

## 5. HF - Az MVVM minta és az MVVM Toolkit alkalmazása

### Bevezetés

A házi feladatban a 3. XAML laboron megvalósított személy regisztrációs alkalmazást alakítjuk át olyan módon, hogy az MVVM mintára épüljön, valamint megismerkedünk az MVVM Toolkit alkalmazásával.

Az önálló feladat a WinUI előadássorozat végén elhangzott MVVM témakörre épít. Megjegyzés: az **5. labor – MVVM** labor nagyon szerteágazó, és egy komplexebb alkalmazás kontextusában mutat példát az MVVM minta alkalmazására, sok más témakör mellett. Jelen házi feladat sokkal fókuszáltabb, kisebb lépésekben építkezik: estünkben esetben inkább a jelen házi feladat megoldása segíti az **5. labor – MVVM** kapcsolódó részeinek könnyebb megértését.

Az kapcsolódó előadásanyag feldolgozásával, jelen önálló gyakorlat feladatai a feladatleírást követő rövidebb iránymutatás segítségével (néha alapértelmezetten összecsukva) önállóan elvégezhetők.

Az önálló gyakorlat célja:

- Az MVVM minta használatának gyakorlása
- NuGet referenciák alkalmazása
- Az MVVM Toolkit alapjaival való ismerkedés
- XAML technikák gyakorlása

A szükséges fejlesztőkörnyezetről [itt](#) található leírás, megegyezik a 3. házi feladatával (XAML alapok).

### A beadás menete

- Az alapfolyamat megegyezik a korábbiakkal. GitHub Classroom segítségével hozz létre magadnak egy repository-t. A meghívó URL-t Moodle-ben találod (a tárgy nyitóoldalán a "*GitHub classroom hivatkozások a házi feladatokhoz*" hivatkozásra kattintva megjelenő oldalon látható). Fontos, hogy a megfelelő, ezen házi feladathoz tartozó meghívó URL-t használd (minden házi feladathoz más URL tartozik). Klónozd le az így elkészült repository-t. Ez

tartalmazni fogja a megoldás elvárt szerkezetét. A feladatok elkészítése után commit-old és push-old a megoldásod.

- A kiclónozott fájlok között a `HelloXaml.sln`-t megnyitva kell dolgozni.
- **!** A feladatok kérik, hogy készíts **képernyőképet** a megoldás egy-egy részéről, mert ezzel bizonyítod, hogy a megoldásod saját magad készítetted. **A képernyőképek elvárt tartalmát a feladat minden esetben pontosan megnevezi.** A képernyőképeket a megoldás részeként kell beadni, a repository-d gyökérmappájába tedd (a neptun.txt mellé). A képernyőképek így felkerülnek GitHub-ra a git repository tartalmával együtt. Mivel a repository privát, azt az oktatókon kívül más nem látja. Amennyiben olyan tartalom kerül a képernyőképre, amit nem szeretnél feltölteni, kitakarhatod a képről.
- **!** Ehhez a feladathoz érdemi előellenőrző nem tartozik: minden push után lefut ugyan, de csak a neptun.txt kitöltöttségét ellenőrzi. Az érdemi ellenőrzést a határidő lejárta után a laborvezetők teszik majd meg.

## Kikötések

### **! MVVM minta kötelező alkalmazása!**

Jelen házi feladatban az MVVM mintát gyakoroljuk, így a feladatok megoldásában kötelező az MVVM minta alkalmazása. Az ettől való eltérés a feladatok értékelésének elutasítását vonja maga után.

## Feladat 0 - Kiinduló állapot áttekintése

A kiinduló állapot alapvetően megegyezik a **3. A felhasználói felület kialakítása** végállapotával. Vagyis egy olyan alkalmazás, melyben egy listában személyek adatait lehet rögzíteni. A labor végállapotához képest egy kisebb változást tartalmaz. Laboron a felület teljes leírását a `MainWindow.xaml` (és a kapcsolódó code-behind fájl) tartalmazta. Jelen kiinduló megoldásban az a különbség, hogy ez át lett mozgatva a `Views` mappában levő `PersonListPage.xaml` (és code behind) fájlba. A `PersonListPage` nem egy `Window`, hanem egy `Page` leszármazott osztály (ellenőrizzük ezt a code behind fájlban). De semmi más változás nincs! Mint a neve is utal rá, a `Page` egy "oldalt" reprezentál az alkalmazásban: önmagában nem tud megjelenni, hanem pl. egy ablakon kell elhelyezni. Előnye, hogy az ablakon - megfelelő navigáció kialakításával - lehetőség van oldalak (különböző `Page` leszármazottak) között navigálni. Ezt mi nem fogjuk kihasználni, egyetlen oldalunk lesz csak. Az oldal bevezetésével a célunk mindössze az volt, hogy szemléltessük: az MVVM architektúrában a nézeteket nem csak `Window` (teljes ablak), hanem pl. `Page` objektumokkal is meg lehet valósítani.

Mivel mindent átmozgattunk a `MainWindow`-ból a `PersonListPage`-be, a `MainWindow.xaml`-ban

már semmi más nincs, mint egy ilyen `PersonListPage` objektum példányosítása:

```
<views:PersonListPage/>
```

Ellenőrizd a kódban, hogy valóban ez a helyzet!

## Főablak fejléce

! A főablak fejléce az "MVVM" szöveg legyen, hozzáfűzve a saját Neptun kódod: (pl. "ABCDEF" Neptun kód esetén "MVVM - ABCDEF"), fontos, hogy ez legyen a szöveg! Ehhez a főablakunk `Title` tulajdonságát állítsuk be erre a szövegre a `MainWindow.xaml` fájlban.

## Feladat 1 - MVVM Toolkit alkalmazása

A meglévő alkalmazásban a `Models` mappában levő `Person` osztály már implementálja az `INotifyPropertyChanged` (becenevén INPC) interfészt (így rendelkezik egy `PropertyChanged` eseménnyel), valamint a `Name` és az `Age` setterében jelzi is a tulajdonság változását a `PropertyChanged` esemény elsütésével (nézd meg ezt alaposan a `Person.cs` fájlban).

Bemelegítésképpen/ismétlésképpen - a kódot ( `PersonListPage.xaml` és `PersonListPage.xaml.cs` ) alaposan átnézve és az alkalmazást futtatva - fogalmazd meg magadban, miért is volt erre az alkalmazásban szükség!



### A válasz (ismétlés)



Az alkalmazásban a `PersonListPage.xaml`-ben a `TextBox`-ok `Text` tulajdonsága (ez a cél tulajdonság) hozzá vannak kötve a code behindban levő `Person` típusú `NewPerson` tag `Age` és `Name` tulajdonságaihoz (ezek a források a két adatkötésben). Nézzük meg a kódban, hogy a `NewPerson.Name` és `NewPerson.Age` forrás tulajdonságokat **változtatjuk is a kódban**: a vezérlő csak akkor tud ezekről értesülni (és így szinkronban maradni a forrással), ha ezekről a `Name` és `Age` változásokról értesítést kap. Emiatt az `Age` és `Name` tulajdonságokat tartalmazó osztálynak, vagyis a `Person`-nek meg kell valósítania az `INotifyPropertyChanged` interfészt, és a tulajdonságok változásakor el kell sütnie a `PropertyChanged` eseményt megfelelően paraméterezve.

Az alkalmazást futtatva ellenőrizd, hogy a '+' és '-' gombok hatására eszközölt `NewPerson.Age` változások valóban érvényre jutnak az életkort megjelenítő `TextBox`-ban.

A `Person` osztályban látszik, hogy az `INotifyPropertyChanged` megvalósítása és a kapcsolódó kód igencsak terjengős. Nézd meg az előadásanyagban, milyen alternatívák vannak az interfész

megvalósítására (az "INPC példa 1" című diától kezdődően kb. négy dia a négy lehetőség illusztrálására)! A legtömörebb megoldást az MVVM Toolkit alkalmazása jelenti. A következő lépésben jelen terjedősebb "manuális" INPC megvalósítást átalakítjuk MVVM toolkit alapúra.

## Feladat 1/a - MVVM Toolkit NuGet referencia felvétele

Első lépésben NuGet referenciát kell tenni az MVVM Toolkitre annak érdekében, hogy használni lehessen a projektben.

**Feladat:** Vegyél fel egy NuGet referenciát a projektben a "CommunityToolkit.Mvvm" NuGet csomagra. Ez a Visual Studio oldal írja le, hogyan lehet egy NuGet referenciát a projektbe felvenni **NuGet Package Manager**. Az előző link az oldalon belül a "NuGet Package Manager" fejezetre ugrik, az itt megadott négy lépést kell követni (természetesen azzal a különbséggel, hogy nem a "Newtonsoft.Json" hanem a "CommunityToolkit.Mvvm" csomagra kell a referenciát felvenni).

Most, hogy a projektünkbe felvettük ezt a NuGet referenciát, a következő build során (mivel annak részeként lefut egy NuGet restore lépés!) letöltődik a NuGet csomag, kicsomagolódnak a benne levő DLL-ek a kimeneti mappába, így azok már szerves részét képezik az alkalmazásnak (egy NuGet csomag tulajdonképpen egy zip állomány). Fontos megemlíteni, hogy Git-be sem a NuGet zip, sem a benne levő dll-ek nem kerülnek fel, a solution gyökerében levő `.gitignore` fájlt ezeket kiszűri. Pont ez a NuGet koncepció lényege: a repository kicsi maradhat, mert a projektfájl csak hivatkozásokat tartalmazza a NuGet csomagokra, és amikor valaki egy frissen clone-ozott solution-t buildel, csak ekkor töltődnek le az online NuGet forrásokból a hivatkozott NuGet csomagok.

⚠ A fenti NuGet-re vonatkozó koncepciók ismerete fontos, a tananyag fontos részét képezik!

Egy NuGet referencia tulajdonképpen csak egy sor a `.csproj` projektleíró fájlban. A Solution Explorerben a "HelloXaml" projekt csomópontra kattintva nyisd meg a `.csproj` projektfájlt, és ellenőrizd, benne van ez a sor (a verzió lehet más lesz):

```
<PackageReference Include="CommunityToolkit.Mvvm" Version="8.2.2" />
```



A `csproj` fájl megnyitása nélkül is ellenőrizd a NuGet referenciánkat: Solution Explorerben nyisd le a "HelloXaml"/"Dependencies"/"Packages" csomópontot: ha minden rendben van, alatta látható egy "CommunityToolkit.Mvvm (verzió)" csomópont.

## Feladat 1/b - INPC megvalósítás MVVM Toolkit alapokon

Most már tudjuk használni az MVVM Toolkit NuGet package-ben levő osztályokat, interfészeket, attribútumokat stb., így át tudunk térni az MVVM Toolkit alapú INPC megvalósításra.

- Kommentezd ki a `Person` osztályt teljes egészében.
- A kikommentezett rész felett vedd fel az osztályt újonnan, de MVVM Toolkit alapú INPC megvalósítással.
  - A megvalósításban a "INPC példa 4 - MVVM Toolkittel" előadásdia segít.
  - Partial class kell legyen (vagyis az osztály részei több fájlban is definiálhatók).
  - A Toolkit-beli `ObservableObject`-ből származzon: ez az ős valósítja meg az `INotifyPropertyChanged` interfészt, így nekünk már nem kell.
  - `Name` és `Age` tulajdonságok helyett `name` és `age` tagváltozókat vezessünk be, `ObservableProperty` attribútummal ellátva.

Meg is vagyunk.

 **A megoldás ellenőrzése** 

```
public partial class Person : ObservableObject
{
    [ObservableProperty]
    private string name;

    [ObservableProperty]
    private int age;
}
```

Ez a kód, egy fordítást követően, alapjaiban ugyanazt a megoldást eredményezi, mint a korábbi, sokkal terjengősebb, immár kikommentezett forma. Vagyis (még ha nem is látjuk egyelőre) születik `Name` és `Age` tulajdonság, megfelelő `PropertyChanged` esemény elsütésekkel. Hogyan lehetséges ez?

- Egyrészt az `ObservableObject` ős már megvalósítja az `INotifyPropertyChanged` interfészt, így a `PropertyChanged` esemény tagot is tartalmazza, ezt a származtatás révén "megörökli" az osztályunk.
- A fordítás során lefut az MVVM Toolkit kódgenerátora, mely minden `ObservableProperty` attribútummal ellátott tagváltozóhoz generál egy ugyanolyan nevű, de nagybetűvel kezdődő tulajdonságot az osztályba, mely tulajdonság settere elsüti megfelelő feltételek mellett és megfelelő paraméterekkel a `PropertyChanged` eseményt. Hurrá, ezt a kódot akkor nem nekünk kell megírni.
- Kérdés, hol keletkezik ez a kód. Az osztályunk egy másik "partial" részében. Egy fordítást követően Visual Studio-ban jobb gombbal kattintsunk a `Person` osztály nevén, majd a felugró

menüben "Go to Definition". Ekkor egy alsó ablakban két találatot is kapunk: az egyik az általunk írt fenti kód, a másik ("public class Person") a generált részre ugrik egy duplakatt hatására: látszik, hogy viszonylag terjedős kódot generált a kódgenerátor, de ami nekünk fontos, hogy itt található a `Name` és `Age` tulajdonság, benne - többek között - a `OnPropertyChanged` elsütésével.

! A kódgenerátor szokásosan az osztályunk másik "partial" felébe dolgozik, annak érdekében, hogy ne keveredjen az általunk írt és a generált kód! A partial classokat leggyakrabban a kézzel írt és a generált kód "különválasztására" használjuk.

Mivel sokkal kevesebb kódot kell írni, a gyakorlatban az MVVM Toolkit alapú megoldást szoktuk használni (de a manuális megoldást is tudni kell, ez alapján érthető, mi is történik a színfalak mögött).



### BEADANDÓ

Készíts egy képernyőmentést `f1b.png` néven az alábbiak szerint:

- Indítsd el az alkalmazást. Ha szükséges, méretezd át kisebbre, hogy ne foglaljon sok helyet a képernyőn,
- a „háttérben” a Visual Studio legyen, a `Person.cs` megnyitva.

## Feladat 2 - Áttérés MVVM alapú megoldásra

Az előző lépésben, bár az MVVM Toolkitet használtuk, még nem tértünk át MVVM alapú megoldásra (a toolkitet csak az INPC egyszerűbb megvalósítására használtuk).

A következőkben átalakítjuk az alkalmazásunk architektúráját, hogy az MVVM koncepcióját kövesse. Az egyszerűbb megvalósítás érdekében építünk az MVVM Toolkitre.

**Feladat:** Dolgozd fel a kapcsolódó előadásanyagot (WinUI anyagrészt végén található):

- Értsd meg az MVVM minta alapkonceptióit.
- Az előadásdiákon található példák teljes kódja elérhető az [Előadás GitHub repository](#) "04-05 WinUI\DancerProfiles" mappában ("RelaxedMVVM" és "StrictMVVM"), ezek segíthetnek a megértésben és a későbbi feladatok megoldásában.

Mit is jelent az MVVM minta a példákra vetítve:

- A model osztály a `Models` mappában levő `Person` osztály, egy személy adatait reprezentálja (UI logikát NEM tartalmaz, független mindenféle megjelenítéstől).

- Jelen pillanatban minden, megjelenítéshez kapcsolódó leírás/logika a `PersonListPage`-ben van. **!** A mostani `PersonListPage`-et **kettévágjuk**:
  - A `PersonListPage.xaml` és a code behindja lesz a View.
  - Bevezetünk egy a `PersonListPage`-hez tartozó ViewModel-t `PersonListPageViewModel` néven.
    - **!** Kulcsfontosságú: a `PersonListPage` code behindből minden megjelenítési logikát átmozgatunk a `PersonListPageViewModel`-be. **A minta lényege az, hogy a View csak tisztán a felület leírását tartalmazza, a megjelenítési logikának a ViewModelben van a helye.**
- A minta másik alappillére: a View-nk tartalmaz egy hivatkozást a ViewModeljére (mégpedig egy tulajdonság formájában).
  - A példánkban azt jelenti, hogy a `PersonListPage`-nek kell legyen egy `PersonListPageViewModel` tulajdonsága.
  - **!** Ez azért kulcsfontosságú, mert `PersonListPage` xaml fájlunkban ezen tulajdonságon keresztül tudunk adatkötést megvalósítani a ViewModel-be átmozgatott tulajdonságokra és eseménykezelőkre!
- A `PersonListPageViewModel` "dolgozik" a modellel és kezeli a felhasználói interakciókat (eseménykezelők).
- Mivel a Relaxed, és nem a Strict MVVM mintát használjuk, a `Person` modellosztályunk köré már nem vezetünk be egy `PersonViewModel` csomagolót.

Feladat: alakítsd át a meglévő logikát így, hogy a fenti elveket követő MVVM mintát kövesse. A `PersonListPageViewModel` osztályt egy újonnan létrehozott `ViewModels` mappába tedd. Próbáld magad kidolgozni a megoldást a fenti segítség alapján! Ehhez egy előzetes tippel adunk, mert erre nehezebb rájönni: Az eseményekhez az eseménykezelő műveleteket adatkötéssel is meg lehet adni: lásd előadás dia "Események és funkciók kötése" címmel (az átalakítás után az eseménykezelőket csak így tudjuk megadni). Az is fontos, hogy adatkötni csak publikus tulajdonsághoz/művelethez lehet, ennek kapcsán is lesz átalakítandó!

**Tippek / megoldás visszaellenőrzése**

1. `PersonListPage.xaml.cs` code-behind fájlból szinte mindent (kivéve `this.InitializeComponent()` hívás a konstruktorban) át kell mozgatni az újonnan bevezetett `PersonListPageViewModel`-be, mert ez mind UI logika.
2. A `PersonListPageViewModel` publikus osztály legyen.
3. A `PersonListPage` code behindba fel kell venni egy `ViewModel` nevű, `PersonListPageViewModel` típusú, csak getterrel rendelkező auto implementált tulajdonságot, és ezt egy új objektumra inicializálni is kell. Vagyis a view hozza létre és tartalmazza a `ViewModel`-t!
4. A `PersonListPage.xaml`-ben a két `TextBox` adatkötését megfelelően igazítani kell (a `NewPerson.Name` és `NewPerson.Age` már egy szinttel mélyebben, a code behind `ViewModel` tulajdonságán keresztül érhető el).
5. A `PersonListPage.xaml`-ben az eseménykezelők (`Click`) igazítása három helyen. Ezt trükkösebb. Eseménykezelő függvény az eddig alkalmazott szintaktikával nem adható már meg, mert az eseménykezelők nem a code behindban találhatók (átkerültek a `ViewModel`-be).
  - Az eseményekhez az eseménykezelő műveleteket adatkötéssel is meg lehet adni! Lásd előadás dia "Események és funkciók kötése" címmel. Ez nekünk azért jó, mert a code behind `ViewModel` tulajdonságában ott a `PersonListPageViewModel` objektum, melyben ott vannak az eseménykezelők (`AddButton_Click`, `IncreaseButton_Click`, `DecreaseButton_Click`), ezeket kell kötött tulajdonságként megadni az adatkötésben (pl. `ViewModel.AddButton_Click` stb.).
  - Fontos, hogy az eseménykezelő függvények legyenek publikusak, máskülönben nem működik az adatkötés (át kell alakítani privátról).

További lényeges átalakítandók:

- A `ViewModel`-ben jelenleg a `Click` eseménykezelők nevei: `AddButton_Click`, `IncreaseButton_Click` és `DecreaseButton_Click`. Ez nem szerencsés. A `ViewModel`-ben "szemantikailag" nem eseménykezelőkben gondolkodunk. Helyette módosító műveletekben, melyek módosítják a `ViewModel` állapotát. A fentiek helyett ennek megfelelően sokkal jobban passzoló és kifejező nevek az `AddPersonToList`, `IncreaseAge` és `DecreaseAge`. Nevezd át a függvényeket ennek megfelelően! Persze a továbbiakban is adatkötéssel ezeket kell kötni a XAML fájlban a `Click` eseményekhez.
- A fenti függvények paraméterlistája egyelőre az "object sender, RoutedEventArgs e". Ugyanakkor ezeket a paramétereket nem használjuk semmire. Szerencsére a x:Bind esemény adatkötés rugalmas annyira, hogy paraméter nélküli művelet is megadható, azzal is jól működik. Ennek tudatában távolítsd el a fenti felesleges paramétereket a `ViewModel`ünk



három függvényéből. Így egy letisztultabb megoldást kapunk.

Ellenőrizd, hogy az átalakítások után is pontosan ugyanúgy működik az alkalmazás, mint előtte!

Mit nyertünk azzal, hogy korábbi megoldásunkat MVVM alapúra alakítottuk át? A választ az előadásanyag adja meg! Pár dolog kiemelve:

- Szépen különválnak (nem keverednek) a különböző felelősségű részek, így jobban megérthető:
  - UI független logika (model és kapcsolódó osztályok).
  - UI logika (ViewModel)
  - UI pusztta megjelenés (View)
- Mivel a UI logika külön van, lehet(ne) hozzá unit teszteket írni

Minél komplexebb egy alkalmazás, annál inkább igazak ezek.



#### BEADANDÓ

Készíts egy képernyőmentést `f2.png` néven az alábbiak szerint:

- Indítsd el az alkalmazást. Ha szükséges, méretezd át kisebbre, hogy ne foglaljon sok helyet a képernyőn,
- a „háttérben” a Visual Studio legyen, a `PersonListPageViewModel.cs` megnyitva.

## Feladat 3 - Vezérlők tiltása/engedélyezése

Jelen állapotban kissé furcsán viselkedik az alkalmazás: a "-" gombbal negatív tartományba is vihető egy életkor, vagy a "+"-szal 150 fölé, illetve a "+Add" gombbal olyan személy is felvehető, mely értelmetlen tulajdonságokkal rendelkezik. Ezeket a gombokat le kellene tiltani, amikor az általuk kiváltott műveletnek nincs értelme, illetve engedélyezni, amikor van.

A következő lépésben valósítsuk meg a "-" gomb tiltását/engedélyezését ennek megfelelően. A gomb akkor legyen csak engedélyezett, ha a személy életkora 0-nál nagyobb.

Próbáld ezt első körben magadtól megvalósítani, legalábbis az alapjait lefektetni! Mindenképpen adatkötés alapú megoldásban gondolkozz, csak ez fogadható el! Ha elakadsz, a megoldásod nem "akar" működni, akkor gondold át, mi lehet az oka, a megoldást pedig az alábbiaknak megfelelően alakítsd ki.

A problémára többféle megoldás is kidolgozható. Mindben közös, hogy a "-" gomb `IsEnabled` tulajdonságát kövjük valamilyen módon. Az általunk választott megoldásban egy a

`PersonListPageViewModel`-ben újonnan bevezetett `bool` tulajdonsághoz kössük.

#### `PersonListPageViewModel.cs`

```
public bool IsDecrementEnabled
{
    get { return NewPerson.Age > 0; }
}
```

#### `PersonListPage.xaml`-be a '-' gombhoz

```
IsEnabled="{x:Bind ViewModel.IsDecrementEnabled, Mode=OneWay}"
```

Próbáljuk ki! Sajnos nem működik, a "-" gomb nem tiltódik le, amikor 0 vagy kisebb értékű lesz az életkor (pl. a gomb sokszori kattintásával). Ha töréspontot teszünk az `IsDecrementEnabled` belsejébe, és így indítjuk az alkalmazást, azt tapasztaljuk, hogy a tulajdonság értékét csak egyszer kérdezi le a kötött vezérlő, az alkalmazás indulásakor: utána hiába kattintunk pl. a "-" gombon, többször nem. Próbáld is ki!

Gondold át, mi okozza ezt, és csak utána haladj tovább az útmutatóval!



#### Indoklás



A korábban tanultaknak megfelelően az adatkötés csak akkor kérdezi le a forrástulajdonság (esetünkben `IsDecrementEnabled`) értékét, ha annak változásáról az `INotifyPropertyChanged` segítségével értesítést kap! Márpedig, jelen megoldásunkban hiába változik a `NewPerson` objektum `Age` tulajdonsága, ennek megtörténtekor a semmiféle értesítés nincs az erre épülő `IsDecrementEnabled` tulajdonság megváltozásáról!

A következő lépésben valósítsd meg a kapcsolódó változásértesítést a

`PersonListPageViewModel` osztályban:

- MVVM Toolkit "alapokon" valósítsd meg az `INotifyPropertyChanged` interfészt
  - `ObservableObject` származtatást használj.
  - Az `IsDecrementEnabled` tulajdonság maradhat a mostani formájában (egy getter only property), nem szükséges `[ObservableProperty]` alapúra átírni (de az is jó megoldás, és a házi feladat tekintetében is teljesen elfogadható, csak kicsit másként kell dolgozni a következő lépésekben).
- Próbáld magadtól megvalósítani a következőt a ViewModel osztályban (a `Person` osztály marad változatlan): amikor a `NewPerson.Age` változik, akkor az `ObservableObject` ősből

örökölt `OnPropertyChanged` hívásával jelezzük a `IsDecrementEnabled` tulajdonság változását. Tipp: a `Person` osztály már rendelkezik `PropertyChanged` eseménnyel, hiszen maga is megvalósítja az `INotifyPropertyChanged` interfészt, erre az eseményre fel lehet iratkozni! Az egyszerűség érdekében az nem zavar minket, ha az `IsDecrementEnabled` változását esetleg akkor is jelezzük, ha tulajdonképpen "logikailag" esetleg nem is változik.

- A fentieket külön eseménykezelő függvény bevezetése nélkül is meg lehet oldani (tipp: eseménykezelő megadása lambda kifejezéssel).

Teszteld is a megoldásod! Ha jól dolgoztál, a gombnak akkor is le kell tiltódnia, ha a `TextBox`-ba kézzel írsz be negatív életkor értéket (és utána kikattintasz a `TextBox`-ból). Gondold át, miért van ez így!

A "+" gombra és a "+Add" gombra is dolgozz ki hasonló megoldást!

- Az életkor maximális "elfogadható" értéke 150 legyen.
- A név csak akkor elfogadható, ha van benne legalább egy nem whitespace karakter (ez utóbbi ellenőrzésére a `string` osztály `IsNullOrWhiteSpace` statikus műveletét használd).
- Azzal az esettel nem kell foglalkozni, hogy ha a felhasználó az életkor `TextBox`-ba nem érvényes számot ír be (ezt jelen megoldással nem is lehet kezelni).

A tesztelés során azt tapasztaljuk, hogy ha pl. kitöröljük a nevet a név `TextBox`-ban, a "+Add" gomb állapota nem azonnal változik, hanem csak ha elhagyjuk a `TextBox`-ot? Miért van ez? Módosítsd a megoldásod, hogy ez minden szöveg változáskor, a `TextBox` elhagyása nélkül is megtörténjen.

Tipp: lásd előadásanyag "x.Bind mikor frissül az adat?" című dia.



#### BEADANDÓ

Készíts egy képernyőmentést `f3.png` néven az alábbiak szerint:

- Indítsd el az alkalmazást. Ha szükséges, méretezd át kisebbre, hogy ne foglaljon sok helyet a képernyőn,
- az életkor legyen 0-ra lecsökkentve az alkalmazásban,
- a „háttérben” a Visual Studio legyen, a `PersonListPageViewModel.cs` megnyitva.

## Feladat 4 - Command használata

Jelen pillanatban a "-" gomb vonatkozásában esetében két feladatunk van:

- A `Click` esetén az eseménykezelő művelet futtatása

- A gomb tiltása/engedélyezése az `IsEnabled` tulajdonság segítségével

Bizonyos vezérlők - ilyen a gomb is - támogatják, hogy ezt a kettőt, a Command mintára építve, egy parancs objektum segítségével adhassuk meg. A Command tervezési minta koncepciójával a "Tervezési minták 3" előadás alapján lehet részletesebben megismerni (bár ott csak az alap Command mintával ismerkedtünk meg, mely a parancs futtatását támogatja, tiltását/engedélyezését nem). A Command minta MVVM specifikus megvalósításával a WinUI előadássorozat vége felé, a "Command minta" című diától kezdve lehet megismerni.

Az alapelv a következő: a gombnál a `Click` és `IsEnabled` "megadása" helyett a gomb `Command` tulajdonságát állítjuk egy `ICommand` interfészt megvalósító command objektumra. A futtatás, illetve tiltás/engedélyezés már ezen command objektum feladata.

Alapesetben egy alkalmazásban minden parancshoz egy külön `ICommand` implementációt kellene készíteni. Ez azonban sok parancs esetén sok osztály bevezetését igényli. Az MVVM Toolkit ebben is a segítségünkre siet. Biztosít egy `RelayCommand` osztályt, mely megvalósítja az `ICommand` interfészt. Ez az osztály bármilyen parancs/kód futtatására használható, így nem kell további command osztályokat bevezetni. Hogyan lehetséges ez? Úgy, hogy a `RelayCommand`-nak konstruktor paraméterekben, két delegate formájában tudjuk a végrehajtáshoz és a tiltáshoz/engedélyezéshez tartozók kódot:

- Első paraméterben a parancs futtatásakor végrehajtandó kódot adjuk meg.
- Második paraméterben (ez opcionális) azt a kódot, melyet a command hív annak ellenőrzésére, hogy engedélyezni/tiltani kell magát (az itt megadott függvénynek bool-lal kell visszatérnie, true esetben engedélyezett lesz a parancs).

A következő lépésben a "-" gomb kezelését alakítjuk át command alapúra. Először próbáld a nagyját önállóan megvalósítani a kapcsolódó WinUI előadásanyag alapján. A parancs futtatása egyszerűbb, de a parancs tiltás-engedélyezéshez lesz még teendők. Főbb lépések:

- Egy csak getterrel rendelkező publikus `RelayCommand` tulajdonság felvétele a ViewModel-be, pl. `DecreaseAgeCommand` néven. Az előadásanyaggal ellentétben esetünkben nem kell a `RelayCommand`-nak generikus paramétert megadni, mert a parancskezelő függvényünknek (`DecreaseAge`) nincs paramétere.
- Az újonnan bevezetett tulajdonságnak a ViewModel konstruktorban értéket adni. A `RelayCommand` konstruktor paramétereit add meg megfelelően.
- A `PersonListPage.xaml`-ben a "-" gombnál a `Click` és `IsEnabled` adatkötésére nincs már szükség, ezek törölendők. Helyette a gomb `Command` tulajdonságát kösd a ViewModel-ben az előző lépésben bevezetett `DecreaseAgeCommand` tulajdonsághoz.

Ha kipróbáljuk, a parancs futtatás működik, a tiltás/engedélyezés viszont még nem: ha jól

megfigyeljük, a gomb mindig engedélyezett marad megjelenésében. Ennek, kicsit jobban belegondolva, logikus oka van: a `RelayCommand` meg tudja ugyan hívni a második konstruktor paraméterében megadott műveletet az állapot ellenőrzéséhez, de nem tudja, hogy minden `NewPerson.Age` változáskor meg kellene ezt tennie! Ezen tudunk segíteni. A `ViewModel`-ünk konstruktorában már feliratkoztunk korábban a `NewPerson.PropertyChanged` eseményre: erre építve, amikor változik az életkor (vagy amikor változhat, az nem probléma, ha néha feleslegesen megtesszük) hívd meg a `DecreaseAgeCommand` `NotifyCanExecuteChanged` műveletét. Ennek a műveletnek nagyon beszédes neve van: értesíti a parancsot, hogy megváltoz(hat)ott azon állapot, mely alapján a parancs tiltott/engedélyezett állapota épít. Így a parancs frissíteni fogja magát, pontosabban a parancshoz tartozó gomb állapotát.

Írd át "+" gomb kezelését is hasonlóan, parancs alapúra! A "+Add" gomb kezelését ne változtasd meg!



#### BEADANDÓ

Készíts egy képernyőmentést `f4.png` néven az alábbiak szerint:

- Indítsd el az alkalmazást. Ha szükséges, méretezd át kisebbre, hogy ne foglaljon sok helyet a képernyőn,
- a név `TextBox` legyen üres az alkalmazásban,
- a „háttérben” a Visual Studio legyen, a `PersonListPageViewModel.cs` megnyitva.

## Feladat 5 - Command használata MVVM Toolkit alapú kódgenerálással

Az előző feladatban a command tulajdonságok bevezetését és azok példányosítását "manuálisan" oldottuk meg. Az MVVM Toolkit ezt le tudja egyszerűsíteni: megfelelő attribútum alkalmazása esetén a tulajdonságot és a példányosítást automatikusan le tudja generálni.

Alakítsuk át a `DecreaseAgeCommand` kezelést (csak ezt, az `IncreaseAgeCommand` maradjon!) generált kód alapúra:

1. Lásd el a `PersonListPageViewModel` osztályt a `partial` kulcsszóval.
2. Töröld ki a `DecreaseAgeCommand` tulajdonságot és ennek példányosítását a konstruktorból.
3. A `DecreaseAge` műveletet lásd el ezzel az attribútummal: `[RelayCommand(CanExecute = nameof(IsDecrementEnabled))]`.
  - Ennek hatására a kódgenerátor bevezet egy `RelayCommand` tulajdonságot az osztályban,

melynek neve a műveletünk neve ( `DecreaseAge` ), hozzáfűzve a "Command" stringet. Ezzel meg is kapjuk a korábban kézzel bevezetett `DecreaseAgeCommand` nevű tulajdonságot.

- A `CanExecute` attribútum tulajdonságban egy string formában annak a boollal visszatérő műveletnek vagy tulajdonságnak a nevét lehet megadni, melyet a generált kód a parancs tiltásának/engedélyezésének során használ (a `RelayCommand` konstruktor második paramétere lesz). Nekünk már van ilyen tulajdonságunk, "IsDecrementEnabled" néven. Azért nem egyszerű string formájában adjuk meg, mert ha utólag valaki átnevezi az `IsDecrementEnabled` műveletet, akkor a mostani "IsDecrementEnabled" már nem jó műveletre mutatna. A `nameof` kifejezés használatával ez a probléma elkerülhető. A `CanExecute` megadása általánosságában nem kötelező (nem adjuk meg, ha nem akarjuk a parancsot soha tiltani).
4. Teszteld a megoldást (életkor csökkentése), ugyanúgy kell működnie, mint korábban. Egyrészt csökkentenie kell az életkort, másrészt 0 elérésekor le kell tiltania a gombot. Ha ez utóbbi nem működik, akkor egy lehetséges ok, hogy a `DecreaseAgeCommand -ra a NotifyCanExecuteChanged hívását törölted az átalakítás során. Erre most is szükség van, hiszen átalakításunk csak arról szólt, hogy a DecreaseAgeCommand-ot MVVM toolkit alapokon kódgenerátorral, egyszerűbben állítjuk elő.`



#### BEADANDÓ

Készíts egy képernyőmentést `f5.png` néven az alábbiak szerint:

- Indítsd el az alkalmazást. Ha szükséges, méretezd át kisebbre, hogy ne foglaljon sok helyet a képernyőn,
- a „háttérben” a Visual Studio legyen, a `PersonListPageViewModel.cs` megnyitva.

## Feladat 6 - Strict MVVM

Jelen megoldásunk a Relaxed MVVM megközelítést követi. A következő lépésekben átgondoljuk, mit is jelent ez pontosan, és mit jelentene a Strict MVVM megközelítésre való áttérés (megvalósítani nem fogjuk).

Jelen megoldásunk a Relaxed MVVM megközelítést követi, vagyis a View-ban közvetlenül a `Person` modell osztályhoz adatkötünk (és a `PersonPageViewModel`-ben is a `Person` modell osztályt használjuk). Ennek előnye az egyszerűség. De van egy hátránya is: a `Person` modell osztályunkban kénytelenek voltunk megvalósítani az `INotifyPropertyChanged` interfészt (mégha az MVVM toolkit segítségével is), különben nem működne jól az adatkötés. Vannak olyan helyzetek, amikor a modell osztályunkat nem szeretnénk ilyen, némiképpen már a felületet

kiszolgáló logikával "szennyezni", hanem a lehető legtisztább formában szeretnénk a modell osztályunkat tartani. Ekkor a Strict MVVM megközelítés jelenti a megoldást (lásd "Strict MVVM - beágyazás" előadásdia). Mit jelentene ez a számunkra, mit kellene a kódon változtatni? Gondold át az előadásdia alapján a szükséges változtatásokat! Megvalósítani/dokumentálni nem kell, ez csak egy átgondolós feladat 😊!



### Strict MVVM alapú megoldás



- A `Person` modell osztályban már nem valósítanánk meg az `INotifyPropertyChanged` interfészt, az osztály leegyszerűsödik, csak egyszerű tulajdonságokat tartalmazna (ez volt a cél).
- Be kellene vezetni egy `PersonViewModel` osztályt (mely egy `Person` modell objektumot csomagolna be). Ebben:
  - Be kellene vezetni a `Name` és `Age` tulajdonságokat.
  - Meg kellene valósítani az `INotifyPropertyChanged` interfészt:
    - `ObservableObject` származtatás
    - A tulajdonságok setterében a `SetProperty` ősből örökölt segédfüggvény használata (hogy kiváltódjon a `PropertyChanged` esemény)
- A Viewhoz tartozó `PersonPageViewModel`-ünket át kellene alakítani, hogy ne a `Person` modell, hanem az új `PersonViewModel`-t használja.



2024-05-12



Szerzők

