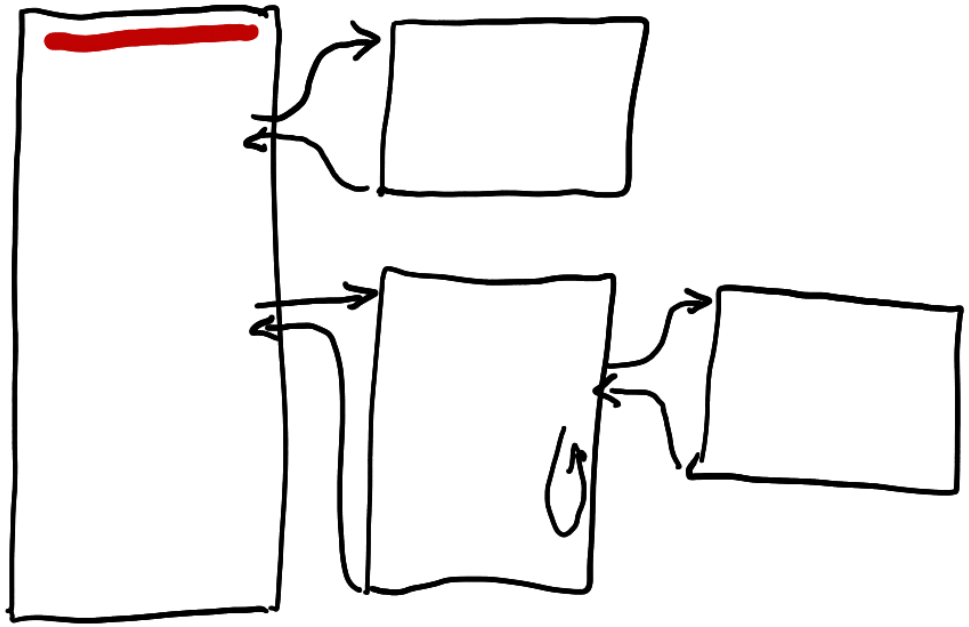
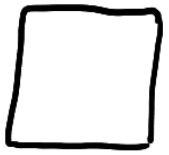
Propertyk, események, INotifyPropertyChanged

Csorba Kristóf

Struktúrált programozás…



Objektum orientált program, események



delegate, event

```
internal void RenderCurrentImage(Area favorite)
{
    LastRenderedArea = CurrentArea.Clone();
    CurrentImage = LastRenderedArea.Render(
        CurrentImageRenderWidth, CurrentImageRenderHeight);
    Notify(nameof(CurrentImage));
    // Notify others in case they want to do something, like
    //  resetting a ScrollViewer which we should not know about here.
    CurrentImageRerendered?.Invoke();
}

// EVIP: defining an event
// Event fired if the CurrentImage has been rerendered and so the
//  ScrollViewer needs to be reset to the default position and zoom level.
public delegate void CurrentImageRerenderedDelegate();
public event CurrentImageRerenderedDelegate CurrentImageRerendered;
```

Mandelbrot\FavoriteMandelbrots\ViewModel\MainViewerViewModel.cs

eventre feliratkozás

```
// EVIP: subscribe to event using method
Transformations.CollectionChanged += Transformations_CollectionChanged;
Transformations.Add(new Interpolation());
// EVIP: subscribe to event using lambda expression
Points.CollectionChanged +=
    (object sender, System.Collections.Specialized.NotifyCollectionChangedEventArgs e)
    => NotifyPropertyChanged(nameof(Points));
```

SpirographLab\SpirographLib\Stroke.cs

Property vs attribútum

```
public class Field : ObservableObject
{
    // EVIP: property without explicit private member
    public int Row { get; set; }
    public int Column { get; set; }
```

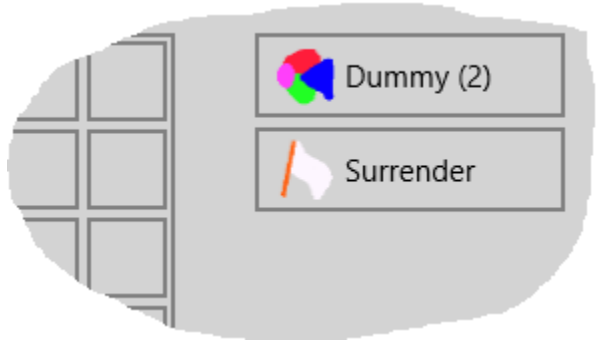
Interfészben, setter nélkül...

```
public interface IBooster : IOperation, ICommand
{
    // Note: needed to set the view model after
    //  constructor (instantiated by Activator
    GameViewModel GameViewModel { get; set; }

    // EVIP: interface does not require setter
    BitmapImage Image { get; }
```

AttaxxPlus\AttaxxPlus\Boosters\IBooster.cs

Property override példa



Ősosztályban (BoosterBase):

```
public abstract string Title { get; }
```

Leszármazottakban:

```
// EVIP: overriding abstract property in base class.
public override string Title { get => $"Dummy ({UsableCounter})"; }
```

```
public class SurrenderBooster : BoosterBase
{
    // EVIP: compact override of getter for Title returning constant.
    public override string Title => "Surrender";
```

AttaxxPlus\AttaxxPlus\Boosters\DummyBooster.cs

Property: további esetek

```
// EVIP: property with protected setter (will be set by derived classes ctor)
public int NumberOfPlayers { get; protected set; }
```

```
// EVIP: property with private member and getter-setter.
private int currentPlayer = 1;
public int CurrentPlayer
{
    get => currentPlayer;
    set
    {
        if (currentPlayer != value)
        {
            currentPlayer = value;
            Notify();
        }
    }
}
```

AttaxxPlus\AttaxxPlus\Model\GameBase.cs

INotifyPropertyChanged (INPC)

```
private bool isSelected = false;
/// <summary>
/// Note: this property is kept up-to-date by GameViewModel.
/// Being a command tool, it has nothing to do with the
/// game rules or game logic.
/// </summary>
public bool IsSelected
{
    get => isSelected;
    set
    {
        // EVIP: setter checking for true change and calling
        //  derived Notify with automatic property name
        if (isSelected != value)
        {
            isSelected = value;
            Notify();
        }
    }
}
```

AttaxxPlus\AttaxxPlus\ViewModel\FieldViewModel.cs

A “Notify”, most az ősosztályban

```
set
{
    // EVIP: setter checking
    //  derived Notify with a
    if (isSelected != value)
    {
        isSelected = value;
        Notify();
    }
}
```

```
// EVIP: base class for INPC, Notify using CallerMemberName
public class ObservableObject : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void Notify([CallerMemberName] string propertyName = "")
    {
        // EVIP: ?. for invoking event. Event is null if there are not subscribers.
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}
```

AttaxxPlus\AttaxxPlus\Model\ObservableObject.cs

```
private int? winner = null;
/// <summary>
/// null means the game is still running.
/// 0 means draw.
/// </summary>
public int? Winner
{
    get => winner;
    private set
    {
        if (winner != value)
        {
            winner = value;
            Notify();
            // EVIP: setter notifies about change of multiple properties
            Notify(nameof(IsGameRunning));
        }
    }
}

// EVIP: read-only property as body expression
public bool IsGameRunning => Winner is null;
```

AttaxxPlus\AttaxxPlus\Model\GameBase.cs

Modell lecserélése, leiratkozás

```
internal void UpdateModel(Area newFavorite)
{
    Model.PropertyChanged -= Model_PropertyChanged;
    Model = newFavorite.Clone();
    Model.PropertyChanged += Model_PropertyChanged;
    // EVIP: all properties may have changed,
    //  so we send notification about all of them.
    NotifyAllPropertiesChanged();
}

public void NotifyAllPropertiesChanged()
{
    Model.NotifyAllPropertiesChanged();
}
```

Mandelbrot\FavoriteMandelbrots\ViewModel\AreaViewModel.cs

Egyebek: INPC továbbítása

```
public class FieldViewModel : ObservableObject
{
    public FieldViewModel(Field model)
    {
        Model = model;
        // EVIP: VM observes Model and translates PropertyChanged events
        model.PropertyChanged += Model_PropertyChanged;
    }

    private void Model_PropertyChanged(object sender,
        System.ComponentModel.PropertyChangedEventArgs e)
    {
        // EVIP: forwarding INPC
        // EVIP: nameof operator
        if (e.PropertyName == nameof(Field.Owner))
            Notify(nameof(Owner));
    }

    // EVIP: read-only property. Needs to emit (forward)
    //  PropertyChanged if underlying model changes.
    public int Owner { get => Model.Owner; }
```

AttaxxPlus\AttaxxPlus\ViewModel\FieldViewModel.cs

Egyebek: INPC továbbítása

```
public readonly GameBase Model;

public GameViewModel(GameBase model) : base()
{
    this.Model = model;
    Model.PropertyChanged += Model_PropertyChanged;
```

```
public int CurrentPlayer { get => Model.CurrentPlayer; }
public int? Winner { get => Model.Winner; }
public bool IsGameRunning { get => Model.IsGameRunning; }

#region INPC forwarding
// EVIP: INPC forwarding for a list of properties stored in a string[].
private readonly string[] dependentPropertyNames =
    { nameof(GameBase.Winner), nameof(GameBase.IsGameRunning),
      nameof(GameBase.CurrentPlayer) };

private void Model_PropertyChanged(object sender,
    System.ComponentModel.PropertyChangedEventArgs e)
{
    // Note: forwarding INPC of wrapped model properties.
    // Using array instead of a list of "if" conditions.
    if (dependentPropertyNames.Contains(e.PropertyName))
        Notify(e.PropertyName);
}
#endregion
```

AttaxxPlus\AttaxxPlus\ViewModel\GameViewModel.cs