9. Аналогичным способом добавьте обработчик события *RunWorkerCompleted,* в теле обработчика добавьте следующий код:

```
if (!(e.Cancelled))
   System.Windows.Forms.MessageBox.Show("Run Completed!");
    else
System.Windows.Forms.MessageBox.Show("Run Cancelled");
```

10. Для кнопки **Start** откройте обработчик события *Click*. Добавьте следующий код:

```
if (!(textBox1.Text == ""))
    {
int i = int.Parse(textBox1.Text);
backgroundWorker1.RunWorkerAsync(i);
    }
```

11. Для кнопки **Cancel** откройте обработчик события *Click*. Добавьте следующий код

```
  backgroundWorker1.CancelAsync();
```

12. Постройте и выполните приложение, и проверьте его функциональность.

Упражнение 2. Использование делегатов

Выполнив это упражнение, вы создадите приложение, подобное тому, что было создано в упражнении 1. Оно будет выполнять продолжительную операцию в отдельном потоке с возможностью отмены, показывать продвижение операции и уведомлять пользователя о ее завершении. Для реализации этой функциональности вы используете делегаты и асинхронный вызов.

1. Создайте новое Windows-приложение и назовите его WinAsynchDelegate.

2. Повторите шаги 2 – 4 предыдущего упражнения для создания требуемой формы.

3. Добавьте в приложение следующий метод:

```
private void TimeConsumingMethod(int seconds)
        {
            for (int j = 1; j <= seconds; j++)
             System.Threading.Thread.Sleep(1000);
      }
```

4. Используйте в приложении делегат, соответствующий методу *TimeConsumingMethod:*

```
    private delegate void TimeConsumingMethodDelegate(int
seconds);
```

5. Добавьте сообщающий о продвижении операции метод, который устанавливает значение элемента управления *ProgressBar* потокобезопасным способом, и делегата этого метода:

```
public delegate void SetProgressDelegate(int val);
        public void SetProgress(int val)
        {
            if (progressBar1.InvokeRequired)
            {
          SetProgressDelegate del = new
SetProgressDelegate(SetProgress);
                this.Invoke(del, new object[] { val });
            }
            else
            {
                progressBar1.Value = val;
            }
        }
```

6. Для сообщения о продвижении операции добавьте следующую строку кода в цикл *For* метода *TimeConsumingMethod:*

```
SetProgress((int)(j * 100) / seconds);
```

7. Добавьте в приложение логическую переменную с именем *Cancel*:

```
bool Cancel;
```

8. Добавьте следующие строки кода в цикл *For* метода *TimeConsumingMethod:*

```
if (Cancel)
    break;
```

9. Добавьте следующий код после цикла *For* метода *TimeConsumingMethod:*

```
if (Cancel)
 {
System.Windows.Forms.MessageBox.Show("Cancelled");
Cancel = false;
   }
 else
   {
System.Windows.Forms.MessageBox.Show("Complete");
            }
```

10. В конструкторе дважды щелкните кнопку GO!, чтобы открыть для нее обработчик события *Click* по умолчанию, и добавьте следующий код:

```
TimeConsumingMethodDelegate del = new
TimeConsumingMethodDelegate(TimeConsumingMethod);
      del.BeginInvoke(int.Parse(textBox1.Text), null, null);
```

11. В конструкторе дважды щелкните кнопку Cancel, чтобы открыть для нее обработчик события *Click* по умолчанию, и добавьте следующий код:

```
Cancel = true;
```

12. Скомпилируйте и проверьте ваше приложение.

Упражнение 3. Асинхронный запуск произвольного метода

При разработке программного обеспечения наиболее часто требуется запускать асинхронно собственные методы. В .NET Framework можно асинхронно вызывать любой метод. Для этого необходимо определить делегат с той же сигнатурой, что и у вызываемого метода.

Среда CLR автоматически определяет для этого делегата методы **BeginInvoke** и **EndInvoke** с соответствующими сигнатурами.

Для асинхронного запуска нужно проделать следующие шаги:

1. Создать и запустить делегат с необходимой сигнатурой. После этого можно работать со своим методом так же, как и с методами со встроенной поддержкой асинхронной модели программирования.
2. Выбрать механизм оповещения о завершении и подготовить для него все необходимое.
3. Запустить метод асинхронно.
4. Получить результаты в основном потоке и обновить пользовательский интерфейс.

Хотя компонент **BackgroundWorker** обеспечивает удобный способ выполнения простых задач в фоновом потоке, иногда может потребоваться осуществить более тонкий контроль за фоновыми процессами. В этом упражнении вы научитесь асинхронно выполнять методы с использованием делегатов.

1. Создайте новое Windows-приложение и назовите его WinAsynchMethod.
2. Установите свойствам формы Size значение 425;200 и Text – "Асинхронный запуск".
3. Добавьте на форму три надписи, два текстовых поля и две кнопки, и установите им следующие свойства:

Свойство	**Значение**
button1	
Name	btnRun
Location	16; 64
Text	Сумма
button2	
Name	btnWork
Location	120; 128
Text	Работа
label1	
Name	lblA
Location	8; 24
Text	Значение A
label2	
Name	lblB
Location	216; 24
Text	Значение B

Свойство	Значение
textBox1	
Name	txbA
Location	88; 24
Text	
textBox2	
Name	txbB
Location	296; 24
Text	

4. Создайте делегат:

```
private delegate int AsyncSumm(int a, int b);
```

5. Создайте метод Summ, в котором будут складываться числа, вводимые в два текстовых поля, и укажите задержку операции на 9 секунд:

```
private int Summ(int a, int b)
{
      System.Threading.Thread.Sleep(9000);
      return a+b;
}
```

6. Реализуйте обработчик кнопки **btnRun**, который будет включать также действия по организации асинхронного вызова:

Создайте экземпляр делегата и проинициализируйте его методом Summ:

```
AsyncSumm summdelegate = new AsyncSumm(Summ);
```

Для использования механизма **Callback** создайте экземпляр делегата AsyncCallBack:

```
AsyncCallback cb = new AsyncCallback(CallBackMethod);
```

После того как делегат инициализирован методом, можно запускать прикрепленный к делегату метод асинхронно с помощью метода **BeginInvoke**. Этот метод принимает две переменные типа int a и b, экземпляр cb делегата AsyncCallback и экземпляр summdelegate делегата SummDelegate:

```
summdelegate.BeginInvoke(a, b, cb, summdelegate);
```

7. В итоге обработчик кнопки **btnRun** будет выглядеть следующим образом:

```
private void btnRun_Click(object sender, System.EventArgs e)
     {
          int a, b;
     try
          {
               // Преобразование типов данных.
          a = Int32.Parse(txbA.Text);
          b = Int32.Parse(txbB.Text);
          }
     catch(Exception)
          {
     MessageBox.Show("При выполнении преобразования типов
возникла ошибка");
          txbA.Text = txbB.Text = "";
          return;
          }
```

```
            AsyncSumm summdelegate = new AsyncSumm(Summ);
            AsyncCallback cb = new
AsyncCallback(CallBackMethod);
            summdelegate.BeginInvoke(a, b, cb, summdelegate);
            }
```

8. Создайте метод CallBackMethod, который привязан к делегату summdelegate:

```
      private void CallBackMethod(IAsyncResult ar)
            {
            string str;
            AsyncSumm summdelegate = (AsyncSumm)ar.AsyncState;
            str = String.Format("Сумма введенных чисел равна
{0}", summdelegate.EndInvoke(ar));
      MessageBox.Show(str, "Результат операции");
            }
```

9. Для демонстрации асинхронности выполнения метода реализуйте обработчик нажатия кнопки **Работа**, например, следующим образом:

```
      MessageBox.Show("Работа кипит!!!");
```

10. Постройте и запустите приложение. После нажатия кнопки **Сумма,** пока будет выполняться операция, нажмите кнопку **Работа**. Проверьте, что метод суммирования действительно реализован асинхронно.

Лабораторная работа 8. Повышение удобства использования приложений

Цель работы

Изучение средств для повышения удобства работы пользователей и получение навыков по созданию контекстной справки, всплывающей подсказки, а также файлов со справочной информацией.

Упражнение 1. Создание контекстной справки

Важной частью любого приложения является понятная и точная документация. Снабдить ваше приложение справкой позволяет компонент *HelpProvider.*

1. Откройте Windows-приложение WinAsynchMethod.
2. Откройте форму в режиме конструктора.
3. Выберите пункт меню **View□ToolBox**.
4. Добавьте ЭУ **HelpProvider** на форму.
5. Выделите поле **txbA** для отображения ее свойств.
6. Для свойства **HelpString on helpProvider1** задайте значение **For input integer A.**
7. Постройте и запустите приложение.
8. Переместитесь по форме, используя клавишу **Tab**, до тех пор, пока поле **txbA** не окажется в фокусе.
9. Нажмите на клавишу **F1** для отображения контекстной справки для поля **txbA.**

Простые формы обычно в своем заголовке имеют кнопку с вопросительным знаком, при нажатии на которую курсор меняет свой вид на изображение с вопросом. При щелчке на выбранном элементе управления появляется его краткое описание (подсказка).

Создайте подобную функциональность на форме проекта WinAsynchMethod:

1. Добавьте к имеющимся свойствам формы следующие свойства:

Свойство	**Значение**
MaximizeBox	False
MinimizeBox	False
HelpButton	True
FormBorderStyle	FixedDialog

2. Для полей ввода **txbA**, **txbB** и двух кнопок в свойстве **ShowHelp on helpProvider1** каждого из этих элементов установите значение **True**.

3. Текст, введенный в поле свойства **HelpString on helpProvider1**, будет появляться в качестве подсказки для конкретного элемента. Установите следующие значения этого свойства для каждого элемента:

txbA	For input integer A
txbB	For input integer B
btnRun	Sum
btnWork	Start work

4. Постройте и запустите приложение.

5. Для активации контекстной справки нажмите на кнопку “?”, расположенную в правом верхнем углу приложения.

6. Нажмите на любую кнопку, появится маленькое окошко, объясняющее, что происходит при ее нажатии.

Упражнение 2. Использование справочного файла

1. Откройте Windows-приложение WinAsynchMethod.
2. Откройте форму в режиме конструктора.
3. Выберите пункт меню **View□ToolBox**.
4. Добавьте ЭУ **HelpProvider** на форму (если он не был добавлен ранее).
5. В папке с решением создайте файл справки, например, документ Microsoft Word. Текст укажите произвольный.
6. Для элемента **helpProvider1** в свойстве **HelpNamespace** укажите путь к файлу справки.
7. Реализуйте возможность вызова файла справки созданием либо команды меню, либо кнопки (и команда меню и кнопка может называться, например, **help**).
8. Создайте обработчик события выбора файла справки. В теле обработчика укажите следующую строку:

```
Help.ShowHelp(this,helpProvider1.HelpNamespace);
```

9. Постройте и запустите приложение.

10. Выберите команду вызова справки. Проверьте, что открылся требуемый файл.

Упражнение 3. Добавление всплывающих подсказок

Компонент *ToolTip* позволяет назначить элементам управления подсказки. Они появляются в окнах, когда мышь находится над элементом управления, и могут предоставлять пользователю краткие сведения о нем.

1. Откройте Windows-приложение WinAsynchMethod в режиме конструктора.
2. Выберите пункт меню **View→ToolBox**.
3. Добавьте на форму элемент управления **ToolTip**.
4. В окне **Properties** расположенных на форме элементов и в самой форме появилось свойство **ToolTip on toolTip1**. Установите следующие значения этого свойства для каждого из элементов:

txbA	For input integer A
txbB	For input integer B
btnRun	Sum
btnWork	Start work

5. Постройте и запустите приложение. Проверьте, что при наведении курсора на элемент управления появляется его подсказка.

Упражнение 4. Автоматический выбор языка при запуске приложения

При распространении приложения часто бывает необходимо обеспечивать пользователям возможность работать в своей языковой среде. В связи с этим, при разработке приложений приходится задумываться о переводе пользовательского интерфейса на другие языки. На практике используется два способа решения данной проблемы, первый: создается локальная версия целиком на одном языке и второй: программы содержат многоязычный интерфейс, позволяющий менять оформление приложения непосредственно в ходе работы.

В ходе установки операционной системы Windows при определении региональных параметров пользователю предлагается выбрать язык стандартов и форматов. Выбранное значение доступно для изменения в дальнейшем — меню “Пуск” | “Панель управления” | “Язык и региональные параметры” | вкладка “Региональные параметры”.

В этом упражнении вы создадите приложение, которое автоматически будет определять установленный язык стандартов и выводить соответствующий пользовательский интерфейс.

1. Создайте новое Windows-приложение и назовите его WinLanguage.
2. Добавьте на форму кнопку и главное меню.
3. Установите следующие значения свойства **Text** для элементов:

   главное меню – **menu**,

   первая команда – **command one**,

   вторая команда – **command two**,

кнопка – **Close**,

для самой формы – **Form**.

4. Для кнопки реализуйте обработчик, закрывающий форму:

```
this.Close();
```

5. Для формы установите свойству **Localizable** значение **True**. Это свойство разрешает поддержку многоязычного интерфейса.

6. В свойстве **Language** выберите значение **English (United States)**.

7. Постройте приложение. В итоге получилась версия программы с интерфейсом на английском языке.

8. В свойстве **Language** выберите **Russian (Russia).**

9. Измените свойство **Text** элементов, заменив названия на английском языке соответствующими названиями на русском.

10. Перейдите в код формы и подключите пространство имен **Threading**:

```
using System.Threading;
```

11. В конструкторе формы до InitializeComponent(); установите культуру пользовательского интерфейса равной текущей культуре:

```
Thread.CurrentThread.CurrentUICulture =
Thread.CurrentThread.CurrentCulture;
```

12. Постройте и запустите приложение. Среда CLR проверяет установленный язык и выводит приложение с интерфейсом на языке, установленном на вкладке "Региональные параметры" в настройках инструмента "Язык и региональные параметры".

13. Закройте приложение. Перейдите в "Панель управления" и установите другой язык (например, "Английский (США)") на вкладке "Региональные параметры" в настройках инструмента "Язык и региональные параметры". Снова запустите приложение. Теперь интерфейс приложения должен быть на другом языке (например, английском).

14. При локализации приложения среда Visual Studio .NET создает сборку, в которой хранятся все данные о приложении. В окне Solution Explorer нажмите на кнопку (**Show All Files**) для просмотра добавленных файлов. Названия файлов-ресурсов (Form1. en-US и Form1.ru-RU) содержат в себе указание на язык (первая часть – en или ru) и регион (вторая часть US или RU).

Упражнение 5. Локализация приложения

Реализовать локализацию, т.е. предоставить пользовательский интерфейс, характерный для текущего региона, можно с помощью встроенных в Visual Studio средств локализации.

Visual Studio позволяет создавать альтернативные версии культурозависимых форм и автоматически управляет поиском ресурсов, соответствующих данной культуре.

Для пользовательского интерфейса культура предоставляется экземпляром *CultureInfo* и отличается от свойства

CulturInfo.CurrentCulture. В то время как оно определяет формат, применяемый к системно-форматируемым данным, *CurrentUICulture* определяет ресурсы, загружаемые в локализованные формы во время выполнения. Культура пользовательского интерфейса устанавливается в свойстве *CurrentThread. CurrentUICulture*

В этом упражнении вы локализуете пользовательский интерфейс приложения и добавите в него локализованные строковые ресурсы.

Локализация формы Windows-приложения

1. Откройте стартовый проект UsabilityDemo.sln из папки **install_folder**\Practices\Mod08\Mod08_04\Starter_2008.
2. Откройте файл UsabilityDemo.cs в режиме конструктора.
3. Нажмите на кнопку **Show All Files** в окне **Solution Explorer**.
4. Обратите внимание на значение свойства формы **Localizable**. Оно установлено в **True**, что означает, что форма может быть локализована.
5. Для свойства формы **Language** задайте значение **French(France).**

В данный момент вы видите английскую версию формы, но теперь вы можете преобразовать ее во французскую. Обратите внимание на то, что в окне **Solution Explorer** под файлом UsabilityDemo.cs появилось несколько новых файлов ресурсов (UsabilityDemo.fr.resx и UsabilityDemo.fr-FR.resx). Обратите внимание на то, что данная форма также была локализована для Германии и Японии.

6. Для свойства формы **Text** задайте значение **Démonstration de l'utilisation**.

***Замечание*:** Текст, необходимый для создания французской версии формы можно скопировать из файла UsabilityDemoLocalizedStrings.rtf, расположенного в папке **install_folder**\Practices\Mod08\Mod08_04.

7. Для свойства **Text** меню **Help** и пункта меню **Help** задайте значение **Aide.**
8. Для свойства **Text** кнопки **Choose a Culture** задайте значение **Choisir une langue.**
9. Для свойства **Text** кнопки **Show Date/Time** задайте значение **Afficher la date/l'heure.**
10. Для свойства **Text** кнопки **Show Currency** задайте значение **Afficher la devise.**
11. Для свойства **Text** кнопки **Show a String** задайте значение **Afficher une chaîne.**
12. Для свойства **Text** кнопки **Exit** задайте значение Quitter**.**

Добавление в приложение файл строковых ресурсов

1. В окне **Solution Explorer** ПКМ по проекту **UsabilityDemo | Add | Add New Item.**
2. В окне **Add New Item** выберите **Assembly Resource File.**
3. Задайте для файла ресурсов имя **UsabilityDemoText.fr-FR.resx** и нажмите на кнопку **Open**.
4. В строке ресурсов задайте для **Name** значение **SimpleTextString.**

5. В строке ресурсов задайте для **Value** значение **Voici du texte.**

6. Сохраните и закройте файл ресурсов.

Добавление кода для получения значений строковых ресурсов

1. Откройте файл с кодом UsabilityDemo.cs.

2. В окне Task List отобразите комментарии TODO. Для этого выберите пункт меню **View | Show Tasks | All.**

3. Добавьте три директивы **using,** для возможности поддержки локализации приложения.

```
using System.Globalization;
using System.Resources;
using System.Threading;
```

4. Найдите в коде второй комментарий TODO. Объявите **private** переменную типа **ResourceManager**.

```
private ResourceManager RM;
```

5. Найдите в коде следующий комментарий TODO. Создайте экземпляр класса **ResourceManager.** Код добавляется в первый конструктор приложения:

```
RM = new ResourceManager("UsabilityDemo.UsabilityDemoText",
Assembly.GetExecutingAssembly());
```

6. Найдите в коде следующий комментарий TODO. Создайте экземпляр класса **ResourceManager.** Код добавляется во второй конструктор приложения:

```
RM = new ResourceManager("UsabilityDemo.UsabilityDemoText",
Assembly.GetExecutingAssembly());
```

7. Найдите в коде следующий комментарий TODO. Добавьте код использования менеджера ресурсов для получения строки текста из файла ресурсов и отображения ее в текстовом поле.

```
OutputTextBox.Text = RM.GetString("SimpleTextString");
```

8. Найдите в коде следующий комментарий TODO. Добавьте код для задания свойствам **Culture** и **UICulture** значений, выбранных пользователем.

```
    Thread.CurrentThread.CurrentUICulture = new
CultureInfo(ChosenCulture, false);
    Thread.CurrentThread.CurrentCulture = new
CultureInfo(ChosenCulture, false);
```

9. Сохраните проект.

Тестирование работы приложения

1. Постройте и запустите приложение.

2. Обратите внимание на то, что при старте приложения все, что вы можете – это либо выбрать язык, либо завершить работу приложения.

3. Нажмите на кнопку **Choose a Culture**.

4. Выберите один из языков. Если вы не выберите ни один из них, по умолчанию будет использоваться English.

5. После нажатия на кнопку **OK** на форме **Culture Chooser** эта форма исчезает и появляется та версия формы **UsabilityDemo,** которая соответствует выбранным настройкам.

Лабораторная работа 9. Развертывание windows приложений

Цель работы

Изучение средств для управления глобальным хранилищем сборок и получение навыков по созданию дистрибутивов, предоставляющих возможность распространять программы с минимумом усилий со стороны конечного пользователя.

Упражнение 1. Использование строго именованной сборки

В этом упражнении вы сгенерируете цифровую подпись и создадите ссылку на данный файл из библиотеки CalculatorEngine. Далее вы создадите приложение WindowsCalculator, ссылающееся на строго именованную сборку. С помощью программы ILDASM вы просмотрите метаданные для CalculatorEngine.dll и WindowsCalculator.exe.

Создание строго именованной сборки

1. Откройте окно командной строки, выбрав **Start |All Programs | Microsoft Visual Studio .NET | Visual Studio .NET Tools | Visual Studio .NET Command Prompt**.
2. В окне командной строки перейдите в директорию **install_folder**\Practices\Mod09\Mod09_01\Starter\CalculatorEngine.
3. Наберите команду **sn –k CalcKey.snk.** Она сгенерирует файл с цифровой подписью, которую вы сможете использовать при создании строго именованной сборки.
4. В Visual Studio .NET откройте проект CalculatorEngine.sln из папки **install_folder**\Practices\Mod09\Mod09_01\Starter\CalculatorEngine.
5. Просмотрите комментарии TODO в окне Task List. Для этого выберите пункт меню **View | Show Tasks | All.**
6. Откройте файл AssemblyInfo.cs
7. Перейдите к фрагменту кода, связанному с первым комментарием TODO. Измените номер версии с 2.0.1.1 на **3.0.1.1.**

```
[assembly: AssemblyVersion("3.0.1.1")]
```

8. Перейдите к следующему комментарию TODO. Внизу файла добавьте новый атрибут для ссылки на файл, содержащий цифровую подпись для создания строго имени.

```
[assembly: AssemblyKeyFile("CalcKey.snk")]
```

9. Откройте файл с кодом Calculator.cs.
10. Перейдите к комментарию TODO. Измените информацию о версии с v2.0.1.1 на **v3.0.1.1.**

```
private static string versionInfo = "Calculator v3.0.1.1";
```

11. Перестройте сборку и закройте Visual Studio .NET.
12. В окне командной строки Visual Studio .NET перейдите в директорию **install_folder**\Practices\Mod09\Mod09_01\Starter\CalculatorEngine\bin\Debug.
13. В окне командной строки запустите ILDASM для CalculatorEngine.dll.

14. Раскройте узел MANIFEST.

15. Обратите внимание, что под **.assembly CalculatorEngine** указан **.publickey.** Это указывает на то, что CalculatorEngine является строго именованной сборкой.

Создание приложения, ссылающееся на строго именованную сборку

1. Откройте проект WindowsCalculator.sln из папки **install_folder**\Practices\Mod09\Mod09_01\Starter. Перестройте проект.

2. Запустите приложение и обратите внимание на то, что номер версии 3.0.1.1. Это номер версии CalculatorEngine.dll.

3. В окне командной строки Visual Studio .NET перейдите в директорию **install_folder**\Practices\Mod09\Mod09_01\Starter\bin\Debug.

4. Запустите ILDASM для WindowsCalculator.exe, набрав следующую команду:

```
ildasm WindowsCalculator.exe
```

5. Раскройте узел MANIFEST. Обратите внимание, что для **.assembly extern CalculatorEngine** указаны номер версии 3:0:1:1 и **.publickeytoken.** Это указывает на то, что приложение ссылается на внешнюю строго именованную сборку.

Упражнение 2. Работа с глобальным кэшем сборок

В этом упражнении вы добавите в глобальный кэш сборок (GAC) сборку CalculatorEngine. Затем вы построите приложение WindowsCalculator. Перед запуском приложения Вы удалите локальные копии CalculatorEngine.dll. После чего Вы убедитесь в успешном запуске приложения WindowsCalculator, т.к. ему удастся обнаружить нужную версию CalculatorEngine в глобальном кэше сборок.

Внимание: В процессе разработки Вы можете использовать утилиту GACUtil.exe для добавления сборки в глобальный кэш сборок на этапе тестирования. Это программа используется для удобства и не должна использоваться при развертывании production-версии приложения. В этом случае необходимо добавлять сборки в глобальный кэш сборок с помощью Windows Installer или .NET Framework configuration tool Mscorcfg.msc.

Построение новой версии сборки CalculatorEngine

1. Откройте проект CalculatorEngine.dll из папки **install_folder**\Practices\Mod09\Mod09_02\Starter\ CalculatorEngine.

2. Просмотрите комментарии TODO в окне Task List. Для этого выберите пункт меню **View | Show Tasks | All.**

3. Откройте файл AssemblyInfo.cs

4. Перейдите к фрагменту кода, связанному с первым комментарием TODO. Измените номер версии с 3.0.1.1 на **4.0.1.1.**

```
[assembly: AssemblyVersion("4.0.1.1")]
```

5. Откройте файл с кодом Calculator.cs.

6. Перейдите к комментарию TODO. Измените информацию о версии с v3.0.1.1 на **v4.0.1.1.**

```
private static string versionInfo = "Calculator v4.0.1.1";
```

7. Перестройте сборку и закройте Visual Studio .NET.

Добавление сборки в глобальный кэш сборок, используя утилиту GACUtils

1. Откройте окно командной строки, выбрав **Start | All Programs | Microsoft Visual Studio.NET | Visual Studio .NET Tools | Visual Studio .NET Command Prompt**.
2. В окне командной строки перейдите в директорию **install_folder**\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug.
3. Добавьте сборку CalculatorEngine в глобальный кэш сборок с помощью следующей команды:

```
gacutil –i CalculatorEngine.dll
```

4. Вы должны получить сообщение **Assembly successfully added to the cache.**
5. Просмотрите содержимое глобального кэша сборок с помощью следующей команды:

```
gacutil –l
```

6. Найдите в списке CalculatorEngine. Запись должна отражать следующую информацию: “CalculatorEngine, Version=4.0.1.1, Culture-neutral, PublicKeyToken=***, Custom=null”.

Построение приложения, ссылающееся на строго именованную сборку, располагающуюся в глобальном кэше сборок

1. Откройте проект WindowsCalculator.dll из папки **install_folder**\Practices\Mod09\Mod09_02\Starter.
2. В окне Solution Explorer раскройте узел References.
3. Выберите ПКМ **References | Add Reference.**
4. В ОД **Add Reference** нажмите на кнопку **Browse** и перейдите папку **install_folder**\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug, где располагается CalculatorEngine.dll. Нажмите на кнопку **Open**, а затем на кнопку **OK**.
5. Откройте файл с кодом CalcUI.cs.
6. Просмотрите комментарии TODO в окне Task List. Для этого выберите пункт меню **View | Show Tasks | All.**
7. Перейдите к фрагменту кода, связанному с комментарием TODO. Добавьте директиву **using** для доступа к Calculator.

```
using Calculator;
```

8. Постройте приложение WindowsCalculator.
9. В окне Windows Explorer переименуйте локальные копии **CalculatorEngine.dll** в **OLD_Calculator.dll.** Это относится к **install_folder**\Practices\Mod09\Mod09_02\Starter\bin\Debug, **install_folder**\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug и **install_folder**\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\obj\Debug.

10. В окне Windows Explorer перейдите в папку **install_folder**\Practices\Mod09\Mod09_02\Starter\bin\Debug.

11. Запустите приложение WindowsCalculator.exe. Обратите внимание на то, что приложение WindowsCalculator успешно запустится, несмотря на то, что локальная копия сборки CalculatorEngine.dll не найдена. Это происходит потому, что среда выполнения находит нужную сборку в глобальном хранилище сборок и загружает ее.

Упражнение 3. Создание и использование файлов конфигурации приложения

В этом упражнении вы добавите в глобальный кэш сборок новую версию сборки **CalculatorEngine.** Далее Вы построите файл конфигурации приложения для приложения WindowsCalculator, заставляющий приложение ссылаться на более новую версию **CalculatorEngine.** Вам не придется перестраивать приложение WindowsCalculator для того, чтобы оно использовало более новую версию **CalculatorEngine.**

Добавление новой версии сборки CalculatorEngine в глобальный кэш сборок

1. Откройте окно командной строки.
2. В окне командной строки перейдите в директорию **install_folder**\Practices\Mod09\Mod09_03\Starter.
3. Удалите CalculatorEngine.dll. Это версия 4.0.1.1, на которую в данный момент ссылается WindowsCalculator.
4. Скопируйте **CalculatorEngine.dll.v5011** в **CalculatorEngine.dll.**
5. Добавьте сборку **CalculatorEngine** в глобальный кэш сборок, используя следующую команду:

```
gacutil –i CalculatorEngine.dll
```

6. Вы должны получить сообщение **Assembly successfully added to the cache.**
7. Просмотрите содержимое глобального кэша сборок с помощью следующей команды:

```
gacutil –l
```

8. Найдите в списке **CalculatorEngine**. В списке должно быть две версии **CalculatorEngine:** версия 4.0.1.1 и версия 5.0.1.1.

Создание файла конфигурации приложения для приложения WindowsCalculator

1. Запустите .NET Framework configuration tool (**Mscorcfg.msc**).
2. Раскройте узел **Applications**.
3. Выберите **Add an Application to Configure.**
4. В ОД **Configure an Application** выберите **Other** для указания местонахождения сборки.
5. Перейдите в папку **install_folder**\Practices\Mod09\Mod09_03\Starter, выберите **WindowsCalculator.exe** и нажмите **Open.**

6. В Mscorcfg.msc под Applications раскройте узел приложения WindowsCalculator.exe и выберите **Configured Assemblies**.

7. На правой панели щелкните по ссылке **Configure an Assembly**.

8. В ОД **Configure an Assembly** выберите **Choose an assembly from the assembly cache**.

9. Нажмите на кнопку **Choose Assembly**.

10. В ОД **Choose Assembly from Assembly Cache,** найдите сборку CalculatorEngine версии 5.0.1.1. Выберите CalculatorEngine и нажмите на кнопку **Select**.

11. Нажмите на кнопку **Finish.**

12. Появится ОД **CalculatorEngine Properties**.

13. Выберите вкладку **Binding Policy.**

14. В столбце **Requested Version** наберите **4.0.1.1**

15. В столбце **New Version** наберите **5.0.1.1**

16. Нажмите на кнопку **Apply**, а затем на кнопку **OK**.

17. С помощью программы Notepad откройте файл **install_folder**\Practices\Mod09\Mod09_03\Starter\WindowsCalculator.exe.config и посмотрите его содержимое. Обратите внимание на тэг **<bindingRedirect>.** Закройте Notepad.

18. Запустите приложение **WindowsCalculator.exe.** Убедитесь в том, что оно использует новую версию сборки **CalculatorEngine.** Номер версии будет 5.0.1.1.

Упражнение 4. Создание и использование Windows Installer Setup Project

В этом упражнении вы создадите пакет установки **Windows Installer Setup Project** и добавите в него приложение WindowsCalculator. Затем Вы протестируете корректность установки и работы приложения.

Создание setup-проекта

1. Запустите Visual Studio .NET. На стартовой странице выберите **New Project.**

2. В окне **New Project** на панели **Project Type** выберите **Setup and Deployment Projects**.

3. Укажите тип создаваемого проекта **Setup Project.**

4. Укажите место расположения **install_folder**\Practices\Mod09\Mod09_04\Starter.

5. Задайте имя **WinCalc** и нажмите **OK.**

6. В дереве **File System** указаны папки для приложения, ярлыков рабочего стола и главного меню. Добавьте папку для хранения сборки, которая при развертывании приложения добавится в глобальный кеш сборок. Откройте контекстное меню **File System on Target Machine,** далее **Add Special Folder | Global Assembly Cache Folder.**

7. Откройте контекстное меню папки **Global Assembly Cache Folder | Add | File…**

8. В ОД **Add Files** перейдите в папку **install_folder**\Practices\Mod09\Mod09_04\Starter и добавьте сборку **CalculatorEngine.dll.**

9. Добавьте в папку **Application Folder** приложение **WindowsCalculator.exe**: в контекстном меню **Application Folder** выберите **Add | File**. В ОД **Add Files** укажите **WindowsCalculator.exe** и нажмите **Open.**

10. Добавьте в папку **Application Folder** файл **KEYS03.ico.**

11. В окне свойств **Properties** для папки **Application Folder** для свойства **DefaultLocation** задайте значение **[ProgramFilesFolder]\[ProductName].** Для свойства **AlwaysCreate** задайте значение **True**.

12. В окне **Solution Explorer** разверните **Detected Dependencies.** В папке **Detected Dependencies** в контекстном меню сборки **CalculatorEngine.dll** выберите **Exclude.**

13. Для создания ярлыка на рабочем столе пользователя:
    - Выберите **Application Folder** и щелкните по **WindowsCalculator.exe.**
    - В контекстном меню выберите команду **Create ShortCut to WindowsCalculator.exe.** Переименуйте его в **Windows Calculator.**
    - В окне **Properties** убедитесь, что для свойства **Target** задано значение **WindowsCalculator.**
    - Для свойства **Icon** выберите **Browse**.
    - В ОД **Icon** выберите **Browse.**
    - Из выпадающего списка **Look In** выберите **Application Folder | OK** и щелкните по **KEYS03.ico.**
    - Нажмите **OK,** а затем еще раз **OK** в ОД **Icon.**
    - Перетащите ярлык из **Application Folder** в узел **User's Desktop**.

14. Для создания ярлыка в меню **Program** повторите предыдущие шаги, создав ярлык с тем же именем. Перетащите ярлык в узел **User's Program Menu**

15. В окне **Properties** для проекта **WinCalc** задайте следующие значения для указанных свойств.

Свойство	**Значение**
Author	I'm
Manufacturer	LIMTU
ProductName	WinCalc
Title	Windows Calculator

16. Сохраните и постройте setup-проект.

Установка и запуск приложения

1. В окне Windows Explorer перейдите в папку **install_folder**\Practices\Mod09\Mod09_04\Starter\WinCalc\Debug.

2. Двойным щелчком по **Setup.exe** запустите инсталляцию приложения Windows Calculator.

3. Три раза нажмите **Next.** Приложение установится на ваш компьютер.

4. После окончания установки нажмите **Close.**

5. Запустите с рабочего стола программу **Windows Calculator.** Должно запуститься приложение, использующее версию 6.0.1.1. сборки **CalculatorEngine**.

6. Закройте приложение.

7. Выберите **Start | All Programs | Windows Calculator.** Должно запуститься приложение, использующее версию 6.0.1.1. сборки **CalculatorEngine**.

8. Закройте приложение.

9. В окне Windows Explorer перейдите в папку C:\Program Files\WinCalc.

10. В этой папке вы увидите WindowsCalculator.exe, но там не будет CalculatorEngine.dll.

11. В окне Windows Explorer перейдите в папку C:\WINDOWS\assembly. В этой папке вы увидите в глобальном кэше сборок сборку CalculatorEngine версии 6.0.1.1.

Упражнение 5. Публикация приложения с помощью ClickOnce в сетевой папке

В этом упражнении вы опубликуете приложение в сетевой папке, используя ClickOnce, настроите место публикации и загрузки, а затем опубликуете приложение и загрузите его, используя ClickOnce.

1. В корневом каталоге создайте общую папку с именем C:\ClickOnce.

2. Откройте приложение, которое хотите опубликовать.

3. В Solution Explorer щелкните проект правой кнопкой мыши и затем щелкните команду **Publish...(Опубликовать**). Откроется мастер публикации.

4. В окне Publish Location укажите место публикации C:\ClickOnce, нажмите кнопку **Next** (**Далее)**.

5. В следующем окне установите способ установки приложения пользователями, выбрав вариант из общей папки и указав значение в виде ***computername*\ClickOnce**, где ***computername*** является именем вашего компьютера. Нажмите кнопку **Next** (**Далее)**.

6. Выберите вариант доступности в автономном режиме (оставьте по умолчанию) Нажмите кнопку **Next** (**Далее)**.

7. Прочтите информацию об установке и нажмите **Finish** (**Готово**), чтобы опубликовать ваше приложение в сетевой папке. ClickOnce проверяет требования приложения и публикует его, после чего открывается папка публикации.

8. Откройте папку C:\ClickOnce и дважды щелкните **Setup**. Вы можете увидеть предупреждение о безопасности. Если это так, щелкните

Установить. Приложение устанавливается и открывается на вашем компьютере. Обратите внимание, что к вашему меню **Пуск** добавляется новая группа.

8. В меню **Пуск** откройте **Панель управления** и выберите **Установка и удаление программ**. Щелкните значок своего приложения и выберите **Заменить/удалить**. Следуйте экранным инструкциям для удаления приложения с вашего компьютера.

Лабораторная работа 10. Подключение к базе данных

Цель работы

Изучение классов, предоставляющих службы доступа к данным и получение навыков по использованию компонентов ADO.NET.

Упражнение 1. Организация доступа к данным и работа с объектом DataReader

В этом упражнении вы получите объекты **DataReader** с помощью объекта **DataCommand** для чтения данных непосредственно в приложение.

Создание нового соединения

Прежде всего, следует создать соединение с БД при помощи окна Server Explorer.

1. Создайте в Visual Studio новое приложение Windows Forms. Назовите его **WinBD**.

2. В окне Server Explorer щелкните правой кнопкой узел **Data Connections** и выберите команду **Add Connection** — при первом добавления подключения откроется диалоговое окно **Choose Data Sourse**. Если вместо диалогового окна **Choose Data Source** появится диалоговое окно **Add Connection**, щелкните кнопку **Change**, в случае необходимости в изменении источника данных.

3. В диалоговом окне **Choose Data Source** выберите источник данных, к которому хотите подключиться – **Microsoft Access Database File (Файл базы данных Microsoft Access)**, а также провайдер данных для подключения. Обратите внимание, как нужный провайдер данных автоматически заполнился при указании источника данных.

4. В окне **Add Connection** в поле **Database file name** щелкните кнопку **Brouse**…

5. Найдите файл Конфетная фабрика.mdb в папке \Labs_БД и выберите его, щелкнув кнопку **Open**.

6. Щелкните кнопку **Test Connection** (Проверка соединения), чтобы проверить соединение, — должно появиться сообщение об успешной проверке. Если такого сообщения вы не получили, вернитесь к пункту 5 и убедитесь, что БД выбрана правильно.

7. Щелкните кнопку ОК — в окне Server Explorer появится новое соединение.

Создание и настройка объекта *DataCommand*

1. Добавьте на панель Toolbox компоненты **OleDbCotnmand** и **OleDbConnection**, для этого щелкните правой кнопкой мыши в разделе Data панели Toolbox и выберите команду **Choose Items**…(**Выбрать элементы**). В списке компонентов NET отметьте нужные компоненты и нажмите OK.

2. Перетащите с вкладки **Data** панели Toolbox на форму экземпляр класса **OleDbConnection** — к приложению добавится новый объект с именем **oleDbConnection1.** В свойстве ConnectionString этого объекта укажите в выпадающем списке файл «Конфетная фабрика».

3. Перетащите с вкладки **Data** панели Toolbox на форму экземпляр класса **OleDbCommand** — к приложению добавится новый объект **OleDbCotnmand** с именем **oleDbCommand1.**

4. Присвойте свойству **Connection** объекта **oleDbCommand1** значение oleDbConnectionl, выбрав его в списке Existing (существующие). Для свойства **CommandText** укажите `SELECT * FROM Заказчики`.

5. Перетащите с вкладки Windows Forms панели Toolbox на форму элементы управления **Button** (в верхнюю часть) и **ListBox** (в середину). Для **Button1** задайте свойство **button1.Text** «Click to Execute DataReader». Увеличьте ширину кнопки для оптимального размещения надписи. Для ЭУ **ListBox** свойство **Dock** установите **Bottom**.

6. В окне дизайнера дважды щелкните объект **button1**, чтобы создать обработчик по умолчанию для события **button1.Click** и открыть его в редакторе кода. Добавьте в обработчик следующий код, извлекающий объект **DataReader** и заполняющий элемент управления ListBox:

```
System.Data.OleDb.OleDbDataReader myReader;
string CustomerString;
oleDbConnection1.Open();
myReader = oleDbCommand1.ExecuteReader();
while (myReader.Read())
{
// Извлечь список имен и фамилий из таблицы
// Заказчики и выполнить их контактенацию.
 CustomerString = myReader[1].ToString() + " " +
      myReader[2].ToString();
// Добавить результат в список ListBox,
      listBox1.Items.Add(CustomerString);
}
      myReader.Close();
oleDbConnection1.Close();
```

7. Сохраните и протестируйте приложение. По щелчку кнопки элемент управления **ListBox** должен заполниться именами и фамилиями заказчиков, взятыми из таблицы **Заказчики**.

Объекты **DataReader** быстро извлекают данные, но не позволяют модифицировать содержимое БД.

Упражнение 2. Извлечение и обновление данных с помощью объектов DataAdapter и DataSet

В этом упражнении вы реализуете доступ к данным для чтения и записи с помощью объектов **DataAdapter** и **DataSet**.

Вы должны заполнить объект **DataSet** *с* помощью объектов **DataAdapter**, связать **DataSet** с элементом управления **DataGrid** и обновить БД модифицированными данными.

Реализация доступа к БД для чтения и записи

1. Перетащите с вкладки Windows Forms панели Toolbox на поверхность формы элемент управления **Button** и установите его свойство **Text** в «Click here for Exercise 2». Увеличьте ширину кнопки для оптимального размещения надписи.

2. В окне дизайнера дважды щелкните элемент управления **button2** чтобы создать обработчик по умолчанию для события **Button2.Click**, и добавьте к нему следующий код:

```
Form2 Exercise2 = new Form2();
Exercise2.Show();
```

3. В меню **Project** выберите команду **Add New Windows Form** и щелкните **ADD**, чтобы добавить новую форму.

4. В этой версии Visual Studio объекты **DataAdapter** были удалены из Toolbox, так что добавьте **OleDbDataAdapter** обратно в Toolbox (см. предыдущее упражнение).

5. Перетащите объект **OleDbDataAdapter** на форму **Form2**, чтобы запустить Data Adapter Configuration Wizard (Мастер настройки адаптера данных).

6. На странице Choose Your Data Connection (Выбор подключения баз данных) выберите подключение к базе данных Конфетная фабрика (или создайте новое подключение в случае необходимости).

7. На странице Choose a Command Type (Выбор типа команды) оставьте настройку по умолчанию Use SQL Statements и щелкните Next.

8. На странице Generate the SQL statements (Создание инструкций SQL) введите следующее предложение SQL: SELECT * FROM Заказчики.

9. Щелкните Finish для завершения мастера и добавления экземпляра настроенного **OleDbDataAdapter** к форме.

10. Генерируйте строго типизированный **DataSet**, основанный на настроенном адаптере, для чего выберите **Generate Dataset** (Создать набор данных) в меню Data.

11. Обратите внимание, что в диалоговом окне выбран **oleDbDataAdapter1**. Он содержит сведения, которые будут использоваться для генерирования строго типизированного **DataSet**.

12. Щелкните OK для создания нового DataSet и добавления его к проекту. Обратите внимание на имя нового объекта – **dataSet11**.

13. Перетащите с вкладки Windows Forms панели Toolbox на форму **Form2** две кнопки и **DataGridView**. Установите для свойств, перечисленных в таблице, указанные в таблице значения:

Объект	Свойство	Значение
button1	Text	Get Data
button2	Text	Update Data
DataGridView	DataSource	dataSet11
	Dock	Bottom
	DataMember	Заказчики

14. В окне конструктора дважды щелкните элемент управления **button1**, чтобы вызвать редактор кода с обработчиком по умолчанию для события **button1.Click**. Добавьте к нему следующий код:

```
oleDbDataAdapter1.Fill(dataSet11.Заказчики);
```

15. Аналогичным образом реализуйте обработчик события **button2.Click,** добавив следующий код:

```
oleDbDataAdapter1.Update(dataSet11);
```

16. Сохраните и протестируйте приложение. Открыв первую форму приложения, щелкните кнопку с надписью «Click here for Exercise 2», чтобы открыть созданную в этом форму. По щелчку кнопки с надписью «Get Data» в элемент управления **DataGrid** будут загружены данные.

Упражнение 3. Использование объектов DataView

Выполнив это упражнение, вы научитесь использовать объекты **DataView** для сортировки и фильтрации данных.

1. Увеличьте размер формы **Form2**: свойству **Size** установите значения (470;300).
2. Добавьте на форму два элемента **label** и свойству **Text** задайте значения *Сортировка* и *Фильтрация* соответственно.
3. Рядом с соответствующими элементами **label** расположите два элемента **TextBox**.
4. Для первого элемента установите значения свойств:

Объект	Name	Text
textBox1	SortTextBox	Фамилия
textBox2	FilterTextBox	Город = 'Пушкин'

5. Создайте новый источник данных, выбрав Add New Data Source в меню Data.
6. Выберите Database и щелкните Next.
7. Выберите допустимое подключение к базе данных Конфетная фабрика.
8. Выбирайте значения по умолчанию, пока не появится страница Choose Your Database Objects.
9. Выберите таблицу *Заказчики* и щелкните Finish.
10. Постройте проект.
11. Найдите в Toolbox компоненты **ЗаказчикиTableAdapter** и **Конфетная_фабрикаDataSet** и перетащите их на форму.
12. Создайте объект DataView для таблицы *Заказчики*, указав перед обработчиком события button1_Click следующий код:

```
DataView ЗаказчикиDataView;
```

13. Замените существующий код в обработчике события button1_Click на следующий:

```
    // Загрузка таблицы данными:
заказчикиTableAdapter1.Fill(конфетная_фабрикаDataSet1.Заказчики);
        // Настройка объекта DataView
        ЗаказчикиDataView = new
  DataView(конфетная_фабрикаDataSet1.Заказчики);
        // Настройка dataGridView для отображения данных
                dataGridView1.DataSource = ЗаказчикиDataView;
        // Присвоения исходного порядка сортировки
                ЗаказчикиDataView.Sort = "Фамилия";
```

14. Постройте и выполните приложение. По нажатию кнопки «Get Data» должны загрузиться данные, отсортированные по столбцу Фамилия.

15. Расположите на форме кнопку с текстом «Сортировка и фильтрация», и добавьте следующий код к обработчику события щелчка кнопки:

```
        ЗаказчикиDataView.Sort = SortTextBox.Text;
          ЗаказчикиDataView.RowFilter = FilterTextBox.Text;
```

16. Постройте и выполните приложение. Загрузите данные и отфильтруйте их. Просмотрите результаты.

17. Измените название города на ‘Санкт-Петербург’ и нажмите кнопку «Сортировка и фильтрация».

Упражнение 4. Связывание данных с элементами управления

Связывание элементов управления с данными является просто описанием процесса отображения данных (таких как данные из базы данных) в элементах управления Windows Forms.

Простое связывание данных описывает отображение одного элемента данных в элементе управления, например отображение в **TextBox** значения одного столбца таблицы, типа названия компании.

В этом упражнении Вы создадите приложение Windows и выполните простое связывание с данными элементов управления (связывание данных настраивается в коде).

1. Создайте приложение Windows и назовите его WinDataBinding.

Реализация доступа к БД

2. В окне **Data Sources** (Источник данных) щелкните **Add New Data Source** (создает новый типизированный набор данных, выполняя Data Source Configuration Wizard).

3. Оставьте выбранный по умолчанию Database (на странице Choose a Data Source Type) и щелкните Next.

4. На странице **Choose Your Data Connection** создайте подключение к базе данных Конфетная фабрика.

5. Щелкайте Next сохраняя значения по умолчанию, пока вы не дойдете до страницы **Choose Your Database Objects** (Выбор объектов базы данных), где выберите *Сотрудники* в узле *Tables.*

6. Щелкните **Finish** для добавления набора данных к вашему проекту.

7. Соберите проект.

8. Перетащите объекты **Конфетная_фабрикаDataSet** и **СотрудникиTableAdapter** из Toolbox на форму.

Создание интерфейса для просмотра данных

Теперь, когда имеется набор данных (и **TableAdapter** для его заполнения), создайте несколько элементов управления для отображения данных набора данных*)* и перемещения по этим данным:

10. Добавьте три элемента управления **TextBox** к форме и присвойте их свойствам **Name** значение **FamtextBox**, **NametextBox** и **SectiontextBox** соответственно.

11.Слева от каждого элемента **TextBox** добавьте элемент **label** и укажите свойству **Text** значения: Фамилия, Имя, Отдел соответственно.

12. Добавьте две кнопки для перемещения по записям.

13. Для первой кнопки установите следующие свойства:

свойству **Name** значение *Previousbutton,* свойству **Text** значение *Previous.*

13. Для второй кнопки также установите свойства

Name – *Nextbutton,* **Text** – *Next.*

Настройка связывания данных

14.Дважды щелкните пустую область на форме, и создайте обработчик события Form1_Load.

15.Перед обработчиком Form1_Load объявите **BindingSource** для таблицы Сотрудники:

```
private BindingSource sotrBindingSourse;
```

16. Добавьте код к обработчику события Form1_Load для настройки связывания данных:

```
// Загрузка таблицы данными:
сотрудникиTableAdapter1.Fill(конфетная_фабрикаDataSet1.Сотрудники);
// Создание BindingSource для таблицы Сотрудники:
 sotrBindingSourse = new
BindingSource(конфетная_фабрикаDataSet1, "Сотрудники");
// Настройка связывания для элементов TextBox:
FamtextBox.DataBindings.Add("Text", sotrBindingSourse,
"Фамилия_сотрудника");
                NametextBox.DataBindings.Add("Text",
sotrBindingSourse, "Имя_сотрудника");
                SectiontextBox.DataBindings.Add("Text",
sotrBindingSourse, "Отдел");
```

17.Дважды щелкните кнопку *Previous* и добавьте код, который перемещает к предыдущей записи в источнике данных *BindingSource:*

```
sotrBindingSourse.MovePrevious();
```

18.Дважды щелкните кнопку *Next* и добавьте код, который перемещает к следующей записи в источнике данных *BindingSource:*

```
sotrBindingSourse.MoveNext();
```

19. Постройте и запустите приложение. Протестируйте его работу.

Упражнение 5. Создание связанной с данными формы в мастере источников данных

Мастер источников данных (Data Source Configuration Wizard) создает в приложении типизированный **DataSet** и заполняет окно Data Sources объектами, выбранными во время работы мастера. После выполнения мастера остается еще переместить элементы в вашу форму для создания экземпляров объектов, которым необходим доступ к данным.

В этом упражнении Вы создадите приложение Windows, создадите источник данных и свяжите элементы управления с данными, перетаскивая элементы из окна Data Sources.

1. Создайте приложение Windows и назовите его WinDataSourcesWizard.
2. Запустите Data Source Configuration Wizard, выбрав **Add New Data Source** в меню **Data**.
3. На странице Choose a Data Source Type оставьте выбранный по умолчанию Database и щелкните Next.
4. На странице Choose Your Data Connection выберите подключение к базе данных Конфетная фабрика и создайте, если нужно, новое подключение.
5. Щелкайте Next, сохраняя значения по умолчанию, пока не дойдете до страницы Choose Your Database Objects (Выбор объектов базы данных), и в узле **Tables** выберите таблицы *Сотрудники* и *Заказы*.
6. Щелкните Finish для добавления набора данных к проекту.
7. В меню **Data** выберите **Show Data Sources**, чтобы отобразить окно **Data Sources**, проверьте, что отображаются требуемые таблицы.
8. Перетащите узел *Сотрудники* из окна Data Sources на форму **Form1.**
9. К форме добавятся **DataGridView** и **BindingNavigator**, и в области компонентов появится несколько относящихся к данным объектов.
10. Постройте приложение.
11. В этот момент вы имеете рабочее приложение с **DataGridView**, связанным с данными таблицы *Сотрудники.* Если в интегрированной среде разработки вы переключитесь в режим кода, то увидите, что был добавлен код к событию загрузки формы для заполнения таблицы *Сотрудники* данными, а в **BindingNavigator** — к сохраняющей данные кнопке для отправки обновлений обратно в базу данных.
12. Запустите приложение. Выполняющееся приложение должно отобразить данные таблицы *Сотрудники*.
13. Остановите приложение и откройте форму в режиме Design.
14. Разверните узел *Сотрудники* в окне Data Sources.
15. Перетащите на свободное место формы узел *Заказы*, вложенный в узел *Сотрудники.*
16. Обратите внимание на **BindingSource** и **TableAdapter**, добавленные в область компонентов.

17. Постройте и выполните приложение. Щелкните строку в таблице *Сотрудники.* Обратите внимание, что **ЗаказыDataGridView** отображает все заказы, оформленные данным сотрудником.

Список литературы

1. *Эндрю Троелсен*. Язык программирования C# 2010 и платформа .NET 4.0. – М.,Вильямс, 2010 г. .– 1392 с.

2. *Голощапов А*. Microsoft Visual Studio 2010 (+ CD-ROM).–БХВ-Петербург, 2011.– 544с.

3. *Ник Рендольф, Дэвид Гарднер, Майкл Минутилло, Крис Андерсон* Visual Studio 2010 для профессионалов.– М.,Диалектика, 2011. – 1184 с.

4. *Алекс Макки*. Введение в .NET 4.0 и Visual Studio 2010 для профессионалов. – Издательство: Вильямс - 2010 – 416с.

5. *Джо Майо*. Microsoft Visual Studio 2010. Самоучитель – БХВ – Петербург,-2010- 450 с.

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

КАФЕДРА ПРОГРАММНЫХ СИСТЕМ

Кафедра **Программных систем** входит в состав нового факультета **Инфокоммуникационные технологии**, созданного решением Ученого совета университета 17 декабря 2010 г. по предложению инициативной группы сотрудников, имеющих большой опыт в реализации инфокоммуникационных проектов федерального и регионального значения.

На кафедре ведется подготовка бакалавров и магистров по направлению **210700 «Инфокоммуникационные технологии и системы связи»:**

210700.62.10 – ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ (Бакалавр)

210700.68.10 – ИНТЕЛЛЕКТУАЛЬНЫЕ ИНФОКОММУНИКАЦИОННЫЕ СИСТЕМЫ (Магистр)

Выпускники кафедры получают фундаментальную подготовку по: математике, физике, электронике, моделированию и проектированию инфокоммуникационных систем (ИКС), информатике и программированию, теории связи и теории информации.

В рамках профессионального цикла изучаются дисциплины: архитектура ИКС, технологии программирования, ИКС в Интернете, сетевые технологии, администрирование сетей Windows и UNIX, создание программного обеспечения ИКС, Web программирование, создание клиент-серверных приложений.

Область профессиональной деятельности бакалавров и магистров включает:

- сервисно-эксплуационная в сфере современных ИКС;

- расчетно-проектная при создании и поддержке сетевых услуг и сервисов;
- экспериментально-исследовательская;
- организационно-управленческая – в сфере информационного менеджмента ИКС.

Знания выпускников востребованы:

- в технических и программных системах;
- в системах и устройствах звукового вещания, электроакустики, речевой, и мультимедийной информатики;
- в средствах и методах защиты информации;
- в методах проектирования и моделирования сложных систем;
- в вопросах передачи и распределения информации в телекоммуникационных системах и сетях;
- в методах управления телекоммуникационными сетями и системами;
- в вопросах создания программного обеспечения ИКС.

Выпускники кафедры Программных систем обладают компетенциями:

- проектировщика и разработчика структур ИКС;
- специалиста по моделированию процессов сложных систем;
- разработчика алгоритмов решения задач ИКС;
- специалиста по безопасности жизнедеятельности ИКС;
- разработчика сетевых услуг и сервисов в ИКС;
- администратора сетей: UNIX и Windows;
- разработчика клиентских и клиент-серверных приложений;
- разработчика Web – приложений;
- специалиста по информационному менеджменту;
- менеджера проектов планирования развития ИКС.

Трудоустройство выпускников:

1. ОАО «Петербургская телефонная сеть»;
2. АО «ЛЕНГИПРОТРАНС»;
3. Акционерный коммерческий Сберегательный банк Российской Федерации;
4. ОАО «РИВЦ-Пулково»;
5. СПБ ГУП «Петербургский метрополитен»;
6. ООО «СоюзБалтКомплект»;
7. ООО «ОТИС Лифт»;
8. ОАО «Новые Информационные Технологии в Авиации»;
9. ООО «Т-Системс СиАйЭс» и др.

Кафедра сегодня имеет в своем составе высококвалифицированный преподавательский состав, в том числе:

- 5 кандидатов технических наук, имеющих ученые звания профессора и доцента;

- 4 старших преподавателя;
- 6 штатных совместителей, в том числе кандидатов наук, профессиональных IT - специалистов;
- 15 Сертифицированных тренеров, имеющих Западные Сертификаты фирм: Microsoft, Oracle, Cisco, Novell.

Современная техническая база; лицензионное программное обеспечение; специализированные лаборатории, оснащенные необходимым оборудованием и ПО;качественная методическая поддержка образовательных программ; широкие Партнерские связи существенно влияют на конкурентные преимущества подготовки специалистов.

Авторитет специализаций кафедры в области компьютерных технологий подтверждается Сертификатами на право проведения обучения по методикам ведущих Западных фирм - поставщиков аппаратного и программного обеспечения.

Заслуженной популярностью пользуются специализации кафедры ПС по подготовке и переподготовке профессиональных компьютерных специалистов с выдачей **Государственного Диплома** о профессиональной переподготовке по направлениям: **"Информационные технологии (инженер-программист)"** и **"Системный инженер"**, а также Диплома о дополнительном (к высшему) образованию с присвоением квалификации: **"Разработчик профессионально-ориентированных компьютерных технологий "**. В рамках этих специализаций высокопрофессиональные преподаватели готовят компетентных компьютерных специалистов по современным в России и за рубежом операционным системам, базам данных и языкам программирования ведущих фирм: Microsoft, Cisco, IBM, Intel, Oracle, Novell и др.

Профессионализм, компетентность, опыт, и качество программ подготовки и переподготовки IT- специалистов на кафедре ПС неоднократно были удостоены **высокими наградами «Компьютерная Элита» в номинации лучший учебный центр России**.

Партнеры:

1. **Microsof**t Certified Learning Solutions;
2. **Novell** Authorized Education Center;
3. **Cisco** Networking Academy;
4. **Oracle** Academy;
5. **Sun Java** Academy и др;
6. **Prometric**;
7. **VUE**.

Мы готовим квалифицированных инженеров в области инфокоммуникационных технологий с новыми знаниями, образом мышления и способностями быстрой адаптации к современным условиям труда.

Никита Алексеевич Осипов

Разработка Windows приложений на C#

УЧЕБНОЕ ПОСОБИЕ

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики
и оптики
197101, Санкт-Петербург, Кронверкский пр., 49