Újdonságok a Java nyelvben

Kozsik Tamás

kto@elte.hu
http://kto.web.elte.hu/

Eötvös Loránd Tudományegyetem
Programozási Nyelvek és Fordítóprogramok Tanszék

2008.

Tartalom

Tartalom

Tartalom

Tartalom

Tartalom

Tartalom

Tartalom

Tartalom

Tartalom

Sablonok

- Generikusok, generic
- Típussal való paraméterezhetőség

```
public interface List<E> extends Collectioni<E> {
    ...
    <T> T[] toArray(T[] a);
}
```

- A legfontosabb újítás
- Generikus programozás
  - Újrafelhasználhatóság
  - Típusbiztonság
- C++ template, Ada generic stb.

Auto-(un)boxing

- Automatikus átalakítás primitív típusok és csomagoló osztályok között
- Szintaktikus cukorka
- Kényelmes, de elfedi, hogy *költséggel jár*
- Kényelmes a sablonok használatánál

Oda-vissza működik

```
int i = new Integer(3);
Integer j = 3;
Integer k = new Integer(2) + new Integer(5);
ArrayList<Integer> seq = new ArrayList<Integer>();
seq.add(5);
```

Megizmosított léptetős ciklus

- Enhanced for-loop
- Iterálás adatszerkezeteken
- Szintaktikus cukorka
- Bizonyos iterálási idiómák kiváltására
- Rövidebbé, frappánsabbá, áttekinthetőbbé teszi a kódot

Szintaxis

```
for( Elemtípus változó: adatszerkezet ) törzs
```

Iterálás tömbökön

Régebben...

```
static double sum( double[] array ){
    double sum = 0.0;
    for( int i=0; i<array.length; ++i ){
        sum += array[i];
    }
    return sum;
}
```

Újabban:

```
static double sum( double[] array ){
    double sum = 0.0;
    for( double element: array ){
        sum += element;
    }
    return sum;
}
```

Iterálás tömbökön

Régebben...

```
static double sum( double[] array ){
    double sum = 0.0;
    for( int i=0; i<array.length; ++i ){
        sum += array[i];
    }
    return sum;
}
```

Újabban:

```
static double sum( double[] array ){
    double sum = 0.0;
    for( double element: array ){
        sum += element;
    }
    return sum;
}
```

Iterálás egyéb adatszerkezeteken

Kell, hogy `java.lang.Iterable` leszármazottja legyen

Ezt kell megvalósítani:

```
public interface Iterable<T> {
    java.util.Iterator<T> iterator();
}
```

Például a Collections Framework ilyen

- `java.util.Collection`
- És leszármazottjai...

Iterálás például egy sorozaton

Régebben...

```
static double sum( List seq ){
  double sum = 0.0;
  Iterator it = seq.iterator();
  while( it.hasNext() )
     sum += ((Double)it.next()).doubleValue();
  return sum;
}
```

Iterálás például egy sorozaton

Vagy akár...

```
static double sum( List seq ){
  double sum = 0.0;
  for( Iterator it = seq.iterator(); it.hasNext(); )
     sum += ((Double)it.next()).doubleValue();
  return sum;
}
```

Iterálás például egy sorozaton

Sablonokkal:

```
static double sum( List<Double> seq ){
  double sum = 0.0;
  for( Iterator<Double> it = seq.iterator(); it.hasNext();
     sum += it.next().doubleValue();
  return sum;
}
```

Iterálás például egy sorozaton

... és automatikus konverzióval:

```
static double sum( List<Double> seq ){
  double sum = 0.0;
  for( Iterator<Double> it = seq.iterator(); it.hasNext();
     sum += it.next();
  return sum;
}
```

Iterálás például egy sorozaton

Enhanced for-loop:

```
static double sum( List<Double> seq ){
  double sum = 0.0;
  for( Double element: seq )
     sum += element;
  return sum;
}
```

Változó számú argumentumok

- A C-ből ismert `printf` *olyan, de olyan* praktikus
- Kicsit veszélyes ugyan a végrehajtási veremmel játszani
- Lehet hasonlót csinálni egy szigorú(bb)an típusos nyelvben?
- Na, ez Javában a *varargs*
- Szintaktikus cukor
- Azonos típusú paraméterből 0, 1 vagy több, a paraméterlista végén
- Altípusosság? `Object`? Auto-(un)boxing?

Hogyan?

Definíció

```
static String concat( String... strings ){
    StringBuilder sb = new StringBuilder();
    for( String s: strings ){
        sb.append(s);
    }
    return sb.toString();
}
```

Használat

```
String s2 = concat("Hello ", "World!");
String s3 = concat("I ", "Say ", s2);
String s1 = concat(s2);
String s0 = concat();
```

Altípusosság?

Definíció

```
static String concat( Object... objects ){
    StringBuilder sb = new StringBuilder();
    for( Object o: objects ){
        sb.append(o.toString());
    }
    return sb.toString();
}
```

Használat

```
String s2 = concat("Hello ", "World!");
String s3 = concat(vector, s2, thread);
```

Auto-(un)boxing

Definíció

```
static double sum( Double... numbers ){
  double sum = 0.0;
  for( Double num: numbers ){
     sum += num;
  }
  return sum;
}
```

Használat

```
double d = sum( 3.1, 4.2, 5.6 );
```

printf

java.io.PrintStream

```
public PrintStream printf( String format,
                           Object... args )
```

Használat

```
System.out.printf( "The sum of %d and %d is %d.%n",
                   a, b, a+b );
```

Eredmény

```
The sum of 3 and 5 is 8.
```

sprintf?

java.lang.String

```
public static String format(String format,
                            Object... args)
```

Használat

```
String.format("<A HREF=\"%s\">%s</A>",address,tag)
```

Eredmény

```
<A HREF="http://kto.web.elte.hu/">My homepage</A>
```

Formázási lehetőségek gazdag választéka

Például

```
Calendar c = ...
String format =
       "Duke's Birthday: %1$th %1$te,%1$tY";
String s = String.format(format, c);
```

Eredmény

```
Duke's Birthday: May 23, 1995
```

Szintaxis

```
%[argument_index$][flags][width]conversion
```

Háttér

- Tömbre fordul, de meg van különböztetve a tömbtől
- Ugyanaz a megizmosított léptetős ciklus, mint tömb és `Iterable` esetén

Vararg

```
static double sum( Double... numbers ){
  double sum = 0.0;
  for( Double num: numbers ){ sum += num; }
  return sum;
}
```

Jelentése

```
static double sum( Double[] numbers ){
  double sum = 0.0;
  for( Double num: numbers ){ sum += num; }
  return sum;
}
```

Háttér

Vararg

```
static double sum( Double... numbers ){ ... }

sum( 3.0, 6.0 )
```

Jelentése

```
static double sum( Double[] numbers ){ ... }

sum( new Double[]{new Double(3.0), new Double(6.0)} )
```

Nem hívható így:

```
sum( new Double[]{3.0, 6.0, 7.0, 1.0} )
sum( new Double[]{new Double(3.0), new Double(6.0)} )
```

Kovariáns visszatérési érték

- Nyelvi változás (nem szintaktikus cukor)
- A lehetőség korábban is benne volt a JVM-ben
- Metódus felüldefiniálásánál szűkíthető a visszatérési érték típusa
  - Kovariáns: együtt változó
- Ugyanúgy, ahogy eddig a kiváltható ellenőrzött kivételek (speciális visszatérési értékek)
- A paraméterek típusa továbbra is invariáns
- A hozzáférési kategória is kontravariáns
- Vö. C++

Példa – mély másolat

```
interface Copyable {
  Copyable copy();
}

class Tree implements Copyable {
  double content;
  Tree left, right;
  Tree(double content){this.content = content;}
  public Tree copy(){
    Tree result = new Tree(content);
    if( left  != null ){ result.left  =  left.copy(); }
    if( right != null ){ result.right = right.copy(); }
    return result;
  }
}
```

Példa – mély másolat

```
interface Copyable {
  Copyable copy();
}

class Tree implements Copyable {
  double content;
  Tree left, right;
  Tree(double content){this.content = content;}
  public Tree copy(){
    Tree result = new Tree(content);
    if( left  != null ){ result.left  =  left.copy(); }
    if( right != null ){ result.right = right.copy(); }
    return result;
  }
}
```

Példa – Cloneable

```
class Tree implements Clonable {
    ...
    public Tree clone(){   // deep copy
        try {
            Tree result = (Tree)super.clone();
            if( left  != null ){ result.left  =  left.clone(); }
            if( right != null ){ result.right = right.clone(); }
            return result;
        } catch(CloneNotSupportedException e){
            assert false : "This simply cannot happen.";
            return null;
        }
    }
}

class SortTree extends Tree {
    ...
    public SortTree clone(){ return (SortTree)super.clone(); }
}
```

Kovariáns paraméter jó lenne?

```
interface Food {}

interface Animal {
    void eat( Food food );
}

class Grass implements Food {}

class Antelope implements Food, Animal {
    public void eat( Food food ){}
}

class Lion implements Animal {
    public void eat( Food food ){}
}
```

Kovariáns paraméter: szerencsére fordítási hiba

```
interface Food {}
interface Animal { void eat( Food food ); }
class Grass implements Food {}

class Antelope implements Food, Animal {
    public void eat( Grass food ){}
}

class Lion implements Animal {
    public void eat( Antelope food ) {}
}

class Problematic {
    public static void main( String[] args ){
        Animal lion = new Lion();
        lion.eat(new Grass());
    }
}
```

Kovariáns paraméternek tűnik, pedig túlterhelés

```
interface Food {}
class Animal { void eat( Food food ){} }
class Grass implements Food {}

class Antelope extends Animal implements Food {
    public void eat( Grass food ){}
}

class Lion extends Animal {
    public void eat( Antelope food ) {}
}

class Problematic {
    public static void main( String[] args ){
        Animal lion = new Lion();
        lion.eat(new Grass());
    }
}
```

Fordítási hiba kikényszerítése annotációval

```
interface Food {}
class Animal { void eat( Food food ){} }
class Grass implements Food {}

class Antelope extends Animal implements Food {
    @Override public void eat( Grass food ){}
}

class Lion extends Animal {
    @Override public void eat( Antelope food ) {}
}
```

Sablonok segítségével

```
interface Food {}

interface Animal<F extends Food> {
    void eat( F food );
}

class Grass implements Food {}

class Antelope implements Food, Animal<Grass> {
    public void eat( Grass food ){}
}

class Lion implements Animal<Antelope> {
    public void eat( Antelope food ) {}
}
```

Statikus tagok importálása

- static import
- Új nyelvi elem (nem csak szintaktikus cukor)
- Importálás: rövid névvel használhatóvá tevés
- Ezentúl nem csak típusokat lehet „importálni”

Korábbi import utasítások

- Típusok importálása
- Akár beágyazott típusoké is
- Importálható egy utasítással
  - Egy bizonyos típus megadása, vagy
  - Minden közvetlenül tartalmazott típus egy adott egységhez képest

Példák

```
import java.util.Vector;
import java.io.*;
import java.util.Map.Entry;
import java.text.Format.*;
```

Újdonság

- Nem csak típusok, hanem statikus tagok is importálhatók
  - Statikus metódusok
  - Statikus attribútumok
- `import static` utasítás
- Importálható egy utasítással
  - egy név, vagy
  - egy típus összes statikus tagja

Példa

```
import static java.util.Arrays.sort;
class Sorting {
    public static void main( String[] args ){
        sort(args);
        for( String s: args ){ System.out.println(s); }
    }
}
```

Felsorolási típusok

- Nyelvi újdonság
- A korábbi „szegény ember felsorolási típusa” helyett
- Sokkal biztonságosabb, mint a C++ `enum`-ja
- Kompatibilitás
  - Lényegében osztály
  - Szintaktikus és szemantikus támogatással
- Használat
  - Egyszerű esetekben tömör
  - Igény szerint cifrázható

Szegény ember felsorolási típusa

Nem típusbiztos

```
package java.awt;
public class Font implements java.io.Serializable {
    public static final int PLAIN  = 0;
    public static final int BOLD   = 1;
    public static final int ITALIC = 2;
    ...
}
```

Többszörözési problémák

```
package java.awt;
import java.io.Serializable;
public class Color implements Serializable, Paint {
    public static final Color RED;
    public static final Color BLACK;
    ...
}
```

Többszörözési problémák

„Típusbiztos enum”

```
public class Color implements Serializable, Paint {
    public static final Color RED;
    public static final Color BLACK;
    ...
}
```

- A konstruktort le kell tiltani (pl. priváttá tenni)
- Jól működik az `==`?
  - Különböző osztálybetöltők akkor is többszörözik
  - Reflection segítségével többszörözhető
  - Ha szerializálható, akkor úgy is többszörözhető
- Mindenesetre legalább az `equals`-t jól meg kell írni

Mindig félhetünk...

```
Color c1, c2;   ...
if( c1 != c2 ){ /* lehet, hogy mindkettő Color.BLACK */ }
```

Szegény ember felsorolási típusa és a statikus tagok importálása

Felsorolási típus

```
package java.awt;
import java.io.Serializable;
public class Color implements Serializable, Paint {
    public static final Color RED;
    public static final Color BLACK;
    ...
}
```

Használó kód

```
import static java.awt.Color;
import static java.awt.Color.*;

    Color c = BLACK;
```

A felsorolási típus megjelentével

Típusdefiníció

```
public enum Napok {
   HÉTFŐ, KEDD, SZERDA, CSÜTÖRTÖK, PÉNTEK,
   SZOMBAT, VASÁRNAP;
}
```

- Többszörözési problémák nem lépnek fel
  - Nem lehet példányosítani (fordítási hiba)
  - Nem lehet klónozni (`final clone`)
  - Szerializációs támogatás
  - Reflektív példányosítás is kizárva
- Az == és a != jól működik
- Nem lehet belőlük származtatni (`final`)
- Rögzített implicit szülőosztály (`java.lang.Enum<E>`)

Műveletek

- Megörökölt műveletek

```
public abstract class Enum<E extends Enum<E>>
implements Comparable<E>, Serializable
```

  - sorszám (`ordinal`)
  - összehasonlítás (`compareTo`)
  - stringkonverziók (`toString`, `valueOf`)
- Tömb a típusértékekből (automatikusan jár)

```
public static Napok[] values()
```

- Definiálhatunk további műveleteket
  - És persze konstruktorokat, attribútumokat stb.
  - Mint a közönséges konkrét osztályok esetén

Néhány szép kódrészlet

Iterálás a típusértékeken

```
for( Napok nap: Napok.values() ) System.out.println(nap);
```

Iterálás a típusértékek egy részsorozatán

```
for( Napok nap: EnumSet.range(Napok.HÉTFŐ,Napok.PÉNTEK) )
   System.out.println(nap);   // olcsó megoldás
```

Támogatás elágazáshoz

```
int munkaÓrák;
switch(nap){
   case HÉTFŐ:
   case KEDD:
   case CSÜTÖRTÖK: munkaÓrák = 9; break;
   case SZERDA:    munkaÓrák = 7; break;
   case PÉNTEK:    munkaÓrák = 6; break;
   default:        munkaÓrák = 0;
}
```

Adatszerkezetek

Hatékony halmaz és leképezés

- `EnumSet<E extends Enum<E>>`
  - Bitvektorral ábrázolva
  - Kisebb típusértékhalmaz esetén egy `long`
  - Még a metszet, unió stb. műveletek is konstans idejűek tudnak így lenni
- `EnumMap<K extends Enum<K>,V>`
  - Tömbbel ábrázolva

Saját műveletek definiálása

```
public enum Coin {
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);
    Coin(int value) { this.value = value; }
    private final int value;
    public int value() { return value; }
}
class CoinTest {
    public static void main(String[] args) {
        for (Coin c : Coin.values())
            System.out.println(c+"\t"+c.value()+"c\t"+color(c));
    }
    private enum CoinColor { COPPER, NICKEL, SILVER }
    private static CoinColor color(Coin c) {
        switch(c) {
          case PENNY: return CoinColor.COPPER;
          case NICKEL: return CoinColor.NICKEL;
          case DIME: case QUARTER: return CoinColor.SILVER;
          default: throw new AssertionError("Unknown: "+c);
        }
    }
```

Még példák

- `Planet`
- `Operation`

Annotációk

Máshol...

Programkönyvtárak

- Konkurrencia
- JWS
- Management
- XML
- Tools