Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут"
Інститут телекомунікаційних систем

Системи розподіленої обробки інформації в телекомунікаційних мережах

Методичні вказівки
до виконання лабораторних робіт
для студентів спеціальності "Інформаційні мережі зв'язку"

Затверджено Вченою радою ІТС НТУУ «КПІ»

Київ
НТУУ "КПІ"
2013

Системи розподіленої обробки інформації в телекомунікаційних мережах [Текст]: метод. вказівки до викон. лаборатор. робіт для студентів спеціальності "Інформаційні мережі зв'язку" / Уклад.: Л.С.Глоба. – К.: НТУУ "КПІ", 2013. – 60 с.

Гриф надано Вченою радою ІТС НТУУ «КПІ»
(Протокол № 7 від 29.08.2013)

Навчально-методичне видання

Системи розподіленої обробки інформації в телекомунікаційних мережах

Методичні вказівки
до виконання лабораторних робіт
для студентів спеціальності: 8.05090301 "Інформаційні мережі зв'язку"

Укладачі: *Глоба Лариса Сергіївна*, д-р техн. наук, проф.

Відповідальний редактор: *А.В. Доровських*, д-р техн. наук, проф.

Рецензенти: *С.О. Кравчук*, д-р техн. наук
М.Ю. Терновой, канд. техн. наук.

Зміст

Вступ

Сучасні вимоги до інженерів з телекомунікацій ґрунтуються на глибоких науково-технічних знаннях та вміннях розв’язувати практичні задачі, в тому числі задачі щодо проектування корпоративних інформаційно-телекомунікаційних систем та мереж, а також створення інформаційних та обчислювальних ресурсів в телекомунікаційному середовищі.

Метою методичних вказівок є надання допомоги студентам в отриманні практичних навичок роботи з системами розподіленої обробки інформації в телекомунікаційних мережах. Виконання лабораторних робіт забезпечить закріплення теоретичного та лекційного матеріалу.

Під час виконання лабораторних робіт студенти отримають навички та познайомляться з:

- платформою Microsoft.Net;
- Стеком технологій WCF;
- Стеком технологій WPF;
- розробкою прикладного програмного забезпечення з використанням можливостей WCF/WPF;
- розробкою Web-частин;
- роботою з інструментальними панелями Expression Blend;
- налаштування служб Excel.

.

Загальні методичні вказівки

1. У кожній лабораторній роботі визначено: мету роботи, рекомендації з підготовки до роботи, програму і порядок її виконання.

2. Напередодні кожної лабораторної роботи необхідно:

- вивчити теоретичний матеріал до лабораторної роботи
- усвідомити мету, зміст і порядок виконання;
- у лабораторії перевірити наявне лабораторне устаткування.

3. До виконання лабораторної роботи допускаються тільки підготовлені студенти після тестування (усного чи письмового), що проводиться викладачем перед початком виконання роботи.

Лабораторні роботи виконуються самостійно кожним студентом.

Звіт з роботи акуратно оформлюється і подається під час захисту лабораторної роботи. Оформлення звіту має відповідати вимогам стандартів.

4. Студенти, відсутні на заняттях, виконують роботу у час, погоджений з викладачем і інженером лабораторії після тестування.

5. Перед початком робіт кожному студенту необхідно вивчити правила техніки безпеки, здати залік, за що розписатися в журналі.

Вимоги до оформлення звіту з лабораторної роботи

Звіт оформлюється на аркушах формату А4 і повинен містити:

1. Титульну сторінку з номером та назвою лабораторної роботи, прізвищем студента, номером групи та номером варіанта (якщо це необхідно).
2. Особливості завдання згідно варіанта роботи (якщо це необхідно).
3. Опис виконання завдання.
4. Висновки по роботі.

Інструкція з техніки безпеки

1. Вимоги з техніки безпеки перед початком роботи
   1.1 Провести огляд з зовні електророзеток, шнурів, вилок підключення до мережі живленя та заземлення (занулення).
   1.2 Забороняється працювати на несправному устаткуванні.
   1.3 За необхідності отримати додаткове устаткування у викладача та перевірити його справність.

2. Вимоги з техніки безпеки під час виконання робіт
   2.1 Необхідно виконувати лише ту роботу, з якої був проведений інструктаж, забороняється передоручати свою роботу іншим особам.
   2.2 Забороняється:
   - експлуатація кабелів та проводів з пошкодженною ізоляцією або такою, що втратила захисні властивості за час експлуатації;
   - залишати під напругою кабелі та проводи з неізольованими провідниками;
   - застосовувати саморобні подовжувачі, що не відповідають вимогам ПВЕ для переносних електропровідників;
   - користуватися пошкодженими розетками, розгалужувальними та з'єднувальними коробками, вимикачами та іншими електровиробами, а також лампами, скло яких має слід затемнення або випинання;
   - використовувати електроустаткування та прилади в умовах, що не відповідають інструкції з експлуатації підприємств-виробників;
   - залишати пристрої, що працюють без нагляду на тривалий час;
   - переносити пристрої, що підключені до електромережі;
   - забороняється самостійно ремонтувати апаратуру;
   - класти будь-які предмети на апаратуру комп'ютера, напої на клавіатуру або поруч з нею - це може вивести їх з ладу.

3. Вимоги з техніки безпеки після закінчення роботи
   3.1. Зберегти необхідні файли на жорсткий диск комп'ютера або на переносний носій.
   3.2 Виключити комп'ютер.
   3.3 Виключити додаткове устаткування та віддати його викладачу.
   3.4 Прибрати робоче місце.

Лабораторна робота №1

Тема: "Створення WPF програм"

Мета: отримати теоретичні знання стосовно платформи Windows Presentation Foundation (WPF), отримати навички створення WPF програм.

Склад робочого місця

1. Обладнання: IBM-сумісний персональный компьютер (ПК).
2. Програмне забезпечення:
   - операційна система Windows;
   - Visual Studio 2008 з встановленим компонентом Visual C#.

Теоретична частина

Windows Presentation Foundation (WPF) надає уніфіковану програмну модель для створення ємних та інтелектуальних програм Windows, що об'єднують інтерфейс користувача, мультимедіа і документи.

Xaml –основний елемент WPF, а тому, щоб почати створення простих програм на WPF необхідно оволодіти елементарними принципами роботи з Xaml. Для прикладу створемо нову WPF-програму. Для цього перейдіть у Visual Studio Ctrl-Shift-N, або в меню File перейдіть в підпункт меню New, натисніть New Project.

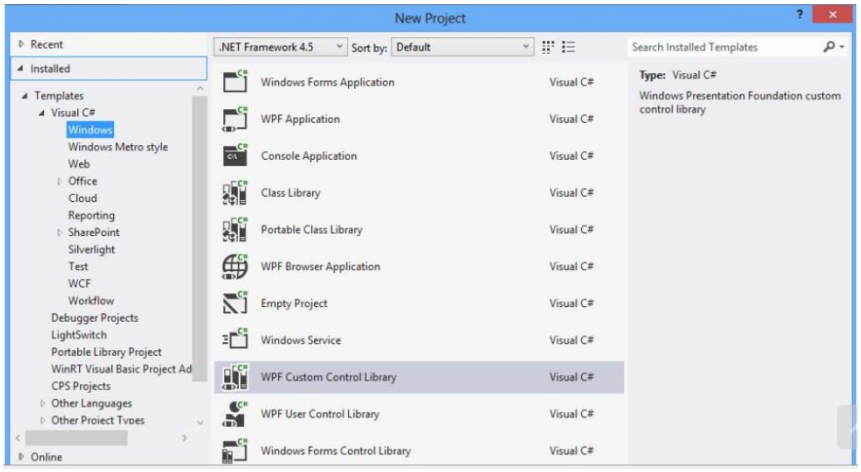


Рис. 1. Створення нової WPF-програми.

Тут необхідно обрати WPF Application, знизу ввести директорію для проекту програми, а також назву.

VS створить файл логіки, де знаходиться код (*.cs), і файл, що описує дизайн форми (*.xaml). Якщо перейти в xaml файл, то можна побачити пусте вікно, знизу вікна має бути Xaml размітка, тип, наведеної нижче:

```
<Window x:Class="WorkWithXaml.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="350" Width="525">
    <Grid>
    </Grid>
</Window>
```

Форма поки не містить елементів, і тому Xaml розмітка невелика.

Код містить кореневий об’єкт Window, в Xaml розмітці однієї форми він може бути представленим лише в одному екземплярі. Об’єкт Window може містити лише один об’єкт, наприклад контейнер компоновки (Grid – один з видів таких контейнерів). Контейнери компоновки в свою чергу містять інші компоненти.

Розмістивши будь-яких елемент, перетягнувши його з панелі інструментів у Visual Studio на форму, дизайнер генерує чітку позицію компонента, що задається в параметрах Left, Top. Але у WPF не має бути чітких координат об’єктів, проте кожний елемент має знаходитись у відповідному контейнері. Така потреба дає можливість позиціонувати елементи при змінній формі вікна програми. Всього контейнерів компоновки 5: Grid, StackPanel, WrapPanel, Canvas, DockPanel.

Grid- найбільш поширений контейнер, дозволяє швидко розміщувати елементи у рядках і стовбцях.

```
<Grid>
    <Grid.RowDefinitions>
        <RowDefinition Height="*"></RowDefinition>
        <RowDefinition Height="auto"></RowDefinition>
        <RowDefinition Height="100"></RowDefinition>
    </Grid.RowDefinitions>
</Grid>
```

У Grid елемента для визначення рядків є властивість RowDefinitions, де задають висоту рядка. В Height можна вказати числове значення, відносне і автоматичне «auto». Символ * означає, що рядок буде розміщено у всьому вільному просторі. Auto використовується для того, щоб виділити стільки місця,скільки потрібно для розміщення всіх елементів, розташованих всередині рядка.

Щоб розмістити компоненти в тому чи іншому рядку необхідно використати властивість Grid.Row, що є у кожного елемента.

Для прикладу, після </Grid.RowDefinitions> допишіть:

```
<TextBox Grid.Row="0">textBox</TextBox>
```

<Button Grid.Row="1">Button</Button>
<Label Background="Aqua" Grid.Row="2"></Label>

На рис.2. видно результат.

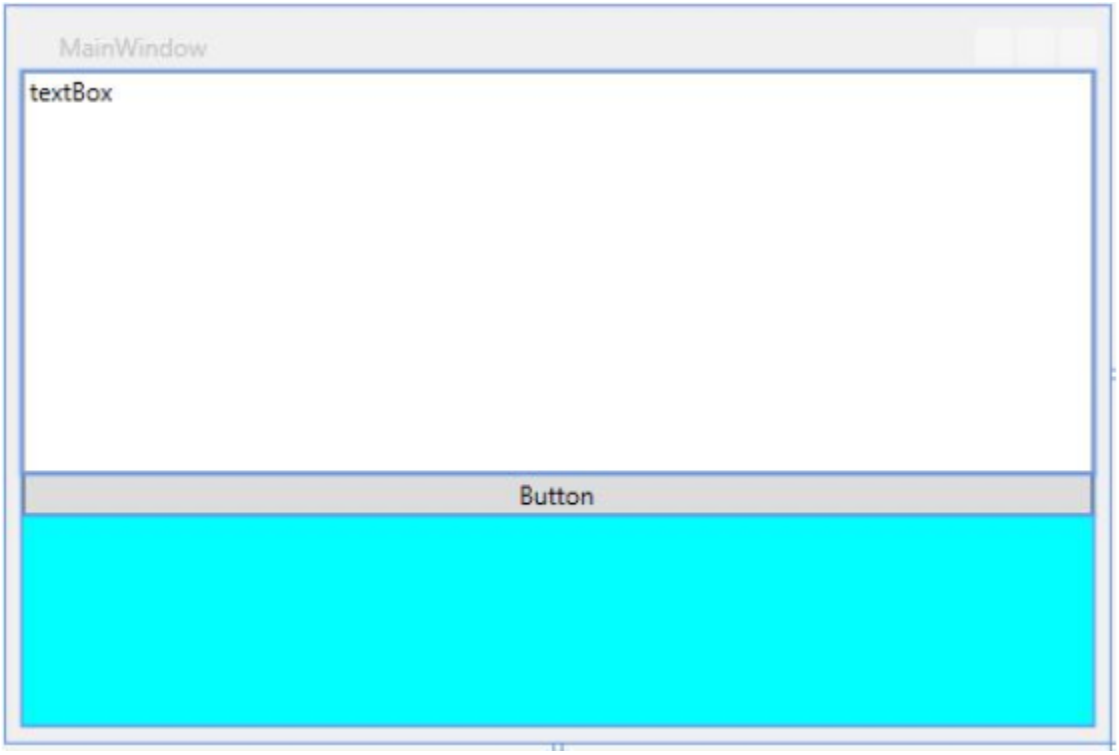


Рис. 2. Grid і його Rows.

Grid містить також і стовбці

```
<Grid>
<Grid.RowDefinitions>
        <RowDefinition Height="*"></RowDefinition>
        <RowDefinition Height="auto"></RowDefinition>
        <RowDefinition Height="100"></RowDefinition>
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
    <TextBox Grid.Row="0">textBox</TextBox>
    <Button Grid.Row="0" Grid.Column="1">Button</Button>
    <Label Background="Aqua" Grid.Row="2"></Label>
    <Ellipse Fill="Bisque" Grid.Row="1" Grid.Column="1" Height="50"></Ellipse>
 </Grid>
```

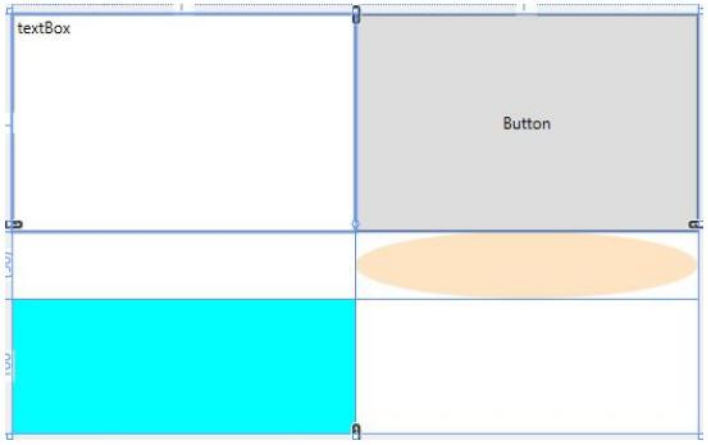


Рис. 3. Grid і його стовбці.

Якщо не вказано, до якого рядка чи стовпця прикріплено елемент, то він розміщується у нульовому рядку і нульовому стовпці.

StackPanel. Даний контейнер використовують коли необхідно розмістити елементи послідовно вертикально, або горизонтально. Цей компонент містить властивість Orientation, що має значення Vertical за замовчуванням, але може бути зміненим на Horizontal. Даний контейнер може бути використаним наприклад для формування головного меню програми:

```
<Grid>
    <StackPanel>
      <TextBox>textbox</TextBox>
      <Button>Button</Button>
      <Label>Label</Label>
      <Ellipse Height="50" Fill="Bisque"></Ellipse>
    </StackPanel>
  </Grid>
```

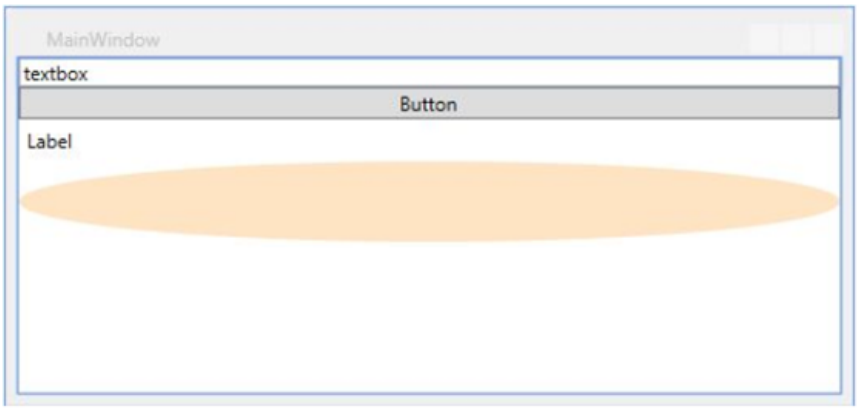


Рис. 4 StackPanel із вертикальною орієнтацією.

Змінивши Orientation на Horizontal, результат змінюється.

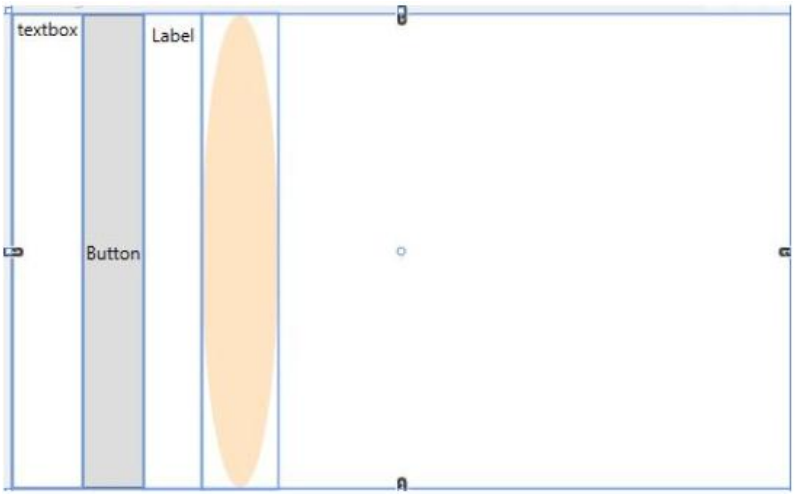


Рис. 5. StackPanel із горизонтальною орієнтацією.

WrapPanel. Даний контейнер відрізняється тим, що може переносити елементи на новий рядок, в залежності від ширини доступного простору. Подібно до StackPanel тут присутня властивість Orientation. Коли користувач зменшує розміри форми, то компоненти всередині WrapPanel будуть переноситися на новий рядок, реорганізовуючись відповідно до нових розмірів форми.

```
<Grid>
    <WrapPanel>
        <TextBlock>text</TextBlock>
        <Button>button</Button>
        <TextBox MinWidth="200">textbox</TextBox>
        <Ellipse Fill="Orange" Height="50" Width="30"></Ellipse>
        <ComboBox MinWidth="100"></ComboBox>
    </WrapPanel>
</Grid>
```

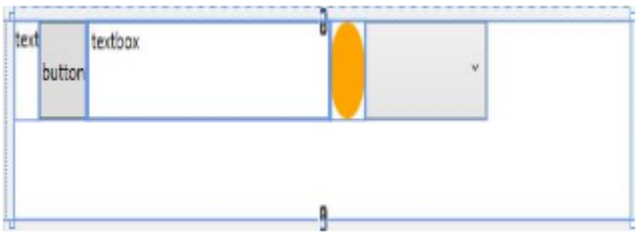


Рис. 6 WrapPanel.

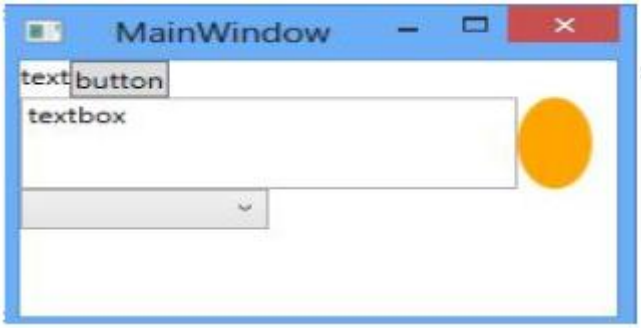


Рис 7. WrapPanel при зміні розмірів форми.

DockPanel дає можливість розмістити компонент вздовж однієї із сторін своєї поверхні. У елементів, що розміщено всередині DockPacel зявляється властивість DockPanel.Dock, яке може бути встановлено у значення left, right, top, bottom.

```
<Grid>
    <DockPanel LastChildFill="False">
      <Button DockPanel.Dock="Right" Height="auto" Width="auto">x</Button>
      <Button DockPanel.Dock="Left" Width="20"></Button>
    </DockPanel>
  </Grid>
```

На рис.8. зображено результат xaml розмітки.

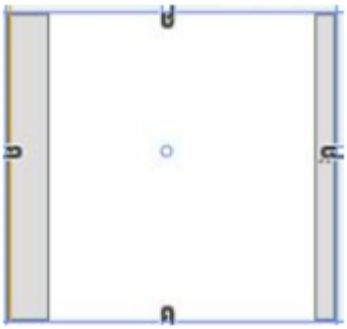


Рис. 8 DockPanel.

Canvas. Кожний елемент, що розміщено у цьому контейнері має властивості Canvas.Top, Canvas.Bottom, Canvas.Left і Canvas.Right. Кожна із властивостей дає можливість встановити абсолютні координати, які не будуть змінюватись навіть при зміні розмірів форми.

```
<Grid>
    <Canvas Background="Aqua">
      <TextBox Canvas.Top="0">text</TextBox>
      <Button Canvas.Top="20">button</Button>
    </Canvas>
```

</Grid>

На рис.9. зображено дану форму.

Рис. 9 Canvas.

GridView. Режим відобраяження GridView є одним із режимів для елементу управління ListView. Класс GridView використовують для відображення колекцій елементів у таблицях. В таких таблицях найчастіше використовуються кнопки-заголовки для стовпців.

В режимі відображення GridView список елементів даних транслюється на екран завдяки прив’язки даних бо стовпціві виводу заголовка стовпця для ідентифікації поля. За замовченням у стилі GridView заголовки стовпців реалізуються у вигляді кнопок. Використовуючи кнопки в якості заголовків стовпців, можна реалізувати важливі можливості взаємодії із користувачем. Наприклад, може натиснути на заголовок стовпця для сортування даних GridView у відповідності до вмісту стовпця.

На рис 10 показано GridView.

Name	Time	Artist	Disk
Song1	3:54	Singer1	Disk1
Song2	4:31	Singer2	Disk3
Recommended			
Song3	5:06	Singer3	Disk1
Strongly Recommended			
Song4	4:18	Singer3	Disk2
Song5	6:15	Singer1	Disk3
Strongly Recommended			

Рис. 10 GridView

Стовпці GridView представлені об’єктами GridViewColumn, размір котрих може автоматично змінюватися відповідно до їх вмісту. За необхідності можна явно задавати ширину об’єкта GridViewColumn. Можна змінити розміри стовпців,рухаючи межу між заголовками стовпців. Можна також динамічно додати, видалити, змінити порядок стовпців, так як ця функція належить класу GridView. Однак в класі GridView немає підтримки даних, що безпосередньо відображаються.

Далі показано, як створити об’єкт GridView, де відображаються дані співробітника. Тут в об’єкті ListView задано EmployeeInfoDataSource як властивість ItemsSource.

```
<ListView ItemsSource="{Binding Source=
                {StaticResource EmployeeInfoDataSource}}">
  <ListView.View>
    <GridView AllowsColumnReorder="true"
          ColumnHeaderToolTip="Employee Information">
      <GridViewColumn DisplayMemberBinding=
                  "{Binding Path=FirstName}"
                Header="First Name" Width="100"/>
              <GridViewColumn DisplayMemberBinding=
                  "{Binding Path=LastName}"
                Width="100">
                <GridViewColumnHeader>Last Name
                  <GridViewColumnHeader.ContextMenu>
                  <ContextMenu  MenuItem.Click="LastNameCM_Click"
                          Name="LastNameCM">
                    <MenuItem Header="Ascending" />
                    <MenuItem Header="Descending" />
                  </ContextMenu>
                  </GridViewColumnHeader.ContextMenu>
                </GridViewColumnHeader>
              </GridViewColumn>
              <GridViewColumn DisplayMemberBinding=
                  "{Binding Path=EmployeeNumber}"
                Header="Employee No." Width="100"/>
    </GridView>
  </ListView.View>
</ListView>
```

На рис 11 показано таблицю, котру створено.

Об’єкт GridView, де відображено дані із ItemsSource

First Name	Last Name	Employee No.
Jesper	Aaberg	12345
Dominik	Paiha	98765
Yale	Li	23875
Muru	Subramani	49392

Рис.11 GridView и дані

ObservebleCollection.

У багатьох випадках, дані, з якими працює користувач, є колекцією об'єктів. Наприклад, поширеним сценарієм у прив'язці даних є використання об'єктів ItemsControl, таких як ListBox, ListView або TreeView для відображення колекції записів.

Користувач може виконати перерахування елементів будь-якої колекції, що реалізує інтерфейс IEnumerable . Однак, щоб налаштувати динамічні прив'язки таким чином, щоб вставки і видалення елементів в колекції автоматично оновлювали UI, в колекції має бути реалізований інтерфейс INotifyCollectionChanged. Цей інтерфейс надає подія CollectionChanged, яке має бути викликано при зміні базової колекції.

WPF надає клас ObservableCollection<T>, який є вбудованою реалізацією колекції даних, що реалізує інтерфейс INotifyCollectionChanged.

До реалізації власної колекції варто розглянути можливість використання класу ObservableCollection<T> або одного з існуючих класів колекцій, таких як List<T>, Collection<T> і BindingList<T> (серед багатьох інших). Якщо є складний сценарій і потрібно реалізувати свою власну колекцію, слід розглянути можливість використання інтерфейсу IList, який надає неуніверсальність колекції об'єктів. До об'єктів цієї колекції можна окремо звертатися за індексом . Реалізація IList забезпечує найкращу продуктивність в роботі з ядром прив'язки даних.

Для повної підтримки передачі значень даних від об'єктів джерела прив'язки до об’єкту прив'язки, кожен об'єкт колекції, який підтримує властивості прив'язки, повинен реалізовувати відповідний механізм повідомлення про зміну властивостей, такий як інтерфейс INotifyPropertyChanged.

ObservableCollection<T> можна використовувати як елемент об'єкта XAML в Windows Presentation Foundation (WPF), у версії 3.0 і 3.5 . Проте використання має суттєві обмеження.

• Колекція ObservableCollection <T> повинна бути кореневим елементом, оскільки атрибут x:TypeArguments, за допомогою якого задається тип обмеження для універсального класу ObservableCollection<T>, підтримується тільки для кореневого елемента об'єкта.

• Необхідно оголосити атрибут x:Class (який вказує на те, що операцією побудови для даного XAML - файлу повинна бути Page або яка-небудь інша операція побудови, яка веде до компіляції XAML).

• Колекція ObservableCollection <T> знаходиться в просторі імен і збірці, які спочатку не поставлена у відповідність простору імен XML за замовчуванням. Необхідно зіставити префікс простору імен і збірки, а потім скористатися префіксом тега об'єкта елемента для колекції ObservableCollection<T>.

Можливостями ObservableCollection<T> для застосування XAML в додатку можна скористатися, оголосивши власний клас колекцій, похідний від ObservableCollection<T>, і

обмеживши його певним типом . Потім слід зіставити збірку, що містить цей клас, і додати посилання на цю збірку в елемент об'єкта в XAML.

Приклад:

```
public class NameList : ObservableCollection<PersonName>
{
   public NameList() : base()
   {
      Add(new PersonName("Willa", "Cather"));
      Add(new PersonName("Isak", "Dinesen"));
      Add(new PersonName("Victor", "Hugo"));
      Add(new PersonName("Jules", "Verne"));
   }
}

public class PersonName
{
   private string firstName;
   private string lastName;

   public PersonName(string first, string last)
   {
      this.firstName = first;
      this.lastName = last;
   }

   public string FirstName
   {
      get { return firstName; }
      set { firstName = value; }
   }

   public string LastName
   {
      get { return lastName; }
      set { lastName = value; }
   }
}
```

Хід роботи

1. У вікні **Visual Studio** в меню **File** оберіть пункт **New Project**.
2. В діалоговому вікні **New Project** розгорніть вузол **Visual C#**, виберіть пункт **WPF**, далі - **WPF Aplication**. Натисніть кнопку **ОК** для відкриття проекту.
3. У коді програми створіть довільний клас MyClass за варіантом.
4. У коді програми ініціалізуйте екземпляр класу ObservableCollection<MyClass>
5. У розмітці програми створіть GridView і зв'яжіть його із створеною колекцією.
6. Продемонструйте роботу інтерфейсу INotifyPropertyChanged
7. Створіть кнопки для додавання і видалення елементів з колекції.

2. Контрольні питання

1. Що таке Wpf?
2. Що таке елементи компоновки?
3. Які елементи компоновки ви знаєте? Поясніть їх особливості.
4. Для чого використовують інтерфейс INotifyPropertyChanged?
5. Для чого використовують клас ObservableCollection<T>?
6. Що таке Binding?

Лабораторна робота №2

Тема: " Створення клієнт-серверної програми з використання технологій wcf/wpf "

Мета: отримати теоретичні знання стосовно платформи Windows Communication Foundation (WCF), отримати навички створення WCF сервісу та його тестування.

Склад робочого місця

1. Обладнання: IBM-сумісний персональный компьютер (ПК).
2. Програмне забезпечення:
   - операційна система Windows;
   - Visual Studio 2008 з встановленим компонентом Visual C#.

Теоретична частина

Windows Communication Foundation (WCF) представляє собою платформу для побудови сервіс-орієнтованих прикладних програм. За допомогою WCF можна відправляти дані у вигляді асинхронних повідомлень від однієї кінцевої точки служби до іншої. Кінцева точка служби може входити в постійно доступну службу, розміщувану в IIS, або представляти службу, розміщувану у прикладній програмі. Кінцева точка може бути клієнтом служби , яка запитує дані від кінцевої точки служби. Повідомлення можуть представляти один символ або одне слово, що відправляється у форматі XML, або мати вигляд складного потоку двійкових даних. Далі представлено кілька зразків сценаріїв:

- Захищена служба для обробки бізнес-транзакцій.
- Служба, що передає іншим об'єктам поточні дані, такі як звіт про трафік, або інша служба спостереження.
- Служба бесід, яка дозволяє двом користувачам спілкуватися і обмінюватися даними в реальному часі.
- Прикладна програма панелі моніторингу, яка опитує одну або кілька служб і дає логічне представлення отриманих даних.
- Надання доступу до робочого процесу, реалізованого за допомогою Windows Workflow Foundation, у вигляді служби WCF.
- Додаток Silverlight для запиту останніх каналів даних у службі.

Такі програми можна було створювати і до появи WCF, однак WCF істотно спрощує розробку кінцевих точок. Таким чином, платформа WCF реалізує керований підхід до створення веб-служб і клієнтів веб-служб.

Можливості WCF

У WCF входить наступний набір можливостей :

- **Сервіс-орієнтованість**

Застосування стандартів WS в WCF дозволяє створювати сервіс-орієнтовані прикладні програми. Сервіс-орієнтована архітектура (SOA) передбачає застосування веб-служб для відправки та отримання даних. Загальною перевагою служб є слабка зв'язаність замість жорсткої запрограмованості для різних прикладних програм . Слабкий зв'язок означає, що будь-який клієнт, створений на будь-якій платформі, може підключатися до будь-якої служби за умови, що виконуються необхідні контракти.

- **Взаємодія**

WCF реалізує сучасні галузеві стандарти для сумісності з веб-службою.

- **Кілька шаблонів повідомлень**

Обмін повідомленнями виконується по одному з декількох шаблонів. Найчастіше використовується шаблон « запит -відповідь», коли одна кінцева точка запитує дані від іншої кінцевої точки, друга кінцева точка відповідає. Існують і інші шаблони , наприклад одностороннє повідомлення, коли одна кінцева точка відправляє повідомлення, не чекаючи відповіді. Більш складним є шаблон дуплексного обміну, коли дві кінцеві точки встановлюють з'єднання і відправляють дані в прямому і зворотному напрямках, подібно до програми обміну миттєвими повідомленнями.

- **Метадані служби**

WCF підтримує публікацію метаданих служби з використанням форматів, зазначених у галузевих стандартах, таких як WSDL, схеми XML і WS- Policy. За допомогою таких метаданих можна автоматично створювати і настроювати клієнти для доступу до служб WCF. Метадані можуть публікуватися через HTTP і HTTPS або з використанням стандарту обміну метаданими веб-служб.

- **Контракти даних**

Оскільки платформа WCF побудована на основі .NET Framework, в неї входять зручно використовувані в коді методи передачі контрактів, які потрібно примусово виконувати. Одним з універсальних типів контрактів є контракт даних. Якщо код служби створюється на мові Visual C # або Visual Basic, то найпростішим способом обробки даних фактично є створення класів, які представляють сутність даних з властивостями, що належать сутності даних. WCF включає складну систему для роботи з даними цим зручним способом. Після створення класів, що представляють дані, служба автоматично створює метадані, які дозволяють клієнтам забезпечувати відповідність заданим типам даних.

- **Безпека**

Повідомлення можна шифрувати для захисту конфіденційності та вимагати від користувачів проходити перевірку автентичності перед прийомом повідомлень. Можна реалізувати широко відомі стандарти безпеки, такі як SSL і WS-SecureConversation.

- **Кілька транспортів і кодувань**

Повідомлення можуть відправлятися по будь-якому з декількох вбудованих транспортних протоколів у різних кодуваннях. Найпоширенішим варіантом є передача повідомлень SOAP в текстовому кодуванні по протоколу HTTP для використання в Інтернеті. Крім того, WCF дозволяє відправляти повідомлення по протоколу TCP, через іменовані канали або MSMQ. Повідомлення можна кодувати у вигляді тексту або використовувати оптимізований двійковий формат. Двійкові дані можна ефективно відправляти з використанням стандарту MTOM. Якщо жоден з наданих транспортів і кодувань не підходить до поточних вимог, можна створити власний користувацький транспорт або кодування.

- **Надійні повідомлення та повідомлення в черзі**

WCF підтримує надійний обмін повідомленнями з використанням надійних сеансів, реалізованих на базі схеми WS-Reliable, і обмін з використанням MSMQ.

- **Стійкі повідомлення**

Стійкі повідомлення не губляться у разі перебоїв зв'язку. Повідомлення, надіслані зі стійкого шаблону, завжди зберігаються в базі даних. Якщо відбувається перебій зв'язку, база даних дозволяє відновити обмін повідомленнями після відновлення з'єднання. Стійкі повідомлення також можна створити за допомогою Windows Workflow Foundation.

- **Транзакції**

WCF також підтримує транзакції, відповідно до однієї з трьох моделей: WS-AtomicTtransactions, API-інтерфейси простору імен System.Transactions і координатор розподілених транзакцій (Майкрософт).

- **Підтримка AJAX і REST**

REST – це приклад розвитку технології Web 2.0. WCF можна налаштувати для обробки «звичайних» XML-даних, що не запаковані в конверт протоколу SOAP. WCF також можна розширити для підтримки певних форматів XML , таких як ATOM (поширений стандарт RSS), і навіть форматів, відмінних від XML, таких як нотація об'єктів JavaScript (JSON).

- **Розширюваність**

Архітектура WCF передбачає ряд точок для розширення. Якщо потрібні додаткові можливості, підтримуються точки входу, за допомогою яких можна налаштувати поведінку служби.

Кінцеві точки: адреси, прив'язки, контракти, набір поведінки

Зв'язок зі службою Windows Communication Foundation (WCF) здійснюється через кінцеві точки служби. Кінцеві точки забезпечують доступ клієнтів до функціональних можливостей служби WCF.

Кожна кінцева точка складається з чотирьох властивостей:

- **адреси (address)**, що показує, де можна знайти кінцеву точку;
- **прив'язки (binding)**, що показує, як клієнт може зв'язатися з кінцевою точкою;
- **контракту (contract)**, який визначає доступні операції;
- **набору поведінки (set of behaviors)**, який задає відомості про локальну реалізацію кінцевої точки.

Структура кінцевої точки

Кожна кінцева точка складається з:

Адреси. Адреса однозначно визначає кінцеву точку і вказує потенційним споживачам на місце розташування служби. У об'єктній моделі WCF адресу представлено класом EndpointAddress. Клас EndpointAddress містить:

- властивість Uri, що представляє адресу служби;
- властивість Identity, що представляє посвідчення безпеки служби і колекцію необов'язкових заголовків повідомлень. Необов'язкові заголовки повідомлень використовуються для виведення додаткової і більш докладної інформації, необхідної для ідентифікації кінцевої точки або взаємодії з нею.

Прив'язка . Прив'язка задає спосіб зв'язку клієнта з кінцевою точкою. У тому числі наступне:

- використовуваний транспортний протокол (наприклад, TCP або HTTP);
- використовуване в повідомленнях кодування (наприклад, текст або двійкове кодування);
- необхідні вимоги безпеки (наприклад, безпека повідомлень SSL або SOAP).

Прив'язка в об'єктній моделі WCF представлена абстрактним базовим класом Binding. У більшості сценаріїв користувачі можуть використовувати тільки одну з передбачених системою прив'язок.

Контракти. Контракти показують, які функціональні можливості дає клієнтові кінцева точка. У контракті задається наступне:

- операції , які можуть бути викликані клієнтом;
- форма повідомлення;
- тип вхідних параметрів або даних, необхідних для виклику операції;
- тип обробки або відповідь повідомлення, який може очікувати клієнт.

Набір поведінки. Поведінку кінцевої точки можна використовувати для налаштування локальної поведінки кінцевої точки. Поведінка кінцевої точки виконує це шляхом участі в процесі створення середовища виконання WCF. Прикладом поведінки є властивість ListenUri, що дозволяє вказувати відмінний від адреси SOAP або WSDL адресу прослуховування.

Кінцеву точку для служби можна вказати або імперативним методом (за допомогою коду), або декларативним (через налаштування).

Типи контрактів

Існують наступні типи контрактів:

- **Service Contract** – визначає операції і поведінку сервісу.
- **Data Contract** – визначає, які типи даних приймають і передаються службою.
- **Operation Contract –** вказує, що метод визначає операцію, яка є частиною контракту служби в Windows Communication Foundation (WCF) прикладній програмі.
- **Message Contract** – визначення application-specific заголовків і контенту для повідомлень.
- **Fault Contract** – визначає специфічні помилки для служби, їх обробку і передачу клієнту.

Data Contract – формальна угода між службою та клієнтом, яка абстрактно описує дані, обмін якими відбувається. Це означає, що для взаємодії клієнт і служба не зобов'язані спільно використовувати одні й ті ж типи даних, досить спільно використовувати одні й ті ж контракти даних. Контракт даних для кожного параметра і типу, що повертається, чітко визначає, які дані серіалізуються (перетворюються на XML) для обміну.

За замовчуванням в Windows Communication Foundation (WCF) для серіалізації і десеріалізації даних (перетворення в XML і назад) використовується модуль серіалізації, що називається серіалізатор контракту даних. Всі типи даних – примітиви .NET Framework, такі як integer і string, а також деякі типи, які обробляються як примітиви, такі як DateTime і XmlElement, можуть бути серіалізовані без додаткової обробки і вважаються типами, які за замовчуванням містять контракти даних. Багато типів .NET Framework також містять контракти даних.

Для серіалізації нових створених складних типів необхідно визначити контракти даних. За замовчуванням DataContractSerializer визначає контракт даних і серіалізуються всі відкриті типи даних. Всі відкриті властивості читання/запису і поля типу серіалізуються. Можна виключати члени з серіалізації за допомогою IgnoreDataMemberAttribute. Також можна явно створювати контракт даних за допомогою атрибутів DataContractAttribute і DataMemberAttribute. Зазвичай це робиться за допомогою застосування атрибута DataContractAttribute до типу. Даний атрибут може бути застосований до класів, структур і перерахувань. Після цього необхідно застосувати атрибут DataMemberAttribute до кожного члена типу контракту даних, щоб вказати, що він є членом даних, який необхідно серіалізувати.

Існують деякі моменти, які необхідно враховувати при створенні контрактів даних:

- Атрибут IgnoreDataMemberAttribute враховується тільки при використанні в невідмічених типах. Сюди входять типи, які не зазначені ні одним з атрибутів DataContractAttribute, SerializableAttribute, CollectionDataContractAttribute, EnumMemberAttribute або відзначені як серіалізовані яким-небудь іншим способом (наприклад, IXmlSerializable).
- Атрибут DataMemberAttribute застосовується до полів та властивостей.
- Рівні спеціальних можливостей членів (внутрішній, закритий, захищений або відкритий) ніяк не впливають на контракт даних.
- Атрибут DataMemberAttribute ігнорується, якщо він застосований до статичного члену.
- Під час серіалізації для членів даних властивостей викликається код property-get, який повертає значення властивостей, які серіалізуються.
- Під час серіалізації спочатку створюється не ініціалізований об'єкт без виклику будь-яких конструкторів типу. Потім десеріалізуються всі члени даних.
- Під час десеріалізації для членів даних властивостей викликається код property-set, що задає значення властивостям, які серіалізуються.
- Щоб контракт даних був допустимим, всі його члени даних повинні бути такими, що серіалізуються.

Для його визначення контракту даних необхідно відзначити атрибути:

[DataContract] – клас, який визначає контракт даних.

[DataMember] – кожне поле цього класу, яке буде брати участь в обміні даними.

Хід роботи

1. У вікні **Visual Studio** в меню **File** оберіть пункт **New Project**.
2. В діалоговому вікні **New Project** розгорніть вузол **Visual C#**, виберіть пункт **WCF**, далі - **WCF Service Library**. Натисніть кнопку **ОК** для відкриття проекту.

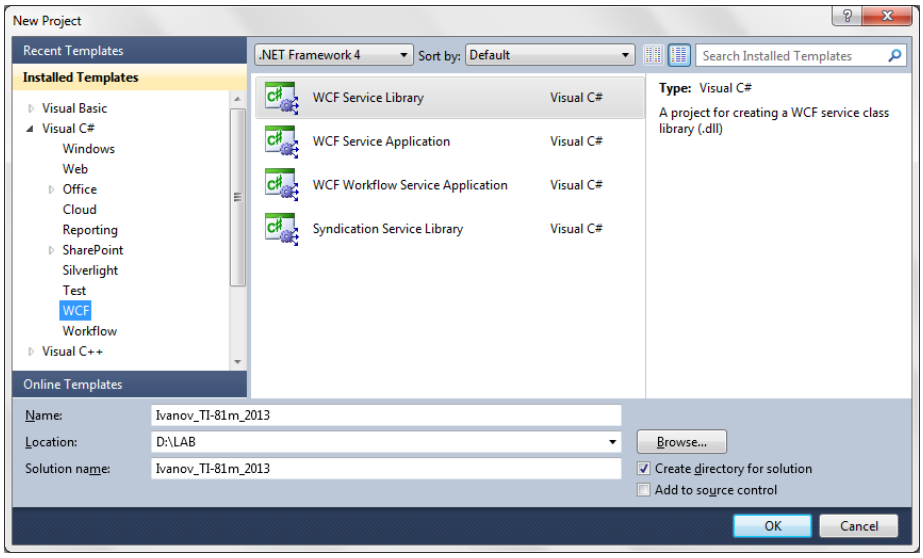


Рис. 2.1 Створення нового проекту WCF

3. В **Solution Explorer** оберіть файл **IService1.cs** і знайдіть наступний рядок:

```
[OperationContract]
string GetData(int value);
```

Змініть тип параметра **value** на string:

```
[OperationContract]
string GetData(string value);
```

4. В **Solution Explorer** оберіть файл **Service1.cs** і знайдіть наступні рядки:

```
public string GetData(int value)
{
    return string.Format("You entered: {0}", value);
}
```

Змініть тип параметра значення на string:

```
public string GetData(string value)
{
    return string.Format("You entered: {0}", value);
}
```

5. Щоб запустити сервіс, натисніть клавішу F5. Форма **WCF Test Client** з'явиться на екрані та завантажить сервіс.

6. В формі **WCF Test Client** двічі натисніть метод **GetData()** під вузлом **IService1**. З'явиться вкладка **GetData**.

7. В області **Request** виберіть поле **Value** і введіть Hello.

8. Натисніть кнопку **Invoke**. Якщо з'явиться діалогове вікно попередження системи безпеки, натисніть кнопку **ОК**. Результат буде виведено в область **Response**.

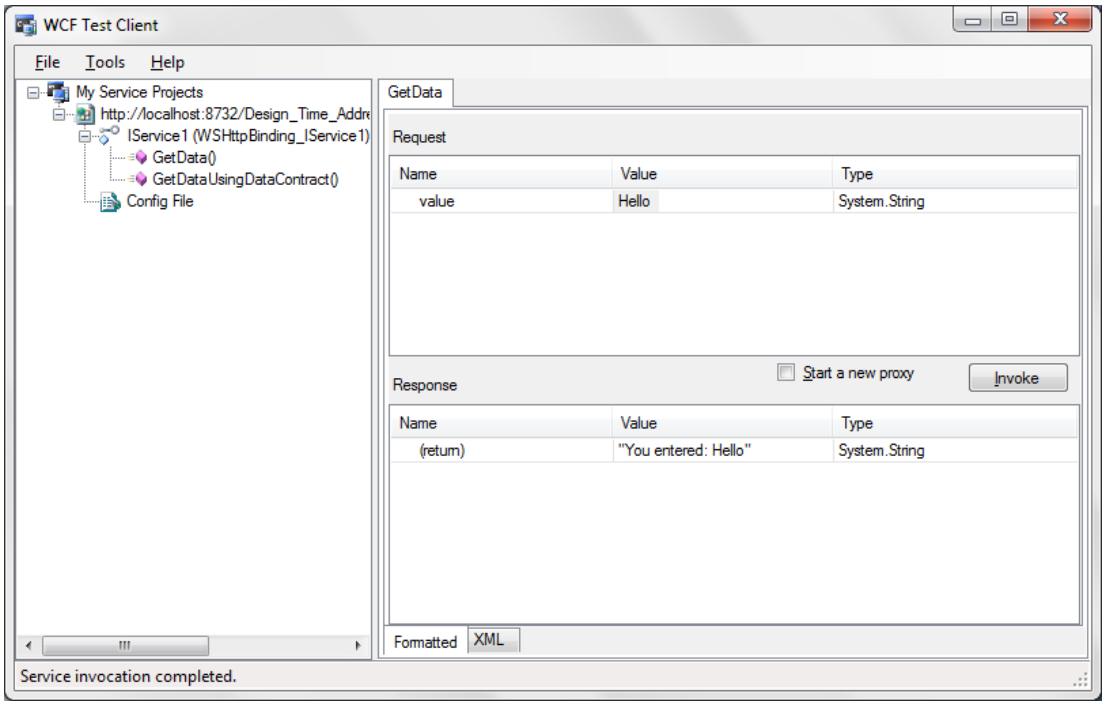


Рис. 2.2 Форма WCF Test Client (GetData)

9. Закрийте тестову форму.

10. В **Solution Explorer** оберіть файл **IService1.cs** і знайдіть наступний рядок:

```
[OperationContract]
CompositeType GetDataUsingDataContract(CompositeType composite);
```

та код:

```
[DataContract]
public class CompositeType
{
    bool boolValue = true;
    string stringValue = "Hello ";

    [DataMember]
    public bool BoolValue
    {
        get { return boolValue; }
        set { boolValue = value; }
    }
    [DataMember]
    public string StringValue
    {
        get { return stringValue; }
        set { stringValue = value; }
    }
```

```
}
```

Змініть назви користувацьких класів та типи і назви полів класів визначених як [DataContract], а також змініть назву методу, назву класу, до якого належить метод і параметри методу визначеного в [OperationContract] (відповідно до варіанту завдання). Після змін, отримаємо:

```
[OperationContract]
Student GetStudentUsingDataContract(Student composite, ExistedGroups groupname);
```

та:

```
[DataContract]
    public class Student
    {

        [DataMember]
        public int ID;
        [DataMember]
        public string Name;
        [DataMember]
        public string Surname;
        [DataMember]
        public float Rating;
        [DataMember]
        public string Group;
    }


[DataContract]
    public enum ExistedGroups
    {
        [EnumMember]
        TI,
        [EnumMember]
        TS,
        [EnumMember]
        TZ
    }
```

11. В **Solution Explorer** оберіть файл **Service1.cs** і знайдіть наступні рядки:

```
public CompositeType GetDataUsingDataContract(CompositeType composite)
{
   if (composite.BoolValue)
```

```
{
    composite.StringValue += "Suffix";
}
return composite;
    }
```

Змініть назву методу та назву класу, до якого він належить, а також параметри та код методу (відповідно варіанту завдання):

```
public Student GetStudentUsingDataContract(Student composite, ExistedGroups groupname)
{
    composite.Group = groupname + "-" + composite.Group;
    return composite;
}
```

12. Натисніть клавішу F5, щоб знову запустити сервіс.

13. В формі **WCF Test Client** двічі натисніть метод **GetStudentUsingDataContract()** під вузлом **IService1**. З'явиться вкладка **GetStudentUsingDataContract**.

14. В області **Request** задайте у полі **Value** відповідні значення полів Group, ID, Name, Rating, Surname та виберіть для groupname відповідно: TI, TS або TZ.

15. Натисніть кнопку **Invoke**. Якщо з'явиться діалогове вікно попередження системи безпеки, натисніть кнопку **ОК**. Результат буде виведено в область **Response**.

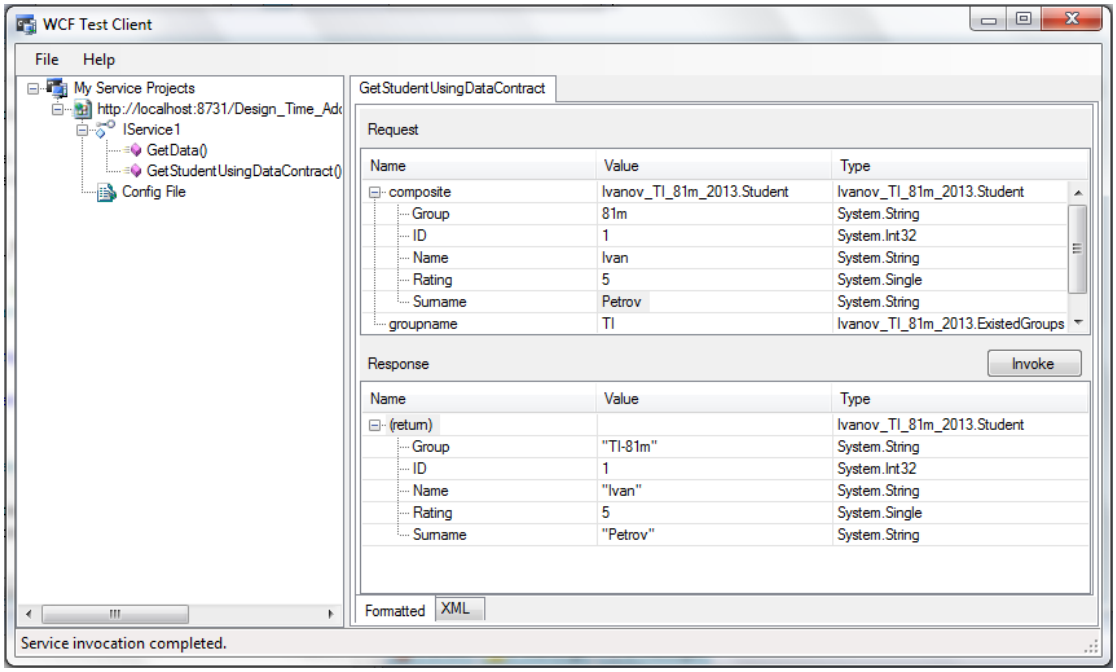


Рис. 2.3 Форма WCF Test Client (GetStudentUsingDataContract)

16. Закрийте тестову форму.

Завдання

17. Назва проекту задається по шаблону: Прізвище_Група_Рік.

18. Назва сервісу повинна відповідати предметній області, відповідно варіанту завдання (табл. 2.1).

Табл.2.1

№ варіанту	**Завдання**		
	Рядок виводу методу GetData	Предметна область для виводу методу GetDataUsingDataContract	Типи даних
1	"Hello, value"	Зоопарк	int, float/double, string
2	"value, You will learn something new today"	Банк	int, float/double, string
3	"Do You want to receive value mark?"	Лікарня	int, float/double, string
4	"value, great!"	Бібліотека	int, float/double, string
5	"value, this is the best laboratory work"	Аптека	int, float/double, string
6	"Aloha value!"	Ресторан	int, float/double, string
7	"value, have a nice day"	Автомобільний салон	int, float/double, string
8	"value, be ready to work hard"	Книжковий магазин	int, float/double, string
9	"Your word was 'value' - am I right?"	Картинна галерея	int, float/double, string
10	"This laboratory work is made by value!"	Аквапарк	int, float/double, string

19. Створити хоча б один клас та одне перерахування, визначені, як [DataContract].

Контрольні питання

1. Що представляє собою Windows Communication Foundation (WCF)?
2. Перерахуйте основні можливості WCF?
3. Для чого потрібні кінцеві точки служби та з яких властивостей вони складаються?
4. Що показує властивість address?
5. Що показує властивість binding?
6. Що визначає властивість contract?
7. Що задає властивість set of behaviors?

8. Перерахуйте типи контрактів в WCF?
9. Що таке Service Contract?
10. Що таке Data Contract?

Лабораторна робота №3

Тема: «Фільтрація даних. LINQ»

Цель: отримати теоретичні знання щодо LINQ-запитів, отримати навички роботи із запитами LINQ.

1. Состав рабочего места

1.1 Персональний комп`ютер

1.2 ПЗ Windows

1.3 Visual Studio 2010.

2. Теоретична частина.

2.1 Загальна інформація щодо LINQ запитів.

Технологія LINQ (Language Integrated Query, запит, інтегрований в мову) дозволяє писати безпечні з точки зору типізації структуровані запити до локальних колекцій об'єктів і віддалених джерел даних. Це нова функціональна можливість мови C # 3.0 і платформи Framework.

LINQ дозволяє будувати запити до будь-якої колекції, що реалізує інтерфейс IEnumerable<>, чи то масиву, списку, колекції XML DOM або віддаленого джерела даних, таких як таблиці на SQL - сервері.

LINQ – це розширення мови програмування, яке дає можливість формувати запити даних з контролем типів. За допомогою LINQ можна організувати 2 види запиту - локальний (тобто коли йде обробка локальної колекції) та інтерпретативний (тобто коли йде обробка віддаленого джерела даних).

Основною одиницею є послідовність – це будь-яка колекція, яка реалізує інтерфейси IEnumerable<> і IQueryable<>.

Оператор запиту – це метод, який перетворює послідовність.

Запит – вираз, за допомогою якого відбувається перетворення послідовності за допомогою операторів запиту.

Оператор запиту ніколи не змінює вхідну послідовність, замість цього він повертає нову.

При роботі з LINQ на мові програмування C# варто звернути увагу на такі розширення мови, як лямбда-вирази, анонімні типи, методи розширення, дерево виразів, так як вони є потужними інструментами створення запитів.

LINQ використовується при створенні запитів для sql, xml, entity і внутрішніх колекцій. Так само варто зауважити, що всі змінні в запиті мають чіт вику типізацію, але в деяких випадках можна не вказувати тип, і компілятор сам його визначить .

За допомогою LINQ можна реалізувати два різних види синтаксису - синтаксис запитів і методів.

Приклад використання синтаксису запитів :

```
int [] msv = { -1, 2, 3, -4 } ;
var res = from n in msv
        where n > 0
        select n ; / / 2, 3
```

Приклад використання синтаксису методів:

```
int [] msv = { -1, 2, 3, -4 } ;
var res = msv.Where ( n => n > 0 ) ; / / 2, 3
```

Дуже важливо те, що при написанні запиту – він повинен починатися з from in і закінчуватися select або group. Багато з методів здатні приймати у вигляді параметра лямбда-вирази і делегати. Так само їх можна застосовувати один за одним. Ще одним важливим зауваженням є те, що будь-який запит, написаний за допомогою LINQ може бути реалізований з використанням звичайних методів мови програмування, тобто без використання запитів.

Різниця між запитами LINQ і методами полягає в тому, що запити, створені на LINQ є більш зрозумілими для людини. Для машини ж, немає ніякої різниці і ваш вибір не вплине на продуктивність, якщо власних синтаксис буде правильним. Так само варто знати, що запити і методи можна поєднувати один з одним.

2.2. Основи LINQ

Найпростіший підхід до LINQ полягає в розгляді його роботи з колекціями, які знаходяться в пам'яті. Йдеться про LINQ to Objects – найпростішої формі LINQ. По суті LINQ to Objects дає можливість замінити ітеративну логіку (таку як блок
foreach) декларативним виразом LINQ. Наприклад, коли необхідно отримати список всіх співробітників, чиї прізвища починаються з літери D. Використовуючи функціональний код C #, код матиме вигляд:

```
/ / Отримати повну колекцію співробітників від допоміжного методу.
List<EmployeeDetails> employees = db.GetEmployees ();
/ / Знайти придатних співробітників.
```

```
List<EmployeeDetails> matches = new List<EmployeeDetails> ();
foreach (EmployeeDetails employee in employees)
{
 if (employee.LastName.StartsWith ("D"))
 {
 matches.Add (employee);
 }
}
```

Потім можна виконати інше завдання з колекцією збігів, або ж відобразити її на сторінці, як показано нижче:

```
gridEmployees.DataSource = matches;
gridEmployees.DataBind ();
```

Такий самий результат можна отримати за допомогою виразу LINQ. Наступний приклад показує, як можна переписати код, замінивши блок foreach запитом LINQ:

```
List<EmployeeDetails> employees = db.GetEmployees ();
IEnumerable<EmployeeDetails> matches;
matches = from employee in employees
 where employee.LastName.StartsWith ("D")
 select employee;
gridEmployees.DataSource = matches;
gridEmployees.DataBind ();
```

Запит LINQ використовує набір нових ключових слів, включаючи from, in, where і select. Він повертає нову колекцію, яка містить лише відповідають умові запиту результати. Кінцевий результат цього коду той же самий – колекція імен matches, яка заповнена співробітниками, чиї прізвища починаються на D, і яка потім відображається в сітці.

2.3. Відкладене виконання

Очевидна різниця між підходом foreach і кодом, що використовує вираз LINQ, полягає в способі типізації колекції matches. У коді foreach вона створюється як колекція спеціального типу – в даному випадку типізований List<T>. У прикладі з LINQ колекція matches представляється тільки через інтерфейс IEnumerable<T>, який вона реалізує. Цю відмінність продиктовано тим, що LINQ використовує відкладене виконання. В свою чергу об'єкт matches не є просто колекцією, яка містить відповідні об'єкти EmployeeDetails.

Замість цього він являє собою спеціалізований об'єкт LINQ, що володіє здатністю отримати дані тоді, коли в них з'явиться потреба .

У попередньому прикладі об'єкт matches – це екземпляр WhereIterator<T>, котрий являє собою приватний клас, вкладений всередину класу System.Linq.Enumerable. Залежно від специфічного запиту, LINQ може повертати різні об'єкти. Наприклад, об'єднаний вираз, що комбінує дані з двох різних колекцій, поверне екземпляр приватного класу UnionIterator<T>. Якщо ж спростити вираз, виключивши з нього конструкцію where, то буде задіяно простий SelectIterator<T>.

Примітка. Немає потреби знати, який саме конкретний клас ітератора використовує код, тому що взаємодія з результатами відбувається через інтерфейс IEnumerable<T>. Але можна визначити тип об'єкта під час виконання, використовуючи режим зневаження (debug) Visual Studio (навівши курсор на змінну в режимі перерваного виконання).

Об'єкти ітераторів LINQ додають додатковий шар між визначенням виразу LINQ та його виконанням. Як тільки здійснюється крок ітерації через ітератор LINQ, подібний WhereIterator<T>, він отримує необхідні дані. Наприклад, якщо ви пишете блок foreach, який проходить по колекції matches, ця дія змушує обчислювати результат виконання виразу LINQ. Залежно від точного типу виразу, LINQ може виконуватися за один крок, або ж частина за частиною – в міру ітерації. У попередньому прикладі дані можуть бути вилучені частинами, але в разі отримання інформацію з бази даних або застосування порядку сортування до результатів LINQ використовував би іншу стратегію і отримав би всі результати на початку циклу.

Не існує технічних причин того, що LINQ застосовує відкладене виконання, але є багато причин вважати це хорошим підходом. У багатьох випадках він дозволяє LINQ використовувати техніку оптимізації продуктивності, яка в іншому випадку була б неможливою. Наприклад, коли використовуються бази даних з LINQ to SQL, можнф уникнути завантаження зв'язаних даних, які насправді не використовуєте .

2.4. LINQ-вирази

Значення, що повертається виразами LINQ є об'єктом ітератора, що реалізує IEnumerable<T>. Коли перераховуються елементи об'єкта ітератора, LINQ виконує свою роботу. У LINQ є важлива симетрія. Вирази LINQ працюють з об'єктами, що реалізують IEnumerable<T> (такими як колекція List<EmployeeDetails> з попереднього прикладу), і повертають LINQ об'єкти, що реалізують Enumerable<T> (як WhereIterator<T> з попереднього прикладу). Тому можна передати результат одного виразу LINQ іншому виразом LINQ і т.д. Такий ланцюжок виразів LINQ обчислюється в кінці, коли виконується ітерація за кінцевими даними. Залежно від типу джерела даних LINQ часто

може "сплавити" ланцюжок виразів в одну операцію, і таким чином, виконати її найбільш ефективним способом.

Вирази LINQ характеризуються зовнішньою схожістю із запитами SQL, хоча порядок складових їх конструкцій змінений. Всі вирази LINQ повинні мати конструкцію from, що задає джерело даних, і конструкцію select, що вказує дані, які необхідно отримати (або group, визначальну групи, в які слід помістити видобувні дані).

Конструкція from є найпершою в синтаксисі:

```
matches = from employee in employees... ;
```

Конструкція from ідентифікує дві частини інформації. Слово після in, ідентифікує джерело даних, в даному випадку – об'єкт колекції з іменем employees, що містить екземпляри EmployeeDetails. Слово, яке знаходиться відразу після from, призначає псевдонім, який представляє індивідуальні елементи з джерела даних. Для поточного виразу кожен EmployeeDetails називається employee. Можна пізніше використовувати цей псевдонім при побудові інших частин виразу, таких як конструкції вибірки і фільтрації.

Розглянемо найпростіший з можливих запит LINQ. Він просто отримує повний набір даних з колекції employees :

```
IEnumerable<EmployeeDetails> matches ;
matches = from employee in employees
          select employee ;
```

2.5. Проекції

Конструкцію select можна змінити, щоб отримати підмножину даних. Наприклад, ось як можна отримати список імен співробітників :

```
IEnumerable<string> matches ;
matches = from employee in employees
 select employee.FirstName ;
```

або список рядків, що включають імена та прізвища:

```
matches = from employee in employees
 select employee.FirstName + employee.LastName ;
```

Можна використовувати стандартні операції C# для числових даних і рядків для модифікації інформації під час вибірки. Що ще цікавіше, що можна динамічно визначити новий клас – оболонку тільки для тієї інформації, яку ви хочете повернути. Наприклад, якщо ви хочете отримати і ім'я, і прізвище, але в окремих рядках, можна створити усічену версію класу EmployeeDetails, що складається тільки з властивостей FirstName і LastName. Щоб зробити це, необхідно використати анонімний тип. Базова техніка полягає в додаванні ключового слова new до select. Ось приклад:

```
var matches = from employee in employees
```

```
select new { First = employee.FirstName, Last = employee.LastName } ;
```

Цей вираз при виконанні поверне набір об'єктів неявно створеного класу. Кожен об'єкт має дві властивості: First і Last. Ви ніде не побачите визначення цього класу, тому що він генерується компілятором і отримує автоматично сгенерированное ім'я. Однак можна локально використовувати цей клас, звертаючись до властивостей First і Last, і навіть використовувати прив'язку даних (у цьому випадку C# отримує відповідні значення за іменем властивості, використовуючи рефлексію). Здатність трансформувати дані в результати зі зміненою структурою називається проекцією.

У роботі цього прикладу присутній один трюк. Вираз LINQ повертає об'єкт – літератор. Клас ітератора є узагальненим, а це означає, що він прив'язаний до певного типу елементів, в даному випадку до анонімного класу, що має дві властивості – First і Last. Однак оскільки ви не визначаєте цей клас, ви не можете визначити коректне посилання IEnumerator<T>. Рішення полягає у використанні ключового слова var. Ключове слово var також потрібно застосовувати, коли необхідно посилатися на індивідуальний об'єкт. Один приклад – коли виконується код ітерації по набору, поверненого попереднім виразом LINQ :

```
foreach ( var employee in matches )
{
 / / ( Щось робити з employee.First і employee.Last. )
}
```

Нагадаємо, що ключове слово var дозволяється під час компіляції і не може бути використано як змінна – член класу. У результаті такий підхід не дає можливості передавати примірник анонімного класу між методами. Проте не обов'язково використовувати анонімні типи при виконанні проекції. Можна визначити тип формально і потім використовувати його у виразі. Наприклад, якщо створити наступний клас EmployeeName:

```
public class EmployeeName
{
 public string FirstName
 { Get ; set ;}

 public string LastName
 { Get ; set ;}
}
```

то можна замінити об'єкти EmployeeDetails на EmployeeName у запиті:

```
IEnumerable<EmployeeName> matches = from employee in employees
select new EmployeeName { FirstName = employee.FirstName ,
```

LastName = employee.LastName } ;

Цей запит працює тому, що властивості FirstName і LastName є загальнодоступними як для читання, так і для запису. Після створення об'єкта EmployeeName LINQ встановлює ці властивості. Альтернативно можна додати набір дужок після класу EmployeeName і вказати аргументи для параметризованого конструктора:

```
IEnumerable<EmployeeName> matches = from employee in employees
 select new EmployeeName(FirstName, LastName);
```

2.6. Фільтрація и сортування

У першому прикладі LINQ показано, як конструкція where дозволяє фільтрувати результати, щоб отримати тільки ті, що відповідають певній умові. Наприклад, наведений нижче код служить для знаходження співробітників, прізвища яких починаються з певної літери:

```
IEnumerable<EmployeeDetails> matches ;
matches = from employee in employees
 where employee.LastName.StartsWith ( "D" )
 select employee ;
```

Конструкція where приймає умову, що обчислюється для кожного елемента. Якщо вона справджується, елемент включається в результат. Однак і в цьому випадку LINQ зберігає ту ж модель відкладеного виконання, а це означає, що конструкція where не обчислюються до тих пір, поки дійсно не буде виконано ітерацію по результату. Можна комбінувати безліч умовних виразів з операціями " І " (&&) і " АБО" (| |), а також застосовувати операції порівняння (такі як<,<= ,> і > =). Наприклад, можна створити наступний запит, щоб відфільтрувати продукти дорожчі певної порогової ціни :

```
IEnumerable<Product> matches ;
matches = from product in products
 where product.UnitsInStock > 0 && product.UnitPrice > 3.00M
 select product ;
```

Одна цікава властивість виразу LINQ полягає в тому, що можна вбудовувати в нього свої власні методи. Наприклад, можна створити функцію по імені TestEmployee(), яка перевіряє співробітника і повертає true або false залежно від того, потрібно його включати в результат чи ні:

```
private bool TestEmployee( EmployeeDetails employee )
{
 return employee.LastName.StartsWith ( "D" )
}
```

Потім можна використовувати метод TestEmployee( ) наступним чином:

```
IEnumerable<EmployeeDetails> matches ;
matches = from employee in employees
where TestEmployee( employee )
select employee ;
```

Операція orderby настільки ж очевидна. Вона моделює синтаксис оператора ORDER BY з SQL. Ви просто надаєте список з одного або більше значень для використання в сортуванні, розділяючи їх комами. Можна додати слово decending після імені поля, щоб сортувати в порядку убування.

Ось приклад сортування:

```
IEnumerable<EmployeeDetails> matches;
matches = from employee in employees
 orderby employee.LastName, employee.FirstName
 select employee;
```

2.7. Групування і агрегація

Групування дозволяє стиснути великий набір інформації в мінімальний набір сумарних результатів. Групування є різновидом проекції, тому що об'єкти в результуючій колекції відрізняються від об'єктів у вихідній колекції. Наприклад, припустимо, є колекція об'єктів Product і необхідно розбити їх на цінові групи, кожна з яких представляє підмножину продуктів певного цінового діапазону. Кожна група реалізує інтерфейс IGrouping<T, K> з простору імен System.Linq .

Щоб використовувати групування, потрібно прийняти два рішення. По-перше, вирішити, який критерій слід застосовувати для створення групи. По-друге, потрібно вирішити, яку інформацію відображати для кожної групи. Перша задача проста. Необхіно використати ключові слова groupby і into для вибору об'єктів, що підлягають групуванню, того, як визначаються групи, і який псевдонім використовувати для посилання на індивідуальні групи. Нижче наведено приклад, що працює з колекцією об'єктів EmployeeDetails і групуючий їх на основі вмісту поля TitleOfCourtesy (Mr., Ms. Тощо).

```
var matches = from employee in employees
```

```
group employee by employee.TitleOfCourtesy into g
...
```

Але корисніше те, що можна використовувати агрегатні функції для виконання обчислень над даними групи. Агрегатні функції LINQ нагадують агрегатні функції бази даних і дозволяють підраховувати суми даних у групі, знаходити мінімум, максимум і середнє значення. Можна також фільтрувати групи на основі цих обчислюваних значень.
У наступному прикладі повертається новий анонімний тип, що включає значення ключа групи і кількість об'єктів в групі. При цьому використовується вбудований виклик методу по імені Count().

```
var matches = from employee in employees
group employee by employee.TitleOfCourtesy into g
select new { Title = g.Key, Employees = g.Count() } ;
```

Наведений вираз LINQ відрізняється від того, що було розглянуто раніше, тому що використовує розширює метод. По суті, методи, що розширюють функціонал (розширюючі методи) – центральна частина функціональності LINQ, яка не представлена виділеними операціями C#. Замість цього необхідно викликати метод безпосередньо. Count() – приклад розширюючого методу.
Що відрізняє розширюючі методи від звичайних методів – так це той факт, що Розширені методи не визначені в класі, использующем їх. Замість цього LINQ включає клас System.Linq.Enumerable, що визначає кілька десятків розширюючих методів, які можна викликати на будь-якому об'єкті IEnumerable<T>. Ці розширюючі методи також працюють з IGrouping<T, K>, тому що він розширює IEnumerable<T>. Іншими словами, наступна частина попереднього виразу LINQ :

```
select new { Title = g.Key, Employees = g.Count() } ;
```

повідомляє LINQ, що треба викликати System.Linq.Enumerable.Count(), щоб обчислити кількість елементів у групі. Поряд з Count() в LINQ також визначені більш потужні розширюючі методи, які можна використовувати в сценаріях групування – агрегатні функції Max(), Min() і Average(). Вирази LINQ, що використовують ці методи, трохи більш складні, тому що вони також використовують інший засіб C#, відомий як лямбда-виразів, що дозволяє застосовувати додаткові параметри до розширюючого методу. У разі методів Max(), Min() і Average() лямбда-вираз дозволяє вказувати, яку властивість треба використати в обчисленнях. Нижче наведено приклад, що використовує ці розширюючі методи для обчислення максимальної, мінімальної та середньої ціни елемента в кожній категорії :

```
var categories = from p in products
 group p by p.Category into g
 select new { Category = g.Key ,
 MaximumPrice = g.Max ( p => p.UnitPrice ) ,
 MinimumPrice = g.Min ( p => p.UnitPrice ) ,
 AveragePrice = g.Average ( p => p.UnitPrice ) } ;
```

Хоча цей приклад інтуїтивно зрозумілий, синтаксис лямбда-виразу виглядає дещо незвично. Хоча LINQ використовує нові ключові слова C# (такі як from, in і select), їх реалізація забезпечується іншими класами. Фактично кожен запит LINQ транслюється в серію викликів методів. Замість того, щоб покладатися на цю рансляцію, можна явно самостійно викликати методи. Наприклад, наступний простий вираз LINQ :

```
matches = from employee in employees
select employee ;
```

може бути переписано так :

```
matches = employees.Select ( employee => employee ) ;
```

Синтаксис тут виглядає досить незвично. Це схоже на те, що метод Select() викликається на колекції співробітників. Однак колекція співробітників – це звичайна колекція List<T>, і вона не включає в себе даного методу. Замість цього Select() є розширюючим методом, який автоматично надається всім класам IEnumerable<T>.

2.8. Розширені методи.

Розширені методи – новий засіб мови, що з'явився в C # 2008. По суті, розширені методи дозволяють визначити метод в одному класі, а викликати його так, нібито він визначений в іншому класі. Методи виразів LINQ визначенні в класі System.Linq.Enumerable, але вони можуть викликатися з будь-яким об'єктом IEnumerable<T>. Оскільки розширені методи LINQ визначені в класі System.Linq.Enumerable, вони доступні тільки в тому випадку, якщо цей клас доступний в поточному контексті. Якщо не імпортувати простір імен System.Linq, то не можна написати явно або неявно ніяких виразів LINQ – в цьому випадку компілятор згенерує помилку. Найпростіший шлях до розуміння цієї техніки – поглянути, що розширює метод. Ось визначення розширеного методу Select() у класі System.Linq.Enumerable:

```
public static IEnumerable<TResult> Select<TSource, TResult> (
 this IEnumerable<TSource> source, Func<TSource, TResult> selector )
{... }
```

Розширені методи керуються невеликим набором правил. Всі розширені методи повинні бути статичними. Розширені методи можуть повертати будь-які типи даних і приймати будь-яку кількість параметрів. Проте перший параметр – це завжди посилання на об'єкт, на якому викликається розширений метод (і вказується ключовим словом this). Тип даних для цього параметра визначає класи, для яких доступний цей розширений метод .

Наприклад, в розширеному методі Select() перший параметр - IEnumerable<T>:

```
public static IEnumerable<TResult> Select<TSource, TResult> (
 this IEnumerable<TSource> source, Func<TSource, TResult> selector )
```

Це говорить про те, що даний розширений метод може бути викликаний на екземплярі будь-якого класу, що реалізовує IEnumerable<T> (включаючи колекції зразок List<T>). Тобто метод Select<T> приймає ще один параметр – делегат для вказівки обраної інформації. І, нарешті, що повертається значення методу Select() – це об'єкт IEnumerable<T>; в даному випадку це екземпляр приватного класу SelectIterator.

Ось як виглядає повний код, що використовується LINQ в методі Enumerable.Select():

```
public static IEnumerable<TResult> Select<TSource, TResult>(
 this IEnumerable<TSource> source, Func<TSource, TResult> selector)
{
 if (source == null)
 {
 throw new ArgumentNullException("source");
 }
 if (selector == null)
 {
 throw new ArgumentNullException("selector");
 }
 return SelectIterator<TSource, TResult>(source, selector);
}
```

2.9. Лямбда-вирази

Лямбда-вираз – це ще одне нововведення синтаксису C # 2008, що засновано на методах виразах LINQ. Лямбда-вираз передається методу Select(), як показано нижче:

```
matches = employees.Select ( employee => employee ) ;
```

Під час виклику методу Select() об'єкт співробітника передається йому в першому параметрі. Це – джерело запиту. Другий параметр вимагає делегата, який, вказує на метод. Цей метод виконує роботу з вибору і викликається по одному разу для кожного екземпляра в колекції. Метод вибірки приймає оригінальне значення (в даному випадку – об'єкт співробітника) і повертає обраний результат. Попередній приклад виконує найбільш прямолінійну логіку вибірки – бере оригінальний об'єкт співробітника і повертає його без змін. Метод Select() очікує делегата. Можна просто застосувати звичайний делегат, який вказує на іменований метод, який створено класі, але це зробило б код досить громіздким.
Простий підхід полягає в застосуванні анонімного методу, що дозволяє визначити метод вбудованим чином там, де його необхідно використати – як аргумент методу Select(). Анонімні методи починаються зі слова delegate, за яким йде оголошення сигнатури методу, а за ним – набір дужок, що містять код методу. Ось як міг би виглядати попередній приклад, при використанні анонімного методу:

```
var matches = employees.Select(
 delegate ( EmployeeDetails employee )
 { Return employee ;}
 ) ;
```

Лямбда-вирази – простий спосіб скоротити цей код. Лямбда-вираз складається з двох частин, розділених символами =>. Перша частина ідентифікує параметри, що приймаються методом. У даному прикладі лямбда-вираз приймає кожен об'єкт з колекції і представляє його посиланням на ім'я employee. Друга частина лямбда-виразу визначає значення, яке необхідно повернути. Щоб краще це зрозуміти, подивимося, що станеться, якщо створити більш складну логіку вибірки, яка виконує проекцію. Як відомо, LINQ забезпечує гнучкість в отриманні тільки тих властивостей, які потрібні, або навіть в оголошенні нового типу. Наприклад, явне вираження LINQ отримує дані кожного співробітника і вставляє їх у новий анонімний тип, що включає тільки інформацію про ім'я та прізвища:

```
var matches = employees.Select(
 delegate ( EmployeeDetails employee )
 { Return new { First = employee.FirstName ,
 Last = employee.LastName } ;
 }
 ) ;
```

Тепер можна скоротити цей код, замінивши анонімний метод лямбда-виразом, який виконує ту ж роботу:

```
var matches = employees.Select ( employee =>
 new { First = employee.FirstName, Last = employee.LastName } ) ;
```

Звичайно, більшість виразів LINQ більш деталізовані, ніж розглянуті в даному розділі приклади. Більш повний вираз LINQ може додавати сортування або фільтрацію, як показано нижче:

```
IEnumerable<EmployeeDetails> matches = from employee in employees
 where employee.LastName.StartsWith ( "D" )
 select employee ;
```

Можна переписати цей вираз з наступним синтаксисом :

```
IEnumerable<EmployeeDetails> matches = employees
. Where ( employee => employee.LastName.StartsWith ( "D" ))
. Select ( employee => employee ) ;
```

Одна з властивостей явного синтаксису LINQ полягає в тому, що він прояснює порядок операцій. У попередньому прикладі легко побачити, що все починається з колекції співробітників, потім йде виклик методу Where() і, нарешті, виклик методу Select(). Якщо використовується більше операцій LINQ, то необхідно скласти довшу серію викликів методів. Метод Where() працює подібно методу Select(). І Where(), і Select() – розширені методи, і обидва використовують лямбда-вирази, які застосовують простий метод. Метод Where() застосовує лямбда-вираз, що перевіряє кожен елемент і повертає true, якщо він має бути включений в результат. Метод Select() застосовує лямбда-вираз, що трансформує кожен елемент даних в потрібне уявлення. У класі System.Linq.Enumerable є багато розширених методів, що працюють аналогічним чином. Найчастіше використовують неявний синтаксис для створення виразів LINQ. Однак може бути чимало випадків, коли знадобиться явний синтаксис. У кожному випадку розуміння того, як вирази відображаються на викликах методів, як розширені методи включаються в об'єкти IEnumerable<T>, і як лямбда-вирази інкапсулюють фільтрацію, сортування, проекцію та інші деталі, значно прояснює внутрішню роботу LINQ .

3. Хід роботи

1. Підключіть простір імен LINQ до проекту.
2. Продемонструйте роботу методу Select()
3. Продемонструйте роботу методу Where()
4. Продемонструйте роботу методу Groupby()
5. Створіть розширений метод на типовому класі .NET.
6. Продемонструйте роботу методів LINQ за допомогою двох варіантів синтаксису.

4. Контрольні запитання

1.1. Які способи написання запитів LINQ ви знаєте

1.2. На якому типі базується робота LINQ?

1.3. Поясніть, що таке відкладене виконання?

1.4. Що таке лямбда-вираз?

1.5. Що таке проекція?

Лабораторна робота № 4

Тема: «Багатопотокові програми»

Мета: вивчити технологію створення багато потокових додатків у сучасних системах програмування.

1. Склад робочого місця:

1.1 Персональний комп'ютер

1.2 Microsoft Visual Studio 2010 або 2012

2. Теоретична частина.

Для застосування багато поточності існує кілька причин. Припустимо, у додатку робиться звернення до якогось сервера в мережі, яка може зайняти певний час. Навряд чи захочеться, щоб користувальницький інтерфейс через це блокувався, і користувачеві довелося просто чекати моменту, коли від сервера повернеться відповідь. Користувач може виконувати в цей час якісь інші дії чи взагалі скасувати відправлений серверу запит. У таких ситуаціях застосування багатопоточності приносить користь.

Для всіх видів активності, що вимагають очікування, наприклад, через одержання доступу до файлу, бази даних або мережі, може запускатися новий потік, що дозволяє виконувати в цей же час інші завдання. Нить може допомогти, навіть якщо є одні тільки насичені в плані обробки завдання. Численні потоки одного і того ж процесу можуть одночасно виконуватися різними ЦП або, що частіше зустрічається в наші дні, різними ядрами одного многоядерного ЦП.

Зрозуміло, необхідно знати особливості одночасного виконання безлічі потоків. Через те, що вони виконуються в один і той же час, при отриманні ними доступу до одних і тих же даних можуть виникати проблеми. Щоб цього не відбувалося, повинні бути реалізовані механізми синхронізації.

Потік (thread) являє собою незалежну послідовність інструкцій в програмі. Потоки відіграють важливу роль як для клієнтських, так і для серверних додатків. Наприклад, під час введення коду C# у вікні редактора Visual Studio проводиться аналіз на предмет різних синтаксичних помилок. Цей аналіз здійснюється окремим фоновим потоком. Те ж саме відбувається і в засобі перевірки орфографії в Microsoft Word. Один потік очікує введення даних користувачем, а інший в цей час виконує у фоновому режимі деякий аналіз. Третій потік може зберігати записувані дані в тимчасовий файл, а четвертий – завантажувати додаткові дані з Інтернету.

У додатку, яке функціонує на сервері, один потік завжди очікує надходження запиту від клієнта і тому називається потоком-слухачем (listener thread). При отриманні запиту він відразу ж пересилає його окремому робітникові потоку (worker thread), який

далі сам продовжує взаємодіяти з клієнтом. Поток-слухач після цього негайно повертається до своїх обов'язків по очікуванню надходження наступного запиту від чергового клієнта.

Кожен процес складається з ресурсів, таких як віконні дескриптори, файлові дескриптори і інші об'єкти ядра, має виділену область у віртуальній пам'яті і містить як мінімум один потік. Потоки плануються до виконання операційною системою. У будь-якого потоку мається пріоритет, лічильник команд, який вказує на місце в програмі, де відбувається обробка, і стек, в якому зберігаються локальні змінні потоку. Стек у кожного потоку виглядає по-своєму, але пам'ять для програмного коду і купа розділяються серед всіх потоків, які функціонують усередині одного процесу.

Це дозволяє потокам всередині одного процесу швидко взаємодіяти між собою, оскільки всі потоки процесу звертаються до однієї і тієї ж віртуальної пам'яті. Однак це також і ускладнює справу, оскільки дає можливість безлічі потоків змінювати одну і ту ж область пам'яті.

3. Основи багатопоточної обробки

Розрізняють два різновиди багатозадачності: на основі процесів і на основі потоків. У зв'язку з цим важливо розуміти відмінності між ними. Процес відповідає за управління ресурсами, до числа яких належить віртуальна пам'ять і дескриптори Windows, і містить як мінімум один потік. Наявність хоча б одного потоку є обов'язковим для виконання будь-якої програми. Тому багатозадачність на основі процесів – це засіб, завдяки якому на комп'ютері можуть паралельно виконуватися дві програми і більше.

Потік являє собою координовану одиницю виконуваного коду. Своїм походженням цей термін зобов'язаний поняттю "потік виконання". При організації багатозадачності на основі потоків у кожного процесу повинен бути принаймні один потік, хоча їх може бути і більше. Це означає, що в одній програмі одночасно можуть вирішуватися два завдання і більше. Наприклад, текст може форматироваться в редакторі тексту одночасно з його висновком на друк, за умови, що обидва ці дії виконуються в двох окремих потоках.

Відмінності в багатозадачності на основі процесів і потоків можуть бути зведені до наступного: багатозадачність на основі процесів організується для паралельного виконання програм, а багатозадачність на основі потоків – для паралельного виконання окремих частин однієї програми.

Головна перевага багатопотокової обробки полягає в тому, що вона дозволяє писати програми, які працюють дуже ефективно завдяки можливості вигідно використовувати час простою, неминуче виникає в ході виконання більшості програм. Як відомо, більшість пристроїв введення-виведення, будь то пристрої, підключені до мережевих портів, накопичувачі на дисках або клавіатура, працюють набагато повільніше, ніж центральний процесор (ЦП). Тому більшу частину свого часу програмі доводиться

очікувати відправки даних на пристрій вводу-виводу або прийому інформації з нього. А завдяки многопоточной обробці програма може вирішувати якусь іншу задачу під час вимушеного простою.

Наприклад, у той час як одна частина програми відправляє файл через з'єднання з Інтернетом, інша її частина може виконувати читання текстової інформації, що вводиться з клавіатури, а третя – здійснювати буферизацию чергового блоку даних, що відправляються.

Потік може знаходитися в одному з декількох станів. В цілому, потік може бути що виконується; готовим до виконання, як тільки він отримає час і ресурси ЦП; призупиненим, тобто тимчасово не виконуються; поновленим надалі; заблокованим в очікуванні ресурсів для свого виконання; а також завершеним, коли його виконання закінчено і не може бути відновлено.

У середовищі. NET Framework визначені два різновиди потоків: пріоритетний і фоновий. За замовчуванням створюваний потік автоматично стає пріоритетним, але його можна зробити фоновим. Єдина відмінність пріоритетних потоків від фонових полягає в тому, що фоновий потік автоматично завершується, якщо в його процесі зупинені всі пріоритетні потоки.

У зв'язку з організацією багатозадачності на основі потоків виникає потреба в особливого роду режимі, який називається синхронізацією і дозволяє координувати виконання потоків цілком певним чином. Для такої синхронізації в C# передбачена окрема підсистема.

Всі процеси складаються хоча б з одного потоку, який зазвичай називають основним, оскільки саме з нього починається виконання програми. З основного потоку можна створити інші потоки.

У мові C# і середовищі. NET Framework підтримуються обидва різновиди багатозадачності: на основі процесів і на основі потоків. Тому засобами C# можна створювати як процеси, так і потоки, а також управляти і тими й іншими. Для того щоб почати новий процес, від программирующего потрібно зовсім небагато зусиль, оскільки кожен попередній процес абсолютно відокремлений від наступного.

Набагато важливішою надається підтримка в C# багатопотокової обробки, завдяки якій спрощується написання високопродуктивних, багатопоточних програм на C# в порівнянні з деякими іншими мовами програмування.

Класи, що підтримують багатопоточне програмування, визначені в просторі імен System.Threading. Тому будь-яка багатопотокова програма на C# включає в себе наступний рядок коду:

using System.Threading;

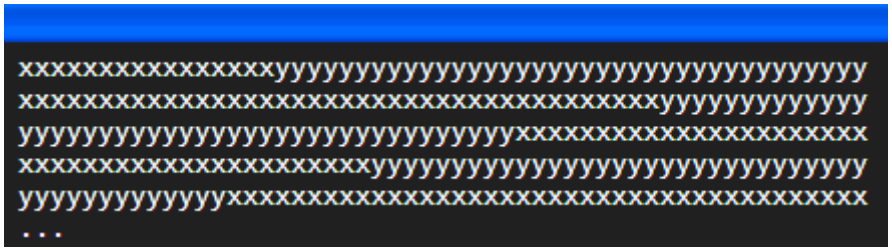
Простір назв System.Threading містить різні типи, що дозволяють створювати багатопотокові програми.

Реалізуємо простий приклад для пояснень багатопоточності:

```
using System;
using System.Threading;
using System.Threading.Tasks;
```

```
1.  class ThreadTest
2.  {
3.    static void Main()
4.    {
5.      Task t = new Task (WriteY); // объявление треда, в котором выполняется WriteY
6.      t.Start();                  // запуск треда
7.
8.      // Паралельно ему продолжает выполняется программа.
9.      for (int i = 0; i < 1000; i++) Console.Write ("x");
10.   }
11.
12.   static void WriteY()
13.   {
14.     for (int i = 0; i < 1000; i++) Console.Write ("y");
15.   }
16. }
```

Виведення програми:



Не можна сказати, що паралелізм даних не був можливий раніше засобами класу Thread. Можливість такої організації обробки даних була, однак вона вимагала чимало зусиль і часу. Бібліотека TPL значно спростила цей процес.

Паралелізм даних у бібліотеці паралельних завдань здійснюється за допомогою методів For ( ) і ForEach ( ), визначених у класі Parallel.

Метод For () використовується для того , щоб розподілити на кілька процесорів (якщо така можливість є) виконання коду в циклі. Але слід бути обережним, тому що використання даного методу може як підвищити, так і знизити продуктивність програми. Зниження продуктивності буде відбуватися в тих випадках, коли буде проведена спроба розподілити дрібні цикли, або ж коли метод, що виконується на кожному кроці циклу, тривіальний. У цих випадках витрати розподілу циклу по потоках перевищуватимуть заощаджений час, і користі від такого використання багатопоточності не буде. Необхідно бути дуже обережним.

```
1.  using System;
2.  using System.Threading;
3.  using System.Threading.Tasks;
4.
5.  namespace ParallelFor
6.  {
7.  class Program
8.  {
9.      static void Main(string[] args)
10.     {
11.         Console.WriteLine("Using C# For Loop \n");
12.
13.         for(int i=0; i <=10; i++){
14.             Console.WriteLine("i = {0}, thread = {1}",
15.                 i, Thread.CurrentThread.ManagedThreadId);
16.             Thread.Sleep(10);
17.         }
18.
19.         Console.WriteLine("\nUsing Parallel.For \n");
20.
21.         Parallel.For(0, 10, i =>
22.         {
23.             Console.WriteLine("i = {0}, thread = {1}", i,
24.             Thread.CurrentThread.ManagedThreadId);
25.             Thread.Sleep(10);
26.         });
27.
28.         Console.ReadLine();
```

```
    }
}
}
```

Виведення програми:

```
Using C# For Loop

i = 0, thread = 10
i = 1, thread = 10
i = 2, thread = 10
i = 3, thread = 10
i = 4, thread = 10
i = 5, thread = 10
i = 6, thread = 10
i = 7, thread = 10
i = 8, thread = 10
i = 9, thread = 10
i = 10, thread = 10

Using Parallel.For

i = 0, thread =  10
i = 5, thread =  7
i = 1, thread =  10
i = 6, thread =  7
i = 7, thread =  7
i = 2, thread =  10
i = 3, thread =  11
i = 8, thread =  7
i = 4, thread =  11
i = 9, thread =  7
```

```
static void Main(string[] args)
{
    foreach (int i in Enumerable.Range(1, 10))
    {
        Console.WriteLine("{0} - {1}", i, GetTotal());
    }
}

Parallel.ForEach(Enumerable.Range(1, 10), i =>
{
    Console.WriteLine("{0} - {1}", i, GetTotal());
});

static long GetTotal()
{
    long total = 0;
    for (int i = 1; i < 1000000000; i++)    // Adjust this loop according
    {                                       // to your computer's speed
```

```
19.             total += i;
20.         }
21.         return total;
22.     }
```

/* OUTPUT foreach (int i in Enumerable.Range(1, 10))

1 - 499999999500000000
2 - 499999999500000000
3 - 499999995000000000
4 - 499999995000000000
5 - 499999995000000000
6 - 499999995000000000
7 - 499999995000000000
8 - 499999995000000000
9 - 499999995000000000
10 - 499999999500000000

/* OUTPUT Parallel.ForEach(Enumerable.Range(1, 10),

2 - 499999999500000000
1 - 499999999500000000
3 - 499999999500000000
5 - 499999999500000000
4 - 499999999500000000
6 - 499999999500000000
9 - 499999999500000000
7 - 499999999500000000
8 - 499999999500000000
10 - 499999999500000000

Параметри start'a:
public void Start()

Запускает задачу Task, планируя ее выполнение в текущем планировщике TaskScheduler.

public void Start(

TaskScheduler scheduler

)

Запускает задачу Task, планируя ее выполнение в заданном планировщике TaskScheduler.

4. Завдання на лабораторну роботу

Організувати отримання і фільтрацію даних в окремих потоках. Виконання чого б то не було в паралельно потоці виконується однією коммандой Task. Використовуйте для цього бібліотеку TPL.

5. Контрольні запитання

1) Що собою представляє потік та потік-слухач?
2) Які є види багатозадачності?
3) Відмінність між багатозадачністю на основі процесів і багатозадачністю на основі потоків?
4) Які основні переваги багатопотокової обробки?
5) Які різновиди потоків є в середовищі .NET Framework?

Лабораторна робота №5

Тема: Імпорт та експорт даних

Мета: отримати теоретичні знання та практичні навички стосовно ефективного застосування методів імпорту/експорту даних з/в Excel в C#

I. Теоретична частина.

1. Загальна інформація стосовно використання стовпців в SharePoint.
2. Введення в типи контенту.

II. Практична частина.

1. Завдання на лабораторну роботу

Склад робочого місця: персональний комп'ютер, ОС Windows, Visual Studio 2010, Microsoft Excel.

I. Теоретична частина.

У даній лабораторній продемонстровано як засобами мови **C#** і платформи. **NetFramework** можна організувати роботу з **Excel** (імпорт-експорт даних з / в документ Excel).

Багатьом розробникам рано чи пізно доводиться стикатися з завданнями , які мають на увазі використання Microsoft Excel. Наприклад, до Вас приходить замовник і каже , що йому дуже важко копіювати з вашої програми дані через буфер обміну для подальшого використання (наприклад , для складання якогось звіту) . Як правило , замовник не хоче розбиратися «у ваших там программізмах» і хоче інформацію тут і зараз і «щоб було зрозуміло». Я думаю з такою ситуацією стикається, рано чи пізно, кожен. Погодьтеся, делікатна виходить ситуація, адже коли замовник (або ще гірше начальник) щось просить, то відмовити йому дуже складно. Особливо тоді, коли рішення його проблеми впирається лише в ваше незнання. Тому дана лабораторна допоможе Вам подружитися з цим страшним механізмом - експорт / імпорт даних з / в Excel.
Мета: реалізувати роботу з Microsoft Excel за допомогою засобів мови C# платформи .NetFramework. Програма повинна здійснювати імпорт або експорт даних з/в документ Excel безпосередньо з коду самої програми.

Microsoft Excel (також іноді називається Microsoft Office Excel) - програма для роботи з електронними таблицями, створена корпорацією Microsoft для Microsoft Windows, Windows NT і Mac OS. Вона надає можливості економіко-статистичних розрахунків,

графічні інструменти і, за винятком Excel 2008 під Mac OS X, мова макропрограмування VBA (Visual Basic для додатків). Microsoft Excel входить до складу Microsoft Office і на сьогоднішній день Excel є одним з найбільш популярних додатків в світі.

Для автоматизації роботи з Office клієнтська програма або створює новий екземпляр програми, або отримує посилання на вже існуючий . Як правило , корпорація Microsoft рекомендує використовувати перший варіант. Проте в певних ситуаціях виникає необхідність автоматизації вже запущеного екземпляра додатка Office. У цьому випадку клієнт автоматизації отримує з таблиці ROT (Running Object Table) посилання на COM-об'єкт сервера автоматизації . Після отримання посилання або ж запуску стає доступна вся об'єктна модель Excel (рис.1). З точки зору програміста вона виглядає так :

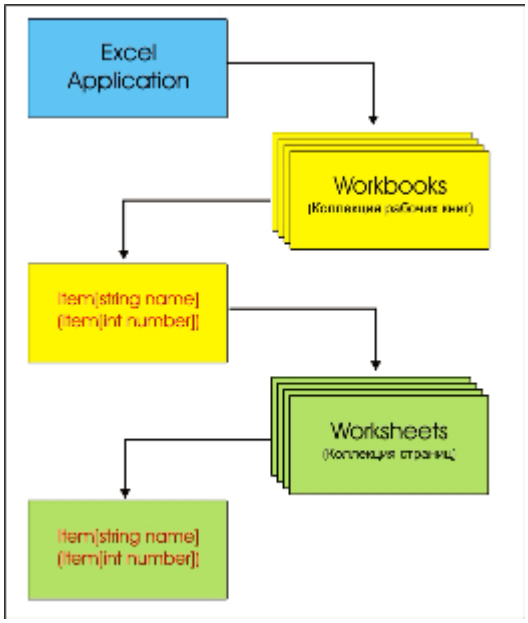


Рис. 1. Об'єктна модель Excel

Для роботи разом з об'єктом **Excel** необхідно отримати посилання на його колекцію книг. З її допомогою можна отримати доступ до будь книзі. У кожної книги є колекція сторінок, посилання на яку також потрібно отримати для доступу до конкретної сторінці. Хочу відразу зауважити, що доступ до книг і до сторінок можна отримати як за їх імені, так і за їх порядковому номеру.

Примітка!

Нумерація книг і сторінок в колекції починається з одиниці, а не з нуля (як прийнято нумерувати масиви в. NET).

Основні моменти реалізації роботи з Excel на C#

Для роботи з **Excel** в **.NetFramework** засобами мови **C#** потрібно створити об'єкти з **COM** структури самого **Excel**. А саме:

```
private Microsoft.Office.Interop.Excel.Application ObjExcel;<br>
private Microsoft.Office.Interop.Excel.Workbook ObjWorkBook;<br>
private Microsoft.Office.Interop.Excel.Worksheet ObjWorkSheet;
```

Тепер давайте розглянемо як правильно читати дані з **Excel**. Для цього потрібно правильно відкрити думент і провести ініціалізацію оголошених раніше об'єктів. Експортовані дані, для прикладу, будуть записуватися в табл (**Grid**). І так розглянемо як це все реалізовувати на **C#**:

```
01 //Диалоговое окно выбора файла с фильтром
02 OpenFileDialog openDialog = new OpenFileDialog();
03 openDialog.Filter = "Файл Excel|*.XLSX;*.XLS";
04 openDialog.ShowDialog();
05 try
06 {
07     //Приложение самого Excel
08     ObjExcel = new Microsoft.Office.Interop.Excel.Application();
09     //Книга.
10     ObjWorkBook = ObjExcel.Workbooks.Open(openDialog.FileName);
11     //Таблица.
12     ObjWorkSheet = ObjExcel.ActiveSheet as Microsoft.Office.Interop.Excel.Worksheet;
13     //Ячейка
14     Microsoft.Office.Interop.Excel.Range rg = null;
15
16     Int32 row = 1;
17     dataGridViewMain.Rows.Clear();
18     while (ObjWorkSheet.get_Range("a" + row, "a" + row).Value != null)
19     {
20         // Читаем данные из ячейки
21         rg = ObjWorkSheet.get_Range("a" + row, "c" + row);
22         foreach (Microsoft.Office.Interop.Excel.Range item in rg)
23         {
24             try
```

```
25              {
26                  arr.Add(item.Value.ToString().Trim());
27              }
28              catch { arr.Add(""); }
29          }
30          dataGridViewMain.Rows.Add(arr[0], arr[1], arr[2]);
31          arr.Clear();
32          row++;
33      }
34      MessageBox.Show("Файл успешно считан!", "Считывания excel файла");
35 }
```

```
36 catch (Exception ex) { MessageBox.Show("Ошибка: " + ex.Message, "Ошибка при
   считывании excel файла"); }
```

Як бачите зчитувати дані з файлу **Excel** зовсім не складно! Записувати дані в **Excel** файл ще легше! Для прикладу дані будемо зчитувати з тієї ж таблиці (**Grid**). І так розглянемо як це все реалізовувати на **C#**:

```
fileName = System.Windows.Forms.Application.StartupPath + "\\" + textBoxFileName.Text +
".xlsx";
try
{

    //Приложение самого Excel
    ObjExcel = new Microsoft.Office.Interop.Excel.Application();
    //Книга.
    ObjWorkBook = ObjExcel.Workbooks.Add(System.Reflection.Missing.Value);
    //Таблица.
    ObjWorkSheet = (Microsoft.Office.Interop.Excel.Worksheet)ObjWorkBook.Sheets[1];

    for (int i = 0; i < dataGridViewMain.Rows.Count; i++)
    {
        DataGridViewRow row = dataGridViewMain.Rows[i]; // строки
        for (int j = 0; j < row.Cells.Count; j++) //цикл по ячейкам строки
```

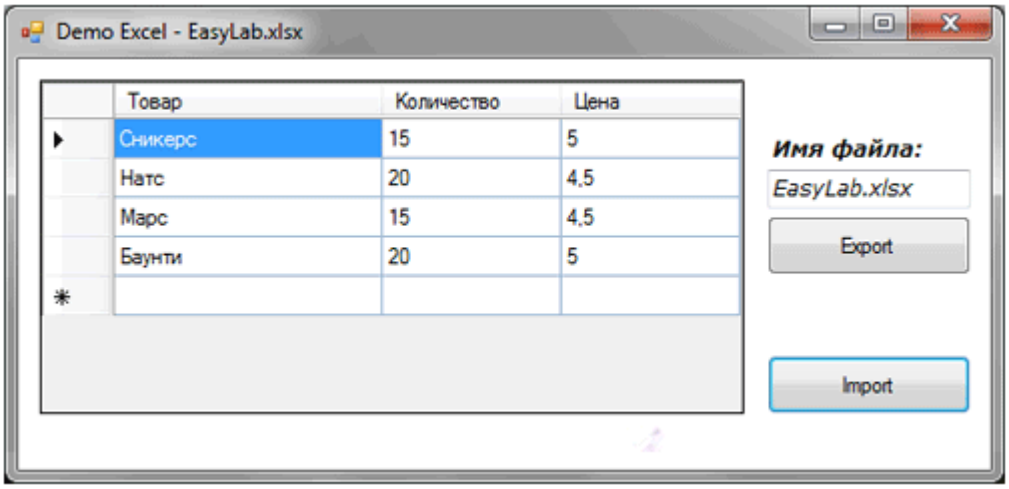
```
        {
            ObjExcel.Cells[i + 1, j + 1] = row.Cells[j].Value;
        }
    }
    ObjWorkBook.SaveAs(fileName);
}
catch (Exception ex) { MessageBox.Show(ex.Message, "Error");}
```

Для коректного завершення роботи з **Excel**, необхідно виконати наступне:

```
01 //Закрытие книгу Excel.
02 ObjWorkBook.Close();
03 //Закрытие приложения Excel.
04 ObjExcel.Quit();
05 //Обнуляем созданые объекты
06 ObjWorkBook = null;
07 ObjWorkSheet = null;
08 ObjExcel = null;
09 //Вызываем сборщик мусора для их уничтожения и освобождения памяти

10 GC.Collect();
```

Приклад роботи з Excel на C#



I. Практична частина

Завдання на лабораторну роботу – реалізувати загрузку та вигризку даних з/в Excel використовуючи Microsoft.Office.Interop.Excel. (вихідні дані для формування файлу взяти з бази сформованої в попередніх лабораторних роботах)

Контрольні запитання:

2. Розкрийте порядок імпорту даних з Excel за допомогою засобів мови C#.
3. Розкрийте порядок експорту даних в Excel за допомогою засобів мови C#.
4. Назвіть основну бібліотеку C#, для організації роботи з Excel

Навчально-методичне видання

Глоба Лариса Сергіївна

Системи розподіленої обробки інформації
в телекомунікаційних мережах
Методичні вказівки
до виконання лабораторних робіт з дисципліни

За редакцією укладачів
Надруковано з оригінал-макету замовника