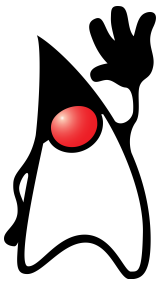
Programozási nyelvek – Java
C-ből Javába

Kozsik Tamás

A Java nyelv

- C-alapú szintaxis
- Objektumelvű (object-oriented)
  - Osztályalapú (class-based)
- Imperatív
  - Újabban kis FP-beütés
- Fordítás bájtkódra, JVM
- Erősen típusos
- Statikus + dinamikus típusrendszer
- Generikus, konkurens nyelvi eszközök

Java Language Specification



A Java nyelv sok mindenben hasonlít a C nyelvre, de nem procedurális, hanem objektum-orientált (vagy más szóval objektumelvű). Ezen paradigmán belül is az úgynevezett osztály-alapú nyelvek közé tartozik. Ez azt jelenti, hogy a program logikai felépítése a programban használt adatokon alapul. Az összetartozó adatok és a rajtuk értelmezett műveletek egységét nevezzük objektumnak. Az objektumok sematikus leírására használjuk az osztályokat.

A nyelv alapvetően imperatív, bár a mai kor szellemének megfelelően fokozatosan megjelennek benne a funkcionális programozásra jellemző konstrukciók is.

A Java forrásszöveget úgynevezett bájtkóda fordítjuk a Java fordítóval. Ez egy platformfüggetlen gépi kód: a programokat a Java Virtuális Gépen (Java Virtual Machine, JVM) futtatjuk: a bájtkód ennek a virtuális gépnek a gépi kódja. Ez a gyakorlatban azt jelenti, hogy a lefordított Java programot egy segédprogrammal futtatjuk.

Fontos még kiemelni, hogy a Java típusrendszere elég szigorú, és már fordítási időben sok programhibát ki tudunk a segítségével szűrni. Annak érdekében, hogy a nyelv kényelmesebb, rugalmasabb legyen, bizonyos típusellenőrzéseket elhalaszthatunk futási időre (dinamikus típusozás), így nem csak biztonságos, hanem rugalmas is a Java nyelv.

Más programozási paradigmák is megemlítendők a Java nyelv kapcsán. A félév során tanulni fogunk a generikus programozásról, ami a típussal való paraméterezésnek felel meg. A konkurens programozásról viszont egy másik tárgy keretében tanulunk majd.

A Java nyelv definíciója a Java Language Specification dokumentum, mely erről a linkről letölthető.

A fent látható figura a Java kabalafigurája, Duke.

Jellemzői

- Könnyű/olcsó szoftverfejlesztés
- Gazdag infrastruktúra
  - Szabványos és egyéb programkönyvtárak
  - Eszközök
  - Kiterjesztések
  - Dokumentáció
- Platformfüggetlenség (JVM)
  - Write once, run everywhere
  - **Compile** once, run everywhere
- Erőforrásintenzív

JavaZone videó

A Java nyelv meglehetősen magas szintű, ezért elég kényelmes benne programozni. A professzionális szoftverfejlesztésben nagyon szeretik, mert viszonylag olcsón, viszonylag jól rendelkezésre álló munkaerővel viszonylag jó minőségű termékeket lehet előállítani viszonylag rövid idő alatt.

A platformfüggetlenség szintén nagyon vonzó tulajdonság: a lefordított bájtkód tetszőleges olyan platformon végrehajtható, ahol a JVM rendelkezésre áll.

Annak, hogy a nyelv magas szintű, kényelmes és biztonságos, van egy hátulütője: mind végrehajtási időben, mind memóriafelhasználásban magasabbak lehetnek az erőforrásigényei, mint egy hasonló funkcionalitású C, vagy akár C++ programnak. A nyelv fordítóprogramjába és a Java Virtuális Gépbe beépített optimalizációs technikák ezen sokat segítenek, így az alkalmazási területek jelentős részén a hatékonyságbeli hátrány nem érzékelhető, vagy nem igazán jelentős.

Történelem

James Gosling és mások, 1991 (SUN Microsystems)

Oak → Green → **Java**

- Java 1.0 (1996)
- Java Community Process (1998)
- Java 1.2, J2SE (1998)
- J2EE (1999)
- J2SE 5.0 (2004)
- JVM GPL (2006)
- Oracle (2009)
- Java SE 8 (2014)
- Java SE 11 (2018) LTS
- Java SE 15 (2021)
- Game of Codes, Javazone 2014

Jelen pillanatban vagy a legfrissebb, azaz 15-ös fejlesztői környezetet, vagy a 11-es long-term support környezetet érdemes használni.

Fentebb látható a gőzölgő kávéscsésze, mely a Java nyelv logója.

Java Virtual Machine

- Alacsonyszintű nyelv: bájtkód
- Sok nyelv fordítható rá (Ada, Closure, Eiffel, Jython, Kotlin, Scala…)
- Továbbfordítható
  - Just In Time compilation
- Dinamikus szerkesztés
- Kódmobilitás

Java Virtual Machine Specification

A Java nyelv elválaszthatatlan a Java Virtuális Géptől. Az utóbbi segítségével valósul meg az, hogy lefordított Java programokat bárhol lefuttathatunk, akár alkalmazásokon belül is küldözgethetünk egyik számítógépről a másikra (gyenge kódmobilitás).

Egy hasznos optimalizációs technika a Just In Time compilation (JIT), ami egy a JVM-ben futó bájtkódot futás közben lefordít az adott platform gépi kódjára, így a bájtkód "interpretálása" helyett annak "teljesen lefordított" végrehajtása valósulhat meg. A HotSpot JVM a bájtkód végrehajtása során felismeri a gyakran végrehajtódó kódrészleteket, és ezeket a "forró pontokat" fordítja gépi kódra.

Amit még fontos megemlíteni: a JVM logikája mentén a Java programokat nem statikusan, hanem dinamikusan szerkesztjük. Azaz a JVM az adott program végrehajtásához szükséges kódokat különböző, bájtkódot tartalmazó (és `.class` kiterjesztésű) fájlokból futás közben, igény szerint szedegeti össze és szerkeszti programmá.

C és Java hasonlósága

```
int lnko( int a, int b ){
    while( b != 0 ){
        int c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

Ez a kód C-ben is és Javában is tökéletesen megállja a helyét: egy olyan függvényt definiál, amely két egész szám legnagyobb közös osztóját határozza meg az euklideszi algoritmussal. A két nyelv nagyon sok mindenben hasonlít egymáshoz. Az alaptípusok, mint például az `int`, a kifejezések és a bennük előforduló operátorok, mint például az `a % b`, az értékadás, a vezérlési szerkezetek, mint például a `while`, és maga a függvénydefiníció is a hasonlóságot illusztrálják.

Az utasítások szintaxisa és persze szemantikája is nagyon hasonlít ahhoz, amit a C nyelv kapcsán tanultunk. BNF-ben felírva:

```
<statement> ::= <expression-statement>
              | <while-statement>
              | <if-statement>
              | …

<while-statement> ::=  while (<expression>) <statement>

<if-statement> ::=  if (<expression>) <statement>
                    <optional-else-part>

<optional-else-part> ::= ""
                       | else <statement>
```

Ami elsőre nem látszik: az `if` elágazás és a `while` ciklus feltételében egy `boolean` típusú kifejezésnek kell állnia. Ez egy önálló típus, ami teljesen független az `int` típustól.

C és Java különbsége

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

Ebben a példában már megfigyelhetünk egy fontos különbséget a C nyelvhez képest: Javában a tömbök tárolják a saját méretüket, és ezt a méretet futási időben le is lehet kérdezni. Így egy tömböket feldolgozó függvénynek nem kell a tömb hosszát is átadni paraméterként, mint a C-ben, csak magát a tömböt. Ettől az apróságtól eltekintve ez a Java kódrészlet érvényes C kód is lehetne.

Persze a háttérben ennél nagyobb a különbség: a C-ben a tömböket másodrendű állampolgárnak neveztük, míg Javában teljes jogú állampolgárnak (first-class citizen) tekinthetjük őket. C-ben egy tömb helyett az első elemére mutató pointer adódik át paraméterként, míg Javában maga a tömb objektum.

C és Java különbsége - hangsúlyosabban

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

Tovább hangsúlyozhatjuk a két nyelv közötti különbséget, ha a Javában jobban megszokott szintaxist használva a tömb típusú változót (itt: paramétert) jelző szögletes zárójelpárt nem a változónév mögé tesszük, hanem az elemtípus mögé. Mindkét jelölés megengedett Javában, de az utóbbi az elterjedtebb.

Egy másik nagy újítás az a for-ciklus, amely adatszerkezetek bejárására lett kialakítva. (Angolul *enhanced for-loop* a neve.) Ez a fajta iteráció kényelmesebb és könnyebben olvasható, mint a C-ben megszokott stílusú, és ráadásul biztonságosabb is: nem kell attól tartani, hogy a ciklusmagban véletlenül elállítjuk a ciklusváltozót.

Ahogy ez a példa is mutatja, a vezérlési szerkezetekben is van azért eltérés a C és a Java között. A kifejezések kiértékelésében is tapasztalni fogunk különbségeket – de ezeket majd egy későbbi előadáson tárgyaljuk majd.

Hello World!

```
class HelloWorld {
    public static void main( String[] args ){
        System.out.println("Hello world!");
    }
}
```

A szokásos „első program'' így nézhet ki Javában. Kulcsszavak itt a `class`, a `public`, a `static` és a `void`. A Java osztályalapú objektumelvű nyelv, ezért a kód egy úgynevezett *osztálydefinícióba* van szervezve, amit a `class` kulcsszó vezet be. A programunk nevének az osztálynevet tekintjük, esetünkben ez a `HelloWorld`. A főprogram egy olyan metódus (így hívjuk az alprogramokat a Javában) lesz, amelynek a neve `main`, és lényegében pont úgy kell deklarálni, mint ahogy ebben a példában tettük: kell elé a `public static void`, egyetlen paramétere kell legyen, és ennek a paraméternek a típusa „sztringekből álló tömb'' kell legyen. Természetesen ez felel majd meg a parancssori argumentumoknak.

Szakítva a C-beli `char *` kényelmetlenségével, a Java egy számos hasznos művelettel felszerelt szabványos könyvtári típust biztosít szövegek kezeléséhez: `String`. A sztringliterálok ilyen típusba tartoznak a

Javában. A `+` operátornak létezik olyan változata, amely két sztringet összefűz, és az eredmény is egy sztring. Sőt, ez az operátor úgy van kitalálva, hogy ha az egyik operandusa sztring, akkor a másikat is sztringé alakítja, és a két sztringet összefűzi: `100 + " bolha"`.

A program tagolásában fontos szerephez jutnak a kapcsos zárójelek. Nem csak a metódusok törzsét és a blokk utasításokat, de az osztály törzsét is kapcsos zárójelekkel határoljuk. A kapcsos zárójelek elhelyezését szabályozó kódolási konvenciót jól megfigyelhetjük ezen a példán.

Egy másik kódolási konvenció az azonositók megválasztását szabályozza. Az osztályok, metódusok, változók nevét camel-case-ben írjuk. Az osztályok neve nagybetűvel, a metódusok és változók neve kisbetűvel szokott kezdődni.

A képernyőre történő kiíráshoz a `System.out.println` műveletet használjuk. Ez felel meg a Python `print`-jének, illetve a C `printf`-jének. A `println` végén a „-ln'' a *line* szóra utal, arra, hogy egy soremelést is kiír.

A `void` kulcsszó ismerős lehet C-ből: ez jelzi azt, hogy a `main` nem ad vissza semmit. (Szemben a C-vel, ahol a `main`-nek egy egész számot kell visszaadnia.)

A `static` kulcsszót is használjuk C-ben, de itt picit más a jelentése. Egyelőre tekintsünk rá úgy, mint annak a jelzése, hogy a szóban forgó metódus nem „igazi'' metódus, hanem inkább olyan, mint a C-ben egy (globálisan definiált) függvény.

A `public` kulcsszó értelme is kitalálható: az adott művelet láthatóságát terjeszti ki. Arra emlékeztethet bennünket, mint amikor a C-ben azt mondtuk egy függvényre, hogy külső szerkesztésű (external linkage), azaz nem `static`. Könnyű itt picit összezavarodni, mert a kulcsszavak és a fogalmak hasonlóak ugyan a két nyelvben, de ugyanaz a kulcsszó egész más fogalmat jelölhet.

1 Objektumelvű programozás

Objektumelvű programozás

Object-oriented programming (OOP)

- Objektum
- Osztály
- Absztrakció
  - Egységbe zárás (encapsulation)
  - Információ elrejtése
- Öröklődés
- Altípusosság, altípusos polimorfizmus
- Felüldefiniálás, dinamikus kötés

A Java nyelven történő programozás alfája és omegája az OOP paradigma. Erre a paradigmára a fenti fogalmak a jellemzőek. A félév során ezekkel fogunk megismerkedni. Kezdjük is az elsővel.

Egységbezárás: objektum

Adat és rajta értelmezett alapműveletek (v.ö. C-beli `struct`)

- "Pont" objektum
- "Racionális szám" objektum
- "Sorozat" objektum
- "Ügyfél" objektum

```
p.x = 0;
p.y = 0;
p.move(3,5);
System.out.println( p.x );
```

Egy objektum valamilyen (logikailag összetartozó) adathalmazt zár egy egységbe az adatokon értelmezett alapműveletekkel.

- "Pont" objektum "eltolás", "tükrözés" stb. műveletekkel
- "Racionális szám" aritmetikai műveletekkel
- "Sorozat" objektum "beszúrás", "törlés", "összefűzés" stb. művelettel
- "Ügyfél" objektum adatkarbantartó, -validáló, adatbázisba mentő műveletekkel

Metódus

Java

```
p.x = 0;
p.y = 0;
p.move(3,5);
```

C

```
p.x = 0;
p.y = 0;
move(p,3,5);
```

Ebben a paradigmában az alprogramokat metódusnak is szoktuk nevezni. A paradigma jellemzője, hogy a metődust egy objektumon hajtjuk végre: azon az objektumon, amelyhez a szóban forgó művelet logikailag tartozik. Ha a pont objektumokhoz definiáltuk az eltolás műveletét, akkor azt nem úgy fogjuk meghívni, mint ahogy C-ben tennénk, azaz átadva a műveletnek a pontot és az eltolás vektorát, hanem a pont paramétert speciálisan jelölve, arra, mint *kitüntetett paraméterre* hívjuk meg. A szokásos aktuális paraméterlistába pedig már csak a további paraméterek kerülnek.

Osztály

Objektumok típusa

- "Pont" osztály
- "Racionális szám" osztály
- "Sorozat" osztály
- "Ügyfél" osztály

```
class Point {
    int x, y;
    void move( int dx, int dy ){...}
}
```

A Java egy osztály alapú OO nyelv, így az objektumok szerkezetének (típusának) megadásához *osztálydefiníciókat* használunk. Az osztálydefiníció az eddigiek értelmében szintaktikus egységbe zárja a belső állapotot leíró adatok és a rajtuk értelmezett műveletek definícióját.

Figyeljük meg, hogy a kitüntetett paraméter a metódus formális paraméterlistájában sem szerepel. Az is látható (érezhető?), hogy a metódus definíciója az objektum adatait definiáló változódeklarációk hatókörén belül található. Használhatjuk is majd ezeket a "változókat" a metódus törzsében…

Példányosítás (instantiation)

- Objektum létrehozása osztály alapján
- Javában: mindig a heapen

```
Point p = new Point();
```

Az osztály és az objektum közötti kapcsolatot a példányosítás fogalma testesíti meg. A Java nyelvben objektumokat úgy (csak úgy) hozhatunk létre, hogy az osztályt példányosítjuk egy `new`-kifejezés segítségével. Szoktuk az objektumot az osztály egy *példányának* (instance) is nevezni.

A `new` segítségével létrehozott objektumok a dinamikus tárhelyen (heapen) tárolódnak. Azaz a C-ben használt `malloc` megfelelője a Javában a `new`. A `p` változó a fenti példában egy *referencia* lesz, mellyel hivatkozhatunk egy `Point` objektumra. A Java referenciák rokonságban állnak a C pointerekkel: indirekt adatelérést tesznek lehetővé. Erről később még sok szó esik majd.

Példa: szövegek

```
String name = "James Arthur Gosling";
String[] names = name.split(" ");
String abbrev = names[names.length-1] + ", "
                + names[0].charAt(0) + ".";
```

Az objektumokra jó példa a `String` típus, mely segítségével szövegeket reprezentálunk Javában. Itt látható néhány hasznos művelet sztringekkel. Az első sorban szövegliterált látunk. A másodikban egy szöveget a `split` művelettel szavakra bontunk (a szóközök mentén). A negyedik sorban azt látjuk, ahogy a `charAt` művelettel egy adott indexű karaktert kiválasztunk egy szövegből. A harmadik és a negyedik sor azt is bemutatja, hogy szövegek `+` jellel történő összefűzésével újabb szöveget állíthatunk elő. Később majd megtanuljuk, hogy szövegek összefűzésére ez egy kényelmes, de nem túl hatékony megoldás.

2 Java programok fordítása és futtatása

Java programok felépítése

(első blikkre)

- [modul (`module`)]
- csomag (`package`)
- osztály (`class`)
- tag (member)
  - adattag (mező, field)
  - metódus (method)

Az eddigiekben láttuk, hogy a Java programok osztálydefiníciókból állnak, melyekek belül adattagok és metődusok szerepelnek. A metódusok törzsében a C-hez hasonlóan utasítások állnak, és az utasítások felépítésében a kifejezéseknek is fontos szerep jut.

Az osztályoknál nagyobb logikai egységeket csomagokkal fogjuk kifejezni. Ezekkel is rövidesen megismerkedünk. Még nagyobb egységet jelentenek a modulok, de ezekről ebben a tárgyban nem beszélünk.

Java forrásfájl

- Osztálynévvel
- `.java` kiterjesztés
- Fordítási egység
- Csomagjának megfelelő könyvtárban
- Karakterkódolás

A Java programokat forrásfájlokban szoktuk tárolni. Első közelítésben azt az ökölszabályt követjük majd, hogy a fájl neve a benne tárolt osztály nevével egyezzen meg (a kis- és nagybetűk is számítanak), a fájl kiterjesztése pedig `.java` kell legyen. Egy ilyen forrásfájl egy fordítási egység lesz.

A dolgokat tovább komplikálják majd a csomagok. A csomagoknak megfelelő elrendezésben alkönyvtárakat kell majd létrehozni a munkakönyvtárunkban, és az alkönyvtárakba kell tenni a forrásfájlokat.

A forrásfájlok olyan szövegfájlok, amelyek tartalmazhatnak ékezetes betűket is. Az adott platformtól (operációs rendszertől, annak beállításaitól) függhet, hogy milyen karakterkódolást kívánunk használni.

Fordítás, futtatás

- A „tárgykód'' a JVM bájtkód (`.class`)
- Nem szerkesztjük statikusan
- Futtatás: bájtkód interpretálása + JIT

Parancssorban

```
$ ls
HelloWorld.java
$ javac HelloWorld.java
$ ls
HelloWorld.class  HelloWorld.java
$ java HelloWorld
Hello world!
$
```

A forrásfájlokból (pl. `HelloWorld.java`) a fordítóval (`javac`) hozhatjuk létre a tárgykódot, ami a Java esetén `.class` kiterjesztésű fájlban lévő bájtkód lesz. Ez egy bináris fájl: ha sima szövegszerkesztővel belenézünk, sok fura katyvaszt látunk majd, de pár értelmes szövegrészlet is a szemünkbe ötlik majd. A `HelloWorld` osztályból a `HelloWorld.class` fájl jön létre. Ezt lefuttatni a `java` paranccsal tudjuk, ez indítja el a virtuális gépet, amely betölti a megadott class fájlt, és végrehajtja a benne lévő `main` metódust.

Java programok futása

- Végrehajtási verem (execution stack)
  - Aktivációs rekordok
  - Lokális változók
  - Paraméterátadás
- Dinamikus tárhely (heap)
  - Objektumok tárolása

A program futása sok szempontból hasonlít a C-ben írt programok futására. A virtuális gép nyilvántart egy végrehajtási vermet, amelyben az alprogramhívásokat jellemző aktivációs rekordokat tárolja. Ezekben a metódusok lokális változói is megtalálhatók, valamint ennek segítségével történik az (alapvetően érték szerinti, de erről még később beszélni fogunk) paraméterátadás is.

Az objektumok programvégrehajtás közbeni tárolására a heap szolgál.