C++/CLI zh segédlet

Nagy Dániel

November 2019

Tartalomjegyzék

0.1. Használati útmutató

- Ne olvasd végig! Akkor vedd elő ha programozás közben elakadtál, a tartalomjegyzőkből keresd ki a szükséges dolgot, majd menj az adott oldalra! Remélem a struktúra egyértelmű lett.
- A példakódok csak *kódrészletek*, egész kódot nem tettem bele, hogy ne kelljen senkinek 100 oldalt kinyomtatnia. Ezért nem kell meglepődni ha esetleg valami nincs előre definiálva vagy ilyesmi, próbáltam elkerülni ezt, de előfordulhat.
- A táblázatok szerintem bemutatásra szorulnak, vegyük példának a 4.14 résznél leírt `Timer` vezérlőt, amihez a `timer1` objektum tartozik.
  - `Property` ez új példa az 1.1.1 alatt. A lényeg hogy ezeket a következő módon érjük el: `timer1->Interval = 200;` amikor beállítjuk vagy `int intervals = timer->Interval;` ha lekérjük.
  - `Method` ez talán nem új, tagfüggvényt jelent. A következő módon érjük el a metódusokat[1]: `timer1->Start();`
  - `Event` ezeket ne hozzuk létre kézzel mi. `Jobb klikk` a vezérlőre, majd `Properties` jobb oldalt beugrik az ablak, ott válasszuk ki az `Events` fület (villámjel), keressük ki a nekünk szükséges eventet, és dupla kattintással legenerálja a VisualStudio.

[1] Itt hozzá kell tenni hogy természetesen lehetnek argumentumai a függvénynek és a visszatérési értékek is vannak, de ez itt éppen `void`

1. CLI sajátosságok

1.1. Referencia osztály és property-k

1.1.1. property-k

1. Referencia osztályon belül
2. property létrehozása:

```
property int Ertek
{ kifejtes... }
```

3. set és get függvények:

```
int get() {return this->ertek; }
void set(int ertek) {this->ertek = ertek; }
```

1.1.2. Példa egy referencia osztályra

```
ref class sikidom
{
private:
    String ^ nev; //a referenciat ^ jeloli
    double oldalhossz;
public:
    sikidom(String ^ nev, double oldalhossz){
        this->nev = nev;
        this->oldalhossz = oldalhossz;
    }
    //irhato es olvashato property
    property double Oldalhossz{
    double get() { return this->oldalhossz; }
    void set(double oldalhossz) { this->oldalhossz = oldalhossz; }
    };
    // csak olvashato property (set hianyzik)
    property String ^ Nev{
    String ^ get() { return this->nev; }
    };
};
```

1.2. Dinamikus managelt memóriafoglalás

Memóriafoglalás a gcnew operátorral:

```
String ^ s = gcnew String("Hello");
```

1.3. Az override specifier

Létező függvények felüldefiniálása. Egy tipikus példa:

```
String ^ ToString() override
{
    String ^s = gcnew String("Hello world");
    return String;
}
```

Így a `ToString` metódus úgy fog viselkedni ahogy mi szeretnénk.

1.4. Hibakezelés

Hibakezelés a `try`, `catch` blokkal. Például hogyha rossz inputot kap a programunk.

```
try
{
    //itt pl a ToDouble-nal lephet fel hiba
    double t = Convert::ToDouble(textBox1->Text);
}
catch (Exception ^ e)
{
    MessageBox::Show("Message: " + e->Message, "Hiba"); // hiba kijelzese
}
```

1.5. Öröklődés

Csak *egy* osztályból lehet örököltetni egyszerre. Örököltessük az 1.1.2. részben leírt kódot:

```
ref class Test: sikidom
{
private:
    double magassag;
public:
    Test(String ^ nev,double oldalhossz,double magassag):
        sikidom(nev,oldalhossz), magassag(magassag){};
};
```

1.6. Az `interface`-ek

Hogyha szeretnénk, hogy egy class több helyről is örökölhessen tulajdonságokat. Öröklődés után a benne található `property`-ket és metódusokat *ki kell fejteni*. Minden adattag public!

```
interface class sikidom
{
    property String ^nev;
    double ker();
    property double ter
    {
        double get();
    };
};
```

Létrehozunk egy 2D pontokat tároló osztályt.

```
ref class p2D
{
private:
    double x, y;
public:
    property double X
    {
        double get() { return x; }
        void set(double x) { this->x = x; }
    }
    property double Y
    {
        double get() { return y; }
        void set(double y) { this->y = y; }
    }
};
```

Öröklődés, `interface` és osztály egyszerre:

```
ref class kor : p2D, sikidom
{
private:
    double r;
public:
    kor(double x, double y, double r) : r(r)
    {
    X = x;
    Y = y;
    nev = "Kor";
    }
    // sikidom osztalybol orokolt elemek
    virtual property String ^ nev;
    virtual property double ter
    {
        double get() { return r * r* 3.14; }
    };
    virtual double ker()
    {
        return 2 * r* 3.14;
    }
};
```

2. Tárolók

2.1. `List` tároló

Ez egy dinamikusan bővíthető lista, hasonlít a `Native C++`os `Vector` tárolóhoz.
A `System::Collections::Generic` névterületen található.

`List<type>(int meret)`	Constructor	Adott típusú lista létrehozása, ami adott memóriát foglal le
`Add(type adat)`	Method	Elem hozzáadása a listához
`RemoveAt(int index)`	Method	Adott indexű elem eltávolítása
`Clear()`	Method	Összes elem törlése a listáról
`type [int index]`	Method	indexelés pl `int elem = list[3]`
`int Count`	Property	Elemszám a listában
`int Capacity`	Property	Maximális elemszám a listában újraméretezés nélkül

Példa: néhány művelet a listákkal

```
List<double>^ a = gcnew List<double>(5);
//5 elemet szeretnek majd a listaba tenni
Console::WriteLine(a->Count); //0-t kapunk
//elemek hozzaadasa
a->Add(1.0); a->Add(2.3); a->Add(-3.2); a->Add(1e10); a->Add(20.);

Console::WriteLine(a->Count); //mostmar 5 elem van
Console::WriteLine(a->IndexOf(2.3)); //1et ad vissza
a->RemoveAt(3); //1e10 torlese a listarol
Console::WriteLine(a[3]); //20at ir ki
a->Clear();
```

2.2. `Array` tároló

Statikus listák, úgy használjuk őket mint a `C/C++` tömböket. Lehetnek több dimenziósak is, ezt a `rank` adja meg.

`List<type,rank>(int meret)`	Constructor	Adott típusú lista létrehozása, adott elemszámmal
`type [int index]`	Method	indexelés pl `int elem = list[3]`, rögtön használható
`int Length`	Property	A lista hossza

Példa: Egy 10×10-es `array` létrehozása, és feltöltése.

```
array<double, 2> ^ c = gcnew array<double, 2>(10,10);
for (int i = 0; i < 10; i++)
{
	for (int j = 0; j < 10; j++)
	{
		//indexeles
		c[i, j] = i * 10 + j;
	}
}
int hossz = c->Length; //hossz 100 lesz
```

3. Stringek és fájlok

3.1. `String` osztály

`String(String)`	Constructor, String inicializálása
`int Contains(String)`	tartalmaz-e adott substringet, visszatérés a substring poziciója
`String Replace(String mit,String mire)`	mit cseréljen le mire
`String += String`	Stringek egymás után fűzése
`Array<String ^ > Split(char elv)`	String darabolása listába, adott karakternél
`String Substring(int kezd,int hossz)`	Részlet kiszedése, adott kezdettel és hosszal
`String IndexOf(char c)`	Megkeresi a c karekter *első* előfordulását
`String LastIndexOf(char c)`	Megkeresi a c karekter *utolsó* előfordulását

1. táblázat. Metódusok a `String` osztályban

Példa: Fájlnév átírása a kiterjesztés előtt. `adatok.txt` → `adatok_output.txt`

```
String ^ filename = gcnew String("3.felev/prog/adatok.txt");
int pos = filename->LastIndexOf("."); //itt biztos a lastIndexOf kell!
String ^ saveFilename = filename->Substring(0, pos)
                  + "_output" + filename->Substring(pos);
```

3.2. `StreamReader` osztály

Fájlok beolvasására alkalmas. A `System::IO` névterületen helyezkedik el.

`StreamReader(String ^ fnev)`	Constructor	Melyik fájlt olvassa be
`EndOfStream`	Property	`true` ha a fájl vége amúgy `false`
`String ^ ReadLine()`	Method	Sor beolvasása egy `String`-be
`Close()`	Method	A fájl bezárása

Példa: Fájl beolvasása

```
String ^ filename = gcnew String("hello.txt");
StreamReader ^ sr = gcnew StreamReader(filename);//fajl megnyit
while (!sr->EndOfStream)
{
       String ^ sor = sr->ReadLine(); //sor beolvas
       Console::WriteLine(sor); //sorok kiirasa konzolra
}
sr->Close() //fajl bezar
```

3.3. `StreamWriter` osztály

Fájlba írásra alkalmas. A `System::IO` névterületen helyezkedik el.

`StreamWriter(String ^ fnev)`	Constructor	Melyik fájlt olvassa be
`WriteLine(String ^ )`	Method	Egy adott String-et kiir a fáljba
`Flush()`	Method	Minden adatot kiment a fáljba
`Close()`	Method	A fájl bezárása

3.4. `String::Format` függvény

A C-ben megszokott String formázás. Az idézőjelen belül kapcsos zárójelbe kerülnek az argumentumok, amiket szeretnénk kiírni a megadott helyre. A tizedesjegyek száma is állítható {0:f3} az f mögé kerülő számmal. https://www.tutlane.com/tutorial/csharp/csharp-string-format-method
Példa: Formázott String kiírása fájlba:

```
StreamWriter ^ sw = gcnew StreamWriter("kimenet.txt");
sw->WriteLine(String::Format("{0:f3}\t{1:f3}\t{2:f2}\t",
                      adatsor->a, adatsor->b, adatsor->c));
sw->Flush();
sw->Close();
```

4. Vezérlők

Minden vezérlőhöz az alábbiak tartoznak:

1. Properties (Tulajdonságok) pl.: Szöveg, betűtípus, szín, méret...
2. Events (Események) pl.: Gombnyomás, megváltozik a szöbeg stb...
3. Methods (Metódusok) függvények a vezérlőhöz pl.: Grafika objektum elkérése

4.1. `Label` vezérlő

Szöveg megjelenítésére alkalmas, a felhasználó nem írhatja át.

`Text`	Property	A megjelenő szöveg, `String` formátum
`Font`	Property	Betűtípus beállítások

4.2. `Button` vezérlő

Gomb, a felhasználó meg tudja nyomni.

`Text`	Property	A megjelenő szöveg, `String` formátum
`Click`	Event	A gomb megnyomásáház tartozó esemény

4.3. `RadioButton` vezérlő

Egyszerre egy `RadioButton` lehet kiválasztva. Célszerű egy `Panel`-en elhelyezni az összetartozó `RadioButton`-öket, így minden `Panel`-en egy ki lehet jelölve.

`Text`	Property	A megjelenő szöveg, `String` formátum
`bool Checked`	Property	`true` ha be van jelölve a doboz, egyébként `false`
`CheckedChanged`	Event	A kijelölés megválozásához tartozó esemény

4.4. `CheckBox` vezérlő

Olyan vezérlők, amelyekből egyszerre többet is kiválaszthat a felhasználó.

`Text`	Property	A megjelenő szöveg, `String` formátum
`bool Checked`	Property	`true` ha be van jelölve a doboz, egyébként `false`
`CheckedChanged`	Event	A kijelölés megválozásához tartozó esemény

4.5. `TextBox` vezérlő

A felhasználó egysoros szöveget írhat be ebbe a vezérlőbe.

`Text`	Property	A megjelenő szöveg, `String` formátum, így tudjuk lekérni
`TextChanged`	Event	Ha a szöveg a dobozban megváltozik
`SelectionStart`	Property	A kurzor helye, ha `=Text->Length` akkor a végén lesz

Példa: TextBoxA vezérlőbe, ha ',' karaktert írunk lecseréli '.' karakterre.

```
System::Void textBoxA_TextChanged(System::Object^ sender, System::EventArgs^ e)
{
		if (textBoxA->Text->Contains(","))
		{
			textBoxA->Text = textBoxA->Text->Replace(',', '.');
			//kurzor jo helyre allitasa
			textBoxA->SelectionStart = textBoxA->Text->Length;
		}
	}
```

4.6. `NumericUpDown` vezérlő

Számok bekérésére megfelelő, a számok `Decimal` formátumban kérhetőek le. Lehet kézzel is állítani fel és le, a lépésköz, és korlátok is megadhatóak.

`Minimum`	Property	Az alsó korlát
`Maximum`	Property	A felső korlát
`Decimal Value`	Property	A jelenleg tárolt érték `Decimal` formátumban
`DecimalPlaces`	Property	Megjelenő helyi értékek száma
`Increment`	Property	Gombnyomásra ennyit nő/csökken az érték.

4.7. `TrackBar` vezérlő

Csúszka, amit a felhasználó bizonyos értékekre állíthat be.

`Minimum`	Property	Az alsó korlát
`Maximum`	Property	A felső korlát
`int Value`	Property	A jelenleg tárolt érték `int` formátumban
`TickFrequency`	Property	A beosztások gyakorisága

4.8. `ComboBox` vezérlő

Legördülő listás vezérlő, ahol a felhasználó egy elemet tud kiválasztani

`Items->Add(String ^ mit)`	Method	Ezzel lehet elemet hozzáadni a listához
`Items->Remove(String ^ mit)`	Method	Ezzel lehet egy adott nevű elemet eltávolítani
`Items->Remove(int index)`	Method	Ezzel lehet egy adott *indexű* elemet eltávolítani
`Items->Count`	Property	Tartalmazott elemek száma
`int SelectedIndex`	Property	A kijelölt indexe, `set()` és `get()`-el is rendelkezdik
`DropDownStyle`	Property	`DropDownList` akkor nem írható be bármi
`String Text`	Property	A beírt szöveg (`DropDown` módban)
`SelectedIndexChanged`	Event	Érzekeli ha új elem lett kiválasztva

4.9. `ListBox` vezérlő

Több sort tartalmaz, ahol a felhasználó kiválaszthat egy sort. Célszerű ezzel párhuzamosan az adatokat egy listában tárolni (ami már a számértékeket tárolja), mert ez a vezérlő csak `String` típusokat tárol!

`Items`	Property	A tartalmazott elemek listája, *indexelhető*!
`Items->Add(String ^ mit)`	Method	Ezzel lehet elemet hozzáadni a listához
`Items->Count`	Property	Tartalmazott elemek száma
`SelectedIndex`	Property	A kijelölt indexe, `set()` és `get()`-el is rendelkezdik, `int` típusú
`Click`	Event	Érzekeli ha rákattintottak a vezérlőre
`DoubleClick`	Event	Érzekeli ha duplán kattintottak a vezérlőre
`SelectedIndexChanged`	Event	Érzekeli ha új elem lett kiválasztva

4.10. `OpenFileDialog` vezérlő

Fájlok kiválasztására alkalmas, hogyha majd meg akarjuk őket nyitni.

`FileName`	Property	Megjelenő fájlnév, majd ide kerül a kiválasztott név is!
`Filter`	Property	Beírhatunk támogatott fájltípusokat
`ShowDialog()`	Method	Felugrik az ablak, `System::Windows::Forms::DialogResult` a visszatérési érték

Példa: Létező fájl nevének bekérése `OpenFileDialog`-al

```
openFileDialog1->FileName = "";
openFileDialog1->Filter = "Adatfajlok (*.txt)|*.txt"; //igy lehet pl .txt-re szurni
System::Windows::Forms::DialogResult valasz = openFileDialog1->ShowDialog();
//ha a felhasznalo OK-ot nyomott
if (System::Windows::Forms::DialogResult::OK == valasz)
{
	filename = openFileDialog1->FileName;
	//filename fajlt valasztotta a felhasznalo
}
```

4.11. `SaveFileDialog` vezérlő

Fájlokat tudunk elmenteni ezzel a vezérlővel.

`FileName`	Property	Megjelenő fájlnév, majd ide kerül a kiválasztott név is!
`Filter`	Property	Beírhatunk támogatott fájltípusokat
`ShowDialog()`	Method	Felugrik az ablak, `System::Windows::Forms::DialogResult` a visszatérési érték

4.12. `ContextMenuStrip` vezérlő

Jobb egérkattintán hatására felugró menü, lehet rajta kiválasztható és kattintható elem. *Fontos* hogy hozzá kell rendelni egy vezérlőhoz vagy a formhoz!

Példa: `ContextMenuStrip` hozzárendelése a formhoz.

```
this->ContextMenuStrip = contextMenuStrip1;
```

Az adatokat a `ContextMenuStrip`-re kézzel vigyük fel. A kattintás eseményt az adott elemre való dupla kattintással célszerű létrehozni. A következő táblázat az *egy adott elemhez tartozó* Property és Eventeket tartalmazza.

`CheckOnClick`	Property	Ha `true` akkor ha rákattintunk a pipa megjelenik/eltűnik
`bool Checked`	Property	Be van-e pipálva az adott elem
`Click`	Event	Rákattintottak az elemre

Példa: az Alma elem kipipálásanak ellenőrzése a hozzá tartozó `almaToolStripMenuItem` objektumon keresztül.

```
bool allapot = almaToolStripMenuItem->Checked;
```

4.13. `MenuStrip` vezérlő

Felső sorban lévő menü, lehet rajta kiválasztható és kattintható elem. Az adatokat a `MenuStrip`-re kézzel vigyük fel. A kattintás eseményt az adott elemre való dupla kattintással célszerű létrehozni. A következő táblázat az *egy adott elemhez tartozó* Property és Eventeket tartalmazza.

`CheckOnClick`	Property	Ha `true` akkor ha rákattintunk a pipa megjelenik/eltűnik
`bool Checked`	Property	Be van-e pipálva az adott elem
`Click`	Event	Rákattintottak az elemre

Példa: a Felbontás elem kipipálásanak ellenőrzése a hozzá tartozó `felbontásToolStripMenuItem` objektumon keresztül.

```
bool allapot = felbontasToolStripMenuItem->Checked;
```

4.14. `Timer` vezérlő

Megadott időközönként meghívja a `Tick` eseményt, ha elindítottuk.

`Interval`	Property	`Tick` események gyakorisága `millisec` mértékegységben
`Start()`	Method	Elindítja a `Timer`-t
`Stop()`	Method	Leállítja a `Timer`-t
`Tick`	Event	Az esemény ami adott időnként meghívódik

5. Chartok/Plotok

Az ehhez tartozó vezérlő: `Chart`. A szükséges osztályok a `System::Windows::Forms::DataVisualization::Charting` namespace-n találhatóak.

`Series`	Property	Adatsorokat tartalmazza
`ChartArea`	Property	A diagramterület formázása
`Color BackColor`	Property	A diagram háttérszíne
`Color Legends[i]->Text`	Property	i. adatsorhoz megjelenő szöveg
`Color Legends[i]->BackColor`	Property	A jelmagyarázat háttérszíne

5.1. `ChartArea` referencia osztály

A `Chart` vezérlőhöz tartozik. Ez az a terület, ahol a sorozatok ki lesznek a rajzolva.

`String AxisX->Title`	Property	Az x tengely neve
`double AxisX->Minimum`	Property	Minimum érték a tengelyen
`double AxisX->Maximum`	Property	Maximum érték a tengelyen
`double AxisX->Interval`	Property	Beosztások gyakorisága
`Color BackColor`	Property	Háttérszín pl: `Color::Transparent`
`Clear`	Method	Minden eddigi beállítás törlése

5.2. `Series` referencia osztály

A `Chart` vezérlőhöz tartozik. Ezek a `Chart`-on megjelenő adatsorok.

`String ChartArea`	Property	Itt lehet megadni hogy melyik `ChartArea`-hoz tartozzon
`ChartType`	Property	Pont/vonal: `SeriesChartType::Point/Line`
`Color Color`	Property	A vonal/marker színe
`int MarkerSize`	Property	A pontok mérete `SeriesChartType::Point` esetén
`Points->AddXY(double x,double y)`	Method	`x` és `y` értékek hozzáadása a grafikonhoz
`Points->Add(double y)`	Method	`y` érték hozzáadása a grafikonhoz (`x = 0,1,2 ...`)
`Clear`	Method	Minden eddigi beállítás törlése

Példa: Egy egyszerű `Chart` elkészítése, ha rányomunk egy gombra.

```
private: System::Void buttonRajzol_Click(System::Object^ sender, System::EventArgs^ e)
{
        //eddigi sorozatok es teruletek torlese
        chart1->Series->Clear();
        chart1->ChartAreas->Clear();

    //hatterek atlatszora allitasa
        chart1->BackColor = Color::Transparent;
        chart1->Legends[0]->BackColor = Color::Transparent;

    //igy tobb lapot is hozzaadhatunk
        ChartArea ^ charea = chart1->ChartAreas->Add("Lap1");
        //elkerjuk a chart1 chartAreas objektumat (ez nem kotelezo)
        charea->AxisX->Title = "X";
        //chart1->ChartAreas["Lap1"]->AxisX->Title = "X"; szinten jo lenne
```

```
        charea->AxisY->Title = "Y";
        //charea->AxisX->LabelStyle->Format = "{0.00}ms"; //tengely formazasa

        //range beallitas
        charea->AxisX->Interval = 1;
        charea->AxisX->Maximum = 0;
        charea->AxisX->Minimum = 10;
        charea->BackColor = Color::White;

    //sorozat letrehozas (szinten nem kotelezo)
        Series ^ kijelolt = chart1->Series->Add("Adatok");
        kijelolt->ChartArea = "Lap1"; //ezen a chartArea-n lesz
        kijelolt->ChartType = SeriesChartType::Point;
        kijelolt->Color = Color::Black;
        kijelolt->MarkerSize = 0;
        for(int i = 0; i < Xvals->Count; i++)
        {
                kijelolt->Points->AddXY(Xvals[i],Yvals[i]);
        }
}
```

6. Grafika

A grafikával kapcsolatos osztályok és függvények a `System::Drawing::Drawing2D` névterületen találhatóak A grafika készítés több módon is lehetséges

1. Egy tetszőleges vezérlőre rajzolással, pl: `Panel`, az adott vezérlő `Graphics` objektumának az elkérésével.
2. `Form` grafika objektumának az elkérésével
3. `Bitmap`-re rajzolással, ez kimenenthető lesz, és akkor frissítjük a megjelenítést amikor mi szeretnénk, nem lesz vibráló kép. *Ezt célszerű használni!*

A továbbiakban a 3. módszer lesz használva! A grafika készítés lépesei:

1. `Graphics ^` objektum definiálása, lehetőleg globálisan!
2. Egy `BitMap` objektum létrehozása, amire a rajz készül majd
3. A `Graphics` objektum elkérése a `BitMap`től a `Graphics::FromImage(BitMap)` függvénnyel.
4. Az eddigi dolgok felülírása a `Graphics->Clear(Color)` függvénnyel.
5. Az összes transzformáció visszaállítása a `Graphics->ResetTransform()` függvénnyel.
6. `Bitmap` kiküldése egy `PictureBox` vezérlőre

Ezután egy teljesen üres objektumot kapunk, amire nyugodtan elkezdhetünk rajzolni.

6.1. `Graphics` osztály

A grafika osztály, aminek a segítségével a rajzolás történik, tartalmazza az origót és a tengelyeket, van rengeteg beépített metódus a transzformációkhoz és a rajzoláshoz.

6.1.1. Transzformációk

A transzformációk segítségével a koordinátarendszert eltolhatjuk a nekünk megfelelő helyre, így lényegesen könnyebben tudunk rajzolni.

`ScaleTransform(float x,float y)`	Method	Átskálázás x és y irányba, a megadott faktorral
`TranslateTransform(float x,float y)`	Method	Origó eltolása x és y értékekkel a tengelyek mentén
`RotateTransform(float angle)`	Method	Jobbkéz szabály szerint elforgatja a megadott mátrixot a megadott fokkal
`ResetTransform()`	Method	Alaphelyzetbe állítás

6.1.2. Rajz metódusok

`Clear()`	Rajzterület letörlése egy színnel
`DrawLine(Pen, Point, Point)`	Két pont összekötése vonallak.
`DrawRectangle(Pen,int x, int y1 int w, int h)`	Origó felöli pont[2] `(x,y)` és szélesség, magasság `(w,h)`
`DrawEllipse(Pen,int x, int y, int w, int h)`	Bal felső pont `(x,y)` és szélessége, magassága a befoglaló téglalapnak `(w,h)`
`DrawPolygon(Pen, Point[])`	Sokszög rajzolása, `Point[]` egy `array` a csúcspontokkal
`DrawString(String, Font, Brush, Point)`	Adott `String` kirajzolása, adott `Point` helyre.
`DrawImage(BitMap, int x, int y)`	Kép rajzolása adott `(x,y)` helyre.

A legtöbb `Draw` metódusnak létezik `Fill` párja, a különbség hogy `Pen` helyett egy `Brush` objektumot kell átadni.

6.2. `Bitmap` osztály

Egy kép tárolására alkalmas osztály

`Bitmap(String ^ fajlnev)`	Constructor	Egy képfájl inicializálása
`Bitmap(int width, int height)`	Constructor	Üres `Bitmap` megadott méretekkel

6.3. `PictureBox` vezérlő

Kép megjelenítésére alkalmas vezérlő.

`int Height`	Property	A `PictureBox` magassága
`int Width`	Property	A `PictureBox` szélessége
`Image`	Property	A megjelenő kép, átadhatunk `Bitmap, Image` formátumot

6.4. `Pen` osztály

Toll, amivel vonalakat tudunk húzni a grafika objektumon.

`Pen(Color)`	Constructor	Toll létrehozása megadott színnel
`Color Color`	Property	A toll színe
`float Width`	Property	A toll vastagsága
`DashStyle`	Property	A vonal típusa a `DashStyle enum`-ból.

6.5. `SolidBrush` osztály

Ecset, ami egy adott színnel kitölti a rajzlap megadott régióit.

`SolidBrush(Color)`	Constructor	Ecset létrehozása megadott színnel
`Color Color`	Property	Az ecset színe

6.6. `Color` osztály

Egy osztály amely rengeteg különböző előre definiált színt tartalmaz, de sajátot is létrehozhatunk. A legtöbb rajzzal kapcsolatos osztálynak szüksége van `Color` propertyre.

`Color::Red`	Piros szín, rengeteg féle másik van
`Color::Transparent`	Átlátszó szín
`Color::FromArgb(int red, int green, int blue)`	Saját RGB szín

6.7. `Matrix` osztály

Egy 3×3 mátrixot tartalmaz amely valami transzformációt ír le. Nem kell magunknak kiszámolni a mátrix elemeit. A transzformációkat függvényekkel végezhetjük el. A `System::Drawing::Drawing2D` névterületen helyezkedik el.

`Matrix()`	Constructor	Egységmátrix létrehozása
`Rotate(float angle)`	Method	Jobbkéz szabály szerint elforgatja a megadott mátrixot a megadott fokkal
`Scale(float x,float y)`	Method	Átskálázás x és y irányba, a megadott faktorral
`Translate(float x,float y)`	Method	Origó eltolása x és y értékekkel a tengelyek mentén
`Reset()`	Method	Egységmátrix visszaállítása

6.8. Grafika példa

```
using namespace System::Drawing::Drawing2D;
...
int rudVastagsag = 20;
int szelTavolsag = 30;
Bitmap ^ rajz;
Graphics ^ gr;

void formload(...)
{
    ...
    //gcnew csak egyszer!
    rajz = gcnew Bitmap(width, height);
    ...
}
/*
Ket rud rajzolasa a kepernyo(pictureBox1) ket szelere
*/
void rajzol()
{
       //meretek
```

```
        int width = pictureBox1->Width;
        int height = pictureBox1->Height;

        //rajzeszkozok
        gr = Graphics::FromImage(rajz);
        Pen ^ vekony = gcnew Pen(Color::Black);
        vekony->Width = 2;

        //hatter torol
        gr->Clear(Color::White);
        gr->ResetTransform();

        //origo bal also sarokba transzformal
        gr->TranslateTransform(0,(float)height);
        gr->ScaleTransform(1.0f, -1.0f);

        //szelso rudak
        gr->DrawRectangle(vekony, szelTavolsag, 1, rudVastagsag, height-2);
        gr->DrawRectangle(vekony, width-rudVastagsag-szelTavlsag, 1,
               rudVastagsag, height - 2);

        //rajz megjelenitese
        pictureBox1->Image = rajz;
}
```

7. 3D grafika

Ehhez a http://delta.inflab.bme.hu/∼szakaly/mmsz/ oldalon, a 7. gyakorlat alatt található kódra lesz szükség. Ez tartalmazza a számunkra szükséges vetítési módokat. Nekünk elég a vetítés osztályokból objektumot készíteni, majd ezt átadni a `Draw3DLine` függvénynek. A függvény:

```
Draw3DLine(Graphics ^ grafika, Pen ^ toll, p3D ^ A, p3D ^ B,
               Vetites ^ vetitesmod, WindowViewPort ^ windowViewPort)
```

- `grafika`, objektum amire a rajz készül
- `toll`, amivel rajzolunk
- `A` egyik végpont
- `B` másik végpont
- `vetitesmod`, a kiválasztott vetítésmód (Cavalier, Izometrikus, Centrális)
- `windowViewPort`, transzformáció ami megfelelő helyre teszi a rajzot.

Példa: 3D koordinátatengelyek rajzolása

```
//transzformaciok definialasa
auto cavalier = gcnew Axonometrikus(0.5, 1, 1, 45, 0, 0); //cavalier
auto cent = gcnew Centralis(2); // centralis
```

```
auto izo = gcnew Axonometrikus(1, 1, 1, 30, 30, 0); // izometrikus

//windowviewport, origo kozepre helyezese, nagyitas nelkul
int width = pictureBox1->Width;
int height = pictureBox1->Height;
auto wv = gcnew WindowViewPort(width / 2, height / 2 + 50, 1);

//rajz es grafika elkeszitese
auto rajz = gcnew Bitmap(width, height);
gr = Graphics::FromImage(rajz);
gr->Clear(Color::White);

//toll defimialasa
Pen ^ toll = gcnew Pen(Color::Black, 2.0);

//koordinatarendszer
p3D ^ origo = gcnew p3D(0.f, 0.f, 0.f);
p3D ^ xmax = gcnew p3D(200.f, 0.f, 0.f);
p3D ^ ymax = gcnew p3D(0.f, 200.f, 0.f);
p3D ^ zmax = gcnew p3D(0.f, 0.f, 200.f);

//koordinata tengelyek
Draw3DLine(gr, toll, origo, xmax, cavalier, wv);
Draw3DLine(gr, toll, origo, ymax, cavalier, wv);
Draw3DLine(gr, toll, origo, zmax, cavalier, wv);

//kirajzol
pictureBox1->Image = rajz;
```

8. Egyéb

8.1. `MouseClick` event

Érzékelhetjük ha a felhasználó megnyom egy gombot, szinte bármely vezérlőhöz rendelhetjük. A koordinátákat az `X` és `Y` property-k adják meg, a vezérlő saját koordináta rendszerében. Tehát ha `this` objektumon nézzük a kattintást, de a koordinátákkal egy `PictureBox` objektumra rajzolunk, az katasztrófa lesz.

Példa: A kattintás helyének elkapása a `pictureBox1` vezérlőn.

```
Void pictureBox1_MouseClick(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e)
{
    if (e->Button == System::Windows::Forms::MouseButtons::Left) //bal gomb
    {
        csinald_ezt();
    }
    else if (e->Button == System::Windows::Forms::MouseButtons::Right) //jobb gomb
    {
        csinald azt();
    }
```

```
}                                                                    12
```

8.2. `Random` osztály

Random számok generálására.

`Next(int min, int max)`	Method	Két határ közé eső random szám
`Next(int max)`	Method	Random szám max értékkel

Példa: Random szám generálása

```
Random ^ rand = gcnew Random();                                      1
int szam = rand->Next(10);                                           2
```

8.3. `Math` namespace

Matematikai műveletek a `System::Math` névterület alatt.

Példa: $\sin \pi$ meghatározása

```
double ertek = Math::Sin(Math::PI));                                 1
```

8.4. `Convert` namespace

A konvertáló függvények a `System::Convert` névterület alatt találhatóak. Az összes függvény `To...(bármi)` alakú, szinte bármit tudnak bármibe konvertálni.[3]

Példa: szövegből `double` kiszedése.

```
String ^ szoveg = gcnew String("1.2345");                            1
double ertek = Convert::ToDouble(szoveg);                            2
```

[3] Azért ezzel óvatosan, `String, double, float, int` a többit hagyjuk.