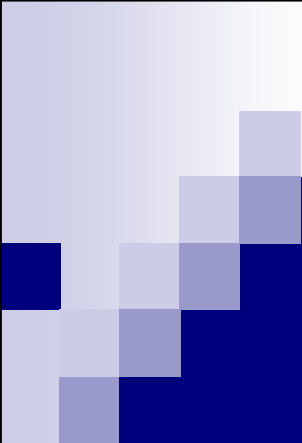




Objektumorientált szoftvertervezés

1. Java ismételés, szálak, kollekciók, reflection

© BME IIT, Goldschmidt Balázs



Az objektumorientáltság helye

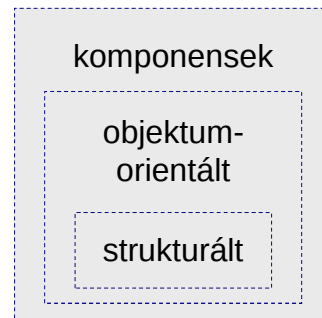
- egyszerűség
 - KISS: keep it simple, stupid
- nincs jó vagy rossz
 - használhatóság a kérdés
- mindig a célnak megfelelő eszközt használjuk

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

2

Az objektumorientáltság helye 2

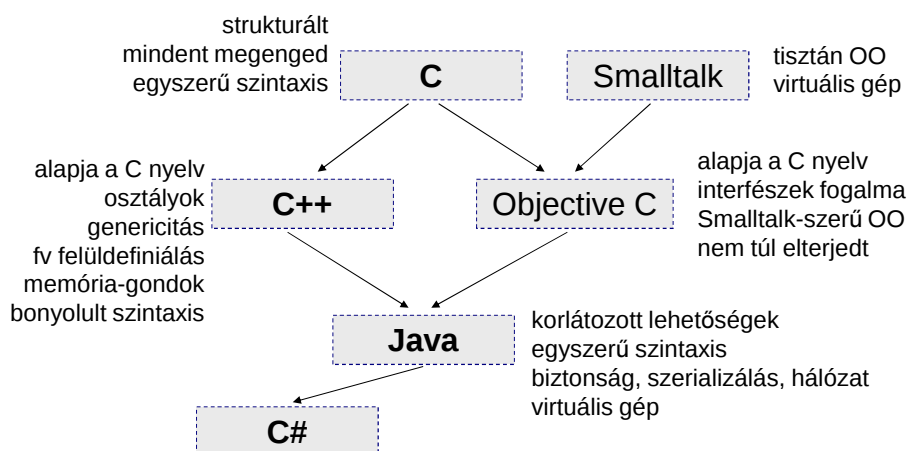
- gépi kód
 - assembly
- makrók
 - "strukturált" assembly
- strukturált programozás
 - függvények, adatszerkezetek
- objektumorientáltság
 - objektumok
- komponensek (pl. beanek)



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

3

A Java története



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

4

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

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
    }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

7

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
    }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

8

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
    }
}
```

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
    }
}
```

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  
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

11

Objektumok egyedisége

- == operátor
- **boolean equals(Object o)**
 - tartalom alapú összehasonlítás
 - rekurzió

`a == b` ↔ `a.equals(b)`

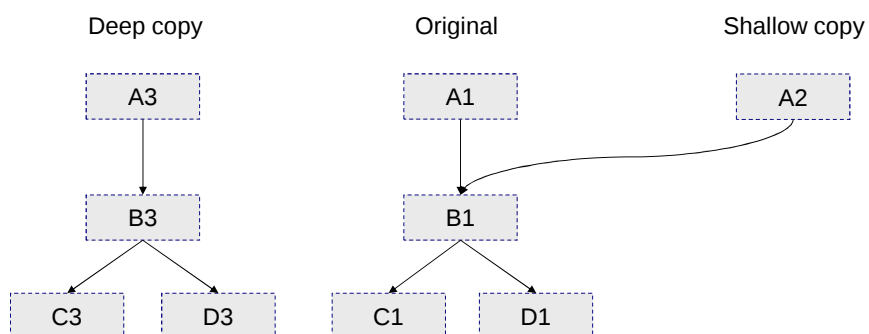
Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

12

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



Másolás öröklés nélkül

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

Másolás öröklés esetén

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

Copy constructor

```
public class A {  
    B b;  
    public A(A a) {  
        this = a.clone(); // NE!!!  
    }  
    public A(A a) {  
        this.b = a.b.clone(); // ctr vs clone  
    }  
    public A(A a) {  
        this.b = new B(a.b); // öröklés?  
    }  
    ...  
}
```

Deep clone vs. Copy ctr

	Előny	Hátrány
Deep clone	absztrakt típusokra is jó	nem new -t használ
Copy ctr	egységes, mert new	absztrakt típusoknál gond

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
}
```

Boxing megvalósítása

```
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)
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

21

Boxing kollekciókban

```
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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

22

Boxing kollektciókban folyt.

```
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);  
}
```

Boxing hibák

```
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
```

Interfészek

- Csak a metódusok deklarációja
 - nincs semmifajta implementáció
- Fontos az újrahasználáshoz
- 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*

API implementáció

Object equals és toString

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

String hash 1.1

```
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;
    }
}
```

String hash 1.5

```
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;
    }
}
```

Integer

```
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);
    }
}
```

Általános hash függvény

```
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;
    }
}
```

Memóriakezelés

Memóriakezelés

■ C már szenvedett a memóriahibáktól

- ☐ pointerek + aritmetika $a[3] \equiv *(a+3) \equiv *(3+a) \equiv 3[a]$
- ☐ void*
- ☐ malloc/calloc/realloc/free

■ C++ próbált javítani, de inkább rontott

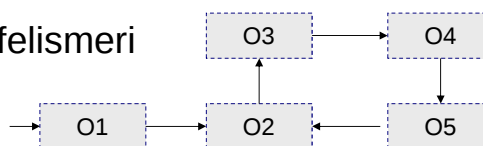
- ☐ copy konstruktor
 - ☐ virtuális destruktork
 - ☐ értékadás
 - ☐ new/delete
- ```
class C : A, virtual B {
 int l; Complex c;
public:
 C(Complex k, int i)
 : A(i), c(k), l(i)
 { l++; }
};
```

# 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



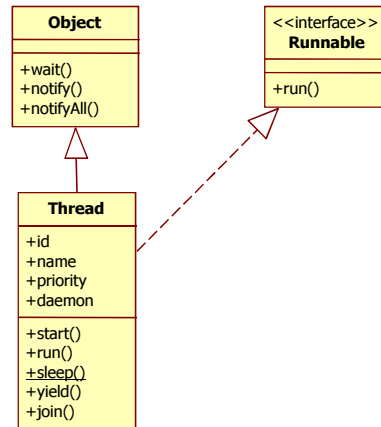
- Defragmentálást is végezhet
  - csúsztatás, rejtett kettős indirekció
  - másolás, dupla tár
- Optimalizálás is kellhet
  - [http://java.sun.com/docs/hotspot/gc5.0/gc\\_tuning\\_5.html](http://java.sun.com/docs/hotspot/gc5.0/gc_tuning_5.html)

# 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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

39

# 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 szálát
- **interrupt()**
  - megszakítja a szál várakozását
  - pl. wait, sleep, stb esetén **InterruptedException**

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

40

# java.lang.Thread stop

## ■ stop implementálása

```
private volatile boolean running;
MyThread(){ running = true; }
public void stop() { running = false; }
public void run() {
 while (running) {
 try {
 thisThread.sleep(interval);
 } catch (InterruptedException e){}
 repaint();
 }
}
```

# java.lang.Thread

## ■ 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

## ■ boolean isDaemon()

- daemon-e

# java.lang.Thread

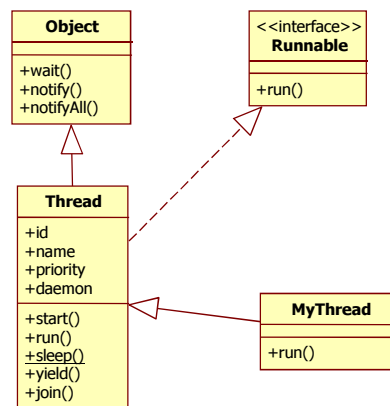
- **int getId()**
  - a szál azonosítója
- **set/getName()**
  - szál neve
- **set/getPriority()**
  - szál prioritása

# java.lang.Thread

- **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: öröklés

## ■ A Thread osztályból származtatunk



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

45

# Szál készítése: öröklés

## ■ 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); }
 }
}
```

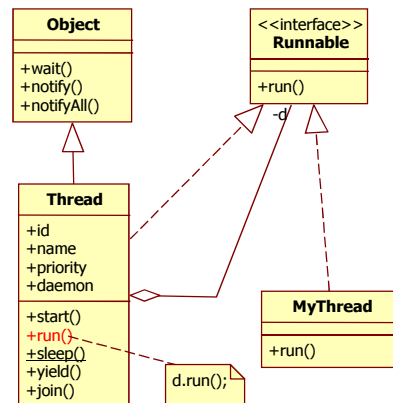
```
MyThread mt = new MyThread(1000);
mt.start();
mt.join();
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

46

## Szál készítése: delegáció

### ■ A **Runnable** interfészt valósítjuk meg



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

47

## Szál készítése: delegáció

### ■ 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();
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

48

## Tervezési kérdés ( $A \rightarrow B$ )

### ■ Öröklés

- ☐ B hozzáfér A nem privát tagjaihoz
  - közös az állapotuk
- ☐ B statikusan cserélhető

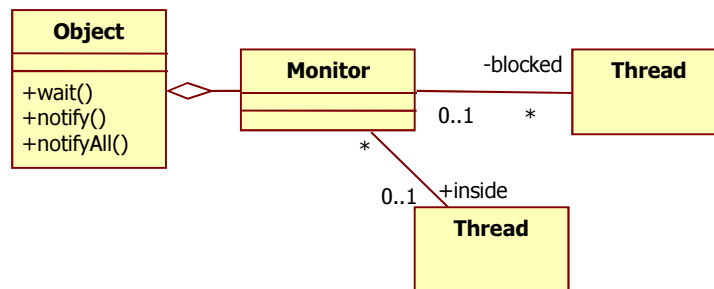
### ■ Delegáció

- ☐ B nem fér könnyen A-hoz
  - sokszor nem is ismeri
- ☐ B dinamikusan cserélhető

## 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: monitor



## Kölcsönös kizárás: *synchronized*

- 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

## ■ 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

- Minden szál egy memória-másolaton dolgozik
  - ☐ caching
- A másolatokat szinkronizálni kell
  - ☐ synchronized blokkban megtörténik
  - ☐ volatile kulcsszó
    - a változó minden módosítása és elérése szinkron

## 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

## Feladat

- Készítsünk szálbiztos Singleton-t
  - legfeljebb egy példány
  - több szálból is elérhető a létrehozás
  - minél kevesebb kölcsönös kizárással
  - sokáig tart az inicializáció

## Singleton és szálak: naívan

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }

 public static Singleton getInstance() {
 if (obj == null) {
 obj = new Singleton();
 }
 return obj;
 }
}
```

## Singleton és szálak: synch'd

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }

 public static synchronized Singleton getInstance() {
 if (obj == null) {
 obj = new Singleton();
 }
 return obj;
 }
}
```

## Singleton és szálak: synch'd 2

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }

 public static Singleton getInstance() {
 if (obj == null) {
 synchronized (Singleton.class) {
 obj = new Singleton();
 }
 }
 return obj;
 }
}
```

## Singleton és szálak: synch'd 3

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }

 public static Singleton getInstance() {
 if (obj == null) {
 synchronized (Singleton.class) {
 if (obj == null) {
 obj = new Singleton();
 }
 }
 }
 return obj;
 }
}
```

## Singleton és szálak: synch'd 4

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }

 public static Singleton getInstance() {
 if (obj == null) {
 synchronized (Singleton.class) {
 if (obj == null) {
 Singleton tmp = new Singleton();
 obj = tmp;
 }
 }
 }
 return obj;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

61

## Singleton és szálak: flaggel

```
public class Singleton {
 private static Singleton obj;
 private Singleton() { ... }
 private static boolean existing = false;
 public static Singleton getInstance() {
 if (!existing) {
 synchronized (Singleton.class) {
 if (!existing) {
 obj = new Singleton();
 existing = true;
 }
 }
 }
 return obj;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

62

## 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

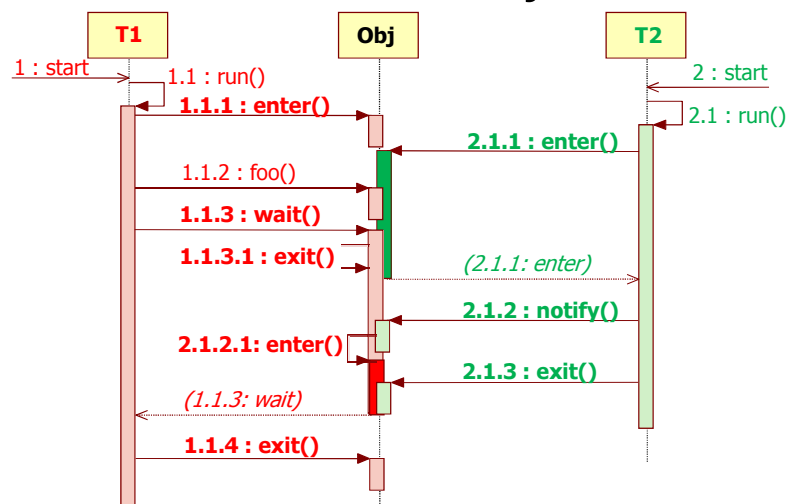
### ■ **Object.notifyAll()**

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

# Monitor és wait-notify

- `wait/notify` csak szinkronizáltan
  - a szálnak a monitorban kell lennie
  - különben *`IllegalMonitorStateException`*
- `wait` alatt kilép a monitorból
  - más szál hozzáfér az objektumhoz
  - `wait` után csak ha visszalépett

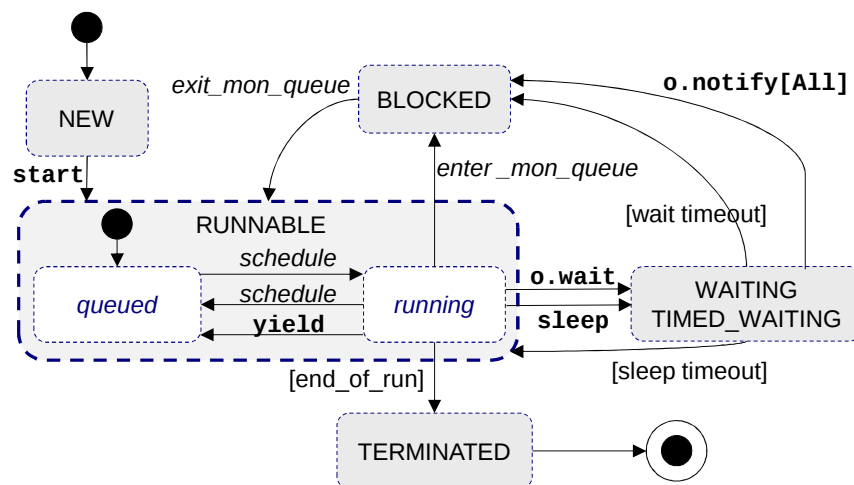
# Monitor és wait-notify



# Szálak állapotai

- **NEW**
  - újonnan létrehozva
- **RUNNABLE (+running)**
  - futásra kész
- **BLOCKED**
  - monitorra vár
- **WAITING, TIMED\_WAITING**
  - várakozó szál (*Object.wait*, *Thread.sleep*)
- **TERMINATED**
  - befejezte a működését

# Szálak állapotai



## Példa: Szemafor pszeudokód

```
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();
 }
}
```

## Példa: sorompó feladat

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

## Példa: sorompó megoldás

```
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();
 }
 }
}
```

## Vizsgapélda

- Egy programban két Java szálunk van, *t1* és *t2*.
- Mindkettő ugyanazt a Qux objektumot ismeri (*q*).
- A *t1* meghívja a *q.foo()*, majd 1 mp múlva *t2* a *q.bar()* metódusát.
- A metódusok lefutása után a szálak megállnak.
- Milyen sorrendben zajnak le a számozott sorok, és az egyes sorok futásakor milyen állapotba kerülnek a szálak?

## Vizsgapélda (folyt.)

```
public class Qux {
 synchronized
 void foo() throws InterruptedException { //1
 Thread.sleep(10000); // 10 mp-ig alszik //2
 wait(); //3
 } //4

 synchronized void bar() { //5
 notifyAll(); //6
 } //7
}
```

# 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 példa: Object

```
public class Store{
 Object[] os; int size;
 public Store(int i) {
 v = new Object[i];
 size = 0;
 }
 public void put(Object o) {
 os[size]=o; size++;
 }
 public Object get(int i) {
 return os[i];
 }
 public int size() {
 return size;
 }
}
```

```
Store s = new Store(10);

s.put("hello ");
s.put("world");
s.put("!");

for (int i = 0;
 i < s.size();
 i++) {

 String l = (String)s.get(i);
 System.out.print(l);

}
```

## Generikus osztályok

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

## Generikus példa: Template

```
public class Store<T>{
 T[] os; int size;
 public Store(int i) {
 v = (T[]) (new Object[i]);
 size = 0;
 }
 public void put(T o) {
 os[size]=o; size++;
 }
 public T get(int i) {
 return os[i];
 }
 public int size() {
 return size;
 }
}
```

```
Store<String> s =
 new Store(10);

s.put("hello ");
s.put("world");
s.put("!");

for (int i = 0;
 i < s.size();
 i++) {

 String l = s.get(i);
 System.out.print(l);

}
```

## Genericitás megvalósítása

```
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);
 }
}
```

## Genericitás megvalósítása

```
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
```

## Genericitás megvalósítása

```
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
```

## Metainformációk

### ■ Tekintsük a következő kódot

```
public Object get1(int i) {
 return os[i];
}
public T get2(int i) {
 return os[i];
}
```

```
String a,b;

a = store.get1(0);
b = store.get2(0);
```

### ■ Honnan tudja a fordító, mikor kell *cast*?

- ☐ template-típus tárolva van (T)

## 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

## Gondok az örökléssel (kollekció)

```
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 leszármazottja!**
- **Inkompatibilis a típusuk**

## 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

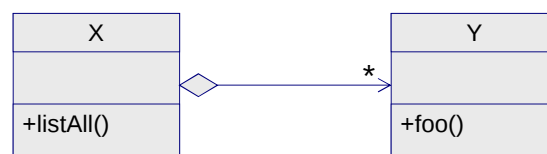
- Mi van akkor, ha a generikus típus egyszerre kell, hogy X és Y interfészű legyen?

```
<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.

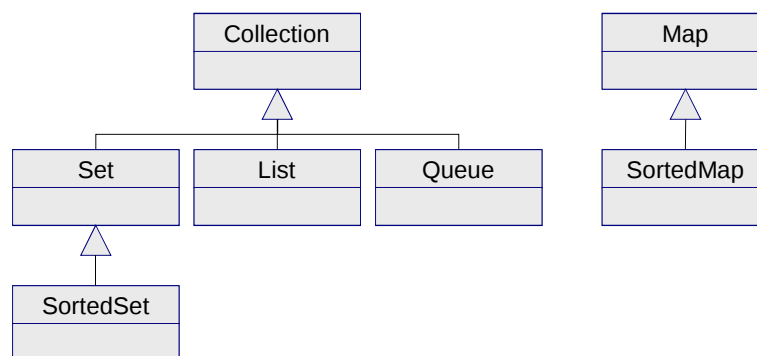
- Láncolt lista
  - egy- vagy kétirányú, gyűrű, strázsa, fésűs lista
- Bináris (n-es fa) fa
  - egyensúly, AVL fa
- Szófa
- Hash-tábla
  - hash függvény jósága

## 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*
- **void addAll(Collection<? extends E> c)** *optional*
  - hozzáadás, shallow copy
- **boolean remove(E e)** *optional*
- **boolean removeAll(Collection<? extends E> c)** *optional*
  - csak a referenciák törölődnek, nincs destruktorhívás!
- **boolean retainAll(Collection<? extends E> c)** *optional*
  - csak c-ben is szereplő objektumokat tartja meg

## Collection interfész 3

- **boolean contains(E e)** *optional*
- **boolean containsAll(Collection<? extends E> c)** *optional*
  - tartalmazás
- **void clear()**
  - összes objektum referenciájának törlése
- **int size()**
- **boolean isEmpty()**
  - méret, üresség

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

103

## Collection interfész 4

- **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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

104

# 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

105

# 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
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

106

## 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 döntik el, hogy mi a sorrend
  - természetes vagy **Comparator** minta
  - (*Strategy*)
- Az iterátor az adott sorrendben adja vissza az elemeket
- Tipikus implementáció: **TreeSet**

## Összehasonlítás

- Természetes
  - megvalósítja a `Comparable<T>` interfészt
  - `int compareTo(T o)`
    - `this`  $\Delta$  `o`  $\leftrightarrow$  `this.compareTo(o)`  $\Delta$  `0`
  - csak egyféleképpen működik
  - fordítási időben eldől
    - trükközni lehet 😊

# Összehasonlítás

## ■ Comparatorral

- *Strategy* minta: kivesszük *T* kezéből
- `interface Comparator<T>`
- `int compare(T o1, T o2)`
  - *T*-ket hasonlít
- `o1  $\Delta$  o2  $\leftrightarrow$  cmp(o1,o2)  $\Delta$  0`
- `boolean equals(Object obj)`
  - comparatorokat hasonlít

# 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
- Kétfajta 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()**
  - visszacja, 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 kulcsokat
- **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 kollekció

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

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

## For ciklus kollekció 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 szálbiztosak
- Speciális kollekciók a korrekt konkurenciához
  - `java.util.concurrent` csomagban
    - `ConcurrentMap<K, V>`
    - `CopyOnWriteArrayList<E>`
    - `Semaphore`
    - ...

## Konkurens kollekció-elérés 2

- Csomagoló osztályok (decorator minta)
- `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)`

## Kollekciókezelés

- Collections utility osztály

- ☐ `emptyList (Map, Set)`, `singletonList (Map, Set)`
- ☐ `fill`, `copy`, `list (from Enumeration)`, `disjoint`
- ☐ `max`, `min`, `frequency`, `binarySearch`
- ☐ `sort`, `shuffle`, `reverse`, `rotate`
- ☐ wrapperek
  - `concurrent`, `unmodifiable`, `checked`

## Keretrendszer használata

```
List<Student> s = new ArrayList<Student>();
```

### ■ Statikus típus

- ☐ funkcionális követelmények
- ☐ minél absztraktabb legyen
- ☐ tervezésnél erre kell koncentrálni

### ■ Dinamikus típus

- ☐ nem funkcionális követelmények
- ☐ akár futási időben cserélhető
- ☐ megvalósításnál erre kell koncentrálni

## *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()**

- ☐ ősosztály 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

137

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

138

## 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*

## 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

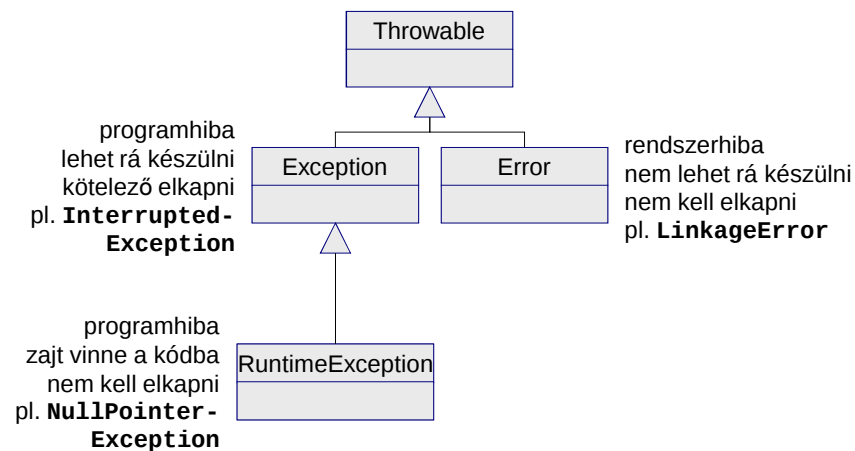
```
void foo(String s) throws IOException {
 ...
}
```

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

147

## Kivételek hierarchiája



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

148

## 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
- Kiírás 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
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

153

## Enum: valódi típus

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

154

## 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

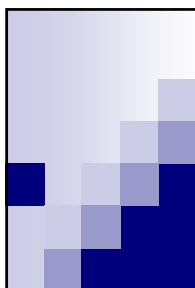
```
void foo(String s, Object[] oa) { for (Object o : oa) ...; }
```

- 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");
```



# Objektumorientált szoftvertervezés

## 3. Perzisztencia

© BME IIT, László Zoltán, 2012

---

---

---

---

---

---

---

---



## 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)*

Objektumorientált SW-tervezés © BME IIT, László Zoltán

---

---

---

---

---

---

---

---



## Bevezetés

- **Objektumok**
  - Viselkedés
  - Szerkezet
  - Állapot
  - Egyediség
- **Élettartam**
  - Automatikus változók: blokk
  - Attribútumok: objektum
  - Osztályváltozók: program futása

Objektumorientált SW-tervezés © BME IIT, László Zoltán

---

---

---

---

---

---

---

---

## Bevezetés

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

4

## Bevezetés

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

5

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

6

## Bevezetés

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

7

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

8

---

---

---

---

---

---

---

---

## *Egy egyszerű feladat*

Objektumorientált SW-tervezés © BME IIT, László Zoltán

9

---

---

---

---

---

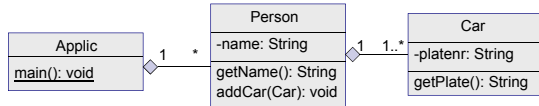
---

---

---

## Feladatdefiníció

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



Objektumorientált SW-tervezés © BME IIT, László Zoltán

10

## Tranziens megoldás

Objektumorientált SW-tervezés © BME IIT, László Zoltán

11

## Szerializálás

Objektumorientált SW-tervezés © BME IIT, László Zoltán

12

## 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*

Objektumorientált SW-tervezés © BME IIT, László Zoltán

13

---

---

---

---

---

---

---

---

## 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) { ... }
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

14

---

---

---

---

---

---

---

---

## 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 !

Objektumorientált SW-tervezés © BME IIT, László Zoltán

15

---

---

---

---

---

---

---

---

## 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**

Objektumorientált SW-tervezés © BME IIT, László Zoltán

16

## Szabályok, folyamat

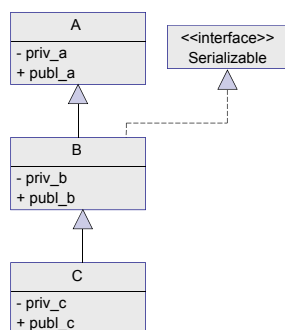
- `writeObject` – tranzitív
  - az első előfordulás tárolt, a többi csak platformfüggetlen azonosító
  - ismételt kiírás – NEM felülírás
 

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

17

## Öröklés



Objektumorientált SW-tervezés © BME IIT, László Zoltán

18

## 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");
}

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

19

---

---

---

---

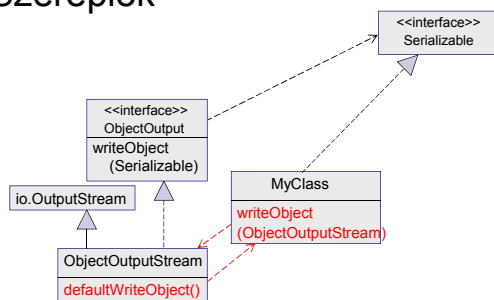
---

---

---

---

## Szereplők



Objektumorientált SW-tervezés © BME IIT, László Zoltán

20

---

---

---

---

---

---

---

---

## Saját szerializálás

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

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

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

21

---

---

---

---

---

---

---

---

## 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;
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

22

---

---

---

---

---

---

---

---

## ObjectInput

```
public interface ObjectInput extends DataInput
{
 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;
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

23

---

---

---

---

---

---

---

---

## Serializálható mezők

### ■ serialPersistentFields

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

24

---

---

---

---

---

---

---

---

## 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);
}
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

25

---

---

---

---

---

---

---

---

## Helyettesítés

### ■ Interceptor – serializálás közben

- visszat egy objektumot,
  - amiből
  - amibe serializálunk

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

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

26

---

---

---

---

---

---

---

---

## 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;
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

27

---

---

---

---

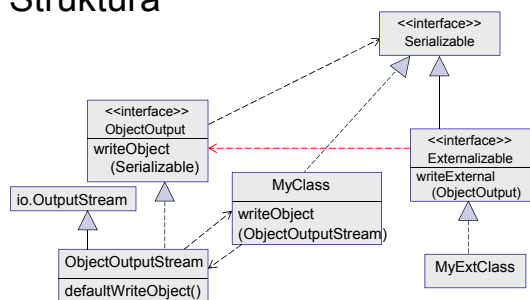
---

---

---

---

## Struktúra



Objektumorientált SW-tervezés © BME IIT, László Zoltán

28

## Verziók

### ■ Kompatibilis

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

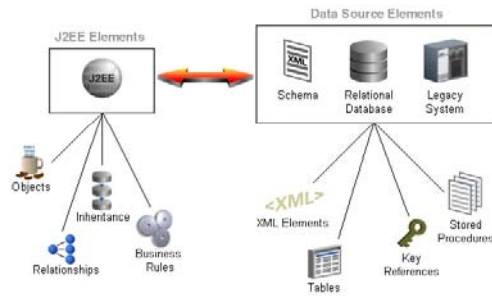
29

## Objektum relációs adatbázis

Objektumorientált SW-tervezés © BME IIT, László Zoltán

30

## Probléma lényege



Objektumorientált SW-tervezés © BME IIT, László Zoltán

31

## Hibernate 3.0

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

32

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

33

## Jellemzők

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

34

---

---

---

---

---

---

---

---

## 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)*

Objektumorientált SW-tervezés © BME IIT, László Zoltán

35

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

36

---

---

---

---

---

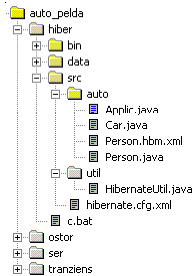
---

---

---

## A feladat megoldása

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



Objektumorientált SW-tervezés © BME IIT, László Zoltán

37

## Hibernate 3.0 definiálása

- Leképezés
  - ☐ Tag-ek, típusok, kollekciók
  - ☐ Asszociáció, komponens
  - ☐ Öröklés
- Működés
  - ☐ Architektúra, objektumok kezelése
  - ☐ Tranzakciók, konkurencia
  - ☐ HQL

Objektumorientált SW-tervezés © BME IIT, László Zoltán

38

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

39

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

40

---

---

---

---

---

---

---

---

## Leképezések / tag-ek 3

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

41

---

---

---

---

---

---

---

---

## Leképezések / típusok 1

- **Entitás**
  - egyed, **függetlenül**, hogy van-e rá referencia
  - explicit kell menteni és törölni (cascade !)
  - entitás referenciákból és Value-kból áll
- **Value**
  - primitív, kollekció, komponens, spec. obj.
  - komponens: saját osztály + value szemantika
  - mentés, törlés a tulajdonossal együtt

Objektumorientált SW-tervezés © BME IIT, László Zoltán

42

---

---

---

---

---

---

---

---

## Leképezések / típusok 2

### ■ Java – Hibernate leképezés

Entitás → `<class>`

Value → `<component>` `<property>`

Objektumorientált SW-tervezés © BME IIT, László Zoltán

43

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

44

## Leképezések / kollekciók 1

### ■ Kollektciók **interface** típusúak

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

### ■ Hibernate-nek saját implementációja van.

### ■ Tilos a konkrét típusra cast-olni !

### ■ Kollektció entitás komponense. Magában nem áll.

Objektumorientált SW-tervezés © BME IIT, László Zoltán

45

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

46

## Leképezések / kollekciók 3

### ■ Kollektók elemei

#### □ Value-k

- <element> Or <composite-element>

#### □ entitások referenciái

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

### ■ Példák

Objektumorientált SW-tervezés © BME IIT, László Zoltán

47

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

48

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

49

## 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)
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

50

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

51

## 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)
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

52

---

---

---

---

---

---

---

---

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

53

---

---

---

---

---

---

---

---

## 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)
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

54

---

---

---

---

---

---

---

---

## 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

55

## Leképezések / komponens

### ■ Érték, nem entitás referencia

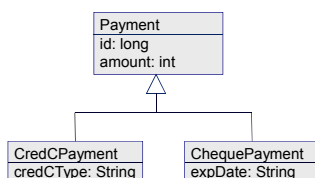
```
public class Person {
 private Name name;...}

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

56

## Leképezések / öröklés 1



Objektumorientált SW-tervezés © BME IIT, László Zoltán

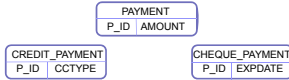
57

## Leképezések / öröklés 2

### ■ table-per-class-hierarchy



### ■ table-per-subclass



### ■ table-per-concrete-class



Objektumorientált SW-tervezés © BME IIT, László Zoltán

58

## 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>

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

59

## 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>

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

60

## Leképezések / öröklés 5

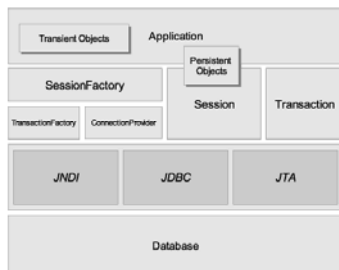
### ■ table-per-concrete-class

```
<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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

61

## Működés / Architektúra



Objektumorientált SW-tervezés © BME IIT, László Zoltán

62

## Architektúra

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

63

## Objektum állapotok

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

64

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

65

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

66

## 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

67

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

68

---

---

---

---

---

---

---

---

## 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()

Objektumorientált SW-tervezés © BME IIT, László Zoltán

69

---

---

---

---

---

---

---

---

## Objektumok kezelése 3

- Perzisztens objektumok módosítása
  - ☐ táblába felír: `session.flush()`
- Lepakcsolt 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)`

Objektumorientált SW-tervezés © BME IIT, László Zoltán

70

## 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();
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

71

## 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



Objektumorientált SW-tervezés © BME IIT, László Zoltán

72

## Tranzakciók 2

- Optimista zárolás
  - Alkalmazás szinten
    - Új session, új verzió – új vs. régi ?
  - Hosszú session, automata verzió
  - Lepakcsolt objektumok, automata verzió
- Pesszimista zárolás - haladóknak

Objektumorientált SW-tervezés © BME IIT, László Zoltán

73

---

---

---

---

---

---

---

---

## 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");
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

74

---

---

---

---

---

---

---

---

## 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);
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

75

---

---

---

---

---

---

---

---

## HQL 1

- Hibernate Query Language

- From

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

- Join

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

76

## HQL 2

- Select

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

- Aggregáló funkciók

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

- Where – kifejezések

- order by, group by

Objektumorientált SW-tervezés © BME IIT, László Zoltán

77

## 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();
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

78

## Java Persistence API

Objektumorientált SW-tervezés © BME IIT, László Zoltán

79

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

80

## Entitások

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

81

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

82

---

---

---

---

---

---

---

---

## Persistence Unit

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

83

---

---

---

---

---

---

---

---

## 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.

Objektumorientált SW-tervezés © BME IIT, László Zoltán

84

---

---

---

---

---

---

---

---

## 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)

Objektumorientált SW-tervezés © BME IIT, László Zoltán

85

## A feladat megoldása

### ■ Derby adatbáziskezelő

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

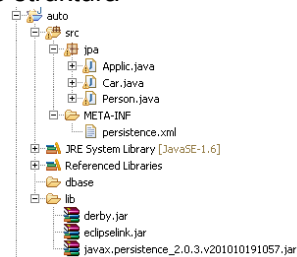
- nyílt forráskódú
- relációs adatbáziskezelő
- tiszta Java
- Apache License, Version 2.0
- csak 2.6 MB
- beágyazott és kliens/szerver mód
- egyszerű install

Objektumorientált SW-tervezés © BME IIT, László Zoltán

86

## A feladat megoldása

### ■ Eclipse struktúra



Objektumorientált SW-tervezés © BME IIT, László Zoltán

87

## JPA részletesebben

- Entitások
- Relációk
- Lekérdezések

Objektumorientált SW-tervezés © BME IIT, László Zoltán

88

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

89

---

---

---

---

---

---

---

---

## Entitások

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

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

90

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

91

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

92

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

93

## Entitások életrciklusa

### ■ 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>
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

94

## Entitások életrciklusa

### ■ Entity Manager műveletei:

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

### ■ Transzitivitás

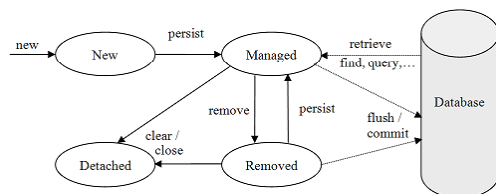
- ☐ reláció paramétere
- ☐ cascade = (műveletek) + ALL

Objektumorientált SW-tervezés © BME IIT, László Zoltán

95

## Entitások életrciklusa

### ■ Entitás állapotai



Objektumorientált SW-tervezés © BME IIT, László Zoltán

96

## Entitások egyéb műveletei

### ■ Entity Manager egyéb műveletei:

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

97

## Relációk

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

98

## 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();
 }

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

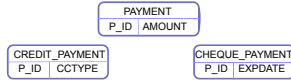
99

## Öröklés - Hibernate

### ■ table-per-class-hierarchy



### ■ table-per-subclass



### ■ table-per-concrete-class



Objektumorientált SW-tervezés © BME IIT, László Zoltán

100

## 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();

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

101

## 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){...}

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

102

## 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();
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

103

---

---

---

---

---

---

---

---

## 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();
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

104

---

---

---

---

---

---

---

---

## 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); }
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

105

---

---

---

---

---

---

---

---



## **OO adatbázisok**

Objektumorientált SW-tervezés © BME IIT, László Zoltán

106

---

---

---

---

---

---

---

---



### **Objectstore – PSEPro 1**

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

107

---

---

---

---

---

---

---

---



### **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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

108

---

---

---

---

---

---

---

---

## Alapfogalmak 1

### ■ Session

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

109

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

110

---

---

---

---

---

---

---

---

## 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)

Objektumorientált SW-tervezés © BME IIT, László Zoltán

111

---

---

---

---

---

---

---

---

## Alapfogalmak 4

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

112

---

---

---

---

---

---

---

---

## A feladat megoldása

- Perzisztens osztályok kiegészítése
  - perzisztens kollekciók
  - importálás
- Alkalmazás átalakítása
  - adatbázis adminisztráció
    - adatbázis, session, tranzakció
  - gyökerek, kollekciók
  - query-k, paraméterek

Objektumorientált SW-tervezés © BME IIT, László Zoltán

113

---

---

---

---

---

---

---

---

## PSEPro részletek

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

114

---

---

---

---

---

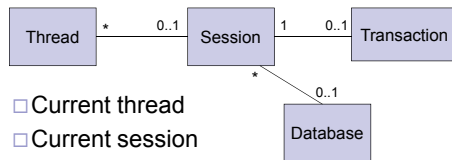
---

---

---

## Szálak és session-ök 1

### ■ Környezet és API



Objektumorientált SW-tervezés © BME IIT, László Zoltán

115

---

---

---

---

---

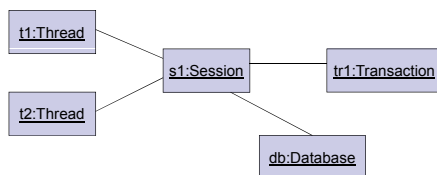
---

---

---

## Szálak és session-ök 2

### ■ Szálak osztoznak a session-ön



Objektumorientált SW-tervezés © BME IIT, László Zoltán

116

---

---

---

---

---

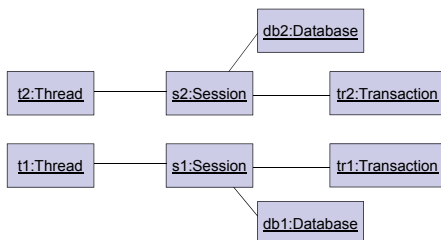
---

---

---

## Szálak és session-ök 3

### ■ Konkurens szálak – különböző adatbázis



Objektumorientált SW-tervezés © BME IIT, László Zoltán

117

---

---

---

---

---

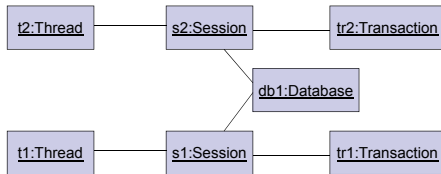
---

---

---

## Szálak és session-ök 4

- Konkurens szálak – ugyanaz az adatbázis



Objektumorientált SW-tervezés © BME IIT, László Zoltán

118

## 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 száaktól
- szál bármikor jöhet-mehet

Objektumorientált SW-tervezés © BME IIT, László Zoltán

119

## Szálak és session-ök 6

- egy időben egy tranzakció egy session-höz
- egy időben 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
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

120

## 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
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

121

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

122

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

123

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

124

## Adatbázisok 4

### ■ Személygyűjtés – 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

125

## Adatbázisok 5

### ■ Séma evolúció (verziózás)

- ☐ Perzisztens objektum módosítása

- ☐ Mikor KELL ?

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

126

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

127

## Tranzakciók 2

### ■ Zárás

#### ■ Update-re commit, RO-ra abort is jó

- Commit

- mentés, változások tárolása
- tranzitív perzisztencia
- perzisztens objektumok állapota
- lock-ok felszabadítása

Objektumorientált SW-tervezés © BME IIT, László Zoltán

128

## Tranzakciók 3

### ■ Zárás

- Commit

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

- retain: ObjectStore.

- \_STALE
- \_HOLLOW
- \_READONLY
- \_UPDATE
- \_TRANSIENT

Objektumorientált SW-tervezés © BME IIT, László Zoltán

129

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

130

## Objektumok 1

### ■ Hogyan lesz perzisztens a tranzienst ?

- alkalmazás root-tá teszi
- másik perzisztens referálja
- fizikailag felír - `objectStore.migrate()`

### ■ Memóriába betöltés

- root-ból vesszük
- external referencia
- lekérdezésből
- fizikai paraméterek alapján olvas

Objektumorientált SW-tervezés © BME IIT, László Zoltán

131

## 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");
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

132

## 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);
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

133

## Objektumok 4

### ■ External references

- session, tranzakció nélkül referál
- nem root is hivatkozható

- létrehozása (session, tranzakció)

- objektumból

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

- Stringből

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

- referált objektum

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

134

## Kollekciók

Class	Implements
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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

135

## 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);
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

136

## 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));
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

137

## Query 3

- Végrehajtás – query metódusai

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

138



## Összefoglalás

Objektumorientált SW-tervezés © BME IIT, László Zoltán139

---

---

---

---

---

---

---



## Tanulság

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán140

---

---

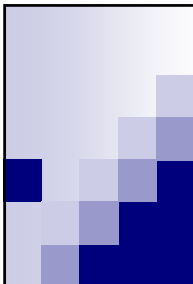
---

---

---

---

---



# Objektumorientált szoftvertervezés

## 6. OO tervezés

© BME IIT, László Zoltán

---

---

---

---

---

---

---



### 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 ?

Objektumorientált SW-tervezés © BME IIT, László Zoltán 2

---

---

---

---

---

---

---



### Tartalomjegyzék

- Tervezési elvek
- Becslés
- CDP minták
- Egyéb minták

Objektumorientált SW-tervezés © BME IIT, László Zoltán 3

---

---

---

---

---

---

---

## Tervezési elvek

Objektumorientált SW-tervezés © BME IIT, László Zoltán

4

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

5

## Csatolás (coupling)

- OO világban
  - Két osztály (package, object ?) közötti kapcsolat
- Adat-csatolás
  - A osztályból elérjük B osztály attribútumát
- Osztály csatolás
  - Függőség (dependencia)
  - Specializáció
  - Asszociáció

Objektumorientált SW-tervezés © BME IIT, László Zoltán

6

## Csatolás (coupling)

### ■ Mértékek

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

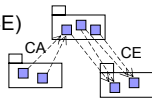
Objektumorientált SW-tervezés © BME IIT, László Zoltán

7

## 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):  $CE / (CA + CE)$



Objektumorientált SW-tervezés © BME IIT, László Zoltán

8

## 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)

Objektumorientált SW-tervezés © BME IIT, László Zoltán

9

## 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)

Objektumorientált SW-tervezés © BME IIT, László Zoltán

10

---

---

---

---

---

---

---

---

## Osztály csatolás

### ■ Specializáció

- öröklés
- interfész implementálás

### □ Mértékek

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

11

---

---

---

---

---

---

---

---

## Osztály csatolás

### ■ Asszociáció

- referencia egy másik osztályra

### □ Mértékek

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

12

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

13

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

14

## 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  $A2 \cap A3 \neq \emptyset$ .
    - $|P| = 0 < |Q| = 3$ ,  $LCOM = 0$ .

Objektumorientált SW-tervezés © BME IIT, László Zoltán

15

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

16

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

17

## Egyéb OO mértékek

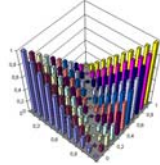
- Comment Percentage - kommentezettség
  - CP – total nbr. of comments/(LOC – BL)
- Weighted methods per class (WMC)
  - egy osztályhoz tartozó metódusok CC-je összesen
  - az osztály elkészítésének bonyolultsága ?
- Response for a class (RFC)
  - egy osztályhierarchiában az elérhető metódusok száma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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

18

## 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



Objektumorientált SW-tervezés © BME IIT, László Zoltán

19

## 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](http://www.laatuk.com/tools/metric_tools.html)
- Kereshető tool-ra  
<http://www.stickyminds.com/tools.asp>

Objektumorientált SW-tervezés © BME IIT, László Zoltán

20

## Eclipse - metrics

- Példa: lemming feladat

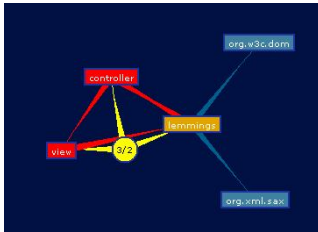
Metric	Total	Mean	Std. Dev.	Maximum	Resource loading Maximum	Method
1. Number of Abstract Methods (lengths per type)	77	0.761	0.421	3	java/lemming/StatInfo.java	
2. Number of Attributes (lengths per type)	96	1.211	0.908	9	java/lemming/Level.java	
3. Number of Children (lengths per type)	38	0.487	1.715	12	java/lemming/Proto.java	
4. Number of Classes (lengths per package/fragment)	78	26	23,338	59	java/lemming	
5. Method Lines of Code (lengths per method)	2222	4.471	6.919	62	java/lemming/LevelStore.java	createLemmingActivity
6. Number of Methods (lengths per type)	472	6.051	7.112	33	java/lemming/GameElement.java	
7. Nested Block Depth (lengths per method)	1,222	0.688	0.688	6	java/lemming/Stone.java	col
8. Depth of Inheritance Tree (lengths per type)	1,923	0.931	0.931	5	java/controller/GIApplet.java	
9. Number of Packages	3	4	4,243	10	java/lemming	
10. Effort Coupling (lengths per package/fragment)	12	4	6,657	12	java/lemming	
11. Number of Interfaces (lengths per package/fragment)	12	4	6,657	12	java/lemming	
12. McCabe Cyclomatic Complexity (lengths per method)	1,531	1.408	0.751	15	java/lemming/GameElement.java	checkCollision
13. Total Lines of Code	3651					
14. Instability (lengths per package/fragment)	0,737	0.108	0.089	5	java/lemming/StatInfo.java	StatInfo
15. Number of Parameters (lengths per method)	0,738	0.796	0.796	5	java/lemming/StatInfo.java	
16. Lack of Cohesion of Methods (lengths per type)	0,131	0.26	0.823	14	java/lemming/Level.java	
17. Effort Coupling (lengths per package/fragment)	9,667	7.04	19	java/lemming		
18. Number of Static Methods (lengths per type)	25	0.321	1.748	14	java/lemming/Legger.java	
19. Normalized Distance (lengths per package/fragment)	0,137	0.142	0.333	java/controller		
20. Abstractness (lengths per package/fragment)	0,126	0.11	0.268	java/lemming		
21. Specialization Index (lengths per type)	0,107	0.314	2	java/controller/GIApplet.java		
22. Weighted methods per Class (lengths per type)	761	9,756	13,862	76	java/lemming/GameElement.java	
23. Number of Static Attributes (lengths per type)	34	0,436	1,228	7	java/lemming/Proto.java	

Objektumorientált SW-tervezés © BME IIT, László Zoltán

21

## Eclipse - metrics

- Példa: lemming feladat – függőségi gráf



Objektumorientált SW-tervezés © BME IIT, László Zoltán

22

---

---

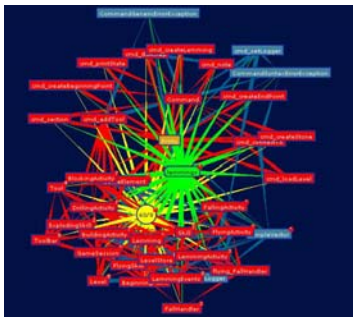
---

---

---

---

## Eclipse - metrics



Objektumorientált SW-tervezés © BME IIT, László Zoltán

23

---

---

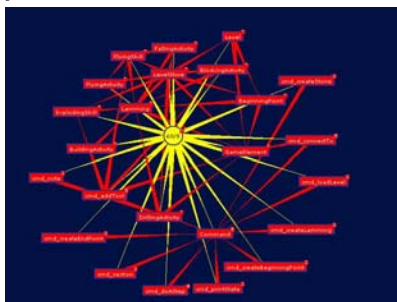
---

---

---

---

## Eclipse - metrics



Objektumorientált SW-tervezés © BME IIT, László Zoltán

24

---

---

---

---

---

---

## Becslés

Objektumorientált SW-tervezés © BME IIT, László Zoltán

25

---

---

---

---

---

---

---

---

## Cocomo 2.0

### ■ Constructive Cost Model

#### ■ Célkitűzések

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

26

---

---

---

---

---

---

---

---

## Cocomo 2.0

### ■ Célcsoportok

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

27

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

28

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

29

## Cocomo 2.0 lépései

### 2. fejlesztési idő

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

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

30

## Cocomo 2.0 tényezők

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

### ■ E – skálázási tényezők

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

31

## Cocomo 2.0 hangolás

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

### ■ EAF – hangolási tényezők (7, 17)

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

32

## 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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

33

## 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)

Objektumorientált SW-tervezés © BME IIT, László Zoltán

34

## 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](http://sunset.usc.edu/research/COCOMOII/expert_cocomo/expert_cocomo2000.html)
  - COSTAR
    - <http://www.softstarsystems.com/>

Objektumorientált SW-tervezés © BME IIT, László Zoltán

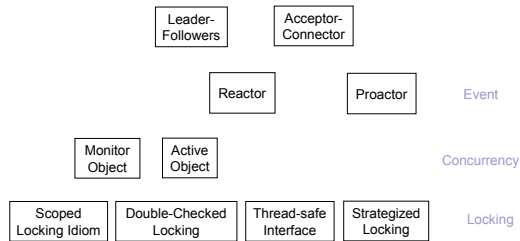
35

## *CDP minták*

Objektumorientált SW-tervezés © BME IIT, László Zoltán

36

## CDP - áttekintés



Objektumorientált SW-tervezés © BME IIT, László Zoltán

37

## 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);

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

38

## 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_;
};

```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

39

## 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

---

---

---

---

---

---

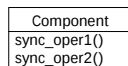
---

---

## Locking minták

### ■ Thread-safe Interface

- **probléma:** komponensen belülről és kívülről is egyaránt használható legyen. Cél
  - holtpont elkerülése
  - hatékonyság



Objektumorientált SW-tervezés © BME IIT, László Zoltán

43

---

---

---

---

---

---

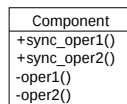
---

---

## Locking minták

### ■ Thread-safe Interface

- **megoldás:** két interfész
  - külső interfész ellenőriz
  - implementáció privát és elfogad



Objektumorientált SW-tervezés © BME IIT, László Zoltán

44

---

---

---

---

---

---

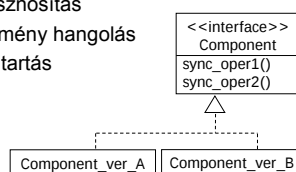
---

---

## Locking minták

### ■ Strategized Locking

- **probléma:** a lock-olás beégetése a programba akadály:
  - újrahaznosítás
  - teljesítmény hangolás
  - karbantartás



Objektumorientált SW-tervezés © BME IIT, László Zoltán

45

---

---

---

---

---

---

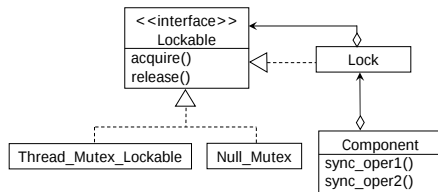
---

---

## Locking minták

### ■ Strategized Locking

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



Objektumorientált SW-tervezés © BME IIT, László Zoltán

46

## Konkurencia minták

### ■ Monitor Object

- **probléma:** több kliens szál párhuzamosan
- **megoldás:**
  - Minden Java object implicit Monitor
  - Synchronized metódus

Objektumorientált SW-tervezés © BME IIT, László Zoltán

47

## Konkurencia minták

### ■ Aktív objektum

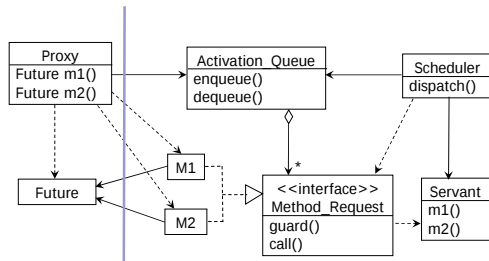
- **probléma:** konkurrens kérések kezelése
  - "guarded" vagy "concurrent" szemantika szerint
  - nem blokkoló metódushívás
- **megoldás:**
  - a metódus meghívása és végrehajtásának szétcsatolása
  - végrehajtás saját szálon

Objektumorientált SW-tervezés © BME IIT, László Zoltán

48

## Konkurencia minták

### ■ Aktív objektum

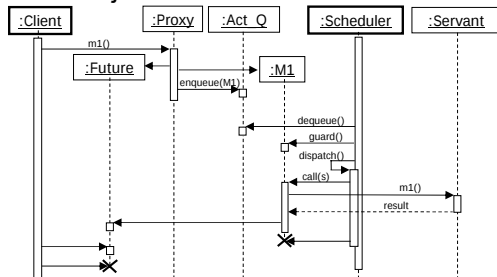


Objektumorientált SW-tervezés © BME IIT, László Zoltán

49

## Konkurencia minták

### ■ Aktív objektum



Objektumorientált SW-tervezés © BME IIT, László Zoltán

50

## Esemény minták

### ■ Reactor

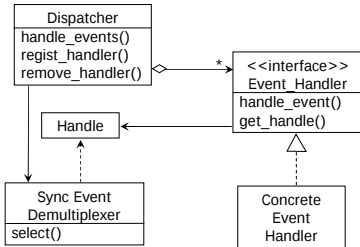
- **probléma:** szerver alkalmazás több klienst szolgál ki
- **megoldás:** integrálni
  - a szinkron esemény-demultiplexelést
  - az event handler-ek kezelését

Objektumorientált SW-tervezés © BME IIT, László Zoltán

51

## Esemény minták

### ■ Reactor

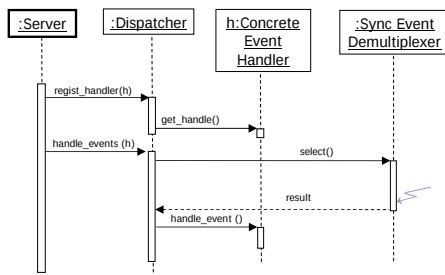


Objektumorientált SW-tervezés © BME IIT, László Zoltán

52

## Esemény minták

### ■ Reactor



Objektumorientált SW-tervezés © BME IIT, László Zoltán

53

## CDP minták

### ■ Vezető-követők (leader-followers)

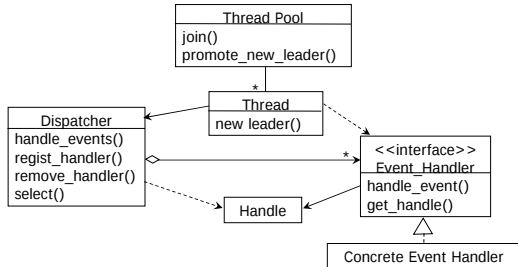
- **probléma:** több eseményt konkurens módon kezelő alkalmazás implementálása
- **megoldás:**
  - szálakból egy csokrot készítünk
  - az eseményt demultiplexeljük
  - szinkron lekezeljük

Objektumorientált SW-tervezés © BME IIT, László Zoltán

54

## CDP minták

### ■ Vezető-követők (leader-followers)

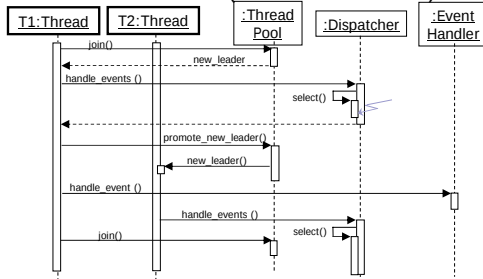


Objektumorientált SW-tervezés © BME IIT, László Zoltán

55

## CDP minták

### ■ Vezető-követők (leader-followers)

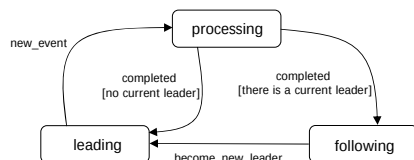


Objektumorientált SW-tervezés © BME IIT, László Zoltán

56

## CDP minták

### ■ Vezető-követők (leader-followers)



Objektumorientált SW-tervezés © BME IIT, László Zoltán

57

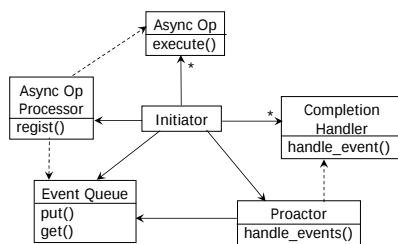
## Esemény minták

- Proactor

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

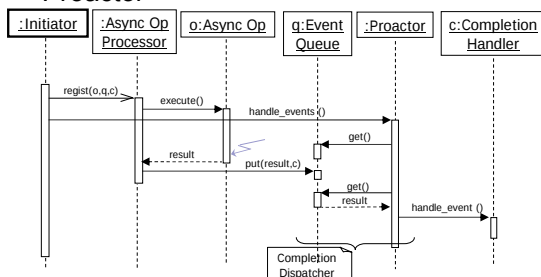
## Esemény minták

- Proactor



## Esemény minták

- Proactor



## CDP minták

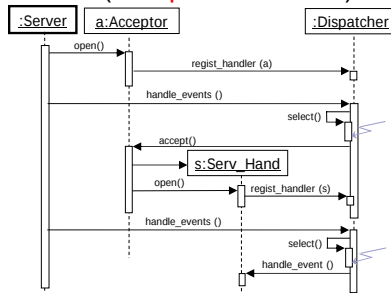
- Csatlakozó (acceptor-connector)
  - **probléma**: összefonódik a kapcsolódó és a kommunikációs szerep
  - **megoldás**: szétválasztani
    - a kapcsolódást és inicializálást
    - a szolgáltatás nyújtását

Objektumorientált SW-tervezés © BME IIT, László Zoltán

61

## CDP minták

- Csatlakozó (**acceptor**-connector)

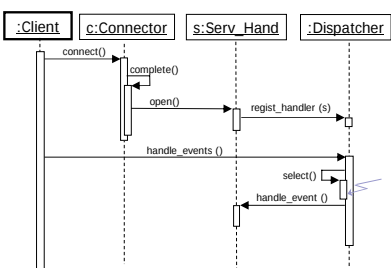


Objektumorientált SW-tervezés © BME IIT, László Zoltán

62

## CDP minták

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

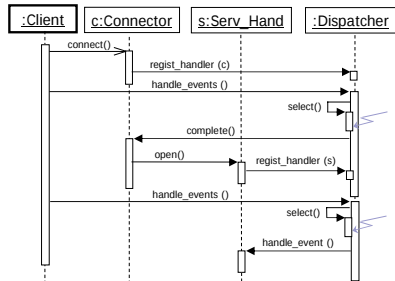


Objektumorientált SW-tervezés © BME IIT, László Zoltán

63

## CDP minták

### ■ Csatlakozó (acceptor-connector<sup>(aszinkron)</sup>)



Objektumorientált SW-tervezés © BME IIT, László Zoltán

64

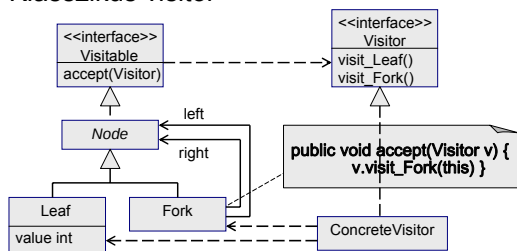
## Egyéb minták

Objektumorientált SW-tervezés © BME IIT, László Zoltán

65

## Visitor Combinator

### ■ Klasszikus visitor



Objektumorientált SW-tervezés © BME IIT, László Zoltán

66

## Visitor Combinator

### ■ Klasszikus visitor problémák

- Ki vezérli a bejárást ?
  - Külső program ?
  - Accept ?
  - Visitor ?
- Újrahasználatosság ?
  - Öröklés – specializálás
  - Több helyről - többszörös öröklés ????

### ■ Megoldás: Visitor Combinator

Objektumorientált SW-tervezés © BME IIT, László Zoltán

67

## 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

68

## 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); }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

69

## 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()); }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

70

---

---

---

---

---

---

---

---

## 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();}
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

71

---

---

---

---

---

---

---

---

## 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); }
 }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

72

---

---

---

---

---

---

---

---

## 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()); }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

73

---

---

---

---

---

---

---

---

## Visitor Combinator

### ■ Kombinátor algebra

Sequence(Identity, v) = v  
 Sequence(v, Identity) = v  
 Sequence(Fail, v) = Fail  
 Sequence(v, Fail) = Fail (ha v-nek nincs mellékhatása)  
 Choice(Fail, v) = v  
 Choice(v, Fail) = v  
 Choice(Identity, v) = Identity

Try(v) = Choice(v, Identity)  
 IfZeroAddOne = Try(Sequence(IsZero, AddOne))

Objektumorientált SW-tervezés © BME IIT, László Zoltán

74

---

---

---

---

---

---

---

---

## 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);
 }
}
```

Objektumorientált SW-tervezés © BME IIT, László Zoltán

75

---

---

---

---

---

---

---

---

## Visitor Combinator

### ■ Bejáró kombinátorok

TopDown(v) = Sequence(v, All(TopDown(v)))

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

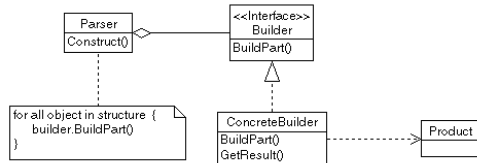
BottomUp(v) = Sequence(All(BottomUp(v)), v)

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

Objektumorientált SW-tervezés © BME IIT, László Zoltán

76

## Builder minta



Objektumorientált SW-tervezés © BME IIT, László Zoltán

77

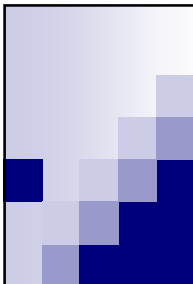
## Builder minta

```
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 SW-tervezés © BME IIT, László Zoltán

78



# Objektumorientált szoftvertervezés

## Ablakkezelés, SWING

© BME IIT, Goldschmidt Balázs

---

---

---

---

---

---

---



## Ablakkezelés Java-ban

- AWT
  - Abstract Windowing Toolkit
  - natív ablakkezelés és widgetek
- SWING
  - Java Foundation Classes
  - light-weight widgetek
- SWT, GWT stb
  - SWT: Standard Widget Toolkit
    - Eclipse és barátai
  - GWT: Google Web toolkit

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

---

---

---

---

---

---

---



## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

---

---

---

---

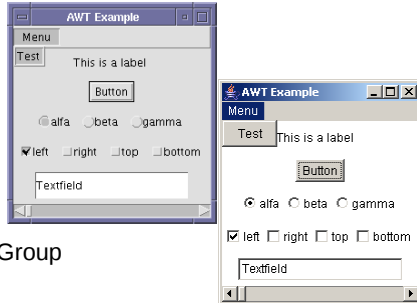
---

---

---

## Alap AWT widgetek

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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

4

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

5

---

---

---

---

---

---

---

---

## Konténer felelőssége

- Komponensek megtalálása
  - adott pozíción levő
  - adott sorszámu
  - lista az összesről
- A fókusz továbbadása
  - ki az aki a billentyűzetről jövő információt kezeli
- A komponensek elhelyezése
  - az alkalmazott algoritmus a *LayoutManager*-ben van implementálva (*Strategy pattern*)
  - rekurzív méretszámítás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

6

---

---

---

---

---

---

---

---

# 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

10

---

---

---

---

---

---

---

---

## Eseménykezelés 1.0 példa

```
public class MyButton extends Button {

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

}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

11

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

12

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

19

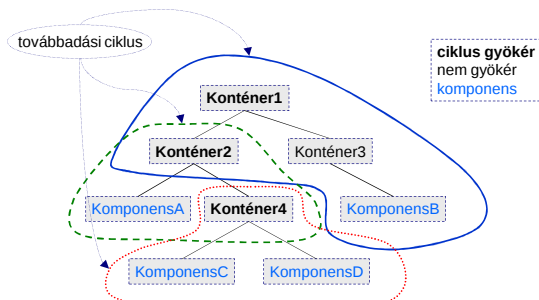
## Fókusz alapfogalmak 3

- 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

20

## Fókusz alapfogalmak 4



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

21

## 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 . redispachEvent( java.awt.Component , java.awt.AWTEvent )**
- A **KeyboardFocusManager**-nél kell regisztrálni

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

22

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

23

---

---

---

---

---

---

---

---

## 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 fókusz
- **FocusEvent**
  - FOCUS\_GAINED / FOCUS\_LOST
    - **Component** kapja, ha megkapja vagy elveszti a fókusz

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

24

---

---

---

---

---

---

---

---

## Fókuszváltás-események 2

- Sorrendiség
  - ha nem a Java alkalmazásé a fókusz
  - **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 fókuszot
  - 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 manager*ek 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)**

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

34

---

---

---

---

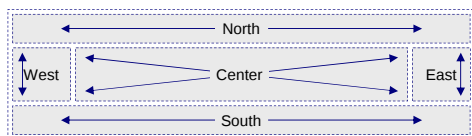
---

---

---

---

## BorderLayout



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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

35

---

---

---

---

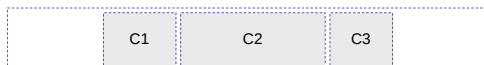
---

---

---

---

## 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**

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

36

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

37

---

---

---

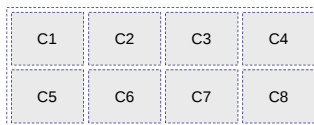
---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

38

---

---

---

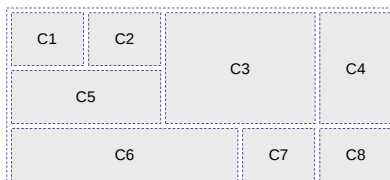
---

---

---

---

## GridBagLayout



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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

39

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

40

---

---

---

---

---

---

---

---

## 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)

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

41

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

42

---

---

---

---

---

---

---

---

## 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\*

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

43

---

---

---

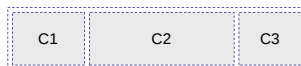
---

---

---

---

## BoxLayout (swing)



- Komponensek 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

44

---

---

---

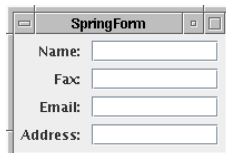
---

---

---

---

## SpringLayout (swing)



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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

45

---

---

---

---

---

---

---

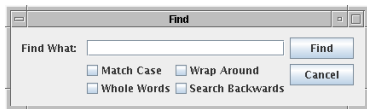
## SpringLayout (swing) 2

```
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
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

46

## GroupLayout (swing)

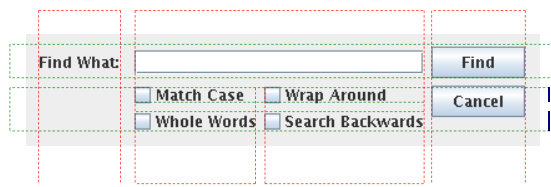


- Horizontális és vertikális dimenziók független kezelése
  - minden komponens kétszer
- Hierarchikus: csoportok egymásba ágyazva
  - Szekvenciális és paralel: egymás mellé vagy fölé
  - Builder minta alkalmazásával
- Komponens cseréje:
  - `void replace(Component oldc, Component newc)`

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

47

## GroupLayout (swing) 2



--- vertikális  
--- horizontális

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

48

## 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))
);
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

49

---

---

---

---

---

---

---

---

## 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))
);
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

50

---

---

---

---

---

---

---

---

## GroupLayout gyakorlat

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

51

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

55

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

56

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

57

---

---

---

---

---

---

---

---

## 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 leszármazottjai
- ☐ **JComponent** a **Container** leszármazottja

### ■ A kirajzolás a Java 2D API-n alapul

### ■ Kerülni kell a keveredést

- ☐ elfedhetik egymást a komponensek

---

---

---

---

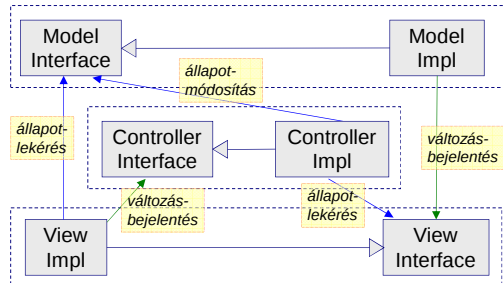
---

---

---

---

## MVC minta



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

61

## Áttérés AWT-ről

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

62

## Áttérés példa

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Test2 implements ActionListener {
 Button b;
 TextArea ta;
 TextField 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();
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

63

## Áttérés 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();
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

64

---

---

---

---

---

---

---

---

## Áttérés 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);
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

65

---

---

---

---

---

---

---

---

## Áttérés 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);
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

66

---

---

---

---

---

---

---

---

## Áttérés 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

---

---

---

---

---

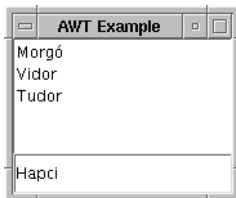
---

---

---

## Áttérés példa: *List* → *JList*

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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

70

---

---

---

---

---

---

---

---

## 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();
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

71

---

---

---

---

---

---

---

---

## 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();
}
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

72

---

---

---

---

---

---

---

---

## Swing megoldás

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

73

---

---

---

---

---

---

---

---

## 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*)

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

74

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

75

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

76

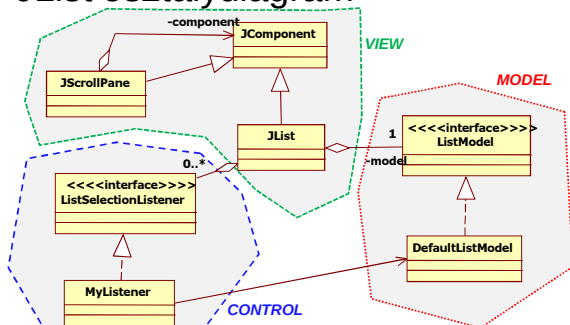
## DefaultListModel

- 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)**
  - ...

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

77

## JList osztálydiagram



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

78

## 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();
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

82

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

83

---

---

---

---

---

---

---

---

## 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)

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

84

---

---

---

---

---

---

---

---

## 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 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

---

---

---

---

---

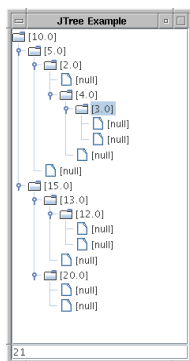
---

---

---

## 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

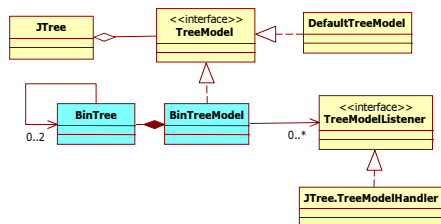
- a JTree ezt használja modelljének
- BinTree a háttérben
  - `BinTree root = new BinTree()`
- TreeModelListener-ek kezelése
  - `Vector<TreeModelListener> listeners`
  - `public void addTreeModelListener(TreeModelListener l)`
  - `public void removeTreeModelListener(TreeModelListener l)`

## 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 osztálydiagram



## 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);
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

97

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

98

---

---

---

---

---

---

---

---

## 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

99

---

---

---

---

---

---

---

---

## 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

100

## JTree példa 2: regisztrálás

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

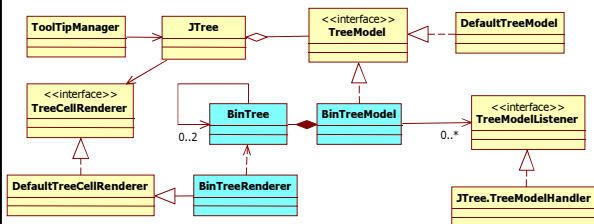
ToolTipManager.sharedInstance()
 .registerComponent(tree);

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

101

## JTree osztálydiagram



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

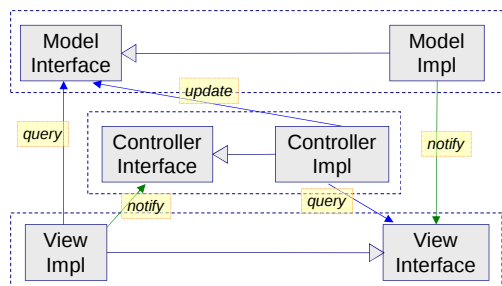
102

## MVC vs MVP

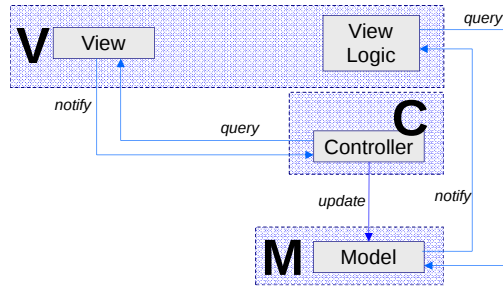
## Rétegezési kérdések

- Vékony kliens vs. vastag kliens
  - hol van a logika?
  - MVC: mennyi logika van a View-ban?
- Több rétegű architektúra
  - böngésző, alkalmazáserver, adatbázis
  - Megjelenítés, szolgáltatás, üzleti logika, infrastruktúra
- Okos böngésző
  - Ajax, GWT, stb: böngésző megjelenít és futtat is
  - vékony vagy vastag?

## Klasszikus MVC



## MVC vs. Model-View-Presenter



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

106

---

---

---

---

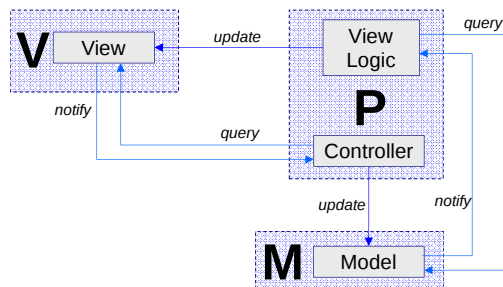
---

---

---

---

## MVC vs. Model-View-Presenter



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

107

---

---

---

---

---

---

---

---

## *Drag and Drop*

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

108

---

---

---

---

---

---

---

---

## Drag and drop



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

109

## Drag and Drop

### ■ `JComponent.setDragEnabled(boolean b)`

#### alapból tudják

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

#### kis segítséggel tudják

- ☐ `JList`
- ☐ `JTable`
- ☐ `JTree`

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

110

## Drag and drop beállítása

### ■ `setDragEnabled(boolean b)`

- ☐ bekacsolja 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

111

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

112

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

113

---

---

---

---

---

---

---

---

## **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)**

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

114

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

115

---

---

---

---

---

---

---

## TransferSupport 2

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

116

---

---

---

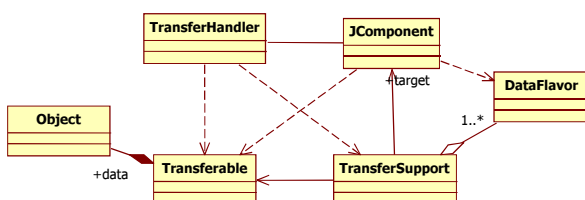
---

---

---

---

## Drag'n'Drop osztálydiagram



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

117

---

---

---

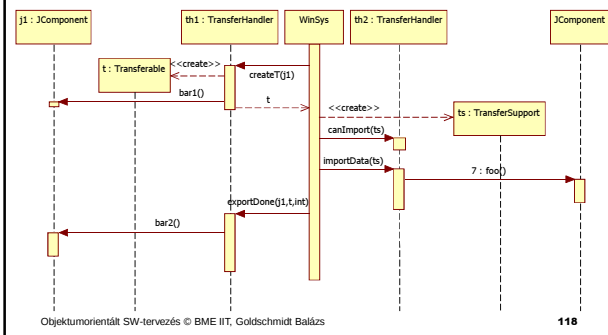
---

---

---

---

## Drag'n'Drop szekvencia



## Drag and drop példa

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

119

## 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
 }
 ...
}

```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

120

## 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;
}
...
```

Objektorientált SW-tervezés © BME IIT, Goldschmidt Balázs

121

---

---

---

---

---

---

---

---

## 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;
}
}
```

Objektorientált SW-tervezés © BME IIT, Goldschmidt Balázs

122

---

---

---

---

---

---

---

---

## 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);
```

Objektorientált SW-tervezés © BME IIT, Goldschmidt Balázs

123

---

---

---

---

---

---

---

---

## 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();
 }
 });
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

127

---

---

---

---

---

---

---

---

## Szálkezelés Swing alatt 4

- Modellek kezelésekor is
  - ☐ modelleket csak az eseménykezelő szálon keresztül módosítsunk
  - ☐ ha nem, akkor az esetleges kivételek tönkreteszik 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

128

---

---

---

---

---

---

---

---

## SwingWorker<T, V>

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

129

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

130

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

131

---

---

---

---

---

---

---

---

## SwingWorker<T,V> 4

- Ha valamit az eseménykezelő szálban 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 szálból hívva
  - a `publish()` metódusban átadott értékeket kapjuk itt meg

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

132

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

133

## Multithreaded GUI?

- Miért lehet csak egy eseménykezelő szál?
  - miért bonyolítottunk *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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

134

## Whiteboard minta

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

135

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

136

---

---

---

---

---

---

---

---

## 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 megfelelően
  - amikor az alkalmazás megáll, úgyis megszűnik minden

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

137

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

138

---

---

---

---

---

---

---

---

## Beágyazott Java jellemzői 2

- Folyamatosan futó VM
  - nem ragadhatnak be objektumok  
→ a korrekt életciklus-kezelés elengedhetetlen
  - nem igaz az, hogy „*amíg van referenciánk egy objektumra, az nem szűnik meg*”
    - erőforrás-kezelés kapcsán sokszor dinamikus 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*

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

142

---

---

---

---

---

---

---

## Swing Look and Feel

- Default L&F
  - ☐ Steel, Ocean
  - ☐ GTK+, Motif
  - ☐ Windows
- Beállítás
  - ☐ parancssorból
  - ☐ programból
  - ☐ property-fájlból

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

143

---

---

---

---

---

---

---

## 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`

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

144

---

---

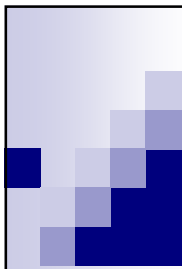
---

---

---

---

---



# Objektumorientált szoftvertervezés

Elosztott OO, RMI, CORBA

© BME IIT, Goldschmidt Balázs

---

---

---

---

---

---

---



## Hálózati forgalom

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

2

---

---

---

---

---

---

---



## 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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

3

---

---

---

---

---

---

---

## Socketes megoldás (szerver)

```
struct sockaddr_in cIntAddr;
int cIntLen = sizeof(cIntAddr);
int cIntSock =
 accept(servSock, (struct sockaddr *) &cIntAddr,
 &cIntLen);

char echoBuffer[RCVBUFSIZE];
int recvMsgSize =
 recv(cIntSock, echoBuffer, RCVBUFSIZE, 0);
printf("%s\n", echoBuffer);

close(cIntSock);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

4

---

---

---

---

---

---

---

---

## 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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

5

---

---

---

---

---

---

---

---

## 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(cIntSock, cp, sizeof(str), 0);
str* s = (str*)cp;
printf("%d %s %lf\n", s->a, s->b, s->c);
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

6

---

---

---

---

---

---

---

---

## 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

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

stack

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

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs 10

---

---

---

---

---

---

---

---

## Függvényhívás

■ Lokális

Egy memóriatér

Kliens

*másolás  
stack-re/ről*

interfész

implementáció

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs 11

---

---

---

---

---

---

---

---

## Függvényhívás

■ Távoli

Kliens memóriatér

Kliens

*másolás  
stack-re/ről*

interfész

?????

Szerver memóriatér

????

*másolás  
stack-re/ről*

?????

implementáció

hálózat

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs 12

---

---

---

---

---

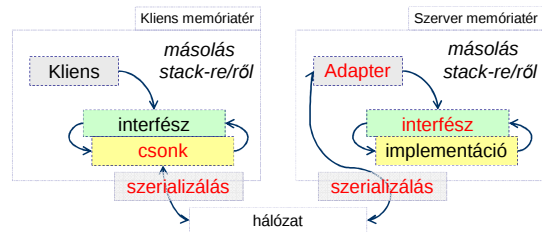
---

---

---

## Függvényhívás

### ■ Távoli



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

13

---

---

---

---

---

---

---

---

## 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;
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

14

---

---

---

---

---

---

---

---

## Távoli függvényhívás

### ■ Adapter (szerver oldal)

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

15

---

---

---

---

---

---

---

---

## 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 pointerok
  - `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)

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

19

---

---

---

---

---

---

---

## NameService

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

20

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

21

---

---

---

---

---

---

---

## 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

---

---

---

---

---

---

---

## Szerializálás 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
- ...

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

28

---

---

---

---

---

---

---

## 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 párai vannak a finomhangoláshoz

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

29

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

30

---

---

---

---

---

---

---

## 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

---

---

---

---

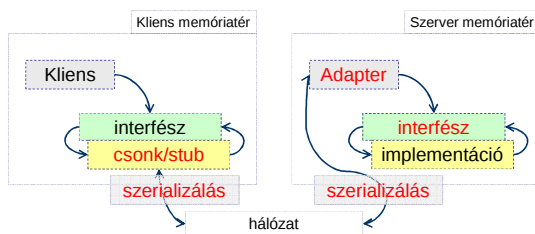
---

---

---

---

## RMI architektúra



---

---

---

---

---

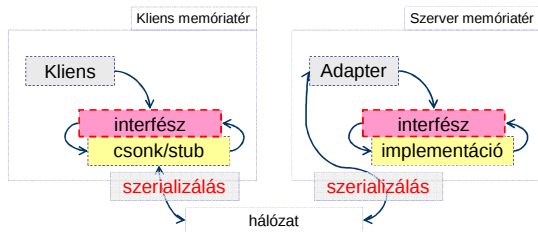
---

---

---

## RMI architektúra

### ■ Távoli objektum interfésze



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

34

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

35

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

36

## Távoli objektum interfésze 3

### ■ Példa:

```
import java.rmi.*;

public interface Hello extends Remote {

 void sayHello(String s)
 throws RemoteException;

}
```

---

---

---

---

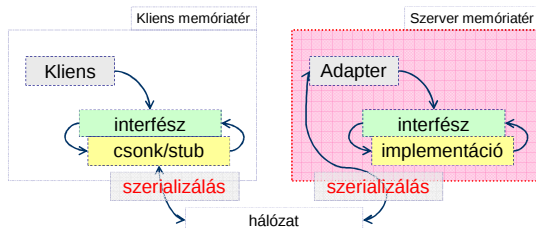
---

---

---

## RMI architektúra

### ■ Szerver oldal



---

---

---

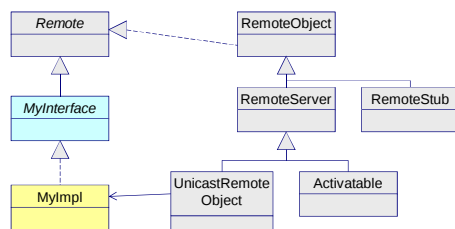
---

---

---

---

## Szerver oldal



---

---

---

---

---

---

---

## RemoteObject

### ■ Távoli objektumok és csontok (stubok) őssztálya

- ☐ **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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

40

---

---

---

---

---

---

---

## RemoteServer

### ■ Lehetővé teszi a távoli objektumok létrehozását és exportálását

- ☐ **static String getClientHost()**
  - távoli hívás során visszatér a kliens host azonosítójával
- ☐ **static void setLog(OutputStream os)**
  - az RMI hívások os-be lesznek naplózva.
- ☐ **static PrintStream getLog()**
  - visszaadja az RMI napló referenciáját

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

41

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

42

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

43

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

44

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

45

---

---

---

---

---

---

---

## 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);
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

46

---

---

---

---

---

---

---

---

## 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();}
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

47

---

---

---

---

---

---

---

---

## 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);
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

48

---

---

---

---

---

---

---

---

## Remote objektum exportálása

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

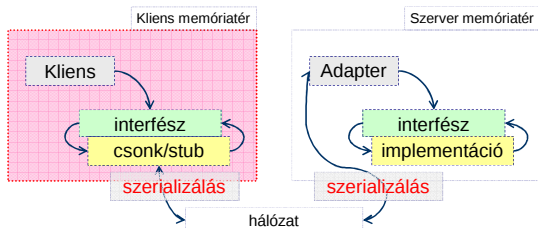
public class Server {
 static public void main(String[] args) {
 try {
 HelloImpl hello_i = new HelloImpl();
 // exportálás megtörtént a konstruktorban
 Registry registry = LocateRegistry.getRegistry();
 // paraméterátadásakor létrejön a stub
 // és csak ő adódik át
 registry.rebind("hello", hello_i);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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

49

## RMI architektúra

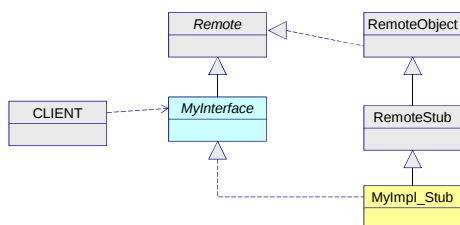
### Kliens oldal



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

50

### Kliens oldal



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

51

## 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 eljáráshívás során
  - visszatérési értéként távoli eljáráshívás során
  - névszolgáltatás segítségével
    - `java.rmi.Naming`

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

52

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

53

---

---

---

---

---

---

---

## 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();}
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

54

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

55

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

56

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

57

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

58

---

---

---

---

---

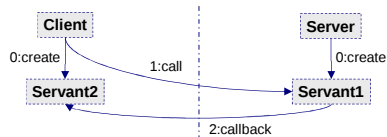
---

---

---

## Callback

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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

59

---

---

---

---

---

---

---

---

## Callback példa

- Egyszerű chat-szerver
  - kliensek regisztrálhatnak
  - üzenetet küldhetnek
  - a regisztráltak értesülnek a többiek üzeneteiről

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

60

---

---

---

---

---

---

---

---

## 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);
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

64

---

---

---

---

---

---

---

## 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();
 }
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

65

---

---

---

---

---

---

---

## 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);

 ...
 }
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

66

---

---

---

---

---

---

---

## 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();
}
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

67

---

---

---

---

---

---

---

## Callback példa futtatás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

68

---

---

---

---

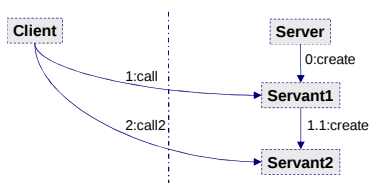
---

---

---

## Factory

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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

69

---

---

---

---

---

---

---

## 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);
}
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

73

---

---

---

---

---

---

---

## 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;
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

74

---

---

---

---

---

---

---

## 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;
}
...
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

75

---

---

---

---

---

---

---

## 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: *Hol a hiba?*

```
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");
 a1.placeAmount(10000);
 a1.transferAmount(3000, a1);
 System.out.println(a1.getAmount());
 a1.close();
 } catch (Exception e) {e.printStackTrace();}
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

79

---

---

---

---

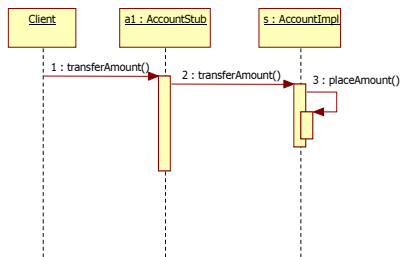
---

---

---

---

## Factory példa naívan



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

80

---

---

---

---

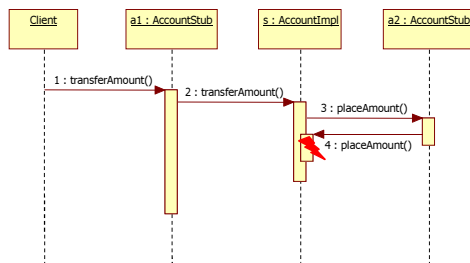
---

---

---

---

## Factory példa elosztottan



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

81

---

---

---

---

---

---

---

---

## Factory példa futtatás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

82

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

83

---

---

---

---

---

---

---

## 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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

84

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

85

---

---

---

---

---

---

---

## 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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

86

---

---

---

---

---

---

---

## DGC példa futtatás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

87

---

---

---

---

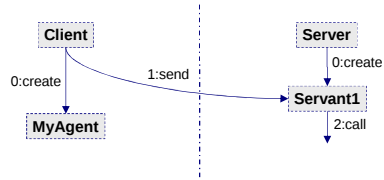
---

---

---

## Mobile agent

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



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

88

## Agent példa: interfészek

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

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

89

## 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);
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

90

## 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

91

---

---

---

---

---

---

---

## 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();
 }
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

92

---

---

---

---

---

---

---

## 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();}
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

93

---

---

---

---

---

---

---

## Agent példa futtatás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

94

---

---

---

---

---

---

---

## 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;
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

95

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

96

---

---

---

---

---

---

---

## 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_S
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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

100

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

101

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

102

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

103

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

104

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

105

---

---

---

---

---

---

---

## 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?

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

106

---

---

---

---

---

---

---

## 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 ügynököket átküldeni

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

107

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

108

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

109

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

110

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

111

---

---

---

---

---

---

---

## 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,
 MarshallableObject 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;
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

112

---

---

---

---

---

---

---

## 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/";
 ...
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

113

---

---

---

---

---

---

---

## 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);
...
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

114

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

118

---

---

---

---

---

---

---

## CORBA

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

119

---

---

---

---

---

---

---

## CORBA alapok

- Nyelvfüggetlen
  - C, C++, Java, COBOL, Ada, ...
- Szállítófüggetlen
  - IONA, Sun, OpenORB, TAO, ...
- Operációsrendszer-független
  - Windows \*, Linux, Solaris, ...

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

120

---

---

---

---

---

---

---

## 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 értse egy átvitt típusú adaton
  - szabványos végpont-elérés
    - az objektumok megszólítása, referálása azonos legyen
  - szabványos implementációs csatlakozás
    - ha ORB-t váltunk, ne kelljen semmit újraírni

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

121

---

---

---

---

---

---

---

## CORBA alapok 2

- Hálózati protokoll
  - GIOP: általános protokoll (pl. double sorosítás)
  - 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

122

---

---

---

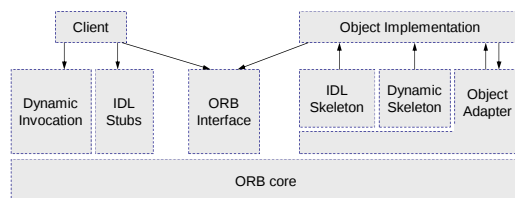
---

---

---

---

## CORBA alapok 3



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

123

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

124

---

---

---

---

---

---

---

## 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);
 };
};
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

125

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

126

---

---

---

---

---

---

---

## 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*

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

127

---

---

---

---

---

---

---

---

## 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 tilva a névtér egyszeres mélységében

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

128

---

---

---

---

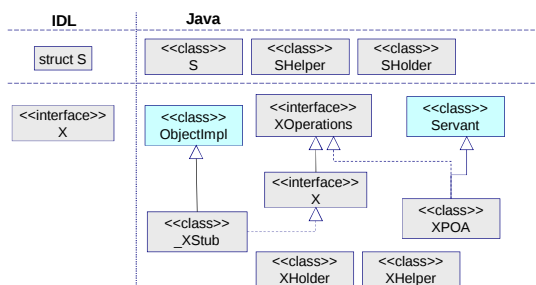
---

---

---

---

## IDL fordítása Java-ra



Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

129

---

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

130

---

---

---

---

---

---

---

## 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);
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

131

---

---

---

---

---

---

---

## 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
 }
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

132

---

---

---

---

---

---

---

## 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) {...}
 ...
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

133

---

---

---

---

---

---

---

---

## 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";
 }
}
```

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

134

---

---

---

---

---

---

---

---

## Szerver oldal 2

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

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

135

---

---

---

---

---

---

---

---

## POA

- *Portable Object Adapter*
- Célja, hogy a szervant újírása 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

136

---

---

---

---

---

---

---

## POA 2

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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

137

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

138

---

---

---

---

---

---

---

## 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

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

139

---

---

---

---

---

---

---

## 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.

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

140

---

---

---

---

---

---

---

## ***CORBA Naming Service***

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

141

---

---

---

---

---

---

---

## Naming Service alapok

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

142

---

---

---

---

---

---

---

## *CORBA Trading Service*

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

143

---

---

---

---

---

---

---

## Trading Service alapok

- Naming Service
  - white pages
  - név → referencia
- Trading Service
  - yellow pages
  - szolgáltatás → referencia lista

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

144

---

---

---

---

---

---

---

## Trading Service fogalmak

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

145

---

---

---

---

---

---

---

---

## Trading Service alapfogalmak 2

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

146

---

---

---

---

---

---

---

---

## Példa szolgáltatás

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

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

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

147

---

---

---

---

---

---

---

---

## Trader Use-case-ei

- Ajánlatok kezelése
  - publikálás: „insert”
  - módosítás: „update”
  - törlés: „delete”
  - keresés: „select”
- Trader attribútumok állítása
  - számosságok (search, match, return)
  - kéréstovábbítás

Objektumorientált SW-tervezés © BME IIT, Goldschmidt Balázs

148

---

---

---

---

---

---

---

## 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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

149

---

---

---

---

---

---

---

## CORBA Event Service

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

150

---

---

---

---

---

---

---

## Bevezetés

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

151

---

---

---

---

---

---

---

## 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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

152

---

---

---

---

---

---

---

## 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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

153

---

---

---

---

---

---

---

## 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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

154

---

---

---

---

---

---

---

## A csatorna 2

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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

155

---

---

---

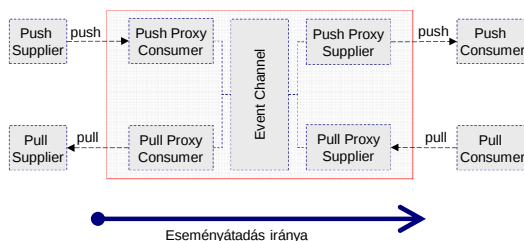
---

---

---

---

## A csatorna modellje



Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

156

---

---

---

---

---

---

---

## 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

Objektumorientált fejlesztés © BME IIT, Goldschmidt Balázs

157

---

---

---

---

---

---

---

## 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 SW-tervezés © BME IIT, Goldschmidt Balázs

158

---

---

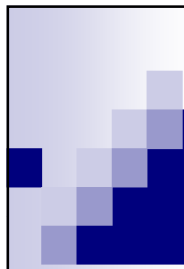
---

---

---

---

---



Objektum-orientált tervezés

eXtensible Markup Language  
(XML, namespaces, XSD)

---

---

---

---

---

---

---



Outline

- XML
- XML Namespaces
- XSD
- XML processing

© BME IIT, Simon Balázs 2

---

---

---

---

---

---

---



***eXtensible Markup Language  
(XML)***

© BME IIT, Simon Balázs

---

---

---

---

---

---

---

## What is XML?

- eXtensible Markup Language
- Markup language like HTML
- Designed to carry data, not to display data
- Tags are not predefined, they are specific to the application that understands them
- Self-descriptive

© BME IIT, Simon Balázs

4

---

---

---

---

---

---

---

## XML vs. HTML

- XML is not a replacement for HTML
- XML and HTML were designed with different goals:
  - XML was designed to transport and store data, with focus on what data is
  - HTML was designed to display data, with focus on how data looks
- HTML is about displaying information, while XML is about carrying information

© BME IIT, Simon Balázs

5

---

---

---

---

---

---

---

## Simple example

- This is a message from Bob to Alice:

```
<message time="15:12">
 <to>Alice</to>
 <from>Bob</from>
 <subject importance="3">Hello</subject>
 <body>Hello, Alice!</body>
</message>
```
- It is just information
- The application has to understand it

© BME IIT, Simon Balázs

6

---

---

---

---

---

---

---

## XML tags are defined by the application

- XML tags are not defined by any standard
- XML has no predefined tags
- The application decides what tag names it uses
- The application decides the structure of the XML document
- The tags in HTML (e.g. <table>, <image>, etc.) are predefined: they must be understood by browsers
- XML does not replace HTML
- XML is for storing and transporting information

© BME IIT, Simon Balázs

7

---

---

---

---

---

---

---

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

8

---

---

---

---

---

---

---

## Parts of an XML document

XML declaration (this is metadata, not a tag)

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

9

---

---

---

---

---

---

---

## Parts of an XML document

root: opening tag

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

\* There is exactly one root tag!  
\* tag=element

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

opening/ start tag  
closing/ end tag  
\* Every opened tag must be closed!  
\* Tag names are case sensitive.

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

simple content: text  
mixed content: subtags and text

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

empty tag

\* Empty tags close themselves.

\* They have no children, no content.

## Parts of an XML document

```
root tag
<?xml version="1.0" encoding="utf-8"?>
child tags
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

sub-child tags

\* XML is a tree structure of tags. They must be properly nested. No overlapping!

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time='15:12'>
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="3">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

comment

## Parts of an XML document

```
<?xml version="1.0" encoding="utf-8"?>
<message date="2013.09.18." time="015:120">
 <to>Alice</to>
 <from>Bob</from>
 <!-- comment -->
 <subject importance="030">Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line/>
 <line index="2">How are you?</line>
 <line index="3"/>
 <line>Bye, Bob</line>
 </body>
</message>
```

Annotations in the original image:

- attribute**: points to `date="2013.09.18."`
- name**: points to `from` in `<from>Bob</from>`
- value**: points to `Bob` in `<from>Bob</from>`
- comment**: points to `<!-- comment -->`
- be quoted between either quotes (") or apostrophes (')**: points to `importance="030"`
- \* Attribute values must**: points to `index="2"`

16

## Names of tags and attributes

- can contain letters, numbers and other characters
- cannot start with a number or a punctuation character
- cannot start with the letters „xml“ (also XML, Xml, etc.)
- cannot contain spaces (tab, new line, etc.) or colons (":")
- **best practice:**
  - ☐ use the english alphabet, numbers and underscores (e.g. book\_title or BookTitle)
  - ☐ do not use dots (e.g. book.title) or dashes (e.g. book-title)

© BME IIT, Simon Balázs

17

## XML tags/elements

- Elements can contain:
  - ☐ other elements
  - ☐ attributes
  - ☐ text
  - ☐ or a mix of all of the above

© BME IIT, Simon Balázs

18

## XML attributes

- Attributes provide additional information about elements
- Usually this information is not part of the data
- There is no rule when to use attributes or names. All of these are allowed:

<pre>&lt;message date="2013.09.18."&gt;   &lt;to&gt;Alice&lt;/to&gt; &lt;/message&gt;</pre>	<pre>&lt;message&gt;   &lt;date&gt;     &lt;year&gt;2013&lt;/year&gt;     &lt;month&gt;09&lt;/month&gt;     &lt;day&gt;18&lt;/day&gt;   &lt;/date&gt;   &lt;to&gt;Alice&lt;/to&gt; &lt;/message&gt;</pre>
---------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

19

## XML attributes best practices

- Use attributes only when their value is metadata (not part of or relevant to the data to be stored). Example: identifiers

```
<body>
 <line id="525">Hello, Alice!</line>
 <line id="221">How are you?</line>
 <line id="312">Bye, Bob</line>
</body>
```

- Attributes cannot contain multiple values. Elements can.
- Attributes cannot contain structured values. Elements can.

© BME IT, Simon Balázs

20

## XML content

- Element content or attribute value
- Can be any text
- But: "<" and "&" **must** be escaped
- Usually ">", "&", """ and "'" should be escaped. Like this:

sign	escaping sequence
< (less than)	&lt;
> (greater than)	&gt;
& (ampersand)	&amp;
" (quotation mark)	&quot;
' (apostrophe)	&apos;

- Examples:

OK →

```
<line>count < 100</line>
<line>count < 100</line>
<line name="my" name="">hello</line>
<line name='my' name=''>hello</line>
<line name="my" "name=">hello</line>
```

error →

21

## XML: well formed vs. valid

- Well formed XML: has the correct syntax (what we have talked about until now)
  - ☐ exactly one root element
  - ☐ elements are closed and nested properly
  - ☐ element names are case-sensitive
  - ☐ attribute values are quoted
- Valid XML: well formed and conforms to a document that describes *the structure and the type* of elements and attributes in the XML document

© BME IIT, Simon Balázs

22

## Valid XML

- Earlier we saw: tags are not predefined, they are specific to the application that understands them
- The application can specify these tags and their structure, so that the XML can be validated
- The specification is another document. It can be one of the following:
  - ☐ DTD = Document Type Definition
  - ☐ XSD = XML Schema Definition

© BME IIT, Simon Balázs

23

## DTD = Document Type Definition

```

<!DOCTYPE message
[
 <!ELEMENT message (to,from,subject,body)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT subject (#PCDATA)>
 <!ELEMENT body (line*)>
 <!ELEMENT line (#PCDATA)>
]>

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message SYSTEM "Message.dtd">
<message>
 <to>Alice</to>
 <from>Bob</from>
 <subject>Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line>How are you?</line>
 <line>Bye, Bob</line>
 </body>
</message>

```

© BME IIT, Simon Balázs

24

## XSD = XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 <xs:element name="message" type="tns:messageType"/>
 <xs:complexType name="messageType">
 <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="subject" type="xs:string"/>
 <xs:element name="body" type="tns:bodyType"/>
 </xs:sequence>
 </xs:complexType>
 <xs:complexType name="bodyType">
 <xs:sequence>
 <xs:element name="line" type="xs:string"
 minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

25

## XML Namespaces

© BME IIT, Simon Balázs

## Conflicting tag names

- What if we want to use the same tag names with different contents and/or meanings in the same XML?

- Examples:

<pre>&lt;beams&gt;   &lt;beam&gt;     &lt;material&gt;wood&lt;/material&gt;     &lt;length&gt;10m&lt;/length&gt;   &lt;/beam&gt;   &lt;beam&gt;     &lt;material&gt;concrete&lt;/material&gt;     &lt;length&gt;15m&lt;/length&gt;   &lt;/beam&gt; &lt;/beams&gt;</pre>	<pre>&lt;beams&gt;   &lt;beam&gt;     &lt;kind&gt;light&lt;/kind&gt;     &lt;wave&gt;300nm&lt;/wave&gt;   &lt;/beam&gt;   &lt;beam&gt;     &lt;kind&gt;laser&lt;/kind&gt;     &lt;wave&gt;500nm&lt;/wave&gt;   &lt;/beam&gt; &lt;/beams&gt;</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

© BME IIT, Simon Balázs

27

## XML name prefix

- Name conflicts can be resolved by a prefix
- E.g.:

```

<beams>
 <c:beam>
 <c:material>concrete</c:material>
 <c:length>15m</c:length>
 </c:beam>
 <em:beam>
 <em:kind>light</em:kind>
 <em:wave>300nm</em:wave>
 </em:beam>
</beams>

```

for construction beams

for electromagnetic beams

© BME IIT, Simon Balázs

28

## XML namespaces

- For each prefix, a namespace has to be defined
- The namespace for the prefix is defined as an attribute on an XML tag like this: xmlns:prefix="URI"
- The name of the prefix is arbitrary
- The namespace URI is what counts. This identifies the namespace

© BME IIT, Simon Balázs

29

## XML namespaces

- Example:

```

<beams>
 <c:beam xmlns:c="http://www.iit.bme.hu/soi/construction">
 <c:material>concrete</c:material>
 <c:length>15m</c:length>
 </c:beam>
 <em:beam xmlns:em="http://www.iit.bme.hu/soi/emwaves">
 <em:kind>light</em:kind>
 <em:wave>300nm</em:wave>
 </em:beam>
</beams>

```

namespace prefix

URI of the construction namespace

prefix also applicable for the descendant elements

© BME IIT, Simon Balázs

30

## Namespace URI

- It is just an identifier
- It does not point anywhere
- Some companies may use namespace URIs to point to their websites
- Some standards may use namespace URIs to point to the standards' websites
- But this is not a rule, just a coincidence

© BME IIT, Simon Balázs

31

---

---

---

---

---

---

---

## XML namespaces

- Namespaces can be declared in the elements they are used
- Or in an ancestor element
- Even in the root tag:

```
<beams xmlns:c="http://www.iit.bme.hu/soi/construction"
 xmlns:em="http://www.iit.bme.hu/soi/emwaves">
 <c:beam>
 <c:material>concrete</c:material>
 <c:length>15m</c:length>
 </c:beam>
 <em:beam>
 <em:kind>light</em:kind>
 <em:wave>300nm</em:wave>
 </em:beam>
</beams>
```

© BME IIT, Simon Balázs

32

---

---

---

---

---

---

---

## Prefix name

- The prefix name is arbitrary
- The same namespace URI can be assigned to multiple prefixes at the same time:

```
<beams xmlns:c1="http://www.iit.bme.hu/soi/construction"
 xmlns:c2="http://www.iit.bme.hu/soi/construction"
 xmlns:em="http://www.iit.bme.hu/soi/emwaves">
 <c1:beam>
 <c2:material>concrete</c2:material>
 <c3:length
 xmlns:c3="http://www.iit.bme.hu/soi/construction">
 15m
 </c3:length>
 </c1:beam>
 <em:beam>
 <em:kind>light</em:kind>
 <em:wave>300nm</em:wave>
 </em:beam>
</beams>
```

alászs

33

---

---

---

---

---

---

---

## Prefix name

- Prefix names can be redefined in a descendant element to denote another namespace:

```
<beams xmlns:b="http://www.iit.bme.hu/soi/construction">
 <b:beam>
 <b:material>concrete</b:material>
 <b:length>15m</b:length>
 </b:beam>
 <b:beam xmlns:b="http://www.iit.bme.hu/soi/emwaves">
 <b:kind>light</b:kind>
 <b:wave>300nm</b:wave>
 </b:beam>
</beams>
```

© BME IIT, Simon Balázs

34

---

---

---

---

---

---

---

---

## Default namespace

- If the prefix name is omitted, the namespace URI will be the default namespace for the defining element and for all of its descendants:

```
<beams>
 <beam xmlns="http://www.iit.bme.hu/soi/construction">
 <material>concrete</material>
 <length>15m</length>
 </beam>
 <beam xmlns="http://www.iit.bme.hu/soi/emwaves">
 <kind>light</kind>
 <wave>300nm</wave>
 </beam>
</beams>
```

© BME IIT, Simon Balázs

35

---

---

---

---

---

---

---

---

## XML Schema Definition (XSD)

© BME IIT, Simon Balázs

---

---

---

---

---

---

---

---

## XML Schema

### ■ XML schema defines:

- ☐ elements that can appear in a document
- ☐ attributes that can appear in a document
- ☐ which elements are child elements
- ☐ the order of child elements
- ☐ the number of child elements
- ☐ whether an element is empty or can include text
- ☐ data types for elements and attributes
- ☐ default and fixed values for elements and attributes

© BME IIT, Simon Balázs

37

---

---

---

---

---

---

---

---

## XML Schema vs. DTD

### ■ XML schema is better than DTD because:

- ☐ it is extensible to future additions
- ☐ it is richer and more powerful than DTDs
- ☐ it is written in XML
- ☐ it supports data types
- ☐ it supports namespaces

© BME IIT, Simon Balázs

38

---

---

---

---

---

---

---

---

## DTD = Document Type Definition

```

<!DOCTYPE message
[
 <!ELEMENT message (to,from,subject,body)>
 <!ELEMENT to (#PCDATA)>
 <!ELEMENT from (#PCDATA)>
 <!ELEMENT subject (#PCDATA)>
 <!ELEMENT body (line*)>
 <!ELEMENT line (#PCDATA)>
]>

<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE message SYSTEM "Message.dtd">
<message>
 <to>Alice</to>
 <from>Bob</from>
 <subject>Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line>How are you?</line>
 <line>Bye, Bob</line>
 </body>
</message>

```

© BME IIT, Simon Balázs

39

---

---

---

---

---

---

---

---

## XSD = XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 <xs:element name="message" type="tns:messageType"/>
 <xs:complexType name="messageType">
 <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="subject" type="xs:string"/>
 <xs:element name="body" type="tns:bodyType"/>
 </xs:sequence>
 </xs:complexType>
 <xs:complexType name="bodyType">
 <xs:sequence>
 <xs:element name="line" type="xs:string"
 minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

40

## XSD root element: <schema>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 ...
 ...
</xs:schema>
```

© BME IIT, Simon Balázs

41

## XSD root element: <schema>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 ...
 ...
</xs:schema>
```

Namespace for the XML schema definition

Prefix for the XML schema definition elements

The root also belongs to the XML schema definition

© BME IIT, Simon Balázs

42

## XSD root element: <schema>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 ...
 ...
</xs:schema>
```

This is "our" namespace:  
this is the namespace where  
the elements and types we  
define in this XML schema  
will be belonging to

© BME IIT, Simon Balázs

43

---

---

---

---

---

---

---

## XSD root element: <schema>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 ...
 ...
</xs:schema>
```

We also assign a prefix to  
our namespace so that we  
can reference the elements  
and types we define in this  
XML schema definition

© BME IIT, Simon Balázs

44

---

---

---

---

---

---

---

## XSD root element: <schema>

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 ...
 ...
</xs:schema>
```

This means that the elements in  
the XML document described by  
this schema must be namespace  
qualified.

© BME IIT, Simon Balázs

45

---

---

---

---

---

---

---

## Types that can be defined in an XSD

- Simple types
  - ☐ elements
  - ☐ attributes
  - ☐ restrictions
- Complex types
  - ☐ elements
  - ☐ empty elements
  - ☐ text content
  - ☐ mixed content
  - ☐ any content

© BME IIT, Simon Balázs

46

---

---

---

---

---

---

---

---

## Elements with simple types

- These elements can contain only text conforming to the type defined for the elements
- Examples:
 

```

<xs:element name="first_name" type="xs:string"/>
<xs:element name="age" type="xs:int"/>
<xs:element name="birth_date" type="xs:date"/>

```

XSD prefix: this means  
that this type is a built-in  
type in XSD
- Examples for the XML described by the XSD:
 

```

<first_name>Bob</first_name>
<age>27</age>
<birth_date>1986.08.11.</birth_date>

```

© BME IIT, Simon Balázs

47

---

---

---

---

---

---

---

---

## Built-in simple types

- Some simple built-in types in XSD are:
  - ☐ xs:string
  - ☐ xs:decimal
  - ☐ xs:integer
  - ☐ xs:boolean
  - ☐ xs:date
  - ☐ xs:time

© BME IIT, Simon Balázs

48

---

---

---

---

---

---

---

---

## Default and fixed values for elements

### ■ Elements can have default values

#### □ XSD:

```
<xs:element name="count" type="xs:int" default="100"/>
```

#### □ XML:

```
<count/> ← value: 100
<count>200</count> ← value: 200
```

### ■ And fixed values:

#### □ XSD:

```
<xs:element name="lang" type="xs:string" fixed="us"/>
```

#### □ XML:

```
<language>us</language> ← value: us
<language>de</language> ← ERROR!
```

© BME IIT, Simon Balázs

49

## Attributes with simple types

### ■ Attributes must always have simple types

### ■ The value of the attribute must conform to its type

### ■ Attributes can only be defined for complex elements

#### ■ XSD:

```
<xs:element name="person">
 <xs:complexType>
 <xs:attribute name="first_name" type="xs:string"/>
 <xs:attribute name="age" type="xs:int"/>
 <xs:attribute name="birth_date" type="xs:date"/>
 </xs:complexType>
</xs:element>
```

#### ■ XML:

```
<person first_name="Bob" age="27"
 birth_date="1986.08.11."/>
```

© BME IIT, Simon Balázs

50

## Default and fixed values for attributes

### ■ Attributes can have default values

#### □ XSD:

```
<xs:attribute name="count" type="xs:int" default="100"/>
```

#### □ XML:

```
<list/> ← value of the count attribute: 100
<list count="200"/> ← value: 200
```

### ■ And fixed values:

#### □ XSD:

```
<xs:attribute name="lang" type="xs:string" fixed="us"/>
```

#### □ XML:

```
<book lang="us"/> ← value: us
<book lang="de"/> ← ERROR!
```

© BME IIT, Simon Balázs

51

## Required attributes

- Attributes are optional by default
- To make them required:

□ XSD: required attribute

```
<xs:attribute name="count" type="xs:int" required="true"/>
```

□ XML:

```
<list/> ← ERROR!
```

```
<list count="200"/> ← OK
```

© BME IIT, Simon Balázs

52

---

---

---

---

---

---

---

---

## Restrictions on values

- The interval of values can be specified:

□ XSD:

```
<xs:element name="age">
 <xs:simpleType>
 <xs:restriction base="xs:integer">
 <xs:minInclusive value="0"/>
 <xs:maxInclusive value="120"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

□ XML:

```
<age>27</age> ← OK
<age>-13</age> ← ERROR!
<age>214</age> ← ERROR!
```

© BME IIT, Simon Balázs

53

---

---

---

---

---

---

---

---

## Restrictions on a set of values

- The set of values can be specified:

□ XSD:

```
<xs:element name="color">
 <xs:simpleType>
 <xs:restriction base="xs:string">
 <xs:enumeration value="red"/>
 <xs:enumeration value="green"/>
 <xs:enumeration value="blue"/>
 </xs:restriction>
 </xs:simpleType>
</xs:element>
```

□ XML:

```
<color>red</color> ← OK
<color>yellow</color> ← ERROR!
```

© BME IIT, Simon Balázs

54

---

---

---

---

---

---

---

---

## Naming and reusing types (simple and complex types)

```
<xs:element name="color" type="tns:colorType"/>
```

reference to our own type

the prefix refers to our namespace since the type is defined in our namespace

The name of our own type. Remember: it is defined in our targetNamespace!

```
<xs:simpleType name="colorType">
 <xs:restriction base="xs:string">
 <xs:enumeration value="red"/>
 <xs:enumeration value="green"/>
 <xs:enumeration value="blue"/>
 </xs:restriction>
</xs:simpleType>
```

© BME IIT, Simon Balázs

55

---

---

---

---

---

---

---

---

## Restrictions

### ■ There are also other kind of restrictions:

- ☐ intervals
- ☐ set of values
- ☐ number format (digits and decimal places)
- ☐ length of the content
- ☐ regular expression

© BME IIT, Simon Balázs

56

---

---

---

---

---

---

---

---

## Complex elements

### ■ Complex elements can contain:

- ☐ child elements
- ☐ attributes
- ☐ text
- ☐ mix of all of the above

© BME IIT, Simon Balázs

57

---

---

---

---

---

---

---

---

## Child elements: sequence

### ■ XSD:

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 <xs:element name="age" type="xs:int"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

the order of the elements is important

### ■ XML:

```
<person>
 <firstname>John</firstname>
 <lastname>Schmidt</lastname>
 <age>58</age>
</person>
```

child elements:  
only in this order

58

## Child elements: choice

### ■ XSD:

```
<xs:element name="person">
 <xs:complexType>
 <xs:choice>
 <xs:element name="husband" type="xs:string"/>
 <xs:element name="wife" type="xs:string"/>
 </xs:choice>
 </xs:complexType>
</xs:element>
```

exactly one of the child elements

### ■ XML:

```
<person>
 <husband>Schmidt</husband>
</person>
```

child elements:  
exactly one of them

59

## Child elements: all

### ■ XSD:

```
<xs:element name="person">
 <xs:complexType>
 <xs:all>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 <xs:element name="age" type="xs:int"/>
 </xs:all>
 </xs:complexType>
</xs:element>
```

the order of the elements is not important

### ■ XML:

```
<person>
 <firstname>John</firstname>
 <age>58</age>
 <lastname>Schmidt</lastname>
</person>
```

child elements:  
in any order

60

## Child elements: multiple elements

### ■ XSD:

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="name" type="xs:string"
 minOccurs="2" maxOccurs="unbounded"/>
 <xs:element name="age" type="xs:int"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

at least 2x      at most infinity-times

### ■ XML:

```
<person>
 <name>Walter</name>
 <name>Bruce</name>
 <name>Willis</name>
 <age>58</age>
</person>
```

name multiple times

61

## Mixed content

### ■ XSD:

```
<xs:element name="person">
 <xs:complexType mixed="true">
 <xs:sequence>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 <xs:element name="age" type="xs:int"/>
 </xs:sequence>
 </xs:complexType>
</xs:element>
```

mixed content

### ■ XML:

```
<person>
 His name is <firstname>John</firstname>
 <lastname>Schmidt</lastname>. He
 is <age>58</age> years old.
</person>
```

62

## Any content

- To allow other elements not defined in our schema

- It makes extension possible

```
<xs:element name="person">
 <xs:complexType>
 <xs:sequence>
 <xs:element name="firstname" type="xs:string"/>
 <xs:element name="lastname" type="xs:string"/>
 <xs:any minOccurs="0"/>
 </xs:sequence>
 <xs:anyAttribute/>
 </xs:complexType>
</xs:element>
```

any other elements even from other namespaces

any other attributes even from other namespaces

© BME IIT, Simon Balázs

63

## Validating an XML against our XSD

- The first element defined in the XSD is the root element of the XML
- The same namespace has to be specified in the XML as the targetNamespace in the XSD
- The XSD can be referenced from the XML:
  - The path to the XSD file has to be specified with the schemaLocation attribute
- Example on the next slides

© BME IIT, Simon Balázs

64

---

---

---

---

---

---

---

---

## XSD = XML Schema Definition

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://www.iit.bme.hu/soi/message"
 xmlns:tns="http://www.iit.bme.hu/soi/message"
 elementFormDefault="qualified">
 <xs:element name="message" type="tns:messageType"/>
 <xs:complexType name="messageType">
 <xs:sequence>
 <xs:element name="to" type="xs:string"/>
 <xs:element name="from" type="xs:string"/>
 <xs:element name="subject" type="xs:string"/>
 <xs:element name="body" type="tns:bodyType"/>
 </xs:sequence>
 </xs:complexType>
 <xs:complexType name="bodyType">
 <xs:sequence>
 <xs:element name="line" type="xs:string"
 minOccurs="0" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

65

---

---

---

---

---

---

---

---

## Example XML

the first element defined in the XSD is the root

the targetNamespace defined in the XSD

```
<?xml version="1.0" encoding="utf-8"?>
<message xmlns="http://www.iit.bme.hu/soi/message"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="
 http://www.iit.bme.hu/soi/message Message.xsd">
 <to>Alice</to>
 <from>Bob</from>
 <subject>Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line>How are you?</line>
 <line>Bye, Bob</line>
 </body>
</message>
```

© BME IIT, Simon Balázs

66

---

---

---

---

---

---

---

---

### Example XML

```
<?xml version="1.0" encoding="utf-8"?>
<message xmlns="http://www.iit.bme.hu/soi/message"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation=
 "http://www.iit.bme.hu/soi/message Message.xsd">
 <to>Alice</to>
 <from>Bob</from>
 <subject>Hello</subject>
 <body>
 <line>Hello, Alice!</line>
 <line>How are you?</line>
 <line>Bye, Bob</line>
 </body>
</message>
```

the standard namespace for schema validation

namespace-path pairs separated by whitespace where:

- \* namespace is an XML namespace URI
- \* path is the path to the XSD defining this namespace

© BME IIT, Simon Balázs 67

---

---

---

---

---

---

---

---

## XML Processing

© BME IIT, Simon Balázs

---

---

---

---

---

---

---

---

## XML Processing

- General, untyped
  - ☐ event-driven
  - ☐ stream-based
  - ☐ tree in memory
- Specific, typed
  - ☐ Java: JAXB
  - ☐ .NET: DataContract, XmlSerializer

© BME IIT, Simon Balázs 69

---

---

---

---

---

---

---

---

## Event-driven

- SAX = Simple API for XML
- Implement a callback interface
- Push model: the parser calls the functions when an XML artifact is found
  - opening tag, end tag, attribute, text content, etc.
- Properties:
  - serial access
  - low memory footprint
  - no positioning
- Java:
  - callback interface: `org.xml.sax.ContentHandler`
  - parser: `javax.xml.parsers.SAXParserFactory`, `javax.xml.parsers.SAXParser`

© BME IIT, Simon Balázs

70

## Stream-based

- StAX = Streaming API for XML
- Reading as a stream
- Pull model: reading XML artifacts in a cycle
- Properties:
  - serial access
  - low memory footprint
  - no positioning
  - more intuitive than the event-driven push model: no complex state machine
- Java: `javax.xml.stream.XMLStreamReader`
- .NET: `System.Xml.XmlReader`

© BME IIT, Simon Balázs

71

## Tree in memory

- DOM = Document Object Model
- The whole document is loaded into a tree structure in the memory
- Properties:
  - random access
  - enables editing
  - large memory footprint
  - enables validation
- Java:
  - `javax.xml.parsers.DocumentBuilderFactory`, `javax.xml.parsers.DocumentBuilder`, `org.w3c.dom.Document`
- .NET:
  - `System.Xml.XmlDocument`
  - `System.Xml.Linq.XDocument`

© BME IIT, Simon Balázs

72

## Typed

- XML document is mapped to typed objects in the memory
- Types can be mapped to XSD
- Properties:
  - ☐ random access
  - ☐ enables editing
  - ☐ enables validation
  - ☐ typed objects are more convenient to use
  - ☐ faster, and lower memory footprint than DOM
  - ☐ only certain XML and XSD artifacts are supported
  - ☐ very specific, cannot be used in general
- Java: JAXB
- .NET: DataContract, XmlSerializer

© BME IIT, Simon Balázs

73

---

---

---

---

---

---

---

---

## JAXB

© BME IIT, Simon Balázs

---

---

---

---

---

---

---

---

## JAXB

- Java Architecture for XML Binding
- JSR-222
- Why?
  - ☐ XML handling: SAX, DOM
  - ☐ SAX: event-driven, serial access, fast, small memory footprint
  - ☐ DOM: builds a tree representation in memory, the tree is changeable, and can be validated
  - ☐ Problems:
    - they are too general
    - access is not statically typed
    - no compile-time check

© BME IIT, Simon Balázs

75

---

---

---

---

---

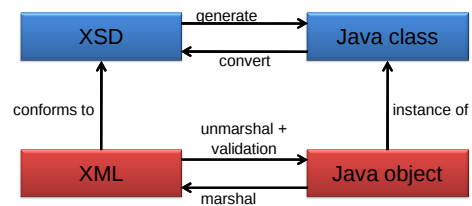
---

---

---

## JAXB

- Mapping between XSD and Java
- Annotations
- Model:



© BME IT, Simon Balázs

76

## JAXB

- Specification:
  - Subset of XML Schema
  - Default mappings:
    - XML names ↔ Java names
    - Schema atomic types ↔ Java types
    - Schema structured types ↔ Java classes
  - Java package: `javax.xml.bind`
- Compiler:
  - XML Schema-to-Java Compiler (xjc)
  - Part of the JDK:
    - `%JAVA_HOME%\bin\xjc.exe`

© BME IT, Simon Balázs

77

## Atomic types

XML Schema	Java
string	java.lang.String
boolean	boolean
byte	byte
short	short
int	int
long	long
float	float
double	double
integer	java.math.BigInteger
decimal	java.math.BigDecimal
unsignedByte	short
unsignedShort	int
unsignedInt	long
base64Binary	byte[]
hexBinary	byte[]
date	java.util.Date
time	java.util.Date
dateTime	java.util.Calendar
QName	javax.xml.namespace.QName

## Custom bindings

- Customizable:
  - ☐ generated package names, class names, method names
  - ☐ return type of a method
  - ☐ field types
  - ☐ which elements belong to a class
  - ☐ which element to omit
  - ☐ fields are mapped to elements or attributes
- How:
  - ☐ Special elements with a predefined JAXB namespace in the XML Schema part
  - ☐ Separate XML descriptor (recommended)

© BME IIT, Simon Balázs

79

## XML Schema support

- Some XML Schema constructs are not supported:
  - ☐ wildcard types
    - any, anyAttribute
  - ☐ identity-constraints:
    - key, keyref, unique
  - ☐ other constraints:
    - restriction
  - ☐ ...

© BME IIT, Simon Balázs

80

## Packages and namespaces

**WsApiSample.xsd:**

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ik.bme.hu/WsApi"
 xmlns="http://ik.bme.hu/WsApi"
 xmlns:wsapi="http://ik.bme.hu/WsApi">
</xs:schema>
```

**wsApi\sample\package-info.java:**

```
@javax.xml.bind.annotation.XmlSchema(
 namespace = "http://ik.bme.hu/WsApi",
 elementFormDefault =
 javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package wsApi.sample;
```

### JAXB sample

**wsApi\sample\Person.java:**

```
package wsApi.sample;

import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlType;

@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Person", propOrder = {
 "age",
 "name"
})
public class Person {
 @XmlElement(name = "Age")
 protected int age;
 @XmlElement(name = "Name", required = true, nillable = true)
 protected String name;

 // + getter/setter methods
}
```

**WsApiSample.xsd:**

```
<xs:schema ...>
 <xs:element name="Person"
 nillable="true"
 type="wsapi:Person"/>
 <xs:complexType name="Person">
 <xs:sequence>
 <xs:element name="Age"
 type="xs:int"/>
 <xs:element name="Name"
 nillable="true"
 type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

82

---

---

---

---

---

---

---

---

### JAXB sample: list

**package wsApi.sample;**

```
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlElements;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "Names", propOrder = {
 "name"
})
public class Names {
 @XmlElements(name = "Name")
 protected List<String> name;

 // + getter/setter methods
}
```

**can be the root of the XML document**

```
<xs:schema ...>
 <xs:complexType name="Names">
 <xs:sequence>
 <xs:element name="Name"
 type="xs:string"
 minOccurs="0"
 maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

83

---

---

---

---

---

---

---

---

### JAXB serialization

■ **Serialization:**

```
Names names = new Names();
OutputStream os = new FileOutputStream("names.xml");
JAXBContext context = JAXBContext.newInstance(Names.class);
Marshaller m = context.createMarshaller();
m.marshal(names, os);
os.close();
```

■ **Deserialization:**

```
InputStream is = new FileInputStream("names.xml");
JAXBContext context = JAXBContext.newInstance(Names.class);
Unmarshaller um = context.createUnmarshaller();
Names names = (Names) um.unmarshal(is);
is.close();
```

84

---

---

---

---

---

---

---

---



## WCF DataContract

©BME IIT, Simon Balázs

---

---

---

---

---

---

---



## WCF DataContract

- Mapping between .NET types and XML Schema
- Metadata on classes: .NET attributes
- Similar to JAXB

© BME IIT, Simon Balázs 86

---

---

---

---

---

---

---



## Atomic types

XML Schema	C#
string	string
boolean	bool
byte	byte
short	short
int	int
long	long
float	float
double	double
integer	long
decimal	decimal
unsignedByte	short
unsignedShort	int
unsignedInt	long
base64Binary	byte[]
hexBinary	string
date	string
time	string
dateTime	DateTime
QName	XmlQualifiedName

---

---

---

---

---

---

---

## Packages and namespaces

### WsApiSample.xsd:

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 targetNamespace="http://ik.bme.hu/WsApi"
 xmlns="http://ik.bme.hu/WsApi"
 xmlns:wsapi="http://ik.bme.hu/WsApi">
</xs:schema>
```

### WsApiSample.cs:

```
namespace WsApiSample
{
 ...
}
```

Annotated on the classes

## WCF DataContract sample

### WsApiSample.xsd:

```
<xs:schema ...>
 <xs:element name="Person"
 nillable="true"
 type="wsapi:Person"/>
 <xs:complexType name="Person">
 <xs:sequence>
 <xs:element name="Age"
 type="xs:int"/>
 <xs:element name="Name"
 nillable="true"
 type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

### WsApiSample.cs:

```
namespace WsApiSample
{
 ...
 [DataContract(Namespace="http://ik.bme.hu/WsApi")]
 public class Person
 {
 [DataMember(Name="Age")]
 public int Age { get; set; }
 [DataMember(Name="Name")]
 public String name { get; set; }
 }
 ...
}
```

89

## WCF DataContract sample: list

```
<xs:schema ...>
 <xs:complexType name="Names">
 <xs:sequence>
 <xs:element name="Name"
 type="xs:string"
 minOccurs="0"
 maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
```

### WsApiSample.cs:

```
namespace WsApiSample
{
 ...
 [CollectionDataContract(Namespace="http://ik.bme.hu/WsApi",
 Names="Names", ItemName="Name")]
 public class Names : List<string>
 {
 }
 ...
}
```

© BME IIT, Simon Balázs

90

## WCF DataContract serialization

### ■ Serialization:

```
Names names = new Names();
FileStream fs = new FileStream("names.xml", FileMode.Create,
 FileAccess.Write);

DataContractSerializer dcs =
 new DataContractSerializer(typeof(Names));
dcs.WriteObject(fs, names);
fs.Close();
```

### ■ Deserialization:

```
FileStream fs = new FileStream("names.xml", FileMode.Open,
 FileAccess.Read);

DataContractSerializer dcs =
 new DataContractSerializer(typeof(Names));
Names names = (Names)dcs.ReadObject(fs);
fs.Close();
```

91

## Summary

### ■ XML

- designed to carry data, not to display data
- tags are not predefined, they are specific to the application that understands them
- structured

### ■ XML namespaces

- to avoid name collisions

### ■ XSD

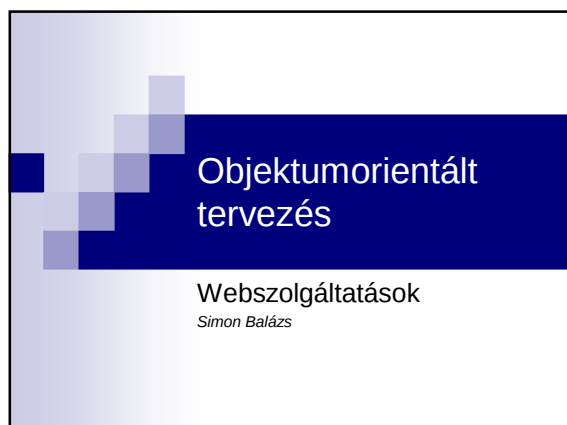
- defines the structure and types of elements and attributes in an XML document

### ■ XML processing

- event-driven, stream-based, DOM tree, typed

© BME IT, Simon Balázs

92



---

---

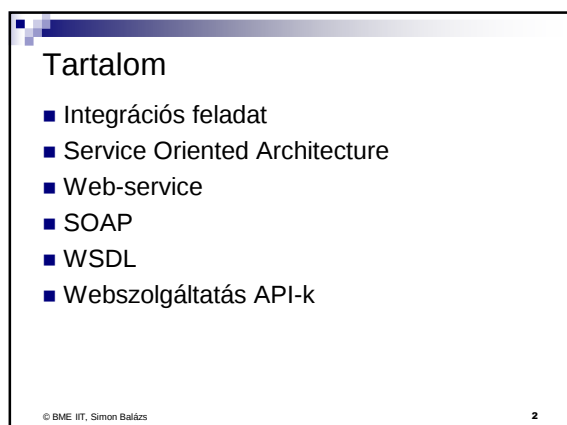
---

---

---

---

---



---

---

---

---

---

---

---



---

---

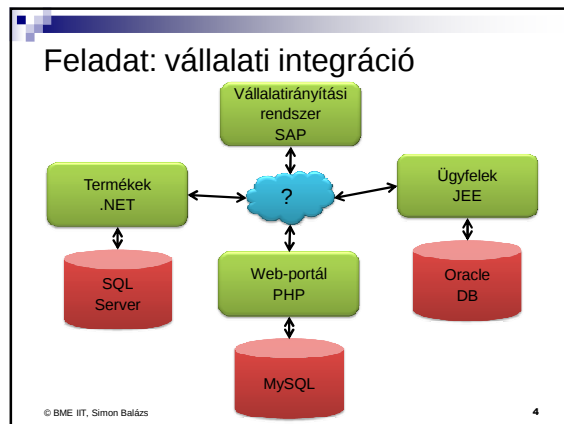
---

---

---

---

---




---

---

---

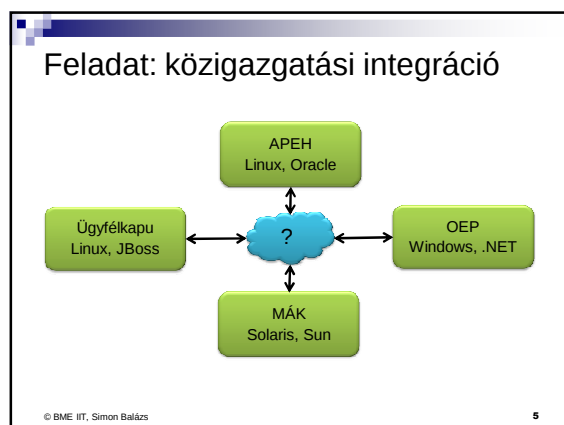
---

---

---

---

---




---

---

---

---

---

---

---

---

- 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, Webszolgáltatások

---

---

---

---

---

---

---

---

## 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
- Elrejtje az implementáció részleteit
- Lehetőleg minél vékonyabb réteg
- Fontos a jó tervezés:
  - ☐ a kiejánlott 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

© BME IIT, Simon Balázs

10

---

---

---

---

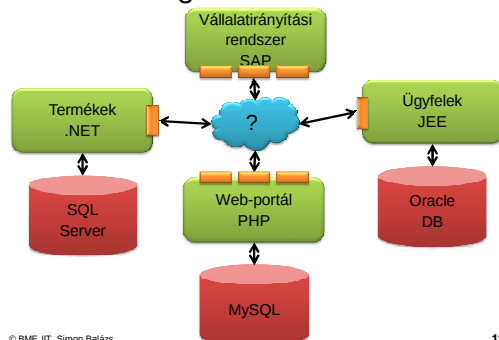
---

---

---

---

## Vállalati integráció



© BME IIT, Simon Balázs

11

---

---

---

---

---

---

---

---

## Példák: közigazgatás

### ■ szolgáltatások:

- ☐ lakcím megváltoztatása
- ☐ adóbevallás fogadása
- ☐ állampolgár adótartozásának lekérdezése
- ☐ anyasági támogatás igénylése

### ■ folyamatok:

- ☐ lakcímváltozás bejelentése
- ☐ adóbevallás benyújtása
- ☐ vállalkozás alapítása

© BME IIT, Simon Balázs

12

---

---

---

---

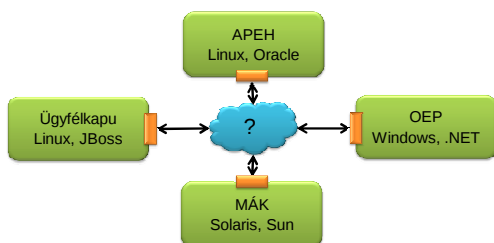
---

---

---

---

## Közigazgatási integráció



© BME IIT, Simon Balázs

13

---

---

---

---

---

---

---

## Webszolgáltatás

14

Objektum-orientált szoftvertervezés  
© BME IIT, Simon Balázs

---

---

---

---

---

---

---

## Webszolgáltatás

- 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

---

---

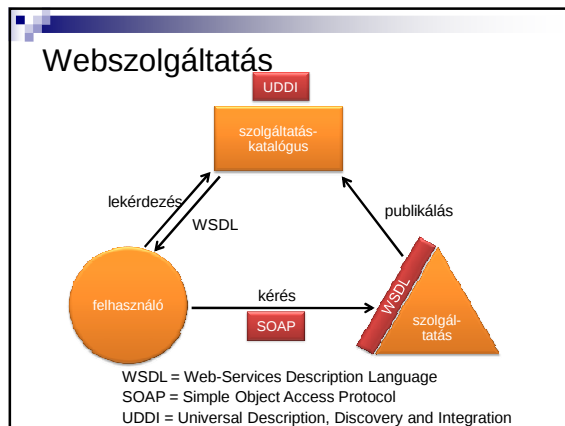
---

---

---

---

---




---

---

---

---

---

---

---

---




---

---

---

---

---

---

---

---




---

---

---

---

---

---

---

---

## Példa: WSDL 1/7

```

interface IHelloWorld
{
 string SayHello(string name);
}

<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
 targetNamespace="http://www.iit.bme.hu/soi"
 xmlns:xs="http://www.w3.org/2001/XMLSchema"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:wsaw="http://www.w3.org/2006/05/addressing/wsdl"
 xmlns:tns="http://www.iit.bme.hu/soi"
 xmlns:ns0="http://www.iit.bme.hu/soi">
 ...
</wsdl:definitions>

```

19

---

---

---

---

---

---

---

---

## Példa: WSDL 2/7

```

interface IHelloWorld
{
 string SayHello(string name);
}
...
<wsdl:types>
 <xs:schema targetNamespace="http://www.iit.bme.hu/soi"
 xmlns:tns="http://www.iit.bme.hu/soi"
 xmlns:ns0="http://www.iit.bme.hu/soi"
 elementFormDefault="qualified">
 <xs:element name="SayHello" nillable="true"
 type="ns0:SayHello"/>
 <xs:complexType name="SayHello">
 <xs:sequence>
 <xs:element name="name" type="xs:string"
 nillable="true"/>
 </xs:sequence>
 </xs:complexType>
 </xs:schema>
</wsdl:types>

```

20

---

---

---

---

---

---

---

---

## Példa: WSDL 3/7

```

interface IHelloWorld
{
 string SayHello(string name);
}
...
 <xs:element name="SayHelloResponse" nillable="true"
 type="ns0:SayHelloResponse"/>
 <xs:complexType name="SayHelloResponse">
 <xs:sequence>
 <xs:element name="SayHelloResult" type="xs:string"
 nillable="true"/>
 </xs:sequence>
 </xs:complexType>
</xs:schema>
</wsdl:types>
...

```

21

---

---

---

---

---

---

---

---

## Példa: WSDL 4/7

```

interface IHelloWorld
{
 string SayHello(string name);
}

...
<wsdl:message name="IHelloWorld_SayHello_InputMessage">
 <wsdl:part name="parameters" element="ns0:SayHello"/>
</wsdl:message>

<wsdl:message name="IHelloWorld_SayHello_OutputMessage">
 <wsdl:part name="parameters"
 element="ns0:SayHelloResponse"/>
</wsdl:message>
...

```

22

---

---

---

---

---

---

---

## Példa: WSDL 5/7

```

interface IHelloWorld
{
 string SayHello(string name);
}

...
<wsdl:portType name="IHelloWorld">
 <wsdl:operation name="SayHello">
 <wsdl:input wsaw:action=
 "http://www.iit.bme.hu/soi/IHelloWorld/SayHello"
 message="ns0:IHelloWorld_SayHello_InputMessage"/>
 <wsdl:output wsaw:action=
 "http://www.iit.bme.hu/soi/IHelloWorld/SayHelloResponse"
 message="ns0:IHelloWorld_SayHello_OutputMessage"/>
 </wsdl:operation>
</wsdl:portType>
...

```

23

---

---

---

---

---

---

---

## Példa: WSDL 6/7

```

...
<wsdl:binding name="IHelloWorld_BasicHttpBinding_Binding"
 type="ns0:IHelloWorld">
 <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
 <wsdl:operation name="SayHello">
 <soap:operation style="document" soapAction=
 "http://www.iit.bme.hu/soi/IHelloWorld/SayHello"/>
 <wsdl:input>
 <soap:body use="literal"/>
 </wsdl:input>
 <wsdl:output>
 <soap:body use="literal"/>
 </wsdl:output>
 </wsdl:operation>
</wsdl:binding>
...

```

24

---

---

---

---

---

---

---

### Példa: WSDL 7/7

```
interface IHelloWorld
{
 string SayHello(string name);
}

...
<wsdl:service name="HelloWorld">
 <wsdl:port name="IHelloWorld_BasicHttpBinding_Port"
 binding="ns0:IHelloWorld_BasicHttpBinding_Binding">
 <soap:address location=
 "http://www.iit.bme.hu/Services/HelloWorld"/>
 </wsdl:port>
</wsdl:service>
```

25

---

---

---

---

---

---

---

---

### Webszolgáltatás

- 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

26

---

---

---

---

---

---

---

---

### 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 (szöveges) alapúak
- adatformátumok túl alacsony szintűek: byte-ok igazításával is foglalkozni kell
- tűzfalak, proxy szerverek általában korlátozzák ezeket

27

---

---

---

---

---

---

---

---



## SOAP

*(Simple Object Access Protocol)*

Objektum-orientált szoftvertervezés  
© BME IIT, Simon Balázs

28

---

---

---

---

---

---

---

---



## 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

29

---

---

---

---

---

---

---

---



## 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)

30

---

---

---

---

---

---

---

---

## 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:Body>
 <env:Fault>
 ...
 </env:Fault>
</env:Envelope>
```



31

---

---

---

---

---

---

---

---

## 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

32

---

---

---

---

---

---

---

---

## 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

33

---

---

---

---

---

---

---

---

## WSDL

### (Web-Services Description Language)

34

Objektum-orientált szoftvertervezés  
© BME IIT, Simon Balázs

---

---

---

---

---

---

---

---

## WSDL

- Web Services Description Language
- Leíró Webszolgáltatásokhoz
  - interfész
  - meta-adatok
  - szolgáltatások címei
- W3C gondozásában
- Használt verziók: 1.1 és 2.0

35

---

---

---

---

---

---

---

---

## 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

36

---

---

---

---

---

---

---

---

## 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")<sub>37</sub>

## 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]"/>*
 </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>
```

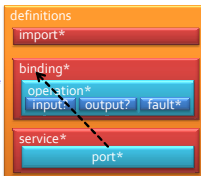


## WSDL 1.1: konkrét rész

```
<?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

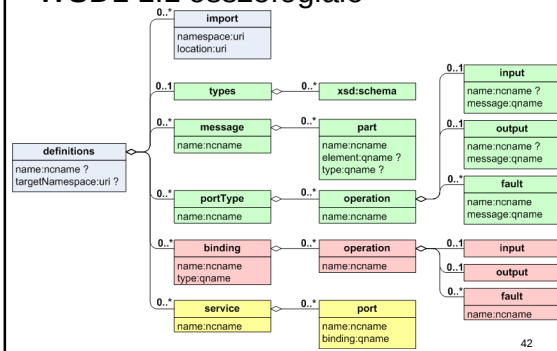
	van output	nincs output
van input	request-reply	one-way (aszinkron)
nincs input	solicit-response	-

## 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

41

## WSDL 1.1 összefoglaló



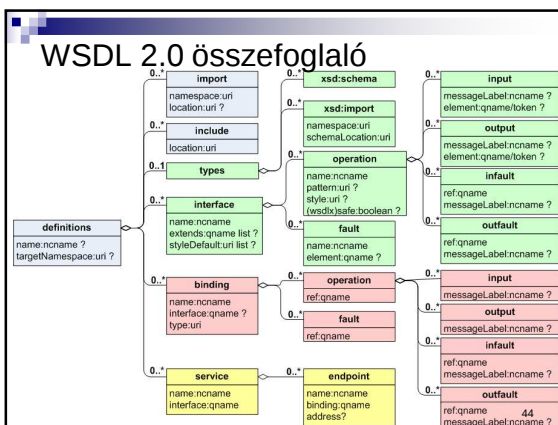
42

## 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)

43

## WSDL 2.0 összefoglaló



## Webszolgáltatás API-k

45

Objektum-orientált szoftvertervezés  
© BME IIT, Simon Balázs

## JAX-WS

- Java API for XML-based Web-Services (JDK6+)
- Célja:
  - Webszolgáltatások egyszerű implementálása
  - WSDL és Java osztályok közti leképezés
  - Annotációkkal
- Primitív típusok átadása alapértelmezetten támogatott
- Saját összetett típusokat JAXB annotációkkal kell ellátni

© BME IIT, Simon Balázs

46

## Egyszerű Java interfész

```
public class MathException extends Exception {
}

public interface ICalculator
{
 public double divide(double left, double right)
 throws MathException;
}
```

## JAX-WS példa: egyszerű webszolgáltatás

```
@WebFault
public class MathException extends Exception {
}

@WebService
public interface ICalculator
{
 @WebMethod
 public double divide(double left, double right)
 throws MathException;
}
```

### JAX-WS példa: elnevezések, névterek

```
@WebFault(name = "MathException",
 targetNamespace = "http://ik.bme.hu/WsApi")
public class MathException extends Exception {
}

@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 MathException;
}
```

---

---

---

---

---

---

---

---

### JAX-WS példa: implementáció

```
@WebService(endpointInterface = "ICalculator")
public class Calculator implements ICalculator
{
 @Override
 public double divide(double left, double right)
 throws MathException
 {
 if(right == 0)
 {
 throw new MathException();
 }
 return left / right;
 }
}
```

© BME ITT, Simon Balázs

50

---

---

---

---

---

---

---

---

### 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 (MathException ex) {
 ex.printStackTrace();
}
finally {
 ((Closeable)calculator).close();
}
```

© BME ITT, Simon Balázs

51

---

---

---

---

---

---

---

---

## WCF

- Windows Communication Foundation (.NET 3.0+)
- Célja:
  - Webszolgáltatások egyszerű implementálása
  - WSDL és .NET osztályok közti leképezés
  - .NET attribútumokkal
- Primitív típusok átadása alapértelmezetten támogatott
- Saját összetett típusokat DataContract annotációkkal kell ellátni

© BME IIT, Simon Balázs

52

---

---

---

---

---

---

---

## Egyszerű C# interfész

```
public class MathFault : Exception
{
}

public interface ICalculator
{
 double Divide(double left, double right);
}
```

© BME IIT, Simon Balázs

53

---

---

---

---

---

---

---

## WCF példa: egyszerű webszolgáltatás

```
public class MathFault : Exception
{
}

[ServiceContract]
public interface ICalculator
{
 [OperationContract]
 [FaultContract(typeof(MathFault))]
 double Divide(double left, double right);
}
```

© BME IIT, Simon Balázs

54

---

---

---

---

---

---

---

## WCF példa: elnevezések, névterek

```
public class MathFault : Exception
{
}

[ServiceContract(Namespace = "http://ik.bme.hu/WsApi")]
public interface ICalculator
{
 [OperationContract]
 [FaultContract(typeof(MathFault), Name = "MathFault")]
 double Divide(double left, double right);
}
```

© BME IIT, Simon Balázs

55

---

---

---

---

---

---

---

---

## WCF példa: implementáció

```
public class Calculator : ICalculator
{
 public double Divide(double left, double right)
 {
 if (right == 0)
 {
 throw new FaultException<MathFault>(
 new MathFault());
 }
 return left / right;
 }
}
```

© BME IIT, Simon Balázs

56

---

---

---

---

---

---

---

---

## 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();
}
```

57

---

---

---

---

---

---

---

---

## Összefoglalás

- SOA
- Webszolgáltatások
  - SOAP
  - WSDL
- Webszolgáltatás API-k
  - Java: JAX-WS
  - .NET: WCF

© BME IIT, Simon Balázs

58

---

---

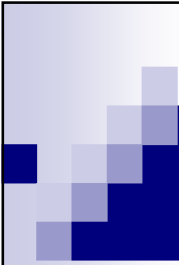
---

---

---

---

---



# Objektumorientált fejlesztés

REpresentational State Transfer (REST)

Simon Balázs, BME IIT, 2014.

---

---

---

---

---

---

---



## Tartalom

- HTTP
- REST
- REST kritikák
- JAX-RS
- WCF és REST
- Elosztott technológiák

(C) Simon Balázs, BME IIT, 2011.

2

---

---

---

---

---

---

---



# HTTP

(C) Simon Balázs, BME IIT, 2011.

3

---

---

---

---

---

---

---

## HTTP GET



The screenshot shows a Mozilla Firefox browser window with the address bar displaying `http://www.google.com/index.html`. Below the browser window, the HTTP request and response are shown. The request is a GET for `/index.html` with a status of 200 OK. The response headers include `Content-Type: text/html; charset=UTF-8`, `Content-Encoding: gzip`, `Server: gws`, and `Content-Length: 10200`. The response body starts with `<!doctype html><html><head>...`.

**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>...
```

(C) Simon Balázs, BME IIT, 2011. 4

---

---

---

---

---

---

---

---

## HTTP GET:

`http://www.abc.com/login?user=xy&pass=123`

HTTP Method	Local URL	Version
GET	<code>/login?user=xy&amp;pass=123</code>	HTTP/1.1

Host: `www.abc.com`  
User-Agent: `Mozilla/5.0`  
Connection: `keep-alive`

Headers+ Empty line	Host name	Query params

(C) Simon Balázs, BME IIT, 2011. 5

---

---

---

---

---

---

---

---

## HTTP POST:

`http://www.abc.com/login`

HTTP Method	Local URL	Version
POST	<code>/login</code>	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

(C) Simon Balázs, BME IIT, 2011.

---

---

---

---

---

---

---

---



# REST

(C) Simon Balázs, BME IIT, 2011.

7

---

---

---

---

---

---

---



# 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

(C) Simon Balázs, BME IIT, 2011.

8

---

---

---

---

---

---

---



# REST alapelvei

- Minden erőforráshoz azonosító rendelése
- Dolgok összekapcsolása
- CRUD műveletek használata
- Többféle adatrepresentáció
- Állapotmentes kommunikáció

(C) Simon Balázs, BME IIT, 2011.

9

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

10

---

---

---

---

---

---

---

---

## Erőforrások azonosítóival

- **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.)

(C) Simon Balázs, BME IIT, 2011.

11

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

12

---

---

---

---

---

---

---

---

## 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 interfészleíró nyelvre
- Az erőforrások manipulálására elegendő a négy művelet: GET, POST, PUT, DELETE
- Ez a „Hypertext As the Engine of State Transfer” (HATEOS) elv

(C) Simon Balázs, BME IIT, 2011.

13

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

14

---

---

---

---

---

---

---

---

## Műveletek erőforrásokon

	CRUD	safe	idempotent	cacheable
GET	read	<b>igen</b>	<b>igen</b>	<b>igen</b>
POST	create	nem	nem	nem
PUT	update/ create	nem	<b>igen</b>	nem
DELETE	delete	nem	<b>igen</b>	nem

POST: a szerver rendel hozzá azonosítót  
 PUT: a kliens határozza meg az azonosítót

(C) Simon Balázs, BME IIT, 2011.

15

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

16

---

---

---

---

---

---

---

---

## Állapotmentes kommunikáció

- A REST önmagában állapotmentes
- De az alkalmazásnak lehet állapota:
  - erőforrásban tárolva (nem a memóriában)
  - kliens oldalon (üzenetcserenél mindig átküldve)
- Előnyök:
  - skálázhatóság: nem kell session-t fenntartani
  - különböző szerverpéldányok is kiszolgálhatnak egymás utáni kéréseket
  - a szerver leállítható, újraindítható két kérés között

(C) Simon Balázs, BME IIT, 2011.

17

---

---

---

---

---

---

---

---

## REST kritikák

(C) Simon Balázs, BME IIT, 2011.

18

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME ITT, 2011.

19

---

---

---

---

---

---

---

---

## 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.

(C) Simon Balázs, BME ITT, 2011.

20

---

---

---

---

---

---

---

---

## 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)

(C) Simon Balázs, BME ITT, 2011.

21

---

---

---

---

---

---

---

---

## Túl sok belső részlet

- A CRUD műveleteken kívül lehet más is
  - ez tetszőlegesen bonyolult lehet, akár csak 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

(C) Simon Balázs, BME IIT, 2011.

22

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

23

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

24

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

25

---

---

---

---

---

---

---

---

## JAX-RS

(C) Simon Balázs, BME IIT, 2011.

26

---

---

---

---

---

---

---

---

## 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

(C) Simon Balázs, BME IIT, 2011.

27

---

---

---

---

---

---

---

---

## 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 {
}
```

(C) Simon Balázs, BME IIT, 2011.

28

---

---

---

---

---

---

---

---

## Számológép példa

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
```

```
@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);
 }
}
```

Csúnya, de a double  
nem működik:

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

(C) Simon Balázs, BME IIT, 2011.

30

---

---

---

---

---

---

---

---

## Param példák

```
@Path("calculator")
public class Calculator {
 @POST
 @Path("add")
 public String add(@PathParam("left") double left,
 @PathParam("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`

(C) Simon Balázs, BME IIT, 2011.

31

## Visszatérési érték

- Lehetséges visszatérési értékek:
  - **void, null:** üres válasz „204 No Content” státuskóddal
  - **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`

(C) Simon Balázs, BME IIT, 2011.

32

## 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;
}
```

(C) Simon Balázs, BME IIT, 2011.

33

## 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);
 }
}
```

(C) Simon Balázs, BME IIT, 2011.

34

---

---

---

---

---

---

---

## MessageBodyWriter példa

```
...
@Override
public long getSize(Object t, Class type,
 Type genericType,
 Annotation[] annotations,
 MediaType mediaType)
{
 return -1;
}
...
```

(C) Simon Balázs, BME IIT, 2011.

35

---

---

---

---

---

---

---

## 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();
}
}
```

(C) Simon Balázs, BME IIT, 2011.

36

---

---

---

---

---

---

---

## Számológép példa revisited

```
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;

@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;
 }
}
```

Mostmár működik

<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

(C) Simon Balázs, BME IIT, 2011.

39

## 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

(C) Simon Balázs, BME IIT, 2011.

39

## XML és JSON példa

**sample/package-info.java:**

```
@javax.xml.bind.annotation.XmlSchema(
 namespace="",
 elementFormDefault=javax.xml.bind.annotation.XmlNsForm.QUALIFIED)
package sample;
```

**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>();
 }
}
```

(C) Simon Balázs, BME ITT, 2011. 40

Kliens oldali  
JSON miatt üres!

## 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;
 }
}
```

(C) Simon Balázs, BME ITT, 2011. 41

## 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>
```

(C) Simon Balázs, BME ITT, 2011. 42

## 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"}]}
```

(C) Simon Balázs, BME IT, 2011.

43

## 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

(C) Simon Balázs, BME IT, 2011.

44

## 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();
 }
}
```

(C) Simon Balázs, BME IT, 2011.

45

## 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;
}
}
```

(C) Simon Balázs, BME IIT, 2011.

46

---

---

---

---

---

---

---

## 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();
 }
 }
}
```

(C) Simon Balázs, BME IIT, 2011.

47

---

---

---

---

---

---

---

## WCF és REST

(C) Simon Balázs, BME IIT, 2011.

48

---

---

---

---

---

---

---

## WCF és REST

- WCF használható REST-re is
  - ahogy SOAP webszolgáltatásokra is
- Az attribútumok ugyanazok: ServiceContract, OperationContract, DataContract, stb.
  - További attribútumok: WebGet, WebInvoke
    - az URL formátumát definiálják
- A konfiguráció hasonló
  - A használandó binding: webHttpBinding
- További információ:
  - <http://msdn.microsoft.com/en-us/magazine/dd315413.aspx>

(C) Simon Balázs, BME IIT, 2011. 49

---

---

---

---

---

---

---

---

## *Elosztott szolgáltatások megvalósítására alkalmas technológiák*

(C) Simon Balázs, BME IIT, 2011. 50

---

---

---

---

---

---

---

---

## Elosztott technológiák

- RMI
- CORBA
- WS
- REST

(C) Simon Balázs, BME IIT, 2011. 51

---

---

---

---

---

---

---

---

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	

(C) Simon Balázs, BME IT, 2011. 52

---

---

---

---

---

---

---

---

Elosztott technológiák				
	RMI	CORBA	WS	REST
Protokoll	RMI	IIOP	SOAP	HTTP
Interfészleíró	Java interface	IDL	WSDL	nincs/ WADL
Katalógus	JNDI	Naming Service	UDDI	-

(C) Simon Balázs, BME IT, 2011. 53

---

---

---

---

---

---

---

---