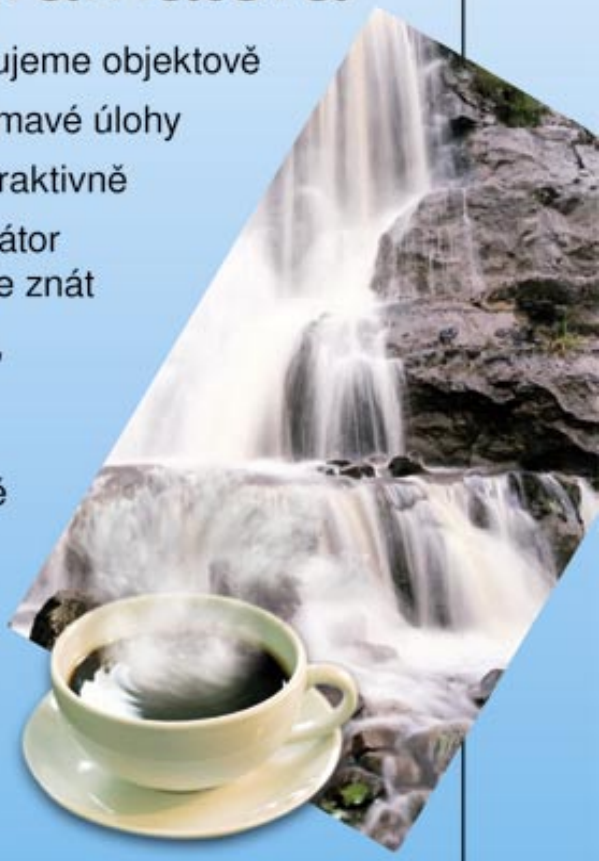
EDICE myslíme v ...

knihovna programátora

- Od samého počátku programujeme objektově
- Již od prvních kapitol řeší zajímavé úlohy
- Ukazuje, jak programovat interaktivně
- Probírá oblasti, které programátor ve své praxi opravdu potřebuje znát
- Věnuje se návrhovým vzorům, refaktorování, programování řízenému testy
- Doporučená učebnice na řadě středních a vysokých škol



myslíme objektově v jazyku

RUDOLF PECINOVSKÝ

Java

kompletní učebnice pro začátečníky, 2., aktualizované a rozšířené vydání

GRADA

Upozornění pro čtenáře a uživatele této knihy

O autorovi

Rudolf Pecinovský patří ke špičkovým odborníkům na výuku programování. Publikoval již 39 učebnic, které byly přeloženy do pěti jazyků, a nepřeberné množství článků a příspěvků na odborných konferencích. Je autorem metodiky výuky programování *Karel*, navazující metodiky *Baltík* a moderní metodiky výuky objektově orientovaného programování známé pod anglickým názvem *Design Patterns First*. Učí programování na VŠE a současně pracuje jako Senior EDU Expert ve firmě ICZ a.s., kde má na starosti doškolování profesionálních programátorů.

O knize

Tato kniha je druhým vydáním populární učebnice programování, která je na našem trhu zcela ojedinělá. Na rozdíl od ostatních učebnic se totiž neomezuje na výuku syntaxe jazyka a práce s knihovnami, ale učí čtenáře doopravdy programovat. Učí jej, jak má při programování myslet.

Kniha je sice primárně určena začátečníkům, ale ohlasy na první vydání ukázaly, že v ní najdou poučení i zkušení programátoři. Většina učebnic a kurzů programování totiž vyvolává falešnou představu, že objektově programovat znamená používat třídy a dědičnost. Tato kniha je první, která ukazuje, že objektově orientované programování přináší především jiný způsob myšlení. Jak výstižně napsal jeden čtenář: „*Myslel jsem si, že nejsem žádné programátorské ucho. Když jsem ale přečetl vaši učebnici, otevřel jsem oči a hubu. Konečně jsem pochopil věci, které mi ostatní učebnice nedokázaly vysvětlit.*“

Kniha vznikla na základě dlouholetých autorových zkušeností se školením profesionálních programátorů, výukou programování na univerzitě i vedením žákovských programátorských kroužků. Autor v ní uvádí čtenáře krok za krokem do tajů objektově orientovaného programování a ukazuje mu, jak možnosti této moderní technologie co nejlépe využít a kde si dát naopak pozor na její úskalí.

Výklad je postaven na příkladech, které autor spolu s čtením postupně řeší a přitom čtenáře učí nejenom základním programátorským návykům a dovednostem, ale předvede mu i nejrůznější užitečné triky, z nichž mnohé nikde jinde vysvětlené nenajdete. Současně upozorňuje na nejčastější začátečnické chyby, které před svými čtenáři ostatní učebnice většinou tají. Navíc probírá i řadu témat (např. návrhové vzory), které patří do základní výbavy objektového programátora, přestože jsou většinou probírána až v pokročilých nebo dokonce nadstavbových kurzech.

Rudolf Pecinovský

Myslíme objektově v jazyku Java

kompletní učebnice pro začátečníky, 2., aktualizované a rozšířené vydání

Vydala Grada Publishing a.s.
U Průhonu 22, Praha 7
jako svoji 3472. publikaci

Odborní lektoři Prof. RNDr. PhDr. Antonín Slabý, CSc.,
Doc. Ing. Vojtěch Merunka, Ph.D., Ing. Alena Buchalcevová, Ph.D.
Odpovědná redaktorka Jaroslava Palasová
Návrh vnitřního layoutu Rudolf Pecinovský
Zlom Jana Davídková, Rudolf Pecinovský
Počet stran 576
První vydání, Praha 2009

Vytiskly Tiskárny Havlíčkův Brod, a.s.
Husova Ulice 1881, Havlíčkův Brod

ISBN 978-80-247-2653-3 (tištěná verze)
ISBN 978-80-247-7025-3 (elektronická verze ve formátu PDF)

Mé ženě Jarušce a dětem
Štěpánce, Pavlínce, Ivance a Michalovi

Stručný obsah

Podrobný obsah

Poděkování

Vím, že se v českých knížkách většinou neděkuje, ale tahle kniha byla spojena s takovými oběťmi řady lidí z mého blízkého i vzdálenějšího okolí, že bych měl velkou újmu na duši, kdybych tak neučinil.

Chtěl bych především nesmírně poděkovat své ženě Jarušce, která byla po celou dobu mojí největší oporou a jejíž nekonečná trpělivost a vstřícnost mi pomohla dokončit knihu v termínu, který se příliš nelišil od toho, jenž jsme původně s nakladatelem dohodli, a ne až někdy za rok po něm. Původně jsem se domníval, že s druhým vydáním nebude moc práce. Šeredně jsem se zmýlil, protože veškeré úpravy, které jsem se rozhodl do knihy zanést (a že jich bylo požehnaně), musely zapadnout do předchozího textu. O to náročnější byla redakční práce, při níž bylo třeba mimo jiné zkontrolovat, že tomu tak doopravdy je.

Nemenší poděkování patří i dětem, které se mi také snažily v rámci svých možností pomáhat. Především Ivance, která pomáhala mamince s některými redakčními pracemi a našli mi v rukopise řadu těžkopádných formulací, Michalovi, který zanášel připomínky čtenářů a Pavlínce, která mne osvobodila od starostí o administrativu a účetnictví.

Na vylepšování textu nového vydání se ale podílela řada dalších lidí. Mezi nimi musím poděkovat především těm, kteří si dali tu práci a při objevení chybky v minulém vydání mi o ní napsali. Mezi nimi pak především Jaromíru Tesařovi, Bohumíru Zámečníkovi a Jiřímu Jílkovi, kteří si dali tu práci, a opravdu zaznamenávali všechny nesrovnalosti, na které při čtení narazili.

Nemalý podíl na úpravách textu mají i Jarmila Pavlíčková a Luboš Pavlíček, s nimiž jsem v posledních letech několikrát probíral styl výuky i použité příklady.

Svůj podíl na kvalitě výsledného textu má i Alena Buchalcevová, které se podařilo v téměř hotovém textu najít ještě několik nepřesností.

Poděkování patří Janě Davídkové, která s níž jsme vtiskávali knize konečnou grafickou podobu. Přes záplavu nejrůznějších poznámek, programů a odboček, kterými se text hemží, se jí podařilo vše uspořádat do podoby, která v neznalém čtenáři vyvolá dojem, že text byl napsán tak, aby šel dobře zalomit.

Stejně bych chtěl poděkovat o Evě Steinbachové z redakce počítačové literatury nakladatelství Grada, která celou práci průběžně sledovala, zprostředkovávala komunikaci mezi mnou a nakladatelstvím a v neposlední řadě upozorňovala na drobná opomenutí, která by mohla způsobit problémy při následné výrobě knihy..

Rád bych touto cestou poděkoval i Michaelu Köllingovi a jeho spolupracovníkům, jejichž myšlenky mne kdysi inspirovaly k novému uspořádání výkladu a jejichž vývojový nástroj *BlueJ* realizaci takového výkladu vůbec umožnil.

Na závěr pak musím vyjádřit svůj velký dík veškerému osazenstvu oddělení Development ve firmě ICZ. Tito lidé mě k Javě přivedli, a po celou dobu přípravy knihy mě všestranně podporovali. Bez jejich podpory by kniha nevznikla.

Předmluva k prvnímu vydání

Rudu Pecinovského jsem poprvé potkal v době, kdy jsme oba studovali na Jaderné fakultě ČVUT v Praze. Doopravdy jsme se ale poznali až mnohem později, když jsme na počátku devadesátých let spolupracovali na překladu manuálů k jistému dodnes populárnímu programovému prostředí. Brzy jsme zjistili, že máme jeden společný zájem – učit lidi, jak kvalitně psát programy.

V současné době dominuje při tvorbě aplikací objektově orientované programování. Moderní vývojové nástroje, které jsou na trhu k dispozici, jeho znalost předpokládají, aplikační knihovny z něj vycházejí, softwarové firmy ho vyžadují, nově vznikající programovací jazyky jsou čistě objektové. A když už jsme u těch jazyků: Java je dnes asi nejpoužívanější jazyk pro vývoj nových aplikací a zcela určitě to je jazyk, který se nejdynamičtěji rozvíjí. Přesto téměř všechny učebnice Javy, které na trhu najdete, začínají procedurálním programováním a k objektově orientovanému programování se dostanou až ke konci. Objekty pak často vypadají jako nepříliš pohodlná nadstavba nad procedurálním programováním.

Řekl jsem, že tak vypadají *téměř* všechny knihy. Kniha Rudy Pecinovského je totiž velmi příjemnou výjimkou. Je to učebnice, která objekty opravdu začíná a prvních několik kapitol se ani ničím jiným nezabývá. Teprve poté, co zvládnete základní pojmy a dovednosti objektově orientovaného programování, se začne zabývat konstrukcemi, jako je cyklus nebo podmínka. Tento postup, který si autor vyzkoušel na začínajících programátorech v programátorských kroužcích a který používá při výuce profesionálů, vás naučí od počátku myslet objektově. Ukazuje objekty jako něco opravdu přirozeného, jako něco, co výrazně usnadňuje přemýšlení o řešené úloze.

Při čtení Rudovy knihy jsem občas litoval, že už umím programovat, a tak jen doufám, že slibované další díly budou stejně dobré.

M. Virius

Úvod

Otevíráte knížku, která vás chce naučit programovat moderním, objektově orientovaným stylem. Stylem, jímž se v dnešní době vyvíjí drtivá většina klíčových aplikací, ale k jehož výuce ještě řada škol nedospěla. Po nastudování této knížky budou proto mnozí z vás vědět o moderním programování víc než leckterý z učitelů.

Komu je kniha určena

Tato knížka je určena pro ty, kteří to se svoji touhou naučit se moderně programovat myslí vážně a chtějí se naučit programovat dobře. Nejsem přítelem stručných náznaků. Naopak, budu se vám v ní snažit předvést všechny klíčové dovednosti krok za krokem a ukázat vám úskalí, která vás mohou očekávat při tvorbě vašich vlastních programů.

Knížka je primárně určena těm, kteří ještě nikdy neprogramovali. První vydání jsem psal na základě zkušeností z výuky programování na základních a středních školách doplněných o další zkušenosti získané z vedení kurzů pro profesionální programátory přecházející z klasického programování na programování objektově orientované.

Při vedení těchto kurzů jsem si uvědomil, že to, co děti (a částečně i dospělí, kteří s programováním teprve začínají) pochopí poměrně snadno, zvládají programátoři s předchozími zkušenostmi z neobjektového programování obtížně[1]. Spoustu úsilí totiž musí věnovat tomu, aby se nejprve odnaučili mnohé z toho, co si před tím pracně osvojili. Teprve pak začnou pomalu vstřebávat jiný způsob programátorského myšlení, jenž objektově orientované programování vyžaduje.

Od čtenářů jejího předchozího vydání jsem dostal řadu mailů, v nichž mi psali, že jsou sice zkušenými programátory, ale teprve zde pochopili některé věci, které jim ostatní učebnice a kurzy nedokázaly vysvětlit. Uvědomil jsem si přitom, že knihu mohou s výhodou použít i ti, kteří již nějakou dobu programují a potřebují se přeškolit z klasického, strukturovaného programování na programování objektově orientované (80 % účastníků mých kurzů). Při úpravách pro toto vydání jsem se proto snažil, aby se kniha stala co nejužitečnější pro obě skupiny čtenářů.

Zanedlouho po vyjití prvního vydání jsem začal učit programování na katedře informačních technologií Fakulty statistiky a informatiky VŠE v Praze. Ke svému milému překvapení jsem zjistil, že tu kniha patří mezi doporučené učebnice stejně jako na řadě dalších univerzit, které se snaží naučit své studenty programovat objektově. K tomu, aby se stala plnohodnotnou učebnicí, jí však chyběl výklad některých pasáží, které se již do minulého vydání nevešly. Domluvili jsme se proto s nakladatelstvím, že toto vydání upravím a rozšířím tak, aby pokrylo látku prvního semestru výuky programování na FSI VŠE.

Co se naučíte

Musím vás upozornit na to, že se od běžných učebnic, s nimiž se můžete v současné době v knihkupectví setkat, poněkud liší. Současné učebnice programování jsou totiž většinou především učebnicemi nějakého programovacího jazyka. Jejich autoři se proto ve svém výkladu soustředí hlavně na výklad vlastností popisovaného jazyka a jeho knihoven. Dozvíte se z nich, kde použít kulaté

[1] Statistiky uvádějí, že typická doba přechodu je 12 až 18 měsíců, přičemž čím je programátor zkušenější, tím déle mu přerod trvá.

a kde hranaté závorky, kam patří středník a řadu další důležitých informací. Bohužel se v nich ale nedozvíte skoro nic o tom, jak při vývoji programů přemýšlet, aby vás nezaskočily náhlé změny zadání, kterými je současné programování pověstné. Autoři totiž předpokládají, že se při čtení jejich knihy naučíte programovat nějak sami od sebe.

Zkušenosti s programátory, kteří navštěvují moje kurzy ve firmě či na VŠE, však ukazují, že tohoto výsledku bývá dosaženo jen zřídka. Většina programátorů, kteří přicházejí do mých kurzů, zná poměrně dobře konstrukce nějakého objektově orientovaného programovacího jazyka, bohužel skoro nikdo z nich v něm neumí objektově programovat. Obdobné je to i se studenty, kteří s programováním začali před tím, než přišli na vysokou školu.

Dopředu proto říkám: toto není učebnice programovacího jazyka, toto je učebnice programování. Mým cílem není naučit vás specialitám použitého programovacího jazyka, ale naučit vás především efektivně navrhovat a vytvářet spolehlivé a snadno udržovatelné programy. Jinými slovy: chci vás naučit dovednostem, které budete používat, ať budete programovat v jakémkoliv objektově orientovaném jazyce. Jazyky přicházejí a odcházejí. Základní programátorské techniky a způsob myšlení však žijí daleko déle než jazyky, se kterými byly zavedeny.

Díky tomu, že se místo na programovací jazyk soustředím spíše na vlastní programování, vznikl v knize prostor pro výklad řady programátorských technik, o nichž se klasické učebnice vůbec nezmiňují, a to dokonce ani učebnice pro zkušené programátory; technik, které se většinou přednášejí až v nadstavbových kurzech. Přitom vůbec nejde o techniky složité, které by začátečník nezvládl pochopit. Učebnice je pomíjejí pouze proto, že nesouvisí přímo se syntaxí jazyka, ale týkají se obecného programování.

Na školách se setkáte se dvěma přístupy k výuce programování. Jeden vychovává studenty tak, aby vyhrávali programátorské soutěže. Učí je různé finty a triky, které se dají upotřebit při řešení opravdu zapeklitých problémů, ale vedle toho již nezbývá místo na získání návyků umožňujících efektivně řešit 95 % běžných úloh.

Druhý styl výuky učí studenty programovat tak, aby si programováním dokázali co nejlépe vydělávat. Učí studenty základní pravdě, že jediné, na co se mohou při vývoji programů spolehnout, je skutečnost, že zadání úlohy se zanedlouho změní, a že proto musí navrhovat programy tak, aby je tyto změny nezaskočily, ale aby je dokázali do svého návrhu rychle a efektivně začlenit.

Tato učebnice se soustředí na druhý způsob výuky. Nebude vám proto ukazovat skoro žádné triky, které použijete v 5 % opravdu zapeklitých úloh, ale zato vás naučí řadu „triků“, které vám pomohou efektivně vyřešit běžné problémy, které by na vás mohly číhat při řešení těch zbylých 95 % úloh. Naučíte se s ní vytvářet programy tak, aby s vámi mohly růst a aby vás nezaskočily rychle se měnící požadavky zákazníků (a připravte se na to, že tyto měnící se požadavky vás potkají i v případě, kdy vaším zákazníkem jste vy sami).

Styl výuky

Před chvílí jsem řekl, že kniha pokrývá látku prvního semestru výuky programování na FSI VŠE. Není to však odborným stylem psaná vysokoškolská učebnice. Naopak. Snažil jsem se ji psát tak, aby se dobře četla i středně bystrému středoškolákovi či středoškolačce a aby, jak s oblibou říkám, vykládaná látka „dobře tekla do hlavy“.

Už podle tloušťky knihy jste nejspíše odhadli, že není určena těm, kteří hledají dvousetstránkovou rychloučebnici, s jejíž pomocí se naučí programovat za víkend. Jestli jste tuto knihu otevřeli, tak asi patříte k těm, kteří vědí, že taková učebnice neexistuje. Dvousetstránková knížka bude možná levná, ale může věci pouze naznačovat, takže se její čtenář dostane ke skutečnému poznání až po následném dlouhém a usilovném samostudiu. Předpokládám proto, že už víte, že tenké učebnice patří ve skutečnosti k těm nejdražším, protože to, co ušetříte při jejich koupi, mnohonásobně ztratíte při následném zdlouhavém osvojování si stručně a náznakově probrané látky.

Tato kniha se od ostatních učebnic (a řady univerzitních kurzů) programování liší ještě v jedné věci: většina ostatních učebnic a kurzů objektově orientovaného jazyka sice na začátku vysvětlí, co je to objektové programování, ale pak na něj na chvíli zapomene a začne výkladem klasických programovacích konstrukcí. Když se pak pracně probojují k výkladu objektových konstrukcí, studenti již mají nacvičené strukturované programování, takže nadále vyvíjejí strukturované programy, pouze v nich používají objektové konstrukce. Neuvědomují si přitom, že používat třídy a objekty ještě neznamená programovat objektově[1].

My to uděláme právě obráceně. Jestli se vám má dostat objektově orientované myšlení pod kůži, musíme s jeho výkladem začít hned a nezatěžovat vás napřed klasickými konstrukcemi, které by vaše myšlení směřovaly trochu jinam, takže byste se museli po chvíli zase přeorientovávat. Na klasické konstrukce samozřejmě nezapomeneme, ale dojde na ně řada až v době, kdy již budete mít za sebou několik objektově orientovaných programů, které budete moci pomocí těchto konstrukcí dále vylepšovat.

Oproti běžným zvyklostem spolu v této učebnici nebudeme řešit pouze jednoduché úlohy, jejichž hlavním účelem je demonstrovat vysvětlovanou vlastnost jazyka (i když se jim nebudu vyhýbat), ale budu se vám naopak snažit předkládat i úlohy složitější, a to i za cenu toho, že část úlohy, která bude používat doposud nevysvětlené konstrukce, za vás budu muset předem vyřešit sám. Takovéto úlohy daleko lépe odpovídají těm, s nimiž se budete v praxi setkávat. Typickou úlohou programátora totiž není navrhnout kompletní řešení nějakého jednoduchého problému, ale naopak doplnit stávající, většinou velmi složitý a někým jiným napsaný program o nějakou novou funkci.

Při práci na takovýchto příkladech si vyzkoušíte další potřebnou dovednost, kterou je schopnost orientovat se v programu, který je mnohem složitější, než byste sami dokázali v daném okamžiku naprogramovat.

Programovací jazyk

Sliboval jsem, že se nebudu soustředit na výuku jazyka, ale na výuku programování. Výuce programovacího jazyka se však nevyhneme. Budete-li si chtít vyzkoušet to, co jste se naučili, nezbude vám, než program v nějakém programovacím jazyce vytvořit. Pro demonstraci látky vysvětlované v této učebnici budu používat programovací jazyk Java. Zvolil jsem jej z několika důvodů:

- v současné době je to nejrozšířenější programovací jazyk[2],
- je to moderní programovací jazyk, na němž lze demonstrovat použití všech důležitých postupů,
- oproti jiným současným jazykům je doopravdy jednoduchý, takže se jej snadno naučíte,
- překladač i ostatní vývojové nástroje je možné získat zdarma,
- vytvořené programy nejsou omezeny na jediný operační systém, ale můžete je přenášet mezi různými operačními systémy,
- je k němu k dispozici vývojový nástroj specializovaný pro výuku, který je dokonce lokalizován do češtiny.

---

[1] Výzkum z přelomu století ukázal, že pouze 10 % programů psaných v objektově orientovaných jazycích je navrženo opravdu objektově. (Goddard, D. 1994. Is it really object oriented?. *Data Based Advis.* 12, 12 (Dec. 1994), 120-123.) Od té doby se situace trochu zlepšila, nicméně strukturovaně navržené programy psané v objektových jazycích stále převažují.

[2] Zájemce odkážu na **http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html**, kde najdou vývoj popularity jednotlivých programovacích jazyků od roku 2001.

Jak jsem již řekl, v této učebnici se chci soustředit spíše na to jak programovat a programovací jazyk používám pouze jako prostředek k tomu, abychom si mohli vysvětlené věci hned také vyzkoušet. Zkoušet ale budeme hodně, takže se v průběhu výuky naučíte nejenom programovat, ale zároveň získáte potřebnou praxi při řešení nejrůznějších úloh v programovacím jazyku Java.

Uspořádání

Kniha je rozdělena do tří částí. V první části se seznámíte s třídami a objekty a naučíte se pracovat s vývojovým prostředím. Druhá část před vámi odkryje klíčová zákoutí objektově orientovaného programování a naučí vás přemýšlet při tvorbě programů objektově. Třetí část pak doplní vaše znalosti o klasické programové konstrukce a ukáže vám, jak se řeší složitější úlohy.

Většina úloh, na jejichž řešení vykládanou látku demonstruji, jsou simulace nebo jednoduché hry. Říkal jsem si, že pro vás budou takovéto příklady mnohem zajímavější než obsluha bankovních účtů a další praktické úlohy, s nimiž se v učebnicích často setkáváme. Navíc se domnívám, že se na nich dá vysvětlovaná látka předvést mnohem názorněji.

Úlohy v první části knihy budou, pravda, celkem triviální, protože toho ještě nebudete moc umět, i když i tady si zkusíte naprogramovat část jednoduché hry. Druhá část už přinese zajímavější úlohy s nejrůznějšími animacemi, ale stále půjde převážně pouze o součásti větších projektů. Ve třetí části se pak konečně objeví úlohy, které budete moci vyřešit od počátku do konce.

Žádná z úloh zadaných v této knize není pouze zadána. **Všechny zadané úlohy jsou vyřešeny, abyste si mohli porovnat své řešení se vzorovým** a abyste se v případě, kdy se dostanete do těžkostí a nebudete si vědět rady, měli kam obrátit pro inspiraci. Budete-li se však chtít opravdu naučit programovat, doporučuji vám vyřešit všechny tyto doplňkové úlohy vlastní hlavou.

Čeština

Někteří čtenáři mne občas napadají za to, že ve svých textech používám důsledně českou terminologii. Za dlouhou dobu své učitelské praxe jsem si však vyzkoušel, že používání původních anglických termínů v začátečnických kurzech není dobré řešení. Začátečníci mívají problémy s pochopením vlastní látky a přidání termínů, kterým nerozumějí (znalost angličtiny u nás stále není na takové úrovni, jakou bychom rádi viděli), jim situaci pouze ztěžuje.

Praxe pak ukazuje, že mnozí „anglicky hovořící“ programátoři schovávají za anglické termíny to, že problému sami nerozumějí a při používání slangu působí na neznalé okolí jako odborníci. Když na začátečníka vybafnu např. název *singleton*, málokterý bude vědět, co to slovo znamená, a nezbude mu, než si je zapamatovat jako nějaký nový, cizí termín. Když se pak po pár týdnech výuky zeptám, jaké vlastnosti má návrhový vzor singleton, začnou žáci nejprve tápat, který z probraných vzorů to je, a v řadě případů jej zamění s nějakým jiným.

Když naproti tomu použiji pro daný návrhový vzor termín *jedináček*, všichni si jej ihned pevně spojí se svojí představou jedináčka. Nejenom že jej pak i po týdnech správně vyloží, ale navíc i lépe pochopí jeho podstatu.

Prosím proto čtenáře, kteří jsou hrdí na svoji znalost angličtiny, aby se smířili s tím, že budu vycházet vstříc většině, která konstrukce označené českými termíny lépe pochopí a daleko lépe si je zapamatuje. Ti, kteří můj počeštěný výklad nepotřebují, se jistě již dávno poučili z některé anglicky psané učebnice.

Protože je však programátorský svět veskrz anglický[1], uvedu u každého termínu při jeho zavedení i příslušný anglický ekvivalent. Všem vám pak doporučuji si tento ekvivalent zapamatovat, protože řada českých i slovenských autorů z nejrůznějších důvodů trvá na používání anglických termínů doplněných českými, resp. slovenskými koncovkami.

Proč je kniha tlustá

Na závěr tohoto úvodu dovolte ještě jednu omluvu. Vím, že kniha je výrazně tlustší, než začátečnické učebnice obvykle bývají. Není to proto, že bych toho v ní chtěl vysvětlit tak moc, je to spíš proto, že se snažím projít všechna krizová místa spolu s vámi krok za krokem.

Nechtěl jsem před vás jen předhodit hotová řešení, ale chtěl jsem vám názorně ukázat, jak byste mohli k takovémuto řešení sami dospět. Obdobně jsem nechtěl jen naznačovat, na co si máte dát pozor, ale u většiny častých chyb jsem také uvedl příklad, v němž se chyba nebo nepříjemná situace vyskytuje, a zároveň ukázal řešení nebo potřebnou reakci. Protože je ale takových míst hodně, kniha trochu narostla. Doufám, že i vy budete tyto podrobné vysvětlivky považovat za užitečné.

Knize přidalo na tloušťce i to, že jsem občas zabrousil do oblastí, které se sice do začátečnických příruček obvykle neprobojují, avšak jejich neznalost pak dělá začátečníkům o to větší problémy.

Kniha vznikala pomalu a dlouho. Přes veškeré úsilí, které jsme ji já i moji spolupracovníci věnovali, nemohu vyloučit, že v textu či doprovodných příkladech zůstaly skryté nějaké chyby. Předem se za ně omlouvám a prosím vás, abyste mi o nich napsali na adresu rudolf@pecinovsky.cz. Já bych je opravil a opravu vystavil na webové stránce této knihy http://knihy.pecinovsky.cz/mojj5.

Potřebné vybavení

Pro úspěšné studium této knihy budete potřebovat čtyři věci:

- dostatečně výkonný počítač,
- základní vývojovou sadu Javy,
- vývojové prostředí *BlueJ*,
- programy používané v příkladech,
- budete-li se chtít podívat na doprovodné animace, tak budete potřebovat internetový prohlížeč, který je schopen přehrát flashové animace.

Doporučená konfigurace

Používáte-li operační systém *Windows*, měla by to být minimálně verze Windows 2000 (Java 5.0 je schopná běhat i na Windows 98). Používáte-li systém *Linux*, lze předpokládat, že budete mít dostatečně novou verzi. Jste-li uživateli počítačů *Macintosh*, měli byste mít systém podporující minimálně Javu 5.0.

Pro úspěšné spouštění programů pod Javou 5.0 na *Windows 98* byste měli vystačit s 64 MB paměti, při přechodu na novější operační systémy a novější verze Javy však nároky na paměť rychle

[1] Když jsem po škole nastupoval v akademii, položil mi můj školitel otázku: „Umíte anglicky?“ Než jsem si zformuloval odpověď, která by charakterizovala úroveň mých znalostí, odpověděl si sám: „No ono je to jedno – buďto budete umět anglicky nebo změníte zaměstnání.“ A totéž platí pro všechny, kteří se chtějí vážně zabývat programováním.

rostou. Pro provoz pod *Windows XP* bych vám doporučoval 256 MB, u pažravých *Windows Vista* 1024 MB. S uvedenými konfiguracemi byste měli při troše trpělivosti vystačit. Těm méně trpělivým bych však pro pohodlnou práci doporučoval raději dvojnásobek.

Paměťové nároky pro *Linux* a *Macintosh* jsou obdobné – vždy záleží na verzi používaného systému. Jsou však řádově srovnatelné s nároky systémů z rodiny *Windows*.

Vývojová sada JDK, o níž budu hovořit za chvíli, zabere na disku po rozbalení (v závislosti na verzi) přibližně 300 až 400 MB a dokumentace dalších 250 až 350 MB. Budete-li si chtít rozbalit soubor se zdrojovými kódy, abyste se mohli podívat, jak je to či ono v Javě naprogramováno, budete potřebovat dalších 70 až 90 MB.

Sada JDK (Java Development Kit)

K vývoji programů budete potřebovat vývojovou sadu JDK, kterou můžete zdarma stáhnout na webové adrese http://java.sun.com/j2se nebo získat na CD přibaleném k některému počítačovému časopisu. Nabízí-li vám někdo starší počítač s již nainstalovanou Javou, zkontrolujte, že se jedná o Javu 5.0 nebo mladší. Na starších verzích Javy naše programy pracovat nebudou.

Každopádně bych vám doporučoval použít co nejnovější verzi. Od verze 5.0 sice v jazyku nepřibyly žádné vlastnosti ani funkce, které by se dotkly programátorských začátečníků (hovořím o verzích 6.0 a 7.0, které byly známé v době vydání knihy), ale každá novější verze udělala výrazný krok v efektivitě. Po verzi 5.0 proto doporučuji sáhnout pouze v případě, že používáte operační systém *Windows 98*, který již mladší verze nepodporují, nebo pracujete na počítači *Macintosh* a nechce se vám investovat do nové verze operačního systému.

K vývojové sadě si nezapomeňte stáhnout a nainstalovat i dokumentaci. Je přibližně stejně rozsáhlá, jako samotná vývojová sada (ale nebojte se, nemusíte ji číst celou) a při jakémkoliv programování za hranicemi příkladů z učebnice se bez ní neobejdete.

Vývojové prostředí

S JDK při vývoji programů teoreticky vystačíte (některé učebnice ani nic jiného nepoužívají), nutí vás však starat se o řadu konfiguračních detailů, které vás odvádějí od vlastního programování. Převážná většina programátorů proto používá vývojové prostředí, které tyto detaily vyřeší za ně a navíc jim pomůže i v řadě dalších oblastí.

Já budu při výkladu používat vývojové prostředí *BlueJ*, které můžete (rovněž zdarma) stáhnout na adrese http://www.bluej.org. Ve srovnání s vlastní Javou je toto prostředí nenáročné. Instalační soubory mají necelé 4 MB a po instalaci zaberou přibližně dvojnásobek.

Někteří čtenáři minulého vydání si stěžovali, že je práce v tomto prostředí obtěžuje, protože prostředí se výrazně odlišuje od běžného standardu. Opravdu se odlišuje – jeho možnosti můžete skrz naskrz prozkoumat tak za 20 minut, kdežto naučit se dokonale některé standardní vývojové prostředí je zhruba stejně těžké jako naučit se Javu. Navíc prostředí *BlueJ* umožňuje velmi jednoduše znázornit strukturu programu, což jiná prostředí neumožňují.

Ty, kterým *BlueJ* nesedne, k jeho používání nenutím. Budete je potřebovat pouze v 2. kapitole, protože žádné jiné prostředí prozatím možnost interaktivního režimu nenabízí. Od 3. kapitoly se však už můžete přepnout na libovolné jiné a pasáže, v nichž vysvětluji některé postupy vázané na prostředí, prostě přeskočit a nastudovat si tyto postupy pro prostředí vámi používané. Mohu-li vám však doporučit, zůstaňte během studia této učebnice u *BlueJ* (alespoň do konce druhé části). Pak už budete mít dostatečné návyky, abyste mohli přejít na libovolné jiné – nejlépe na *NetBeans* nebo *Eclipse*.

Konfigurační soubor pro prostředí BlueJ

K prostředí *BlueJ* doporučuji stáhnout a instalovat konfigurační soubor, který najdete na adrese http://knihy.pecinovsky.cz/mojj5 spolu s návodem, jak jej použít a konfiguraci *BlueJ* upravit. Po instalaci tohoto konfiguračního souboru se *BlueJ* lokalizuje a začne se s vámi bavit česky. Oproti původní konfiguraci budou naše programy navíc i trochu barevnější, což vám umožní se v nich lépe orientovat. Poslední výhodou této konfigurace je, že přepne celé prostředí do kódování UTF-8, takže přestanete mít problémy s přenášením souborů mezi jednotlivými platformami a uživatelé používající *Windows* si budou moci vesele vyměňovat své programy s uživateli systémů *Linux* či *Macintosh*, a to nezávisle na tom, používají-li diakritiku.

Doprovodné programy

Na stránce s konfiguračním souborem, tj. na adrese http://knihy.pecinovsky.cz/mojj5, najdete také dva soubory s projekty použitými v knize. První z nich obsahuje zdrojové soubory programů přesně v té podobě, jak jsou uvedeny v knize, tj. včetně diakritiky. Druhý je pak určen pro ty, kteří diakritiku v programech nesnáší.

Abyste si mohli navíc převést do „nediakritického" tvaru i jiné programy, je pro vás na webu připraven program `Odhackuj` (jeho název je schválně bez diakritiky), který zkopíruje soubory uložené v kódování UTF-8 do zadaného adresáře a přitom zbaví jejich názvy i jejich obsah veškeré diakritiky. Tento program je napsaný v Javě, takže by měl chodit na všech počítačích.

Na této stránce bude zároveň průběžně aktualizovaný návod na stažení Javy a *BlueJ* určený pro ty, kteří budou mít s jejich stažením problémy. Kdyby ani tento návod nestačil, pošlete mail a pokusíme se vaše instalační problémy nějak vyřešit.

Doprovodné animace

Sami jistě ze zkušenosti víte, že dobrý příklad je v řadě situací lepší než řada slov. Od minulého vydání jsem proto připravil celou řadu animovaných ukázek, v nichž krok za krokem demonstruji některé vysvětlované postupy. Najdete je na adrese http://vyuka.pecinovsky.cz/animace.

Tyto animace využijete hlavně při čtení první části knihy, protože demonstrují především práci s prostředím *BlueJ* a tvorbu základních konstrukcí. Ukazují jak tyto konstrukce použít a předvádějí chování programů, v nichž jsou vysvětlené konstrukce použity.

Animace však nejsou navázány na konkrétní kapitoly knihy. Vznikly jako doprovod k mým přednáškám na VŠE a VOŠIS. Vedle českých animací najdete na zmíněné stránce i řadu anglických, které zase slouží jako doprovod k mému kurzu programování, který běží na stránkách firmy *Sun*.

Použité konvence

K tomu, abyste se v textu lépe vyznali a také abyste si vykládanou látku lépe zapamatovali, používám několik prostředků pro odlišení a zvýraznění textu.

Objekty	První výskyt nějakého termínu a další texty, které chci zvýraznit, vysazuji **tučně**.
Názvy	Názvy firem a jejích produktů vysazuji *kurzivou*. Kurzivou vysazuji také názvy kapitol, podkapitol a oddílů, na které se v textu odkazuji.
Citace	Texty, které si můžete přečíst na displeji, např. názvy polí v dialogových oknech či názvy příkazů v nabídkách, vysazuji **tučným bezpatkovým písmem**.

Adresy Názvy souborů a internetové adresy vysazuji **obyčejným bezpatkovým písmem.**

Program Texty programů a jejich částí vysazuji `neproporcionálním písmem.`

Kromě částí textu, které považuji za důležité zvýraznit nebo alespoň odlišit od okolního textu, najdete v textu ještě řadu doplňujících poznámek a vysvětlivek. Všechny budou v jednotném rámečku, který bude označen ikonou charakterizující druh informace, kterou vám chce poznámka či vysvětlivka předat.

Symbol jin-jang bude uvozovat poznámky, s nimiž se setkáte na počátku každé kapitoly. Zde vám vždy prozradím, co se v dané kapitole naučíte.

Otevřená schránka s dopisy označuje informace o projektu, s nímž budeme v dalším textu pracovat nebo v něm najdete vzorové řešení.

Obrázek knihy označuje poznámku týkající se používané terminologie. Tato poznámka většinou upozorňuje na další používané termíny označující stejnou skutečnost nebo na konvence, které se k probírané problematice vztahují. Seznam všech terminologických poznámek najdete v rejstříku pod heslem „terminologie“.

Obrázek počítače označuje zadání úkolu, který máte samostatně vypracovat. Seznam všech úloh najdete v rejstříku pod heslem „úloha“.

Píšící ruka označuje obyčejnou poznámku, která pouze doplňuje informace z hlavního proudu výkladu o nějakou zajímavost.

Ruka s hrozícím prstem upozorňuje na věci, které byste měli určitě vědět a na které byste si měli dát pozor, protože jejich zanedbání vás většinou dostane do problémů.

Usměváček vás bude upozorňovat na různé tipy, kterými můžete vylepšit svůj program nebo zefektivnit svoji práci.

Mračoun vás naopak bude upozorňovat na různá úskalí programovacího jazyka nebo programů, s nimiž budeme pracovat, a bude vám radit, jak se těmto nástrahám vyhnout či jak to zařídit, aby vám alespoň pokud možno nevadily.

Brýle označují tzv. „poznámky pro šťouraly“, ve kterých se vás snažím seznámit s některými zajímavými vlastnostmi probírané konstrukce nebo upozorňuji na některé souvislosti, avšak které nejsou k pochopení látky nezbytné.

Symbol znamení raka označuje poznámku, ve které poukazuji na interpretaci nějakého obratu či konstrukce v nějaké analogii, nejčastěji v analogii ze světa robotů, kterou zavádím v podkapitole *Analogie* na straně 53. Seznam odkazů na tuto analogii najdete v rejstříku pod heslem „analogie“.

Odbočka

Občas je potřeba vysvětlit něco, co nezapadá přímo do okolního textu. V takových případech používám podšeděný blok se silnou čarou po straně. Tento podšeděný blok je takovou drobnou odbočkou od ostatního výkladu. Nadpis podšeděného bloku pak najdete i v podrobném obsahu mezi nečíslovanými nadpisy.

Část 1: Zapouzdření

Kapitola 1
Seznamujeme se s nástroji

Co se v kapitole naučíme
V této kapitole se seznámíte s nástroji, které budete při studiu dalších částí knihy potřebovat. Nejprve vám povím o historii a současných trendech v programování a prozradím vám, proč jsem pro výuku vybral programovací jazyk Java. Pak vám ukážu vývojové prostředí *BlueJ* a naznačím, jak je máte nainstalovat na svůj počítač. Poté vám vysvětlím, jak jsou organizovány doprovodné projekty k učebnici, a ukážu vám, jak je připravit, abyste s nimi mohli v průběhu studia učebnice snadno pracovat. Na závěr vám předvedu, jak otevřít projekt ve vývojovém prostředí *BlueJ*, prozradím vám, co je to diagram tříd, a naučím vás, jak je možno tento diagram v prostředí *BlueJ* upravovat.

1.1 Trochu historie

První počítače

Historie počítačů (tj. strojů, u nichž je postup výpočtu řízen programem) a programování se začala psát již v devatenáctém století. Charles Babbage tehdy dostal zakázku od anglického námořnictva na vylepšení svého počítacího stroje pro výpočet navigačních tabulek, které se hemžily chybami. V roce 1848 dokončil návrh počítacího stroje, který byl řízený programem zadávaným na děrných štítcích. Zbytek života se pak věnoval jeho konstrukci. Stroj byl mechanický a měl být poháněn parním strojem, ale z programátorského hlediska již umožňoval značnou část operací, kterými se honosí současné počítače.

Zajímavé je, že prvním publikujícím programátorem byla žena – Augusta Ada Lovelance (mimochodem dcera známého romantického básníka lorda Byrona), která se s Babbagem spřátelila a pomáhala mu s některými výpočty. Jednou se na výletě v Itálii setkala s italským poručíkem (později generálem a předsedou vlády sjednocené Itálie) Menabreou, který pro místní smetánku přednášel o Babbageově stroji. Aby Babbageův stroj zpopularizovala, přeložila přednášku, která popisovala hardware tohoto počítače, do angličtiny a na Babbageův popud překlad doplnila „poznámkami překladatele", kde vysvětlila možnosti stroje ze softwarového hlediska. Svůj výklad doplnila i krátkým programem na výpočet Bernoulliho čísel – prvním publikovaným počítačovým programem na světě[1]. (Na její počest byl v osmdesátých letech minulého století pojmenován programovací jazyk Ada.)

[1] Poznámky překladatele doplnila až po Babbageově přemlouvání. Babbage ji chtěl ušetřit práci a program pro ni napsal, jenže ona mu v něm našla chybu a definitivní, publikovaná verze je již její dílo. Není tedy prvním člověkem, který program napsal, ale je prvním člověkem, který svůj program publikoval.

Babbageovi se jeho počítač nepodařilo rozchodit, protože vyžadoval součástky z materiálu s pevností pancéřové oceli opracované s hodinářskou přesností, a to bylo pro tehdejší technologii příliš velké sousto[1]. Rozchodil jej až jeho vnuk, který byl americkým generálem, a při odchodu do penze se rozhodl, že všem ukáže, že dědeček nebyl blázen a jeho stroj by byl schopen provozu.

První funkční počítač postavil až v roce 1938 v Německu Konrád Zuse. Další počítače se objevily v průběhu druhé světové války. Nejslavnějším z nich byl ENIAC, který byl vyroben v roce 1944 a byl prvním čistě elektronickým počítačem (ostatní ještě používaly relé[2] či dokonce mechanické prvky). Skutečné počítačové (a tím i programátorské) orgie však začaly až v padesátých letech, kdy se začaly počítače sériově vyrábět a počítačová věda (computer science) se zabydlela na všech univerzitách.

Problémem prvních počítačů byla jejich vysoká poruchovost. Antonín Svoboda tehdy přišel s řadou konstrukčních vylepšení umožňujících vytvořit z nespolehlivých součástek spolehlivý počítač[3]. Konstruktéři se postupně naučili vytvářet vysoce spolehlivé počítače. Nejslabším článkem celého komplexu se tak stával program.

Co je to program

Program bychom mohli charakterizovat jako v nějakém programovacím jazyce zapsaný předpis, popisující, jak má procesor, pro nějž je program určen (v našem případě počítač) splnit zadanou úlohu.

Cílovým procesorem nemusí být vždy počítač. Oblíbeným příkladem programů jsou např. kuchařské předpisy. V předpočítačových dobách zaměstnávaly některé instituce (např. armáda) velké skupiny počtářů, které na mechanických kalkulačkách řešili podle zadaných programů výpočty, které dnes předhazujeme počítači.

Na programu je důležité, že musí být napsán v nějakém programovacím jazyce, kterému rozumí programátor i počítač. Vybrat jazyk pro kuchařský předpis je jednoduché, vybrat jazyk pro počítač je mnohem složitější. Kvalita použitého jazyka totiž naprosto zásadně ovlivňuje i kvalitu výsledných programů. Proto také prošly jak programovací jazyky, tak metodiky jejich používání celou řadou revolučních změn.

Program musí být především spolehlivý

Počítače byly postupně nasazovány v dalších a dalších oblastech a programátoři pro ně vytvářeli dokonalejší a dokonalejší programy. Programy byly čím dál rafinovanější a složitější a to začalo vyvolávat velké problémy. Programátoři totiž přestávali být schopni své programy rozchodit a když je vítězně rozchodili, nedokázali z nich v rozumném čase odstranit chyby, které uživatelé v programu objevili.

Tato krize vedla postupně k zavádění nejrůznějších metodik, které měly jediný cíl: pomoci programátorům psát spolehlivé a snadno upravovatelné programy. V padesátých letech minulého století se tak prosadily vyšší programovací jazyky, v šedesátých letech modulární programování, v sedmdesátých letech na ně navázalo strukturované programování a v průběhu osmdesátých a zejména pak devadesátých let ovládlo programátorský svět objektově orientované programování, jehož vláda pokračuje dodnes. (Jednotlivé termíny si za chvíli probereme podrobněji).

---

1 Takhle se to alespoň dlouho tvrdilo. V roce 1991 však Doron Swade, kurátor londýnského muzea vědy, sestavil Babaggeův počítač pouze s použitím technologií dostupných v polovině devatenáctého století.

2 Relé je elektromechanický prvek, kde průchod proudu cívkou zapříčiní sepnutí nebo rozpojení kontaktů. Rychlá relé dokázala sepnout i 100krát za sekundu – to byla také maximální rychlost tehdejších počítačů.

3 V tehdejší době (tj. v padesátých letech) byla naše republika na špičce světového vývoje počítačů. Svobodovy myšlenky byly později uplatněny v počítačích řídících americké kosmické lodě. Jenže to už bylo v době, kdy byl z naší republiky vypuzen, emigroval do USA a tam působil jako špičkový počítačový odborník.

Hlavním cílem programátorů v počátcích programování bylo, aby jejich programy spotřebovaly co nejméně paměti a byly co nejrychlejší. Tehdejší počítače totiž měly paměti málo, byly velice pomalé a jejich strojový čas byl drahý. Se stoupající složitostí programů však byly takto psané programy stále méně stabilní a stále hůře udržovatelné. Současně s tím, jak klesala cena počítačů, jejich strojového času i paměti, začínal být nejdražším článkem v celém vývoji člověk.

Cena strojového času a dalších prostředků spotřebovaných za dobu života programu začínala být pouze zlomkem ceny, kterou bylo nutno zaplatit za jeho návrh, zakódování, odladění a následnou údržbu. Začal být proto kladen stále větší důraz na produktivitu programátorů i za cenu snížení efektivity výsledného programu.

Prakticky každý program zaznamená během svého života řadu změn. Tak, jak se průběžně mění požadavky zákazníka na to, co má program umět, je program postupně upravován, rozšiřován a vylepšován. Celé současné programování je proto vedeno snahou psát programy nejenom tak, aby pracovaly efektivně, tj. rychle a s minimální spotřebou různých zdrojů (operační paměť, prostor na disku, kapacita sítě atd.), ale aby je také bylo možno kdykoliv jednoduše upravit a vylepšit.

Předchozí zásady krásně shrnul Martin Fowler ve své knize Refactoring:

„Napsat program, kterému porozumí počítač, umí i hlupák.
Dobrý programátor píše programy, kterým porozumí i člověk."

Mějte při tvorbě svých programů tuto větu neustále na paměti.

1.2 Objektově orientované programování – OOP

Vývoj metodik programování

Jak jsem řekl, v průběhu doby se prosadilo několik metodik, které doporučovaly, jak programovat, abychom byli s programem co nejdříve hotovi a výsledný program byl co nejkvalitnější. Tyto metodiky na sebe postupně navazovaly. Každá z nich těžila ze zkušeností získaných při aplikaci předchozí metodiky a stavěla na nich.

První revolucí bylo zavedení vyšších programovacích jazyků, které programátory osvobodily od zapisování programů v podobě, která přesně odpovídala použitým instrukcím použitého počítače, a zavedly vyjadřování, které bylo daleko bližší zvyklostem lidí.

Zavedení vyšších programovacích jazyků umožnilo tvorbu složitějších programů. Vyvíjené programy byly zanedlouho tak složité, že se v nich programátoři přestali vyznávat. Modulární programování ukazovalo, že rychlost vývoje i kvalitu výsledného programu zvýšíme, když vhodně rozdělíme velký projekt do sady menších, rozumně samostatných modulů.

Produktivita programátorské práce se zvýšila, ale záhy se začalo ukazovat, že programátoři mají problémy nejenom s rozchozením obřích programů, ale i jednotlivých modulů. Strukturované programování se proto ponořilo do hloubky kódu a ukazovalo, že dalšího zvýšení produktivity vývoje i kvality výsledných programů dosáhneme dodržením několika jednoduchých zásad při vlastním psaní kódu. Jeho hlasatelé předváděli, že opuštěním nejrůznějších fint, kterými se programátoři snažili o optimalizaci kódu, a maximálním zpřehledněním vytvářeného programu dosáhneme nejenom vyšší produktivity, ale v mnoha případech i vyšší efektivity výsledného programu.

Se stále rostoucí složitostí vyvíjených programů se začalo ukazovat, že jednou z největších překážek v efektivní tvorbě kvalitních programů je tzv. *sémantická mezera* mezi tím, co chceme vytvořit a tím, co máme k dispozici. Naše programy mají řešit široké spektrum úkolů od řízení mikrovlnné trouby přes nejrůznější kancelářské a grafické programy a hry až po složité vědecké úlohy, kosmické lety či protivzdušnou obranu kontinentu. Ve svém rozletu jsme ale odkázáni na stroje, které si umějí pouze hrát s nulami a jedničkami. Čím budou naše vyjadřovací možnosti blíže zpracovávané skutečnosti, tím rychleji a lépe dokážeme naše programy navrhnout, zprovoznit a udržovat.

Jinými slovy: Kvalita programu a rychlost jeho tvorby je velice úzce svázána s hladinou abstrakce, kterou při jejich tvorbě používáme. Budeme-li např. programovat ovládání robota, bude pro nás výhodnější programovací jazyk, v němž můžeme zadat příkazy typu „zvedni pravou ruku“, než jazyk, v němž musíme vše vyjadřovat pomocí strojových instrukcí typu „dej do registru A dvojku a obsah registru A pak pošli na port 27“.

Objektově orientované programování (v dalším textu budu využívat zkratku OOP) se proto obrátilo do vyšších hladin a ukázalo, že vhodným zvýšením abstrakce dokážeme rychle a spolehlivě navrhovat a vyvíjet ještě větší a komplikovanější projekty.

OOP přichází s výrazovými prostředky, které nám umožňují maximálně zvýšit hladinu abstrakce, na které se „bavíme“ s našimi programy a tím maximálně zmenšit onu sémantickou mezeru mezi tím, co máme k dispozici a co bychom potřebovali. Chceme-li však jeho výhod plně využít, nesmíme zapomínat na nic z toho, s čím přišly předchozí metodiky.

Principy OOP

Objektově orientované programování vychází z myšlenky, že všechny programy, které vytváříme, jsou simulací buď skutečného nebo nějakého námi vymyšleného virtuálního světa. Čím bude tato simulace přesnější, tím bude výsledný program lepší.

Všechny tyto simulované světy jsou ve skutečnosti světy objektů, které mají různé vlastnosti a schopnosti a které spolu nějakým způsobem komunikují (říkáme, že si navzájem posílají zprávy). Touto komunikací se přitom nemyslí jen klasická komunikace mezi lidmi či zvířaty, ale i mnohem obecnější vzájemné interakce (např. židle a podlaha spolu komunikují tak, že židle stojí na podlaze a naopak podlaha podpírá židli, aby se skrz ní neprobořila).

Budeme-li chtít, aby naše programy modelovaly tento svět co nejvěrněji, měly by být schopny modelovat obecné objekty spolu s jejich specifickými vlastnostmi a schopnostmi a současně modelovat i jejich vzájemnou komunikaci (tj. vzájemné posílání zpráv). Čím bude tento model věrnější, tím budou naše programy přesnějším modelem skutečností a tím i použitelnější, ale také spolehlivější a snáze udržovatelné.

Máme-li být schopni rychle vytvářet kvalitní programy, měli bychom mít k dispozici jazyk, jehož jazykové konstrukce nám umožní se vyjadřovat co nejpřirozeněji a co nejméně se nechat znásilňovat omezeními počítače, na kterém má program běžet. Musí umožnit co nejvyšší míru abstrakce, při níž se můžeme vyjadřovat tak, abychom mohli přirozeně popsat modelovanou skutečnost.

Všechny moderní programovací jazyky se honosí přídomkem „objektově orientované“. Tím se nám snaží naznačit, že nabízejí konstrukce, které umožňují rozumně modelovat náš okolní, objekty tvořený svět (a nejen jej).

1.3 Překladače, interprety, platformy

Tato podkapitola je určena těm, kteří se nespokojí jen s tím, že věci fungují, ale chtějí také vědět, jak fungují. Naznačíme si v ní, jak je to v počítači zařízeno, že programy pracují.

Operační systém a platforma

Operační systém je sada programů, jejímž úkolem je zařídit, aby počítač co nejlépe sloužil zadanému účelu. Operační systémy osobních počítačů se snaží poskytnout co největší komfort a funkčnost

jak lidským uživatelům, tak programům, které operační systém nebo tito uživatelé spouští. (Teď nehodnotím, jak se jim to daří.)

Operační systém se snaží uživatele odstínit od hardwaru použitého počítače. Uživatel může střídat počítače, avšak dokud bude na všech stejný operační systém, bude si se všemi stejně rozumět.

Při obsluze lidského uživatele to má operační systém jednoduché: člověk komunikuje s počítačem pomocí klávesnice, obrazovky, myši a případně několika dalších zařízení. Ty všechny může operační systém převzít do své správy a zabezpečit, aby se nejrůznější počítače chovaly vůči uživateli stejně.

U programů to má ale složitější. Programy totiž potřebují komunikovat nejenom s operačním systémem (např. když chtějí něco přečíst z disku nebo na něj zapsat), ale také přímo s procesorem, kterému potřebují předat své instrukce k vykonání. Problémem ale je, že různé procesory rozumí různým sadám instrukcí.

Abychom věděli, že náš program na počítači správně poběží, musíme vědět, že počítač bude rozumět té správné sadě instrukcí a že na něm poběží ten správný operační systém. Kombinaci *operační systém* + *procesor* budeme v dalším textu označovat jako **platforma**.

Nejrozšířenější platformou současnosti je operační systém *Windows* na počítačích s procesory kompatibilními s procesorem Intel Pentium. Další vám známou platformou bude nejspíš operační systém *Linux* na počítačích s procesory kompatibilními s Pentiem. Oba zmíněné operační systémy však mají své verze běžící na počítačích s jinými procesory.

Programovací jazyky

Jak asi všichni víte, pro zápis programů používáme nejrůznější programovací jazyky. Ty jsou vymýšleny tak, aby v nich mohl člověk co nejlépe popsat svoji představu o tom, jak má počítač splnit požadovanou úlohu.

Program zapsaný v programovacím jazyku pak musíme nějakým způsobem převést do podoby, které porozumí počítač. Podle způsobu, jakým postupujeme, dělíme programy na překládané a interpretované.

U **překládaných programů** se musí napsaný program nejprve předat zvláštním programu nazývanému překladač (někdo dává přednost termínu kompilátor), který jej přeloží (zkompiluje), tj. převede jej do podoby, s níž si již daná platforma ví rady, tj. musí jej přeložit do kódu příslušného procesoru a používat instrukce, kterým rozumí použitý operační systém. Přeložený program pak můžeme kdykoliv na požádání spustit.

Naproti tomu **interpretovaný program** předáváme v podobě, v jaké jej programátor vytvořil, programu označovanému jako interpret. Ten obdržený program prochází a ihned jej také provádí.

Výhodou překládaných programů je, že většinou běží výrazně rychleji, protože u interpretovaných programů musí interpret vždy nejprve přečíst kus programu, zjistit, co má udělat a teprve pak může tento požadavek vykonat.

Výhodou interpretovaných programů bývá na druhou stranu to, že jim většinou tak moc nezáleží na tom, na jaké platformě běží. Stačí, když na daném počítači běží potřebný interpret. Mohli bychom říci, že platformou těchto programů je právě onen interpret. Vytvoříte-li pro nový počítač interpret, můžete na něj vzápětí přenést i všechny vytvořené programy. Kdykoliv tyto programy po systému něco chtějí, požádají o to svoji platformu (interpret) a ta jim příslušné služby zprostředkuje. Takovým programům pak může být jedno, na jakém procesoru a pod jakým operačním systémem běží, protože se beztak „baví“ pouze se svojí platformou.

Naproti tomu překládané programy se většinou musí pro každou platformu trochu (nebo také hodně) upravit a znovu přeložit. Při implementaci programu pro více platforem bývá pracnost přizpůsobení programu jednotlivým platformám srovnatelná s pracností vývoje jeho první verze.

Vedle těchto základních druhů programů existují ještě **hybridní programy**, které jsou současně překládané i interpretované a které se snaží sloučit výhody obou skupin. Hybridní program se nejprve přeloží do jakéhosi mezijazyka, který je vymyšlen tak, aby jej bylo možno co nejrychleji interpretovat. Takto přeložený program je potom interpretován speciálním interpretem označovaným často jako virtuální stroj[1].

Hybridní programy spojují výhody obou kategorií. K tomu, aby v nich napsané programy mohly běžet na různých platformách, stačí pro každou platformu vyvinout potřebný virtuální stroj. Ten pak vytváří vyšší, mnohem univerzálnější platformu. Je-li tento virtuální stroj dostatečně „chytrý“ (a to jsou v současné době prakticky všechny), dokáže odhalit často se opakující části kódu a někam stranou je přeložit, aby je nemusel pořád kolem dokola interpretovat.

Ty nejchytřejší virtuální stroje dokonce sledují, co program dělá, a zjistí-li, že je to výhodné, přeloží rychle část programu znovu tak, aby využila některých právě platících speciálních podmínek, a mohla běžet ještě rychleji. Na těchto strojích pak může běžet hybridní program skoro stejně rychle jako překládaný a v některých speciálních případech dokonce i rychleji. Přitom si stále udržuje svoji nezávislost na hardwarové platformě a použitém operačním systému.

Na překládané, interpretované a hybridní bychom měli dělit programy, avšak často se takto dělí i programovací jazyky. Je sice pravda, že to, zda bude program překládaný, interpretovaný nebo hybridní není závislé na použitém jazyku, ale je to především záležitostí implementace daného jazyka, nicméně každý z jazyků má svoji typickou implementaci, podle které je pak zařazován.

Prakticky všechny jazyky sice mohou být implementovány všemi třemi způsoby a řada jich opravdu ve všech třech podobách existuje (jako příklad bychom mohli uvést jazyky Basic, Java nebo Pascal), ale u většiny převažuje typická implementace natolik výrazně, že se o těch ostatních prakticky nemluví. Z vyjmenované trojice je např. klasický Basic považován za interpretovaný jazyk, Java za hybridní a Pascal za překládaný.

Hybridní implementace jazyků se v posledních letech výrazně prosadily a jsou dnes králi programátorského světa. Vyvíjí v nich převážná většina programátorů a procento implementací v těchto jazycích neustále vzrůstá.

1.4 Java a její zvláštnosti

O splnění zásad objektově orientovaného programování se snaží tzv. objektově orientované jazyky. Mezi nimi je v současné době nejpopulárnější programovací jazyk Java, který se narodil v roce 1995. Hned po svém vzniku zaznamenal velký ohlas a během několika let v něm začalo vyvíjet své programy více než 3 milióny programátorů a toto číslo stále stoupá. V roce 2003 se podle průzkumů renomovaných společností stal nejpoužívanějším programovacím jazykem a tuto pozici si od té doby stále udržuje. Na převážné většině univerzit se Java používá jako vstupní jazyk pro výuku programování.

[1] Virtuální stroj se mu říká proto, že se vůči programu v mezijazyku chová obdobně, jako se chová procesor vůči programu v čistém strojovém kódu.

Klíčové vlastnosti Javy

Jaké jsou vlastnosti tohoto programovacího jazyka, že v tak krátké chvíli tak významně ovlivnil programátorský svět? Důvodů je celá řada. Zde uvedu jen ty z nich, které se výhodně projeví při výuce programování.

Objektově orientovaná

Java plně podporuje objektově orientované programování. Na rozdíl od C++ a dalších jazyků „klasického ražení" již neumožňuje napsat program, který by nebyl objektově orientovaný. Její autoři se přitom do ní snažili začlenit převážnou většinu zásad objektově orientovaného programování.

Předchozí tvrzení musím trochu zlehčit. Platí obecná zásada, že jako čuně mohu programovat v jakémkoliv programovacím jazyce. Java (a s ní i dalších moderní programovací jazyky) nedovoluje zapsat program, který by nebyl alespoň formálně objektově orientovaný. Bude-li objektově orientovaný doopravdy, tj. i svou architekturou a duchem, to už záleží na programátorovi. To za vás žádný programovací jazyk neudělá.

Jednoduchá

Java je velice jednoduchý jazyk. Základy jeho syntaxe může člověk, který umí programovat, zvládnout během několika hodin. Pak už se může soustředit na poznání a pochopení funkce klíčových knihoven a na jejich co nejefektivnější využití.

Jednoduchost jazyka však neznamená jeho omezenost. Jak jsem již řekl, Java je v současnosti nejpoužívanějším programovacím jazykem. Programují se v ní aplikace všech rozměrů od drobných programů pro čipové karty, přes programy pro mobilní telefony a nejrůznější zabudovaná zařízení, až po obří projekty rozprostřené na řadě vzájemně komunikujících počítačů.

Multiplatformní

Java se řadí mezi hybridní jazyky. To znamená, že je současně překládaná i interpretovaná. Programy v jazyku Java se nejprve přeloží do speciálního tvaru nazývaného *bajtkód*, který pak analyzuje a interpretuje speciální program nazývaný *virtuální stroj Javy* (Java Virtual Machine – používá se pro něj zkratka VM).

Virtuální stroj umožňuje, aby jeden a týž program běhal na různých počítačích a operačních systémech. Pro každou konfiguraci hardwaru a operačního systému lze definovat její vlastní virtuální stroj. Ten se pak stará o správný běh programů, takže se program (a s ním i uživatel) vůbec nemusí starat o to, na jakém hardwaru a operačním systému zrovna běží.

Java běhá pod systémy *Windows*, *Unix*, *Linux*, *MacOS*, *Solaris* a řadou dalších operačních systémů. Vytvoříte-li program v Javě, můžete jej spustit téměř na libovolném počítači. Jak mnozí z vás vědí, programy v Javě je možné spouštět i na řadě mobilních telefonů a dokonce ovládají i miliony čipových karet.

Java je jazyk i platforma

Jedním z klíčových záměrů autorů Javy bylo vytvořit nejenom jazyk, ale celou platformu . Tuto platformu realizuje výše zmíněný virtuální stroj spolu se základní knihovnou nejpoužívanějších funkcí.

Asi bych se zde měl zmínit o tom, že Java není jediná platforma, ale jsou to hned čtyři platformy:

- J2SE (Java 2 Standard Edition) označuje základní platformu určenou pro vývoj desktopových aplikací a jednodušších verzí serverových aplikací. Tuto edici budeme používat i my.
- J2EE (Java 2 Enterprise Edition) označuje nadstavbu nad J2SE obsahující další knihovny specializované pro tvorbu rozsáhlých distribuovaných aplikací.
- J2ME (Java 2 Micro Edition) označuje zjednodušenou verzi určenou pro vývoj aplikací pro malá zařízení, jejichž typickými představiteli jsou mobilní telefony.
- Java Card je ještě osekanější verze používaná při vývoji aplikací určených pro čipové karty. Tato platforma bývá někdy zařazována pod J2ME.

Vývojářská sada

Jako vývojáři musíte rozlišovat dvě verze programových balíků, které můžeme stáhnout:

- Jednodušší **JRE** (Java Runtime Environment – běhové prostředí Javy) poskytuje vše potřebné pro běh programů.
- Komplexnější **JDK** (Java Development Kit – sada pro vývoj v Javě) označovaná někdy také **SDK** (Software Development Kit – sada pro vývoj softwaru) je vlastně sada JRE doplněná o základní vývojové nástroje (překladač, generátor dokumentace, ladící program a další) a poskytuje tak vše potřebné pro vývoj programů.

Vy se chcete naučit pomocí této učebnice programovat, tak budete potřebovat mít instalované JDK (připomínám, že se musí jednat o verzi 5.0 či vyšší). Uživatelům, kteří pak budou spouštět vaše programy, stačí JRE. V dalším textu budu předpokládat, že máte JDK nainstalováno.

1.5 Vývojové prostředí BlueJ

Mnozí autoři učebnic vystačí při výkladu s nástroji z SDK doplněnými o nějaký editor, ve kterém píší zdrojové texty programů. Obdobným způsobem pracuje i část programátorů (podle některých průzkumů takto pracuje asi 7 % programátorů). Většina programátorů však dává přednost speciálním vývojovým prostředím označovaným souhrnně zkratkou **IDE** – Integrated Development Environment (integrované vývojové prostředí), což je sada programů, která nabízí řadu funkcí, jež vývoj programů výrazně zefektivňují.

IDE dáme přednost i my. V této učebnici budu používat vývojové prostředí *BlueJ* (čtěte blúdžej), které bylo vyvinuto speciálně pro výuku objektově orientovaného programování a oproti jiným IDE nabízí několik vlastností, jež se nám budou při výkladu základů OOP hodit.

- Je maximálně jednoduché, takže se začátečníci mohou soustředit na své programy a nejsou rozptylováni záplavou možností, kterými je zahrnují profesionální vývojová prostředí a které na počátku beztak neumí využít.
- Je názorné, protože slučuje možnosti klasického textového zápisu programů s možností definice jeho architektury v grafickém prostředí. Tato koncepce se stává základním postupem návrhu programů. *BlueJ* je schopno vytvořit na základě grafického návrhu kostru programu

a průběžně zanášet změny v programu do jeho grafické podoby a naopak změny v grafickém návrhu do textové podoby programu, aniž by ovlivnilo ty části programu, které programátor zadal „ručně“.

☞ Je interaktivní – umožňuje přímou práci s objekty. Ostatní prostředí vám většinou povolují pouze vytvořit program, který můžete spustit. Přímou komunikaci s jednotlivými součástmi programu, tj. vytváření objektů a zasílání zpráv řízené interaktivně uživatelem, však většinou neumožňují.

K používání tohoto prostředí ve vstupních kurzech programování se veřejně hlásí zhruba tisícovka univerzit a školicích středisek po celém světě. Jak se můžete přesvědčit na stránkách **http://www.bluej.org**, jeho výhodné vlastnosti přivedly řadu programátorů k tomu, že je začali používat i pro vývoj menších profesionálních aplikací. Tito programátoři oceňují zejména jeho malou paměťovou náročnost (*BlueJ* vystačí v nouzi se 64 MB RAM, zatímco požadavky běžných profesionálních vývojových prostředí jsou mnohonásobně větší) a originální, jinde nenabízené možnosti interaktivního vývoje.

V dalším textu budu předpokládat, že máte toto prostředí nainstalováno spolu s rozšířením, o němž jsem se zmiňoval v pasáži *Potřebné vybavení* na straně 27.

1.6 Projekty a BlueJ

V současných vývojových prostředích již nepracujeme s programy, ale s projekty. Projekt může obsahovat jeden program (tj. něco, co spustíme a ono to běží) nebo několik programů – to podle toho, jak se nám to zrovna hodí.

Typickým příkladem jednoduchého projektu sestávajícího z několika programů byl vítězný projekt mladší kategorie z programátorské soutěže BB2002[1]. Byl jím soubor deskových her, který obsahoval 5 programů: hry Dáma a Reversi vytvořené v programovacím jazyku Baltík, hru Piškvorky vytvořenou v jazyku Baltazar, hlavní program umožňující volbu hry vytvořený opět v jazyku Baltík a skript umožňující přepínání programů vytvořený v jazyku JavaScript.

Jednotlivé hry byly v případě potřeby spustitelné samostatně. Jejich sloučením do většího celku a doplněním o nadstavbové uživatelské rozhraní pro výběr hry vzniklo takové malé herní centrum, které mělo pro řadu uživatelů větší užitnou hodnotu, než sada samostatných, na sobě nezávislých her.

Umístění projektů na disku

Doporučuji vám, abyste si pro projekty zřídili novou složku. Do ní pak budete umisťovat jak svoje projekty, tak projekty, které budou součástí tohoto seriálu.

Jednou z možností je zřídit na datovém disku složku **Java** s podsložkami **Texty** a **Projekty**. Do složky **Texty** si můžete vložit texty týkající se programování a Javy, ve složce **Projekty** si pak zřiďte pro každý projekt novou složku, ve které budou umístěny všechny soubory, které k němu budou patřit.

Jakmile složku pro své projekty zřídíte, můžete do ní hned stáhnout projekty k této učebnici, zmiňované v pasáži *Potřebné vybavení* na straně 27.

[1] Program si můžete stáhnout na stránkách **www.sgp.cz**.

Windows a substituované disky

V operačním systému *Windows* můžete používat 26 logických disků – pro každé písmeno abecedy jeden. Většina uživatelů však používá pouze zlomek tohoto počtu. Disky A: a B: bývaly donedávna vyhrazeny pro mechaniky pružných disků, na disku C: má většina uživatelů operační systém. Pak na počítači nejdete ještě pár dalších jednotek vyhrazených pro další oddíly velkého disku, pro CD-ROM a případně ještě nějaké síťové disky či zařízení, která se vůči počítači tváří jako další disk (např. paměť Flash-RAM) a tím výčet končí. Průměrný uživatel tak má většinu písmen abecedy nevyužitých.

Operační systém umožňuje použít tyto názvy pro tzv. **substituované disky**, což jsou složky, které se rozhodnete vydávat za logický disk. Protože o této možnosti většina uživatelů neví a přitom je to funkce velice užitečná, ukážu vám, jak ji můžete využít.

Substituované disky se definují pomocí příkazu

```
SUBST název_disku substituovaná_složka
```

Nejjednodušší způsob, jak definovat ve *Windows* např. substituovaný disk J:, je vložit do složky, kterou budete chtít substituovat jako disk J:, dávkový soubor s příkazem k substituci. (Písmeno J se pro Javu hodí nejlépe, ale můžete si vybrat jakékoliv jiné.)

Pokud jste ještě nepracovali s dávkovými soubory, tak vězte, že to jsou obyčejné textové soubory, do nichž zapisujete příkazy pro operační systém. Jejich název může být libovolný, ale musí mít příponu bat (zkratka ze slova batch – dávka).

V dávkovém souboru budou následujícím příkazy (na velikosti písmen nezáleží):

```
SUBST J: /d
SUBST J: .
```

První příkaz má za úkol zrušit případnou doposud nastavenou substituci disku J: (není-li v daném okamžiku označený disk substituován, systém vypíše chybovou zprávu, ale jinak se nic nestane), druhý příkaz pak substituuje aktuální složku jako disk J:.

Soubor umístíte do složky, z níž budete chtít udělat substituovaný disk. Kdykoliv pak tento dávkový soubor spustíte, dávka substituuje složku, v níž je umístěna, jako příslušný disk. Dávka se přitom spouští obdobně jako aplikace – např. poklepáním na ikonu jejího souboru v *Průzkumníku*.

Budete-li chtít mít danou substituci nastavenu trvale, můžete umístit zástupce dávkového souboru do startovací nabídky do složky **Start → Programy → Po spuštění** a systém spustí příslušnou dávku po každém restartu počítače.

Po spuštění příslušného dávkového souboru (spouští se poklepáním jako jakýkoliv jiný program nebo skript) se rozšíří používané disky o právě substituovaný disk – v našem případě o disk J:.

Kdykoliv od této chvíle budete pracovat s diskem J:, budete ve skutečnosti pracovat s obsahem substituované složky. A naopak: cokoliv uděláte s obsahem substituované složky, uděláte zároveň s obsahem disku J:.

Abyste mohli složku substituovat jako nějaký disk, nesmí váš operační systém používat disk označený tímto písmenem pro nějaké hardwarové zařízení (např. pevný disk, CD-ROM, Flash-disk apod.). Písmeno může být použito nejvýše pro jiný substituovaný disk, protože tuto substituci můžete před nastavením nové substituce zrušit (pro tento případ je v dávkovém souboru první příkaz).

Substituované složky umožňují sjednotit prostředí několika počítačů. Tuto knihu např. připravuji na několika počítačích, přičemž každý z nich má jinak uspořádané složky (vadí mi to, ale nemohu s tím nic dělat). V každém z nich jsou ale definovány následující substituované disky:

☞ J: pro složku s vývojovými nástroji Javy,

☞ S: pro složku, ve které je text knihy,

☞ V: pro složku, ve které jsou programy pro knihu.

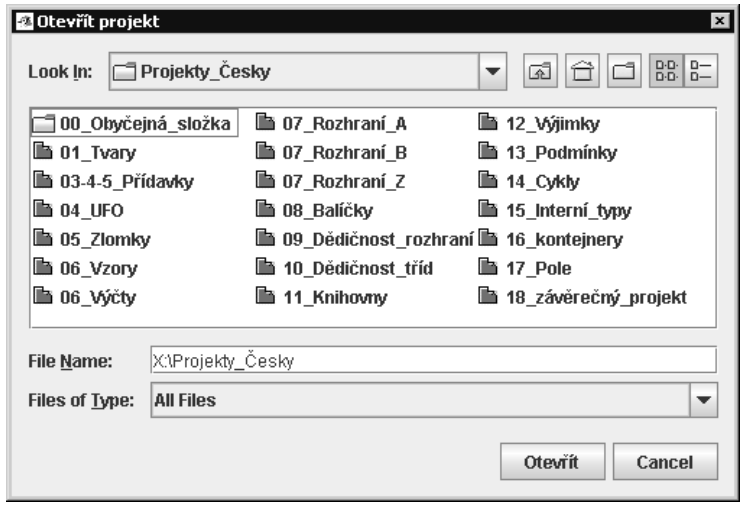
Po této úpravě mi již nutnost přecházení mezi různými počítači nevadí, protože vím, že na každém z nich najdu potřebné nástroje a dokumentaci na discích J:, S: a V:.

Používáte-li operační systém *Windows*, můžete urychlit budoucí vyhledání složky s projekty právě tím, že pro ni zřídíte zvláštní substituovaný disk. Kdykoliv se pak obrátíte na příslušný disk, obrátíte se ve skutečnosti k příslušné složce. Pomocí substituce si tak můžete zkrátit cestu k často používaným složkám.

Vyhledání a otevření projektu

V následujícím textu budeme pracovat s projektem **01_Tvary**, který je určen pro první seznámení s prostředím *BlueJ*, třídami a objekty a s nímž budeme pracovat celou příští kapitolu.

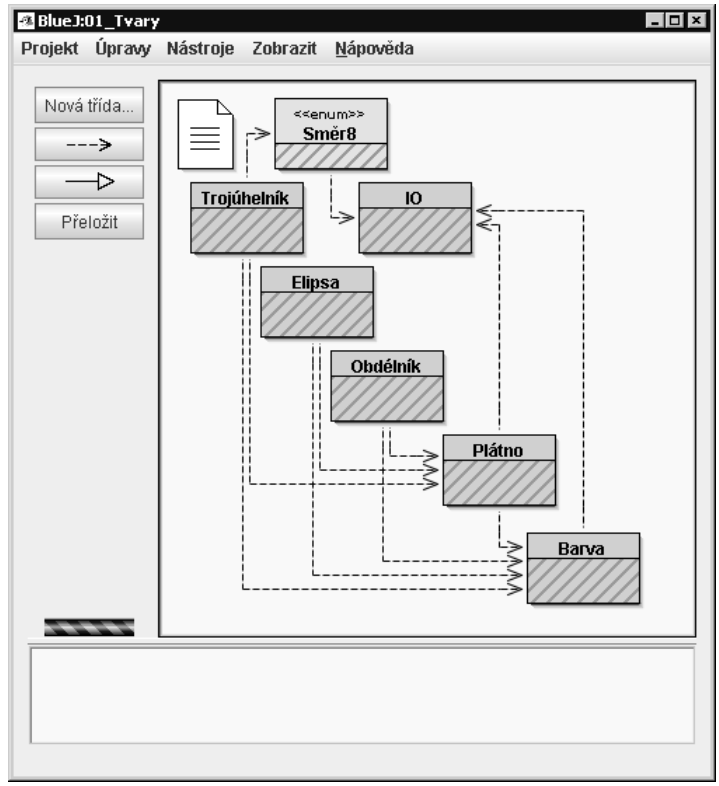
Předpokládám, že máte instalováno vše, o čem jsem se zmiňoval v úvodu v pasáži *Potřebné vybavení* na straně 27. Spusťte *BlueJ*, zadejte příkaz **Projekt → Otevřít**. Po zadání příkazu se otevře dialogové okno **Otevřít projekt** (viz obr. 1.1), v němž vyhledáte a označíte složku s projektem **01_Tvary** a stisknete tlačítko **Otevřít**. Po otevření projektu by okno *BlueJ* mělo vypadat obdobně jako na obrázku 1.2.



Obrázek 1.1
Otevření existujícího projektu

Na obrázku 1.1 si všimněte, že ikony projektů (přesněji ikony složek, v nichž je uložen projekt) vypadají jinak než ikony obyčejných složek. *BlueJ* totiž do složky, ve které budou soubory jeho projektu, přidá svůj vlastní soubor `bluej.pkg`, do nějž si ukládá informace o grafickém uspořádání objek-

tů v zobrazovaném diagramu. Podle přítomnosti tohoto souboru pak pozná, zda se jedná o složku s jeho projektem nebo o nějakou obyčejnou složku.



Obrázek 1.2
Okno BlueJ po otevření projektu 01_Tvary

1.7 Diagram tříd

Velký obdélník zabírající většinu okna projektu obsahuje tzv. **diagram tříd** našeho projektu. Diagram tříd popisuje strukturu našeho programu a vzájemné vazby mezi jeho částmi.

Malé obdélníky v diagramu tříd představují části programu, které nazýváme **třídy** (za chvíli si o nich budeme povídat podrobněji). Jsou znázorněny podle konvencí grafického jazyka UML[1] (vybarvení obdélníků do konvence nepatří, ale zvyšuje přehlednost a názornost diagramu). Čárkované šipky prozrazují, kdo koho používá – např. šipky vedoucí od tříd `Obdélník`, `Trojúhelník` a `Elipsa` ke třídě `Plátno` naznačují, že tyto třídy třídu `Plátno` používají (jak se vzápětí dozvíte, tyto obrazce se na plátno kreslí).

[1] UML je zkratkou z anglického Unified Modeling Language – sjednocený modelovací jazyk. Je to grafický jazyk, ve kterém programátoři navrhují své aplikace před tím, než začnou psát program. Více se o tomto jazyku můžete dozvědět např. v knize *Schmuller Joseph: Myslíme v jazyku UML*, Grada, 2001 (ISBN 80-247-0029-8).

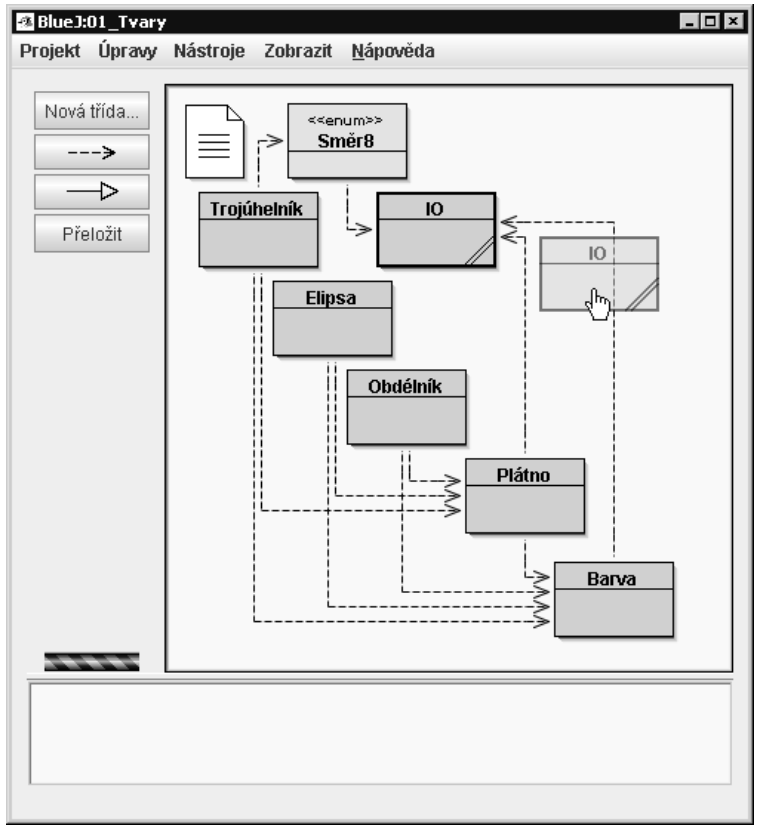
Bílý obrázek listu papíru v levém horním rohu diagramu představuje textový soubor se základním popisem celého projektu. Když na něj poklepete, otevře se okno editoru, v němž si budete moci tyto informace přečíst a v případě potřeby je i upravit či doplnit.

Šrafování spodní části obdélníků symbolizuje to, že třídy ještě nejsou přeloženy do bajtkódu. To lze snadno napravit: stiskněte tlačítko **Přeložit** v levé části aplikačního okna. Uvidíte, jak jedna třída za druhou ztmavne (tím prostředí naznačuje, že se překládá), aby pak opět zesvětlela a její šrafování zmizelo (již je přeložena). Od této chvíle můžete třídu používat.

Manipulace s třídami v diagramu

Polohu jednotlivých tříd v diagramu můžete měnit. Najedete-li ukazatelem myši na obdélník představující danou třídu, změní se podoba kurzoru ze šipky na ukazující ruku. V tomto okamžiku můžete obdélník uchopit (tj. stisknout a podržet primární [většinou levé] tlačítko myši) a přesunout jej do požadované pozice, kde jej upustíte (pustíte tlačítko myši).

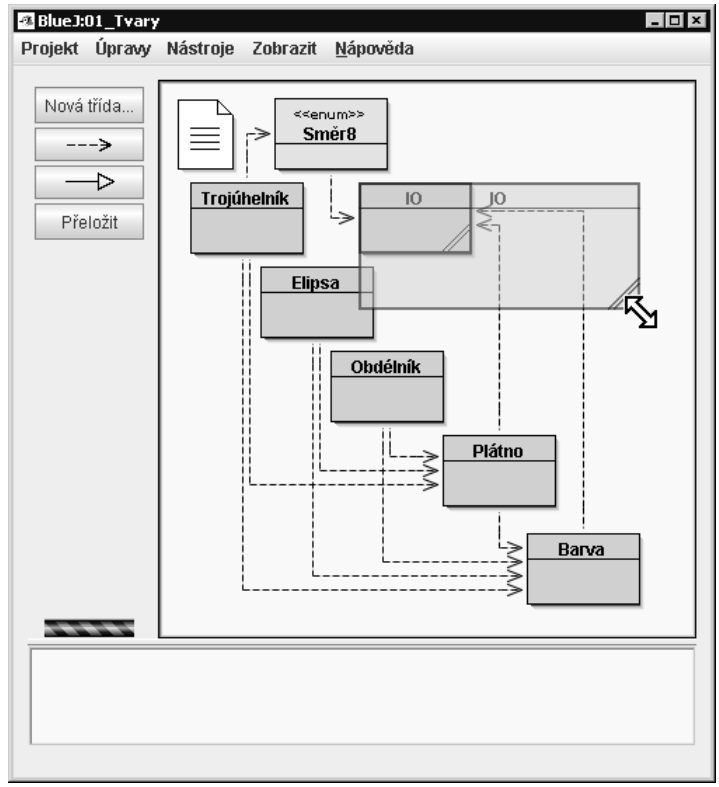
Po stisku tlačítka myši rám obdélníku ztuční. V průběhu přesunu se s ukazatelem myši bude přesouvat průsvitná kopie příslušného objektu, takže budete průběžně znát původní i nově nastavovanou pozici obdélníku (viz obrázek 1.3).



Obrázek 1.3
Posunutí ikony třídy v diagramu

Jsou-li zástupci tříd navzájem spojeni šipkami, bude *BlueJ* vaše přesuny sledovat a po umístění obrázku třídy do nové pozice šipky vždy příslušně překreslí. Můžete si tak diagram upravit do podoby, v níž vám bude připadat nejpřehlednější a nejnázornější.

Obdélníky zastupující třídy můžete nejenom přesouvat, ale můžete měnit i jejich rozměr. Všimněte si, že poté, co na obdélník klepnete, objeví se u pravého dolního rohu ztučnělého rámu dvojité přeškrtnutí. Najedete-li ukazatelem myši do oblasti označené tímto přeškrtnutím, změní se jeho podoba na dvojitou šikmou šipku. Nyní můžete uchopit roh obdélníku, přesunout jej do nové pozice a upravit tak velikost obdélníku (viz obrázek 1.4).

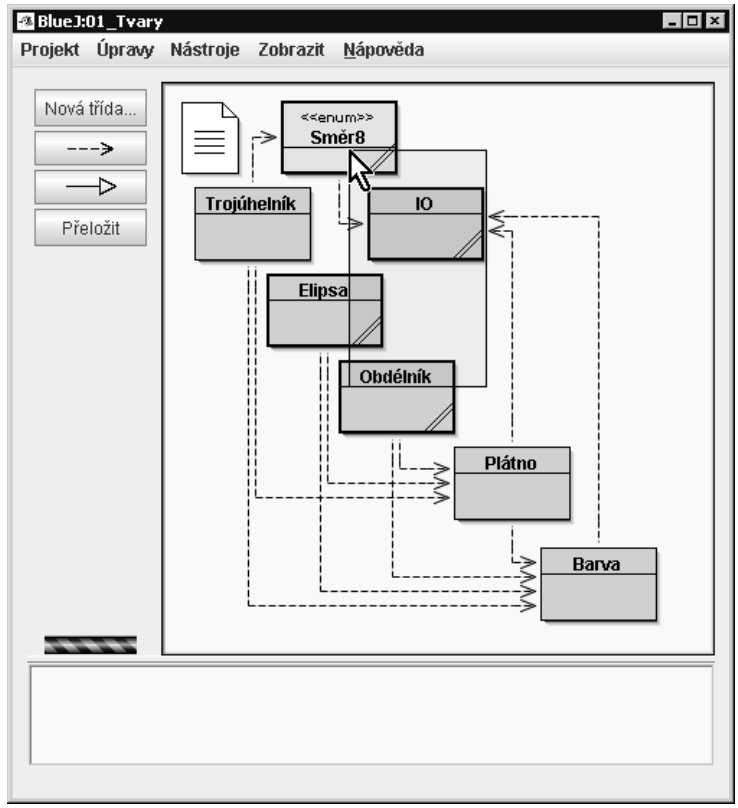


Obrázek 1.4
Změna velikosti ikony třídy v diagramu

Někdy se stane, že se nám projekt rozrůstá a rozrůstá a najednou bychom potřebovali posunout více tříd najednou, abychom udělali místo pro třídy, které se chystáme do projektu přidat. Postup je jednoduchý, ale nejprve se musíme naučit vybírat skupiny tříd.

Třídy jde zařazovat a vyřazovat ze skupiny vybraných tříd několika způsoby, které můžete kombinovat:

- Stisknete přeřaďovač CTRL nebo SHIFT (je to jedno) a klepete postupně na obdélníky, které chcete zahrnout od výběru.
- Najedete ukazatelem myši do volného prostoru a stisknete tlačítko myši. Při stisknutém tlačítku pak popojedete ukazatelem a za ním se roztáhne výběrový podšeděný obdélník. Která třída do něj padne alespoň svojí částí, ta se zařadí mezi vybrané a její okraj se zvýrazní (viz obr. 1.5).



Obrázek 1.5
Výběr skupiny tříd pomocí „výběrového obdélníku“

☞ Stisknete-li před předchozí operací přeřaďovač CTRL nebo SHIFT, budou se třídy „zasažené výběrovým obdélníkem“ přidávat ke třídám dříve vybraným.

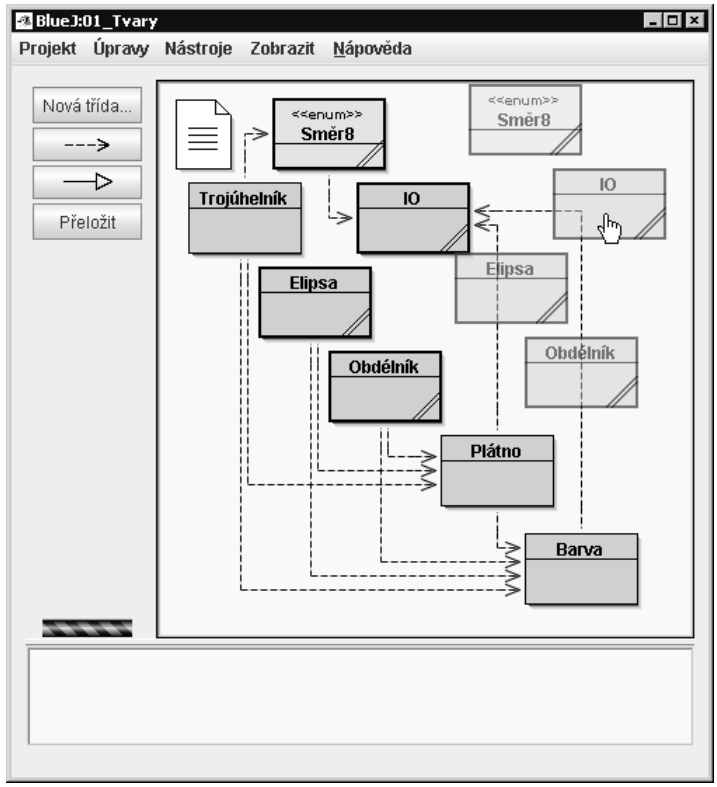
Někdy toho vyberete víc než potřebujete, a budete chtít některé třídy z výběru vyjmout. Pak máte dvě možnosti:

☞ Klepnutím myší do volného prostoru zrušíte výběr všech tříd.

☞ Klepnutím na třídu při stisknutém přeřaďovači CTRL nebo SHIFT změníte výběr dané třídy – vybranou „odvyberete“ a nevybranou vyberete.

Celou operaci přesunu skupiny tříd si můžeme vyzkoušet:

1. Roztáhněte trochu okno projektu, aby bylo kam třídy přesunout.
2. Vyberte třídy tak, jak je naznačeno na obr. 1.5.
3. Najeďte na některou z vybraných tříd tak, aby se ukazatel myši změnil na ukazující ruku (předpokládám standardní nastavení podoby ukazatelů).
4. Uchopte myší tuto třídy (a s ní i ostatní vybrané) a přesuňte je do požadované cílové pozice – viz obr. 1.6.
5. Uchopte znovu některou z vybraných tříd a pokuste se ji přesunout někam jinam (nejlépe do původní pozice). Ověřte si, že dokud výběr nezrušíte, jsou všechny třídy vybrány a přesouvají se jako celek.



Obrázek 1.6
Přesun bloku tříd

1.8 Shrnutí – co jsme se naučili

- Program je předpis zapsaný v nějakém programovacím jazyce a definující, jak má procesor, pro nějž je určen, splnit zadanou úlohu.
- U programů je nejdůležitější jejich spolehlivost a snadná udržovatelnost.
- Každý program je jakousi simulací reálného nebo virtuálního světa.
- Efektivita programování a robustnost vyvinutých programů je silně ovlivněna mírou abstrakce použité při jejich tvorbě. Čím může být naše vyjadřování blíže modelované skutečnosti, tím lépe se nám podaří napsat správný, spolehlivý a robustní program.
- Programy dělíme na překládané, interpretované a hybridní. Někdy se obdobné dělení používá i pro programovací jazyky.
- Překládané programy bývají rychlejší, interpretované zase bývají méně závislé na použité platformě.
- Platformou se nazývá kombinace procesor-operační systém. U interpretovaných jazyků však tuto platformu nahrazuje interpret, prostřednictvím nějž program komunikuje se systémem.

- ☞ Hybridní programy spojují výhody obou. Překládají se do mezijazyka optimalizovaného pro rychlou interpretaci. Ten je pak interpretován programem označovaným jako virtuální stroj.
- ☞ Moderní, objektově orientované jazyky se snaží zohlednit skutečnost, že jsou modelem světa tvořeného vzájemně interagujícími objekty.
- ☞ Java je moderní programovací jazyk, který je jednoduchý, objektově orientovaný a jeho programy jsou přenositelné mezi platformami.
- ☞ Javu zařazujeme mezi hybridní jazyky. Programy napsané v Javě se překládají do bajtkódu, který je pak interpretován virtuálním strojem.
- ☞ Java je nejenom jazyk, ale také platforma, přesněji čtyři platformy: J2SE, J2EE, J2ME a Java Card. Pro výuku se používá J2SE.
- ☞ Pro vývoj programů v Javě potřebujeme instalovat programovou sadu nazývanou SDK nebo JDK. Pro úspěšné spuštění programů v této učebnici potřebujete verzi 5.0 nebo vyšší (mladší).
- ☞ Při vývoji programů se většinou používají speciální vývojová prostředí označovaná zkratkou IDE.
- ☞ V této učebnici budeme používat vývojové prostředí *BlueJ*.
- ☞ Současná vývojová prostředí organizují vyvíjené programy do tzv. projektů.
- ☞ Při umísťování projektů na disku můžeme pod operačním systémem *Windows* s výhodou použít substituce.
- ☞ Strukturu programu můžeme znázornit v diagramu tříd.
- ☞ Při znázorňování struktury programu používáme grafický jazyk UML.
- ☞ Třídy jsou v diagramu tříd znázorněny rozdělenými obdélníky, v jejichž horní části je název třídy. Spodní část je v *BlueJ* prázdná.
- ☞ Prostředí *BlueJ* nám umožňuje měnit v diagramu tříd umístění i velikost obdélníků představujících jednotlivé třídy.
- ☞ Jsou-li mezi třídami natažené šipky, *BlueJ* je po změně umístění či velikosti obdélníků natáhne znovu, aby odpovídaly jejich novým pozicím a rozměrům.

Kapitola 2
Třídy a objekty v interaktivním režimu

Co se v kapitole naučíme
Tato kapitola předpokládá, že již máte otevřen projekt **01_Tvary**, o kterém jsme se bavili v minulé kapitole. Nejprve si povíme, co to vlastně ty třídy a objekty jsou, a seznámíme se s novým termínem *instance*. Pak se naučíme pracovat s třídami a objekty v prostředí *BlueJ*. Vysvětlíme si, jak se objekty vytvářejí, jak reagují na zprávy, co to jsou jejich atributy a že vedle zpráv posílaných instancím existují i zprávy posílané celé třídě. Naučíte se vytvářet instance tříd, posílat jim zprávy a prohlížet si hodnoty jejich atributů.

2.1 Nejprve trocha teorie

Než se vrhneme na vlastní programování, povíme si nejprve trochu o nejdůležitějších termínech, s nimiž se budeme v dalším výkladu setkávat. Abych vás tou teorií příliš neunavoval, tak tyto termíny opravdu jen zavedu a nebudu je nijak rozpitvávat – povím vám pouze to, co budete potřebovat vědět pro porozumění textu této a následující kapitoly. K řadě z nich se pak ještě vrátíme a vysvětlíme si je podrobněji i s ukázkami na příkladech.

Třídy a jejich instance

Jak jsem již naznačil, výchozím bodem celého objektově orientovaného programování je tvrzení, že svět je světem objektů, které si navzájem posílají zprávy. **Objekty**, s nimiž se ve svém okolí setkáváme, můžeme rozdělit do skupin, které mají nějaké společné vlastnosti a které v programování označujeme jako **třídy**. Vlastní objekty pak označujeme jako **instance** příslušné třídy – např. židle, na které sedíte, je instancí třídy židlí. Mohli bychom tedy říci, že instance (objekt) je nějakou konkrétní realizací obecného vzoru definovaného v příslušné třídě (židle, na které sedím, je konkrétní realizací obecné židle).

Pojem objekt a instance jsou ve skutečnosti synonyma, která můžete s klidným svědomím zaměnit. Termínu objekt se dává většinou přednost tehdy, hovoříme-li o obecných objektech, kdežto termínu instance dáváme přednost v situacích, kdy chceme zdůraznit, do jaké třídy objekty, o nichž hovoříme, patří.

Třídy popisují společné vlastnosti svých instancí a definují také nástroje pro jejich vytváření. Přirovnáme-li programování k hraní si na pískovišti, pak třída je něco jako formička a její instance jsou bábovičky, které za pomoci této formičky vytváříme. Při jiném přirovnání bychom mohli prohlásit, že třída je vlastně továrna na objekty – své instance.

Třída může mít obecně libovolný počet instancí. Existují však i třídy, které dovolí vytvoření pouze omezeného počtu instancí, někdy dokonce povolí jen jedinou instanci – jedináčka. Nevyskytují se příliš často, ale na druhou stranou nejsou žádnou exotickou výjimkou (s jednou se potkáme hned v prvním projektu).

Zprávy

Objekty si navzájem posílají zprávy, ve kterých se žádají o různé služby, přičemž onou službou může být často jen informace. Můžeme se např. židle zeptat, je-li čalouněná. V praxi bychom to realizovali nejspíše tak, že bychom k ní poslali upřený pohled, v programu k ní vyšleme zprávu.

V minulé kapitole jsme si řekli, že program je předpis popisující, jak splnit zadanou úlohu, a zapsaný v nějakém programovacím jazyce (dokud není zapsaný v programovacím jazyce, není to program, ale algoritmus). Tato definice je možná přesná, nicméně je tak obecná, že je pro naše účely prakticky nepoužitelná. Definujme si proto, že **objektově orientovaný program je v nějakém programovacím jazyce zapsaný popis použitých tříd, objektů a zpráv, které si tyto objekty posílají, doplněný u složitějších programů ještě o popis umístění programů na jednotlivých počítačích a jejich svěření do správy příslušných služebních programů** (např. operačních systémů či aplikačních serverů).

Budete-li slyšet programátory velkých aplikací hovořit o „deploymentu“, tak vězte, že hovoří o výše zmíněném rozmísťovacím aspektu svých programů.

Předchozí definici jsem uváděl proto, aby si ti, kteří mají nějaké zkušenosti s klasickým programováním, uvědomili, že u objektově orientovaného programování vstupují do trochu jiného programátorského vesmíru, než ve kterém se pohybují ti, kteří programují klasicky. Do vesmíru, v němž sice budou nadále používat mnohé z toho, co používá klasické, neobjektové programování, ale v němž se základní úvahy o koncepci a konstrukci programu ubírají naprosto jinými cestami, než na které byli doposud zvyklí.

Metody

Zde se objevuje drobný terminologický guláš. Java převzala terminologii jazyka C++ a nehovoří o posílání zpráv, s nímž přišli zakladatelé objektově orientovaného programování, ale o volání metod, které je bližší programátorům pracujícím v klasických jazycích.

Pokud jste již někdy programovali, tak pro ty z vás, kteří programovali v Baltíkovi, budou metody něco podobného jako Baltíkovi pomocníci, ti, kteří programovali v jiných jazycích, je mohou považovat ze ekvivalenty procedur a funkcí.

Metoda je část programu, kterou instance spustí v reakci na obdržení zprávy. Každá zpráva má v programu přiřazenu svoji vlastní metodu. Programátor v metodě definuje, jak bude objekt na příslušnou zprávu reagovat.

Svým způsobem je jedno, jestli řeknete, že instanci pošlete zprávu nebo že zavoláte její metodu. Termín *zpráva* budu proto používat spíš v souvislosti s popisem chování programu (tj. prakticky celou tuto kapitolu), termínu *metoda* pak budu dávat přednost ve chvíli, kdy budu hovořit o konkrétní části programu realizující odpověď na zaslání příslušné zprávy. Jak jsem ale řekl, oba termíny jsou si ekvivalentní.

2.2 Analogie

V dopisech čtenářů a v rozhovorech na různých internetových konferencích jsem se setkal se dvěma reakcemi čtenářů minulého vydání. První prohlašovali, že jim uváděné analogie výrazně pomohly pochopit vysvětlovaný problém, druzí naopak tvrdili, že jim tyto analogie pouze odváděly pozornost.

Nepatříte-li k těm, kteří mají rádi teoretické koncepty ilustrované analogiemi z reálnějšího světa a domníváte-li se naopak, že by taková analogie vše jen zatemnila, klidně tuto podkapitolu přeskočte a následující „analogické“ poznámky ignorujte.

Prozatím jsme se o třídách, objektech, zprávách a atributech bavili jako o něčem abstraktním, co se sice projevuje způsobem, který dokážeme vnímat, ale ne vždy jej dokážeme také pochopit. Zkusím vám proto vše připodobnit k něčemu hmatatelnému – třeba tato představa některým z vás pomůže při chápání některých konstrukcí, které budeme ve zbytku knihy vytvářet.

Představte si svůj projekt jako město robotů. Každá třída v něm vystupuje jako továrna vyrábějící roboty. Jejími instancemi jsou vozidla vybavená robotími osádkami – takovou robotí pracovní četou. Jedna továrna může vyrábět třeba vozidla s četou hasičů, druhá vozidla s četou opravářů, třetí vozidla s četou malířů.

Každá továrna vyrábí všechna vozidla stejná s naprosto stejnými osádkami. Každý člen osádky vozidla je specializovaný na jedinou funkci, kterou je reakce na konkrétní zprávu a představuje tak ekvivalent metody.

Vozidlo představující instanci je vybaveno radiostanicí, takže může z okolního světa přijímat zprávy. Když přijme zprávu, která vyžaduje splnění některého z úkolů, pro které bylo postaveno, pověří příslušného specializovaného robota (metodu), aby úkol splnil. Robot může při plnění úkolu posílat zprávy libovolné osádce (včetně té svojí), a požádat ji, aby vykonala nějakou akci potřebnou pro splnění úkolu.

2.3 Třídy a jejich instance

Vytváříme svou první instanci

V jazyku Java vytváříme instance tříd zasláním dvojice zpráv:

1. Nejprve pošleme virtuálnímu stroji zprávu sestavenou z klíčového slova `new`, za nímž uvedeme název třídy, jejíž instanci vytváříme. Zprávou `new` jej žádáme o vytvoření nové instance, které

spočívá především ve vyhrazení (alokaci) paměti pro vytvářený objekt. Název třídy následující za klíčovým slovem `new` pak předává informaci o tom, kolik místa bude třeba v paměti vyhradit.

2. Pouhé vyhrazení paměti k vytvoření instance nestačí. To je pouze nutnou podmínkou k tomu, aby bylo možno vytvářený objekt připravit – inicializovat. Po uvedeném vyhrazení paměti proto musí **vždy** následovat zaslání speciální (bezejmenné) zprávy žádající inicializaci připraveného objektu, která spočívá v uložení potřebných počátečních hodnot a spuštění potřebných podpůrných akcí.

   Reakci na tuto zprávu, tj. inicializaci vyhrazené paměti má na starosti speciální metoda označovaná jako **konstruktor**. Konstruktor obdrží od virtuálního stroje odkaz na právě alokovanou paměť a zařídí vše potřebné, aby vznikla plnohodnotná instance jeho třídy. Svoji činnost ukončí tím, že volajícímu programu vrátí odkaz (ten, který před chvílí obdržel od virtuálního stroje), prostřednictvím nějž se bude zbytek programu na nově vytvořenou instanci obracet.

   Protože činnost konstruktoru je pro vytvoření instance klíčová, nebudu se bránit obratu *konstruktor vytvoří instanci*. Berte to tak, že alokovaná paměť je pouze surovinou, z níž tvůrce – konstruktor teprve vytvoří výsledné „umělecké dílo" – objekt. Je to obdobné jako když hrnčíř vytvoří z dodané hlíny džbánek nebo když kovář udělá z dodaného železa pluh.

Program v Javě nikdy nepracuje s instancí přímo, ale vždy pouze prostřednictvím odkazu na ni. Použijeme-li naši analogii, pak bychom mohli říci, že nikdy nebudete mít v držení celou pracovní četu (= instanci), protože byste ji ani neunesli. Budete mít vždy pouze její telefonní číslo (= odkaz), na které jí můžete zavolat, kdykoliv od ní budete něco potřebovat (budete jí chtít poslat zprávu). Podrobnosti o tom, proč tomu tak je, se dozvíte v podkapitole *Instance versus odkaz* na straně 61.

Volání konstruktoru naznačujeme dvojicí závorek uvedených za názvem třídy následujícím za operátorem `new`. Protože každá instance musí být inicializovaná, musí být součástí každého vytváření nové instance i volání konstruktoru. Máme-li na inicializaci instance nějaké zvláštní požadavky, uvádíme je v těchto závorkách.

Vyzkoušíme si vše v praxi. Budeme pokračovat v práci na projektu **`01_Tvary`**, který jsme v minulé kapitole otevřeli. V tomto projektu je šest tříd:

- `Plátno` – Instance třídy `Plátno` představuje kreslící plátno, na kterém se budou zobrazovat námi vytvořené tvary. Tato instance je jedináček, tj. je jedinou instancí své třídy, která pak začala používat „antikoncepci", takže už nikdy žádnou jinou instanci mít nebude.

- `Obdélník`, `Trojúhelník` a `Elipsa` – Jejich instance představují objekty, které lze na plátno nakreslit a z nichž můžeme sestavovat složitější tvary.

- `Barva` – Tato třída má právě sedmnáct předem vytvořených instancí představujících osm základních barev (černou, modrou, červenou, fialovou, zelenou, azurovou, žlutou a bílou) a devět dalších doplňujících barev (krémovou, šedou, ocelovou, růžovou, hnědou, khaki, cihlovou a stříbrnou). Tyto barvy se používají pro vybarvení plátna a zobrazovaných tvarů. Pokud vám nabídka barev nevyhovuje, můžete vytvořit další a používat pak ty.

  Každá třída má definovanou svoji oblíbenou (implicitní) barvu. Časem si ukážeme, jak barvu instance ovlivnit.

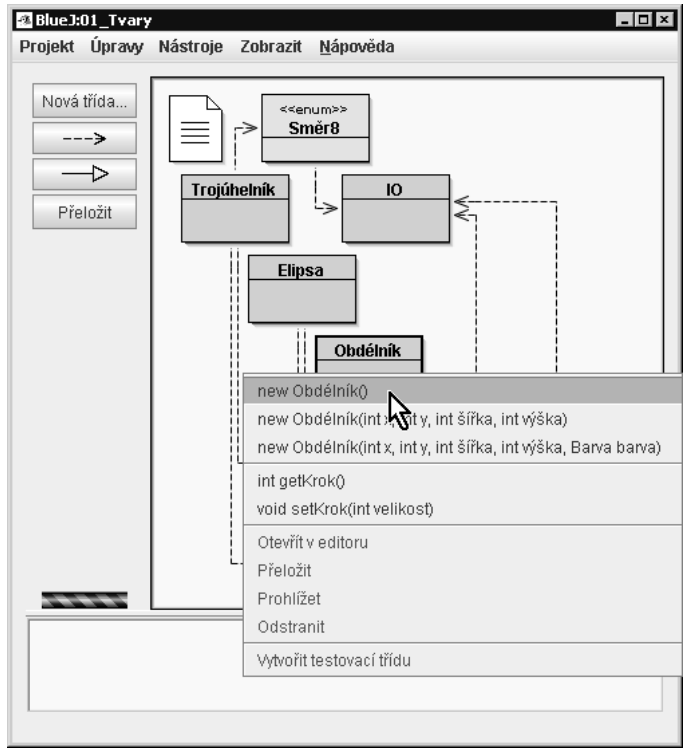
- `Směr8` - Instance třídy `Směr8` představují čtyři hlavní a čtyři vedlejší světové strany, tj. východ, severovýchod, sever, severozápad, západ, jihozápad, jih a jihovýchod. Pomocí těchto instancí je možno definovat směr, do kterého bude vytvářený trojúhelník natočen. Nezadáme-li směr

natočení, vytvoří se rovnoramenný trojúhelník s vrcholem otočeným na sever. Na rozdíl od barvy již žádný další směr vytvořit nemůžete.

Vyjmenované třídy jsou v diagramu tříd zobrazeny jako vodorovně rozdělené obdélníky, v jejichž horní části je zapsán název třídy. Jazyk UML, od jehož pravidel se způsob kreslení tříd v *BlueJ* odvozuje, sice definuje i obsah spodní části, ale autoři prostředí *BlueJ* vás nechtěli hned zpočátku zahrnovat záplavou informací a v zájmu maximální přehlednosti diagramu tříd ponechali spodní část obdélníku prázdnou.

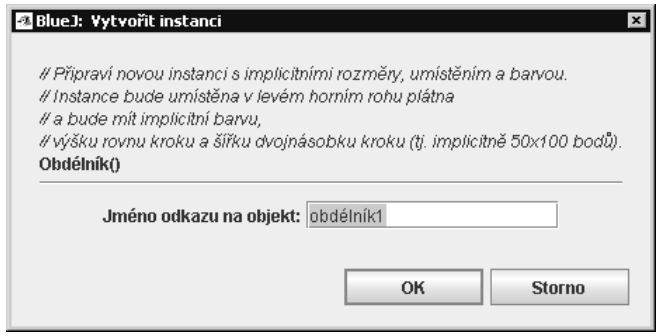
Šťouralové se asi hned zeptají, proč třída `Směr8` vypadá jinak než ostatní třídy. Prozatím jim prozradím, že je to proto, že definuje konečný počet předem známých instancí, k nimž už nemůžete žádnou přidat. Jinak je to stejná třída jako ostatní. Podrobněji si o tomto druhu tříd povíme v kapitole *Výčtové typy* na straně 237.

Zkuste nyní vytvořit např. instanci obdélníku. Klepněte v diagramu tříd pravým tlačítkem myši na obdélník představující třídu `Obdélník`. Rozbalí se místní nabídka, v jejíž horní části je seznam příkazů začínajících klíčovým slovem `new`, které slouží k vytvoření a inicializaci nových instancí. Klepněte na ten z příkazů pro vytvoření nové instance, který má za sebou prázdné závorky, takže inicializuje čerstvě vytvořenou instanci prostřednictvím konstruktoru, který neočekává žádné speciální požadavky a který proto označujeme jako **implicitní** (viz obr. 2.1).



Obrázek 2.1
Vytvoření instance třídy Obdélník

BlueJ otevře dialogové okno (viz obr. 2.2), v němž vás seznámí se základními informacemi o vyvolávaném konstruktoru a jím inicializované instanci. Zároveň se vás zeptá, jak budete chtít pojmenovat odkaz na vytvořený objekt, přičemž vám nabídne jméno odvozené od jména třídy, jejíž instanci vytváříte.



Obrázek 2.2
Dialogové okno po zavolání bezparametrického konstruktoru.

Pravidla pro tvorbu identifikátorů v jazyce Java

Jména odkazů, tříd a dalších pojmenovaných částí programu se označují termínem **identifikátory**, protože nám tyto části identifikují. Každý programovací jazyk definuje sadu pravidel, jimž musí jeho identifikátory vyhovovat. V jazyku Java musí identifikátory vyhovovat následujícím pravidlům:

- Smějí obsahovat pouze písmena, číslice a znaky „_“ (podtržítko) a „$“ (dolar). Za písmeno je přitom považován jakýkoliv znak, který sada UNICODE považuje za písmeno. Sem patří nejen písmena s diakritikou, ale také např. čínské či japonské znaky.
- Nesmějí začínat číslicí.
- Nesmějí být shodné s žádným klíčovým slovem[1], tj. s žádným ze slov:

```
abstract    assert      boolean     break       byte
case        catch       class       const       continue
default     do          double      else        enum
extends     final       finally     float       for
goto        char        if          implements  import
instanceof  int         interface   long        native
new         package     private     protected   public
return      short       static      strictfp    super
switch      synchronized this        throw       throws
transient   try         void        volatile    while
```

Délka identifikátorů (počet jejich znaků) není v jazyku Java omezena – to u řady jiných programovacích jazyků neplatí.

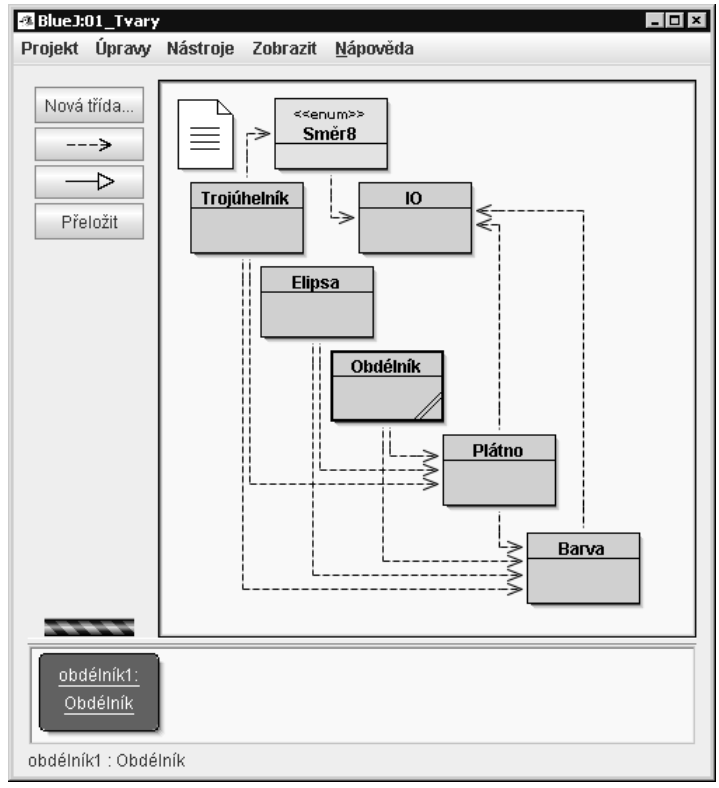
[1] Klíčová slova jsou identifikátory, které mají v jazyku předem daný speciální význam a nesmějí se proto použít na nic jiného.

Java patří mezi jazyky, kterým záleží na tom, zda napíšete identifikátor velkými či malými písmeny. Identifikátory něco, Něco a NĚCO jsou proto chápány jako tři různé identifikátory.

Jak jsem již řekl, podle definice jazyka patří mezi písmena všechny znaky včetně českých, ruských či japonských, nicméně některá vývojová prostředí se s neanglickými znaky příliš nekamarádí. Největší problémy dělají v názvech souborů. *BlueJ* jako výukové prostředí nemá s diakritikou problémy. Budete-li však chtít používat jiná prostředí, ověřte si proto tuto skutečnost před tím, než začnete identifikátory s diakritikou používat.

Vytváříme svou první instanci – pokračování

Máme tedy otevřené dialogové okno z obr. *2.2* na straně 56. Protože nemáme žádný pádný důvod definovat nějaký zvláštní vlastní název, můžeme akceptovat název, který nám program nabídne, tj. název `obdélník1`. Po potvrzení jména odkazu vytvoří *BlueJ* instanci třídy `Obdélník`, odkaz na ni umístí do **zásobníku odkazů**, který se nachází pod diagramem tříd a pojmenuje jej zadaným názvem (viz obr. 2.3).



Obrázek 2.3
Odkaz na první instanci v zásobníku odkazů