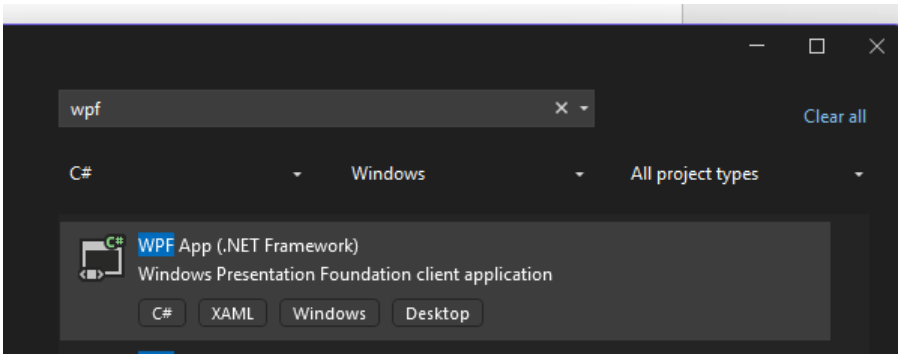
Лабораторная работа №16. Разработка приложения конвертера валют

Цель: Создать приложение для преобразования валют с использованием WPF

Шаг 1. Создание проекта

1. Создайте новый проект. В этот раз нам потребуется **WPF App (.NET Framework)**



2. Назовите проект **CurrencyConverter**.

3. Создайте **Git Repository** для проекта.

Шаг 2. Создание дизайна приложения

2.1. Настройка основного окна

В файле XAML измените свойства Window:

- Сначала задайте для свойства **Title** значение **«Currency Converter».**
- Удалите значения по умолчанию **Height** и **Width** из свойств тега окна и добавьте свойство **SizeToContent="WidthAndHeight",** чтобы задать форму окна в соответствии с размером содержимого.
- Установите свойство **WindowStartupLocation** на **«CenterScreen»,** чтобы задать центральное положение окна.

```
Title="Currency Converter"
SizeToContent="WidthAndHeight"
WindowStartupLocation="CenterScreen">
```

2.2. Настройка сетки (Grid)

Добавьте определение строк в Grid:

```
<Grid.RowDefinitions>
    <RowDefinition Height="60"></RowDefinition>
    <RowDefinition Height="80"></RowDefinition>
    <RowDefinition Height="150"></RowDefinition>
    <RowDefinition Height="100"></RowDefinition>
    <RowDefinition Height="*"></RowDefinition>
</Grid.RowDefinitions>
```

- Панель Grid представляет собой гибкую область, состоящую из строк и столбцов. Дочерние элементы можно организовать в табличной форме. Элементы можно добавлять в любую строку и столбец с помощью свойств **Grid.Row** и **Grid.Column**.
- По умолчанию панель «Сетка» создаётся с одной строкой и одним столбцом. Несколько строк и столбцов создаются с помощью свойств **RowDefinitions** и **ColumnDefinitions**.

2.3. Добавление декоративного элемента Border

Границы в **WPF** работают немного иначе. В **XAML** у элемента управления **Border** есть собственный элемент управления, который можно применять к другим элементам управления или элементам **XAML**. Для размещения границы вокруг элемента в **WPF** предусмотрен элемент **Border**, аналогичный другим элементам **WPF**. У **Border** есть свойства **Width**, **Height**, **Background**, а также **HorizontalAlignment** и **VerticalAlignment**.

3.3.1. Добавляем Border (Рамку)

```
    </Grid.RowDefinitions>
    <Border Grid.Row="2" Width="800"
            CornerRadius="10" BorderThickness="5">
    </Border>
</Grid>
```

- **Grid.Row="2"** – помещает элемент во 2-ю строку нашей сетки (Grid).
- **Width="800"** – задает ширину рамки 800 пикселей.
- **CornerRadius="10"** – скругляет углы рамки на 10 единиц.
- BorderThickness="5" – делает обводку толщиной 5 пикселей.

(Пока рамка пустая, у нее только контур, но нет содержимого и цвета границы.)

3.3.2: Добавляем градиентную обводку BorderBrush

```
<Border Grid.Row="2" Width="800"
        CornerRadius="10" BorderThickness="5">
    <Border.BorderBrush>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Color="#ec2075" Offset="0.0" />
            <GradientStop Color="#f33944" Offset="0.50" />
        </LinearGradientBrush>
    </Border.BorderBrush>
</Border>
```

- **Border.BorderBrush** – определяет, чем закрашивается граница рамки.
- **LinearGradientBrush** – создает линейный градиент (плавный переход цветов).
    - **StartPoint="0,0"** – градиент начинается в левом верхнем углу.
    - **EndPoint="1,0"** – заканчивается в правом верхнем углу (горизонтальный градиент).
- **GradientStop** – определяет цвет в определенной точке градиента:
    - **Color="#ec2075"** – розовый цвет (в начале градиента, Offset="0.0").
    - **Color="#f33944"** – красный цвет (в середине градиента, Offset="0.50").

(Теперь у рамки красивая градиентная обводка от розового к красному.)

3.3.3: Добавляем Rectangle (Прямоугольник) внутрь Border

```
        </LinearGradientBrush>
    </Border.BorderBrush>
    <Rectangle Grid.Row="2">

    </Rectangle>
</Border>
```

- **Grid.Row="2"** – также размещается во 2-й строке Grid (как и Border).
- Пока это просто пустой прямоугольник, который займет все пространство внутри Border.

3.3.4: Заливаем Rectangle градиентом

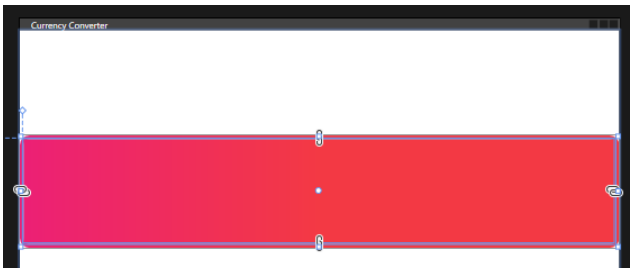
```
<Rectangle Grid.Row="2">
    <Rectangle.Fill>
        <LinearGradientBrush StartPoint="0,0" EndPoint="1,0">
            <GradientStop Color="#ec2075" Offset="0.0" />
            <GradientStop Color="#f33944" Offset="0.50" />
        </LinearGradientBrush>
    </Rectangle.Fill>
</Rectangle>
```

- **Rectangle.Fill** – определяет заливку прямоугольника.
- Используется тот же **LinearGradientBrush**, что и для обводки **Border**:
    - Градиент идет слева направо (**StartPoint="0,0", EndPoint="1,0").**
    - Цвета: **#ec2075 (розовый) → #f33944 (красный**).

(Теперь внутри рамки находится прямоугольник с таким же градиентом, как и сама рамка.)

Итоговый результат:

- Получается **рамка (Border)** с:
  - Толщиной обводки **5px**.
  - Скругленными углами (**10px**).
  - Градиентной обводкой (от розового к красному).
- Внутри нее — **прямоугольник (Rectangle)**, залитый таким же градиентом.



Теперь давайте спроектируем StackPanel.

2.4. Проектирование StackPanel для заголовка приложения

2.4.1. Создаем StackPanel (контейнер)

Добавим контейнер **StackPanel** для автоматического упорядочивания дочерних элементов либо вертикально (по умолчанию), либо горизонтально:

```
    </Border>
    <StackPanel Grid.Row="0"
                Orientation="Horizontal"
                HorizontalAlignment="Center"
                VerticalAlignment="Center"
                Height="50" Width="1000" >
    </StackPanel>
</Grid>
```

- **Grid.Row="0"** - размещает элемент в первой строке нашего Grid
- **Orientation="Horizontal"** - располагает дочерние элементы горизонтально (в ряд)
- **HorizontalAlignment="Center"** - выравнивает панель по центру по горизонтали
- **Height="50"** - фиксированная высота 50 пикселей
- **Width="1000"** - ширина 1000 пикселей (шире экрана, чтобы создать эффект "на всю ширину")
- **VerticalAlignment="Center"** - выравнивает по центру по вертикали

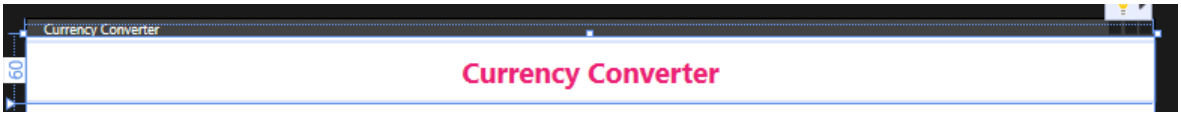
2.4.2. Добавляем Label (надпись)

Внутрь нашего контейнера добавим панель с заголовком "Currency Converter"

```
<StackPanel Grid.Row="0" Orientation="Horizonta
  HorizontalAlignment="Center"
            VerticalAlignment="Center" Height="
    <Label Content="Currency Converter"
           Height="50" Width="1000"
           HorizontalContentAlignment="Center"
           VerticalContentAlignment="Center"
           FontSize="25" FontWeight="Bold"
           Foreground="#ec2075"/>
</StackPanel>
```

- **Height="50"** - высота совпадает с высотой StackPanel
- **Width="1000"** - ширина совпадает с шириной StackPanel
- **HorizontalContentAlignment="Center"** - выравнивание текста по центру горизонтально
- **VerticalContentAlignment="Center"** - выравнивание текста по центру вертикально
- **Content="Currency Converter"** - текст заголовка
- **FontSize="25"** - размер шрифта 25 пунктов
- **Foreground="#ec2075"** - цвет текста (розовый, как в градиенте)
- **FontWeight="Bold"** - полужирное начертание

Итог:



2.5. Проектирование StackPanel для блока результата конвертации

Создаём новую **StackPanel**:

```
<!--Блок результата конвертации-->
<StackPanel Grid.Row="1"
            Orientation="Vertical"
            HorizontalAlignment="Center"
            Height="80" Width="1000">
    <Label Content="Converted Currency"
           Height="40" Width="1000"
           HorizontalContentAlignment="Center"
           VerticalContentAlignment="Center"
           FontSize="20"/>
    <Label Name="lblCurrency"
           Height="40" Width="1000"
           HorizontalContentAlignment="Center"
           VerticalContentAlignment="Center"
           FontSize="20"/>
</StackPanel>
```

Назначение: Отображение результата конвертации.

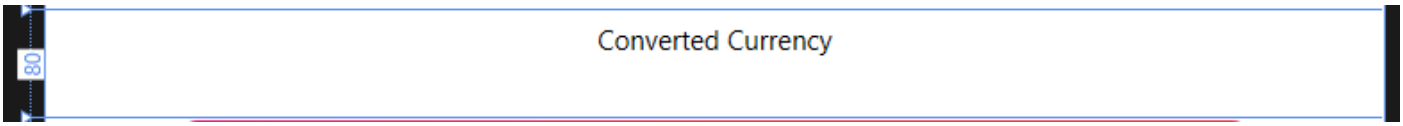
Элементы:

- Статический **Label** с подписью.
- Динамический **Label** (**lblCurrency**) для вывода суммы.

Особенности:

- Вертикальное расположение элементов.
- Ширина 1000px для растягивания на весь экран.

Итог:



2.6. Проектирование StackPanel для подписи полей ввода

Создаём новую **StackPanel**:

```
<!--Подписи полей ввода-->
<StackPanel Grid.Row="2"
            Orientation="Horizontal"
            HorizontalAlignment="Center"
            VerticalAlignment="Top"
            Height="60" Width="800">
    <Label Content="Enter Amount : "
           Height="40" Width="150"
           Margin="35 0 0 0"
           VerticalAlignment="Bottom"
           Foreground="White"
           FontSize="20"/>
    <Label Content="From : "
           Height="40" Width="150"
           Margin="110 0 0 0"
           VerticalAlignment="Bottom"
           Foreground="White"
           FontSize="20"/>
    <Label Content="To : "
           Height="40" Width="150"
           Margin="130 0 0 0"
           VerticalAlignment="Bottom"
           Foreground="White"
           FontSize="20"/>
</StackPanel>
```

Назначение: Подписи для полей ввода/выбора валют.

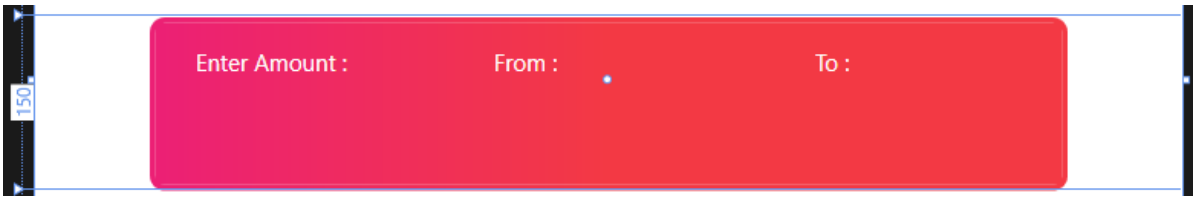
Элементы:

- Три белых (Foreground="White") Label.

Особенности:

- Горизонтальное расположение.
- Отступы (Margin) для выравнивания над полями ввода.

Итог:



2.7. Проектирование StackPanel для полей ввода и выбора валют

```
<!--Поля ввода и выбора валют-->
<StackPanel
    Grid.Row="2"
    Orientation="Horizontal"
    HorizontalAlignment="Center"
    Height="90" Width="800"
    VerticalAlignment="Bottom">
    <TextBox Name="txtCurrency"
             Width="200" Height="30"
             Margin="40 0 0 0"
             PreviewTextInput="NumberValidationTextBox"
             FontSize="18"
             VerticalContentAlignment="Center"
             VerticalAlignment="Top"/>
    <ComboBox Name="cmbFromCurrency"
              Width="170" Height="30"
              Margin="60 0 40 0"
              FontSize="18"
              VerticalContentAlignment="Center"
              VerticalAlignment="Top"
              MaxDropDownHeight="150"/>
    <ComboBox Name="cmbToCurrency"
              Width="170" Height="30"
              Margin="40 0 0 0"
              VerticalContentAlignment="Center"
              VerticalAlignment="Top"
              MaxDropDownHeight="150"
              FontSize="18" />
```

Назначение: Ввод суммы и выбор валют.

Элементы:

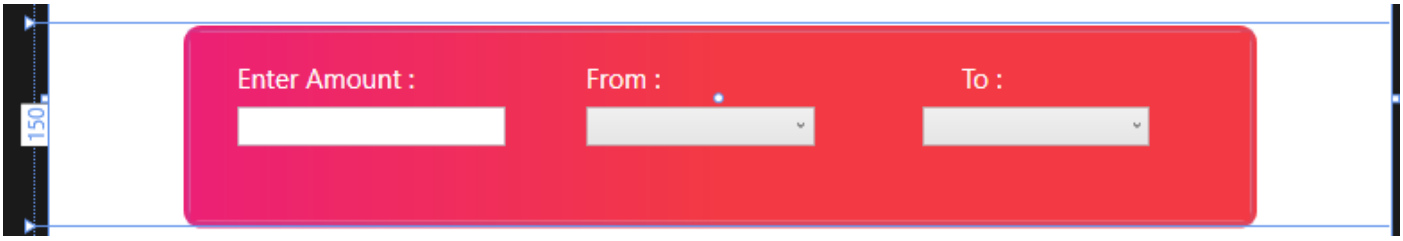
- **TextBox** для ввода числа (с валидацией NumberValidationTextBox).

- Два **ComboBox** для выбора исходной и целевой валют.

Особенности:

- Фиксированные размеры (**Width="170-200px").**
- Вертикальное выравнивание по верхнему краю (**VerticalAlignment="Top"**).

Итог:



2.8. Проектирование StackPanel для кнопок действий

Создаём новую **StackPanel**:

```
<!--Кнопки действий-->
<StackPanel Grid.Row="3"
            Height="100" Width="1000"
            Orientation="Horizontal">
    <Button Name="Convert"
            Height="40" Width="150"
            Content="Convert"
            Click="Convert_Click"
            Margin="350 0 20 0"
            Foreground="White"
            FontSize="20"
            BorderBrush="#FF707070"
            Background="#FFEA0000"/>
    <Button Name="Clear"
            Height="40" Width="150"
            Content="Clear"
            Click="Clear_Click"
            Foreground="White"
            FontSize="20"
            BorderBrush="#FF707070"
            Background="#FFEA0000"/>
</StackPanel>
```

Назначение: Управление приложением.

Элементы:

- Кнопка **Convert** для выполнения конвертации.
- Кнопка **Clear** для сброса данных.

Особенности:

- Красный фон (Background="#FFEA0000") с белым текстом.
- Обработчики событий Click.

Итог:

Чтобы исправить ошибки компилятора добавьте обработчики событий:

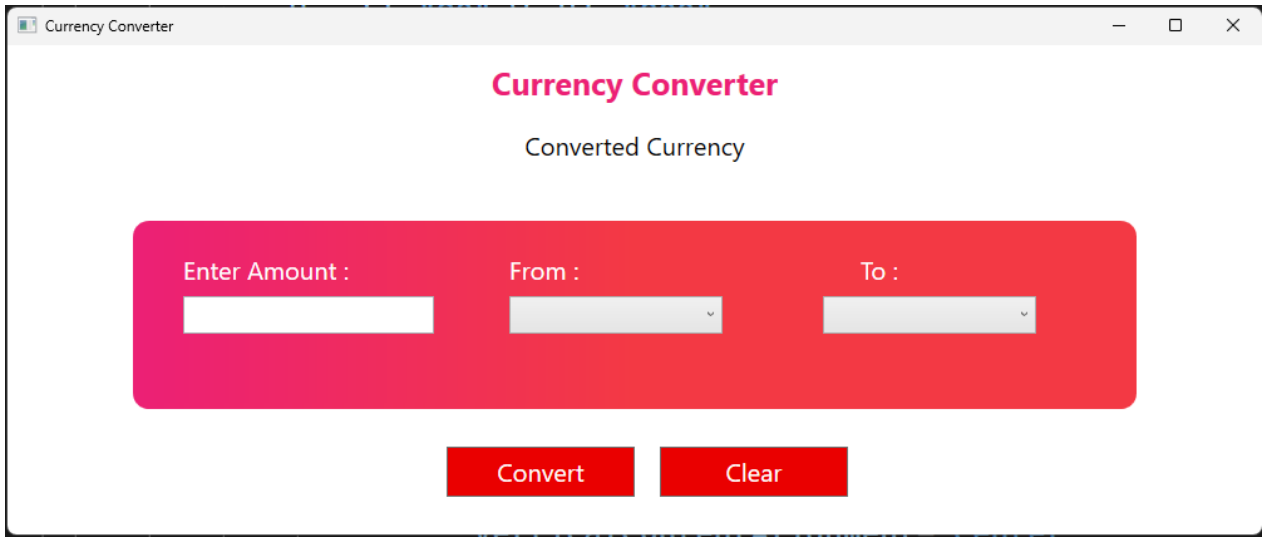
```
private void Convert_Click(object sender, RoutedEventArgs e) {

}
1 reference
private void Clear_Click(object sender, RoutedEventArgs e) {

}
1 reference
private void NumberValidationTextBox(object sender, TextCompositionEventArgs e) {

}
```

TextCompositionEventArgs — это класс в пространстве имён **System.Windows.Input**, который используется в WPF для обработки событий **ввода текста**, таких как:

- **PreviewTextInput**
- **TextInput**

Он содержит информацию о **символах**, которые пользователь вводит с клавиатуры, включая те, которые ещё не были добавлены в элемент управления.

Запустите приложение, у вас должно получиться следующее окно:



Шаг 3. Написание кода

1. Отформатируем код

Для начала уберём лишние комментарии и неиспользуемые пространства имён, оставив только минимально необходимое:

```
using System.Windows;
using System.Windows.Input;

namespace CurrencyConverter2 {

    public partial class MainWindow : Window {
        public MainWindow() {
            InitializeComponent();
        }
        private void BindCurrency() {

        }
        private void Convert_Click(object sender, RoutedEventArgs e) {

        }
        private void Clear_Click(object sender, RoutedEventArgs e) {

        }
        private void ClearControls() {
        }
        private void NumberValidationTextBox(object sender, TextCompositionEventArgs e) {

        }
    }
}
```

На этом этапе у нас есть чистый каркас — приложение запускается, но ещё ничего не делает.

2. Подключаем необходимые пространства имён

```
using System.Data; // Для работы с DataTable
using System.Text.RegularExpressions; // Для валидации чисел
using System.Windows; // Базовые WPF-компоненты
using System.Windows.Input; // Для обработки ввода
```

3. Создаем каркас класса

Внутри класса заготовим **пустые методы**:

- **BindCurrency**() — заполнение списка валют;
- **ClearControls**() — очистка всех полей и результата.

```
private void BindCurrency() {

}
0 references
private void ClearControls() {

}
```

Также подключим их вызовы:

- **BindCurrency**() и **ClearControls**() вызываем в конструкторе, чтобы при старте окно уже имело подготовленные списки и чистое состояние;

```
public MainWindow() {
    InitializeComponent();
    ClearControls();
    BindCurrency();
}
```

- **ClearControls**() вызываем и в кнопке Clear.

```
private void Clear_Click(object sender, RoutedEventArgs e) {
    ClearControls();
}
```

4. Реализуем метод BindCurrency()

В начале создаём таблицу для хранения данных о валютах:

```
private void BindCurrency() {
    DataTable dtCurrency = new DataTable();
```

Добавляем в неё два столбца:

- "**Text**" — отображаемое название валюты в списке;
- "**Value**" — числовой курс относительно базовой валюты (например, INR).

```
dtCurrency.Columns.Add("Text");
dtCurrency.Columns.Add("Value");
```

Затем заполняем строки:

- первая строка — --SELECT--, как подсказка для пользователя;
- следующие строки — валюты с примерными курсами (INR, USD, EUR, SAR, POUND, DEM, RUB и т.д.).

```
dtCurrency.Rows.Add("--SELECT--", 0);
dtCurrency.Rows.Add("RUB", 1.2);
dtCurrency.Rows.Add("INR", 1);
dtCurrency.Rows.Add("USD", 75);
dtCurrency.Rows.Add("EUR", 85);
dtCurrency.Rows.Add("SAR", 20);
dtCurrency.Rows.Add("POUND", 5);
dtCurrency.Rows.Add("DEM", 43);
```

Далее настраиваем **первый ComboBox** (откуда переводим):

- указываем, что его источник данных — наша таблица;
- задаём, что показывать (**DisplayMemberPath** = "Text") и какое значение хранить (**SelectedValuePath** = "**Value**");
- выставляем начальный выбор на первую строку (подсказку).

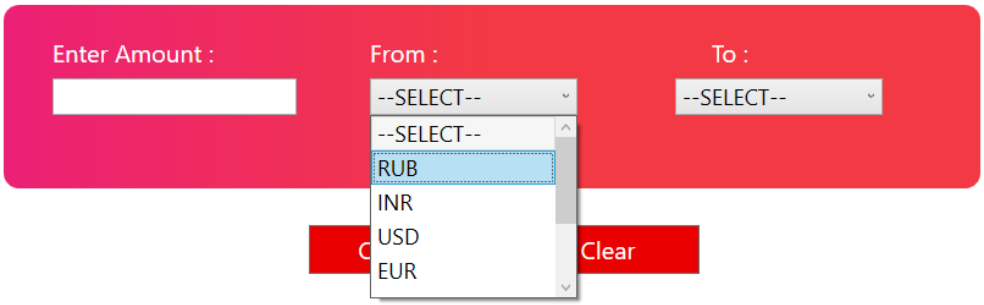
```
cmbFromCurrency.ItemsSource = dtCurrency.DefaultView;
cmbFromCurrency.DisplayMemberPath = "Text";
cmbFromCurrency.SelectedValuePath = "Value";
cmbFromCurrency.SelectedIndex = 0;
```

Аналогично настраиваем **второй ComboBox** (в какую валюту переводим).

```
cmbToCurrency.ItemsSource = dtCurrency.DefaultView;
cmbToCurrency.DisplayMemberPath = "Text";
cmbToCurrency.SelectedValuePath = "Value";
cmbToCurrency.SelectedIndex = 0;
```

Итоговый код:

```
private void BindCurrency() {
    DataTable dtCurrency = new DataTable();
    dtCurrency.Columns.Add("Text");
    dtCurrency.Columns.Add("Value");
    dtCurrency.Rows.Add("--SELECT--", 0);
    dtCurrency.Rows.Add("RUB", 1.2);
    dtCurrency.Rows.Add("INR", 1);
    dtCurrency.Rows.Add("USD", 75);
    dtCurrency.Rows.Add("EUR", 85);
    dtCurrency.Rows.Add("SAR", 20);
    dtCurrency.Rows.Add("POUND", 5);
    dtCurrency.Rows.Add("DEM", 43);

    cmbFromCurrency.ItemsSource = dtCurrency.DefaultView;
    cmbFromCurrency.DisplayMemberPath = "Text";
    cmbFromCurrency.SelectedValuePath = "Value";
    cmbFromCurrency.SelectedIndex = 0;

    cmbToCurrency.ItemsSource = dtCurrency.DefaultView;
    cmbToCurrency.DisplayMemberPath = "Text";
    cmbToCurrency.SelectedValuePath = "Value";
    cmbToCurrency.SelectedIndex = 0;
}
```



В окне: после запуска видим, что оба списка заполнены валютами. Можно выбрать, например, USD в первом и RUB во втором.

5. Реализуем кнопку очистки

В методе **ClearControls**() мы хотим:

- очищать поле ввода суммы;
- сбрасывать оба **ComboBox** на первый элемент (--SELECT--);
- удалять текст результата;
- возвращать фокус в поле суммы, чтобы можно было сразу ввести новое значение.

Реализуем это:

```
private void ClearControls() {
    txtCurrency.Text = string.Empty;
    if (cmbFromCurrency.Items.Count > 0)
        cmbFromCurrency.SelectedIndex = 0;
    if (cmbToCurrency.Items.Count > 0)
        cmbToCurrency.SelectedIndex = 0;
    lblCurrency.Content = "";
    txtCurrency.Focus();
}
```

Проверяем в действии

- Запускаем приложение
- Выбираем, например, **USD → RUB**
- Вводим сумму, жмём **Convert** — в поле результата появится пересчитанная сумма.
- Теперь кликаем **Clear** и видим, что наши поля очистились.

6. Добавляем обработчик

В метод **Convert_Click** вписываем логику, которая:

- Проверяет, ввёл ли пользователь сумму.
- Проверяет, выбраны ли обе валюты.
- Считает результат по формулам.

6.1. Проверка ввода пользователя

```
private void Convert_Click(object sender, RoutedEventArgs e) {
    if (string.IsNullOrWhiteSpace(txtCurrency.Text)) {
        MessageBox.Show("Please enter amount");
        return;
    }
```

- Проверяет, ввел ли пользователь сумму для конвертации
- **string.IsNullOrWhiteSpace** проверяет, является ли строка пустой или состоящей только из пробелов
- Если проверка не проходит, показывает сообщение и выходит из метода

6.2. Проверка выбора валют

```
if (cmbFromCurrency.SelectedIndex <= 0 ||
  cmbToCurrency.SelectedIndex <= 0) {
    MessageBox.Show("Please select currencies");
    return;
}
```

- Проверяет, выбрал ли пользователь валюты для конвертации
- **SelectedIndex** <= 0 означает, что либо ничего не выбрано (индекс -1), либо выбран первый элемент (индекс 0), который может быть заголовком типа "Select **currency**"
- Если проверка не проходит, показывает сообщение и выходит из метода

6.3. Получение введенной суммы

```
double amount = double.Parse(txtCurrency.Text);
```

- Преобразует текст из текстового поля **txtCurrency** в число типа **double**

6.4. Получение курсов валют

```
double fromValue = double.Parse
  (cmbFromCurrency.SelectedValue.ToString());
double toValue = double.Parse
  (cmbToCurrency.SelectedValue.ToString());
```

- Получает выбранные значения из комбобоксов валют (**cmbFromCurrency** и **cmbToCurrency**)
- Преобразует их в числа типа **double**
- Предполагается, что **SelectedValue** содержит числовое значение курса валюты

6.5. Расчет результата

```
double result = (amount * fromValue) / toValue;
```

- Выполняет конвертацию по формуле: (сумма * курс исходной валюты) / курс целевой валюты
- Это стандартная формула для конвертации через базовую валюту (обычно USD)

6.6. Вывод результата

```
lblCurrency.Content = $"{cmbToCurrency.Text} {result:N3}";
```

- Форматирует результат с точностью до 3 знаков после запятой (N3)
- Отображает в формате "[Код валюты] [сумма]", например "EUR 123.456"
- Выводит результат в содержимое элемента lblCurrency

Итоговый код:

```
private void Convert_Click(object sender, RoutedEventArgs e) {
    if (string.IsNullOrWhiteSpace(txtCurrency.Text)) {
        MessageBox.Show("Please enter amount");
        return;
    }
    if (cmbFromCurrency.SelectedIndex <= 0 ||
      cmbToCurrency.SelectedIndex <= 0) {
        MessageBox.Show("Please select currencies");
        return;
    }

    double amount = double.Parse(txtCurrency.Text);
    double fromValue = double.Parse
      (cmbFromCurrency.SelectedValue.ToString());
    double toValue = double.Parse
      (cmbToCurrency.SelectedValue.ToString());
    double result = (amount * fromValue) / toValue;

    lblCurrency.Content = $"{cmbToCurrency.Text} {result:N3}";
}
```

7. Ограничиваем ввод только цифрами

В метод **NumberValidationTextBox** вставляем проверку:

```
private void NumberValidationTextBox(object sender,
  TextCompositionEventArgs e) {
    // Разрешаем цифры, точку и запятую
    Regex regex = new Regex("^[0-9.,]+$");
    // Блокируем, если символ не подходит
    e.Handled = !regex.IsMatch(e.Text);
}
```

- **^[0-9.,]+$** — регулярное выражение, разрешающее любую комбинацию цифр, точки и запятой.
- **e.Handled = !regex.IsMatch(...)** — если введённый символ не подходит под шаблон, мы его не принимаем.

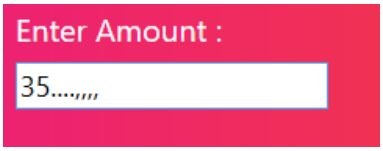
Проверяем в работе

- Запускаем проект, проверяем что у нас печатаются только цифры.

Шаг 4. Рефакторинг кода

При проверке приложения вы могли обнаружить ряд ошибок.

Например:



Это валидация точки/запятой. Сейчас **regex** пропускает и точку, и запятую, но пользователи могут ввести обе сразу или несколько разделителей.

Также если вы вспомните, то для денежных расчётов лучше использовать тип данных decimal, т.к. **double** даёт двоичную погрешность на деньгах.

Также можно локализовать наше приложение.

1. Обновим метод NumberValidationTextBox():

```
var tb = txtCurrency;
// Разрешаем только цифры, точку, запятую
if (!Regex.IsMatch(e.Text, @"^[0-9.,]$")) {
    e.Handled = true;
    return;
}
// Разрешаем только один разделитель (точка или запятая)
if ((e.Text == "." && tb.Text.Contains(".")) ||
    (e.Text == "," && tb.Text.Contains(",")) ||
    (tb.Text.Contains(".") && e.Text == ",") ||
    (tb.Text.Contains(",") && e.Text == ".")) {
    e.Handled = true;
}
```

2. Обновим метод **Convert_Click**

```
private void Convert_Click(object sender, RoutedEventArgs e) {
    if (string.IsNullOrWhiteSpace(txtCurrency.Text)) {
        MessageBox.Show("Введите сумму", "Информация",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }
```

```
    if (cmbFromCurrency.SelectedIndex <= 0 ||
cmbToCurrency.SelectedIndex <= 0) {
        MessageBox.Show("Выберите валюты (из/в)", "Информация",
MessageBoxButton.OK, MessageBoxImage.Information);
        return;
    }

    // Пробуем разобрать сумму с учётом текущей и инвариантной
культур
    if (!decimal.TryParse(txtCurrency.Text, NumberStyles.Number,
CultureInfo.CurrentCulture, out decimal amount) &&
        !decimal.TryParse(txtCurrency.Text, NumberStyles.Number,
CultureInfo.InvariantCulture, out amount)) {
        MessageBox.Show("Неверный формат числа", "Ошибка",
MessageBoxButton.OK, MessageBoxImage.Error);
        return;
    }

    decimal fromValue =
decimal.Parse(cmbFromCurrency.SelectedValue.ToString());
    decimal toValue =
decimal.Parse(cmbToCurrency.SelectedValue.ToString());
    decimal result = (amount * fromValue) / toValue;

    lblCurrency.Content = $"{cmbToCurrency.Text} {result:N3}";
}
```

Самостоятельные задания

Цель: Создать WPF-приложение, в котором пользователь вводит имя, возраст и пароль. После нажатия кнопки данные проверяются, и выводится сообщение об успешной регистрации или ошибке.

Требования к заданию

1. **Создать новый проект WPF Application** Назовите его SimpleRegistrationApp.
2. **Интерфейс должен содержать:**
    - Заголовок "Регистрация"
    - **TextBox** для имени
    - **TextBox** для возраста (только цифры)
    - **PasswordBox** для пароля
    - **Button** "Зарегистрироваться"
3. **Логика регистрации:**

- Имя не должно быть пустым
- Возраст должен быть числом от 1 до 120
- Пароль должен быть не короче 6 символов
- При успешной проверке — показать "Регистрация успешна"
- При ошибке — показать соответствующее сообщение

Пример реализации: