**Mérési segédlet**

**IV. mérés – Windows Forms I**

A mérés során a hallgatók a .NET keretrendszer felületkezelésével, felhasználói interfészével ismerkedhetnek meg. Mivel a mérés igen sokrétű, ezért fontos, hogy valamilyen formában (papíros vagy elektronikus) a mérésen a hallgatók rendelkezzenek a segédlettel!

#### Form - felülettervezés

A Form osztály az ablakot reprezentálja, és egyben a konténer-vezérlő kapcsolatban a legfelső konténer.

A felület kialakítása szempontjából az alkalmazásunk lehet:

* Dialógus alapú: Kizárólag vezérlőket helyezünk el a form-on, mintha egy dialógus ablak lenne. Ha szükséges új ablakot nyitunk az egyes funkcióknak. Pl. Üzleti alkalmazások.
* SDI, vagy MDI: Dokumentum alapú alkalmazás, amelyben a form a dokumentum megjelenítője, és esetleg szerkesztője, az egyéb vezérlőket/funkciókat a menübe és a toolbar-ra tesszük. Az SDI (Single Document Interface) egy objektumot kezel egy időben, az MDI (Multiple Document Interface) pedig többet. Pl. szövegszerkesztők.
* Vegyes: Az ilyen jellegű alkalmazásokban a dokumentum szerkesztése a cél csakúgy, mint az SDI/MDI változatokban, azonban az ablak egy része fenn van tartva vezérlők számára, ahol könnyen elérhetjük a funkciókat. Pl. CAD alkalmazások.

#### Konténer-vezérlő hierarchia

Egy ablak hierarchikus (fa) felépítésű, amelyben a gyökérobjektum maga a Form. Alatta újabb konténerek lehetnek egymásba ágyazva, vagy csak egymás mellett. A hierarchia alján vannak a vezérlők, de lehetnek vezérlő nélküli konténerek is.

Az egymásba ágyazás azért szükséges, hogy egységként lehessen kezelni a konténereket és a gyerekeiket, így például odébb húzva a konténert mennek vele a vezérlők is. A másik fontos ok a tulajdonságöröklés, amely lehetővé teszi, hogy ha megváltoztatjuk valamelyik konténer örökölhető tulajdonságát (pl. Font), akkor azt a gyerekei is megöröklik. Ez nem a szokásos objektum-orientált öröklés, de a szülő-gyermek viszony azonos elvre épül.

#### Üzenetkezelés

A WinForms alapú alkalmazások üzenetkezelésre épülnek, amelynek a hátterében az operációs rendszer felépítése áll. Az üzenetkezelő ciklus a Main függvényben van (Application.Run), amely csak akkor lép ki, ha bezárjuk az alkalmazásunkat.

A vezérlők is üzenetekkel kommunikálnak, amelyek tulajdonságként jelennek meg számunkra. A TextBox.Text tulajdonság beállítása például üzenetet küld a vezérlőnek, amely erre átállítja a szöveget.

#### Visual Studio Designer

A designer a felhasználói felület szerkesztője, amelyben lehetőségünk van új elemeket felvenni, a meglévőket módosítani és törölni. A designer fontosabb elemei/kellékei:

* Toolbox: Erről lehet a konténerek és vezérlőket ráhúzni a form-ra.
* Properties ablak: A kijelölt vezérlő tulajdonságait és eseményeit mutatja, és itt lehet szerkeszteni is őket.
* Smart tag: Minden vezrélő jobb felső sarkában található egy kis nyíl, amely előhozza. Ebben vannak a fontosabb tulajdonságok összegyűjtve.
* Document outline ablak: A konténer-vezérlő hierarchiát mutatja. Itt ki lehet jelölni az elemeket és mozgatni is lehet őket a hierarchiában átrendezve ezzel őket a form-on is.

#### Feladat – Menük

Hozzunk létre egy Blank Solution-t, majd adjuk hozzá a projekteket!

Készítsünk egy új Windows Forms alkalmazást a tervező segítségével.

Vizsgáljuk meg a teljes alkalmazást! Nézzük át, hogy a tervező mit generált!

A felülettervezés következő feladata a menük megszerkesztése. Ehhez végezzük el a következő lépéssorozatot.

1. A Toolbox-ról húzzunk rá a Form-ra egy MenuStrip-et (Menus & Toolbars kategóriában van)
2. A MenuStrip smart tag-jét kinyitva (kicsi nyíl a jobb felső sarkában) nyomjunk rá az Insert Standard Items-re.
3. Ismételjük az első kép lépést most ToolStrip-pel.
4. Majd helyezzünk fel alulra egy StatusStrip-et is.

Teszteljük az alkalmazást, vegyük észre, hogy a ToolStrip-nek van kis fogantyúja, azonban azt hiába fogjuk meg, nem mozog. Ekkor jön segítségünkre a ToolStripContainer, amely egyben jól szemlélteti a konténer-vezérlő felépítést is.

1. A menü smart tag-kében válasszuk ki az Embed In ToolStripContainer parancsot, amely feltesz egy ToolStripContainer-t, és a menüt egyből bele is teszi. A form káoszossá válik, mivel a többi strip nem került bele.
2. A ToolStripContainer smart tag-jén válasszuk a Dock Fill in Form funkciót. Akkor látszólag minden a helyére kerül, leszámítva, hogy a menü és a toolbar fel vannak cserélve.
3. Rendezzük a konténer-vezérlő hierarchiát! Nyissuk meg a Document Outline ablakot (View/Other Windows/Document Outline). Magyarázzuk el a működését.
4. Húzzuk át a ToolStrip-et és a StatusStrip-et a ToolStripContainer felső és alsó paneljébe.
5. A MenuStrip smart tag-jében állítsuk át a a Grip Style-t Visible-re, ekkor már a menü is mozgatható.

Próbáljuk ki az alkalmazást, a ToolStrip és a MenuStrip mozgatható lett (oldalra és alulra is).

Példaként implementáljuk a File/Exit menüelemet.

1. Duplán klikkeljünk a menüelemen, majd a kódba írjuk ezt:

Application.Exit();

Ebben a feladatban egy Windows Explorer-hez hasonló alkalmazás felületét készítjük el. Ehhez megismerjük a SplitContainer-t a TreeView-t és a ListView-t.

Az első részben tervezzük meg a designer-rel az AddressBar-t:

1. Rakjunk fel egy SplitContainer-t a form-ra, ez fogja a középső területet felosztani. Vegyük észre, hogy a konténernek két panelja van, amelyek szintén konténerek.
2. A SplitContainer smart tag-kéből válasszuk a Horizontal Splitter Orientation funkciót.
3. Tegyünk ki egy TextBox-ot, rakjuk rá a felső panelre, ez lesz az AddressBar. Vegyünk még fel egy Label-t is, erre írjuk rá, hogy Address. Rendezzük el őket igényesen. A SplitContainer felső paneljét vegyünk olyan kicsire, amilyenre csak lehet.
4. A SplitContainer smart tag-jéből válasszuk a Dock in parent container funkciót.

Amennyiben valamit nehéz kiválasztani, vagy csak elkavarodtunk a vezérlők között, hívjuk segítségül a Document Outline ablakot, ahol jól láthatóm hogy mit hova tettünk, és a kiválasztásban is segít.

A következő részben tervezzük meg a lényeget:

1. Tegyünk az alsó panelre egy újabb SplitContainer-t, egyből válasszuk a Dock in parent container-t a smart tag-en.
2. A bal oldali panel-re rakjunk rá egy TreeView-t. A smart tag-jében válasszuk a Dock in parent container funkciót.
3. A jobb oldali panelre rakjunk egy ListView-t. A smart tag-jében válasszuk itt is a Dock… funkciót.

Ezzel el is készült a MiniExplorer, legalábbis a felülete.

# Feladat 5 – Felület programozása

Mivel készen van a felület, a következő feladat azt kitölteni.

1. Duplán klikkeljünk a form fejlécén, ezzel tudjuk implementálni a Form.Load eseményét. Itt fogjuk kitölteni a fát:

private void Form1\_Load(object sender, EventArgs e)

{

TreeNode root = treeView1.Nodes.Add("Local Disk (C:)");

root.Tag = new DirectoryInfo("c:\\");

root.Nodes.Add("Kamu elem");

}

Magyarázzuk el, hogy a TreeView-ban mit jelent a Nodes, és a Tag tulajdonság.

1. Menjünk vissza a designer-be, válasszuk ki a TreeView-t, majd a Properties ablakban váltsunk esemény nézetre. Duplán klikkeljünk a BeforeExpand eseményen, hogy implementálhassuk:

private void treeView1\_BeforeExpand(object sender, TreeViewCancelEventArgs e)

{

DirectoryInfo parentDI = e.Node.Tag as DirectoryInfo;

e.Node.Nodes.Clear();

try

{

foreach (DirectoryInfo di in parentDI.GetDirectories())

{

TreeNode node = new TreeNode(di.Name);

node.Tag = di;

node.Nodes.Add("Kamu elem");

e.Node.Nodes.Add(node);

}

}

catch { }

}

Magyarázzuk a kódot, a fájlkezelésbe nem igazán belemenve.

Teszteljük az alkalmazást.

A jobb oldal kitöltése ugyanilyen egyszerű:

1. Az események között most kezeljük le az AfterSelect-et, amelynek az implementációja:

private void treeView1\_AfterSelect(object sender, TreeViewEventArgs e)

{

DirectoryInfo parentDI = e.Node.Tag as DirectoryInfo;

listView1.Items.Clear();

try

{

foreach (FileInfo fi in parentDI.GetFiles())

listView1.Items.Add(fi.Name);

}

catch { }

}

A hibakezelő rész fontos, mert nem minden könyvtárba fogunk tudni belépni.

Értelmezzük a kódot! Csak akkor csinálunk valamit, ha ez a könyvtár még nincs kitöltve, azaz benne van a kamu elem. Ezt onnan tudjuk, hogy csak az van benne és a Tag-je null. Ekkor töltsük fel a már ismert módon a fát, és tegyünk be kamu elemeket a nem üres könyvtárakba.

Magyarázzuk el, hogy a ListView-ban mit jelent az Items tulajdonság.

1. Ugyanitt ki is tudjuk tölteni az Address részt is:

textBox1.Text = parentDI.FullName;

Ha marad idő, akkor oldjuk meg, hogy a jobb oldal részletes listát mutasson:

1. A kódrészletet módosítsuk, hogy ne csak a nevét adja meg a fájlnak, hanem egyéb paramétereit is. Az Add függvényt paraméterezzük így:

listView1.Items.Add(new ListViewItem(new string[] { fi.Name,

fi.Length.ToString(), fi.LastWriteTime.ToString() }));

1. Mivel még nem jelenik meg semmi új dolog sem, a ListView-t módosítani kell a designer-ben, hogy mutassa a részleteket is. Ehhez hozzuk elő a smart tag-jét, és állítsuk át rajta a View-t Detail-re.
2. Az oszlopokat nekünk kell létrehoznunk, amihez a smart tag-jében válasszuk az Edit Columns funkciót, majd a megfejelenő listát töltsük fel 3 új elemmel, amelyeknek a Text tulajdonsága legyen: Name, Size, Modified. Rendezzük el szépen az oszlopok szélességét, hogy minden kiférjen majd a feltöltés után is.

#### Feladat - Custom Control készítése

Adjunk hozzá a Solution-höz egy uj projektet!

Adjunk hozzá a projektünkhöz egy Custom Controlt!

Készítsünk el két Property-t a következő módon:

private int amplitudo =50;

public int Amplitudo

{

get { return amplitudo; }

set { amplitudo = value; }

}

private int period; =100;

public int Period

{

get { return period; }

set { period = value; }

}

Ezek után definájuk felül a Controll OnPaint metódusát, és rajzoljuk ki a controlunkat.

protected override void OnPaint(PaintEventArgs pe)

{

Point[] points = new Point[2 \* (this.Width / period + 2)];

for (int i = 0; i < points.Length; i++)

{

points[i].X = period / 2 \* i;

points[i].Y = this.Height / 2 + (2 \* (i % 2) - 1) \* amplitude;

}

pe.Graphics.DrawLines(new Pen(Color.Red, 5), points);

if (this.DesignMode)

pe.Graphics.DrawString("[Design]", this.Font, new SolidBrush(Color.Red), 5, 5); // Design módban lehet feliratozni!

// Calling the base class OnPaint

base.OnPaint(pe);

}

Majd fordítsuk le az alkalmazásunkat!

A Formunkra húzzuk rá az elkészített ZigZag controlunat.

Vizsgáljuk meg az elkészített alkalmazásunkat.

Állítsuk át az Amplitudo értékét! A control nem frissül autómatikusan!

Ahhoz, hogy a control autómatikusan frissüljön, írjuk át a property-ket a következő módon:

public int Amplitude

{

get { return amplitude; }

set { amplitude = value; *this.Invalidate();* }

}

public int Period

{

get { return period; }

set { period = value; *this.Invalidate();* }

}

Futtassuk a projektünket, és a futó alkalmazás közben méretezzük át az ablakot!

Mit tapasztalunk?

A probléma megoldására módosítsuk a Contorlnak a konstruktorát a következő képen, adjuk hozzá a következő sort:

this.SetStyle(ControlStyles.ResizeRedraw, true);

Futtassuk a projektünket, és a futó alkalmazás közben méretezzük át az ablakot!

Mit tapasztalunk?

A probléma megoldására módosítsuk a Contorlnak a konstruktorát a következő képen, adjuk hozzá a következő sort:

this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);

#### Feladat

Az elkészített Controlunkat írjuk át úgy, hogy a Panel osztályból származzon.

public partial class ZigZag : Panel

Tegyünk ki rá egy buttont és vegyük észre, hogy ezúttal a gomb együtt mozog a panellel!

Állítsuk át a Form Opacity property-jét 50%-ra!

Állítsuk át a Form Transparency key értékét System/Control értékre!

Figyeljük meg, hogy az egérrel való kattintás a Form mögötti ablakra működik.