Java 7 vitafórum

Ezek voltak a tegnapi Java7 vitafórum témái. Lehet folytatni!

Generikus típusok futásidben

```
if (list instanceof List<String>) {
...
}
```

Ez most nem fordul, pedig jó lenne ha ki lehetne deríteni futásidben a generikus szerkezetek paramétereinek típusait.
Elvileg valami trükkel mégiscsak ki lehet szedni! Aki véletlenül tudja ezt a trükköt, ne tartsa magában!

Enum-ok ordinary értékei

Ha enum-okat definiálok, akkor az int reprezentációjuk mindig 0-tól egyesével növekszik, pedig jó lenne, ha kézzel meg lehetne adni értékeket. Pl. Utólag szeretnék beszúrni egy elemet középre. Ha simán beszúrom elcsúsznak az értékek mögötte. A program verziók nem lesznek kompatibilisek! Vagy egy meglév rendszer konstansait szeretném beállítani az enum értékeinek.

Ez nem igazán zavart senkit, könnyen kiküszöbölhet. Pl. hibernate-ben nem a számértékeket, hanem a név-értékeket érdemes eltárolni.

Autoboxing NPE

```
Integer i = null;
int x = i
```

NullPointerException-t okoz. Tapasztalataim szerint nem könny észrevenni a fentinél kevésbé triviális helyzetekben.
Lehetne az NPE üzenetrészében legalább valami "autoboxing" vagy hasonló.

Ez sem zavart senkit.

Triviális kasztolások

```
byte[] b = new byte[] {0xFF, 0xFF };
byte[] b = new byte[] {(byte)0xFF, (byte)0xFF };
```

(Megj.: amikor alacsonyszint Java-t programoztam (Embedded java) majdnem szétkapartam az arcomat kínomban emiatt.)

Más is találkozott már vele, ez van.

switch-be nem lehet String-et írni

```
switch (str) {
   case "ONE": return 1;
   case "TWO": return 2;
   case "THREE": return 3;
 }
```

Ezen elfilóztunk kicsit. A switch JVM utasítás bels szerkezete csak az int összehasonlításnak kedvez és tulajdonképpen egy függvénytáblát használ. Azért syntax sugar-nek meg lehetne csinálni. De akkor is, mikor használjunk equals-t és mikor ==-t?

Typedef

Ha sok helyen használunk egy adott generikus típust lehessen egy typedef-szer dolgot megadni:

```
import java.util.Map<String, Integer> as CodeValueMap;
```

(Megj: Ha sok helyen használunk egy adott generikus típust, akkor akár külön osztályt is csinálhatunk rá.)

No way.

Rövidítés

Mi lenne, ha nem kellene kiírni a List-ben a get-et:

```
List<String> list; ...
String s = list.get(3000); helyett
String s = list[3000]
```

vagy

```
Map<String, Integer> map; ...
Integer i = map["egy"];
```

(Mi lenne ha lenne operator overloading...)

Nem tartottuk fontosnak. Aki C#-ban akar programozni az programozzon C#-ban-ban.

Property-k

Van aki nem szeret gettereket írogatni.

```
public class Person {
  private String _forename;
  public property int age;
 }
```

(Megj.: Jobb IDE megírja öt kattintásra a gettereket. A fentinél bonyolultabb esetekben a property-knek is kell getter.)

Azért van aki mellette volt. Jobban belegondolva tényleg csak fölösleges "kódzaj" a sok getter és setter, azokban az esetekben, amikor úgyis csak beállítja-visszaadja a privát mez értékét.

BigDecimal, BigInteger

Kényelmetlen a használata: metódushívások helyett kényelmesebb lenne operátorokat használni.

```
BigInteger sum = a.add(b.add(c.add(d))); helyett
BigInteger sum = a + b + c + d;
```

Elvileg már tervezik.

XML support

Lehetne valami nyelvi szint XML szolgáltatás, mint a Stringeknél.

```
Node n = <Person><Name>Joe</Name><Country>Netherlands</Country></Person>;
```

(Láttam valahol egy videót, ahol 45 percen keresztül taglalja a témát egy ipse.)

Gáznak tartottuk. Ha megtalálom a videót belinkelem majd / aki ismeri linkelje be.

Closures

Bels függvények, lokális függvények. Néha jól jönnének.

(http://tronicek.blogspot.com/2007/12/closures-closure-is-form-of-anonymous_28.html)

```
public class SimpleClosure {
  public static void main(String[] args) {
    // function with no arguments;
    int answer = { => 42 }.invoke();
    System.out.println(answer);
  }
}
```

(Megj.: ez nem valami jó példa.)

Szintén, aki Scala-ban akar programozni, programozzon Scala-ban.

KIVÉTELEK:

Multi-catch

Ha két kivétel catch ágában ugyanazt a kódot akarjuk használni, lehessen így egyszersíteni:

```
try {
} catch (IOException | SQLException ex) {
}
```

(Megj.: ha kiemeljük metódusba a használandó kódot az is segít.)

És akkor mi az "ex" változó típusa...?

Checked exception: Szükséges?

Kemény téma volt mindig is.

Néha úgysem tudunk értelmes catch kódot írni.
Sok helyen látok üres (vagy ami még roszabb alibi) catch blokkot.
Mi lenne, ha a checked exception kezelésének kihagyása csak warningot eredményezne?

Jó az a checked exception.

Stacktrace

```
row.getPerson().getAddress().getCity();

java.lang.NullPointerException:
    at myprogram.Main(Main.java:365)
```

Jó lenne tudni a soron belül hol történt a kivétel.

Ami null, annak kiírni a típusát, vagy a metódus nevét (Delphi csinálja így?).
Megadni valahogy, hogy mi kerüljön a stackTrace-be. Valakit zavarnak a hosszú stacktrace-ek is.
Egyébként le lehet cserélni az el-nem-kapott-exception kezelt sajátra, ami azt csinál amit akarunk a tömbben megkapott StackTraceElement-ekkel.

Egyebek

Több érték visszatérési értékek. Egy metódus tudjon visszaadni több értéket. (Akkor már kimen paraméterrel is megoldható.)
return type overloading. (Biztos sok további kérdést felvet.)
JVM lassú, modularizáció elkelne. *El.*
GC-hez közelebbi hozzáférés. Pl. cache-ek írásánál. Néha a WeakReference sem elég.
Appletek használhatatlanok. *Majd a javaFX!*
Dátumkezelés lassú. *Már tervezik a javítást!*
String ne legyen final. *No way.*
Minimalista API-k. *(Már nem annyira jellemz.)*