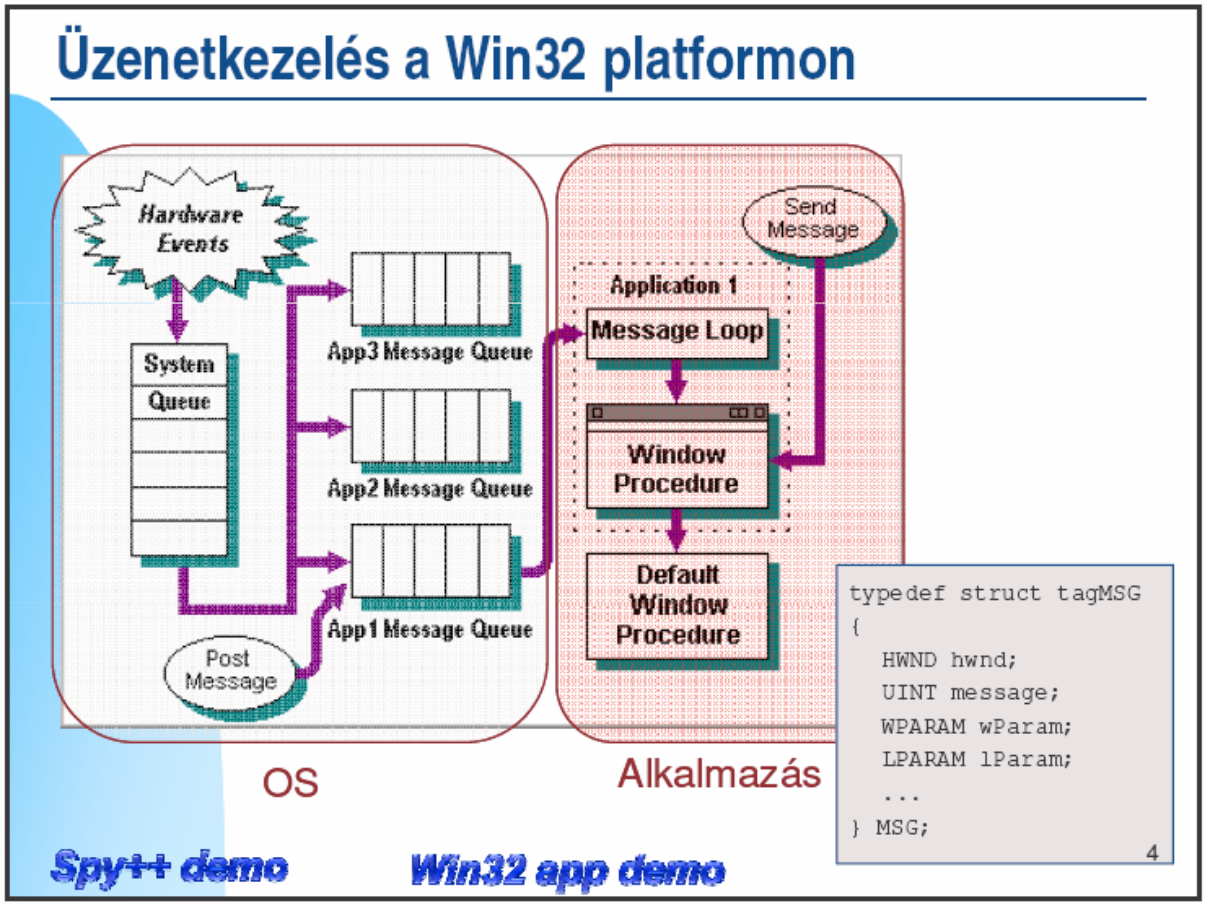
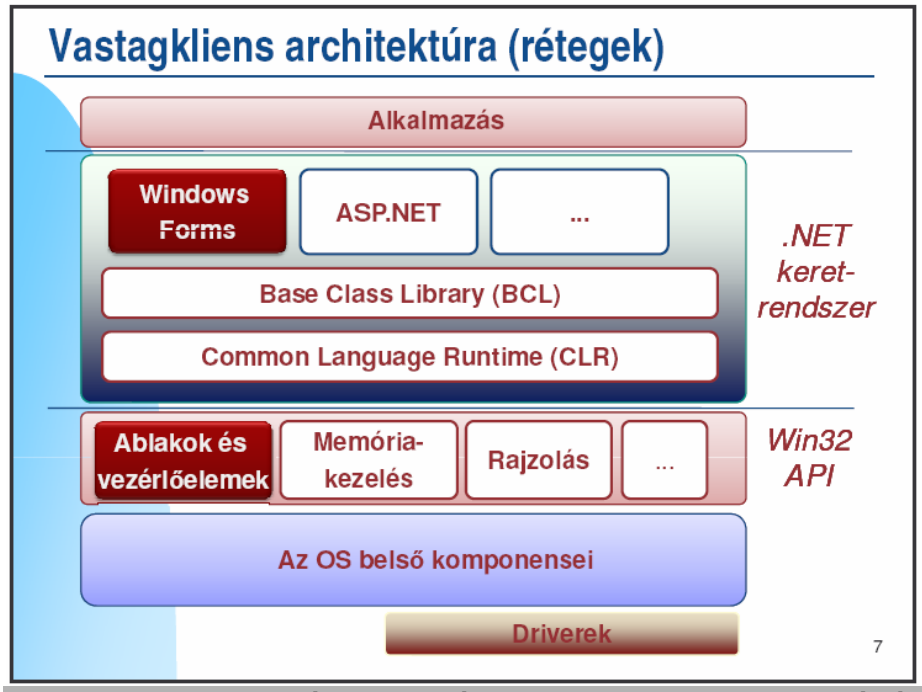
Szoftvertechnikák – összefoglaló a kérdések alapján

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



Ehhez akinek van jegyzete, az egészítse már ki egy kis szöveggel. Köszi!

Ismertesse a .NET vastagkliens alkalmazások architektúráját (rétegek)!



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

Csak .NET 2.0-tól
A fordító fésüli össze (nem lehetnek a részek külön szerelvényben)
Fő területe: a generált és a kézzel írt kód különválasztása

Ismertesse a Windows Forms alkalmazások architektúráját!

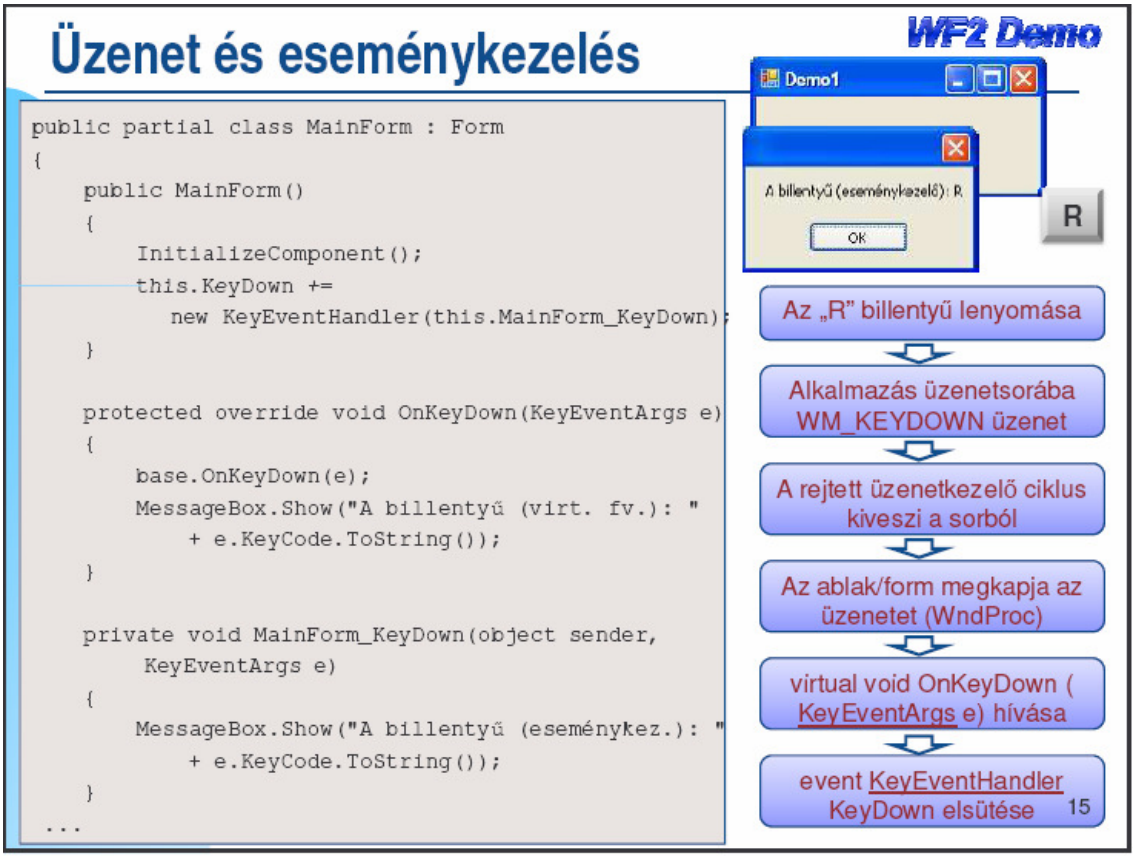
System.Windows.Forms névtér
_ Minden ablak egy Form leszármazott osztály
_ A Form
_ Számos tulajdonsággal rendelkezik (pl. BackColor, Text, Size, ...)
_ Számos eseményt publikál (Load – betöltéskor, Click – egérkattintás,Resize – átméretezés, Move – Mozgatás, KeyDown –billenty_lenyomás, ...)

_ 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
_ Az 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

Ismertesse az üzenet és eseménykezelés kapcsolatát (pl.billentyű_lenyomás 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
  _ Bizonyítsuk be!
  _ Manuális futtatás: Application.DoEvents(): ez feldolgozza a sorban levő üzeneteket



Hogyan lehet menüt és eszközsávot Windows Forms alkalmazásokban készíteni?

_ Menüsáv
  _ MenuStrip
  _ Elemek: ToolStripMenuItem, ToolStripTextBox, ToolStripComboBox,separator
_ Eszközsáv
  _ ToolStrip
  _ Elemek: ToolStripButton, stb.

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)
_ Új form felvétele: jobb katt. a projekten + Add/New Form
  _ Form keret stílus
  _ FormBorderStyle tulajdonság

FormBorderStyle.Sizeable (def.) - átméretezhet_
_ FormBorderStyle.FixedDialog – nem átméretezhet_
_ ...

_ Párbeszédablak

_ 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, ...

Kódok:

Párbeszédablakok felépítése

Párbeszédablakok felépítése

```
public partial class SettingsForm : Form
{
    private int interval;
    public int Interval
    {
        get { return interval; }
        set
        {
            interval = value;
            textBox1.Text = value.ToString();
        }
    }

    private void bOk_Click(object sender, EventArgs e)
    {
        if (!int.TryParse(textBox1.Text, out interval))
            MessageBox.Show(„Érvénytelen érték!");
        else
            this.DialogResult = DialogResult.OK;
    }
}
```

Párbeszédablakok megjelenítése

```
private void settingsToolStripMenuItem_Click(object sender,
EventArgs e)
{
      SettingsForm form = new SettingsForm();
      form.Interval = timer1.Interval;
      if (form.ShowDialog() == DialogResult.OK)
      {
            MessageBox.Show("Changed!");
            timer1.Interval = form.Interval;
      }
}
```

_ Beállítjuk a tulajdonságokat
_ Modális megjelenítés: Form.ShowDialog
_ Ennek visszatérési értékét vizsgáljuk meg: Ok-kal zárták-e be az ablakot (DialogResult enum típus)

Ismertesse a nemmodális ablak fogalmát és kódrészlettel illusztrálja megjelenítését!

Aktiválható más ablak is
_ Megjelenítés

```
SettingsForm form = new SettingsForm();
form.Show();
```

_ A Show-nak megadhatunk egy owner (birtokos) ablakot
_ Nem kerülhet az owner (birtokos) ablak elé Z-orderben

```
SettingsForm form = new SettingsForm();
form.Show(this); // A this egy Form leszármazott
```

Ismertesse a közös párbeszédablak fogalmát és ismertesse fajtáit!

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

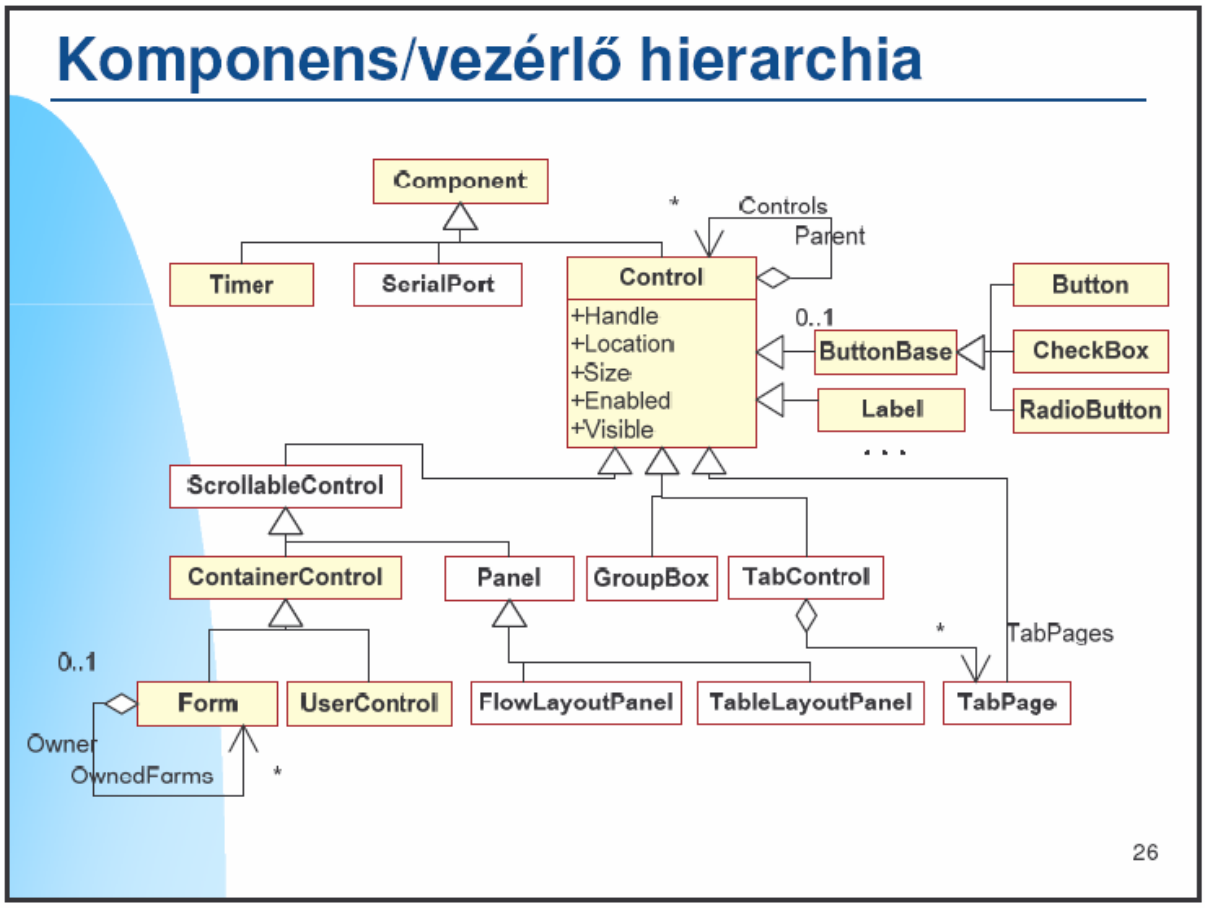
Sorolja fel a fontosabb elemi vezérlőket!

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

Sorolja fel a fontosabb tároló vezérlőket!

GroupBox, Panel, SplitContainer, TabControl, FlowLayoutPanel

Rajzolja fel a komponens/vezérlő hierarciha fontosabb osztályait és kapcsolatukat! Ismertesse a fontosabb osztályokat (Component, Control, ContainerControl)!



Akinek van hozzá jegyzete, az ismertesse plz amit kell.

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

Szülő – gyerek viszony

_ Control.Controls tulajdonság

_ A tartalmazott vezérl_k gyerekablakok (child window)

_ A megszokott viselkedést ez biztosítja

_ A gyerekablak együtt mozog a szülővel

_ Ha megszűnik a szülő, a gyerekablakok isautomatikusan megszűnnek

_ A gyerekablak nem lóghat ki a szül_ablakból (clipping)

_ A szülő elrejtése/megjelenítése automatikusan magával vonja a gyerekablakok elrejtését/megjelenítését

_ A szül_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)

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

A delegate típusa általában (ha nincs esemény paraméter)

```
public delegate void EventHandler ( Object sender, EventArgs e )
```

_ A sender: az esemény kiváltója
_ EventArgs: esemény paraméterek
_ Az EventArgs nem hordoz információt
_ Ebb_l kell leszármaztatni, ha van esemény paraméter
_ Billenty_ események (Control event-ek):
KeyDown, KeyPress, KeyUp

```
this.KeyDown += new KeyEventHandler(this.MainForm_KeyDown);
...
private void mainForm_KeyDown(object sender, KeyEventArgs e)
{
// Ha az ALT + E került lenyomásra
      if(e.Alt && e.KeyCode == Keys.E) { ... }
```

Ismertesse a saját vezérlők készítésének lehetőségeit!

1. Control osztályból leszármaztatás (ha teljesen új vezérlőre van szükség, csak az alap tulajdonságokat és műveleteket kapjuk meg)
2. Adott control osztályból származtatás (csak az új/módosított részeket kell megírnunk, a vezérlő alapértelmezett működése megmarad)
3. UserControl létrehozása

Ismertesse a UserControl felhasználásának lehetőségeit!

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

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

Mikor írunk destruktort?
_ Memóriát nem kell persze felszabadítani...
_ Csak ha „drága”, nem felügyelt erőforrást foglal az objektum (pl. natív ablak, adatbáziskapcsolat, file, lock, stb.) .
_ Egyébként ne írjunk, mert a destruktorral rendelkező osztályok megszűnés előtt bekerülnek a finalizer queue-ba: lassít!
_ Finalize/destruktor jellemzők
_ A végrehajtás ideje nem ismert!
_ A sorrend nem ismert!
_ A szál nem ismert!
_ A destruktor csak a külső hivatkozásait engedheti el (pl. nem felügyelt erőforrások, mert a felügyelteket már lehet felszabadította a szemétgyűjtő)
_ A TAGVÁLTOZÓKHOZ, MELYEK FELÜGYELT ERŐFORRÁSRA

HIVATKOZNAK, ENNEK MEGFELŐEN NE IS FÉRJÜNK HOZZÁ!

Ismertesse a Dispose minta lényegét!

Dispose minta
_ Implementáljuk az IDisposable interfészt!
_ Írjuk meg a void Dispose() műveletet!
_ Ennek törzsében szabadítsuk fel a nem felügyelt erőforrásokat!
_ A Dispose explicit hívható és hívandó: ha már nem használjuk az erőforrást, hívjunk rá Dispose()-t -> EZ VOLT A CÉL
_ De hívjuk a destruktorból is, hogy ha explicit nem

Mutasson példát a Dispose minta megvalósítására!

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

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());
        }

    }
```

Egy űrlapon egy timer1 nevü Timer időzítő komponens, egy label1 nevü 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 egyel növekvő egész számra állítsa a label1 címke szövegét. (Az eseménykezelők bekötéséről is önnek kell gondoskodnia!)

Designer.cs, illetve Form load-ba:

```
this.mStop.Click += new System.EventHandler(this.mStop_Click);
this.mStart.Click += new System.EventHandler(this.mStart_Click);
this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
```

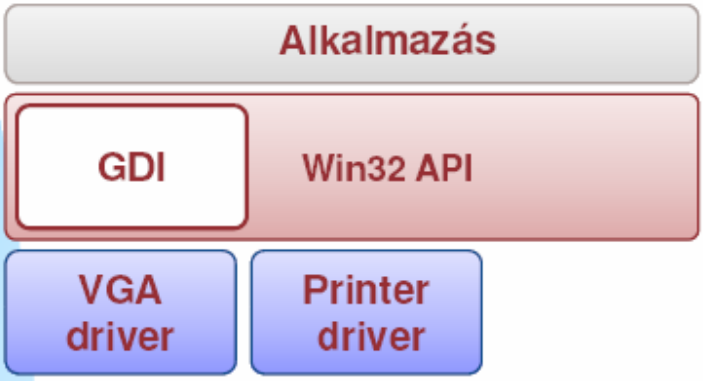
Form-ba:

```
private int c=0;

private void mStart_Click(object sender, EventArgs e) {
    timer1.Interval = 1000;
    timer1.Start();
}

private void mStop_Click(object sender, EventArgs e) {
    timer1.Stop();
}

private void timer1_Tick(object sender, EventArgs e) {
    label1.Text = (c++).ToString();
}
```

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

GDI architektúra

- Eszközfüggetlen grafikus megjelenítést támogat
  - Felbontás és színmélység függetlenül rajzolunk
  - „Ugyanúgy” rajzolunk képernyőre, mint nyomtatóra
    - Nyomtatásnál mi a mi feladatunk -> oldalakra tördelés



- Az eszközfüggetlenség alapja a Device Context, vagyis eszközkapocsolat

Ismertesse az eszközkapcsolat fogalmát!

Reprezentál egy grafikus eszközt ()
_**HDC leíró azonosítja**
_**Lépések**
_1. Eszközkapcsolat létrehozás
_2. Rajzolás az eszközkapcsolatra
_3. Eszközkapcsolat lezárás
_**„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

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 lev, láthatóvá vált ablakrészek (pl. átméretezés, Z-orderben el_bb 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

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***: újrafestend_ 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.

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 kés_bb

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 el_redefiniáltak, pl. *Pens.Blue* – kék folytonos 1px vastag vonal
_ Egyszerű: nem kell létrehozni, felszabadítani!

BRUSH:
Az ecset az alakzatok kitöltési színét, mintáját határozza meg, ***Brush*** _sosztály, 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);
```

Vannak el_redefiniáltak:

_ Pl. ***Brushes.Blue*** – kék tele ecset

Ismertesse a metafájl fogalmát!

Előre rögzített, lejátszható vektorgrafikus műveletek.

Mik a GDI+ erőforrások? Mire kell használatukkor odafigyelni?

A Graphics, a Pen, a Brush és az Image (valamit az ebből leszármazott Bitmap) mind drága, nem felügyelt erőforrásokat zárnak egységbe, így implementálják az IDisposable interfészt! Ha nem használjuk tovább, mielőbb hívjunk rájuk Dispose-t!

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

Egyszerű, az űrlap bet_típusát 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 fels_ 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

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: virtuális ablakok módszere. Alapelve: egy memória
Graphics objektumra rajzolunk, az ***OnPaint-ben*** ennek tartalmát másoljuk a képernyőre.

Fő lépések:
0. Eltároljuk a bonyolult megjelenítendő grafikát egy bitmap-ben (🡐 ez saját megj.)
1. Ha változik a rajz: a memóriában egy bitmapre elkészítjük az új rajzot, a bitmapet egy tagváltozóban őrizzük meg.
2. Az OnPaint-ben kirajzoljuk a bitmapben előkészített rajzot (ez nagyon gyors, nem is villog)

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

```
private int n = 0;

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

Példa: Írjon olyan C# nyelv alkalmazást, ami a (10,10) koordinátában másodpercenként egyel növelve megjeleníti egy számláló értékét!

```
// Előfeltétel 'timer1' nevű Timer legyen felvéve a formra

private int n = 0;

// Belinkeljük a timer eseményét, inicializáljuk, és elindítjuk
protected override void OnLoad(EventArgs e) {
    base.OnLoad(e);
    this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
    this.timer1.Interval = 1000;
    this.timer1.Start();
}

// Paint eseményt felülírjuk, hogy mindig kiírja a számláló akt. értékét
protected override void OnPaint(PaintEventArgs e) {
    e.Graphics.DrawString(n.ToString(), Form1.DefaultFont, Brushes.Black,
10, 10);
    base.OnPaint(e);
}

// Növeljük a számlálót és érvénytelenítjük a form területét (hogy
újrarajzolja)
private void timer1_Tick(object sender, EventArgs e) {
    n++;
    Invalidate();
}
```

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

```
// Előfeltétel 'timer1' nevű Timer legyen felvéve a formra
private bool b = false; // ha false, akkor kéket, ha true akkor zöldet
rajzolunk

// Belinkeljük a timer eseményét, inicializáljuk, és elindítjuk
protected override void OnLoad(EventArgs e) {
    base.OnLoad(e);
    this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
    this.timer1.Interval = 1000;
    this.timer1.Start();
}

// Paint eseményt felülírjuk, hogy mindig a megfelelő négyzetet rajzolja ki
protected override void OnPaint(PaintEventArgs e) {
    // b-től függően zöld vagy kék brusht használunk
    e.Graphics.FillRectangle( b ? Brushes.Green : Brushes.Blue
,10,10,30,30);
    base.OnPaint(e);
}

// Negáljuk az állapot-változót
private void timer1_Tick(object sender, EventArgs e) {
    b = !b;
    Invalidate();
}
```

1. Soroljon fel három C++ tulajdonságot, amelyek alkalmatlanná teszik a nyelvet lazán csatolt komponensek fejlesztésére!

1. nem szabványos a memóriaképe
2. fordítás ideju függvényhívás nem megoldott
3. statikus <-> dinamikus kódnak nem lehet közös adata

2. Hasonlítsa össze a statikus és a dinamikus programkönyvtárakat!

A dinamikus libek nem linkelési (fordítási) időben, vagy a program indulásakor töltődnek be, hanem egy API segítségével futásidőben.

3. Ismertesse a dinamikus könyvtárak elnevezésikonvenció-hierarchiáját!

->libakarmi.so: "linker name", szimlink a soname-re, ez alapján keresgél a linker
->libakarmi.so.3: "soname", általában szimlink a valódi névre, a vége a verziószám
(ami az interface változásakor nő)
->libakarmi.so.3.2.1: valódi név - soname + minor number + release number, ez már konkrétan a lib

4. Mutasson egy Linux VAGY Windows alatt futó példát dinamikus programkönyvtárak betöltésére, felszabadítására és egy könyvtári függvény meghívására! Hol jelenik meg a fordító szintjén lév6 kapcsolódás?

Win:

```
#include <windows.h>

typedef int (*MY_MAX_FUNC)(int,int);
```

és ezeket a main()-ben vagy akárhol:

```
HMODULE hDLL = LoadLibrary("myDll.dll");
MY_MAX_FUNC fvMax;

if(hDLL == NULL)
{
  fprintf(stderr, "Cannot find DLL\n");          //kiírjuk az stderr-re, hogy baj van
  return -1;                                     //végül kilépünk
}

fvMax = (MY_MAX_FUNC)GetProcAddress(hDLL, "max");

if (fvMax == NULL)
{
  fprintf(stderr, "Cannot find function\n");
  return -1;
}

int max = (*MY_MAX_FUNC)(a, b);

FreeLibrary(hDLL);
```

Linux:

```
#include <dlfcn.h>

typedef int (*MY_MAX_FUNC)(int,int);

int main()
{
 MY_MAX_FUNC fvMax;

 void *handle;
 char *error;

 handle = dlopen("./libcomplex.so", RTLD_LAZY);
 if(!handle)
 {
  fputs(dlerror(), stderr);
  exit(1);
 }

 fvMax = dlsym(handle, "max");
 if((error = dlerror()) != NULL)
 {
  fputs(error, stderr);
  exit(1);
 }

 int max = (*fvMax)(a, b);

 dlclose(handle);
 return 0;
}
```

1. Hogyan lehet egy osztály metódusait/tagváltozóit lekérdezni? Mutasson egy C# VAGY egy Java példát!

```
    import java.lang.reflect.*;
    import java.awt.*;

    class SampleField {

       public static void main(String[] args) {
      GridBagConstraints g = new GridBagConstraints();
      printFieldNames(g);
       }

      static void printFieldNamesAndMethods(Object o) {
      Class c = o.getClass();
      Field[] publicFields = c.getFields();
      Method[] theMethods = c.getMethods();

// Tagvaltozok lekerdezese
    for (int i = 0; i < publicFields.length; i++) {
        String fieldName = publicFields[i].getName();
        Class typeClass = publicFields[i].getType();
        String fieldType = typeClass.getName();
        System.out.println("Name: " + fieldName +
          ", Type: " + fieldType);
        }
      }
// Metodusok lekerdezese
      for (int i = 0; i < theMethods.length; i++) {
     String methodString = theMethods[i].getName();
     System.out.println("Name: " + methodString);
     String returnString =
       theMethods[i].getReturnType().getName();
     System.out.println("    Return Type: " + returnString);
     Class[] parameterTypes = theMethods[i].getParameterTypes();
     System.out.print("   Parameter Types:");
     for (int k = 0; k < parameterTypes.length; k ++) {
        String parameterString = parameterTypes[k].getName();
        System.out.print(" " + parameterString);
     }
     System.out.println();
      }
   }
    }
```

2. Hasonlítsa össze a C/C++ nyelv; bináris komponenseket a modern futtatókörnyezetek megoldásaival! Miért van szükség reflexióra?

Azert van szukseg reflexiora, mert igy lehet pl. debuggereket,class browsereket es GUI buildereket irni.

3. Mutasson példát attribútumokra C# nyelven VAGY annotációkra Java nyelven (saját létrehozása, használat, lekérdezés)!

```
// Letrehozas, ez ugye innentol .Net es attributum:
[AttributeUsage(AttributeTargets.Class,
    AllowMultiple=false)]
```

```
public class TableAttribute : System.Attribute
{
   private string _name;
   public string Name
   {
   get
   {
      return _name;
   }
   set
   {
      _name = value;
   }
   }

   public TableAttribute(string name)
   {
   _name = name;
   }
}

// Hasznalat:
[Table("Customer")]
public class Customer
{
   // Anything...
}
// Lekerdezes:
private string GetTable(Type type)
{
   object[] tables = type.
     GetCustomAttributes(typeof(TableAttribute), true);

   if (tables.Length == 1)
   return (tables[0] as TableAttribute).Name;
   else
   return null;
}
```

1. Mutasson egy példát a három anomáliatípusra! Küszöbölje ki a példa anomáliáit BCNF dekompozícióval!

-törlési anomália-Egy szükségtelen adat törlése magával ránt hasznos információt is.
- módosítási anomália-Redundánsan tárolt adat megváltoztatásához az összes tárolási ponton változtatni kell
-beszúrási anomália-Inkonzisztens adatok szúrhatók be a táblába, pl.:

Név	Osztály száma	Osztály neve
Ede	42	Takarítók
Gizi	42	IT

2. Mutassa be az objektum-relációs leképezést! Adjon meg példaként osztálydiagramot,amely tartalmaz 1-1, 1-több, több-több kapcsolatot! Képezze le ezeket adatbázistáblákba!

Egyednek vagy ***entitás***nak nevezünk egy, a valós világban létező dolgot, amit tulajdonságokkal akarunk leírni. Esetünkben egyed lehet egy könyv a könyvtárban, illetve egy adott olvasó. Általánosított fogalmakat használva beszélhetünk "könyv" egyedről és "olvasó" egyedről is.

Tulajdonságnak vagy ***attribútum***nak nevezzük az egyed egy jellemzőjét. Például a könyv, mint egyed legfontosabb tulajdonságai a címe, és a szerző neve.

Egy egyed attribútumainak azt a minimális részhalmazát, amely egyértelműen meghatározza az egyedet, ***kulcs***nak nevezzük és *aláhúzás*sal jelöljük. Esetünkben a „könyv" egyed kulcsa a *könyvszám*, az „olvasó" egyedé az olvasószám.

?????

1. Ismertesse egy rövid C# nyelv; példán keresztül az ADO.NET kapcsolatalapú adathozzáférését!

```
SqlConnection conn = null;
try
{
// Kapcsolódás azadatbázishoz
conn = new SqlConnection(@"Data Source=LAPTOP\SQLEXPRESS;Initial
Catalog=Northwind;Integrated Security=True");

// A kapcsolat megnyitása
conn.Open();

// Az adatbázis parancs létrehozása
SqlCommand command = new SqlCommand("SELECT ShipperID, CompanyName, Phone
FROM Shippers");

// Adatbázis kapcsolat megadása
command.Connection = conn;
Console.WriteLine("{0,0}{1,15}{2,15}", "ShipperID", "CompanyName",
"Phone");
Console.WriteLine("-------------------------------------------------------
---------");
// Az adatok lekérdezése és kiiratása
using (SqlDataReader reader = command.ExecuteReader())
{
while (reader.Read())
Console.WriteLine("{0,4}{1,20}{2,20}", reader["ShipperID"].ToString(),
reader["CompanyName"].ToString(), reader["Phone"].ToString());
}
}
catch (Exception ex)
{
// Kivétel szövegének kiiratása
Console.WriteLine(ex.Message);
}
finally
 {
// Az adatbázis kapcsolat lezárása, ha meg lett nyitva
 if((conn!=null)&&(conn.State==System.Data.ConnectionState.Open))
conn.Close();
}
```

2. Ismertesse egy rövid C# nyelv; példán keresztül az ADO.NET kapcsolat nélküli adathozzáférését!

```
DataSet ds = new DataSet();
SqlDataAdapter da = new SqlDataAdapter("SELECT...", conn);
da.Fill(ds, "Customers");
da.Update(ds, "...");
```

3. Ismertesse az adatkötés fogalmát!

Def1: Az adatkötés az adatforrásokból származó adatok összerendelése az adatmegjelenítő vezérlőkkel.
Def2: Adatkötés alatt azt a folyamatot értjük, mely segítségével kiolvassuk a szükséges adatokat az adatforrásból és dinamikusan egy vizuális elem egyik tulajdonságához "kötjük", azaz a vizuális elem szóban forgó tulajdonságának értékét az adatforrás bizonyos adataitól tesszük (kölcsönösen) függővé.

Adatbázis objektumok kapcsolása WindowsForm Controlokhoz (Control-BindingSource-DataSource-TableAdapter-DB).

4. Mutasson egy példát egy osztály perzisztenssé tételére a Java Persistence API segítségével! Ismertesse a szükséges annotációk jelentését! Mire használjuk a @Transient annotációt?

Java Persistence API, azaz szabványos Obejtum-Relációs mapping

```
@Entity
public class Customer implements Serializable {
@Id protected Long id;
protected String name;
protected Address address;
protected PreferredStatus status;
@Transient protected int orderCount;
public Customer() {}
public Long getId() {return id;}
protected void setId(Long id) {this.id = id;}
public String getName() {return name;}
public void setName(String name) {this.name = name;}
 ..
}
```

@Entity: Entitás osztály jelölése
@Basic: lehetnek egyszer? típusok: primitív/wrapper, szerializálható, enum, byte[], char[], …
@Embedded: lehetnek összetett, beágyazott objektumok
@Id, @EmbeddedId, @IdClass: Els?dleges kulcs kötelez?

Entitások állapota perzisztens kivéve a: @Transient vagy transient attribútumokat

@OneToOne, @OneToMany, @ManyToOne, @ManyToMany: 1-1, 1-több, több-1, több-több entitások között