A JAVA MINT MAGASSZINTÛ PROGRAMNYELV

Lencse Zsolt, *lencse@math.klte.hu*
Szkiba Iván, *szkiba@math.klte.hu*
Almási Béla, *almasi@math.klte.hu*
KLTE Matematikai Intézet, Információtechnológia tanszék

Abstract

Sun Microsystem's new programming language gets become the leading Internet authoring language. By the use of JAVA users can create, access and support dynamic interactive Web content. In this paper we will examine how JAVA works as a genaral purpose programming language. Can this highly platform independent language be used to create "serious" applications? What are the strengths and weeknesses?

I. A Java nyelv fõbb jellemzõi

A Sun Microsystems új programnyelve és szoftvere forradalmasítja a személyi számítástechnika és a szoftverkészítés elveit. Használatával a számítógép-programok bárhova, bárkinek elküldhetõk az INTERNET-en. Segítségével a programok tárolhatók elérhetõk vagy eladhatók a hálózaton, platformtól és operációs rendszertõl függetlenül mûködtethetõk. Elõadásunkban a Javat mint magasszintû programnyelvet vizsgáljuk. Egy programozási nyelvnek az alábbi kritériumokat kell teljesítenie (Barbara Liskov, MIT):

1. Jól definiált szintaxis és szemantika
2. Megbízhatóság
3. Gyors fordítás lehetõsége
4. Hatékony tárgykód
5. Ortogonalitás
6. Gépfüggetlenség
7. Bizonyíthatóság
8. Általánosítás
9. Összhang az általánosan használt jelölésekkel
10. Részhalmazok
11. Egyöntetûség
12. Kiterjeszthetõség

Vizsgáljuk meg, hogy a JAVA milyen jellemzõkkel rendelkezik, és milyen mértékben felel meg a fenti pontoknak!

1 Jól definiált szintaxis és szemantika

A JAVA szintaktikája pontosan definiált, környezetfüggetlen nyelvtannal leírt [3] nyelv. Alapvet õen interpretálásra létrehozott objektum-orientált eszköz. A JAVA forráskódot egy úgynevezett bytecode-ra fordítja a JAVA fordító, amelyet egy “elméleti gép” a Java Vitual Machine (JVM) interpretál. Az interpretálásból következõen rendkívül portábilis és elkerül olyan problémákat amelyek a processzorkódra fordításkor jönnek elõ más objektum orientált nyelveknél (pl. sérülékeny superosztályok problémája). Bár az érdeklõdõ általában, kissé hiányos természetes nyelvvel leírt szemantikával találkozik, a fentiekbõl is láthatjuk,

hogy itt egy absztrakt géppel (JVM, bytecode) leírt pontos szemantikával állunk szemben. Mindez hatalmas lökést ad a JAVA rendszerek fejlesztésére, hiszen gyorsabban elérhetõ egy szabvány állapot. Gondoljunk csak bele, hogy pl. C-nél ez évekig tartott és az 1989-ben kitalált C++ is csak most kerül szabvány állapotba.

2. Megbízhatóság

Azaz a nyelvet úgy kell megalkotni, hogy a szintaktikai és a logikai hibák könnyen felfedezhetõk legyenek. Aki ismeri a JAVA-t az tudja, hogy a nyelv egyfajta leegyszerûsített C++, bár további nyelvi elemeket is tartalmaz. Ennek megfelelõen a hibák gyors felismeréséhez szükséges kommentárok hasonló szintaktikával kerülnek a nyelvbe mint a C++-ban Kivétel ez alól a /** típusú megjegyzés, amely az ráadásul automatikus dokumentációt szolgálja. Az a tény azonban, hogy a függvények mint nyelvi elemek hiányoznak, arra kényszeríti a programozókat, hogy függvényhívás helyett osztályok statikus metódusait hívják meg:

```
a = Math.abs(b);
```

Ebbõl a célból "fal" osztályokat kell létrehozni (pl. Math, System), ami kifejezésben kevésbé olvasható, az osztály elhagyása gyakori hibaforrás.

3. Gyors fordítás lehetõsége

Egy egyszerûen elemezhetõ nyelv nemcsak a fordítóprogram, hanem a programozó számára is elõny. A JAVA lévén egyszerûsített C++, amelyben nincsenek olyan elemek mint a fordító direktívák, függvények mutatók stb. gyors fordításra adnak alkalmat. Az írás idejében meglévõ fordítók (javac) azonban meglepõen lassan hozzák létre és szerkesztik a `.class` kiterjesztésû bytecode fájlokat. Erre rakódik rá még az interpreter többszintû bytecode-ellenõrzése. A fordító lassúsága azért is meglepõ mert nem hozza létre a memória-elrendezést és nem fordítja be az osztályhivatkozásokat, hanem meghagyja a szimbolikus hivatkozásokat. Ez utóbbi két feladat a java interpreter dolga, ami miatt viszonylag lassú a betöltés is.

4. Hatékony tárgykód

Mivel még nem jöttek létre "nagy" alkalmazások JAVA-ban, és az írás idején gyakorlatilag csak a SUN fordítóprogramja állt rendelkezésre (az elõadás idejére már kijön a Borland Development Suite-ban lévõ JAVA fordító is) nem sokat lehet tudni a fordítók által létrehozott kód hatékonyságáról. Az a tény, hogy egy alapvetõen interpretálásra létrehozott nyelvrõl van szó, valamint az interpreter bytecode-ellenõrzése miatt lassan betöltõdõ kód nyilvánvalóvá teszi, hogy a JAVA-bytecode interpretálva nem alkalmas mûszaki, tudományos számításokra. A SUN tervez egy JAVA-chipet létrehozni, amely felgyorsítaná a kódvégrehajtást. A JAVA mint nyelv erõsen tipizált lévén egyrészt támogatja a hatékony kód generálását A komponensméretek és -típusok kötelezõ megadása miatt hatékony kódot lehet létrehozni a komponensek elérésére, másrészt az explicit memóriakezelést kiváltó automatikus szemétgyûjtés plusz munkát jelent a futó kód számára. Ezt a java rendszerek a szemétgyûjtõ független szálon való futtatásával próbálják ellensúlyozni. Mindez azt jelenti, hogy egy JAVA-ból létrehozott tárgykód (akár gépi kód is lehet) sosem lesz olyan hatékony mint a neki megfelelõ C++ kód. Ezt jelzi az a tény is, hogy a JAVA más nyelvekhez hasonlóan támogatja a natív C kód beszerkesztését. Az így létrejött alkalmazás azonban nyilvánvalóan elveszti a JAVA fõ elõnyét: a nagyfokú platformfüggetlenséget. Ez a lehetõség ugyanakkor kiváló lehetõséget nyújt arra az esetre ha meglévõ kódunkat akarjuk egyszerre több platformra átvinni:
A futási idõ nagyobb részében futó kritikus kódot, ami általában a kód kisebb része C-ben hagyjuk, és a fennmaradó kisebb részben futó, a kód nagyobb részét kitevõ felhasználói interakciókat JAVAban valósítjuk meg.

5. Ortogonalitás

Ez a tulajdonság azt fejezi ki, hogy a nyelv elemei külön-külön elérhetõek legyenek és kombinálásukkor ne legyen közöttük kölcsönhatás. Ezt természetesen a C++-hoz hasonlóan a JAVA is teljesíti. Az egyik tulajdonság megléte nem vonja maga után a másik tulajdonság meglétét:

Gondoljunk például a többszálon futó osztályok szinkronizálására. Mind a többszálúság támogatása (szinkronizáció), mind az objektum-alapúság (osztályok) ortogonális tulajdonságai a nyelvnek.

6. Gépfüggetlenség

A szemantika JVM-re való megadásával rendkívüli portábilitásra ad lehetõséget a JAVA. Gondoljunk csak bele, hogy pl. a K&R C esetén az int adattípus egy gépi szót jelentett azaz 32 bites szóhosszúságú gépeknél 32 bitet, 16 bites szóhosszúságú gépeknél pedig 16 bitet. Ugyanakkor a JAVA egyértelmûen definiálja az elemi (primitive) adattípusok ábrázolását. (Pl. az operandusok a Motorla és RISC chipekben használatos *bid endian* kódolási sémát követik, egy *int* 32 bit). A nyelv maga teljesen portábilis, csak a futtató környezetet (JVM) kell egy adott operációs rendszerre átvinni. Mindez az ANSI C-ben SUN java könnyen megtehetõ minden olyan operációs rendszer esetén, amely a többszálúságot támogatja. (A Java ugyanis mind nyelvi, mind alap-objektumkönyvtár szinten támogatja a többszálúságot).
A JAVA azon kevés rendszerek közé tartozik, amely támogatja az UNICODE-ot. Ezzel a portábilitás másik gyakori problémáját, a karakterkódok használatát is áthidalja. Gyakorlatilag ma az egyetlen nyelv amely nyelvi szinten támogatja az unicode-ot (megadhatók unicode literálok).
Gondoljunk csak bele, mekkora problémát jelent egy szoftverház saját, nemzeti nyelvû karaktereket támogató programjának a használata és áttétele más más platformra (Pl. Microsoft Windows kódlapok, Lotus LMBCS).

7. Bizonyíthatóság

A JAVA-ban írt programok, lévén "igazi" nyelv, bonyolultan verifikálhatók. Azzal, hogy hiányoznak a mutatók és a goto utasítás, a JAVA nagyban könnyíti a program helyességének matematikai ellenõrzését. Ezt segíti a fordító direktívák és a függvények hiánya. Ez utóbbi lehetõvé teszi, hogy a programok helyességét elõször osztály szinten vizsgáljuk (az osztályt verifikáljuk), majd az objektumok közötti üzenetek elemzésével lássuk be a program adott input-feltételtõl való helyességét (Csak üzenetek vannak, nincsenek "friend" függvények stb.). A többszálúság támogatása konkurrens programok verifikálására alkalmas eszközkészletet követel. Ezt bonyolítja az a helyzet is, hogy a többszálúságot egyrészt a standard `Thread` osztályból való származtatással, másrészt a `Runnable` interface használatával is el lehet érni.
Mindezek persze azok a lehetõségek, amelyeket a nyelv tesz lehetõvé, de az ilyen programok verifikálása nagyon idõigényes (ezért költséges is, tehát rendszerint elmarad).

8. Általánosítás

Az az ötlet, hogy minden jellemzõ néhány alapkoncepcióból áll össze. A JAVA az objektum-orientált nyelvek (osztály, öröklõdés, stb.) és a párhuzamos programok (szinkronizáció) alapkoncepcióit használja.

9. *Összhang az általánosan használt jelölésekkel*

Ez a követelmény különösen fontos a nyelv elsajátításában. A JAVA egyrészt használja az általánosan elfogadott operátor-jelöléseket, másrészt kiterjeszti azokat, a meglévõ osztályokra (pl. sztringkonkatenáció). Visszalépés azonban, hogy származtatott (és egyéb létrehozott) osztályokra nem lehet operátorokat (át)definiálni. A C++ból jól ismert operátor-átdefiniálás itt hiányzik. Ennek a hatékony eszköznek a hiányát azon kívül semmi nem indokolja, hogy így egyszerûbb nyelvtannal lehet leírni a nyelvet. (Objektumokra csak üzenethívás van, az osztálynál megfelelõ metódust kell létrehozni, amely a megfelelõ elemeken megfelelõ módon operál. A jelölésmód zavaróbb, de így könnyebb leírni egy kifejezés szintaktikáját)

10 Részhalmazok

A részhalmazok ötlete, vagyis az, hogy nem kell a teljes nyelvet ismerni a használathoz, az olyan "nagy", bonyolult programozási nyelvekbõl mint a PL/1 vagy az ADA (nem hivatalosan) jönnek.
Ennek oka, az hogy nehéz, a teljes nyelvet "ismerõ" hatékony fordítót létrehozni. Ez a veszély a JAVA-t nem érinti, mivel lényegében (néhány plusztól eltekintve) egy létezõ nyelv egyszerûsített változata.

11. Egyöntetûség

Ez egy minimális követelmény egy jól definiált nyelvnél. Emiatt pl. a JAVAból hiányzik a C++ struktúrája és unionja. (A C++ struktúrája gyakorlatilag a C-vel való kompatibilitás miatt maradt meg. A C++-ban a struktúra gyakorlatilag abban különbözik az osztálytól, hogy más a tagelemek "rejtettségi" szintjének alapértelmezése). A struktúrák kiválthatók osztálydefiníciókkal, az unionok kiválthatók osztálydefinícióval, kis trükkel és típuskényszerítéssel. Az unionok használata lényegében annyira ritka, hogy nyelv egyszerûsödése megéri azt az árat ami a unionok hiánya okoz.

12 Kiterjeszthetõség

A kiterjeszthetõségen a további adattípusok és operátorok létrehozásának a lehetõségét értjük. A JAVA objektum-orientált lévén természetesen bõvíthetõ további adattípusokkal, amelyek ráadásul az adattípus elemein operáló algoritmusokat (metódusok) is tartalmazzák (osztály). Az operátordefinícióból eredõ hiány pedig kiváltható a metódushívással.

A fentiekbõl is következik, hogy a JAVA egy jól definiált, jól használható magasszintû programozási nyelv, amely a korszerû objektum-orienált technológiát és a többszálúságot támogatja. A biztonsági okokból kihagyott mutatók miatt azonban nem alkalmas láncolt adatszerkezetek megvalósítására (AV-L-fa, B-fa, stb.). Emiatt a teljesen általános használhatóságtól el kell tekintenünk.

II. Objektum-orientált technológia a Java-ban

A JAVA legfontosabb jellemzõje, hogy objektum-orientált. Bár a JAVA nagymértékben hasonlít a C++-ra, teljesen objektum-orientált. (A C++ inkább hibrid nyelvnek tekinthetõ). Az objektum-orientáltság számos elõnnyel jár mint:

- A kód újrahasználhatósága
- Bõvíthetõség
- Dinamikus alkalmazások létrehozása

A JAVA objektum egy dinamikusan betölthetõ elem, ami a JAVA alapvetõ végrehajtási eleme. (Itt jegyezzük meg, hogy a JAVA terminológia ezt az elemet osztálynak (`class`) hívja, de ez az OO terminológiában objektum) A JAVA interpreter határozza meg az osztályok (`class`) memória-elrendezését. A JAVA-ban megvan a hagyományos öröklõdés. Mivel az interpreter határozza meg a memória-elrendezést, nem jelentkezik a bázisosztályra való hivatkozások újrafordításának, vagyis a "törékeny szuperosztályok problémája". A JAVA emellett nem támogatja a többszörös öröklõdést. Ez nem hiba, vagy gyengeség, egyszerûen szemlélet kérdése, hiszen a többszörös öröklõdés aggregátumok használatával kikerülhetõ. A JAVA futtatókörnyezetben, mint már írtuk, dinamikusan betölthetõk a objektumok. Ezzel meglévõ alkalmazások új objektumokat láncolhatnak be további funkcionalitást adva az alkalmazásnak. Például, az Internetet böngészve egy JAVA-t (bytecode-ot) támogató böngészõvel, találhatunk olyan típusú fájlt, amelyhez nincs "megjelenítõ" alkalmazásunk. Ekkor a böngészõ például kérhet a fájlt szolgáltató szervertõl egy olyan alkalmazást (objektumot), amelyik kezelni tudja a fájlt. Ezt az objektumot a fájlal együtt letöltve megjeleníthetõ a fájl.
A JAVA teljesen objektum-orientált. Ebbõl a megközelítésbõl nem tekinthetõ a C++ kiterjesztésének (ami maga is a C egy kiterjesztése), mivel ellentétben a C++-szal a JAVA-ban egyáltalán nem lehet procedurális módon programozni. Nincsenek függvények, minden objektum egy õsobjektumtól (`Object`) származik.
Az objektumok konstruktorral és terminátorral (`finalizer`) rendelkez(het)nek. Mivel a JAVA futtatókörnyezet automatikus hulladékgyûjtést végez, a JAVA finalizereknek kisebb szerepe van (esetleg pl. fájllezárásra használhatjuk) mint más nyelveknél a destruktoroknak.
A JAVA objektumok négy féle hozzáférési szinttel rendelkeznek: *Public, Protected, Private és Friendly* . A Firendly elemekhez a csomag (package) bármely más objektuma hozzáférhet. A Public elemek

hagyományosan mindenki által hozzáférhetõk, a Protected elemek csak az adott objektumban és a származtatott objektumokban érhetõk el.

III. Biztonság

A JAVA tervezésekor nagy hangsúlyt fektettek a hálózati mûködésbõl adódó biztonsági követelményekre. Mivel a bytecode letölthetõ az Internetrõl, biztosítani kell nehogy a futó kód betörjön a rendszerünkbe. Ha erre lehetõség nyílik, akkor senki sem fog JAVA kódot letöltetni a hálózatról hiszen nem lehet biztos benne, hogy az például nem trójai faló. Ennek a célnak alárendelve nincsenek a rendszerben mutatók illetve nincs mutatóaritmetika. Ezzel biztosítva van hogy a vezérlés nem kerül ki (nem lehet kiugrani) a JAVA program számára biztosított memóriaterületrõl. Továbbá a JAVA interpretereknek ellenõriznie kell, hogy a letöltött kód JAVA bytekód. Ezt a betöltéskor induló bytecode-verifier biztosítja. Az interpreternek ezen kívül biztosítania kell, minden egyes objektumra egy független névterületet a véletlen névütközések kizárására. A bytecode-ellenõrzõ az alábbi elemeket ellenõrzi:

- "ál" mutatók
- hozzáférési szabályok megsértése
- helytelen objektumhivatkozás
- illegális adatkonverzió (cast-tal)
- érvénytelen paraméterszám

Mindezek az elemek biztosítják, hogy a vezérlés nem kerül az alkalmazáson (objektumon) kívülre, és ezzel nem okoz rendszerösszeomlást. Ez különösen fontos akkor, ha a JAVA interpreter nem végrehajtja, hanem tárgykódra fordítja a bytecode-ot és a tárgykódot futtatja egy adott rendszerben (just-in-time compiler).

IV. Java browserek a világban

A JAVA-t jelenleg a SUN HotJava és a Netscape Navigátor programja támogatja. A közeljövõben kerülhet be a Microsoft Internet Explorerébe és a Lotus Notes szerverének böngészõjébe. Miután a Netscape lévén a legelterjedtebb WEB-böngészõ, nagy lökést ad a JAVA elterjedésének.

Irodalomjegyzék

1. E. Horowitz: magasszintû programnyelvek.
Mûszaki könyvkiadó. Budapest 1987.
2. James Gosling, Henry McGilton: The JAVA language Environment
Sun Microsystems Coputer Company 1995 október
3. The JAVA language specification
Sun Microsystems Coputer Company 1995 október