

Objektumorientált szoftvertervezés

1. Java ismételés, szálak,
kollekciók, reflection

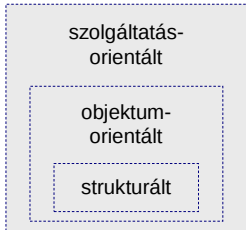
Az objektumorientáltság helye

- *Entia non sunt multiplicanda praeter necessitatem*
 - William Ockham, 1288-1347
- egyszerűség
 - KISS: keep it simple, stupid
- nincs jó vagy rossz
 - használhatóság a kérdés
- mindig a célnak megfelelő eszközt használjuk

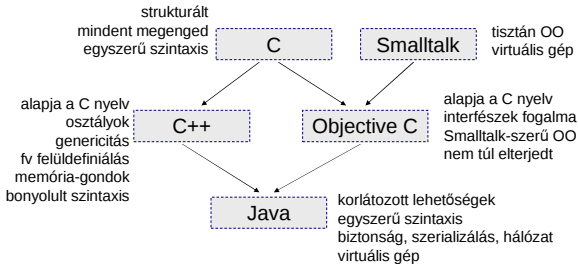


Az objektumorientáltság helye 2

- gépi kód
- makrók
- strukturált programozás
- objektumorientáltság
- szolgáltatásorientáltság



A Java története



Write Once, Run Anywhere

- C, C++, stb:

- ☐ *write once, compile everywhere*

- Java:

- ☐ forrás fordítása bytecode-ra, amit
- ☐ virtuális gép futtat
- ☐ nem kell újrafordítani

- write once, debug everywhere

- ☐ fontos a jó tervezés
- ☐ könnyű platformfüggetlen alkalmazást készíteni

Alapvető jellemzők

- (Majdnem) minden osztály
- Kétfajta típus
 - primitív (int, double, boolean, ...): változóban az érték
 - objektum (String, Vector, ...): változóban a referencia
- Szintaxis erősen C-ből örököelve
 - operátorok (+, -, >>, ...)
 - vezérlési szerkezetek (for, while, switch)
 - függvényhívás
- De
 - *nincs pointer, nincs goto*
 - *külön byte, char és boolean típus*

Alapvető jellemzők 2

- Tömbök is objektumok
 - hossz (length) → run-time ellenőrzés
- Csak érték szerinti paraméterátadás
 - pointer helyett referencia: *holder* osztályok

```
class StringHolder {  
    public String value;  
}
```



Objektumok, Osztályok és Interfészek

Osztályok

- **egységbezárás (attribútumok és metódusok)**
 - *public, protected, private, package* láthatóság
 - *static*: osztályszintű attribútum vagy metódus
- **csak egyszeres öröklés**
 - csak virtuális függvények
 - *abstract*: meg kell valósítani
 - *final*: nem lehet felüldefiniálni

Virtuális függvények

```
class A {  
    public void foo() { System.out.println("A"); }  
    public void bar() { foo(); }  
}  
class B extends A {  
    public void foo() { System.out.println("B"); }  
}  
...  
A a1 = new A();    // statikus típus A, din. tip. A  
A a2 = new B();    // statikus típus A, din. tip. B  
a1.foo();           // "A"  
a2.foo();           // "B"  
a1.bar();           // "A"  
a2.bar();           // "B"
```

Object őszosztály

- Mindenkinek az őse
- Metódusok:
 - **boolean equals(Object o)**
 - tartalom alapú összehasonlítás
 - vö: `a == b` `a.equals(b)`
 - **int hashCode()**
 - hash kulcs generálása
 - **void finalize()**
 - mint C++ destruktork, GC hívja

Object őssosztály 2

■ Metódusok (folyt.)

- **void wait(...)**

- hívó szál az objektum váró sorába kerül

- **void notify()**

- **void notifyAll()**

- objektum sorában váró szál(ak)at felébreszti

- **Class getClass()**

- reflection-höz szükséges metaadatot adja vissza

Object őszosztály 3

■ Metódusok (folyt.)

□ **String toString()**

- string reprezentációt állít elő
- főleg debug-nál hasznos
- meghívódik, ahol String-et várunk

```
"az autóm: "+myCar+";"
```

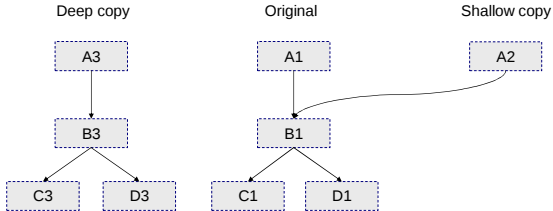
□ **Object clone()**

- az objektum másolatát adja vissza

Másolás

- **Cloneable** interfész megvalósításával
- **Object.clone()** felüldefiniálásával
 - **super.clone()**-t illik meghívni
- *Shallow copy*
 - csak a referenciákat másoljuk
 - pl. Vector másolata ugyanazokra az objektumokra hivatkozik
- *Deep copy*
 - rekurzív másolás
 - pl. C++ string példa

Másolás 2



Másolás 3

```
public class A {  
    B b;  
    ...  
    Object clone() { // shallow  
        A a2 = new A();  
        a2.b = b;  
        return a2;  
    }  
    Object clone() { // deep  
        A a3 = new A();  
        a3.b = b.clone();  
        return a3;  
    }  
    ...  
}
```

Boxing

- Java 5.0 óta
- Primitív típusok és csomagolóik automatikus konverziója

```
Vector v = new Vector();  
int a = 5;  
v.add(a);           // itt valójában egy  
                    // Integer kerül a vektorba  
Integer b = 3;      // létrejön egy új Integer  
a = a + b;          // itt b-ből int lesz: a=8
```

Boxing 2

- Csak akkor használjuk, ha muszáj
- Nem hatékony

```
public static void main(String args[]) {  
    Integer i1 = Integer.parseInt(args[0]);  
        // boxing  
    Integer i2 = Integer.parseInt(args[0]);  
        // boxing  
    i1.equals(i2) // igaz, nincs boxing  
    i1 == i2;     // csak ha -128 < i1 ≤ 127  
    i1 <= i2;     // igaz, boxing  
    i1++;         // növel, boxing  
    i1 > 120;     // boxing  
}
```

Interfészek

- Csak a metódusok deklarációja

- nincs semmifajta implementáció

- Fontos az újrahazsnáláshoz

- Attribútumai lehetnek

- automatikusan *public static final* (globális konstans)

```
public static final int maxLength = 100;
```

- Többszörös öröklés (interfészek között)

- csak egyértelmű attribútumokkal

- Többszörös megvalósítás

- Nem ugyanaz, mint az absztrakt osztály!

Belső osztályok

- Osztályon belül van definiálva
 - akár metóduson belül is
- Szintek száma nincs megszabva
- Eléri a külső osztály(ok) attribútumait és metódusait
- Céljuk: egységbezárás
 - kis segédosztályok a főosztály közelében
- Formális paraméterek, lokális változók vagy kivételkezelő paraméterek: kötelezően *final*

Tagosztály (*member class*)

- van saját neve
- elérhető kívülről (láthatóságától függően)
- közvetlenül a külső osztályban van deklarálva
- static vs. non static

```
class In1 {  
    int k;  
    class In2 {int x = k;}  
    void bar() {  
        In2 i2 = new In2();  
        k = i2.x++;  
    }  
}
```

```
...  
In1 i1 = new In1();  
i1.k++;  
In1.In2 i3 =  
    new In1().new In2();  
...
```

Lokális osztály

- van saját neve
- blokkon belül van deklarálva, csak innen érhető el

```
public class Test {  
    int i = 10;  
    void xxx() { i++; }  
    void foo(final int a) {  
        class In1 {  
            int k = a;  
            int j = i;  

```

```
        void bar() {  
            k = i++;  
            xxx();  
        }  
    }  
    In1 i1 = new In1();  
    i1.j++;  
}
```

Kombinált példa

```
public class Test {  
    int i = 10;  
    void xxx() { i++; }  
    void foo(final int a) {  
        class In1 {  
            int k = a;  
            int j = i;  
            class In2 {  
                int x = k;  
                int y = i;  
                int z = a;  
            }  
        }  
    }  
}
```

```
void bar() {  
    In2 i2 = new In2();  
    k = i2.z++;  
    xxx();  
}  
}  
In1 i1 = new In1();  
i1.j++;  
In1.In2 i3 =  
    new In1().new In2();  
}  
}
```

Anonim osztályok

- nem *abstract*, nem *static*, mindig *final*
- nincs deklarált konstruktora

```
public class MyFrame extends Frame {  
    MyFrame() {  
        super("MyFrame");  
        addWindowListener(new WindowListener() {  
            public void windowClosing(WindowEvent e) {  
                System.exit(0);  
            }  
            ... // a többi metódus  
        })  
    }  
}
```



Memóriakezelés

Memóriakezelés

■ C már szenvedett a memóriahibáktól

- pointerek + aritmetika

`a[3] ≡ *(a+3) ≡ *(3+a) ≡ 3[a]`

- void*

- malloc/calloc/realloc/free

■ C++ próbált javítani, de inkább rontott

- copy konstruktor

- virtuális destruktork

- értékadás

- new/delete

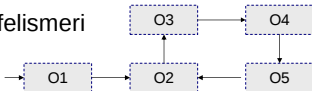
```
class C : A, virtual B {  
    int l; Complex c;  
public:  
    C(Complex k, int i)  
        : A(i), c(k), l(i)  
        { l++; }  
};
```

Memóriakezelés

- Java-ban beépített Garbage Collector (GC)
 - **new** van: heap-en foglal helyet
 - **delete** nincs: GC szabadít fel
- Azt törli, amire már senki sem hivatkozik
 - **void finalize()** meghívódik
- Explicit indítás:
 - **System.gc()** vagy **Runtime.gc()**

Garbage collector

- Gyűrűket is felismeri



- Defragmentálást is végezhet

- ☐ csúsztatás, rejtett kettős indirekció
- ☐ másolás, dupla tár

- Optimalizálás is kellhet

- ☐ http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html



Szálkezelés

Szálkezelés

- Beépített tulajdonság
 - `java.lang.Thread`, `java.lang.Runnable`
- Prioritásos ütemezés
 - nem szigorú
- Csoportok (**ThreadGroup**)
 - szálak egységes kezeléséhez
- Konkurens memóriahasználat
 - kölcsönös kizárás monitorral

java.lang.Thread

- **run()**

- ☐ a szál futtatható kódja

- **start()**

- ☐ a szál indítása

- **yield()**

- ☐ a többi szálnak adja át a futási jogot

- **sleep(long millis [, int nanos])**

- ☐ alvó állapotba viszi a szálat

java.lang.Thread 2

■ **interrupt()**

- megszakítja a szál várakozását
- pl. wait, sleep, stb esetén **InterruptedException**

■ **suspend()** *deprecated*

- a szál futását felfüggeszti

■ **resume()** *deprecated*

- a szál futását újra engedélyezi

■ **stop()** *deprecated*

- megállítja a szálát

java.lang.Thread 3

■ stop implementálása

```
private volatile Thread blinker;  
MyThread(){ blinker = Thread.currentThread(); }  
public void stop() { blinker = null; }  
public void run() {  
    Thread thisThread = Thread.currentThread();  
    while (blinker == thisThread) {  
        try {  
            thisThread.sleep(interval);  
        } catch (InterruptedException e){}  
        repaint();  
    }  
}
```

java.lang.Thread 4

■ `join([long millis [,int nanos]])`

- az adott szál befejeződésére (adott ideig) vár

■ `setDaemon(boolean on)`

- daemon szálként lesz indítva
- a JVM leáll, ha már csak daemon szálak futnak
- hívás az indítás előtt

java.lang.Thread 5

- **int getId()**

- ☐ a szál azonosítója

- **set/getName()**

- ☐ szál neve

- **set/getPriority()**

- ☐ szál prioritása

- **boolean isDaemon()**

- ☐ daemon-e

java.lang.Thread 6

- **boolean isAlive()**
 - fut-e még
- **static Thread currentThread()**
 - az aktuális szál referenciája
- **static int activeCount()**
 - a szál csoportjában aktív szálak száma
- **ThreadGroup getThreadGroup()**
 - a szál csoportját adja vissza

Szál készítése

■ A Thread osztályból származtatunk

```
public class MyThread extends Thread {  
    int a;  
    int b;  
    public MyThread(int i) { b=i; }  
    public void run() {  
        for (a = 0; a < b; a++) { System.out.println(a); }  
    }  
}
```

```
MyThread mt = new MyThread(1000);  
mt.start();  
mt.join();
```

Szál készítése 2

■ A **Runnable** interfészt valósítjuk meg

```
public class MyThread implements Runnable {  
    int a;  
    int b;  
    public MyThread(int i) { b=i; }  
    public void run() {  
        for (a = 0; a < b; a++) { System.out.println(a); }  
    }  
}
```

```
MyThread mt = new MyThread(1000);  
Thread t = new Thread(mt); // kell egy szál, ami futtat  
t.start();  
t.join();
```

Kölcsönös kizárás

- minden objektumhoz egy monitor
 - JVM-ben *lock* és *unlock* utasítás
 - lock/unlock mindig párban
- egyszerre egy szál lehet a monitorban
- többi szál várakozási sorban
- rekurzív belépés engedélyezett
- **static boolean holdsLock(Object o)**
 - igaz, ha az aktuális szálnál van a monitor

Kölcsönös kizárás 2

- megvalósítás a **synchronized** kulcsszóval

```
Hashtable<String, Integer> ht = ...;  
public void increment(String s) {  
    ...  
    synchronized (ht) {  
        int i = ht.get(s);  
        i++;  
        ht.put(s,i);  
    }  
    ...  
}
```

Kölcsönös kizárás 3

■ **synchronized**

□ lehet blokk előtt

- ekkor objektum referencia kell

□ metódus előtt

- ekkor a monitor azé az objektumé, akin a metódust hívjuk
- egyenértékű a teljes metódust felölelő blokkal

```
void foo() {  
    synchronized(this) {  
        ...  
    }  
}
```

```
synchronized void foo() {  
    ...  
}
```

Volatile

```
class Test {  
    static int i = 0, j = 0;  
    static void one() { i++; j++; }  
    static void two() {  
        System.out.println("i=" + i + " j=" + j);  
    }  
}
```

- Egy-egy szál hívja **one()**-t és **two()**-t
- Előfordul, hogy **j** hamarabb nő, mint **i**
 - a változók értéke nem biztos, hogy időben frissül
 - a környezet optimalizálhatja a frissítés sorrendjét

Volatile 2

```
class Test {  
    static int i = 0, j = 0;  
    static synchronized void one() { i++; j++; }  
    static synchronized void two() {  
        System.out.println("i=" + i + " j=" + j);  
    }  
}
```

- Garantált, hogy **two()** mindig egyforma **i**-t és **j**-t lát
- Nem biztos, hogy hatékony

Volatile 3

```
class Test {  
    static volatile int i = 0, j = 0;  
    static void one() { i++; j++; }  
    static void two() {  
        System.out.println("i=" + i + " j=" + j);  
    }  
}
```

- Garantált, hogy **i** növelése mindig atomi, és megelőzi **j** növelését, mert egyből frissül
- Lehet, hogy **two()** különböző értékeket nyomtat
 - ha az értékek elkérése közben fut le **one()**

Konkurens double, long kezelés

- double, long: 64 bites értékek
- a JVM *load*, *store*, *read*, *write* műveletei 32 bites word-on dolgozhatnak
 - emiatt egy egyszerű double értékkezeléséhez két művelet kellhet
 - több szál esetén hazárdos érték fordulhat elő
- ha a változó *volatile*, akkor a fenti műveletek garantáltan atomiak
 - több szál esetén is mindig korrekt értékek

Szálak szinkronizálása

■ **Object.wait([long millis [,int nanos]])**

- a hívó szál vár, amíg nem értesítik vagy timeout
- a szálnak az objektum monitorában kell lennie
- a várakozás alatt a monitorból kilép

```
synchronized (obj) {  
    ...  
    try {  
        obj.wait(); // monitor szabad  
    } catch (InterruptedException ie) {...}  
    ...  
}
```

Szálak szinkronizálása 2

■ **Object.notify()**

- felébreszt egyet az objektumon váró szálak közül
- a szálnak az objektum monitorában kell lennie
- a felébresztett szál csak akkor folytatja futását, ha a hívó szál kilépett a monitorból

■ **Object.notifyAll()**

- felébreszti az objektumon váró összes szálát
- a kölcsönös kizárás érvényesül
- a szálnak az objektum monitorában kell lennie

Szálak állapotai

■ NEW

- újonnan létrehozva

■ RUNNABLE

- futásra kész

■ BLOCKED

- monitorra vár

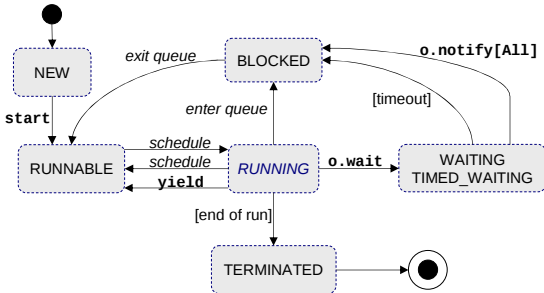
■ WAITING, TIMED_WAITING

- várakozó szál (*Object.wait()*)

■ TERMINATED

- befejezte a működését

Szálak állapotai



Példa: Szemafor pseudokód

```
Init(Semaphore s, Integer v){  
    s := v;  
}  
P(Semaphore s) {    // Acquire Resource  
    wait until s > 0,  
    then s := s-1;  
    /* must be atomic once s > 0 is detected */  
}  
  
V(Semaphore s) {    // Release Resource  
    s := s+1;        /* must be atomic */  
}
```

Példa: Szemafor Javaban

```
public class Semaphore {  
    private int s;  
    public Semaphore (int v) {  
        s=v;  
    }  
    synchronized public void P()  
        throws InterruptedException {  
        while (s <= 0) {wait();}  
        s--;  
    }  
  
    synchronized public void V() {  
        s++;  
        notify();  
    }  
}
```



Genericitás

Genericitás

- Cél: ugyanaz a kód többféle típusra is egyformán működjön
- C megoldásai
 - **void* malloc(size_t s)**
 - kasztolgatás elég veszélyes
 - a hiba nem biztos, hogy kiderül
 - **#define max(a,b) ((a)>(b)?(a):(b))**
 - mellékhatások megnehezítik a használatát

Genericitás 2

- Cél: ugyanaz a kód többféle típusra is egyformán működjön
- C++ megoldása
 - `template <class T> T max(T a, T b) {
 return (a>b)?a:b;
}`
 - szemantikai megkötések `T` típusra, doksi fontos
 - fordításkor kiderül a hiba

Genericitás 3

- Cél: ugyanaz a kód többféle típusra is egyformán működjön
- **Java** eredeti megoldása (pre 5.0)
 - minden objektum az Object leszármazottja
 - **C-s void*** filozófia új köntösben
 - kód tele van kasztolásokkal
 - hiba csak futási időben derül ki

Generikus osztályok

- Java 5.0 óta vannak
- C++ template-hez hasonlók
 - De: nincs minden paraméterhez új osztály
- egyértelműen jelzik az elvárt interfészt
- programozó tudja, mit várunk el
- fordítási időben hibajelzés
- kasztolás-mentes kód

Egyszerű generikus példa

```
public class Fifo {  
    Vector v;  
  
    public Fifo() {  
        v = new Vector();  
    }  
    public void push(Object o) {  
        v.add(o);  
    }  
    public Object pop()  
    throws NoSuchElementException {  
        try { return v.remove(0); }  
        catch (Exception e) {throw new  
            NoSuchElementException();}  
    }  
}
```

■ Object-tel

```
Fifo s =  
    new Fifo();  
s.push(1);  
s.push(2);  
int i1 =  
    (Integer)f.pop();  
int i2 =  
    (Integer)f.pop();
```

Egyszerű generikus példa

```
public class Fifo<E> {  
    Vector<E> v;  
  
    public Fifo() {  
        v = new Vector<E>();  
    }  
    public void push(E e) {  
        v.add(e);  
    }  
    public E pop()  
    throws NoSuchElementException {  
        try { return v.remove(0); }  
        catch (Exception e) {throw new  
            NoSuchElementException();}  
    }  
}
```

■ <E>-vel

```
Fifo<Integer> f =  
    new Fifo<Integer>();  
f.push(1);  
f.push(2);  
int i1 =  
    f.pop();  
int i2 =  
    f.pop();
```

Gondok az örökléssel

```
void printFifo(Fifo<Object> of) {  
    while (!of.isEmpty()) {  
        System.out.println(of.pop());  
    }  
}
```

```
Fifo<String> sf = new Fifo<String>();  
Fifo<Object> of = sf;  
of.push(new Integer(13));
```

- **Fifo<Object>**-nek a **Fifo<String>** nem leszarmazottja!
- Inkompatibilis a típusuk

Gondok az örökléssel (tömb)

```
Object[] oa = new String[10];  
oa[0] = "Hello";  
oa[1] = new Integer(2);  
//ArrayStoreException
```

- **Object[]** helyettesíthető pl. **String[]**-bel
- De csak a dinamikus típusnak megfelelő objektumokat tárolhat

Dzsóker (*wildcard*): ?

```
void printFifo(Fifo<?> of) {  
    while (!of.isEmpty()) {  
        System.out.println(of.pop());  
    }  
}
```

```
Fifo<String> sf = new Fifo<String>();  
Fifo<?> of = sf;  
of.push(new Integer(13)); // ford. hiba  
of.push("Hello");        // ford. hiba
```

- <?> konvertálható bármivé, így <Object>-té is
- visszafele nem megy

Kötött dzsóker: *extends*

```
void pushAll(Fifo<E> s) {  
    while (!s.isEmpty()) { push(s.pop()); }  
}
```

- Mi legyen, ha **s** típusa **Fifo<F>**, ahol **F** az **E** leszármazottja?

```
void pushAll(Fifo<? extends E> s) {  
    while (!s.isEmpty()) { push(s.pop()); }  
}
```

Kötött dzsóker: *super*

```
public E popTo(Fifo<E> f1,  
              Fifo<E> f2){  
    E last = null;  
    while (!f1.isEmpty()){  
        last = f1.pop();  
        f2.push(last);  
    }  
    return last;  
}
```

```
Fifo<String> fs = ...;  
Fifo<Object> fo = ...;  
String last = popTo(fs, fo); // hibás hívás
```

Kötött dzsóker: *super* 2

```
public E popTo(Fifo<? extends E> f1,  
              Fifo<E> f2){  
    E last = null;  
    while (!f1.isEmpty()){  
        last = f1.pop();  
        f2.push(last);  
    }  
    return last;  
}
```

```
Fifo<String> fs = ...;  
Fifo<Object> fo = ...;  
String last = popTo(fs, fo); // hibás visszatérés
```

Kötött dzsóker: *super* 3

```
public E popTo(Fifo<E> f1,  
               Fifo<? super E> f2){  
    E last = null;  
    while (!f1.isEmpty()){  
        last = f1.pop();  
        f2.push(last);  
    }  
    return last;  
}
```

```
Fifo<String> fs = ...;  
Fifo<Object> fo = ...;  
String last = popTo(fs,fo); // hibátlan működés
```

Generikus metódusok

```
static public <E> E popTo(Fifo<E> f1,  
                          Fifo<? super E> f2){  
    E last = null;  
    while (!f1.isEmpty()){  
        last = f1.pop();  
        f2.push(last);  
    }  
    return last;  
}
```

Többszörös típusok

- Mi van akkor, ha a generikus típus egyszerre kell, hogy **X** és **Y** interfészű legyen?

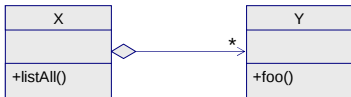
```
<T extends X & Y>
```

```
public static  
<T extends E & Comparable<? super F>>  
T max(Collection<? extends T> coll)
```



Kollekciók

Kollekciók



Kollekciók: tömb

- beépített típus
- mérete rögzített
 - C/C++ nem ellenőriz, van **realloc**
 - Java ellenőriz, nincs **realloc**
 - objektumok (referenciák) másolása elkerülhetetlen
- típuskompatibilitás öröklésnél kérdéses
- kényelmes leírni

Kollekciók: dinam. adatszerk.

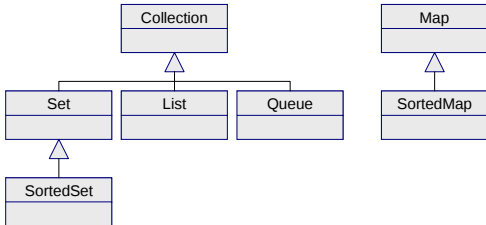
- Láncolt lista (☺ nád © Goldschmidt)
 - egy- vagy kétirányú, gyűrű, strázsa, fésűs lista
- Bináris (n-es fa) fa
 - egyensúly, AVL fa
- Szófa
- Hash-tábla
 - hash függvény jósága

Kollekciók: általánosítás

- Alapműveletek azonosak
 - beszúrás, keresés, törlés
- Alapfunkciók azonosak
 - iterálás
- Implementáció-függő egyedi funckiók
 - sorrendiség
 - halmaz/lista
 - deep/shallow copy
 - optimalizálás: pl. beszúrásra v. keresésre

Kollekció keretrendszer

Az interfészek



Collection interfész

- Általánosított tároló interfész
- Lehet, hogy
 - rendezett vagy nem
 - egyszeres vagy többszörös tárolás
- Definíciója generikus:

Collection<E>

Collection interfész 2

- **void add(E e)** *optional*
 - új objektum hozzáadása
- **void addAll(Collection<? extends E> c)** *optional*
 - **c** kollekció összes objektumának hozzáadása
 - csak a referenciák másolódnak: shallow copy
- **boolean remove(E e)** *optional*
 - objektum törlése, **false** ha nem tartalmazta
- **boolean removeAll(Collection<? extends E> c)** *optional*
 - **c**-ben levő összes objektum törlése a kollekcióból
 - csak a referenciák törlődnek, nincs destruktorhívás!

Collection interfész 3

- **boolean contains(E e)** *optional*
 - igaz, ha tartalmazza az objektumot
- **boolean containsAll(Collection<? extends E> c)** *optional*
 - igaz, ha tartalmazza c összes objektumát
- **int size()**
 - tartalmazott objektumok száma
- **boolean isEmpty()**
 - igaz, ha a kollekció üres
- **void clear()**
 - összes objektum referenciájának törlése

Collection interfész 4

- **boolean retainAll(*Col*<? *ext* *E*> *c*)** *optional*
 - csak *c*-ben is szereplő objektumokat tartja meg
- **boolean equals(*E* *e*)**
 - azonosság: **List** és **Set** nem lehet azonos
 - kötelezően szimmetrikus kell legyen
- **Object[] toArray()**
 - a kollekció elemeit tartalmazó tömbbel tér vissza
- **<T> T[] toArray(T[] *ta*)**
 - a kollekció elemeit tartalmazó, *ta* típusú tömbbel tér vissza
- **Iterator<*E*> iterator()**
 - egy **Iterator** interfészű objektumot ad vissza

Iterator interfész

- Iterál egy kollekció elemein
 - nem ismert a sorrend
- Definíciója: **Iterator<E>**
- **boolean hasNext()**
 - igaz, ha van még iterálandó elem
- **E next()**
 - a következő elemmel tér vissza
- **void remove()**
 - törli az utoljára visszaadott elemet a kollekcióból

Iterator interfész 2

■ Konkurencia (többszálúság) esetén

- ha elemet törölünk, az megzavarhatja a többi iterátort
- ilyenkor **ConcurrentModificationException** lehet a büntetés

```
Collection c = ...;  
...  
Iterator i1 = c.iterator();  
Iterator i2 = c.iterator();  
i1.next();  
i2.next();  
i2.remove();  
i1.next();    // itt kapjuk a kivételt
```

Iterator interfész 3

■ Tipikus használata:

```
Collection<Integer> c = ...;
...
Iterator<Integer> i = c.iterator();

while (i.hasNext()) {
    int a = i.next(); // outboxing
    if (a < 0) {
        i.remove();
    }
}
```

Set interfész

- Halmaz, vagyis minden elem csak egyszer szerepel
- A sorrendjükről semmit sem tudunk
- Az iterátor is tetszőlegesen adhatja vissza az elemeket
- Nincs a kollekciót bővítő metódusa
- Tipikus implementáció: **HashSet**

SortedSet interfész

- Olyan halmaz, amiben az elemek sorrendje rögzített
- Az elemek döntenek el, hogy mi a sorrend
 - természetes, vagy **Comparator** minta
- Az iterátor az adott sorrendben adja vissza az elemeket
- Tipikus implementáció: **TreeSet**

SortedSet interfész 2

- **Comparator<? super E> comparator()**
 - visszaadja a rendező objektumot, vagy **null**-t ha természetes rendezés
- **E first()** első elem
- **E last()** utolsó elem
- **SortedSet<E> headSet(E to)**
 - **to**-ig az elemek (exkluzívan)
- **SortedSet<E> tailSet(E from)**
 - **from**-tól az elemek (inkluzívan)
- **SortedSet<E> subSet(E from, E to)**
 - **from**-tól **to**-ig az elemek

List interfész

- Elemek sorozata (szekvencia)
- Egy elem többször is
- Minden elem helye ismert
 - index alapján elérhetők
- Kereshetők az elemek
- **ListIterator**-t ad a hatékonyabb eléréshez
 - **ListIterator** az **Iterator** leszármazottja
- Tipikus implementáció: **ArrayList**

List interfész 2

- **boolean add(E e)**
 - a lista végére fűzi **e**-t
- **void add(int i, E e)**
 - a lista **i**. helyére teszi **e**-t, a többi shifteli
- **boolean addAll(int i, *Col<? ext E>* e)**
 - a lista **i**. helyétől berakja **e** elemeit, a többi shifteli
- **E set(int i, E e)**
 - a lista **i**. helyére teszi **e**-t, a korábbi elemet visszaadja
- **E get(int i)**
 - visszaadja az **i**. elem referenciáját

List interfész 3

- **int indexOf(Object o)**
 - visszaadja az objektum első előfordulásának helyét
- **int lastIndexOf(Object o)**
 - visszaadja az objektum utolsó előfordulásának helyét
- **E remove(int i)**
 - a lista *i*. helyéről törli az elemet, a referenciáját visszaadja
- **List<E> subList(int from, int to)**
 - visszaadja egy listában az intervallum elemeit
- **ListIterator<E> listIterator([int i])**
 - [adott indextől] lista-iterátort ad vissza

ListIterator interfész

- **void add (E e)**
 - hozzáadja az objektumot a listához az aktuális pozíción
- **void set(E e)**
 - utoljára visszaadott elemet cseréli
- **E previous()**
 - a legutoljára visszaadott elem előtti elemet adja vissza
- **boolean hasPrevious()**
 - igaz, ha **previous** tud még elemet adni
- **int nextIndex() | int previousIndex()**
 - a következő **next/previous** hívással visszaadandó elem indexe

Queue interfész

- Puffer implementálásához
- Lehet FIFO, LIFO, *Comparator*-vezérelt
- Két fajta metódus, hiba esetén
 - kivételt dob
 - visszatérési értékben jelez
- Tipikus implementáció: **LinkedList**

Queue interfész 2

- **boolean add(E e)** (*IllegalStateException*)
- **boolean offer(E e)**
 - berakja az objektumot a sorba
- **E element()** (*NoSuchElementException*)
- **E peek()**
 - visszadja, de nem törli a sor elején levő elemet
- **E remove()** (*NoSuchElementException*)
- **E poll()**
 - visszaadja és törli a sor elején levő elemet

Map interfész

- Kulcsok és értékek párait tárolja
- Kulcs alapján értéket egyértelműen visszaadja
- Módosítható (mutable) objektumok használata kontraindikált
- Három nézet
 - kulcsok halmaza
 - értékek halmaza
 - kulcs-érték párok halmaza
- Tipikus implementáció: **HashMap**

Map interfész 2

- Definíció: **Map<K, V>**

- ☐ **K** a kulcsok típusa
- ☐ **V** az értékek típusa

- Ami ugyanaz, mint a **Collection**-nél:

- ☐ **void clear()**
- ☐ **boolean equals()**
- ☐ **boolean isEmpty()**
- ☐ **int size()**

Map interfész 3

- **V put(K key, V value)**
 - hozzáadja a kulcs-érték párt a leképezéshez
- **void putAll(Map<? ext K, ? ext V> m)**
 - a megadott leképezés elemeit hozzáadja a leképezéshez
- **V get(K key)**
 - visszaadja a kulcshoz tartozó értéket, nem törli
- **V remove(K key)**
 - törli és visszaadja a kulcshoz tartozó értéket

Map interfész 4

- **boolean containsKey(Object key)**
- **boolean containsValue(Object value)**
 - igaz, ha leképezés tartalmazza a kulcsot/értéket
- **Set<K> keySet()**
 - visszaadja a leképezésben szereplő
- **Collection<V> values()**
 - visszaadja a leképezésben szereplő értékeket
- **Set<Map.Entry<K, V>> entrySet()**
 - visszaadja a párok halmazát

Miért Set és Collection?

SortedMap interfész

- A Set kibővítése úgy, hogy a kulcsok rendezve legyenek
 - vagy természetes rendezés
 - vagy **Comparator** alapján
- A visszaadott nézetek (**keys()**, **values()**, **entrySet()**) ezt tükrözik
 - az iterátorok a kulcsok sorrendjében lépnek
- Tipikus implementáció: **TreeMap**

SortedMap interfész 2

- **Comparator<? super K> comparator()**
 - visszaadja a kulcsok összehasonlítóját
- **K firstKey()**
- **K lastKey()**
 - visszaadja az első/utolsó kulcsot
- **SortedMap<K,V> headMap(K from)**
- **SortedMap<K,V> subMap(K from, K to)**
- **SortedMap<K,V> tailMap(K to)**
 - visszaadja a leképzés elejét/közepét/végét

For ciklus kollekción

- Java 1.5 előtt iterátorral
 - `hasNext()` és `next()` metódusokkal
- Java 1.5 bevezette az egyszerűsített *for* ciklust

```
Collection<Integer> c = ...;  
...  
for (Integer i : c) {  
    System.out.println(i);  
}
```

For ciklus kollekción 2

■ Előnyei

- ☐ olvashatóbb, tisztább kód
- ☐ tömbön is működik

```
static public void main(String[] args) {  
    for (String s : args) { System.out.println(s); }  
}
```

■ Hátránya

- ☐ nincs **remove()**

Konkurens kollekció-elérés

- Több szál esetén fokozott figyelem kell a kollekciókhoz
- Alap kollekciók nem szál-biztosak
- Speciális kollekciók a korrekt konkurenciához
 - **java.util.concurrent** csomagban
 - **ConcurrentMap<K, V>**
 - **CopyOnWriteArrayList<E>**
 - **Semaphore**
 - ...

Konkurens kollekció-elérés 2

■ `java.util.Collections`

- `Collection synchronizedCollection(Collection c)`
- `List synchronizedList(List c)`
- `Map synchronizedMap(Map c)`
- `Set synchronizedSet(Set c)`
- `SortedMap synchronizedSortedMap(SortedMap c)`
- `SortedSet synchronizedSortedSet(SortedSet c)`

Read-only kollekciók

- Kollektciók paraméterként

- ☐ Java: shallow copy → változtatható
- ☐ C++-ban: *const* kulcsszó

- **java.util.Collections**

- ☐ `Collection unmodifiableCollection(Collection c)`
- ☐ `List unmodifiableList(List c)`
- ☐ `Map unmodifiableMap(Map c)`
- ☐ `Set unmodifiableSet(Set c)`



Reflection

Reflection

- Lehetővé teszi az osztályok, objektumok futási időben történő vizsgálatát és módosítását
 - bővíthetőségi lehetőségek
 - futási időben külső osztályok betöltése, objektumok elérése
 - osztálymegjelenítők
 - osztályok metainformációinak futási idejű vizsgálata
 - debuggerek, teszteszközök
 - a futás során az alkalmazás elemeinek ellenőrzése
 - objektumok elérése

Reflection 2

■ Óvatosan kell vele bánni

□ teljesítményromlás

- reflection használata megnöveli az erőforrásigényt és a futási időt

□ biztonsági megkötések

- a reflection használatához engedélyek kellenek, amik egyes SecurityManagerek mellett nem állnak rendelkezésre

□ egységbezárás megbontása

- olyan dolgokat is meg lehet tekinteni, lehet módosítani, ami sérti az OO elveket

Osztályok metainformációi

■ **Object.getClass()**

- visszaadja az adott objektum osztályát leíró **Class** objektumot

■ **.class** elérés

- **boolean.class** visszaadja a **boolean** primitív típus objektumát
- **String.class** visszaadja a **String** osztályt leíró objektumot

■ **Class.forName()**

- **Class.forName("java.lang.String")** visszaadja a **String** osztályt leíró objektumot

Class<T> metódusai

- **Class<? super T> getSuperClass()**
 - őssosztály leírója
- **Class<?>[] getClasses()**
- **Class<?>[] getDeclaredClasses()**
- **Class<?>[] getEnclosingClasses()**
 - tag/deklarált/beágyazott osztályok leírói

Class<T> metódusai 2

- **Constructor<T>**
getConstructor(Class<?>... parameterTypes)
 - adott paraméterű *publikus* konstruktor leírója
- **Constructor<?>[] getConstructors()**
 - összes *publikus* konstruktor leírója
- **Constructor<T>**
getDeclaredConstructor(Class<?>... parameterTypes)
 - adott paraméterű konstruktor leírója
- **Constructor<?>[] getDeclaredConstructors()**
 - összes konstruktor leírója

Class<T> metódusai 3

- **Field** `get[Declared]Field(String name)`
- **Field[]** `get[Declared]Fields()`
 - adott nevű / összes mező leírója
- **Method** `get[Declared]Method(String name, Class<?>... parameterTypes)`
- **Method[]** `get[Declared]Methods()`
 - adott szignatúrájú / összes metódus leírója

Class<T> metódusai 4

- **String getName()**
 - az osztály neve
- **Package getPackage()**
 - az osztály csomagja
- **int getModifiers()**
 - az osztály módosítói (**public**, **static**, **abstract**, stb), kódolva
- ...

Method metódusai

- **String getName()**
 - metódus neve
- **int getModifiers()**
 - metódus módosítói (pl. public) kódolva
- **Class<?>[] getParameterTypes()**
 - paramétereinek típusa
- **Class<?> getReturnType()**
 - visszatérési érték típusa

Method metódusai 2

- **Class<?>[] getExceptionTypes()**
 - kivételek típusa
- **Class<?> getDeclaringClass()**
 - az osztály leírója, amiben a metódus szerepel
- **Object invoke(Object obj, Object... args)**
 - metódus meghívása
 - adott objektumon
 - adott paraméterekkel

Field metódusai

- **String getName()**
 - attribútum neve
- **Class<?> getType()**
 - attribútum típusa
- **Object get(Object obj)**
- **int getInt(Object obj)**
 - attribútum értékének lekérése (általánosan és primitív típusra)
- **void set(Object obj, Object value)**
- **void setInt(Object obj, int i)**
 - attribútum értékének beállítása (általánosan és primitív típusra)



Kivételkezelés, Felsorolás, Vararg, Annotációk

Kivételkezelés története: C

- hibajelzés a visszatérési értékben

```
while ((c=getchar()) != EOF) {  
    c = toupper(c);  
    putchar(c);  
}
```

- keveredik a típus és a metatípus
- **setjmp()** és **longjmp()** primitív kivételkezelést ad

Kivételkezelés története: C++

- megjelenik a klasszikus kivételkezelés

```
try {  
    ...  
} catch (E e) { // E típ. kiv.  
} catch (...) { // minden más  
    ...  
    throw;  
}
```

- nincs egyértelmű kivétel-típus

Kivételkezelés története: **Java**

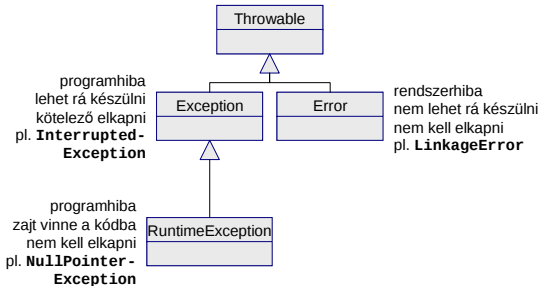
- a C++ változaton alapul
- van saját kivételosztály, csak ez dobható
 - **Throwable**
- kötelező definiálni, mit nem kap el
 - **throws**
- elkapandó és nem-elkapandó kivételek is
 - **Exception, RuntimeException, Error**
- záró blokk: **finally**

Kivételkezelés

```
void foo(String s) throws IOException {  
    ...  
}
```

```
FileInputStream fis = new FileInputStream("a");  
try {  
    ...  
    foo(s);  
    ...  
} catch (IOException e) {  
    ...  
    throw e;  
} finally {  
    fis.close();  
}
```

Kivételek hierarchiája



Láncolt kivételek

```
public AST parseFile(File f) throws ParseException {  
    AST ast = new AST();  
    try {  
        ...  
    } catch (IOException e) {  
        ParseException pe = new ParseException();  
  
        throw pe;  
    }  
    ...  
    return ast;  
}
```

```
try { ast = parseFile(f); }  
catch (ParseException pe) { pe.printStackTrace(); }
```

Láncolt kivételek 2

```
public AST parseFile(File f) throws ParseException {  
    AST ast = new AST();  
    try {  
        ...  
    } catch (IOException e) {  
        ParseException pe = new ParseException();  
        pe.initCause(e);  
        throw pe;  
    }  
    ...  
    return ast;  
}
```

```
try { ast = parseFile(f); }  
catch (ParseException pe) { pe.printStackTrace(); }
```

Láncolt kivételek 3

```
ParseException  
    at Test2.parseFile(Test2.java:19)  
    ...  
  
Caused by: IOException  
    at Test2.nextString(Test2.java:12)  
    ...  
    at Test2.parseFile(Test2.java:17)
```

- a **Throwable**-ben Java 1.4 óta beállítható egy *cause*
- a *stack-trace*-ben is megjelenik

Enum: integerekkel

- Enum intekkel

```
public static final int TUDOR=0;  
public static final int VIDOR = 1;  
public static final int HAPCI = 2;  
...
```

- Nem típusérzékeny

- minden **int**

- Módosítás újrafordítást igényel

- Kiíratás nem informatív

Enum: pattern

■ *Typesafe enum pattern*

```
public final class Enum implements java.io.Serializable {  
    public static final Enum TRUE = new Enum (true);  
    public static final Enum FALSE = new Enum (false);  
    public String toString () {  
        return String.valueOf (m_value).toUpperCase ();  
    }  
    private Enum (boolean value) { m_value = value; }  
    private Object readResolve ()  
        throws java.io.ObjectStreamException {  
        return (m_value ? TRUE : FALSE);  
    }  
    private boolean m_value;  
} // end of class
```

Enum: valódi típus

- valódi felsorolás típus (Java 1.5)
- saját osztállyal
- attribútumokkal
- metódusokkal
- szerializálható
- kiíratható
- rövid *for* ciklus működik
- switch-case működik

Enum példa

```
public enum Törpe {  
    TUDOR, VIDOR, SZENDI, SZUNDI, HAPCI, MORGÓ, KUKA  
}
```

```
public enum Törpe {  
    TUDOR(210), VIDOR(120), SZENDI(90), SZUNDI(95),  
    HAPCI(110), MORGÓ(170), KUKA(10);  
    private final int iq;  
    Törpe(int i) { iq = i; }  
    public int iq { return iq; }  
    public String ének() { return "-Hejhó! -"+iq; }  
}
```

```
for (Törpe t : Törpe.values()) {  
    System.out.println(t+": "+t.ének());  
}
```

Enum példa 2

```
enum Betuk {A, B, C}
```

```
Betuk e = Betuk.A;  
switch (e) {  
    case A: System.out.println("A!"); break;  
    case B: System.out.println("B!"); break;  
    case C: System.out.println("C!"); break;  
}
```

```
if (e == Betuk.B) { ... }
```

Enum metódusai

- **String name()**

- enum konstans neve

- **int ordinal()**

- konstans sorszáma

- **static <T extends Enum<T>> T
valueOf([Class<T> enumType,]
String name)**

- adott [típusú és] nevű konstanst ad vissza

- **static <T extends Enum<T>> T[]
values()**

- az enum típus konstansait adja vissza

Enum genericitása

```
Enum<E extends Enum<E>>
```

- enumot csak enum leszármazottjával lehet paraméterezni
- speciális metódusokat örököl

```
public final int compareTo(E o){ ... }
```

- ezek a metódusok csak a konkrét enum típuson értelmesek

Változó hosszúságú paraméterlista

- Pre 1.5: tömböt kellett átadni
 - kényelmetlen

```
void foo(String s, Object[] oa) { for (Object o : oa) ...; }
```

- 1.5: *varargs*
 - tömbként és felsorolva is átadható
 - csak a paraméterlista végén állhat

```
void foo(String s, Object... oa) { for (Object o : oa) ...; }
```

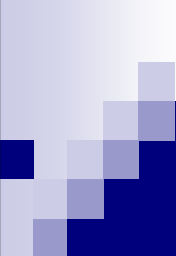
```
Object[] oo = {"a", "b", "c"};  
foo("X", oo);  
foo("X", "j", "k", "l", "m", "n");
```

Annotációk

- A forráskód metainformációkkal való kibővítéséhez

```
public @interface Copyright {  
    String value() default "2008 Me";  
}  
@Copyright("2008 Bytemongers Limited")  
public class Árvíztűrőtükörfúrógép { ... }
```

- Interfész @ jellel kezdve
- Metódusok írják le az annotáció elemeit
 - visszatérési érték primitív, String, Class, enum
- Az elemnek lehet alapértelmezett értéke



Objektumorientált szoftvertervezés

Java gyakorlat

1. Boxing

```
Integer a = 3;  
Integer b = a;  
boolean B1 = (a==b);           //true  
boolean B2 = (a.equals(b));    //true  
  
a += 2;  
  
boolean B3 = (a==b);           //false  
boolean B4 = (a.equals(b));    //false
```

1. Boxing

```
a += 2;
```

```
5:   aload_1
6:   invokevirtual   #3;
    //java/lang/Integer.intValue:()I
9:   iconst_2
10:  iadd
11:  invokestatic     #2;
    //Integer.valueOf:(I)Ljava/lang/Integer;
14:  astore_1
```

```
int tmp = a.intValue();      (5,6)
tmp += 2;                    (9,10)
a = Integer.valueOf(tmp);    (11,14)
```

1. Boxing

```
HashMap<String,int> hm =  
    new HashMap<String,int>();  
while(true) {  
    String s = readLine();  
    if (s==null) break;  
    if (hm.containsKey(s)) {  
        hm.get(s)+=1;  
    } else hm.put(s,1);  
}
```

1. Boxing

```
HashMap<String, Integer> hm =  
    new HashMap<String, Integer>();  
while(true) {  
    String s = readLine();  
    if (s==null) break;  
    if (hm.containsKey(s)) {  
        hm.get(s)+=1;  
    } else hm.put(s,1);  
}
```

1. Boxing

```
hm.get(s) += 1;
```

```
Integer i = hm.get(s);
```

```
i += 1;
```

```
j = hm.get(s);
```

```
boolean B1 = (i == j); false
```

```
boolean B2 = (i.equals(j)); false
```

2. Öröklés

```
public class A {  
    private void f1(){System.out.println("A.f1");}  
    public void f2() {System.out.println("A.f2");}  
    public void f3() { f1(); }  
    public void f4() { f2(); }  
}  
public class B extends A {  
  
    public static void main(String[] args) {  
        A a1 = new A();  
        a1.f3();        // A.f1  
        a1.f4();        // A.f2  
    }  
}
```

2. Öröklés

```
public class A {  
    private void f1(){System.out.println("A.f1");}  
    public void f2() {System.out.println("A.f2");}  
    public void f3() { f1(); }  
    public void f4() { f2(); }  
}  
public class B extends A {  
    private void f1() {System.out.println("B.f1");}  
    public void f2() {System.out.println("B.f2");}  
    public static void main(String[] args) {  
        A a2 = new B();  
        a2.f3();        // A.f1  
        a2.f4();        // B.f2  
    }  
}
```

2. Öröklés

```
public class A {  
    private void f1(){System.out.println("A.f1");}  
    public void f2() {System.out.println("A.f2");}  
    private void f5() { f2(); }  
    public void f6() { f5(); }  
}  
public class B extends A {  
    private void f1() {System.out.println("B.f1");}  
    public void f2() {System.out.println("B.f2");}  
    public static void main(String[] args) {  
        A a3 = new A(); A a4 = new B();  
        a3.f6();          // A.f2  
        a4.f6();          // B.f2  
    }  
}
```

2. Öröklés (konstruktor, Java)

```
public class A {  
    void foo() { System.out.println("A.foo");}  
    public A() { foo(); }  
}  
  
public class B extends A {  
    void foo() { System.out.println("B.foo");}  
    public B() { super(); }  
  
    public static void main(String[] args) {  
        A a = new A();    // A.foo  
        B b = new B();    // B.foo  
    }  
}
```

2. Öröklés (konstruktor, C++)

```
class A {  
public:  
    virtual void foo() { cout << "A.foo" << endl; }  
    A() { foo(); }  
};  
class B : public A {  
public:  
    virtual void foo() { cout << "B.foo" << endl; }  
    B() : A() { }  
};  
  
int main() {  
    A a();    // A.foo  
    B b();    // A.foo  
}
```

3. API implementáció

```
public class Object {  
    ...  
    public boolean equals(Object obj) {  
        return (this == obj);  
    }  
    public String toString() {  
        return getClass().getName()  
            + "@"  
            + Integer.toHexString(hashCode());  
    }  
    ...  
}
```

3. API implementáció

```
public class String { // java 1.1
    private int hash = 0;
    public int hashCode() { int h = hash;
        if (h == 0) {
            char val[] = value;
            int len = count; int skip = Math.max(1, len/8);
            for (int i = 0; i < len; i += skip) {
                h = 37*h + val[i];
            }
            hash = h;
        }
        return h;
    }
}
```

3. API implementáció

```
public class String { // java 1.5
    private int hash = 0;
    public int hashCode() { int h = hash;
        if (h == 0) {
            int off = offset; char val[] = value;
            int len = count;
            for (int i = 0; i < len; i++) {
                h = 31*h + val[off++];
            }
            hash = h;
        }
        return h;
    }
}
```

3. API implementáció

```
public class Integer {  
    private final int value;  
    public int hashCode() {return value;}  
    private static class IntegerCache {  
        private IntegerCache(){}  
        static final Integer cache[] =  
            new Integer[-(-128) + 127 + 1];  
        static {for(int i = 0; i < cache.length; i++)  
            cache[i] = new Integer(i - 128);  
        }  
    }  
    public static Integer valueOf(int i) {  
        final int offset = 128;  
        if (i >= -128 && i <= 127) {  
            return IntegerCache.cache[i + offset]; }  
        return new Integer(i);  
    }  
}
```

3. API implementáció

```
class Test {  
    Object o1;  
    Object o2;  
    ...  
    Object on;  
    public int hashCode() {  
        int h = 0;  
  
        h = 31*h+o1.hashCode();  
        h = 31*h+o2.hashCode();  
        ...  
        h = 31*h+on.hashCode();  
  
        return h;  
    }  
}
```

4. Konkurencia: sorompó

```
public class Barrier { // N szál megvárja egymást
    ...
    public Barrier(int i) {...}
    public void waiting() {
        ...
    }
}
```

4. Konkurencia: sorompó

```
public class Barrier {  
    private int n, in;  
    public Barrier(int i) {n = i; in = 0;}  
    public synchronized void waiting() {  
        in++;  
        if (in < n) {  
            wait();  
        } else {  
            in = 0;  
            notifyAll();  
        }  
    }  
}
```

5. Genericitás

```
public class Proba {  
    static public void main(String[] args) {  
        Vector<Integer> v = new Vector<Integer>();  
        Integer a = 3;  
        v.add(a);  
        Integer b;  
        b = v.get(0);  
    }  
}
```

5. Genericitás

```
Vector<Integer> v = new Vector<Integer>();
```

```
0:  new #2; //class java/util/Vector
```

```
3:  dup // konstruktor paramétere
```

```
4:  invokespecial #3; //java/util/Vector."<init>":()V
```

```
7:  astore_1
```

```
Integer a = 3;
```

```
8:  iconst_3
```

```
9:  invokestatic  #4;
```

```
    //Integer.valueOf:(I)Ljava/lang/Integer;
```

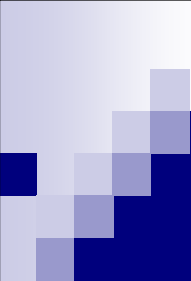
```
12: astore_2
```

5. Genericitás

```
v.add(a);  
13: aload_1  
14: aload_2  
15: invokevirtual #5;  
    //java/util/Vector.add:(Ljava/lang/Object;)Z  
18: pop // visszatérési értéket eldobjuk  
b= v.get(0);  
19: aload_1  
20: iconst_0  
21: invokevirtual #6;  
    //java/util/Vector.get:(I)Ljava/lang/Object;  
24: checkcast      #7; //class java/lang/Integer  
27: astore_3  
// return  
28: return
```

6. API leírások

- Java API dokumentáció böngészőben
 - **HashMap**: idők, szinkronitás, shallow copy
 - **ArrayList**: idők, szinkronitás, shallow copy
 - **IdentityHashMap**: referencia szerinti azonosság
 - **TreeSet**: elérés ideje, rendezés és equals konformitása
 - **Collections**: hatékony kereső, rendező, keverő metódusok, konkurens kollekciók



Objektumorientált szoftvertervezés

Ablakkezelés, SWING

Ablakkezelés Java-ban

■ AWT

- Abstract Windowing Toolkit
- natív ablakkezelés és widgetek

■ SWING

- Java Foundation Classes
- light-weight widgetek

■ SWT

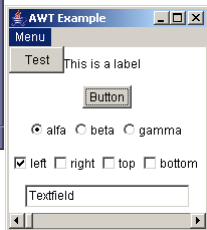
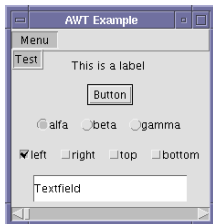
- Standard Widget Toolkit
- Eclipse és barátai

Abstract Windowing Toolkit

- JDK 1.0-tól
- Abstract: az interfésze független a platformtól
 - mindenhol natívan néz ki
 - natív implementációval
 - inkompatibilitás
- Netscape nyomására
- Emiatt kezdetben sok hiba
 - eseménykezelés átgondolatlan
 - sok bug

Alap AWT widgetek

- Canvas
- Button
- List
- TextBox
- Menu
- Choice
- Checkbox
- CheckboxGroup
- ...



Konténer (**Container**)

- A komponenseket konténerekbe tesszük
 - **Panel**
 - **ScrollPane**
 - **Window**
 - **Frame**
 - ...
- A konténer is komponens
 - konténerbe másik konténer is tehető
 - hierarchikus tartalmazás jön létre
 - klasszikus fa szerkezet

Konténer felelőssége

- Komponensek megtalálása

- ☐ adott pozíción levő
- ☐ adott sorszámú
- ☐ lista az összesről

- A fókuszt továbbadása

- ☐ ki az aki a billentyűzetről jövő információt kezeli

- A komponensek elhelyezése

- ☐ az alkalmazott algoritmus a *LayoutManager*-ben van implementálva (*Strategy pattern*)
- ☐ rekurzív méretszámítás



Eseménykezelés

Eseménykezelés 1.0

■ Component osztály metódusai

- `public boolean handleEvent(Event e)`
- `public boolean mouseDown(Event e, int x, int y)`
- `public boolean keyDown(Event e, int key)`
- `public boolean action(Event e, Object what)`
- ...

■ AWT az érintett komponensnek meghívja a **handleEvent** metódusát

■ A **handleEvent** default módon meghívja az érintett egyebet

Eseménykezelés 1.0 (2)

- Az érintett kezelő metódus *true*-val tér vissza, ha kezelt
- Ha *false*, akkor a hierarchiában eggyel feljebb levőt hívja
- A fejlesztő leszármaztat a komponensből, és amit akar, felüldefiniál

Eseménykezelés 1.0 (3)

- Az összes esemény akár a hierarchia tetejéig is eljuthat
 - hatalmas overhead
- Keveredik az eseménykezelés és a megjelenítés
- Alapból minden komponenshez egy eseménykezelő
 - ez persze megvalósíthatja az observer mintát
- A metódusok paraméterei rondák
- Csak egy **Event** osztály

Eseménykezelés 1.0 példa

```
public class MyButton extends Button {  
    public boolean action(Event e, Object o) {  
        System.out.println("Button pressed");  
        return true;  
    }  
}
```

Eseménykezelés 1.1

■ Observer minta

- az események iránt érdeklődők regisztrálnak a komponensnél
- többen is megkaphatják az eseményt
- egy observer több komponenshez is regisztrálhat

■ Többfajta eseménytípus (újak is)

- `java.util.EventObject`
- `java.awt.AWTEvent`
- `java.awt.event.MouseEvent`

■ Az események jól elkülöníthetők

- a felelősség osztható

Eseménykezelés 1.1 (2)

■ *XEventListener*

- interfész
- megvalósítását komponenshez lehet regisztrálni
 - `addXEventListener(XEventListener e1)`
- *XEvent*-et kell feldolgozni
 - `MouseEvent`, `KeyEvent`, `AdjustmentEvent`, `FocusEvent` stb.
- gyakori az anonim, belső osztály használata
 - kényelmes, de nem menedzselhető

Eseménykezelés 1.1 (3)

■ *XEventAdapter*

- *XEventListener* megvalósítása
 - üres metódusokkal
- kényelmi osztály, hogy ne kelljen az üres metódusokat megírni
- használatukkal áttekinthetőbb kód nyerhető

Eseménykezelés 1.1 példa

```
public class MyActionListener implements ActionListener {  
    public void actionPerformed(ActionEvent ae) {  
        System.out.println("Button pressed");  
    }  
}
```

```
...  
Button b = new Button("Hello");  
ActionListener al = new MyActionListener();  
b.addActionListener(al);  
...
```



Fókusz-kezelés

Fókusz

- Akinél a fókusz van, az kapja meg a billentyűzetről érkező adatokat
- JDK 1.4 előtt hibásan működő, ad-hoc, platformfüggő fókuszkezelés volt
- JDK 1.4 óta jól átgondolt, korrekt rendszer
- A soron következő komponens meghatározása a **KeyboardFocusManager** dolga
 - **DefaultKeyboardFocusManager**

Fókusz alapfogalmak

- Fókusz tulajdonosa (*focus owner*)
 - az a komponens, akinél éppen a fókusz van
- Permanens tulajdonos (*permanent focus owner*)
 - az a komponens, akinél esetleg csak időlegesen nincs a fókusz
- Fókusz ablaka (*focused window*)
 - az az ablak, amiben a fókusz tulajdonosa található
- Aktív ablak (*active window*)
 - vagy nála a fókusz, vagy az első *Frame* vagy *Dialog*, akinél a fókusz ablaka van

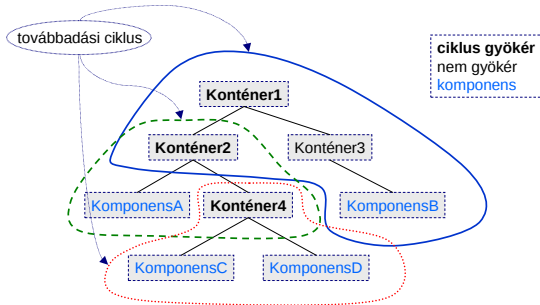
Fókusz alapfogalmak 2

- Fókusz-továbbadás (*focus traversal*)
 - a fókusz átvitele komponensről komponensre
 - tipikusan TAB-bal
 - lehet előre vagy hátra
 - elő lehet idézni programozottan is
- Fókusz-továbbadási ciklus (*focus traversal cycle*)
 - a konténer hierarchia egy darabja, amiben a fókusz-továbbadás minden komponenst érint, de más komponenseket nem

Fókusz alapfogalmak 3

- Fókusz ciklus gyökere (*focus cycle root*)
 - konténer, amely egy adott *fókusz-továbbadási ciklus* gyökere
 - alapértelmezésben a **Window**
 - normál fókusz-továbbadás esetén a fókusz nem kerülhet a fókusz ciklus gyökere fölé
 - ehhez speciális fel- és leléptető parancs kell
- Fókusz-továbbadást szabályozó (*focus traversal policy provider*)
 - konténer, amelynek saját fókusz-továbbadási szabálya van

Fókusz alapfogalmak 4



Billentyűzet-események

- Előfordul, hogy valaki más is kíváncsi a kapott eseményre, mielőtt a *focus owner* megkapja
- **KeyEventDispatcher** interfészt kell megvalósítani
 - eldöntheti, hogy az esemény továbbadódjon-e
 - **boolean dispatchKeyEvent(KeyEvent e)**
 - ha true, más már nem kapja meg
 - ha másnak akarja küldeni: **KeyboardFocusManager.redispatchEvent(java.awt.Component, java.awt.AWTEvent)**
- A **KeyboardFocusManager**-nél kell regisztrálni

Billentyűzet-események 2

- Előfordul, hogy valaki más is kíváncsi a kapott eseményre, miután feldolgozták
 - pl. menü *short-cut* esetén
- **KeyEventPostProcessor** interfészt kell megvalósítani
 - eldöntheti, hogy az esemény továbbadódjon-e
 - **boolean postProcessKeyEvent(KeyEvent e)**
 - ha true, más már nem kapja meg
 - akkor is kaphat, ha senkinél sincs fókusz
- A **KeyboardFocusManager**-nél kell regisztrálni

Fókuszváltás-események

■ WindowEvent

- WINDOW_ACTIVATED / WINDOW_DEACTIVATED
 - **Frame** vagy **Dialog** kapja, ha aktívvá válik vagy már nem aktív
- WINDOW_GAINED_FOCUS / WINDOW_LOST_FOCUS
 - **Window** kapja, ha megkapja vagy elveszti a fókuszt

■ FocusEvent

- FOCUS_GAINED / FOCUS_LOST
 - **Component** kapja, ha megkapja vagy elveszti a fókuszt

Fókuszváltás-események 2

■ Sorrendiség

- ha nem a Java alkalmazásé a fókus
- **Frame F1**-ben **K1** komponensre kattintunk
 - **F1**: WINDOW_ACTIVATED
 - **F1**: WINDOW_GAINED_FOCUS
 - **K1**: FOCUS_GAINED

Fókuszváltás-események 3

■ Sorrendiség

- előzőek után
- **Frame F2**-ben **K2** komponensre kattintunk
 - **K1**: FOCUS_LOST
 - **F1**: WINDOW_LOST_FOCUS
 - **F1**: WINDOW_DEACTIVATED
 - **F2**: WINDOW_ACTIVATED
 - **F2**: WINDOW_GAINED_FOCUS
 - **K2**: FOCUS_GAINED

Fókuszváltás-események 4

- Minden esemény csak az előző feldolgozása után
- Minden esemény az ellentétessel párban
 - pl. FOCUS_GAINED után nem jöhet újabb FOCUS_GAINED
- Csak informáló események
 - megváltoztatásukra nincs lehetőség
 - ha mégis nagyon szükséges, akkor **VetoableChangeListener**

Fókuszváltás-események 5

- Túloldali komponens és ablak
 - ha kíváncsiak vagyunk arra, hogy ki kapta az esemény-pár másik felét
 - `FocusEvent.getOppositeComponent()`
 - `WindowEvent.getOppositeWindow()`

Átmeneti fókuszváltás

- FOCUS_GAINED és FOCUS_LOST *temporary*-nek lehet jelölve
- Akkor lehet, amikor csak rövid időre veszítjük el a fókuszt
 - pl. menü megnyitása, scrollbar görgetése
- A túloldali nem biztos, hogy szintén *temporary*-t lát
- Hasznos, ha *commit*-olni kellene
 - pl. **TextField** esetén



Layout management

Hová kerülnek a komponensek?

- Hová kerülnek a widget-ek egy konténeren?
 - ahova rakjuk őket?
 - *this page is optimized for 800x600 resolution*
 - mi történik átméretezéskor?
- Java-ban konténerekhez *layout managerek* vannak rendelve
 - különválnak a tartalmazás és az elrendezés felelőssége
- Vannak megoldások, ahol a komponens felelőssége, hogy merre köt

Layout managerek

■ Container

- **void setLayout(LayoutManager mgr)**
 - beállítja a layout managert
- **LayoutManager getLayout()**
 - visszaadja a layout managert
- **void validate()**
 - rekurzívan aktualizálja a konténer komponenseinek elhelyezését
- **Component add(Component comp [,int index])**
- **void add(Component c, Object constraint, int index)**
 - hozzáadja a komponenst a konténerhez

Layout managerek 2

■ **LayoutManager**

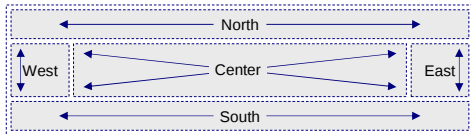
- **void addLayoutComponent(String name, Component comp)**
 - hozzáadja *comp*-ot
- **void removeLayoutComponent(Component comp)**
 - leveszi *comp*-ot
- **void layoutContainer(Container parent)**
 - berendezi *parent*-et
- **Dimension minimumLayoutSize(Container parent)**
- **Dimension preferredLayoutSize(Container parent)**

Layout managerek 2

■ **LayoutManager2**

- **void addLayoutComponent(Component comp, Object constraints)**
 - megkötéssel ad hozzá komponenst
- **float getLayoutAlignmentX(Container target)**
- **float getLayoutAlignmentY(Container target)**
 - X és Y irányú igazítás
- **void invalidateLayout(Container target)**
 - ha a konténer tárolt infót a komponensei elhelyezéséről, az már nem érvényes
- **Dimension maximumLayoutSize(Container target)**

BorderLayout



- Öt mező
 - north, south, west, east, center
- **Frame**-ek alapértelmezett kiosztása
- A nyilak szerint nyúlhatnak a komponensek
- Egy mezőbe legfeljebb egy komponens kerülhet

FlowLayout

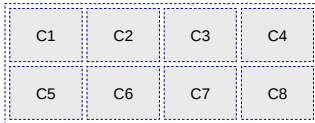


- Egymás mellé helyezi a komponenseket
- Új sort nyit, ha nem férnek el
- Nem nyújt rajtuk
- Balról jobbra vagy jobbról balra a konténertől függ
 - `ComponentOrientation.LEFT_TO_RIGHT`,
`RIGHT_TO_LEFT`
- Igazítással
 - `LEFT`, `RIGHT`, `CENTER`, `LEADING`, `TRAILING`

CardLayout

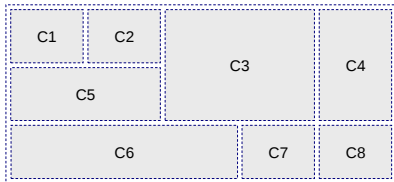
- Kártyapakliként kezeli a komponenseket
- Egy komponens egy lap
- Mindig csak a legfelső lap látszik
- A layout manager metódusaival mozoghatunk a pakliban
- Elnevezhetjük a komponenseket a gyorsabb elérés kedvéért

GridLayout



- Egy $n \times m$ -es mátrixban helyezi el az elemeket
- Mindegyik egyforma méretet kap
 - amelyiket kell, nyújtja
- Sorrendiség a konténertől
 - **LEFT_TO_RIGHT - RIGHT_TO_LEFT**
- Ha a sorok száma rögzített, az oszlopoké nem számít

GridBagLayout



- Egy $n \times m$ -es mátrixba teszi az elemeket
- De egy elem több egységet is elfoglalhat

GridBagConstraints

- Az elemek elhelyezkedését a **GridBagLayout** esetén **GridBagConstraints** paraméter szabja meg
- **GridBagConstraints.gridx/y**
 - a komponens bal felső (jobb felső) elemének pozíciója
 - ha az érték **RELATIVE**, akkor a megelőző elem után (default)
- **GridBagConstraints.gridwidth/height**
 - az elem mérete: ennyi sort/oszlopot foglal el (default: 1)
 - ha **REMAINDER**, akkor a sor/oszlop végéig ér

GridBagConstraints 2

■ **GridBagConstraints.weightx/y**

- a maradék extra helyek a súlyok arányában lesznek szétosztva (default 0)
- ha mindenkinek nulla, akkor a szélre kerül az extra hely

■ **GridBagConstraints.ipadx/y**

- ennyi hozzá lesz adva a komponens minimális méretéhez (default 0)

GridBagConstraints 3

■ **GridBagConstraints insets**

- a komponens körül levő minimális üres rész mérete (default 0)

■ **GridBagConstraints fill**

- ha a kért terület nagyobb, mint az elem preferált mérete, akkor mi történjen
- **NONE**: semmi (default)
- **HORIZONTAL**: széltében nőhet, függőlegesen nem
- **VERTICAL**: függőlegesen nőhet, széltében nem
- **BOTH**: minden irányban nőhet

GridBagConstraints 4

■ **GridBagConstraints.anchor**

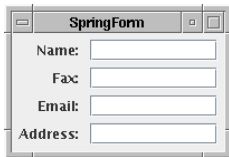
- hova kell helyezni a komponenst a területén
- abszolút
 - center (default) north, south, west, east, northeast, northwest, southeast, southwest
- relatív (függ a jobb-bal iránytól)
 - page_start, page_end, line_start, line_end, first_line_start, first_line_end, last_line_start, last_line_end
- alapvonal (baseline)
 - baseline, baseline_leading, baseline_trailing
 - above_baseline*, below_baseline*

BoxLayout (swing)



- Komponentek vízszintes vagy függőleges elrendezése
- Nem tör a sor végén
- Négyféle leosztás
 - **X_AXIS – Y_AXIS**: horizontális vagy vertikális
 - **LINE_AXIS – PAGE_AXIS**: figyelembe veszi a **ComponentOrientation**-t is

SpringLayout (swing)

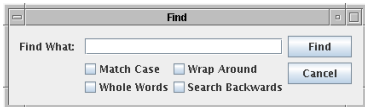


- Rugalmas táblázat megjelenítésére
- Alapvetően a komponensek élei közötti kapcsolatot kell specifikálni
- GUI builder-ek számára
 - kézzel nem könnyű használni

SpringLayout (swing) 2

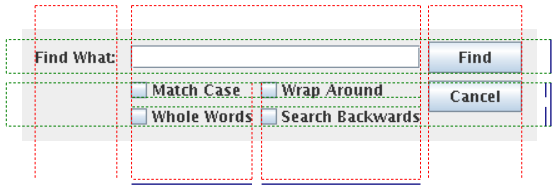
```
String[] labels = {"Name: ", "Fax: ", "Email: ", "Address: "};
int numPairs = labels.length;
//Create and populate the panel.
JPanel p = new JPanel(new SpringLayout());
for (int i = 0; i < numPairs; i++) {
    JLabel l = new JLabel(labels[i], JLabel.TRAILING);
    p.add(l);
    JTextField textField = new JTextField(10);
    l.setLabelFor(textField);
    p.add(textField);
}
//Lay out the panel.
SpringUtilities. // itt van a kutya elásva
makeCompactGrid(p, numPairs, 2, // container, rows, cols
                6, 6, 6, 6); // initX, initY, xPad, yPad
```

GroupLayout (swing)



- Horizontális és vertikális dimenziók független kezelése
 - minden komponens kétszer
- Hierarchikus: csoportok egymásba ágyazva
- Szekvenciális és parallel: egymás mellé vagy fölé
- Komponens cseréje:
 - **`void replace(Component oldc, Component newc)`**

GroupLayout (swing) 2



----- vertikális
----- horizontális

GroupLayout (swing) 3

```
layout.setHorizontalGroup(layout.createSequentialGroup()  
    .addComponent(label)  
    .addGroup(layout.createParallelGroup(LEADING)  
        .addComponent(textField)  
        .addGroup(layout.createSequentialGroup()  
            .addGroup(layout.createParallelGroup(LEADING)  
                .addComponent(caseCheckBox)  
                .addComponent(wholeCheckBox)  
            .addGroup(layout.createParallelGroup(LEADING)  
                .addComponent(wrapCheckBox)  
                .addComponent(backCheckBox)))  
    .addGroup(layout.createParallelGroup(LEADING)  
        .addComponent(findButton)  
        .addComponent(cancelButton)  
    );
```

GroupLayout (swing) 4

```
layout.setVerticalGroup(layout.createSequentialGroup()  
    .addGroup(layout.createParallelGroup(BASELINE)  
        .addComponent(label)  
        .addComponent(textField)  
        .addComponent(findButton))  
    .addGroup(layout.createParallelGroup(LEADING)  
        .addGroup(layout.createSequentialGroup()  
            .addGroup(layout.createParallelGroup(BASELINE)  
                .addComponent(caseCheckBox)  
                .addComponent(wrapCheckBox))  
            .addGroup(layout.createParallelGroup(BASELINE)  
                .addComponent(wholeCheckBox)  
                .addComponent(backCheckBox)))  
        .addComponent(cancelButton))  
    );
```

GroupLayout gyakorlat

- *GroupLayout* felhasználásával helyezzünk el 5 db komponenst (north, south, west, east, center) egy konténeren a *BorderLayout*-nak megfelelő kiosztásban!

Megoldás

```
GridLayout layout = new GridLayout(getContentPane());
    getContentPane().setLayout(layout);
layout.setAutoCreateGaps(true);
layout.setAutoCreateContainerGaps(true);

layout.setHorizontalGroup
    (layout.createParallelGroup(CENTER)
        .addComponent(north)
        .addGroup(layout.createSequentialGroup()
            .addComponent(west)
            .addComponent(center)
            .addComponent(east)
        )
        .addComponent(south));
```

Megoldás folyt.

```
layout.setVerticalGroup(layout.createSequentialGroup()  
    .addComponent(north)  
    .addGroup(layout.createParallelGroup(CENTER)  
        .addComponent(west)  
        .addComponent(center)  
        .addComponent(east)  
    )  
    .addComponent(south));  
  
layout.linkSize(SwingConstants.VERTICAL, north, south);  
layout.linkSize(SwingConstants.HORIZONTAL, west, east);
```



Swing alapok

Swing

- Widget készlet

- 1996 Internet Foundation Classes (Netscape)
 - 1997 Java Foundation Classes (Netscape+Sun)
 - JSE 1.2-től standard könyvtár

- **javax.swing** package

- AWT-től eltérő filozófia

- Java nyelven írva, MVC, Look and feel váltás stb.

- Átgondolt tervezés

Swing jellemzők

■ Platformfüggetlen

- widget-ek Java-ban írva
- mindenhol ugyanúgy néz ki

■ Bővíthető

- tagolt architektúra
- programozók saját megoldással bővíthetik a keretrendszert
 - meglevő felüldefiniálásával
 - új megalkotásával

Swing jellemzők 2

■ Komponens-orientált

- aszinkron eseményküldés
- kötött property-k
- jól definiált parancsok
- minden komponens egyben *Java Bean* is

■ Testreszabható

- szabványos elemkészletből épülő komponensek
 - pl. keret, dekoráció, háttér
- programból szabhatók az egyes elemek
 - property-ként érhetők el és módosíthatók

Swing jellemzők 3

■ Konfigurálható

- indirekt kompozíció és futás közbeni mechanizmusok segítik a futás közbeni konfigurálást
- meglevő kód újrafordítás nélkül is alakítható
- pl. look and feel bármikor lecserélhető

■ Lightweight UI

- a konfigurálhatóság oka, hogy nem natív megoldásokat használ
- Java-ból rajzolja ki az egyes elemeket a Java 2D API segítségével

Swing jellemzők 4

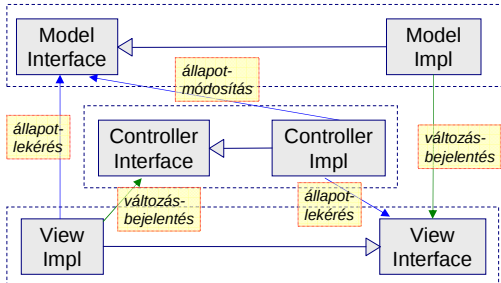
■ Lazán csatolt

- MVC minta komoly alkalmazása az egész rendszerben
- szétcsatolja a tényleges megjelenítést a megjelenítendő modelltől
- a legtöbb elemhez saját modell interfész tartozik
- ezekhez egyszerű implementációkat ad

Kapcsolat az AWT-vel

- Cél volt a kapcsolat minimalizálása
 - eseménykezelés azonos
 - **JContainer**, **JApplet**, **JDialog**, **JFrame**, **JWindow** a megfelelő AWT elemek leszármazottai
 - **JComponent** a **Container** leszármazottja
- A kirajzolás a Java 2D API-n alapul
- Kerülni kell a keveredést
 - elfedhetik egymást a komponensek

MVC minta



Áttérés AWT-ről

- Egyszerű komponensek esetén triviális
 - **JLabel**
 - **JButton**
 - **JScrollbar**
 - **JTextField, JTextArea**
 - **JPanel, JFrame, JWindow**
- Plusz szolgáltatásokat nyújtanak

Átérés példa

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Test2 implements ActionListener{
    JButton b;
    JTextArea ta;
    JTextField tf;
    public void actionPerformed(ActionEvent ae) {
        if (ae.getActionCommand().equals("Test")) {
            ta.append(tf.getText()+"\n");
            tf.setText("");
        }
    }
    static public void main(String args[]) {
        (new Test2()).run();
    }
    ...
}
```

Átérés példa

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Test2 implements ActionListener{
    JButton b;
    JTextArea ta;
    JTextField tf;
    public void actionPerformed(ActionEvent ae) {
        if (ae.getActionCommand().equals("Test")) {
            ta.append(tf.getText()+"\n");
            tf.setText("");
        }
    }
    static public void main(String args[]) {
        (new Test2()).run();
    }
    ...
}
```

Átérés példa

```
...  
public void run() {  
    Frame f = new Frame("AWT Example");  
  
    b = new Button("Test");  
    tf = new TextField();  
    ta = new TextArea("", 10, 30);  
  
    b.addActionListener(this);  
  
    f.add(b, BorderLayout.NORTH);  
    f.add(new Scrollbar(), BorderLayout.EAST);  
    f.add(ta, BorderLayout.CENTER);  
    f.add(tf, BorderLayout.SOUTH);  
    ...  
}
```

Átérés példa

```
...  
public void run() {  
    JFrame f = new JFrame("AWT Example");  
  
    b = new JButton("Test");  
    tf = new JTextField();  
    ta = new JTextArea("", 10, 30);  
  
    b.addActionListener(this);  
  
    f.add(b, BorderLayout.NORTH);  
    f.add(new JScrollBar(), BorderLayout.EAST);  
    f.add(ta, BorderLayout.CENTER);  
    f.add(tf, BorderLayout.SOUTH);  
    ...  
}
```

Átérés példa

```
...  
Menu m = new Menu("Menu");  
m.add(new MenuItem("Test"));  
MenuBar mb = new MenuBar();  
mb.add(m);  
f.setMenuBar(mb);  
f.pack();  
f.show();  
}  
}
```

Áttérés példa

```
...  
JMenu m = new JMenu("Menu");  
m.add(new JMenuItem("Test"));  
JMenuBar mb = new JMenuBar();  
mb.add(m);  
f.setJMenuBar(mb);  
f.pack();  
f.show();  
}
```

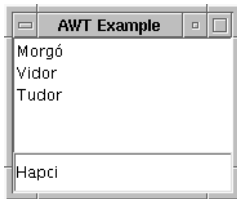
```
}
```

Áttérés AWT-ről 2

- Összetettebb komponensek esetén nem elég a névváltás
 - **JList**
 - **JComboBox**
 - **JChoice**
- Általában szükséges a modell implementálása az MVC minta korrekt működéséhez

Áttérés példa: *List* $\rightarrow$ *JList*

- **Feladat:** készítsünk alkalmazást, amiben szövegmezőbe írt sorok megjelennek egy listában



AWT megoldás

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class AWTList implements ActionListener{
    TextField tf;
    List list;
    public void actionPerformed(ActionEvent ae) {
        list.add(tf.getText());
        tf.setText("");
    }
    static public void main(String args[]) {
        (new AWTList()).run();
    }
    ...
}
```

AWT megoldás

```
...  
public void run() {  
    Frame f = new Frame("AWT Example");  
    tf = new TextField("", 20);  
    list = new List(5);  
    tf.addActionListener(this);  
    f.add(tf, BorderLayout.SOUTH);  
    f.add(list, BorderLayout.CENTER);  
    f.pack();  
    f.show();  
}  
}
```

Swing megoldás

- Szokásos cserékkel majdnem megy
 - **Frame** → **JFrame**
 - **TextField** → **JTextField**
 - eseménymodell nem változik
- Kivétel: **JList**
 - nincs **add(...)** metódusa
 - a mérete nem állítható

Swing megoldás: **JList**

- MVC minta szerint működik

- A modell lehet:

- **ListModel**

- **Vector**

- **Object[]**

- Nem görgethető, a mérete nem állítható

- JScrollPane-be kell tenni

- **JScrollPane scrollPane = new JScrollPane(list);**

- *(Decorator minta)*

ListModel

■ interface javax.swing.ListModel

- Object getElementAt(int index)

- visszaadja az indexedik elemet

- int getSize()

- megadja a tárolt elemek számát

- void removeListDataListener
(ListDataListener l)

- void addListDataListener
(ListDataListener l)

- listenert regisztrál/töröl a modellben

ListDataListener

- Amikor a modell változik, értesíti a *listener*-eket
- A **JList**-ben van egy ilyen:
 - **BasicListUI.ListDataHandler**
- Metódusai
 - **void intervalAdded(ListDataEvent e)**
 - **void intervalRemoved(ListDataEvent e)**
 - az *e*-ben meghatározott intervallum megváltozott
 - **void contentsChanged(ListDataEvent e)**
 - bonyolultabb változás történt

DefaultListModel

- Megvalósítja a **ListModel** interfészt
- **java.util.Vector**-ből ismerős műveletek egy részét is támogatja
 - `void add(int index, Object o)`
 - `int size()`
 - `Object get(int index)`
 - `Object remove(int index)`
 - ...

Swing megvalósítás

```
public class SwingList implements ActionListener{
    TextField tf;
    List list;

    public void actionPerformed(ActionEvent ae) {
        list.add(tf.getText());
        tf.setText("");
    }
    static public void main(String args[]) {
        (new SwingList()).run();
    }
    ...
}
```

Swing megvalósítás

```
public class SwingList implements ActionListener{
    JTextField tf;
    JList list;
    DefaultListModel model;
    public void actionPerformed(ActionEvent ae) {
        model.addElement(tf.getText());
        tf.setText("");
    }
    static public void main(String args[]) {
        (new SwingList()).run();
    }
    ...
}
```

Swing megvalósítás

```
...  
public void run() {  
    Frame f = new Frame("AWT Example");  
    tf = new TextField("", 20);  
  
    list = new List(5);  
  
    tf.addActionListener(this);  
    f.add(tf, BorderLayout.SOUTH);  
    f.add(list, BorderLayout.CENTER);  
    f.pack();  
    f.show();  
}  
}
```

Swing megvalósítás

```
...  
public void run() {  
    JFrame f = new JFrame("AWT Example");  
    tf = new JTextField("", 20);  
    model = new DefaultListModel();  
    list = new JList(model);  
    JScrollPane pane = new JScrollPane(list);  
    tf.addActionListener(this);  
    f.add(tf, BorderLayout.SOUTH);  
    f.add(pane, BorderLayout.CENTER);  
    f.pack();  
    f.show();  
}  
}
```



JList példa bemutatás

Feladat-elemzés

- **Model: DefaultListModel**

- ☐ elemek kollekcióját kezeli
- ☐ ha változik, értesíti a *View*-t

- **View: JList** (*+BasicListUI.ListDataHandler*)

- ☐ csak megjelenít

- **Controller: SwingList** (*implements ActionListener*)

- ☐ események és a *View* állapota alapján módosítja a *Model*-t

JTable

- Táblázat-nézet
- **JScrollPane** kell a görgetéshez
- Modellje a **TableModel**
- Oszlopok tetszőlegesen mozgathatók
- Sorok tetszőleges oszlop alapján rendezhetők
 - **TableRowSorter**
 - a modell tartalmát nem befolyásolja
 - összehasonlítás a *sorter* dolga
 - `setComparator(int column, Comparator<?> comparator)`

JTable példa

- **Feladat:** hallgatók rekordjait jelenítsük meg táblázatosan

1) saját modell kell, mert egy sor egy objektum

2) rendezni lehessen oszlopok szerint

a) a rekord csak tárol

b) a modell tudja, hogy hány mező, milyen névvel, stb.

JTable példa: *Student* rekord

```
class Student {  
    String name;  
    String neptun;  
    double average;  
    public Student(String s1, String s2, double s3) {  
        name = s1;  
        neptun = s2;  
        average = s3;  
    }  
    public String toString() {  
        return name+" (" +neptun+"): "+average;  
    }  
}
```

JTable példa: *StudentModel*

```
class StudentModel extends AbstractTableModel {
    Vector<Student> vector;
    public StudentModel() {super();
        vector = new Vector<Student>();
    }
    public void add(Student s) {vector.add(s);}
    public boolean isCellEditable(int r, int c) {
        return (c!=1);
    } // különben nem editálhatóak a mezők
    public String getColumnName(int col) { ... }
    public void setValueAt(Object aValue,
        int rowIndex, int columnIndex) { ...
    } // ne felejtsük el a fireTableXXX() metódust!
    public Object getValueAt(int r, int c) { ... }
    public int getColumnCount() {return 3;}
    public int getRowCount() {return vector.size();}
}
```

JTable példa: *TableModelListener*

```
...
public void tableChanged(TableModelEvent e) {
    System.out.println(e);
    System.out.println(e.getColumn()
        +" "+e.getFirstRow()+" "+e.getType());
    if (e.getType() == e.UPDATE) {
        System.out.println
            (((TableModel)e.getSource()).
                getValueAt(e.getFirstRow(),
                    e.getColumn()));
    }
}
...
```

JTable példa: *TableExample*

```
...  
JFrame f = new JFrame("Swing Example");  
  
StudentModel model = new StudentModel();  
model.addTableModelListener(this);  
model.add(new Student("Gáz Géza", "ABCDEF", 3.5));  
... // további adatok hozzáadása  
  
JTable table = new JTable(model);  
table.setRowSorter(new TableRowSorter(model));  
JScrollPane pane = new JScrollPane(table);  
f.add(pane, BorderLayout.CENTER);  
...
```



JTable példa bemutatás

JTable példa elemzés

■ Model: StudentModel

- kezeli a táblázat elemeit
- megadja a táblázat adatait (sorok száma, oszlopok neve); eldönti, hogy mi szerkeszthető
- metaadatok: *View vs Model*

■ View: JTable

- megjelenít
- egyben *Controller* is: szerkesztéshez nem kell más

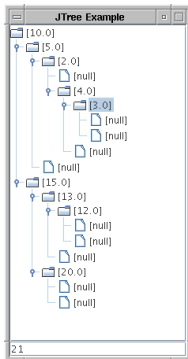
■ Controller: -

JTree

- Hierarchikus adatszerkezet megjelenítésére
- Modellje a **TreeModel**
- Inicializálható egy **TreeNode**-fával is
 - a gyökérelem megadásával

JTree példa

- **Feladat:** készítsünk bináris fát megjelenítő alkalmazást!
 - a fában valósokat tárolunk
 - új elemet hozzáadhatunk



JTree példa

■ BinTree

- bináris fa implementáció
- a null elemet null objektum képviseli
 - egyszerűbb az implementáció
 - *Null Object minta*
 - pl. ilyen a láncolt lista strázsája is
- beszúrásnál vissza kell tudni adni az új elem elérési útját

JTree példa

■ **BinTreeModel**

- a **JTree** ezt használja modelljének
- **BinTree** a háttérben
 - **BinTree root = new BinTree()**
- **TreeModelListener**-ek kezelése
 - **Vector<TreeModelListener> listeners**
 - **public void
addTreeModelListener(TreeModelListener l)**
 - **public void
removeTreeModelListener(TreeModelListener l)**

JTree példa

■ **BinTreeModel** (folyt)

□ a modell leírása

- `public Object getChild(Object parent,
int index)`
- `public int getChildCount(Object parent)`
- `public int getIndexOfChild(Object parent,
Object child)`
- `public Object getRoot()`
- `public boolean isLeaf(Object node)`
- *`public void valueForPathChanged(TreePath
path, Object newValue)`*
- `public void insert(double d)`

JTree példa: alkalmazás

```
JTextField tf;  
BinTreeModel btm;
```

```
public void actionPerformed(ActionEvent ae) {  
    double d = Double.parseDouble(tf.getText());  
    btm.insert(d);  
    tf.setText("");  
}
```

```
tf = new JTextField("", 20);  
btm = new BinTreeModel();  
JTree tree = new JTree(btm);  
JScrollPane scrollPane = new JScrollPane(tree);  
tf.addActionListener(this);
```



JTree példa bemutatás

JTree példa bővítés

■ Feladat

- módosítsuk úgy az alkalmazást, hogy a faelemek *tooltip*-je az adott részfa mélységét írja ki!

■ Megoldás

- a fa-elemeknek mélységet kell számolnia
- implementálni kell egy **TreeCellRenderer**-t
 - ez felelős a fa-elemek megjelenéséért
 - **getTreeCellRendererComponent** metódust kell megvalósítani
- a *renderer*-t regisztrálni kell a fánál
- a fát regisztrálni kell a **TooltipManager**-nél

JTree példa 2: mélységszámolás

```
class BinTree {  
    ...  
    public int getDepth() {  
        if (isNull()) { return 1; }  
  
        int l = left.getDepth();  
        int r = right.getDepth();  
        int m = (l>r) ? l : r;  
        return m+1;  
    }  
}
```

JTree példa 2: renderer

```
class BinTreeRenderer extends DefaultTreeCellRenderer {  
    public Component getTreeCellRendererComponent(  
        JTree tree, Object value, boolean sel,  
        boolean expanded, boolean leaf, int row,  
        boolean hasFocus) {  
  
        super.getTreeCellRendererComponent(  
            tree, value, sel, expanded, leaf, row, hasFocus);  
  
        BinTree bt = (BinTree)value;  
        setToolTipText("(" + bt.getDepth() + ")");  
  
        return this;  
    }  
}
```

JTree példa 2: regisztrálás

```
JTree tree = new JTree(btm);
```

```
ToolTipManager.sharedInstance()  
    .registerComponent(tree);
```

```
tree.setCellRenderer(new BinTreeRenderer());
```



JTree példa 2 bemutatás



Drag and Drop

Drag and drop



Drag and Drop

■ `JComponent.setDragEnabled(boolean b)`

alapból tudják

- ☐ `JEditorPane`
- ☐ `JFormattedTextField`
- ☐ `JPasswordField`
- ☐ `JTextArea`
- ☐ `TextField`
- ☐ `JTextPane`
- ☐ `JColorChooser`

kis segítséggel tudják

- ☐ `JList`
- ☐ `JTable`
- ☐ `JTree`

Drag and drop beállítása

■ **setDragEnabled(boolean b)**

- bekapcsolja a Drag-N-Dropot

■ **setDropMode(DropMode dm)**

- **INSERT**: két elem közé szúrja
- **INSERT_COLS** / **INSERT_ROWS**: ugyanez csak táblában
- **ON**: elemre rakja
- **USE_SELECTION**: elem saját kiválasztó módján
- **ON_OR_INSERT**, **ON_OR_INSERT_COLS**, **ON_OR_INSERT_ROWS**: kombinációban

Kis segítség: **TransferHandler**

- **setTransferHandler(TransferHandler th)**
 - a **TransferHandler**-ben implementálhatjuk az átadás részleteit
 - **int getSourceActions(JComponent)**
 - COPY, MOVE és LINK
 - **Transferable**
createTransferable(JComponent)
 - elkészíti az átvendő adatot
 - **void exportDone(JComponent c, Transferable t, int action)**
 - meghívódik, amikor az exportnak vége

Kis segítség: **TransferHandler**

- **boolean canImport(TransferHandler.
TransferSupport ts)**
 - megadja, hogy a fogadó fél képes-e az adat fogadására
- **boolean importData(TransferHandler.
TransferSupport ts)**
 - meghívódik, amikor az importnak le kell zajlania
 - visszatérési érték jelzi, hogy sikeres volt-e

Transferable, DataFlavor

■ Transferable

- **Object** `getTransferData(DataFlavor flavor)`
 - visszaadja az adott adattípusnak megfelelő adatot
- **DataFlavor[]** `getTransferDataFlavors()`
 - visszaadja a támogatott adattípusokat
- **boolean** `isDataFlavorSupported(DataFlavor flavor)`
 - megadja, hogy az adott adattípust támogatja-e

■ DataFlavor

- *imageFlavor, javaFileListFlavor, stringFlavor*
- **DataFlavor(Class<?> representationClass, String humanPresentableName)**

TransferSupport

- **Component** **getComponent()**
 - visszaadja a cél komponenst
- **int** **getDropAction()**
 - COPY, MOVE vagy LINK
- **int** **getSourceDropActions()**
 - megadja, hogy a forrás milyen akciókat támogat
- **DataFlavor[]** **getDataFlavors()**
 - mint a **Transferable**-nél

TransferSupport 2

- **boolean**
isDataFlavorSupported(DataFlavor)
 - mint a **Transferable**-nél
- **Transferable** **getTransferable()**
 - visszaadja az adatot
- **DropLocation** **getDropLocation()**
 - megadja, hogy hova kell tenni
- . . .

Drag and drop példa

- **Feladat:** legyen két listánk és a kettő között tudjunk sorokat pakolni
- Kell:
 - saját **TransferHandler**
 - *INSERT* beszúrással

Drag and drop példa

```
class HanoiHandler extends TransferHandler {  
  
    public int getSourceActions(JComponent c) {return MOVE;}  
  
    public Transferable createTransferable(JComponent c) {  
        JList l = (JList)c;  
        DefaultListModel m = (DefaultListModel)l.getModel();  
        int i = l.getMinSelectionIndex();  
        if (i < 0) return null;  
        return new StringSelection(""+m.get(i));  
        // előállít egy String-et tartalmazó Transferable-t  
    }  
    ...  
}
```

Drag and drop példa

```
...
public void exportDone(JComponent c, Transferable t,
                       int action) {
    if (action == MOVE) {
        JList l = (JList)c;
        DefaultListModel m = (DefaultListModel)l.getModel();
        int i = l.getMinSelectionIndex();
        if (i < 0) return;
        m.remove(i);
    }
}
public boolean canImport(TransferHandler.
                        TransferSupport support) {
    return true;
}
...
```

Drag and drop példa

```
...
public boolean importData(TransferHandler.
                           TransferSupport support) {
    try {
        JList l = (JList)support.getComponent();
        String s = (String)support.getTransferable()
            .getTransferData(DataFlavor.stringFlavor);
        JList.DropLocation drop =
            (JList.DropLocation)support.getDropLocation();
        DefaultListModel m = (DefaultListModel)l.getModel();
        m.add(drop.getIndex(), s);
        return true;
    } catch (Exception e) {e.printStackTrace();}
    return false;
}
```

Drag and drop példa

```
JList list1, list2;  
DefaultListModel model1, model2;  
JFrame f = new JFrame("DND Example");  
model1 = new DefaultListModel();  
model2 = new DefaultListModel();  
list1 = new JList(model1);  
list2 = new JList(model2);  
JScrollPane pane1 = new JScrollPane(list1);  
JScrollPane pane2 = new JScrollPane(list2);  
list1.setDragEnabled(true);  
list1.setDropMode(DropMode.INSERT);  
HanoiHandler hh = new HanoiHandler();  
list1.setTransferHandler(hh);  
...  
f.add(pane1, BorderLayout.WEST);  
f.add(pane2, BorderLayout.EAST);
```



Drag and drop bemutatás



Szálkezelés és GUI

Szálkezelés Swing alatt

- Swing nem szál-biztos (thread-safe)
- Alapból az eseménykezelő szálát kell használni
- Legtöbbször nincs probléma
 - pl. **JButton** megnyomása a listener-eket hívja meg egyenként
- Gond lehet a GUI felépítésekor
 - **main** metódusból ne építsünk
 - helyette
 - `SwingUtilities.invokeLater(Runnable r)`
 - `SwingUtilities.invokeAndWait(Runnable r)`

Szálkezelés Swing alatt 2

```
public class MyApp implements Runnable {  
    public void run() {  
        // Az eseménykezelő szálból hívva.  
        // GUI építése, megjelenítése.  
    }  
  
    public static void main(String[] args) {  
        SwingUtilities.invokeLater(new MyApp(args));  
    }  
}
```

Szálkezelés Swing alatt 3

```
public class MyApp {  
    MyApp(String[] args) {  
        // Inicializálás. Eseménykezelő szálból hívva.  
    }  
    public void show() {...} // GUI megjelenítés  
    public static void main(final String[] args) {  
        SwingUtilities.invokeLater(new Runnable() {  
            public void run() {  
                new MyApp(args).show();  
            }  
        });  
    }  
}
```

Szálkezelés Swing alatt 4

■ Modellek kezelésekor is

- modelleket csak az eseménykezelő szálon keresztül módosítsunk
- ha nem, akkor az esetleges kivételek tönkretehetik a GUI-t

■ Ha szükséges, használjuk a **SwingWorker** osztályt (***javax.swing** csomag*)

- pl. hosszú I/O művelet, számítás stb. esetén
- különben nem reagál az alkalmazás

SwingWorker<T, V>

- Úgy viselkedik, mint egy szál
 - párhuzamosan fut az eseménykezelővel
 - csak egyszer futtatható
- Visszatérési értéke van
 - ez a futás végeztével elérhető
 - tetszőleges típus lehet (genericitás)
- Képes kódot futtatni az eseménykezelő szálban
 - hasznos, ha modellt kell módosítani

SwingWorker<T, V> 2

- **protected abstract T doInBackground()**
 - az elvégzendő munka (mint `Thread.run()`)
 - kötelező implementálni
- **void execute()**
 - elindítja a szálát (mint `Thread.start()`)
- **protected void done()**
 - meghívva, ha a szál véget ért
- **boolean isDone()**
 - true, ha a szál véget ért
- **T get(long timeout, TimeUnit unit)**
 - visszaadja a `doInBackground()` által visszaadott értéket

SwingWorker<T, V> 3

- **void setProgress(int i)**
 - beállítja az előrehaladást mutató értéket (0-100)
- **int getProgress()**
 - visszaadja a fent beállított értéket
- **void cancel(boolean mayInterruptIfRunning)**
 - cancelled állapotba viszi a folyamatot
- **boolean isCancelled()**
 - true, ha **cancel()** meg lett hívva

SwingWorker<T, V> 4

- Ha valamit az eseménykezelő számban kell végezni
- **protected final void publish(V... chunks)**
 - chunks-ot gyűjti és átadja process()-nek
 - aszinkron a kapcsolatuk
- **protected void process(List<V> chunks)**
 - az eseménykezelő számból hívva
 - a publish() metódusban átadott értékeket kapjuk itt meg

SwingWorker<T, V> 5

- **Property**-ben lekérdezhető és megfigyelhető az állapot
 - **progress** és **state**
- **PropertyChangeListener** ...
- **public final SwingWorker.StateValue getState()**
 - **PENDING** indítás előtt
 - **STARTED** fut, de még nem állt le
 - **DONE** megállt

Multithreaded GUI?

- Miért lehet csak egy eseménykezelő szál?
 - miért bonyolítunk *SwingWorker*-rel?
- Sokszor próbálták
 - Cedar GUI – XEROX, korai 80-as évektől
- Failed dreams
 - GUI: *top-down* és *bottom-up* keverve
 - események alulról jönnek
 - kezelés felülről programozva
 - vagy *deadlock* vagy *race condition*
 - egyszerűbb ha maradunk a *single-thread*-nél

Whiteboard minta

Listener modell jellemzői

■ Előnyök

- elegáns, szép megoldás
- szépen elválik a forrás (view) és a feldolgozás (control)

■ Hátrányok

- nagy az overhead
 - >130 event, adapter és listener interface
 - legtöbbször csak egy listener, mégis sokra készülünk
 - szörnyosztályok: **AWTEventMulticaster**
- nincs gond, ha van elég memória és CPU

Listener modell jellemzői 2

■ Hátrányok

- függőség az esemény forrása és a listener között – életciklus-problémák
 - ha bármelyik megszűnne, a másiknak ezt tudomásul kell vennie
- normál Java alkalmazásban nem nagyon jön elő a baj
 - inicializálás triviális
 - lecsatlakozásról mindenki megfeledkezik
 - amikor az alkalmazás megáll, úgyis megszűnik minden

Beágyazott Java jellemzői

■ Kis eszközök

- kevés memória, relatíve lassú CPU
→ nem pazarolhatunk
- nincsenek adapter osztályok
- nem töltünk be felesleges objektumot

■ Kollaborációs modell

- nem alkalmazások futnak, hanem *bundle*-ök (csomagok)
- szolgáltatások a *service registry*-be regisztrálnak

Beágyazott Java jellemzői 2

■ Folyamatosan futó VM

- nem ragadhatnak be objektumok
→ a korrekt élelciklus-kezelés elengedhetetlen
- nem igaz az, hogy „*amíg van referenciánk egy objektumra, az nem szűnik meg*”
 - erőforrás-kezelés kapcsán sokszor dinamikusán jönnek létre és szűnnek meg objektumok

■ A fenti jellemzők miatt a klasszikus listener eseménykezelés nem alkalmazható

Whiteboard minta

- Az egyedi eseménykezelés helyett a *registry*-t használjuk
- A *listener*-ek a *registry*-be regisztrálnak
- Ha jön egy esemény, nem a *listener*-t értesítjük, hanem a *registry*-t
- Az eseményforrás nem regisztrál

Whiteboard minta 2

■ Registry előnyei

☐ hibakeresés

- a fejlesztőeszközök támogatják a *registry*-be való betekintést

☐ biztonság

- *ServicePermission*-nel állítható a regisztrált *listener*-ek hozzáférése az eseményekhez

☐ Property-k

- segítségükkel válogathatunk a *listener*-ek között



Swing Look and Feel

Swing Look and Feel

■ Default L&F

- ☐ Steel, Ocean
- ☐ GTK+, Motif
- ☐ Windows

■ Beállítás

- ☐ parancssorból
- ☐ programból
- ☐ property-fájlból

Swing Look and Feel 2

■ Implementálás

- programból

 - **`javax.swing.plaf.basic`**

- XML-fájlal

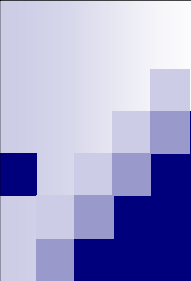
 - **`javax.swing.plaf.synth`**

- multiplex – más UI-ra ráültetve

 - **`javax.swing.plaf.multi`**



Swing Look and Feel bemutató



Objektumorientált szoftvertervezés

3. Perzisztencia

Bevezetés

■ Perzisztencia:

- kitartás, állhatatosság, szívósság

■ Persistence

- , the characteristic of data that outlives the execution of the program that created it. Without this capability, data only exists in memory, and will be lost when the memory loses power. (*wikipedia*)
- is the ability of the programmer to have her/his data survive the execution of a process, in order to eventually reuse it in another process.
(*OODB System Manifesto, Univ. Leeds*)

Bevezetés

■ Objektumok

- ☐ Viselkedés
- ☐ Szerkezet
- ☐ Állapot
- ☐ Egyediség

■ Élettartam

- ☐ Automatikus változók: blokk
- ☐ Attribútumok: objektum
- ☐ Osztályváltozók: program futása

Bevezetés

- Hogyan élhetjük túl a restartot ?
 - Mi az, hogy túlélni ?
 - Viselkedés és struktúra ?
 - Állapot visszaállítása
 - Mi van a referenciákkal ?

Bevezetés

- Hogyan élhetjük túl a restartot ?
- Megoldások
 - Szerializálás, fájlkezelés
 - Egyszerű, gyors, nincs query

Bevezetés

- Hogyan élhetjük túl a restartot ?
- Megoldások
 - Relációs adatbázis
 - gyors, query, transzformáció
 - "Using tables to store objects is like driving home and then disassembling the car to put in the garage. It can be assembled again in the morning, but one eventually asks whether this is the most efficient way to park a car." *Esther Dyson*
 - Relációs séma kontra objektum modell

Bevezetés

- Hogyan élhetjük túl a restartot ?
- Megoldások
 - Objektum orientált adatbázisok
 - Egyszerű, gyors, query
 - Nincs transzformáció, OO modell

Tartalomjegyzék

- Egy egyszerű feladat
- Megoldás erőből
 - Szerializálás
- Éljen a relációs adatbázis !
 - Hibernate 3.0
- Még absztraktabban !
 - Java Persistence API, EclipseLink
- Maradjunk az OO-nál !
 - Objectstore - PSEPro

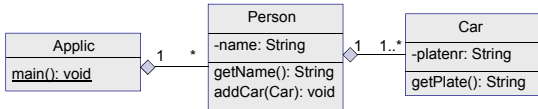


Egy egyszerű feladat

Feladatdefiníció

■ autók nyilvántartása

- vegyünk fel autót (tulajdonossal)
- listázzuk az autókat tulajdonosonként
- korlátozások
 - rendszám egyedi
 - tulajdonos egyedi





Tranziens megoldás



Serializálás

Terminológia

- folyamat (*serialization*), amelyben egy objektum attribútumait bináris formába alakítjuk át.
 - *deflating, marshalling*
- az ellentétes folyamatban (*deserialization*) az adatstruktúrát helyreállítjuk a bináris formából
 - *inflating, unmarshalling*

Kimentés

```
import java.io.Serializable;
<mod> class SerializableClass
    implements Serializable
{ ... }

import java.io.*;
try {
    FileOutputStream f =
        new FileOutputStream("filename");
    ObjectOutputStream out =
        new ObjectOutputStream(f);
    out.writeObject(_SerializableClass);
    out.close();
}
catch(IOException ex) { ... }
```

Visszaállítás

```
import java.io.*;
try {
    FileInputStream f =
        new FileInputStream("filename");
    ObjectInputStream in =
        new ObjectInputStream(f);
    o_ins = (SerializableClass)in.readObject();
    in.close();
}
catch(IOException ex) { ... }
catch(ClassNotFoundException ex) { ... }
```

■ Nincs konstruktor hívás !

Szabályok, folyamat

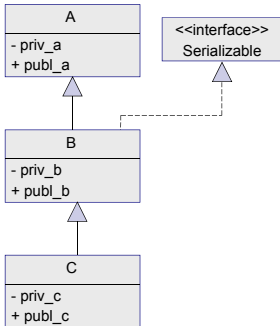
- Csak a `Serializable` interface-t implementáló osztályok (és azok leszármazottai) szerializálhatóak
- `Serializable` – formális
- tömbök **IGEN**
- `java.lang.Object` **NEM**
- `Thread`, `OutputStream`, `Socket` **NEM**

Szabályok, folyamat

- `writeObject` – tranzitív
 - az első előfordulás tárolt, a többi csak platformfüggetlen azonosító
 - ismételt kiírás – NEM felülírás

```
out.writeObject(_a);  
out.writeObject(_a);
```
- mindegyik nem `transient`, nem `static` attribútum
- legősibb szerializálhatótól indul

Öröklés

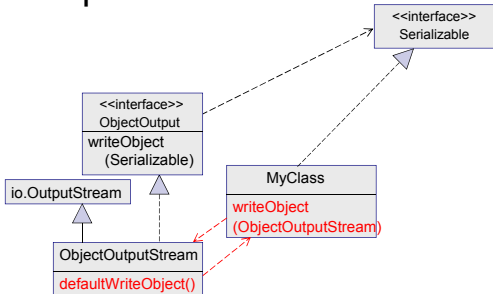


Öröklés állj !!

```
private void writeObject(ObjectOutputStream out)
                        throws IOException
{
    throw new NotSerializableException("X");
}

private void readObject(ObjectInputStream in)
                        throws IOException, ClassNotFoundException
{
    throw new NotSerializableException("X");
}
```

Szereplők



Saját szerializálás

```
private void writeObject(ObjectOutputStream out)
                        throws IOException
{
    out.defaultWriteObject();
}

private void readObject(ObjectInputStream in)
                        throws IOException, ClassNotFoundException
{
    in.defaultReadObject();
    // inicializálás
}
```

ObjectOutput

```
public interface ObjectOutput extends DataOutput
{
    public void writeObject(Object obj) throws
                        IOException;
    public void write(int b) throws IOException;
    public void write(byte b[]) throws IOException;
    public void write(byte b[], int off, int len)
                        throws IOException;
    public void flush() throws IOException;
    public void close() throws IOException;
}
```

ObjectInput

```
public interface ObjectOutput extends DataOutput
{
    public Object readObject()
        throws ClassNotFoundException, IOException;
    public int read() throws IOException;
    public int read(byte b[]) throws IOException;
    public int read(byte b[], int off, int len)
        throws IOException;
    public long skip(long n) throws IOException;
    public int available() throws IOException;
    public void close() throws IOException;
}
```

Serializálható mezők

■ serialPersistentFields

```
class Datum implements Serializable {
    int ev, honap, nap;
    ...
    private static final ObjectOutputStreamField[]
        serialPersistentFields =
        { new ObjectOutputStreamField("datum", int.class) };

    private void writeObject( ObjectOutputStream out )
        throws IOException {
        ObjectOutputStream.PutField adatok = out.putFields();
        adatok.put("datum", 10000*ev+100*honap+nap);
        out.writeFields();
    }
}
```

Serializálható mezők

■ serialPersistentFields

```
private void readObject( ObjectInputStream in ) throws  
    IOException, ClassNotFoundException {
```

```
    ObjectInputStream.GetField adatok = in.readFields();  
    int d = adatok.get("datum",0);  
    nap   = (int) (d % 100);  
    honap = (int) ((d / 100) % 100);  
    ev    = (int) (d / 10000);
```

```
    }  
}
```

Helyettesítés

- Interceptor – serializálás közben
 - visszatér egy objektumot,
 - amiből
 - amibe serializálunk

```
<mod> Object writeReplace() throws  
                                ObjectOutputStreamException;
```

```
<mod> Object readResolve() throws  
                                ObjectOutputStreamException;
```

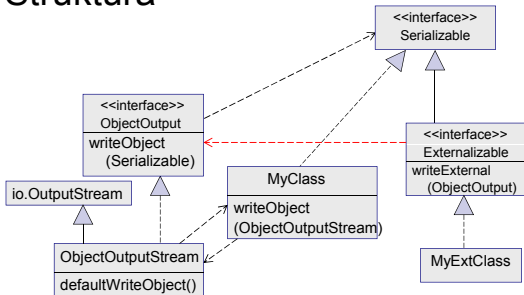
Externalizálás

- Csak az osztály azonosítása mentődik, a többi a user kezében.

```
public interface Externalizable extends Serializable
{
    public void writeExternal(ObjectOutput out)
        throws IOException;

    public void readExternal(ObjectInput in)
        throws IOException,
            java.lang.ClassNotFoundException;
}
```

Struktúra



Verziók

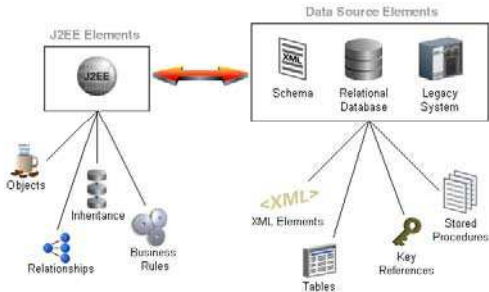
■ Kompatibilis

- Új mezőkkel bővül
- `static` -> `pers`, `transient` -> `pers`
- hozzáférés módosítás
- öröklési fa bővítése, szűkítése
- új `writeObject`, `readObject`



Objektum relációs adatbázis

Probléma lényege



Hibernate 3.0

- Open-source
- Objektumok táblákra leképezve
- Lekérdező nyelv
- A standard DB-hez kapcsolódik
- Swing, Servlet, J2EE kapcsolat
- nagy teljesítmény

Jellemzők

- (LGPL) Lesser General Public License
- Lekérdezés
 - Hibernate Query Language
 - Hibernate Criteria Query API
 - konkrét DB SQL-je
- Eclipse support
- Teljesítmény monitor: JMX, helyi API
- Automatikus kulcs generálás

Jellemzők

- Támogatja az OO-t
 - Öröklés, polimorfizmus, Java Collection
- Skálázható

Megközelítés

- „To use Hibernate, it is required to create Java classes that represents the table in the database and then map the instance variable in the class with the columns in the database. Then Hibernate can be used to perform operations on the database like select, insert, update and delete the records in the table. Hibernate automatically creates the query to perform these operations.” *(Hibernate Reference Documentation 3.2.6)*

A feladat megoldása

■ Perzisztens osztályok kiegészítése

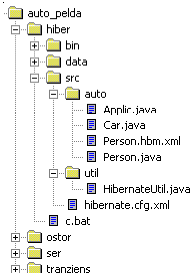
- `Long id` – attribútum - ajánlott
- standard `get`, `set` metódusok - ajánlott
- `public no-arg constructor`

■ Alkalmazás átalakítása

- adatbázis adminisztráció
- `car-on` rendszámot, majd `Person`-t keresünk
- `list` - ugyanaz

A feladat megoldása

- Perzisztens osztály + asszociációk → tábla mapping
- Hibernate configuration – xml
- HSQL DB indítása
- Alkalmazás indítása



Hibernate 3.0 definiálása

■ Leképezés

- ☐ Tag-ek, típusok, kollekciók
- ☐ Asszociáció, komponens
- ☐ Öröklés

■ Működés

- ☐ Architektúra, objektumok kezelése
- ☐ Tranzakciók, konkurencia
- ☐ HQL

Leképezések / tag-ek

- `<hibernate-mapping>` gyökérelem
 - A leírás egészére vonatkozó paraméterek
- `<class>` perzisztens osztály
 - 21 paraméter – name, table,where
- `<id>` azonosító (ne kompozit !)
 - name, type, column (natural-id külön)
 - `<generator>` opció, Java class
 - 12 generátor algoritmus: native, sequence, foreign

Leképezések / tag-ek 2

- `<property>` Java stílusban
 - ☐ name, type, column - 11 paraméter
- `<properties>` property-k csoportja
- `<many-to-one>` reláció
 - ☐ name, type, column - 14 paraméter
- `<one-to-one>` reláció
 - ☐ name, class – 10 paraméter

Leképezések / tag-ek 3

- `<component>` komponens
- `<join>` egy osztály több táblába

Leképezések / típusok 1

■ Entitás

- ☐ egyed, **függetlenül**, hogy van-e rá referencia
- ☐ explicit kell menteni és törölni (cascade !)
- ☐ entitás referenciákból és Value-kből áll

■ Value

- ☐ primitív, kollekció, komponens, spec. obj.
- ☐ komponens: saját osztály + value szemantika
- ☐ mentés, törlés a tulajdonossal együtt

Leképezések / típusok 2

■ Java – Hibernate leképezés

Entitás →

`<class>`

Value →

`<component>`

`<property>`

Leképezések / típusok 3

- Java – tábla oszlop
- Standard mapping ➡
- User defined mapping

Java	SQL
int	INTEGER
String	VARCHAR

- `org.hibernate.usertype` implementálása
- példa: `dupla_string`

```
<property name="twoStrings"  
    type="org.hibernate.test.DoubleStringType">  
    <column name="first_string"/>  
    <column name="second_string"/>  
</property>
```

Leképezések / kollekciók 1

- Kollekciók **interface** típusúak

```
public class Product {  
    private String serialNumber;  
    private Set parts = new HashSet();  
    ...  
}
```

- Hibernate-nek saját implementációja van.
- Tilos a konkrét típusra cast-olni !
- Kollekció entitás komponense. Magában nem áll.

Leképezések / kollekciók 2

■ Mapping

```
<class name="Product">  
  <id name="serialNumber" column="SN"/>  
  <set name="parts">  
    <key column="SN" not-null="true"/>  
    <one-to-many class="Part"/>  
  </set>  
</class>
```

<set>, <list>, <map>, <bag>, <array>, <p>-array>

Leképezések / kollekciók 3

■ Kollektciók elemei

□ Value-k

- `<element>` Or `<composite-element>`

□ entitások referenciái

- `<one-to-many>` Or `<many-to-many>`

■ Példák

Leképezések / kollekciók 4

■ String-ek halmaza

```
<set name="names" table="person_names">  
  <key column="person_id"/>  
  <element column="person_name" type="string"/>  
</set>
```

■ Integerek bag-je

```
<bag name="sizes" table="item_sizes"  
      order-by="size asc">  
  <key column="item_id"/>  
  <element column="size" type="integer"/>  
</bag>
```

Leképezések / asszociáció 1

■ Rendszerezés

- ☐ egy- vagy kétirányú
- ☐ multiplicitás (1:1, 1:n, n:1, n:m)
- ☐ join táblával, vagy nélküle

■ Példákban SQL típus nincs (pl. `bigint`)

■ A leggyakoribb eset (kétirányú, 1:n) az autós példában

Leképezések / asszociáció 2

■ Példa: egyirányú, n:1

```
create table Person (personId not null primary key,  
                    addressId not null)  
create table Address (addressId not null primary key)
```

Leképezések / asszociáció 3

■ Egyirányú, n:1

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
  <many-to-one name="address",
    column="addressId", not-null="true"/>
</class>

<class name="Address">
  <id name="id" column="addressId">
    <generator class="native"/>
  </id>
</class>
```

Leképezések / asszociáció 4

■ Példa: egyirányú, 1:1

□ idegen kulccsal

```
create table Person (personId not null primary key,  
                    addressId not null unique)  
create table Address (addressId not null primary key)
```

□ közös kulccsal

```
create table Person (personId not null primary key)  
create table Address (personId not null primary key)
```

Leképezések / asszociáció 5

■ Egyirányú, 1:1

```
<class name="Person">
  <id name="id" column="personId">
    <generator class="native"/>
  </id>
</class>

<class name="Address">
  <id name="id" column="personId">
    <generator class="foreign">
      <param name="property"> person </param>
    </generator>
  </id>
  <one-to-one name="person"
               constrained="true"/>
</class>
```

Leképezések / asszociáció 6

■ Példa: egyirányú, 1:n, join table

```
create table Person (personId not null primary key)
create table PersonAddress (personId not null,
                           addressId not null primary key )
create table Address (addressId not null primary key)
```

Leképezések / asszociáció 7

■ Egyirányú, 1:n, join table

```
<class name="Person">
  <id name="id" column="personId"/>
  <set name="addresses" table="PersonAddress">
    <key column="personId"/>
    <many-to-many column="addressId"
      unique="true" class="Address"/>
  </set>
</class>

<class name="Address">
  <id name="id" column="addressId"/>
</class>
```

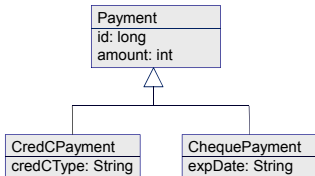
Leképezések / komponens

■ Érték, nem entitás referencia

```
public class Person {  
    private Name name;...}
```

```
<class name= ....  
    <component name="Name" class="Name">  
        <property name="first"/>  
        <property name="last"/>  
    </component>  
</class>
```

Leképezések / öröklés 1

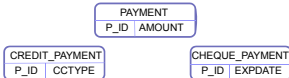


Leképezések / öröklés 2

■ table-per-class-hierarchy

PAYMENT				
P_ID	PAYMT_TYPE	AMOUNT	CCTYPE	EXPDATE

■ table-per-subclass



■ table-per-concrete-class



Leképezések / öröklés 3

■ table-per-class-hierarchy

```
<class name="Payment" table="PAYMENT">
  <id name="id" type="long" column="P_ID"/>
  <discriminator column="PAYMT_TYPE" type="string"/>
  <property name="amount" column="AMOUNT"/>
  ...
  <subclass name="CredCPayment" discriminator-value="CRDT">
    <property name="credCType" column="CCTYPE"/>
    ...
  </subclass>
  <subclass name="ChequePayment" discriminator-value="CHEQ">
    <property name="expDate" column="EXPDATE"/>
    ...
  </subclass>
</class>
```

Leképezések / öröklés 4

■ table-per-subclass

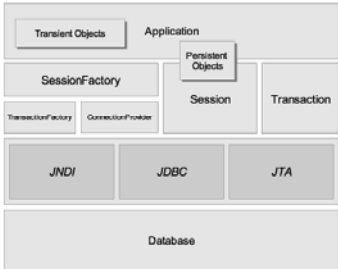
```
<class name="Payment" table="PAYMENT">
  <id name="id" type="long" column="P_ID"/>
  <property name="amount" column="AMOUNT"/>
  ...
  <joined-subclass name="CredCPayment"
                    table="CREDIT_PAYMENT">
    <key column="P_ID"/>
    <property name="credCType" column="CCTYPE"/>
    ...
  </joined-subclass>
  <joined-subclass name="ChequePayment"
                    table="CHEQUE_PAYMENT">
    <key column="P_ID"/>
    <property name="expDate" column="EXPDATE"/>
    ...
  </joined-subclass>
</class>
```

Leképezések / öröklés 5

■ table-per-concrete-class

```
<class name="Payment">
  <id name="id" type="long" column="PAYMENT_ID">
    <generator class="sequence"/> </id>
  <property name="amount" column="AMOUNT"/>
  ...
  <union-subclass name="CredCPayment"
    table="CREDIT_PAYMENT">
    <property name="credCType" column="CCTYPE"/>
    ...
  </union-subclass>
  <union-subclass name="ChequePayment"
    table="CHEQUE_PAYMENT">
    <property name="expDate" column="EXPDATE"/>
    ...
  </union-subclass>
</class>
```

Működés / Architektúra



Architektúra

- SessionFactory szálbiztos, DB
- Session egyszálú, rövid életű
- Pers. Obj – rövid életű, egy Session-höz
- Tranzakció – atomicitás
- ConnectionProvider – opció
- TransactionFactory - opció

Objektum állapotok

- Session-höz (*persistence context*) képest
- Tranziens
 - **még nem** kapcsolódott (nincs id-je)
- Perzisztens
 - Az adatbázis tábláival összekötve
- Lekapcsolt (detached)
 - **már nem** kapcsolódik

Konfigurálás

- leképezés a Java osztályok és a DB között
- **Configuration** – nem változtatható
 - Paraméterek, erőforrások megadása
 - kódban, fájl, paraméter, property
- **SessionFactory** – nem változtatható
- **Session**
 - JDBC kapcsolat, JNDI elérés
- Egyéb property-k

Azonosság (identitás)

- DB `foo.getId().equals(bar.getId())`
- JVM `foo == bar`
- Session-on belül Hibernate biztosítja, közöttük problémák lehetnek
- Megoldás:
 - `equals()` és `hash()` felüldefiniálása
 - business key equality

Azonosság (identitás)

```
public class X {  
    public boolean equals(Object other) {  
        if (this == other) return true;  
        if (!(other instanceof X)) return false;  
  
        final X x = (X)other;  
        if (!x.getY().equals(getY())) return false;  
        return true;  
    }  
    public int hashCode() {  
        int result;  
        result = getY().hashCode();  
        result = 29 * result + getId();  
        return result;  
    }  
}
```

Objektumok kezelése 1

■ új perzisztens objektum

```
X x = new X(...);  
x.setY = ...;  
session.save(x); OR session.persist(x);
```

■ betöltés (ID-t ismerni kell !)

```
X x = (X)session.load(X.class, ID); OR  
session.load(x, ID); OR  
X x = (X)session.get(X.class, ID);
```

□ load – exception, get – null pointer

Objektumok kezelése 2

■ Query visszatér

```
X x = (X)session.createQuery(..).uniqueResult();  
List l = session.createQuery(..).list();  
Iterator i = session.createQuery(..).iterate();
```

■ Mi van a listában ?

- ☐ objektum
- ☐ tuple – többes (tömbként)
- ☐ skalár – attribútumok, pl. `count()`

Objektumok kezelése 3

- Perzisztens objektumok módosítása
 - táblába felír: `session.flush()`
- Lekapcsolt objektumok módosítása
- Hogyan kapcsoljuk rá egy új session-re ?
 - `session.update(x)` - nem reloaded
 - `session.saveOrUpdate(x)`
 - `session.merge(x)`
- Objektum törlése
 - `session.delete(x)`

Objektumok kezelése 4

■ Replikálás

```
Session session = sessionFactory1.openSession();
Transaction tx = session.beginTransaction();
Item item = (Item) session.get(Item.class, new
                                     Long(1234));

tx.commit();
session.close();
```

```
Session session2 = sessionFactory2.openSession();
Transaction tx2 = session2.beginTransaction();
session2.replicate(item,
                   ReplicationMode.LATEST_VERSION);
tx2.commit();      (IGNORE, OVERWRITE, EXCEPTION)
session2.close();
```

Tranzakciók 1

- DB elérés csak tranzakción belül
- session – DB-objektum kapcsolat
- *session-per-request* policy az ajánlott
- session nem szálbiztos – szinkronizálni
- memória objektumokra nincs zár



Tranzakciók 2

■ Optimista zárolás

☐ Alkalmazás szinten

- Új session, új verzió – új vs. régi ?

☐ Hosszú session, automata verzió

☐ Lekapcsolt objektumok, automata verzió

■ Pesszimista zárolás - haladóknak

Lekérdezések 1

■ Query paraméterezése

□ név

```
Query q = s.createQuery("      :x ");  
q.setString("x", "param");
```

□ pozíció

```
Query q = s.createQuery("      ?  ? ");  
q.setString(0, "param0");  
q.setString(1, "param1");
```

Lekérdezések 2

■ Query paraméterezése

□ paraméterlista

```
List names = new ArrayList();  
names.add("param1");  
names.add("param2");  
Query q = s.createQuery(".. in (:namesList)");  
q.setParameterList("namesList", names);
```

HQL 1

- Hibernate Query Language
- From

```
from Department as dept  
from Department, Employee
```

- Join
 - ☐ inner
 - ☐ left outer
 - ☐ right outer
 - ☐ full outer

HQL 2

- Select

- ☐ objektumok és property-k
- ☐ tuple-k és listák

- Aggregáló funkciók

- ☐ avg(), sum(), min(), max(), count()

- Where – kifejezések

- order by, group by

Criteria Query

```
Criteria crit = s.createCriteria(X.class);  
crit.setMaxResults(50);  
List x1 = crit.list();
```

```
List x2 = crit  
    .add(Restrictions.like("name", "Z"))  
    .add(Restrictions.between("y", minY, maxY))  
    .list();
```



Java Persistence API

Java Persistence API (JPA)

- JSR 220: EJB 3.0

- <http://jcp.org/en/jsr/detail?id=220>

- Cél: EJB egyszerűsítése

- Források:

- EclipseLink (Toplink)

- Hibernate

- Lényeg

- POJO technológia, Java EE, SE támogatás

Entitások

- “An entity is a lightweight persistent domain object.” (JSR 220)
- POJO – serializálható (más nem kell)
 - öröklés, polimorfizmus - OK
- Szabvány O-R kapcsolat
- Lekérdezhetőség
- Entity Manager (EM) a felügyelő
 - rajta keresztül érhetők el a szolgáltatások

Entity Manager

- Persistence context (PC): A perzisztens objektumok futási környezete. Objektum állapotok benne értelmezhetőek.
- EM jeleníti meg
- Session (hibernate) == EM (JPA) ???
- PC élettartam
 - konténer vagy az alkalmazás menedzseli
 - ☐ tranzakció alapú
 - ☐ kiterjesztett

Persistence Unit

- telepítő csomag
- O-R leképezés
- Leképezések, relációk hatásköre
- Java annotáció és/vagy XML
- standard könyvtár szerkezet
- Persistence provider konfigurálása:
 - persistence.xml

A feladat megoldása

- EclipseLink

<http://www.eclipse.org/eclipselink/>

- is an advanced object-persistence and object-transformation framework that provides development tools and run-time capabilities that reduce development and maintenance efforts and increase enterprise application functionality.

A feladat megoldása

■ EclipseLink

□ Integrálható

- Oracle WebLogic Server
- Glassfish
- JBoss
- IBM WebSphere application server
- SAP NetWeaver
- Oracle OC4J
- various web containers (Apache Tomcat, IBM WebSphere CE, SpringSource tcServer)

A feladat megoldása

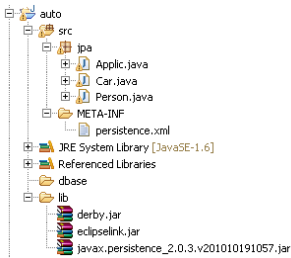
- Derby adatbáziskezelő

<http://db.apache.org/derby/>

- ☐ nyílt forráskódú
- ☐ relációs adatbáziskezelő
- ☐ tiszta Java
- ☐ Apache License, Version 2.0
- ☐ csak 2.6 MB
- ☐ beágyazott és kliens/szerver mód
- ☐ egyszerű install

A feladat megoldása

■ Eclipse struktúra



JPA részletesebben

- Entitások
- Relációk
- Lekérdezések

Entitások

- @Entity annotáció (vagy XML)
- public vagy protected no-arg konstruktor
- final kizárva (osztály, metódus, változó)
- Serializable-t implementálni (érték pm)
- Örökölhet entitástól vagy POJO-tól
- POJO örökölhet entitástól
- Konténeren kívül is használhatóak

Entitások

```
@Entity
public class MyClass implements Serializable
{
    @Basic
    Date birthday;
}
```

```
<entity name="mydomain.MyClass">
    <attributes>
        <basic name="birthday"/>
    </attributes>
</entity>
```

Perzisztens elemek

- Mező és property alapon is elérhető
 - hierarchián belül egységes

```
@Entity
public class MyClass implements Serializable
{
    @Basic
    Date getBirthday(){...}
    void setBirthday(Date date){...}
}
```

- nem static
- nem @Transient vagy transient

Perzisztens elemek

- Mezők és property-k (@Basic)
 - primitív típusok, String, wrapperek
 - byte[], char[], enum, szerializálhatók
 - Collection, Set, List, Map (generic is)
- Összetett, beágyazott (@Embedded)
- Betöltés: Eager és Lazy
 - predefinit: Eager
 - kivétel: 1:1, 1:N relációk

Kulcs

- Primary Key – kötelező
- Egyszerű kulcs - @Id
 - Primitív, wrapper, String, Date
- Összetett kulcs
 - Primary Key osztály - @IdClass
 - Kulcs elem : @EmbeddedId
- Kulcsgenerálás
 - @GeneratedValue(strategy = GenerationType.X)
 - X = AUTO, SEQUENCE, IDENTITY, TABLE

Entitások élelciklusa

■ Entity Manager

```
EntityManagerFactory factory =  
    Persistence.createEntityManagerFactory(p_u_name);  
EntityManager em = factory.createEntityManager();
```

persistence.xml

```
<persistence-unit name="p_u_name",  
    transaction-type="RESOURCE_LOCAL">  
    <class>Car</class>  
    <properties>  
        <property ...="...."/>  
    </properties>  
</persistence-unit>
```

Entitások élelciklusa

■ Entity Manager műveletei:

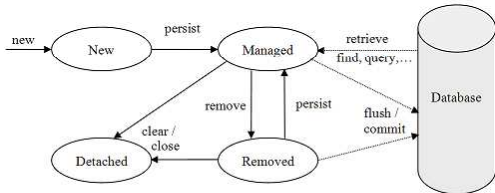
- ☐ persist
- ☐ refresh
- ☐ remove
- ☐ merge

■ Tranzitivitás

- ☐ reláció paramétere
- ☐ cascade = (műveletek) + ALL

Entitások életciklusa

■ Entitás állapotai



Entitások egyéb műveletei

■ Entity Manager egyéb műveletei:

- ☐ find
- ☐ getReference
- ☐ flush
- ☐ clear
- ☐ Query - később

Relációk

- Megegyezik a Hibernate-tel
- Annotáció
 - @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
- Egy- és kétirányú
- Kétirányúnál
 - tulajdonos oldal – idegen kulcs
 - inverz oldal – referencia (mappedBy =)

Relációk

```
@Entity
public class Car implements Serializable {
    @ManyToOne
    private Person owner;
```

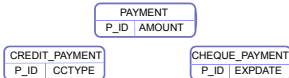
```
@Entity
public class Person implements Serializable {
    @OneToMany(mappedBy = "owner")
    private Collection<Car> cars = new HashSet();
```

Öröklés - Hibernate

■ table-per-class-hierarchy

PAYMENT				
P_ID	PAYMT_TYPE	AMOUNT	CCTYPE	EXPDATE

■ table-per-subclass



■ table-per-concrete-class



Lekérdezések

- Java Persistence Query Language (JPQL)
- Statikus lekérdezések
 - @NamedQuery, @NamedNativeQuery

```
@Entity
@NamedQuery(name = "carByplate", query = "SELECT
    c FROM Car c WHERE c.plate = :rsz")

Car c = (Car) em.createNamedQuery("carByplate").
    setParameter("rsz", plate).
    getSingleResult();
```

Lekérdezések

■ Dinamikus lekérdezések

- run time string komponálás,
paraméterezéssel

```
Query query = em.createQuery  
    ("SELECT p FROM Product p WHERE p.param2 <  
    :threshold ORDER BY p.param1 ascending");  
  
query.setParameter("threshold", my_threshold);  
  
List results = query.getResultList();  
..  
try {Object o = q1.getSingleResult();}  
catch (NoResultException e){....}
```

Lekérdezések

■ Natív SQL lekérdezés

- A konkrét DB nyelve szerint
- számozott paraméterezéssel

```
Query query = em.createNativeQuery("SELECT * FROM  
Product p WHERE p.param2 < ?1");  
  
query.setParameter(1, my_threshold).  
setMaxResults(10);  
List results = query.getResultList();
```

Lekérdezések

■ Criteria Query - Hibernate

- Builder minta szerint futás közben

```
CriteriaQuery cq1 = em.getCriteriaBuilder().  
                        createQuery();  
cq1.select(cq1.from(Person.class));  
  
Iterator iter = ((List<Person>)  
                em.createQuery(cq1).getResultList()).  
                iterator();
```

Lekérdezések

■ SQL – Többszörös SELECT

```
Query q = em.createQuery("SELECT p.name, SUM(c.km)  
FROM Car c, Person p  
WHERE c.owner=p GROUP BY p.name");
```

```
List<Object[]> result = q.getResultList();
```

```
for (Object[] resultElement : result) {  
    String namee = (String) resultElement[0];  
    Long sumkm = (Long) resultElement[1];  
    System.out.println(namee + " " + sumkm); }
```



OO adatbázisok

Objectstore – PSEPro 1

- Java objektumok perzisztenciája
- 50 MB adatbázis kezelés
- egy-felhasználós alkalmazásokhoz
- konkurens session-ök
- az alkalmazáson belül fut, nincs külső adatbázis
- személygyűjtés megoldva

Objectstore – PSEPro 2

- query
- kollekciók kezelése
- adatbázis helyreállítás – recovery
- API a kezelésre
 - adatbázis műveletek
 - session-ok, tranzakciók
 - root-ok kezelése (name service)
 - objektumok írása, olvasása

Alapfogalmak 1

■ Session

- ☐ Hozzáférés az API-hoz
- ☐ Környezet az adatbázis eléréséhez
- ☐ aktív – létrehozott és nem törölt
- ☐ Konkurencia
 - JVM-en belül
 - JVM-ek között

Alapfogalmak 2

- Perzisztencia-képes (persistence capable)
 - adatbázisban tárolható
 - annotáció – posztprocesszálassal
 - nem öröklődik
- Perzisztens objektum
 - Az adatbázisban tárolt objektum
 - Állapotai:
 - üres (hollow)
 - tok, amibe beolvasható az adatbázisból az objektum
 - lusta betöltés

Alapfogalmak 3

■ Perzisztens objektum

□ Állapotai:

■ aktív

- Az adatbázisban levő objektum másolata induláskor
- tiszta vagy koszos (clean or dirty)

■ lejárt (stale)

- elveszti a kapcsolatot az adatbázissal

■ Perzisztencia-tudatos (persistence aware)

- egy osztály, ha maga nem perzisztencia-képes, de hozzáfér egy perzisztens osztály attribútumához (perzisztens tömb eleméhez)

Alapfogalmak 4

- **Tranziens objektum**
 - nincs az adatbázisban
- **Tranzitív perzisztencia**
 - perzisztens objektum által hivatkozott objektumok szintén perzisztensek
- **Annotáció**
 - Posztprocesszor bytekódot injektál
- **Gyökér (root)**
 - Adatbázisbeli objektum elnevezése
 - Naming service

A feladat megoldása

■ Perzisztens osztályok kiegészítése

- perzisztens kollekciók
- importálás

■ Alkalmazás átalakítása

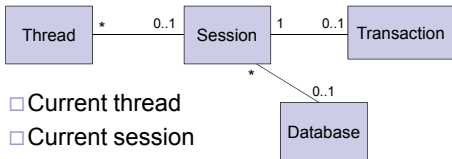
- adatbázis adminisztráció
 - adatbázis, session, tranzakció
- gyökerek, kollekciók
- query-k, paraméterek

PSEPro részletek

- Szálak és session-ök
- Adatbázisok kezelése
- Tranzakciók
- Objektumok használata
- Kollekción
- Lekérdezések

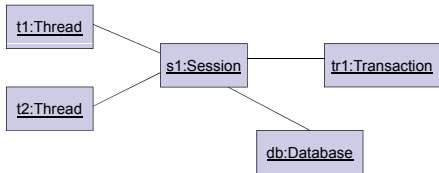
Szálak és session-ök 1

■ Környezet és API



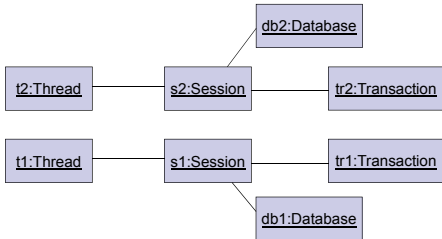
Szálak és session-ök 2

- Szálak osztoznak a session-ön



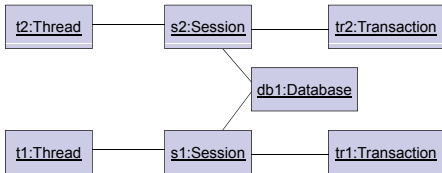
Szálak és session-ök 3

■ Konkurens szálak – különböző adatbázis



Szálak és session-ök 4

- Konkurens szálak – ugyanaz az adatbázis



Szálak és session-ök 5

- session létrehozása

```
public static Session create(String host,  
                             java.util.Properties props)
```

- szálak kapcsolódása

```
public void join()  
public static void leave()
```

- session-höz kapcsolódó objektumok léteznek függetlenül a szálaktól
- szál bármikor jöhet-mehet

Szálak és session-ök 6

- egyidőben egy tranzakció egy sessionhoz
- egyidőben egy adatbázishoz kapcsolódhat
 - tetszőleges számú *read-only* tranzakciót végrehajtó session
 - egy *update* tranzakciót végző session

```
public boolean inTransaction() //Session
public Transaction currentTransaction() //Session
public Session getSession() // Transaction
```

Szálak és session-ök 7

- Session terminálás

```
public void terminate() //Session
```

- A session-höz továbbra is kapcsolódnak az API és perzisztens objektumok

- Aktív-e a session ?

```
public boolean isActive() //szálból
```

Adatbázisok 1

■ Létrehozás

- ☐ előfeltétel: aktív session, tranzakcióban

```
public static Database create(String name,  
                             int fileMod)
```

- ☐ name = XXX.odb
- ☐ fileMod = oprendszer-függő
 - ObjectStore.ALL_READ, ObjectStore.ALL_WRITE
 - ObjectStore.OWNER_READ,
- ☐ DatabaseAlreadyExistsException

Adatbázisok 2

■ Megnyitás

- ☐ nem számít a tranzakció, lehet már nyitott

```
public static Database open(String name,  
                           int openMode)
```

- ☐ openMod =
 - ObjectStore.UPDATE, ObjectStore.READONLY
- ☐ Többszörös nyitás – ugyanaz jön vissza
- ☐ XXX.**odx** – az aktuális directoryban - lock

Adatbázisok 3

■ Zárás

- ☐ nincs tranzakcióban, adatbázis nyitva

```
public void close() // stale objektumok vagy  
public void close(boolean RetainAsTransient)  
Session.terminate() vagy ObjectStore.shutdown()
```

- ☐ zárás után is megmarad
 - ismételt nyitáskor ugyanazt kapjuk vissza

Adatbázisok 4

■ Szemétgyűjtés – JVM-től független

- ☐ adatbázis zárva

```
public Properties GC() // database
```

- ☐ konkurens session-t megvárja, lock update-re

■ Törlés

- ☐ db update-re nyitva, nincs tranzakció

```
db.destroy() // pers. obj - stale
```

- ☐ fájlok törlése kívül

Adatbázisok 5

■ Séma evolúció (verziózás)

- Perzisztens objektum módosítása

- Mikor KELL ?

- perzisztens attribútum felvétele, törlése
- perzisztens attribútum típusváltás
- perzisztens attribútumok sorrendjének változása

Tranzakciók 1

- Konzisztens, biztonságos programrészlet
- Start
 - nem előírt a nyitott adatbázis

```
public static Transaction begin(int type)
```

- type =
 - ObjectStore.UPDATE_ *NON_BLOCKING*
 - ObjectStore.READONLY_ *NON_BLOCKING*
- lock-ok

Tranzakciók 2

- Zárás
- Update-re commit, RO-ra abort is jó
 - Commit
 - mentés, változások tárolása
 - tranzitív perzisztencia
 - perzisztens objektumok állapota
 - lock-ok felszabadítása

Tranzakciók 3

■ Zárás

☐ Commit

```
public void commit(int retain)
```

☐ retain: ObjectStore.

- _STALE
- _HOLLOW
- _READONLY
- _UPDATE
- _TRANSIENT

Tranzakciók 4

■ Zárás

- ☐ Abort – rollback CSAK az adatbázisban

```
public void abort(int retain)
```

■ Konkurencia

- ☐ zár kezelés – adatbázis és tranzakció együtt
- ☐ több read-lock, egy update-lock
- ☐ sorban állnak a kérések

Objektumok 1

- Hogyan lesz perzisztens a tranzien ?
 - ☐ alkalmazás root-tá teszi
 - ☐ másik perzisztens referálja
 - ☐ fizikailag felír - `objectStore.migrate()`
- Memóriába betöltés
 - ☐ root-ból vesszük
 - ☐ external referencia
 - ☐ lekérdezésből
 - ☐ fizikai paraméterek alapján olvas

Objektumok 2

■ Root

- create – update tranzakt, adatbázis nyitva

```
public void createRoot(String name, Object object)
```

- *name* – egyedi az adatbázisban
- ua. objektum több *name*-hez is kapcsolódhat
- primitív root

```
db.createRoot("foo", new Integer(5));
```

- retrieve

```
X x = (X)db.getRoot("foo");
```

Objektumok 3

■ Root

- összerendelés változtat – update tranzak.

```
public void setRoot(String name, Object object)
```

- root törlése – update tranzakció

```
db.destroyRoot(String name);
```

 object NEM

- root mögötti objektum törlése

```
Object object = db.getRoot("xxx");  
db.setRoot("xxx", null);  
ObjectStore.destroy(object);
```

Objektumok 4

■ External references

- session, tranzakció nélkül referál
- nem root is hívható

□ létrehozása (session, tranzakció)

- objektumból

```
public ExternalReference(Object obj)
```

- stringből

```
public static ExternalReference fromString(String s)
```

□ referált objektum

```
public Object getObject() //Ext.ref - UAZ. session
```

Kollekciók

<i>Class</i>	<i>Implements</i>
OSHashBag	Collection
OSHashMap	Map
OSHashSet	Set
OSHashtable	None (Hashtable)
OSTreeMapByteArray	Map
OSTreeMapDouble	Map
OSTreeMapFloat	Map
OSTreeMapInteger	Map
OSTreeMapLong	Map
OSTreeMapString	Map
OSTreeSet	Set
OSVector	Collection
OSVectorList	List

Query 1

- *Collection*-t implementáló szerkezeten
- minden elemre kiértékelhető boole értékű
- operandus – joker is
 - literál – Java primitívek
 - name
 - publikus attribútum és metódus (static is !)
 - free variable

```
Query q = new Query(Employee.class, "getSalary()<50000");  
Collection employees =(Collection)db.getRoot("employees");  
Set result = q.select(employees);
```

Query 2

■ Paraméterezés – free variables

```
public Query(Class elType, String queryExpr  
              [,FreeVariables freeVariables])
```

■ példa

```
FreeVariables fv = new FreeVariables();  
fv.put("IS", Integer.TYPE);  
Query q = new Query(Person.class, "salary>=IS", fv);  
FreeVariableBindings fvb = new FreeVariableBindings();  
fvb.put("IS", new Integer(20000));
```

Query 3

■ Végrehajtás – query metódusai

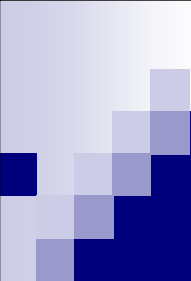
```
public Set select(Collection coll,  
                  [FreeVariableBindings fvb])  
public Object pick(Collection coll,  
                  [FreeVariableBindings fvb])
```



Összefoglalás

Tanulság

- Szerializálás
 - lábbal hajtós
- Relációs adatbázis
 - azért a tábla az ÚR
- JPA
 - a keret takar
- OO adatbázis
 - kompromisszum



Objektumorientált szoftvertervezés

6. OO tervezés

Bevezetés

- Mikor jó egy OO program ?
- Beszélhetünk jó stílusról ?
- Vannak szabályok, formulák, amelyek szerint eljárva jó programot kapunk ?
- Vannak a program jóságának mérőszámai ?
- Mik a jó program jellemzői ?

Tartalomjegyzék

- Tervezési elvek
- Becslés
- CDP minták
- Egyéb minták



Tervezési elvek

Csatolás (coupling)

- Egy szoftver elem változása mekkora hatást okoz másik szoftver elemen ?
- Strukturális és moduláris rendszerekben
 - modulok közötti csatolás – fv hívás
 - dimenziók:
 - paraméterek komplexitása
 - paraméterek száma
 - csatolás ideje
- Laza csatolás - karbantarthatóság

Csatolás (coupling)

- OO világban

- Két osztály (package, object ?) közötti kapcsolat

- Adat-csatolás

- A osztályból elérjük B osztály attribútumát

- Osztály csatolás

- Függőség (dependencia)
 - Specializáció
 - Asszociáció

Csatolás (coupling)

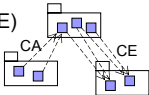
■ Mértékek

- D(ARP)C - Direct (Attribute, Reference, Parameter) Based Coupling
 - azon különböző osztályok száma, amelyeknek (attribútumát, metódusát, paraméterként) érjük el
- DCC - Direct Class Coupling
- CBO – Coupling Between Objects
 - azon különböző osztályok száma, amelyeket elérünk

Csatolás (coupling)

■ Mértékek

- Afferent Coupling (CA): Azon package-en kívüli osztályok száma, amelyek a package-beli osztálytól függenek.
- Efferent Coupling (CE): Azon package-beli osztályok száma, amelyek package-en kívüli osztálytól függenek.
- Instability (RMI): $CE / (CA + CE)$



Osztály csatolás

■ Függőség

- paraméter
- lokális változó
- return value (new is !)

□ Mértékek

- Number of used classes by dependency relation (NUCD)
- Total number of evidences for 'Used classes by dependency relation' (TNUCD)
- Ratio of NUCD to TNUCD (RNUCD)

Osztály csatolás

□ Mértékek

- Number of user classes for a class through dependency relation (NUCC)
- Total number of evidences for 'User classes through dependency relation' (TNUCC)
- Ratio of NUCC to TNUCC (RNUCC)

Osztály csatolás

■ Specializáció

- öröklés
- interfész implementálás

□ Mértékek

- Depth of class in inheritance tree (DIT)
 - root = 0, többszörös = áganként
- Number of children (NSC)

Osztály csatolás

■ Asszociáció

- referencia egy másik osztályra

□ Mértékek

- Number of associated classes with a class (NAC)
- Total associated class Usages (TACU)

Kohézió

- Egy egységbe (modul, osztály, blokk) tartozó elemek közötti kapcsolat erőssége
 - Metóduson belül
 - funkcionális - OK
 - szekvenciális - elfogadott
 - kommunikációs - switch a műveletre
 - procedurális - `instanceOf`
 - temporális
 - logikai
 - esetleges

Kohézió - LCOM

■ Lack of Cohesion Metric (LCOM, IEEE TSE 94)

- Legyen $p(X)$ X tulajdonsága, akkor
- $\sigma(X, Y) = p(X) \cap p(Y)$ a hasonlóság mértéke $p()$ -ben
- Legyen $p()$ az O operáció által használt attribútumok halmaza
- Class C operációi: $O_1, O_2, \dots, O_n$ attribútumai: $a_1, \dots$
- Mindegyik O_i egy A_i attribútumhalmazon operál
- Legyen $P = \{ (A_i, A_j) \mid A_i \cap A_j = \emptyset \}$
 - üres metszet, az operációk nem hasonlítanak
- Let $Q = \{ (A_i, A_j) \mid A_i \cap A_j \neq \emptyset \}$
 - Nem üres metszet, valamekkora hasonlóság
- $LCOM = |P| - |Q|$ if $|P| > |Q|$
 = 0 egyébként

Kohézió - LCOM

■ Példa

□ Class C három operáció: M1, M2, and M3.

□ Példa 1:

- $A1 = \{a, b, c, d, e\}$, $A2 = \{a, b, e\}$, and $A3 = \{x, y, z\}$.
- $A1 \cap A2 \neq \emptyset$, but $A1 \cap A3 = A2 \cap A3 = \emptyset$.
- So $|P| = 2$, and $|Q| = 1$
- $LCOM = 2 - 1 = 1$.

□ Példa 2:

- $A1 = \{a, b, c, d, e\}$, $A2 = \{a, b, e\}$, and $A3 = \{c, d, e\}$.
- $A1 \cap A2 \neq \emptyset$, $A1 \cap A3 \neq \emptyset$, and $A2 \cap A3 \neq \emptyset$.
- $|P| = 0 < |Q| = 3$, $LCOM = 0$.

Kohézió – package szinten

☐ Foundation

- Abstract/Math – Complex, ..
- Physical Unit – date, time, point, line, money, ...
- Structural – stack, list, tree, ...

☐ Architectural

- HCI – Window, CommandButton, TextField, ...
- Database manipulations – Transaction, Session, ...
- Communication – Port, ORB, NameService, ...

☐ Business

- Role – Customer, Patient, Invoice, ...
- Relationship – AccountOwner, PatientSupervisor,

☐ Application

Egyéb nem OO mértékek

□ Cyclomatic Complexity (CC)

- egy metódus bonyolultsága
- a bejárások száma - tesztelés
- $CC = \#edge - \#node + 2$
- alacsony CC a jó

□ Size

- LOC – mindegyik sor,
- NCNB – no comment, no blank lines
- EXEC – végrehajtható utasítások

Egyéb OO mértékek

- Comment Percentage - kommentezettség
 - $CP = \text{total nbr. of comments} / (\text{LOC} - \text{BL})$
- Weighted methods per class (WMC)
 - egy osztályhoz tartozó metódusok CC-je összesen
 - az osztály elkészítésének bonyolultsága ?
- Response for a class (RFC)
 - egy osztályhierarchiában az elérhető metódusok száma
 - csak public vagy mind

Egyéb OO mértékek

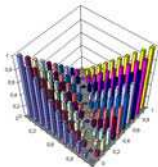
□ Abstractness (RMA)

- Az absztrakt és összes osztályok számának aránya a package-ben

□ Normalized Distance (RMD): $|RMA + RMI - 1|$

□ Különféle elemek száma:

- static attribútumok
- static metódusok
- átadott paraméterek
- blokkok mélysége



OO metrika eszközök

- LocMetrics - C#, C++, Java, SQL

<http://www.locmetrics.com/>

- Eclipse plug-in

<http://metrics.sourceforge.net/update>

- Összefoglaló tábla

http://www.laatuk.com/tools/metric_tools.html

- Kereshető tool-ra

<http://www.stickyminds.com/tools.asp>

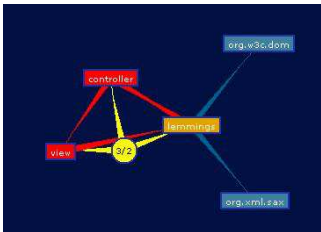
Eclipse - metrics

■ Példa: lemming feladat

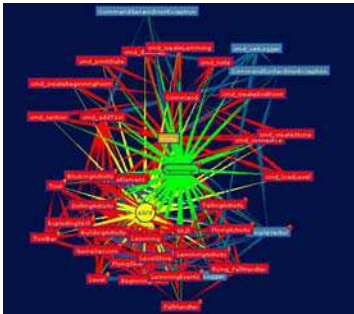
Metric	Total	Mean	Std. Dev.	Maximum	Resource Loading Maximum	Method
Number of Overridden Methods (avg/max per type)	27	0,346	0,748	3	/pelda/lemming/BuildingActivity.java	
Number of Attributes (avg/max per type)	96	1,231	1,908	8	/pelda/lemming/Level.java	
Number of Children (avg/max per type)	30	0,487	1,715	12	/pelda/lemming/Photo.java	
Number of Classes (avg/max per package/fragment)	78	26	23,338	59	/pelda/lemming	
Method Lines of Code (avg/max per method)	2222	4,471	6,919	62	/pelda/lemming/LevelClass.java	createLemmingActivity
Number of Methods (avg/max per type)	472	6,051	7,112	33	/pelda/lemming/GameElement.java	
Nested Block Depth (avg/max per method)		1,223	0,086	6	/pelda/lemming/Stone.java	isL
Depth of Inheritance Tree (avg/max per type)		1,822	0,331	5	/pelda/controller/Graphic.java	
Number of Packages	3					
Alienent Coupling (avg/max per package/fragment)		4	4,243	10	/pelda/lemming	
Number of Interfaces (avg/max per package/fragment)	12	4	5,657	12	/pelda/lemming	
McCabe Cyclomatic Complexity (avg/max per method)		1,531	1,408	15	/pelda/lemming/GameElement.java	checkCollisionWith
Total Lines of Code	3651					
Instability (avg/max per package/fragment)		0,737	0,108	0,899	/pelda/view	
Number of Parameters (avg/max per method)		0,738	0,706	5	/pelda/lemming/StatusInfo.java	StatusInfo
Lack of Cohesion of Methods (avg/max per type)		0,131	0,26	0,621	/pelda/lemming/Level.java	
Efficient Coupling (avg/max per package/fragment)		9,647	7,04	19	/pelda/lemming	
Number of Static Methods (avg/max per type)	25	0,321	1,765	14	/pelda/lemming/Logger.java	
Normalized Distance (avg/max per package/fragment)		0,137	0,142	0,333	/pelda/controller	
Abstractness (avg/max per package/fragment)		0,128	0,11	0,268	/pelda/lemming	
Specialization Index (avg/max per type)		0,107	0,314	2	/pelda/controller/Graphic.java	
Weighted methods per Class (avg/max per type)	761	9,756	13,562	76	/pelda/lemming/GameElement.java	
Number of Static Attributes (avg/max per type)	34	0,438	1,228	7	/pelda/lemming/Photo.java	

Eclipse - metrics

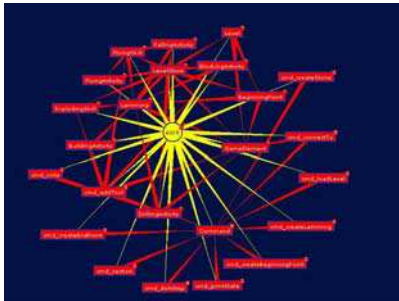
- Példa: lemming feladat – függőségi gráf



Eclipse - metrics



Eclipse - metrics





Becslés

Cocomo 2.0

■ Constructive Cost Model

■ Célkitűzések

- Költség és ütemezés becslése a 200x évek gyakorlatában bevett életciklus modellekre
- A fejlődést követni képes támogató eszközök és adatbázisok kiépítésének háttere
- A fejlesztés érettségének mérésére alkalmas módszerek és eszközök háttere

Cocomo 2.0

■ Célcsoportok

- ☐ Application composition – 1. fázis
 - ☐ Alkalmazás generálás
 - ☐ Rendszerintegráció
 - ☐ Infrastruktúra
- } 3 fázisú modell

Cocomo 2.0 fázisai

■ Fázisok

1. Application Composition – prototípus
 - Object Points –ből indul
2. Early Design – architektúra változatok
 - Function Pointból és SLOC-ból
3. Post-Architecture – megoldás ismert
 - Function Pointból és SLOC-ból

Cocomo 2.0 lépései

1. munka mennyisége

$$\text{Effort} = 2.94 * \text{EAF} * (\text{KSLOC})^E$$

- Effort (PersonMonth, PM)
- EAF = Effort Adjustment Factor
- KSLOC = kilo source LOC
- E = exponens

Cocomo 2.0 lépései

2. fejlesztési idő

$$\text{Duration} = 3,67 * (\text{Effort})^{\text{SE}}$$

- ☐ Duration – (M-ben)
- ☐ Effort (PersonMonth, PM)
- ☐ SE = schedule equation exponent

Cocomo 2.0 tényezők

$$\text{Effort} = 2.94 * \text{EAF} * (\text{KSLOC})^E$$

■ E – skálázási tényezők

- ☐ precedens
- ☐ fejlesztés rugalmassága
- ☐ kockázatkezelés
- ☐ csoport kohézió
- ☐ process érettsége

Cocomo 2.0 hangolás

$$\text{Effort} = 2.94 * \text{EAF} * (\text{KSLOC})^E$$

■ EAF – hangolási tényezők (7, 17)

- ☐ termék
- ☐ platform
- ☐ stáb
- ☐ project

Object points

- Célja: Effort számítása
- Lépései:
 1. Ernyők, listák, 3GL komp. becslése
 2. Osztályozás, súlyozás
 3. Object point számítása – reuse
 4. PROD ráta becslése
 5. PM számítása

Funkciópont elemzés

- FPA - <http://www.ifpug.org/>
- Komponensek:
 - External Inputs
 - External Outputs
 - External Inquiries
 - Internal Logical Files
 - External Interface Files
- +14 Value Adjustment Factor (VAF)

Cocomo 2.0 - eszközök

■ Tool-ok (példák)

■ egyszerű

<http://www.cms4site.ru/utility.php?utility=cocomoi>

■ COCOMO 2000 site

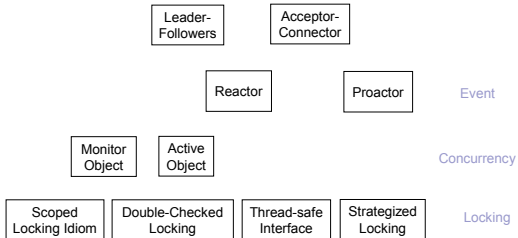
http://sunset.usc.edu/research/COCOMOII/expert_cocomo/expert_cocomo2000.html

■ COSTAR

<http://www.softstarsystems.com/>

CDP minták

CDP - áttekintés



Locking minták

■ Scoped Locking Idiom

- **probléma**: biztonságos lock felszabadítás

```
void sync_oper1(...) {  
    my_lock.acquire();  
    .... // critical section  
    my_lock.release();  
}
```

- **megoldás**: Guard class

```
Thread_Mutex_Guard guard(my_lock);
```

Locking minták

■ Scoped Locking Idiom

```
class Thread_Mutex_Guard {  
    public:  
        Thread_Mutex_Guard(Thread_Mutex &lock)  
            :lock_(lock) {result_ = lock_.acquire();}  
        ~Thread_Mutex_Guard(void); {  
            if (result_ != -1) lock_.release();}  
    private:  
        Thread_Mutex &lock_;  
        int result_;  
};
```

Locking minták

■ Double-Checked Locking

```
if (flag == FALSE){  
    my_lock.acquire();  
    .... // critical section  
    flag = TRUE;  
    my_lock.release();  
}
```

□ **probléma:** Singleton implementációja

Locking minták

■ Double-Checked Locking

```
my_lock.acquire();  
    if (flag == FALSE){  
        .... // critical section  
        flag = TRUE;  
    }  
my_lock.release();
```

□ **probléma:** hatékonyság

Locking minták

■ Double-Checked Locking

```
if (flag == FALSE){  
    my_lock.acquire();  
    if (flag == FALSE){  
        .... // critical section  
        flag = TRUE;  
    }  
    my_lock.release();  
}
```

□ **probléma:** compiler optimalizálás

Locking minták

■ Thread-safe Interface

□ **probléma**: komponensen belülről és kívülről is egyaránt használható legyen. Cél

- holtpont elkerülése
- hatékonyság

Component
sync_oper1()
sync_oper2()

Locking minták

■ Thread-safe Interface

□ **megoldás**: két interfész

- külső interfész ellenőriz
- implementáció privát és elfogad

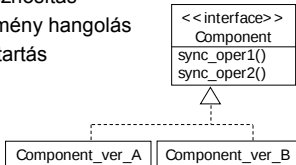
Component
+ sync_oper1() + sync_oper2() - oper1() - oper2()

Locking minták

■ Strategized Locking

□ **probléma**: a lock-olás beégetése a programba akadály:

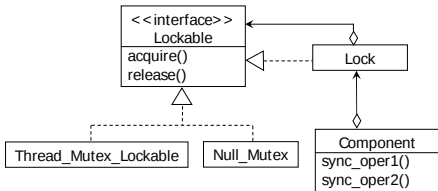
- újrahasznosítás
- teljesítmény hangolás
- karbantartás



Locking minták

■ Strategized Locking

- **megoldás**: run-time delegálható szinkronizációs aspektus



Konkurencia minták

- Monitor Object

- **probléma**: több kliens szál párhuzamosan

- **megoldás**:

- Minden Java object implicit Monitor
 - Synchronized metódus

Konkurencia minták

- Aktív objektum

- **probléma**: konkurrens kérések kezelése

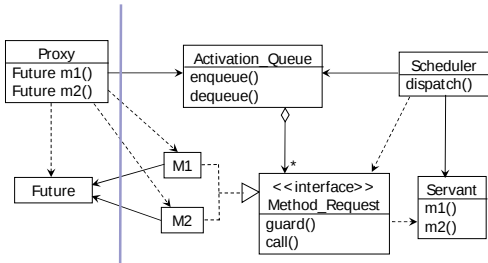
- “guarded” vagy “concurrent” szemantika szerint
 - nem blokkoló metódushívás

- **megoldás**:

- a metódus meghívása és végrehajtásának szétcsatolása
 - végrehajtás saját szálon

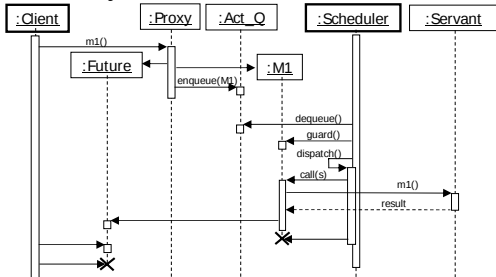
Konkurencia minták

■ Aktív objektum



Konkurencia minták

■ Aktív objektum



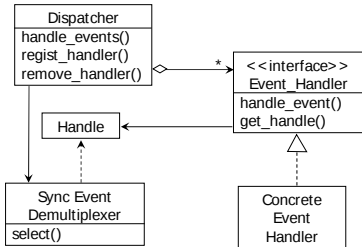
Esemény minták

■ Reactor

- **probléma**: szerver alkalmazás több klienst szolgál ki
- **megoldás**: integrálni
 - a szinkron esemény-demultiplexelést
 - az event handler-ek kezelését

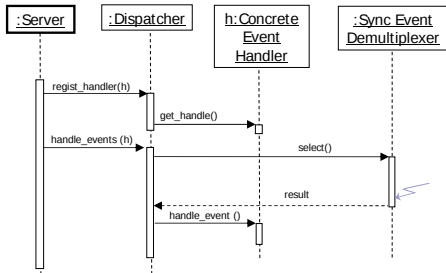
Esemény minták

■ Reactor



Esemény minták

■ Reactor

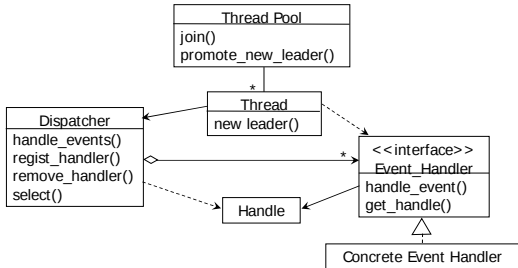


CDP minták

- Vezető-követők (leader-followers)
 - **probléma**: több eseményt konkurens módon kezelő alkalmazás implementálása
 - **megoldás**:
 - szálakból egy csokrot készítünk
 - az eseményt demultiplexeljük
 - szinkron lekezeljük

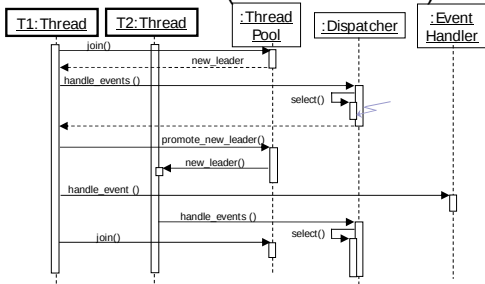
CDP minták

■ Vezető-követők (leader-followers)



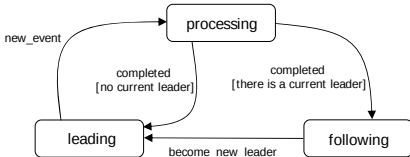
CDP minták

■ Vezető-követők (leader-followers)



CDP minták

■ Vezető-követők (leader-followers)



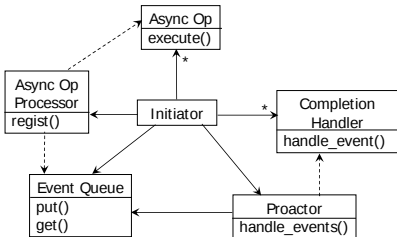
Esemény minták

■ Proactor

- **probléma**: szerver alkalmazás több klienst szolgál ki (mint a Reactor-nál)
- **megoldás**: integrálni
 - az aszinkron esemény-demultiplexelést
 - az event handler-ek kezelését

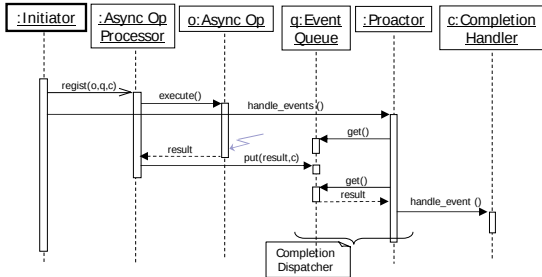
Esemény minták

■ Proactor



Esemény minták

■ Proactor

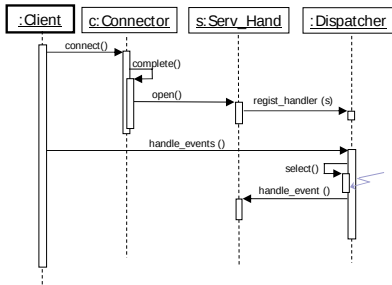


CDP minták

- Csatlakozó (acceptor-connector)
 - **probléma**: összefonódik a kapcsolódó és a kommunikációs szerep
 - **megoldás**: szétválasztani
 - a kapcsolódást és inicializálást
 - a szolgáltatás nyújtását

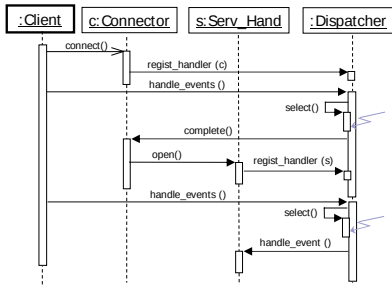
CDP minták

■ Csatlakozó (acceptor-connector(**szinkron**))



CDP minták

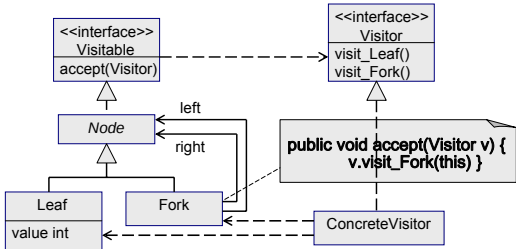
■ Csatlakozó (acceptor-connector(aszinkron))



Egyéb minták

Visitor Combinator

■ Klasszikus visitor



Visitor Combinator

- Klasszikus visitor problémák

- Ki vezérli a bejárást ?

- Külső program ?

- Accept ?

- Visitor ?

- Újrahasználhatóság ?

- Öröklés – specializálás

- Több helyről - többszörös öröklés ????

- Megoldás: Visitor Combinator

Visitor Combinator

■ Identity - minek ????

```
class Identity implements Visitor {  
    public void visit_Leaf(Leaf leaf) {}  
    public void visit_Fork(Fork fork) {}  
}
```

■ AddOne – egy példa

```
class AddOne extends Identity {  
    public void visit_Leaf(Leaf leaf) {  
        leaf.value = leaf.value + 1;  
    }  
}
```

Visitor Combinator

■ Sequence

```
class Sequence implements Visitor {  
    Visitor first;  
    Visitor then;  
    public Sequence(Visitor _first, Visitor _then){  
        first = _first;  
        then = _then; }  
    public void visit_Leaf(Leaf leaf) {  
        leaf.accept(first);  
        leaf.accept(then); }  
    public void visit_Fork(Fork fork) {  
        fork.accept(first);  
        fork.accept(then); }  
}
```

Visitor Combinator

■ Mire jó a Sequence ?

```
class Twice extends Sequence {  
    public Twice(Visitor v){  
        super(v, v); }  
}
```

■ másik példa

```
class AddTwo extends Twice {  
    public AddTwo(){  
        super(new AddOne()); }  
}
```

Visitor Combinator

- Alternatív kombináció
- Hiba a látogatáskor

```
public class VisitFailure extends Exception {}
```

- Fail

```
class Fail implements Visitor {  
    public void visit_Leaf(Leaf l) throws VF {  
        throw new VisitFailure();  
    }  
    public void visit_Fork(Fork f) throws VF {  
        throw new VisitFailure();  
    }  
}
```

Visitor Combinator

■ Choice

```
class Choice implements Visitor {  
    Visitor first;  
    Visitor then;  
    public Choice(Visitor _first, Visitor _then){  
        first = _first;  
        then = _then; }  
    public void visit_Leaf(Leaf leaf) {  
        try {leaf.accept(first);}  
        catch (VF f) {leaf.accept(then); }  
    public void visit_Fork(Fork fork) {  
        try {fork.accept(first);}  
        catch (VF f) {fork.accept(then); }  
    }  
}
```

Visitor Combinator

■ IsZero

```
class IsZero extends Fail {  
    public void visit_Leaf(Leaf _l) throws VF {  
        if (_l.value != 0) {  
            throw new VisitFailure();  
        }  
    }  
}
```

■ Try

```
class Try extends Choice {  
    public Try(Visitor v){  
        super(v, new Identity());  
    }  
}
```

Visitor Combinator

■ Kombinator algebra

$\text{Sequence}(\text{Identity}, v) = v$

$\text{Sequence}(v, \text{Identity}) = v$

$\text{Sequence}(\text{Fail}, v) = \text{Fail}$

$\text{Sequence}(v, \text{Fail}) = \text{Fail}$ (ha v -nek nincs mellékhatása)

$\text{Choice}(\text{Fail}, v) = v$

$\text{Choice}(v, \text{Fail}) = v$

$\text{Choice}(\text{Identity}, v) = \text{Identity}$

$\text{Try}(v) = \text{Choice}(v, \text{Identity})$

$\text{IfZeroAddOne} = \text{Try}(\text{Sequence}(\text{IsZero}, \text{AddOne}))$

Visitor Combinator

■ Bejáró kombinátorok

- mindegyik közvetlen gyermekre

```
class All implements Visitor {  
    Visitor v;  
    public All(Visitor _v){  
        v = _v; }  
    public void visit_Leaf(Leaf leaf) throws VF {  
    }  
    public void visit_Fork(Fork fork) throws VF {  
        fork.left.accept(v);  
        fork.right.accept(v);  
    }  
}
```

Visitor Combinator

■ Bejáró kombinátorok

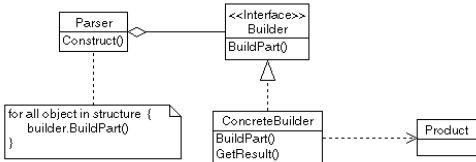
$\text{TopDown}(v) = \text{Sequence}(v, \text{All}(\text{TopDown}(v)))$

```
class TopDown extends Sequence {  
    public TopDown(Visitor v) {  
        super(v, null);  
        then = new All(this); }  
}
```

$\text{BottomUp}(v) = \text{Sequence}(\text{All}(\text{BottomUp}(v)), v)$

```
class BottomUp extends Sequence {  
    public BottomUp(Visitor v) {  
        super(null, v);  
        first = new All(this); }  
}
```

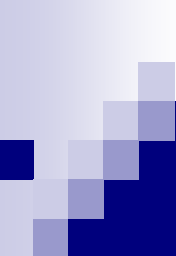
Builder minta



Builder minta

```
public class Complex {
    private final double x;
    private final double y;
    public static class Builder {
        private double x;
        private double y;
        public Builder() {x = 0; y = 0;}
        public Builder x(double xx)
            {x = xx; return this;}
        public Builder y(double yy)
            {y = yy; return this;}
        public Complex build() {return new
                                Complex(this);}
    }
    private Complex(Builder builder)
        {x = builder.x; y = builder.y;}
}

Complex c1 = new
    Complex.Builder().x(3).y(7).build();
```



Objektumorientált szoftvertervezés

Elosztott OO, RMI, CORBA

Hálózati forgalom

- TCP/IP socket használatával
 - bedrótozott protokoll
 - alacsonyszintű adatkezelés
 - lábbalhajtós
 - small footprint
- Keretrendszerrel
 - sok minden transzparens
 - szabványos (?)
 - sok plusz szolgáltatás
 - large footprint

Socketes megoldás (szerver)

```
int main(int argc, char *argv[]) {
    int servSock =
        socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);

    struct sockaddr_in servAddr;

    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family = AF_INET;
    servAddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servAddr.sin_port = htons(5678);

    bind(servSock, (struct sockaddr *) &servAddr,
        sizeof(servAddr));
    listen(servSock, MAXPENDING);
}
```

Socketes megoldás (szerver)

```
struct sockaddr_in clntAddr;  
int clntLen = sizeof(clntAddr);  
int clntSock =  
    accept(servSock, (struct sockaddr *) &clntAddr,  
           &clntLen);  
  
char echoBuffer[RCVBUFSIZE];  
int recvMsgSize =  
    recv(clntSocket, echoBuffer, RCVBUFSIZE, 0);  
printf("%s\n", echoBuffer);  
  
close(clntSocket);  
}
```

Socketes megoldás (kliens)

```
int main(int argc, char *argv[]) {
    int sock =
        socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
    struct sockaddr_in servAddr;
    memset(&servAddr, 0, sizeof(servAddr));
    servAddr.sin_family      = AF_INET;
    servAddr.sin_addr.s_addr = inet_addr(127.0.0.1);
    servAddr.sin_port        = htons(5678);
    connect(sock, (struct sockaddr *) &servAddr,
              sizeof(servAddr));
    send(sock, "hello world", 12, 0);
    close(sock);
    exit(0);
}
```

Adatátvitel – szerializálás

```
typedef struct {  
    int a;  
    char[32] b;  
    double c;  
} str;
```

```
//client  
str s;  
s.a = 13;  
strncpy("hello world", s.b);  
s.c = 3.14;  
char* cp = (char*)&s;  
send(sock, cp, sizeof(s), 0);
```

```
//server  
char[sizeof(str)] cp;  
recv(clSock, cp, sizeof(str), 0);  
str* s = (str*)cp;  
printf("%d %s %lf\n", s->a, s->b, s->c);
```

Adatátvitel – szerializálás

- Mi történik, ha mutatót akarunk átvinni?

```
typedef struct {  
    int a;  
    char* b;  
    double c;  
} str;
```

- Túl alacsonyszintű a hozzáférés
- Mindig ki kell találni a kereket
- Inkább csináljunk egy keretrendszert



Remote Procedure Call

RPC történet

- RPC először 1976-ban
 - RFC 707
- ONC RPC (Sun)
 - Open Network Computing Remote Procedure Call
 - RFC 1831
- NFS (Sun)
 - RFC 1094, 1813, 3530
- DCOM (Microsoft)
- CORBA (OMG)

Függvényhívás

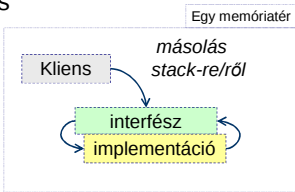
stack	x=1
	y=2
	z=3
	t=6
	a=1
	b=2
	c=3
	r=6

```
int a = 1;  
int b = 2;  
int c = 3;  
int r;  
...  
r = foo(a, b, c);  
...  
printf("%d\n", r);
```

```
int foo(int x, int y, int z){  
    int t = x+y+z;  
    return t;  
}
```

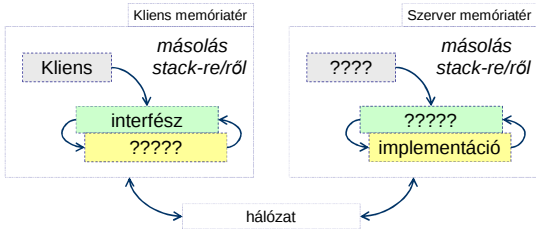
Függvényhívás

■ Lokális



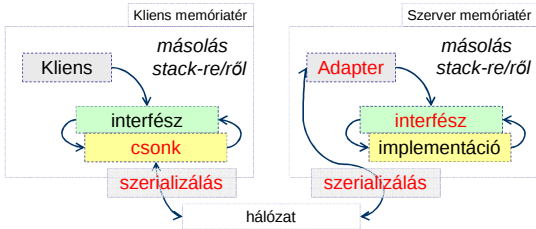
Függvényhívás

■ Távoli



Függvényhívás

■ Távoli



Távoli függvényhívás

■ Csonk (kliens oldal)

```
int foo(int x, int y, int z) {  
    Stub stub = getStub("foo");  
    Stream s = stub.getOutputStream();  
    s.writeInt(x);  
    s.writeInt(y);  
    s.writeInt(z);  
    stub.invoke();  
    s = stub.getInputStream();  
    int t = s.readInt();  
    return t;  
}
```

Távoli függvényhívás

■ Adapter (szerver oldal)

```
void invoke(Skeleton skeleton) {  
    Stream s = skeleton.getInStream();  
    int x = s.readInt();  
    int y = s.readInt();  
    int z = s.readInt();  
    int t2 = foo(x,y,z);  
    s = skeleton.getOutStream();  
    s.writeInt(t2);  
}
```

Új problémák

- memóriakezelés
 - külön címtér
- hiba a hálózaton
 - idempotens műveletnél nincs gond
 - általában:
 - at_most_once
 - at_least_once
 - exactly_once
- hogyan találjuk meg
 - nameservice, trader, stb

Memóriakezelés

- Külön címtér
- Primitív típusok
 - nincs nagy különbség
- Összetett típusok és pointerek
 - deep copy szükséges minden esetben
 - saját foglaló/felszabadító függvények
 - külön érdekesség az *inout* paraméterek kezelése
- Java: elosztott GC

Hálózati hibák

- Csupán jelezzük a hibát
 - *at_most_once* filozófia
 - a kliens eldönti, mi legyen
 - nem transzparens
- Hibajavítás: ismételt hívás
 - *at_least_once*: addig küldjük, amíg jó nem lesz
 - idempotens metódusoknál működik
 - *exactly_once*: igen nehéz implementálni

Szerver megtalálása

■ Bedrótozva

- Lehet protokollfüggő
 - TCP/UDP portok
 - /etc/services
- Ad-Hoc
 - kódba ágyazva

■ Indirekt módon megadva

- fájlrendszer segítségével
- NameService (white pages)
- TradingService (yellow pages)

NameService

- white pages telefonkönyv
- nevekhez a szolgáltató
 - név → telefonszám
- hol van a telefonkönyv
 - tudakozó száma bedrótozva
- hierarchikus névtér elengedhetetlen
- pl. DNS

TradingService

- Mi van ha a szolgáltatás érdekel?
 - ki hangol zongorát a városban?
- Yellow pages
 - szolgáltatások jegyzéke
 - megadhat plusz paramétereket is
 - mennyiért hangol
 - milyen típusú zongorát
 - mennyi időn belül



Remote Method Invocation

Serializálás

- **java.io.Serializable** interfész
- vagy default serializálás
- vagy programozó által megadott

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException

private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;

private void readObjectNoData()
    throws ObjectStreamException;
```

Serializálás 2

- Rekurzív

- attribútumok
 - ha referencia, a referált objektum is

- Megszakítható

- transient
 - az asszociáció végén lévő objektum nem serializálódik

- Verziókezeléssel

- `static final long serialVersionUID = 42L;`

- Osztályattribútumok nem serializálódnak

Serializálás 3

```
FileOutputStream fos = new FileOutputStream("t.tmp");
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeInt(12345);
oos.writeObject("Hello World");
oos.writeObject(new Date());
oos.close();
```

```
FileInputStream fis = new FileInputStream("t.tmp");
ObjectInputStream ois = new ObjectInputStream(fis);
int i = ois.readInt(); // primitív típusra is jó
String hello = (String) ois.readObject();
Date date = (Date) ois.readObject();
ois.close();
```

SecurityManager

- Java-ban kezdettől hangsúlyt fektettek a biztonságra
- A különböző biztonsági szabályok betartásáért a *SecurityManager* felelős
 - `java.lang.SecurityManager`
 - `SecurityManager System.getSecurityManager()`
 - `System.setSecurityManager(SecurityManager sm)`

SecurityManager 2

- **checkXXX()** nevű metódusok segítségével ellenőrizhető a jogosultság
- a metódusok SecurityException-t dobhatnak elutasítás esetén
 - **void checkAccess(Thread t)**
 - *stop, suspend, resume, setPriority, setName, setDaemon*
 - **void checkRead(String file)**
 - **void checkWrite(String file)**
 - **void checkDelete(String file)**
 - fájl olvasása, írása, törlése

SecurityManager 3

- **void checkConnect(String host, int port)**
 - kapcsolat nyitása
- **void checkAccept(String host, int port)**
 - kapcsolat fogadása
- **void checkListen(int port)**
 - kapcsolat várása
- **void checkExit(int status)**
 - **System.exit()** metódus meghívása
- **void checkExec(String cmd)**
 - parancs végrehajtása
- **void checkPermission(java.security.Permission)**
 - általános ellenőrző metódus
- ...

Szabályok beállítása

- *SecurityManager* felüldefiniálása
 - a megfelelő metódus felüldefiniálásával egyedi szabályokat alkothatunk
- Engedélyek (permission) beállítása
 - *policy* fájl beállításával
 - típusok: *File*, *Socket*, *Net*, *Security*, *Runtime*, *Property*, *AWT*, *Reflect*, *Serializable*
 - mindegyikhez egy **XXXPermission** osztály tartozik
 - mindegyiknek saját név-érték párpai vannak a finomhangoláshoz

Policy fájl

■ Feldolgozási sorrend

- `file: java.home/lib/security/java.policy`
- `file: user.home/.java.policy`
- opció: `-Djava.security.policy=someURL`

■ Szerkesztése

☐ kézzel

- könnyű szintaktikai hibát véteni

☐ *policytool* segítségével

- JDK része
- kicsit lábbalhajtós

Policy fájl 2

■ Tartalom

```
grant codeBase "file:/home/sysadmin/" {  
    permission java.io.FilePermission "/tmp/abc", "read";  
};  
grant signedBy "Duke" {  
    permission java.io.FilePermission "/tmp/*",  
    "read,write";  
};
```

■ Részletek

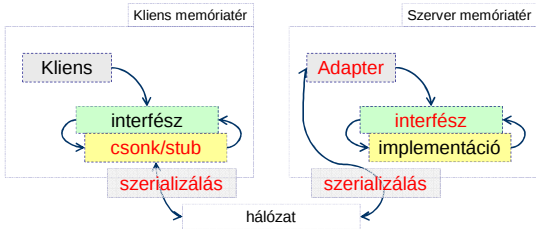
<http://java.sun.com/javase/6/docs/technotes/guides/security/PolicyFiles.html>

Permission objektumok

```
permission java.io.FilePermission "/tmp/*", "read,write";
```

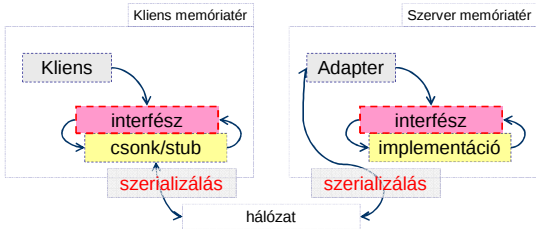
- **String getActions()**
 - "read,write"
- **String getName()**
 - "/tmp/*"
- **boolean implies(Permission permission)**
 - **FilePermission("/tmp/foo", "read");**
→ **boolean**

RMI architektúra



RMI architektúra

■ Távoli objektum interfésze



Távoli objektum interfésze

- **java.rmi.Remote** leszármazottja
 - üres interfész, jelzi, hogy az objektum távoliként elérhető
- minden metódusnak jelezni kell a **java.rmi.RemoteException** dobását
 - vagy az őst, a **java.io.IOException**,
java.lang.Exception kivételek egyikét
- más interfészekről is örökölhet
 - ha betartja a kivételkezelési feltételt

Távoli objektum interfésze 2

- Paraméter és visszatérési érték
 - primitív típus
 - érték szerint, mint lokálisan
 - **Serializable** interfészt megvalósító objektum
 - deep copy
 - szerver oldali módosítása nem hat vissza a hívó oldalra
 - távoli objektum referenciája
 - csak a stub adódik át
 - referenciaszámlálás segíti a GC-t

Távoli objektum interfésze 3

■ Példa:

```
import java.rmi.*;

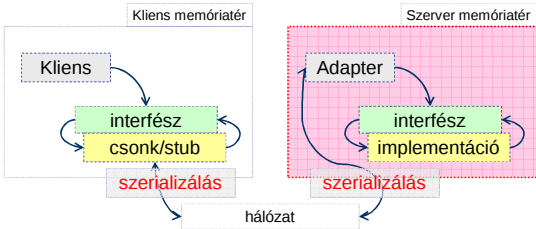
public interface Hello extends Remote {

    void sayHello(String s)
        throws RemoteException;

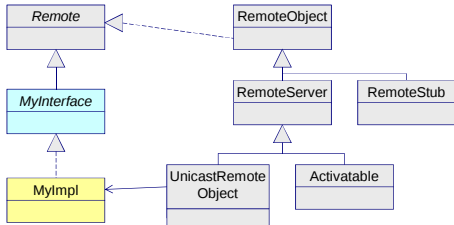
}
```

RMI architektúra

■ Szerver oldal



Szerver oldal



RemoteObject

- Távoli objektumok és csontok (stubok) osztálya
 - **boolean equals(Object o)**
 - összehasonlítás stub-okon is korrekt
 - **RemoteRef getRef()**
 - **int hashCode()**
 - **String toString()**
 - **static Remote toStub(Remote obj)**
 - a távoli objektum stub-ját adja vissza
 - csak akkor hívható, ha a szervant exportálva lett

RemoteServer

- Lehetővé teszi a távoli objektumok létrehozását és exportálását
 - **static String getClientHost()**
 - távoli hívás során visszatér a kliens hoszt azonosítójával
 - **static void setLog(OutputStream os)**
 - az RMI hívások **os**-be lesznek naplózva.
 - **static PrintStream getLog()**
 - visszaadja az RMI napló referenciáját

UnicastRemoteObject

- Távoli objektum exportálása és a hozzá tartozó stub elérése
 - az exportálás teszi lehetővé a távoli objektum távoli elérését
 - a stub küldhető át a kliens oldalra
- Statikus metódusai segítségével exportálhatunk
- Ha szükséges, belőle örököltethetünk
 - ekkor felhasználhatjuk a konstruktorait

UnicastRemoteObject 2

- **Object clone()**
- **static RemoteStub exportObject(Remote r)**
 - exportálja r-t, és visszaad egy rámutató stub-ot
- **static Remote exportObject(Remote r, int port [,RMIClientSocketFactory csf, RMIServerSocketFactory ssf])**
 - mint fent, de a megadott porton fogadja a hívásokat
 - ha kell, speciális socketen keresztül (pl. SSL)
- **static boolean unexportObject(Remote r, boolean force)**
 - siker esetén r nem érhető el távoliként
 - ha force igaz, akkor nem várja meg a futó hívások feldolgozását

UnicastRemoteObject 3

- **protected UnicastRemoteObject(
 int port,
 RMIClientSocketFactory csf,
 RMIServerSocketFactory ssf)**
 - ha meg van adva, adott porton
 - ha meg vannak adva, adott típusú socketekkel
 - automatikusan exportálódik az objektum

Activatable

- Perzisztens elérésű távoli objektumokhoz
 - az objektum stub-jai elérik az objektumot a szerver újraindítása után is
- Szükség esetén aktiválható
- Ha szükséges, értesülhetünk a deaktiválásról is

Remote objektum megvalósítása

■ Delegációval

```
import java.rmi.*;
public class HelloImpl implements Hello {

    public HelloImpl() throws RemoteException {
        super();
    }

    public void sayHello(String s)
        throws RemoteException {
        System.out.println(s);
    }
}
```

Remote objektum exportálása

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

public class Server {
    static public void main(String[] args) {
        try {
            HelloImpl hello_i = new HelloImpl();
            // exportálás, stub létrehozása explicit!
            Hello stub =
                (Hello)UnicastRemoteObject.exportObject(hello_i, 0);
            Registry registry = LocateRegistry.getRegistry();
            registry.rebind("hello", stub);
        } catch (Exception e) {e.printStackTrace();}
    }
}
```

Remote objektum megvalósítása

■ Örökléssel

```
import java.rmi.*;
public class HelloImpl extends UnicastRemoteObject
implements Hello {
    public HelloImpl() throws RemoteException {
        super();
    }

    public void sayHello(String s)
        throws RemoteException {
        System.out.println(s);
    }
}
```

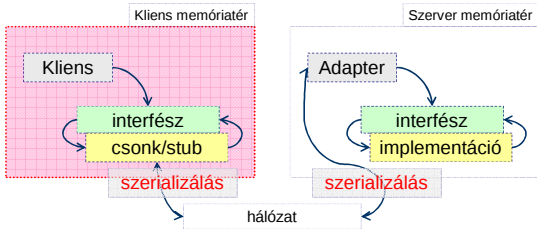
Remote objektum exportálása

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

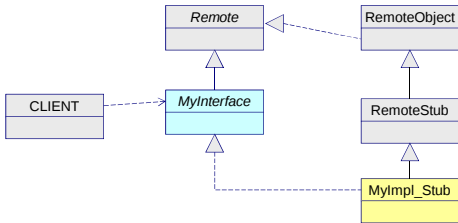
public class Server {
    static public void main(String[] args) {
        try {
            HelloImpl hello_i = new HelloImpl();
            // exportálás megtörtént a konstruktorban
            Registry registry = LocateRegistry.getRegistry();
            // paraméterátadáskor létrejön a stub
            // és csak ő adódik át
            registry.rebind("hello", hello_i);
        } catch (Exception e) {e.printStackTrace();}
    }
}
```

RMI architektúra

■ Kliens oldal



Kliens oldal



Kliens oldal

- A stub referál a távoli objektumra
- A stub maga is egy objektum!
 - az implementációja automatikusan generálódik
 - `MyImpl_Stub`
- A stub megvalósítja a távoli objektum interfészét
 - `MyInterface`
- A stubot többféleképpen meg lehet kapni
 - paraméterként távoli eljárás hívás során
 - visszatérési értéként távoli eljárás hívás során
 - névszolgáltatás segítségével
 - `java.rmi.Naming`

Stub generálása: RMIC

■ Java 1.0-1.1

- generálja a stub és a szkeleton osztályokat
- stub osztály a kliens alkalmazásnál meg kell legyen

■ Java 1.2-1.4

- csak a stub-ot generálja

■ Java 1.5-

- nincs szükség stub generálásra sem, minden automatikus
- stub implementációja (bytecode) a kliens oldalra dinamikusan átkerül

Remote objektum használata

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
public class Client {
    static public void main(String[] args) {
        try {
            Registry registry = LocateRegistry.getRegistry();
            Hello hello = (Hello)registry.lookup("hello");

            hello.sayHello("Hello RMI!");

        } catch (Exception e) {e.printStackTrace();}
    }
}
```

RMIRegistry

- Feladata távoli objektumok névhez kötése
- Név-stub párokat tárol
- *LocateRegistry* osztály
 - ha kell, létrehozza
 - megtalálja adott hoszton, porton, protokollal
- *Registry* interfész
 - biztosítja a szolgáltatást
 - nem perzisztens, nincs hozzáférés-kezelés

RMIRegistry 2

■ *LocateRegistry*

- **static Registry *createRegistry*(int port, RMIClientSocketFactory csf, RMIServerSocketFactory ssf)**

- létrehoz egy registry-t

- **static Registry *getRegistry*(String host, int port, RMIClientSocketFactory csf)**

- megtalál egy registry-t

RMIRegistry 3

■ *Registry*

- **void bind(String name, Remote obj)**
 - beregisztrálja obj-et name-mel
- **String[] list()**
 - kilistázza a regisztrált neveket
- **Remote lookup(String name)**
 - megtalálja obj-et adott néven
- **void rebind(String name, Remote obj)**
 - ha kell, felülírja a korábbi obj-stub-ot
- **void unbind(String name)**
 - törli a nevet a listából

Legfontosabb tervezési minták

■ Callback

- *observer* minta elosztott környezetben
 - a kliens visszahívást vár
- aszinkron hívásoknál majdnem elengedhetetlen

■ Factory

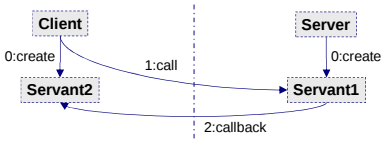
- új szerver oldali objektum létrehozása
- *destroy* metódus kellhet
 - az elosztott GC, *Unreferenced* interfész segít

■ (Mobile) Agent

- a kliensről a szerverre küldött objektum, amelyben adat és futtatható kód együtt kerül át

Callback

- A kliens-szerver szerepek keverednek
 - a kliens szerverként is,
 - a szerver kliensként is viselkedik



Callback példa

■ Egyszerű chat-szerver

- ☐ kliensek regisztrálhatnak
- ☐ üzenetet küldhetnek
- ☐ a regisztráltak értesülnek a többiek üzeneteiről

Callback példa: interfészek

```
public interface Center extends Remote {  
    void regist(String name, Receiver rec)  
        throws RemoteException;  
    void unregist(String name)  
        throws RemoteException;  
    void send(String who, String message)  
        throws RemoteException;  
}
```

```
public interface Receiver extends Remote {  
    void receive(String from, String message)  
        throws RemoteException;  
}
```

Callback példa: *CenterImpl* (S)

```
public class CenterImpl extends UnicastRemoteObject
implements Center {
    private Hashtable<String, Receiver> table;
    public CenterImpl() throws RemoteException {
        super();
        table = new Hashtable<String, Receiver>();
    }
    public void regist(String name, Receiver rec)
        throws RemoteException {
        table.put(name, rec);
    }
    public void unregist(String name)
        throws RemoteException {
        table.remove(name);
    }
    ...
}
```

Callback példa: *CenterImpl* 2

```
...  
public void send(String who, String message)  
    throws RemoteException {  
    System.out.println "["+who+"] "+message);  
    for (Receiver r : table.values()) {  
        r.receive(who, message);  
    }  
}  
}
```

Callback példa: *ReceiverImpl* (C)

```
public class ReceiverImpl
extends UnicastRemoteObject
implements Receiver
{
    public ReceiverImpl() throws RemoteException {
        super();
    }
    public void receive(String from, String message)
        throws RemoteException {
        System.out.println("[ "+from+" ] " + message);
    }
}
```

Callback példa: *Server*

```
public class Server {  
    static public void main(String[] args) {  
  
        try {  
  
            CenterImpl center = new CenterImpl();  
            Registry registry = LocateRegistry.getRegistry();  
            registry.rebind("chat", center);  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Callback példa: *Client*

```
public class Client {  
  
    static public void main(String[] args) {  
  
        try {  
            Registry registry = LocateRegistry.getRegistry();  
            Center center = (Center)registry.lookup("chat");  
  
            ReceiverImpl rec = new ReceiverImpl();  
            center.regist(args[0], rec);  
  
            . . .  
        }  
    }  
}
```

Callback példa: *Client 2*

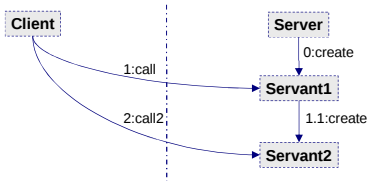
```
...  
  
while (true) {  
    String m = System.console().readLine();  
    if (m.equals("end")) break;  
    center.send(args[0], m);  
}  
  
center.unregister(args[0]);  
  
} catch (Exception e) {  
    e.printStackTrace();  
}  
}  
}
```



Callback példa futtatás

Factory

- A szerver a hívás hatására új szervantot készít
- A kliens az új szervantot használhatja



Factory példa

■ Bankszámla

- ☐ kliens nyithat bankszámlát
- ☐ bankszámla azonosítóval elérhető
- ☐ bankszámlára pénz átutalható
- ☐ bankszámla megszüntethető

Factory példa: interfészek

```
public interface Bank extends Remote {  
    Account openAccount(String anumber)  
        throws RemoteException;  
    Account getAccount(String anumber)  
        throws RemoteException;  
}
```

```
public interface Account extends Remote {  
    void placeAmount(double d) throws RemoteException;  
    boolean transferAmount(double d, Account to)  
        throws RemoteException;  
    double getAmount() throws RemoteException;  
    void close() throws RemoteException;  
}
```

Factory példa: *BankImpl* (S)

```
public class BankImpl extends UnicastRemoteObject
implements Bank {
    Hashtable<String, AccountImpl> table;
    public BankImpl() throws RemoteException {
        table = new Hashtable<String, AccountImpl>();
    }
    public Account openAccount(String anumber)
        throws RemoteException {
        if (!table.containsKey(anumber)) {
            AccountImpl a = new AccountImpl(this, anumber);
            table.put(anumber, a);
            return a;
        } else {return null;}
    }
    ...
}
```

Factory példa: *BankImpl* 2

```
...  
public Account getAccount(String anumber)  
    throws RemoteException {  
    return table.get(anumber);  
}  
public void closeAccount(String anumber) {  
    AccountImpl a = table.get(anumber);  
    try {  
        unexportObject(a, false);  
    } catch (Exception e) {}  
    table.remove(anumber);  
}  
}
```

Factory példa: *AccountImpl* (C)

```
public class AccountImpl extends UnicastRemoteObject
implements Account {
    double amount;
    BankImpl bank;
    final String number;
    public AccountImpl(BankImpl b, String n)
        throws RemoteException {
        super();
        bank = b;
        number = n;
        amount = 0.0;
    }
    ...
}
```

Factory példa: *AccountImpl* 2

```
...  
synchronized public void placeAmount(double d)  
    throws RemoteException {  
    amount += d;  
}  
synchronized public boolean  
transferAmount(double d, Account to)  
    throws RemoteException {  
    if (to == null) return false;  
    amount -= d;  
    to.placeAmount(d);  
    return true;  
}  
...
```

Factory példa: *AccountImpl* 3

```
...  
public double getAmount() throws RemoteException {  
    return amount;  
}  
public void close() throws RemoteException {  
    bank.closeAccount(number);  
}  
}
```

Factory példa: *Server*

```
public class Server {  
    static public void main(String[] args) {  
        try {  
            BankImpl bank = new BankImpl();  
            Registry registry = LocateRegistry.getRegistry();  
            registry.rebind("bank", bank);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Factory példa: *Client*

```
public class Client {  
    public static void main(String[] args) {  
        try {  
            Registry registry = LocateRegistry.getRegistry();  
            Bank bank = (Bank)registry.lookup("bank");  
            Account a1 = bank.openAccount("acc01");  
            Account a2 = bank.openAccount("acc02");  
            a1.placeAmount(10000);  
            a1.transferAmount(3000, a2);  
            System.out.println(a2.getAmount());  
            a1.close();  
            a2.close();  
        } catch (Exception e) {e.printStackTrace();}  
    }  
}
```



Factory példa futtatás

Listaiterátor-minta

- Feladat: egy naplószervertől kérjünk el naplóbejegyzéseket

```
public interface Logger extends Remote {  
    ...  
    Log[] getLogs(Date from, Date to);  
}
```

- Probléma: túl sok adat jöhet válaszul

Listaiterátor-minta

■ Megoldás: iteráljunk (lapozzunk)

```
public interface LogIterator extends Remote {  
    boolean hasMore();  
    Log[] nextN(int n);  
}
```

```
public interface Logger extends Remote {  
    ...  
    LogIterator getLogs(Date from, Date to);  
}
```

Elosztott *garbage collection*

- A távoli objektumokra a kliens oldali stub-oknak is van referenciájuk
- Ha az objektum távoli referenciái megszűntek, az objektumot az RMI *weak reference*-ként tartja számon
- Értesítés a referenciák megszűntéről:
 - `java.rmi.server.Unreferenced` interfész
 - `public void unreferenced()` metódusa

Elosztott *garbage collection*

```
// Client
Account a1 = bank.openAccount("acc01");
...
a1.close(); a1 = null;
```

```
public void close() throws RemoteException {
    bank.closeAccount(number);
}
```

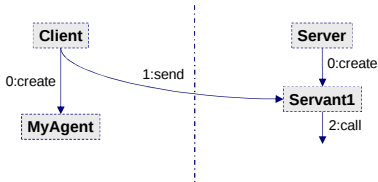
```
public void closeAccount(String anumber) {
    AccountImpl a = table.get(anumber);
    try { unexportObject(a, false); }
    catch (Exception e) {}
    table.remove(anumber);
}
}
```



DGC példa futtatás

Mobile agent

- A kliens által küldött objektum futtatható kódja nincs meg a szerveren
- Adat és kód együtt kerül a szerverhez



Agent példa: interfészek

```
public interface Agent extends Serializable {  
    void run();  
}
```

```
public interface Agency extends Remote {  
    public Agent accept(Agent a)  
        throws RemoteException;  
}
```

Agent példa: *MyAgent* (C)

```
public class MyAgent implements Agent {  
    Integer i;  
    public MyAgent() {i = 1;}  
  
    public void run() {  
        System.out.println(i);  
        i++;  
        System.out.println(i);  
    }  
  
    public void bar() {  
        System.out.println(i);  
    }  
}
```

Agent példa: *MyAgency* (S)

```
public class MyAgency extends UnicastRemoteObject
implements Agency {

    public MyAgency() { super(); }

    public Agent accept(Agent a)
        throws RemoteException {
        System.out.println("Accepting agent");
        a.run(); // lokális hívás!!!
        System.out.println("Done.");
        return a;
    }
}
```

Agent példa: *Server*

```
public class Server {  
    static public void main(String[] args) {  
        if (System.getSecurityManager() == null) {  
            System.setSecurityManager(new SecurityManager());  
        }  
        try {  
            MyAgency mya = new MyAgency();  
            Registry registry = LocateRegistry.getRegistry();  
            registry.rebind("agency", mya);  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Agent példa: *Client*

```
public class Client {  
    static public void main(String[] args) {  
        try {  
            Registry registry =  
                LocateRegistry.getRegistry();  
            Agency agency =  
                (Agency)registry.lookup("agency");  
            MyAgent a = new MyAgent();  
            a = (MyAgent)agency.accept(a);  
            a.bar();  
            a = (MyAgent)agency.accept(a);  
            a.bar();  
        } catch (Exception e) {e.printStackTrace();}  
    }  
}
```



Agent példa futtatás

Agent példa, genericitás

```
public interface Agency extends Remote {  
    public <A extends Agent> A accept(A a)  
        throws RemoteException;  
}
```

```
public class MyAgency implements Agency {  
    public <A extends Agent> A accept(A a)  
        throws RemoteException {  
        System.out.println("Accepting agent");  
        a.run();  
        System.out.println("Done.");  
        return a;  
    }  
    ...  
}
```

Kódmigráció (dynamic class loading)

- Klasszikus példa: *applet*
- RMI hívás során is automatikusan
- Az elérhető bytecode-ok helyét meg kell adni
 - **-Djava.rmi.server.codebase=<URL>**
 - Az URL-nek a kliens számára kell érthetőnek lennie
 - A kliens oldali *ClassLoader* innen tudja, hogy honnan töltsse be a hiányzó osztályokat
- Java 1.5 óta a stub-ok is ezzel a technikával kerülnek át

Kódmigráció 2

- A marshalling során a codebase leírás is átmegy
- Az osztálybetöltés menete
 - lokális osztályok között keresünk
 - ebben a korábban dinamikusan átküldött osztályok is benne vannak!
 - a **java.rmi.server.codebase** property-ben megadott helyeken keres
 - hibát jelez

Kódmigráció: ha hiba történt

■ Szerver oldal, regisztrálás során:

```
java.rmi.ServerException: RemoteException occurred in
server thread; nested exception is:
    java.rmi.UnmarshalException: error unmarshalling
arguments; nested exception is:
        java.lang.ClassNotFoundException: Agency
...
Caused by: java.rmi.UnmarshalException: error
unmarshalling arguments; nested exception is:
    java.lang.ClassNotFoundException: Agency
    at
sun.rmi.registry.RegistryImpl_Skel.dispatch(Unknown
Source)
...
```

Kódmigráció: ha hiba történt

■ Kliens oldal, visszatérési érték

```
java.rmi.UnmarshalException: Return value class not  
found; nested exception is:  
    java.lang.ClassNotFoundException: MyImpl_Stub  
    at  
sun.rmi.registry.RegistryImpl_Stub.lookup(RegistryImpl_Stub.java:109)  
    at java.rmi.Naming.lookup(Naming.java:60)  
    at RmiClient.main(MyClient.java:28)
```

Mitől ügynök az ügynök?

- Aktív és autonóm

- ☐ saját szálon fut
- ☐ a döntéseit maga hozza
- ☐ a környezet figyelembevételével

- Kapcsolatképes (reaktív)

- ☐ más ügynökökkel kommunikálhat

- Tanulékony

- ☐ a tapasztalatait összegzi

- Mobil

- ☐ képes az ügynökségek közötti közlekedésre

Hogyan érkezik az ügynök?

- hogyan inicializálódik
 - **void init()** metódus
 - elég-e a **run()** metódus ehhez?
- hogyan regisztrál
 - ügynökség automatikusan regisztrálja
 - neki kell regisztrálni
- hogyan állítja be a jogosultságokat
 - mit tehet az ügynök
 - mit tehetnek vele mások

Hova menjen az ügynök?

■ Itinerary minta

- az ügynök magával visz egy **Itinerary** objektumot
- megadja az ügynök útvonalát
- **getNextAgency()**
 - referencia
 - név
 - küldheti is az ügynököket

■ On-the-fly döntés

- Itinerary módosításával vagy önállóan

Hogyan távozik az ügynök?

- magától megy tovább
 - egyszerű eset
 - az ügynök felkészülhet a távozásra
- ügynökség (esetleg itinerary) küldi
 - bonyolult eset
 - megszakítható-e a **run()** metódus
 - **finalize()**-szerű jelzés kellhet
 - dönthet-e az ügynök, ha még lenne dolga
 - **goTo(Agency a)** metódus

Kivel kommunikál?

- Ügynökséggel

- ☐ **static Agency Agency.getLocalAgency()**
- ☐ hoszt neve
- ☐ a lokálisan elérhető többi ügynök lekérése
- ☐ proxy regisztrálása

- Más ügynökökkel

- ☐ ügynökségtől kéri el
- ☐ viszi magával a proxy referenciákat

- Fontos a megbízhatóság

- ☐ jogosultságok kezelésével

Hogyan érhető el?

■ Lokálisan

- ügynökségen keresztül
- a fogadott ügynökök láthatják egymást
- fontos a hozzáféréskezelés

■ Távolról

- proxy objektumokat hagy hátra
- ezeknek jelzi, ha mozgott
- a távoli klienseknek nem kell ismerni az aktuális pozícióját

Megoldandó problémák

■ Classloading

- hogyan távolítjuk el a távozó ügynök bytecode-ját?

■ Biztonság

- ügynökség: kinek engedjük meg a belépést
- ügynök: kinek milyen módosításokat engedünk
- rendszer: az ügynök milyen erőforrásokat használhat
- SecurityManager módosításával finomhangolható

■ Hálózati hibák kezelése

- mi lesz az ügynökkel, ha migráció közben leáll a fogadó gép?

Előnyök

- Számítás adatokhoz küldése
 - hálózati forgalom csökken
- Aszinkron végrehajtás
 - pl. GRID rendszerekben
- Dinamikus adaptáció
 - a fogadó rendszer állapotától függő végrehajtás
- Hálózati hibák elviselése
 - szétcsatolt működés, nem kell folyamatos kliens-szerver kapcsolat
- Rugalmas karbantartás
 - a működés módosításához elég az ügynököt átküldeni

Tipikus használat

■ Erőforráskezelés

- rendelkezésreállítás (availability)
- felkutatás (discovery)
- monitorozás (monitoring)

■ Információgyűjtés

- pl. egy kép adatbázisból egy adott feltételnek megfelelő képek kigyűjtése
- cél a hálózati forgalom csökkentése

Tipikus használat 2

■ Hálózati felügyelet

- az erőforráskezelés egy speciális esete
- nem csak a statikus (lokális), hanem a dinamikus (migráció során szerzett) információk feldolgozása

■ Dinamikus szoftvertelepítés

- távoli felügyelethez ideális
- nincs szükség a hoszt gép rendszerének módosítására
- a portabilitás nő, rendszerkövetelmények kevésbé szigorúak

RMI: futás közbeni aktiválás

- Csak akkor példányosodik, ha meghívják
 - a stub-jai léteznek
 - nem foglalja az erőforrásokat
- Perzisztencia-képes
 - két példányosodás közben nem vesznek el az adatai

Futás közbeni aktiválás

- **java.rmi.activation** package
- **Activatable** osztály
 - öröklés
 - **Activatable(ActivationID id, MarshalledObject data)**
 - statikus metódusok használata
 - **exportObject(...)**
- Activation csoport és leíró készítése
- rmid démon

Interfész és implementáció

```
public interface Hello extends Remote {  
    public String sayHello(String s) throws RemoteException;  
}
```

```
public class HelloImpl extends Activatable  
implements Hello {  
    public HelloImpl(ActivationID id,  
        MarshalledObject data) throws RemoteException {  
        super(id, 0);  
        // perzisztencia a data paraméter segítségével  
    }  
    public String sayHello(String s) throws RemoteException {  
        System.out.println(s);  
        return (new java.util.Date())+": "+s;  
    }  
}
```

Szerver oldali exportálás

```
public class Server {  
    public static void main(String[] args)  
        throws Exception {  
  
        if (System.getSecurityManager() == null) {  
            System.  
                setSecurityManager(new SecurityManager());  
        }  
  
        String implClass = "HelloImpl";  
        String implCodebase = "file:/server/";  
        ...  
    }  
}
```

Szerver oldali exportálás

```
...  
Properties props = new Properties();  
props.put("java.security.policy",  
         "file:/server/policy");  
props.put("impl.codebase", implCodebase);  
  
ActivationGroupDesc groupDesc =  
    new ActivationGroupDesc(props, null);  
  
ActivationGroupID groupID = ActivationGroup.  
    getSystem().registerGroup(groupDesc);  
...
```

Szerver oldali exportálás

```
...  
MarshaledObject data = null;  
ActivationDesc desc =  
    new ActivationDesc(groupId, implClass,  
                        implCodebase, data);  
  
Remote stub = Activatable.register(desc);  
  
String name = "hello";  
LocateRegistry.getRegistry().rebind(name, stub);  
}  
}
```

Kliens oldali hívás

```
public class Client {  
    static public void main(String[] args) {  
        try {  
            Registry registry =  
                LocateRegistry.getRegistry();  
            Hello hello = (Hello)registry.lookup("hello");  
  
            String s = hello.sayHello("Hello RMI!");  
            System.out.println(s);  
  
        } catch (Exception e) {  
            e.printStackTrace();  
        }  
    }  
}
```

Aktiváló démon indítása

- A kliens hívásakor

- ha létezik a távoli objektum

- a hívás lezajlik a szokott módon

- ha nem létezik az objektum

- a stub a megadott aktivátorhoz fordul

- amely az objektumot példányosítja és továbbítja a hívást

- A démon indítása

- `rmid -J-Djava.security.policy=policyFile`

- A démonnak már futnia kell az exportáláskor



Aktiválás példa futtatása



CORBA

CORBA alapok

- Nyelvfüggetlen

- C, C++, Java, COBOL, Ada, ...

- Szállítófüggetlen

- IONA, Sun, OpenORB, TAO, ...

- Operációsrendszer-független

- Windows *, Linux, Solaris, ...

CORBA alapok

- Common Object Request Broker Architecture
 - elosztott objektum-orientált architektúra
 - RPC objektumorientált környezetben
 - OMG szabvány, széleskörű támogatás
- Cél: platform- és nyelvfüggetlen kommunikáció
 - szabványos adatátvitel
 - mindenki ugyanazt értsse egy átvitt típusú adaton
 - szabványos végpont-elérés
 - az objektumok megszólítása, referálása azonos legyen
 - szabványos implementációs csatlakozás
 - ha ORB-t váltunk, ne kelljen semmit újraírni

CORBA alapok 2

■ Hálózati protokoll

- GIOP: általános protokoll (pl. double sorosítása)
- IIOP: TCP/IP specialitások kezelése (pl. socket)

■ Interfész leírás

- mindenki számára ugyanazt jelentse → IDL

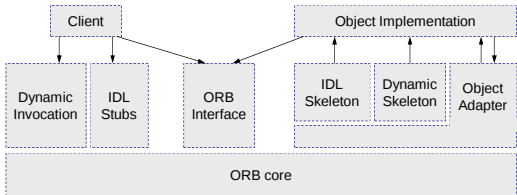
■ Programnyelvi leképezés

- mindenki a saját nyelvén tudjon hozzáférni a többiek objektumaihoz, implementálni sajátot

■ Szolgáltatások

- gyakori problémákra szabványos megoldások

CORBA alapok 3



Interface Description Language

- C++-hoz hasonló nyelv
 - C, C++, Java háttérrel könnyű megtanulni
- IDL-ben adjuk meg
 - az interfészeket
 - a metódusokat
 - az adatstruktúrákat
- Csak leírás, implementáció nem
 - implementációk saját nyelven

IDL példa

```
module Math {  
    struct Complex {  
        double re;  
        double im;  
    };  
    typedef sequence<Complex> complexSeq;  
    interface Calculator {  
        exception DivByZero {};  
        Complex sum(in complexSeq cs);  
        Complex prod(in complexSeq cs);  
        Complex div(in Complex c1, in Complex c2)  
            raises (DivByZero);  
        void normalize(inout Complex c);  
    };  
};
```

IDL jellemzői

- Névterek

- modulok és interfészek, fa szerkezet

- Többféle típus

- **primitív**: char, octet, boolean, short, long, double, ...
 - **összetett**: enum, string, struct, union, sequence, array, valuetype, exception, ...

- 3 féle paraméterátadás

- *in*: csak oda
 - *inout*: oda-vissza
 - *out*: csak vissza

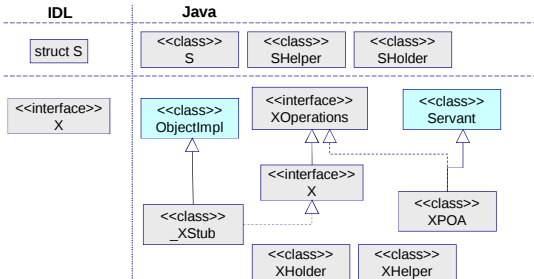
IDL és a natív nyelv

- IDL interfészeket le kell fordítani
 - implementáció natív nyelven
 - tipikusan natív metódusok felüldefiniálásával
 - hívás natív nyelven
 - natív objektumként viselkedő stub-okon
 - adattípusok natív megfelelőivel
 - a kliens és a szerver mind a saját natív típusait használja
 - bizonyos IDL konstrukciók csak körülményesen használhatók
 - pl. Java-ban *enum*, *inout* paraméterek, *union*

IDL és a natív nyelv 2

- Szabványos leképezés több nyelvre
 - Ada, C, C++, COBOL, Java, Lisp, PL/I, Python, Smalltalk
- Nem szabványos leképezés szinte minden nyelvre
- A nyelvfüggetlenség miatt sok megkötés az IDL nevek használatában
 - kis-nagybetűk
 - névtér neve tiltva a névtér egyszeres mélységében

IDL fordítása Java-ra



IDL fordítása Java-ra 2

■ *SHelper*

- segédfüggvények
 - CORBA marshallinghoz
 - általános Any típusba csomagoláshoz

■ *SHolder*

- segédosztály az **inout** és **out** paraméterekhez
- Java-ban csak referencia-átadás, így wrapper kell
 - **value** attribútumban az **S** érték

CORBA példa 1 (interface)

```
// hello.idl
interface Hello {
    string sayHello(in string s1,
                    inout string s2);
};
```

```
// HelloOperations.java
public interface HelloOperations {
    String sayHello (String s1,
                    StringHolder s2);
}
```

CORBA példa 2 (client stub)

```
public interface Hello
extends HelloOperations,
org.omg.CORBA.Object,
org.omg.CORBA.portable.IDLEntity
{} // gyűjtő interfész
```

```
public class _HelloStub
extends ObjectImpl
implements Hello {
    public String sayHello (String s1,
                           StringHolder s2) {
        // marshalling, invoke, unmarshall
    }
    ...
}
```

CORBA példa 3 (server skeleton)

```
public abstract class HelloPOA extends Servant
implements HelloOperations, InvokeHandler
{
    public OutputStream
    _invoke (String method, InputStream in,
            ResponseHandler rh) {
        //delegálja a hívást a leszármazotthoz
    }

    public Hello _this() {...} // aktiválás
    public Hello _this(ORB orb) {...}
    ...
}
```

Szerver oldal

- Az **XPOA** osztálytól kell örököltetni a szervantot
 - az **XOperations**-ben definiált metódusokat kell felüldefiniálni

```
public class MyHello extends HelloPOA{  
    public String sayHello(String s1,  
                           StringHolder s2){  
        System.out.println(s1+s2.value);  
        s2.value = s1+" "+s2.value;  
        return s1+" vissza";  
    }  
}
```

Szerver oldal 2

- A szervantot a **_this()** metódussal lehet a legegyszerűbben aktiválni
 - visszatérési érték a stub
 - ez már átadható CORBA metódushívással

```
...  
    MyHello mh = new MyHello();  
    Hello h = mh._this();  
...
```

POA

- *Portable Object Adapter*
- Célja, hogy a szervant újraírása nélkül ORB implementációt lehessen váltani
 - korábban Basic Object Adapter (BOA)
 - nem szabványos megoldások
 - váltáskor módosítani kellett a szervant kódját
- Feladata
 - szervantok aktiválása
 - explicit és implicit, akár hívás közben
 - kérések szervanthoz juttatása

POA 2

- Szabványos, IDL-ben specifikált metódusok
 - szabványos működés
 - az metódusok szemantikája rögzített
 - policy-kel szabályozható
 - pl. szálkezelés, élettartam, szervantok elérése
- Nyelvi sajátosságok a nyelvi mappingben specifikálva
 - szervantokhoz való csatlakozás, interfészek
 - memóriakezelés, referenciák

Kliens oldal

- Az objektumot a stub-on keresztül érjük el
 - natív hívásokkal
 - natív típusokkal
 - natív kivételekkel
- Nincs távoli referenciaszámlálás
 - a stub megszűnése nem látszik szerver oldalon
 - a szervant megszűnése nem hat a stub-ra

Szabványos szolgáltatások

- Gyakran előforduló problémák
 - pl. üzenetküldés, objektum-keresés
- Szabványos megoldás (OMG)
 - IDL-ben specifikált interfész
 - bárki bármelyik implementációt használhatja
- Normális CORBA szerverként futnak
 - elérésük, használatuk a szabványos CORBA szintaxis és szemantika szerinti

Szabványos szolgáltatások 2

■ Naming Service

- névszolgáltatás: név alapú keresés
- white pages

■ Trading Service

- szolgáltatás típusok alapján keresés
- yellow pages

■ Event és Notification Service

- eseményátadás, szétcsatolt kommunikáció

■ Collection, Concurrency, Time, Transaction stb.



CORBA Naming Service

Naming Service alapok

- Név – elem párok
- Hierarchikus névtér
 - context – „directory”
 - object – „file”
- Összekapcsolás
 - context maga is távoli objektum
 - más naming service alkalmazás belinkelhető



CORBA Trading Service

Trading Service alapok

■ Naming Service

- white pages
- név → referencia

■ Trading Service

- yellow pages
- szolgáltatás → referencia lista

Trading Service fogalmak

- **ServiceType** – szolgáltatás típusa
 - név
 - interfész
 - attribútumok (properties)
- **Offer** – ajánlat
 - konkrét referencia
 - konkrét attribútum-értékek

Trading Service alapfogalmak 2

- Service Type $\leftrightarrow$ SQL tábla
 - kötelező elem a CORBA interfész
- Offer $\leftrightarrow$ SQL tábla egy sora
 - kötelező elem a referencia
- Keresés $\leftrightarrow$ SQL query
 - megadható, hogy egy szolgáltatástípusban milyen konkrét attribútumértékű ajánlatot keresünk
 - *select REFERENCE, ATTRIBUTE*
from SERVICE_TYPE
where EXPRESSION*

Példa szolgáltatás

```
//IDL  
interface PI {  
    string getValue();  
};
```

```
//STDL  
service PiService {  
    interface PI;  
    property short digits;  
};
```

```
//SQL  
create table PiService (  
    _reference PI,  
    digits short  
);
```

Trader Use-case-ei

■ Ajánlatok kezelése

- ☐ publikálás: „*insert*”
- ☐ módosítás: „*update*”
- ☐ törlés: „*delete*”
- ☐ keresés: „*select*”

■ Trader attribútumok állítása

- ☐ számosságok (search, match, return)
- ☐ kérestovábbítás

Összekapcsolt traderek

- Különböző helyen futó traderek összekapcsolhatók
 - federation, link
- Az egyikben kezdeményezett keresés a többibe továbbadódhat
- A kapcsolat egyirányú
- A kapcsolat beállítása lehet
 - **local_only**: csak az aktuális traderben keresünk
 - **if_no_local**: ha az aktuálisban nincs, mehet tovább
 - **always**: minden elérhető traderben keresünk



CORBA Event Service

Bevezetés

- Alap CORBA kommunikáció
 - erős csatolás
 - egy-egy kommunikáció
- Néha nagyobb szétcsatolás kell
 - esemény-alapú kommunikáció
 - több-több kommunikáció
- Ezt támogatja az Event Service

Event Service alapok

■ Szerepek

- **termelő** (supplier) termeli az eseményeket
- **fogyasztó** (consumer) dolgozza fel

■ Kommunikációs modell

- push modell:
 - termelő kezdeményez
 - fogyasztó passzív
- pull modell
 - termelő passzív
 - fogyasztó kezdeményez

Modellek implementálása

■ Push modell

- termelő, ha van új esemény, meghívja a fogyasztó *push* metódusát
- a fogyasztó a *push* metódust implementálja és vár

■ Pull modell

- a fogyasztó, ha eseményt szeretne, meghívja a termelő *pull* metódusát
- termelő implementálja a *pull* metódust és vár

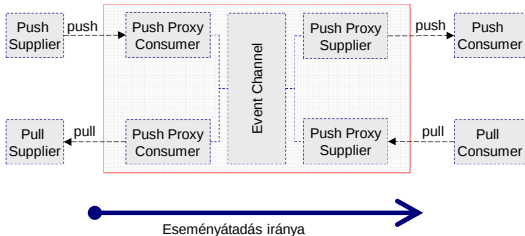
A csatorna

- Az eddigiek nem különböztek az egy-egy alapú CORBA kommunikációtól
 - push-pull eddig is ment
- A szétcsatoláshoz kell egy csatorna
 - többen csatlakozhatnak rá
 - nincs közvetlen kapcsolat termelők és fogyasztók között
 - nincs szükség másik oldali szereplőre
 - ilyenkor nincs hibajelzés
 - plusz szolgáltatásokat nyújthat

A csatorna 2

- A csatlakozáshoz proxy kell
 - interfészt nyújt (push-pull)
 - a termelő felé fogyasztóként
 - a fogyasztó felé termelőként viselkedik
 - tárolja a felgyűlt üzeneteket
 - puffereelés
- A kommunikációs modellek keverhetők

A csatorna modellje



A proxy-k

- A **supplier proxy-k** saját FIFO pufferrel rendelkeznek
 - ha tele van a puffer, az új esemény bekerül, a legrégebbit eldobja
- A **push consumer proxy** a *push* hatására a csatornához továbbítja az eseményt
- A **push supplier proxy** a következő eseményt push hívással adja át a fogyasztónak
- A **pull supplier proxy** a pull hatására vagy ad egy új eseményt a pufferből, vagy blokkol
- A **pull consumer proxy** pull hívással kér új eseményt a termelőtől

Notification Service

■ Plusz szolgáltatások

- csatorna-factory

- események szűrése

- QoS

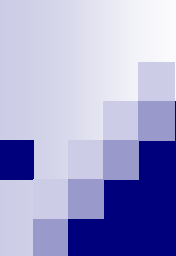
 - garantált átvitel

 - prioritás, időzítés, timeout

 - queue-length, max consumers, stb

- NotifyPublish/Subscribe

 - feliratkozhatunk szolgáltatástípusok adására/vételére



Objektumorientált szoftvertervezés

XML a gyakorlatban

Bevezető

- eXtensible Markup Language
- Cél: szabványos üzenetátadás, tárolás
- HTML sikerén felbuzdulva
- Visszanyúlik a szöveghez
 - vö. `toString()`
- Ma: "Bármilyen kérdés, XML a válasz"

XML történet

■ GML

- generalized markup language
- IBM, '60-as évek
- dokumentumleírás

```
:h1.Chapter 1:  Introduction
:p.GML supported hierarchical containers, such as
:ol
:li.Ordered lists (like this one),
:eol.
as well as simple structures.
:p.Markup minimization allowed the end-tags to be
omitted for the "h1" and "p" elements.
```

XML történet

■ SGML

- standard generalized markup language
- 1986
- lexikonok (Magyar Nagylexikon), szótárak (OED), adatbázisok
- eléggé XML-szerű
- DTD megjelenik

XML történet

■ HTML

- hypertext markup language
- 1991
- weboldalak
 - áttörés
 - előtte gopher, stb
- nem eléggé rugalmas
 - kompatibilitási problémák
 - *"this page is optimized for ObscureBrowser 11.3a"*

XML jellemzői

- Hierarchikus fa struktúra
 - tag-ek, attribútumok és szöveg
- W3C szabvány
 - nyelvtan
 - parse-olási szabályok
 - jólformáltság és validitás
- Meta-struktúra, tetszőleges elemekkel bővíthető

XML jólfomáltság

■ Szintaktikai szabályoknak való megfelelés

- opcionális fejléc

```
<?xml version="1.0" encoding="UTF-8"?>
```

- minden tag-nek van záró-párja

```
<p>bekezdés  </p>
```

- van gyökérelem (dokumentum-elem)
- tag-ek egymásba ágyazva

XML jólformáltság 2

- megjegyzés

```
<!-- ez egy megjegyzés -->
```

- tag attribútum

```

```

- mindig idézőjelek között

- speciális jelek

```
&amp;  &  
&lt;    <  
&gt;    >  
&apos; '  
&quot;  ''
```

XML validitás

- Szemantikai megkötéseknek való megfelelés
- Megkötések megadása
 - séma (schema)
 - DTD (document type definition)
- Megadják a dokumentum általános struktúráját
 - milyen elemek engedélyezettek
 - milyen hierarchiában
 - milyen attribútumokkal
 - ...

DTD – Document Type Definition

- XML 1.0 szabvány óta
- SGML-től örököelve
- Igen elterjedt
- Elavult
 - újabb dolgokat (pl. namespace) nem támogatja
 - nem XML-ben ír le
 - van, amit nem lehet benne kifejezni

DTD 2

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate?, gender?,
    socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>27/11/2008</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```



XML Namespaces

XML névterek

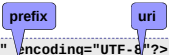
- Mint a package Java-ban, vagy a namespace .NET-ben: logikai szeparáció
- Névütközések elkerülése
- XML-re épülő szabványok erőteljesen használják
 - következő órán: webszolgáltatások, SOAP, WSDL, WS-* szabványok

XML névterek

■ Két fő jellemző:

- prefix: ez csak ideiglenes név
- URI: ez a lényeg

■ Példa:



```
<?xml version="1.0" encoding="UTF-8"?>
<lib:library xmlns:lib="http://example.com/library">
  <lib:book>
    <lib:title>Anathem</lib:title>
    <lib:author>Neal Stephenson</lib:author>
  </lib:book>
</lib:library>
```

Default névtér

■ Prefix nélkül

- az adott csúcsnak és minden leszármazottnak ez lesz a névtére, amíg felül nem bíráljuk

■ Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://example.com/library">
  <book>
    <title>Anathem</title>
    <author>Neal Stephenson</author>
  </book>
</library>
```

Üres névtér

- Ha nem használunk névtereket, akkor üres az alapértelmezett
- Példa:

mintha itt sem lenne

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="">
  <book>
    <title>Anathem</title>
    <author>Neal Stephenson</author>
  </book>
</library>
```

Egy URI, több prefix

- Egy közös URI-t többféle prefixszel is elnevezhetünk, a jelentés ugyanaz marad
- Ugyanis csak az URI számít, a prefix csak ideiglenes név
- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:library xmlns:lib="http://example.com/library">
  <lib:book>
    <lib2:title xmlns:lib2="http://example.com/library">
      Anathem
    </lib2:title>
    <lib:author>Neal Stephenson</lib:author>
  </lib:book>
</lib:library>
```

Prefix újradefiniálása

- A prefix csak ideiglenes: felüldefiniálható
- Ekkor a felüldefiniált prefix részfájában az új névtér érvényes
- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:library xmlns:lib="http://example.com/library">
  <lib:book>
    <lib:title>Anathem</lib:title>
    <lib:author>
      <lib:person xmlns:lib="http://example.com/person">
        <lib:name>Neal Stephenson</lib:name>
      </lib:person>
    </lib:author>
  </lib:book>
</lib:library>
```

Default névtér újradefiniálása

- Ekkor a felüldefiniált részében az új névtér érvényes
- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<library xmlns="http://example.com/library">
  <book>
    <title>Anathem</title>
    <author>
      <person xmlns="http://example.com/person">
        <name>Neal Stephenson</name>
      </person>
    </author>
  </book>
</library>
```

Prefixek előre definiálása

- Prefixeket előre is lehet definiálni, nem kötelező a legmélyebb ponton
- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:library xmlns:lib="http://example.com/library"
             xmlns:pers="http://example.com/person">
  <lib:book>
    <lib:title>Anathem</lib:title>
    <lib:author>
      <pers:person>
        <pers:name>Neal Stephenson</pers:name>
      </pers:person>
    </lib:author>
  </lib:book>
</lib:library>
```

Prefixes-default keverhető

- Prefixes és default névterek keverhetők
- Példa:

```
<?xml version="1.0" encoding="UTF-8"?>
<lib:library xmlns:lib="http://example.com/library">
  <lib:book>
    <lib:title>Anathem</lib:title>
    <lib:author>
      <person xmlns="http://example.com/person">
        <name>Neal Stephenson</name>
      </person>
    </lib:author>
  </lib:book>
</lib:library>
```

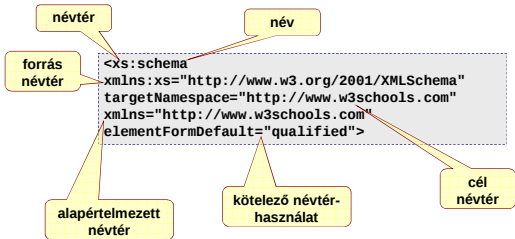


XML Schema Definition

XML Schema

- W3C ajánlás (2001)
- XML-leírást ad a dokumentum szerkezetéről
 - szótár (elemek és attribútumok nevei)
 - tartalmi modell (kapcsolatok és struktúra)
 - adattípusok
- *Post-Schema-Validation Infoset (PSVI)*
 - "típust" ad a dokumentumunknak
 - objektumként kezelhetjük a dokumentumot
- Az eredmény egy *XML schema definition* (XSD)

XSD: Gyökérelem



XSD: Egyszerű elem

```
<xs:element name="xxx" type="yyy"/>
```

■ Sok fajta beépített típus

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time
- ...

```
<xs:element name="lastname"  
  type="xs:string"/>  
<xs:element name="dateborn"  
  type="xs:date"/>
```

```
<lastname>Refsnes</lastname>  
<dateborn>1970-03-27</dateborn>
```

XSD: Kezdő és fix értékek

- Egyszerű elem kezdő és fix értékkel

```
<xs:element name="xxx" type="yyy" default="zzz"/>
```

```
<xs:element name="xxx" type="yyy" fix="qqq"/>
```

- Példa:

```
<xs:element name="color" type="xs:string"  
  default="blue"/>  
<xs:element name="pi" type="xs:decimal"  
  fix="3.14159265358979323844"/>
```

XSD: Attribútumok

```
<xs:attribute name="xxx" type="yyy"/>
```

```
<xs:attribute name="xxx" type="yyy" default="qqq"/>
```

```
<xs:attribute name="xxx" type="yyy" fix="qqq"/>
```

■ Kötelező attribútum

```
<xs:attribute name="xxx" type="yyy" use="required"/>
```

XSD: Típusmegkötések

- Beágyazott típusdefinícióval

```
<xs:element name="password">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:length value="8"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

XSD: Típusmegkötések

- Külön típusdefinícióval

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Zhiguli"/>
    <xs:enumeration value="Moskvitch"/>
    <xs:enumeration value="Zaporozhets"/>
  </xs:restriction>
</xs:simpleType>
```

XSD: Típusmegkötések

- ☐ totalDigits
- ☐ fractionDigits
- ☐ minExclusive
- ☐ maxExclusive
- ☐ minInclusive
- ☐ maxInclusive
- ☐ enumeration
- ☐ length
- ☐ minLength
- ☐ maxLength
- ☐ pattern
- ☐ whiteSpace

XSD: Összetett elemek

```
<xs:element name="employee" type="personinfo"/>  
  
<xs:complexType name="personinfo">  
  <xs:sequence>  
    <xs:element name="firstname" type="xs:string"/>  
    <xs:element name="lastname" type="xs:string"/>  
  </xs:sequence>  
</xs:complexType>
```

- beágyazott megadás is lehet

XSD: Típusöröklés

```
<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

XSD: Keverett összetett elemek

- Szöveg és elemek egyszerre

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid"
                  type="xs:positiveInteger"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD: Indikátorok

■ Sorrend:

- **all**: minden elem, tetszőleges sorrendben
- **choice**: pontosan egy elem
- **sequence**: minden elem, adott sorrendben

■ Gyakoriság:

- **maxOccurs**: legfeljebb ennyiszer ismétlődhet
- **minOccurs**: legalább ennyiszer kell ismétlődnie

■ Csoportosítás:

- **group name**: referencia létrehozása elemek egy csoportjára
- **attributeGroup name**: referencia létrehozása attribútumok egy csoportjára

XSD: Indikátor példák

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD hivatkozása XML-ből

Alapértelmezett névtér

```
<?xml version="1.0"?>
```

```
<note
```

```
xmlns="http://www.w3schools.com"
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:schemaLocation="http://www.w3schools.com note.xsd">
```

```
<to>Tove</to>
```

```
<from>Jani</from>
```

```
<heading>Reminder</heading>
```

```
<body>Don't forget me this weekend!</body>
```

```
</note>
```

Sémapéldány

Névtér + séma elérése

XSD példa: az alap XML dok.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>Gáz Géza</orderperson>
  <shipto> <name>Cserép Virág</name>
           <address>Tér köz 3</address>
           <city>4567 Bükkösbánat</city>
        </shipto>
  <item> <title>U.Eco: Foucault-inga</title>
         <note>Nem kell a borító</note>
         <quantity>1</quantity>
        </item>
  <item> <title>DJ Dick</title>
         <quantity>1</quantity>
        </item>
</shiporder>
```

XSD példa

■ Séma létrehozása

XSD dokumentum gyökere

```
<?xml version="1.0" encoding="ISO-8859-1" ?>  
<xs:schema  
  xmlns:xs="http://www.w3.org/2001/XMLSchema">  
  
  ... // shiporder elem definíció  
  
</xs:schema>
```

XML dokumentum gyökere

XSD példa

■ Shiporder elem definiálása

```
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      ... // belső elemek:
      ... //  orderperson, shipto,
      ... //  item*
    </xs:sequence>
  </xs:complexType>
  ... // attribútumok: orderid
</xs:element>
```

XSD példa

■ Shiporder elem tartalma 1

```
<xs:element name="orderperson" type="xs:string"/>
```

```
<xs:element name="shipto">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="name" type="xs:string"/>  
      <xs:element name="address" type="xs:string"/>  
      <xs:element name="city" type="xs:string"/>  
    </xs:sequence>  
  </xs:complexType>  
</xs:element>
```

XSD példa

■ Shiporder elem tartalma 2

Ismétlődő

```
<xs:element name="item" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="title" type="xs:string"/>
      <xs:element name="note" type="xs:string"
        minOccurs="0"/>
      <xs:element name="quantity"
        type="xs:positiveInteger"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Opcionális

XSD példa

■ Shiporder elem tartalma 3

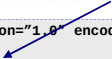
- az attribútumot mindig utolsóként kell megadni!

```
<xs:attribute name="orderid"  
              type="xs:string"  
              use="required"/>
```

XML rekurzív beágyazása

```
<?xml version="1.0" encoding="UTF-8" ?>
<po id="123">
  <billTo id="addr1">
    <company>XXX Warehouse</company>
    <street>Bell Park 12456.</street>
    <city>Boston</city>
    <postalCode>41776</postalCode>
  </billTo>
  <shipTo href="addr1">
</po>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<example>
  .
  .
  .
</example>
```



Megoldás 1.

```
<?xml version="1.0" encoding="UTF-8" ?>
<example>
    &lt;?xml version=&quot;1.0&quot;
        encoding=&quot;UTF-8&quot; ?&gt;
    &lt;po id=&quot;123&quot;&gt;
        &lt;billTo id=&quot;addr1&quot;&gt;
            . . .
        &lt;/billTo>
    &lt;/po>
</example>
```

Megoldás 2.

```
<?xml version="1.0" encoding="UTF-8" ?>
<example><![CDATA
  <?xml version="1.0" encoding="UTF-8" ?>
  <po id="123">
    <billTo id="addr1">
      <company>XXX Warehouse</company>
      <street>Bell Park 12456.</street>
      <city>Boston</city>
      <postalCode>41776</postalCode>
    </billTo>
    <shipTo href="addr1">
  </po>
]]></example>
```

XPath és XSLT

XPath

- XML dokumentumban található elemek leírására
- Hasonlóan a könyvtárstruktúra leírásához
- Több mint 100 függvényt definiál
- Kényelmesen tudunk a dokumentumban elemeket elérni, referálni

XPath elérések

- csomópontnév
 - adott nevű csomópont gyerekei
- /
 - gyökérelem
- //
 - az aktuális elemtől választ, mindegy, hogy milyen mélyen
- .
 - aktuális elem
- ..
 - az aktuális elem szülője
- @
 - attribútum

XPath elérések példái

- **bookstore**

- kiválasztja a **bookstore** nevű elem gyerekeit

- **/bookstore**

- kiválasztja a gyökerben levő **bookstore** elemet

- **bookstore/book**

- kiválasztja a **bookstore** elem **book** gyerekét

- **//book**

- minden **book** elemet kiválaszt

- **bookstore//book**

- minden **book** elemet kiválaszt **bookstore** alatt

- **//@lang**

- kiválasztja az összes **lang** attribútumot

XPath predikátumok

- Szűkítéshez, mindig szögletes zárójelben []
- Sorrend fontos
- Példák
 - **/bookstore/book[1]**
 - bookstore első book gyereke (IE5!!!)
 - **/bookstore/book[last()]**
 - bookstore utolsó book gyereke
 - **/bookstore/book[last()-1]**
 - bookstore utolsó előtti book gyereke
 - **/bookstore/book[position()<6][lang='en']**
 - bookstore első 5 book gyereke közül az angol nyelvűek

XPath predikátumok

■ Példák (folyt)

- `//title[@lang]`

- minden **title** elem, amiben van **lang** attribútum

- `//title[@lang='hu']`

- minden **title** elem, amiben a **lang** attribútum értéke **'hu'**

- `/bookstore/book[price>35.00]`

- minden **book** a **bookstore**-ban, aminek a **price** eleme nagyobb, mint 35.00

- `/bookstore/book[price>35.00]/title`

- minden olyan **book**-nak a **title**-je a **bookstore**-ban, aminek a **price** eleme nagyobb, mint 35.00

XPath dzsóker-karakterek

■ A shell-helyettesítéshez hasonlóan



- bármely elem
- pl. `/bookstore/*` a **bookstore** összes gyereke
- pl. `/a/b/*/c`



- bármely attribútum
- pl. `//title[@*]` az összes olyan **title**, aminek van attribútuma

XPath csomópont függvények

■ Csomópont típusokra

□ **node()**

- bármely csomópont (elem vagy attribútum)

□ **text()**

- szöveg

□ **comment()**

- megjegyzés

□ **processing-instruction()**

- `<?php echo $a; ?>`

XPath ág (axis)

- Kiválaszt egy adott elem-halmazt az aktuális elemtől nézve
 - **self**
 - az elem maga
 - **parent**
 - a szülő
 - **ancestor / ancestor-or-self**
 - az összes ő a fában (plusz az elem)
 - **child**
 - a gyermekek
 - **descendant / descendant-or-self**
 - az összes leszármazott a fában (plusz az elem)

XPath ág (axis) 2

- **following**
 - minden, ami az elem után jön az XML dokumentumban
- **following-sibling**
 - az elem utáni testvér-elemek
- **preceding**
 - minden, ami az elem előtt van az XML dokumentumban
- **preceding-sibling**
 - az elem előtti testvérek
- **namespace**
 - az elem összes névtere
- **attribute**
 - az elem attribútumai

XPath elérési utak

- Abszolút út
 - `/lépés/lépés/...`
- Relatív út
 - `lépés/lépés/...`
- Lépés
 - `ágnév::elemszűkítés[predikátum]`

XPath elérési utak 2

■ Példák

- **child::book**

- összes **book** gyerek

- **attribute::lang**

- összes **lang** attribútum

- **child::***

- az aktuális elem összes gyereke

- **attribute::***

- az aktuális elem összes attribútuma

XPath elérési utak 3

- **child::text()**
 - az összes **text** gyerek
- **child::node()**
 - az összes csomópont gyerek
- **descendant::book**
 - az összes leszármazott, ami **book**
- **ancestor-or-self::book**
 - az összes ő, aki **book**, beleértve magát is
- **child::* / child::price**
 - az összes **price** unoka

XPath elérési utak 4

■ Rövidítések

- **attribute** @
- //a/@href
- **self, self::node()** .
- **descendant-or-self** //
- **parent** ..
- **child**

- alapértelmezett, ha az ág nincs megadva

XPath elérési utak 4

- Rövidítés példák

descendant-or-

self::node()/child::a/attribute::href

//a/@href

child::A/ descendant-or-

self::node()/child::B[position()=1]

A//B/*[1]

XPath operátorok

- alapműveletek

- `+` `-` `*` `/` `div` `mod`

- pl. `//item[@price > 2*@discount]`

- logikai műveletek

- `and` `or`

- relációs műveletek

- `=` `!=` `<` `<=` `>` `>=`

- halmaz-művelet (unió)

- `|` (függőleges vonal)

- pl. `v[x or y] | w[z]`

XPath függvények

■ Közel 100 függvény

- `fn:round(num)`
- `fn:concat(string, string, ...)`
- `fn:substring(string, start, len)`
- `fn:month-from-dateTime(datetime)`
- `fn:name(nodeset)`
- `fn:index-of((item, item, ...), searchitem)`
- `fn:reverse((item, item, ...))`
- `fn:avg((arg, arg, ...))`
- `fn:current-time()`
- ...

XSLT

- Xml StyLe sheets Transformations
 - maga is XML-ben megadva
- XML dokumentumok transzformációját írja le
 - az dokumentumban levő fát alakítja át új dokumentummá
 - az átalakítás szabályainak megadásával
- XPath leírással navigálhatunk
 - az egyes szabályokban az XPath-ban definiált módon adhatjuk meg az elemeket
- W3C ajánlás



Simple API for XML

SAX

- Simple API for XML
- Esemény-vezérelt
 - callback minta
- Soros elérés
 - alacsony memória-igény
 - nagy sebesség
 - nem lehet előre- vagy visszaugrani
- Állapotfüggetlen feldolgozás

SAX parser implementálás

- Események feldolgozása
 - dokumentum eleje/vége
 - elem (tag) eleje/vége
 - prefixMapping eleje/vége
 - karakterek (szöveg)
 - kihagyott entitás
 - whitespace-ek
 - feldolgozási utasítások

ContentHandler interfész

- **org.xml.sax.ContentHandler**
- Ezt kell a parsernek megvalósítani
- Az események feldolgozása callback alapon
- Üres implementáció
 - **org.xml.sax.helpers.DefaultHandler**
 - minden metódus törzse üres

ContentHandler

- **void startDocument() – void endDocument()**
 - dokumentum kezdete/vége
- **void startElement(String uri, String localName, String qName, Attributes atts)**
- **void endElement(String uri, String localName, String qName)**
 - elem kezdete/vége: teljes és relatív név, attribútumok
- **void startPrefixMapping(String prefix, String uri)**
- **void endPrefixMapping(String prefix)**
 - az előtag névtér megadása

ContentHandler

- **void characters(char[] ch, int start, int length)**
 - sima szöveg (törhet is!)
- **void ignorableWhitespace(char[] ch, int start, int length)**
 - elem tartalmában levő whitespace
- **void processingInstruction(String target, String data)**
 - a gyökérellem előtt/után előforduló feldolgozási utasítások
- **void skippedEntity(String name)**
 - kihagyott elemek
- **void setDocumentLocator(Locator locator)**
 - az események forrását lehet megkapni

Locator

- Szükség lehet információra, hogy merre is járunk a fájlban
- **void setDocumentLocator(Locator l)**
 - **int getColumnNumber()**
 - aktuális elem oszlopa
 - **int getLineNumber()**
 - aktuális elem sora
 - **String getPublicId()**
 - a dokumentum publikus azonosítója, ha van
 - **String getSystemId()**
 - a dokumentum neve (pl. fájlnev) URL formában

Dokumentum validálás

- Az eddig használt parser nem validált

```
SAXParserFactory factory = SAXParserFactory.newInstance();  
factory.setValidating(true);  
factory.setNamespaceAware(true);
```

```
SAXParser p = factory.newSAXParser();  
String JAXP_SCHEMA_LANGUAGE =  
    "http://java.sun.com/xml/jaxp/properties/schemaLanguage";  
String W3C_XML_SCHEMA =  
    "http://www.w3.org/2001/XMLSchema";  
p.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

Hibakezelés

■ Három hibatípus

☐ fatal error

- a dokumentum nem jólformált

☐ error

- a dokumentum nem valid

☐ warning

- figyelmeztetés
- pl. ha kétszer definiálunk egy típust

Hibakezelés 2

■ Szintaktikai hiba

- *fatal error*

- **SAXParseException** dobódik

```
org.xml.sax.SAXParseException: The element type "level3"  
must be terminated by the matching end-tag "</level3>".
```

```
...
```

```
at javax.xml.parsers.SAXParser.parse(SAXParser.java:395)
```

```
at javax.xml.parsers.SAXParser.parse(SAXParser.java:331)
```

```
at MyParser.main(MyParser.java:16)
```

Hibakezelés 3

- Szemantikai hiba
 - *(non-fatal) error*
 - csak ha van validálás
 - hibát le kell kezelni:

```
public void error(SAXParseException e)
throws SAXParseException {
    throw e;
}
```

```
org.xml.sax.SAXParseException: cvc-complex-type.2.4.a:
Invalid content was found starting with element 'b'. One
of '{level3}' is expected.
```



Document Object Model

DOM

- *Document Object Model*
- Felépíti a dokumentum alapján a fa reprezentációt
- A reprezentáció módosítható
- Képes validálni

DOM bevezető példa

```
try {
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    factory.setValidating(true);
    factory.setNamespaceAware(true);

    DocumentBuilder builder =
        factory.newDocumentBuilder();

    Document document =
        builder.parse(new java.io.File(args[0]));
    ...
} catch (Exception e) {
    e.printStackTrace();
}
```

Document

- A dokumentumot reprezentáló objektum
 - nem a gyökér elem!
- Lekérdezhetők alapadatok
 - doctype, XML verzió, stb
- Gyártani tud elemeket
 - attribútum, element, comment, stb.
- Maga is egy **Node**

Node

- A fa egy eleme
 - element, attribútum, comment, text, stb.
- Lekérhetjük és módosíthatjuk az adatait
 - nevét, típusát, értékét, stb.
- Navigálhatunk benne
 - felfele: szülő és dokumentum
 - lefele: gyerekek
 - oldalt: testvérek
- Módosíthatjuk a gyerekeit (hozzáadás, törlés)

Segédosztályok

■ **NodeList**

- mikor egy **Node** gyerekeit kérjük el
 - **NodeList** **Node.getChildren()**
- **int** **getLength()**
 - a lista mérete
- **Node** **item(int index)**
 - az index-edik elem

Segédosztályok

■ **NamedNodeMap**

- mikor egy **Node** attribútumait kérjük el
 - **NamedNodeMap** **Node.attributes()**
- **int** **getLength()**
 - a halmaz mérete
- **Node** **item(int index)**
 - az index-edik Node
- **Node** **getNamedItem(String name)**
- **Node** **getNamedItemNS(String namespaceURI, String localName)**
 - az adott nevű Node (névtérrel együtt)

Segédosztályok

■ **NamedNodeMap** (folyt.)

- **Node removeNamedItem(String name)**
- **Node removeNamedItemNS(String namespaceURI, String localName)**
 - törli az adott elemet a halmazból és a **Node**-ból
 - ha van default érték, arra cseréli
- **Node setNamedItem(Node arg)**
- **Node setNamedItemNS(Node arg)**
 - módosítja az elem értékét
 - a **nodeName** attribútum alapján tárol

Példa: egyszerű kiíratás

```
static void print(Node n, String tab) {  
    System.out.println(tab+"("+n.getNodeName()+") \\"  
        +n.getNodeValue()+"\"");  
    if (n.hasAttributes()) {  
        NamedNodeMap map = n.getAttributes();  
        for (int i = 0; i < map.getLength(); i++) {  
            Node n1 = map.item(i);  
            print(n1, tab+"attr: ");  
        }  
    }  
    NodeList nl = n.getChildNodes();  
    for (int i = 0; i < nl.getLength(); i++) {  
        Node n1 = nl.item(i);  
        print(n1, tab+"  ");  
    }  
}
```

Rekurzió

DOM specialitások

- XML *tag* neve a **Node** neve
 - `getNodeName()`
- XML *tag* értéke a **Node** **#text** nevű gyerek értéke
 - `getNodeValue()`
- XML attribútum neve és értéke a **Node** neve és értéke
 - de van egy **#text** gyereke is az értékkel

JDOM

- Java- és XML-közelibb objektum-modell
 - **Attribute, CDATA, Comment, Content, DefaultJDOMFactory, DocType, Document, Element, EntityRef, Namespace, ProcessingInstruction, Text**
- Attribútumok, leszármazottak könnyebben elérhetők, módosíthatók
 - nincs **NamedNodeMap, NodeList**
 - helyette **java.util.List**

JDOM

- Szűrők is beállíthatók a leszármazott-kereséshez
 - **org.jdom.filter.Filter**
 - **boolean matches(java.lang.Object obj)**
- XSL transzformációk támogatása
 - **org.jdom.transform.XSLTransformer**
- XPATH keresések támogatása (Jaxan)
 - **org.jdom.xpath.XPath**

JDOM

- A beolvasást másra hagyja
 - **DOMBuilder**
 - DOM-ból épít
 - **SAXBuilder**
 - SAX eseményekből épít
- A dokumentum kimenthető
 - XML dokumentumként
 - DOM modellként
 - SAX esemény-generátorral

JDOM

```
SAXBuilder builder = new SAXBuilder();  
Document doc = builder.build(new File("..."));  
Element e = doc.getRootElement();  
  
List<Attributes> attrList = e.getAttributes();  
...  
List<Element> books = e.getChildren();  
...
```

JDOM Element 1.

- **Attribute** `getAttribute(String name)`
 - A megadott nevű attribútumot kéri le
- **List<Attribute>** `getAttributes()`
 - Az összes attribútumot visszaszítja listában
- **String** `getAttributeValue(...)`
 - A megadott nevű attribútum értékét adja
- **Element** `getChild(String name)`
 - A megadott nevű gyerekelemet adja
- **List<Element>** `getChildren()`
 - Az összes gyereket visszaadja listában
- **String** `getChildText(String name)`
 - A megadott gyerekelemhez tartozó szöveget adja vissza.

JDOM Element 2.

- **String getName()**

- Az elem nevét adja vissza (névtér nélkül)

- **String getText()**

- Az elem szövegrészé(ei)t adja vissza

- **Namespace getNamespace()**

- Az adott elem névterét adja vissza

- **boolean isRootElement()**

- gyökérelem-e az adott elem

JDOM Attribute

- **getName()**

- Az attribútum nevét adja vissza

- **getValue()**

- Az attribútum értékét stringben adja vissza

- **getBooleanValue(), getIntValue(),
getDoubleValue(), getFloatValue()**

- Az attribútum értékét a megadott típusra alakítva adja vissza.

- **getNamespace(), getNamespacePrefix(),
getNamespaceURI()**

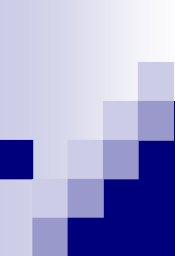
- A névtér adatait adják vissza

JDOM Namespace

- Egy névtér objektum a prefixet és az URI-t tartalmazza:
 - `getPrefix()`
 - `getURI()`

JDOM kimenet

```
Element e;  
e.addContent(...);  
e.setAttribute(...);  
...  
XMLOutputter outputter = new XMLOutputter();  
outputter.output(doc, new FileWriter("..."));
```



Objektum-orientált tervezés

Web-szolgáltatások

Simon Balázs

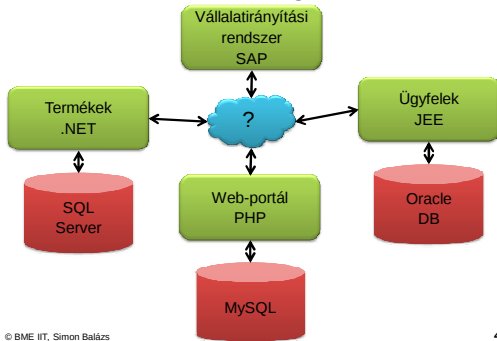
Tartalom

- Integrációs feladat
- Service Oriented Architecture
- Web-service
- SOAP
- WSDL
- Web-szolgáltatás API-k
- DEMO

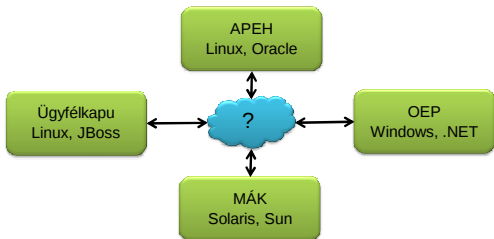


Integrációs feladat, Service Oriented Architecture

Feladat: vállalati integráció



Feladat: közigazgatási integráció





Feladat

- Alkalmazások közötti kommunikáció
- Különböző programnyelvek
- Különböző operációs rendszerek
- Különböző szoftvergyártók
- Sokfajta meglévő, működő rendszer
- Integráció
- Üzleti folyamatok elektronizálása

Követelmények

- Egyszerű
- Szabványos
- Programnyelvektől, operációs rendszertől független
- Széles körű támogatás
- Middleware feladatok:
 - megbízható üzenetküldés
 - titkosítás, digitális aláírás
 - tranzakciókezelés
- Megoldás: SOA, web-szolgáltatások

Service Oriented Architecture

- Architektúrafejlesztési paradigma
- Alapegység: szolgáltatás
- Szereplők (szerepek):
 - szolgáltató
 - felhasználó
- OASIS definíció:
 - paradigma
 - elosztott erőforrások és képességek
 - szervezésére és hasznosítására
 - felkínálás, felderítés, interakció

Szolgáltatás

- Az architektúra alapegysége
- Erőforrás vagy képesség publikálása
- Jól definiált, szabványos interfész
- Elrejtí az implementáció részleteit
- Lehetőleg minél vékonyabb réteg
- Fontos a jó tervezés:
 - a kiajánlott funkcionalitás
 - a megfelelő granularitás
 - általánosság
 - újrafelhasználhatóság



Példák: vállalat

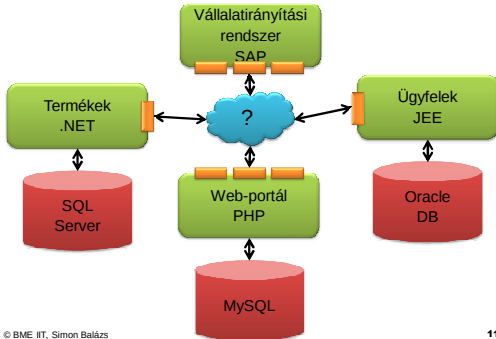
■ szolgáltatások:

- ☐ ügyfél adatainak lekérdezése
ügyfélazonosító alapján
- ☐ termék adatainak lekérdezése
termékazonosító alapján
- ☐ új ügyfél felvétele
- ☐ termék készletbe vétele
- ☐ termék eladása

■ folyamatok:

- ☐ ügyfél felad egy rendelést

Vállalati integráció



Példák: közigazgatás

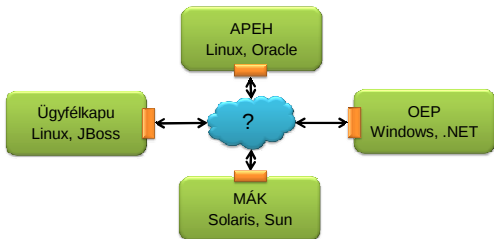
■ szolgáltatások:

- ☐ lakcím megváltoztatása
- ☐ adóbevallás fogadása
- ☐ állampolgár adótartozásának lekérdezése
- ☐ anyasági támogatás igénylése

■ folyamatok:

- ☐ lakcímváltozás bejelentése
- ☐ adóbevallás benyújtása
- ☐ vállalkozás alapítása

Közigazgatási integráció



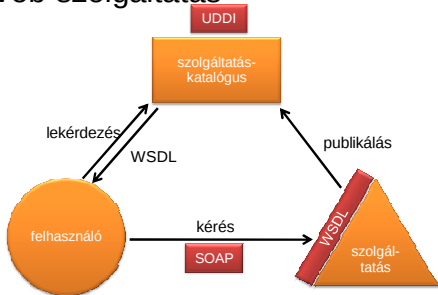


Web-szolgáltatás

Web-szolgáltatás

- Konkrét technológia szolgáltatások megvalósításához
- Szabványos (OASIS, W3C)
- Programnyelvektől, operációs rendszerektől független
- XML alapú
- Szállító protokoll: tipikusan HTTP
- Elterjedt, széles körben támogatott
- Számos szabvány middleware feladatokra

Web-szolgáltatás



WSDL = Web-Services Description Language

SOAP = Simple Object Access Protocol

UDDI = Universal Description, Discovery and Integration

Web-szolgáltatás

- Definíció:

- SOAP üzeneten keresztül meghívható szolgáltatás

- Üzenetformátum:

- SOAP = Simple Object Access Protocol
 - Használt verziók: 1.1 és 1.2

- Interfészleíró:

- WSDL = Web-Services Description Language
 - Használt verziók: 1.1 és 2.0

- Szolgáltatáskatalógus:

- UDDI = Universal Description Discovery and Integration
 - Használt verziók: 2.0 és 3.0

Miért?

- Korábban: CORBA, DCOM
- DCOM: csak Windows
- CORBA:
 - különböző gyártók implementációi közti együttműködés kérdéses
 - ha sikerülne is, a biztonság és a tranzakciókezelés problémás
- nem XML alapúak
- adatformátumok túl alacsony szintűek: byte-ok igazításával is foglalkozni kell
- tűzfalak, proxy szerverek általában korlátozzák ezeket



SOAP

(Simple Object Access Protocol)

SOAP történet

■ Eredetileg:

- 1998. Microsoft
- SOAP = Simple Object Access Protocol
- Eredeti cél: XML alapú RPC

■ Ma:

- W3C gondozásában
- Használt verziók: 1.1 és 1.2
- Már csak a SOAP rövidítés maradt meg, mivel a „Simple Object Access Protocol” félrevezető

SOAP

- Kommunikációs protokoll
- Alkalmazások között
- XML-re épül
- Platformfüggetlen
- Programnyelvfüggetlen
- Egyszerű
- Kiterjeszthető
 - Id. később: WS-* szabványok
- Független az alatta lévő kommunikációs csatornától, de általában HTTP felett alkalmazzák (így a tűzfalon is átmegy)

SOAP envelope

```
<?xml version="1.0" encoding="utf-8"?>
<env:Envelope xmlns:env="[SOAP névtér]">
  <env:Header>
    <myns:MyHeader1 xmlns:myns="[MyNamespace]"/>
    <myns:MyHeader2 xmlns:myns="[MyNamespace]"
      env:mustUnderstand="1"/>
    ...
  </env:Header>
  <env:Body>
    ...
    <env:Fault>
      ...
    </env:Fault>
  </env:Body>
</env:Envelope>
```

Envelope

Header?

Saját fejléc*

Body

Saját tartalom?
vagy Fault?

SOAP 1.1

- SOAP üzenet névtér:

- <http://schemas.xmlsoap.org/soap/envelope/>

- WSDL névtér:

- <http://schemas.xmlsoap.org/wsdl/soap/>

- HTTP fejlécek:

- **POST** [Lokális URL] HTTP/1.1

- Content-Type: text/xml; charset="utf-8"**

- SOAPAction: [Action]**

- a SOAPAction kötelező

SOAP 1.2

- SOAP üzenet névtér:
 - <http://www.w3.org/2003/05/soap-envelope>
- WSDL névtér:
 - <http://schemas.xmlsoap.org/wsdl/soap12/>
- A SOAP 1.1-hez képest más a Fault szerkezete
- HTTP fejlécek:
 - **POST** [Lokális URL] HTTP/1.1
Content-Type: application/soap+xml;
charset=utf-8;
action="[Action]"
 - az action opcionális
 - a GET is támogatott

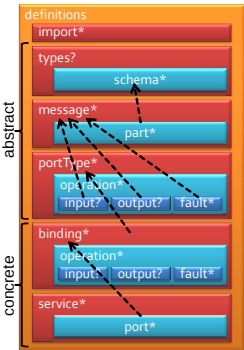


WSDL *(Web-Services Description Language)*

WSDL

- Web Services Description Language
- Leíró web-szolgáltatásokhoz
 - interfész
 - meta-adatok
 - szolgáltatások címei
- W3C gondozásában
- Használt verziók: 1.1 és 2.0

WSDL 1.1



- **definitions:** gyökér elem
- **import:** más WSDL importálása
- **types:** a felhasznált típusok
 - XML schema (XSD)
- **message:** paraméterlista
 - **part:** paraméter
- **portType:** interfész
 - **operation:** művelet
 - **input:** bemenő paraméterlista
 - **output:** kimenő paraméterlista
 - **fault:** kivétel
- **binding:** hívási konvenció
 - protokoll, adatformátum, kódolás
- **service:** implementáció
 - **port:** az adott binding alapján hívható szolgáltatás, konkrét helye

WSDL 1.1: típusok XSD-ben

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="[MyNamespace]"
            xmlns:myns="[MyNamespace]"
            elementFormDefault="qualified">

  <xsd:import schemaLocation="..." namespace="[MyNamespace]"/>*

  <xsd:element name="[ncname]" type="[qname]"/>*

  <xsd:complexType name="[ncname]"/>*

</xsd:schema>
```

Jelmagyarázat:

[ncname] = név definíció (pl. "Add")

[qname] = név hivatkozás névtérprefixszel együtt (pl. "myns:Add")

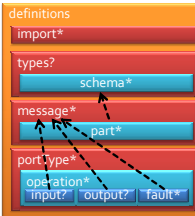
WSDL 1.1: absztrakt rész

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="PilotTest" targetNamespace="[MyNamespace]"
  xmlns:myns="[MyNamespace]" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:import location="..." namespace="[MyNamespace]"/>

  <wsdl:types?>
    <xsd:schema targetNamespace="[MyNamespace]" elementFormDefault="qualified">
      <xsd:import schemaLocation="..." namespace="[MyNamespace]"/>
    </xsd:schema>
  </wsdl:types>

  <wsdl:message name="[ncname]">
    <wsdl:part name="[ncname]" element="[qname]"/>
  </wsdl:message>

  <wsdl:portType name="[ncname]">
    <wsdl:operation name="[ncname]">
      <wsdl:input message="[qname]"/>
      <wsdl:output message="[qname]"/>
      <wsdl:fault name="[ncname]"
        message="[qname]"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```



WSDL 1.1: konkrét rész

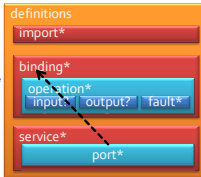
```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions name="PilotTest" targetNamespace="[MyNamespace]"
  xmlns:myns="[MyNamespace]" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"

  <wsdl:import location="..." namespace="[MyNamespace]"/>

  <wsdl:binding name="[ncname]" type="[qname]">*
    <wsdl:operation name="[ncname]">*
      <wsdl:input>...</wsdl:input>?
      <wsdl:output>...</wsdl:output>?
      <wsdl:fault name="[ncname]">...</wsdl:fault>*
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="[ncname]">*
    <wsdl:port name="[ncname]" binding="[qname]">...</wsdl:port>*
  </wsdl:service>

</wsdl:definitions>
```



WSDL 1.1 MEP

- MEP = Message Exchange Pattern
- Az input és output operációk meglététől függően

	van output	nincs output
van input	request-reply	one-way (aszinkron)
nincs input	solicit-response	-

Két szolgáltatás interfészének ekvivalenciája

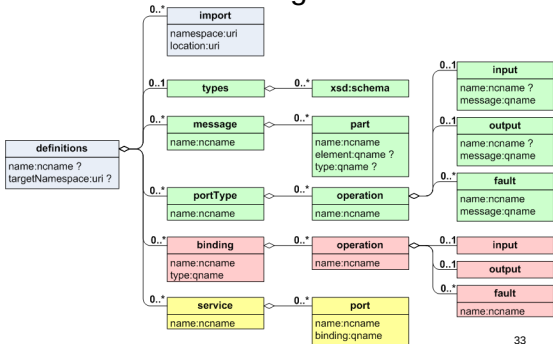
■ Egyeznie kell:

- a types részben felhasznált XSD típusoknak, elemeknek és ezek XML névterének
- az Action-nek
- a binding beállításainak
- a policy beállításoknak
- ha az Action nem explicit adott:
 - a WSDL targetNamespace-ének, a portType, operation, input, output és fault nevének

■ Nem szükséges egyeznie:

- a message nevének, a part nevének
- a binding nevének
- a service nevének, címének
- ha az Action explicit adott:
 - a WSDL targetNamespace-ének, a portType, operation, input, output és fault nevének

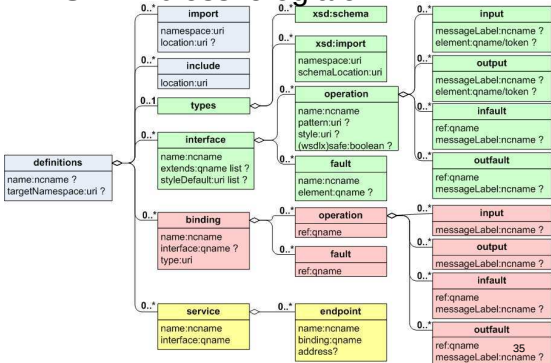
WSDL 1.1 összefoglaló



WSDL 2.0

- Még nem szabvány, csak ajánlás
- Nem igazán elterjedt, nem nagyon támogatott
- Egyszerűbb, mint az 1.1-es verzió
- Gyökérelem: **definitions** helyett **description**
- Van **import** és **include** is
- Csökkent a redundancia: eltűnt a **message** rész
- Készíthetők újrahasznosítható **binding**-ok
- Jobban hasonlít a hagyományos programnyelvi interfészekhez: **portType** helyett **interface**
- Megjelent az öröklődés interfészek között (többszörös öröklődés is támogatott)

WSDL 2.0 összefoglaló





Web-szolgáltatás API-k

JAX-WS

- Java API for XML-based Web-Services
- Célja:
 - Web-szolgáltatások egyszerű implementálása
 - WSDL és Java osztályok közti leképezés
 - Annotációkkal
- Referencia implementáció:
 - Sun: GlassFish ESB
 - <https://jax-ws.dev.java.net/>

JAX-WS példa: interfész

```
@WebService(name = "ICalculator",  
            targetNamespace = "http://ik.bme.hu/WsApi")  
public interface ICalculator  
{  
    @WebMethod(operationName = "Divide")  
    public double divide(  
        @WebParam(name = "left") double left,  
        @WebParam(name = "right") double right)  
        throws MathFaultFaultMessage;  
}
```

JAX-WS példa: implementáció

```
@WebService(endpointInterface = "ICalculator")
public class Calculator implements ICalculator
{
    @Override
    public double divide(double left, double right)
        throws MathFaultFaultMessage
    {
        if(right == 0)
        {
            MathFault fault = new MathFault();
            fault.setReason("Division by zero.");
            throw new MathFaultFaultMessage("Error", fault);
        }
        return left / right;
    }
}
```

JAX-WS példa: szolgáltatás meghívása

- A `wsimport` által generált `CalculatorService` proxy osztály segítségével
- Példa:

```
CalculatorService service = new CalculatorService();
Calculator calculator = service.getCalculatorPort();
try {
    double result = calculator.divide(5, 2);
}
catch (MathFaultFaultMessage ex) {
    ex.printStackTrace();
}
finally {
    ((Closeable)calculator).close();
}
```

WCF

- Windows Communication Foundation (.NET 3.0)
- Célja:
 - Web-szolgáltatások egyszerű implementálása
 - WSDL és .NET osztályok közti leképezés
 - .NET attribútumokkal

WCF példa: interfész

```
[ServiceContract(Namespace = "http://ik.bme.hu/WsApi")]  
public interface ICalculator  
{  
    [OperationContract]  
    [FaultContract(typeof(MathFault), Name = "MathFault")]  
    double Divide(double left, double right);  
}
```

WCF példa: implementáció

```
public class Calculator : ICalculator
{
    public double Divide(double left, double right)
    {
        if (right == 0)
        {
            throw new FaultException<MathFault>(
                new MathFault() { Reason = "Division by zero." });
        }
        return left / right;
    }
}
```

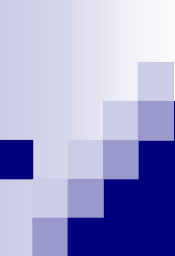
WCF példa: szolgáltatás meghívása

- Az **SvcUtil** által generált **CalculatorClient** proxy osztály segítségével
- Példa:

```
CalculatorClient calculator = new CalculatorClient();  
try  
{  
    double result = calculator.Divide(5, 2);  
}  
catch (FaultException<MathFault> ex)  
{  
    Console.WriteLine(ex);  
}  
finally  
{  
    calculator.Close();  
}
```



DEMO: JAX-WS, WCF



Objektum-orientált tervezés

Web-szolgáltatás szabványok

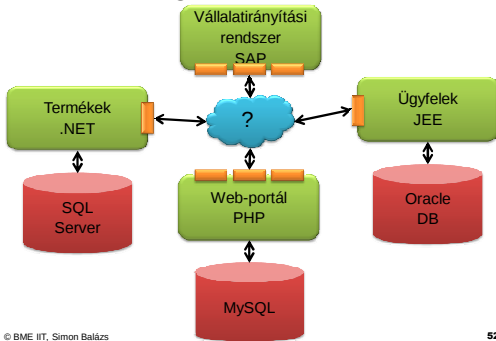
Simon Balázs

Tartalom

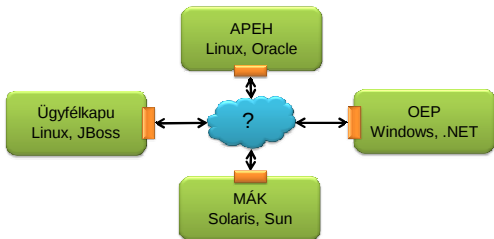
- Követelmények
- WS-* szabványok
- DEMO

Integrációs követelmények

Vállalati integráció



Közigazgatási integráció

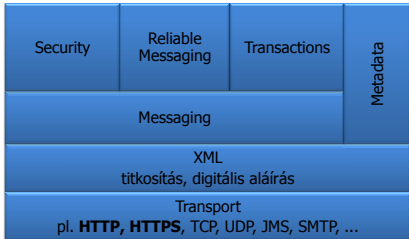


Követelmények

- Vállalaton belüli integráció:
 - tranzakciókezelés
- E-Közigazgatási integráció, vállalatok közti integráció:
 - biztonság: titkosítás, digitális aláírás
 - megbízhatóság: üzenet nem veszik el
- Szabványos megoldás

WS- szabványok*

Web-Service szabványok



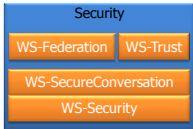
Messaging

- Üzenetkezelés
- WS-Addressing: címzés
 - SOAP fejlécek:



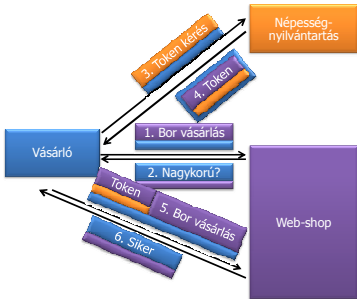
- Action
 - To
 - From
 - ReplyTo
 - FaultTo
 - MessageId
 - RelatesTo
- MTOM: bináris adatok küldése byte-folyamként külön MIME részben

Security



- **WS-Security:**
 - titkosítás, digitális aláírás
- **WS-SecureConversation:**
 - szimmetrikus kulcs generálása, titkosított adatcsere
 - (analógia: SSL)
- **WS-Trust:**
 - tokenek kérése, kibocsátása
 - (analógia: Kerberos)
- **WS-Federation**
 - trusted domain-eken túli azonosítás
 - single sign-on

WS-Federation példa



Reliable messaging

- analógia: TCP

- WS-Reliability:

 - ☐ eredeti változat

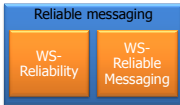
 - ☐ nem veszi figyelembe a többi protokollt

- WS-ReliableMessaging:

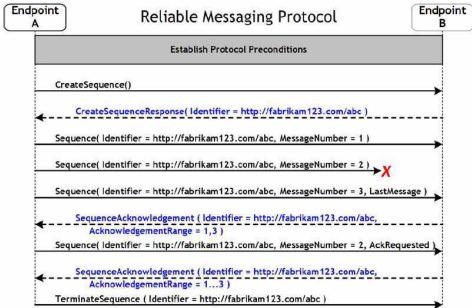
 - ☐ jobban támogatott

 - ☐ figyelembe veszi a többi protokollt is

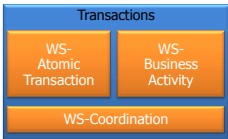
 - pl. transactions, security, ...



WS-ReliableMessaging

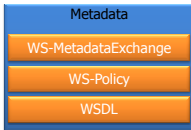


Transactions



- **WS-Coordination:**
 - tranzakció levezénylése
- **WS-AtomicTransaction:**
 - rövid lejáratú tranzakciók
 - 2PC
- **WS-BusinessActivity:**
 - hosszú lejáratú tranzakciók
 - rollback: kompenzáció

Metadata



- **WS-Policy:**
 - egy szolgáltatás képességeit és követelményeit írja le
 - kibővíti a WSDL leírást
 - példák:
 - WS-Security Policy
 - WS-ReliableMessaging Policy
 - WS-AtomicTransaction Policy
- **WS-MetadataExchange:**
 - WSDL dokumentumok felderítése
 - Policy-információk cseréje
 - dinamikus protokolfelismeréshez

WS-* szabványok összesítés

Security

WS-Federation

WS-Trust

WS-SecureConversation

WS-Security

Reliable Messaging

WS-Reliability

WS-ReliableMessaging

Transactions

WS-Atomic
Transaction

WS-Business
Activity

WS-Coordination

Metadata

WS-Metadata
Exchange

WS-Policy

WSDL

Messaging

WS-Transfer

WS-Enumeration

WS-EventNotification

MTOM

WS-Addressing

SOAP

XML

XML Encryption

XML Digital Signature

XML

XML Schema

XML Namespaces

Transport

HTTP

HTTPS

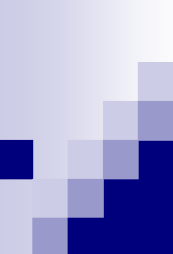
SMTP

TCP

...



DEMO



Objektumorientált szoftvertervezés

REpresentational State Transfer
(REST)

Simon Balázs, BME IIT, 2011.

Tartalom

- HTTP
- REST
- REST kritikák
- JAX-RS
- Elosztott technológiák



HTTP

HTTP GET



Request:

GET /index.html HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0
Connection: keep-alive

Response:

HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 10200

<!doctype html><html><head>...

HTTP GET:

http://www.abc.com/login?user=xy&pass=123

HTTP Method

Local URL

Version

GET /login?user=xy&pass=123 HTTP/1.1

Host: www.abc.com

User-Agent: Mozilla/5.0

Connection: keep-alive

Headers+
Empty line

Host name

Query params

HTTP POST:

http://www.abc.com/login

HTTP Method

Local URL

Version

POST /login HTTP/1.1

Host: www.abc.com

User-Agent: Mozilla/5.0

Content-Type: application/x-www-form-urlencoded

Content-Length: 16

user=xy&pass=123

Headers+
Empty line

Host name

HTTP Body:
Post parameters



REST

REST

- REpresentational State Transfer
- RESTful HTTP
- HTTP protokoll kibővítése
 - GET, POST, PUT, DELETE
- Bemenő paraméterek:
 - URL része
 - URL query string
 - POST paraméter
 - HTTP body
- Visszatérési érték:
 - HTTP body
- Nagyon egyszerű: böngészőből is tesztelhető

REST alapelvei

- Minden erőforráshoz azonosító rendelése
- Dolgok összekapcsolása
- CRUD műveletek használata
- Többféle adatrepresentáció
- Állapotmentes kommunikáció

Erőforrások azonosítóival

■ URI: Universal Resource Identifier

- mindennek az alapja (URN, URL)

■ URN: Universal Resource Name

- olyan URI, amely nem tartalmaz helyinformációt
- pl. `urn:isbn:0307346617`
- előny: soha nem romlanak el
- hátrány: nem tartalmaznak információt arról, hogyan kell feloldani őket

■ URL: Universal Resource Locator

- olyan URI, amely helyinformációt is tartalmaz
- hátrány: könnyen elromolhatnak, főleg, ha utalnak a mögöttes technológiára, pl.
`http://someserver.com/ActionServlet?blah=blah`
- de lehet jól is használni őket, pl.
`http://company1.com/customer/123456`

Erőforrások azonosítóval

■ Használjunk URL-t!

- az URL egy erőforrás egyértelmű azonosítója
- helyinformáció miatt könnyű feloldani
- legyen független a mögöttes technológiától

■ Erőforrások lehetnek:

- dokumentumok (blogok, hírek, stb.)
- adatok (számítás eredménye, metaadatok, stb.)
- szolgáltatások (SOAP web service, REST, stb.)
- fogalmak (emberek, szervezetek, stb.)

Dolgok összekapcsolása

- Jó URL címeket kell választani
- Ennek sok előnye van:
 - hatékonyan továbbadhatók
 - az adatok utólag betölthetők
 - analógia: C++ pointerok
 - biztonságosabb megoldás: az adatok hozzáférési jogosultságait könnyebb ellenőrizni

Erőforrás-URL-ek értelmezése

- Az URL hierarchikusnak tűnik
- Kliens:
 - nem szabad értelmeznie az URL-t
 - csak azonosítóként felhasználnia
 - úgy kell viselkednie, mint egy böngészőnek
 - a struktúra ugyanis változhat
- Nincs szükség interfészeíró nyelvre
- Az erőforrások manipulálására elegendő a négy művelet: GET, POST, PUT, DELETE
- Ez a „Hypertext As the Engine of State Transfer” (HATEOS) elv

Műveletek erőforrásokon

- CRUD: create, read, update, delete
- Tulajdonságok (a HTTP specifikációból):
 - safe (biztonságos): a kliens olyan műveletet hajt végre, ami csak lekérdez, és nem tehető felelőssé az okozott mellékhatásokért
 - idempotent: a művelet ismételt végrehajtása ugyanazt az eredményt produkálja
- Különböző idempotens műveletek egymás után történő ismételt végrehajtása nem feltétlenül ugyanazt az eredményt adja
 - pl. olvas-töröl-olvas
- Mellékhatások nélküli műveletek ismételt végrehajtása mindig ugyanazt az eredményt adja

Műveletek erőforrásokon

	CRUD	safe	idempotent	cacheable
GET	read	igen	igen	igen
POST	create	nem	nem	nem
PUT	update/ create	nem	igen	nem
DELETE	delete	nem	igen	nem

POST: a szerver rendel hozzá azonosítót

PUT: a kliens határozza meg az azonosítót

Többféle adatrepresentáció

- HTML:
 - csak emberek számára
 - gyakran változhat a struktúra
 - számítógép számára formálisabb kell (pl. XML, JSON, stb.)
- Választani kell tudni a különböző reprezentációk között
- Egy lehetséges, de rossz megoldás:
 - `http://company1.com/2009/report.html`
 - `http://company1.com/2009/report.xml`
 - `http://company1.com/2009/report.xls`
- A helyes megoldás: "Accept" HTTP fejléc, pl.
 - `GET /2009/report HTTP/1.1`
 - `Host: company1.com`
 - `Accept: application/xml`
- Nem támogatott MIME típus esetén: HTTP 406 Error

Állapotmentes kommunikáció

- A REST önmagában állapotmentes
- De az alkalmazásnak lehet állapota:
 - erőforrásban tárolva (nem a memóriában)
 - kliens oldalon (üzenetcsereénél mindig átküldve)
- Előnyök:
 - skálázhatóság: nem kell session-t fenntartani
 - különböző szerverpéldányok is kiszolgálhatnak egymás utáni kéréseket
 - a szerver leállítható, újraindítható két kérés között



REST kritikák

REST kritikák

- CRUD műveleteken kívül másra nem alkalmas
- Nincs interfészleíró
- Túl sok belső részletet elárul
- Tervezési guideline-ok hiánya
- Middleware funkciók hiánya
- Nincs publish-subscribe ill. aszinkron kommunikáció

CRUD műveleteken kívül más

- Lehet másra is használni, pl.:
 - `http://example.com/sum?a=2&b=3`
- De ez csalás:
 - az URI-nek egyértelműen azonosítania kell az erőforrást
- Mégsem csalás:
 - az erőforrás a kettő és a három összege
- Kérdés: melyik HTTP metódust használjuk itt?
 - GET jó lesz: hivatkozható, cache-elhető, nincs mellékhatás, safe, idempotent
- De: sok esetben szükség van mellékhatásra. Ekkor melyik HTTP metódust használjuk?
 - célszerűen: POST
 - a szerver válasza pedig egy URI az eredményre
 - az eredmény így redirect-tel megkapható, újrahasznosítható, cache-elhető, stb.

Nincs interfészleíró

■ Interfészleíró:

- leírja a műveleteket és a paraméterek típusát
- a szemantikát nem
- tipikus használat: kliens stub, szerver skeleton generálása

■ REST:

- csak 4 fajta művelet
- adatok: tipikusan XML-ben => XSD

■ Ajánlott megoldás:

- szöveges leírás a műveletek szemantikájáról
- pl. HTML leírás jön vissza egy GET hatására

■ Ha mégis formális leírás kell:

- WADL: Web Application Description Language
- WSDL 2.0
- de ezek nem nagyon támogatottak (Java világban van rá példa, .NET-ben egyáltalán nem)

Túl sok belső részlet

- A CRUD műveleteken kívül lehet más is
 - ez tetszőlegesen bonyolult lehet, akárcsak egy SOAP vagy RPC hívás
- A REST nem azt jelenti, hogy a belső adatrepresentációt publikálni kell
 - csak a szemlélet más: műveletközpontú helyett adatközpontú
- Szabály: az erőforrásokat csak URI-n keresztül szabad publikálni
 - így akár még egyszerűbb is a védelmük

Tervezési guideline-ok hiánya

- Nincsenek hivatalos best-practice-ek
- Nincs szabványos módja annak, hogyan oldjuk meg a felmerülő problémákat
- Nincs ajánlás arra, hogyan transzformáljuk át a szolgáltatásainkat REST-re
- Nincs ajánlás arra, hogyan nézzen ki egy URI
- Valóban, ezek a vádak igazak, de a józan ész és a tapasztalat segít

Middleware funkciók hiánya

- Nincs tranzakciókezelés:

- valóban

- Nincs megbízható üzenetküldés

- nem tudni, hogy egy művelet sikeres volt-e

- ha HTTP 200 OK válasz jött: sikerült

- ha nem jön válasz: nem tudni

- de: az idempotens műveletek (GET, PUT, DELETE) újraküldhetők

- POST esetén azonban vigyázni kell

Publish-subscribe, aszinkronitás

- REST: kliens-szerver modell

- Publish-subscribe:

- RSS egy lehetséges megoldás
- de: itt is a kliens kezdeményez
- GET-tel, ami hatékonyan cache-elhető
- ez a notification-by-polling

- Aszinkronitás:

- ha a szerver hosszú műveletet végez
- megoldás: válaszként HTTP 202 Accepted
- lehetőségek:

- a szerver visszaadja az eredmény URI-jét, amit a kliens pollozhat
- a kliens ad egy URI-t a kérésben, ahol a szerver értesítheti őt



JAX-RS

JAX-RS

- JAX-RS: Java API for RESTful Web Services (JSR-311)
- Java osztályok leképezése REST erőforrásokra
- Java annotációk segítségével
- Fontosabb implementációk:
 - hivatalos (Sun-Oracle): Jersey
 - JBoss: RESTeasy
 - Apache CXF
 - Restlet

JAX-RS példa

■ Application servlet:

- általános servlet egy RESTful alkalmazáshoz
- megadja, hogy mely osztályok vannak REST-ként publikálva
- kell hozzá web.xml is
- mindig így néz ki:

```
@javax.ws.rs.ApplicationPath("resources")  
public class ApplicationConfig extends javax.ws.rs.core.Application {  
}
```

Számológép példa

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
```

Csúnya, de a double
nem működik:

```
@Path("calculator")
public class Calculator
{
```

```
    @GET
```

```
    @Path("add")
```

```
    @Produces("text/plain")
```

```
    public String add(
```

```
        @QueryParam("left") double left,
```

```
        @QueryParam("right") double right)
```

```
    {
```

```
        return ""+(left+right);
```

```
    }
```

```
}
```

javax.ws.rs.WebApplicationException
A message body writer for Java type,
class java.lang.Double, and MIME
media type, text/plain, was not found

<http://localhost:8080/RestApp1/resources/calculator/add?left=3.0&right=5.0>

Paraméterek

■ Paraméter annotációk:

- **@PathParam:** URI template paraméter (az URI egy darabja)
- **@QueryParam:** URI query paraméter
- **@MatrixParam:** URI mátrix paraméter
- **@FormParam:** POST paraméter
- **@CookieParam:** Cookie paraméter
- **@HeaderParam:** HTTP header paraméter

■ Támogatott típusok:

- 1. primitív típusok
- 2. olyan T típusok, amelyeknek van egy darab String paraméterrel rendelkező konstruktoruk
- 3. olyan T típusok, amelyek statikus függvényként tartalmazzák a következő szignatúrájú metódusok valamelyikét:
 - `public static T valueOf(String)`
 - `public static T fromString(String)`
- 4. `List<T>`, `Set<T>`, `SortedSet<T>`, ahol T a fenti 2-es vagy 3-as esetből kerül ki

Param példák

```
@Path("calculator")
public class Calculator {
    @POST
    @Path("add")
    public String add(@???Param("left") double left,
                     @???Param("right") double right) {...}
}
```

- `@Path("add")`, `@QueryParam`
 - `http://.../calculator/add?left=3.0&right=5.0`
- `@Path("add/{left}/{right}")`, `@PathParam`
 - `http://.../calculator/add/3.0/5.0`
- `@Path("add")`, `@MatrixParam`
 - `http://.../calculator/add;left=3.0;right=5.0`
- `@Path("add")`, `@PostParam`
 - `http://.../calculator/add`

Visszatérési érték

■ Lehetséges visszatérési értékek:

- **void, null:** üres válasz „204 No Content” státuszkóddal
- **Response:** közvetlenül írható a válasz stream
- **GenericEntity:** generikus típusok visszaadására (type erasure miatt)
- egyéb típus

■ Nem minden Java típus támogatott

- pl. String igen, Double nem
- megoldás:
 - `MessageBodyReader`
 - `MessageBodyWriter`

MessageBodyWriter

```
public interface MessageBodyWriter<T extends Object> {  
  
    public boolean isWriteable(Class<?> type, Type genericType,  
        Annotation[] annotations, MediaType mediaType);  
  
    public long getSize(T t, Class<?> type, Type genericType,  
        Annotation[] annotations, MediaType mediaType);  
  
    public void writeTo(T t, Class<?> type, Type genericType,  
        Annotation[] annotations, MediaType mediaType,  
        MultivaluedMap<String, Object> httpHeaders,  
        OutputStream entityStream)  
        throws IOException, WebApplicationException;  
}
```

MessageBodyWriter példa

```
@Produces("text/*")
@Provider
public class SimpleTypeWriter implements MessageBodyWriter
{
    @Override
    public boolean isWriteable(Class type,
                               Type genericType,
                               Annotation[] annotations,
                               MediaType mediaType)
    {
        return type.equals(Integer.class) ||
               type.equals(Double.class);
    }
}
```

...

MessageBodyWriter példa

...

```
@Override
public long getSize(Object t, Class type,
                    Type genericType,
                    Annotation[] annotations,
                    MediaType mediaType)
{
    return -1;
}
```

...

MessageBodyWriter példa

...

```
@Override
public void writeTo(Object t,
                    Class type,
                    Type genericType,
                    Annotation[] annotations,
                    MediaType mediaType,
                    MultivaluedMap httpHeaders,
                    OutputStream entityStream)
    throws IOException, WebApplicationException
{
    BufferedWriter writer =
        new BufferedWriter(
            new OutputStreamWriter(entityStream));
    writer.write(t.toString());
    writer.flush();
}
```

Számológép példa revisited

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
```

Mostmár működik

```
@Path("calculator")
public class Calculator
{
    @GET
    @Path("add")
    @Produces("text/plain")
    public double add(
        @QueryParam("left") double left,
        @QueryParam("right") double right)
    {
        return left+right;
    }
}
```

<http://localhost:8080/RestApp1/resources/calculator/add?left=3.0&right=5.0>

HTTP methods, mime-types

■ HTTP method annotációk:

- @GET, @POST, @PUT, @DELETE, @HEAD

■ HTTP content-type annotációk:

- @Consumes: milyen formátumú bemeneteket fogad az operáció
- @Produces: milyen formátumú kimeneteket képes előállítani az operáció

XML és JSON

- Bemenő paraméterek és az eredmény formátuma legtöbbször XML vagy JSON
- Jó hír: egységesen kezelhetők
- JAXB technológia:
 - Java-XML leképezés, sorosítás
 - eredetileg SOAP-típusú web szolgáltatásokhoz
 - de XML mellett mostmár JSON-ra is alkalmas

XML és JSON példa

sample/package-info.java:

```
@javax.xml.bind.annotation.XmlSchema(  
    namespace="",  
    elementFormDefault=javax.xml.bind.annotation.XmlNsForm.QUALIFIED)  
package sample;
```

Kliens oldali
JSON miatt üres!

sample/Person.java:

```
@XmlRootElement  
@XmlType  
@XmlAccessorType(XmlAccessType.FIELD)  
public class Person {  
    @XmlElement    private String name;  
    @XmlElement    private int age;  
    @XmlElement    private List<Person> friends;  
  
    public Person() {  
        this.friends = new ArrayList<Person>();  
    }  
...  
}
```

XML és JSON példa

```
@Path("person")
public class PersonService {
    @GET
    @Path("getpb")
    @Produces("application/json")
    public Person getPerson() {
        Person pb = new Person();
        pb.setName("Peter Bishop");
        pb.setAge(27);
        Person od = new Person();
        od.setName("Olivia Dunham");
        od.setAge(26);
        pb.getFriends().add(od);
        Person bod = new Person();
        bod.setName("Bolivia Dunham");
        bod.setAge(26);
        pb.getFriends().add(bod);
        return pb;
    }
}
```

XML eredmény

```
curl -v -X GET -H"Accept: application/xml"  
http://localhost:8080/RestApp1/resources/person/getpb
```

```
> GET /RestApp1/resources/person/getpb HTTP/1.1  
> User-Agent: curl/7.20.1 (i686-pc-cygwin) ...  
> Host: localhost:8080  
> Accept: application/xml  
>  
< HTTP/1.1 200 OK  
< X-Powered-By: Servlet/3.0  
< Server: GlassFish Server Open Source Edition 3.0.1  
< Content-Type: application/xml  
< Content-Length: 227  
< Date: Sun, 13 Mar 2011 12:21:17 GMT  
<  
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>  
<person><name>Peter Bishop</name><age>27</age>  
<friends><name>Olivia Dunham</name><age>26</age></friends>  
<friends><name>Bolivia Dunham</name><age>26</age></friends>  
</person>
```

JSON eredmény

```
curl -v -X GET -H"Accept: application/json"  
http://localhost:8080/RestApp1/resources/person/getpb
```

```
> GET /RestApp1/resources/person/getpb HTTP/1.1  
> User-Agent: curl/7.20.1 (i686-pc-cygwin) ...  
> Host: localhost:8080  
> Accept: application/json  
>  
< HTTP/1.1 200 OK  
< X-Powered-By: Servlet/3.0  
< Server: GlassFish Server Open Source Edition 3.0.1  
< Content-Type: application/json  
< Transfer-Encoding: chunked  
< Date: Sun, 13 Mar 2011 12:21:19 GMT  
<  
{  
  "name": "Peter Bishop", "age": "27",  
  "friends": [  
    {  
      "name": "Olivia Dunham", "age": "26"},  
    {  
      "name": "Bolivia Dunham", "age": "26"}  
  ]  
}
```

REST kliens

- Nem része a JAX-RS specifikációnak
- Mindenki máshogyan implementálja
- Jersey (Sun-Oracle):
 - csak alacsony szintű hozzáférés
- RESTeasy (JBoss):
 - van alacsony szintű hozzáférés is
 - de lehet típusosan (annotált interfész) is használni

Jersey kliens

```
import com.sun.jersey.api.client.Client;
import com.sun.jersey.api.client.WebResource;
import com.sun.jersey.api.client.config.ClientConfig;
import com.sun.jersey.api.client.config.DefaultClientConfig;
import javax.ws.rs.core.MediaType;

public class PersonClient {
    private WebResource webResource;
    private Client client;
    private static final String BASE_URI =
        "http://localhost:8080/RestApp1/resources";

    public PersonClient() {
        ClientConfig config = new DefaultClientConfig();
        client = Client.create(config);
        webResource = client.resource(BASE_URI).path("person");
    }
    public void close() {
        client.destroy();
    }
}
```

Jersey kliens

...

```
public Person getPersonPB_XML() {
    Person pb = webResource.
        path("getpb").
        accept(MediaType.APPLICATION_XML_TYPE).
        get(Person.class);

    return pb;
}

public Person getPersonPB_JSON() {
    Person pb = webResource.
        path("getpb").
        accept(MediaType.APPLICATION_JSON_TYPE).
        get(Person.class);

    return pb;
}
}
```

Jersey kliens főprogram

```
public class Main {  
    public static void main(String[] args) {  
        PersonClient pc = new PersonClient();  
        try{  
            Person pbx = pc.getPersonPB_XML();  
            System.out.println("XML: "+pbx.getName());  
            Person pbj = pc.getPersonPB_JSON();  
            System.out.println("JSON: "+pbj.getName());  
        } finally {  
            pc.close();  
        }  
    }  
}
```



Elosztott szolgáltatások megvalósítására alkalmas technológiák

Elosztott technológiák

- RMI
- CORBA
- WS
- REST

Elosztott technológiák

	RMI	CORBA	WS	REST
Elosztott	X	X	X	X
Szabványos		X	X	X
Programnyelv-független		X	X	X
Széles körű támogatás		X	X	X
Egyszerű	X			X
Egyszerű API	X		X	X
Gyors	X	X		X
Biztonság, tranzakciók			X	

Elosztott technológiák

	RMI	CORBA	WS	REST
Protokoll	RMI	IIOP	SOAP	HTTP
Interfészleíró	Java interface	IDL	WSDL	nincs/ WADL
Katalógus	JNDI	Naming Service	UDDI	-