WPF - Data Binding

Exercise #1

Exercise 1: Basic One-Way Binding

1. **Objective**: Display the value of a `Slider` in a `TextBlock`.
2. **Steps**:
3. Open `MainWindow.xaml`.
4. Inside the `<Grid>` tag, add a `Slider` and a `TextBlock` below it.
5. Set the `Minimum` and `Maximum` properties of the `Slider` (e.g., `Minimum="0"` and `Maximum="100"`).
6. Add a `TextBlock` and bind its `Text` property to the `Value` property of the `Slider`:

```
<Slider x:Name="slider" Minimum="0" Maximum="100" />
<TextBlock Text="{Binding ElementName=slider, Path=Value, StringFormat=F0}" />
```

1. Run the application.
2. **Expected Outcome**: Moving the slider updates the `TextBlock` to show the current slider value.

```xml
<UserControl x:Class="DataBindingExercices.Exe1.Exe1"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe1"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="50*"/>
            <RowDefinition Height="50*"/>
        </Grid.RowDefinitions>
        <Slider x:Name="slider" Minimum="0" Maximum="100" VerticalAlignment="Top" Height="20"
                Background="LightPink" Margin="10" Width="600"/>
        <TextBlock Grid.Row="1" Text="{Binding ElementName=slider , Path=Value, StringFormat=F0}" VerticalAlignment="Center"
                   HorizontalAlignment="Center" FontSize="16" FontWeight="Bold" Background="LightPink" Margin="10"/>
    </Grid>
</UserControl>
```

Slider - המאות שם כדי שנוכל לשנות אליו ב-data binding מסוג TextBlock.
הגדרתי הטווח מ-0 עד 100.

TextBlock - מציגה את הערך של ה-Slider ומעגלה אותו.
F0 - StringFormat משמש להגדרת המספר כערך שלם ללא נקודה עשרונית.
מגדירים ה-TextBlock קשור למאפיין הערך של ה-Slider. עם הגדרת ה-Slider המאפיין Value משתנה וה-TextBlock מציג את הערך המעודכן בזמן אמת.

Exercise #2

Exercise 2: Two-Way Binding

1. **Objective**: Create two synchronized TextBox elements so that typing in one updates the other.
2. **Steps**:
3. In MainWindow.xaml, add two TextBox controls in the <Grid>.
4. Bind the Text property of each TextBox to a shared property called SharedText in the code-behind:

```
<TextBox Text="{Binding SharedText, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" Width="200" />
<TextBox Text="{Binding SharedText, Mode=TwoWay,
UpdateSourceTrigger=PropertyChanged}" Width="200" Margin="0,10,0,0" />
```

1. In MainWindow.xaml.cs, define a property SharedText and set DataContext to this in the constructor:

```
public partial class MainWindow : Window
{
private string _sharedText;
public string SharedText
{
get => _sharedText;
set
{
_sharedText = value;

OnPropertyChanged(nameof(SharedText));
}
}
public MainWindow()
{
InitializeComponent();
DataContext = this;
}
}
```

1. Run the application.
2. **Expected Outcome**: Typing in either TextBox automatically updates the other with the same text.

```
<UserControl x:Class="DataBindingExercices.Exe2.Exe2"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe2"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="50*"/>
            <RowDefinition Height="50*"/>
        </Grid.RowDefinitions>
        <TextBox Text="{Binding SharedText, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" Width="200" VerticalAlignment="Center"
           HorizontalAlignment="Center" FontSize="16" FontWeight="Bold" Background="LawnGreen" Margin="10"/>
        <TextBox Grid.Row="1" Text="{Binding SharedText, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" Width="200"
                 VerticalAlignment="Center" HorizontalAlignment="Center" FontSize="16" FontWeight="Bold" Background="LawnGreen" Margin="10"/>
    </Grid>
</UserControl>
```

ה- Grid בנוי מ-2 שורות כל שורה מכילה TextBox.

- Mode=TwoWay מאפשר binding דו כיווני, כך שכל שינוי בתיבת הטקסט הא לא ביטוי ל- SharedText ולהפך

UpdateSourceTrigger=PropertyChanged מעדכנת את ה- SharedText בכל פעם שהטקסט משתנה ומספקת סנכרון בזמן אמת בין שתי ה- TextBox.

```
using System.ComponentModel;
using System.Windows.Controls;


namespace DataBindingExercices.Exe2
{
    2 references
    public partial class Exe2 : UserControl ,INotifyPropertyChanged
    {
        0 references
        public Exe2()
        {
            DataContext = this;
            InitializeComponent();
        }

        private string _sharedText;
        1 reference
        public string SharedText
        {
            get => _sharedText;
            set
            {
                _sharedText = value;
                OnPropertyChanged(nameof(SharedText));
            }
        }

        public event PropertyChangedEventHandler? PropertyChanged;

        1 reference
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

הוספה את כל האובייקטים במחלקה לצורך data binding ב- XAML.

הגדרת מאפשר לה מאפשרת למחלקה לבודא למשתמש האחר מאפיין מאוגד משותף, דבר החיוני ל binding data דו כיווני.

מאפיין זה מחובר לשני רכיבי TextBox

בעצה למשתמש על השינוי המשתנה

הפעלה מפעילה את ה- event PropertChanged. מאפשר המשתנה מקשיב לאירוע זה כדי לעדכן את כל הרכיבים הקשורים ל- SharedText כל פעם שערכו משתנה.

Exercise #3

Exercise 3: Binding to a Data Context

1. **Objective**: Bind a TextBox's Text property to a property in the code-behind.
2. **Steps**:
3. In `MainWindow.xaml.cs`, define a public `string` property called `Name`.
4. Implement `INotifyPropertyChanged` for `MainWindow`.
5. Set `DataContext` to `this` in the constructor of `MainWindow` so that the UI can access properties.
6. In `MainWindow.xaml`, add a `TextBox` and bind its `Text` property to `Name`:

```
<TextBox Text="{Binding Name, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
Width="200" />
```

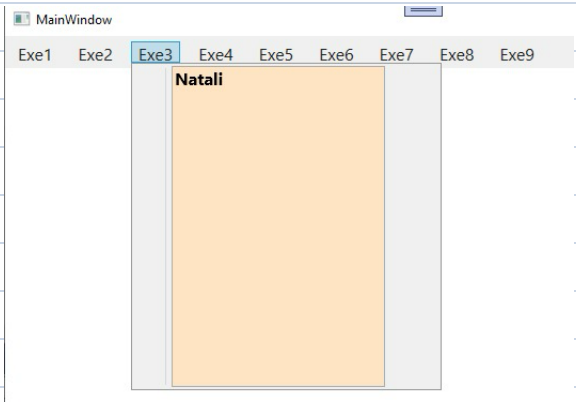
1. Run the application and type in the TextBox.
2. **Expected Outcome**: Any changes in the TextBox reflect in the Name property in the code-behind and vice versa.

```
<UserControl x:Class="DataBindingExercices.Exe3.Exe3"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe3"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <TextBox Text="{Binding UserName , Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}" Width="200"
                 Height="300" FontSize="16" FontWeight="Bold" Background="Bisque"/>
    </Grid>
</UserControl>
```

מאזין בזמן אמת

TextBox - המאופיין Text קשור לשם userName ב- backend.
Mode = TwoWay - מבטיח שינויים בתיבת הטקסט מעדכנים את המאופיין userName ולהפך.

```
Output
Show output from: Debug
Name changed to:
Name changed to: N
Name changed to: Na
Name changed to: Nat
Name changed to: Nata
Name changed to: Natal
Name changed to: Natali
```




```
using System.ComponentModel;
using System.Windows.Controls;

namespace DataBindingExercices.Exe3
{
    2 references
    public partial class Exe3 : UserControl , INotifyPropertyChanged
    {
        0 references
        public Exe3()
        {
            DataContext = this;
            InitializeComponent();
        }

        private string _userName;

        public event PropertyChangedEventHandler? PropertyChanged;

        1 reference
        public string UserName
        {
            get { return _userName; }
            set
            {
                if(_userName != value)
                {
                    _userName = value;
                    OnPropertyChanged(nameof(UserName));
                    System.Diagnostics.Debug.WriteLine($"Name changed to: {_userName}");
                }
            }
        }
        1 reference
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

מאשר השינוי מאזין ל- event כדי לדעת מתי בוצע שינוי.

המאופיין שאליו נקשר הה TextBox

בדיקה האם הערך החדש שונה מ- userName הנוכחי. אם שונה הוא מעדכן את ה- userName ומעלה את האירוע PropertyChanged

הפונקציה אומרת למשתמש להעדכן בכל פעם שה- userName משתנה.

Exerci

Exercise #4

Exercise 4: Binding with Converters

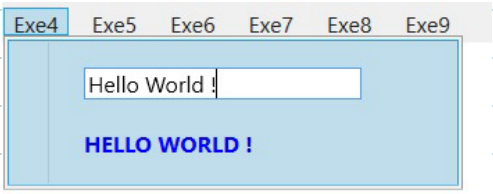
1. **Objective:** Use a value converter to convert text to uppercase.
2. **Steps:**
3. Create a new class called `UpperCaseConverter` that implements `IValueConverter`.
4. In `Convert`, return the input text in uppercase.
5. Register the converter in `Window.Resources`:

```
<Window.Resources>
<local:UpperCaseConverter x:Key="UpperCaseConverter" />
</Window.Resources>
```

1. Bind a `TextBox` to a `TextBlock` with the converter:

```
<TextBox x:Name="inputTextBox" Width="200" />
<TextBlock Text="{Binding ElementName=inputTextBox, Path=Text, Converter=
{StaticResource UpperCaseConverter}}" Width="200" />
```

1. Run the application.
2. **Expected Outcome:** Text entered in the `TextBox` appears in uppercase in the `TextBlock`.




```
using System.Globalization;
using System.Windows.Data;

namespace DataBindingExercices.Exe4
{
    0 references
    public class UpperCaseConverter : IValueConverter
    {
        0 references
        public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
        {
            if (value is string input)
            {
                return input.ToUpper(); // Convert the text to uppercase
            }
            return value;
        }

        0 references
        public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
        {
            // No need to implement ConvertBack in this case since we are only doing one-way binding
            return value;
        }
    }
}
```

המחלקה הזו מבצעת את ההמרה מאותיות קטנות לגדולות ומג'ב במשקל המשתנה.
המחלקה מישמת את ממשק IValueConverter המאפשר הפיכה של נתונים במהלך ה-binding.

שיטה זו משמשת לקישור דו כיווני אך אנו רוצים בתרגיל רק קישור חד כיווני מ-TextBox ל-TextBlock.

```
<UserControl x:Class="DataBindingExercices.Exe4.Exe4"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe4"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <UserControl.Resources>
        <!-- Register the UpperCaseConverter as a resource -->
        <local:UpperCaseConverter x:Key="UpperCaseConverter" />
    </UserControl.Resources>

    <StackPanel HorizontalAlignment="Center" VerticalAlignment="Center" Margin="10">
        <!-- Input TextBox -->
        <TextBox x:Name="inputTextBox" Width="200" Margin="10" />

        <!-- Output TextBlock with the converter applied -->
        <TextBlock Text="{Binding ElementName=inputTextBox, Path=Text, Converter={StaticResource UpperCaseConverter}}"
                   Width="200" FontSize="16" FontWeight="Bold" Foreground="Blue" Margin="10"/>
    </StackPanel>
</UserControl>
```

הוספת משאב שנשתמש בו (binding)

TextBox שם המשתנה יקליד את הטקס

TextBlock תתיבה בה ישתנה הטקסט לאותיות גדולות (UpperCaseConverter) על קשר למאפיין Text של inputTextBox.

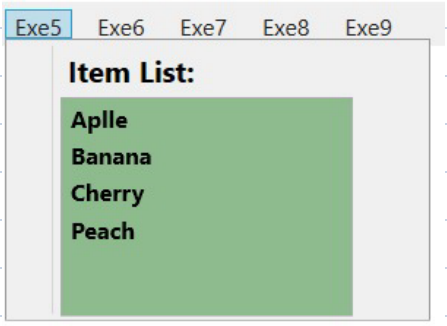
Exercise #5

Exercise 5: One-Way Binding with a Collection

1. **Objective**: Bind a list of strings to a ListBox.
2. **Steps**:
3. In MainWindow.xaml.cs, create an ObservableCollection<string> called Items.
4. Populate Items with sample strings in the constructor.
5. Bind the ItemsSource of a ListBox in MainWindow.xaml to Items:

<ListBox ItemsSource="{Binding Items}" Width="200" Height="150" />

1. Run the application.
2. **Expected Outcome**: The ListBox displays each string in the Items collection.




```xml
<UserControl x:Class="DataBindingExercices.Exe5.Exe5"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe5"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="20*"/>
            <RowDefinition Height="80*"/>
        </Grid.RowDefinitions>
        <Label Content="Item List:" FontSize="20" FontWeight="Bold"/>
        <ListBox Grid.Row="1" ItemsSource="{Binding Items}" Width="200" Height="150" Background="DarkSeaGreen"
                 FontSize="16" FontWeight="Bold" VerticalAlignment="Center" HorizontalAlignment="Center"/>
    </Grid>
</UserControl>
```

ListBox - בתוך רכיב זה יש אוסף ItemSource שקשור לאוסף של Items
שנמצא בקוד ב- backend.
בגלל ש- ItemSource הוא bound, כל שינוי בפריטים יבוא לידי ביטוי
אוטומט ב- ListBox.

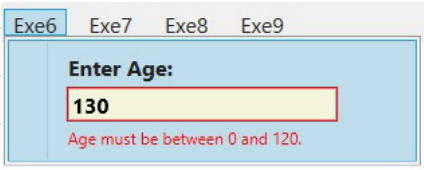
```
using System.Collections.ObjectModel;
using System.Windows.Controls;


namespace DataBindingExercices.Exe5
{
    2 references
    public partial class Exe5 : UserControl
    {
        1 reference
        public ObservableCollection<string> Items { get; set; }

        0 references
        public Exe5()
        {
            InitializeComponent();

            Items = new ObservableCollection<string>
            {
                "Aplle", "Banana", "Cherry", "Peach"
            };

            DataContext = this;
        }
    }

}
```

הגדרת אוסף ObservableCollection
של Items מודיע אוטומט -
למשתמש המשתמש על כל שינוי,
כגון הוספה או הסרה של פריטים.

Exercise #6

Exercise 6: Two-Way Binding with Validation Rules

1. **Objective**: Validate input in a TextBox using custom validation.
2. **Steps**:
3. Create a TextBox bound to an Age property.
4. Implement a custom validation rule class for age validation.
5. Add this rule to the ValidationRules of the TextBox in XAML.
6. Run the application.
7. **Expected Outcome**: Entering invalid data shows an error message.



```
using System.ComponentModel;
using System.Windows.Controls;

namespace DataBindingExercices.Exe6
{
    2 references
    public partial class Exe6 : UserControl , INotifyPropertyChanged
    {
        private int _age;

        public event PropertyChangedEventHandler? PropertyChanged;

        1 reference
        public int Age
        {
            get { return _age; }
            set
            {
                _age = value;
                OnPropertyChanged(nameof(Age));
            }
        }

        0 references
        public Exe6()
        {
            InitializeComponent();
            DataContext = this;
        }

        1 reference
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

מחלקה המכילה את המאפיין Age

השיטה מופעלת בשביל המאפיין כדי שתתעדכן ומבטיחה שמשתמש המשתמש מתעדכן אוטומטית.

```
using System.Globalization;
using System.Windows.Controls;

namespace DataBindingExercices.Exe6
{
    0 references
    public class AgeValidationRule : ValidationRule
    {
        0 references
        public override ValidationResult Validate(object value, CultureInfo cultureInfo)
        {
            if (int.TryParse(value.ToString(), out int age))
            {
                if (age >= 0 && age <= 120) // Define a valid age range
                {
                    return ValidationResult.ValidResult;
                }
                else
                {
                    return new ValidationResult(false, "Age must be between 0 and 120.");
                }
            }
            else
            {
                return new ValidationResult(false, "Invalid input. Please enter a number.");
            }
        }
    }
}
```

המחלקה נועדה לבדוק האם קלט הכל הוא מספר חוקי ושלם.

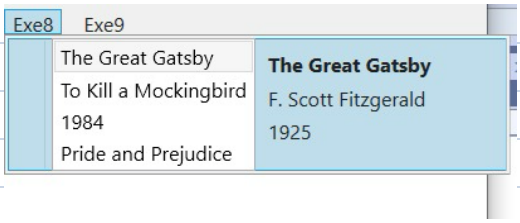
```
<UserControl x:Class="DataBindingExercices.Exe6.Exe6"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe6"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">

    <UserControl.Resources>
        <!-- Register the custom validation rule -->
        <local:AgeValidationRule x:Key="AgeValidationRule"/>
    </UserControl.Resources>
    <StackPanel Margin="10">
        <TextBlock Text="Enter Age:" FontSize="16" FontWeight="Bold" Margin="0,0,0,5"/>

        <!-- Corrected TextBox binding with validation rules -->
        <TextBox x:Name="ageTextBox" Width="200" FontSize="16" FontWeight="Bold" Background="Beige">
            <TextBox.Text>
                <Binding Path="Age" Mode="TwoWay" UpdateSourceTrigger="PropertyChanged">
                    <Binding.ValidationRules>
                        <StaticResource ResourceKey="AgeValidationRule" />
                    </Binding.ValidationRules>
                </Binding>
            </TextBox.Text>
        </TextBox>

        <!-- Display validation error message -->
        <TextBlock Foreground="Red" FontSize="12" Margin="0,5,0,0"
                   Text="{Binding ElementName=ageTextBox, Path=(Validation.Errors)[0].ErrorContent, FallbackValue=''}" />
    </StackPanel>
</UserControl>
```

רושם את המחלקה כמשאב שיהיה ניתן להשתמש בו (Binding).

TextBox להכנסת הגיל בדיקה תיקשרה האם הגיל תקין

במידה והגיל אינו תקין נקבל שגיאה בתוך ה- TextBlock.

Exercise #8



Exercise 8: Master-Detail Binding

1. **Objective**: Bind ListBox and detail view.
2. **Steps**:
3. Bind a list of Book objects to a ListBox.
4. Display selected book details in TextBlocks using SelectedItem.
5. Run the application.
6. **Expected Outcome**: Selecting a book shows its details.

```
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Windows.Controls;

namespace DataBindingExercices.Exe8
{
    2 references
    public partial class Exe8 : UserControl , INotifyPropertyChanged
    {

        9 references
        public class Book
        {
            4 references
            public string Title { get; set; }
            4 references
            public string Author { get; set; }
            4 references
            public int Year { get; set; }
        }

        private ObservableCollection<Book> _books;
        2 references
        public ObservableCollection<Book> Books
        {
            get => _books;
            set
            {
                _books = value;
                OnPropertyChanged(nameof(Books));
            }
        }

        private Book _selectedBook;
        1 reference
        public Book SelectedBook
        {
            get => _selectedBook;
            set
            {
                _selectedBook = value;
                OnPropertyChanged(nameof(SelectedBook));
            }
        }

        0 references
        public Exe8()
        {
            InitializeComponent();
            Books = new ObservableCollection<Book>
            {
                new Book { Title = "The Great Gatsby", Author = "F. Scott Fitzgerald", Year = 1925 },
                new Book { Title = "To Kill a Mockingbird", Author = "Harper Lee", Year = 1960 },
                new Book { Title = "1984", Author = "George Orwell", Year = 1949 },
                new Book { Title = "Pride and Prejudice", Author = "Jane Austen", Year = 1813 }
            };
            DataContext = this;
        }

        public event PropertyChangedEventHandler PropertyChanged;
        2 references
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
        }
    }
```

מחלקה המייצגת ספר עם המאפיינים שלו.

המאפיין מכיל את רשימת הספרים ב- ListBox. משתמש ב- ObservableCollection מודיע אוטומטית למשתמש המשתמש רק הפריטים מתווספים/מוסרים או שהאוסף מתעדכן.

נקראת כאשר Books מושנה, מה שמבטיח שמשתמש המשתמש מתעדכן אם הפנייה לאוסף משתנה.

מאפיין זה מחזיק את הספר שנבחר כרגע ב-ListBox כאשר הבחירה משתנה OnPropertyChanged נקרא, אשר מעדכן את הפרטים ב-StackPanel.

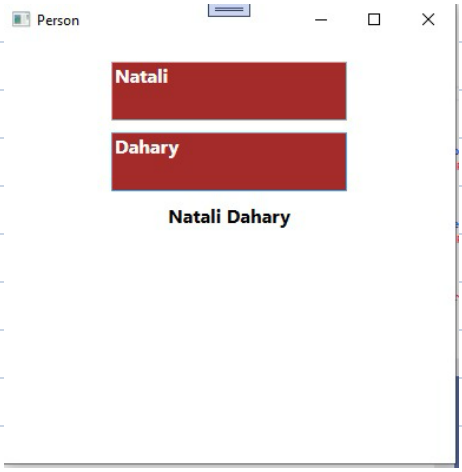
יצירת ספרים לדוגמה.

```
<UserControl x:Class="DataBindingExercices.Exe8.Exe8"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:DataBindingExercices.Exe8"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="2*" />
        </Grid.ColumnDefinitions>

        <ListBox Grid.Column="0" ItemsSource="{Binding Books}" DisplayMemberPath="Title"
                 SelectedItem="{Binding SelectedBook, Mode=TwoWay}" />

        <StackPanel Grid.Column="1" Margin="10">
            <TextBlock FontWeight="Bold" FontSize="16" Text="{Binding SelectedBook.Title}" />
            <TextBlock Text="{Binding SelectedBook.Author}" Margin="0,5,0,0"/>
            <TextBlock Text="{Binding SelectedBook.Year}" Margin="0,5,0,0"/>
        </StackPanel>
    </Grid>
</UserControl>
```

ה- Grid מחולק ל-2 עמודות העמודה הראשונה מכילה את ה- ListBox שמקשר את ה ListBox לאוסף של פריטי ספר ב- backend. זה אומר ל- ListBox לדעת את המאפיין שלה Title של כל ספר ברשימה. הפריט הנבחר ב- ListBox קשור למאפיין SelectedBook ב- backend.

הצגת המאפיינים של כל ספר (כותרת, מחבר, שנה). בתוך StackPanel.

Exercise #9



Exercise 9: Binding with INotifyPropertyChanged

1. **Objective**: Implement INotifyPropertyChanged.
2. **Steps**:
3. Create a Person class with FirstName, LastName, and FullName.
4. Bind FirstName and LastName to TextBoxes.
5. Run the application.
6. **Expected Outcome**: FullName updates when either name changes.

```
<Window x:Class="Exe9.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:Exe9"
        mc:Ignorable="d"
        Title="Person" Height="400" Width="400">
    <StackPanel Margin="10">

        <!-- First Name TextBox -->
        <TextBox Text="{Binding FirstName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                 Width="200" Margin="0,10,0,0" Background="Brown" Height="50" Foreground="White"
                 FontSize="16" FontWeight="Bold"/>

        <!-- Last Name TextBox -->
        <TextBox Text="{Binding LastName, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"
                 Width="200" Margin="0,10,0,0" Background="Brown" Height="50" Foreground="White"
                 FontSize="16" FontWeight="Bold"/>

        <!-- Full Name Display -->
        <TextBlock Text="{Binding FullName}" FontWeight="Bold" FontSize="16" Margin="0,10,0,0" VerticalAlignment="Center"
                   HorizontalAlignment="Center"/>

    </StackPanel>
</Window>
```

מדכין בזמן אמת

FirstName ל- TextBox
LastName -ו
Mode = TwoWay שינוי
בתיבות טקסט מעדכנים את
המאפיין ב- backend
ולהפך.

FullName קשור ל- TextBlock
משלב את השם פרטי שם המשפחה לתחתיו אותו.

```
using System.ComponentModel;
using System.Windows;

namespace Exe9
{
    2 references
    public partial class Person : Window , INotifyPropertyChanged
    {
        private string _firstName;
        private string _lastName;

        2 references
        public string FirstName
        {
            get => _firstName;
            set
            {
                if (_firstName != value)
                {
                    _firstName = value;
                    OnPropertyChanged(nameof(FirstName));
                    OnPropertyChanged(nameof(FullName)); // Update FullName when FirstName changes
                }
            }
        }
        public string LastName
        {
            get => _lastName;
            set
            {
                if (_lastName != value)
                {
                    _lastName = value;
                    OnPropertyChanged(nameof(LastName));
                    OnPropertyChanged(nameof(FullName)); // Update FullName when LastName changes
                }
            }
        }

        2 references
        public string FullName => $"{FirstName} {LastName}";

        public event PropertyChangedEventHandler PropertyChanged;

        4 references
        protected void OnPropertyChanged(string propertyName)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```

מודיע למאזין המתאים
בכל פעם שקובץ שינה.

המחלקה Person מגדירה 3 מאפיינים
FirstName, LastName ו- FullName.

בדיקה אם הערך החדש שונה מהערך
הנוכחי. אם כן שונה נעדכן את המאפיין
ונפעיל את השיטה OnPropertyChanged עבור
על המאפיין וגם על ו- FullName.

בכל פעם שלא פרטי או שם משפחה
מתעדכנים, שם מלא גם מתעדכן
דרך ה- OnPropertyChanged.