Bevezető a Java világába

Simon Károly
simon.karoly@codespring.ro

Java

- Magas szintű, bytecode alapú, objektumorientált.
- Virtuális gép (VM - Virtual Machine): egy számítógép szoftver implementációjának tekinthető, amely képes végrehajtani a bytecode utasításait.
- JIT (Just in Time) fordítók alkalmazása, optimalizálás.
- AOT (Ahead of Time) fordítók alkalmazásának lehetősége.

Java életút „dióhéjban”

- 1960-as évek Simula 67: objektumok; 1970-es évek, XEROX Parc, Smalltalk: OOP; 1983 C++.
- 1991 Sun Microsystems, Stealth (Green) Project: „okos”, hálózatba köthető elektronikai készülékek összekapcsolása, egy heterogén de központilag irányítható rendszerbe.
- Bill Joy, Mike Sheridan, Patrick Naughton, **James Gosling**
- Oak → Java
- 1994 Patrick Naughton: Hot Java Browser (WebRunner); Netscape támogatás
- 1995 Sun JavaSoft
- 1996 JDK (Java Development Kit) 1.0

Java életút „dióhéjban”

- 1998 J2SE 1.2 (Java 2 Platform Standard Edition) (JDK 1.2), J2EE (Java 2 Platform Enterprise Edition), J2ME (Java 2 Platform Micro Edition)
- J2EE: alkalmazásszerverek, komponens alapú fejlesztések, osztott rendszerek, web-programzás (EJB – Enterprise Java Beans, JPA – Java Persistence API, JTA – Java Transaction API, JMS – Java Message Service, Servlet API, JSP – JavaServer Pages, JSF – JavaServer Faces stb.)
- 2005: 2.5 milliárd elektronikus eszközön van jelen, kb. 4.5 millió fejlesztő használja, a fejlesztők 75% tekinti „elsődleges” programozási nyelvének.
- 2009-2010: a Sun Microsystems megvásárlása az Oracle által
- Közel egymilliárd JRE letöltés évente, közel 10 millió fejlesztő, több milliárd eszköz (több mint 3 milliárd mobil).

Java életút „dióhéjban”

- Fejlesztői környezetek: Borland JBuilder, Oracle JDeveloper, IntelliJ IDEA, Xinox JCreator stb.
- Érdekesség: Microsoft J++ (MSJVM, a Visual Studio 6.0-nak még része), utódja a J# (a .NET keretrendszer részét képezte).
- NetBeans (netbeans.org):
  - 1997-ben indul egyetemi projektként.
  - 1999-ben a Sun megvásárolja majd nyílt forráskódúvá teszi.
- Eclipse (eclipse.org):
  - 1998-ban indul IBM fejlesztésként, 2001-ben megalapul a több nagyvállalatból álló Eclipse konzorcium, majd 2004-ben az Eclipse Foundation.
  - 2001-től nyílt forráskódú, „lelke” az OSGi (Open Service Gateway Initiative) standard Equinox implementációja.
- „Rokon” nyelvek: Groovy, JRuby, Jython stb.

Java SE verziótörténet

- 1996: JDK 1.0
- 1997: JDK 1.1 - belső osztályok, JDBC, RMI, reflection stb.
- 1998: J2SE 1.2 (Playground) – SWING, JCF, JIT fordító stb.
- 2000: J2SE 1.3 (Kestrel) – HotSpot, RMI-IIOP, JNDI stb.
- 2002: J2SE 1.4 (Merlin) – „láncolt” kivételek, java.nio, logging, JAXP, biztonsággal és kriptográfiával kapcsolatos kiegészítések stb.
- 2004: J2SE 5.0 (Tiger) – generics, enumerations, auto-boxing, annotációk, for each ciklus, concurrency csomag stb.
- 2006: Java SE 6 (Mustang) – több hatékonyságot növelő átalakítás, javítások, kiegészítések.
- 2011: **Java SE 7** (Dolphin) – apró nyelvi átalakítások (kivételkezelés, switch stb.), további kiegészítések.
- 2014: Java SE 8 (Spider…) – lambda kifejezések, alapértelmezett metódusok, dátum és idő API, ismétlődő annotációk stb.

Fontosabb alapelvek

- Platformfüggetlenség
- Megbízhatóság:
  - Példa: egyszerű számítógépes program „lefagyása” vs. mobiltelefon/bankautomata szoftvere → cél a programozók tévedési lehetőségeinek csökkentése.
  - Csak objektumok használhatóak adatstruktúraként.
  - A kivételkezelés nagyobb szerepet kap.
  - A többszörös öröklés nem megengedett.
  - A memóriakezelést automatikus „szemétgyűjtő” (garbage collector) teszi hatékonyabbá.
- Biztonság
  - Pl. nem használunk pointereket, így „kártékony” programok nem férhetnek hozzá „nem megengedett” memóriarészekhez.
  - Homokverem (sandbox) mechanizmus.

A színfalak mögött

- Futási környezet: JRE (Java Runtime Environment) – tartalmazza a JVM –et, a futtatáshoz szükséges indítót (launcher) és dinamikusan betölthető osztályokat tartalmazó osztálykönyvtárakat.
- A JVM nyílt forráskódú, Sun implementációja a HotSpot (kliens és szerver fordító, JIT fordító alkalmazása).
- Alternatív lehetőség: AOT fordító alkalmazása (Excelsior Jet, GNU Compiler).
- A Java program bytecode formájában, .class állományokban, vagy csomagokban (.zip vagy .jar állományok) jut el a felhasználóhoz.
- Indításkor a JRE megkapja egy .class állomány nevét, elvégez bizonyos ellenőrzéseket, majd megkeresi és (ha van) futtatja a main metódust.

A színfalak mögött

- Általában szükség van külső osztályokra, amelyek lehetnek a Java Platform standard osztályai (bootstrap osztályok) (pl. a java.lang és java.util csomagok osztályai stb.), kiterjesztések (opcionális csomagok), vagy a felhasználó saját osztályai.
- A JVM-nek meg kell találnia ezeket az osztályokat:
  - A bootstrap osztályok és kiterjesztések keresése automatikus (a JRE főkönyvtárának /lib és /lib/ext alkönyvtáraiban találhatóak, a legtöbb fontos, központi osztály a /lib/rt.jar csomagban).
  - A saját osztályok esetében meg kell határozni az útvonalat az operációs rendszer CLASSPATH környezeti változójának beállításával.

Példa

- Hello World helyett: a parancssor argumentumainak kiírása a konzolra:

```
public class Example {
        public static void main(String[] args) {
                for (int i = 0; i < args.length; i++)
                        System.out.println(args[i]);
                }
        }
```

- Example.java
- Fordítás: javac Example.java (→ Example.class), indítás: java Example
- Megjegyzések: UNICODE karakterek, állomány neve, kódolási és elnevezési konvenciók

Primitív adattípusok

- Primitív adattípusok: byte (előjeles egész, 8 bit), short (előjeles egész, 16 bit), int (előjeles egész, 32 bit), long (előjeles egész, 64 bit), char (egész, 16 bit, UTF 16 kódolás), float (lebegőpontos, 32 bit), double (lebegőpontos, 64 bit), boolean (logikai, általában 8 bit – JVM függő, értékkészlet: true/false)
- Burkoló osztályok (wrapper classes): Byte, Short, Integer, Long, Character, Float, Double, Boolean

Példák:

```
int i1 = 5;
Integer i2 = new Integer(5);
Integer i3 = new Integer(i1);
String s = "10";
int i1 = Integer.parseInt(s);
```

```
i2 = new Integer(i1);
float f = i2.floatValue();
char c = 'a';
if (Character.isLowerCase(c))
    c = Character.toUpperCase(c);
```

Referencia típusok

- Objektumok beazonosítása: referenciák segítségével.
- Referencia: a pointer fogalmához áll közel, de nem adja meg a C++ esetében rendelkezésünkre álló „szabadságot” (pointer operátorok).
- Példák:

```
String s1 = "Hello";
String s2 = s1;
String s3;
s3 = new String("Hello");
```

- A referencia implicit értéke null, a helyfoglalás a new kulcsszóval történik

Referencia típusok

- **Figyelem** az értékek másolásának és összehasonlításának esetében (clone és equals metódusok)!
- A tömbök és objektumok beazonosítása mindig referenciák által történik, de a paraméterátadás **mindig érték általi** (pass by value) (nem pass by reference): ha egy objektumot adunk át, a referencia érték általi átadásáról beszélhetünk (átadáskor a referenciáról másolat készül), a referencián keresztül történő változtatások a metódusból való kilépés után is érvényben maradnak

Autoboxing és auto-unboxing

- Tiger: autoboxing és auto-unboxing mechanizmusok bevezetése
- Példa:

```
int i;                          int i;
Integer j;                      Integer j;
i = 1;                    →     i = 1;
j = new Integer(2);             j = 2;
i = j.intValue();               i = j;
j = new Integer(i);             j = i;
```

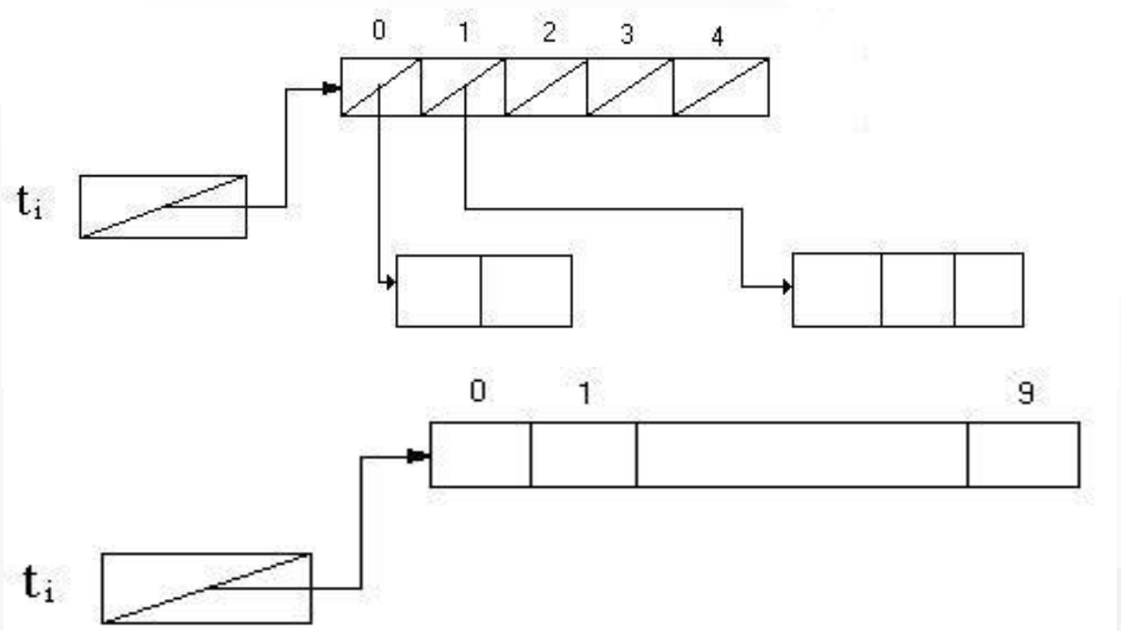
- Alkalmazás példa:

```
int i1 = 10;
Vector v = new Vector();
v.add(i1);
```

- Kényelmesebb írásmódot tesznek lehetővé, de a megfelelő „óvatossággal” kezelendőek. Pl. a „==” logikai operátor továbbra is az objektumok azonosságát fogja tesztelni, az érték szerinti összehasonlítás equals metódussal történik (érdemes erre nagyon odafigyelni, főként, hogy a wrapper class caching mechanizmus további félreértésekhez vezethet)

Tömbök

- Objektumok, az Object ősosztály leszármazottai
- A tömbök elemei azonos típusúak, méreteik mindig ismertek, a tömbhatárok mindig ellenőrzésre kerülnek, átlépésük kivételt eredményez.
- A tömbök elemei lehetnek tömbök (a többdimenziós tömbök tulajdonképpen tömbökből álló tömbök).
- Példák:
  - ```
    int ti[];
    ti = new int[5];
    ti = {1, 2, 3, 4, 5};
    ti = new int[10];
    for (int i = 0; i < 10; i++)
            ti[i] = i;
    ```
  - ```
    int ti[][] = new int[5][];
    ti[0] = new int[2];
    ti[1] = new int[3];
    ```



Kivételkezelés dióhéjban

```
try {
    // Kódrészlet, amely kivételt eredményezhet
} catch(Exception1 object1) {
    // Az Exception1 kivétel kezelésének megfelelő kód
} catch(Exception2 object2) {
    // Az Exception2 kivétel kezelésének megfelelő kód
} finally {
    // Ez a kódrészlet minden esetben végre lesz hajtva
}
```

- A kivételek valamilyen ágon az Exception ősosztály leszármazottai
- **Figyelem:** egy kivétel felléptekor az adott kivétel típusának megfelelő első catch ágon belüli kód kerül lefuttatásra. Ha a figyelt kivétel típusok egymás leszármazottai fontos a catch ágak sorrendje (elérhetetlen kód → fordítási hiba).

Kivételkezelés példa

```
public class ExceptionExample {
    public static void main(String[] args) {
        int i;
        try {
            i = Integer.parseInt(args[0]);
        } catch (ArrayIndexOutOfBoundsException e1) {
            i = 10;
        } catch (NumberFormatException e2) {
            i = 20;
        } finally {
            i++;
        }
        System.out.println(i);
    }
}
```

Gyakorlatok

1. A Java fejlesztői csomag (JDK) letöltése és telepítése után töltsük le a megfelelő dokumentációt, és első feladataink megoldásával párhuzamosan tanulmányozzuk át annak szerkezetét, hogy a későbbiekben könnyen használhassuk. Töltsünk le, és (ha szükséges) telepítsünk két-három fejlesztői környezetet (pl. Eclipse, NetBeans). Az első feladataink megoldása során próbáljuk ki, és próbáljuk meg összehasonlítani ezeket, kiválasztva a számunkra legmegfelelőbbet a munkánk hatékonyabbá tételének szempontjából.
2. Amennyiben szükséges, próbáljuk meg felfrissíteni az objektumorientált programozással és az általunk ismert objektumorientált programozási nyelvekkel (pl. C++) kapcsolatos ismereteinket.

Gyakorlatok

3. Írjunk programot, amely kiszámolja a parancssor argumentumainak összegét, csak az egész számokat véve figyelembe (kivételkezelést alkalmazzunk). A programot egészítsük ki olyan módon, hogy külön számolja ki a páratlan, illetve páros argumentumok összegeit.
4. Írjunk programot, amely kiírja a konzolra a parancssor argumentumait, a kisbetűket nagybetűkbe, a nagybetűket kisbetűkbe alakítva. Útmutatás: egy *String* objektum esetében egy adott karaktert a *charAt(index)* metódus segítségével kérdezhetünk le, az ellenőrzés és átalakítás a *Character* osztály statikus metódusainak segítségével történhet.

Gyakorlatok

5. Hozzunk létre egy tömbökből álló tömböt, amelynek első sora 1, második sora 2, *n*-edik sora *n* elemet tartalmaz. Az elemek egész számok 1-től *n*(n+1)/2*-ig, ahol *n*, a sorok száma, a parancssor argumentuma. Amennyiben nem adunk meg argumentumot (vagy az nem egy numerikus érték), a sorok alapértelmezett száma legyen 10. Figyeljünk arra, hogy minden tömb esetében csak annyi elemnek foglaljunk helyet, amennyire valóban szükség van. A tömb elemeit írassuk ki a konzolra az alábbi példához hasonlóan:

```
1
2 3
4 5 6
...
```