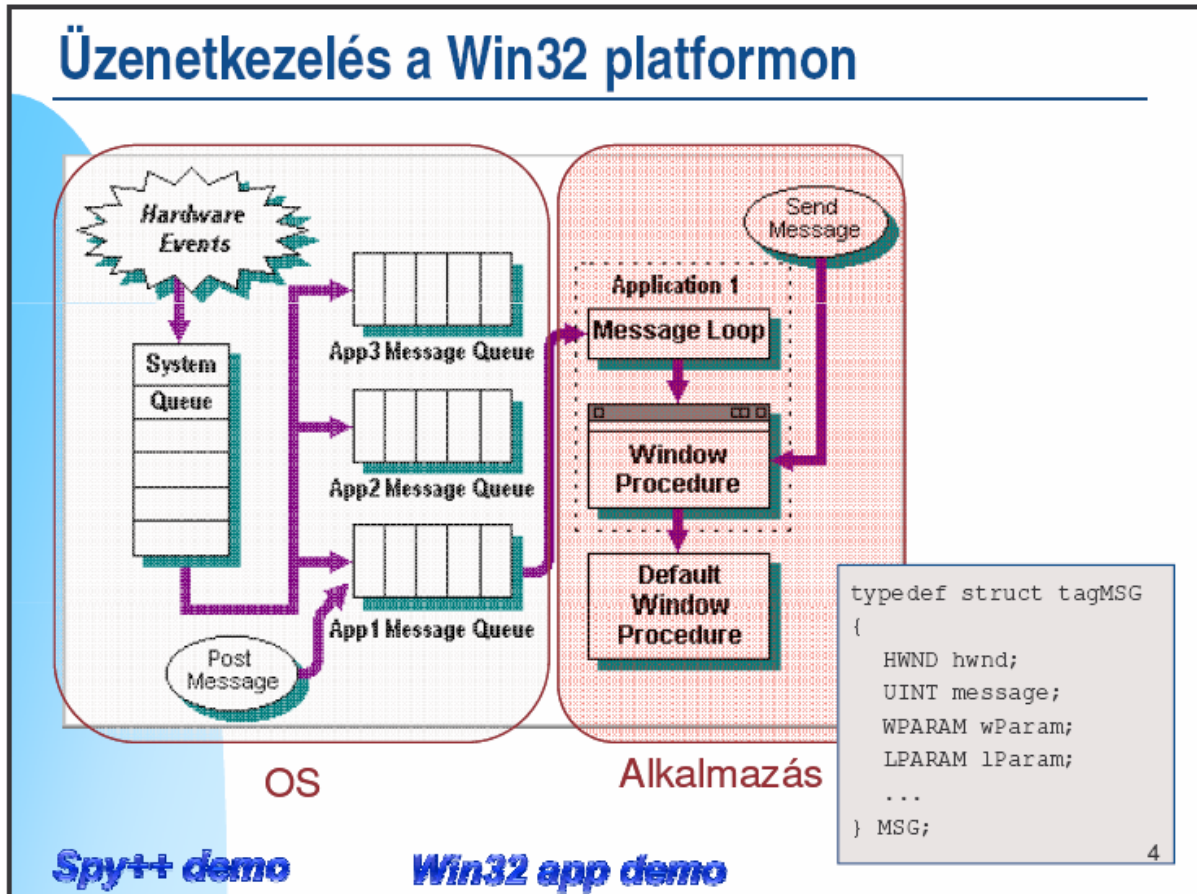


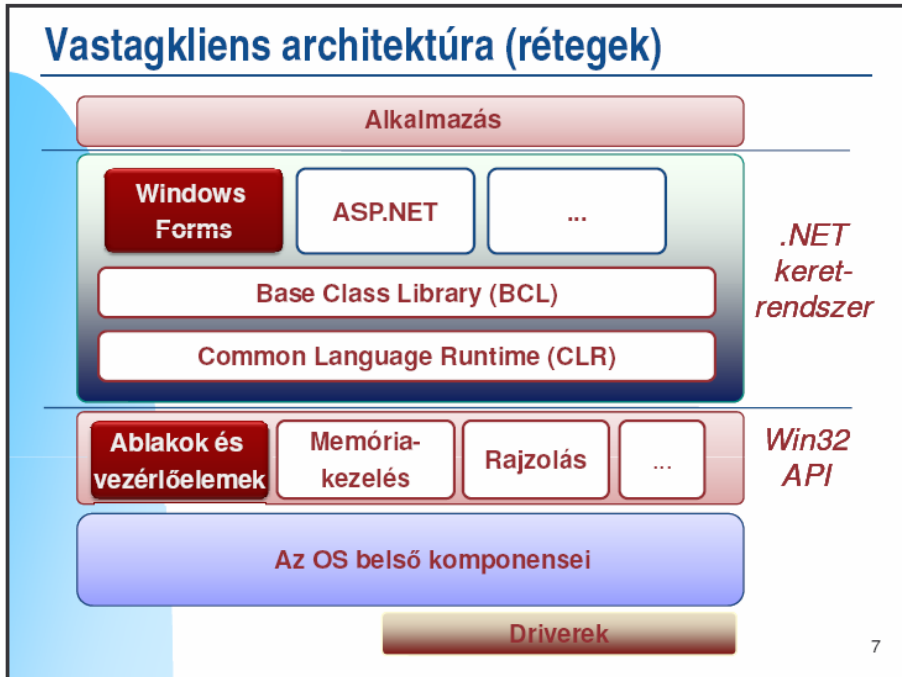
Szoftvertechnikák – összefoglaló a kérdések alapján

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!



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

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 – egérekattintás, Resize – átméretezés, Move – Mozgatás, KeyDown – billentyűnyomás, ...)

_ 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

Üzenet és eseménykezelés

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

    protected override void OnKeyDown(KeyEventArgs e)
    {
        base.OnKeyDown(e);
        MessageBox.Show("A billentyű (virt. fv.): "
            + e.KeyCode.ToString());
    }

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



Az „R” billentyű lenyomása

Alkalmazás üzenetsorába WM_KEYDOWN üzenet

A rejtett üzenetkezelő ciklus kiveszi a sorból

Az ablak/form megkapja az üzenetet (WndProc)

virtual void OnKeyDown (KeyEventArgs e) hívása

event KeyEventHandler KeyDown elsütése 15

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 (=dialogusablakok) 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, FolderBrowserDialog, FontDialog

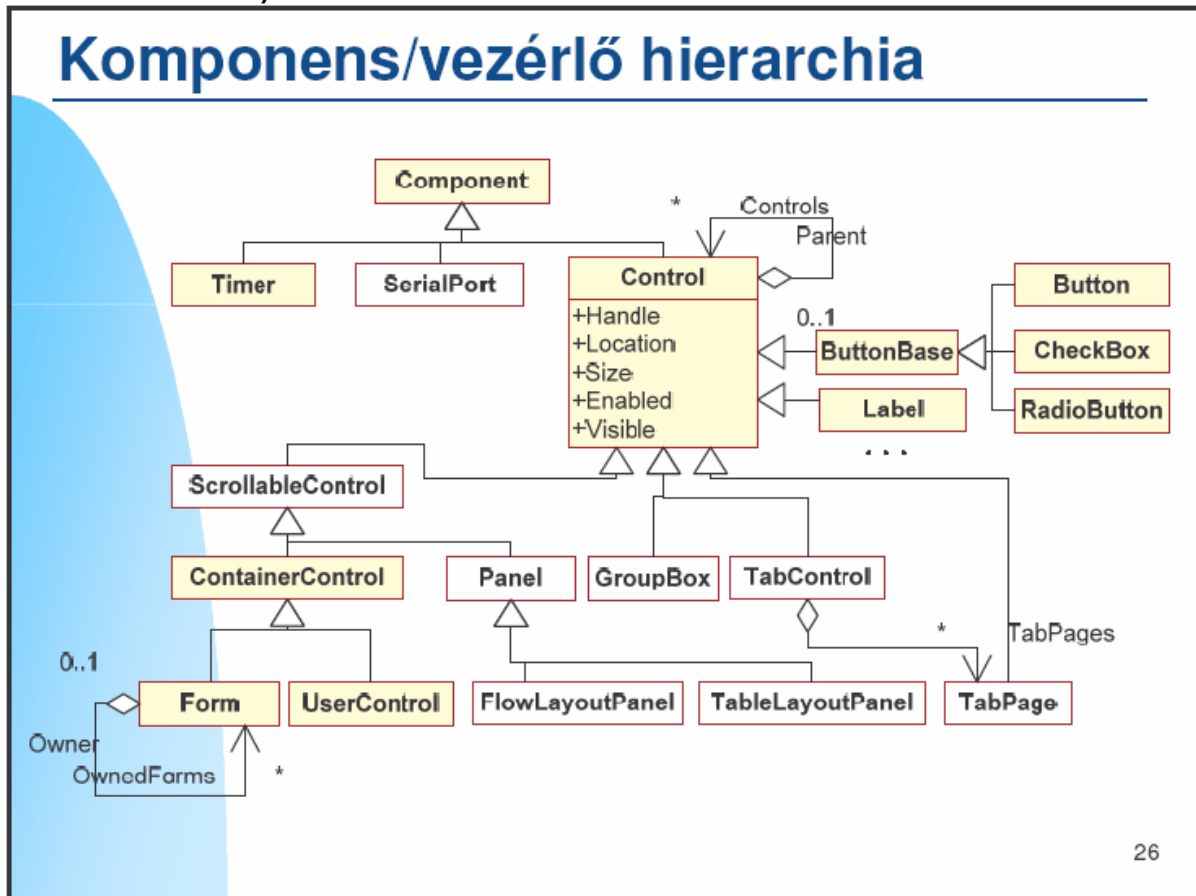
Sorolja fel a fontosabb elemi vezérlőket!

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

Sorolja fel a fontosabb tároló vezérlőket!

GroupBox, Panel, SplitContainer, TabControl, FlowLayoutPanel

Rajzolja fel a komponens/vezérlő hierarchia 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 vezérlő_k 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 szülő_ablakból (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 (+ugyanaz 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
 - _ Ebből kell leszámaztatni, ha van esemény paraméter
- _ Billentyű események (Control event-ek):
 - KeyDown, KeyPress, KeyUp

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

Ismertesse a saját vezérlők készítésének lehetőségeit!

Ismertesse a UserControl felhasználásának lehetőségeit!

Ismertesse a Finalize metódus és a destruktorkapcsolatát, 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 destruktorkal rendelkező osztályok megszűnés előtt bekerülnek a finalizer queue-ba: lassít!

_ Finalize/destruktorkarakterisztikák

- _ A végrehajtás ideje nem ismert!
- _ A sorrend nem ismert!
- _ A szál nem ismert!
- _ A destruktork csak a külső hivatkozásait engedheti el (pl. nem felügyelt erőforrások, mert a felügyelteteket már lehet felszabadította a szemétyűjtő)
- _ **A TAGVÁLTOZÓKHOZ, MELYEK FELÜGYELT ERŐFORRÁSRA HÍVATKOZNAK, ENNEK MEGFELŐEN NE IS VÉRJÜNK 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 használuk 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 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 egyel növekvő egész számra állítsa a label1 címke szövegét. (Az eseménykezelők bekötéséről is önnek kell gondoskodnia!)

valaki?

Saját vezérlők készítése

_ Ismertesse a saját vezérlők készítésének lehetőségeit!

- Háromféle módszer
- _ Contol osztályból leszármaztatás
- _ Adott vezérlőből leszármaztatás
- _ UserControl készítés

_ 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.

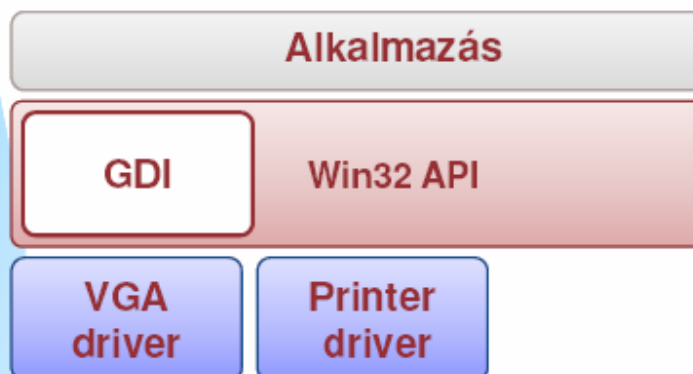
_ Példa *FilePicker* vezérlő: tipikusan együtt előforduló vezérlőelemek összekötése

Grafikus megjelenítés

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 eszközkapcsolat**

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
- _ Metafile-ra
- _ 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 levő, láthatóvá vált ablakrészek (pl. átméretezés, Z-orderben elbbr került az ablak, stb.)

_ Miért nem jegyzi meg a rajzot az OS? Sok ablak létezik 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: újrafestendő terület

Form.ClientRectangle: ablak kliens területe

__ Nemcsak az _rlapoknál, hanem saját vezérl_ elemeknél is ugyanígy rajzolunk!

__ Az **OnPaint**-tet ne hívjuk explicit (az OS optimalizál)! Helyette: **Invalidate**-et hívjunk, ami érvényteleníti a területet és kiváltja az újrajzolást.

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 szint 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**_osztály, 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-!

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?

??

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

Kód jöhet.

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

Kód jöhet.

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

Kód jöhet.

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

Kód jöhet.

Szoftvertechnikák tételsor Levendovszky Tihamér

NB: A ZH-, ill. vizsgakérdések ezekb_1 a kérdésekb_1 és azok kombinációjából állnak össze.

I. Bináris komponensek evolúciója 1.

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

- újraimplementálsz virtuális függvényeket
- hozzáadsz vagy elveszel virtuális tagfüggvényeket
- megváltoztatod az osztályhierarchiát (kivéve ha csak új leveleket adsz hozzá)
- elveszel/hozzáadsz privát adattagokat,
- eltávolítasz public vagy protected nem inline metódust
- inlineként valósítasz meg régi (nem inline) public/protected metódusokat
- megváltoztatod egy inline metódus működését úgy, hogy a régi verzió így már nem működik megfelelően

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 változásokor 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 lévő kapcsolódás?

```
#include <stdlib.h>
#include <stdio.h>
#include <dlfcn.h> // Ez a header tartalmazza a dinamikus linkeleshez szükséges dolgokat
                // dlopen(), dlsym(), dlclose()

int main(int argc, char **argv) {
    void *handle;
    double (*cosine)(double);
    char *error;

    handle = dlopen ("lib/libm.so.6", RTLD_LAZY); // lib betöltése
    if (!handle) {
        fputs (dlerror(), stderr);
        exit(1);
    }

    cosine = dlsym(handle, "cos"); // fv. kikeresése
    if ((error = dlerror()) != NULL) {
        fputs(error, stderr);
        exit(1);
    }

    printf ("%f\n", (*cosine)(2.0)); //fv meghívása
    dlclose(handle); //lib bezárása
}
```

Listáról, WIN-re:

```
HMODULE hDLL;
hDLL = LoadLibrary ("valami.dll");
FreeLibrary (hDLL);
GetProcAddress (hDLL, "valamifuggveny");
```

II. Bináris komponensek evolúciója 2.

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 így lehet pl. debuggereket, class browsereket es GUI buildereket írni.

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

```

// Letrehozás, ez ugye innetől .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;
}

```

III. Adatkezelés 1.

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.

-beszúrási anomália-Inkonzisztens adatok szűrhetők be a táblába, pl.:

| Név | Osztály száma | Osztály neve |

```

-----
| Ede | 42      | Takarítók  |
| Gizl| 42      | IT          |

```

- módosítási anomália-Redundánsan tárolt adat megváltoztatásához az összes tárolási ponton változtatni kell

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

IV. Adatkezelés 2.

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

```
SqlConnection conn = null;
try
{
    // Kapcsolódás az adatbázishoz
    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 kiírása
    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 kiírása
    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 adathozzáférést!

3. Ismertesse az adatkötés fogalmát!

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