Different Ways of Raising OnPropertyChanged

OnPropertyChanged in the setter of the property

In this course, we'll mainly use the format `OnPropertyChanged("SomeProperty")` to raise the event that the property value was changed, and that will update the UI.

To avoid having calls to the `OnPropertyChanged("SomeProperty")` in multiple places, what's usually recommended is to have a backing field for the property. That field holds the value of the property. After that field is updated in the setter of the property, call `OnPropertyChanged()`.

Then the property name in the `OnPropertyChanged()` is optional since the method will use the property name from which it's being called.

Example:

```
private string _someProperty;

public string SomeProperty
{
    get => _someProperty;
    set
    {
        _someProperty = value;
        OnPropertyChanged();
    }
}
```

And we update the property:

```
SomeProperty = "John";
```

And that's it!

OnPropertyChanged with an expression

The second approach is to use helper methods that will raise the `OnPropertyChanged` based on the passed expression.

```
public void RaisePropertyChanged<T>(Expression<Func<T>> property)
{
    var name = GetMemberInfo(property).Name;
    OnPropertyChanged(name);
}

private MemberInfo GetMemberInfo(Expression expression)
{
    MemberExpression operand;
    LambdaExpression lambdaExpression = (LambdaExpression)expression;
    if (lambdaExpression.Body as UnaryExpression != null)
    {
        UnaryExpression body =
(UnaryExpression)lambdaExpression.Body;
        operand = (MemberExpression)body.Operand;
    }
    else
    {
        operand = (MemberExpression)lambdaExpression.Body;
    }
    return operand.Member;
}
```

(We usually place these two methods in a class from which all view model classes inherit common behavior. Don't worry, you will find out more about this class later in the course).

What these two methods do is to raise the `OnPropertyChanged` event based on the passed expression. In the expression we state for which property we want to raise the update event.

```
private string _someProperty;

public string SomeProperty
{
    get => _someProperty;
    set
    {
        _someProperty = value;
        RaisePropertyChanged(() => SomeProperty);
    }
}
```

OnPropertyChanged with passing storing field

The third way is to pass the field that stores the actual value of the property to the custom method, which will do all the work for us:

```
protected bool SetProperty<T>(ref T backingStore, T value,
                              [CallerMemberName]string propertyName = "",
                              Action onChanged = null)
{
    if (EqualityComparer<T>.Default.Equals(backingStore, value))
    {
        return false;
    }
    backingStore = value;
    onChanged?.Invoke();

    OnPropertyChanged(propertyName);
    return true;
}
```

The example usage:

```
private string _someProperty;

public string SomeProperty
{
    get => _someProperty;
    set
    {
        SetProperty(ref _someProperty,value);
    }
}
```

The method `SetProperty` does all the work: it sets the `_someProperty` to a new value and raises the OnPropertyChanged event for us.

Neat!

Conclusion

So, which approach you should use? Well, they are all doing the same thing.

The second and the third way are better than the first approach, since the first relies on calling the **OnPropertyChanged** with the string argument, and it's easier to make a mistake when typing the string.