C# alapok

Csorba Kristóf

Előzetes online anyagok (ism.)

A .NET mint keretrendszer

Ism.: KonzolosHelloWorld (lab)

- Assembly (külön leforduló projekt)
- Namespace és using, Class, Interface, öröklés, láthatóságok (private, protected, internal, public)
- érték és referencia típusok

Visual Studio

- Code Lens
- Ctrl+. (fix usings, refactorings)
- F1 (help), F12 (goto definition)
- Snippets, pl. “ctor” + Tab-Tab

```
using System.Collections.Generic;
using System.ComponentModel;

namespace SpirographLib.Transformations
{
    9 references | Kristof, 175 days ago | 1 author, 1 change
    public abstract class Transformation : ObservableBase
    {
        3 references | Kristof, 175 days ago | 1 author, 1 change
        public abstract IEnumerable<SPoint> Transform(IEnumerable<SPoint> points);
    }
}
```

```
public class FieldCommand : CommandBase
{
    // EVIP: readonly, assigned only in ctor
    private readonly GameViewModel vm;

    private readonly CloneMoveOperation cloneMove;
    private readonly JumpOperation jump;

    public FieldCommand(GameViewModel vm) : base()
    {
        this.vm = vm;
        cloneMove = new CloneMoveOperation(vm.Model);
        jump = new JumpOperation(vm.Model);
    }
```

AttaxxPlus\AttaxxPlus\ViewModel\FieldCommand.cs

```
// EVIP: Constant attributes to avoid hardwired and redundant
//  (and hard-to-read) numeric constants throughout the code.
public const int CurrentImageRenderWidth = 1000;
public const int CurrentImageRenderHeight = 1000;
```

Mandelbrot\FavoriteMandelbrots\ViewModel\MainViewerViewModel.cs

```
// EVIP: array initializer
// EVIP: highly specialized assert method
AssertOpacities(new double[] { 0, 0, 0 });
```

svg2pptx\Tests\CommandManagementTests.cs

```
private int? winner = null;
```

AttaxxPlus\AttaxxPlus\Model\GameBase.cs

```
public bool AppliesToFrame(int frame)
{
    // EVIP: default value if null
    // EVIP: int.MaxValue
    return ((FirstFrame ?? 0) <= frame
        && frame <= (LastFrame ?? int.MaxValue));
}
```

svg2ppts/AnimationCommands/AnimationCommand.cs

```
// EVIP: equality check of doubles using threshold
Assert.AreEqual(expectedOpacity, opacity, 0.01);
```

svg2pptx\Tests\CommandManagementTests.cs

```
// EVIP: using out and variable declaration together.
Assert.IsTrue(double.TryParse(opacityString, out double opacity));
```

svg2pptx\Tests\CommandManagementTests.cs

```
// EVIP: compact object initialization.
rowList.Add(
    new FieldViewModel(Model.Fields[row, col])
        { FieldCommand = fieldCommand }
    );
```

AttaxxPlus\AttaxxPlus\ViewModel\GameViewModel.cs

```
// EVIP: Elvis operator in chain
if (TryExecute(GameViewModel.SelectedField?.Model, null))
    GameViewModel.EndOfTurn();
```

AttaxxPlus\AttaxxPlus\Boosters\BoosterBase.cs

```
// EVIP: interface implementing several other interfaces
// EVIP: for compactness, a Booster is both view model and command.
public interface IBooster : IOperation, ICommand, INotifyPropertyChanged
{
    // Note: needed to set the view model after calling the parameterless
    //  constructor (instantiated by Activator using reflection)
    GameViewModel GameViewModel { get; set; }

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

    string Title { get; }

    void InitializeGame();
}
```

AttaxxPlus\AttaxxPlus\Boosters\IBooster.cs

```
// EVIP: virtual method and not abstract.
// It has a default not doing anything,
//  so override is not mandatory.
public virtual void InitializeGame() { }
```

AttaxxPlus\AttaxxPlus\Boosters\BoosterBase.cs

```
// EVIP: unified interface for all operations in the game
public interface IOperation
{
    /// <summary>
    /// Tries to execute this operation
    /// </summary>
    /// <param name="selectedField">A field previously selected by the
    /// <param name="currentField">The field currently clicked</param>
    /// <returns>True if executed. False if cannot execute (no side ef
    bool TryExecute(Field selectedField, Field currentField);
}
```

AttaxxPlus\AttaxxPlus\Model\Operations\IOperation.cs

```
// EVIP: abstract base class implementing an interface
//  and providing helper methods and game model reference.
public abstract class OperationBase : IOperation
{
    // EVIP: base class stores dependency in protected field.
    protected GameBase game;
    public OperationBase(GameBase game)
    {
        this.game = game;
    }

    // EVIP: abstract method
    public abstract bool TryExecute(Field selectedField, Field currentField);
```

AttaxxPlus\AttaxxPlus\Model\Operations\OperationBase.cs

```
// EVIP: deriving from GameBase to specify exact situation and startup playing field.
public class SimpleGame : GameBase
{
    public SimpleGame(int size) : base()
    {
        // Note: InitializeGame expects these objects already created.
        // EVIP: instantiating and initializing 2D array (row, column)
        Fields = new Field[size, size];
        for (int row = 0; row < size; row++)
            for (int col = 0; col < size; col++)
                Fields[row, col] = new Field() { Row = row, Column = col, Owner = 0 };
        // EVIP: setting property with protected setter
        NumberOfPlayers = 2;

        InitializeGame();
    }
```

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

AttaxxPlus\AttaxxPlus\Model\SimpleGame.cs

```
namespace AttaxxPlus.ViewModel
{
    // EVIP: default implementation for an interface
    public abstract class CommandBase : ICommand
    {
        // EVIP: temporarily disable a warning
#pragma warning disable 67
        public event EventHandler CanExecuteChanged;
#pragma warning restore 67
        public bool CanExecute(object parameter) => true;
        public abstract void Execute(object parameter);
    }
}
```

AttaxxPlus\AttaxxPlus\ViewModel\CommandBase.cs

```
public class AreaViewModel : ObservableObject
{
    public AreaViewModel(Area model, MainViewerViewModel mainViewerVM)
    {
        this.Model = model;
        model.PropertyChanged += Model_PropertyChanged;
        ShowCommand = new ShowInMainViewerCommand(model, mainViewerVM);
    }

    // EVIP: Nested class (command not meant to be used elsewhere)
    public class ShowInMainViewerCommand : CommandBase
    {
        public ShowInMainViewerCommand(Area model, MainViewerViewModel mainViewerVM)
            : base(mainViewerVM)
        {
            this.model = model;
        }

        private Area model;
```

Mandelbrot\FavoriteMandelbrots\ViewModel\AreaViewModel.cs

```
// EVIP: More complex ctor chaining
public SvgWrapper(string filename)
    : this(XElement.Load(filename))
{
}
```

svg2pptx\svg2pptx\SvgWrapper.cs

```
public class VisibleCommand : AnimationCommand
{
    public const double Opacity = 0.4;

    // EVIP: ctor calling base class ctor and passing over the parameters.
    public VisibleCommand(int? param, int? firstFrame, int? lastFrame)
        : base(param, firstFrame, lastFrame)
    {
    }

    public override void Apply(SvgWrapper svgWrapper, XElement element, int frameIndex)
    {
        // EVIP: overridden method should call base class implementation!
        //  Even if it does not do anything. Later someone may change that.
        base.Apply(svgWrapper, element, frameIndex);
        if (AppliesToFrame(frameIndex))
            element.SetStyle("opacity", Opacity.ToString("F2"));

        // We cannot set opacity to 0.0 here, as there may be multiple VisibleCommands
    }
}
```

svg2pptx\svg2pptx\AnimationCommands\VisibleCommand.cs

```
// EVIP: simple access to a given frequently used attribute via extension method.
var style = svg.GetElementWithLabel(elementLabel).Style().Value;
```

```
private const string styleAttributeName = "style";
public static XAttribute Style(this XElement element)
{
    XAttribute s = element.Attribute(styleAttributeName);
    if (s is null)
    {
        s = new XAttribute(XName.Get("style"), "");
        element.Add(s);
    }
    return s;
}
```

svg2pptx\Tests\CommandManagementTests.cs
e:\Projektek\evipdev\svg2pptx\svg2pptx\XElementSvgExtensionMethods.cs

```
// EVIP: operator overload
public static bool operator==(SPoint obj1, SPoint obj2)
{
    return obj1.Equals(obj2);
}
```

```
private const double epsilon = 0.001;
public override bool Equals(object obj)
{
    SPoint other = obj as SPoint;
    if (other is null)
        return false;
    return (Math.Abs(X-other.X)<epsilon
        && Math.Abs(Y-other.Y)<epsilon
        && Math.Abs(T-other.T)<epsilon
        && LineWidth == other.LineWidth
        && Color.R == other.Color.R
        && Color.G == other.Color.G
        && Color.B == other.Color.B);
}
```

SpirographLab\SpirographLib\SPoint.cs

```
public class AnimationCommand
{
    // EVIP: getter-only property initialized in ctor (like readonly)
    public int? Param { get; }
    public int? FirstFrame { get; }
    public int? LastFrame { get; }

    public AnimationCommand(int? param, int? firstFrame, int? lastFrame)
    {
        Param = param;
        FirstFrame = firstFrame;
        LastFrame = lastFrame;
    }
```

svg2ppts/AnimationCommands/AnimationCommand.cs

```
// EVIP: property with further setter responsibilities.
private FieldViewModel selectedField;
public FieldViewModel SelectedField
{
    get => selectedField;
    set
    {
        if (selectedField != value)
        {
            // Note: both old and new value can be null!
            if (selectedField != null)
                selectedField.IsSelected = false;
            selectedField = value;
            if (selectedField != null)
                selectedField.IsSelected = true;
            Notify();
        }
    }
}
```

AttaxxPlus\AttaxxPlus\ViewModel\GameViewModel.cs

```
public int GetLastFrameIndex()
{
    // EVIP: SelectMany to concatenate arrays in dictionary values.
    var allCommands = Commands.SelectMany(keyValuePair => keyValuePair.Value).ToArray();
    int lastFrameIndex = -1;

    // EVIP: local function having access to local variables of parent method
    void registerFrameIndex(int? frameIndex)
    {
        if (frameIndex.HasValue && lastFrameIndex < frameIndex)
            lastFrameIndex = frameIndex.Value;
    }

    foreach(var cmd in allCommands)
    {
        // Without nested method, registerFrameIndex would need a
        //  class level attribute to store the maximum (or a ref parameter).
        registerFrameIndex(cmd.FirstFrame);
```

svg2pptx\svg2pptx\SvgWrapper.cs

```
protected SPoint InterpolatePoint(SPoint a, SPoint b, double t)
{
    // EVIP: throwing exception
    if (a.T > t || b.T < t)
        throw new ArgumentException(
            "t must be between timestamp of 'a' and 'b' to interpolate.");
    SPoint p = new SPoint();
```

SpirographLab\SpirographLib\Transformations\Interpolation.cs

```
// EVIP: initializing a dictionary with values
readonly Dictionary<InitialShapeEnum, SPoint[]> initialShapes =
    new Dictionary<InitialShapeEnum, SPoint[]>()
    {
        { InitialShapeEnum.Rectangle, new SPoint[] {
            new SPoint(50,50,0.0),
            new SPoint(450,50,0.25),
            new SPoint(450,450,0.5),
            new SPoint(50,450,0.75),
            new SPoint(50,50,1.0)
            }
        },
        { InitialShapeEnum.Triangle, new SPoint[] {
            new SPoint(50,50,0.0),
            new SPoint(450,50,0.33),
            new SPoint(50,450,0.67),
            new SPoint(50,50,1.0)
            }
        }
    };
```

```
// EVIP: enum
public enum InitialShapeEnum
{
    Rectangle, Triangle
}
```

SpirographLab\SpirographViewer\ViewModel\StrokeVM.cs

OPC

- Tesztelés alapjai
- Continuous Integration (CI) jelentése