EDICE
myslíme v ...

knihovna programátora

- Podrobný výklad vlastností jazyka od naprostých základů až po pokročilé, běžně neprobírané konstrukce
- Vysvětluje nejenom jak probírané konstrukce používat, ale také proč jsou právě takové
- Využívá zabudované REPL prostředí pro demonstraci vykládaných konstrukcí bez zbytečného pomocného kódu
- Může sloužit současně jako učebnice i referenční příručka
- Ukazuje, jak efektivně experimentovat a využitím prostředí JShell získat okamžité odpovědi

RUDOLF PECINOVSKÝ

Java 9

Kompletní příručka jazyka

GRADA

EDICE
myslíme v ...

knihovna programátora

RUDOLF PECINOVSKÝ

Java 9

Kompletní příručka jazyka

GRADA
Publishing

Rudolf Pecinovský

Java 9
Kompletní příručka jazyka

Vydala Grada Publishing, a.s.
U Průhonu 22, Praha 7
obchod@grada.cz, www.grada.cz
tel.: +420 234 264 401, fax: +420 234 264 400
jako svou 6768. publikaci

Odpovědný redaktor: Jaroslava Palasová
Návrh vnitřního layoutu: Rudolf Pecinovský
Zlom: Rudolf Pecinovský
Počet stran 560
První vydání, Praha 2018
Vytiskly Tiskárny Havlíčkův Brod, a. s.

ISBN 978-80-271-0930-2 (pdf)
ISBN 978-80-271-0715-5 (print)

Všem, kteří se chtějí něco naučit

Stručný obsah

Podrobný obsah

Seznam výpisů programů

Seznam obrázků

Seznam odboček – podšeděných bloků

Úvod

Tato kniha seznamuje s devátou verzí jazyka *Java*. Soustředí se především na výklad vlastností jazyka a minimalizuje výklad probírající obsah standardní knihovny – tomu se budou věnovat další příručky z této série.

Komu je kniha určena

Kniha je určena všem, kteří se chtějí naučit jazyk *Java*. Nevyžaduje od čtenáře žádné předchozí znalosti programování a snaží se, byť stručně, vysvětlit vše, co je potřeba.

Naprostým začátečníkům bych sice doporučil, aby začali s některou z mých příruček úvodu do programování, v nichž nevysvětluji, jak se program zapisuje, ale soustředím se především na to, jak se vymýšlí. V těchto příručkách spolu navrhujeme architekturu programu. Zakódování navrženého programu (tj. jeho zápis v programovacím jazyce) přenecháváme generátoru kódu, který je integrální součástí použitého vývojového prostředí. K výkladu zápisu programu v kódu, na nějž se soustředí tato učebnice, pak přejdeme až v okamžiku, kdy složitost navrhovaných programů překročí možnosti onoho generátoru. V té době ale již mají účastníci kurzu základy objektově funkcionálního paradigmatu zažité a nedělají proto chyby, s nimiž zápasí studenti, kteří začali studiem syntaxe použitého programovacího jazyka.

Na druhou stranu nechci nikoho zrazovat, a proto jsem se v této knize pokusil důkladně vysvětlit i základní programové konstrukce a doporučené způsoby jejich využití, takže i naprostí začátečníci zde najdou všechny potřebné informace.

Jak už jsem řekl, kniha se soustředí na výklad jazyka. Tím, že výklad knihoven (např. práce s kolekcemi, datovody, soubory a datovými proudy, regulárními výrazy či vlákny) odložím do jiných příruček, ušetřím prostor, který mohu věnovat podrobnému výkladu těch vlastností jazyka, na které v jiných příručkách často nezbývá místo. Jejich neznalost ale vede k hodinám marného pátrání po skutečném původci vzniklé chyby a následným experimentálním změnám programu s myšlenkou: „Co kdyby to náhodou vyšlo?“

Koncepce výkladu

Značná část učebnic začíná definicí programu typu *Hello world*, což je koncepce převzatá z historické učebnice jazyka C vydané v roce 1978. Základní nevýhodou této koncepce je, že na počátku používá několik programových konstrukcí, které čtenáři vysvětlí až někde v průběhu dalšího výkladu.

Tehdy to dost dobře jinak ani nešlo. Od té doby se objevila řada nástrojů, které umožňují koncipovat výklad tak, aby se v něm používaly pouze konstrukce, které již byly vysvětleny. O to se pokouším i v této knize a používat doposud nevysvětlené konstrukce opravdu výjimečně pouze v situacích, kdy to výrazně zpřehlední výklad. Aby se mi to podařilo, používám k výkladu program *JShell*, který je standardní součástí vývojářské sady.

Knihu jsem se navíc pokusil napsat tak, aby mohla sloužit stejně dobře jako učebnice jazyka i jako referenční příručka, ve které by bylo možno v případě potřeby najít klíčové informace rychleji a snáze než na internetu. Pasáže vysvětlující jednotlivé programové konstrukce jsem se proto snažil jasně oddělit od pasáží probírajících teorii, jak řešit tu kterou třída problémů, a sloužících především začátečníkům, kteří ještě nemají s programováním žádné zkušenosti.

Rozdělení textu

Text je rozdělen do čtyř částí. První část začíná úvodem do prostředí *JShell*, které bude v celé první části využíváno ke spouštění demonstračních programů. Ve zbytku první části pak probírám algoritmické konstrukce a prostředky, které *Java* nabízí k jejich realizaci.

Druhá část se soustředí na základy objektově orientovaného programování. Začíná teoretickou kapitolou vysvětlující hlavní zásady objektově orientovaného paradigmatu. Po ní následují kapitoly postupně probírající základní objektové konstrukce a jejich účel.

Třetí část je věnována výkladu nadstavbových objektových konstrukcí včetně rozboru některých základních pravidel, jejichž nedostatečné pochopení dělá studentům často problémy. Závěr třetí části je věnován koncepci modularity, která je novinkou deváté verze. Modularita sice není součástí jazyka, ale platformy; nicméně její osvojení přináší výrazné zvýšení spolehlivosti vyvíjených programů a do jisté míry i efektivity jejich vývoje i následného provozu.

Závěrečná část je věnována přílohám. První vás seznámí s omezeními jazyka *Java*, druhá vám představí klíčové datové typy knihovny, kterou začneme zhruba od poloviny knihy používat.

Terminologie

Při zavádění české terminologie se bohužel potýkáme často s tím, že v anglické literatuře zavádějí mnozí autoři terminologii vlastní. V oblasti, kterou se má zabývat tato publikace, je terminologický guláš obzvlášť vydatný. Budu-li proto uvádět ve svém výkladu anglické ekvivalenty zaváděných termínů, budu používat terminologii zavedenou v oficiální referenční publikaci *Java® Language Specification – Java SE 9 Edition* ([3]), na níž se budu občas odvolávat zkratkou JLS.

Použité nástroje

Pro úspěšné studium knihy budete potřebovat několik programů, které musíte nejprve stáhnout a instalovat.

Vývojová sada JDK 9

Především budete potřebovat mít instalovanou vývojovou sadu pro *Javu 9* označovanou JDK, což je zkratka z anglického *Java Development Kit*. Stáhnete ji ze stránky http://www.oracle.com/technetwork/java/javase/downloads/. Jenom musíte před vlastním stažením nastavit přepínač u seznamu stáhnutelných souborů do polohy **Accept License Agreement**, protože jinak vás stránka daný program stáhnout nenechá.

Spolu s vývojovou sadou vřele doporučuji stáhnout i dokumentaci. Tu najdete na téže stránce, jenom musíte popojet trochu níže do sekce **Additional Resources**, ve které najdete podsekci **Java SE 9 Documentation**.

Vývojové prostředí *JShell*

Pro většinu výkladu budu využívat prostředí *JShell*, které je součástí JDK a se kterým vás seznámí úvodní kapitola. (Podrobnější seznámení s tímto prostředím najdete v příručce *Java 9 – JShell*.) Výhodou tohoto přístupu je, že součástí programů nemusí být nejrůznější vata, bez které byste standardní program v *Javě* nespustili. Navíc již nemusíte instalovat žádné další vývojové prostředí, takže nebudete muset zápasit s důsledky známé skutečnosti, že zvládnutí současných profesionálních vývojových prostředí je náročnější než zvládnutí základů jazyka.

V závěru knihy budu potřebovat vysvětlit některé konstrukce, které se v prostředí *JShell* dost dobře vysvětlit nedají. Až na to dojde, tak vás upozorním.

Samostatné vývojové prostředí

Od kapitoly *12 Vývojová prostředí a vytvoření aplikace* na straně 248 začnu vedle prostředí *JShell* používat i prostředí *NetBeans*.Budu v něm demonstrovat vlastnosti, které se projeví až u samostatně vyvíjených programů. Nelpím přitom na tom, abyste používali právě toto prostředí, ale doporučuji je těm, kteří ještě nemají s profesionálními vývojovými prostředími žádné zkušenosti.

Každopádně budete-li chtít experimentovat s profesionálním prostředím hned od počátku, můžete si je instalovat. Adresy, z nichž si můžete stáhnout některé z doporučovaných prostředí, najdete na počátku dané kapitoly.

Doprovodné programy

Všechny doprovodné programy zmiňované a používané v textu najdete na stránce knihy na adrese http://knihy.pecinovsky.cz/57_j9lang. Zde je umístěn generátor, který vám požadované programy dodá.

Doprovodné programy jsou zpočátku skripty prostředí *JShell*, v druhé části to pak jsou projekty pro vývojové prostředí *NetBeans*, které lze snadno transformovat na projekty pro vaše oblíbené vývojové prostředí.

Knihu jsem se snažil zalomit tak, aby všechny výpisy programů a komunikace s počítačem, které se vejdou na jednu stránku, byly vždy na jedné stránce a nelámaly se přes hranu stránky. Čtenářům papírové verze by se tak měly minimalizovat problémy při studiu doprovodných textů diskutujících obsah těchto výpisů.

Čtenáři elektronické verze jsou na tom s listováním hůře. Těm bych doporučil, aby si text knihy otevřeli dvakrát a umístili oba dokumenty vedle sebe. (Na většině současných počítačů s širokými monitory by to neměl být problém.) Pak lze mít v jednom dokumentu otevřenou stránku s výpisem a ve druhém dokumentu stránku s rozborem obsahu daného výpisu.

Problémy s klávesnicí

Až do verze *Java* SE 9.0.1 se prostředí *JShell* nekamarádilo se standardní českou klávesnicí, přesněji neumožňovalo zadat znak &. Obávám se, že tento stav bude trvalý, a to proto, že tento znak se na standardní české klávesnici zadává prostřednictvím klávesové zkratky ALTGR+C, která je ekvivalentní s klávesovou zkratkou ALT+CTRL+C, a ta je zřejmě (z hlediska *JShell*) příliš podobná zkratce CTRL+C.

Tento problém nastává naštěstí pouze při používání standardního rozložení. Pokud je ale používáte, tak je jedno, jestli se pokoušíte zadat daný znak z klávesnice, anebo jestli jej zadáváte prostřednictvím schránky „oblíbenou" kombinací CTRL+C, CTRL+V. Existuje několik způsobů řešení tohoto problému:

- Nejhorším řešením je přepínání mezi českým a americkým rozložením kláves. Pak se vám totiž pod rukama střídají dvě klávesnice, a to vede k jisté klávesnicové schizofrenii.
- Trváte-li na standardním rozložení kláves, můžete tento znak zadat prostřednictvím numerické klávesnice zadáním ALT+038.

Historie rozložení *České (QWERTY)*

Před rokem 1989 jsem pracoval v *Tesle Elstroj*, kde se vyvíjel počítač SAPI 86, jenž se měl stát českým ekvivalentem počítačů PC. Hrubě se nám ale nelíbil návrh rozložení kláves, který pro češtinu navrhl v roce 1986 *Microsoft*. Byl jsem proto pověřen navrhnout alternativní rozložení, které by více odpovídalo potřebám našich uživatelů.

Můj návrh vycházel z klávesnice psacího stroje, která byla nedotknutelná. Znaky, které na psacím stroji nebyly, jsem umístil na stejná místa, na nichž se nacházely na US klávesnici, jenom je bylo třeba vyvolat stisknutím přeřaďovače ALTGR, resp. jemu ekvivalentní kombinaci přeřaďovačů ALT+CTRL. Tím se na klávesnicích s US potiskem, kterých byla tehdy většina, minimalizovaly problémy s hledáním nestandardních znaků. Toto rozložení se mělo stát standardem pro naši republiku. *Tesla Jihlava* již začala tuto klávesnici vyrábět (ještě ji mám schovanou 😊).

Pak ale přišla revoluce a vše se změnilo. Naše řešení byla zavržena a byla dána přednost řešením z USA. Jenomže pak se firma *Apple* rozhodla lokalizovat svůj operační systém pro českou republiku a k mému příjemnému překvapení zvolila pro klávesnici mnou navržený layout.

Když později *Microsoft* lokalizoval *Windows NT*, členy lokalizačního týmu se stalo i několik uživatelů počítačů *Apple*, kteří si na moji klávesnici zvykli, a tak ji do české lokalizace *Windows* propašovali. Protože je však rozčilovalo stěhování znaků Y a Z při přecházení mezi počítači s českou a anglickou lokalizací, tak umístění těchto znaků prohodili, aby odpovídalo anglické klávesnici. Tak vzniklo rozložení označované jako *České (QWERTY)*.

Možnost volby tohoto rozložení kláves se od té doby stalo standardní součástí operačního systému *Windows*. Jistě chápete, jak mne těší, že se mi podařilo zasít do systémů *Windows* a *MacOS* své semínko.

- Kompromis mezi střídáním klávesnic a standardním českým rozložením představuje rozložení, které je k dispozici pod názvem *České (QWERTY)*, o němž se můžete podrobněji dozvědět v podšeděném rámečku *Historie rozložení České (QWERTY)*.

Syntaktické definice a diagramy

V testu je uváděna řada nejrůznějších programových konstrukcí, jejichž zápis se řídí přesně danými syntaktickými pravidly. Většina učebnic možné způsoby zápisu většinou formálně nedefinuje, anebo používá syntaktické definice odvozené z Backusovy normální formy. Syntaktické definice jsou výhodné pro strojové zpracování, lidé se v nich často špatně orientují. Rozhodl jsem se proto zobrazovat syntaktická pravidla zápisu jednotlivých konstrukcí pomocí syntaktických diagramů.

Syntaktický diagram ukazuje, jak je možno zobrazovanou konstrukci zapsat. Pojedete-li po čarách, tak jakýkoliv průjezd generuje syntakticky správnou konstrukci. Toho, kdo syntaktické diagramy ještě nezná a chtěl by rychle některý vidět, bych odkázal např. na diagram na obrázku 2.1 na straně 64.

Použité typografické 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.

Termíny První výskyt nějakého termínu a další texty, které chci zvýraznit, vysazuji **tučně**.

Název Názvy firem a jejích produktů vysazuji *kurzivou*. Kurzivou vysazuji také názvy kapitol, podkapitol a oddílů, na které se v textu odkazuji.

Odkaz Celá kniha je prošpikovaná křížovými odkazy na související pasáže. Není-li odkazovaný objekt (kapitola, obrázek, výpis programu, …) na některé ze sousedních stránek, je pro čtenáře tištěné verze doplněn o číslo stránky, na níž se nachází. Čtenářům elektronické verze stačí, když na něj klepnou, a použitý prohlížeč by je měl na odkazovaný objekt ihned přenést.

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`	Identifikátory a další části programů zmíněné v běžném textu vysazuji `neproporcionálním písmem`, které je v elektronických verzích pro zvýraznění tmavě červené.
`metoda(?)`	Při odkazech na metody budu v závorkách za názvem metody vždy uvádět seznam typů jejich parametrů – např. `equals(Object)`. Nebude-li v danou chvíli jasné, jaké má zmiňovaná metoda parametry, budu do závorek psát otazník – např. `metoda(?)`.
`zadání`	Ve výpisech komunikace s prostředím *JShell* bude zadání uživatele vysazeno tučně (v elektronických verzích pro zvýraznění tmavě červeně), a odpovědi prostředí netučně (a černě).

Kromě výše zmíněných 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 jing-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 aplikující probranou látku. Příslušný projekt získáte pomocí generátoru projektů popsaného výše.

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.

Odbočka – podšeděný blok

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.

Zpětná vazba

Přes veškeré úsilí, které jsme knize já i moji spolupracovníci věnovali, nemohu vyloučit, že v textu či doprovodných příkladech zůstaly skryté nějaké chyby. Předem se za ně omlouvám. Objevíte-li proto v knize nějakou chybu nebo budete mít návrh na nějaké její vylepšení, neostýchejte se napsat na adresu rudolf@pecinovsky.cz. Pokusím se na stránku knihy http://knihy.pecinovsky.cz/57_j9lang co nejdříve zanést příslušná errata s opravou, kterou pak zapracujeme do případného dalšího vydání.

Pokud vám bude někde připadat text nepříliš srozumitelný nebo budete mít nějaký dotaz, ať už k vykládané látce či použitému vývojovému prostředí, pošlete mail s předmětem 57_Java9-UcRef_DOTAZ. Bude-li se dotaz týkat něčeho obecnějšího, zveřejním na stránce knihy odpověď i pro ty ostatní, které by mohl obdobný dotaz napadnout za pár dní, anebo jsou natolik ostýchaví, že se netroufnou sami zeptat.

Část I: Neobjektové konstrukce

Tato část vás nejprve seznámí s prostředím *JShell*, které budu ve svém příkladu využívat k tomu, abychom získali okamžitou odpověď na zadané programové obraty. Budeme tak moci od začátku zkoušet probírané konstrukce, aniž bychom museli používat něco, co jsme ještě neprobrali. Poté postupně probereme algoritmické konstrukce, které vyšší programovací jazyky nabízely ještě před příchodem objektového paradigmatu, kterému bude věnována druhá část.

Kapitola 1
Prostředí JShell

Co se v kapitole naučíte

Tato kapitola vás seznámí s prostředím *JShell*, které bude v následujících kapitolách používáno pro demonstraci základních programových konstrukcí jazyka a jeho pravidel. V závěru pak ještě vysvětlí některá nejzákladnější syntaktická pravidla jazyka a seznámí vás s komentáři, kterými budou doprovázeny uložené výpisy programů.

vy

Tato kapitola bude obsahovat jen velmi stručný úvod, jenž vás seznámí s klíčovými vlastnostmi prostředí *JShell*, které budu v následujícím výkladu používat. Jak už bylo řečeno v úvodu, podrobné informace o tomto prostředí včetně návodu na jeho začlenění do vlastního programu najdete v publikaci *Java 9 – JShell*.

1.1 Charakteristika programu a prostředí *JShell*

Program *JShell* zařazujeme do kategorie nástrojů typu REPL, což je zkratka z anglického *read-evaluate-print-loop* (česky: *cyklus přečti-vyhodnoť-vytiskni*). Tyto nástroje se používají jako poskytovatelé prostředí (platformy) pro rychlou interakci s podkladovým prostředím, kterým může být jak operační systém, tak nějaký program. Uživatel platformy typu REPL může komunikovat s podkladovým prostředím v programovacím jazyce dané platformy bez potřeby překladu a samostatného spuštění.

1.2 Příprava programu *JShell* a první spuštění

Zdánlivě nejjednodušší je otevřít okno konzoly (terminálu) a přesunout se v něm do podsložky bin složky, do které jste instalovali JDK. Tam najdete soubor jshell.exe, který můžete spustit příkazem

```
jshell -v
```

kde parametrem -v spouštěný program žádáme, aby nám o všech provedených činnostech referoval co nejpodrobněji.

Zabíhání do této složky však většinu uživatelů obtěžuje. Chcete-li proto program používat opravdu jednoduše, pak se nabízejí dvě cesty:

- Upravit systémovou proměnnou PATH definující seznam složek, které systém při hledání zadaného souboru prohledá.
- Definovat jednoduchý skript, který program *JShell* spustí v aktuální složce a případně mu i zadá potřebné parametry.

Pro tento kurz dám přednost druhé cestě. Jak znám uživatele systému *Linux* a jeho verzí (sem patří i MacOS), tak ti dávkové soubory hojně používají a žádný výklad nepotřebují. Proberu proto tvorbu dávkových souborů pouze pro *Windows*, jejichž uživatelé o této možnosti často ani nevědí.

Dávkové soubory pro *Windows*

Windows používají dva druhy dávkových souborů: z dob systému DOS převzaly soubory s příponou BAT (zkratka z anglického *batch* – dávka, skupina, …) a od verze *Windows* XP přidaly ještě soubory s příponou PS1 (PS je zkratka z názvu programu *PowerShell*, ale co tam dělá ta jednička, netuším).

Obě verze se spouští v konzolovém okně. Jeho velikost, umístění, použité písmo a barvy si můžete nastavit ve vlastnostech dosažitelných ze systémové nabídky okna[1]. My budeme používat první z nich, tj. soubor s příponou BAT. Postupujte následovně:

1. Otevřete složku, do které jste prostřednictvím generátoru doprovodných programů uložili soubor Start.jsh.
2. Ve svém oblíbeném textovém editoru[2] vytvořte prázdný textový soubor.
3. Do souboru napište příkaz

```
chcp 1250
```

[1] Systémovou nabídku aktivujete klepnutím na ikonu na levém okraji titulkové lišty okna.

[2] Já používám program PAPad, který můžete stáhnout z adresy http://pspad.com v některé z osmi lokalizací. Dáváte-li ale přesnost jinému, nebudu vám bránit.

který nastaví správnou kódovou stránku. Česká a slovenská Windows totiž implicitně používají stránku 852, ale spuštěnému programu tvrdí, že používají stránku 1250. Proto je třeba na tuto kódovou stránku nastavit i příkazový panel.

4. Na další řádek napište příkaz

```
<JDK>\bin\jshell -v
```

kde `<JDK>` zastupuje cestu ke složce, do které jste instalovali JDK *Javy*. Tím spustíte program *JShell* a jím definovanou platformu, kterou budeme v učebnici používat.

5. Na další řádek napište příkaz

```
pause
```

Ten zabezpečí, že po opuštění platformy *JShell* zůstane příkazový panel otevřen, abyste si mohli před jeho ukončením prohlédnout případné závěrečné informace.

6. Soubor uložte pod názvem `J6_JShell.bat`. Vlastní jméno souboru není důležité – doporučuji jej pouze proto, abych se na ně mohl v dalším textu odvolávat. Důležitá je přípona `bat`.

Řada začínajících programátorů ponechává nastavení *Windows* pro laické uživatele, které matou přípony souborů. Proto jsou přípony známých typů implicitně skryty. Necháváte-li ale operační systém, aby před vámi skrýval přípony, tak se stává, že vám sice průzkumník ukazuje, že se ve složce nachází soubor `J9_JShell.bat`, ale ve skutečnosti je tam uložen soubor `J6_JShell.bat.txt`. Přípona `txt` je pro operační systém známá, a proto ji nezobrazuje.

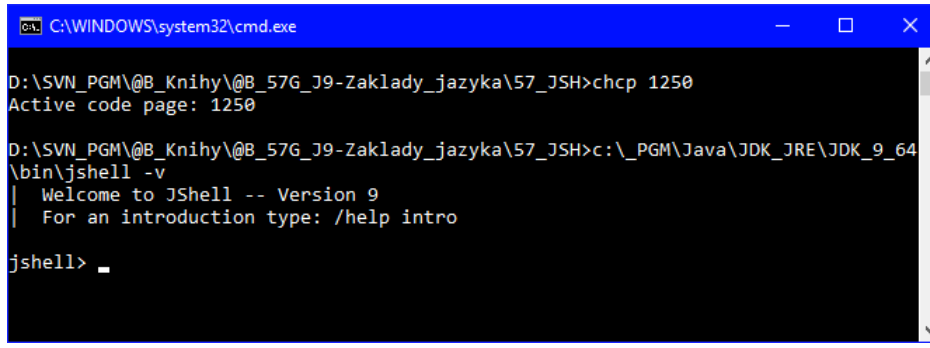
Po spuštění

Po spuštění programu se otevře konzolové okno podobné oknu na obrázku 1.1. Lišit se budou pouze v cestách k aktuální složce vypisovaných na počátku prvního a čtvrtého řádku a v cestě k programu `jshell.exe` ve čtvrtém řádku. Uživatelům systémů *Linux* a *MacOS* budou chybět první tři řádky (nastavení kódové stránky, oznámení nastavené stránky a oddělující prázdný řádek).

Pravda, já mám na svém počítači doplněnou systémovou proměnnou `PATH` o cestu k programu *JShell*[3], takže jsem ve čtvrtém řádku celou cestu k programu

[3] Návod, jak se to dělá, určený pro ty, kteří nemají s nastavováním této proměnné žádné zkušenosti, je v příloze výše zmíněné příručky *Java 9 – JShell*.

`jshell.exe` uvádět nemusel. Navíc se pak celý příkaz nevešel na řádek, takže pokračuje na dalším řádku okna, ale logicky je to stále jeden řádek. Chtěl jsem ale hlavně pro ty, kteří s konzolovým oknem ještě nemají velké zkušenosti, ukázat výpis kompletní.



Obrázek 1.1:
Konzolové okno otevřené po spuštění dávky `J9_JShell.bat` na mém počítači

V dalším textu již nebudu ukazovat výpisy z okna ve formě obrázků, ale ve formě klasických výpisů programů s číslovanými řádky, abych se mohl na čísla řádků snáze odvolávat. Obsah okna na obrázku 1.1 je ve výpisu 1.1.

Výpis 1.1: *Obsah konzolového okna otevřeného po spuštění dávky `J9_JShell.bat` na mém počítači*

```
1 D:\SVN_PGM\@B_Knihy\@B_57G_J9-Zaklady_jazyka\57_JSH>chcp 1250
2 Active code page: 1250
3
4 D:\SVN_PGM\@B_Knihy\@B_57G_J9-Zaklady_jazyka\57_JSH>c:\_PGM\Java\JDK_JRE\JDK_9_64
  \bin\jshell -v
5 |  Welcome to JShell -- Version 9
6 |  For an introduction type: /help intro
7
8 jshell>
```

Na posledním řádku výpisu je již výzva programu/prostředí *JShell*. Za ní můžete začít psát své úryvky a příkazy.

Nelekněte se, když se program *JShell* nespustí hned, přesněji když hned nevypíše své přivítání. Je to proto, že se na počátku načítá startovní skript a podle něj se prostředí konfiguruje. Přivítání se vypisuje až poté, co se startovní skript načte a provede.

1.3 Úryvky (snippets)

Výrazy (expressions), příkazy (statements), deklarace (declarations) a definice (definitions) jazyka *Java*, které za tuto výzvu napíšete, se hromadně označují jako **úryvky** (snippets). Kdykoliv napíšete nějaký úryvek, prostředí *JShell* jej ihned vyhodnotí a uloží výsledek. Ve výpisu 1.2 najdete zadání dále popsaných úryvků spolu s odpověďmi prostředí.

Zkuste napsat jednoduchý aritmetický výraz, např. `3+5` (řádek 1 výpisu) a potvrdit jej stiskem `<ENTER>`. Jak už jsem řekl, prostředí *JShell* výraz spočítá a na dalším řádku oznámí jak výsledek, tak to, že jej uložilo do pomocné proměnné nazvané `$1`.

Úplným začátečníkům prozradím, že **proměnná** je místo v paměti, kam si můžeme uložit nějakou hodnotu pro budoucí použití. Proměnných může být v programu více, a proto má každá přidělené svoje jméno. Když pak později chceme uloženou hodnotu využít, odvoláme se na ni jménem příslušné proměnné.

Prostředí *JShell* nám navíc na řádku 3 přidalo informaci, že vytvořená pomocná proměnná je typu `int`, což je zkratka z anglického *integer* a prozrazuje to, že obsah oné proměnné bude v programu vždy interpretován jako celé číslo.

Úryvky se nemusejí nutně vejít na jeden řádek. Když prostředí vyhodnotí, že zadávání úryvku ještě neskončilo (ve výpisu 1.2 řádky 5 a 6), zahájí další řádek pokračovací výzvou. Vyhodnocení zadaného výrazu zahájí až v okamžiku, kdy je zadání zkompletováno, k čemuž dojde na řádku 7, takže se na řádcích 8 a 9 dozvíme výsledek.

Výpis 1.2: *První tři pokusné úryvky*

```
 1 jshell> 6+5
 2 $1 ==> 11
 3 |  created scratch variable $1 : int
 4
 5 jshell> 6 +
 6    ...> 7+
 7    ...> 9
 8 $2 ==> 22
 9 |  created scratch variable $2 : int
10
11 jshell> $1+$2
12 $3 ==> 33
13 |  created scratch variable $3 : int
14
15 jshell>
```

Použití proměnných

Jak jste si jistě všimli, na řádku 3 jsem místo čísel použil názvy dříve vytvořených proměnných. Jak už jsem řekl v poznámce, objeví-li se ve výrazu název proměnné, překladač danou proměnnou najde a na dané místo ve výrazu vloží hodnotu, která je v proměnné uložena.

Identifikace úryvků

JShell vytvořené úryvky postupně čísluje. Tato čísla se pak používají jako identifikátory úryvků. V dalším textu je budu označovat jako ID úryvků.

Obsahuje-li zadaný úryvek výraz, *JShell* tento výraz vyhodnotí a pro výsledek vytvoří novou pomocnou proměnnou, kterou pojmenuje znakem $ (dolar) následovaným ID zadaného úryvku. O tom jste se mohli přesvědčit např. ve výpisu 1.2 na řádcích 3, 9 a 13.

Definujeme-li úryvek, který obsahuje nějaký pojmenovaný objekt (proměnnou, metodu, datový typ), tak se v příkazech pro prostředí *JShell* můžeme na tento objekt odvolávat jak jeho jménem, tak prostřednictvím ID jeho úryvku (až budeme probírat příkazy pro prostředí *JShell*, tak si to předvedeme).

Středník

Java převzala od jazyka C pravidlo, že každý příkaz musíme ukončit středníkem. Platí, že závěrečný středník dělá z výrazu příkaz, který je třeba provést.

Zapomenutý středník patří mezi „oblíbené“ chyby. Zadáme-li proto výraz bez ukončujícího středníku (prozatím tak byly zadány všechny doposud zadané výrazy), překladač ohlásí chybu. *JShell* se pak podívá, jestli by nepomohlo přidání středníku na konec řádku. Pokud je po přidání středníku překladač spokojen, *JShell* nás žádným chybovým hlášením neobtěžuje a prostě si daný úryvek zapamatuje i přidaným středníkem.

Více objektů na řádku, zavlečené chyby

Definujeme-li více pojmenovaných objektů na jednom řádku, *JShell* vytvoří pro každý z těchto objektů samostatný úryvek. Aby však poznal, kde jedna definice končí a jiná začíná, musíme všechny definice s výjimkou té poslední ukončit středníkem. Poslední definici středníkem ukončovat nemusíme (ale samozřejmě můžeme), protože, jak jsme si řekli před chvílí, závěrečný středník je *JShell* ochoten vložit na konec řádku za nás.

Ve výpisu 1.3 je na řádku 1 pokus vložit více úryvků, aniž bych první ukončil středníkem. Jak vidíte, *JShell* ohlásil na řádku 2 chybu, na řádku 3 vysvětlil, že mu chybí očekávaný středník a na řádku 5 dokonce ukázal, kde si myslí, že by se měl onen středník vložit do úryvku vypsaného na řádku 4.

Pak se sice pokusil vyhodnocovat dál, ale první chyba jej poněkud rozhodila, takže další chyby, na něž mne na následujících řádcích upozorňuje, jsou tzv. **zavlečené chyby**, což jsou chyby vzniklé v důsledku špatného pochopení programu zapříčiněného předchozí chybou.

Když jsem na řádku 24 poslechl radu z řádků 4 a 5 a doplnil středník, *JShell* upravené zadání akceptoval a vytvořil proměnné $4 a $5, do nichž uložil hodnoty výrazů v zadaných úryvcích.

Všimněte si, že proměnná $4 vznikla až při vyhodnocení prvního úryvku na daném řádku. Důležité však je, že v okamžiku, kdy se začal vyhodnocovat druhý úryvek na řádku, již byla vytvořena, takže ji druhý úryvek mohl použít.

Výpis 1.3: *Více úryvků na jednom řádku*

```
1  jshell> $1+$3  $1+$4
2  |  Error:
3  |  ';' expected
4  |  $1+$3  $1+$4
5  |       ^
6  |  Error:
7  |  not a statement
8  |  $1+$3  $1+$4
9  |         ^---^
10 |  Error:
11 |  cannot find symbol
12 |    symbol:   variable $4
13 |  $1+$3  $1+$4
14 |            ^^
15 |  Error:
16 |  unreachable statement
17 |  $1+$3  $1+$4
18 |         ^---^
19 |  Error:
20 |  missing return statement
21 |  $1+$3  $1+$4
22 |  ^----------^
23
24 jshell> $1+$3; $1+$4
25 $4 ==> 44
26 |  created scratch variable $4 : int
27 $5 ==> 55
28 |  created scratch variable $5 : int
29
30 jshell>
```

1.4 Příkazy (commands)

Jednou za čas potřebujeme po prostředí i něco jiného než vyhodnocení zadaného úryvku. Potřebujete si připomenout, co už jsme zadali, uložit to do souboru nebo naopak ze souboru nějakou zapamatovanou skupinu úryvků načíst. Pak využijeme příkazy.

Žádný výraz ani příkaz programu v jazyce *Java* nemůže začínat lomítkem. Všechny příkazy prostředí *JShell* naopak lomítkem začínají. Prostředí tak snadno pozná, jestli se chystáme zadat další úryvek nebo příkaz. V této podkapitole se seznámíme pouze s několika základními příkazy, které se nám budou hodit v dalším výkladu.

Vyloučení úryvku: `/drop`

Občas se dostaneme do situace, kdy by bylo nejlepší nějaký úryvek odstranit, přesněji **vyloučit ze seznamu aktivních** (úplně odstranit nejde, *JShell* si pamatuje i ty vyloučené). K tomu slouží příkaz `/drop`, jemuž v parametru předáme identifikační číslo úryvku. Vytváří-li daný úryvek objekt s nějakým názvem, název, můžeme místo identifikačního čísla úryvku zadat název vytvořeného objektu.

Přehled aktivních úryvků: `/list`

Zadáním příkazu `/list` požádáte prostředí o vypsání všech aktivních úryvků. Úryvky, při jejichž zadávání jste udělali chybu, ani úryvky, které jste vyloučili nebo nahradili novější verzí, se nevypisují.

Při výpisu úryvků se dodržuje formátování, v jakém jste úryvky zadali. Když jsem proto druhý úryvek zadal rozprostřený do tří řádků s různým oddělením čísel a operátoru `+` mezerami, tak se tak také vypíše – viz výpis 1.4, řádky `4` až `6`. Stejně tak u úryvku `5` zobrazeného na řádku `9` vidíte, že do něj *JShell* zahrnul všechny znaky následující za středníkem ukončujícím předchozí úryvek včetně úvodní mezery (zadání viz řádek `24` ve výpisu 1.3).

Když jsem však druhý úryvek vyloučil z aktivních (řádky `11` a `12`), tak ho příkaz `/list` již nevypsal (řádky `16` až `19`),

Přehled aktivních úryvků: `/list -all`

O vypsání všech úryvků včetně těch chybných a těch vyloučených požádáte zadáním příkazu `/list -all` (stačí `/list -a`). Ten vypíše všechny zadané úryvky včetně úryvků startovního skriptu (jejich `ID` začíná písmenem `s`) spouštěného při spuštění programu *JShell*. Za ním následuje seznam všech zadaných úryvků včetně těch neaktivních, tj. těch, při jejichž zadávání jste udělali chybu, nebo které jste nahradili novější verzí – viz výpis 1.5.

Výpis 1.4: *Výpisy úryvků příkazem* `/list`

```
jshell> /list

   1 : 6+5
   2 : 6 +
       7+
       9
   3 : $1+$2
   4 : $1+$3;
   5 :  $1+$4

jshell> /drop 2
|  dropped variable $2

jshell> /list

   1 : 6+5
   3 : $1+$2
   4 : $1+$3;
   5 :  $1+$4

jshell>
```

Výpis 1.5: *Výpisy úryvků příkazem* `/list -all`

```
jshell> /list -all

  s1 : import java.io.*;
  s2 : import java.math.*;
  s3 : import java.net.*;
  s4 : import java.nio.file.*;
  s5 : import java.util.*;
  s6 : import java.util.concurrent.*;
  s7 : import java.util.function.*;
  s8 : import java.util.prefs.*;
  s9 : import java.util.regex.*;
 s10 : import java.util.stream.*;
   1 : 6+5
   2 : 6 +
       7+
       9
   3 : $1+$2
  e1 : $1+$3  $1+$4
   4 : $1+$3;
   5 :  $1+$4

jshell>
```

Chybně zadané úryvky poznáte ve výpisu podle toho, že jejich `ID` začíná písmenem `e` (první písmeno slova *error* = chyba). Já jsem prozatím zadal jediný chybný

úryvek, a to, když jsem zadával dva výrazy na jednom řádku a neukončil první středníkem. Tento úryvek dostal `ID e1` a ve výpisu 1.5 je na řádku 18.

Bohužel, úryvky vyloučené z aktivních (např. úryvek s `ID=2`) nejsou ve výpisu nijak označeny, takže je musíte odhalit porovnáním se seznamem aktivních úryvků.

Přehled objektů daného druhu

Někdy nás nezajímají ani tak vlastní úryvky, ale spíše objekty, které jsme zadáním úryvku vytvořili, resp. stav, který je právě nastaven.

Už jsme se setkali s tím, že jsme vytvořili proměnnou. V dalším textu se naučíte vytvářet (= definovat) i metody a datové typy a zadávat různé importy. Pro každý z těchto druhů objektů existuje příkaz pro zobrazení objektů daného typu.

- Příkaz `/vars`, resp. `/vars -all` zobrazí aktivní, resp. všechny proměnné. S proměnnými se seznámíte v kapitole 3 *Proměnné* na straně 78.
- Příkaz `/methods`, resp. `/methods -all` zobrazí aktivní, resp. všechny definované metody. Definovat metody se naučíte v kapitole 5 *Definice metod* na straně 111.
- Příkaz `/types`, resp. `/types -all` zobrazí aktivní, resp. všechny datové typy definované uživatelem. Definovat vlastní datové typy se začnete učit v kapitole 11 *Třídy a jejich členy* na straně 224.
- Příkaz `/imports` zobrazí zadané importy. O importování různých částí programu si povíme v kapitole 13 *Balíčky a knihovny* na straně 263.

Definice metod, datových typů ani importů jsme doposud neprobírali, takže si na použití odpovídajících příkazů musíte počkat[4]. Můžete si ale vyzkoušet výpis všech aktivních proměnných. Výsledek zobrazuje výpis 1.6. Všimněte si, že zadáte-li argument `-all`, bude příkaz `/vars -all` vypisovat i ty neaktivní.

Uložení aktivních úryvků: `/save <file>`

I při práci s prostředím *JShell* potřebujeme občas odběhnout a práci na tuto dobu uložit. K uložení práce slouží příkaz `/save`, který je schopen uložit všechny aktivní úryvky. Jako parametr příkazu se zadává cesta k cílovému souboru, přičemž relativní cesta se odvozuje od složky, v níž jste program spustili. Chcete-li soubor uložit do aktuálního adresáře, stačí napsat pouze jeho název.

Uložené soubory jsou vnímány jako skripty a používá se pro ně přípona `jsh` jako zkratka z názvu *JShell*. Je to sice jenom konvence, ale doporučuji vám ji používat.

---

[4] Myslel jsem tím smysluplné použití. Použít je můžete hned, ale *JShell* vám v odpovědi zobrazí pouze prázdný seznam.

Výpis 1.6: *Výpis aktivních proměnných a všech proměnných*

```
jshell> /vars
|    int $1 = 11
|    int $3 = 33
|    int $4 = 44
|    int $5 = 55

jshell> /vars -all
|    int $1 = 11
|    int $2 = (not-active)
|    int $3 = 33
|    int $4 = 44
|    int $5 = 55

jshell>
```

Zadávat můžete jak absolutní, tak relativní cestu. Zadáte-li pouze název souboru, uloží se do složky, z níž jste program *JShell* spustili. Zadáte-li název souboru i s celou cestou, uloží se tam, kam jste si objednali.

Uložení všech zadaných úryvků: `/save -all <file>`

Zadáním příkazu `/save -all` uložíte všechny zadané úryvky včetně těch neaktivních (chybových nebo přepsaných). To se může hodit např. tehdy, chcete-li se o svých chybách s někým poradit.

Uložení dosavadního průběhu seance: `/save -history <file>`

Zadáním příkazu `/save -history` uložíte vše, co jste v průběhu seance zadali; nejenom úryvky, ale i příkazy. Zadáte-li příkaz `/reset` nebo `/reload` (budou vysvětleny za chvíli), uloží se i ten. Načtením skriptu uloženého tímto příkazem můžete zopakovat kompletní historii seance od spuštění programu *JShell* až do chvíle uložení historie. Tímto příkazem proto budu ukládat průběh jednotlivých kapitol.

Načtení skriptu: `/open <file>`

Uložený skript můžete znovu načíst. K tomu slouží příkaz `/open`, jehož jediným parametrem je cesta k načítanému souboru. Opět můžete zadávat absolutní i relativní cestu k souboru. Nejvýhodnější proto je mít načítané soubory uloženy tamtéž, co dávkový soubor, s jehož pomocí jste *JShell* spustili.

V některých kapitolách vám v poznámce o použitém projektu občas doporučím, abyste načetli nějaký soubor, v němž je připraven užitečný kód, který v dané kapitole použiju. Budete-li chtít zkoušet vše přesně tak, jak to ve výpisech zadávám, budete tento kód potřebovat.

Ukončení seance: /exit

Příkaz ukončí běh programu *JShell*. Nicméně *JShell* si mezi seancemi pamatuje dříve zadané příkazy, takže při následující seanci můžete pomocí šipek aktivovat příkazy z minulé seance, aniž byste je museli znovu celé vypisovat.

Restart: /reset

Občas se dostaneme do situace, v níž bychom nejraději vše zapomněli a začali zcela znovu. Po příkazu `/reset` *JShell* všechny zapamatované úryvky smaže, restartuje virtuální stroj a znovu načte startovní skript.

Znovuzavedení: /reload -restore

Příkaz `/reload -restore` použijete ve chvíli, kdy potřebujete počítač vypnout a po čase se k rozdělané práci vrátit. Zadáte-li po opětovném spuštění programu jako první příkaz `/reload -restore`, načtou se znovu všechny úryvky, které byly při ukončení předchozí seance aktivní.

Natavení startovního skriptu: /set -start <file>

Po spuštění programu *JShell* a po každém resetu se nejprve načte startovní skript. Jeho úryvky se ale příkazem `/list` nezobrazí. Zobrazit byste je mohli pouze příkazem `/list -all` nebo `/list -start`. V tomto skriptu si můžete připravit sadu definic úryvků, které pak budete v průběhu seance používat.

Startovní skript můžete kdykoliv změnit. Nově nastavený se začne načítat od příštího příkazu `/reset` nebo `/reload` a bude platit až do konce seance. Startovní skript nastavíte zadáním příkazu

```
/set -start <file>
```

kde `<file>` zastupuje soubor či skupinu souborů, které se při startu načtou.

V některých kapitolách vám bude na počátku doporučeno, abyste nastavili zadaný startovní skript, resp. startovní skripty. Jsou v nich připraveny úryvky, které budu v průběhu dané kapitoly využívat.

Nápověda: /?

Nápovědu můžeme vyvolat dvěma příkazy: výše uvedeným příkazem `/?`, anebo příkazem `/help`. Za příkaz `/help` můžeme dát parametr s názvem objektu, o kterém se chceme dozvědět něco podrobněji.

Nápovědu můžete získat i tak, že rozepíšete úryvek nebo příkaz a stisknete klávesu <Tab>. Prostředí se „zamyslí“, jestli vám může radit. Pokud ano, tak doplní zadávaný příkaz a/nebo vypíše pod něj seznam možných pokračování.

Napovídací schopnosti prostředí umožní nezadávat příkazy v plném znění. Stačí napsat dostatečný počet znaků, aby prostředí zadávaný příkaz poznalo, a zadat jej.

1.5 Základní syntaktická pravidla

Na závěr představování programu *JShell* bych vás rád seznámil se dvěma základními součástmi kódu, které pomohou zpřehlednit skripty se záznamem průběhu jednotlivých kapitol.

Jak jistě víte, kód je správně zapsaná posloupnost znaků, přičemž ono „správně“ znamená, že je zapsaná podle syntaktických pravidel daného jazyka. V této pasáži vám představím ta hlavní pravidla, v dalším textu se pak budeme postupně seznamovat s dalšími.

Bílé znaky

Kód programu je tvořen posloupností symbolů, mezi něž patří identifikátory, operátory (např. `+ - =`) a oddělovače (např. `, ; :` – čárka, středník, dvojtečka).

Mezi tyto symboly je možno vkládat libovolný počet bílých znaků, přičemž definice jazyka specifikuje bílý znak jako jeden ze znaků:

- mezera (`'\u0020'`),
- vodorovný tabulátor (`'\u000C'`),
- konec stránky – zde jsou tři možnosti:
  - znak LF = `'\u000A'`,
  - znak CR = `'\u000D'`,
  - dvojice znaků CRLF = `"\u000D\u000A"`).

Libovolný počet je i nula. Pokud by se však v případě, kdy mezi dva symboly nevložíte žádný bílý znak, tyto dva symboly slily a definovaly tak symbol jiný, je třeba mezi ně bílý znak vložit.

Když např. napíšete `a+b`, nemusíte znak `+` od okolních identifikátorů oddělovat, protože tento znak nemůže být součástí identifikátoru, takže je zcela zřejmé, kde jeden symbol končí a druhý začíná.

Když ale budete potřebovat napsat `int i` (za chvíli to použijeme), tak mezi tyto dva identifikátory bílý znak vložit musíte, protože `inti` by překladač považoval za identifikátor jediný.

Komentáře

Komentář je část programu, kterou překladač ignoruje a která slouží pouze k tomu, aby čtenář získal o programu nějaké informace, které by mu při čtení kódu nemusely dojít. Komentář můžeme v programu napsat všude tam, kam můžeme napsat mezeru. Překladači je pak jedno, jestli na dané místo napíšeme komentář nebo mezeru – můžeme se na jeho práci dívat tak, že před vlastním překladem nahradí všechny komentáře mezerami.

Java definuje dva druhy komentářů: blokové, které mohou zabírat více řádků, a řádkové, které jsou na jediném řádku.

Řádkový komentář začíná dvojicí lomítek (//) a končí s koncem řádku. Potřebujeme-li pokračovat s informací i na dalším řádku, musíme před pokračovací text znovu vložit dvojici lomítek.

Blokový komentář začíná „*otevírací komentářovou závorkou*“ tvořenou znaky /* (lomítko následované hvězdičkou) a končí „*zavírací komentářovou závorkou*“ tvořenou znaky */ (hvězdička následovaná lomítkem). Uvnitř komentáře mohou být libovolné znaky (včetně přechodu na nový řádek) s výjimkou posloupnosti znaků tvořících zavírací komentářovou závorku. Z toho vyplývá, že blokové komentáře nemůžeme vnořovat. Vložíme-li do blokového komentáře posloupnost /*, bude to mít stejný vliv, jako kdybychom tam vložili cokoliv jiného s výjimkou zavírací komentářové závorky.

Speciálním případem blokového komentáře je **dokumentační komentář**, což je blokový komentář začínající trojicí znaků /**. Dokumentační komentáře slouží (jak název napovídá) k dokumentaci označeného kódu. Podrobněji vás s nimi seznámím, až bude jejich použití smysluplné, konkrétně v podkapitole *14.1 Dokumentační komentáře a API* na straně 285.

Komentáře můžete začlenit jako součást úryvku. Hodí se to např. tehdy, když si budete chtít seanci uložit a uložený skript později prohlížet. Komentář vám může připomenout, co jste zadáním daného úryvku sledovali či předat jinou informaci, kterou byste mohli zapomenout.

Ukázky řádkového i blokového komentáře si můžete prohlédnout ve výpisu 1.7. Dokumentační komentář jsem nepředváděl – později se mu budeme věnovat podrobněji. Na řádku 7 je předvedeno, jak lze komentář použít jako součást úryvku.

Ve výpisu si všimněte, že řádky obsahující pouze komentáře příkaz `/list` nevypisuje. Když je ale komentář součástí nějakého úryvku, příkaz `/list` jej vypíše jako součást daného úryvku – viz řádek 17.

K úryvkům obsahujícím pouze komentáře se *JShell* chová obdobně jako k příkazům (ty také začínají lomítkem). Nezobrazuje ani příkaz `/list -all`, ale pouze příkaz `/list -history`, který vypíše celou konverzaci včetně případných řádků obsahujících pouze komentáře.

Výpis 1.7: Ukázky použití komentářů

```
jshell> //Řádkový komentář

jshell> /* Několikařádkový blokový komentář
   ...>    /* Vložení nové otevírací závorky neodstartuje vnořený komentář
   ...>    vše se ukončí zapsáním zavírací závorky. */

jshell> $1 + $5     //Komentář jako součást úryvku
$6 ==> 66
|  created scratch variable $6 : int

jshell> /list

   1 : 6+5
   3 : $1+$2
   4 : $1+$3;
   5 :  $1+$4
   6 : $1 + $5     //Komentář jako součást úryvku

jshell>
```

1.6 Ovládání

Při editaci úryvků a příkazů používáme nejčastěji editační šipky: šipkami doprava a doleva se přesouváme po aktuálně zadávaném textu, šipkou nahoru a dolů procházíme seznam doposud zadaných úryvků a příkazů.

Prostředí *JShell* sice nabízí řadu dalších klávesových zkratek, ale domnívám se, že byste je stejně nepoužívali. Koho zajímají, ten je najde v již několikrát zmíněné příručce *Java 9 – JShell*.

Použití editoru

Začnete-li používat *JShell* intenzivněji, budete chtít definovat složitější úryvky než ty prostoduché, které jsme definovali doposud. Pro takovéto případy nabízí *JShell* jednoduchý zabudovaný editor, v němž můžete vytvářet a upravovat složitější definice. Editor aktivujete zadáním příkazu

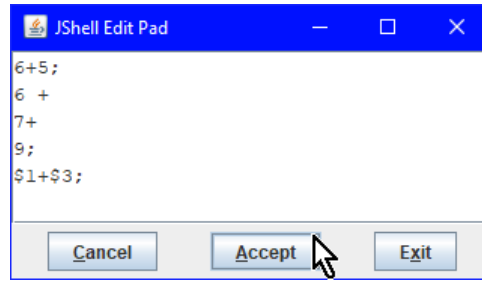
```
/edit <ID>
```

kde parametr `<ID>` může zastupovat jako identifikační číslo úryvku, tak název definovaného objektu.

Můžete dokonce uvést několik názvů či `ID` za sebou oddělených mezerami, přičemž můžete uvést jak aktivní úryvky, tak ty vyloučené. Editor pak otevře všechny jmenované. Na obrázku 1.2 je okno editoru otevřené po zadání příkazu

```
/edit 1 $2 4
```

Příkaz můžete zadat i bez parametru. Pak se v editoru otevřou všechny aktivní úryvky, ale to by byl v současné situaci pouze úryvek 1.



Obrázek 1.2:
Okno zabudovaného editoru otevřené po zadání příkazu `/edit 1 $2 4`

Okno se ovládá tak, že si v něm prohlédnete zobrazený kód a v případě potřeby jej upravíte. Význam tlačítek je následující:

- Tlačítko **Cancel** použijete tehdy, když si svoji úpravu rozmyslíte nebo když jste si chtěli definice jen prohlédnout. Okno editoru se zavře a vy můžete pokračovat v práci v konzolovém okně, aniž by se zaměnil původní stav.
- Tlačítko **Exit** stisknete tehdy, budete-li chtít své úpravy potvrdit. I po jeho stisku se okno editoru zavře, ale před zavřením se zadaný text předá k vyhodnocení.

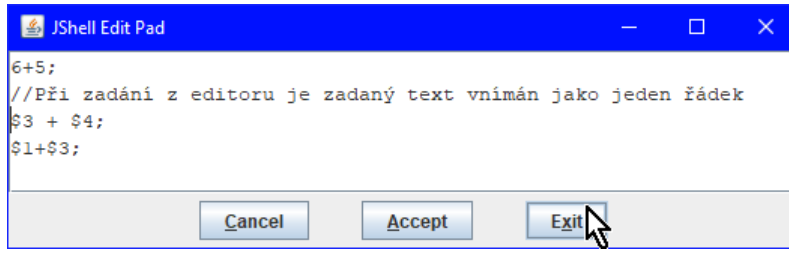
  Je ale dobré vědět, že pokud v editačním okně nic nezměníte (anebo něco sice změníte, ale pak vrátíte text do původního stavu), nic se po stisku tohoto tlačítka nezadá.
- Tlačítko **Accept**, které na obrázku 1.2 stiskává myš, použijete ve chvíli, kdy si nejste jisti tím, že vaše zadání je bezchybné a chcete ho nejprve prověřit. Vaše zadání se pak předá k vyhodnocení, vy si v konzolovém okně můžete prohlédnout výsledek a pokračovat v editaci.

  I zde platí, že shoduje-li se výsledný text s výchozím, tak se platformě nic nezadává.

Když jsem ale upravil podobu druhého (tj. vyloučeného) úryvku podle obrázku 1.3, *JShell* jej akceptoval a uložil upravenou verzi jako úryvek s `ID=7` – viz výpis 1.8.

Ve výpisu 1.8 bych vás chtěl upozornit na několik drobností. Jak naznačuje komentář v okně editoru, *JShell* akceptuje text obdržený z editoru obdobně, jako bychom ho celý zadali v jediném řádku. Z toho vyplývají následující vlastnosti:

- *JShell* si pamatuje podobu zobrazovaného textu a zpracovává pouze tu část, která se změnila – v našem případě pouze prostřední dva řádky definující nový úryvek.



Obrázek 1.3:
Okno zabudovaného editoru otevřené po zadání příkazu `/edit 1 $2 4`

Výpis 1.8: *Příkaz editace a výpis úryvků po editaci v okně na obrázku 1.3*

```
1  jshell> /edit 1 $2 4
2  $7 ==> 77
3  |  created scratch variable $7 : int
4
5  jshell> /list
6
7     1 : 6+5
8     3 : $1+$2
9     4 : $1+$3;
10    5 :  $1+$4
11    6 : $1 + $5     //Komentář jako součást úryvku
12    7 : //Při zadání z editoru je zadaný text vnímán jako řádek
13        $3 + $4;
14
15 jshell>
```

- Středník si můžete odpustit pouze na posledním řádku. Pokud by nebyl některý z úryvků zadaných na předchozích řádcích ukončen středníkem, *JShell* by ohlásil chybu (nebo celou sérii chyb). Stejně jako tomu bylo ve výpisu 1.3 na straně 46.
- Chcete-li, aby se komentář stal součástí úryvku, musíte jej zadat před daným úryvkem. Napíšete-li jej za úryvek (přesněji za ukončující středník), bude jej *JShell* zpracovávat jako součást následujícího úryvku.

Vyzkoušejte si, jaká bude reakce prostředí, pokud byste v editoru smazali středník ukončující první úryvek a jaká bude reakce prostředí, přidáte-li vysvětlující komentář na konec druhého úryvku za jeho ukončující středník.

Nastavení vlastního editoru

Připadá-li vám zabudovaný editor příliš jednoduchý a prostý, můžete použít svůj vlastní. Ten zadáte použitím příkazu

```
/set editor -retain -wait <command>
```

V tomto příkazu parametr `-retain` zadává, že si vaše nastavení bude *JShell* pamatovat a při příštím spuštění bude tento editor již přednastaven. Nechcete-li zadávat editor jako trvalý, ale budete-li jej zadávat pouze pro danou seanci, můžete parametr `-retain` vynechat.

Zadáním parametru `-wait` upravíte reakci prostředí *JShell* tak, že nebude čekat na zavření zavolaného editoru, ale na to, až opět aktivujete okno, v němž běží *JShell* (pravděpodobně konzolové okno). Poté stisknete dvakrát klávesu ENTER. Pomocný soubor určený k editaci přitom může zůstat v osloveném editoru nadále otevřený.

Parametr `<command>` reprezentuje příkaz operačního systému, kterým se daný editor spouští. *JShell* za tento příkaz doplní název souboru s editovaným textem.

Kdybyste měli s nastavením svého oblíbeného editoru nějaké problémy, zkuste si sehnat knihu *Java 9 – JShell*, v níž je tato problematika poměrně podrobně rozebrána.

1.7 Záznamy lekcí

Z lekcí, v nichž bude probíraná látka demonstrována na úryvcích programu *JShell*, budu vytvářet záznamy jako skripty obsahující všechny zadané úryvky a příkazy. Oproti skriptům, které bychom získali zadáním příkazu `/save -history`, budou tyto skripty obsahovat navíc komentáře, které budou naznačovat, v jakém výpisu budou zobrazeny jednotlivé úryvky zadávané v dané kapitole.

1.8 Shrnutí

Skript uchovávající naše akce z této kapitoly spolu s výše uvedenými informačními komentáři je uložen v souboru **01S_c_JShell_Intro.jsh**.