TARTALOM – HIPERHIVATKOZÁSOKKAL ELLÁTVA – SAJÁT FELELŐSSÉG!

1. **Nyelvi alapok és ismétlés**
    1. Ismertesse az automatikus tulajdonság fogalmát, írjon rá egy példát.
    2. Mi az indexelő és milyen korlátozások vonatkoznak rá?
    3. Mi a bővítő metódus (ismertetés)?
    4. Írjon egy példát bővítő metódusra.
    5. Mit jelent az, hogy egy metódus statikus? (itt legalább két fontos tulajdonságot kell megadni)
    6. Ismertesse a virtuális metódus fogalmát és jellemzőit.
    7. Mi a lambda kifejezés (ismertetés)?
    8. Írjon egy példát lambda kifejezésre.
    9. Mi a metódusreferencia (ismertetés)?
    10. Ismertesse a közzétevő/feliratkozó eseménykezelési modellt.
    11. Hogyan adhatunk át paramétereket egy eseménykezelő metódusnak? (ismertesse úgy az egyparaméteres mint a kétparaméteres eseménykezelő esetét)
2. **WPF**
    1. Hogyan írják le a felhasználói felületet a WPF alkalmazások, amikor nem C# kódból definiáljuk a felületet? Készítsen egy példát, amelyben van egy három menüpontos főmenü, alatta egy eszköztár három gombbal, és az alatt egy szövegmező.
    2. Fordítást követően hol és milyen formában tárolódik a felület leírása (WPF)?
    3. Mi a logikai fa (WPF)? Rajzoljon egy példát, és magyarázza el annak segítségével a logikai fa felépítését.
    4. Ismertesse az Application osztály (WPF) öt előadáson tanult tulajdonságát.
    5. Miért van szükség rétegmenedzserekre WPF-ben? Ismertesse a 6 tanult rétegmenedzsert (feladat+jellemzők).
    6. Ismertesse a DependencyProperty fogalmát és jellemzőit (WPF).
    7. Ismertesse, hogy hogyan oldható meg az ikonok befordítása a szerelvénybe illetve képek tárolása a szerelvényen kívül. (WPF)
    8. Milyen célt szolgálnak a "parancsok" (WPF)? Mi a CommandBinding feladata?
    9. Ismertesse a három WPF esemény típust.
    10. Mit jelent a vezérlők csatolása (WPF)?
    11. Ismertesse a validálási szabályokat. (WPF)
3. **Grafikus megjelenítés WPF felületen**
    1. Mutassa be a 2D rajzolás lehetőségeit (szintjeit) WPF-ben. Mindegyiknél nevezzen meg legalább egy előnyös és egy hátrányos tulajdonságot.
    2. Mutassa be egy-egy mondatban a transzformációhoz kapcsolódó 6 osztályt (WPF) és ábrák segítségével mutassa be a SkewTransform paramétereinek értelmezését.
    3. Ismertesse a három animáció típust egy-egy mondatban + nevezzen meg mindegyikhez legalább egy tipikus osztályt (WPF).

4. Ismertesse részletesen a lineáris animációt (WPF).
5. Ismertesse részletesen az útvonal alapú animációt (WPF).

4. Adatbázisok elérése

1. Ismertesse az ODBC-n és az OLE DB-n keresztüli adatbázis elérési módokat.
2. Ismertesse a három rétegű alkalmazás fogalmát és az egyes rétegek feladatait.
3. Hasonlítsa össze a két tanult adatelérési modellt (direct access, disconnected access).
4. Ismertesse a "Provider" objektumokat és fontosabb jellemzőiket.
5. Ismertesse a "Consumer" objektumokat és fontosabb jellemzőiket kiemelve a típusos objektumok előnyeit.
6. Ismertesse kapcsolat alapú (direct access) adatbáziselérés esetén a lekérdezés, beszúrás, módosítás és törlés lépéseit.
7. Ismertesse kapcsolat nélküli adatbáziselérés esetén a lekérdezés lépéseit, a táblák közötti kapcsolat létrehozását, valamint a módosítások végrehajtását és érvényesítését.
8. Hasonlítsa össze a típusos és típus nélküli DataSet-eket. Melyik alkalmazása előnyösebb és miért?
9. Milyen interfészt kell implementáljon egy gyűjtemény ahhoz, hogy LINQ lekérdezést hajthassunk végre rajta? Ha ezt az interfészt nem implementálja egy gyűjtemény, akkor milyen eszközzel és hogyan készíthetünk belőle egy olyan gyűjteményt, ami már lekérdezhető LINQ segítségével? Írjon egy egyszerű példát erre.
10. Tervezzen meg egy két, egymással kapcsolatban álló táblából álló típusos DataSet-et a DataSet Designer által alkalmazott grafikus jelölésmódot használva. Az ábrán szerepelnie kell a DataSet névnek, a táblaneveknek, a táblaadapter neveknek, a mezőneveknek, kulcsoknak és az idegen kulcsnak. Lássa el szöveges magyarázattal az ábrát. Írjon egy egytáblás lekérdezést, amiben szerepel a where, és az eredményrekordok tartalmaznak projekciót (select new {...}), és magyarázza azt el. Írjon egy kéttáblás (a táblák összekapcsolását tartalmazó) lekérdezést, amiben van csoportosítás (group-by-into) , és magyarázza azt el.
11. Milyen objektumra hívhatóak meg a Distinct<>(), Count<>(), Sum<>(), Min<>() és Max<>() metódusok, és milyen feladatot látnak el?
12. Írjon egy-egy példát lambda kifejezésre FindAll és Any metódusokhoz.
13. Milyen objektumok tartják nyilvánWPF alkalmazásokban azt, hogy az adatforrás mely rekordja/objektuma az aktuális?
14. Ismertesse egy-egy mondatban a három WPF adatkötési módot (OneTime, OneWay,TwoWay).
15. Ismertesse az XML dokumentumok felépítését és mutassa be egy példán keresztül, hogy milyen formában tárolhatunk adatot egy XML dokumentumban. A példában alkalmazzon álnévvel rendelkező névteret is.
16. Mutassa be egy példán keresztül, hogy hogyan állíthatunk elő a memóriában egy XML dokumentumot egy gyűjtemény lekérdezésével.
17. Készítsen egy XML dokumentumot, ami attribútumként és beágyazott formában is tartalmaz adatot. Készítsen egy-egy LINQ lekérdezést mindkét adattípushoz.
18. Mi az Entity Framework? Mit nyújt?
19. Ismertesse a Database First és a Model First koncepciókat. Mit jelent a Lazy Loading megoldás? (Entity Framework)

5. Windows Forms

1. Ismertesse a következő Windows Forms komponensek osztályainak neveit: ablak, nyomógomb, főmenü (menücsík), menüpont, listaablak, kombinált listaablak, állapotsor, jelölőnégyzet, választógomb, szövegmező, csoportablak, gyorsmenü (csak a hibátlan nevek érnek pontot).
2. A fájlmegnyitás példáján keresztül mutassa be a beépített Windows Forms párbeszédablakok használatának lépéseit pszeudokód vagy C# kód segítségével.
3. Egy szövegmezőt tartalmazó saját készítésű párbeszédablak példáján keresztül mutassa be a saját készítésű Windows Forms párbeszédablakok használatának lépéseit pszeudokód vagy C# kód segítségével.
4. Mi az ablak Controls gyűjteményének feladata? Egy rövid kódon keresztül mutassa be a használatát.
5. Mit jelent az, hogy egy párbeszédablak modális vagy nem modális (ShowDialog, Show)?
6. Mi a Validating esemény szerepe? Hogyan használhatjuk egy vezérlőhöz (pl. szövegmező) kapcsolódóan? (Windows Forms)

6.Grafikus megjelenítés Windows Forms felületen

1. Mit jelent az, ha az ablak egy része érvénytelenné válik? Mikor fordulhat ez elő?
2. Ismertesse az "Eltűnő kép" problémát és a négy lehetséges megoldást három-három mondatban. (Windows Forms)
3. Ismertesse részletesen a dupla pufferelés technikáját. Milyen előnye lehet a képrerajzolással szemben? (Windows Forms)
4. Hogyan oldaná meg azt a feladatot, hogy a jobb oldali egérgomb lenyomvatartása és az egér mozgatása folyamatos vonal megrajzolását eredményezze?

7.Windows 8 stílusú alkalmazások fejlesztése

1. Ismertesse a következő vezérlők fontosabb jellemzőit: WrapGrid, VariableSizedWrapGrid, RepeatButton, HyperlinkButton.
2. Mutassa be egy példán keresztül az async-await párossal megvalósított aszinkron hívást.
3. Ismertesse a Windows 8 által támogatott szenzorokat.

8. Adatbázis kezelés Windows Forms

1. Ismertesse az adatkötés típusok osztályozását, és ismertesse mindegyik típust egy-egy mondatban. (Windows Forms)
2. Milyen szerepet játszanak a Format és Parse események? (Windows Forms)
3. Hasonlítsa össze a közvetett és a közvetlen adatkötést kiemelve fontosabb tulajdonságaikat. (Windows Forms)

Nyelvi alapok és ismétlés

1.1 Ismertesse az automatikus tulajdonság fogalmát, írjon rá egy példát.

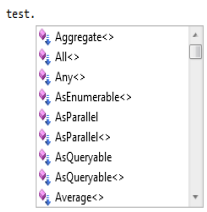
- A C# 3.0 vezette be az automatikus tulajdonságot, mint lehetőséget
- Nem kell létrehoznunk sem az adattagot (mező), sem a hozzá tartozó tulajdonságot, mivel a fordító mindkettőt legenerálja nekünk
- Programkód rövidebb és átláthatóbb lesz
- Példa: class Person { public string Name { get; set; } }
- A fordító automatikusan létrehoz egy private elérésű, string típusú „name” nevű adattagot, és elkészíti hozzá a getter-t/setter-t is
- A hozzáférés módosítható (pl.: private set;)
- Visual Studio-ban prop kulcsszót beírjuk, majd kétszer TAB-ot ütve automatikusan legenerálódik

1.2 Mi az indexelő és milyen korlátozások vonatkoznak rá?

```
class Indexelő
{
    private int[] tömb = new int[100];
    public int this[int index]
    {
        get    { return tömb[index]; }
        set    { tömb[index] = value; }
    }
}
...
Indexelő próba = new Indexelő();
próba[3] = 1;
System.Console.WriteLine(próba[3]);
```

- Az indexelők olyan tulajdonságok, amelyek segítségével egy osztály egyes elemei tömbhöz hasonló indexekkel érhetők el.
- Indexelő deklarálásakor meg kell adni az elemek típusát, majd a **this**-t követi a formális paraméterlista szögletes zárójelben. Ez a paraméterlista határozza meg az indexelés szintaxisát, benne nem szerepelhet **ref** ill. **out** paraméter.
- Az indexelők mindig példány szintűek
- Az indexelők neve mindig this
- Egy osztályhoz eltérő paraméterlistával több indexelőt is megadhatunk
- Minden más szempontból megegyeznek a tulajdonságokkal

1.3 Mi a bővítő metódus (ismertetés)?



- Egy nem általunk írt osztályt ki tudunk egészíteni függvényekkel
- Lambda kifejezéseknél és LINQ-nál sokszor használjuk
- Vannak beépített bővítő függvények, de mi is létre tudunk hozni sajátot
- Intellisense-ben rózsaszín doboz + nyíl

1.4 Írjon egy példát bővítő metódusra.

int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

double avg = numbers.Average(); *//Average egy bővítő metódus*

1.5 Mit jelent az, hogy egy metódus statikus? (itt legalább két fontos tulajdonságot kell megadni)

- Statikus metódust a hagyományos metódusokhoz hasonlóan készítünk, mindössze a static kulcsszóra van szükségünk
- A statikus konstruktortól eltérően rá nem vonatkozik, hogy nem lehetnek paraméterei
- Statikus metódusok nem férnek hozzá az osztály „normális” tagjaihoz, legalábbis direkt módon nem (az minden további nélkül működik, ha egy példány referenciáját adjuk át neki)
- Statikus metódust általában akkor használunk, ha nem egy példány állapotának a megváltoztatása a cél, hanem egy osztályhoz kapcsolódó művelet elvégzése
- Ilyen metódus például az Int32 osztályhoz tartozó Parse statikus metódus is
- A „leghíresebb” statikus metódus a Main

1.6 Ismertesse a virtuális metódus fogalmát és jellemzőit.

- Az ősosztályban deklarált virtuális (vagy polimorfikus) metódusok viselkedését a leszármazottak átdefiniálhatják

- Virtuális metódust a szignatúra elé írt virtual kulcsszó segítségével deklarálhatunk
- A leszármazott osztályokban az override kulcsszóval mondjuk meg a fordítónak, hogy **szándékosan** hoztunk létre az ősosztályéval azonos szignatúrájú metódust, és a leszármazott osztályon ezt kívánjuk használni mostantól

```
class Animal
{
    public virtual void Eat()
    {
        Console.WriteLine("Egy állat eszik...");
    }
}

class Dog : Animal
{
    public override void Eat()
    {
        Console.WriteLine("Kutya csontot rág...");
    }
}
```

  - Egy override-dal jelölt metódus automatikusan virtuális is lesz, így az ő leszármazottai is átdefiniálhatják a működését
  - Az utódosztály metódusának szignatúrája és láthatósága meg kell egyezzen azzal, amit át akarunk definiálni
  - Nem jelölhetünk virtuálisnak statikus, absztrakt és override–dal jelölt tagokat (az utolsó kettő egyébként virtuális is lesz, de ezt nem kell külön jelölni)
- Futási időben dől el, hogy az osztályhierarchián belüli virtuális metóduslánc melyik tagja hajtódik végre
- Virtuális metódus nem lehet private
- Ha definiálunk a leszármazott osztályban ugyanilyen paraméterszignatúrával egy metódust, akkor az:
  - a virtuális metóduslánc folytatása esetén **override kulcsszóval illetjük**
  - a lánctól független metódus esetén **new kulcsszóval illetjük**
  - új virtuális lánc kezdete esetén **new virtual kulcsszóval illetjük**

1.7 Mi a lambda kifejezés (ismertetés)?

- A C# 3.0 vezette be a lambda kifejezéseket
- Egy lambda kifejezés gyakorlatilag megfelel **egy névtelen metódus** „civilizáltabb”, elegánsabb változatának
- **Olvashatóbb** kódot eredményez
- Minden lambda kifejezés tartalmazza az ún. lambda operátort (=>), ennek jelentése nagyjából annyi, hogy **„legyen”**
- Az **operátor bal oldalán a bemenő változók, jobb oldalán a bemenetre alkalmazott kifejezés áll.**
- Mivel névtelen metódus, ezért egy lambda kifejezés **állhat egy delegate értékadásában is** akár(*a következő példában egy delegate fog szerepelni*)

1.8 Írjon egy példát lambda kifejezésre.

PL: (x) => (x * x);

```
public delegate int IntFunc(int x);

static void Main(string[] args)
{
    IntFunc func = (x) => (x * x);

    Console.WriteLine(func(10)); // 100

    Console.ReadKey();
}
```

- *Egy olyan metódusra van tehát szükség, amely egy **int** típusú bemenő paramétert vár, és ugyanilyen típust ad vissza.*
- *A lambda kifejezés bal oldalán a bemenő paraméter (**x**), jobb oldalán pedig a visszaadott értékről gondoskodó kifejezés (**x * x**) áll.*
- *A bemenő paraméternél nem kell (de lehet) explicit módon jeleznünk a típust, azt a fordító magától „kitalálja” (a legtöbb esetre ez igaz, de néha szükség lesz rá, hogy jelöljük a típust).*

1.9 Mi a metódusreferencia (ismertetés)?

- Ha metódusreferenciáról beszélünk, egy metódusra mutató **pointer**-re kell gondolnunk!
- A metódusreferencia másik (talán ismertebb) nevén a **delegate**.
- A metódusreferencia által hivatkozott metódust dinamikusan tudjuk változtatni.
- Definiálása a **delegate** kulcsszóval történik.
- A delegate olyan típus, amely **egy vagy több metódusra hivatkozik.**

- Minden delegate különálló objektum, amely **egy listát tárol a meghívandó metódusokról** (értelemszerűen ez egyúttal erős referencia is lesz a metódust szolgáltató osztályra).
- Nemcsak példány-, hanem **statikus metódusokra is mutathat**.
- Egy delegate deklarációjánál megadjuk, hogy milyen szignatúrával rendelkező metódusok megfelelőek:
  `delegate int TestDelegate(int x);`
- Delegate **nem deklarálható blokkon belül, csakis osztályon belül tagként, illetve osztályokon kívül** (hasonlóan az enum típusokhoz).
- Ez a delegate olyan metódusra mutathat, amelynek visszatérési értéke **int** típusú és egyetlen **int** paramétere van, pl.:
  `static public int Pow(int x){ return (x * x); }`
- **A használata:**
  `TestDelegate dlgt = Pow;`
  `int result = dlgt(10);` *//a result változó értéke 10x10 lesz, azaz 100*
- A delegate-ekhez **egynél több metódust is hozzáadhatunk** a += és + operátorokkal, **valamint elvehetjük őket** a -= és – operátorokkal
- A delegate hívásakor **a listáján lévő összes metódust meghívja a megadott paraméterre**
- A delegate–ek legnagyobb haszna, hogy **nem kell előre megadott metódusokat használnunk**, ehelyett később tetszés szerint adhatjuk meg az elvégzendő műveletet
- **Két delegate egyenlő, ha** mindkettő értéke null, illetve ha a híváslistájukon ugyanazon objektumok ugyanazon metódusai szerepelnek (vagy ugyanazok a statikus metódusok)

1.10 Ismertesse a közzétevő/feliratkozó eseménykezelési modellt.

Az eseménykezelés általános menete:

1. Szükséges képviselő típusok létrehozása
2. Az osztály közzéteszi az eseményeit
3. Az esemény iránt érdeklődő osztályok saját metódusaik átadásával feliratkoznak az eseményre
   Értelemszerűen csak megfelelő szignatúrájú metódussal
   Feliratkozás a +=, leiratkozás a -= operátorral történik
4. Amikor a közzétevő osztályban kiváltódik az esemény, a képviselők segítségével értesíti a feliratkozott osztályokat

1.11 Hogyan adhatunk át paramétereket egy eseménykezelő metódusnak? (ismertesse úgy az egyparaméteres, mint a kétparaméteres eseménykezelő esetét)

- Metódusreferencián keresztül
- Az eseménykezelő metódusok **formális paraméterlistája alapvetően kétféle lehet**
- Mindkét esetben az első (vagy egyetlen) paraméter típusa object kell legyen, és ebben a paraméterben kell megkapja az eseménykezelő az eseményt előidéző objektum referenciáját
- **Amennyiben nem kell külön paramétereket átadnunk az eseménykezelőnek, akkor az eseménykezelőnk egyparaméteres lesz, és ha át kell adnunk paramétert, akkor kétparaméteres lesz**
- A kétparaméteres esetben a második paraméter az EvenArgs osztály leszármazottja kell legyen

WPF

2.1 Hogyan írják le a felhasználói felületet a WPF alkalmazások, amikor nem C# kódból definiáljuk a felületet? Készítsen egy példát, amelyben van egy három menüpontos főmenü, alatta egy eszköztár három gombbal, és az alatt egy szövegmező.

Ebben az esetben a felületet **XAML kód segítségével kell kialakítani**.

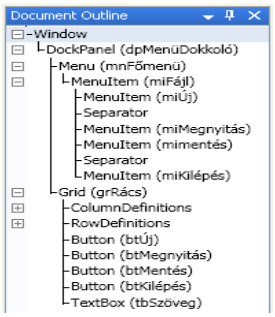
```
<Grid>
<Menu HorizontalAlignment="Stretch" VerticalAlignment="Top" Height="30">
<MenuItem Header="File" FontSize="18"/>
<MenuItem Header="Save" FontSize="18"/>
<MenuItem Header="Open" FontSize="18"/>
</Menu>
<ToolBar HorizontalAlignment="Stretch" Margin="0,30,0,0" VerticalAlignment="Top" Height="30">
<Button Content="Gomb1"></Button>
<Button Content="Gomb2"></Button>
<Button Content="Gomb3"></Button>
</ToolBar>
<Label Content="Szövegmező" Margin="0,60,0,0" VerticalAlignment="Top"
HorizontalAlignment="Left">
</Label>
</Grid>
```

2.2 Fordítást követően hol és milyen formában tárolódik a felület leírása (WPF)?

- Binary Application Markup Language (BAML)-ben tárolódik
- A BAML az XAML-ben leírt felület bináris változata
- Fordításkor keletkezik az XAML-ből
- EXE-be beágyazva
- ~\obj\Debug*.BAML
- Betöltés az InitializeComponent-ben

2.3 Mi a logikai fa (WPF)? Rajzoljon egy példát, és magyarázza el annak segítségével a logikai fa felépítését.

- **Leírja a szülő és a gyerek objektumok közötti összefüggést**
- Rácsban nyomógomb, amiben rács, amiben kép és szöveg
- Fontos a tulajdonságok öröklése szempontjából
- Navigálás: GetParent, GetChildren, FindLogicalNode



2.4 Ismertesse az Application osztály (WPF) öt előadáson tanult tulajdonságát.

System.Windows.Application tulajdonságok:

- **Current** – a futó alkalmazás objektum – pl. alkalmazás szintű változók, leállítás
- **MainWindow** – a fő ablak
- **Windows** – az alkalmazást létrehozó szálból létrehozott ablakok gyűjteménye
- **StartupUri** – az alkalmazás indításakor automatikusan megnyitott ablak/lap
- **Properties** – az alkalmazásból bárhonnan elérhető adatok tárolása-pl. Application.Current.Properties["Név"]=Érték;

2.5 Miért van szükség rétegmenedzserekre WPF-ben? Ismertesse a 6 tanult rétegmenedzsert (feladat+jellemzők).

A rétegmenedzsereket **más néven tárolóknak** nevezzük. A tárolók segítségével **könnyebben alakítható ki a felület, pontosabb, áttekinthetőbb, könnyebb pozícionálást biztosítanak, és a vezérlőket egy egységbe, rétegbe csoportosítják.** A tárolók **egymásba ágyazhatók**, tehát, egy grid tároló tartalmazhat további grid, vagy más tárolókat.

- **Grid** – sorok és oszlopok alakíthatók ki
- **UniformGrid** – minden cella azonos méretű
- **StackPanel** – vízszintesen vagy függőlegesen elhelyezett elemek
- **DockPanel** – dokkolás támogatása
- **Canvas** – koordináta alapú tárolás
- **WrapPanel** – az elemek egyenes vonalban helyezkednek el

2.6 Ismertesse a DependencyProperty fogalmát és jellemzőit (WPF).

A DependencyProperty olyan tulajdonság, mely hasonló a .NET tulajdonságokhoz, de azoknál jóval összetettebb (komplexebb).

Főbb eltérések a tulajdonságokhoz képest:

- A tulajdonság közvetlenül olvassa ki a védett mezőből az adatokat
- Depencency Property dinamikusan gyűjti be az információt a GetValue() metódus meghívásával. Ezt a függvényt a DependencyObject-től örökli

További eltérések:

- Adatmódosításnál (érték beállítása) a DependencyObject-től kapott dictionary-nek (kulcs-érték párok halmaza) adunk egy értékpárt.
- A kulcs lesz az a tulajdonság, amit változtatni akarunk, az érték pedig amire be akarjuk állítani.

Előnyei:

- Kevesebb memóriát fogyasztunk(rengeteg adatot tárolunk a UI vezérlők adattagjaiban)
- Értékek öröklődése

  -Ha lokálisan nincs érték beállítva, akkor a logikai fában felfelé haladva addig megyünk, amíg nem találunk értéket

  -Pl.: FontSize-ot beállítjuk egy gyökér elemben akkor az mindenre ki fog hatni (ha lokálisan nem változtatjuk meg)

- Értesítés a változásokról

  -Beépített változás-értesítés

  -Callback metódus segítségével kapunk értesítést a változásokról

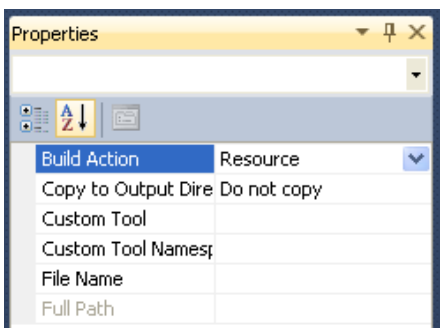
  -Adatkötésre is Dependency property-t használjuk (Binding)

Létrehozása Visual Studioban: propdp <TAB> <TAB> - code snippet

2.7 Ismertesse, hogy hogyan oldható meg az ikonok befordítása a szerelvénybe illetve képek tárolása a szerelvényen kívül. (WPF)

IKONOK:

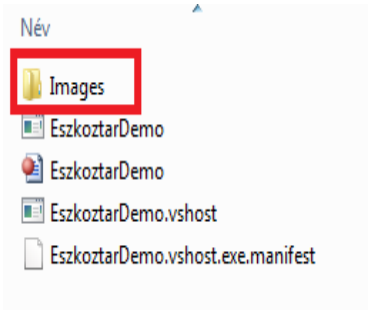
1. Solution Explorerben létre kell hozni egy új mappát az ikonoknak.
2. Az kiválasztott ikonokat be kell másolni a mappába.



3. Ikonokat ki kell jelölni, majd a tulajdonságok (Properties) mezőre kattintva kell beállítani a következő tulajdonságokat:
   Build Action=Resource
   Copy to Output Directory=Do not copy
4. Végül le kell fordítani a projektet. Build/Rebuild Solution

KÉPEK:

Az eljárás hasonló, mint az előbb.



1. Solution Explorerben létre kell hozni egy új mappát az képeknek.
2. Az kiválasztott képeket be kell másolni a mappába.
3. A képeket ki kell jelölni, majd a tulajdonságok (Properties) mezőre kattintva kell beállítani a következő tulajdonságokat:
   Build Action=Content
   Copy to Output Directory=Copy always
4. Végül le kell fordítani a projektet. Build/Rebuild Solution

2.8 Milyen célt szolgálnak a "parancsok" (WPF)? Mi a CommandBinding feladata?

- Ha egy műveletet több módon is előidézhetünk (menü, gyorsbillentyű, eszköztár), kössük őket össze, hogy **egyszerűen megoldható legyen a letiltás/engedélyezés**. Ezt a célt szolgálják a Commands objektumok (amik beépített és programozó által definiált parancs osztályok)
- **A CommandBinding feladata:** ez az osztály kapcsolja össze a parancsot és az eseménykezelőt

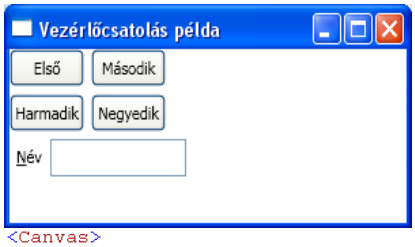
2.9 Ismertesse a három WPF esemény típust.

- **Direkt esemény (Direct Event)** – a WinForm-mal azonos módon működik, az eseményt csak az előidéző objektumban dolgozhatjuk fel
- **Buborék esemény (Bubbling Event)** – először az eseményt kiváltó vezérlő kapja meg, majd az őt tartalmazó rács, majd a nyomógomb, és így tovább felfelé a hierarchiában a gyökér csomópontig
- **Lefutó esemény (Tunneling Event)** – Preview-al kezdődik a neve, a gyökértől kiindulva halad lefele az esemény előidéző vezérlőhöz. Általában az események blokkolására használják

Először a TE (↓)majd a BE (↑) következik be

2.10 Mit jelent a vezérlők csatolása (WPF)?

- Egy vezérlő, Target tulajdonságának segítségével csatolható egy másik vezérlőhöz
- Az aláhúzás jel azt jelenti, hogy az N betű aláhúzva jelenik meg
- A csatolás hatására, ha Alt+N-et nyomunk, akkor a szerkesztőmező kapja meg az input fókuszt




```
<Canvas>
  <Button Name="btElső" Width="55" Height="25" >Első</Button>
  <Button Name="btMásodik" Width="55" Height="25" Canvas.Left="60">Második</Button>
  <Button Name="btHarmadik" Width="55" Height="25"  Canvas.Top="30">Harmadik</Button>
  <Button Name="btNegyedik" Width="55" Height="25" Canvas.Left="60" Canvas.Top="30">Negyedik</Button>
  <Label Canvas.Top="60" Target="{Binding ElementName=tbNév}">_Név</Label>
  <TextBox Name="tbNév" Canvas.Left="30" Canvas.Top="60" Width="100" Height="25"></TextBox>
</Canvas>
```

2.11 Ismertesse a validálási szabályokat. (WPF)

```
<Grid x:Name="grRács" DataContext="{Binding}">
  <TextBox x:Name="tbÁtlag" Grid.Row="1" Grid.Column="1"
           Height="30" Margin="0,5,10,5"
           VerticalContentAlignment="Center">
    <TextBox.Text>
      <Binding  Path="Átlag" UpdateSourceTrigger="PropertyChanged" >
        <Binding.ValidationRules>
          <local:vrÁtlag />
        </Binding.ValidationRules>
      </Binding>
    </TextBox.Text>
  </TextBox>
```

A WPF validációs mechanizmusa úgynevezett ValidationRule-ok segítségével történik, ami az adatkötési fázisban ellenőrzi a bemenő információt, egy adott szabály függvényében. A szabályt definiálni kell, majd egy vezérlőhöz kell kötni. Ha az input érvényes, az adatkötés megtörténhet, egyébként pedig meg kell akadályozni. Továbbá a felhasználónak visszajelzést kell biztosítani arról, hogy tudja érvénytelen adatot próbált meg betáplálni, újra kell kezdenie a kitöltést.

Grafikus megjelenítés WPF felületen

3.1 Mutassa be a 2D rajzolás lehetőségeit (szintjeit) WPF-ben. Mindegyiknél nevezzen meg legalább egy előnyös és egy hátrányos tulajdonságot.

Három darab 2D rajzolási lehetőség közül választhatunk: *(Előny ; Hátrány)*

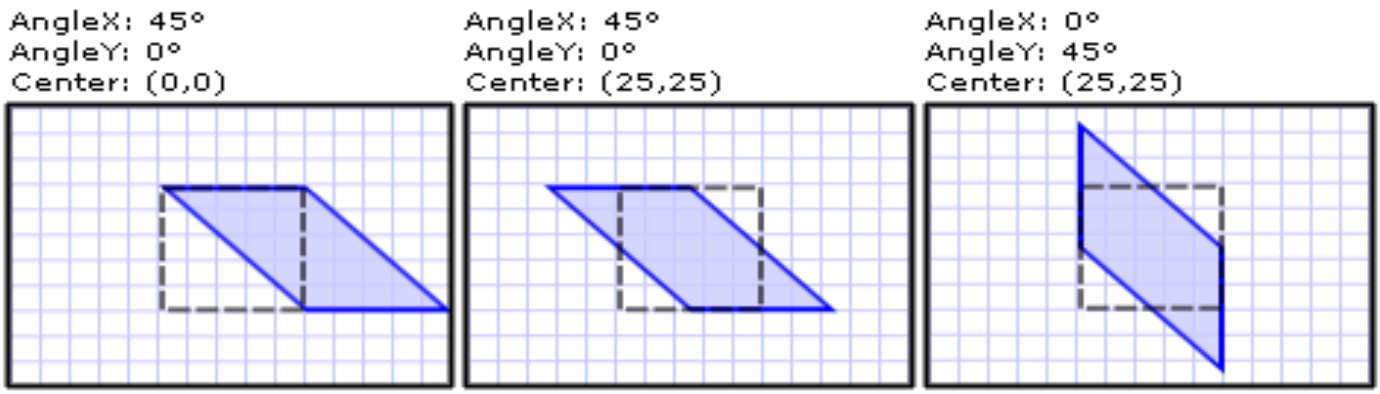
- **Shapes:** ha felhasználói interakció (kölcsönös viszony) szükséges. Pl.: Egér input, vagy ToolTip, stb..
  - (magas szintű, rengeteg metódus, eseménykezelés **;** nagy memóriaigény, lassúság)
- **Drawings & Geometries:** ha felhasználói interakció nem szükséges (bár megoldható), vagy ha komplex, vektor alapú grafikus adatmegjelenítés szükséges
  - (gyorsabb, kisebb erőforrásigény **;** nincs beépített input kezelés, több kód szüksgées, felh. interkació csak jelentős kódolás áran)
- **Visuals:** ha nagy mennyiségű adatot gyorsan akarunk megjeleníteni
  - (leggyorsabb, legkisebb energiaigény **;** alacsony szintű, sok kódolás)

3.2 Mutassa be egy-egy mondatban a transzformációhoz kapcsolódó 6 osztályt (WPF) és ábrák segítségével mutassa be a SkewTransform paramétereinek értelmezését.

- **TranslateTransform**: Eltolja az objektumot, az eredeti helyzethez képest X és Y tengelyeken (külön is).
- **RotateTransform**: Elforgatja az objektumot a megadott szöggel. (Angle)
- **ScaleTransform**: Nyújtja az objektumot. X, Y nyújtás. Akkorára nyújtja, ahány szorosát megadjuk.
- **SkewTransform**: Dönti az objektunot az X, Y tengelyeken megadott szöggel.
- **MatrixTransform**: Olyan egyéni tranzformáció készíthető a segítségével, mely nem hozható létre más transzformációs osztállyal.
- **TransformGroup**: Több transzformációt lehet vele egy (csoport) transzformációként kezelni.

SkewTransform:

- **Angle X**: X tengelybeli döntés szöge.
- **Angle Y**: Y tengelybeli döntés szöge.
- **Center(0,0):** Meghatározza, hogy az objektum mely pontjához viszonyítva történjen a döntés szögének a kiszámítása.



3.3 Ismertesse a három animáció típust egy-egy mondatban + nevezzen meg mindegyikhez legalább egy tipikus osztályt (WPF).

- **Lineáris (egyenletes) változtatás:**
  - Két érték között növeli vagy csökkenti az értéket, megadott időintervallum alatt.
  - *AdattípusAnimation*
  - DoubleAnimation

- **Útvonal alapú:**
  - *AdattípusAnimationUsingPath*
  -DoubleAnimationUsingPath)
  - Egy előredefiniált útvonal mentén történő például helyváltoztatást lehet megvalósítani vele, megadott időintervallumon belül.
- **Előredefiniált értéksor váltogatása:**
  - *AdattípusAnimationUsingKeyFrames*
  - DiscreteStringKeyFrame
  - Különböző értékeket animálhatunk a segítségével. Pl.: Karakterek, vagy stringek, stb értékeit váltogathatjuk, animációként.

3.4 Ismertesse részletesen a lineáris animációt (WPF).

- A lineáris animációval létrehozható például méretváltoztatás. A méretváltozásnak megadhatjuk az idejét egy TimeSpan segítségével.
- Ha WPF kódban létrehozunk egy négyzetet, és elnevezzük ”négyzet” névvel, a következő kóddal tudjuk animálni a szélességét, magasságát 100 pontról 300 pontra 5 másodperc alatt:

```
TimeSpan time = TimeSpan.FromSeconds(5);
DoubleAnimation anim = new DoubleAnimation();
anim.From = 100;
anim.To = 300;
anim.Duration = new Duration(time);
négyzet.BeginAnimation(WidthProperty,anim);
négyzet.BeginAnimation(HeightProperty, anim);
```

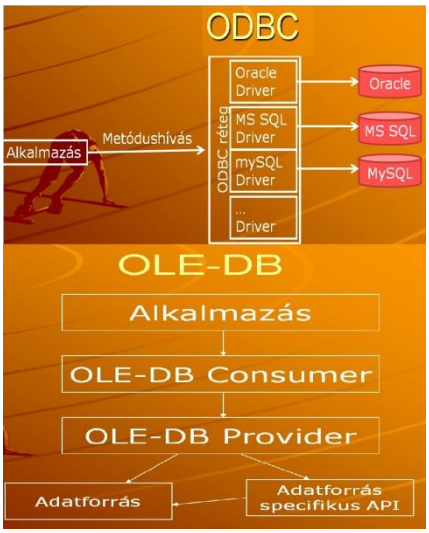
3.5 Ismertesse részletesen az útvonal alapú animációt (WPF).

Az útvonal alapú animációval előre elkészített útvonal mentén mozgathatunk objektumokat.

1. Felület kialakítása: A mozgatni kívánt objektum megrajzolása.
2. Az útvonal megrajzolása olyan módon, hogy az útvonalat alkotó pontok lekérdezhetők legyenek. (pl.: az órán ismertetett archimédeszi spirál)
3. Útvonal geometria objektum definiálása: PathGeometry pgÚtvonal = new PathGeometry();
4. Görbe megrajzolása.
5. Animáció létrehozása (mindkét tengelyre meg kell írni külön, mert nem ugyanazok az értékek kellenek a két tengelyre): DoubleAnimationUsingPath daxAnimáció = new DoubleAnimationUsingPath();
6. Útvonal megadása: daxAnimáció.PathGeometry = pgÚtvonal;
7. Animáció tulajdonságainak beállítása: idő, reverse, koordináta, stb..
8. Másik tengelyre is el kell készíteni az animációt, a fent leírtak alapján.
9. Animáció indítása mindkét tengelyen külön:
   rcNégyzet.BeginAnimation(Canvas.TopProperty,dayAnimáció);
   rcNégyzet.BeginAnimation(Canvas.LeftProperty, daxAnimáció);

Adatbázisok elérése

4.1 Ismertesse az ODBC-n és az OLE DB-n keresztüli adatbázis elérési módokat.



- **Közvetlen elérés** - minden adatbázismotorhoz külön függvénygyűjtemény
- **Absztrakciós rétegen keresztül**
    - **-Open DataBase Connectivity** (ODBC), közös függvényhalmaz, amivel minden DB elérhető. Az ODBC rétegben levő driverek lefordítják.
    - **-Object Linking and Embedding DataBase** (OLE DB) –kifele táblázatos formában mutatja az adatokat. Az adatforrások OLE DB provider-eken keresztül érhetők el COM objektumok segítségével. ODBC-t is elér

4.2 Ismertesse a háromrétegű alkalmazás fogalmát és az egyes rétegek feladatait.

A többrétegű alkalmazás több, egymással együttműködő logikai egységből áll, amelyek szétszórva helyezkednek el a hálózat számítógépein. Ily módon a kliensalkalmazásunk egész vékony lehet, az üzleti logikát pedig központosíthatjuk. A többrétegű alkalmazások 'legegyszerűbb' formája a háromrétegű alkalmazás. A megjelenítési réteg tartalmazza a felhasználói interfészt, az adatszolgáltatási réteg működteti az adatbázis funkciókat és az üzleti szabályok (business rules) rétege jelenti a középső réteget, az itt megvalósított funkciók miatt akarunk tulajdonképpen programot írni. A következő réteg, vagyis az alkalmazás szerver menedzseli az ügyfél és az adatbázis szerver közti adatforgalmat, ezért data broker-nek is nevezik.

- **Presentation tier:** Megjelenítési réteg: windows forms, wpf, web böngésző (web forms)
- **Business tier (logic):** Üzleti logikai réteg: adatokat feldolgozó szerver, adat objektumok, web szolgáltatások, net remoting.
- **Data tier:** Adattároló réteg: DBMS, egyéb erőforrások

4.3 Hasonlítsa össze a két tanult adatelérési modellt (direct access, disconnected access).

- **Kapcsolat alapú (Connected Access)**: pl. DataCommand, DataReader – ha az adatokat azonnal feldolgozzuk. Az adatbázis nem lokális, nincs mentve a gépünkre, ilyenkor kapcsolatot kell teremteni a tároló szerverrel, és onnan végezzük az adatok betöltését.
- **Kapcsolat nélküli (Disconnected Access)**: DataSet – lokális másolat az adatokról. A számítógépünkre másolt adatokal dolgozunk, például egy acces (*.mdb) adatbázisból nyerjük az adatokat.

4.4 Ismertesse a "Provider" objektumokat és fontosabb jellemzőiket.

A Provider objektumok segítségével tudjuk írni és olvasni az adatforrást.

- Connection
- Command
- DataReader
- DataAdapter
- TableAdapter

Connection objektum:

- A DP egyik komponense
- Kapcsolat az adattárhoz, ezen keresztül kommunikál az alkalmazás az adatbázissal

Command objektum:

- Közvetlen hozzáférés a kapcsolt adatbázis adataihoz
- SQL parancsok vagy tárolt eljárások
- Az eredmény adatfolyam, amit DataReader olvashat vagy DataSet-be lehet betölteni
- Parameters tulajdonság: gyűjtemény, az SQL parancsok vagy tárolt eljárások bemenő és kimenő paraméterei
- Command típusok (osztályok)
  -System.Data.SqlClient.SqlCommand
  -System.Data.OleDb.OleDbCommand

DataReader objektum:

- Gyors, csak előrehaladást engedélyező szerver oldali kurzorhoz hozzáférés
- Rekordokat tartalmazó adatfolyamon halad végig
- A Command objektum ExecuteReader metódusa egy DataReader-t ad vissza
- Az aktuális rekord egyes oszlopaiban tárolt adatokat típusuk szerinti metódusokkal lehet lekérdezni (pl. GetDouble)

DataAdapter objektum:

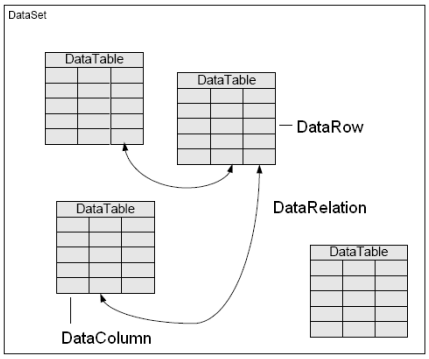
- Híd a tábla és az adatforrás között
- Adatbázis parancsok
- Adatbázis kapcsolatok
- Alap típusok
  -OleDbDataAdapter
  -SqlDataAdapter – SQL Serverhez

TableAdapter objektum:

- Egy generált osztály típusos DataSet kezeléséhez
- Magába foglalja az alábbi objektumokat:
  -DataAdapter
  -Connection
  -Commands
  -Query
  Parameters
- Minden táblához külön TableAdapter

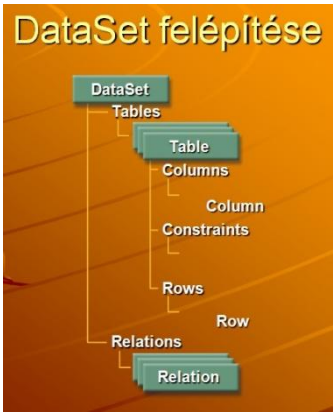
4.5 Ismertesse a "Consumer" objektumokat és fontosabb jellemzőiket kiemelve a típusos objektumok előnyeit.

- DataSet/típusos DataSet
- DataTable/típusos DataTable
- DataRow
- DataColumn
- Relation



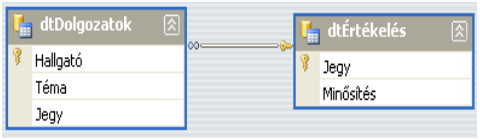
DataSet objektum:

- Kapcsolat nélküli cache-ben tárol adatokat
- Az adatforrás adatainak lokális másolata
- Struktúrája egy relációs adatbázishoz hasonló
- Táblák, kapcsolatok
- A megkötések elsődleges és idegen kulcsokat igényelnek



Típusos DataSet objektum:

- További absztrakciós szint
- Az alap DataSet osztály leszármazottja
- Típusellenőrzést tesz lehetővé fordítási időben
- Gyorsabb hozzáférést biztosít az adathalmazban levő táblákhoz és rekordokhoz
- XML Schema (.xsd) leírás állományokból van generálva az XSD.exe segítségével
- Grafikusan (.xss, .xcs)
- Hozzáférés táblákhoz és oszlopokhoz
  -Típus nélküli esetben: dsNév.Tables["TáblaNév"]
  -Típusos esetben: dsNév.TáblaNév ,
  dsAdatok.dtDolgozatok.HallgatóColumn



4.6 Ismertesse kapcsolat alapú (direct access) adatbáziselérés esetén a lekérdezés, beszúrás, módosítás és törlés lépéseit.

Az eredmény egy szerver oldali kurzorba kerül, abból olvashatunk soronként DataReader segítségével

[Lekérdezés]

Lépések:

- Kapcsolati sztring összeállítása
- Kapcsolat létesítése az adatbázishoz Connection objektum segítségével
- SQL lekérdezés sztring összeállítása
- Kapcsolat megnyitása
- Command objektum létrehozása
- DataReader objektum létrehozása
- Rekordok kiolvasása
- DataReader objektum lezárása
- Kapcsolat lezárása

[Beszúrás, Módosítás, Törlés]

Lépések:

- Kapcsolat objektum létrehozása
- Kapcsolati sztring összeállítása
- Kapcsolat létesítése az adatbázishoz Connection objektum segítségével
- SQL utasítás sztring összeállítása
- Command objektum létrehozása
- Kapcsolat megnyitása
- SQL parancs végrehajtása ExecuteNonQuery
- Kapcsolat lezárása

4.7 Ismertesse kapcsolat nélküli adatbáziselérés esetén a lekérdezés lépéseit, a táblák közötti kapcsolat létrehozását, valamint a módosítások végrehajtását és érvényesítését.

Lekérdezés:

- Kapcsolat létesítése az adatbázissal egy Connection objektum segítségével;
- Command objektum létrehozása és a Connection objektumhoz kapcsolása;
- SQL parancsok összeállítása;
- DataAdapter objektum(ok) létrehozása;
- DataSet objektum(ok) létrehozása;
- Adatok bemásolása a DataSet-be (DataTable) a Fill() metódus meghívásával;

Táblák közötti kapcsolat létrehozása:

- Az ADO.NET nem állítja elő automatikusan az adatbázis táblái közötti kapcsolatokat a DataSet táblái között;
- A kapcsolat vizuális eszközökkel Visual Studio 2005-ben is beállítható;

  Megoldás programból:

```
DataColumn dcSzülő = dsAdat.Tables["ElsőT"].Columns["ElsődlegesKulcs"];
DataColumn dcGyerek = dsAdat.Tables["MásodikT"].Columns["IdegenKulcs"];
DataRelation drKapcsolat = new DataRelation("Kapcs1", dcSzülő, dcGyerek);
dsAdat.Relations.Add(drKapcsolat);
```

Tárolt adatok módosítása:
Minden cella közvetlenül írható;

Sor törlése:

- Meghívjuk DataRow objektum a Delete() vagy Remove() metódusát;
- A Remove() meghívja az AcceptChanges()-t is;

Beszúrás:

- Új sorobjektum előállítása;
- Sorobjektum hozzáadása a tábla Rows gyűjteményéhez;
- A DS tárolja az eredeti és a módosított adatokat;
- Elfogadás (AcceptChanges());
- Visszavonás (RejectChanges());

Változások érvényesítése az adatbázisban:

- A DataAdapter objektum Update() metódusának meghívása;
- A DataAdapter elküldi a megfelelő INSERT, UPDATE, DELETE SQL utasításokat;
- Az SQL utasításokat (a vizuális fejlesztés során automatikusan generálja a fejlesztőrendszer (nem minden adatbázis esetén támogatott, de pl. MS SQL-nél igen));
- CommandBuilder objektummal állítjuk elő, de csak egytáblás adatbázisnál működik, és csak akkor, ha van elsődleges kulcs, a programozó írja meg;

4.8 Hasonlítsa össze a típusos és típus nélküli DataSet-eket. Melyik alkalmazása előnyösebb és miért?

DataSet objektum:

- Kapcsolat nélküli cache-ben tárol adatokat;
- Az adatforrás adatainak lokális másolata;
- Struktúrája egy relációs adatbázishoz hasonló;

- Táblák, kapcsolatok;
- A megkötések elsődleges és idegen kulcsokat igényelnek.

Típusos DataSet:

- További absztrakciós szint;
- Az alap DataSet osztály leszármazottja;
- Típusellenőrzést tesz lehetővé fordítási időben;
- Gyorsabb hozzáférést biztosít az adathalmazban levő táblákhoz és rekordokhoz; XML Schema (.xsd) leírás állományokból van generálva az XSD.exe segítségével;
- Grafikusan (.xss, .xcs);
- Hozzáférés táblákhoz és oszlopokhoz (Típus nélküli esetben: dsNév.Tables("TáblaNév");
- Típusos esetben: dsNév.TáblaNév , dsAdatok.dtDolgozatok.HallgatóColumn).
- A típusos datasetek használata az előnyösebb, mert így egyfajta adatellenőrzést hajtunk végre.

4.9 Milyen interfészt kell implementáljon egy gyűjtemény ahhoz, hogy LINQ lekérdezést hajthassunk végre rajta? Ha ezt az interfészt nem implementálja egy gyűjtemény, akkor milyen eszközzel és hogyan készíthetünk belőle egy olyan gyűjteményt, ami már lekérdezhető LINQ segítségével? Írjon egy egyszerű példát erre.

- Az adatokat tároló objektumnak meg kell valósítania a generikus IEnumerable<T> interfészt.
- A System.Collections névtér gyűjteményei nem implementálják.
- Ebben az esetben saját magunknak kell átalakítani a következő módon:

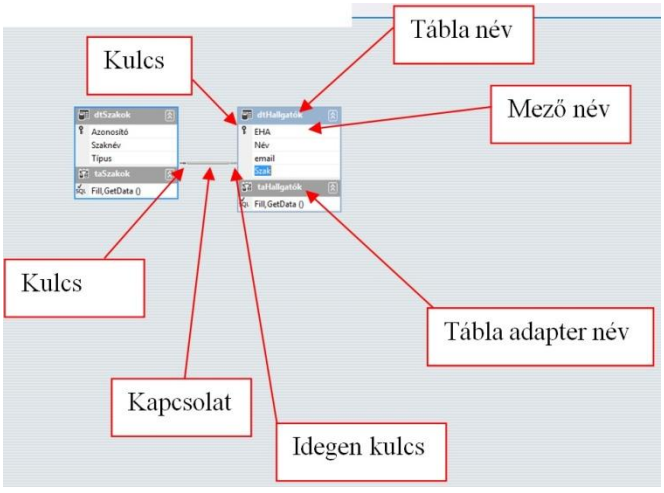
```
ArrayList A = new ArrayList() {2,4,5,1,8,5,7};
IEnumerable<int> I = A.OfType<int>();
IEnumerable<int> E = from x in I
 where x % 2 == 0
 orderby x descending
 select x;
```

4.10 Tervezzen meg egy két, egymással kapcsolatban álló táblából álló típusos DataSet-et a DataSet Designer által alkalmazott grafikus jelölésmódot használva. Az ábrán szerepelnie kell a DataSet névnek, a táblaneveknek, a táblaadapter neveknek, a mezőneveknek, kulcsoknak és az idegen kulcsnak. Lássa el szöveges magyarázattal az ábrát. Írjon egy egytáblás lekérdezést, amiben szerepel a where, és az eredményrekordok tartalmaznak projekciót (select new {...}), és magyarázza azt el. Írjon egy kéttáblás (a táblák összekapcsolását tartalmazó) lekérdezést, amiben van csoportosítás (group-by-into), és magyarázza azt el.

Egytáblás:

```
var er = from x in dsHallgatók.dtHallgatók
where x.Szak==1212
select new
{ x.EHA,
x.Név,
x.email,
x.Szak };
```

Megadjuk, melyik táblából kérdezünk le, az x változóba. Egy var típusú változóban kerül tárolásra az eredmény. Feltétel. Csak olyan sorokat jelenítsen meg, melyek 1212 szak azonosítóval rendelkeznek.



Kiválasztjuk, mely mezők szerepeljenek a lekérdezésben.

Kéttáblás:

Az alábbi lekérdezés azoknak a hallgatóknak adja meg a szakneveit egyszer (ismétlés nélkül), melyek azonosítója 1000-nél nagyobb.

```
var er = from x in dsHallgatók.dtHallgatók
join y in dsHallgatók.dtSzakok on x.Szak equals y.Azonosító
where x.Szak>1000
group y by y.Szaknév into g
select g;
```

x változóba kerül a dsHallgatók.dtHallgatók. y változóba kerül dsHallgatók.dtSzakok tábla és össze lesz kapcsolva az előző táblával a következő feltétel szerint: x.Szak mező egyenlő az y.Azonosító mező értékével. Feltétel. Csoportosítás, és a group értéke a g változóban tárolódik. g változó értéke kerül kiválasztásra.

4.11 Milyen objektumra hívhatóak meg a Distinct<>(), Count<>(), Sum<>(), Min<>() és Max<>() metódusok, és milyen feladatot látnak el?

Olyan osztályok objektumaira hívhatjuk meg a kérdésben szereplő metódusokat, melyek megvalósítják az IEnumerable interfészt. *(Ez nem biztos, hogy jó válasz.)*

- Distinct<>(): A megadott mező elemeit adja vissza ismétlések nélkül.
- Count<>(): Megszámolja a mező elemeit (, hogy hány darab érték szerepel a mezőben).
- Sum<>(): A megadott mező értékeinek összegét adja vissza.
- Min<>(): A megadott mező értékei közül a legkisebbet adja vissza.
- Max<>(): A megadott mező értékei közül a legnagyobbat adja vissza.

4.12 Írjon egy-egy példát lambda kifejezésre FindAll és Any metódusokhoz.

- Any(s => s == 'z');
  ```
  bool seged = textBox.Text.ToCharArray().Any(s => s == 'z');
  ```
- FindAll(s=>s==1);
  ```
  List<int> szamok = new List<int>();
  szamok.Add(0);
  szamok.Add(1);
  szamok.Add(1);
  List<int> eredmeny = new List<int>();
  eredmeny = (szamok.FindAll(s=>s==1));
  ```

4.13 Milyen objektumok tartják nyilván WPF alkalmazásokban azt, hogy az adatforrás mely rekordja/objektuma az aktuális?

A CurrencyManager. A CM-el lekérdezhetjük/beállíthatjuk, hogy melyik az aktuális sor.

4.14 Ismertesse egy-egy mondatban a három WPF adatkötési módot (OneTime, OneWay,TwoWay).

- **OneTime:** legegyszerűbb, nincs változáskövetés, egyszer történik adatszinkronizáció. (Forrás->Cél, egyszeri)
- **OneWay:** felület felé irányú változáskövetés megoldott, ha módosul az adatforrás, nem kell semmit kódolni (ez az alapértelmezett viselkedés). Egyirányú, alapértelmezett. Ha a mód nincs megadva ez lesz használva. Ilyenkor a kötött vezérlő tulajdonságának változásakor a célvezérlő tulajdonsága is változni fog. (Forrás->Cél)
- **TwoWay:** másik irányú változáskövetés is megoldott. Kétirányú. Oda-vissza kötés. (Forrás<->Cél)

4.15 Ismertesse az XML dokumentumok felépítését és mutassa be egy példán keresztül, hogy milyen formában tárolhatunk adatot egy XML dokumentumban. A példában alkalmazzon álnévvel rendelkező névteret is.

Az XML leírja az entitásokat és a köztük levő kapcsolatokat.

XML dokumentumok felépítése:

- Bevezető rész

-Kötelező: XML változat, nyelvkészlet
<?xml version "1.0" encoding="UTF-8"?>
<?xml version "1.0" encoding="ISO-8859-2"?>
-Feldolgozónak szóló direktíva
<?xml-stylesheet type="text/xsl" href="…"?>

- Struktúraleírás

-Dokumentum elem struktúrájának, sémájának leírása
-Dokumentum elem leírás: szerkezet+tartalom

- Záró rész

-Feldolgozónak szóló direktíva
-Megjegyzés
<!-- megjegyzés -->

Példa: Az Álnévvel rendelkező rész félkövér betűvel van szedve.

```
<?XML version="1.0" encoding="UTF-8" ?>

<adatbazis xmlns="http://valami.valahol.org"
xmlns:pz="http://penzugy">

  <cim>A programozás örömei</cim>

  <author>Lélek Búvár</author>

  <pz:ar>1500</pz:ar>

 </konyv >

 <konyv azn="531">

  <cim>Szoftverprojektek</cim>

  <author>Fejlesztő Ernő</author>

  <pz:ar>1500</pz:ar>

 </konyv>

</adatbazis>
```

4.16 Mutassa be egy példán keresztül, hogy hogyan állíthatunk elő a memóriában egy XML dokumentumot egy gyűjtemény lekérdezésével.

```
static List<Hallgató> Hallgatók = new List<Hallgató>();
static void Feltölt()
{
  Hallgatók.Add(new Hallgató("XX.KEFO", "A B", "x@net.hu", 2.25));
  Hallgatók.Add(new Hallgató("XY.KEFO", "C B", "y@net.hu", 4.25));
  Hallgatók.Add(new Hallgató("ZX.KEFO", "D B", "z@net.hu", 3.45));
  Hallgatók.Add(new Hallgató("ZY.KEFO", "E B", "t@net.hu", 3.55));
}

private static void Előállít_lekérdezéssel()
{
  Feltölt();
  xdAdatok = new XDocument(
    new XDeclaration("1.0", "utf-8", "yes"),
    new XComment("Hallgatói adatok"));
  var xml = from x in Hallgatók
            select new XElement("Hallgató",
              new XAttribute("Átlag", x.Átlag),
              new XElement("EHA", x.EHA),
              new XElement("Név", x.Név),
              new XElement("e-mail", x.email));
  xdAdatok.Add(new XElement("Hallgatói_Adatok",xml));
}
// XML dokumentum előállítása lekérdezéssel
Előállít_lekérdezéssel();
Console.WriteLine(xdAdatok);
xdAdatok.Save("Hallgatoi_Adatok.xml");
Console.ReadLine();
```

4.17 Készítsen egy XML dokumentumot, ami attribútumként és beágyazott formában is tartalmaz adatot. Készítsen egy-egy LINQ lekérdezést mindkét adattípushoz.

```
// Az összes e-mail cím.
var email = from x in xdAdatok.Descendants("e-mail") select x.Value;
Console.WriteLine("E-mail címek");
foreach (var e in email)
  Console.WriteLine(e);
Console.ReadLine();

// Lekérdezés attribútum alapján.
// A 3,5-nél nagyobb átlaggal rendelkező hallgatók neve és EHA kódja.
var Jók = from x in xdAdatok.Descendants("Hallgató")
          where (double)x.Attribute("Átlag") > 3.5
          select new
          {
            Név = (string)x.Element("Név").Value,
            EHA = (string)x.Element("EHA").Value
          };
Console.WriteLine("3,5-nél nagyobb átlaggal rendelkező hallgatók "+
  "neve és EHA kódja");
foreach (var j in Jók)
  Console.WriteLine(j.Név + "\t" + j.EHA);
Console.ReadLine();
```

4.18 Mi az Entity Framework? Mit nyújt?

Az Entity Framework egy olyan adatelérési keretrendszer (angolul ORM (Object-Relational Mapper) API), amely segít áthidalni a különbségeket az alkalmazás adatstruktúrája és üzleti objektumai között. Tehát, olyan keretrendszer, mellyel a relációs adatbázist objektum orientáltan tudjuk kezelni. A Linq to SQL testvére, működésük hasonló. MS fejleszti, alapból MS SQL adatbázisokkal működik csak (pl. Access-el se!). Nagyméretű projekteknél igen hasznos.

- **Lekérdező nyelv a LINQ**

  -Ha lassú lenne a lekérdezés készíthetünk magunk beágyazott SQL-t

- **Három lehetőségünk van:**

  -Database first

  A relációs adatbázis már létezik, ebből hozunk létre entitás osztályokat

  -Model first

  Előbb létrehozzuk az entitás modellt, majd ebből generálunk adatbázist

  -Code first

  C# osztályokat hozunk létre, ebből generáljuk a modellt

4.19 Ismertesse a Database First és a Model First koncepciókat. Mit jelent a Lazy Loading megoldás? (Entity Framework)

Database First:

- Az entitás modellt az adatbázis alapján generáljuk.
- A relációs adatbázis már létezik, ebből hozunk létre entitás osztályokat

Modell First:

- Az adatbázist az entitás modellből generáljuk
- Adatelérési réteg
- Külön projekt, melynek típusa osztálykönyvtár
- Az automatikusan keletkező Class1.cs-t töröljük belőle
- Előbb létrehozzuk az entitás modellt, majd ebből generálunk adatbázist

Lazy Loading:

- Késleltetett betöltést jelent.
- Akkor használjuk, ha a lekérdezés definiálásakor az nem fut le ténylegesen, vagy a lekérdezés csak akkor kerül futtatásra, amikor az eredményre szükség van.

Windows Forms

5.1 Ismertesse a következő Windows Forms komponensek osztályinak neveit: ablak, nyomógomb, főmenü (menücsík), menüpont, listaablak, kombinált listaablak, állapotsor, jelölőnégyzet, választógomb, szövegmező, csoportablak, gyorsmenü (csak a hibátlan nevek érnek pontot)!

Ablak:	Form
Nyomógomb:	Button
Főmenű (menücsík):	MenuStrip
Menüpont:	MenuItem
Listaablak:	ListBox
Kombinált listaablak:	ComboBox
Állapotsor:	StatusStrip
Jelölőnégyzet:	CheckBox
Választógomb:	RadioButton
Szövegmező:	Label
Csoportablak:	GroupBox
Gyorsmenü:	ContextMenuStrip

5.2 A fájlmegnyitás példáján keresztül mutassa be a beépített Windows Forms párbeszédablakok használatának lépéseit pszeudokód vagy C# kód segítségével!

1. Első lépésben példányosítani kell az OpenFileDialog osztályt, egy tetszőleges névvel (pl.: dlg (lsd. lentebb)).
2. A következő lépésben a dlg példányra állíthatunk be szűrőt, mely csak a megadott kiterjesztésű fájlokat jeleníti meg a fájl megnyitás ablakban. Pl: *.mp3, stb.
3. Megadhatjuk, hogy egy, vagy több fájlt nyithassunk meg egyszerre. „dlg.Multiselect = true;”
4. Valamilyen eseményhez kell társítani az ablak megjelenítését, például egy nyomógomb klikk eseményének hatására nyíljon meg az ablak. Az eddigi kódunkat beillesztjük egy nyomógomb click eseményére feliratkozó metódusába, majd a kódot kiegészítjük a következő sorral: példánynév.ShowDialog(); melynek hatására megjelenik a fájl megnyitása ablak. Mivel a ShowDialog(); parancsal nyithatjuk csak meg, így addig, míg megnyitott állapotban van az ablak, nem vezérelhetjük azt az ablakot, melyből megnyitottuk a fájl megnyitása ablakot. Tehát a főprogramba való visszatérés érdekében be kell zárnunk a fájl megnyitást. ShowDialog(); modális megnyitási mód. A Show(); nem modális megnyitási mód, a főablak és az abból Show() metódussal nyitott ablak egyaránt vezérelhető (kivéve: MessageBox.Show();). OpenFileDialog nem nyitható meg Show() metódussal, csak a ShowDialog() alkalmazható.
5. Az ablak visszatérési értékét (mely DialogResult típusú (csak ShowDialog() esetben)) vizsgálhatjuk, és egy if elágazás segítségével különböző parancsokat hajthatunk végre a visszatérési értéktől függően (miután bezárult a fájl megnyitása ablak). Az alábbi példa szemlélteti a leírtakat, majd ugyan ez megtalálható az alábbi kód alatt kommentelve.

Példa fájlmegnyitás ablak létrehozására:

```
string [] list; // String tömb létrehozása, amibe majd kerülnek a fájlok elérési útjai, amiket majd megnyitunk.
OpenFileDialog dlg = new OpenFileDialog(); // A feájl megnyitása ablak (objektumosítása) példányosítása.
dlg.Filter = "Audio Files(*.mp3, *.wav)|*.mp3;*.wav"; //Szűrő: Ha csak Audio fájlokat szeretnénk
megnyitni, így lehet a kiterjesztéseket szűrni. Eredménye: Csak az mp3 és a wav kiterjesztésű fájlok
```

```
jelennek meg a megnyitás ablakban (majd).
dlg.Multiselect = true; // Engedélyezzük, hogy a fájl megnyitása ablakkal egyszerre több fájlt is megnyithassunk.
if (dlg.ShowDialog() == DialogResult.OK) { // Itt az if feltételében megadott "dlg.ShowDialog()" meg fogja jeleníteni az ablakot a felhasználó számára melynek visszatérési értékére történik meg a vizsgálat. Ha az "== DialogResult.OK", akkor lefut az igaz ág. Tehát, ha valamit megnyit a felhasználó (ténylegesen, tehát nem a mégse gombra kattint megnyitáskor) akkor fog csak lefutni az igaz ág.
list = dlg.FileNames; // Miután megnyílt az ablak, és tényleges fájlnyitás történt (, mert az igaz ágban vagyunk), a dlg példány, FileNames tulajdonsága értékkel rendelkezik (tehát ezen túl nem "null"), mégpedig a megnyitott összes fájl elérési útjával, így itt a list tömb átveszi a dlg.FileNames értékeit.
string AllList = "";
foreach (string i in list) { // A ciklus bejárja a list tömböt és értékeit az AllList változóba másolja úgy, hogy minden egyes elérési út közé tesz egy új sor karaktert, hogy az alábbi "MessageBox" egymás alatt elkülönülve tudja megjeleníteni (csak szemléltetés céljából) az elérési utakat.
AllList += i + "\n"; } // Az i változóban lévő string az AllList string típusú változó értékéhez lesz fűzve.
MessageBox.Show(AllList); } // Egy beépített párbeszédablak megjeleníti a string típusú AllList értékét.
```

5.3 Egy szövegmezőt tartalmazó saját készítésű párbeszédablak példáján keresztül mutassa be a saját készítésű Windows Forms párbeszédablakok használatának lépéseit pszeudokód vagy C# kód segítségével!

1. Ablak osztály létrehozása. A SolutionExplorer segítségével: Project Properties sorra jobb klikkel kattintva ki kell választani a megjelenő menüben az „Add” parancson belül a Windows Form menüpontot, és nevet kell adni az új ablak osztálynak, majd létre kell hozni az új osztályt. Az így létrehozott ablakot kedvünkre szerkeszthetjük. MGJ: Az ablakon elhelyezett Button DialogResult tulajdonságát beállítva adhatunk az ablak osztályunk ShowDialog() metódusának visszatérési értéket.
2. Ablak osztály példányosítása egy tetszőleges névvel, egy eseménykezelő metódusban.
3. Ablak osztály megjelenítése: példánynév.Show(); (nem modális megjelenítés) vagy példánynév.ShowDialog(); modális megjelenítés.
    a. Modális: Csak a modális módon megnyitott ablak vezérelhető, aktiválható, és rendelkezik DialogResult típusú visszatérési értékkel.
    b. Nem modális: Minden ablak vezérelhető (kiválasztható, aktiválható), void visszatérési értékkel rendelkezik az ablak osztály Show() metódusa.
    c. MGJ: A MessageBox.Show(); Metódus kivételt képez az előbb leírtak alól. A MessageBox.Show() metódus minden esetben DialogResult típusú visszatérési értékkel rendelkezik, és más ablak nem aktiválható mellette.
4. Ha szeretnénk vizsgálni a visszatérési értéket (csak ShowDialog() metódus alkalmazható ebben az esetben), akkor létre kell hoznunk egy DialogResult típusú változót melyben tároljuk az ablak osztály ShowDialog(); metódusának visszatérési értékét. (az előző példa is működhet így)
5. A visszatérési érték függvényében egy elágazás segítségével utasításokat hajthatunk végre.

Példa (C# kóddal):

A példában szereplő ablak osztály neve Form2.cs, ennek megfelelően van példányosítva az alábbi kódban.

```
Form2 ablak = new Form2();
DialogResult ertek = ablak.ShowDialog();
if (ertek == DialogResult.OK){ MessageBox.Show("Az előző ablak az OK gombbal zárult be!");}
else if (ertek == DialogResult.Cancel){MessageBox.Show("Az előző ablak a Mégse gombbal zárult be!");}
```

Ha nem szeretnénk vizsgálni a megnyitandó ablakunk kódját, akkor a megjelenítést végezhetjük az osztály Show() metódusával is. Pl.: ablak.Show (); Ebben az esetben a fenti kód lényegesen leegyszerűsödik:

```
Form2 ablak = new Form2();
ablak.Show ();
```

5.4 Mi az ablak Controls gyűjteményének feladata? Egy rövid kódon keresztül mutassa be a használatát!:

A Controls gyűjtemény, vezérlő objektumok gyűjteménye. Segítségével, például: vezérlőket tudunk elhelyezni futásidőben a formon.

Példakód:

```
RadioButton rb = new RadioButton();
rb.Text = groupBox1.Controls.Count.ToString();
rb.Top = 20 * (groupBox1.Controls.Count+1);
rb.Left = 20;
groupBox1.Controls.Add(rb);
```

5.5 Mit jelent az, hogy egy párbeszédablak modális vagy nem modális (ShowDialog, Show)?

Modális: Ha modális ablakról beszélünk, a ShowDialog(); metódussal hívtuk meg az ablak megjelenítését. A modális megjelenítés két fontos tulajdonsággal bír anem modálissal szemben. Az első, hogy DialogResult visszatérési értékkel rendelkezik, a második, hogy amely ablakból megnyitásra került az új ablak, az nem aktiválható (vezérelhető). Ahhoz hogy visszakapja a fókuszt a főablak, ki kell lépnünk az abból megnyitott új ablakból.

Nem modális: A nem modális ablak a Show(); metódussal került megnyitásra, és visszatérési értéke void típusú. A főablakból Show(); metódussal megnyitott ablak, és a főablak választhatóan vezérelhető. Tetszés szerint inaktívvá téve a mellékablakot, a főablak vezérelhető anélkül, hogy kilépnénk a Show(); metódussal megnyitott mellék (vagy másodlagos) ablakból.

MGJ: A MessageBox Show()-metódusa kivételt képez a leírtak alól, mert visszatérési értéke DialogResult típusú, és a fókuszt is kisajátítja.

5.6 Mi a Validating esemény szerepe? Hogyan használhatjuk egy vezérlőzöz (pl. szövegmező) kapcsolódóan? (Windows Forms):

A Validating eseménnyel a vezérlők ellenőrzése végezhető el. A Validating esemény akkor következik be, mielőtt a vezérlő elveszítené az input fókuszt, ha a vezérlő CausesValidation tulajdonsága True-ra van állítva (ez az alapértelmezett). Segítségével szabályozhatjuk, hogy mikor kerülhet át a fókusz egy másik vezérlőre.

Pl.: Ha a formon elhelyzünk egy textBox-ot, és azt szeretnénk, hogy csak egész számokat fogadjon a mező, akkor a textBox Validating eseményéhez készíthetünk egy feliratkozó metódust, melyben megvizsgáljuk a textBox.Text mezőbe írt szöveget. A vizsgálatra akkor kerül sor, ha a textBox-ból elkattint (vagy tabulátort használ) a felhasználó. Mivel a textBox elveszítené az input fókuszt átkattintáskor, előtte megtörténik a Validating esemény, és lefut a feliratkozott metódus, amiben elvégezhetjük a textBox.Text vizsgálatát. Ha a vizsgálat folyamán a szöveget sikerül átkonvertálni egész számmá, akkor megtörténik a textBox input fókuszának az elvesztése, és a felhasználó áttérhet másik vezérlőre, vagy mást aktívvá tehet. Ha azonban nem egész számot ad meg a felhasználó, az int típusra történő konvertálás hibát dob, és a megfelelően hibakezelt kód figyelmeztetheti a felhasználót a hibára.

6.Grafikus megjelenítés Windows Forms felületen

6.1 Mit jelent az, ha az ablak egy része érvénytelenné válik? Mikor fordulhat ez elő?

A kép megjelenítését a képmegjelenítő komponens Paint eseményéhez kívánjuk kapcsolni. Mikor keletkezik Paint esemény? Egy terület érvénytelenítése Paint eseményt idéz elő. Ez bekövetkezhet úgy, hogy a programozó idézi elő szándékosan egy utasítással, vagy ha a teljes komponens (ablak), vagy annak egy része érvénytelenné válik, pl. takarásból újra látható lesz, vagy átméretezi a felhasználó az ablakot, de ez az esemény az ablak első megjelenésekor is bekövetkezik.

6.2 Ismertesse az "Eltűnő kép" problémát és a négy lehetséges megoldást három-három mondatban! (Windows Forms) :

A megjelenítés problémája az eltűnő kép (sebesség).

Megoldásai:

- Mindig újrarajzoljuk
- Metafájlba lementjük és visszaolvassuk
- Képre rajzolás
- Dupla pufferelés

6.3 Ismertesse részletesen a dupla pufferelés technikáját! Milyen előnye lehet a képre rajzolással szemben? (Windows Forms) :

Dupla pufferelés: A gyorsítókártya - amíg az aktuális frame kirajzolása tart - egy második pufferben, a kép pufferben (frame buffer) tárolja a megjelenítés során a következő frame-et. Az aktuális frame kirajzolását követően rögtön megjeleníthető a következő frame és ez felgyorsítja a megjelenítést.

6.4 Hogyan oldaná meg azt a feladatot, hogy a jobb oldali egérgomb lenyomva tartása, és az egér mozgatása folyamatos vonal megrajzolását eredményezze?

```
int x, y;
private void Form1_MouseMove(object sender, MouseEventArgs e) {
if (e.Button == MouseButtons.Right){
System.Drawing.Graphics formGraphics = this.CreateGraphics();
formGraphics.DrawLine(Pens.Blue, x, y, e.X, e.Y);
x = e.X;
y = e.Y;
formGraphics.Dispose();}}
private void Form1_MouseDown(object sender, MouseEventArgs e){
x = e.X;
y = e.Y;}
```

7 Windows 8 stílusú alkalmazások fejlesztése

7.1 Ismertesse a következő vezérlők fontosabb jellemzőit: WrapGrid, VariableSizedWrapGrid, RepeatButton, HyperlinkButton!

WrapGrid: Olyan tároló, melyben egyforma szélességű és magasságú cellákat hozhatunk létre. Az elemeket horizontálisan és vertikálisan lehet elhelyezni benne. Alapértelmezés szerint először az oszlopokat tölti ki, ha az oszlopban nincs több sor, a következő oszlopot kezdi kitölteni.

VariableSizedWrapGrid: Egy tároló, melyben az elemek balról jobbra, illetve felülről lefelé helyezkednek el. Ha egy elem túlnyúlik a konténer szélén, pozíciója átkerül a következő sorba, vagy oszlopba.

RepeatButton: különlegessége abban rejlik, hogy ha megnyomjuk és nyomva tartjuk, akkor úgy viselkedik, mintha folyamatosan nyomkodnánk. A **Button** vezérlővel szemben, ami csak egyetlen **Click** eseményt küld, ha lenyomjuk, a **RepeatButton** folyamatosan küldi a **Click** eseményeket, amíg csak nyomva tartjuk. Ennek a vezérlőnek két fontos tulajdonsága van, a **Delay** és az **Interval**. Előbbivel ezredmásodpercekben megadhatjuk, hogy a legelső **Click** esemény után mennyi idő teljen el, mielőtt elkezdené „szórni" magából a további **Click** eseményeket. Ez hasonló, mint a szövegszerkesztőknél a törlés, ahol a backspace megnyomásakor először csak egy karaktert töröl vissza, majd ha kellő ideig nyomva tartjuk azt, akkor elkezd nagyobb tempóban törölni. Az **Interval** tulajdonság a tempót fogja meghatározni, azt hogy milyen gyakorisággal küldjön nyomva tartás esetén **Click** eseményeket a gomb. Ezt az értéket szintén ezredmásodpercekben kell megadnunk.

HyperlinkButton: Ennek a gombnak az a különlegessége, hogy a **NavigateUri** tulajdonságán keresztül megadhatunk neki egy címet, és amikor rákattintunk, az adott címre navigál. Ha ez egy weboldal címe, akkor megnyitja a böngészőt. Ha egy „mailto" hivatkozás, akkor megnyitja a levelezőalkalmazást az adott címzettel.

*Lényegében ugyanazt érjük el ennek a gombnak a használatával, mintha a **Launcher** API **LaunchUriAsync** metódusát hívnánk meg egy sima **Button** eseménykezelőjéből.*

7.2 Mutassa be egy példán keresztül az async-await párossal megvalósított aszinkron hívást!

Az új **async** módosítót egy metódusra alkalmazva jelezhetjük a fordítónak, hogy ez a metódus aszinkron metódushívásokat tartalmazhat. A fordítóprogram ebből tudni fogja, hogy ha a metóduson belül találkozik az **await** operátorral, akkor az utána következő metódushívás működését módosítania kell.

```
async Task<string> Metódusnév(int i){ //… }
await Metódusnév (1212);
```

7.3 Ismertesse a Windows 8 által támogatott szenzorokat!

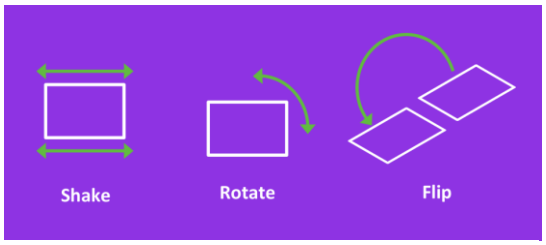
A Windows 8 a különböző hardverek és szenzorok kezelését teljesen új szintre emelte. A táblagépeknek köszönhetően a felhasználó munkakörnyezete jelentősen átalakult.

Nem olyan méretes számítógéppel dolgozunk, amelyhez a szoba bútorzatát és berendezését igazítjuk, hanem egy könnyed kis eszközről beszélünk, amely egész nap segítheti a felhasználót – különböző kihívásoknak megfelelve. A folyamatos környezetváltozáshoz igazodnunk kell, a fényviszonyoknak megfelelő megvilágítást kell adnunk a kijelzőnek, az alkalmazásoknak az aktuális térbeli elhelyezkedésnek megfelelő nézetet kell biztosítanunk, a különböző információk kiszolgálásához pedig tudnunk kell, hogy éppen merre jár a felhasználó. A táblagépek rengeteg olyan szenzorral vannak felszerelve, amelyekkel eddig nem igazán találkozhattunk. A szenzorok különböző típusú adatokat szolgáltatnak vissza, és különböző mérési

9-8 ábra: A Windows 8 által szolgáltatot

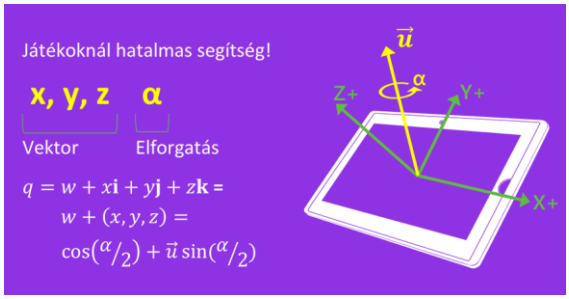
elvárásokkal rendelkeznek. Ha magunknak kellene ebben az információáradatban kiigazodnunk, elég sok munka állna előttünk már egészen egyszerű alkalmazások megírása során is. Azonban a Windows 8 platform szinten támogatja az eszközök kezelését, az általuk mért adatokat pedig a legkényelmesebb formában teszi számunkra elérhetővé. Nem mindig szükséges tudnunk, hogy melyik fizikai eszköz mérte a legutóbbi adatokat, csupán azt kell megmondanunk, hogy mire vagyunk kíváncsiak. A 9-8 ábra a különböző mérési lehetőségeket mutatja be.

Vegyük a következő egyszerű példát: szükségünk van az adott eszköz térbeli elhelyezkedésére, azonban ez a mi esetünkben kettős fogalomnak számít. Nem minden esetben szükséges tudnunk, hogy a Descartes féle koordinátarendszerben különböző tengelyekre nézve mekkora szöggel fordították el a táblagépet, vagy adott tengelyen hány egységgel mozdult el. Mindösszesen arra van szükségünk, hogy tudjuk: álló vagy fekvő üzemmódban használja-e a felhasználó a gépezetet, ezzel az adott orientációhoz legjobban illeszkedő felhasználói felületet jeleníthetjük meg számára. Az egyszerű orientációs adatokra mutat példákat a 9-9 ábra.



9-9 ábra: Az elhelyezkedéssel összefüggő

Következő példának nézzük a számítógépes játékokat! Elképzelhető, hogy egy olyan autó szimulátort írunk, ahol a felhasználó a táblagép forgatásával kormányozhatja autóját. Itt nem elegendő azt tudnunk, hogy éppen fekvő vagy álló állapotban van-e az eszköz, helyette nagyon pontos szögekkel és irány menti gyorsulásokkal kell számolnunk. Különböző szenzorok szükségesek a két említett egység kiméréséhez, és a visszaszolgáltatott adatokat valahogy fel kell dolgoznunk. A Windows 8 bevezeti a *sensor fusion* fogalmát, amely lehetővé teszi, hogy a pontos hardvereszközök ismerete nélkül mérjünk összetett adatokat, esetünkben különböző vektorértékeket és szögelfordulásokat. A visszaérkező információ pedig a játékokban a legkönnyebben felhasználható formában kerül vissza, úgynevezett *kvanterniók*ra leképezve. A 9-10 ábra vizuális formában is bemutatja a leírtakat.



9-10: A mérések egyszerűsödése Win platf

Egy egyszerű példa: kamera és mikrofon használata

A következő példában egy kamerát és mikrofont használó programrészt fogunk elkészíteni. Nincs szükségünk másra, mint egy Windows Store alkalmazássablonra Visual Studióban. Említettem többször is a fejezet során, hogy a homokozónak és a Windows Store elvárásoknak köszönhetően nem férhetünk hozzá szabadon a különböző hardverelemekhez, az alkalmazás által igényelt komponenseket meg kell jelölnünk a **Package.appxmanifest** fájlban. Kattintsunk is rá kettőt, majd a Capabilities fület kiválasztva jelöljük ki az alábbi elemeket: Microphone, Music Library, Pictures Library, Webcam! Ahogyan azt a nevük is mutatja, a rendelkezésünkre álló audio- és videoeszközöket fogjuk használni, majd a rögzített tartalmakat a Windows 8 speciális mappáiba kimenteni. A fent leírt beállításokat a 9-11 ábra mutatja be.

8.Adatbázis kezelés Windows Forms

8.1 Ismertesse az adatkötés típusok osztályozását, és ismertesse mindegyik típust egy-egy mondatban. (Windows Forms)

- Egyszerűen és kényelmesen teremt kapcsolatot egy form vezérlői és egy adathalmaz között
- Kötés létesíthető DataSet-hez, tömbhöz, gyűjteményhez, stb. (támogatnia kell az indexelt hozzáférést)

Osztályozás:

- **Egyszerű kötés (egy rekord egy adata):** Egyszerre egy adatot köt egy vezérlő egy tulajdonságához, avezérlő egy tulajdonságába egy DataSet egy táblája egy oszlopának az aktuális mezője kerül
- **Komplex kötés (több rekord):** Egy listát kötnek egy vezérlőhöz, egyszerre egynél több rekord adatai jeleníthetők meg.
- **Közvetlen kötés**
- **Közvetett kötés (BindingSource):** Megkönnyíti az adatok közötti navigálást és az adatforrás esetleges lecserélését

8.2 Milyen szerepet játszanak a Format és Parse események? (Windows Forms)

Az adatforrásból származó adatok típusa gyakran nem egyezik meg a vezérlő által elfogadott típussal

Format esemény: adat kerül az adatforrásból a vezérlőhöz (adatkötés létrehozásakor, Position értékének változásakor, rendezés, szűrés)

Parse esemény: adat mozog a vezérlőtől az adatforrás felé (a vezérlő Validated eseménye után, az EndCurrentEdit metódus meghívásakor, ha változik a Position)

8.3 Hasonlítsa össze a közvetett és a közvetlen adatkötést kiemelve fontosabb tulajdonságaikat. (Windows Forms)

Közvetlen kötés esetén az adatforrás (DataSet) és a vezérlő egy tulajdonsága között létesítünk kapcsolatot, míg a közvetett esetben a kettő között egy újabb réteg jelenik meg a BindingSource objektum.