МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського

«Харківський авіаційний інститут»

факультет програмної інженерії та бізнесу

кафедра інженерії програмного забезпечення

**Практична робота № 1**

з дисципліни « Архітектура програмного забезпечення .Net »

*назва дисципліни*

на тему: «РОЗРОБКА КОМПОНЕНТІВ»

Виконав: студент 2 курсу групи № 622п

освітньої програми

121 інженерія програмного забезпечення

(шифр і назва ОП)

Зайченко Ярослав Ігорович

(прізвище й ініціали студента)

Прийняв: к. т. н., доцент

Лучшев Павло Олександрович

(посада, науковий ступінь, прізвище й ініціали)

Кількість балів:

Харків – 2023

**ЗМІСТ**

[Постановка завдання 3](#_Toc159492133)

[Порядок виконання роботи 3](#_Toc159492134)

[Виконання роботи 5](#_Toc159492135)

[Висновок 11](#_Toc159492136)

Мета роботи: Створити додаток, який складається з двох програмних одиниць (проектів) – бібліотеки з компонентом (елементом управління) та файлу, що запускається та використовує компонент із бібліотеки. Розробити компонент шляхом наслідування та/або об'єднання декількох стандартних компонентів в один власний. Виконати його встановлення у середовище розробки Visual Studio та продемонструвати можливість повторного використання.

Основні вимоги до роботи наведені в таблиці 1.1, а варіанти функціональних вимог до компонента в таблиці 1.2 при цьому у компонента має бути як мінімум одна властивість та одна подія з рекомендованих у варіанті або запропонованих самим автором роботи. Допускається реалізація цілком власного варіанта компонента з аналогічними вимогами до складності (після обговорення з викладачем).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Складність | Вимоги до роботи | Бали | Оцінка |
|  | Базовий рівень | Реалізація компоненту та основної програми окремими проектами в рамках одного рішення (*Solution*) | 1 | 1 |
|  | Реалізація власних властивостей та подій компонента, що відображаються в редакторі властивостей (*Properties*) | 2 | 2 |
|  | Реалізація функціональності компонента згідно завдання | 2 | 2 |
|  | Підвищений рівень (самостійна робота) | Прив'язка піктограми до компонента для відображення на панелі інструментів | 1 | 1 |
|  | Прив'язка меж та пропорційна зміна внутрішніх елементів щодо загального розміру компонента | 1 | 1 |
|  | Реалізація додаткової програми тестування в окремому рішенні (*Solution*) та підключення компонента через менеджер пакетів *NuGet* | 1 | 1 |

Постановка завдання

Створити додаток, який складається з двох програмних одиниць (проектів) – бібліотеки з компонентом (елементом управління) та файлу, що запускається та використовує компонент із бібліотеки. Розробити компонент шляхом наслідування та/або об'єднання декількох стандартних компонентів в один власний. Виконати його встановлення у середовище розробки Visual Studio та продемонструвати можливість повторного використання.

Основні вимоги до роботи наведені в таблиці 1.1, а варіанти функціональних вимог до компонента в таблиці 1.2 при цьому у компонента має бути як мінімум одна властивість та одна подія з рекомендованих у варіанті або запропонованих самим автором роботи. Допускається реалізація цілком власного варіанта компонента з аналогічними вимогами до складності (після обговорення з викладачем).

Порядок виконання роботи

1. За допомогою діалогу створення нового проекту (меню *File\New\Project* або *Ctrl+Shift+N*) створити порожнє рішення (*Blank Solution*).
2. У створене рішення за допомогою контекстного меню додати два нові проекти (спочатку *Windows Forms Application*, потім *Windows Forms Control Library*).

***Примітки***:

1. Якщо порядок створення проектів було порушено, перед запуском програми необхідно вказати, що стартовим застосунком є Windows Forms Application.

2. Необхідно враховувати, що обидва проекти повинні мати однакову цільову платформу (***Target Framework***) і, крім цього, далі розглядається використання платформи***.NET 5/6/7/8***, яку слід відрізняти від платформи***.Net Framework 4.x.x***. При зовнішній схожості ці платформи мають багато внутрішніх відмінностей.

1. В проекті *Windows Forms Control* *Library* стандартне ім'я компонента *UserControl1* поміняти на свою назву. Якщо потрібно використовувати інший базовий клас, то виконується відповідне коригування коду.
2. Додати до проекту *Windows Forms Control Library* піктограму (ім'я файлу має збігатися з ім'ям класу п.3). *Solution Explorer \ Add \ New Item\ Icon File*
3. У редакторі піктограм додати нове зображення *New Image Type… \ 16x16, 24 біт* та видалити створені за умовчанням *16х16, 4 біт; 32х32, 4 біт* та ін. через контекстне меню *Delete Image Type*…
4. Зв'язати піктограму з компонентом, додавши код із атрибутом класу … [ToolboxBitmap(@"*drive:\fullpath*\***Ім'я класу***.ico")] public partial class ***Ім'я класу*** : UserControl …
5. ***Примітка***: піктограму можна створювати у форматі *bitmap* але обов'язково *16x16, 24 біт* також вказуючи повний шлях в атрибутах: [ToolboxBitmap(@"*drive:\fullpath*\**Ім'я класу**.bmp")].
6. У вікні властивостей файлу піктограми змінити властивість *Build Action* на *Embedded Resource* або *Resource*.
7. Виконати компіляцію всіх проектів.
8. Якщо створений компонент після компіляції не з'явиться автоматично у верхній частині вікна *Toolbox,* то через основне меню *Tools / Options…* виконати налаштування параметра автоматичного підвантаження компонентів для дизайнера віконних форм: Window Forms Designer 🡪General 🡪 Automatically populate Toolbox 🡪 True
9. Реалізувати логіку компонента відповідно до варіанта (див. табл. 1.2), задіявши задані властивості та події. Допускається та вітається власний варіант завдання з описом функціональності компонента, його властивостей та подій.
10. Розмістити створений компонент на формі *Windows Forms Application* і при необхідності доопрацювати програму для тестування функціональності створеного компонента.

Варіант завдання 10: Два елементи TextBox  для відображення одного значення відстані у сантиметрах у першому TextBox  і в дюймах у другому TextBox .

**Властивості**: відстані в сантиметрах та в дюймах

**Події**: зміни відстані у сантиметрах або у дюймах

Виконання роботи

Опис програми:

Мова програмування: С#, операційна система Windows 11 Prо, Версія 23H2, Збірка ОС 22621.1325, процесор: Apple Silicon M1 Pro 3.20 GHz (ядер: 6), компілятор: Microsoft Visual Studio Community 2022 (64-розрядна версія ARM).

Створення компоненту UserControl

Створений компонент у дизайнері середи розробки зображено на рисунку 1

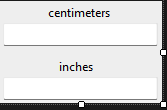


Рисунок 1 – UserControl

Реалізація логіки власного компоненту.

Реалізацію власних властивостей та подій компонента, що відображаються в редакторі властивостей:

**Властивості** CM **та** Inch**:**

Компонент містить дві властивості CM та Inch, які представляють собою значення в сантиметрах і дюймах відповідно.

* Властивість CM перетворює значення в сантиметрах у значення в дюймах і навпаки, властивість Inch робить обернене перетворення.
* Обидві властивості генерують події CMChanged та InchChanged відповідно при зміні своїх значень.

**Події** CMChanged **та** InchChanged**:**

Ці події виникають при зміні властивостей CM та Inch. Вони дозволяють підписникам реагувати на зміни значень сантиметрів та дюймів у компоненті.

**Подія** ErrorOccurred**:**

* Ця подія виникає при виникненні помилок під час обробки користувацького вводу. Вона дозволяє клієнтському коду обробляти помилки, що виникають при введенні даних.

**Методи для генерації подій:**

У компоненті реалізовані методи OnCMChanged(), OnInchChanged() та OnErrorOccurred(Exception ex), які генерують відповідні події. Ці методи надають захищений спосіб виклику подій і їх обробки.

**Обробник події** TextChanged **для текстових полів:**

Реалізований обробник події TextChanged, який реагує на зміни у текстових полях компонента. При зміні значення текстового поля відбувається парсинг введеного значення і його перетворення в сантиметри або дюйми в залежності від вибраного поля.

У випадку помилок вводу або від'ємних чисел генерується подія ErrorOccurred.

**Перевірка на коректність вводу:**

* У обробнику події KeyPress для текстових полів компонента реалізована перевірка на коректність вводу.
* Користувачеві забороняється ввід від'ємних чисел та неприпустимих символів, таких як літери або символи пунктуації.
* Також перевіряється наявність лише одного роздільника (коми або крапки) для уникнення помилок при введенні чисел з плаваючою точкою.

Ці особливості роблять компонент гнучким та зручним у використанні, а його поведінка передбачуваною та інтуїтивно зрозумілою для кінцевих користувачів.

Програмний код компоненту подано нижче

namespace ConverterTxtBoxLibrary

{[ToolboxBitmap(@"C:\Users\Yarik\source\repos\APPZ\_Lab1\_Zaychenko\_622п\WinFormsControlLibrary1\ConverterTxtBox.ico")]

public partial class ConverterTxtBox : UserControl

{

private double cm;

private double inch;

public ConverterTxtBox()

{

InitializeComponent();

textBoxCM.Anchor = AnchorStyles.Left | AnchorStyles.Top | AnchorStyles.Right;

textBoxInch.Anchor = AnchorStyles.Left | AnchorStyles.Top | AnchorStyles.Right;

this.Resize += ConverterTxtBox\_Resize;

}

public double CM

{

get { return cm; }

set

{

if (cm != value)

{

cm = value;

textBoxCM.Text = value.ToString();

inch = value / 2.54;

textBoxInch.Text = inch.ToString();

OnCMChanged();

}

}

}

public double Inch

{

get { return inch; }

set

{

if (inch != value)

{

inch = value;

textBoxInch.Text = value.ToString();

cm = value \* 2.54;

textBoxCM.Text = cm.ToString();

OnInchChanged();

}

}

}

public event EventHandler CMChanged;

public event EventHandler InchChanged;

public event EventHandler<ErrorEventArgs> ErrorOccurred;

protected virtual void OnCMChanged()

{

CMChanged?.Invoke(this, EventArgs.Empty);

}

protected virtual void OnInchChanged()

{

InchChanged?.Invoke(this, EventArgs.Empty);

}

protected virtual void OnErrorOccurred(Exception ex)

{

ErrorOccurred?.Invoke(this, new ErrorEventArgs(ex));

}

private void textBox\_KeyPress(object sender, KeyPressEventArgs e)

{

try

{

TextBox textBox = sender as TextBox;

// Проверка на ввод минуса в начале строки

if (e.KeyChar == '-' && textBox.SelectionStart == 0)

{

e.Handled = true; // Отменяем ввод

throw new Exception("Negative numbers are not allowed.");

}

// Перевірка, чи є введений символ цифрою, крапкою або комою, або клавішею Backspace

if (!char.IsControl(e.KeyChar) && !char.IsDigit(e.KeyChar) && e.KeyChar != ',' && e.KeyChar != 'E' && e.KeyChar != '+' && e.KeyChar != '-')

{

// Якщо введений символ не є цифрою, крапкою, комою або Backspace, скасувати введення

e.Handled = true;

throw new Exception("Atemp of invalid input was stopped");

}

// Перевірка, чи введений символ - крапка або кома, і чи вони вже присутні у тексті

if (e.KeyChar == ',' && (sender as TextBox).Text.IndexOf(',') > -1)

{

// Якщо введено більше однієї крапки або коми, скасувати введення

e.Handled = true;

throw new Exception("More than one separator was typed!");

}

}

catch (Exception ex) { OnErrorOccurred(ex); }

}

private void ConverterTxtBox\_Resize(object sender, EventArgs e)

{

// Размеры компонента изменились, пересчитываем размеры и позиции внутренних элементов

UpdateInternalElements();

}

private void UpdateInternalElements()

{

// Определяем высоту текстового поля как 1/3 высоты компонента

int textBoxHeight = this.Height / 3;

// Устанавливаем размер и позицию для лейблов

labelCM.Location = new Point((this.Width - labelCM.Width) / 2, 5);

labelInch.Location = new Point((this.Width - labelInch.Width) / 2, textBoxHeight + 10);

// Устанавливаем размеры для текстовых полей

textBoxCM.Size = new Size(this.Width, textBoxHeight);

textBoxInch.Size = new Size(this.Width, textBoxHeight);

// Устанавливаем позиции для текстовых полей

textBoxCM.Location = new Point(0, labelCM.Bottom);

textBoxInch.Location = new Point(0, labelInch.Bottom + 5);

}

private void textBox\_TextChanged(object sender, EventArgs e)

{

TextBox textBox = (TextBox)sender;

// Проверяем, не пустое ли значение в текстовом поле

if (string.IsNullOrEmpty(textBox.Text))

{

textBoxCM.Clear();

textBoxInch.Clear();

return;

}

try

{

// Попробуйте распарсить введенное значение как число

double value = double.Parse(textBox.Text);

// Проверяем, если число меньше 0, выводим сообщение об ошибке

if (value < 0)

{

throw new Exception("Negative numbers are not allowed.");

}

// Обработка для поля CM

if (textBox == textBoxCM)

{

CM = value;

}

// Обработка для поля Inch

else if (textBox == textBoxInch)

{

Inch = value;

}

}

catch (FormatException)

{

// Выводим сообщение об ошибке, если введенное значение не является числом

OnErrorOccurred(new Exception("Invalid input. Please enter a valid number."));

}

catch (Exception ex)

{

// Выводим сообщение об ошибке только если это не отрицательное число

if (ex.Message != "Negative numbers are not allowed.")

{

OnErrorOccurred(ex);

}

}

}

}

}

Прив’язка іконки до свого компоненту

Реалізація відбувається написанням атрибуту на початку файлу компоненту. В даному випадку код має такий вид:

[ToolboxBitmap(@"C:\Users\Yarik\source\repos\APPZ\_Lab1\_Zaychenko\_622п\WinFormsControlLibrary1\ConverterTxtBox.ico")]

Прив'язка меж та пропорційна зміна внутрішніх елементів компоненту

Фрагмент коду, який відповідає за реалізацію функціональності пропорційної зміни компоненту подано нижче, та рисунку 2 для демонтрації.

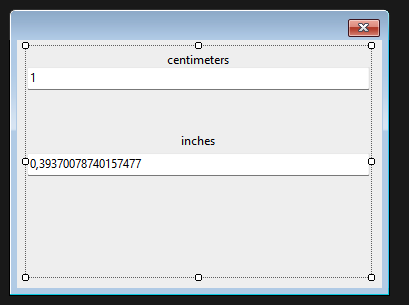
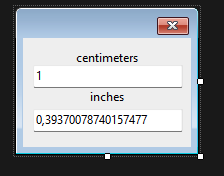


Рисунок 2 – пропорціональна зміна розміру елементів в компоненті

Код вікна WinFoms

Необхідний для обробки виключень компонентом конвертером

using ConverterTxtBoxLibrary;

namespace Lab\_1\_Form

{

public partial class Lab\_1 : Form

{

public Lab\_1()

{

InitializeComponent();

this.ControlBox = true; // Включаем элементы управления окна

this.MinimizeBox = false; // Скрываем кнопку "Свернуть"

this.MaximizeBox = false; // Скрываем кнопку "Развернуть во весь экран"

//this.Resize += Lab\_1\_Resize;

// Подписываемся на событие ErrorOccurred компонента

convCmToInch.ErrorOccurred += ConvCmToInch\_ErrorOccurred;

}

// Метод для обработки события ErrorOccurred

private void ConvCmToInch\_ErrorOccurred(object sender, ErrorEventArgs e)

{

// Получаем информацию об ошибке из аргументов события

Exception error = e.GetException();

// Обрабатываем ошибку

MessageBox.Show(error.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

Підключення компонента через менеджер пакетів NuGet

Сформували пакет NuGet, додали директорію з його розташуванням на диску, та встановили у новий проект. Процес встановлення подано на рисунках 3-5.

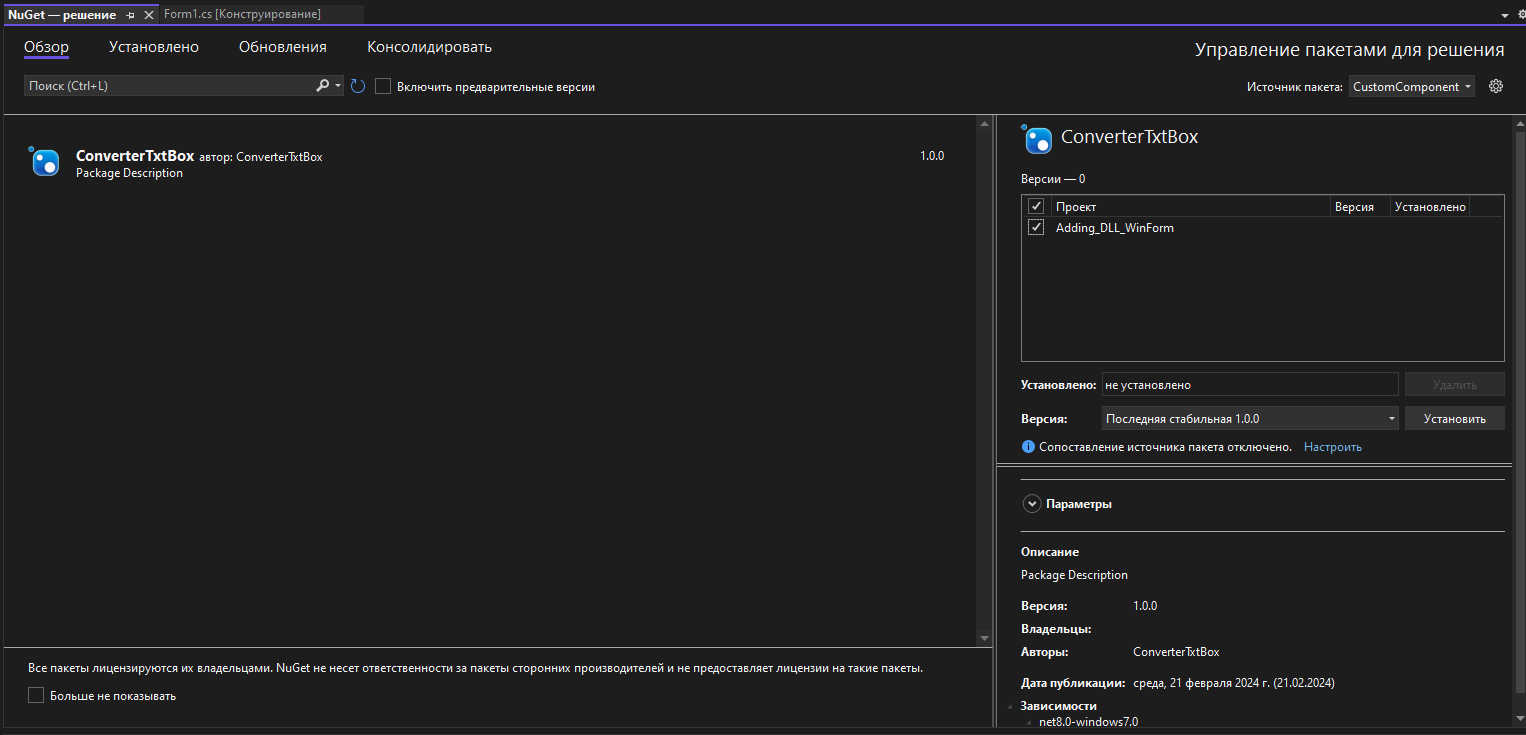


Рисунок 3 – Вікно менеджера пакетів

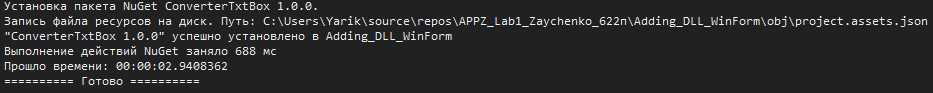


Рисунок 4 – Вивід логів про успішне встановлення пакету в проект

Після встановлення пакету, компонент з’являється у вікні з елементами форми. Подано на рисунку 5.

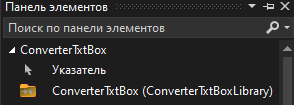


Рисунок 5 – підключений власний компонент до проекту

Висновок

У результаті виконання завдання було створено додаток, який складається з двох програмних одиниць: бібліотеки з компонентом (елементом управління) та файлу, що використовує цей компонент. Компонент був реалізований шляхом наслідування та об'єднання декількох стандартних компонентів у один власний.

Згідно з основними вимогами до роботи, в компоненті було реалізовано мінімум одна властивість та одна подія. У даному випадку були створені властивості `CM` та `Inch`, які дозволяють робити конвертацію відстані між сантиметрами та дюймами. Також були реалізовані події `CMChanged` та `InchChanged`, які сповіщають про зміну значень сантиметрів та дюймів відповідно.

Далі, було виконано встановлення компонента у середовище розробки Visual Studio та продемонстровано його можливість повторного використання. Це дозволяє зручно використовувати створений компонент у різних проектах та забезпечує зручність та ефективність роботи програмістів.

Отже, завдяки виконанню даного завдання було закріплено навички об'єктно-орієнтованого програмування та вивчено особливості створення та використання компонентів та елементів інтерфейсу користувача за допомогою середовища розробки Visual Studio.