James Keogh

Java
bez předchozích znalostí

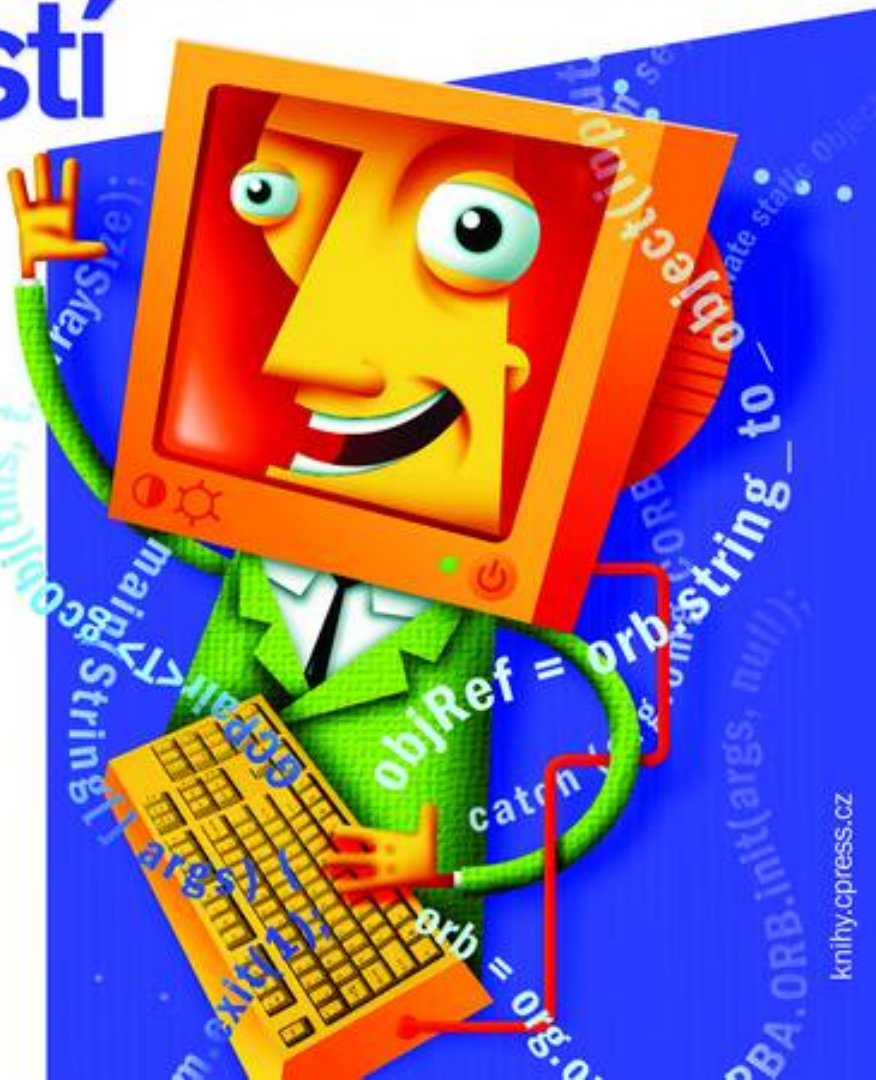
průvodce pro samouky

Základní gramatika a syntaxe jazyka

Srozumitelnou řečí bez zbytečných detailů

Kontrolní otázky a závěrečný test

Naučte se i vy zakrátko vyvářet funkční programy v Javě!



press
Mc Graw Hill
Osborne

James Keogh

Java bez předchozích znalostí
Průvodce pro samouky

Computer Press
Brno
2012

Java bez předchozích znalostí
Průvodce pro samouky

James Keogh

Překlad: Ivo Fořt
Odborná korektura: Bogdan Kiszka
Obálka: Martin Sodomka
Odpovědný redaktor: Ivo Magera
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-0839-0

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 15 927.

Dotisk prvního vydání.

ALBATROS MEDIA a.s.

Obsah

Tato kniha je věnována Anne, Sandy,
Joanne, Amber-Leigh Christine a Graafovi,
bez jejichž pomoci a podpory by nikdy nevznikla.

O autorovi

Jim Keogh je členem profesorského sboru kolumbijské univerzity, na které učí kursy zaměřené na vývoj aplikací v jazyce Java. Je také členem programu Java Community Process Program. Jako první stál u zrodu sekce zabývající se problematikou elektronického obchodování na kolumbijské univerzitě a stal se jejím předsedou. Jim strávil více než deset let vývojem pokročilých systémů pro velké firmy obchodující na Wall Street a je také autorem několika velmi dobře se prodávajících knih o počítačích.

Úvod

Tato kniha je pro všechny, kdo se chtějí naučit základy programování v jazyce Java bez nutnosti absolvovat oficiální kurs. Kniha může sloužit i jako doplňkový materiál při návštěvě některého z těchto kursů. Nejlepších výsledků lze dosáhnout postupným čtením od začátku až do konce.

Ten, kdo má dobré základní znalosti programování, může první kapitolu přeskočit. Měl by si však udělat test na jejím konci, aby zjistil, zda je skutečně připraven ponořit se přímo do programování v jazyku Java.

90 procent správných odpovědí znamená, že jste připraveni začít. Při 75 až 89 procentech správných odpovědí bude stačit zběžné prolistování 1. a 2. kapitoly. Ovšem pokud máte méně než 75 procent správných odpovědí, pak bude nejlépe, najdete-li si tiché místečko, kde si nikým nerušeni budete moci 1. kapitolu v klidu přečíst. Jedině tak se totiž připravíte na zbytek knihy o Javě.

Abychom byli schopni se Javu naučit, musíme mít jisté znalosti o počítačích. Rozhodně nemá smysl tento fakt nějak zatajovat, ale na druhou stranu bychom se jím neměli nechat zastrašit. Žádná z počítačových dovedností, které budeme potřebovat, nevybočuje z rámce základního použití operačního systému a psaní textu v editoru.

V této knize je na konci každé kapitoly test. Testy obsahují otázky podobné těm, které se vyskytují v kursech pro jazyk Java. Když si budete myslet, že danou problematiku zvládáte, zkuste si test napsat. Odpovědi poté předejte kamarádovi. Ten by vám měl sdělit výsledek, ale neměl by vám říkat, ve kterých otázkách jste chybovali. Odpovědi na otázky v testech jsou uvedeny na konci knihy. Danou kapitolu byste měli studovat tak dlouho, dokud testem úspěšně neprojdete.

Na konci knihy je též závěrečná zkouška. Otázky jsou zaměřené prakticky a představují výběr ze všech kapitol. Zkoušku byste měli vykonat až po přečtení všech kapitol a provedení všech testů. Uspokojivý výsledek je alespoň 75 procent správných odpovědí. Opět řekněte kamarádovi, aby vám spočítal výsledek, aniž by vám řekl, ve kterých otázkách jste chybovali.

Doporučujeme strávit s knihou jednu až dvě hodiny denně. Předpokládaná doba na zvládnutí jedné kapitoly je asi tak týden. Je dobré studovat rovnoměrným tempem, budete tak mít čas na vstřebání všech nových informací. Není důvod někam spěchat. Celý obsah knihy se dá zvládnout během několika měsíců a publikace potom může sloužit jako obsáhlá a trvalá referenční příručka.

Do nitra Javy

Někomu se může zdát, že počítačové programování je zahaleno rouškou tajemství a že vypadá jako něco, co přísluší univerzitním vědcům. Tak tomu ale není. S trochou času a úsilí můžeme programování počítače snadno zvládnout a převzít tak kontrolu nad počítačem způsobem, o kterém se nám nikdy ani nesnilo. Pokud si přečtete tuto knihu a uděláte si poznámky, získáte veškeré znalosti a dovednosti potřebné pro napsání počítačového programu. Počítačový program představuje množinu instrukcí, které počítač provádí. Tyto instrukce se píší v jazyce, který se do značné míry podobá angličtině. Existuje velmi mnoho různých počítačových jazyků. Mezi ty populárnější patří i programovací jazyk Java. Kolem Javy už se strhlo mnoho povyku a bylo proneseno mnoho slov o tom, jakou revoluci přinesla do světa počítačového programování. Budeme v čele této revoluce, naučíme-li se řídit počítač pomocí vlastních programů napsaných v jazyce Java. Naši cestu zahájíme na samém počátku úplnými základy. Postupně se však propracujeme přes vše, co je třeba vědět k tomu, abychom mohli psát javové programy.

Počítačové programy

S osobním počítačem se už nejspíš setkal každý. Nicméně osobní počítač představuje pouze jeden druh počítače. Mezi další typy patří osobní digitální asistenti (PDA, personal digital assistant), výkonné podnikové počítače a malé počítače v automobilech, letadlech a domácích spotřebičích.

Všechny počítače mají jedno společné: provádějí výpočty a logická rozhodnutí milionkrát rychleji, než jsme toho schopni my. Z technického hlediska provádějí počítače pouze dva druhy výpočtů – sčítaní a odečítání.

Takže – připraveni na první test? Jak poznáme, zda se dvě čísla rovnají? Odečteme je. Pokud je výsledek nula, čísla jsou stejná. Pokud je výsledek kladná hodnota, je první číslo větší než druhé. V případě záporného výsledku je první číslo menší než druhé. A toto je také způsob, kterým počítač provádí logická rozhodování.

Počítačový programátor říká počítači, jak má provádět výpočty a jakým způsobem se má logicky rozhodovat pomocí počítačového programu. Počítačový program obsahuje všechny kroky, které se musí provést ve specifickém pořadí, aby došlo k výpočtu a bylo dosaženo logického rozhodnutí.

Data

Mnoho počítačových instrukcí po počítači vyžaduje, aby pracoval s informacemi, které mu dodal programátor, osoba, jež počítač používá, nebo jiný počítač. Těmto informacím se říká *data*.

Data počítači předkládáme pokaždé když vyplňujeme své přihlašovací jméno a heslo. Program tyto informace převezme a ověří je ještě před tím, než nám dovolí přístup k počítači. A máme tady další test. Jaký druh výpočtu se používá při ověřování přihlašovacího jména a hesla? Odečítání! Program v počítači odečte zadané jméno a heslo od platného přihlašovacího jména a hesla. Pokud je výsledek nula, poskytli jsme korektní údaje a přístup k počítači je nám povolen.

Když data poskytuje programátor, může je vepsat přímo do instrukce. Například takto vypadá instrukce, která počítači říká, aby sečetl dvě čísla (programátor čísla uloží přímo do instrukcí):

```
10 + 15
```

Pokud jste překvapeni tím, jak jednoduše může taková počítačová instrukce vypadat, máme pro vás radostnou novinu, protože mnoho instrukcí v programovacím jazyku Java je jednoduchých pro pochopení i pro zápis.

Programátor ale mnohokrát nemá v okamžiku psaní programu data pro instrukci k dispozici, jako je tomu i v případě přihlašovacího jména a hesla. V takovém případě musí zadat data do běžícího programu osoba, která jej používá.

Nicméně i v tomto případě se musí při psaní programu zapsat instrukce tak, aby byla schopna pracovat s daty. Programátor proto použije místo dat v instrukci jejich zástupce. Na zástupce dat se lze dívat jako na dočasnou jmenovku určenou pro data. Programátoři říkají těmto jmenovkám *proměnné* a my se o nich dozvíme více později. Následující příklad používá k součtu dvou čísel stejnou instrukci, s výjimkou toho, že písmena `A` a `B` představují jmenovky pro konkrétní hodnoty:

```
A + B
```

Počítač nahradí písmena čísly ve chvíli, kdy člověk zadá do programu čísla nebo tehdy, když program získá hodnoty z jiného počítačového programu.

Programovací jazyky

Počítačový programovací jazyk, jako je například Java, usnadňuje programátorovi psaní instrukcí pro počítač. Je to proto, že tyto instrukce může psát pomocí slov podobných anglickým. Avšak počítač ve skutečnosti anglicky znějícím slovům nerozumí. Rozumí jen instrukcím napsaným ve strojovém jazyce. Instrukce strojového jazyka tvoří kombinace jedniček a nul, kterým rozumí centrální řídicí jednotka (CPU, procesor), což je část počítače, v níž probíhají veškeré výpočty a zpracování.

I když mají programátoři tendenci odkazovat se na strojový jazyk jako na jediný jazyk, existují ve skutečnosti různé verze strojového jazyka. Dá se na ně dívat jako na dialekty. Jednotka CPU rozumí pouze jednomu dialektu strojového jazyka. To znamená, že program napsaný v jednom dialektu se dá spouštět jen na tom počítači, který má procesor, jenž danému dialektu rozumí. To dělá z programů výtvory *závislé na počítači*. Jinými slovy řečeno, program ve strojovém jazyku napsaný pro jeden druh počítače se nedá spustit na jiném druhu počítače.

Nemusíme být vědeckými kapacitami, abychom viděli problémy spjaté s psaním programů ve strojovém jazyce. Za prvé, ruku na srdce, kdo by opravdu chtěl psát programy pomocí samých jedniček a nul? Jsme zvyklí přemýšlet pomocí slov, nikoliv čísel. A také

kdo by chtěl trávit všechen svůj čas psaním programu, který by mohl běžet jen na jednom druhu počítače?

Zavedení jazyka symbolických adres vyřešilo alespoň jeden z těchto problémů. Jazyk symbolických adres je další programovací jazyk, který tvoří anglické zkratky nazývané *instrukce jazyka symbolických adres*. Každá z nich představuje pro počítač elementární operaci.

Programátor se nejprve rozhodne, kterou operaci je potřeba vykonat, a poté použije instrukci jazyka symbolických adres, aby počítači sdělil, že má tuto operaci provést. Počítač stále rozumí pouze jedničkám a nulám, a proto se musí použít sestavovací program zvaný *assembler*, jenž přeloží instrukce jazyka symbolických adres do strojového jazyka.

Podívejme se na ukázku programu napsaného v jazyce symbolických adres. Lze z ní poměrně snadno zjistit, jakou operaci má počítač vykonat. Počítač dostal za úkol sečíst čísla 10 a 15.

```
ADD 10, 15
```

Je potřeba přiznat, že jazyk symbolických adres měl vůči strojovému jazyku velké výhody – programátor mohl pro zapisování instrukcí pro počítač používat slova podobající se angličtině. Nicméně stále existovaly i výrazné nevýhody.

Programátoři se museli u tohoto jazyka naučit spoustu zkratek, které vůbec nebyly intuitivní, jako například POP a PUSH. Pro vykonání základních operací se muselo použít velké množství instrukcí. Největším problémem však stále zůstávala přenositelnost. Každý druh počítače rozuměl jen svému dialektu jazyka symbolických adres. Bylo tudíž prakticky neproveditelné napsat v tomto jazyce program, který by se dal spouštět i na jiných druzích počítačů, aniž by se musel alespoň zčásti přepsat.

Jazyk symbolických adres se postupem času rozvinul do programovacích jazyků vyšších úrovní, které jsou mnohem intuitivnější díky svým angličtině se podobajícím slovům, příkazům (pseudoinstrukcím) a interpunkci, pomocí nichž říkáme počítači, co má udělat. Mezi dnešní oblíbené vysokoúrovňové programovací jazyky patří jazyky C, C++ a Java.

Navíc, pro provedení skupiny souvisejících operací stačí napsat jedinou instrukci. Programátor je tak zproštěn břemene psaní samostatné instrukce pro každou operaci. Programy napsané v jazycích vyšších úrovní lze spouštět na různých počítačích bez toho, aby je programátor musel přepisovat.

Podívejme se, jak takový vysokoúrovňový jazyk funguje: programátor používá klíčová slova programovacího jazyka, aby sdělil počítači, co má vykonat. Klíčová slova se podobají slovům v angličtině, ze kterých formulujeme věty, když chceme někomu říct, co má udělat.

Asi nás ani nepřekvapí, že počítač nerozumí programu napsanému v jazyce vyšší úrovně, protože rozumí pouze strojovému jazyku. I program napsaný ve vysokoúrovňovém jazyce se tedy musí přeložit do strojového jazyka.

Proces překladu tvoří dva kroky. Nejprve se program převede na mezistupeň, který se nazývá *objektový soubor*. Ve druhém kroku objektový soubor konvertuje na program ve strojovém jazyce, který lze spustit na počítači.

Převedení programu do objektového souboru se nazývá *kompilování* a provádí jej překládací program, jemuž říkáme *kompilátor*. Proces převodu objektového souboru na program ve strojovém kódu je označován jako *spojování* (linkování) a provádí jej *spojovací program* (linker).

Nabízí se otázka, proč se program nezkompiluje přímo do strojového jazyka. Důvod je jasný – běžný program tvoří dva nebo i více objektových souborů, které se musejí nejprve spojit dohromady, aby mohly vytvořit program ve strojovém jazyku. Programátoři říkají tomuto spojování objektových souborů *linkování*.

Kompilování a spojování jsou důležité rysy vysokoúrovňových programovacích jazyků, protože umožňují spouštění programu na různých druzích počítačů bez nutnosti jeho přepisu.

Jak je to možné? Pro každý druh počítače existuje jiný kompilátor a spojovací program, který je schopen přeložit specifický vysokoúrovňový programový kód pro specifický druh počítače. Například program napsaný v jazyce C++ se dá zkompilovat a spojit, aby běžel na různých druzích počítačů bez nutnosti přepisování.

A takto vypadá předchozí verze příkladu uvedená v jazyce symbolických adres, nyní však přepsaná do vysokoúrovňového jazyka:

```
10 + 15
```

Na počátku

V padesátých letech minulého století byly velmi populární dva vysokoúrovňové počítačové jazyky – FORTRAN (FORmula TRANslator) a COBOL (COmmon Business Oriented Language). Tyto jazyky se používají dodnes.

FORTRAN, vyvinutý firmou IBM, je vysokoúrovňový programovací jazyk navržený pro vykonávání složitých matematických výpočtů pro vědecké a technické aplikace. Jazyk COBOL, vyvinutý na zakázku federální vlády, je navržen pro zpracování velkých objemů dat a manipulaci s nimi. Přestože oba jazyky splnily velmi dobře požadavky na ně kladené, nebyly dostatečně přizpůsobivé a chyběly jim vlastnosti požadované při tvorbě kompilátorů a operačních systémů.

To vše vedlo ke snahám vyvinout víceúčelový vysokoúrovňový programovací jazyk. Koncem šedesátých let byl Martinem Richardsonem vytvořen nový jazyk, který nesl označení *programovací jazyk BCPL* a sloužil pro psaní kompilátorů. Brzy po jeho uvedení představil Ken Thompson rozšířenou verzi jazyka BCPL a nazval ji *programovací jazyk B*. Programovací jazyk B použila firma Bell Laboratories pro vytvoření první verze operačního systému Unix.

Hlavní nevýhoda programovacích jazyků BCPL a B spočívala ve způsobu, jakým používaly paměť počítače. V dnešní době je paměť počítače relativně levná. Avšak v šedesátých letech minulého století byla počítačová paměť drahá a jazyk BCPL i jazyk B ji využívaly neefektivně.

Představme si paměť počítače jako hromadu skříněk. Do každé skříňky se vejde jedna láhev vína (data). Proto je rozumné rezervovat si deset skříněk, pokud potřebujeme uložit deset lahví vína, a pět skříněk, když máme pět lahví. Jazyky BCPL a B však rezervovaly stále stejný počet skříněk bez ohledu na počet lahví, které ukládaly. V praxi to znamenalo, že vždy, když jsme chtěli uložit jednu láhev vína, museli jsme rezervovat deset skříněk. Z toho plyne, že devět skříněk nebylo použito, a tudíž promrháno.

V roce 1972 vytvořil Dennis Ritchie ze společnosti Bell Laboratories programovací jazyk C, který překonal nedostatky jazyků BCPL a B. Jazyk C v sobě zahrnoval mnoho rysů, které

se vyskytovaly již v jazycích BCPL a B. Obsahoval však i mnoho nových vlastností, mezi které patřila například i možnost určení přesného množství paměti potřebného pro uložení dat do paměti počítače.

Ačkoliv jazyk C vyřešil nedostatky jazyků BCPL a B, někteří programátoři cítili, že mu chybí schopnost napodobit způsob, jakým se my lidé díváme na reálný svět. To je podstatná nevýhoda, protože počítačové programy se vytvářejí z toho důvodu, aby dokázaly uvnitř počítače simulovat skutečný svět. Proto nedokázal programovací jazyk C simulovat reálný svět podle potřeb programátorů.

Lidé vnímají reálný svět jako množinu objektů. Tyto objekty mají různé atributy (data) a chování. Vezměme si na ukázku třeba okno. Rozměry okna jsou jeho atributy. Okno se dá ale také zavřít nebo otevřít, což jsou funkce s ním spojené.

Programovací jazyk C je procedurální jazyk, který se zaměřuje na napodobování chování opravdového světa uvnitř počítače. Naneštěstí není vybaven schopnostmi, jak spojit chování s atributy.

V roce 1980 vytvořil Bjarne Stroustrup z firmy Bell Laboratories nový programovací jazyk a nazval jej C++. Jeho nejvýraznějším vylepšením byla možnost spojovat atributy a funkčnosti do objektů. Od tohoto okamžiku přichází na scénu objektově orientovaný návrh a objektově orientované programování (viz kniha Object-Oriented Programming Demystified, která se tomuto tématu věnuje).

Nabízí se otázka, proč pan Stroustrup použil k odlišení jazyka jen znaky ++ místo toho, aby dal jazyku do vínku zcela nový název. Symbol ++ představuje v jazyce C (a v Javě) přírůstkový operátor, o němž si budeme povídat i my později v této knize. Pro tuto chvíli je důležité vědět, že přírůstkový operátor přičítá ke stávající hodnotě číslo 1. O jazyku C++ se říká, že povyšuje (inkrementuje) programovací jazyk C, protože má v sobě zakomponovány všechny jeho vlastnosti a navíc přináší spoustu nových rysů. Proto se dá na jazyk C++ dívat jako na rozšíření programovacího jazyka C.

Jak se stane počítačový jazyk standardem

Mnohé z nás už asi napadla otázka, jak se vlastně počítačový jazyk vyvíjí? Je k tomu potřeba mnoho vytrvalosti a štěstí. Vraťme se kousek zpět a vzpomeňme si na součásti programovacího jazyka. Všechny programovací jazyky se skládají z klíčových slov a funkčnosti.

Klíčové slovo se podobá slovu z angličtiny. Funkčnost je činnost, kterou provede počítač, pokud se klíčové slovo použije v programu. Na funkčnost se můžeme dívat jako na definici klíčového slova.

Počáteční krok ve vývoji programovacího jazyka spočívá ve vytvoření seznamu klíčových slov a funkčnosti. V ideálním případě by měla klíčová slova a funkčnost nového programovacího jazyka představovat toužebně očekávaná vylepšení vůči stávajícímu programovacímu jazyku. Pokud tomu tak není, nebude nový jazyk používat nikdo s výjimkou jeho autora.

V další fázi je potřeba diskutovat se členy technické komunity a podnítit v nich zájem o nový programovací jazyk. Pokud se o něm bude dostatečně hovořit a bude přinášet skutečný užitek, postarají se čelní představitelé technologie a průmyslu o tlak, který povede k jeho standardizaci.

Standardizace představuje formální proces, ve kterém se technická komunita pomocí standardizační organizace sjednotí na množině klíčových slov a odpovídající funkčnosti. Mezi význačné standardizační organizace patří Americký národní standardizační institut (ANSI, www.ansi.org) a Mezinárodní standardizační organizace (ISO, www.iso.ch). Organizace Java Community Process (www.jcp.org) stanovuje standardy pro programovací jazyk Java.

Jakmile spatří nový standard světlo světa, vytvoří výrobci softwarových nástrojů kompilátory, spojovací programy a další softwarové pomůcky, které rozpoznají programy napsané v novém programovacím jazyku a dokáží je převést na programy v jazyku strojovém, takže je lze spouštět na různých druzích počítačů. Nový jazyk se začne vyučovat ve školách a vzdělávacích institucích, začnou se o něm psát knihy a programátoři ho používají pro psaní programů.

A nakonec Java

Jazyk C++ a další vysokoúrovňové jazyky mají všechny jeden nedostatek: programy v nich napsané se musí překompilovat, aby se daly spustit na různých druzích počítačů. Komerční organizace měly zájem o programovací jazyk, jenž by jim dovolil vytvářet programy, které by mohly běžet na všech druzích počítačů, aniž by bylo potřeba je překompilovat.

Nepřímou cestou se tohoto cíle podařilo dosáhnout společnosti Sun Microsystems, když uvedla programovací jazyk Java. V roce 1991 odstartovala společnost tzv. Zelený projekt (Green project), jehož cílem bylo vyvinout programovací jazyk vhodný pro psaní programů určených pro zařízení spotřební elektroniky, jako jsou televizory a počítače používané v automobilech. Firma viděla v tomto segmentu trhu velký komerční potenciál.

James Gosling, jeden z vedoucích inženýrů Zeleného projektu, vytvořil programovací jazyk Oak, který zadaná kritéria splňoval. Jméno jazyka vzniklo podle stromu, který rostl před jeho kanceláří. Vzápětí se však objevil malý problém. Programovací jazyk s názvem Oak již existoval. Gosling a další členové týmu Zeleného projektu se proto sešli v blízké kavárně s cílem vymyslet pro svůj jazyk nové jméno – a tak vznikla Java.

Potřeba vytvářet programy, které by se spouštěly v domácích elektrických spotřebičích, nikdy nevznikla. Zato však v roce 1993 explodovala bomba zvaná World Wide Web. Co dosud bylo komunikační sítí pro akademiky a vládní úřady, se náhle stalo novým prostředkem vzájemné komunikace mezi obchodníky a širokou veřejností.

Klíčovým prvkem komunikace se staly webové stránky. Každá stránka byla psána ručně pomocí kódu HTML (Hypertext Markup Language) a sloužila pro zobrazení textu a grafiky na vzdáleném počítači. Těmto stránkám se říká statické, protože jejich obsah zůstává při každém prohlédnutí stále stejný.

Avšak návrháři webových stránek požadovali robustní způsob tvorby dynamických webových stránek, jež by se daly přizpůsobit individuálně pro každou osobu, která web navštíví. Chtěli webové stránky generované počítačovým programem, který by mohl také automaticky komunikovat s databázovým a počítačovým systémem používaným v organizaci.

Goslingův programovací jazyk Java se perfektně hodil pro vytváření dynamických webových stránek s interaktivním obsahem, jež dovolovaly individuální přizpůsobení zobrazovaného obsahu pro každého návštěvníka webového serveru. Navíc se Java dala spouštět prakticky na jakémkoliv druhu počítače, aniž by se musela překompilovat.

Společnost Sun Microsystems oficiálně představila Javu v roce 1995 a o čtyři roky později se Java stala typickým programovacím jazykem pro velké podnikové aplikace využívající technologii World Wide Web.

Současný programovací jazyk Java se značně rozrostl a zahrnuje několik verzí zvaných *edice*. Nejčastěji používanou edicí je Java 2 Standard Edition (J2SE), kterou se budeme zabývat i v této knize. Existuje ale také Java 2 Enterprise Edition (J2EE), jež se využívá pro tvorbu podnikových aplikací. Mezi další, často se objevující, edice patří například Java 2 Micro Edition (J2ME), která byla vytvořena k programování aplikací určených pro mobilní zařízení, jako jsou třeba mobilní telefony a osobní digitální asistenti. O edici J2EE se dozvíme více z knihy *J2EE: The Complete Reference*, edice J2ME je zase popsána v knize *J2ME: The Complete Reference*.

Pohled do nitra Javy

Pojďme se podívat na některá základní fakta týkající se Javy. *Program* je série instrukcí zapsaná v programovacím jazyce, která říká počítači, aby vykonal specifickou úlohu. Jako kdybyste popsali kamarádovi cestu, kudy se dostane k vám domů. Každá instrukce musí být přesně zapsána, aby počítač přesně porozuměl tomu, co po něm chceme.

Bezpochyby jste se již setkali s pojmem *počítačová aplikace*. Počítačová aplikace je běžně skupina souvisejících programů, které společně říkají počítači, jak má napodobit skutečný svět. Například v místním supermarketu mají aplikaci pro zpracování transakcí, která se používá pro zaznamenávání, zpracování a vykazování prodejních transakcí. Nejviditelnější částí celé aplikace jsou snímače čárového kódu umístěné na pokladnách. Aplikaci na zpracování transakcí používanou v supermarketu tvoří mnoho programů.

Program v jazyku Java se skládá z jedné nebo více tříd, které jsou napsány v programovacím jazyku Java. Třída se podobá formičce na perníčky a slouží k definování objektů reálného světa uvnitř počítače. Podobně jako formička definuje, jak bude perníček vypadat, definuje i třída vzhled objektu. A stejně jako formička se i třída používá k vytvoření skutečných objektů uvnitř programu. Jak se to dělá, uvidíme v 7. kapitole.

Třídy obsahují instrukce, které říkají počítači, co má udělat, a samozřejmě data, jež počítač pro splnění zadaného úkolu potřebuje. Třídy se zapisují do souboru se zdrojovým kódem Javy pomocí textového editoru. Soubor zdrojového kódu Javy se podobá dokumentu textového procesoru, avšak namísto textu obsahuje soubor se zdrojovým kódem Javy instrukce napsané v programovacím jazyku Java. Textový editor je jednoduchý textový procesor, který nedisponuje všemi těmi úžasnými formátovacími schopnostmi, jež se dají nalézt v normálních textových procesorech.

Píšeme zdrojový kód Javy, který se posléze stane programem. Zdrojový kód se však musí nejprve uložit na disk do souboru s rozšířením `.java`.

Java funguje jinak než C++ a další vysokoúrovňové programovací jazyky. Tyto jazyky se musí nejprve zkompilovat do objektového kódu, který se poté spojí spojovacím programem, čímž se vytvoří program ve strojovém jazyce běžící na počítači.

Zdrojový kód Javy se nekompiluje do objektového kódu. Místo toho se zkompiluje do bajtového kódu a uloží do souboru s příponou `.class`. Kompilátor Javy je součástí vývojového balíku Java 2 Software Development Kit (J2SDK). Ten se dá zdarma stáhnout z webu java.sun.com.