Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ. ЧАСТИНА 2

Домашня контрольна робота

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітніми програмами «Системи забезпечення споживачів електричної енергією» та
«Енергетичний менеджмент та енергоефективні технології»
спеціальності 141 Електроенергетика, електротехніка та електромеханіка

Укладачі: Д. В. Філянін, І. В. Притискач

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.432.2

Укладачі: *Притискач Іван Васильович*, канд. техн. наук, ст. викладач
Філянін Данило Володимирович, канд. техн. наук, ст. викладач.

Рецензент *Босак А. В.*, канд. техн. наук, доц., доцент кафедри автоматизації електротехнічних та мехатронних комплексів НН ІЕЕ КПІ ім. Ігоря Сікорського

Відповідальний редактор *Ярмолюк О. С.*, канд. техн. наук, доц.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 3 від 09.01.2025 р.)
за поданням вченої ради факультету/навчально-наукового інституту
(протокол № 5 від 27.12.2024 р.)

Обчислювальна техніка та програмування. Частина 2. Домашня контрольна робота [Електронний ресурс] : навч. посіб. для здобувачів ступеня бакалавра за освіт. програмами «Системи забезпечення споживачів електричної енергією» та «Енергетичний менеджмент та енергоефективні технології» спеціальності 141 Електроенергетика, електротехніка та електромеханіка / КПІ ім. Ігоря Сікорського ; уклад.: Д. В. Філянін, І. В. Притискач. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 38 с.

Навчальний посібник з дисципліни «Обчислювальна техніка та програмування. Частина 2. Домашня контрольна робота» містить в собі методичні вказівки та завдання до виконання дамашньої контрольної роботи. В посібнику описано основні можливості розробки графічного інтерфейсу користувача з використанням технології WPF, що базується на мові розширеної розмітки додатків XAML. Видання містить теоретичні відомості про створення простого віконного додатку, роботу з діалоговими вікнами, меню та панелями, використання елемента DataGrid та побудову графіків за допомогою бібліотеки класів OxyPlot.
Призначений для здобувачів ступеня бакалавра за освітньою програмою «Системи забезпечення споживачів електричної енергією» та «Енергетичний менеджмент та енергоефективні технології» спеціальності 141 Електроенергетика, електротехніка та електромеханіка.

УДК 004.432.2

Реєстр. № НП 24/25-209. Обсяг 1,2 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

Зміст

Мета

Познайомитися з основними можливостями розробки графічного інтерфейсу користувача з використанням технології WPF. Отримати досвід створення простих віконних додатків. Вивчити базові елементи мови розмітки XAML.

Теоретичні відомості

1. Створення простого віконного додатку

Windows Presentation Foundation (WPF) система для побудови додатків Windows з візуально привабливими можливостями взаємодії з користувачем, графічна (презентаційна) підсистема у складі .NET Framework.

WPF надає засоби для створення візуального інтерфейсу, включаючи мову XAML (Extensible Application Markup Language), елементи управління, прив'язку даних, макети, двомірну і тривимірну графіку, анімацію, стилі, шаблони, документи, текст, мультимедіа і оформлення.

Побудова інтерфейсів користувача для WPF-додатків здійснюється із застосуванням мови розширеної розмітки додатків XAML. XAML-документ містить розмітку, що описує зовнішній вигляд і поведінку вікна або сторінки додатка, а пов'язані з ним файли коду C# – логіку програми. Мова XAML забезпечує поділ процесу дизайну додатки (графічної частини) і розробки бізнес-логіки (програмного коду) між дизайнерами і розробниками.

XAML базується на мові розширеної розмітки XML (Extensible Markup Language) і його синтаксис визначаються наступними правилами:

- кожен елемент XAML-документа відображається на деякий екземпляр класу .NET. Ім'я такого елемента в точності відповідає імені класу. Наприклад, елемент <*Button*> служить інструкцією для побудови об'єкта класу **Button**;
- елементи XAML можна вкладати один в одного. вкладення елементів розмітки зазвичай відображає вкладеність елементів інтерфейсу;

- властивості класу визначаються за допомогою атрибутів або за допомогою вкладених дескрипторів зі спеціальним синтаксисом.

XAML пропонує дуже просту і ясну схему визначення різних елементів і їх властивостей. Кожен елемент, як і будь-який елемент XML, повинен мати відкритий і закритий тег, як у випадку з елементом *Window*:

```
<Window> </ Window>
```

Або елемент може мати скорочену форму з закриваючим слешем в кінці, наприклад:

```
<Window />
```

Наприклад, створимо просте вікно з кнопкою. Для цього, при створенні нового проекту потрібно у лівій частині діалогового вікна вибрати пункт Visual C#, в правій – пункт WPF Application (рис. 1).

Після натискання кнопки "ОК" буде сформований шаблон проекту. При цьому згенерується наступний XAML-документ:

```
<Window x:Class="WPFTest_1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
    <Grid>

    </Grid>
</Window>
```

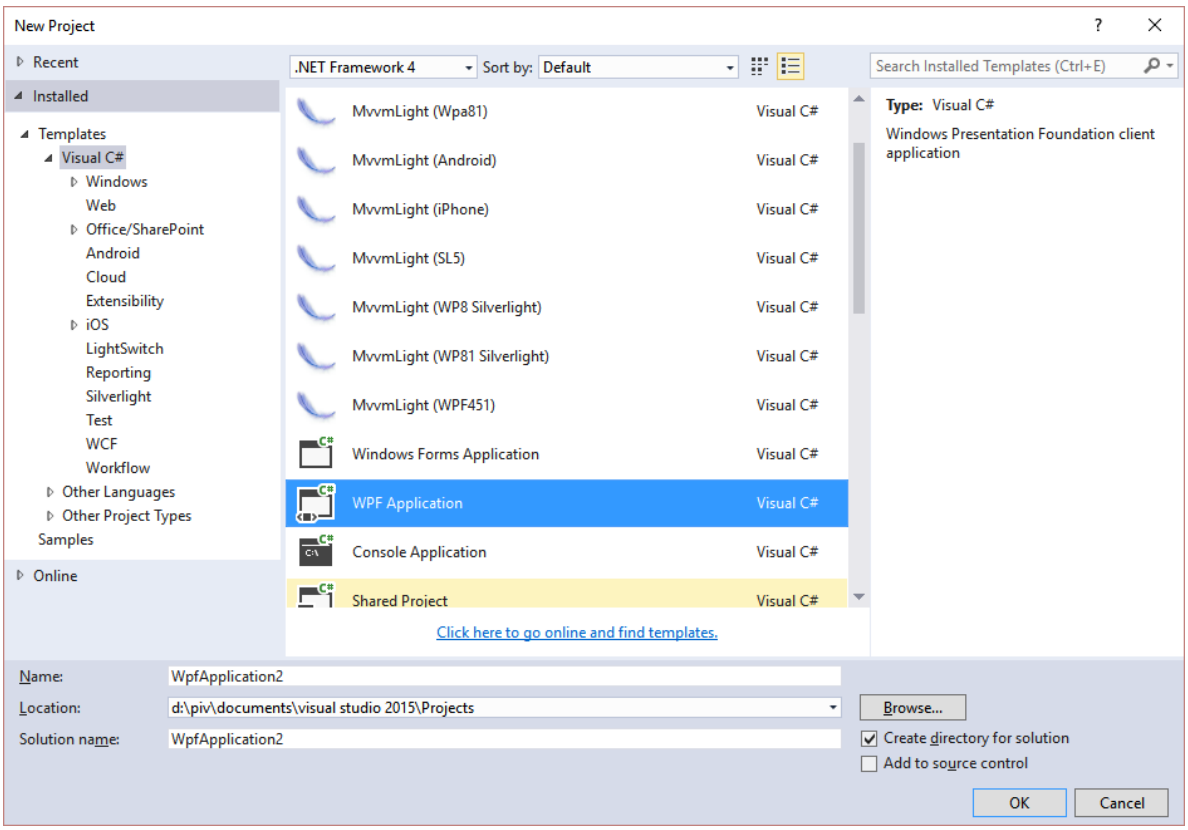

Рис. 1 – Створення проекту

Дизайнер проекту наведено на рис. 2.

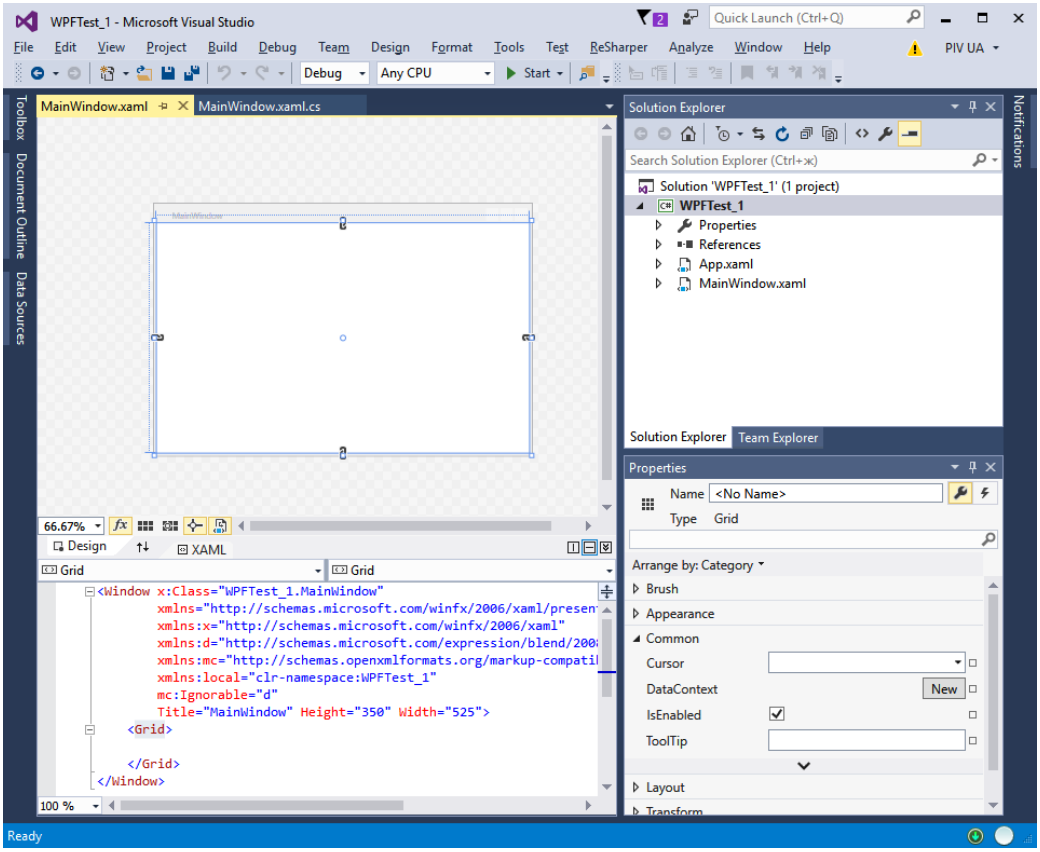

Рис. 2 – Дизайнер проекту

Спочатку йде елемент самого вищого рівня – *Window*, потім йде вкладений елемент *Grid* – контейнер для інших елементів, і в ньому вже визначимо елемент *Button*, який представляє кнопку. Для створення кнопки додаємо такий код між тегами <*Grid*> </*Grid*>:

```
<Button x:Name = "firstButton" Width = "300" Height = "70"
Content = "Натисни мене" />
```

Для кнопки ми можемо визначити властивості у вигляді атрибутів. Тут визначені атрибути *x:Name* (ім'я кнопки), *Width, Height* і *Content*. Подібним чином ми можемо визначити і інші атрибути, які нам потрібні.

При створенні нового проекту WPF на додаток до створюваного файлу *MainWindow.xaml* створюється також файл відокремленого коду *MainWindow.xaml.cs*, де має міститися логіка програми, пов'язана з розміткою. Файли XAML дозволяють нам визначити інтерфейс вікна, але для створення логіки програми все одно доведеться користуватися кодом C #.

У програмі часто потрібно звернутися до якогось елементу управління. Для цього треба визначити у елемента в XAML властивість *Name*.

Ще однією точкою взаємодії між XAML і C# є події. За допомогою атрибутів в XAML ми можемо поставити події, які будуть пов'язані з обробниками в коді C#.

Додаймо до попереднього проекту можливість реагувати на натискання кнопки. Для цього розмістимо на формі три текстових поля і надаймо їм імена (*Name*) *X, Y, Result*. Отримаємо такий код XAML:

```
<Window x:Class="WPFTest_1.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"
        mc:Ignorable="d"
        Title="MainWindow" Height="350" Width="525">
```

```
    <Grid>
        <TextBox x:Name = "X" Width = "150" Height = "30"
VerticalAlignment = "Top" Margin = "20" Text="0" />
        <TextBox x:Name = "Y" Width = "150" Height = "30"
VerticalAlignment = "Top" Margin = "55" Text="0" />
        <TextBox x:Name = "Result" Width = "150" Height = "30"
VerticalAlignment = "Top" Margin = "90" />
        <Button x:Name = "firstButton" Width = "300" Height = "70"
Content = "Натисни мене" Click="firstButton_Click" />
    </Grid>
</Window>
```

Атрибут *Margin* у цьому випадку задає відступ від верхньої межі вікна. Атрибут кнопки

```
     Click="firstButton_Click"
```

задає назву обробника її натискання. В обробнику задаються дії, які потрібно виконати при натискання кнопки.

Змінимо файл коду, додавши в нього обробник натискання кнопки:

```
using System.Windows;

namespace WPFTest_1
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void firstButton_Click(object sender, RoutedEventArgs
e)
        {
            var a = double.Parse(X.Text);
            var b = double.Parse(Y.Text);
            var res = a*a + b*b;
            Result.Text = res.ToString();
        }
    }
}
```

Визначивши імена елементів в XAML, потім ми можемо до них звертатися в коді C#:

```
Result.Text = res.ToString();
```

У обробнику натиснення кнопки просто обчислюємо суму квадратів двох чисел в перших полях і виводимо результат у третє текстове поле.

Розміщення різних елементів інтерфейсу виконується за допомогою компонування (layout). Завдяки компонуванню ми можемо зручним чином налаштувати елементи інтерфейсу та позиціонувати їх. Наприклад, елементи компоновки в WPF дозволяють при ресайзі – стисканні або розтягуванні – дуже зручно автоматично масштабувати елементи.

У WPF компоновка здійснюється за допомогою спеціальних контейнерів. Фреймворк надає нам такі контейнери: *Grid, UniformGrid, StackPanel, WrapPanel, DockPanel і Canvas*. Різні контейнери можуть містити в собі інші контейнери.

Grid це найбільш потужний і часто використовуваний контейнер, що нагадує звичайну таблицю. Він містить стовпці і рядки, кількість яких задає розробник. Для визначення рядків використовується властивість *RowDefinitions*, а для визначення стовпців – властивість *ColumnDefinitions*:

```
<Grid.RowDefinitions>
   <RowDefinition> </ RowDefinition>
   <RowDefinition> </ RowDefinition>
   <RowDefinition> </ RowDefinition>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
   <ColumnDefinition> </ ColumnDefinition>
   <ColumnDefinition> </ ColumnDefinition>
   <ColumnDefinition> </ ColumnDefinition>
</Grid.ColumnDefinitions>
```

Кожен рядок задається за допомогою вкладеного елемента *RowDefinition.* При цьому задавати додаткову інформацію необов'язково. Кожен стовпець задається за допомогою вкладеного елемента *ColumnDefinition*.

Щоб задати позицію елемента управління з прив'язкою до певної комірки *Grid*, в розмітці елемента потрібно прописати значення властивостей *Grid.Column* і *Grid.Row*, тим самим вказуючи, в якому стовпці і рядку буде знаходитися елемент.

```
<Button Grid.Column = "0" Grid.Row = "0" Content =
"Рядок 0 Стовпець 0" />
<Button Grid.Column = "2" Grid.Row = "2" Content =
"Рядок 2 Стовпець 2" />
```

Є кілька варіантів настройки розмірів комірок Grid.

Автоматичні розміри. Тут стовпець або рядок займає те місце, яке їм потрібно

```
<ColumnDefinition Width = "Auto" />
<RowDefinition Height = "Auto" />
```

Абсолютні розміри. В даному випадку висота і ширина вказуються в одиницях, незалежних від пристрою:

```
<ColumnDefinition Width = "150" />
<RowDefinition Height = "150" />
```

Також абсолютні розміри можна задати в пікселях, дюймах, сантиметрах або точках: пікселі (px), дюйми (in), сантиметри (cm), точки (pt). наприклад,

```
<ColumnDefinition Width = "1 in" />
<RowDefinition Height = "10 px" />
```

Пропорційні розміри. Наприклад, нижче задаються два стовпці, другий з яких має ширину в чверть від ширини першого:

```
<ColumnDefinition Width = "*" />
<ColumnDefinition Width = "0.25*" />
```

Якщо рядок або стовпець має висоту, рівну *, то даний рядок або стовпчик буде займати все місце, що залишилося. Якщо у нас є кілька рядків або стовпців, висота яких дорівнює *, то все доступне місце ділиться порівну між усіма такими термінами і стовпцями. Використання коефіцієнтів (0.25*) дозволяє зменшити або збільшити виділене місце на даний коефіцієнт. При цьому всі коефіцієнти складаються (коефіцієнт * аналогічний 1*) і потім весь простір ділиться на суму коефіцієнтів.

Щоб взаємодіяти з користувачем, отримувати від користувача введення з клавіатури або миші і використовувати введені дані в програмі, використовуються елементи управління.

Всі елементи управління можуть бути умовно розділені на кілька підгруп:

- Елементи керування вмістом, наприклад кнопки (*Button*), мітки (*Label*).
- Спеціальні контейнери, які містять інші елементи, але на відміну від елементів *Grid* або *Canvas* не є контейнерами компонування – ScrollViewer, GroupBox.
- Декоратори, чиє призначення створення певного фону навколо вкладених елементів, наприклад, *Border* або *Viewbox*.
- Елементи управління списками, наприклад, *ListBox, ComboBox*.
- Текстові елементи управління, наприклад, *TextBox, RichTextBox*.
- Елементи, засновані на діапазонах значень, наприклад, *ProgressBar, Slider*.
- Елементи для роботи з датами, наприклад, *DatePicker* і *Calendar*.

- Інші елементи управління, які не увійшли в попередні підгрупи, наприклад, *Image*.

Розглянемо деякі з основних властивостей, які є характерними практично для усіх елементів управління.

Name. За визначеним іменем можна звертатися до елементу як в коді програми, так і в XAML розмітці. Приклад використання вже був показаний вище.

Visibility. Ця властивість встановлює параметри видимості елемента і може приймати одне з трьох значень:

- *Visible* – елемент видно і він бере участь в компонуванні.
- *Collapsed* елемент не видно і він не бере участь в компонуванні.
- *Hidden* елемент е видно, але при цьому бере участь в компонуванні.

Властивості шрифтів:

- *FontFamily* визначає сімейство шрифту (наприклад, *Arial, Verdana* та т. д.)
- *FontSize* визначає висоту шрифту
- *FontStyle* визначає нахил шрифту, приймає одне з трьох значень - *Normal, Italic, Oblique*.
- *FontWeight* визначає товщину шрифту і приймає ряд значень, як *Black, Bold* та ін.
- *FontStretch* визначає, як буде розтягувати або стискати текст, наприклад, значення *Condensed* стискає текст, а *Expanded* розтягує.

Кольори фону і шрифту. Властивості *Background* і *Foreground* задають відповідно колір фону і тексту елемента управління.

Найпростіший спосіб завдання кольору в коді XAML:

```
Background = "DarkRed"
```

Як значення властивість *Background* (*Foreground*) може приймати запис у вигляді шістнадцяткового значення в форматі #rrggbb, де rr - червона складова, bb - зелена складова, а bb - синя. Або можна використовувати назви кольорів безпосередньо: *Red, Green, LightGray*.

2. Діалогові вікна, меню та панелі

У WPF доступні для використання декілька стандартних діалогових вікон, таких як **OpenFileDialog** і **SaveFileDialog**. Робота з будь-яким з цих діалогових вікон зводиться до створення об'єкта відповідного класу і викликом методу ShowDialog (). В класах визначені різні члени, що дозволяють встановлювати фільтри файлів і шляхи до каталогів і отримувати доступ до вибраних користувачем файлів.

Розглянемо такий приклад:

```
using System.Windows;
using System.IO;
using Microsoft.Win32;

namespace WPFTest_1
{
    public partial class Window1 : Window
    {
        public Window1()
        {
            InitializeComponent();
        }

        private void SaveBtn_Click(object sender, RoutedEventArgs e)
        {
            var saveDlg = new SaveFileDialog();
            saveDlg.Filter = "Text Files | * .txt";
            if (true == saveDlg.ShowDialog())
            {
                // Зберегти дані з TextBlock в зазначеному файлі
                File.WriteAllText(saveDlg.FileName, text.Text);

            }
```

```
        }

        private void OpenBtn_Click(object sender, RoutedEventArgs e)
        {
            // Створити діалогове вікно відкриття файлу, що відображає
тільки текстові файли
            var openDlg = new OpenFileDialog();
            openDlg.Filter = "Text Files | * .txt";
            if (true == openDlg.ShowDialog())
            {
                // Завантажити вміст вибраного файлу
                string dataFromFile =
File.ReadAllText(openDlg.FileName);
                // Показати рядок в TextBlock
                text.Text = dataFromFile;
            }
        }
    }
}
```

```
<Window x:Class="WPFTest_1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"
        mc:Ignorable="d"
        Title="Window1" Height="300" Width="500">
    <DockPanel>
        <StackPanel>
            <Button x:Name="OpenBtn" Content="Відкрити" Height="50"
Click="OpenBtn_Click"></Button>
            <Button x:Name="SaveBtn" Content="Зберегти" Height="50"
Click="SaveBtn_Click"/>
            <TextBox x:Name="text" TextWrapping="Wrap"
AcceptsReturn="True"></TextBox>
        </StackPanel>
    </DockPanel>
</Window>
```

Як бачите, для роботи додатку необхыдно імпортувати простори імен **System.IO** і **Microsoft.Win32** в файл коду.

У головному вікні програми створено дві кнопки і текстове поле. При натисканні на кнопку «Відкрити» відкривається діалогове вікно вибору файлу. Вміст вибраного файлу відображається у текстовому полі. При натисканні кнопки «Зберегти» вміст текстового поля записується у текстовий файл.

Створення меню в WPF відбувається за допомогою класу **Menu**, який підтримує колекцію об'єктів **MenuItem**. При побудові меню в XAML кожен **MenuItem** може обробляти різні події, з яких найважливішою є **Click**, що виникає, коли користувач вибирає елементи.

Додамо до попереднього прикладу наступну розмітку в контекст елемента **DockPanel**:

```
<Window x:Class="WPFTest_1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"
        mc:Ignorable="d"
        Title="Window1" Height="300" Width="500">
    <DockPanel>
        <Menu DockPanel.Dock="Top" HorizontalAlignment = "Left" Background =
"White" BorderBrush = "Black">
            <MenuItem Header = "_Файл">
                <MenuItem Header = "_Відкрити" Click="OpenBtn_Click" />
                <MenuItem Header = "_Зберегти" Click="SaveBtn_Click" />
                <Separator />
                <MenuItem Header = "_Вихід" Click="MenuItem_Click"/>
            </MenuItem>
            <MenuItem Header = "_Довідка">
                <MenuItem x:Name="AboutMenuItem" Header = "_Про програму"
Click="AboutMenuItem_Click" />
            </MenuItem>
        </Menu>
        <StackPanel DockPanel.Dock="Bottom">
            <Button x:Name="OpenBtn" Content="Відкрити" Height="50"
Click="OpenBtn_Click"></Button>
            <Button x:Name="SaveBtn" Content="Зберегти" Height="50"
Click="SaveBtn_Click"/>
            <TextBox x:Name="text" TextWrapping="Wrap"
AcceptsReturn="True"></TextBox>
        </StackPanel>
    </DockPanel>
</Window>
```

Зверніть увагу, що меню стикуватися до верхньої частини **DockPanel**.

```
DockPanel.Dock="Top"
```

Крім того, елемент **Separator** використовується для додавання в систему меню тонкої горизонтальної лінії безпосередньо перед пунктом *Вихід*. Значення **Header** для кожного **MenuItem** містять символ підкреслення

(наприклад, *_Вихід*). Так вказується символ, який стане підкресленим, коли користувач натисне клавішу <Alt> (для введення клавіатурного скорочення).

Після побудови системи меню необхідно реалізувати різні обробники подій. Перш за все, знадобиться обробник пункту меню *Файл - Вихід*, який просто закриває вікно, що, в свою чергу, призводить до завершення програми, оскільки це вікно самого вищого рівня.

Нижче показаний код доданих обробників подій:

```
private void MenuItem_Click(object sender, RoutedEventArgs e)
{
     Close();
}
private void AboutMenuItem_Click(object sender, RoutedEventArgs e)
{
     MessageBox.Show("Приклад програми");
}
```

Панелі інструментів створюються за допомогою класу **ToolBar,** і зазвичай забезпечують альтернативний спосіб активізації пунктів меню. Перемістимо раніше додаеі кнопки «Відкрити» та «Зберегти» на панель інструментів. Для цього модифікуємо код розмітки головного вікна таким чином:

```
<DockPanel>
        <Menu DockPanel.Dock="Top" HorizontalAlignment = "Left" Background =
"White" BorderBrush = "Black">
            <MenuItem Header = "_Файл">
                <MenuItem Header = "_Відкрити" Click="OpenBtn_Click" />
                <MenuItem Header = "_Зберегти" Click="SaveBtn_Click" />
                <Separator />
                <MenuItem Header = "_Вихід" Click="MenuItem_Click"/>
            </MenuItem>
            <MenuItem Header = "_Довідка">
                <MenuItem x:Name="AboutMenuItem" Header = "_Про програму"
                  Click="AboutMenuItem_Click" />
            </MenuItem>
        </Menu>
        <ToolBar DockPanel.Dock="Top">
            <Button x:Name="OpenBtn" Content="Відкрити"
                  Click="OpenBtn_Click"></Button>
            <Button x:Name="SaveBtn" Content="Зберегти"
                  Click="SaveBtn_Click"/>
        </ToolBar>
        <TextBox x:Name="text" TextWrapping="Wrap"
AcceptsReturn="True"></TextBox>
</DockPanel>
```

Елемент управління **TabControl** відображає набір пов'язаних елементів управління WPF у декільеох вкладках представлених елементами **TabItem**. Два елементи типу вкладок надаються автоматично. Для додавання додаткових вкладок потрібно просто натиснути правою кнопкою миші на вузлі **TabControl** у вікні *Document Outline* і вибрати в контекстному меню пункт *Add TabItem*; можна також натиснути правою кнопкою миші на самому елементі **TabControl** в візуальному конструкторі і вибрати аналогічний пункт меню. Кожен елемент управління **TabItem** містить властивість **Header**, що відповідає за заголовок вкладки. При виборі вкладки для редагування ця вкладка стає активною, і її вміст можна компонувати, перетягуючи елементи управління з панелі інструментів.

Додамо до раніше створеної програми дві вкладки. У першій буде відображатися текстове поле для редагування, а у другому інформація про текст: кількість символів та кількість слів.

```
<Window x:Class="WPFTest_1.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"
        mc:Ignorable="d"
        Title="Window1" Height="300" Width="500">
    <DockPanel>
        <Menu DockPanel.Dock="Top" HorizontalAlignment = "Left" Background = "White"
BorderBrush = "Black">
            <MenuItem Header = "_Файл">
                <MenuItem Header = "_Відкрити" Click="OpenBtn_Click" />
                <MenuItem Header = "_Зберегти" Click="SaveBtn_Click" />
                <Separator />
                <MenuItem Header = "_Вихід" Click="MenuItem_Click"/>
            </MenuItem>
            <MenuItem Header = "_Довідка">
                <MenuItem x:Name="AboutMenuItem" Header = "_Про програму"
Click="AboutMenuItem_Click" />
            </MenuItem>
        </Menu>
        <ToolBar DockPanel.Dock="Top">
            <Button x:Name="OpenBtn" Content="Відкрити" Click="OpenBtn_Click"></Button>
            <Button x:Name="SaveBtn" Content="Зберегти" Click="SaveBtn_Click"/>
        </ToolBar>
        <TabControl>
            <TabItem Header="Текст">
                <TextBox x:Name="text" TextWrapping="Wrap"
AcceptsReturn="True"></TextBox>
            </TabItem>
            <TabItem Header="Статистика">
                <StackPanel>
                    <Label x:Name="LettersCount" ></Label>
```

```
                    <Label x:Name="WordCount" ></Label>
                </StackPanel>
            </TabItem>
        </TabControl>
    </DockPanel>
</Window>
```

```
        private void OpenBtn_Click(object sender, RoutedEventArgs e)
        {
            var openDlg = new OpenFileDialog();
            openDlg.Filter = "Text Files | * .txt";
            if (true == openDlg.ShowDialog())
            {
                // Завантажити вміст вибраного файлу
                string dataFromFile = File.ReadAllText(openDlg.FileName);
                // Показати рядок в TextBlock
                text.Text = dataFromFile;
                LettersCount.Content = "Кількість символів:\t"+dataFromFile.Length;
                WordCount.Content = "Кількість слів:\t\t" +
                                    dataFromFile.Split(new char[] {' ', '\r', '\n'},
                                        StringSplitOptions.RemoveEmptyEntries).Length;
            }
        }
```

Зображення вікна остаточного варіанту програми показане на рис. 3.

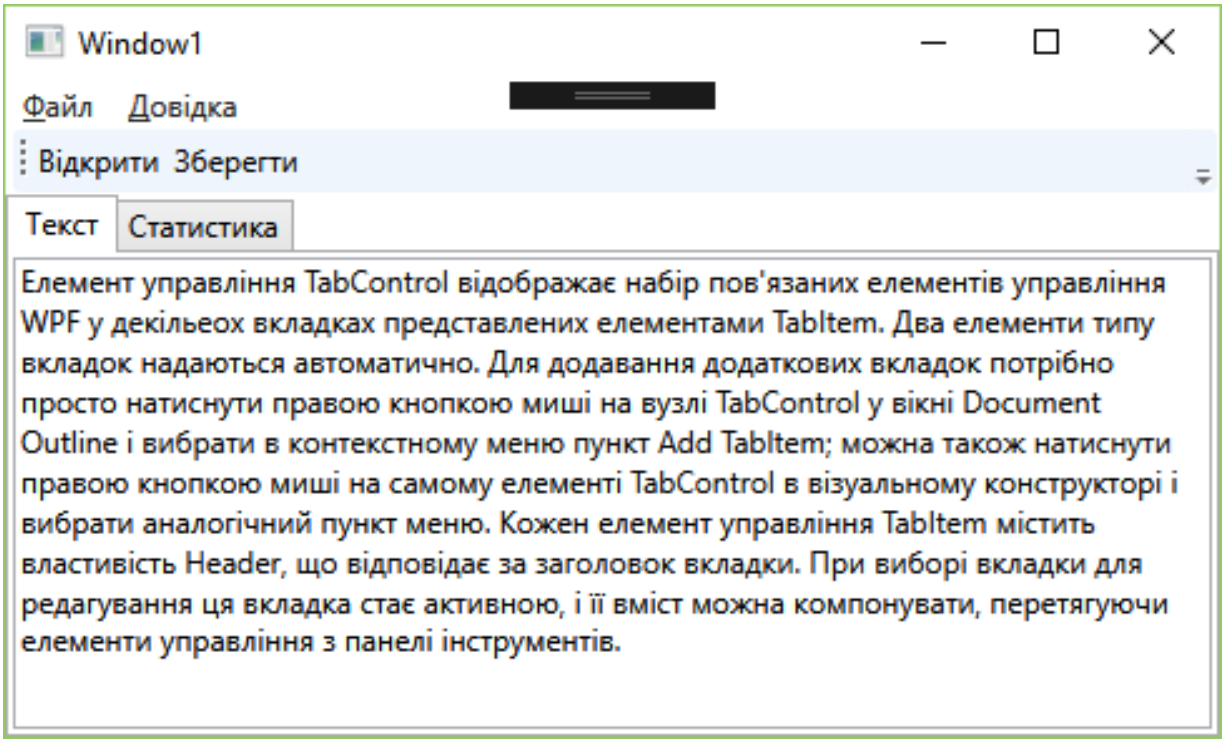

Рис. 3 – Зображення вікна програми

3. Елемент DataGrid

DataGrid – гнучкий елемент управління для відображення даних у вигляді таблиці з декількома стовпцями, що допускає сортування, редагування та інші дії.

Основні типи стовпців, що підтримуються елементом **DataGrid**:

- **DataGridTextColumn** – для відображення рядків; в звичайному режимі використовується елемент **TextBlock**, а в режимі редагування – елемент **TextBox**;
- **DataGridHyperlinkColumn** – представляє звичайний текст в вигляді гіперпосилання;
- **DataGridCheckBoxColumn** – для відображення логічних значень; використовується елемент **CheckBox**, який у позначеному стані відповідає значенню **true**, а в скинутому – значенню **false**.
- **DataGridComboBoxColumn** – для відображення перерахувань; в звичайному режимі використовується елемент **TextBlock**, а в режимі редагування - елемент **ComboBox**, що містить можливі значення.
- **DataGridTemplateColumn** – дозволяє задати довільні шаблони для відображення значення в звичайному режимі і в режимі редагування. Робиться це за допомогою властивостей **CellTemplate** і **CellEditingTemplate**.

Елемент **DataGrid** автоматично підтримує зміну порядку і розміру стовпців і сортування за стовпцями, але будь-яку можливість можна відключити, встановивши значення **false** для таких властивостей: **CanUserReorderColumns**, **CanUserResizeColumns**, **CanllserResizeRows** і **CanUserSortColumns**. Властивості **GridLinesVisibility** і **HeadersVisibility** дозволяють відключити показ ліній сітки і заголовків відповідно.

Якщо об'єкти, які відображаються в елементі **DataGrid**, задаються за допомогою властивості **ItemsSource**, то **DataGrid** намагається автоматично згенерувати стовпці, що відповідають властивостям класу об’єктів. У такому випадку для відображення рядків вибирається стовпець типу **DataGridTextColumn**, для відображення URI – стовпець типу **DataGridHyperlinkColumn**, для відображення логічних величин – стовпець типу **DataGridCheckBoxColumn**, а для відображення перерахувань – стовпець типу **DataGridComboBoxColumn**.

Розглянемо приклад програми, в якій за допомогою елемента **DataGrid** буде відображатися колекція об’єктів класу *Lamp*, що описаний у попередніх практичних роботах. Розмітка вікна надзвичайно проста і містить тільки порожній елемент **DataGrid**:

```
    <Grid>
        <DataGrid x:Name="lampGrid">

        </DataGrid>
    </Grid>
```

Встановимо властивість ItemsSource елемента з назвою `lampGrid` в наступному коді:

```
        public Window2()
        {
            InitializeComponent();

            var lampList = new List<Lamp>();
            lampList.Add(new Lamp("СЛ-21", 15, 220));
            lampList.Add(new Lamp("ПЛ-345", 75, 220));
            lampList.Add(new Lamp("РПС", 6, 12));
            lampList.Add(new Lamp("ДМ45", 45, 24));
            lampList.Add(new Lamp("Е100", 100, 220));

            lampGrid.ItemsSource = lampList;
        }
```

Єдиний візуальний недолік автоматичної генерації стовпців – це назви в заголовках, які збігаються з іменами відповідних властивостей. Результат показаний на рис. 4.

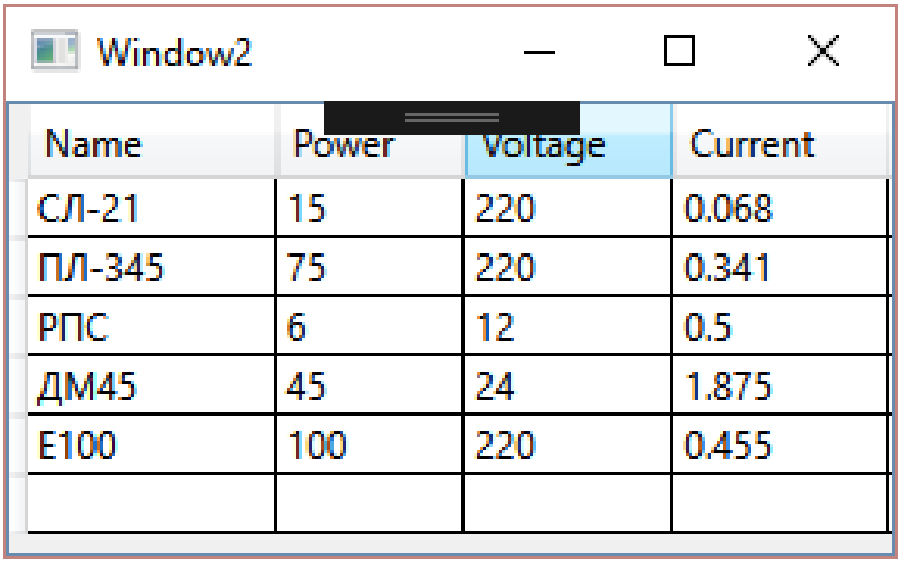

Рис. 4 – Елемент **DataGrid** з автоматично згенерованими стовпцями

DataGrid на рис. 9.1 автоматично підтримує редагування всіх полів кожного елемента. Варто клацнути по будь-якій клітинці, і вона автоматично перетворюється в поле введення **TextBox**. Результат будь-якого завершеного редагування відображається в колекції **ItemsSource**.

Якщо в елементі **DataGrid** присутні явно визначені стовпці, то автоматично згенеровані стовпці додаються після них. Окремі автоматично згенеровані стовпці можна налаштувати або видалити, обробивши подію **AutoGeneratingGolumn**, яке виникає один раз для кожного стовпчика. Після генерації всіх стовпців один раз виникає подія **AutoGeneratedColumns**.

При використанні властивості **ItemsSource** автоматично підтримується редагування даних в окремих об'єктах. Оскільки колекція **ItemsSource** допускає також додавання і видалення об'єктів, то автоматично підтримуються і ці операції. В сітці **DataGrid** внизу присутній порожній рядок, в який можна додати дані. У класі **DataGrid** визначені методи і команди для таких типових дій, як початок редагування (клавіша F2), скасування редагування (клавіша Esc), збереження результатів редагування (клавіша Enter) і видалення рядка (клавіша Delete).

Для запобігання редагування можна присвоїти властивості **IsReadOnly** значення **true**, а щоб заборонити додавання або видалення рядків, потрібно

присвоїти значення **false** властивості **CanUserAddRows** або **CanUserDeleteRows** відповідно.

Щоб скасувати автоматичну генерацію стовпців, потрібно присвоїти властивості **AutoGenerateColumns** значення **false**.

Елемент **DataGrid** підтримує кілька моделей вибору за допомогою двох властивостей – **SelectionMode** і **SelectionUnit**. Властивості **SelectionMode** можна присвоїти значення **Single** – тоді дозволено вибирати тільки один об'єкт, або значення **Extended** – в цьому випадку можна вибирати кілька об'єктів (це стандартний режим). Визначення слова «об'єкт» залежить від значення властивості **SelectionUnit**:

- **Cell** – дозволено вибирати тільки окремі клітинки;
- **FullRow** – дозволено вибирати тільки цілі рядки;
- **CellOrRowHeader** – дозволено вибирати і те і інше (для вибору всього рядку слід клацнути по його заголовку).

У режимі вибору декількох об'єктів натискання клавіші Shift дозволяє вибирати сусідні об'єкти, а, натискання клавіші Ctrl – довільно розташовані об'єкти.

При виборі рядків генерується подія **Selected**, а властивість **SelectedItems** містить колекцію обраних об'єктів. При виборі окремих клітинок генерується подія **SelectedCellChanged**, а властивість **SelectedCells** містить інформацію про відповідні рядки і стовпці.

Клас **DataGrid** підтримує і інші дії, наприклад, взаємодію з буфером обміну та можливість «заморожувати» стовпці.

Взаємодія з буфером обміну. Налаштувати, які саме дані копіюються з **DataGrid** в буфер обміну (наприклад, при натисканні Ctrl + C після вибору об'єктів), дозволяє властивість **ClipboardCopyMode**. Вона може приймати наступні значення:

- **ExcludeHeader** – не включати заголовки стовпців в текст, що копіюється. Це режим за замовчуванням;
- **IncludeHeader** – включати заголовки стовпців в текст, що копіюється;
- **None** – нічого не копіювати в буфер обміну.

Заморожування стовпців. Елемент **DataGrid** дозволяє заморозити будь-яке число стовпців. Це означає, що вони не будуть висунуті за межі області елемента при горизонтальній прокрутці. Приблизно так само заморозка стовпців працює в Microsoft Excel. Щоб заморозити один або декілька стовпців, досить присвоїти властивості **FrozenColumnCount** будь-яке значення, більше 0.

4. Побудова графіків за допомогою бібліотеки класів OxyPlot

В додатках написаних мовою C# є можливість використовувати сторонні класи зібрані у скомпільовані бібліотеки класів.

Існує дуже велика кількість готових бібліотек з відкритим вихідним кодом, що написані програмістами зі всього світу, і які призначені для вирішення найрізноманітніших задач. Ці бібліотеки вільно поширюються в мережі Internet і найзручнішим засобом для їх використання є пакетний менеджер NuGet.

Для використання класів необхідно перш за все додати у проект посилання на збірку, яка містить переносну бібліотеку класів. Якщо бібліотеку додавати через менеджер NuGet то практично всі дії виконуються автоматично. Для цього можна, наприклад, в браузері рішень клацнути по пункту «References» правою кнопкою по назві проекту і вибрати пункт меню «Manage NuGet Packages...».

Далі потрібно в пошуковому полі ввести ім’я бібліотеки або ключові слова для пошуку і з отриманих результатів вибрати бажаний. Додавання бібліотеки відбувається після натискання кнопки «Install».

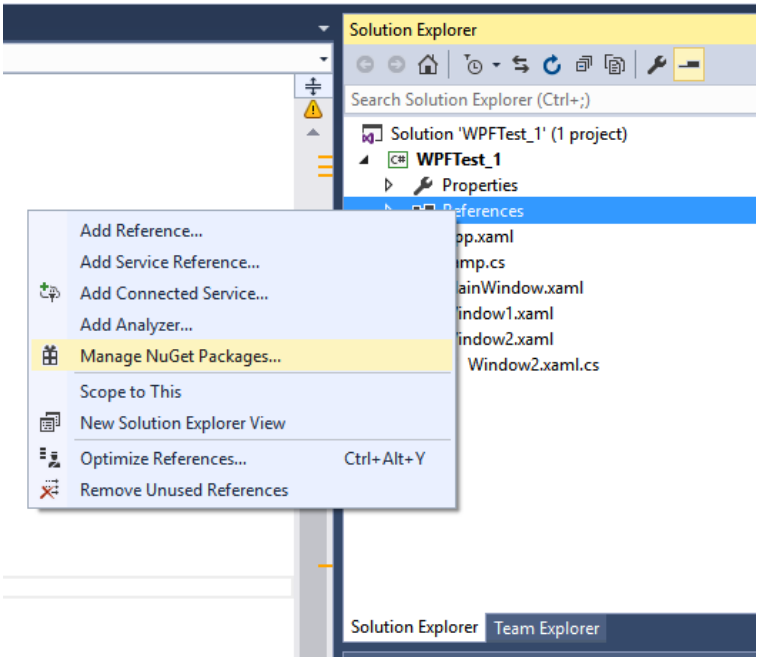

Рис. 5 – Додавання бібліотеки класів через менеджер NuGet

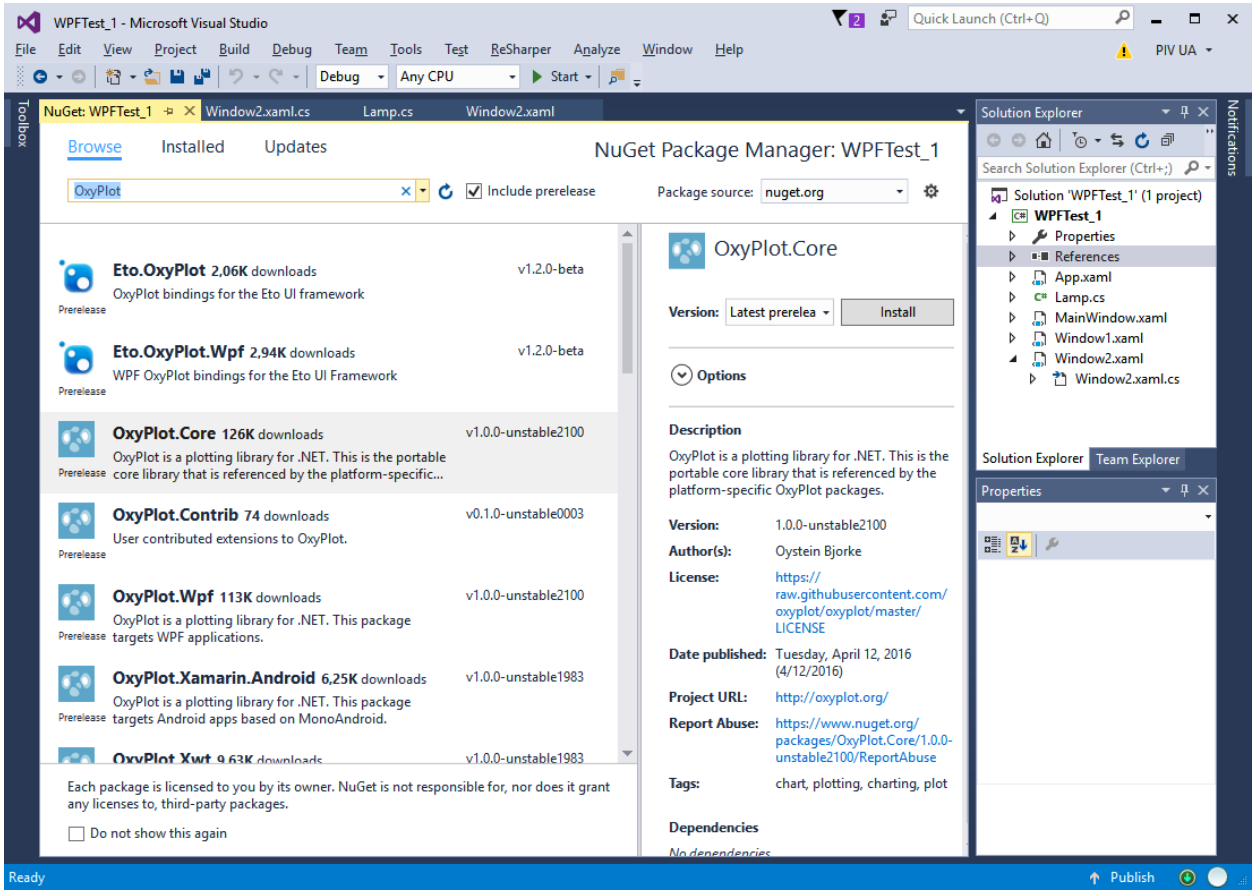

Рис. 6 – Пошук бібліотеки класів через менеджер NuGet

Для додавання бібліотеки вручну, спершу викличемо контекстне меню на пункті «References», і виберемо в ньому «Add Reference…», як показано на малюнку нижче.

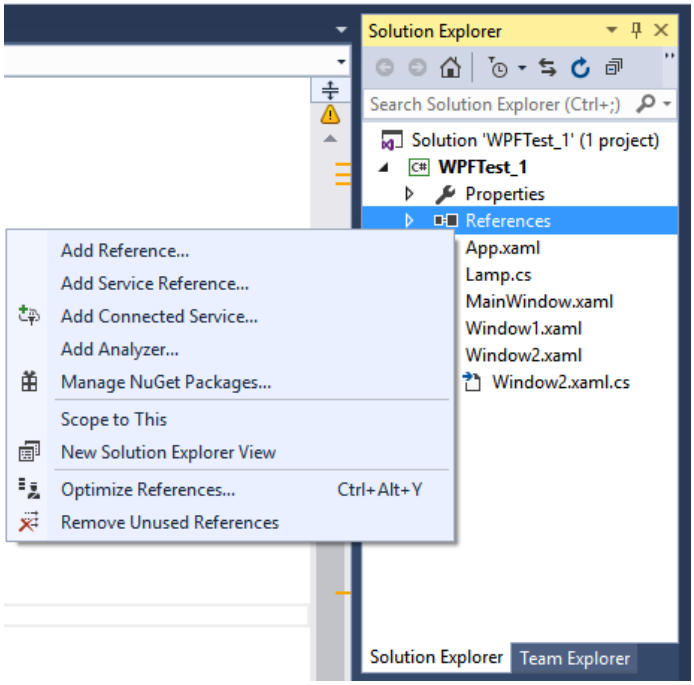

Рис. 7 – Додавання бібліотеки класів

У вікні в лівій області вибираємо пункт «Browse» і в низу вікна, натискаємо на кнопку «Browse…», як показано на малюнку нижче.

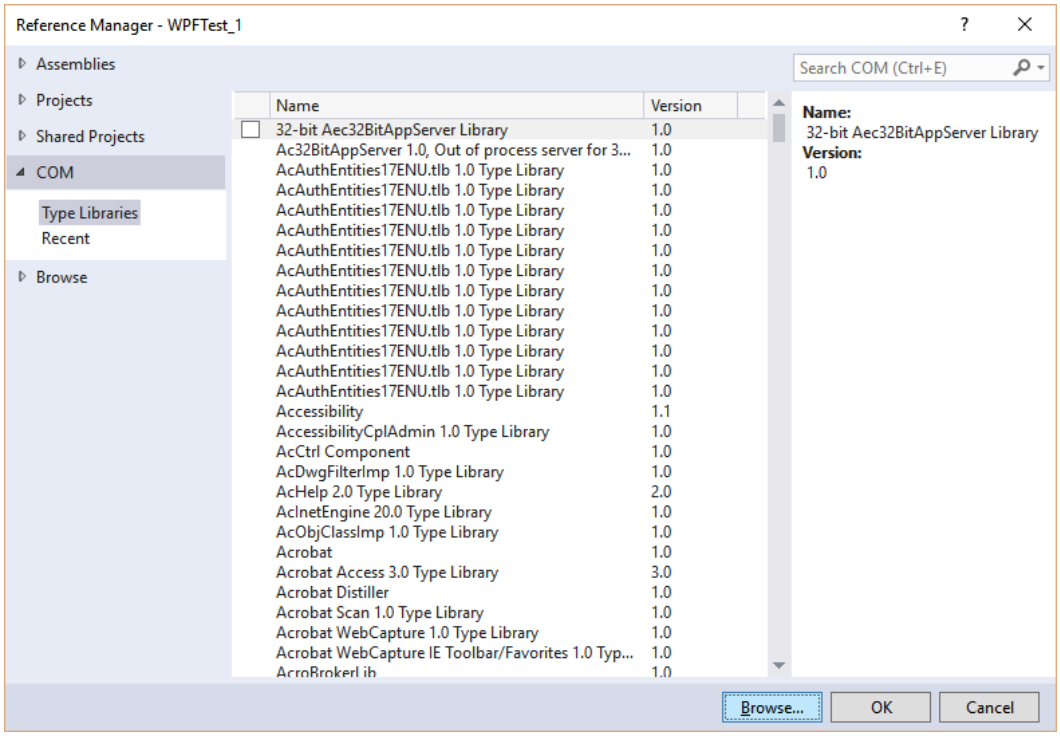

Рис. 8 – Додавання бібліотеки класів

У вікні потрібно перейти в папку, в яке лежить бібліотека (DLL), вибрати цю бібліотеку і натиснути на кнопку «Додати». Після чого, натиснути на кнопку «ОК» в попередньому вікні. В результаті, список посилань проекту, буде поповнено ще однією бібліотекою. Тепер, ми можемо використовувати в програмі класи з підключеної бібліотеки. Проте перед цим потрібно підключити простір імен. Для цього, додаємо рядок

```
using <назва_простору_імен>;
```

в кінець блоку директив **using**, який розташований в самому початку основного файлу проекту.

Використання бібліотек класів розглянемо на прикладі бібліотеки для побудов графіків **OxyPlot**. **OxyPlot** містить багато різних типів графіків, осей, рядів тощо. Всі побудовані графіки можуть бути легко експортовані у різні формати файлів, такі як PNG, PDF і SVG.

Для створення графіку у вікні додатку потрібно додати простір імен і відповідний компонент WPF, який називається **PlotView** (виділено напівжирним шрифтом):

```
<Window x:Class="WPFTest_1.Window3"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-
compatibility/2006"
        xmlns:local="clr-namespace:WPFTest_1"

        xmlns:oxy=http://oxyplot.org/wpf

        mc:Ignorable="d"
        Title="Window3" Height="300" Width="300">
    <Grid>
        <oxy:PlotView x:Name="LampPlot" />
    </Grid>
</Window>
```

У коді програми потрібно створити об’єкт класу **PlotModel** та присвоїти його властивості **Model** компонента **PlotView**:

```
      var LampModel = new PlotModel();
      LampPlot.Model = LampModel;
```

Подальша робота ведеться з об’єктом класу **PlotModel** через відповідні властивості та методи. Наприклад, назву графіку можна задати через властивість **Title**. Ряди даних зберігаються в колекції **Series** і можуть бути додані через метод **Add**(). Для прикладу, на основі раніше створеного списку побудуємо залежність струму від потужності для різних типів ламп:

```
      var LampModel = new PlotModel();
      LampPlot.Model = LampModel;

      LampModel.Title = "Залежність струму від потужності для різних
типів ламп";

      var lampSeries = new LineSeries();
      foreach (var lamp in lampList)
      {
       lampSeries.Points.Add(new DataPoint(lamp.Power, lamp.Current));
      }
      LampModel.Series.Add(lampSeries);
```

Результат зображено на рис. 9.

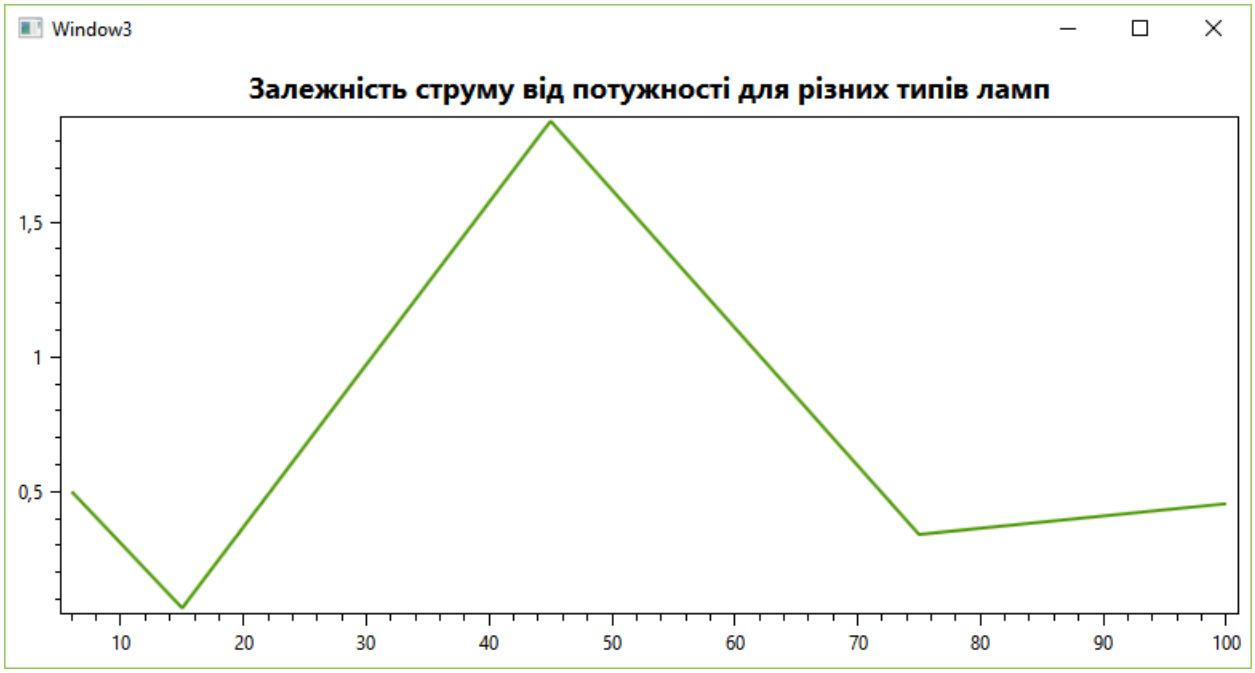

Рис. 9 – Зображення вікна з графіком

Характеристики осей можуть бути задані через властивість **Axes** класу **PlotModel.** Додамо підписи осей:

```
            LampModel.Axes.Add(new LinearAxis
            {
                Position = AxisPosition.Bottom,
                Minimum = 0,
                Maximum = 120,
                Title = "Рном, Вт"
            });
            LampModel.Axes.Add(new LinearAxis
            {
                Position = AxisPosition.Left,
                Minimum = 0,
                Maximum = 3,
                Title = "I, A"
            });
```

Результат зображено на рис. 10.

Рис. 10 – Зображення вікна з графіком

Зберегти зображення графіка у файл можна за допомогою методу класу **PlotView**:

```
SaveBitmap( назва_файлу , ширина , висота , колір );
```

Наприклад:

```
LampPlot.SaveBitmap("plot1.png",700,300,OxyColors.Transparent);
```

Завдання 1

Навчитися створювати прості додатки з графічним інтерфейсом, використовуючи технологію WPF.

1. Створити новий проект типу WPF Application.
2. Додати в проект новий файл коду і скопіювати до нього клас що був розроблений під час виконання практичних робіт. Зверніть увагу, що копіювати потрібно тільки сам клас, а не код методу *Main()*!
3. Додати в основне вікно текстові поля (*Text*), які будуть відповідати полям класу. Біля них розмістити пояснення (опис) у елементах типу *Label*. Розміщення та вирівнювання всіх елементів виконати з використанням рядків і стовпчиків контейнера Grid.
4. Додати багаторядкове текстове поле для виведення результату (*TextBlock*).
5. Кожному елементу присвоїти відповідне ім’я.
6. Для текстових полів, що призначені для введення та виведення інформації, задати різні властивості шрифтів на вибір.
7. Додати кнопку. Для кнопки запрограмувати обробник натискання, в якому виконати такі дії:
    - за допомогою конструктора створити новий екземпляр класу, описаного в попередніх практичних заняттях. Значення полів взяти з відповідних текстових елементів.
    - Вивести текстовий опис створеного об’єкта у текстове поле для результату.

Завдання 2

Навчитися використовувати розширені можливості створення додатків з графічним інтерфейсом, використовуючи технологію WPF.

1. Модифікувати програму додавши у головне вікно меню та панель інструментів.
2. Забезпечити можливість збереження та відкриття файлів з об’єктами класу, що містять серіалізовані дані у бінарному форматі.
3. Забезпечити можливість збереження файлів з описами об’єтів у текстовому форматі.
4. За допомогою елемента управління **TabControl** додати в головне вікно дві вкладки що містять окремо дані полів об’єкту і текстовий опис об’єкту.

Завдання 3

Навчитися використовувати елемент **DataGrid** для відображення та редагування колекцій об’єктів.

1. Модифікувати програму додавши в елемент управління **TabControl** нову вкладку що містить елемент **DataGrid**.
2. Створити та заповнити список об’єктів класу, описаного в попередніх роботах.
3. Зв’язати список з елементом **DataGrid** за допомогою властивості **ItemsSource**.
4. Перевірити можливість редагування значень властивостей об’єктів, а також додавання та видалення об’єктів.
5. Доповнити головне меню програми пунктами, що забезпечують можливість зберігати список у форматі XML.

Завдання 4

Навчитися використовувати бібліотеку класів **OxyPlot** для побудови графіків.

1. Модифікувати програму додавши в проект посилання на бібліотеку **OxyPlot.**
2. Додати в елемент управління **TabControl** нову вкладку що містить елемент **PlotView**.
3. Створити об’єкт класу **PlotModel** та присвоїти його властивості **Model** компонента **PlotView**.
4. Створити новий ряд даних, який представлятиме залежність між двома властивостями на вибір створеного у попередніх роботах класу.
5. Додати створений ряд даних на графік.
6. Задати властивості осей графіка.
7. Створити новий ряд даних з довільними даними на вибір і додати його на графік.
8. На вибір змінити декілька властивостей рядів та осей, таких як колір, шрифт, розмір, товщина тощо.
6. Доповнити головне меню програми пунктами, що забезпечують можливість зберігати графік у файл формату PNG.
9. У протоколі виконання практичного заняття навести код C# та код розмітки XAML, а також зображення вікна запущеної програми.

Варіанти завдань

1. Додаток, що містить відомості про студента: прізвище, ім’я, по батькові, рік народження, зріст.
2. Додаток, що містить відомості про області України: назва, площа, населення, адміністративний центр, населення в адміністративному центрі.
3. Додаток, що містить відомості про модель телефону: виробник, діагональ екрана, кількість процесорних ядер, ціна.

4. Додаток, що містить відомості про електродвигун: тип, номінальна потужність, кількість фаз, маса.
5. Додаток, що містить відомості про монітор комп’ютера: модель, розширення по горизонталі, розширення по вертикалі, яскравість, ціна.
6. Додаток, що містить відомості про лінію електропередачі: тип кабелю, довжина, активний опір, реактивний опір, кількість фаз.
7. Додаток, що містить відомості про будинок: адреса, кількість поверхів, загальна площа, кількість мешканців.
8. Додаток, що містить відомості про футбольну команду: назва, місто, країна, кількість набраних очок за сезон, середня зарплата футболістів.
9. Додаток, що містить відомості про літак: модель, максимальна швидкість, максимальна висота польоту, кількість пасажирів.
10. Додаток, що містить відомості про навчальний предмет: назва, кількість семестрів, кількість лекцій, прізвище викладача.
11. Додаток, що містить відомості про планету сонячної системи: назва, маса, діаметр, кількість супутників.
12. Додаток, що містить відомості про провід ліній електропередачі: марка, площа перерізу, питомий активний опір, матеріал.
13. Додаток, що містить відомості про квартиру: номер, площа, кількість кімнат, споживання електроенергії за місяць.
14. Додаток, що містить відомості про модель ноутбука: назва, виробник, рік випуску, час автономної роботи.
15. Додаток, що містить відомості про студента: прізвище, номер телефону та екзаменаційні оцінки з трьох предметів.
16. Додаток, що містить відомості про прізвища, табельний № та зарплату працівника заводу.
17. Додаток, що містить відомості про автора, назву книги, кількість сторінок та рік її видання.

18. Додаток, що містить коефіцієнти квадратного рівняння, та його запис у текстовому вигляді.
19. Додаток, що містить відомості про розклад руху потягів: номер потяга, назва станції, час прибуття, час відправлення.
20. Додаток, що містить відомості про здачу студентом екзамену: номер групи, прізвище студента, предмет, оцінка.
21. Додаток, що містить відомості про запис студента в бібліотеку: прізвище, ім’я, рік народження, дата запису до бібліотеки.
22. Додаток, що містить відомості про номер автомобіля, марку автомобіля, рік випуску та колір.
23. Додаток, що містить відомості про назву країни, площу країни, населення країни, назву столиці, населення столиці.
24. Додаток, що містить відомості про розклад занять: назва предмета, № пари, день тижня, прізвище викладача.
25. Додаток, що містить відомості про трансформатор: тип, номінальна потужність, напруга первинної обмотки, напруга вторинної обмотки, кількість фаз.

Вимоги до оформлення роботи

Робота оформлюється в друкованому вигляді на папері стандартного формату А4, на одній стороні аркуша з дотриманням полів:

- зліва – 2,5 см
- зверху – 2 см
- знизу – 2 см
- справа – 1,5 см

Шрифт Times New Roman, розмір 14, міжрядковий інтервал 1,5. Всі сторінки мають бути пронумеровані починаючи зі змісту. Абзацний відступ 1,25 см. Текст в абзацах вирівнюється по ширині сторінки.

Заголовки структурних розділів оформлюються великими прописними літерами, вирівнюються по центру сторінки. Кожен структурний розділ повинен починатись з нової сторінки.

Заголовки глав виділяються міжрядковим інтервалом. Може використовуватись курсив або напівжирне форматування, розмір шрифту такий самий, як і загальний текст.

Скорочення слів не допускається, крім загальноприйнятих, при першому вживанні вони супроводжуються розшифровуванням.

Структура роботи:

- Титульна сторінка

  Приклад оформлення титульної сторінки наведено в додатку 1.
- Зміст
- Завдання
- Теоретичний опис використаних методів обчислення визначених інтегралів
- Математичні розрахунки

- Блок-схема алгоритму розв’язку задачі та їх опис.

  Кожний блок повинен бути пронумерований. Усі блоки повинні бути стандартного розміру. Для зв’язку між блоками використовувати стрілочки.
- Текст програми.

  Міжрядковий інтервал для тексту програму одинарний, розмір шрифту 12, вирівнювання тексту з лівого краю.
- Результати роботи програми
- Висновки
- Список використаної літератури

СПИСОК ЛІТЕРАТУРИ

1. ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ: Об’єктно-орієнтоване програмування. Конспект лекцій [Електронний ресурс]: навч. посіб. для студ. спеціальності 141 Електроенергетика, електротехніка та електромеханіка / КПІ ім. Ігоря Сікорського; уклад.: Д. В. Філянін. – Електронні текстові дані (1 файл: 0,869 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 101 с. URL: https://ela.kpi.ua/handle/123456789/47965 (дата звернення 10.06.2023).

2. Andrew Troelsen, Philip Japikse, “Pro C# 9 with .NET 5: Foundational Principles and Practices in Programming”, Apress, P. 1411. April 2021.

3. Joseph Albahari, Ben Albahari, “C# 9.0 Pocket Reference: Instant Help for C# 9.0 Programmers”, 1st edition, O'Reilly Media, P. 264. February 2021.

4. Mark J. Price, “C# 10 and .NET 6 – Modern Cross-Platform Development”, Sixth Edition, Packt Publishing Ltd., P. 825. 2023.

5. Mikael Olsson, “C# 10 Quick Syntax Reference. 4th Ed.”, Apress, P. 196. 2022.

6. ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ: Об’єктно-орієнтоване програмування. Комп’ютерний практикум [Електронний ресурс] : навч. посіб. для студ. спеціальності 141 Електроенергетика, електротехніка та електромеханіка та 184 Гірництво / КПІ ім. Ігоря Сікорського; уклад.: Д. В. Філянін, В. П. Опришко, В. О. Броницький, О. Е. Максименко. – Електронні текстові дані (1 файл: 1,12 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 80 с. URL: https://ela.kpi.ua/handle/123456789/47961 (дата звернення 10.06.2023).

7. Мова програмування C#. URL: http://www.znannya.org/?view=csharp (дата звернення 10.06.2023).

8. C# documentation. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (дата звернення 10.06.2023).

9. Visual C# Guided Tour. URL: https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2008/bb383962(v=vs.90)?redirectedfrom=MSDN (дата звернення 10.06.2023).

10. C# Programming. URL: https://en.wikibooks.org/wiki/C_Sharp_Programming (дата звернення 10.06.2023).

11. C# / CSharp Tutorial. URL: http://www.java2s.com/Tutorial/CSharp/CatalogCSharp.htm (дата звернення 10.06.2023).

12. C#. URL: https://www.bestprog.net/uk/sitemap_ua/c-3/ (дата звернення 10.06.2023).

13. Visual Studio 2022 Community. Безкоштовне повнофункціональне середовище IDE, що розширюється, для створення сучасних програм Android, iOS і Windows, а також веб-додатків і хмарних служб.

ДОДАТОК 1

Приклад титульного аркушу розрахункової роботи

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут»

Навчально-науковий інститут енергозбереження та енергоменджменту

Кафедра електропостачання

Домашня контрольна робота

з дисципліни

«ОБЧИСЛЮВАЛЬНА ТЕХНІКА ТА ПРОГРАМУВАННЯ. Частина 2»

Варіант №

Виконав:

Студент гр. ГЕ-99

Іваненко І. І.

Перевірив:

Петренко П. П.

Київ – 2025