Budapesti Műszaki és Gazdaságtudományi Egyetem

Java technológia

A Java nyelv alapjai

Nyelvi elemek

A nyelv definíciója

- **The Java Language Specification**, Third Edition
  http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html
- JSR-176 (Java5), JSR-270 (Java6), JSR
- A továbbiakban: **JLS**, hivatkozás a JLS megfelelő helyére: [§1.2], vagyis a JLS 1. fejezetének 2. pontja.
  - Kezdetben: JLS, First Edition
  - Java 2 SE 1.2 óta: JLS, Second Edition
  - Java 2 SE 5.0-óta: JLS, Third Edition
- A nyelvben csak egy adott verzió óta létező elemek a verzió számával vannak megjelölve: **[1.4]**, **[5.0]**.

Menetrend: nyelvi elemek

- Lexikai struktúra
- Adattípusok
- Operátorok
- Utasítások

Lexikai struktúra

- Irodalom: JLS 3. fejezet: Lexical Structure

Karakterkódolás 1.

- Cél: a forráskódok is hordozhatók legyenek.
- Megoldás: A forráskódok Unicode karakterekből állnak (következmény: tetszőleges karakter előfordulhat bennük, magyar ékezetes betűk, japán, kínai írásjelek, stb.). [§3.1]
- Mivel a forráskódok szövegfájlok, ezek kódolása operációs rendszertől függ (legalább).
- A forráskódokban előfordulhatnak úgynevezett **Unicode escape szekvenciák**, amelyekkel tetszőleges Unicode karaktert megadhatunk. [§3.3]

Karakterkódolás 2.

- A Unicode escape-ek formátuma a következő:

```
\u1234
```

- ahol az "u" után 4 hexadecimális számjegy áll.
- A Unicode escape-eket a fordító a feldolgozás legelső lépésében értelmezi, és kicseréli a megfelelő Unicode karakterre.
- Következmény: a forráskód tetszőleges lexikai elemét megadhatjuk így (pl. stringet határoló idézőjelet, stb.).

Lexikai alapegységek 1.

- A forrásfájlok
  - **token**ekből,
  - a köztük lévő **üres karakterek**ből (white space),
  - **sorvége jelek**ből,
  - **megjegyzések**ből (comment) és
  - (opcionálisan) **fájl vége jel**ből (EOF, 0x04) állnak.
- A forrásfájlokban a sorok végét az alábbi karakterkombinációk valamelyike jelöli [§3.4]:
  - ASCII LF (0x0a, newline),
  - ASCII CR (0x0d, return),
  - ASCII CR ASCII LF (0x0d 0x0a).

Lexikai alapegységek 2.

- Az üres karakterek az alábbiak [§3.6]:
  - ASCII SP (0x20, space),
  - ASCII HT (0x09, horizontal tab),
  - ASCII FF (0x0c, form feed),
  - sorvége jel.
- A megjegyzéseket kétféleképpen lehet jelölni [§3.7]:
  - egysoros megjegyzés: `//` *megjegyzés*
  - többsoros megjegyzés: `/*` *megjegyzés* `*/`
- Megjegyzések nem ágyazhatók egymásba.

Azonosítók

- **Név**, **azonosító** (identifier) [§3.8]:
  - Első karaktere: amelyre a Character.isJavaIdentifierStart metódus true-t ad (pl. kis- és nagybetűk, _ és $ karakterek, de nem számjegy). Ajánlott: betű
  - További karakterek: amelyre a Character.isJavaIdentifierPart metódus true-t ad (pl. kis- és nagybetűk, _ és $ karakterek, számjegyek).
  - Az azonosító nem lehet kulcsszó (ld. később), boolean literál (`true` és `false`), vagy null literál (`null`).
- Azonosnak "tűnő" karakterek gyakran különböznek, például:
  - latin kis a (Unicode 0x0061),
  - cirill kis a (Unicode 0x0430).

Kulcsszavak

abstract	**assert**	boolean	break
byte	case	catch	char
class	**const**	continue	default
do	double	else	**enum**
extends	final	finally	float
for	**goto**	if	implements
import	instanceof	int	interface
long	native	new	package
private	protected	public	return
short	static	strictfp	super
switch	synchronized	this	throw
throws	transient	try	void
volatile	while		

- A **kulcsszavak** [§3.9] közül a pirossal szedett szavak nem használtak.
- Az **assert** a nyelv **[1.4]**-es, az **enum** pedig az **[5.0]**-ás verziója óta minősül kulcsszónak.

Literálok

- További foglalt nevek:
  - Boolean literálok [§3.10.3]:
    - true és false
  - Null literál [§3.10.7]:
    - null

Adattípusok

- Irodalom: JLS 4. fejezet: Types, Values, and Variables

Adattípusok

- A Java adattípusok két csoportra oszthatók:
  - **Primitív típusok** (primitive types) [§4.2]
    - boolean
    - byte, short, int, long, char
    - float, double
  - **Referencia típusok** (reference types) [§4.3]
    - Objektum
    - Paraméterezett típusok [§4.5] – *későbbi előadáson*

Primitív típusok 1.

- **Egész típusok** (integral types) [§4.2.1]:
  - Egész típusok: byte, short, int, long
  - Karakter típus: char
- **Logikai típus** [§4.2.5]: boolean
- **Lebegőpontos típusok** [§4.2.3]: float, double

Egész típusok

- A Java-ban a karakter típus (char) 16 bites, mivel egy Unicode karaktert tárol UTF-16 kódolásban, a stringek (ld. később) belső reprezentációja ezzel szemben UTF-8 kódolást alkalmaz, **[5.0]** a Java API egyes elemei pedig lehetővé teszik UTF-32 formátumú 32 bites Unicode karakterek használatát is.

Név	Típus	Alapérték	Méret	Értékkészlet
byte	előjeles egész	0	8 bit	-128 … 127
short	előjeles egész	0	16 bit	-32768 … 32767
int	előjeles egész	0	32 bit	-2^{31} … $2^{31}-1$
long	előjeles egész	0	64 bit	-2^{63} … $2^{63}-1$
char	egész	\u0000	16 bit	\u0000 … \uffff

Logikai típus

Név	Típus	Alapérték	Méret	Értékkészlet
boolean	logikai	false	8 bit	false vagy true

Lebegőpontos típusok

- A lebegőpontos típusok speciális értékeket is felvehetnek, melyeket a Float és Double burkoló osztályokban (ld. később) definiált konstansok jelölnek:
  - NaN (Not-a-Number): 0.0 / 0.0
  - POSITIVE_INFINITY: például 1.0 / 0.0
  - NEGATIVE_INFINITY: például -1.0 / 0.0

Név	Típus	Alapérték	Méret	Értékkészlet
float	IEEE 754 lebegőpontos	0	32 bit	±3.40E+38 … ± 1.4E-45
double	IEEE 754 lebegőpontos	0	(legalább) 64 bit	± 1.797E+308 … ± 4.94E-324

Long és double literálok

- Ha csak egy „egyszerű” egész számot írunk be, annak típusa int-ként van értelmezve
- Ennek következménye:

```
double d1 = 1 / 2;  // == 0.0 !!!
long l1 = 10000000000; // Le sem fordul!!!

double d2 = 1.0 / 2.0; // == 0.5
long l2 = 10000000000L; // OK
```

- **A nem int típusú literálokat mindig jelöld a megfelelő végződéssel!!!**
  - **double esetén .0 végződés ha egész**
  - **float esetén .0f végződés ha egész (de „f” mindenképpen kell!)**
  - **long esetén „L” végződés**

Lebegőpontos típusok - NaN

- Valamilyen érvénytelen lebegőpontos értéket jelent, leggyakrabban 0.0/0.0 eredményeképpen
- **Minden NaN értéken végzett művelet eredménye NaN!**
- **Minden NaN értéken végzett összehasonlító művelet értéke NaN! (Még akkor is, ha a másik érték is NaN)**

```
double nan1 = 0.0 / 0.0; // == NaN
double nan2 = nan1 * 0.0; // == NaN !!!

boolean isNan1 = (nan1 == nan2); // == false !!!
boolean isNan2 = (nan1 == Double.NaN); // == false !!!

boolean isNan3 = Double.isNaN(nan1); // == true
```

Burkoló osztályok

- A primitív típusok nem objektumok (ellentétben például a C#-pal), nem lehet rájuk referenciával hivatkozni. *Ennek jelentősége majd a Collection Framework osztályoknál fog megmutatkozni.*
- **Burkoló osztály** (wrapper class) → objektumként kezelhető
- A burkoló neve általában a primitív típus nevének nagy kezdőbetűs változata:
  - byte → Byte, short → Short, int → Integer, long → Long,
  - char → Character,
  - boolean → Boolean,
  - float → Float, double → Double
- Java 1.5 óta „autoboxing” van, azaz a burkolást elvégzik helyettünk

Egész és lebegőpontos literálok

Literál	**Típus**	**Radix**
1 432 -22 0	int	10
01 0777	int	8
0x1 0xffFF	int	16
1l -3L	long	10
0x1L 0xffFFl	long	16
1.0 1e6 1E6 1D 1d	double	10
1.0f 1e6f 1E6F 1F 1f	float	10

Speciális karakterek

Escape szekvencia	**Jelentés**
\\	backslash (\)
\'	aposztróf (')
\"	idézőjel (")
\r	CR (kocsivissza)
\n	LF (újsor)
\f	FF (lapdobás)
\t	HT (tab)
\b	BS (backspace)
\uXXXX	\u0000 és \u00ff közötti kódú karakter oktálisan megadva

Típuskonverziók

- Irodalom: JLS 5. fejezet: Conversions and Promotions

Típuskonverziók 1.

- A típuskonverziók két típusa létezik a Java-ban:
  - **automatikus** (implicit),
  - **nem automatikus** (explicit, type casting).
- Automatikus típuskonverziók csak a bővítő irányban történnek, minden más esetben explicit konverziót kell alkalmazni.

```
long a = 25;                                    // OK, int -> long
int b = 25.0;                                 // Hiba: double -> int
int c = (int) 25.0;                         // OK, explicit konverzió
```

Típuskonverziók 2.

- Kétoperandusú művelet: az operandusok típusa dönti el az eredmény típusát
- byte, short és char operandus a műveletben int-ként vesz részt
- Az eredmény típusa mindig a két operandus közül a bővebb típusú operandus típusával egyezik meg:
  - long és int → long,
  - float és előbbiek → float,
  - double és előbbiek → double.

```
double d = 13/2;                                    // OK, d értéke 6
double e = (double) 13/2;                          // OK, e értéke 6.5
double f = 3/0;                 // Hiba: int eredmény, division by zero
double g = 3.0/0;             // OK, double eredmény, POSITIVE_INFINITY
```

Típuskonverziók 3.

- A típuskonverziók másik lehetséges csoportosítása a konverzióban részvevő típusok szerint történhet:
  - Identitás konverzió. (Azonos típusok közötti "konverzió".)
  - Bővítő/szűkítő primitív konverzió. (Lásd fentebb.)
  - Bővítő/szűkítő referencia konverzió. (Lásd az objektum-orientáltságnál.)
  - Boxing/unboxing konverzió. **[5.0]** (Lásd a következő slide-on.)
  - Ellenőrzetlen konverzió. (Lásd a generikus típusoknál.) **[5.0]**
  - Capture konverzió. (Lásd a generikus típusoknál.) **[5.0]**
  - String konverzió. (Lásd a String osztály tárgyalásánál.)
  - Értékkészlet konverzió. (Lásd a következő slide-on.)

Típuskonverziók 4.

- A **boxing/unboxing konverzió** a primitív típusok és burkoló osztályaik közötti automatikus konverziót jelenti. **[5.0]**
- Boxing konverzió két esetben történhet:
  - egy metódus formális paraméterlistájában burkoló osztály típusú paraméter áll, viszont híváskor a megfelelő primitív típusú paramétert adjuk át (method invocation conversion),
  - egy burkoló osztály típusú változónak, vagy mezőnek primitív típusú értéket adunk (assignment conversion).
- Az unboxing konverzió a fenti két eset ellenkező irányát jelenti.
- **Értékkészlet konverzió** csak lebegőpontos típusok esetén történhet, nem FP-strict kiértékelés esetén. Ekkor részeredményként a lebegőpontos típusok kitevő-tartományán kívül eső értékek is ábrázolhatók. [§15.4]

Operátorok

- Irodalom: JLS 15. fejezet: Expressions

Operátorok

- Az alábbi táblázatok a Java operátorait tartalmazzák.
- A táblázat egyes sorai azonos precedenciájú operátorokat tartalmaznak.
- Az operátorok csökkenő prefecedencia sorrendjében helyezkednek el.

Operátor	Operandus típusa	Zárójelezés	Jelentés
[ ] . ( ) ++ --	tömb, egész referencia, azonosító függvény, argumentumok numerikus numerikus	⇨	index osztály, objektum eleme függvényhívás postincrement postdecrement
++ -- + - ~ ! ( ) new	numerikus numerikus numerikus egész boolean típus, kifejezés típus	⇦	preincrement predecrement előjel bitenkénti negálás logikai negálás explicit típuskonverzió példányosítás
* / %	aritmetikai	⇨	szorzás, osztás, modulo
+ - +	aritmetikai String	⇨	összeadás, kivonás string konkatenáció

Operátor	Operandus típusa	Zárójelezés	Jelentés
<< >> >>>	egész egész egész	⇨	bitléptetés balra bitléptetés jobbra, bal oldalon előjel bit lép be bitléptetés jobbra, bal oldalon 0 lép be
< <= > >= instanceof	aritmetikai aritmetikai referencia, típus	⇨	kisebb reláció nagyobb reláció típus kompatíbilitás vizsgálat
== !=	tetszőleges tetszőleges	⇨	egyenlő nem egyenlő
& &	egész boolean	⇨	bitenkénti AND logikai AND

Operátor	Operandus típusa	Zárójelezés	Jelentés
^ ^	egész boolean	⇨	bitenkénti XOR logikai XOR
| |	egész boolean	⇨	bitenkénti OR logikai OR
&&	boolean	⇨	logikai AND, rövidrezárt kiértékelés
||	boolean	⇨	logikai OR, rövidrezárt kiértékelés
? :	boolean, tetszőleges	⇦	feltételes kifejezés
= *= /= %= += -= <<= >>= >>>= &= ^= |=	változó, kifejezés változó, kifejezés	⇦ ⇦	értékadás művelet és értékadás

Néhány különbség a C és a Java között

C	**Java**
->	nincs
sizeof	nincs
& (címe operátor)	nincs
* (pointer indirekció)	nincs
függvény argumentumok kiértékelése tetszőleges sorrendben	függvény argumentumok kiértékelése balról jobbra
rövidrezárt kiértékelés nem csak logikai AND és OR esetén történhet például: 0*f() esetén f nem biztos, hogy végrehajtódik	rövidrezárt kiértékelés csak logikai AND és OR esetén történik

Utasítások

- Irodalom: JLS 14. fejezet: Blocks and Statements

Utasítások

- A Java utasításai:
  - kifejezés,
  - blokk ({ }),
  - lokális változódeklaráció,
  - lokális osztálydeklaráció (lásd később),
  - feltételes elágazások (if, switch),
  - ciklusszervező utasítások (for, while, do),
  - vezérlésátadó utasítások (return, break, continue),
  - assert utasítás (lásd később) **[1.4]**,
  - kivételkezelő utasítások (try, throw) (lásd később),
  - szikronizáció (synchronized) (lásd később).

A kifejezés utasítás

- A **kifejezés** utasítás alakja [§14.8]:

```
kifejezés ;
```

- A Java nem engedi meg a hatástalan kifejezések utasításként való használatát.
- Az alábbi példában szereplő f függvény egy int értéket ad vissza:

```
int a = f();                                                    // OK.
f();                          // OK, a visszatérési értéket eldobhatom.
2*3;                                    // Hiba, a kódnak nincs hatása.
```

A blokk utasítás

- A **blokk** utasítás alakja [§14.2]:

```
{
   utasítás1;
   utasítás2;
   ...
   utasításN;
}
```

- A blokk utasításban szereplő utasítások szekvenciálisan hajtódnak végre.
- A blokk végrehajtása return, break, continue utasítás, vagy kivétel (ld. később) hatására megszakadhat.

A lokális változódeklaráció

- A **lokális változódeklaráció** alakja [§14.4]:

```
típus azonosító1 [= érték1], ..., azonosítóN [= értékN];
```

- A lokális változódeklaráció utasítás a megadott típusú lokális változók létrehozására szolgál. Egyszerre több, azonos típusú változót is deklarálhatunk.
- A lokális változók érvényességi köre az utasítást befoglaló blokk.
- A lokális változókat használat előtt mindig inicializálni kell, különben fordítási hibát kapunk.

```
int a = 4, b , c;

a = 0;                                                    // OK
c = b;                                                    // Fordítási hiba
```

Feltételes elágazások I.

- Az **if** utasítás alakja [§14.9]:

```
if (kifejezés)
   utasítás;
```

- vagy:

```
if (kifejezés)
   utasítás;
else
   utasítás;
```

Feltételes elágazások II.

- A **switch** utasítás alakja [§14.10]:

```
switch (kifejezés) {
   case konstans1 : ...
   case konstans2 : ...
   ...
   case konstansN : ...
   default : ...
}
```

- A kifejezés és a case címkékben szereplő konstansok csak egész vagy boolean típusúak lehetnek. *Java7: String switch is van, lásd későbbi előadás*
- A case címkék a switch blokkba való belépési pontokat jelölik, a végrehajtás nem áll meg a következő case címkénél!
- Ha a kifejezés értéke egyik case címkének sem felel meg, az opcionális default címkénél kezdődik a végrehajtás.

Feltételes elágazások III.

- Példa a switch utasítás viselkedésére:

```
switch (a) {
   case 1 : System.out.print (1); break;
   default : System.out.print (0);
   case 2 : System.out.print (2); break;
   case 3 : System.out.print (3);
   case 4 : System.out.print (4);
}
```

- a=1 esetén

```
1
```

- a=5 esetén

```
02
```

- a=3 esetén

```
34
```

Ciklusszervező utasítások I.

- A **for** utasítás alakja [§14.13]:

```
for (kifejezés1;kifejezés2;kifejezés3)
   utasítás;
```

- A kifejezés1 a ciklusmag első végrehajtása előtti inicializálást végzi. Lehet egy, vagy több változó deklaráció, vagy ","-vel elválasztva több utasítás.
- A kifejezés2 egy boolean kifejezés, a ciklusmag csak akkor hajtódik végre, ha kifejezés értéke true.
- A kifejezés3-ban ","-vel elválasztva a ciklusmag végrehajtása után végrehajtandó utasítások szerepelnek.
- A három kifejezés bármelyike, vagy akár mindegyike elhagyható, kifejezés2 alapértéke true (végtelen ciklus).

Ciklusszervező utasítások II.

- Példa a for utasítás használatára:

```
for (int i = 0;i < 10;i++)
  utasítás;
```

- A fenti példában az utasítás 10-szer hajtódik végre.

```
for (;;)
  utasítás;
```

- A fenti példa végtelen ciklust valósít meg.

Ciklusszervező utasítások III.

- A for utasítás másik alakja: **[5.0]**

```
for (típus azonosító : kifejezés)
   utasítás;
```

- A kifejezés értéke "iterálható" kell legyen (lásd később).
- A ciklusváltozó a ciklusmag végrehajtásai során végiglépked az "iterálható" kifejezés lehetséges értékein.
- A végrehajtások száma az "iterálható" kifejezés lehetségés értékeinek számával egyezik meg.

Ciklusszervező utasítások IV.

- A **while** utasítás alakja [§14.11]:

```
while (kifejezés)
   utasítás;
```

- A kifejezés egy boolean kifejezés.
- A ciklusmag akkor hajtódik végre, ha a kifejezés értéke true.
- Egy for ciklus megfeleltethető egy while ciklusnak:

```
kifejezés1;
while (kifejezés2) {
   utasítás;
   kifejezés3;
}
```

Ciklusszervező utasítások V.

- A **do** utasítás alakja [§14.12]:

```
do {
   utasítás;
} while (kifejezés);
```

- A kifejezés egy boolean kifejezés.
- A ciklusmag legalább egyszer végrehajtódik, de csak akkor hajtódik végre újra, ha a kifejezés értéke true.

Vezérlésátadó utasítások I.

- A **return** utasítás alakja [§14.16]:

```
return;
```

- vagy:

```
return kifejezés;
```

- A return utasítás kilép a végrehajtás alatt álló függvényből.
- Az első esetben a függvény nem ad vissza értéket, ez akkor lehetséges, ha a függvény visszatérési típusa voidnak deklarált.
- A második esetben a függvény visszatérési típusa a kifejezés értéke lesz.

Vezérlésátadó utasítások II.

- A **break** utasítás alakja [§14.14]:

```
break;
```

- vagy:

```
break címke;
```

- A break utasítás kilép a végrehajtás alatt álló blokkból.
- A második esetben a címkével jelölt helyre lép ki (ld. később).

Vezérlésátadó utasítások III.

- A **continue** utasítás alakja [§14.15]:

```
continue;
```

- vagy:

```
continue címke;
```

- A continue utasítás a végrehajtás alatt álló ciklusmag végére ugrik.
- A második esetben a címkével jelölt ciklus magjából lép ki (ld. később).

Vezérlésátadó utasítások IV.

- A break utasítás használata címkével:

```
for (int i = 0;i < 10;i++) {
   cimke: {
      if ( ... ) break cimke;
   }
   a += 5;
}
```

- A fenti példában a break utasítás nem a ciklusból, hanem csak a cimkével jelölt blokkból lép ki, az a += 5 utasítás még végrehajtódik.
- A continue utasítást esetén a célcímke csak közvetlenül for, do és while ciklusok előtt állhat. Ekkor a continue utasítás a címkével jelölt ciklusutasítás elejére ugrik és új iterációt kezd (természetesen csak akkor, ha a leállási feltétel nem vált igazzá)