Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar

Programozás .NET környezetben

1. gyakorlat

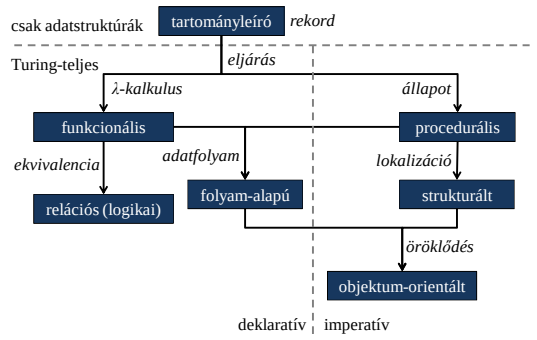
Objektumorientált nyelvek és keretrendszerek

© 2013.02.14. Cserép Máté
mcserep@caesar.elte.hu
http://mcserep.web.elte.hu

Programozási paradigmák
A paradigma jelentősége

- A szoftverek tervezésének és programozásának módszerét nevezzük *programozási paradigmá*nak
  - meghatározza a programozás stílusát és az absztrakciós szintet
- A programozási paradigmákat két csoportba soroljuk:
  - *imperatív*: a program állapotváltozások sorozata, ahol egy állapotot a programban lévő összes változó együttes értéke adja fel, ennek megfelelően az utasításokat szekvenciálisan hajtja végre
  - *deklaratív*: a program a tartalmát, megjelenését írja le, nem pedig a funkció módját, nem alkalmaz változókat, a program működéséhez csak konstans értékeket használ fel, az utasításokat nem szekvenciálisan hajtja végre

Programozási paradigmák
A jelentősebb paradigmák



Programozási paradigmák
A procedurális programozás

- A *procedurális programozás* a felülről lefelé tervezés elvét követi, vagyis a feladatot részfeladatokra bontja
- Ez a koncepció számos problémát felvet:
  - *nem tagolható* kellő mértékben a program
  - az *adatok élettartama* nem elég testre szabható
  - a *feladat módosítása* több funkció módosítását igényelheti, nem elég rugalmas a programszerkezet
- A korlátok kiküszöbölése:
  - a programot a betöltött szerep szerint kell tagolni, az egyes szerepekre külön *programegység*eket létrehozva
  - *függetleníteni* kell az adatokat a főprogramtól, valamint mentesíteni azt a teljes program vezérlésének feladatkörétől

Objektumorientált programozás
Kialakulása

- Megoldás a *felelősség továbbadása*, azaz az adatok továbbadása az őket kezelő programegységek számára
  - így egy programegység alá nem csak a műveletek kerülnek, hanem az általuk manipulált adatok – ezt nevezzük *egységbe zárás*nak (*enkapszuláció*nak) – , így kialakul a programegységen belül a szoros összetartás
  - a programegységek között így már kialakítható egy gyengébb kohézió azáltal, hogy csak bizonyos pontokon engedélyezünk hozzáférést a részegységhez, a többi funkciót, adatot pedig *elrejtjük*
- Ennek eredményeként egy módosítás a feladatban csupán egy programegység módosítását vonja maga után

Objektumorientált programozás
Absztrakció

- Az *absztrakciós szint* megválasztása lehetővé teszi, hogy az adott problémát több szinten kezeljük, azaz az összes lehetséges tulajdonságnak egy alkalmas részhalmazát válasszuk ki, és azokat használjuk a megvalósításban
  - a számunkra lényeges tulajdonságok kiemelését, és ezáltal az objektum meghatározását nevezzük *absztrakció*nak
- *Objektum*nak (object) nevezzük a feladat egy adott tárgyköréért felelős programegységet, amely tartalmazza a tárgykör megvalósításához szükséges adatokat, valamint műveleteket
- Az objektumok viselkedési mintáját – azaz a tárolható adatokat, és azokkal végezhető műveletek halmazát – az *osztály* tartalmazza

Objektumorientált programozás
Objektum-orientált program

- *Objektum-orientált*nak nevezzük azt a programot, amely egymással kommunikáló objektumok összessége alkot
  - minden adat egy objektumhoz tartozik, és minden algoritmus egy objektumhoz rendelt tevékenység, nincsenek globális adatok, vagy globális algoritmusok
  - a program így kellő tagoltságot kap az objektumok mentén
  - az adatok élettartama így összekapcsolható az objektum élettartamával
  - a módosítások általában az objektum belsejében véghezvihetők, ami nem befolyásolja a többi objektumot, így nem szükséges jelentősen átalakítani a programot

Objektumorientált programozás
Alaptulajdonságok

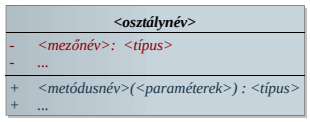
- Az objektum-orientáltság öt alaptényezője:
  - *absztrakció*: az objektum reprezentációs szintjének megválasztása
  - *enkapszuláció*: az adatok és alprogramok egységbe zárása, a belső működés elrejtése
  - *nyílt rekurzió*: az objektum mindig látja saját magát, eléri műveleteit és adatait
  - *öröklődés*: az objektum tulajdonságainak átruházása más objektumokra
  - *polimorfizmus és dinamikus kötés*: a műveletek futási időben történő működéshez kötése, és a viselkedés átdefiniálása

Objektumorientált programozás
Az objektumorientált nyelvek kialakulása

- Az objektumorientált tervezés eszköze a *Unified Modelling Language* (*UML*), amely egy magas szintű tartományleíró nyelv
  - 13 diagramtípus segítségével tervezhető meg a program szerkezete (*statikus tervezés*) és működése (*dinamikus tervezés*)
- A programozási nyelvekben 1967-től megjelent az objektumorientáltság támogatása a *Simula 67*-ben
- Azóta számtalan nyelv biztosított támogatást az objektumorientált paradigma iránt (pl. C++), valamint megjelentek azok a nyelvek, amelyben kizárólag ezt a paradigmát lehetett alkalmazni, utóbbiakat nevezzük *tisztán objektumorientált nyelv*eknek

Objektumorientált programozás
Az UML osztálydiagram

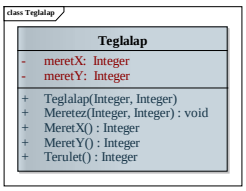
- Az UML diagramok körében az osztályok szerkezetét, a program felépítését az *osztálydiagram* reprezentálja
  - ábrázolja a rendszerben részt vevő osztályokat, azok felépítését (mezők és metódusok bontásában, típusok és láthatóság megjelölésével):



  - az osztályok közötti kapcsolatot relációk formájában, név és multiplicitás megjelölésével

Objektumorientált programozás
Példa

- A téglalap osztály megvalósítása:
  - mezői a két oldal mérete, ezeket elrejtjük
  - műveletei az átméretezés, illetve a méretek és a terület lekérdezése, ezeket láthatóvá tesszük
- A téglalap osztály terve (UML osztálydiagramja):



Objektumorientált programozás
Objektumok közötti kapcsolatok

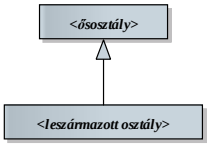
- *Egyszerű kommunikáció* (*asszociáció*): az osztály meghívja más osztály (látható) metódusát, hivatkozik rá a műveletek végrehajtása során, paraméterként, vagy visszatérési értékként

- *Hivatkozás* (*aggregáció*): az osztály egy, vagy több példánya meg van hivatkozva a másik osztály egy, vagy több mezőjében

Objektumorientált programozás
Objektumok közötti kapcsolatok

- *Tartalmazás* (*kompozíció*): az osztály egy példányát a másik osztály tartalmazza egy, vagy több mezőjében

- *Öröklődés*: az osztály egy másik osztály specializációja



Objektumorientált programozási nyelvek
A Smalltalk nyelv

- 1980-ban a *Smalltalk* volt az első tisztán objektumorientált nyelv, és megadta az összes őt követő nyelv alaptulajdonságait, úgymint:
  - teljesen hordozható kódot biztosít *virtuális gép*en történő futtatás segítségével
  - minden implementált elem (változók, konstansok, metódusok) egy objektum, és egyben egy osztály példánya, az osztályok egymástól örökölnek, és egy *teljes származtatási hierarchiá*ban vannak
  - a memóriakezeléshez *szemétgyűjtő*t használ
  - *dinamikus programozás* támogatása, azaz a program futás közben képes manipulálni a programkódot

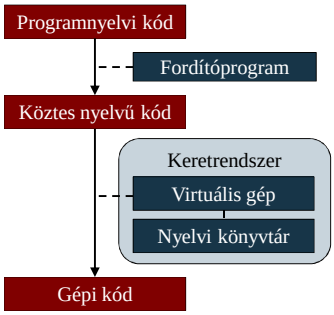
Objektumorientált programozási nyelvek
A virtuális gép

- A hordozhatóság akadálya, hogy a fordítással keletkezett alacsonyszintű programkód gépfüggő, ezért a *Smalltalk* programokat egy gépfüggetlen *köztes nyelv*re (*Intermediate Language*) fordították
- A köztes nyelvű program egy értelmező szoftver segítségével futtatható, amely futás közben alakítja gépi kóddá a programkódot, ezt az értelmezőt nevezzük virtuális gépnek (*Virtual Machine*)
- Ezt a félig fordított, félig értelmezett megoldást nevezzük *futásidejű fordítás*nak, vagy röpfordításnak (*Just In Time Compilation*)
  - a futtatáskor optimalizációk történnek, így az értelmezést követően gyorsabb lehet a futás, mint teljes fordítás esetén

Objektumorientált programozási nyelvek
A szoftver keretrendszer

- Mivel a fordítás egy része, és az összeszerkesztés futási időben történik, ezért nem kötelező minden programkomponenst beágyazni a programba, a hivatkozások feloldása történhet futtatáskor is
  - ezáltal csökkenthető a program mérete és a betöltés ideje
  - a kiemelt komponenseknek jelen kell lenniük a gépen
  - célszerű a beépített könyvtárak kiemelése
- *Szoftver keretrendszer*nek nevezzük a kiemelt programkönyvtár és a virtuális gép együttesét
  - tartalmazza az API gépfüggetlen, absztrakt lefedését
  - felügyeli a programok futásának folyamát
  - biztosítja a memóriakezelést, szemétgyűjtést

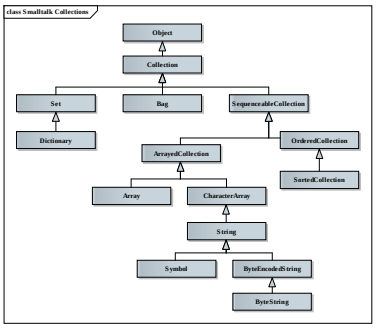
Objektumorientált programozási nyelvek
A szoftver keretrendszer



Objektumorientált programozási nyelvek
Teljes származtatási hierarchia

- Teljes származtatási hierarchiában van egy egyetemes ős osztály, minden más osztály ennek leszármazottja (akkor is, ha nincs megjelölve ősként)
  - az ős definiálja az alapértelmezett viselkedését minden objektumnak (pl. szöveggé alakítás, másolás,…)
  - minden beépített osztály egy származtatási hierarchia mentén van implementálva, és ezek között absztrakt osztályok is találhatóak
  - ezen láncok mentén specializálódnak, illetve kiegészülnek az osztályok viselkedései
  - a hierarchia mentén találhatóak megkötések is a származtathatóságra és a felüldefiniálhatóságra

Objektumorientált programozási nyelvek
Teljes származtatási hierarchia



Objektumorientált programozási nyelvek
Memóriakezelés

- Az objektumorientált nyelvekben célszerű referenciákon, illetve mutatókon keresztül hivatkozni az objektumokra, hatékonysági és élettartam szabályozási okok folytán
  - mutatók esetén a létrehozást és törlést manuálisan kell megírni, ám a törlés sokszor elmarad, ezért a memóriában maradnak a felesleges, nem hivatkozott objektumok (a szemét), ezért célszerű ezt automatikusan elvégezni
- A virtuális gép feladata, hogy felügyelje a program által elfoglalt memóriaterületet, és a benne lévő memóriaszemetet eltávolítsa, ezt nevezzük *szemétgyűjtés*nek (*Garbage Collection*)
  - ciklikusan ellenőrzi a memóriát és a hivatkozásokat, és kiüríti a nem hivatkozott memóriaterületeket

Objektumorientált programozási nyelvek
Metaosztályok és dinamikus programozás

- Mivel minden objektum, maguk az osztályok is objektumok, és az ő viselkedési mintájukat is definiálni kell, erre a célra szolgálnak a *metaosztályok*
  - közös felületet biztosít osztálytulajdonságok és azok részleteinek lekérdezésére, metódusok futtatására, módosítására és példányosításra
  - az objektumok és osztályok lehetőséget adnak a metaosztály lekérdezésére és az osztályhoz tartozó *metaobjektum* létrehozására
  - a metaosztály is egy objektum, a típusa szintén metaosztály
- A metaobjektumokon át bármely osztályt módosíthatunk futás közben, ezáltal dinamikusan programozhatóvá válik a rendszer

Objektumorientált keretrendszerek
A Java

- 1991-ben indult el a *Java* fejlesztése a *Sun Microsystems*-nél, amelynek célja egy objektumorientált, általános célú, hordozható kódú, sokplatformos programozási nyelv megalkotása volt
  - könnyű programozhatóságot, ugyanakkor lassabb programfuttatást eredményezett, a programok a Java virtuális gépen (*JVM*) futottak
  - pár év alatt, különösen a mobil és webes alkalmazásoknak köszönhetően nagy népszerűségre tett szert
  - a *Microsoft* saját virtuális gépet, és nyelvi könyvtárat fejlesztett ki (*Virtual J++*), amely gyorsabb futtatást tett lehetővé

Objektumorientált keretrendszerek
A .NET Framework

- 1998-tól a Microsoft új célja a futásidejű fordítás kiterjesztése a létező Microsoft nyelvekre (Visual C++, Visual Basic), ezáltal egy közös virtuális gépen futhatnak a programok
  - egységes köztes nyelv vezethető be, így a nyelvek tetszőleges mértékben kombinálhatóak egymással
  - egységes programkönyvtárak használhatóak
- 2001-re készült el a *.NET Framework*, és a dedikáltan ráépülő nyelv, a C#
  - azóta integrált része a Windows rendszereknek
  - hozzáférést biztosít az összes Microsoft technológiához (*COM*, *ODBC*, *DirectX*)

Objektumorientált keretrendszerek
A .NET Framework

- Egységes virtuális gép: *Common Language Runtime* (*CLR*)
- Egységes köztes nyelv: *Common Intermediate Language* (*CIL*)
- Egységes típusrendszer: *Common Type System* (*CTS*)
- Teljeskörű programkönyvtár
  - *Base Class Library* (*BCL*): gyűjtemények, I/O kezelés, adatkezelés, hálózat, párhuzamosítás, XML, …
  - *Framework Class Library* (*FCL*): WinForms, ASP.NET, LINQ, WPF, WCF
- Biztosítja a programok védettségét és hordozhatóságát
  - memóriakezelés felügyelete: *Managed Code*
  - köztes kód védelme: *Code Access Security* (*CAS*)

Objektumorientált keretrendszerek
A .NET Framework története

- 1.1 (2003): megnövelt *ASP.NET* funkcionalitás, biztonság. Compact Framework megjelenése. ODBC and Oracle adatbázisok kezelésének beépített támogatása.
- 2.0 (2005): eléri a Java funkcionalitását
  - gyorsabb programfuttatás, 64 bites támogatás
  - sablonok, parciális osztályok, névtelen metódusok támogatása, iterátorok
  - új adatkezelés (ADO.NET)
- 3.0 (2007): új technológiák a kommunikációra, a megjelenítésre, a modellezésre és a hitelesítésre. (WCF, WPF, WF, CardSpace)

Objektumorientált keretrendszerek
A .NET Framework története

- 3.5 (2008): funkcionális programozás, nyelvbe ágyazott lekérdezések (LINQ), AJAX támogatás. Az Entity Framework és ASP.NET MVC keretrendszer bevezetése.
- 4.0 (2010): párhuzamosítás automatizálása (PLINQ, TPL), szerződés alapú programozás (DbC), fejlettebb natív matematikai támogatás (pontosság, komplex számok)
- 4.5 (2012): megjelenik a Windows 8
  - új megjelenítési technológia (Metro felület) és hozzá kapcsolódó keretrendszer (Windows Runtime)
  - nyelvi szintű párhuzamosítás
  - HTML5 + JavaScript alapú fejlesztés
  - az ASP.NET vezérlői támogatják a HTML5-öt

Objektumorientált keretrendszerek
A .NET Framework nyelvei

- Összesen 59 nyelv biztosít támogatást a .NET keretrendszer felé, a hivatalosan támogatott nyelvek:
  - *C#*: a Visual J++ utódnyelve, amely eltávolodik a Java szintaxisától, több ponton visszatér a C++-hoz
  - *Visual Basic .NET*: a Visual Basic továbbfejlesztése
  - *C++/CLI (C++.NET)*: C++ szintaxis kiegészítve a .NET könyvtárakra memóriafelügyelettel
  - *F#*: funkcionális programozási nyelv
  - *J#*: a Visual J++ utódnyelve, megmarad a teljes Java szintaxisnál, a .NET környezetre építve (ma már nem használatos)
  - *JScript.NET*: JavaScript alapú szkriptnyelv

Objektumorientált keretrendszerek
Kritika a .NET-tel szemben

- A futásidejű fordítás miatt
  - szükséges a keretrendszer megléte
  - lassúbb programindítás (nagyságrendi elmaradás a fordított programokhoz képest, azonban jelentős előny a Java-val szemben)
  - a köztes kód teljes mértékben visszafejthető bármely felsőbb szintű nyelvbe (ez valamelyest kivédhető kódzavaró eszközökkel, de nem teljesen)
- A memóriafelügyelet miatt
  - megnövelt erőforrásigény (mutatók folyamatos ellenőrzése)
  - késleltetés a szemétgyűjtő futásakor (valós idejű alkalmazások implementálása lehetetlen)

Objektumorientált keretrendszerek
A .NET Framework kapcsolódó keretrendszerei

- Silverlight: webböngészők, Windows Phone
- XNA Framework: Xbox, Windows Phone
- Windows Runtime: Windows RT és Windows 8
- Mono projekt
  - Mono Framework: Linux és OS X
  - Moonlight: Linux és OS X (Silverlight)
  - MonoTouch: iOS
  - Mono for Android
- Kinect SDK: Xbox, PC