Jazyk Java

Jak jsem potkal Javu (1)

Už když se v roce 1995 objevily první zprávy o Javě, bylo mi jasné, že mne tento jazyk nemine, a nemýlil jsem se. Jeho syntaxe se na první pohled velice podobá jazyku C++, chci se však zmínit o některých problémech, které mohou céčkaře potkat, když se do Javy pustí. Nebudu se pokoušet o hodnocení Javy; že jde o dobrý a užitečný jazyk, se už prokázalo v praxi. Následující povídání proto berte spíše jako hrst dojmů, které mé seznamování s ní doprovázely.

Jak známo, Java je dítkem firmy Sun Microsystems. Jedním z hlavních cílů tvůrců tohoto jazyka byla co nejvyšší nezávislost na platformě, a tedy maximální možná přenositelnost. Své mety dosáhli tím, že vytvořili interpretovaný jazyk. Zdrojový program v Javě se ovšem neinterpretuje přímo. Nejprve se přeloží do tzv. bajtového kódu (bytecode, soubory s příponou class). Bajtový kód pak interpretuje tzv. javský virtuální stroj (Java Virtual Machine, JVM); je asi jasné, že toto řešení je na úkor rychlosti. Firma Sun Microsystems sice slibuje vytvořit procesor, který by uměl bajtový kód provádět přímo, ale ten zatím, pokud vím, na trhu není.

Program v Javě je, alespoň na první pohled, velice podobný programu v C++. Ostatně autoři Javy se tím, že je Jazyk C++ inspiroval, nikdy netajili; často se dokonce setkáme s tvrzením, že Java je C++ po vypuštění potenciálně nebezpečných rysů.

Čistě objektová, ale ne tak docela

O Javě se říká, že to je čistě objektový jazyk. Není to ale tak úplně pravda, neboť čísla, znaky a další “primitivní typy” nejsou objekty. V některých čistě objektových jazycích se jako objekty chovají dokonce i bloky kódu; ani s tím se v Javě nesetkáme. Také třídy (tedy objektové typy) nejsou objekty, takže zde nenajdeme metatřídy, jako např. ve Smalltalku.

Nicméně všechny třídy v Javě jsou členy jedné dědické hierarchie, a pokud v deklaraci nějaké třídy neuvedeme předka, stane se jím automaticky třída Object, která je zde společným (pra)předkem všech tříd. To znamená, že všechny objekty dědí řadu užitečných vlastností (ale také vlastností, o které občas vůbec nestojíme), např. metodu equals pro porovnávání, toString pro převod na řetězec nebo clone pro vytvoření kopie.

Z téměř čisté objektovosti plyne, že v Javě nejsou k dispozici “obyčejné” funkce; vše jsou pouze metody. Existuje ovšem řada funkcí, které si jako metody objektových typů lze představit jen těžko – třeba běžné matematické funkce jako sinus, kosinus atd. Nicméně v Javě to jsou statické metody třídy Math, takže musíme psát Math.sin(x) apod. Nemohu si pomoci, připadá mi to jako přehnaný fundamentalismus, i když někdo bude hovořit spíše o důslednosti a možná i o čistotě nebo eleganci. Zápis libovolného matematického vzorce v Javě se tím vzdálí matematické praxi, a to nutí programátora myslet také na to, jak co píše, a nikoli jen na řešený problém. Ostatně připadá mi to i proti základním idejím objektového programování: Je snad matematika (Math) v nějakém smyslu třída objektů? Sotva – zde prostě dostala přednost snaha po objektovosti před zdravým rozumem. (Nejdůležitější je stát v řadě, to přece známe ze života...)

Ale když už důslednost, proč tedy nejsou třídami také číselné typy nebo třeba bloky kódu? (Pokud po tom však zatoužíme, máme v Javě předdefinované “obalové” třídy Double, Integer atd., jejichž instance v sobě mohou ukrývat hodnoty primitivních typů.)

Pozor, neskákat?

Java neobsahuje klasický příkaz skoku, i když slovo goto patří mezi rezervovaná. (To znamená, že je nesmíme použít jako identifikátor, ale příkaz goto použít nemůžeme a jiný význam toto klíčové slovo nemá.) To vypadá hezky; při podrobnějším pohledu ale zjistíme, že goto zde sice chybí, ale skoky ne. Vedle příkazů break a continue v podobě, v jaké je známe z jazyka C, tu najdeme konstrukci break návěští;, která umožňuje vyskočit z několika do sebe vnořených cyklů. (Napadá mne staré české přísloví o vlku, který se nažral, a o koze, která zůstala celá.)

Balíky

Zdrojový kód je v Javě uspořádán do tzv. balíků (package); to je konstrukce, která nemá v C++ přesnou analogii – nejspíše ji lze srovnat s prostorem jmen (namespace). Balík je vlastně modul,

skupina programových souborů (tříd) tvořících logický celek. Překladem balíku vznikne skupina souborů s příponou class, které za běhu programu načítá a interpretuje JVM (nebo WWW prohlížeč, pokud jde o součást appletu). Jeden balík je obvykle v jednom adresáři. Jméno balíku by mělo být založeno na internetové adrese organizace, v níž byl vytvořen, a v adresářové struktuře počítače, ve kterém je uložen, není to ale nezbytné. (To má význam u balíků sdílených v mnoha aplikacích v prostředí WWW.)

Balík musí být uložen v adresáři, který odpovídá jeho jménu. Jsou-li např. javské soubory uloženy v adresáři C:\JAVA\SRC, musí být balík jménem java.src.b1 uložen v adresáři C:\JAVA\SRC\B1. Podobně jako adresáře mohou obsahovat podadresáře, mohou i balíky obsahovat "podbalíky".

Identifikátory musí být jednoznačné v rámci balíku. Chceme-li použít jméno z jiného balíku, musíme ho kvalifikovat, tj. připojit k němu pomocí tečky jméno balíku. Potřebujeme-li tedy např. deklarovat třídu Ap1 jako potomka třídy Applet, musíme napsat

```
class Ap1 extends java.awt.Applet
{/* ... */ }
```

To je nepohodlné, a proto se často používá příkaz import:

```
imoport java.awt.*;
class Ap1 extends Applet
{/* ... */ }
```

Tento příkaz bývá srovnáván s direktivou #include, ale to není zcela přesné, neboť imoport nezpůsobí načtení importovaného balíku při překladu. Daleko přesnější je srovnání s příkazem using z ISO C++, který také umožňuje vynechat kvalifikaci identifikátorem prostoru jmen.

Třídy

Deklarace třídy v Javě se na první pohled podobá deklaraci objektového typu v C++. Sice chybí struktury a unie, některá klíčová slova se liší a definice metod musíme zapsat celé do těla třídy, za deklarací nepíšeme středník, ale to tolik nevadí. Ovšem jsou tu rozdíly, na které je třeba pamatovat. O prvním z nich jsme už mluvili: V Javě jsou všechny objekty navzájem příbuzné. Pokud v deklaraci neuvedeme předka, použije se automaticky třída Object, která je společným předkem všech objektů v Javě.

Jiným překvapením je automatická pozdní vazba (polymorfismus). Z C++ jsem byl zvyklý, že pokud požaduji pro některou metodu pozdní vazbu, musím si o to říci, tj. musím ji deklarovat jako virtuální. V Javě se pozdní vazba uplatňuje automaticky, můžeme ji ale naopak potlačit tím, že pomocí klíčového slova super předepíšeme volání metody předka. (Automatické použití pozdní vazby je v čistě objektových jazycích obvyklé, vede to ale k nižší efektivitě. Ovšem oproti skutečnosti, že Java je interpretována, je to zpravidla naprosto zanedbatelné.)

Dalším rozdílem je absence vícenásobné dědičnosti. Ono jí ve skutečnosti nejspíš není vůbec třeba, ale je to pohodlný nástroj, který umožňuje programátorům snáze řešit některé složitější situace při skládání objektů. (Když už jsme u toho – ona vlastně není nutná ani dědičnost. A když půjdeme v podobných úvahách dál, zjistíme, že není nezbytné objektové programování, a dokonce se lze obejít i bez vyšších programovacích jazyků. Koneckonců, vše, co lze naprogramovat, lze napsat přímo ve strojním kódu...) Oficiální příručky nás ovšem poučí, že vícenásobná dědičnost je plně nahrazena mechanismem rozhraní (interface).

Zajímavé také je, že dědění lze v Javě omezit. Jestliže v deklaraci třídy použijeme modifikátor final, nebude možné od ní odvodit potomka. Podobně lze pomocí tohoto modifikátoru zakázat předefinování jednotlivých metod v potomkovi.

Přístupová práva

Přístupová práva, tedy nástroje pro omezování neautorizovaného přístupu ke složkám objektů, vypadají v Javě na první pohled velice podobně jako v C++. Najdeme tu klíčová slova public, protected a private a také vysvětlení jejich významu nám bude znít povědomě; skutečnost, že tyto specifikátory zapisujeme před každou datovou složku nebo metodu, nás jistě z míry nevyvede.

Nicméně brzy zjistíme, že zde je několik významných rozdílů. Asi nejdůležitějším z nich je fakt, že se přístupová práva uplatňují nejen na úrovni třídy, ale na úrovni balíku. Složky deklarované jako chráněné (protected) jsou přístupné nejen v metodách dané třídy a jejích potomků, ale v celém balíku.

Přitom ve skutečnosti existují čtyři úrovně přístupových práv; čtvrtou je implicitní hodnota (když nepoužijeme žádný specifikátor). Implicitní přístupová práva jsou svým rozsahem někde mezi hodnotami private a protected.

Navíc při dědění nelze přístupová práva omezit, jen rozšířit. To znamená, že deklarujeme-li např. v předkovi veřejně přístupnou metodu, nelze ji v potomkovi překrýt chráněnou nebo soukromou metodou.

Trochu nezvyklé také je, že každý soubor může obsahovat nejvýše jednu veřejně přístupnou třídu a ta musí být uložena v souboru, který se jmenuje stejně jako tato třída. Trochu to připomíná některá podivná omezení z Pascalu, má to ale svoji logiku – interpreter Javy vyhledává přeložené třídy právě podle jmen souborů.

Rozhraní

Rozhraní je vlastně jakýsi seznam metod. V C++ by asi nejbližším ekvivalentem byla struktura obsahující pouze čistě virtuální metody (a případně konstanty). Třída smí mít jen jednoho předka, vedle toho ale může implementovat libovolný počet rozhraní. (Rozhraní se nedědí, ale implementují; to může vypadat na první pohled jako slovíčkaření, nicméně Java pro to používá jiné klíčové slovo.)

Třída, která implementuje určité rozhraní, se zavazuje implementovat jeho metody. Jméno rozhraní pak můžeme podobně jako jméno třídy použít k definici reference na objekt. (Nelze ho však samozřejmě použít při vytváření instance pomocí operátoru new – viz dále.)

Java a čeština

Svéráznou kapitolou v každém programovacím nástroji je jeho poměr k národním prostředím. I když občas narážím na představu, že všechny programy by se měly chovat jednotně, tedy hovořit anglicky (nejlépe s americkým přízvukem), praxe velkých softwarových firem je jiná a prostředí, které se s češtinou nedokáže snadno a dobře vyrovnat, je u nás špatně prodejné. (Ostatně v mnoha sousedních zemích je vytváření programů nepodporujících národní jazyk v podstatě nemyslitelné a lokalizace zahraničních programů je morální, nebo dokonce i zákonnou povinností; to poslední platí např. ve Francii, ale i v Německu je lokalizace vývojových nástrojů obvyklá.)

Prvním a nejdůležitějším momentem je v tomto ohledu možnost používat národní abecedu. Vzhledem k požadavku maximální přenositelnosti se tvůrci Javy rozhodli ponechat řešení tohoto problému na konkrétních implementacích. To znamená, že v Javě pracujeme pouze s několika “abstraktními” fonty, které se jmenují Serif, Monospaced atd., a jejich přiřazení “konkrétním” fontům je popsáno v souborech s názvy font.properties.xx, kde xx je přípona označující národní prostředí a případně i platformu. Soubor s názvem font.properties (bez přípony) popisuje implicitní přiřazení, které odpovídá americké angličtině.

V JDK 1.1, dodávaném např. v JBuilderu 2, bylo k dispozici značné množství (cca 30) těchto souborů pro nejrůznější jazyky, mezi nimi i pro češtinu a slovenštinu (s příponami cs, resp. sk), polštinu, ruštinu, srbochorvatštinu, maďarštinu atd.

V JDK 1.2 jich je jen devět, z toho tři pro čínštinu; dále tu najdeme korejštinu, hebrejštinu, arabštinu a thajštinu, z evropských jazyků pouze ruštinu. Přitom soubor font.properties.cs, přenesený z předchozí verze pod Windows 95, nefunguje. (Použijete-li font.properties.cs z předchozí verze pod Windows NT, překladač vám oznámí, že nemůže najít potřebné fonty, ale čeština bude fungovat.) Autory jazyka – či spíše jejich obchodní manažery – zřejmě vývojáři v malých evropských zemích prostě nezajímají; člověku se na jazyk derou ošklivé poznámky o aroganci moci, tj. monopolu, a podobně. (Nebo nás také může napadnout, že to prostě nezvládli; je otázka, co je horší.)

Poznamenejme však, že v současné době je již k dispozici částečné řešení, které umožňuje používat češtinu alespoň v některých komponentách a v jednom jediném písmu. Ovšem toto řešení nepochází, pokud vím, od tvůrců Javy.

Příště

Výčet překvapení, která mohou potkat céčkaře, když si začne s Javou, tím samozřejmě nekončí. V příštím čísle se proto k Javě vrátíme ještě jednou.

Miroslav Virius