WPF Data Binding Exercises

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.

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.

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.

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`.

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.
   .

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.

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.

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.