

Szoftvertechnikák önálló gyakorlat

5. Önálló feladat

Document-View architektúra

Tartalom

TARTALOM	2
BEVEZETÉS	3
FELADATOK ÁTTEKINTÉSE	4
FELADAT 1 - A KIINDULÁSI KÖRNYEZET MEGISMERÉSE	5
FELADAT 2 – MÉRÉSI ÉRTÉKEK REPREZENTÁLÁSA, SAJÁT DOKUMENTUM OSZTÁLY, ADATOK MENTÉSE ÉS BETÖLTÉSE	7
1. VEZESSEN BE EGY OSZTÁLYT A JELÉRTÉKEK REPREZENTÁLÁSÁRA	7
2. VEZESSEN BE EGY SAJÁT DOKUMENTUM OSZTÁLYT A DOKUMENTUMHOZ TARTOZÓ JELÉRTÉKEK TÁROLÁSÁRA.	7
3. GONDOSKODJON A DOKUMENTUM ÁLTAL TÁROLT ADATOK ELMENTÉSÉRŐL.....	7
4. BIZTOSÍTSON LEHETŐSÉGET DOKUMENTUM FÁJLBÓL BETÖLTÉSÉRE	9
5. A BETÖLTÉST KÖVETŐEN ELLENŐRIZZE A BETÖLTÉS SIKERESSÉGÉT	10
FELADAT 3 – JELEK GRAFIKUS MEGJELENÍTÉSE, SAJÁT NÉZET OSZTÁLY	10
1. VEZESSEN BE EGY ÚJ NÉZET OSZTÁLYT UserControl FORMÁJÁBAN	10
2. RAJZOLJA KI A KOORDINÁTA TENGELYEKET	12
3. VALÓSÍTSA MEG A JELEK MEGJELENÍTÉSÉT.....	13
4. BIZTOSÍTSON LEHETŐSÉGET A NÉZET NAGYÍTÁSRA ÉS KICSINYÍTÉSÉRE. EHHEZ HELYEZZEN EL EGY KISMÉRETŰ, + ÉS – SZÖVEGET TARTALMAZÓ NYOMÓGOMBOT A NÉZETEN.....	14
OPCIONÁLIS FELADAT – IMSC PONTOKÉRT	15
1. AZ IVIEW EGY INTERFÉSZ, EZÉRT A GetDocument/Update, Stb. KÓDJÁT NEM LEHET IMPLEMENTÁLNI BENNE. HELYETTE MINDEN NÉZETBEN „COPY-PASTE”-TEL DUPLIKÁLNI KELL A MEGFELELŐ KÓDOT. TUDNA-E ELEGÁNSABB MEGOLDÁST JAVASOLNI? (1 PONT).....	15
2. BIZTOSÍTSON LEHETŐSÉGET A GRAFIKON GÖRGETÉSÉRE (1 PONT).....	15
3. BIZTOSÍTSON LEHETŐSÉGET JELEK ÉLŐ GENERÁLÁSÁRA ÉS MEGJELENÍTÉSÉRE (3 PONT).....	16

Bevezetés

A feladat megértése szempontjából kulcsfontosságú a document-view architektúra részletekbe menő ismerete, pl. az előadásanyag alapján.

Kapcsolódó előadások:

- C# property, delegate, event alkalmazástechnikája
- Windows Forms alkalmazások fejlesztésének alapjai (Form, vezérlőelemek, eseménykezelés)
- Grafikus megjelenítés Windows Forms alkalmazásokban
- Document-View architektúra elméleti ismerete (09-10 Architektúrák előadás része) és alkalmazása egyszerű környezetben.
- UserControl és használata

Kapcsolódó laborgyakorlatok:


- „3. Felhasználói felület kialakítása” laborgyakorlat
- „6. Document-View architektúra” laborgyakorlat

Ezeket a laborgyakorlatot a hallgatók a gyakorlatvezető útmutatásával, a gyakorlatvezetővel közösen vezetett módon végzik/végezték el. A laborgyakorlathoz útmutató tartozik, mely részletekbe menően bemutatja az elméleti hátteret, valamint lépésenként ismerteti a megoldás elkészítését.

Az önálló gyakorlat célja:

- UML alapú tervezés és néhány tervezési minta alkalmazása
- A Document-View architektúra alkalmazása a gyakorlatban
- UserControl szerepének bemutatása Window Forms alkalmazásokban, Document-View architektúra esetén
- A grafikus megjelenítés elveinek gyakorlása Window Forms alkalmazásokban (Paint esemény, Invalidate, Graphics használata)

A feladat publikálásának és beadásának alapelvei megegyeznek az előző feladatával, pár kiemelt követelmény:

- A feladathoz tartozó GitHub Classroom hivatkozás: https://classroom.github.com/a/Liw_Mo4k
A munkamenet megegyezik az előző házi feladatával: a fenti hivatkozással mindenkinek születik egy privát repója, abban kell dolgozni és a határidőig a feladatot beadni.
-  Az elindulással **ne várd meg a határidő közeledtét**, legalább a saját repó létrehozásáig juss el mielőbb. Így ha bármi elakadás lenne, még időben tudunk segíteni.

- **i** A repository gyökérmappájában található neptun.txt fájlba írd bele a Neptun kódod, csupa nagybetűvel. **De csak akkor, ha a megoldással elkészültél, a feladatot beadottnak tekinted.**
- **i** A beadott megoldások mellé külön indoklást, illetve leírást nem várunk el, ugyanakkor az elfogadás feltétele, hogy a beadott kódban a „Feladat 3 – Jelek grafikus megjelenítése, saját nézet osztály” fejezet feladatainak a megoldását **kommentekkel kell ellátni. A többi fejezet feladatainak megoldását NEM kell kommentezni.**
- A munka során a kiindulási repóban levő solutionben/projektben kell dolgozni, új projektet/solutiont ne hozz létre.
- Amikor a házi feladatod beadottnak tekinted, célszerű ellenőrizni a GitHub webes felületén a repository-ban a fájlokra való rápillantással, hogy valóban minden változtatást push-oltál-e.
- Emlékeztető: *valamennyi házi feladatot a megadott határidőig a tanszéki portálra (<https://www.aut.bme.hu/>) is fel kell tölteni egy zip csomagban!*
 - A tanszéki portálra feltöltendő zip állománynak tartalmaznia kell a megoldás(ok) forráskódját. További szabályok:
 - A zip állomány nem tartalmazhatja a kimeneti („.exe”) és köztes állományokat! Ezen állományok törléséhez a Solution mappából kézzel törölni kell a „bin” és „obj” mappákat teljes tartalmukkal együtt.
 - A zip állományba ne tegyük bele a „.git” és „.vs” könyvtárat.

Feladatok áttekintése

Feladatkiíráshoz tartozó melléklet

Egy Visual Studio solution, mely a feladat kiindulásául szolgál.

Feladatléírás

- Egy olyan vastagkliens (Windows Forms) alkalmazást kell elkészíteni, amely képes fájlban időbélyeggel tárolt mérési értékek grafikus megjelenítésére. Az alkalmazásnak a Document-View architektúrát kell követnie.
- Egyszerre több dokumentum is meg lehet nyitva, illetve egy dokumentumnak több nézete is lehet. A főablak egy TabControl-t tartalmaz, melyen minden nézet egy külön tabfülön jelenik meg.
- Egy dokumentum létrehozásakor/megnyitásakor egy nézet (tabfül) jön létre hozzá, de utólag a Window/New View menüelem kiválasztásával új nézet/tabfül is létrehozható. Egy dokumentumhoz azért van értelme több nézetet megjeleníteni, mert az egyes nézetek eltérő nagyításban képesek az adott dokumentum jeleit megjeleníteni.
- A jelek kirajzolása mellett jelenítse meg a koordináta tengelyeket is.

Írányelvek

- A megvalósítás során használjon beszédes változóneveket, pl. pixelPerSec.

- Amennyiben a programozási feladatok megvalósítása során „inconsistent visibility”-re vagy „inconsistent accessibility”-re panaszkodó fordítási hibaüzenetekkel találkozunk, ellenőrizzük, hogy valamennyi típusunk (osztályunk, interfészünk) láthatósága publikus-e, a class/interface kulcsszó előtt adjuk meg a public módosítót. Pl.:

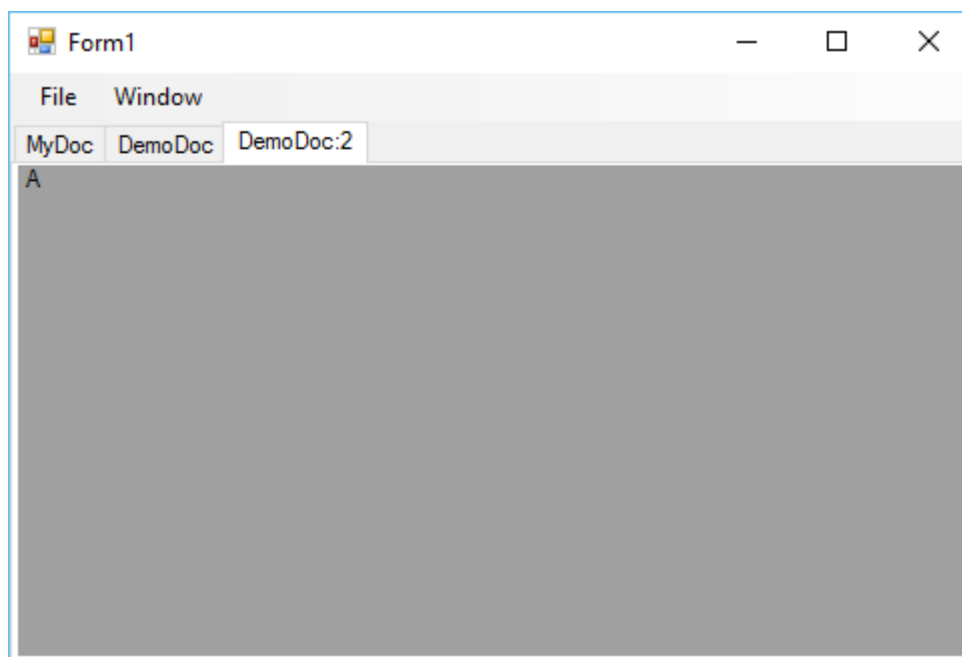
```
public class MyClass  
{ ... }
```

Feladat 1 - A kiindulási környezet megismerése

Töltsük le a tárgy honlapjáról a kiindulási keretet. Ez egy Visual Studio solution, amely egy Document-View keretet tartalmaz. Futtatva teszteljük a kiindulási alkalmazást:

- A File/New menü egy új dokumentumot hoz létre. Első lépésben bekéri a dokumentum nevét, majd létrehozza a dokumentumot és a nézetet a hozzá tartozó tabfüllet.
- A File/Open és File/Save menüelemekhez lényegi implementáció egyelőre nem tartozik.
- A File/Close bezárja az aktuális dokumentumot/tabfület.
- A Window/New View egy új nézetet/tabfület hoz létre az aktuális dokumentumhoz. Amennyiben egy dokumentumhoz több nézet is tartozik, a 2. nézettel kezdve a tabfülön a nézet sorszáma is megjelenik.

A főablakunk a következőképpen néz ki, ha két dokumentumot hoztunk létre, és a másodikhoz két nézetet:



A megjegyzésekkel ellátott forráskódot nézve ismerkedjünk meg a keret architektúrájával, működésével.

A fontosabb osztályok a következők:

- **MainForm osztály:** Az alkalmazás főablaka. Egy TabControl-t tartalmaz, ahol megjelennek az egyes dokumentumok nézetei. Kezeli a MenuStrip eseményeit, a többségük kezelőfüggvényében egyszerűen továbbhív az App osztályba (vagyis a logika nem a form osztályban van megírva).
- **App osztály:** Az alkalmazást reprezentálja. Egy példányt kell létrehozni belőle az Initialize hívásával, ez lesz az alkalmazásunk „root” objektuma. Ez bármely osztály számára hozzáférhető az App.Instance statikus property-n keresztül (erre több példát is látunk a főablak menü eseménykezelőiben).

Tárolja a dokumentumok listáját. Legfontosabb tagjai a következők:

 - documents: Valamennyi megnyitott dokumentumot tartalmazó lista.
 - activeView: Az aktív nézetet adja vissza. Ezt az aktív tabpage határozza meg. Tabváltáskor mindig frissítésre kerül. A TabPage-ek a Tag property-jükben tárolják azt a nézet objektumot, amit megjelenítenek.
 - ActiveDocument: Az aktív dokumentumot adja vissza. Az aktív tabpage meghatározza, melyik az aktív view, a view pedig tárolja a dokumentumot, amihez tartozik.
 - NewDocument: Létrehoz egy új dokumentumot, a hozzá tartozó nézettel. Alaposan tanulmányozzuk át az implementációt, az általa hívott függvényeket is beleértve!
 - CreateViewForActiveDocument: Egy új nézetet hoz létre az aktív dokumentumhoz. A Window/New View menüelem kiválasztásának hatására hívódik meg.
 - CloseActiveView: Bezárja az aktív nézetet.
- **Document osztály:** Az egyes dokumentum típusok őssztálya. Bár esetünkben csak egy dokumentum típus létezik, a későbbi bővíthetőség miatt célszerű külön választani. Tartalmazza a nézetek listáját, melyek a dokumentumot megjelenítik. Az UpdateAllViews művelete valamennyi nézetet értesít, hogy frissítsék magukat. A LoadDocument és SaveDocument üres virtuális függvények, melyek a dokumentum betöltésekor és mentésekor kerülnek meghívásra. A Document leszármazott osztályunkban kell felüldefiniálni és értelemszerűen megvalósítani őket.
- **IView:** Az egyes nézetek közös interfésze. Azért nem osztály, mert a nézetek tipikusan a UserControl-ból származnak le, és egy osztálynak nem lehet több őssztálya .NET környezetben.

- **DemoView:** Egy demo nézet implementáció nézetre. Mintaként szolgálhat saját nézet létrehozásához. A `UserControl` osztályból származik, és implementálja az `IView` interfészt.

Az osztályok közötti kapcsolatok jobb megértését segíti a solutionben található `ClassDiagram1.cd` UML osztálydiagram.

Feladat 2 – Mérési értékek reprezentálása, saját dokumentum osztály, adatok mentése és betöltése

1. Vezessen be egy osztályt a jelértékek reprezentálására

Legyen az osztály neve `SignalValue`, és egy `Value` (`double`) mezőben tárolja a mért értéket, az időbélyeget pedig egy `TimeStamp` (`DateTime`) mezőben. Mivel ezeket nem akarjuk a kezdeti inicializálás után megváltoztatni, definiáljuk őket csak olvashatónak (`readonly` kulcsszó).

Az osztálynak legyen olyan kétparaméteres konstruktora, mely paraméterben megkapja jelértéket és az időbélyeget, és ez alapján inicializálja a tagváltozókat.

Írjuk felül az `Object`-ből örökölt `ToString` műveletet, hogy formázottan jelenítse meg az objektum tagváltozóit. Segítség:

```
public override string ToString()
{
    return string.Format("Value: {0}, TimeStamp: {1}", Value, TimeStamp);
}
```

A formázás során sztring iterpoláció is használható (de ez nem elvárás).

2. Vezessen be egy saját dokumentum osztályt a dokumentumhoz tartozó jelértékek tárolására.

Legyen az osztály neve `SignalDocument`, származzon a `Document` osztályból, és egy `signals` nevű `List<SignalValue>` típusú tagban tárolja a jeleket.

Megjegyzés: az ős `Document` nem rendelkezik default konstruktossal, ezért kell írjunk a leszármazottunkban megfelelő konstruktort:

```
public SignalDocument(string name)
    : base(name)
{
    signals.AddRange(testValues);
}
```

Módosítsuk az `App.NewDocument` függvényt, hogy a leszármazott `SignalDocument`-et példányosítsa.

3. Gondoskodjon a dokumentum által tárolt adatok elmentéséről

A tesztelést segítőként inicializálja a `SignalDocument`-ben tárolt jelérték listát úgy, hogy mindig legyen benne néhány elem. Célszerű ezeket egy külön tagváltozóban felvenni. Az alábbi kód arra is példát mutat, hogyan lehet C# nyelven a tömb elemeit az inicializálás során egyszerűen megadni (a megvalósítás során ne az alábbi példában szereplő értékeket használja):

```

class SignalDocument : Document
{
    ...

    private List<SignalValue> signals = new List<SignalValue>();

    private SignalValue[] testValues = new SignalValue[]
    {
        new SignalValue(10, new DateTime(2017, 1, 1, 0, 0, 0, 111)),
        new SignalValue(20, new DateTime(2017, 1, 1, 0, 0, 1, 876)),
        new SignalValue(30, new DateTime(2017, 1, 1, 0, 0, 2, 300)),
        new SignalValue(10, new DateTime(2017, 1, 1, 0, 0, 3, 232)),
        new SignalValue(-10, new DateTime(2017, 1, 1, 0, 0, 5, 885)),
        new SignalValue(-19, new DateTime(2017, 1, 1, 0, 0, 6, 125))
    };

    public SignalDocument(string name)
        : base(name)
    {
        // Kezdetben dolgozzunk úgy, hogy a signals
        // jelérték listát a testValues alapján inicializáljuk.
        signals.AddRange(testValues);
    }
    ...
}

```

Következő lépésben írja meg az `App.SaveActiveDocument` függvényt a forráskódban található megjegyzéseknek megfelelően. A `SaveFileDialog` használatára az MSDN Library-ben talál példát (célszerű pl. a Google-ben rákeresni). Az MSDN-ben ismertetett példa megtévesztő lehet, mert a dialógus meg is nyitja a fájlt. Esetünkben erre semmi szükség, csak egy fájl útvonalat szeretnénk szerezni, hiszen a fájl megnyitása a dokumentum osztályunk feladata.

Segítség a megoldáshoz:

```

/// <summary>
/// Elmenti az aktív dokumentum tartalmát. Nincs implementálva.
/// </summary>
public void SaveActiveDocument()
{
    if (ActiveDocument == null)
        return;

    // Útvonal bekérése a felhasználótól a SaveFileDialog segítségével.
    // http://msdn.microsoft.com/en-us/library/system.windows.forms.savefiledialog.aspx
    // ...
    SaveFileDialog saveFileDialog = new SaveFileDialog();
    // Megjelenítés előtt paramlíterezzük fel a dialógus ablakot
    saveFileDialog.Filter = "txt files (*.txt)|*.txt|All files (*.*)|*.*";
    saveFileDialog.FilterIndex = 0;
    saveFileDialog.RestoreDirectory = true;

    // Modálisan megjelenítjük a dialógusablakot.
    // Ha a felhasználó nem az OK gommbal zárta be az ablakot,
    // nem csinálunk semmit (visszatérünk)
    if (saveFileDialog.ShowDialog() != DialogResult.OK)
        return;

    // A dokumentum adatainak elmentése.
    // A saveFileDialog.FileName tartalmazza a teljes útvonalat.

```



```

        ActiveDocument.SaveDocument(saveFileDialog.FileName);
    }

```

A következő lépésben definiálja felül a `SignalDocument` osztályban az örökölt `SaveDocument` függvényt, melyben írja ki a tárolt jelértékeket, időbélyeggel együtt. A mentés során arra törekszünk, hogy tömör, mégis olvasható formátumot kapjunk. Ennek megfelelően az XML és a bináris formátum nem javasolt. Kövessük a következő szöveges formátumot:

```

10    2008-12-31T23:00:00.1110000Z
20    2008-12-31T23:00:01.8760000Z
30    2008-12-31T23:00:02.3000000Z
10    2008-12-31T23:00:03.2320000Z

```

Az első oszlopban a jelértékek, a másodikban az időpont található, az oszlopok tabulátor karakterrel szeparáltak ('`\t`'). Az időpont legyen UTC idő, hogy ha a fájlt más időzónában töltik be, akkor is a helyes helyi időt mutassa. Az megfelelő string konverzió a következő:

```

string dt = myDateTime.ToUniversalTime().ToString("o");

```

Szöveges adatok fájlba írására a `StreamWriter` osztályt használjuk. Figyelem: megoldásunkban garantáljuk, hogy kivétel esetén is lezáródjon a fájlunk: használjunk `try-finally` blokkot, vagy alkalmazzunk `using` blokkot:

```

using (StreamWriter sw = new StreamWriter(filePath))
{
    ...
}

```

Az alkalmazást futtatva tesztelje a mentés funkciót. Ennek során ellenőrizze, hogy a fájlban valóban az elvárásoknak megfelelő formátumban kerülnek-e kiírásra az adatok. Ehhez indítsuk el az alkalmazást, hozzunk létre egy új dokumentumot, majd a `File/Save` menü kiválasztásával mentsük el.

4. Biztosítson lehetőséget dokumentum fájlból betöltésére

Írja meg az `App.OpenDocument` függvényt a benne szereplő megjegyzéseknek megfelelően, kövesse az ott megadott lépéseket.

A következő lépésben definiálja felül a `SignalDocument` osztályban az örökölt `LoadDocument` függvényt, melyben töltse fel a tárolt jelérték listát a fájl tartalma alapján. Szöveges adatok fájlból beolvasására a `StreamReader` osztályt használjuk, a mentéshez hasonlóan `try/finally` vagy `using` blokkban. Segítségképpen:

- Amennyiben van egy `sr` nevű `StreamReader` objektumunk, a fájl soronkénti beolvasása a következőképpen lehetséges:

```

while ((line = sr.ReadLine()) != null)
{
    // A line változóban benne van az aktuális sor
    ...
}

```

- Az üres, vagy csak whitespace karaktereket tartalmazó sorokat át kell ugrani. A `string.Trim` használható a whitespace karakterek kiszűrésére, pl.:

```
s = s.Trim();
```

- Az oszlopok tab karakterrel szeparáltak. Egy sztring adott karakter szerinti vágására kényelmesen használható a `string` osztály `Split` művelete, pl.:

```
string[] columns = line.Split('\t');
```

- Sztringből `double`-t, illetve `DateTime` objektumot a `<típusnév>.Parse(str)` függvénnyel lehet pl. kinyerni:

```
double d = double.Parse(strValue);
...
DateTime dt = DateTime.Parse(strValue);
```

- A fájlban UTC időbélyegek szerepelnek, ezt a dokumentum osztályban tárolás előtt konvertáljuk lokális időre:

```
DateTime localDt = utcDt.ToLocalTime();
```

- Miután beolvastuk az adott sort, hozzunk létre egy új `SignalValue` objektumot a beolvasott értékekkel inicializálva, és vegyük fel a `signals` listába.

Megjegyzés: a `LoadDocument` függvény elején a `signals` feltöltése előtt töröljük ki a `Clear` művelettel a benne levő elemeket. Enélkül ugyanis a konstruktorban hozzáadott teszt jelértékek benne maradnának.

5. A betöltést követően ellenőrizze a betöltés sikerességét

Mivel grafikus megjelenítéssel még nem rendelkezik az alkalmazás, más megoldást kell választani. Nyomkövetésre, diagnosztikára a `System.Diagnostics` névtér osztályai használhatók. A `Trace` osztály „Debug” build esetén a `Write/WriteLine` utasítással kiírt adatokat trace-eli, ami alapértelmezésben azt jelenti, hogy megjeleníti a Visual Studio Output ablakában. Írjunk egy `TraceValues` segédfüggvényt a `SignalDocument` osztályba, ami trace-eli a tárolt jeleket:

```
void TraceValues()
{
    foreach (SignalValue signal in signals)
        Trace.WriteLine(value.ToString());
}
```

Hívjuk meg a `TraceValues`-t a betöltő függvényünk (`LoadDocument`) végén, és ellenőrizzük a működést: az F5 billentyű lenyomásával debug módban indítsuk el az alkalmazást, a `File/Open` kiválasztásával töltsünk be egy korábban elmentett fájlt. A művelet végén ellenőrizzük, hogy a Visual Studio Output ablakában (View/Output menüvel jeleníthető meg) kiíródnak-e a fájlból betöltött jelek adatai.

Feladat 3 – Jelek grafikus megjelenítése, saját nézet osztály

Lényeges: ezen főfejezet feladatainak megoldását kommentekkel kell ellátni!

1. Vezessen be egy új nézet osztályt UserControl formájában

A nézetet `UserControl`-ként valósítjuk meg. A téma elméleti háttere az előadásanyagban megtalálható. Következzen pár fontosabb gondolat ismétlésképpen. A `UserControl` alapú megközelítéssel olyan saját vezérlőt készíthetünk, melyek az

űrlapokhoz (`Form`) hasonlóan más vezérlőket tartalmazhatnak. Számos pontban nagyon hasonlítanak az űrlapokhoz, pl.:

- Két forrásfájl tartozik hozzájuk. Egy, amiben mi dolgozunk, és egy `designer.cs` végződésű, melybe a Visual Studio generál kódot. Megjegyzés: a fejlesztők számára dedikált forrásfájlt többféleképpen lehet megnyitni:
 - A *Solution Explorer* összevontan jeleníti meg a forrásfájlokat: ezen jobb gombbal kattintva a *View Code* elemet válasszuk a menüben.
 - Amennyiben duplakattal megnyitottuk a `UserControl`-t szerkesztésre, a szerkesztőfelületen jobb gombbal kattintva válasszuk a *View Code* menüt.
 - F7 billentyű használatával.
- Amikor saját űrlapot készítünk, a beépített `Form` osztályból kell egy saját osztályt leszármaztatni. Saját `UserControl` esetében a beépített `UserControl` osztályból kell származtatni. Ezt ritkán szoktuk manuálisan megtenni, általában a Visual Studio-ra bízunk (pl. *Project/Add UserControl* menü).
- Hasonlóan a *Solution Explorer*-ben duplán kattintva rajtuk tudjuk megnyitni a felületüket szerkesztésre, a *Toolbox*-ból tudunk más vezérlőket elhelyezni a felületükön, melyekből a `UserControl` osztályunkban tagváltozók lesznek.
- Hasonló módon tudunk eseménykezelőket készíteni (magához a `UserControl`-hoz, vagy a rajta levő vezérlőkhöz).
- Ugyanúgy tudunk felületére rajzolni. Vagy a `Paint` eseményhez rendelünk eseménykezelőt, vagy felüldefiniáljuk az `OnPaint` virtuális függvényt.

Abban természetesen különbözik az űrlapoktól, hogy míg az űrlapok, mint önálló ablakok a `Show` vagy `ShowDialog` műveletekkel megjeleníthetők, a `UserControl`-ok vezérlők, melyeket űrlapokon vagy más vezérlőkön kell elhelyezni.

Visszetérve a feladatra a megvalósítás főbb lépései a következők:

- Az új nézet tehát egy `UserControl` legyen. Saját `UserControl`-t felvenni pl. a *Project/Add UserControl* menüvel lehet. Legyen a neve `GraphicsSignalView` (jelezve, hogy ez egy grafikus nézet, és nem karakteresen jeleníti meg a jeleket).
- Bővítse az osztályt a `DemoView` mintájára (többek között implementálja az `IView` interfészt). A `DemoView` a `dokumentumra` és `Document` típusként hivatkozik, lásd `Document document;` tagváltozó. A `GraphicsSignalView`-ban célszerű a specifikusabb, `SignalDocument` típusúnak definiálni a tagváltozót!
- Módosítsa az `App.createView()`-t, hogy `DemoView` helyett `GraphicsSignalView`-t hozzon létre. Hogy ez működhessen, a `GraphicsSignalView`-ba fel kell vennie egy konstruktort a következőnek megfelelően (hagyjuk meg a default konstruktort is):

```
public GraphicsSignalView(SignalDocument document)
{
    InitializeComponent();
    this.document = document;
}
```

Az `App.CreateView` módosításának van még egy trükkje. Mivel a `doc` referenciánk típusa `Document`, a `GraphicsSignalView` meg a leszármazottját várja, a konstruktor hívásakor explicit le kell castoljuk `SignalDocument`-re:

```
GraphicsSignalView view = new GraphicsSignalView((SignalDocument)doc);
```

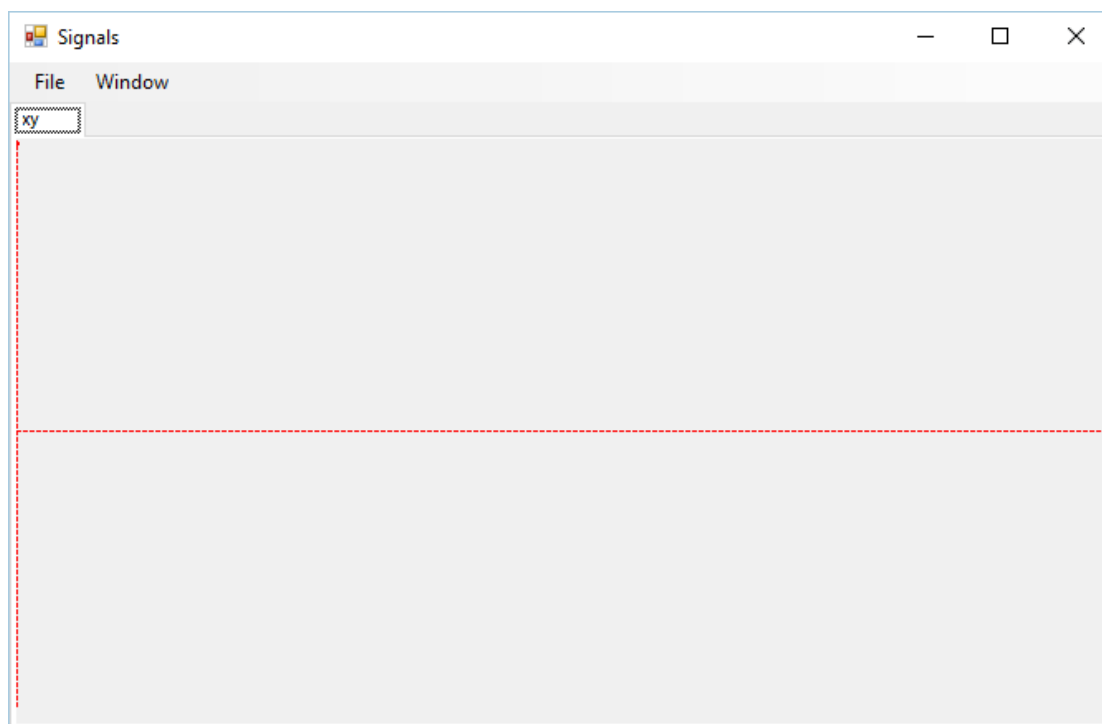
2. Rajzolja ki a koordináta tengelyeket

Legyen az alapelvünk a következő:

- A rajzolófelületünk (vagyis a `GraphicsSignalView` `UserControl`) kliens területének szélességét a `ClientSize.Width`, a magasságát a `ClientSize.Height` lekérdezésével kaphatjuk meg. Vonalat rajzolni a `Graphics` osztály `DrawLine` műveletével lehet.
- Az Y tengelyt a nulla y pixelpozícióba rajzoljuk.
- Az X tengelyt mindig a rajzolófelületünk közepére igazítva rajzoljuk, akárhogy méretezi is a felhasználó az ablakot (segítségképpen: a teljes aktuális magasságot a `ClientSize.Height` adja meg számunkra).
- A koordináta tengelyek színe bármilyen általunk választott szín lehet, de ne legyen fekete. **A tengelyeket szaggatott vonallal rajzoljuk, és a végükön legyen egy kisméretű nyíl.** Erre a beépített `Pen` támogatást nyújt:

```
Pen pen = new Pen(Color.Red, 1);
pen.DashStyle = DashStyle.Dash;
pen.EndCap = LineCap.ArrowAnchor;
```

A munkánk eredményeképpen valami hasonlót kell lássunk futás közben (persze csak ha megnyitunk egy létező vagy létrehozunk egy új dokumentumot, máskülönben nincs is nézetünk!):



3. Valósítsa meg a jelek megjelenítését

Az `GraphicsSignalView`-ban az `OnPaint`-t `override`-olva valósítsa meg a jelek kirajzolását. Először 3×3 pixeles „pontokat” rajzoljon (pl. `Graphics.FillRectangle`-lel), majd a pontokat kösse össze vonalakkal (`Graphics.DrawLine`).

A megvalósításban segíthet a következő:

- Az `OnPaint` művelet a megjelenítés során el kell érje a `SignalDocument`-ben tárolt `SignalValue` objektumokat. Ehhez a `SignalDocument` osztályban vezessünk be egy publikus property-t (a `SignalDocument`-ben a `signals` tag privát, és ez maradjon is így):

```
public IReadOnlyList<SignalValue> Signals
{
    get { return signals; }
}
```

Figyeljük meg, hogy az objektumokat nem `List<SignalValue>`-ként, hanem `IReadOnlyList<SignalValue>` formában adjuk vissza: így a hívó nem tudja módosítani az eredeti listát, nem tudja véletlenül se elrontani a tartalmát.

- Két `DateTime` érték különbsége egy `TimeSpan` típusú objektumot eredményez.
- Egy `DateTime` objektum a `Ticks` property-jében adja vissza legjobb felbontással az általa tárolt időértéket (1 tick = 100 nsec felbontás).
- A rajzolófelületünk (vagyis a `GraphicsSignalView` `UserControl`) nulla x koordinátájában jelenítsük meg a listánkban levő első jelet.
- A megjelenítés során semmiféle követelmény nincs arra vonatkozóan, hogy a jeleket olyan skálátényezőkkal jelenítsük meg, hogy pont kiferjenek a rajzolás során. Helyette a view osztályunkban vezessünk be és használjunk olyan

`pixelPerSec` és `pixelPerValue` skálatényezőket, melyek észre, vagy pár próbálkozás után úgy jelenítsék meg a jeleket, hogy a nézetbe beférjenek, de ne legyen a rajz túl kicsi.

Amennyiben a rajzunk „nem akar” megjelenni, tegyük töréspontot az `OnPaint` műveletbe, és a kódukot lépésenként végrehajva a változók értékét tooltipben vagy a Watch ablakban megjelenítve nyomozzuk, hol csúszik félre a számításunk.

Ha jól dolgoztunk, a következőhöz hasonló kimenetet kapunk:



4. Biztosítson lehetőséget a nézet nagyításra és kicsinyítésére. Ehhez helyezzen el egy kisméretű, + és – szöveget tartalmazó nyomógombot a nézeten.

Lépések:

- Nyissa meg a `GraphicsSignalView UserControl`-t szerkesztésre.
- A Toolbox-ról drag&drop-pal helyezzen el rajta két gombot (`Button`).
- Nevezze el a gombokat megfelelően és állítsa be a szövegüket (`Text` property)
- Rendeljen eseménykezelőt a gombok `Click` eseményéhez (ehhez csak duplán kell a gombokon kattintani a szerkesztőben).
- Vezessen be a view-ban egy `double` típusú skálatényezőt, melynek kezdőértéke legyen 1. Nagyításkor ezt növelje (pl. 1,2-szeresére), kicsinyítéskor csökkentse (pl. ossza 1,2-vel). Az `OnPaint` műveletben, mikor az y és x pixelkoordinátákat számolja, az végső eredmény számításakor a koordinátákat szorozza be az aktuális skálatényezővel. A skálatényező változtatása után ne felejtse el meghívni az `Invalidate` műveletet!

A következőhöz hasonló kimenetet a cél (némi nagyítást követően):



Az alkalmazást futtva a Window menüből ugyanahhoz a dokumenthoz hozzunk létre egy új nézetet, és a nagyítás/kicsinyítés gombokat használva, valamint a nézetek között váltogatva ellenőrizzük, hogy a nézetek ugyanazokat az adatokat jelenítik meg, de eltérő nagyításban.

Optionális feladat – IMsc pontokért

Az egyes feladatok egymástól függetlenül is megoldhatók!

- 1. Az IView egy interfész, ezért a GetDocument/Update, stb. kódját nem lehet implementálni benne. Helyette minden nézetben „copy-paste”-tel duplikálni kell a megfelelő kódot. Tudna-e elegánsabb megoldást javasolni? (1 pont)**

Ötlet: Egy `ViewBase` nevű osztályt kell írni, mely a `UserControl`-ból származik és implementálja az `IView` interfészt. A nézeteinket a `UserControl` helyett a `ViewBase` osztályból kell származtatni.

Alakítsa át a megoldást ennek megfelelően.

- 2. Biztosítson lehetőséget a grafikon görgetésére (1 pont)**

A megvalósításban használhatunk egyedi scrollbar-t is, de ennél egyszerűbb a `UserControl` autoscroll támogatását felhasználni. Ehhez először engedélyezzük az `AutoScroll`-t a properties ablakban a nézethez tartozó `UserControl`-ra. Ezt követően meg kell adni a rajzolófelület nagyságát, aminél ha kisebb a `UserControl`-unk mérete, akkor automatikusan megjelenik a megfelelő scrollbar:

```
this.AutoScrollMinSize = new Size(widthInPixels, heightInPixels);
```

Ezt követően a kirajzolás során a rajzot az aktuális scroll pozíciónak megfelelően el kell tolni. Erre a legegyszerűbb megoldás, ha egy, a scroll pozíciónak megfelelő eltolást

eredményező transzformációs mátrixot állítunk be a `Graphics` objektumra a kirajzolás előtt:

```
Matrix transform = new Matrix(1, 0, 0, 1, AutoScrollPosition.X,  
    AutoScrollPosition.Y);  
g.Transform = transform;  
... rajzolás ...
```

A megközelítés előnye a viszonylagos egyszerűsége. Hátránya, hogy ha nagyon sok jelünk van, de annak csak egy kis szelete látható egy adott pillanatban, attól még a `Paint` függvényünkben a nem látható jeleket is kirajzoljuk. Egy optimalizált megoldásban csak a látható tartományt célszerű megjeleníteni.

3. Biztosítson lehetőséget jelek élő generálására és megjelenítésére (3 pont)

Az átalakítást olyan módon kell végrehajtani, hogy a korábbi feladatok megoldása működőképes maradjon (az alapértelmezett, nem élő módban).

Vezessen be egy `Data` menüelem alatti „Change To Live Data Source Mode” menüelemet. Amikor a felhasználó erre kattint, indítson egy szálat, mely véletlenszerű jelértékeket generál (azt szimulálva, hogy valamilyen adatforrásból, pl. soros port, hálózat, stb. adatok érkeznek) a következőknek megfelelően:

- A „Change To Live Data Source Mode” az aktuális dokumentumra vonatkozik. Vagyis minden dokumentum egymástól függetlenül élő adatforrás módba kapcsolható. Ennek megfelelően a jelek generálását dokumentum szinten célszerű megvalósítani (dokumentumonként külön szál).
- Másodpercenként nagyságrendileg 4 jelérték érkezzen véletlen időközönként (pl. szál altatása véletlen időközre, 500 ms max értékkel), melyek időbélyege legyen az aktuális idő (`DateTime.Now`).
- A generált értékek kerüljenek bele az aktuális dokumentum jelérték halmazába (fűzze a végére).
- A megjelenítés során **nem** kell azzal foglalkoznia, hogy a nézet automatikusan úgy nagyítsa/kicsinyítse/görgeesse a felületet, hogy az érkező adatok láthatóak legyenek. Vagyis semmiféle automatizmust nem kell megvalósítania, ha az érkező adatok a megjelenítési tartományon kívül esnek: a felhasználó feladata, hogy úgy nagyítsa/kicsinyítse/görgeesse a felületet, hogy azok láthatók legyenek.
- A megjelenítés során **nem** kell optimalizációval foglalkoznia. Vagyis nem kell gondoskodni arról, hogy mindig csak az újonnan érkező adatok kerüljenek kirajzolásra, vagy, hogy ne villogjon a felület az újrarajzolás során.

Tippek a megvalósításhoz:

- Egy élő adatforrás módban levő dokumentum esetén nem elvárás, hogy ha új érték születik, a dokumentum nézetei azt mielőbb megjelenítsék. Vagyis teljesen elfogadható - sőt, célszerű - megoldás, ha egy élő módban levő dokumentum nézetei másodpercenként néhányszor (pl. ötször) ellenőrzik, érkezett-e új adat, és szükség esetén frissítik magukat (pull modell). Azt, hogy érkezett-e új adat, egy nézet pl. úgy tudja eldönteni, ha eltárolja az utoljára megjelenített jelek számát, és összehasonlítja az aktuális jelszámmal.