Eötvös Loránd Tudományegyetem
Informatikai Kar

Eseményvezérelt alkalmazások

---

11. előadás

Összetett Avalonia UI alkalmazások fejlesztése

Cserép Máté
mcserep@inf.elte.hu
https://mcserep.web.elte.hu

Összetett Avalonia UI alkalmazások

Időzítés

- Felületi időzítésre használhatjuk a platformfüggő időzítőt (**`DispatcherTimer`**), amely nem egyezik a WPF azonos nevű időzítőjével
  - megadhatjuk az időintervallumot (**`Interval`**), és az időzítésre (**`Tick`**) elvégzendő tevékenységet.
- Felületfüggetlen időzítésre továbbra is használható a **`System.Timers.Timer`** típus
  - az időzítő egy háttérszálon váltja ki az eseményt, az Avalonia UI keretrendszer esetében is igaz, hogy a felületi vezérlőkhöz csak az őket létrehozó szálról férhetünk hozzá
  - használjuk a **`Dispatcher.UIThread.InvokeAsync()`** hívást a szálak közötti szinkronizáláshoz

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk Avalonia UI felülettel egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- a modell (**`ExamGeneratorModel`**) valósítja meg a generálást, tétel elfogadást/eldobást, valamint a történet tárolását, a modellre egy interfészen keresztül (**`IExamGenerator`**) hivatkozunk
- két nézetet hozunk létre, egyik a generáló nézet (**`MainView`**), a másik a beállítási nézet (**`SettingsView`**), ezeket ablakokként is tegyük elérhetővé (**`MainWindow`**, **`SettingsWindow`**)
- a két nézetet ugyanaz a nézetmodell (**`ExamGeneratorViewModel`**) szolgálja ki, amelybe befecskendezzük a modellt

Összetett Avalonia UI alkalmazások

Példa

- a nézetmodell tárolja a start/stop funkcióért, beállítások megnyitásáért és bezárásáért felelős utasításokat
- a nézetmodell kezeli a modell eseményét (**`NumberGenerated`**), és frissíti a megjelenített számot
- a nézetmodell egy listában tárolja a kihúzott tételeket (**`History`**), ehhez létrehozunk egy segédtípus (**`HistoryItem`**), amely tárolja az elem sorszámát, illetve az állapotát (kiadható, vagy sem), ezeket a tulajdonságokat kötjük a nézetre
- az alkalmazás (**`App`**) felel az egyes rétegek példányosításáért, valamint a nézetmodell események kezeléséért

- Megfigyelhetjük, hogy a modell és nézetmodell rétegek a feladat korábbi változataival megegyeznek.

Összetett Avalonia UI alkalmazások

Alkalmazások környezete

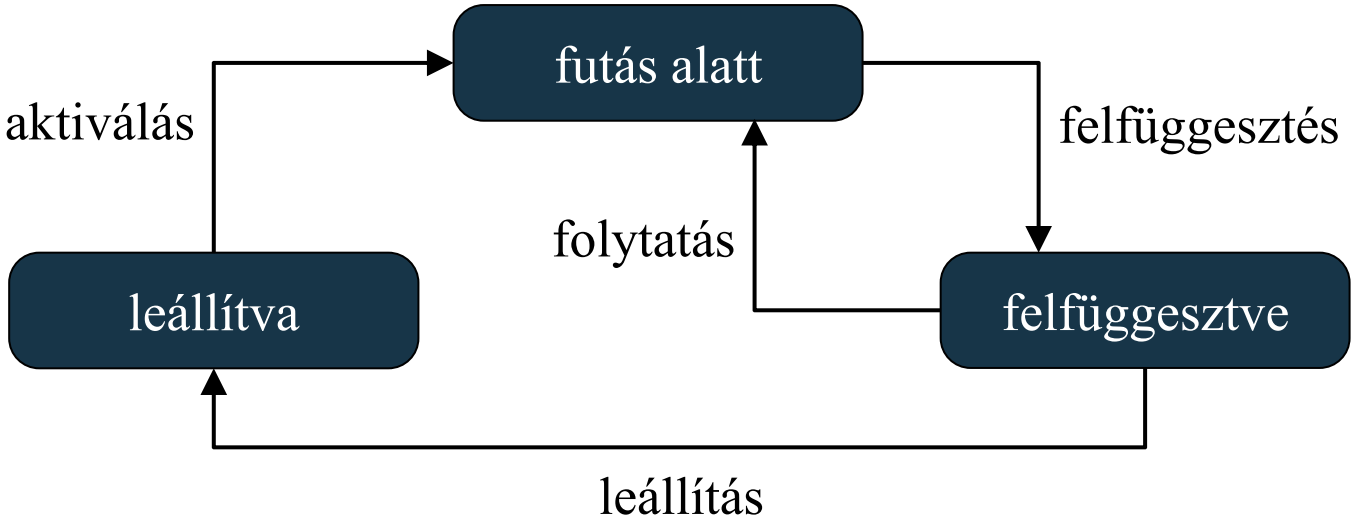
- Az alkalmazások egy biztonságos környezetben futnak
  - nem férhetnek hozzá más alkalmazások adataihoz
  - csak korlátozott módon férhetnek hozzá a rendszer adataihoz (pl. fájlrendszer), és azt is csak engedéllyel
  - szintén engedéllyel használhatják csak az eszközöket (*application manifest*)

- Mindegyik alkalmazás számára rendelkezésre áll egy lokális könyvtár, amely csak az alkalmazás fájljait tárolja
  - Ennek ellenéréséhez nem szükséges külön jogosultság semelyik támogatott platformon
  - Alkalmazás törlésekor a hozzá tartozó adatok is törlődnek

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- A mobil alkalmazások más életciklusban futnak, mint az asztali alkalmazások
  - a *futás alatt (running)* és a *terminált (not running)* állapotok mellett megjelenik a *felfüggesztett* (*suspended*) állapot is, amely akkor lép életbe, ha az alkalmazás a háttérbe (vagy a gép alvó állapotba) kerül, célja a takarékosság



Összetett Avalonia UI alkalmazások

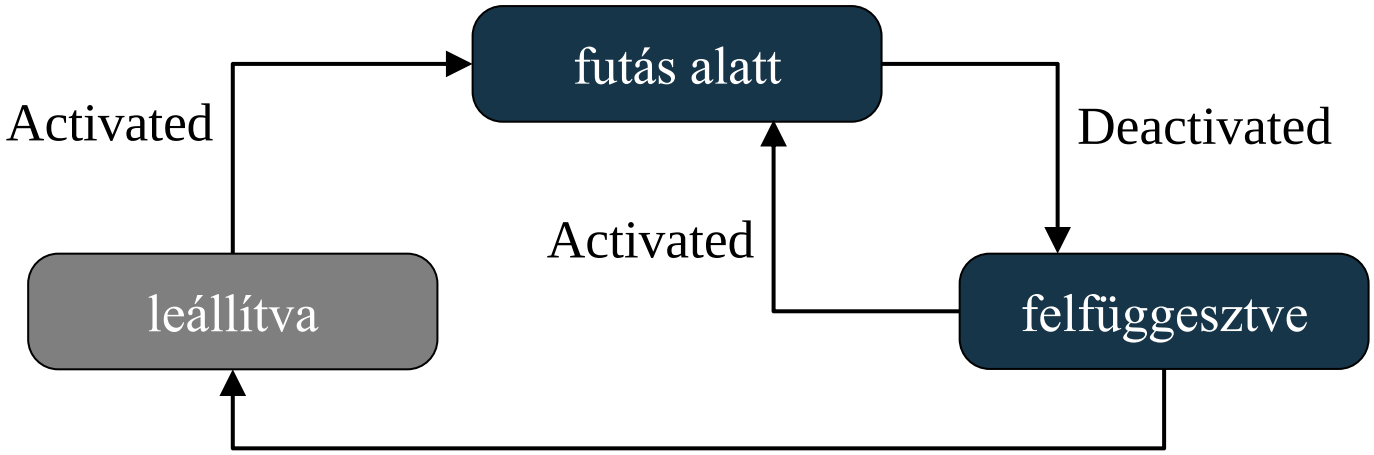
Alkalmazások életciklusa

- A felfüggesztés célja az erőforrásokkal való takarékoskodás
  - a fejlesztőnek törekednie kell rá, hogy felfüggesztett állapotban az alkalmazás minél kevesebb erőforrást igényeljen
  - a futó tevékenységeket célszerű leállítani, az adatokat perzisztálni
- A rendszer úgy is dönthet (pl. ha kevés a memória), hogy a felfüggesztett alkalmazást leállítja, majd újraindítja, ha a felhasználó visszaváltott rá
  - célszerű, hogy a felhasználó ennek ellenére olyan állapotban kapja vissza az alkalmazást, amelyben hagyta, így ezt az állapotot vissza kell állítanunk
  - az állapot eltárolását felfüggesztéskor kell elvégeznünk

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Az Avalonia UI keretrendszerű mobil alkalmazások egységes életciklus kezeléssel rendelkeznek:
  - az **`App`** osztályunk **`OnFrameworkInitializationCompleted`** metódusát felüldefiniálva, eseménykezelőkkel adható meg az életciklus váltáskor végrehajtandó tevékenység (**`Activated`**, **`Deactivated`**), minden platformra (Android, iOS, macOS).



Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Pl.:

```
if (Application.Current
      .TryGetFeature<IActivatableLifetime>()
      is { } activatableLifetime)
{
    activatableLifetime.Activated += (sender, args) =>
    {
       // alkalmazás aktív
    };
    activatableLifetime.Deactivated += (sender, args) =>
    {
       // alkalmazás inaktív
    };
}
```

Összetett Avalonia UI alkalmazások

Alkalmazások életciklusa

- Asztali alkalmazásoknál Windows és Linux operációs rendszerek esetében hasonló módon az alkalmazás elindítása és leállítása kezelhető, mint életciklus események
  - az **`App`** osztályunk **`OnFrameworkInitializationCompleted`** metódusát felüldefiniálva, eseménykezelőkkel adható meg az alkalmazás elindításakor és leállításakor (**`Startup, Exit`**) végzendő tevékenység.
  - Pl.:

```
desktop.Startup += (sender, args) =>
{
    // ...
};
```

Összetett Avalonia UI alkalmazások

Platformfüggetlen perzisztencia

- A **`System.IO`** névtérben korábban már megismert osztályok és eljárások a fájlkezelés összes lehetőségét biztosítják
  - könyvtárak (**`Directory`**) elérését, listázását (**`GetFiles`**, **`GetDirectories`**), benne könyvtárak létrehozását (**`CreateDirectory`**), fájlok és könyvtárak törlését (**`Delete`**)
  - fájlok (**`File`**) létrehozását, megnyitását (**`Open`**), olvasást (**`ReadAllBytes`**, **`ReadAllLines`**, **`ReadAllText`**), írást (**`WriteAllBytes`**, …), másolást (**`Copy`**), …
  - az írás és olvasás történhet adatfolyamok segítségével is (**`StreamReader`**, **`StreamWriter`**), amelyeket a megszokott módon használhatunk

Összetett Avalonia UI alkalmazások

Platformfüggetlen perzisztencia

- Speciális, jól ismert útvonalakat az **`Environment.SpecialFolder`** *enum* segítségével kérhetünk le
  - ügyelni kell arra, hogy az egyes útvonalak platformfüggően mást jelenthetnek, pl.:
    - Dokumentumok könyvtár asztali operációs rendszereken:
      ```
      Environment.GetFolderPath(
        Environment.SpecialFolder.MyDocuments)
      ```
    - alkalmazás saját adatkönyvtára mobil platformokon:
      ```
      Environment.GetFolderPath(
        Environment.SpecialFolder.
        LocalApplicationData)
      ```
      - ez az útvonal asztali operációs rendszeren azonban nem alkalmazás specifikus

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy vizsgatétel generáló alkalmazást, amely ügyel arra, hogy a vizsgázók közül ketten ne kapják ugyanazt a tételt.

- kiegészítjük életciklus kezeléssel az alkalmazást, eltároljuk a modell állapotát, valamint a generálás állapotát az alkalmazás lokális könyvátárba JSON formátumban szerializálva
- mivel nincs külön perzisztencia, közvetlenül a modelltől kérjük el az információkat, és tároljuk el egyenként
- az alkalmazás életciklusát az **`Application`** vezérli, ezért itt definiálunk egy **`SaveState`** és egy **`LoadState`** eljárást
- indításkor, illetve folytatáskor a korábbi állapotot betöltjük, és ennek megfelelően inicializáljuk a modellt

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- a megjelenítést azáltal grafikus alakzatokkal (**`Line`**, **`Ellipse`**, **`Rectangle`**) valósítsuk meg
- ugyanakkor továbbra is gombokat jelenítünk meg (amely kattintható), de felüldefiniáljuk a sablont (**`Template`**) egy egyedi felépítéssel (**`ControlTemplate`**), így a gomb megjelenése teljesen más lesz
  - definiáljunk stílusokat (**`Empty`**, **`X`**, **`O`**), amelyeket a nézetmodell **`TicTacToeField`** osztályának 1-1 logikai tulajdonságához köthetünk (**`IsEmpty`**, **`IsX`**, **`IsO`**)
- a mentés egy beégett útvonalon legyen lehetséges; az alkalmazás leállításakor / háttérbe kerülésekor automatikusan mentsünk

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- Avalonia UI keretrendszerben platformfüggetlen módon kérhetünk fájl betöltéséhez és mentéséhez dialogóusablakot
  - ehhez a **`StorageProvider`** **`OpenFilePickerAsync()`** és **`SaveFilePickerAsync()`** metódusait használhatjuk
    - betöltésnél megadhatjuk, hogy több fájl is kiválasztható-e (**`AllowMultiple`**)
    - mentésnél megadhatjuk a javasolt fájlnevet (**`SuggestedFileName`**) és alapértelmezett kiterjesztést (**`DefaultExtension`**)
    - választható fájltípusokat, ehhez elérhetőek előre definiált beállítások (pl. **`FilePickerFileTypes.ImageJpg`**), de saját is megadható
  - hasonlóan kérhetjük könyvtár tallózását is a **`OpenFolderPickerAsync()`** eljárás használatával

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- A **`StorageProvider`** osztály ún. szolgáltatásként (*service*) érhető el az Avalonia UI keretrendszerben
  - a szolgáltatások a **`TopLevel`** vezérlőn keresztül érhetőek el
  - az ablakok, nézetek, és felületi vezérlők egymásba ágyazásával a vezérlők egy fa struktúrát alkotnak (*visual tree*)
  - ezen fa struktúra gyökér eleme a **`TopLevel`** vezérlő
  - asztali ablakos alkalmazásoknál a **`Window`** a **`TopLevel`**
  - mobil alkalmazásoknál platform specifikus lehet, pl. Android esetén a **`MainActivity`** lesz az
  - platformfüggetlen módon bármely vezérlőhöz lekérdezhető: **`TopLevel.GetTopLevel(control);`**

Összetett Avalonia UI alkalmazások

Dialógus ablakok

- Pl.:

```
var files = await
TopLevel.StorageProvider.OpenFilePickerAsync(
    new FilePickerOpenOptions
    {
        Title = "Select file to load",
        AllowMultiple = false,
        FileTypeFilter = new[]
        {
            new FilePickerFileType("Data file")
            {
                Patterns = new[] { "*.data" }
            }
        }
    });
```

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy Tic-Tac-Toe programot, amelyben két játékos küzdhet egymás ellen.

- a mentéshez az útvonalat a felhasználó választhassa meg
  - csak a **`.data`** kiterjesztésű fájlok mentését és betöltését kínáljuk fel
  - Android operációs rendszer támogatásához az **`AndroidManifest.xml`** állományban a szükséges **`READ_EXTERNAL_STORAGE`** és **`WRITE_EXTERNAL_STORAGE`** jogosultságokat kérjük meg
- az alkalmazás leállításakor / háttérbe kerülésekor továbbra is automatikusan készítsünk egy mentést

Összetett Avalonia UI alkalmazások

Animációk

- A WPF keretrendszerben megismertekhez hasonló módon támogatott
  - stílusként (**`Styles`**) adható meg a nézet vagy vezérlő szintjén
  - Pl. áttetszőség és forgatás egy időben történő animálása:

```
<Animation Duration="0:0:3" IterationCount="4">
  <KeyFrame Cue="0%">
     <Setter Property="Opacity" Value="0.0"/>
     <Setter Property="RotateTransform.Angle"
             Value="0.0"/>
  </KeyFrame>

  <KeyFrame Cue="100%">
    <Setter Property="Opacity" Value="1.0"/>
    <Setter Property="RotateTransform.Angle"
            Value="90.0"/>
  </KeyFrame>
</Animation>
```

Összetett Avalonia UI alkalmazások

Átmenetek

- Avalonia UI keretrendszerben átmeneteket is alkalmazhatunk,
  - ezek működését leginkább a webfejlesztésből ismert CSS animációkhoz hasonlíthatjuk
  - a tulajdonság változása nem pillanatszerű, hanem animált lesz

```
<Window.Styles>
  <Style Selector="Rectangle.red">
    <Setter Property="Fill" Value="Red"/>
    <Setter Property="Opacity" Value="0.5"/>
  </Style>
  <Style Selector="Rectangle.red:pointerover">
    <Setter Property="Opacity" Value="1"/>
  </Style>
</Window.Styles>

<Rectangle Classes="red">
  <Rectangle.Transitions>
    <Transitions>
      <DoubleTransition Property="Opacity" Duration="0:0:0.2"/>
    </Transitions>
   </Rectangle.Transitions>
</Rectangle>
```

Összetett Avalonia UI alkalmazások

Példa

Feladat: Készítsünk egy dinamikus méretezhető táblát, amely három szín között (piros, fehér, zöld) állítja a kattintott gombot, valamint a vele egy sorban és oszlopban lévőket.

- a színt a nézet adja meg, így a nézetmodell nem adhat vissza konkrét színt, csak egy sorszámot (0 és 2 között), amely alapján a szín állítható (`ColorNumber`)
- a színt stílus osztályok segítségével állítsuk a nézetben, a gomb emellett animálódjon, amennyiben a kurzort rávisszük (csak asztali környezetben), illetve ha lenyomva tartjuk
- a stílusokat a nézet erőforrásaként hozzuk létre

Összetett Avalonia UI alkalmazások

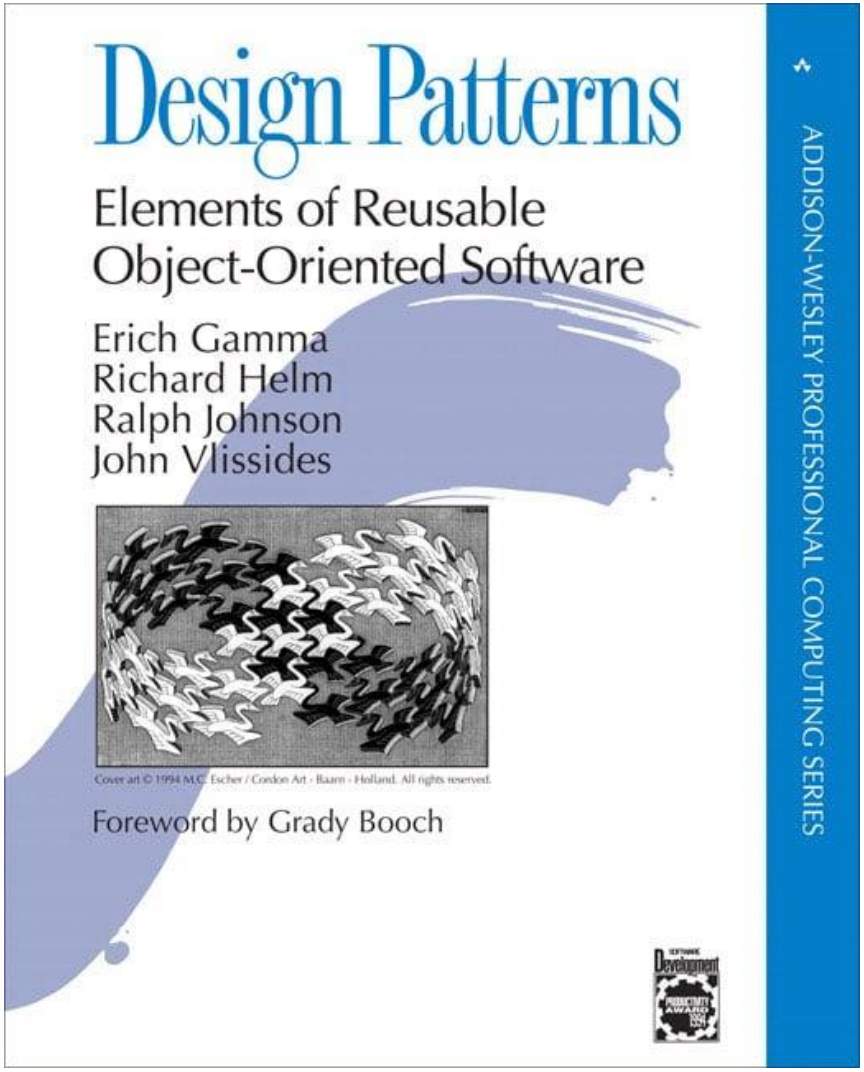
Reaktív programozás

- Az *Avalonia UI* támogatja a *reaktív programozás*t a *ReactiveUI* kerentrendszer használatával (https://www.reactiveui.net/)
  - Az pedig az *Rx.NET*-re épül (*Reactive Extensions for .NET*) https://reactivex.io/
- Mi az a reaktív programozás?
  - *"Reactive programming is a declarative programming paradigm that is based on the idea of asynchronous event processing and data streams."*
- Példa:

```
int a = 5; int b = 10;
int c = a + b;
Console.WriteLine($"c = {c}");
a = 20;
Console.WriteLine($"c = {c}");
```

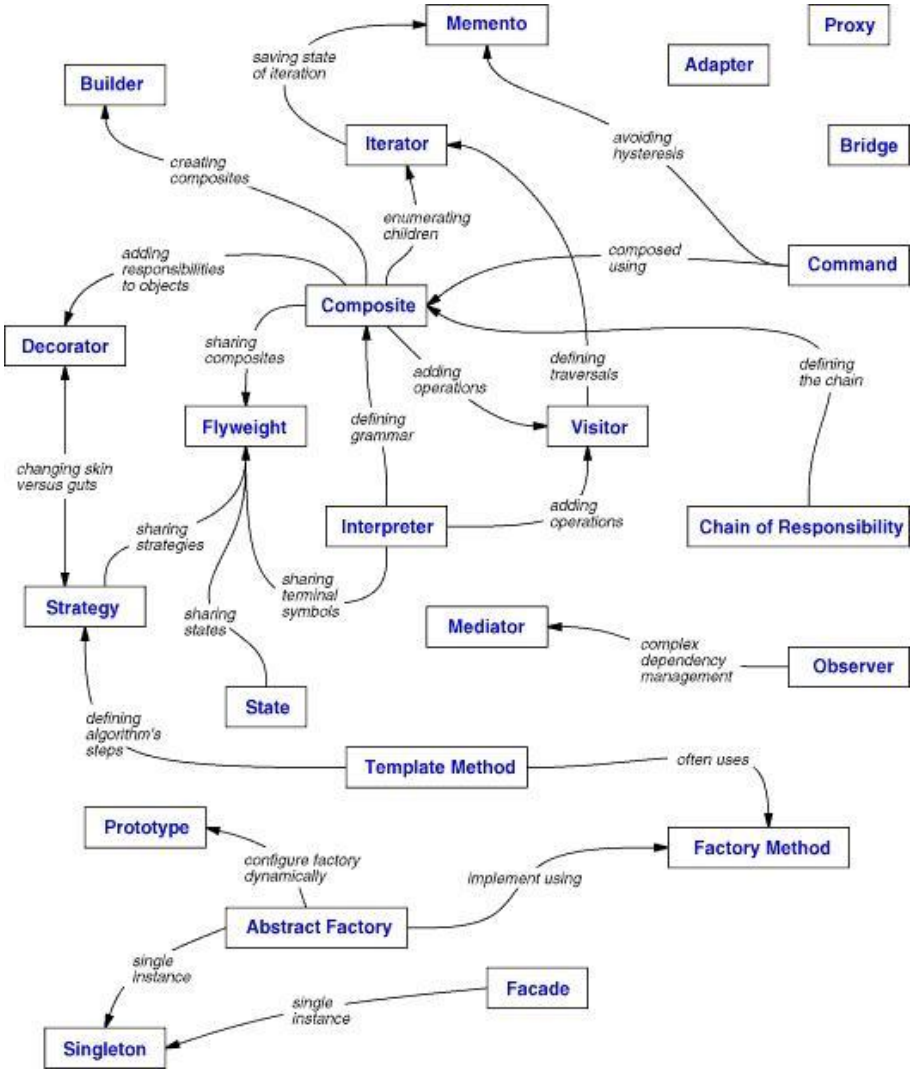
Összetett Avalonia UI alkalmazások

Tervezési minták



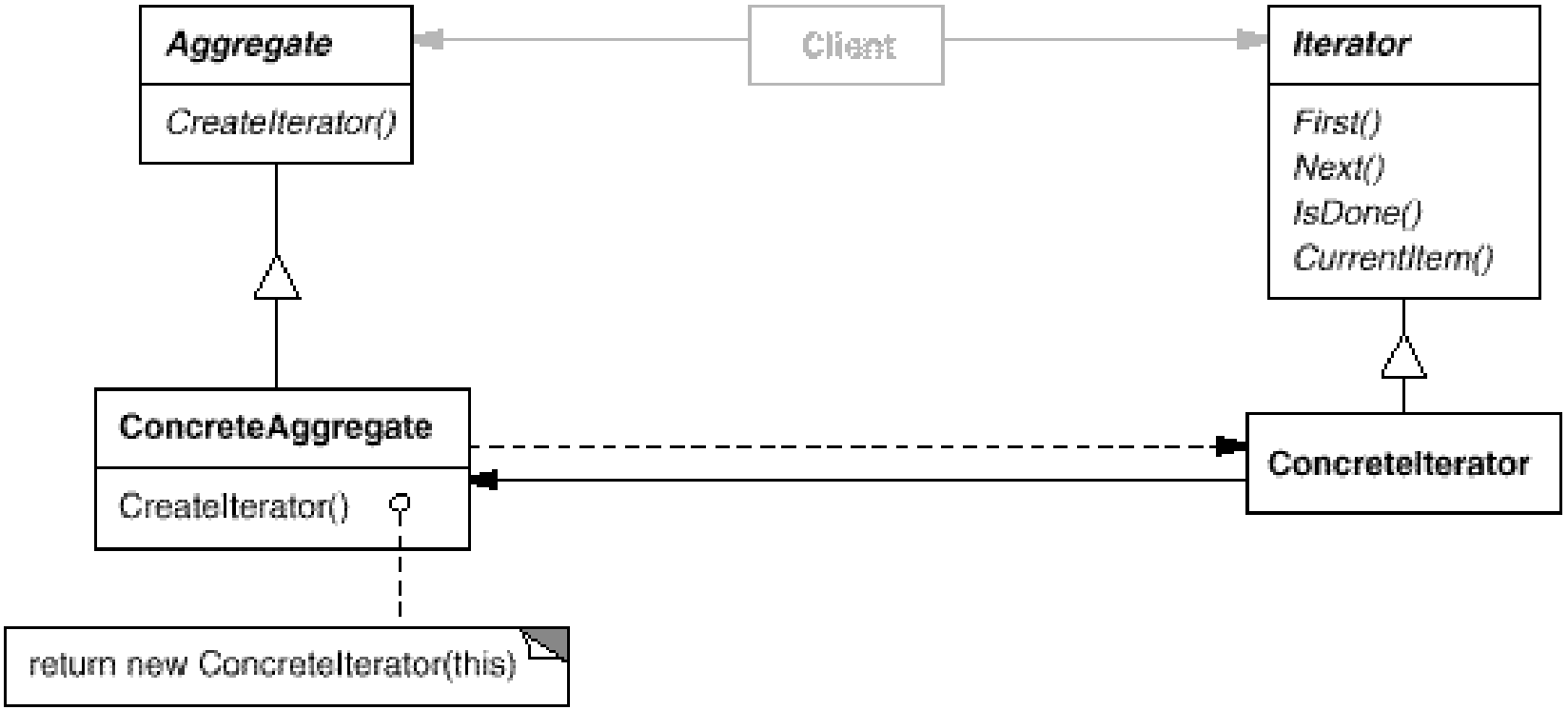
Összetett Avalonia UI alkalmazások

Tervezési minták



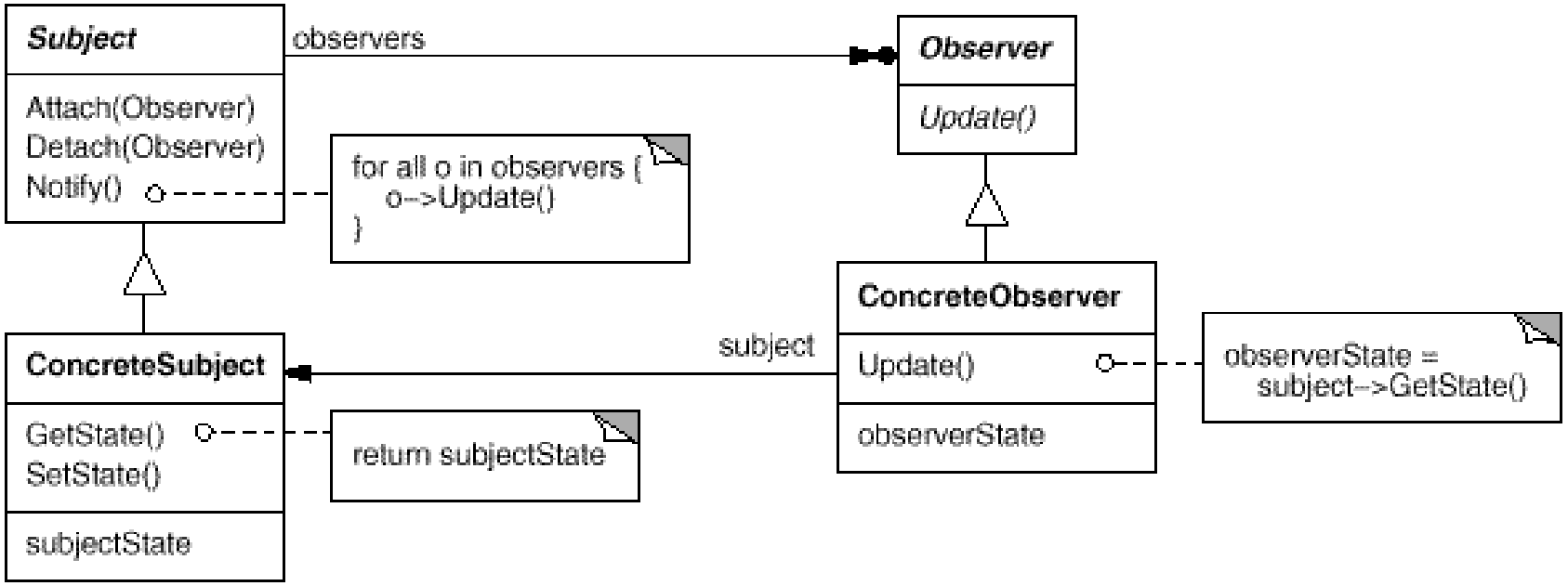
Összetett Avalonia UI alkalmazások

Iterator tervezési minta



Összetett Avalonia UI alkalmazások

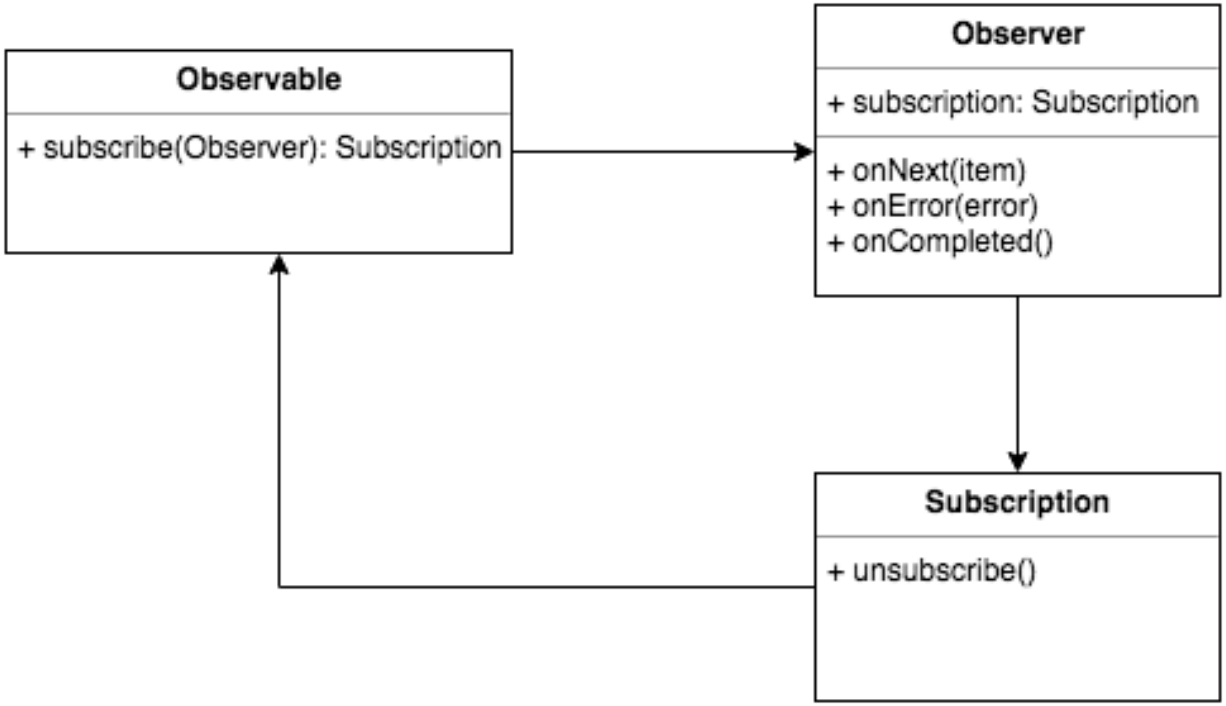
Observer tervezési minta



Összetett Avalonia UI alkalmazások

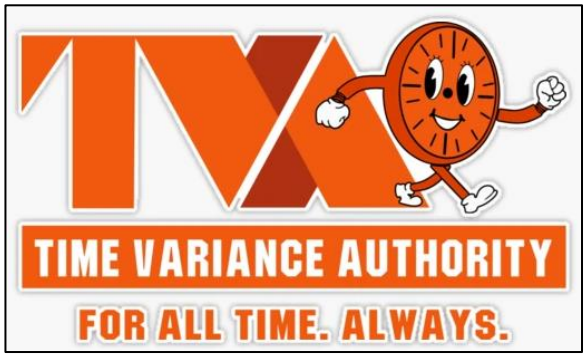
Megfigyelhető felsorolók

- Kombináljuk az *iterator* és az *observer* tervezési mintákat!
  - Mit kapunk eredményül?



Összetett Avalonia UI alkalmazások

Reaktív programozás



- Reaktív programozás legfontosabb elemei:
  - Időben változó, megfigyelhető változók (*time variant variables*), amelyeket adatfolyamokként (*data streams*) is felfoghatunk
  - Aszinkron végrehajtás és ütemezők támogatása (*observerek* és *observable*-ök ütemzése: mit melyik szálon kell végrehajtani)
  - Operátorok
    - Szűrők: *filter*, *skip*, *take*
    - Kombinációs: *concat*, *merge*, *zip*
    - Transzformációs: *map*, *groupby*
    - stb.

Összetett Avalonia UI alkalmazások

ReactiveX

- A reaktív programozáshoz használt egyik népszerű szoftverkönyvtár az eredetileg a Microsoft által kidolgozott, mára nyílt forráskódú *ReactiveX* (röviden: *Rx*) könyvtár.
  - számos programozási nyelvhez elérhető, C#/.NET-hez az *Rx.NET* (*Reactive Extensions for .NET*) könytárban
  - a **`System.Reactive`** NuGet csomaggal adhatjuk a projekteinkhez
- *Rx.NET*-ben a megfigyelhető felsorolók közös ős interfésze az **`IObservable<T>`** típus
  - ilyen megfigyelhető felsorolókat számos különböző módon létrehozhatunk az **`Observable`** osztály gyártó műveleteihez.

Összetett Avalonia UI alkalmazások

ReactiveX

- a legelemibb módja az `Observable.Create<T>()` használata
- példa egy szöveges állomány soronkénti megfigyelhető felsorolására:

```
var myObservable = Observable.Create<string>(observer =>
{
  try {
    using var reader = new StreamReader("file.txt");
    string line;
    while ((line = reader.ReadLine()) != null)  {
      observer.OnNext(line);
      // Minden sor esetén új elemet jelzünk
    }
    observer.OnCompleted();
    // Jelezzük, hogy véget ért a felsorolás
  }
  catch (Exception ex) {
    observer.OnError(ex);
    // Jelezzük, hogy hibával ért véget a felsorolás
  }
  return () => { };
  // a felsoroló felszabadítását végző tevékenység
});
```

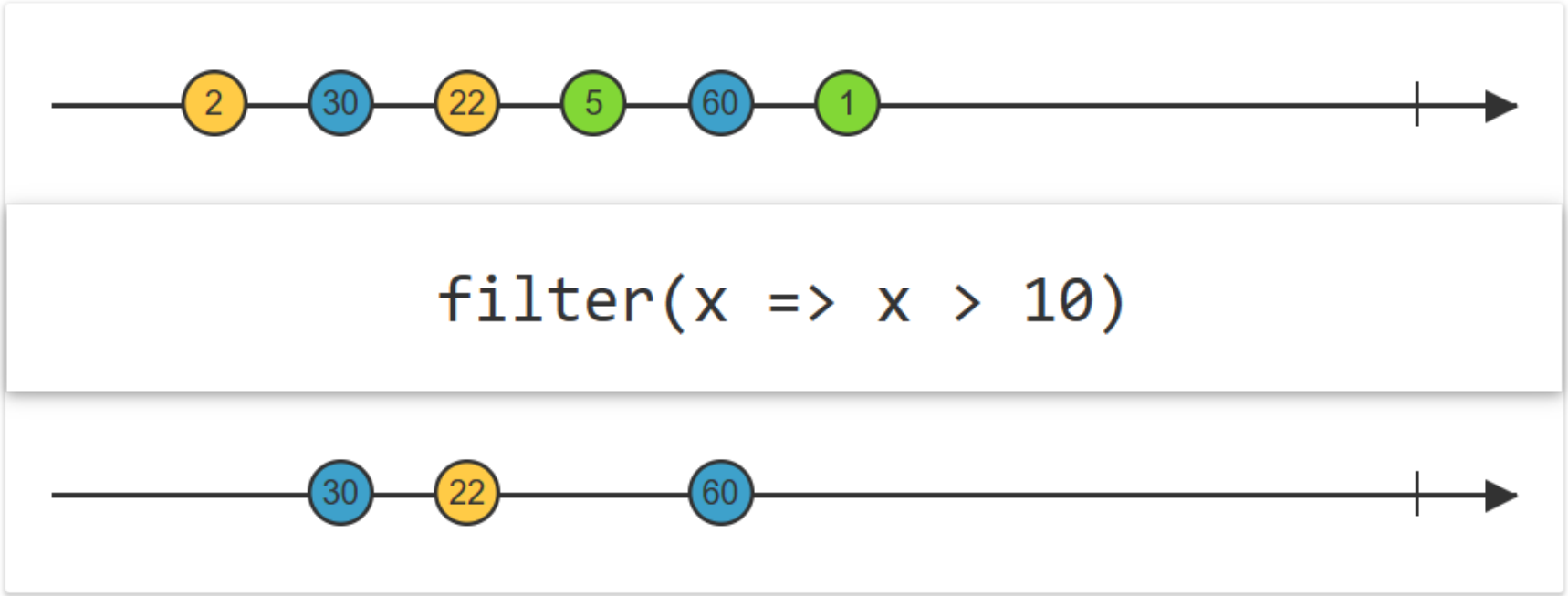
Összetett Avalonia UI alkalmazások

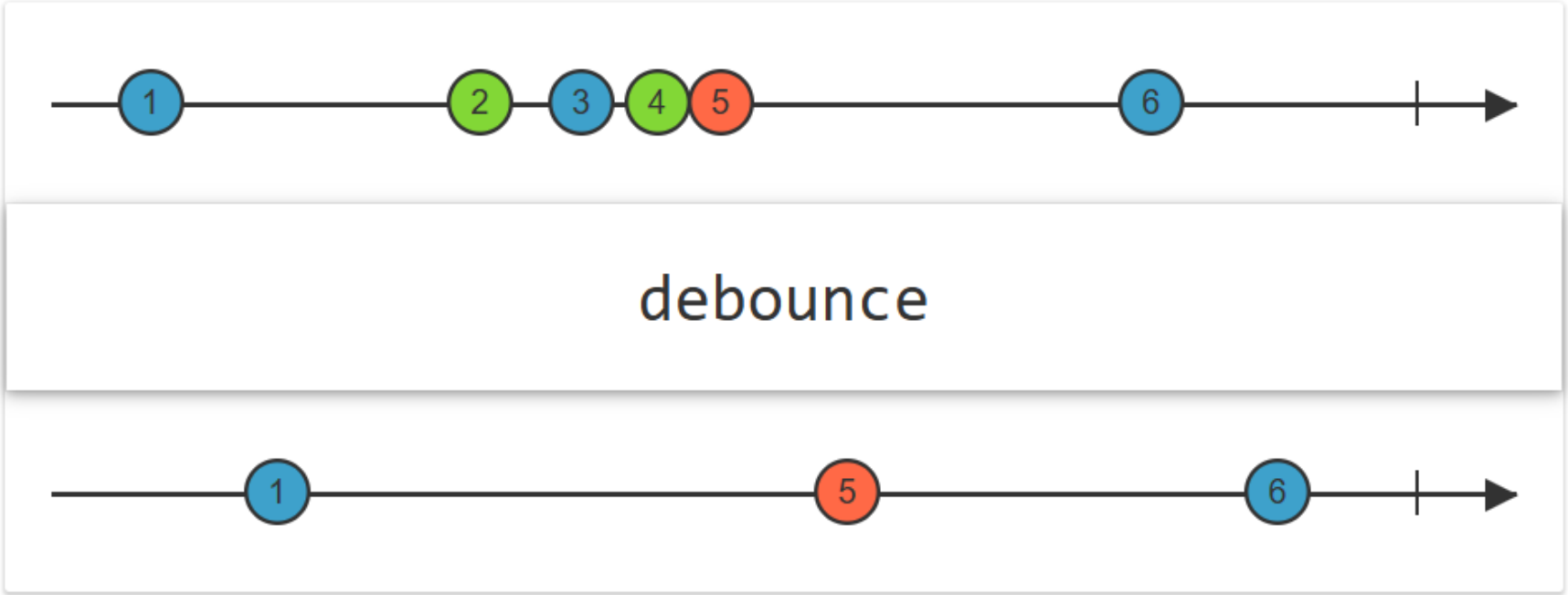
ReactiveX

- számos segéd gyártó eljárás segíti munkánkat, hogy a leggyakoribb feladatokra könnyebben is definiálhassunk megfigyelhető felsorolókat, például:
  - **`Observable.Return()`:** megadott érték egyszeri megfigyelhető felsorolása
  - **`Observable.Generate()`:** megadott kiindulási állapotból, megállási feltétellel és iterációs szabállyal elemek felsorolása
  - **`Observable.Interval()`:** elemek időzített felsorolása megadott időközönként
  - **`Observable.Start()`:** számításigényes eljárás aszinkron végrehajtása és az eredmény megfigyelhető felsorolása
- a felsorolt elemeket operátorokkal szűrhetjük, transzformálhatjuk, csoportosíthatjuk

Összetett Avalonia UI alkalmazások

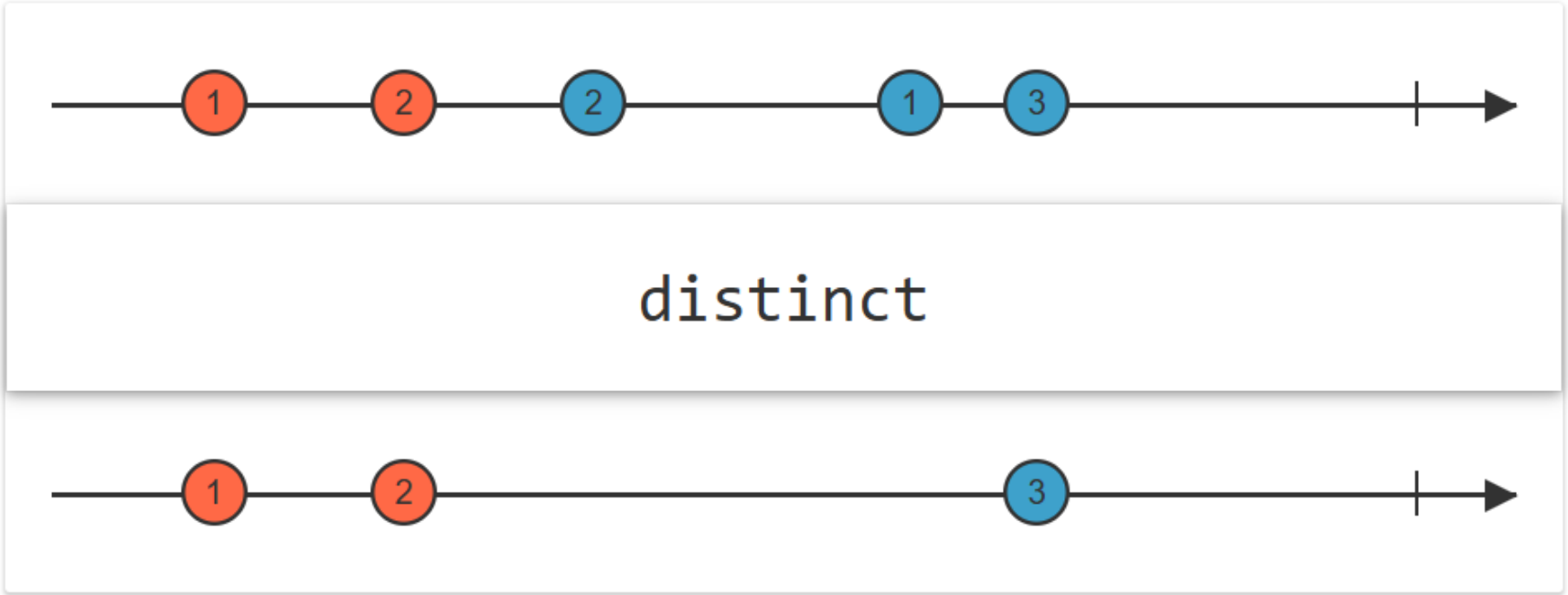
ReactiveX – Operátorok

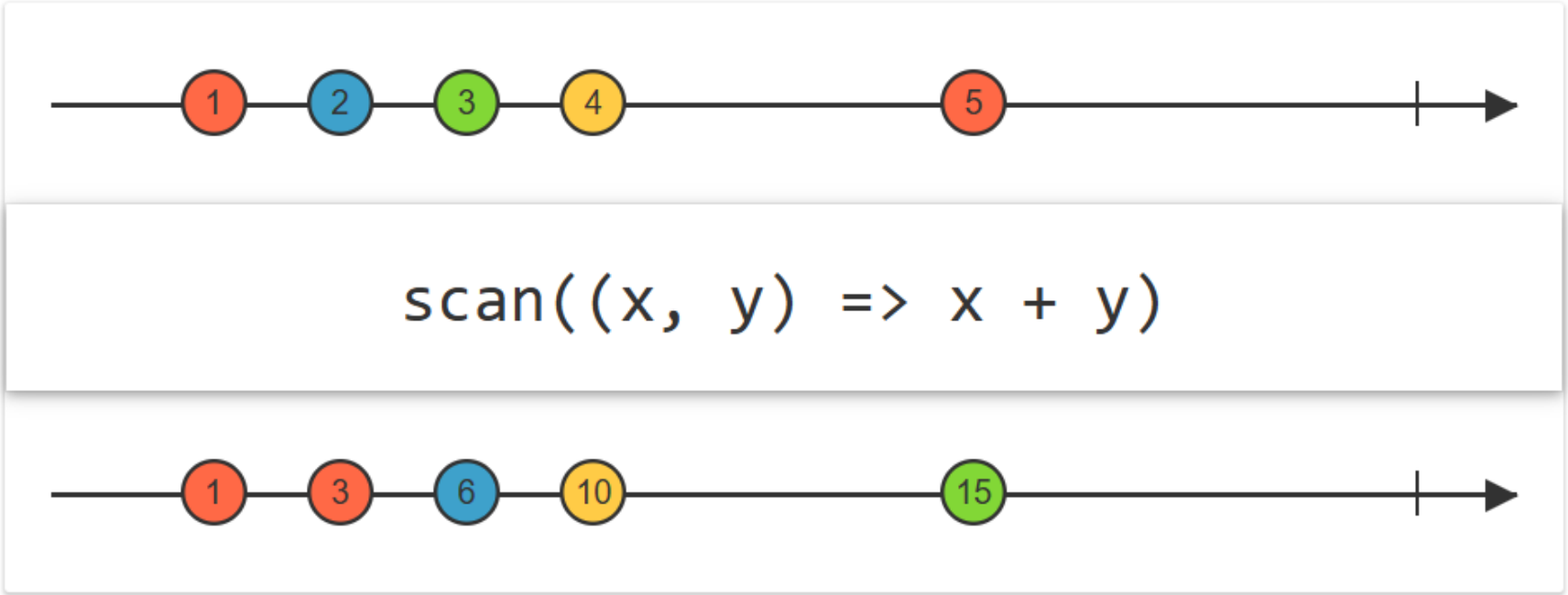



1
2
3
4
5
6
debounce
1
5
6

Összetett Avalonia UI alkalmazások

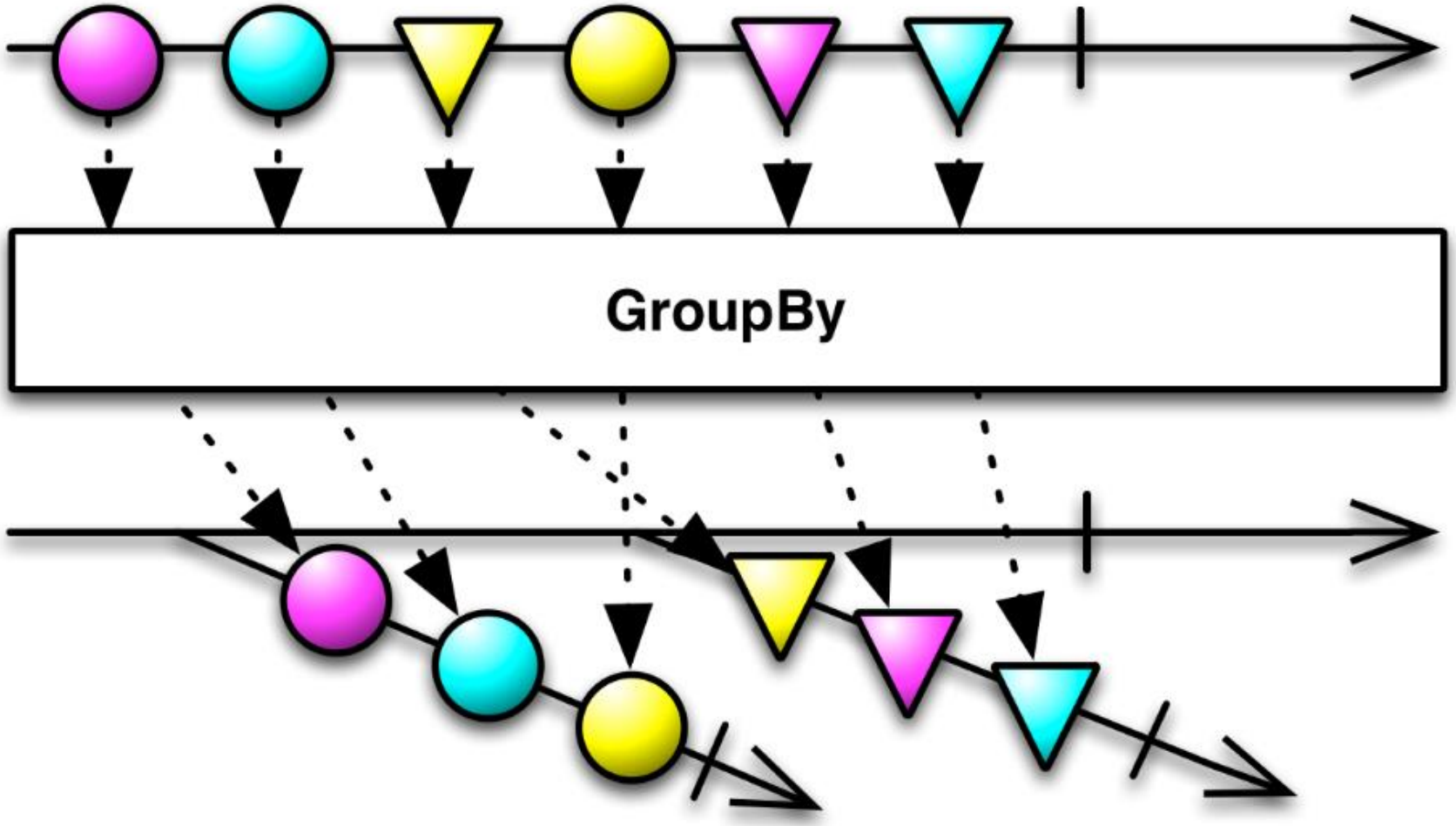
ReactiveX – Operátorok




1
2
3
4
5
scan((x, y) => x + y)
1
3
6
10
15

Összetett Avalonia UI alkalmazások

ReactiveX – Operátorok



Összetett Avalonia UI alkalmazások

ReactiveX

- megfigyelhető felsorolóra feliratkozni a Subscribe() eljárásával tudunk:

```
subscription = myObservable.Subscribe(
  param => /* … */, // onNext
  ex => /* … */,    // onError
  () => /* … */     // onSuccess
);
```

- alapértelmezetten egy új feliratkozáskor a megfigyelhető felsoroló inicializálási logikája végrehajtódik
- *multicast* megoldáshoz megfigyelhető felsorolónk elindítását a `Publish()` eljárással manuálissá alakítjuk
  - az elindítást és a felszabadítást automatizáljuk a `RefCount()` használatával
  - ez egy hivatkozás (referencia) számlálást fog a háttérben végezni, és az első feliratkozáskor indítja a felsorolást, az utolsó feliratkozás felszabadításakor pedig a felsorolót is felszabadítja

Összetett Avalonia UI alkalmazások

ReactiveUI

- Az asztali grafikus alkalmazások felületi eseményei és állapotváltozásai is kezelhetőek reaktív módon
  - .NET keretrendszerben ezt a ReactiveUI könyvtár teszi lehetővé
  - könnyen integrálható amely Windows Forms, Windows Presentation Foundation és Avalonia UI felületű alkalmazásokkal is 1-1 NuGet csomagnak a projekthez adásával
  - a *ReactiveUI* keretrendszer az *Rx.NET* osztálykönyvtárra épül
- MVVM architektúra esetén a nézetmodell megfigyelhető tulajdonságainak (**`INotifyPropertyChanged`** interfész) változása egy megfigyelhető felsorolóként is értelmezhető.
- Hasonló szemlélettel a parancsok (**`ICommand`** interfész) végrehajthatósága (**`CanExecute`**) is tekinthető egy logikai értékeket felsoroló objektumnak, amely változásairól eseményeket küld (**`CanExecuteChanged`**), azaz megfigyelhető.