Objektumorientált fogalmak, felügyelt környezetek

1. Ismertesse a felügyelt környezetek fontosabb szolgáltatásait!

Egy plusz réteg az OS fölött, plusz szolgáltatásokat nyújt

- Szemétgyűjtés (garbage collector, GC): a már nem hivatkozott dinamikusan lefoglalt memóriát szabadítja fel –nem kell „delete”-et hívni a „new”-val dinamikusan lefoglalt memóriára. Kevesebb kód, nincs memóriaszivárgás.
- Típusellenőrzés (pl. érvénytelen konverziók kiszűrése)
- Biztonság, és még rengeteg további szolgáltatás
- Ugyanaz az alkalmazás futhat különböző operációs rendszereken (újrafordítás nélkül)
- Egységes kivételkezelés
- Nyelvi szinten és a CLR-ben definiált hibakezelés Integrálódik a Windows strukturált kivételkezelésével is (SEH)

2. Ismertesse a .NET Keretrendszer architektúráját!

3. Ismertessen a fordítás és végrehajtás lépéseit .NET környezetben!

Forrás kód→ Nyelvi fordító → Szerelvény, assembly:[Köztes kód (IL) + metaadatok (.dll,.exe)]
Végrehajtás: JIT fordító (első futtatásnál ill. telepítésnél)→ natív kód

Azonos kimenet: IL (Intermediate Language)

- Köztes kód
- Olyan, mint egy objektumorientált assembly nyelv
- Processzor és architektúra független
- Továbbfordításra tervezték, nem interpretált!
- Nyelvfüggetlen
- Metaadat: típusok leírása, tagváltozók, metódusok leírása
- „Könnyű" visszafejteni, pl. Reflector

4. Helló világ konzol alkalmazás C# nyelven

```
using System;
namespace HelloWorld
{
        class Program
        {
                static void Main()
                        {
                                Console.WriteLine("Hello world ! ");
                        }
        }
}
```

5. Objektumorientált nyelvi elemek C# nyelven, példákkal: osztály, láthatóság szabályozása, statikus tagok, öröklés, virtuális függvények, behelyettesíthetőség, névtér, generikus típusok használata (írásuk nem)

Osztály:

- Egységbezárás (encapsulation)
- Adat (tagváltozók) + kód (tagfüggvények)
- Tagváltozók - mint a struktúra (struct) esetében
- Tagváltozó, mező, field
- Tagfüggvények
- Globális függvényeket bevisszük a struktúrába
- Globális függvény nincs
- Tagfüggvény, művelet, metódus, operation
- Osztály
- Mint a struktúra, majd később értjük meg a különbséget.
- Az osztály példánya az objektum:
- Osztály: típusnak felel meg
- Objektum: változónak felel meg

Példa:

```
class Rect
    {
        public:
        int x;
        int y;

        void Print() { ... }
    };
```

Láthatóság szabályozása:

- private: csak az adott osztály tagfüggvényei számára látható (ez az alapértelmezett)
- public: minden osztály számára látható
- protected: az adott osztály és leszármazottai számára látható
- internal: csak az adott szerelvényen (assembly) belül látható (ez a későbbi tanulmányok során lesz érthető)
- protected internal: a protected és az internal kombinációja

Példa:

```
private int v;
protected char Q;
```

Statikus tagok:

- **Statikus tagváltozó**: Az osztály minden objektumára közös változó (nem objektumonként foglalódik neki tárhely).
- **Statikus tagfüggvény:** Objektum nélkül hívható, az osztály nevén keresztül

Példa:

-

```
Console.WriteLine
Static int eget;
```

Öröklés:

- Csak egy őse lehet mindenkinek

Példa:

```
public class Circle: Shape
    {}
```

Virtuális függvények:

- Ha azt szeretnénk, hogy futás közben nézze meg a kód, hogy milyen típusú objektumra mutat a referencia (jelen esetben a this), akkor a meghívott függvényt virtuálisnak kell definiálni
- Az ősben virtual kulcsszó
- A leszármazottban override
- Megnézi, hogy milyen típusú objektum van a referencia mögött és az abban definiált függvényt hívja meg

- Ökölszabály: ha egy függvényt felüldefiniálunk,
- legyen virtuális.

Absztrakt függvény és osztály:

- Absztrakt függvény – nem adunk meg törzset neki, csak majd a leszármazott osztályban
- Absztrakt osztály – van legalább egy absztrakt függvénye. Nem példányosítható.

Példa:

```
public virtual void PrintName() { Console.Write("Shape");
….
public override void PrintName() { Console.Write("Circle");
```

Behelyettesíthetőség:

- behelyettesíthetőségi szabály: őspointerrel (illetve C#-ban ősreferenciával) lehet leszármazottra is mutatni.
- Gyerekből szülőre (pl. Rect-re hivatkozás Shape referenciával) automatikus a konverzió
- Ősről leszármazottra (lefele konverzió)
  - Explicit cast
  - Veszélyes, el lehet rontani
- Az „is” operátorral tudjuk ellenőrizni, hogy egy típus kompatibilis-e egy másikkal (bool-lal tér vissza) :

```
// is operátor
Shape s = new Rect(10, 20, 10, 10);
if (s is Rect)
{
   Rect r = (Rect)s; // Explicit cast, biztos OK, hiszen ellenőriztük
   // ...
}
```

- Az „as” operátor működése hasonló, de ez megkísérli a konverziót. Ha nem lehetséges, null-lal tér vissza (de nem dob kivételt):

```
// as operátor
Shape s = new Rect(10, 20, 10, 10);
Rect r = s as Rect;
if (r != null)
```

Példa:

```
Rect r1 = new Rect(10, 20, 10, 20);
Shape s1 = r1;                    // Automatikus Rect -> Shape
Shape s2 = new Circle(1, 2, 3);
```

Névtér:

- Nagyon sok definíció (osztály, interfész, struktúra)
- C-ben minden globális

- C# névterekbe szervezzük. Logikai csoportosítás
- System – legfontosabb általános osztályok, pl. String

Példa:

System.IO – fájlkezelés

Generikus típusok használata:

- Olyan típus, ami nem teljes
- A felhasználás során adjuk meg pl. a listánál az elemtípust
- ArrayList helyett - List<T>
- Típusbiztos – nem kell castolni, amikor elérjük az elemeket.
- Mint a C++-ban a template

Példa:

List<T>

6. Modern nyelvi eszközök C# nyelven, ismertetésük példákkal: interfész, property, delegate, event, attribútum

Interfész:

- Az interfész nem más, mint egy művelethalmaz a műveletek szignatúrájának és a visszatérési érték típusának pontos megadásával. A műveletekhez implementációt (törzset) nem adhatunk. Szintaktikailag olyan, mint egy osztálydefiníció, de a class helyett az interface kulcsszót használjuk
- Egy osztály által implementált interfészeket az osztály neve utáni „:”-ot követően kell megadni
- ha egy osztály implementál egy interfészt, akkor valamennyi műveletét implementálnia kell.
- Egy osztályra (pontosabban objektumaira), hivatkozhatunk bármelyik általa implementált interfészként.
- Az ősosztályokhoz hasonlóan lehetővé teszik a különböző típusú objektumok egységes kezelését.

Példa:

```
public interface IComparable
{
        // Összehasonlítja az objektumot a paraméterként kapott másikkal.
        int CompareTo( Object obj );
}
```

Property:

- Egyszerűsített szintaktikát biztosít a get és set tagfüggvények kiváltására. Tipikusan tagváltozókhoz való szabályozott hozzáférésre, illetve számított értékek visszaadására használjuk. .NET specifikus

Példa:

```
class Person
{
    private string name;
    private int yearOfBirth;

    // Declare a Name property of type string:
    public string Name
    {
       get { return name; }
       set
       {
       if (value == null)
       throw new ArgumentNullException("The Name property can not be
 null.");
         name = value;
       }
 }
```

Delegate (metódusreferencia):

- Olyan, mint a C függvénypointer, csak objektumorientált: tagfüggvényekre mutat
- A delegate kulcsszóval delegate típus definiálása
- Delegate típus alapján változók (delegate objektum)
- Egy delegate objektum több metódusra is mutathat.
- Lehet statikus függvényre is metódusreferencia, nem kell hozzá objektum (interfésznél mindig szükség van egy összehasonlító objektumra).
- Akkor is használható, ha nem tudjuk megoldani, hogy implementálja az adott interfészt az osztály (pl. az IComparable-t)

Példa:

```
delegate bool FirstIsSmallerDelegate(object a, object b);
public static void HyperSort(ArrayList list,
      FirstIsSmallerDelegate firstIsSmaller)
   {
      for (...)
      {
     ...
            if (firstIsSmaller(list[j], list[j - 1]))
      ...

// Egyszerűsített szintaktika
      Sorter.HyperSort(list, FirstIsSmaller_Complex);
```

Event:

- Egy osztálynak nemcsak tagfüggvényei, tagváltozói, tulajdonságai lehetnek, hanem eseményei is.

- Más objektumok előfizethetnek a számukra érdekes eseményekre, amit ha a publikáló osztály elsüt, az előfizetők értesülni fognak.
- Publisher/Subscriber tervezési minta
- Az event kulcsszóval definiálhatók, delegate-ekre épül
- Előfizetés a += operátorral
- Eseményvezérelt programozást tesz lehetővé

Példa:

```
// Esemény elsütése: meghívódik az összes előfizető beregisztrált függvénye.
 if (Log != null)
    Log(msg);

        // Előfizetünk az eseményre két módon is.
   public App()
   {
       log.Log += new LogHandler(writeConsole);
      log.Log += new LogHandler(fileLogListener.WriteToFile);
   }
// Takarításkor leíratkozunk az eseményről.
        public void Cleanup()
   {
      log.Log -= new LogHandler(writeConsole);
      log.Log -= new LogHandler(fileLogListener.WriteToFile);
```

Attribútum:

- Metainformáció rendelhető szinte minden nyelvi elemhez
- Osztályokhoz, metódusokhoz, tulajdonságokhoz, paraméterekhez, mezőkhöz, szerelvényhez, stb.
- Ezek az információk megváltoztathatják a fordító viselkedését (beépített attribútumok) VAGY futási időben lekérdezhetőek
- Használja őket a keretrendszer
- Használhatók saját kódban
- Szintaktika: [XXXXX]

Példa:

```
[Serializable]
class SomeClass
{
[NonSerialized]
int n;
string Test([SomeAttr] string param1) {…}
}
```

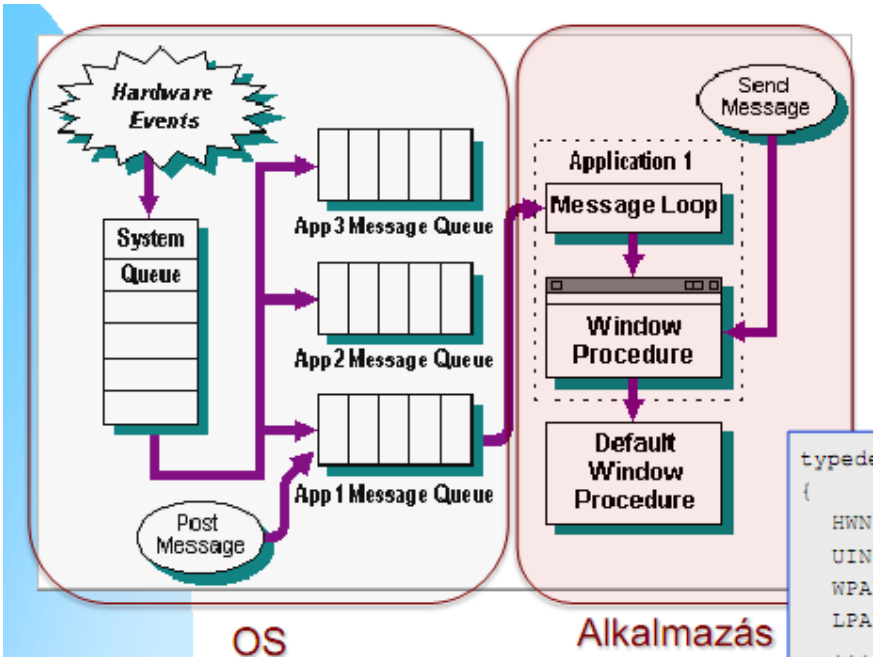
7. Referencia és érték típusok

Érték típus (value type): int, char, decimal, float, bool, enum, stb. egyszerű típusok tartoznak ide, meg amit mit definiálunk a struct kulcsszóval. Inline módon foglalnak helyet összetett típusokban. Lokális változónál a vermen foglalódik nekik hely. Függvényparaméter átadáskor pont úgy kezelődnek, mint C++-ban az int, vagy minden más: másolat készül az eredeti adatról. Gyors az allokációjuk. Korlátozások: nem örökölhetnek, nem lehet belőlük származni. Interfészt viszont implementálhatnak

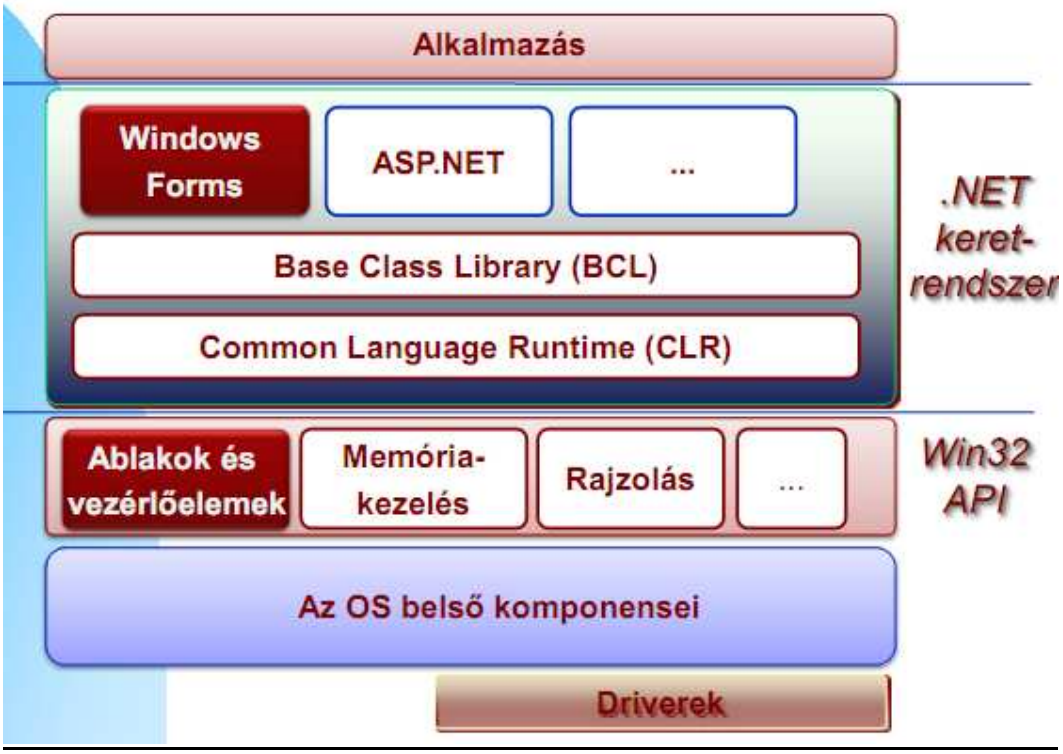
A referencia típus (reference type): két részből áll: egy mutató/hivatkozás, ami alapértelmezésben null, és maga a mutatott/hivatkozott objektum, ami a felügyelt heapen foglal helyet, és a garbage collector gyűjti be, ha már nincs rá hivatkozás. Ez utóbbinak nekünk kell a new-val helyet foglalni. A .NET beépített osztályai (pl. string, File, stb.) ilyenek, a tömbök, meg amit mi hozunk létre a class kulcsszóval. Az interfészek is ide tartoznak, később lesz róluk szó.

Vastagkliens alkalmazások fejlesztése

8. Ismertesse ábrával illusztrálva a natív Win32 platform üzenetkezelését (ismétlés)! Ennek során térjen ki a következő fogalmakra: üzenet, üzenetsor, üzenetkezelő ciklus, ablakkezelő függvény, callback függvény, alapértelmezett ablakkezelő függvény!



9. **Ismertesse a .NET vastagkliens alkalmazások architektúráját (rétegek)!**



10. Ismertesse a .NET részleges típus (partial class) fogalmát és főbb felhasználási területét!

- Pl osztály két külön részletben van megírva
- A fordító fésüli össze (nem lehetnek a részek külön szerelvényben)
- Fő területe: a generált és a kézzel írt kód különválasztása
- Főleg Windows Forms, ahol a Drag&Drop-os szerkesztőből a fordító generál kódot, a fejlesztő meg írja a többit

11. Ismertesse a Windows Forms alkalmazások architektúráját!

- System.Windows.Forms névtér
- Minden ablak egy Form leszármazott osztály
- A Form számos tulajdonsággal rendelkezik (pl. BackColor, Text, Size, ..), számos eseményt publikál (Load – betöltéskor, Click – egérkattintás, Resize – átméretezés, Move – Mozgatás, KeyDown –billentyűlenyomás)

- A Formon vezérlőelemeket helyezünk el (mint pl. TextBox, Label, stb.) vagy vizuálisan a Visual Studio designer-ben, a ToolBox-ról, vagy programozottan

- A vezérlőelemek a Form leszármazott osztály tagváltozói lesznek ,a konstruktorból hívott InitializeComponent-ben példányosítódnak
- Számos tulajdonsággal rendelkeznek (pl. Font property) és számos eseményt publikálnak (pl. a TextBox TextChanged-et)
- A Visual Studio részleges osztályokat generál

12. Ismertesse az üzenet és eseménykezelés kapcsolatát (pl. billentyűlenyomás esetére)

- A .NET alkalmazások ráépülnek a natív üzenet és ablakkezelésre
- Az ablakoknak és a vezérlőknek van ablakleírója (HWND, Control.Handle tulajdonság)
- A form egy csomagoló egy natív ablak körül
- A form és a vezérlők a Windows üzeneteket .NET eseményekké konvertálják
- Az alkalmazásnak van üzenetkezelő ciklusa

13. Ismertessen a fontosabb események kezelésének menetét (EventHandler, EventArgs, billentyű események)!

- A sender: az esemény kiváltója
- EventArgs: esemény paraméterek
- Az EventArgs nem hordoz információt, ebből kell leszármaztatni, ha van esemény paraméter
- Pl.: public delegate void EventHandler (Object sender, EventArgs e)

14. Hogyan lehet menüt és eszközsávot Windows Forms alkalmazásokban készíteni?

- Windows Forms felületen, Drag&Drop módszerrel, oldalról
- MenuStrip, ToolStrip
- Utána lehet Property-ket állítani
- Programozottan is lehet

15. Ismertesse a párbeszédablak fogalmát! Kódrészlettel illusztrálva ismertesse a párbeszédablakok kialakításának és megjelenítésének módszerét!

- Párbeszédablakok (=dialógusablakok) célja
- Tipikusan beállítások megjelenítésére, megváltoztatására
- Modális megjelenítés (nem lehet más ablakra átváltani)
- Az adatoknak tulajdonságok felvétele a Form osztályunkban
- A példánkban Interval néven
- Az Form a DialogResult tulajdonságában jelzi, hogy érvényes-e az új érték (OK gombbal zárta-e be a felhasználó az ablakot).
- Lehetséges értékek:
  DialogResult.Ok, DialogResult.Cancel, DialogResult.Yes, ...
- Modális megjelenítés: Form.ShowDialog
- Addig nem tér vissza, amíg be nem záródik a párbeszédablak.

16. Ismertesse a nemmodális ablak fogalmát és kódrészlettel illusztrálja megjelenítését!

- Aktiválható más ablak is
- Megjelenítés:
  - A Show-nak megadhatunk egy owner (birtokos) ablakot
  - Nem kerülhet az owner (birtokos) ablak elé Z-orderben
- SettingsForm form = new SettingsForm();
  form.Show();
  SettingsForm form = new SettingsForm();
  form.Show(this); // A this egy Form leszármazott

17. Ismertesse a közös párbeszédablak fogalmát és ismertesse fajtáit!

- .NET csomagolók a Win32 API közös párbeszédablakok körül
- pl: OpenFileDialog, SaveFileDialog, ColorDialog, FolderBrowserDialod, FontDialog

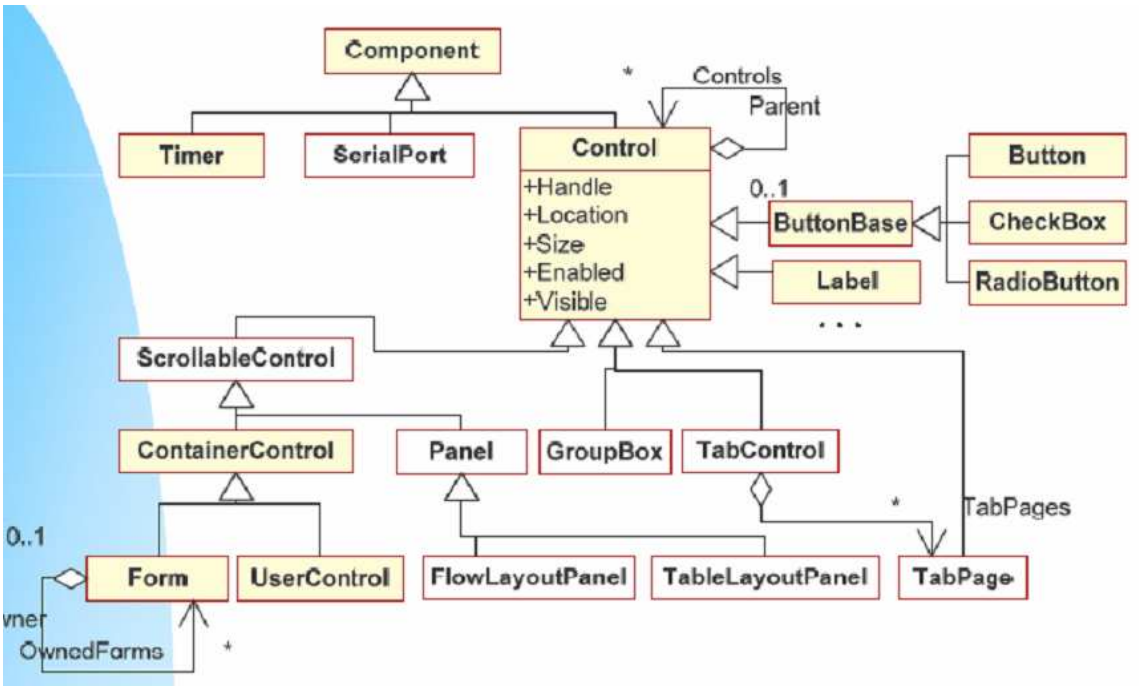
18. Sorolja fel a fontosabb elemi vezérlőket!

- Button, CheckBox, ComboBox, Label, Listbox, ListView, TextBox, RadioButton...

19. Sorolja fel a fontosabb tároló vezérlőket!

- GroupBox, Panel, SplitContainer, TabControl, FlowLayoutPanel
-

20. Rajzolja fel a komponens/vezérlő hierarchia fontosabb osztályait és kapcsolatukat! Ismertesse a fontosabb osztályokat (Component, Control, Form, ...)!



Component (komponens):

- Bármilyen, container (pl. designer) által tartalmazható komponens
- Nem feltétlenül vizuális (pl. SerialPort, Timer), de fel lehet dobni a designerbe, megadhatók vizuálisan a tulajdonságok és események

Control (vezérlő):

- Minden vezérlő őse
- Natív ablak HWND tartozik hozzá (Handle tulajdonság)
- Összes közös tulajdonság
- Visible
  - Enabled
  - Controls – Tartalmazott vezérlők listája (csak az összetett vezérlők)
  - Location, Size, ...
- Összes közös művelet
  - Focus(), ...
- Összes közös esemény

- Click
- KeyDown, ...

21. Ismertesse a vezérlők és űrlapok közötti lehetséges viszonyokat (része hierarchia)!

Szülő – gyerek viszony

A tartalmazott vezérlők gyerekablakok (child window)

- A megszokott viselkedést ez biztosítja
  - _A gyerekablak együtt mozog a szülővel
  - Ha megszűnik a szülő, a gyerekablakok is automatikusan megszűnnek
  - A gyerekablak nem lóghat ki a szülőablakból (clipping)
  - A szülő elrejtése/megjelenítése automatikusan magával vonja a gyerekablakok elrejtését/megjelenítését
- A szülő ablak a Parent tulajdonsággal érhető el (null,ha nincs szülő)

Formok között más: birtokos-birtokolt (owner – owned) viszony

- Lazább, mint a parent-child
- Ha bezáródik a birtokos, a birtokolt ablak is bezáródik (+ugyanez a minimize-ra)
- A Z-orderben a birtokolt ablak a birtokos előtt helyezkedik el (?)
- Birtokolt ablakok: Form.OwnedForms tulajdonság
- Birtokos ablak: Form.Owner tulajdonság (null, ha nincs birtokos)

22. Írjon olyan Windows Forms C# kódot, ami egy üzenetablakban megjeleníti a leütött billentyűt!

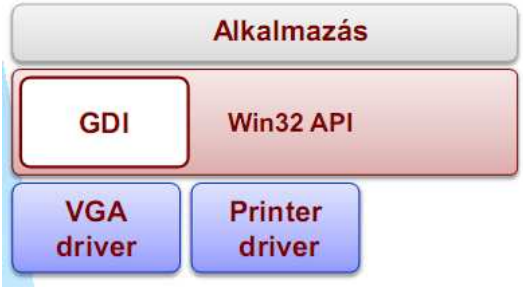
```
public partial class MainForm : Form
   {
      public MainForm()
      {
         InitializeComponent();
         this.KeyDown += new KeyEventHandler(this.MainForm_KeyDown);
      }

      private void MainForm_KeyDown(object sender, KeyEventArgs e)
      {
         MessageBox.Show("A billentyű (eseménykezelő): "       +
e.KeyCode.ToString());
      }

   }
```

23. Egy űrlapon egy timer1 nevű Timer időzítő komponens, egy label1 nevű címke, valamint egy mStart és mStop menuStrip elemekkel rendelkező MenuStrip van elhelyezve. Írjon olyan C# kódot, ami az mStart menüelem kiválasztásakor elindítja, az mStop menüelem kiválasztásakor leállítja az időzítőt. Ha az időzítő fut, másodpercenként eggyel növekvő egész számra állítsa a label1 címke szövegét. (Az eseménykezelők bekötéséről is gondoskodjon!)

```
public partial class Form1 : Form
    {
        int count = 0;

        public Form1()
        {
            InitializeComponent();
            // 01. Frissítés átméretezéskor
            this.SetStyle(ControlStyles.ResizeRedraw, true);
            UpdateStyles();
       this.bStartStop.Click += new System.EventHandler(this.bStartStop_Click);

        }

        protected override void OnPaint(PaintEventArgs e)
        {
            StringFormat format = new StringFormat();
            // Igazítás (Near – balra, Center, Far - jobbra)
            format.Alignment = StringAlignment.Center;
            // Horizontális igazítás (Near – fent, Center, Far - lent)
            format.LineAlignment = StringAlignment.Center;

            e.Graphics.DrawString(count.ToString(), this.Font,
                    Brushes.Black, this.ClientRectangle, format);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            ++count;
            Invalidate();
        }

        private void bStartStop_Click(object sender, EventArgs e)
        {
            timer1.Enabled = !timer1.Enabled;
            bStartStop.Text = timer1.Enabled ? "Stop" : "Start";
        }
    }
```

24. Ismertesse a GDI architektúráját! Mit jelent az eszközfüggetlen grafikus megjelenítés?

Eszközfüggetlen grafikus megjelenítést támogat
 Felbontás és színmélység függetlenül rajzolunk
 „Ugyanúgy” rajzolunk képernyőre, mint nyomtatóra



25. Ismertesse az eszközkapcsolat fogalmát!

Reprezentál egy grafikus eszközt ()
HDC leíró azonosítja
Lépések

- 1. Eszközkapcsolat létrehozás
- 2. Rajzolás az eszközkapcsolatra
- 3. Eszközkapcsolat lezárás

A DC „rajzolófelület „: minden rajzoló függvénynek ez az első paramétere
Eszközkapcsolat létrehozható

- Ablak kliens területre
- Teljes ablakra (pl. fejlécre, statuszbarra, menüre, stb., is tudunk rajzolni)
- Képernyőre
- Memóriába rajzolásra
- Nyomtatóra
- Metafilera
- Érvénytelen területre

26. Ismertesse a megjelenítés mechanizmusát natív környezetben! (Érvénytelen terület, WM_PAINT üzenet, stb.)

Az OS a rajzot nem jegyzi meg!
Ha érvénytelen terület keletkezik, újra kell rajzolni azt!
Érvénytelen terület:

- Korábban takarásban levő, láthatóvá vált ablakrészek (pl. átméretezés, Z-orderben előbb került az ablak, stb.)
- Miért nem jegyzi meg a rajzot az OS? Sok ablak létezhet egyszerre -> memóriakorlát!
- Honnan tudjuk, hogy érvénytelen terület keletkezett egy ablakon?
- WM_PAINT üzenetet kap az ablak

27. Kódrészlettel is illusztrálva ismertesse a megjelenítés mechanizmusát felügyelt környezetben! (Érvénytelen terület, Paint esemény, OnPaint virt. fv., Graphics osztály, ...)

A .NET a GDI+ -t támogatja (GDI + számos extra szolgáltatás)
Nagyon hasonlít a natív Win32 API modellhez
System.Drawing névtér és szerelvény
Eszközkapcsolat (DC) helyett System.Drawing.Graphics objektumra rajzolunk, ami egy rajzolófelületet reprezentál
Érvénytelen területre rajzolás, mint natív esetben
Mi hogyan kezelhetjük?
űrlap/vezérlőelem Paint eseményéhez eseménykezelőt.
Vagy felülbíráljuk (override) az OnPaint műveletét (hatékonyabb)
Ha szükséges: Invalidate() ->> OnPaint

Az Form leszármazott osztályunkban:

```
protected override void OnPaint(PaintEventArgs e)
        {
        e.Graphics.FillRectangle(
        new SolidBrush(Color.Blue), this.ClientRectangle);
        }_
```

PaintEventArgs paraméter
_ PaintEventArgs .Graphics objektum: erre tudunk rajzolni
_ PaintEventArgs.ClipRectangle: újrafestendő terület
Form.ClientRectangle: ablak kliens területe

_ _ Nemcsak az űrlapoknál, hanem saját vezérlőelemeknél is ugyanígy rajzolunk!
_ Az OnPaint-tet ne hívjuk explicit (az OS optimalizál)! Helyette: Invalidate-et hívjunk, ami érvényteleníti a területet és kiváltja az újrarajzolást.

28. Ismertesse a színkezelés alapjait!

Color struktúra
_ Color.Red – piros szín
_ Color.FromArgb
_ 4 színösszetev_ (alpha, red, green, blue), a tartomány mindre 0-255

```
        _ Color.FromArgb(127, Color.Red); // Félig átlátszó piros szín
        _ Color.FromArgb(100, 255, 0, 0); // Félig átlátszó piros szín
```

_ Color.Empty – üres szín (null-ak felel meg)
_ Color.Transparent – A transzparens színt reprezentáló szín (tipikusan háttérszínnek szokás megadni) – lásd később

29. Ismertesse a Pen és a Brush fogalmát, röviden a típusait és egy egyszerű példával illusztrálja használatukat!

A toll vonalak színét, mintáját, vastagságát határozza meg
_ Pen osztály

Pen pen = new Pen(Color.FromArgb(150, Color.Blue), 2)

public void DrawRectangle (Pen pen, int x, int y, int width, int height)

Amit tudni kell: Pen használata DrawLine és DrawRect esetén.
_ Vannak előredefiniáltak, pl. Pens.Blue -- - kék folytonos 1px vastag vonal
_ Egyszerű: nem kell létrehozni, felszabadítani!

BRUSH:
Az ecset az alakzatok kitöltési színét, mintáját határozza meg, Brush ősosztály, leszármazottak
- SolidBrush – adott színnel „tele” kitöltés
_ HatchBrush – vonalmintával kitöltés, pl.
_ TextureBrush – bitmintával kitöltés
_ LinearGradientBrush – lineáris színátmenet

e.Graphics.FillRectangle(new SolidBrush(Color.Green), 10, 30, 200, 25);

30. Ismertesse a metafájl fogalmát!

Előre rögzített, lejátszható vektorgrafikus műveletek.

31. Ismertesse a szöveg megjelenítésének lehetőségeit! Mutasson példát szöveg adott koordinátában való megjelenítésére!

Egyszerű, az űrlap betűtípusát használva, (10, 10) pontban

```
protected override void OnPaint(PaintEventArgs e)
{
        e.Graphics.DrawString("Hello World", this.Font, new SolidBrush(Color.Black), 10,
10);
}
```

Téglalap bal felső sarkában (ami nem fér, levágja)

```
protected override void OnPaint(PaintEventArgs e)
{
        e.Graphics.DrawString("Hello World",25
        new Font("Lucida Sans Unicode", 8),
        new SolidBrush(Color.Blue),
        new RectangleF(100, 100, 250, 350)); // Befoglaló téglalap
}
```

A StringFormat-tal „mindent’’ lehet

32. Ismertesse a virtuális ablakok módszerét! Ennek során térjen ki a következőkre: milyen feltételek esetén célszerű alkalmazni; mik a megoldás főbb lépései?

Ha maga a rajzolás lassú (pl. bonyolult rajz, ábra), a megjelenítés villog, átméretezéskor akadozik. Oka: minden OnPaint esetén lefut a lassú megjelenítő algoritmus.
 Vegyük észre: ezen a duplapufferelés engedélyezése sem segít.
 Megoldás: manuális duplapufferelés (virtuális ablakok módszere). Alapelve: egy memória Graphics objektumra rajzolunk, az OnPaint-ben ennek tartalmát másoljuk a képernyőre (ez nagyon gyors művelet).Így ha sok OnPaint esemény keletkezik is, gyorsan le fog futni mind: nem villog, nem szaggat.
Menete:

1. Előkészítés: a bitmapet egy tagváltozóban őrizzük meg
2. Ha változik a rajz: a memóriában a bitmapre elkészítjük az új rajzot
3. Az OnPaint-ben kirajzoljuk a bitmapben előkészített rajzot (ez nagyon gyors, nem is villog)

33. Példa: Írjon olyan C# nyelvű alkalmazást, ami a (10,10) koordinátában megjeleníti az ablakfrissítések számát!

```
public partial class Form1 : Form
    {
        int count = 0;

        public Form1()
        {
            InitializeComponent();

            // 01. Frissítés átméretezéskor
            this.SetStyle(ControlStyles.ResizeRedraw, true);
            UpdateStyles();

            // 03. Duplapufferelés engedélyezése - ki kell szedni a kommentet
            // DoubleBuffered = true;
        }

        protected override void OnPaint(PaintEventArgs e)
        {
            ++count;
            e.Graphics.DrawString(count.ToString(), this.Font,
                    Brushes.Black, this.ClientRectangle, 10,10);
        }
    }
```

34. Példa: Írjon olyan C# nyelvű alkalmazást, ami a (10,10) koordinátában másodpercenként eggyel növelve megjeleníti egy számláló értékét!

```
public partial class Form1 : Form
    {
        int count = 0;

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled;
            // 01. Frissítés átméretezéskor
            this.SetStyle(ControlStyles.ResizeRedraw, true);
            UpdateStyles();
        }

        protected override void OnPaint(PaintEventArgs e)
        {

            e.Graphics.DrawString(count.ToString(), this.Font,
                    Brushes.Black, this.ClientRectangle, 10,10);
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            ++count;
            Invalidate();
        }
    }
```

35. Írjon olyan C# nyelvű alkalmazást, ami a (10,10) koordinátában másodpercenként eggyel növelve megjeleníti egy számláló értékét!

```
public partial class Form1 : Form
    {
        int color = 0;

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled;
            // 01. Frissítés átméretezéskor
            this.SetStyle(ControlStyles.ResizeRedraw, true);
            UpdateStyles();
        }

        protected override void OnPaint(PaintEventArgs e)
        {
           If (color==1)
            e.Graphics.FillRectangle(new SolidBrush(Color.Blue), 10,10,20,20);
           else
              e.Graphics.FillRectangle(new SolidBrush(Color.Green),
                                       this.ClientRectangle,10,10,20,20);

        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            Color= ! color;
            Invalidate();
        }
```

36. Írjon olyan C# nyelvű alkalmazást, ami másodpercenként felváltva megjelenít egy tele kék és zöld színnel kitöltött négyzetet a (10,10) pontban 20 pixel oldalhosszúsággal!

```
public partial class Form1 : Form
    {
        int color = 0;
              int x=10,y=10;

        public Form1()
        {
            InitializeComponent();
              // 01. Frissítés átméretezéskor
            this.SetStyle(ControlStyles.ResizeRedraw, true);
            UpdateStyles();
            Graphics.DrawRectangle(Pen.Red, x,y,10,10);

              this.KeyDown +=new KeyEventHandler(this.MainForm_KeyDown);

        }

        protected override void OnPaint(PaintEventArgs e)
        {

                  e.Gr Graphics.DrawRectangle(Pen.Red, x, y,10,10);

        }

        private void MainForm_KeyDown(object sender, KeyEventArgs e)
       {

              If(e.KeyCode==Keys.Up)
                     {y=y-1;}
              If(e.KeyCode ==Keys.Down)
                     {y=y+1;}
              If(e.KeyCode ==Keys.Left)
                     {x=x-1;}
              If(e.KeyCode ==Keys.Down)
                     {x=x+1;}




              }
    }
```

37. Írjon olyan C# nyelvű alkalmazást, amely a (10,10) pontban egy piros, 1 pixel vastag folytonos vonallal rajzolt 10 pixel oldalhosszúságú négyzetet jelenít meg. A négyezet a kurzorbillentyűkkel lehessen mozgatni! (A kurzorbillentyűk kódja: Keys.Up, Key.Left, ...)

38. Ismertesse a saját vezérlők készítésének lehetőségeit!

- Contol osztályból leszármaztatás
  Akkor használjuk, ha a egy teljesen új vezérlőelemet szeretnénk létrehozni
  Származtassunk egy új osztályt a Control osztályból

Örököljük a minden Controlra közös tulajdonságokat (Size, Location, stb.)
Adhatunk hozzá új tulajdonságokat (property), eseményeket
A rajzolás is a mi feladatunk
 GDI+ -szal, Paint-ben
Egy példa: egy az aktuális időt mutató címke

 Adott vezérlőből (pl. TextBox) leszármaztatás
  Pl. TextBoxból, stb...
  Akkor használjuk, ha egy már létező vezérlőelemet szeretnénk testreszabni
  Csak a speciális viselkedést kell megvalósítani
  Adhatunk hozzá új tulajdonságokat, eseményeket
  Példa: egy speciális szövegablak, ami élénk háttérrel jelenik meg, ha érvénytelen e-mailcímet írt bele a felhasználó

 UserControl készítés

A vezérlőelem maga is egy űrlap, tartalmazhat vezérlőelemeket
Tervezési időben vizuálisan elkészíthetjük összetett vezérlőelemeinket, pont úgy, ahogy egy formot is elkészítenénk.
Miben más? Űrlapokra, illetve más UserControlokra lehet elhelyezni
Példa FilePicker vezérlő: tipikusan együtt előforduló vezérlőelemek összekötése
A tartalmazott vezérlőelemek private láthatóságúak – így logikus
Hogyan induljuk el? Visual Studio: Add New Item/UserControl

39. Ismertesse a UserControl felhasználásának lehetőségeit!

 Újrafelhasználás, valamint
 Összetett felhasználói felület modularizálásának eszköze!

40. Ismertesse a Finalize metódus és a destruktor kapcsolatát, valamint ismertesse működésüket!

A GC általi felszabadítás során meghívódik az objektumra a Finalize művelet (egységes CLR név) -> ez az ún. Finalizer
 Aminek C# nyelven a destruktor felel meg (~osztálynév {...})! C# nyelven nem definiálhatjuk felül a Finalize nevű műveletet!
Finalizer/destruktor jellemzők
  A végrehajtás ideje nem ismert! (amikor a CLR úgy gondolja, ideje futtatni a GC-t)
  A sorrend nem ismert
  Ha pl. 100 objektum szabadítható fel, milyen sorrendben szabadulnak fel (ezek hivatkozhatnak is egymásra!)...
  A szál nem ismert!

A „MySuperFileReader” ún. NEM FELÜGYELT erőforrást használ (esetünkben egy fájlleírót). Ezeket a GC NEM gyűjti be, felszabadításuk a mi feladatunk. Az ezért felelős kódot tettük a destruktorba.
Mit értünk el?

 A nem felügyelt erőforrások nem szivárognak el, nem maradnak felszabadítatlanul. Ez által nyer létjogosultsága destruktor .NET környezetben.
Nem felügyelt erőforrás pl.:
 Fájl
 Nem felügyelt lockok, mutexek
 Adatbázis-kapcsolat
 Minden natív ablak
Vagyis minden, amit a .NET alatti OS-ben közvetlenül foglalunk le.

41. Ismertesse a Dispose minta lényegét!

Dispose minta lényege
 A nem felügyelt erőforrást foglaló osztály implementálja az IDisposable interfészt. Ebben egy művelet van, a Dispose. Ebben kell felszabadítani a nem felügyelt erőforrást. Pl. ilyen a szövegfájlok olvasására szolgáló StreamReader és az írásra szolgáló StreamWriter osztály.
 Sok esetben beszédesebb néven is elérhető a Dispose() művelet, pl. File esetén a Close() művelet Dispose()-t hív

42. Mutasson példát egy az IDisposable interfészt implementáló osztály helyes használatára!

```
ResourceWrapper r1 = new ResourceWrapper();
try
{
// r1 objektum használata
r1.DoSomething();
}
finally
{
// null feltétel ellenőrzés
if (r1 != null) r1.Dispose();
}
```

Vagy:

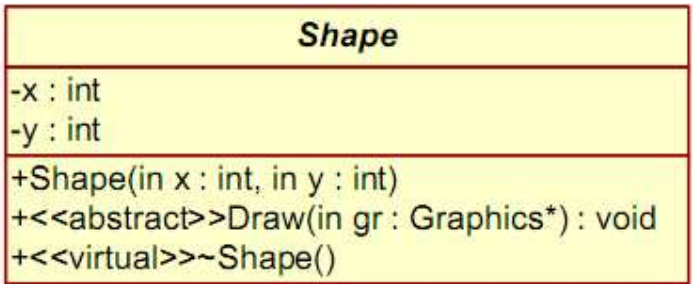
```
using (ResourceWrapper r1 = new ResourceWrapper())
{
// r1 objektum használata
r1.DoSomething();
} // Kilépünk a using blokkból: meghívódik r1-re a Dispose!
```

Szoftvertervezés

43. Mi az UML?

- Modellező nyelv
- Grafikus: szemléletes
- Szabványos: fejlesztők közös nyelve
- NEM mondja meg, hogyan kell objektumorientáltan fejleszteni, csak egy NYELV, JELÖLÉSRENDSZER
- Abban is sokat segít: le kellene dokumentálni valamit (pl. komponsek telepítése): hogyan induljunk el...
- Objektumorientált szemléletmódot támogatja:
- A valóságot objektumokkal, osztályokkal ragadjuk meg
- A szoftverrendszert egymással együttműködő objektumok valósítják meg

44. UML osztálydiagram, osztályok közötti kapcsolatok

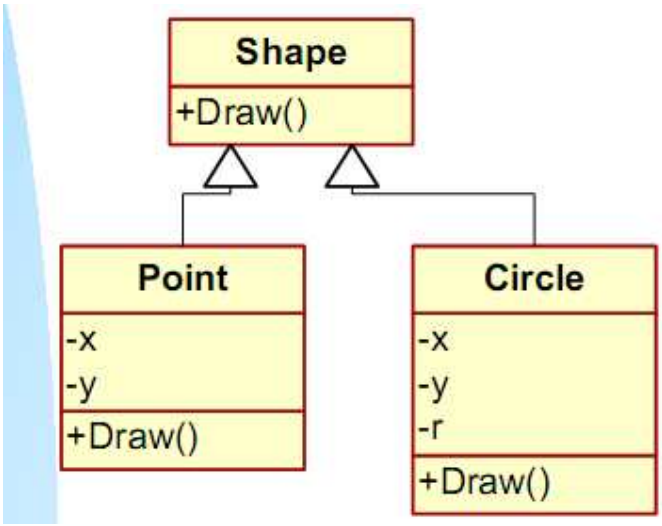


Felosztás:
- Legfelül osztálynév
- Alatta tagváltozók (UML-ben attribútumnak nevezik)
- Alatta műveletek
- Láthatóság

-: privát
+: publikus
#: protected

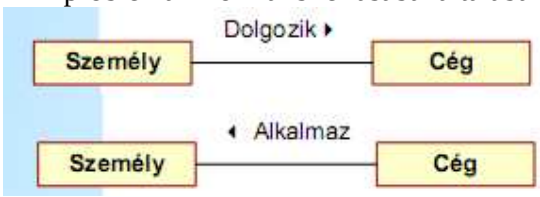
- Sztereotípiák
- Az UML elemek kiterjesztésére szolgál: << >> között bármilyen szöveg megadható, pl. osztálynév, műveletnév, tagváltozónév, paraméter, stb. esetén

I. Általánosítás, specializáció
- A speciálisabb osztály rendelkezik az általánosabb összes műveletét és tagváltozóját
- A programozási nyelvekben örökléssel lehet megvalósítani



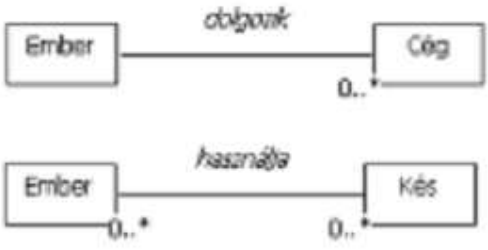
II. asszociáció

- Mindig kommunikációt jelent az osztályok objektumai között.
- Egy adott osztály igénybe szeretné venni egy másik osztály szolgáltatásait.
- A probléma informális leírásábanáltalában igeként jelenik meg:



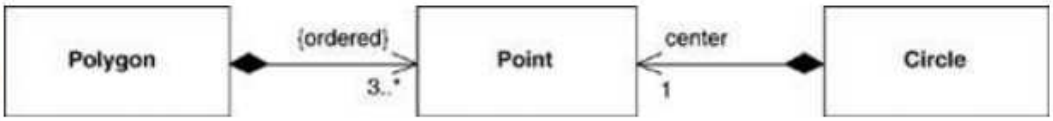
Multiplicitás

- 0..1 (0 vagy egy)
- 1 (egy)
- 0..* (nulla vagy több)
- * (nulla vagy több)
- 1..6



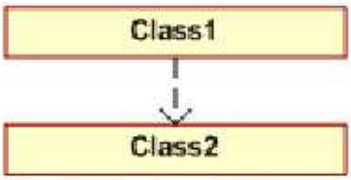
Tartalmazás - aggregáció

Tartalmazás – kompozíció



- Nem megosztott
- Élettartam

Függőség



Class 1 függ Class2-től: használja azt. Ha Class2 megváltozik, lehet, hogy a Class1-et is meg kell változtatni.

45. Példákon keresztül ismertesse az UML osztálydiagram leképezését C# nyelvi elemekre (osztály, osztályok közötti kapcsolatok)!

Adott egy probléma. A feladat: egy adott módszertant (modellező nyelv + folyamat) követve a szoftver termék elkészítése. Már láttuk a lépéseket (követelmény elemzés, analízis, tervezés, implementáció, …)A tervezés későbbi fázisában elkészül a rendszer részletes terve (detailed design), aminek eredménye a részletes terv (vagy implementációs)modell. Ezen a szinten a modellben szereplő bizonyos elemek (pl. osztályok) egyértelműen leképezhetők az adott alrendszer (SW komponens) implementációjául választott programozási nyelv elemeire.
Ha jó a modellező eszközünk: le tudja generálni az osztályok vázát (pl. C++, Java, C# osztályok)

Osztályok leképezése (egyszerű)
- UML osztály -> osztály
- UML attribútum -> attribútum/tagváltozó
- UML művelet -> művelet/metódus

Specializáció:

```
public class Base
{ };
public class Derived: Base
{ };
```

Asszociáció
Mindig kommunikációt jelent az osztályok objektumai között. Egy adott osztályigénybe szeretné venni egy másik osztály szolgáltatásait.

```
class Application
{
WindowManager* windowManager;
};
```

A kliens osztály tartalmaz egy pointert a kiszolgáló osztályra. Ezen a referencián keresztül tudja a kliens osztály objektuma igénybe venni a kiszolgáló objektum szolgáltatását, vagyis meghívni annak műveleteit. Ennek a tagváltozónak neve a szerepkör (role) neve lesz.

N multipcliitású:
Pl, nem egyértelmű

```
class WindowManager
{
List<Window> windows;
};
```

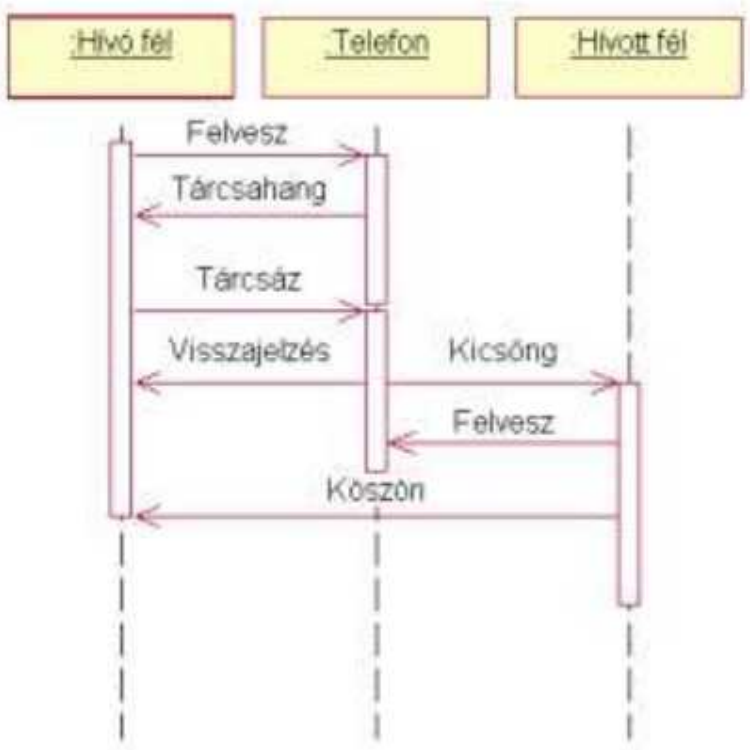
III. Aggregáció (tartalmazás, rész-egész viszony)
A, Megosztott tartalmazás
A kódban ugyanúgy jelenik meg, mint az asszociáció.

B, Kompozíció
A tartalmazott és a tartalmazó élettartama összefügg: a tartalmazó megszűnésekor a tartalmazott objektum is megszűnik.

```
class Rect
{
Point points[4];
};
```

46. Ismertesse az UML szekvenciadiagramot példa segítségével!

Objektumok egymás közötti kommunikációjának leírásárszolgál
A függőleges tengely mentén telik az idő
A vízszintes tengely mentén az egymással kölcsönhatásban levő példányok vannak
Egy osztálynak akár több példánya is megjelenhet egy diagramon belül objektumpéldányok



47. Ismertesse a Document-View architektúrát! Adja meg a minta osztály és szekvenciadiagramját!

Az alkalmazások többsége adatokat jelenít meg, melyet a felhasználó módosíthat.
Alapigazság: ne keverjük bele a GUI-ba az alkalmazáslogikát
- Rettentő hosszú, áttekinthetetlen, nehezen karbantartható form/windows forráskód
- Az alkalmazáslogika nem újrafelhasználható
- Az alkalmazáslogikára nem lehet automatizált teszteket írni

Válasszuk külön az adatok kezeléséért felelős kódot az adatok megjelenítéséért felelős kódtól.
Egy lehetséges megoldás a Document-view architektúra

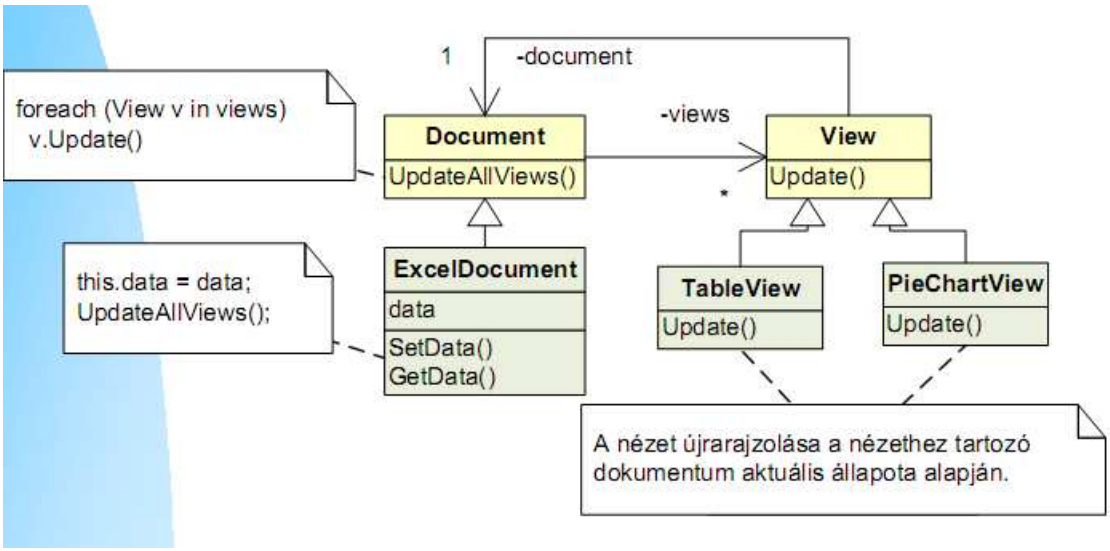
Document (dokumentum)
- Feladata az adatok tárolása, menedzselése. Modellnek is szokás nevezni.

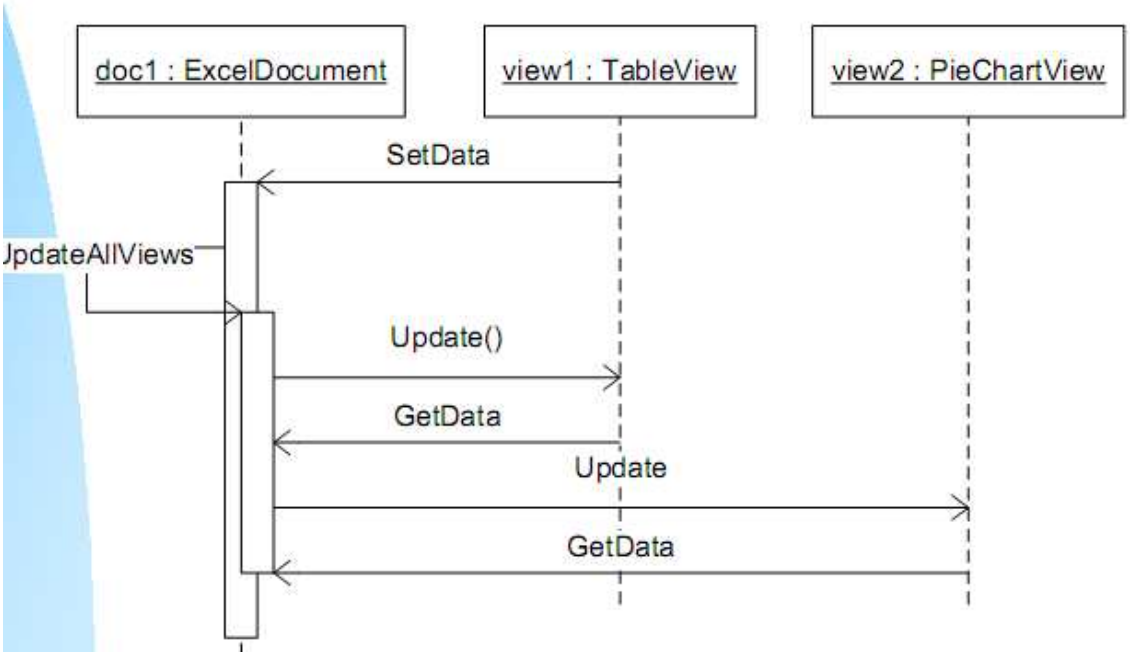
- Olyan osztály(ok), melyek az adatokat tagváltozóikban tárolják, és olyan tagfüggvényekkel rendelkeznek, melyek kezelik ezeket az adatokat (pl. Load, Save), és elérhetővé teszik más osztályok számára (pl. a View részére)

View (nézet)
- Feladata az adatok megjelenítése a dokumentum adatai alapján és a felhasználói interakciók kezelése (pl. menük, egér, billentyűzet). A felhasználói interakciók során általában a nézet a dokumentum tartalmát módosítja
- A view általában egy ablakként, vagy tabfülként jelenik meg a kliensalkalmazásokban

Támogatja a következőket
- Több dokumentum egyidejű megnyitása (pl. Firefox tabok, MS Word dokumentumok)
- Egy dokumentumhoz több és többféle nézet kapcsolódhat
  (pl. Excel, de általában a View/New Window menüvel)

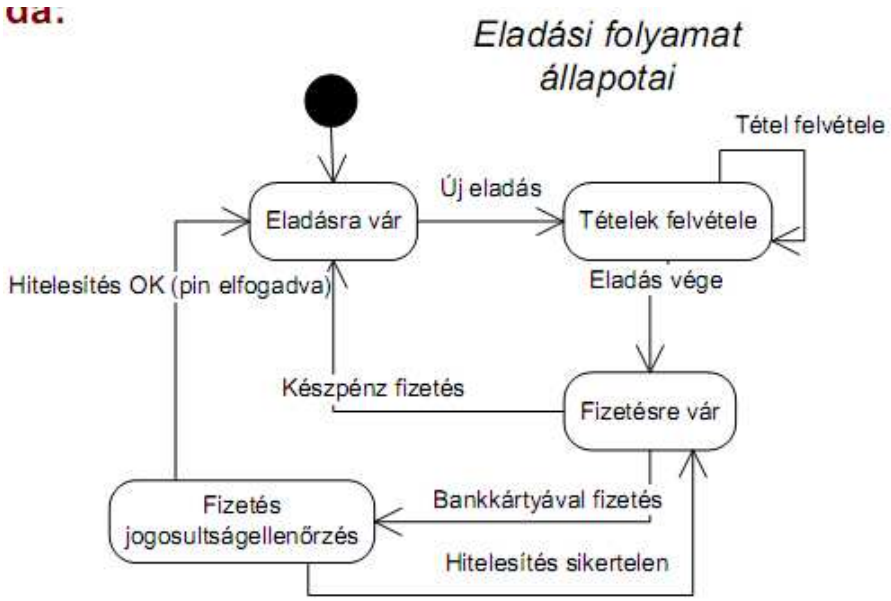
A jó megoldás
- Emeljük ki az adatokat és az azon értelmezett műveleteket egy osztályba, ez lesz a dokumentum (vagy modell)
- A dokumentumhoz különböző view-kat lehet beregisztrálni
- Ha valamelyik view megváltoztatja a dokumentum adatait, a dokumentum értesíti az összes beregisztrált view-t a változásról.
- Az értesítés hatására a view lekérdezi a dokumentum állapotát és frissíti magát
- A dokumentum csak egy közös View interfészen/ősosztályon keresztül tárolja a beregisztrált view-kat (nem függ az egyes típusoktól).




doc1 : ExcelDocument
view1 : TableView
view2 : PieChartView
SetData
UpdateAllViews
Update()
GetData
Update
GetData

48. Ismertesse az UML állapotdiagramot példa segítségével!

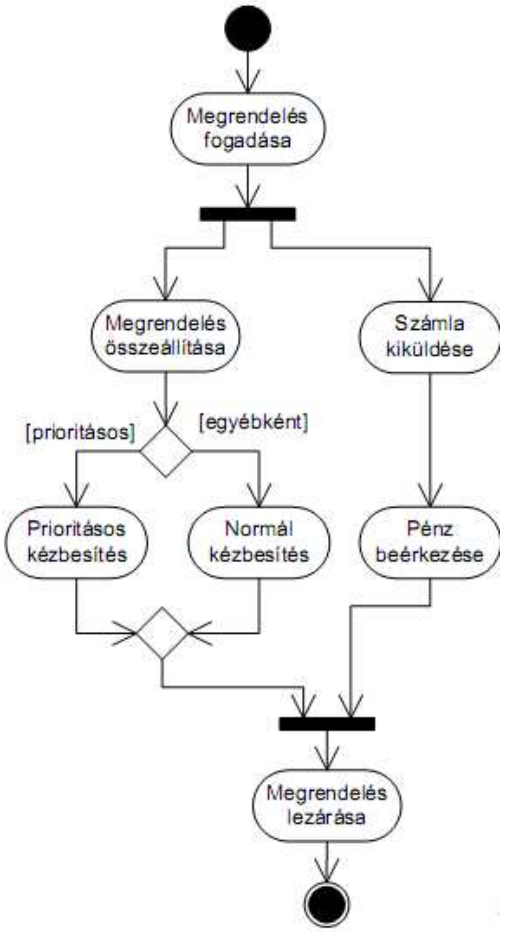
Objektum/alrendszer dinamikus viselkedését írja le
Csak akkor célszerű, ha bonyolult az objektum/alrendszer viselkedése, működése
Állapotok, események/feltételek hatására állapotátmenetek
Pénztárgép példa:



49. Ismertesse az UML tevékenységdiagramot példa segítségével!

Folyamat, algoritmus leírására
- Tipikusan üzleti folyamatok, workflowk leírására
- Egyszerű, mindenki megérti

Elemek
- Tevékenység (activity)
- Átmenet (transition)
- Feltétel (decision)
- Elágazás (fork)
- Csatlakozás (join)
- Kiinduló és záró állapot


Megrendelés fogadása
Megrendelés összeállítása
Számla kiküldése
[prioritásos]
[egyébként]
Prioritásos kézbesítés
Normál kézbesítés
Pénz beérkezése
Megrendelés lezárása

Többszálú alkalmazások fejlesztése

50. Folyamat és szál fogalma, jellemzőik.

Folyamat:
Alkalmazás (program) != folyamat
A folyamat a program egy betöltött példánya
Mi tartozik a folyamathoz
- Egy saját címtartomány(private address space)
- Rendszererőforrások
- Legalább egy szál (thread) fut

Minek az egysége a folyamat?
- A folyamat alapvetően VÉDELMI egység!
- A futási/ütemezési egység a szál
- A folyamatok egymástól elszigeteltek
- Védettek egymástól
- Az OS védett a felhasználói folyamtoktól
- Egy folyamat megszűntetésének nincs hatása a többire/OS-r

Mi védett? A memória!
- Minden folyamat saját kb. 2 GB virtuális címtartományt kap (32 bites OS esetén)
- A teljes címtartomány egy 32 bites OS alatt 232 (4 GB), de az OS kb. 2 GB-ot fenntart magának
- A virtuális címeket a memóriamenedzser fizikai címekre képezi le (RAM)

Nem áll automatikusan rendelkezésre: használat előtt foglalni kell!
- C/C++ globális/statikus változók – A folyamat indulásakor allokálódik hely
- Lokális változók – A Stacken allokálódik hely
- malloc, new – Dinamikusan allokálunk helyet

Folyamatok „programozása”
- API
    - CreateProcess(...) – elindít egy folyamatot, többek között egy futtatható fájl útvonala a paraméter

.NET
- System.Diagnostics.Process osztály
    - Start() – elindít egy folyamatot
    - Kill() – terminálja a folyamatot
- Process.GetCurrentProcess() – aktuális process elérése

A szál fogalma
- A szál (thread) a futási/ütemezési egység
- Az eddigi alkalmazásaink egyszálúak voltak
- A folyamat elindulásakor létrejött a folyamat fő szála (main thread)
- Ebből indíthatunk újabb szálakat, ekkor az alkalmazásunk többszálú (multithreaded) lesz.
- A szál egy folyamaton belüli feladatként fogható fel

Az OS a szálakat ütemezi
- Kiválaszt egyet a futásra kész szálak közül, és a processzort hozzá rendeli
- Egy CPU-n egyidőben egy szál futhat (a processzormagok száma számít)

Ütemezési típusok
 Nem preemptív: egy szál csak akkor kaphat futási jogot, ha önként lemond futási jogáról. Ilyen pl. a Windows 3.1. Kiéheztetés!
 Preemptív: az ütemező egy idő után elveszi a futási jogot a száltól, ha nagyobb prioritású szál futásra kész, vagy lejárt a szál időszelete (n*10 millisec). A legtöbb modern OS ilyen. Egy CPU esetén is látszólagpárhuzamosan futnak a szálak.
Szálak állapota
 Futó
 Felfüggesztett
 I/O műveletre vár

A folyamatok elszigeteltek: a folyamatok közötti kommunikáció nehéz
A szálak egy adott folyamaton belül egy címtartományban vannak, így „könnyen” kommunikálnak
 A C++ a globális, statikus, dinamikus változók közösek
 Minden szál saját stackkel rendelkezik
A szál lokális változóit csak az adott szál látja

51. Szálkezelés C# nyelven: szál indítása, paraméterezett szálindítás, előtér- és háttérszálak, szál altatása,

Szál indítása:

```
Thread t = new Thread( new ThreadStart(WriteY) );
t.Start();
```

Paraméterezett szál indítása:

```
Thread t = new Thread(new ParameterizedThreadStart(WriteAny) );
…..
static void WriteAny(object param)
{
while (true) Console.Write(param);
}
```

Előtérszál:

- Alapértelmezetten így indul
- Egy processz csak akkor lép ki, ha minden előtérszál befejezte a futását.

Háttérszál:

```
t.IsBackground = true;
```

- Nem várja meg, amíg végez

Szál altatása:

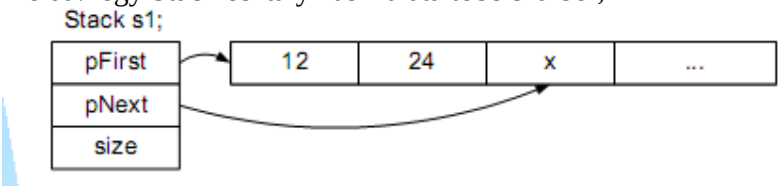
- Thread.Sleep(int millisec) és
- Thread.Sleep(TimeSpan ts) statikus művelet,elaltatja a hívó szálat, nem foglal CPU időt
- Kloroform ☺

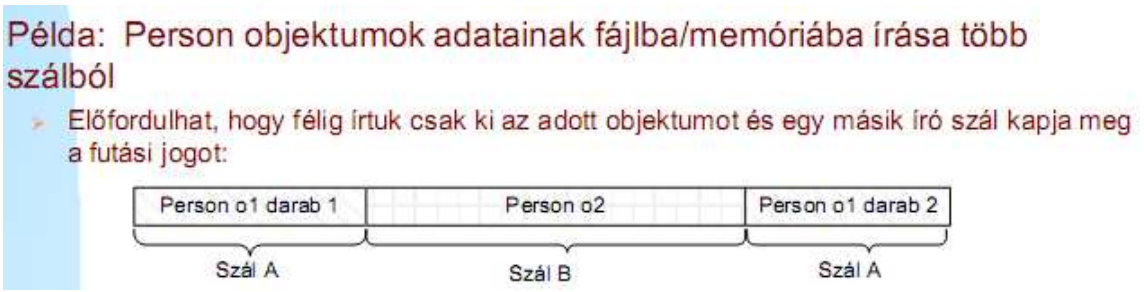
52. Néhány példa kölcsönös kizárás problémájára

Környezet: megosztott erőforráshoz több szál fér hozzá
Megosztott erőforrás: memória (változók, objektumok), fájl, stb.

Meg kell őrizni a konzisztenciát. Egy CPU esetén sem tudjuk, mikor veszi el az ütemező a szál futási jogát!

Példa: egy Stack osztály használata több szálból,





Biztosítanunk kell, hogy a megosztott erőforráshoz egy időben csak egy szál férjen hozzá.

53. Milyen fontosabb zárolási konstrukciókat támogat a .NET Keretrendszer? Jellemezze őket egy-két mondatban!

Név	Cél	Folyamatok között is?	Sebesség
lock C# utasítás (Monitor.Enter/Monitor.Leave)	Biztosítja, hogy egy adott erőforráshoz/kódrészlethez egy időben csak egy szál férhet hozzá.	nem	gyors
Mutex	Mint a lock, de **folyamatok között is**. Pl. annak megoldására, hogy egy alkalmazásból csak egy példány indulhasson.	igen	közepes
Semaphore	Mint a Mutex, de nem egy, hanem **max. N hozzáférést** engedélyez.	igen	közepes
ReaderWriterLockSlim	Sok olvasóra optimalizált megoldás. Egyszerre több olvasó is hozzáférhet az erőforráshoz, de íróból csak egy (illetve az író kizárja az olvasókat is). Pl. ritkán módosított cache megvalósítása.	nem	közepes

54. Példa segítségével magyarázza el a lock használatát!

```
class ThreadSafeClass
{
        static bool done; // Közös erőforrás
        static object syncObject = new object();
        static void Main()
        {
```

```
            new Thread(Run).Start();
            new Thread(Run).Start();
        }
static void Run()
        {
        lock (syncObject)
        {
            if (!done) { done = true; Console.WriteLine("Done");}
        }
        }
        }
```

Egy objektum paramétert kell neki adni
Megvizsgálja, hogy zárolva van-e az objektum
 Ha nincs, atomi módon zárolja, majd tovább fut (a lock tulajdonképpen egy oszthatatlan test_and_set)
 Ha igen, vár (blokkolja a hívó szálat), amíg fel nem szabadul
A várakozás nem használ CPU időt
A várakozó szálak egy sorba kerülnek, „érkezési sorrendben” kapnak hozzáférési jogot
A lock blokkból kilépéskor oldja az objektumon levő zárat

55. Adja meg a szálbiztos (thread-safe) szál fogalmát!

Egy osztály szálbiztos, ha úgy lett megírva, hogy többszálú környezetben is biztonságosan használható.
 Maga garantálja a konzisztenciát
 A felhasználás során már nem kell zárolni...
Tipikus vizsgafeladat: tegyük szálbiztossá az alábbi xxx osztályt!

```
readonly int size;
int current = 0;
T[] items;
object syncObject = new object();
public void Push(T item) {
        lock(syncObject) {
        items[current++] = item;
                        }
}
public T Pop() {
        lock(syncObject) {
        return items[current--];
                        }
```

Interocked műveletek:
Long sum;
Interlocked.Increment(ref sum);
Interlocked.Decrement(ref sum);

56. Mutasson példát szál kiléptetésére (bool flaggel)!

```
static void Run()
{
int counter = 0;
```

```
while (!exit)
{
        Thread.Sleep(300);
        Console.WriteLine(
        counter++.ToString());
}
Console.WriteLine("Exiting from worker thread.");
}
```

57. Milyen jelzésre alkalmas szinkronizációs konstrukciókat támogat a .NET? Ismertesse az AutoResetEvent fogalmát, és használatának főbb lépéseit! Ismertesse a ManualResetEvent fogalmát, és használatának főbb lépéseit!

WaitHandle
Olyan osztályok őse amelyek objektumaira várakozni lehet
Az AutoResetEvent ésManualResetEvent eseményre való várakozást tesz lehetővé

AutoresetEvent:

- Set művelet:
- Jelzettbe állítja az event objektumot, így elenged egy várakozó objektumot. Ha nincs várakozó objektum, jelzett marad. Azt nem tartja nyilván, hányszor volt Set hívás (a több is egynek számít).

WaitOne művelet:

- Ha az event objektum nem jelzett, várakozik (a CPU-t nem terheli)
- Ha az event objektum jelzett: Tovább fut
- AutoResetEvent esetén automatikusan nem jelzettbe állítja az event objektumot (ManualResetEvent esetén nem)

Reset művelet:

- Reseteli (nem jelzett állapotba állítja) az event objektumot

```
class SimpleEventDemo {
static EventWaitHandle wh = new AutoResetEvent (false);
static void Main() {
        new Thread (Run).Start();
        Thread.Sleep (1000); // Várjunk egy kicsit
        wh.Set(); // Ébresztő
                }
static void Run() {
        Console.WriteLine ("Várunk értesítésre ...");
        wh.WaitOne(); // Várakozás
        Console.WriteLine ("Az értesítés megérkezett.");
}
}
```

ManualResetEvent:

- Az AutoResetEvent „ellentéte”
- „Kapuként” funkcionál: ha jelzett, mindenki mehet, ha nem jelzett, mindenki vár.

58. Ismertesse a WaitSleepJoin szálállapotot! Milyen összefüggésben van ez a Thread.Interrupt() művelettel? Miben különbözik a Thread.Interrupt() és a Thread.Abort()?

WaitSleepJoin állapot
- Blokkolt állapot
- Sleep, Join, lock, WaitOne, WaitAny, WaitAll utasításokhatására

Mind blokkol. Meddig? Négy módon léphet ki blokkolt állapotból:
- A várakozási feltétel teljesül
- Timeout lejár (ha adtunk meg)
- A várakozás a Thread.Interrupt hívásával megszakításra kerül
- A várakozás a Thread.Abort hívásával megszakításra kerül

Thread.Interrupt használata
- A szál, amire meghívjuk, kap egy ThreadInterruptedException-t, de csak ha WaitSleepJoin állapotban volt
- Vagyis a while(true); -ből nem fogja kiléptetni a szálat

Pl:
```
catch (ThreadInterruptedException) { }
        Console.WriteLine("Exiting from worker thread.");
```

Thread.Abort:

Thread.Abort() művelet
Hasonlít a Thread.Interrupt-ra, de
- A szál ThreadAbortException-t kap
- Nem csak WaitSleepJoin állapotban, hanem bármilyen állapotban lehet a szál
- Vagyis a while(true); -ból is kilépteti!
- A finally blokkok azért lefutnak (persze ebben lehet még egy végtelen ciklus)

Inkább ne, max programkilépéskor

59. Ismertesse a ThreadPool fogalmát!

- Nagyszámú párhuzamos kérés esetén túl sok szál fut
- Folyamatonként ~100-nál többet nem célszerű, a szálak közötti sok váltás miatt
- A szál indítása viszonylag költséges

Thread-pool alkamazása az igazi megoldás
- Előre elindítunk néhány szálat
- A kiszolgálás során ezekből allokálunk
- A kiszolgálás végeztével a szál visszakerül a poolba, új kérést szolgálhat ki
- Ha nincs szabad szál a poolban:
    - Új szálat indítunk és teszünk a poolba,

 Ha már túl sok szál van (néhányszor tíz) - blokkoljuk a hívót míg nem szabadul fel szál

 Mikor célszerű a ThreadPool használata?

 Csak rövid műveleteket futtatunk. Ne blokkoljuk a ThreadPool szálakat (máskülönben hamar kimerítjük a ThreadPoolt) .

60. Ismertesse a holtpont (deadlock) fogalmát és elkerülésének legjellemzőbb módját!

Holtpont: Kettő (vagy több) szál kölcsönösen egymásra vár. A probléma oka: foglalva várakozás!
Detektálható: Megfelelő irányított gráfot kell detektálni. Nekünk kell megírni. Nem szoktuk. Inkább megpróbáljuk elkerülni.

Elkerülése:

 Az erőforrásokat minden szálban ugyanabban a sorrendben foglaljuk le

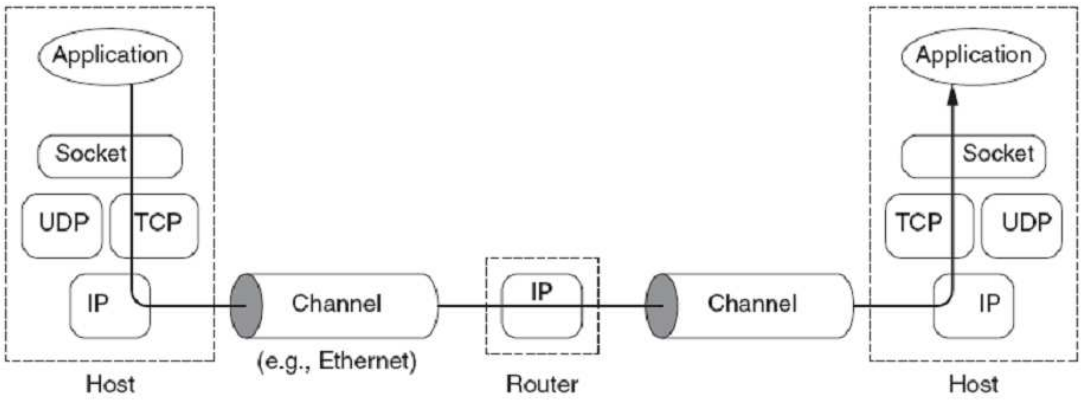
 Nem mindig tehető meg

Alternatív megoldás: Adott időkorlátig várakozunk az erőforrásra (timeout alkalmazása).

Hálózatprogramozás .NET környezetben

61. Ismertesse a TCP/IP protokoll családot. Mik az előnyei a rétegelt szerkezetnek?

A protokoll családok protokolljait célszerű rétegekbe (layer) szervezni. Ez a TCP/IP esetén is így történt. A következő ábra két host és egy közbeiktatott router kommunikációján keresztül mutatja be az adat áramlását az egyes rétegek között. TCP/IP esetén a legalsó réteg maga a kommunikációs csatorna, ami lehet például Ethernet vagy egy telefonos modemes kapcsolat. Az e felett lévő réget a hálózati réteg (network layer), amely a csomagok célbajuttatásáért felelős. A TCP/IP protokollcsaládban az egyetlen protokoll, ami ehhez a réteghez tartozik az IP protokoll. Az IP protokollnak köszönhető, hogy a két host közötti kommunikáció a host-ok számára teljesen átlátszó abból a szempontból, hogy nem kell azzal foglalkozniuk, hogy milyen router-eken és csatornákon keresztül történik a csomagok átvitele. Ők egy egyszerű host-to-host esetet látnak.



IP (hálózati réteg):

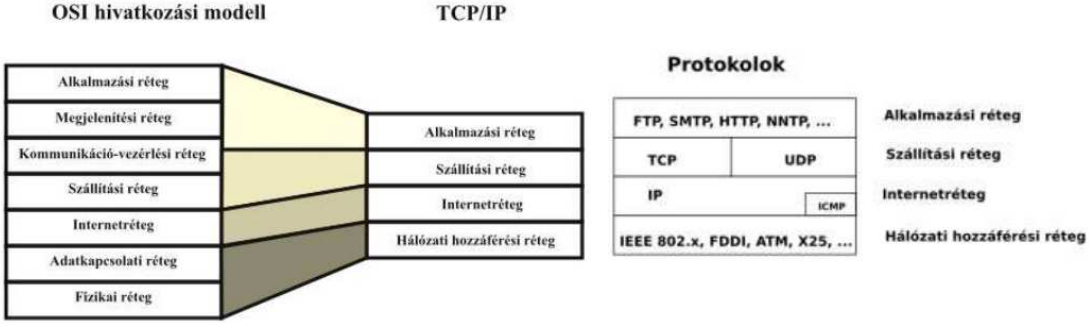
- IP protokoll minden csomagot egymástól függetlenül kezel és kézbesít
- A csomagok kézbesítését azonban nem garantálja a protokoll

- csomagok elveszhetnek, duplikálódhatnak, illetve a csomagok sorrendje is megváltozhat

Szállítási réteg (transport protocol)

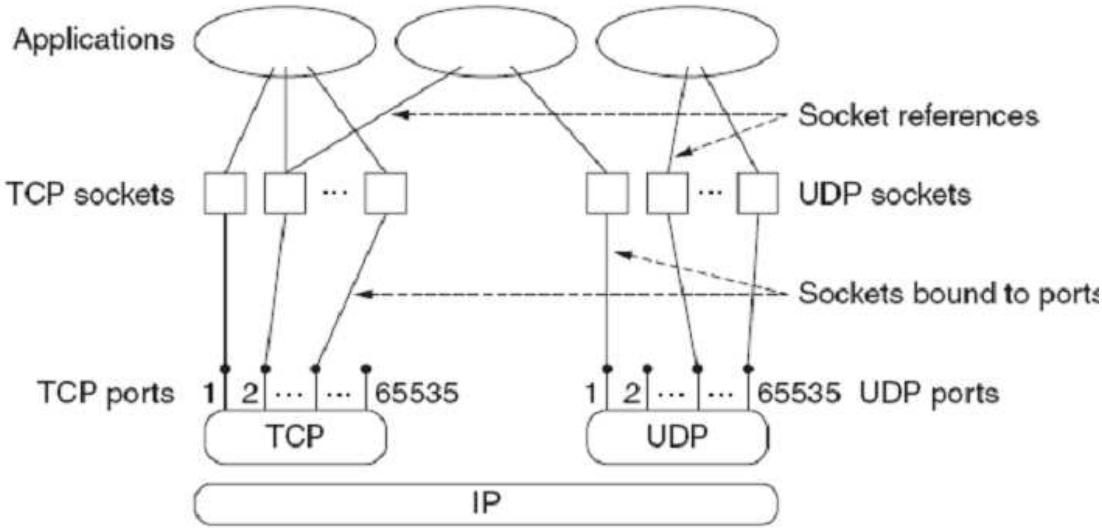
- TCP/UDP
- Míg a hálózati réteg feladata az volt, hogy továbbítsa a csomagokat két host között, addig a szállítási rétegben a címek a host címén kívül egy portszámot is tartalmaznak. Azt szoktuk mondani, hogy a szállítási réteg nem host-okat, hanem alkalmazásokat köt össze, hiszen az alkalmazások választhatnak, hogy a host melyik portját használják

- TCP egy megbízható byte-folyamot épít ki a két host között, kapcsolat-orientált protokoll
- Az UDP ezzel ellentétben az IP réteget csak az újfajta címzési lehetőséggel (t.i.
- portok) egészíti ki, és nem gondoskodik az üzenetek garantált kézbesítéséről.

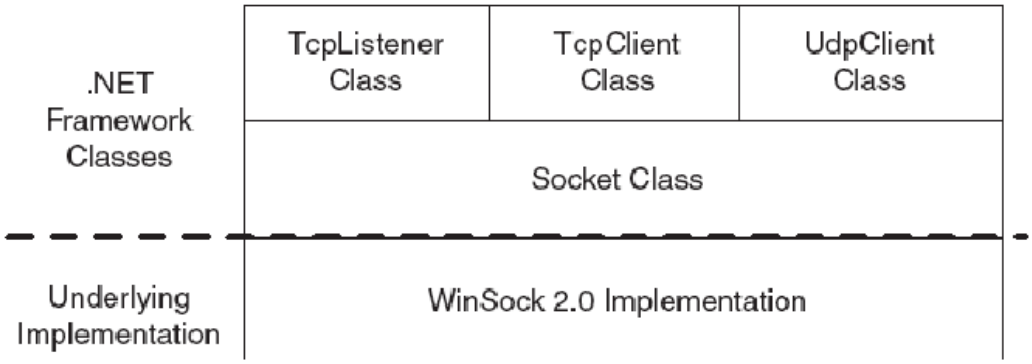


62. Ismertesse a socket fogalmát. Milyen socket implementációk léteznek .NET-ben?

A socket egy olyan dolog, amin keresztül a file-ok írásához és olvasásához hasonló módon adatok küldhetők egy hálózatba vagy adatok fogadhatóak onnan. A TCP/IP világában a két leggyakrabban használt socket típus a stream socket és a datagram socket. A stream socket a TCP és az IP protokollokra épül és egy megbízható byte-folyam alapú szolgáltatást nyújt. A datagram socket pedig az UDP és az IP protokollokra épülve egy nem megbízható, kapcsolat nélküli kommunikációs modellt támogat. Egy TCP/IP socket-ot egyértelműen meghatároz az Internet cím, a szállítási protokoll (TCP vagy UDP), illetve a portszám.



Implementációk:
A jelenlegi verzió a WinSock 2.0. A WinSock interfész kis eltérésektől eltekintve azonos a kezdeti Berkeley sockets C interfészével.



63. Milyen lépésekből áll egy TCP alapú kliens-szerver kommunikáció? Ez milyen osztályokkal és hogyan valósítható meg .NET-ben?

1. Létrehozunk egy kliens oldali TcpClient-et, aminek megadjuk a szerver címét egy IPAddress példány formájában.
2. Szerver oldalon létrehozunk egy TcpListenert egy adott porton.
3. A kliens megpróbál csatlakozni a szerverhez.
4. A szerver elfogadja a csatlakozási kérelmet, és a kapcsolathoz létrehoz egy szerver oldali TcpClient példányt.
5. A kommunikáció a két fél között a két TcpClient példányon keresztül zajlik. Mind a kettőhöz tartozik egy NetworkStream, ami írható és olvasható, akár egy file.
6. Lezárjuk a kliens és a szerver oldali TcpClient-eket.

a TcpClient és a TcpListener.

A kliens alkalmazásban létrehozunk egy TcpClient-et. Ha a konstruktorban megadjuk a szerver címét, a kliens implicit kapcsolódik a szerverhez. Ellenkező esetben nekünk kell meghívni a Connect() metódust a kliens socket objektumon. A konstruktorban opcionálisan

megadható a lokális portszám is. A szervertől kapott válasz olvasásakor érdemes megfigyelni, hogy nem csak egy olvasás történik, hanem addig olvassuk a NetworkStream-et, amíg vissza nem kapunk annyi darab karaktert, amennyit elküldtünk. Erre azért van szükség – és ennek figyelmen kívül hagyása gyakori problémák forrása – hogy a TCP nem őrzi meg a csomaghatárokat. Azaz, egyáltalán nem biztos, hogy egy üzenetben fogjuk megkapni kliens oldalon azt, amit a szerver egy üzenetben küldött el.

A szerverhez történő kapcsolódás során SocketException, a NetworkStream-en történő kommunikáció során pedig IOException típusú kivételekre számíthatunk.
Végül a finally blokkban zárjuk a NetworkStream-et és a socket-ot.

64. Milyen lépésekből áll egy UPD alapú kliens-szerver kommunikáció? Ez milyen osztályokkal és hogyan valósítható meg .NET-ben?

Az UDP socket-oknak nem kell kapcsolódniuk egymáshoz a kommunikáció megkezdése előtt TCP: telefon, UDP pedig olyan, mint egy levél feladása és fogadása. A TCP és az UDP közötti másik fontos eltérés, hogy míg a TCP nem őrizte meg a csomaghatárokat, addig az UDP megőrzi. Azaz, amit egy csomagban küld el az egyik oldal, azt egy csomagban tudja fogadni a másik oldal.
A .NET Framework UdpClient osztályát használhatjuk UDP socket-ek esetén. Ez az osztály használható mind kliens, mind szerver oldalon. Segítségével küldhetünk és fogadhatunk is csomagokat.
Az UdpClient létrehozásakor nem kell megadni a szerver címét, mivel nem épül fel kapcsolat a kliens és a szerver között. A konstruktorban opcionálisan megadható a lokális portszám is. Ha nem adunk meg ilyet,automatikusan választ egy kliens oldali szabad portot. Egy UdpClient-tel több különböző távoli socket-telis tudunk kommunikálni. Az UdpClient-nek is van egy Connect() metódusa, de ezzel csak az alapértelmezett címzett állítható be. A csomagok elküldése a Send() metódussal történik, melynek paraméterként meg kell adni a csomagot egy byte tömbként, illetve a címzettet (ha nincs címzett megadva, akkor azt korábban a Connect()-tel be kellett, hogy állítsuk). A válasz fogadása itt egy lépésben történik a Receive() metódussal, mivel az UDP megőrzi az üzenethatárokat. Egy üzenet fogadásakor a Receive() kimenő paramétereként megkapjuk a csomag küldőjének a címét is. Ha az elküldött vagy a válasz üzenet elveszne, a Receive() örökké várakozna. Ezért a Receive()-nek megadható egy time-out is.
Szerver oldalon is létrehozunk egy UdpClient példányt, aminek paraméterként megadjuk, hogy melyik portot kell figyelnie. A TCP-vel ellentétben azonban ezen a socket-en nem a kapcsolódási kérelmeket figyeljük (mivel erre nincs is szükség), hanem ezen zajlik a tényleges kommunikáció. Hasonlóan kell üzeneteket küldeni és fogadni, mint ahogy azt az UDP kliens esetén is tettük. A kliens üzenetek a Receive() hívással fogadhatóak, amely kimenő paraméterként visszaadják a küldő kliens címét is, innen tudja a szerver, hogy kinek kell visszaküldenie a választ.

65. Ismertesse a Socket .NET-es osztályt, és mutassa be az elemi műveleteit.

A .NET Framework Socket osztálya egy burkoló a WinSock implementáció köré. A TcpClient, TcpListener és UdpClient osztály is ezt a Socket osztályt használja (a Client tulajdonságukon keresztül le is kérhető).

A kliensalkalmazásban először létrehozunk egy Socket példányt. Ennek három paramétere van:
- Address family: IP protokoll esetén AddressFamily.InterNetwork. (A Socket osztály és a socket koncepció nem csak IP protokollt támogatja)
- Socket type: Stream vagy datagram típusú kommunikációt szeretnénk. TCP esetén SocketType.Stream, UDP esetén SocketType.Datagram-ot kell választanunk.
- Protocol type: TCP esetén ProtocolType.Tcp, UDP esetén pedig ProtocolType.Udp.

Socket() konstruktor	Socket létrehozása, a socket típus meghatározása.
Bind()	A socket kötése egy helyi porthoz.
Connect()	A socket kapcsolása egy távoli címhez.
Listen()	A hívás hatására a socket elkezdi figyelni a bejövő kliens kéréseket.
Accept()	Egy klienstől érkező kapcsolódási kérés elfogadása. Visszaad egy Socket példányt, amin keresztül kommunikálhatunk a másik féllel.
Send()	Adatok küldése a másik félnek.
Receive()	Adatok fogadása a másik féltől.
Close()	Kapcsolat bontása.

Socket-eknek számos tulajdonsága állítható be a Socket.SetSocketOption() metódus segítségével. Ilyenek például a buffer méretek, a time-out-ok, hogy a broadcast üzenetek engedélyezettek-e vagy sem, és még számos más. Egy példa a receive-timeout 3 mp-re állítására:

66. Hogyan készíthető többszálú szerver alkalmazás? Hogyan küldhetünk üzeneteket egyszerre több host-nak.

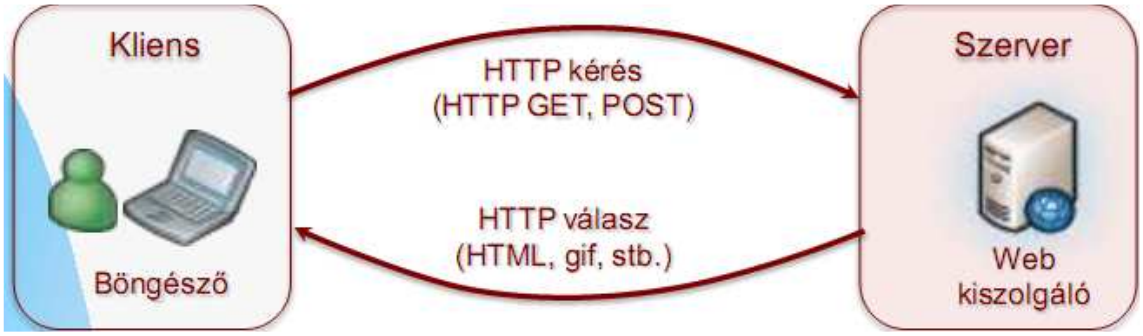
Többszálú szerver alkalmazások esetén alapvetően két lehetőség közül választhatunk:
- Kliens kérésenként létrehozunk egy szálat (thread-per-client),
- Vagy egy Thread-pool szálon hajtjuk végre a műveletet. Ebben az esetben a szálak száma korlátozott, és előre létrehozott szálakat használunk.

A szál létrehozásakor .NET-ben meg kell adni a szálfüggvényt, amelynek a szál indításakor egy paraméter is megadható. A szálfüggvényben kell implementálni egy kliens kiszolgálását. A függvénynek tehát célszerű paraméterként megkapnia azt a szerver oldali socket-et, amelyet az Accept() valamelyik változata hozott létre, és már össze van kötve a kliens socket-tel. Magának a szálfüggvénynek is megadhatjuk ezt a paramétert, de még szebb megoldás, ha egy külön osztályt készítünk a szálfüggvény köré, amely tagváltozóként tartalmazza a függvény összes paraméterét.

A többesküldés UDP esetén használható. Két lehetőségünk van:
- Broadcasting: IP cím: 255.255.255.255. A router-ek általában kiszűrik, így csak a lokális hálózaton küldi el mindenkinek.
- Multicasting: egy csoport címe adható meg címzettnek. A csoportcímeknek a 224.0.0.0 – 239.255.255.255 tartomány van fenntartva. Az üzenetet mindenki megkapja, aki tagja a csoportnak. Egy adott porton ülő UDP socket csatlakozni tud egy csoporthoz, és ki tud lépni onnan. A csoportba be- és kilépés szabályozható socket option-ök segítségével, illetve használható az UdpClient JoinMulticastGroup() és DropMulticastGroup() metódusa.

Webalkalmazások fejlesztése

67. Ismertesse a webalkalmazások architektúráját!



A kliens kiad egy HTTP kérést, amire a webkiszolgáló egy HTTP választ küld

- Pl. HTTP GET esetében az URL-ben megadja, mely szerver erőforrást (pl. html oldal) kéri
- A válasz tipikusan egy HTML oldal, vagy egy hivatkozott kép (pl. gif)
- A kliens oldali böngésző megjeleníti a HTML oldalt és letölti a szükséges hivatkozott kiegészítő fájlokat (pl. képek, stíluslapok)

A kliens is küldhet adatot: HTTP POST (pl. INPUT vezérlők tartalma egy HTML FORM-ban)

Mindig szigorú kérés-válasz: a kliens kérésére a szerver válaszol, a szerver sohasem kezdeményez!

A HTTP egy a TCP/IP-re épülő, szöveges protokoll

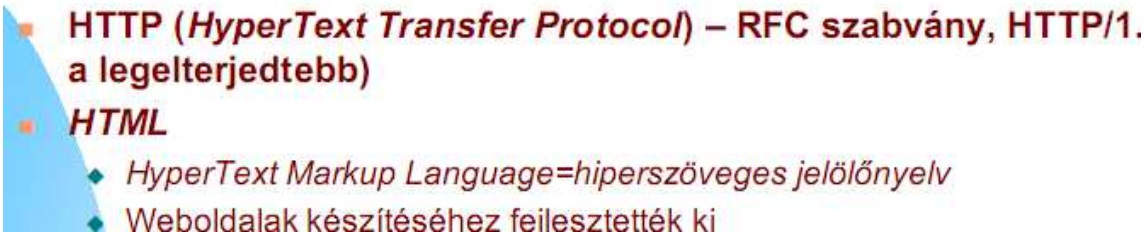
68. Ismertesse a HTML oldalak felépítésének alapjait!

Statikus oldal

a HTML utasításokat a szövegben < és > jelek közé kell zárni.

Egy-egy utasítás - HTML tag - hatását általában a záró utasításpárja szünteti meg, amely megegyezik a nyitó utasítással, csak a / jel vezeti be (természetesen a < és a > jelek között)

Bizonyos karakterek megjelenítésére escape szekvenciát kell használni az „&” karakter követően: pl. a „<” és „>” karakterek „foglaltak”, ezért ezek megjelenítésére az &lt és &gt használandók.

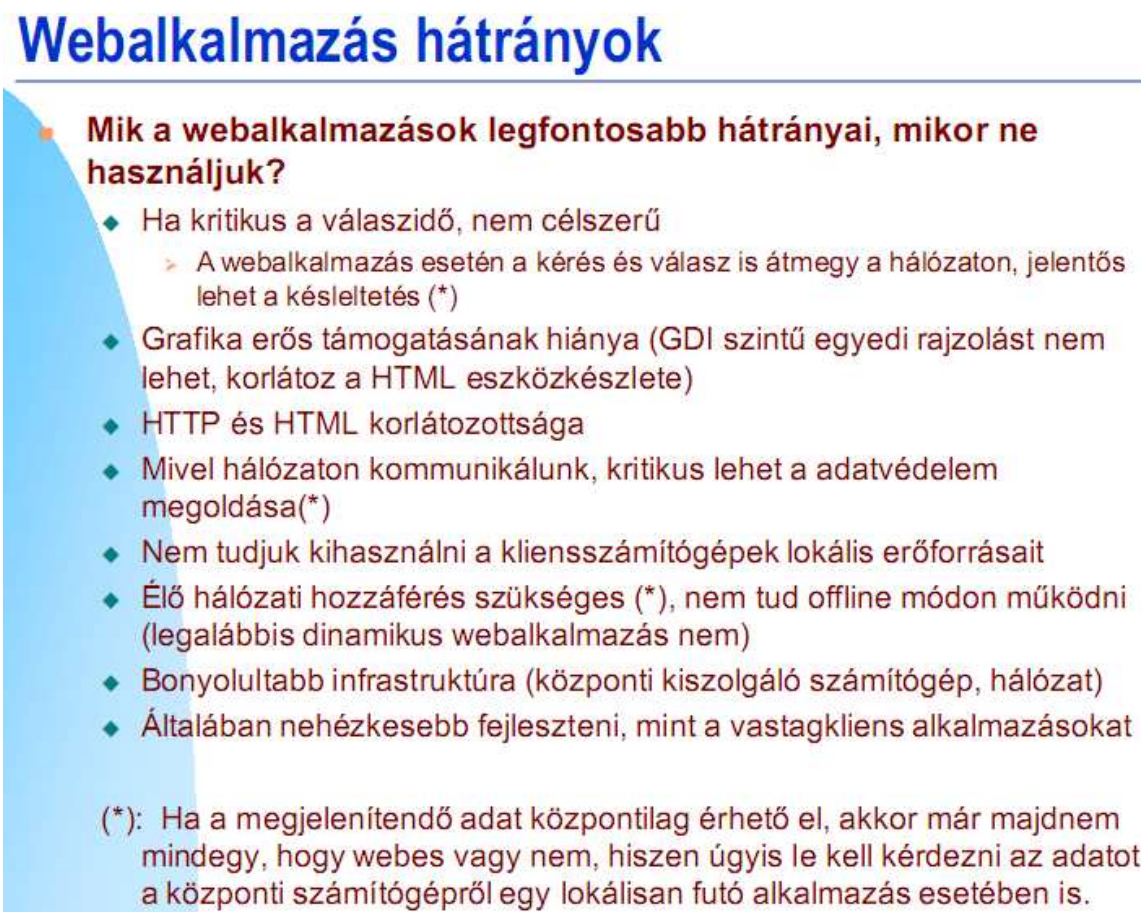


- **A HTML oldalak felépítése (az előző példa alapján)**
  - A **<!DOCTYPE>** elem a böngészőnek szól, a dokumentum sémáját adja meg, ami estünkben a 4.01-es szabványnak megfelelő HTML formátum
  - A **<html>** alatti **<head>** elem technikai és dokumentációs fejlécadatoka tartalmaz, melyeket az böngésző nem jelenít meg
  - A **<html>** alatti **<body>** elem tartalmazza a dokumentumtörzset, vagyis a megjelenítendő adatokat
- **Néhány fontosabb elemtípus**
  - **…**
    Hiperivatkozást tudunk létrehozni. A href megmondja, hogy hova kell ugrani, és a <a> között szereplő rész lesz a link szövege.
  - **<h1>..<h1>**
    Címsor típus, egyszerűen kiemelhetően vele bizonyos szövegek. 1-től 6-ig állítható be.
  - ****
    Kép, alternatív szöveggel.
  - **<table ...>** - táblázat

69. Hasonlítsa össze a webes és nem webes alkalmazásokat (előnyök, hátrányok – mikor érdemes „webesíteni” az alkalmazást és mikor nem)!

Webalkalmazás előnyök

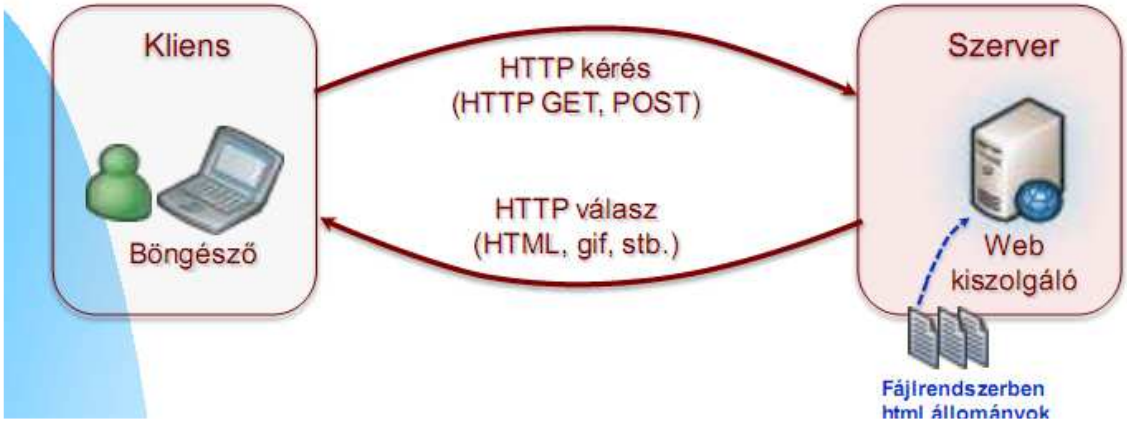
- **Mik a webalkalmazások előnyei, miért terjedtek el rohamosan?**
  - Egyszerű telepítés – csak a szerverre kell, a klienseken elég egy böngésző.
  - Könnyű karbantarthatóság (frissítések, javítások) – elég a szerveren frissíteni, a kliensekhez nem kell nyúlni. Így olcsóbb az üzemeltetés.
  - Kliensoldali platformfüggetlenség– elég egy kliensoldali böngésző, a platform (operációs rendszer) mindegy, hogy mi.
  - Egyéb
    - Szabványos távoli kommunikáció (HTTP)
    - Szabványos titkosítás és hitelesítés (HTTPS esetén titkos SSL vagy TLS protokoll)
    - Távoli adminisztráció lehetősége (elég a szervert, nem kell minden klienst)
    - Mobil és nem mobil kliensek is



70. Ismertesse az okoskliens (smart client) alkalmazások működésének lényegét és főbb jellemzőit!

 A klasszikus vastagkliens alkalmazásokat vértezi fel a webes alkalmazások előnyeivel
 A.NET támogatja
 A lényege, hogy a vastagkliens alkalmazást mindig egy központi helyről indítjuk (pl. egy weboldalba ágyazott hiperhivatkozással, vagy van egy hálózaton megosztott futtatható állományra mutató shortcuttal).
 A futtatókörnyezet ellenőrzi, hogy megvan-e lokálisan is a futtatható állomány és a hivatkozott DLL-ek. Ha nincs, letölti őket és elindítja az alkalmazást. Ha megvannak lokálisan, akkor előbb ellenőrzi, hogy megvan-e a legfrissebb verzió (EXE és DLL-ek is), és ha nincs, akkor az indulás előtt frissíti a lokális példányt. Ha nincs hálózati kapcsolat a verzióellenőrzéshez/frissítéshez, a lokális példány indul automatikusan)
 Előnyök
    - Hordozza a vastagkliens alkalmazások előnyeit (jobb felhasználói élmény, válaszidő, grafika, offline működés megoldható, stb.)
    - Ugyanolyan könnyű telepíteni, karbantartani, mint a webes alkalmazásokat
 Hátrányok
    - Szükség van a kliensen a támogató futtatókörnyezetre (pl. .NET), bár ez is tud automatikusan települni, már ha az operációs rendszer támogatja

71. Ismertesse a statikus web működését és főbb jellemzőit (definíció, kliens oldal, szerver oldal, ábra)!



 A tartalom állandó, nem tartalmaz szerver oldali logikát
 A kiszolgáló tipikusan a kliens által az URL-ben megadott HTML fájlt felolvassa a fájlrendszerből ,és változtatás nélkül elküldi a kliensnek
 Kliens oldali logikát (JavaScriptet) tartalmazhat
 Újabban: a ritkán változó tartalmú, de dinamikus alapokon nyugvó webalkalmazást is szokás statikusnak nevezni.
 Webkiszolgáló pl.: Apache, Microsoft IIS

72. Ismertesse a Javascript főbb jellemzőit és mutasson rá kódrészletet!

Jellemzők
 Az egyetlen, minden böngésző által támogatott programozási nyelv
 Kliens oldali szkript segítségével a böngészőben futtathatunk szkript kódot, amely előállíthatja, vagy módosíthatja az oldalt
 A böngészőben futó kód elérheti és módosíthatja a dokumentum tartalmát és a böngésző bizonyos beállításait is (pl. ablak mozgatás, új ablak megnyitás). A HTML dokumentum forráskódja tehát nem minden esetben egyezik meg a dokumentum objektum modelljének aktuális állapotával.
 Bizonyos műveletek csak a kliens oldalon kezdeményezhetők (szerver oldalon nem): pl. ablak átméretezése, új ablak nyitása, nyomtatás elindítása, jóváhagyó ablak megjelenítése, stb. Ezek megvalósítására csak a kliens oldali kód nyújt megoldást, pl. a Javascript

Egyszerű példa - Egy szövegdobozba bekérünk szöveget. Ha a felhasználó nem írt be semmit, jelezzük egy figyelmeztető ablakban, egyébként köszönjük meg.

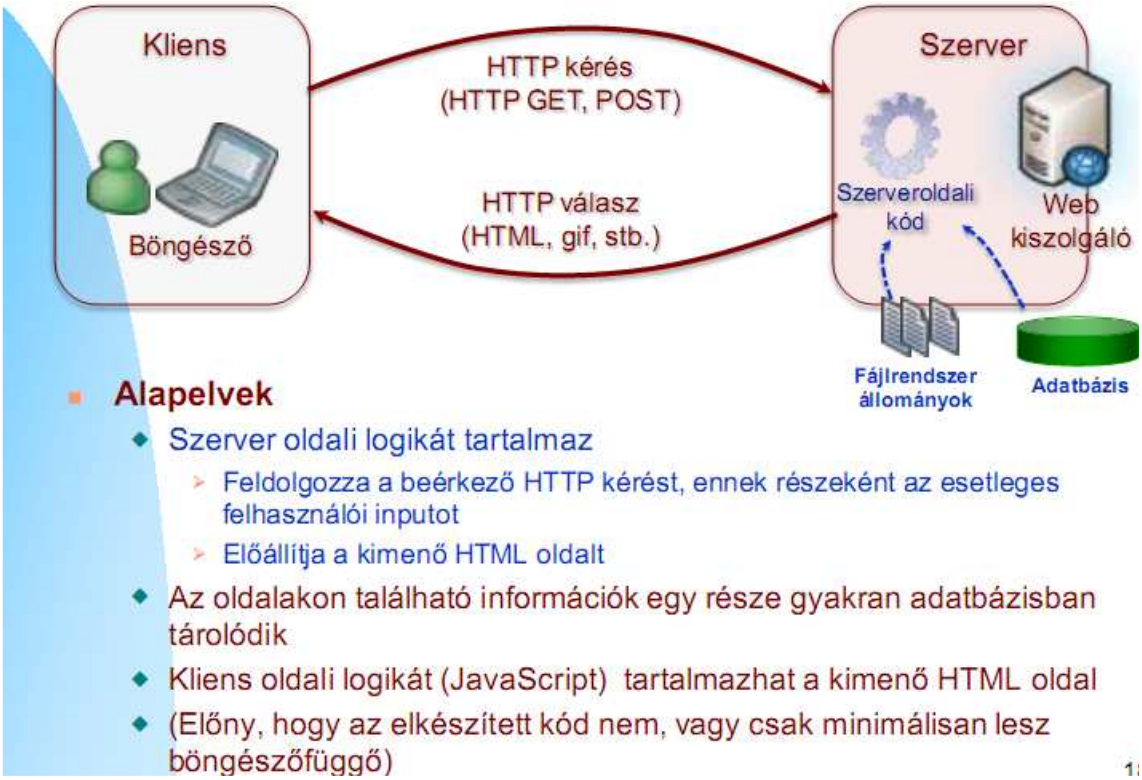
```
<html>
 <head>
  <script language="JavaScript">
     function checkNum(str)
     {
       if (str == "")
       {
         alert("Nem adtál meg semmit!")
         return false
       }
       return true
     }
     function thanks()
     {
        alert("Köszönöm, rendben!")
     }
    </script>
 </head>
```


```
<body>
 <form name="form1">
   Kérlek, írj ide valamit:
   <input type="text" name="num"
    onchange="if (!checkNum(this.value))
      { this.focus(); this.select(); }
       else thanks();">
 </form>
</body>
</html>
```


A példa rövid magyarázata

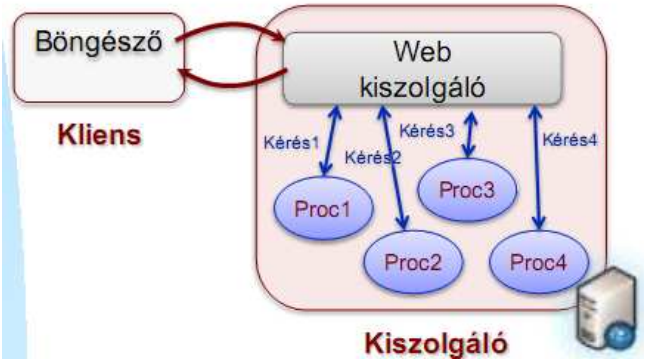
- A <head> alatti <script> tag, a language=”…” attribútum adja meg a szkript nyelvét (pl. JavaScript).
- Ciklust, elágazást, visszatérési értéket ugyanúgy adunk meg mind Java-ban.
- A változók deklarálásánál ki kell tennünk a var kulcsszót.
- Függvény létrehozása
- A function kulcsszó után megadjuk a függvény nevét, majd zárójelek között megadjuk a paramétereket. A függvény törzse kapcsos zárójelek közé kerül.
- Figyelmeztető ablak feldobása
- Az alert kulcsszóval tehetjük meg. Paraméterként a kiírandó szöveget kell átadni.
- A dokumentum egyes elemeinek elérése
- document.getElementById("...");
- A példában szkriptfüggvények meghívása a szövegdoboz onChange eseménykezelőben történik

73. Ismertesse a dinamikus webalkalmazások működését és főbb jellemzőit (definíció, kliens oldal, szerver oldal, ábra)!



74. Jellemezzen röviden néhány elterjedtebb kiszolgálóoldali technológiát! (CGI, PHP, JEE, ASP.NET)

CGI - Common Gateway Interface.
Szabványos protokoll a webkiszolgáló és egy külső alkalmazás kommunikációjára.
 Teljesítményproblémák! Minden kérésre új processzt indít: sok párhuzamos kérés esetén a processzek indítgatása-leállítása magában túlterheli a kiszolgálót!
 Nehéz fejleszteni, nem objektum-orientált, „ősi” (a külső folyamattal környezeti változókon keresztül kommunikálunk és az a szabványos kimeneten üzen vissza)

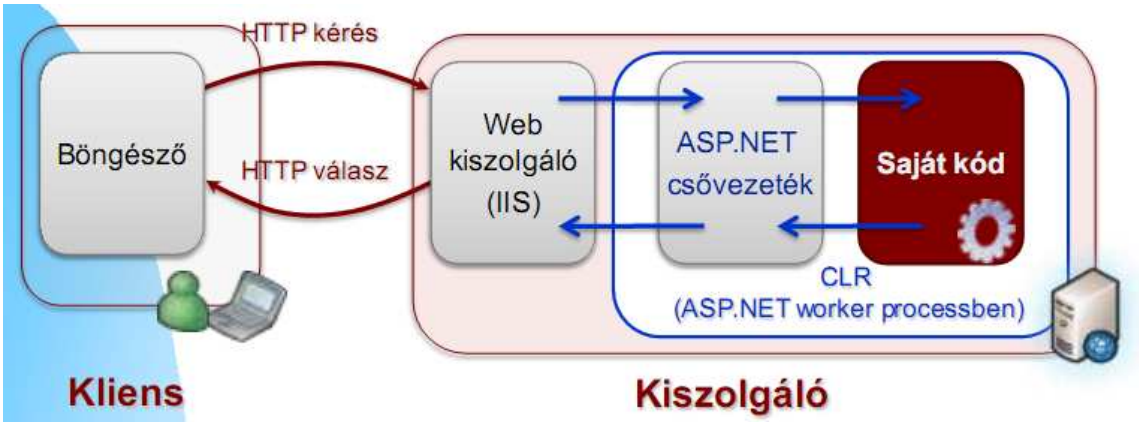


PHP (& társai)
- Szerver oldali szkript
- PHP kódból weboldalkimenet
- Csak weboldalak fejlesztésére
- Korlátozott integrálhatóság

Java
- JEE (Java Enterprise Edition) részeként
- JSP (Java Server Pages) és
- Java Servlet technológiák

ASP.NET
- A .NET Framework része
- CLR által felügyelt, natív kódú végrehajtás
- C#, Visual Basic .NET, stb. nyelvek
- Gyors fejlesztés: olyan eseményvezérelt programozási modell, mint a Windows Form alkalmazások esetében!

75. Ismertesse röviden az ASP.NET architektúráját!



Működés
- A webkiszolgáló processz .aspx kiterjesztésű kérések esetén a kérést továbbítja az ASP.NET worker processznek, amiben áthalad az ASP.NET csővezetéken, és végül a kérést a .NET-ben megírt saját kód dolgozza fel, és állítja elő a választ.

76. Ismertesse az ASP.NET WebForm fogalmát!

A webform összefogja az adott oldalhoz:
- HTML, direktívák, vezérlők, statikus szövegek
- Funkcionalitást megvalósító kód

77. Ismertesse a „szerver oldali inline szkript” fogalmát ASP.NET környezetben, mutasson rá kódrészletet!

Jellemzők

- Minden oldalhoz egy .aspx kiterjesztésű fájl tartozik
- Első lekéréskor ezt lefordítja az ASP.NET (egy temp könyvtárba DLL formájában)
- A lefordított kód feldolgozza a bejövő kérést és előállítja a választ
- Későbbi kéréseket ez a lefordított kód szolgálja ki (gyors)
- Az aspx fájlban vegyesen
  - HTML markup – módosítás nélkül kerül a HTML kimenetbe
  - <% ... %> között .NET (tipikusan C#) nyelvű kód, ami a szerveren fut le, és a Response.Write-tal írhat a HTML kimenetbe
  - <%= ... %> között .NET (tipikusan C#) nyelvű kód, kifejezés, ami a szerveren fut le: egy stringet kell visszaadjon, ami belekerül a HTML kimenetbe.
  - Szerver oldali vezérlők, erről külön, a későbbiekben lesz szó.

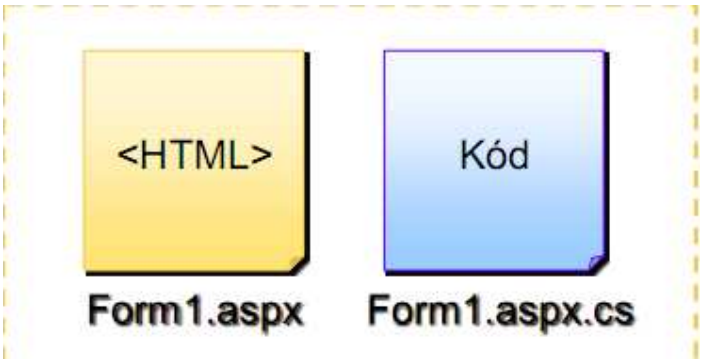
- A kód két sor jelenít meg:
  - Az első sorban ezt írja ki:

    5! = 120

    (vagyis kiírja 5 faktoriálisát)
  - A második sorban kiírja az aktuális szerver oldali időt
- A @Page direktíva Language attribútuma határozza meg, mi a szerver oldali forrásnyelv (c#)
- A nem kiemelt részek módosítatlan formában kerülnek a kimenetbe
- A kiemelt <% %> közötti kódrészek a szerveren futnak le, és beleírnak a HTML kimenetbe

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN">
<html>
<head>
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    5! = <%  int n = 1;
             for (int i = 1; i <= 5; i++)
                 n = n*i;
             Response.Write(n.ToString());
                 %>
     <br>
     Idő: <%=DateTime.Now.ToString() %>
    </div>
    </form>
</body>
</html>
```

78. Ismertesse mögöttes kódfájl alapú ASP.NET WebFormok felépítését, és jellemezze a megoldást!



Minden oldalhoz egy .aspx kiterjesztésű és egy C# nyelvű fájl tartozik
Az aspx fájlban

- HTML markup – módosítás nélkül kerül a HTML kimenetbe
- Szerver oldali vezérlők, erről külön, a későbbiekben lesz szó.
- Lehetnek inline elemek (<% ... %> között), de nem szokás
- A mögöttes kódfájlban: System.Web.UI.Page-ből leszármazott sajátosztály, amiben
- az oldalon levő vezérlőelemek eseménykezelői találhatók, és a további, oldalhoz tartozó logika

Jellemzők
 Nem kavarodik az alkalmazás működéséért felelős kód és a felhasználói felület arculatáért felelős megjelenítés, jobban olvasható, karbantartható a kód
 Szépen különválik a fejlesztő és a felülettervező (designer) feladata, könnyebb az együttműködés

79. Jellemezze az ASP.NET kiszolgáló oldali vezérlőket (fontosabb jellemzők, szerepük, működésük, példakód).

Deklaratív létrehozás: aspx fájlban runat="server" attribútummal

<asp:TextBox ID="tbNumber" runat="server"></asp:TextBox>
<asp:Button ID="btnCalc" runat="server" Text="Számol" onclick="btnCalc_Click" />

Az ID attribútummal meghatározott névvel lehet az oldalhoz tartozó mögöttes kódból a vezérlőket elérni
 A vezérlő egy .NET objektum
Az aspx fájlban HTML attribútumokon keresztül testreszabható a megjelenésük és működésük
 A fenti példában a Button
 Text attribútuma határozza meg a gomb szövegét
 Az onclick attribútum határozza meg, hogy a mögöttes kód osztályának mely tagfüggvénye hívódjon meg a kattintás esemény esetén (btnCalc_Click)Állapotmentesek a szerver oldalon

 Egy vezérlőelem állapota („ViewState”) együtt utazik a lappal (lásd később)

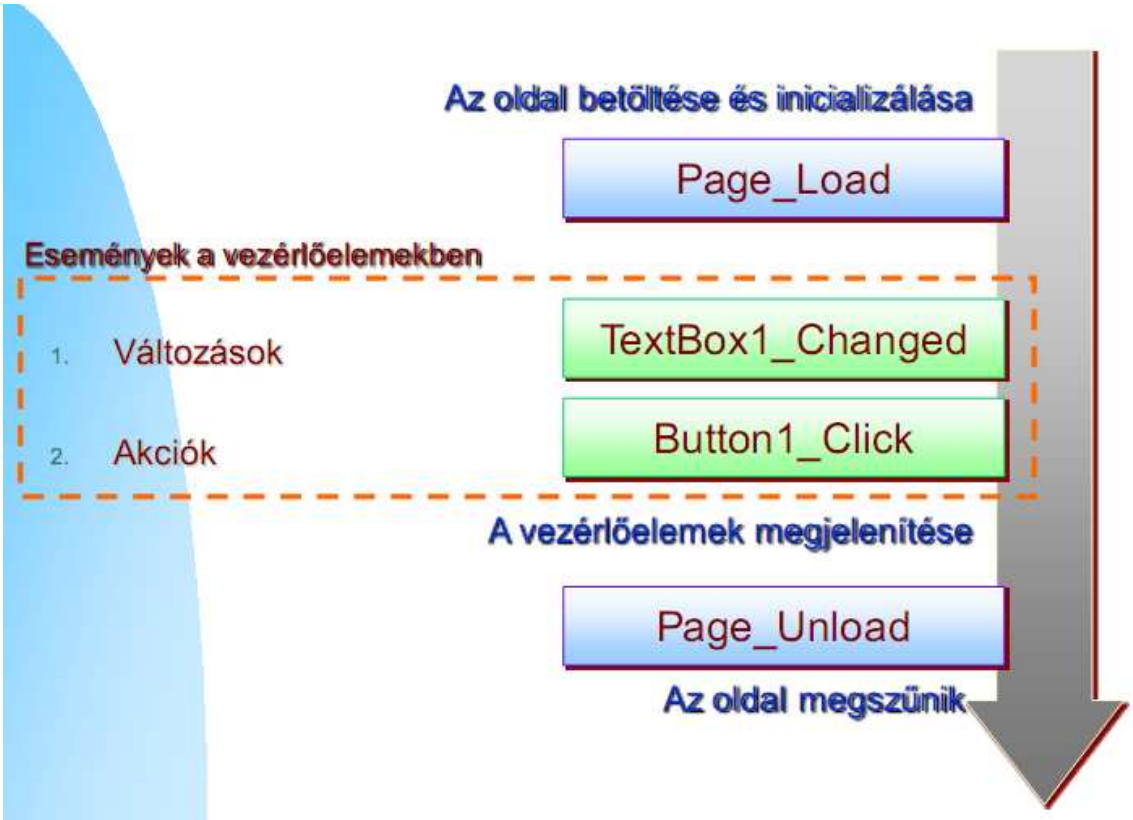
```
<%@ Page Language="C#"
  AutoEventWireup="true" CodeFile="CodeBehindDemo.aspx.cs"
  Inherits="CodeBehindDemo" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
<html>
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Adjon meg egy számot:
        <asp:TextBox ID="tbNumber" runat="server"></asp:TextBox>
        <asp:Button ID="btnCalc" runat="server" Text="Számol"
                onclick="btnCalc_Click" />
        <br />
        A fenti szám faktoriálisa:
        <asp:Label ID="lblResult" runat="server"></asp:Label>
    </div>
    </form>
</body>
</html>
```

```
public partial class CodeBehindDemo : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        // Inicializálásra használható
    }

    protected void btnCalc_Click(object sender, EventArgs e)
    {
        int fact = factorial(int.Parse(tbNumber.Text)) ;
        lblResult.Text = fact.ToString();
    }

    public int factorial(int n)
    {
        int k = 1;
        for (int i = 1; i <= n; i++)
            k = k * i;
        return k;
    }
}
```

A példa rövid magyarázata
CodeBehindDemo.aspx
 A @Page direktíva CodeFile attribútuma határozza meg, mely mögöttes kódfájl tartozik az aspx fájlhoz
 A @Page direktíva Inherits attribútuma határozza meg, hogy a mögöttes kódfájl mely osztálya tartalmazza az aspx oldalhoz tartozó kódot
 A <form> elem runat="server" attribútuma jelzi, hogy az űrlapot a kiszolgálón kell feldolgozni
 Az űrlapon található egy TextBox kiszolgáló oldali vezérlő, ennek azonosítója bNumber (ilyen néven tudjuk a mögöttes kódfájlból elérni). A runat="server" attribútum jelzi, hogy a vezérlőt a kiszolgáló oldalon kell feldolgozni.
 Az űrlapon található még egy gomb (Button) és címke (Label) kiszolgáló oldali vezérlő is. Az előbbinél a onclick="btnCalc_Click" attribútum jelzi, hogy a gombkattintás eseményhez a mögöttes kódfájl osztályának meg tagfüggvénye tartozik.

CodeBehindDemo.aspx.cs
 Az aspx oldalhoz tartozó mögöttes kódfájl osztály neve CodeBehindDemo,a System.Web.UI.Page-ből származik
 A btnCalc gombhoz tartozó btnCalc_Click eseménykezelő a kiszolgálón fut le, amikor a kliens az oldalt a gombkattintás hatására visszaküldi a kiszolgálónak. Ebben először kiolvassuk a tbNumber szövegdobz tartalmát, kiszámoljuk a beolvasott szám faktoriálisát, és a lblResult címke szövegét beállítjuk erre.

80. Ismertesse a webes űrlapok életciklusát!



A webes űrlapoknak életciklusa van, a feldolgozási folyamat kezdetén létrejönnek az objektumok, lefutnak a hozzájuk rendelt eseménykezelők, majd az objektumok megszűnnek.

Kétféle eseményről beszélhetünk:
• Változás- (vagy adat-) típusú esemény: alapértelmezés szerint nem küldik vissza azonnal az oldal tartalmát a kiszolgálóra (postback), de egy más okból bekövetkezett postback után lefutnak a hozzájuk rendelt eseménykezelő a szerveren. Ilyen például egy jelölőnégyzet bejelölése vagy egy elem kiválasztása egy listából.

• Akció típusú esemény: alapértelmezés szerint postbacket generálnak
a szerver felé és azonnal lefuttatják a feliratkozott eseménykezelőket.
Akció típusú esemény tipikusan egy gomb megnyomása.

Fontos tudni, hogy az egyes eseménykezelők milyen sorrendben futnak le szerver oldalon. Először a Page_Load, utoljára pedig a Page_Unload eseménykezelő fut le.

81. Mit értünk állapotkezelésen dinamikus webalkalmazások esetén? Mi a jelentősége? Adja meg a kliensoldali és szever oldali állapotmegőrzés előnyeit/hátrányait, sorolja fel a fontosabb , az ASP.NET nyújtotta lehetőségeket !

A HTTP protokoll állapotmentes
 Minden egyes oldal lekérése független a korábbi kérésektől
 A kiszolgáló létrehozza az oldalt, elküldi a klienshez és „elfelejti”

A felhasználó egy alkalmazással kommunikál: az egyes kérések összefüggenek
 Ha a felhasználó megadta az adatait egy űrlapon:elvárás, hogy őrizzék meg tartalmukat a vezérlők!
Az ASP.NET számos lehetőséget biztosít, egyszerűvé teszi

Állapotkezelés lehetőségek
Kliens oldalon
 Előnyök
    Skálázható, mert a szerver oldalon nem tárol semmit
    Webfarmok esetén is használható
 Lehetőségek
    ViewState
    Rejtett mező (Hidden field)
    Süti (Cookie)
    URL paraméterek (Query String)
Kiszolgáló oldalon
 Előnyök
    „Biztonságosabb”
    Nem nő az oldal mérete
 Lehetőségek
    Application objektum
    Session objektum
    Saját megoldás: adatbázis
    Saját megoldás: fájl

82. Sorolja fel és jellemezze röviden az ASP.NET nyújtotta kliens oldali állapotmegőrzési lehetőségeket !

ViewState

- Beépített szolgáltatása az űrlapoknak: az oldal és a rajtalevő vezérlőelemek állapottárolására szolgál
  - Pl. megjelenési információ (szín, betűtípus, stb.) vagy címke szövege
- Oldalhoz tartozik
- Egy rejtett mezőben utazik
- Mire használható?
  - Beépített vezérlők: „automatikusan” megőrzik az állapotukat segítségével
  - Ha saját vezérlőt írunk, állapot tárolására
  - Egy oldal megírásakor saját adatok tárolására.

Pl.:ViewState["SortExpression"] = "ProductID ASC";

- Kliens odalon tárolódik
  - A szerver oldalon nem igényel plusz memóriát
  - Növeli az oldal méretét: sávszélesség igény!
  - A szervernek minden kérésékor fel kell dolgoznia a kódolt állapotot
  - Biztonsági kérdések merülnek fel
- Kikapcsolható: alkalmazásra (web.config)/oldalra/vezérlőre
- Nem mindig lehet kikapcsolni:
  - A változás eseményeket csak akkor generálja a vezérlő, ha be van kapcsolva
  - Ha dinamikusan állítjuk a vezérlők tulajdonságait
- Nem biztonságos, a klienshez elkerül az adat: hitelkártyaszámot, jelszót ne tároljunk benne
- Integritást HASH-eléssel (digitális aláírás) lehet biztosítani

<%@Page EnableViewStateMAC=true …%>

- A tartalom ettől még látszik, de nem lehet módosítani
- Ha a ViewState érzékeny információkat tárol, 3DES titkosítással lehet védeni

<machineKey validation="3DES" />

Rejtett mező (hidden field)

<input type="hidden" id="myId" value="ez itt rejtett" />

Süti (cookie)

- A kiszolgáló sütibe tehet adatokat, amit a böngésző minden kéréssel visszaküld
- Titkosítva és digitálisan aláírva is tárolhatunk benne adatot
- Lejárati idő megadható, lehet perzisztens is
- Méretkorlát: 4KB
- Elérése a kiszolgálón: Request.Cookies ill. Response.Cookies

URL paraméterek (query string)

- Pl.: http://www.contoso.com/listwidgets.aspx?category=basic&price=100
- Méretkorlát: 255 byte
- Kedvencekbe felvéve megmarad a paraméter!
- Egy oldal egyszerű információt ad át egy másiknak
- Elérése a kiszolgálón: Request.QueryString 42

83. Sorolja fel és jellemezze röviden az ASP.NET nyújtotta kiszolgáló oldali állapotmegőrzési lehetőségeket !

Application objektum

- A kiszolgáló tárolja, minden felhasználóra közös
- Komplex objektumok tárolhatók benne
- Cache-elésre ne használjuk (régi ASP fejlesztők), arra a Cache objektum való!

Konkurens hozzáférés!

- Vagy legyen az objektum szálbiztos

```
Application.Lock;
Application["PageCounter"]+= 1;
lblCount.Text = Application["pageCounter"];
Application.Unlock;
```

  - Vagy zárolni kell
  - Probléma: minden kérésre és benne tárolt objektumra közös a zárolás: skálázhatóság!
- Alkalmazás újraindulásakor elveszik az adat

Session state

- A kiszolgáló tárolja: felhasználónként elkülönítve
- Komplex objektumok tárolhatók benne

```
Session["orderitems"] = orderItems;
```

- Sok erőforrást (memóriát) igényelhet a kiszolgáló oldalon: skálázhatóság
- Nem lehet megosztani a felhasználók között
- Amennyiben nincs feltétlen rá szükség, érdemes kerülni a használatát

```
<%@ Page EnableSessionState="False" %>
```

Biztonság

- Az adat a kiszolgálón: „biztonságban” van?
- Minden felhasználóhoz egy 120 bites SessionID generálódik és beteszi egy sütibe
- A kérdés a következő
  - Ki lehet-e másvalaki SessionID-jét találni egy másik (pl. a sajátunk) ismeretében: NEM
  - El lehet-e lopni egy SessionID-t
    - IGEN, de nem könnyű – visszajátszásos támadás
    - Megoldás
      - Lejárati idő vagy „Log Out” funkció (Session.Abandon())
      - Adatfolyam szintű titkosítás: SSL vagy IPSEC használata
- Süti nélküli mód is megadható (cookieless): URL-t módosítja
- A lejárati idő konfigurálható, 20 perc az alapértelmezett
- Minden Session State konfiguráció: web.config –sessionState bejegyzés