1. UML és kód kapcsolata, Futtatókörnyezetek, .NET szerelvény, .NET biztonsági megoldásai

Milyen fajta biztonsági megoldások vannak a .NET keretrendszerben?

- **Szerep alapú biztonság** (Role based security): A felhasználó rendszerben betöltött szerepén és jogain alapuló biztonsági mechanizmus.
- **Kóderedet alapú biztonság** (Code access security): Képes meghatározni, hogy a kód mit tehet, a meghívott kód mit tehet, valamint képes egyértelműen azonosítani a kódot.

Definiálja és jellmezeze a .NET szerelvény (assembly) fogalmát (szerepe, típusa, azonosítása)!

- általában egy .dll vagy egy .exe, de lehet több is
- IL kód
- metaadatok .NET osztályokból
- erőforrások (.jpg, .txt)

Minden alkalmazás szerelvényekből épül fel:

- egy névtér több szerelvényben is lehet
- egy szerelvényben több névtér is lehet
- hivatkozhat más szerelvényekre

Szerelvény fájlok:

- egy fájl
- memória fájl, pl. scriptelésnél
- több fájl: egy mappában a manifesttel, a biztonságosság miatt hash készül a hivatkozott állományokról

Szerelvény információk: a manifest

- név (általában kiterjesztés nélküli állománynév)
- verzió (major, minor, build number, revision)
- támogatott nyelv és kultúra
- processzor és OS
- szerelvény referenciák: név, verzió, nyilvános kulcs (ha megosztott), hash algoritmus azonosító (ha megosztott), szerelvényhez tartozó modulok listája, hash, egyéb (kiadó, leírás)

Verziókezelés:

- ha új verziót telepítünk, akkor is a régit használja
- a kibocsátó cég átirányíthat
- a rendszergazdák felüldefiniálhatják

1. Privát szerelvény:

- egyetlen alkalmazás használja
- a neve azonosítja
- az alkalmazás mappáiban keresi (konfigurációs állományban más elérési útvonal is megadható)
- egyszerűen telepíthető

2. Azonosított szerelvény (megosztott, erős névvel ellátott)

- több alkalmazás használja --> "dll hell"
- erős név teszi egyedivé: név, fejlesztő cég nyilvános kulcsa, verzió szám, opcionálisan nyelv és kultúra
- digitális aláírás a fejlesztő cég privát kulcsával --> integritásvédelem
- csak azonosított szerelvényekre hivatkozhat
- %WINDIR%\Assembly mappában
- a több alkalmazás által használt komponensek a %WINDIR%\System32 mappában

Adja meg két-három mondatban, hogy mit jelent a szerelvények vonatkozásában a .NET alkalmazások integritásvédelme! (3p)

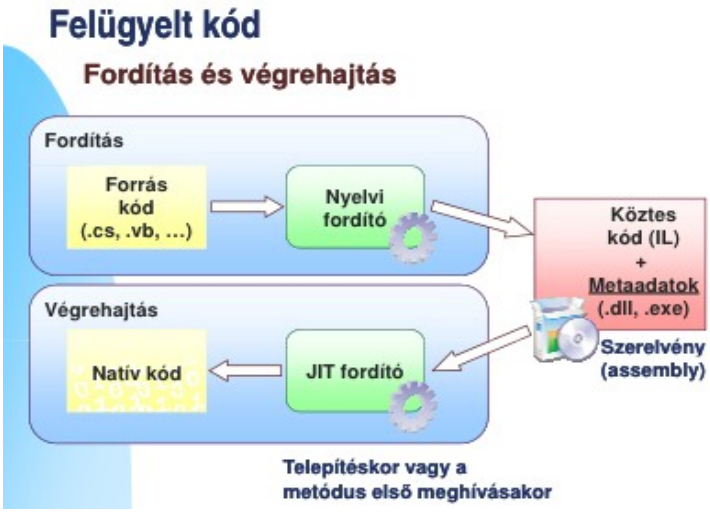
A szerelvények digitális aláírása a fejlesztő cég privát kulcsával
A CLR a szerelvény betöltésekor ellenőrzi, nem sérült-e az integritás

- Nem módosították-e az alkalmazás valamely szerelvényét
- Nem cserélték-e le az alkalmazás valamely szerelvényét

A szerelvény metaadataként szerepel

- A szerelvény aláírása (hash)
- Az aláíráshoz használt titkos kulcs nyilvános párja (az ellenőrzéshez használt betöltéskor)
- Minden hivatkozott szerelvényről részletes információ (ez alapján ellenrizhet, hogy nem cserélték-e le a hivatkozott szerelvényt)!

Ismertesse a fordítási és a végrehajtási (futtatás) lépéseit .NET környezetben. Ennek során adja meg mit értünk IL (Intermediate Language) kód alatt! (6p)



Adja meg a futtatókörnyezetek alkalmazásának három fontos előnyét! (6p)

- Hordozhatóság: IL segítségével n+m (n*m helyett) translator-ra van szükség, hogy n nyelvet m platformra implementáljunk
- Kompaktság: Az IL kód kompaktabb is lehet, mint az eredeti kód
- Hatékonyság: Nem kell az új platformok sajátosságait használni, mindig újra tanulni
- Biztonság: Adat és kód ellenőrzés
- Együttműködés: Többnyelvűség
- Rugalmasság: „metaprograming”, reflection, dynamic code generation, serialization, type

browsing, etc.

2. Modern nyelvi eszközök (Event, Delegate, Property, Attribute)

Lásd gyakorlat jegyzet

3. Eseményvezérelt programozás, Win32 API, Windows Forms

Ismertesse az érvénytelen terület fogalmát. Hogyan kapcsolódik a Paint eseményhez? (7p)

Korábban takarásban levő, újra láthatóvá vált ablakrészek (pl.: átméretezés, Z-orderben előbb került az ablak, stb.)
Ha érvénytelen terület keletkezik, akkor WM_PAINT üzenetet kap az ablak.

Ismertesse röviden a Windows Forms saját vezérlők készítésének a lehetőségét! (7p)

- Control osztályból leszármaztatás. Akkor használjuk, ha egy teljesen új vezérlőelemet szeretnénk létrehozni. Csak a minden Controlra közös tulajdonságokat és műveleteket kapjuk meg. Adhatunk hozzá új tulajdonságokat (property), eseményeket (event). Ez esetben a rajzolás is a mi feladatunk.
- Adott Control osztályból történő származtatás. Akkor használjuk, ha egy már létező vezérlőelemet szeretnénk testre szabni. Csak az új/módosított speciális viselkedést kell megírnunk, a vezérlő alapértelmezett működése megmarad. Adhatunk hozzá új tulajdonságokat, eseményeket.
- Származtatás UserControl osztályból. Vizuálisan elkészíthetjük összetett vezérlőelemeinket, pont úgy, ahogy egy Formot is elkészítenénk. A tartalmazott vezérlőelemek private láthatóságúak. Tipikusan az összetett felhasználói felület modularizálásának eszköze.

4. Folyamatok, Szálkezelés, Konkurens alkalmazások fejlesztése, Generikus típusok

Válasszon ki egy, a kölcsönös kizárást megvalósító konstrukciót, és ismertesse a működését! (4p)
Hasonlítsa össze a Mutex és a C# Lock zárolási konstrukciókat! (4p)
Jellemezze egy mondatban a ReaderWriterLock osztályt. (2p)

Név	Cél	Folyamatok között is?	Sebesség
lock C# utasítás (Monitor.Enter/Monitor.Leave)	Biztosítja, hogy egy adott erőforráshoz/kódrészlethez egyidőben 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
ReaderWriterLock	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

Adja meg a szálbiztos (tread-safe) osztály fogalmát egy-két mondatban! (3p)

Egy osztály szálbiztos, ha úgy lett megírva, hogytöbszálú környezetben is biztonságosan használható

- Maga garantálja a konzisztenciát
- A felhasználás során már nem kell zárolni

Egy mondatban adja meg, miért könnyebb az egy folyamaton belüli szálak kommunikációját megvalósítani, mint a folyamatok közöttit. (3p)

- Folyamatok elszigeteltek, a folyamatok közötti kommunikáció nehéz
- Szálak egy adott folyamaton belül egy címtartományban vannak, így "könnyen" kommunikálnak

Mutasson kódrészletet szál indítására C# vgy Java nyelven! (5p)

```
    class ThreadTestClass
{
    public static void Main(string[] args)
    {
        Thread t = null;
        if (args.Length == 0)
        {
            t = new Thread(new ThreadStart(ThreadMethod1));
            t.Start();
        }
        else
        {
            t = new Thread(new ParameterizedThreadStart(ThreadMethod2));
            t.Start(args[0]);
        }
```

```
    }
    public static void ThreadMethod1()
    {
        Console.WriteLine("Thread without parameter.");
    }

    public static void ThreadMethod2(object param)
    {
        Console.WriteLine("Thread with parameter: {0}", param.ToString());
    }
}
```

Háttérszál indítása:

```
Thread t= new Thread( ... );
t.IsBackground = true;
t.Start();
```

5. Bináris komponensek evolúciója, Adatkezelés

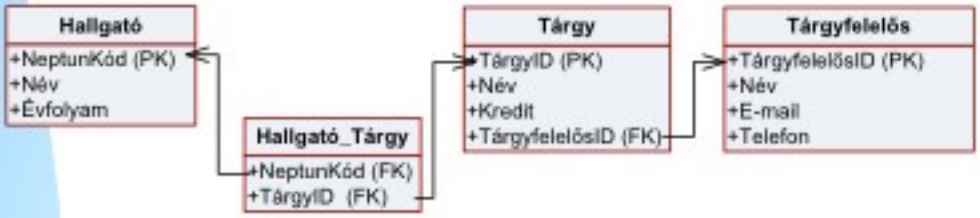
Példán keresztül mutassa be az objektum-relációs leképezést: adjon meg példaként egy osztdiát, amely tartamaz 1-1,1-több, több-több kapcsolatot. Képezze le ezeket adatbázistáblába! (12p)

- **Az adatbázistervezésnek két módszere van**
  - A) „Hagyományos": a relációkat dekompozícióval normalizáljuk. Eddig ezt láttuk.
  - B) Adott egy fogalmi modell (pl. UML osztálydiagram vagy egyed-kapcsolat diagram) → Az osztályokat/entitásokat és a kapcsolataikat képezzük le táblákra.
- **Példa B)-re**
  - Adott az alábbi fogalmi modell (UML osztálydiagram)

  - Ismétlés: első előadáson láttuk, hogy a kapcsolatok „programkódban" hogyan képződnek le (tipikusan legalábbis):
    - Egy-több kapcsolat (Tárgyfelelős-Tárgy): A tárgyfelelős osztályban egy mutató (referencia) lista Tárgy objektumokra, és a Tárgy osztályban egy mutató (referencia) egy Tárgyfelelős objektumra.
    - Több-több kapcsolat (Hallgató-Tárgy): Mindkét osztályban egy mutató (referencia) lista a másik objektumaira.

16

    - Osztály, tagváltozó: Minden osztályhoz felveszünk egy táblát, a tagváltozóknak egy-egy oszlopot.
    - Egy-több kapcsolat (Tárgyfelelős-Tárgy): A többes oldal táblájában (Tárgyf) felveszünk egy idegen kulcsot az egyes oldal táblájának (Tárgyfelelős) elsődleges kulcsára.
    - Több-Több kapcsolat (Hallgató-Tárgy): Felveszünk egy új kapcsolótáblát (Hallgató_Tárgy), amely tartalmaz egy-egy idegen kulcsot az összekapcsolt táblák elsődleges kulcsára.
  - Leképezve adatbázis táblákra



    - Négy táblánk lett: a Hallgató_Tárgy egy kapcsolótábla

Ismertesse egy rövid C# példán keresztül az ADO.NET kapcsolatalapú adathozzáférést! (12p)

```
qlConnection 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();
}
```

6. Tervezési minták

Adja meg röviden, hogy miben és hogyan segítenek a tervezési minták a tervezés során!

A tervezési minták a fejlesztés tervezés fázisában segítenek (az analízis minták az analízis fázisában) Ezekben segítenek a patternek:

- Megfelel objektumokat megtalálni, definiálni
  Objektumok számának, méretének meghatározása
  Objektum interfészek definiálása
  Objektum implementálása
- Újrafelhasználás (design for reuse)
- Változtathatóság, kiterjeszthetség (design for change)

Egy konkrát tervezési mintának az ismertetése

7. Szoftver-architektúrák

Válassza ki, és jellemezze a Model-View-Controller vagy a Document-View architektúrát! Ismertesse az előnyeit-hátrányait!

Model-View-Controller

Felhasználói felülettel rendelkező alkalmazások esetében:

- Model – applikáció objektum
- View – megjelenítésért felelős
- Controller – definiálja, hogy a felhasználói felület hogyan regál a felhasználói bemenetekre

Példa inicializálásra:

1. A főprogram létrehozza a Model-t.
2. A főprogram létrehozza a View-t.
3. A View létrehozza a Controller-t.
4. A View magához csatlakoztatja a Model-t.

Működés:

1. A Controller kezeli az eseményeket.
2. A Controller értesíti az eseményről a Model-t.
3. A Model értesíti megváltozásáról a View-t.
4. A View lekérdezi a Model állapotát, majd megjeleníti azt.

Előnyök:

- többféle nézet ugyanahhoz a modellhez
- függetlenül cserélhető nézetek és vezérlők
- szinkronizált nézetek (mindig aktuális állapotot mutassanak)
- a model megváltoztatása nélkül lehet új nézetet kapcsolni a modellhez

Hátrányok:

- a komplexitás növekszik
- túl sok szükségtelen frissítés
- túl elszigetelt a Model és a Controller --> a megoldás a Document-View architektúra

Document-View

- A view felelős a felhasználói interakciók kezeléséért is.
- A View és a Controller összevonásra kerül.

Ismertesse az információs rendszerek három rétegű architektúráját, melynek során adja meg röviden az egyes rétegek szerepét is. Milyen előnyökkel jár a három rétegű architektúra alkalmazása a két rétegűvel szemben (minimum három szempontot adjon meg)? (14p)

Háromrétegű architektúra

- Külső séma - *alkalmazás*.
- *Alkalmazáslogikai alréteg*: opcionális. Ez egy interfész (homlokzat) a tartománymodellhez. Az interfész illeszkedik a GUI vezérlőelemeihez. A tartomány típusairól logikai típussá konvertál.
- Koncepcionális séma - *tartomány* (üzleti logika).
- Belső séma - *adatbázis*.
- Felépítés:

Alkalmazás	Tartomány	Adatbázis

Előnyök:

- Az alkalmazás kevésbé függ a fizikai adatszerkezettől.
- A referenciális integritás alkalmazásfüggetlenül kezelhető.
- Az adatbázis átszervezhető az alkalmazástól függetlenül.
- Tranzakciókezelés a DBMS feladata – ne nekünk kelljen implementálni.
- Ipari támogatottság (Microsoft, Sun).

Hátrányok:

- Kevesen használják.
- Nagyobb erőforrásigény.
- Többletmunka.
- Az objektumorientált adatbázisok jó felhasználási területe, de ezek teljesítőképessége kétséges.

Kétrétegű architektúra

- Megosztott adatbázis (file-ok, DBMS)
- Alkalmazások (hagymányos vagy 4GL nyelven)
- Felépítés:

Alkalmazás	Megosztott adatbázis

Előnyök:

- Az adatkarbantartás elkülöníthető az alkalmazásoktól.
- A már meglévő adatokat több szempontból (nézetből) is megjeleníthetjük.

Hátrányok:

- Az adatok integritása jól csak az adatbázis szintjén megoldható – ha nincs lehetőség tárolt eljárásokra, akkor ez problematikussá válhat.
- Ha az adatintegritás kezelését „bedrótozzuk” az alkalmazásba, nem lehet megváltoztatni a régi alkalmazásokkal való kompatibilitás miatt.
- Optimalizálás denormalizálással.
- Az alkalmazás az adatbázis fizikai felépítését is figyelembe kell vegye, pedig neki csak a szemantikát kellene.

8. Webalkalmazások fejvesztése

Jellemezze a dinamikus webalkalmazásokat (definíció, kliens oldal, szerver oldal)

- Szerver oldali logikát tartalmaz
- Az oldalakon található információk egy része gyakran adatbázisban tárolódik
- Kliens oldali logikát (JavaScript[2]) tartalmazhat

Kliens oldal

- Tipikusan valamilyen böngésző fut -> HTML kódot, HTTP választ tud feldolgozni
    - Valamilyen szinten implementálja a HTTP szabványt
        - Kommunikálhat más protokollon is a szerverrel
        - Kompatibilitás és tűzfal barátság érdekében
    - Lehet RSS olvasó, letöltő, feltöltő, irodai alkalmazás, webszolgáltatás kliens
    - Lehet oldalba ágyazott kontroll (Flash, Silverlight)
- Letölti a kiegészítő fájlokat (kép, stíluslap, szkript stb.)
- Futtatja a kliens oldali kódot
    - Browser sandboxban
- A felhasználó akciói visszakerülnek a szerverre: postback vagy roundtrip
    - Pl. gomb megnyomására HTTP POST vagy URL paraméterek formájában

Szerver oldal

- Feldolgozza a beérkező HTTP kérést -> input
    - Felhasználó által megadott input (URL, query string, form data)
    - Korábbi műveletek (session, cookie)
    - Kliens paraméterei (pl. User-Agent, Accept-Language header)
- Valamilyen nyelvű kód fut, aminek HTTP választ kell előállítania -> output
- Szerver oldali vagy külső erőforrásokat (adatbázis, web) használhat
- Kliens oldali erőforrásokat nem érhet el (browser sandbox, biztonság!)

Állapotkezelést (session ID) valahol meg kell oldani (HTTP nem támogatja)

- Valamilyen egyedi azonosító alapján
- Kliens oldalon
    - URL paraméter
    - Rejtett mező
    - Cookie
- Szerver oldalon
    - In-process a webszerver memóriájában
    - Külön folyamatban
    - Adatbázisban
- ASP.NET: web.configban tag

ASP.NET kiszolgáló oldali vezérlők(jellemzők,szerepük,működésük,példakód)

- Felületelemek megjelenéséért és viselkedéséért felelnek (TextBox, Label, stb.)
- Feldolgozzák a kliens oldalról érkező adatokat
  - Elérhetővé teszik a böngésző által felküldött felhasználói adatokat (a HTML FORM vezérlőelemek tartalma)
  - Eseményeket generálnak a szerver oldalon
    - Pl. gomb esetén „Click”, TextBox esetén „Changed”
- Generálják a kliensnek küldendő HTML kódot
  - Mind egyszerű HTML elemekre képződik le (például a GridView HTML table-re, a TextBox HTML input-ra, stb.
  - Több böngésző típust is támogathatnak
- Több, mint 50 beépített vezérlőtípus, de kiterjeszthető architektúra
- Deklaratív létrehozás: aspx fájlban runat="server"attribútummal

```
<asp:TextBox ID="tbNumber" runat="server"></asp:TextBox>
<asp:ButtonID="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.

Mik az okos kliens alkalmazások előnyei/hátrányai a webesekkel összehasonlítva? (13p)