

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



knihovna programátora

RUDOLF PECINOVSKÝ

Java 9

Kompletní příručka jazyka

GRADA
Publishing

Upozornění pro čtenáře a uživatele této knihy

Všechna práva vyhrazena. Žádná část této tištěné či elektronické knihy nesmí být reprodukována a šířena v papírové, elektronické či jiné podobě bez předchozího písemného souhlasu nakladatele.

Neoprávněné užití této knihy bude **trestně stíháno**.

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.

© Grada Publishing, a.s., 2018

Cover Design © Grada Publishing, a. s., 2018

Cover Photo © Depositphotos

Názvy produktů, firem apod. použité v knize mohou být ochrannými známkami nebo registrovanými ochrannými známkami příslušných vlastníků.

ISBN 978-80-271-0930-2 (pdf)

ISBN 978-80-271-0715-5 (print)

Všem, kteří se chtějí něco naučit

Stručný obsah

Stručný obsah	6
Podrobný obsah	8
Seznam výpisů programů	22
Seznam obrázků	27
Seznam odboček – podšeděných bloků	29
Úvod	30
Část I: Neobjektové konstrukce	39
1 Prostředí JShell	40
2 Základní datové typy a jejich literály	58
3 Proměnné	78
4 Základní operátory	91
5 Definice metod	111
6 Ostatní operátory	128
7 Pole	152
8 Rozhodování	166
9 Opakování části kódu	181
Část II: Základní objektové konstrukce	203
10 Základy objektově orientovaného paradigmatu	204
11 Třídy a jejich členy	224
12 Vývojová prostředí a vytvoření aplikace	248
13 Balíčky a knihovny	263
14 Dokumentace API	285
15 Konstrukce interface	298
16 Podrobnosti o konstruktorech	316
17 Úvod do dědění implementace	333
18 Viditelnost členů tříd	351
19 Virtuální metody a jejich přebíjení	369
20 Abstraktní třídy	381

Část III: Pokročilé objektové konstrukce	393
21 Výjimky a aserce	394
22 Generické datové typy a metody	422
23 Typové parametry a argumenty	442
24 Interní datové typy	464
25 Výčtové typy	480
26 Lambda-výrazy	494
27 Anotace	505
28 Vlákna a paralelní procesy	519
28 Moduly.....	526
Literatura.....	551
Rejstřík	552

Podrobný obsah

Stručný obsah	6
Podrobný obsah	8
Seznam výpisů programů	22
Seznam obrázků	27
Seznam odboček – podšeděných bloků	29
Úvod	30
Komu je kniha určena	30
Koncepce výkladu	31
Rozdělení textu	31
Terminologie	32
Použité nástroje	32
Vývojová sada JDK 9	32
Vývojové prostředí JShell	32
Samostatné vývojové prostředí	33
Doprovodné programy	33
Problémy s klávesnicí	33
Historie rozložení České (QWERTY)	34
Syntaktické definice a diagramy	35
Použité typografické konvence	35
Odbočka – podšeděný blok	37
Zpětná vazba	38

Část I: Neobjektové konstrukce 39

1 Prostředí JShell	40
1.1 Charakteristika programu a prostředí JShell	40
1.2 Příprava programu JShell a první spuštění	41
Dávkové soubory pro Windows	41
Po spuštění	42
1.3 Úryvky (snippets)	44
Použití proměnných	45
Identifikace úryvků	45
Středník	45
Více objektů na řádku, zavlečené chyby	45
1.4 Příkazy (commands)	47
Vyloučení úryvku: /drop	47
Přehled aktivních úryvků: /list	47
Přehled aktivních úryvků: /list -all	47
Přehled objektů daného druhu	49
Uložení aktivních úryvků: /save <file>	49

	Uložení všech zadaných úryvků: /save -all <file>.....	50
	Uložení dosavadního průběhu seance: /save -history <file>	50
	Načtení skriptu: /open <file>.....	50
	Ukončení seance: /exit.....	51
	Restart: /reset.....	51
	Znovuzavedení: /reload -restore.....	51
	Natavení startovního skriptu: /set -start <file>	51
	Nápověda: /?.....	51
1.5	Základní syntaktická pravidla.....	52
	Bílé znaky	52
	Komentáře	53
1.6	Ovládání	54
	Použití editoru	54
	Nastavení vlastního editoru.....	57
1.7	Záznamy lekcí	57
1.8	Shrnutí.....	57
2	Základní datové typy a jejich literály	58
2.1	Datové typy	58
	Dělení datových typů.....	59
	Primitivní datové typy.....	60
	Objektové datové typy.....	61
	Odkazy na objekty	62
2.2	Literály.....	62
	Literály typu boolean.....	62
	Literály typu int.....	62
	Historická vsuvka – číselné soustavy	63
	Názvy skupin bitů	64
	Literály typu long.....	66
	Literály typu byte a short	66
	Literály typu double.....	66
	Celé číslo s příponou	67
	Obyčejné desetinné číslo.....	67
	Číslo v semilogaritmicím tvaru	67
	Literály typu float.....	68
	Literály typu char.....	69
	Prázdný odkaz null	72
	Literály typu String.....	73
	Literály typu Class.....	75
2.3	Ještě trocha terminologie.....	75
2.4	Nestandardní hodnoty reálných typů	76
2.5	Shrnutí.....	76
3	Proměnné.....	78
3.1	Pravidla pro tvorbu identifikátorů	78
	Používání znaku \$	79
	Konvence pro velikost písmen.....	79
3.2	Druhy typování	80
	Statické × dynamické typování.....	80
	Definice × odvození datového typu.....	81
	Silné (přísné) × slabé typování	81
	Shrnutí.....	82
3.3	Definice × deklarace.....	82
3.4	Deklarace proměnných	83
3.5	Středníky	84
3.6	Současná deklarace více proměnných.....	85

	Reakce prostředí JShell.....	85
3.7	Redeklarace proměnných v JShell.....	87
3.8	Deklarace s přiřazením počáteční hodnoty	87
	Pozor na velikost znaků.....	88
	Zpět k deklaraci s přiřazením počáteční hodnoty.....	89
3.9	Syntaktický diagram	90
3.10	Shrnutí.....	90
4	Základní operátory	91
4.1	Nejprve trocha teorie	91
4.2	Operátor přiřazení =	92
	Přiřazení je výraz	92
4.3	Unární + a -	93
4.4	Aritmetické operátory + - * / %.....	94
	Operátor sčítání.....	94
	Sčítání textových řetězců	94
	Operátor odčítání.....	95
	Operátor násobení.....	95
	Operátor dělení.....	95
	Operátor zbytku po dělení	96
4.5	Kulaté závorky ()	96
	Alternativní řešení	98
4.6	Operátor přetypování (typ)	98
	Implicitní přetypování	98
	Příklady implicitního přetypování.....	99
	Explicitní přetypování	101
	Priorita	101
	Kontrola	102
	Explicitní přetypování hodnot primitivních typů.....	102
	Příklady.....	103
	Přetypování instancí objektových datových typů.....	105
	Univerzální „přetypování“ na String	105
	Textový podpis	107
4.7	Specifika číselných typů.....	107
	Malé celočíselné typy.....	107
	Ztráta přesnosti	109
	Pořadí vyhodnocování.....	109
	První příklad	109
	Druhý příklad	110
4.8	Shrnutí.....	110
5	Definice metod.....	111
5.1	Historické ohlédnutí.....	111
5.2	Definice a volání metody	112
5.3	Metody s parametry	115
	Formální versus skutečné parametry, argumenty.....	115
	Více parametrů.....	116
5.4	Metody vracějící hodnotu	117
5.5	Přetěžování metod	118
5.6	Lokální proměnné metod.....	119
	Parametry jako lokální proměnné.....	121
	Příklady.....	121
	Jídlna	121
	Návratová hodnota	121
	Definice metod v editoru.....	122
5.7	Metody s proměnným počtem argumentů.....	123

5.8	Zásobník návratových adres (ZNA)	124
	Parametry × lokální proměnné	125
	Předávání hodnot parametrů	125
	Životnost lokálních proměnných	125
5.9	Přehled definovaných metod	126
5.10	Syntaktický diagram	126
5.11	Shrnutí	127
6	Ostatní operátory	128
6.1	Inkrementační a dekrementační operátory ++ --	128
6.2	Porovnávací operátory < <= == != >= >	130
	Testování shody desetinných čísel	130
	Zvláštnosti porovnávání textových řetězců – stringů	131
	p12 == false	132
	p13 == true	132
	p23 == false	133
	Porovnávání objektů reprezentujících hodnotu	133
6.3	Logické operátory ! & && 	134
6.4	Bitové operátory ~ & ^ << >> >>>	136
6.5	Složené přiřazovací operátory Op=	139
	Příklady využití přetytování	140
6.6	Ternární operátor :? – podmíněný výraz	140
	Ještě jednou porovnávání reálných čísel	142
6.7	Operátor instanceof	144
6.8	Zbylé operátory: new [] ()	146
	Operátor new	146
	Operátor []	147
	Operátor . (tečka)	148
	Operátor volání metody ()	148
6.9	Priorita, asociativita a komutativita operátorů	148
	Priorita	148
	Asociativita	150
	Komutativita	150
6.10	Shrnutí	151
7	Pole	152
7.1	Strukturovaný datový typ – kontejner – pole	152
7.2	Deklarace a inicializace polí	153
	Syntaxe zděděná od jazyků C/C++	154
7.3	Přiřazení hodnoty poli a přetytování polí	155
7.4	Počet prvků pole	157
7.5	Práce s prvky pole	158
7.6	Vícerozměrná pole – pole polí	159
	Obdélníková pole	160
	Zubatá pole	160
	Inicializace dvourozměrného pole	162
	Inicializace vícerozměrného pole	163
7.7	Proměnný počet argumentů metod	163
7.8	Arrays – knihovna metod pro práci s poli	163
7.9	Pole a moderní programování	164
7.10	Shrnutí	165
8	Rozhodování	166
8.1	Jednoduchý podmíněný příkaz	166
8.2	Blok příkazů (složený příkaz)	168

	Vnořování bloků	169
	Proměnné lokální v bloku	169
8.3	Úplný podmíněný příkaz	173
8.4	Složený podmíněný příkaz	174
8.5	Přepínač	176
8.6	Shrnutí	180
9	Opakování části kódu	181
9.1	Obecný cyklus	181
9.2	Cyklus s ukončovací podmínkou – cyklus do-while	182
9.3	Cyklus s počáteční podmínkou – cyklus while	183
9.4	Cyklus s parametrem – cyklus for	185
	Metody s proměnným počtem argumentů	187
9.5	„Dvojtečkový“ cyklus for (cyklus „for each“)	189
9.6	Vnořování cyklů	191
9.7	Cyklus s prázdným tělem	191
9.8	Nekonečný cyklus	193
9.9	Cyklus s podmínkou uprostřed	194
9.10	Příkaz break s návěstím	195
9.11	Příkaz continue	198
9.12	Rekurze	199
	Princip	199
	Přímá a nepřímá rekurze	200
	Přeplnění zásobníku návratových adres	200
9.13	Shrnutí	202

Část II: Základní objektové konstrukce **203**

10	Základy objektově orientovaného paradigmatu	204
10.1	Předmluva	204
10.2	Trocha historie	205
10.3	Motivace OOP	206
10.4	Objekty	206
	Členy objektů	207
10.5	Třídy a jejich instance	207
10.6	Třída jako objekt	208
10.7	Členy třídy a jejich instancí	209
	Přežívající lokální proměnné	210
10.8	Zprávy	210
10.9	Metody	211
10.10	Entity	212
10.11	Polymorfismus, rozhraní, interfejs	212
	Rozhraní × implementace	213
	Atributy × vlastnosti	213
	Vlastnosti v knihovně/platformě/frameworku JavaFX	214
	Signatura × kontrakt	214
	Rozhraní × interface	215
	Interfejs a jeho instance	216
10.12	Objektové datové typy	217
10.13	Dědění	217
	Přirozené (nativní) dědění	218
	Dědění typu (rozhraní)	218
	Dědění implementace	219

	Problémy s děděním – substituční princip Liskové (LSP)	219
10.14	Vlastní instance třídy a mateřská třída objektu	221
10.15	Tři základní principy OOP	221
10.16	Jazyk UML	222
10.17	Shrnutí	223
11	Třídy a jejich členy	224
11.1	Nejjednodušší definice třídy	224
11.2	Konstruktory	225
	Implicitní konstruktor	225
	Vlastní konstruktor a skrytý parametr <code>this</code>	225
	Proč se liší podpisy	227
	Definice tříd jako úryvky	227
11.3	Třída se všemi členy	228
	Statické (třídní) členy	228
	Instanční členy	230
	Konstrukce objektů	230
11.4	Kvalifikace posílaných zpráv	231
	Implicitní kvalifikace	232
11.5	Přetěžování konstruktorů	233
	Kvalifikace klíčovým slovem <code>this</code>	236
11.6	Modifikátory přístupu a skrývání implementace	237
	Veřejné a „neveřejné“ datové typy	238
11.7	Přístupové metody	238
11.8	Modifikátor <code>final</code>	240
	Konstantní atributy	240
	Konstanty vyhodnotitelné v době překladu	240
	Konstantní lokální proměnné	241
	Efektivní konstanty	241
	Zveřejňování konstantních atributů	241
	Modifikátor <code>final</code> v procesu dědění	241
	Neměnnost objektů	242
11.9	Primitivní a obalové datové typy – autoboxing	242
	Převody textových řetězců na hodnoty primitivních typů	243
11.10	Důležité metody klíčových tříd	244
	Třída <code>Object</code>	244
	<code>Object clone()</code>	245
	Mělké a hluboké kopie objektů	245
	<code>boolean equals(Object)</code>	245
	<code>Class<?> getClass()</code>	245
	<code>int hashCode()</code>	246
	<code>String toString()</code>	246
	Třída <code>String</code>	246
	Třída <code>Class</code>	246
	<code>boolean equals(Object)</code>	246
	<code>String getName()</code>	247
	<code>String getSimpleName()</code>	247
	<code>String toString()</code>	247
11.11	Shrnutí	247
12	Vývojová prostředí a vytvoření aplikace	248
12.1	IDE	248
	BlueJ a BlueJ++	249
	Nejpoužívanější IDE	249
12.2	Instalace a spuštění NetBeans	250
12.3	Vytvoření spustitelného projektu v NetBeans	250

	Vytvoření nového projektu	250
	Vytvoření nové třídy	252
	Definice hlavní metody	254
12.4	Překlad a sestavení projektu	255
	Překlad	255
	Sestavení	256
	Spustitelnost JAR-souboru	256
	Spuštění aplikace	257
12.5	Zobrazování varovných hlášení	259
	Zapnutí podrobných hlášení	260
	Doporučení	261
	Vypnutí konkrétního hlášení	261
	Proč vypínat varování	262
12.6	Shrnutí	262
13	Balíčky a knihovny	263
13.1	Velké programy a jejich problémy	263
13.2	Balíčky	264
	Umístění zdrojových souborů	265
	Kořenový (implicitní, defaultní, nepojmenovaný) balíček	265
	Podbalíčky	266
	Konvence pro názvy balíčků	266
	Balíčky doprovodných programů a knihoven	267
13.3	Balíčky a NetBeans	268
13.4	Rozšiřujeme strom balíčků	269
	Názvy tříd	271
13.5	Explicitní ukončení aplikace	271
13.6	Příkaz import	273
	Import zadaného datového typu	273
	Import všech typů ze zadaného balíčku	274
	Podpora zadávání příkazu import ve vývojových prostředích	274
	Výjimečnost balíčku java.lang	275
13.7	Příkaz import static	275
13.8	Syntaktický diagram	276
13.9	Používání knihoven	276
13.10	Typy se stejným názvem v různých balíčcích	279
	Shrnutí	281
13.11	Použití knihovny v JShell	281
	Nastavení proměnné classpath	282
	Nastavení importů	282
	Násilné ukončení aplikace	283
13.12	Shrnutí	284
14	Dokumentace API	285
14.1	Dokumentační komentáře a API	285
14.2	Proč psát srozumitelné a komentované programy	286
	POBLIOCHA	287
14.3	Jak psát dokumentační komentáře	288
14.4	Pomocné značky pro tvorbu dokumentace	289
14.5	Dokumentace balíčku a modulu	290
14.6	Vytvoření a zobrazení dokumentace	292
14.7	Struktura dokumentace API	293
	Práce s panely	294
	Struktura dokumentace datového typu	294
14.8	Zpřehlednění programu	295

14.9	Zakomentování a odkomentování části programu	297
14.10	Shrnutí	297
15	Konstrukce interface	298
15.1	Definice typického interfejsu	298
	Deklarace abstraktních metod	299
	Příklad	299
15.2	Implementace interfejsu třídou	300
15.3	Interfejs se všemi přípustnými typy členů	302
	Motivace k rozšíření – implicitní metody	303
	Statické členy	303
	Instanční členy	305
15.4	Dědění interfejsů	305
15.5	Příklad	305
	Plynulé posuny	306
	Plynulé změny velikosti	307
	Sloučení knihoven	307
15.6	Výhody implicitních metod při návrhu architektury	309
15.7	Řešení kolizí	309
15.8	Specifikace zdroje použité metody	310
	Možné problémy	312
15.9	Speciální interfejsy	313
	Značkovací interfejsy	313
	java.lang.Cloneable	313
	java.io.Serializable	314
	Současné trendy a doporučení	314
	Funkční interfejsy	314
	Interfejs Iterable	314
15.10	Shrnutí	315
16	Podrobnosti o konstruktorech	316
16.1	Opakování: co víme o konstruktorech instancí	316
16.2	Zavádění třídy – java.lang.ClassLoader	317
16.3	Statický konstruktor – konstruktor třídy	318
	Konstruktor interfejsu	318
16.4	Instanční inicializační blok	319
16.5	Dvě části těla konstruktora instancí	319
16.6	Příklad	320
	Konstruktor třídy	326
	3 až 9: Úvodní statický inicializační blok	326
	25: Předčasné použití atributu	327
	8: Nekorektní použití metod	327
	42: Předčasné použití konstanty	327
	62: Nekorektní volání konstrukturu	328
	Inicializační část konstrukturu instancí	328
	12 až 15: Úvodní instanční inicializační blok	328
	149: Deklarace konstanty loaded	329
	153 až 157: Inicializační výpočet	329
	165: Použití this v inicializaci	329
	266 až 269: Závěrečný inicializační blok	329
	Těla konstrukturu instancí	329
	177 až 182: Bezparametrický konstruktor	330
	190 až 196: Jednparametrický konstruktor	330
	199 až 204: Dvoupametrický konstruktor	330
	213 až 226: Tříparametrický konstruktor	330
16.7	Experimenty	331

16.8	Doporučení	332
	Jediný statický inicializační blok	332
	Bez instančních inicializačních bloků	332
	Inicializovat všechny atributy jednotně	332
16.9	Shrnutí.....	332
17	Úvod do dědění implementace	333
17.1	Úvodní poznámky.....	334
17.2	Definice dceřiné třídy	334
17.3	Rodičovský podobjekt	336
17.4	Konstruktor	337
	Dědění implementace od více rodičů	338
	Konstrukce rodičovského podobjektu.....	338
17.5	Přetížené verze konstruktorů – použití super × this	340
17.6	Konstruktory rodiče a potomka	342
17.7	Demonstrace chování konstruktorů	343
	Definice třídy Graddaughter.....	343
	Provedení akce před příkazem this() nebo super().....	344
	Definice metody constructorReport(Object,Class).....	346
	Spuštění testu	347
	Zavedení třídy.....	347
	Tisk nehotových objektů	347
	Preference vlastních metod.....	348
	Dokončení testu.....	348
	Rodičovský podobjekt je abstrakce	348
17.8	Zákaz vytváření potomků třídy	350
18	Viditelnost členů tříd	351
18.1	Úpravy použitého projektu	351
18.2	Trocha terminologie.....	352
	Posílání zpráv a volání metod	352
	Přetěžování×přebíjení×zakrývání×přepisování×předefinování metod	352
	Přetěžování metod	352
	Přebíjení metod	353
	Zakrývání metod	353
	Přepsání či předefinování metod.....	353
18.3	Chráněné členy – modifikátor přístupu protected	354
	Shrnutí	356
18.4	Dědění metod	357
	Zděděné, dále neupravované metody	357
	Zděděné metody, pro něž potomek definuje „lepší“ implementaci	358
	Kompatibilita signatur.....	358
18.5	Zakrývání metod předka (method hiding)	359
18.6	Metody, které není možno v potomku zakrýt či přebít – modifikátor final	362
18.7	Zakrývání atributů předka	363
18.8	Metody nově definované v potomku	365
	Staticky × dynamicky typované jazyky	366
	Proč je situace jednoduchá jen zdánlivě	367
	Anotace @Override	367
18.9	Závěr	367
18.10	Shrnutí.....	368
19	Virtuální metody a jejich přebíjení	369
19.1	Princip	369
	Časná a pozdní vazba	370
	Virtuální metody	370

19.2	Které metody jsou v Javě virtuální	371
19.3	Chování virtuálních metod.....	372
19.4	Zdokonalení třídy Square	374
	Přebití metody copy()	374
	Problémy s nastavováním velikosti.....	375
	První návrh definice metody setSize(int,int).....	376
	Test prvního návrhu	377
	Oprava.....	378
19.5	Co se nám na dědění nelíbí	379
19.6	Shrnutí.....	380
20	Abstraktní třídy	381
20.1	Abstraktní třídy a jejich role v dědické hierarchii	381
	Vytváříme hybrida.....	382
	Abstraktní třída bez abstraktních metod	383
20.2	Konstruktor abstraktní třídy	383
20.3	Deklarace a implementace abstraktních metod.....	384
20.4	Účel abstraktních tříd	386
20.5	Proč společný rodič	386
20.6	Účel abstraktních metod	387
20.7	Návrhový vzor Šablonová metoda (Template method)	388
	Princip.....	388
	Implicitní metody interfejsů	388
	Architektura balíčku eu.pedu.lib17w.geom.....	389
	Metoda toString()	390
20.8	Shrnutí.....	392

Část III: Pokročilé objektové konstrukce **393**

21	Výjimky a aserce	394
21.1	Co to jsou výjimky.....	395
21.2	Analyza chybové zprávy	395
	Oznámení o chybě.....	395
	Jak chyba vznikla – výpis zásobníku návratových adres.....	396
21.3	Nejdůležitější výjimky	397
21.4	Vyhození výjimky.....	399
	Oddělené vytvoření výjimky	400
21.5	Výjimky a nedosažitelný kód.....	401
21.6	Co výjimky umí	401
21.7	Hierarchie dědění výjimek	402
21.8	Zachycení vyhozené výjimky.....	404
	Chování metody exceptionCatching(int)	405
21.9	Syntaktický diagram bloku try ... catch	406
	Několik současně odchyťovaných výjimek.....	406
	Společná reakce na několik výjimek.....	407
	Společný úklid – blok finally.....	407
	Příklad	409
21.10	Definice vlastních výjimek	409
21.11	Kontrolované výjimky	411
21.12	Převedení kontrolované výjimky na nekontrolovanou	413
21.13	Informace o skutečném původci výjimky	414
21.14	Ověřování podmínek – příkaz assert	416
	Design by Contract.....	417

21.15	Kdopak mne to volal	420
21.16	Shrnutí.....	421
22	Generické datové typy a metody	422
22.1	Motivace	422
22.2	Generické a parametrizované datové typy.....	425
22.3	Definice generických typů	428
22.4	Použití generických typů	429
22.5	Překlad generických datových typů a očišťování	431
22.6	Rizika nepoužití typových argumentů	432
22.7	Varování překladače a jejich potlačení	435
22.8	Generické metody	436
22.9	Shrnutí.....	441
23	Typové parametry a argumenty	442
23.1	Omezení typových argumentů	442
	Typové argumenty s více předky.....	443
	Vzájemné závislosti typových parametrů	443
23.2	Překlad a očišťování podrobněji.....	444
	Doporučené pořadí omezujících interfejsů	444
	Ztráta informace při běhu	446
	Přemostovací metody.....	446
23.3	Zakázané operace	448
	Za typové parametry nelze dosazovat primitivní typy	448
	Typové parametry třídy není možno použít u statických členů.....	448
	Nelze vytvořit instanci typového parametru.....	449
	Nelze vytvořit pole instancí typového parametru ani parametrizovaného typu	449
	Reflexe.....	450
	Výjimky	450
23.4	Proměnný počet argumentů – @SafeVarargs.....	452
	Omezení.....	453
	Vytvoření pole hodnot.....	454
23.5	Nejednoznačnosti a kolize.....	454
	Falešně přetížená metoda	454
	Nová metoda koliduje se zděděnou	455
	Kolize požadovaných interfejsů	456
	Kolize implementovaných interfejsů	457
	Potomci a předci generických typů – špatné pochopení dědičnosti	458
23.6	Žolíky	459
23.7	Příklad: datový typ <code>Interval<T extends Comparable<? super T>></code>	460
23.8	Shrnutí.....	463
24	Interní datové typy	464
24.1	Motivace	464
	Pomocný soukromý typ.....	464
	Objekt znající útroby a implementující veřejné rozhraní	465
	Sdružení souvisejících typů	465
24.2	Terminologie.....	466
24.3	Společné charakteristiky interních typů	467
24.4	Globální interní (členské) datové typy	468
24.5	Vnořené datové typy	469
24.6	Vnitřní třídy	469
	Interní interfejsy a výčtové typy bez modifikátoru <code>static</code>	470
24.7	Příklad na vnořené a vnitřní třídy	470
	Vnořené <code>Elements</code> × vnitřní <code>SAIterator</code>	472

	Veřejná Elements × soukromá SAIterator	472
	Definice třídy SparseArray.Element	472
	Definice třídy SparseArray.SAIterator	472
	Definice třídy SparseArrayTest	475
24.8	Lokální třídy	476
	Pojmenované lokální třídy	477
	Anonymní třídy	477
	Použití anonymních tříd	479
24.9	Shrnutí	479
25	Výčtové typy	480
25.1	Nejjednodušší definice	480
	Prekladačem přidané atributy a metody	482
	Atribut \$VALUES	482
	public static final NázevTypu[] values()	482
	public static NázevTypu valueOf(String name)	482
25.2	Třída Enum	483
	Zděděné metody	483
	public final String name()	483
	public final int ordinal()	483
	public final int compareTo(E o)	484
	public final Class<E> getDeclaringClass()	484
	public static <T extends Enum<T>> T valueOf(Class<T> enumType, String name)	484
	Přebité verze metod zděděných od třídy Object	484
	protected final Object clone() throws CloneNotSupportedException	484
	equals(Object)	484
	hashCode()	485
	public String toString()	485
	Serializace	485
25.3	Použití výčtových typů v programu	485
	Přepínač	485
	Přidaná anonymní třída	486
	Cyklus	487
25.4	Složitější definice výčtových typů	487
25.5	Akční výčtové typu	491
	Class-objekty instancí funkčních výčtových typů	492
25.6	Shrnutí	493
26	Lambda-výrazy	494
26.1	Motivace	494
26.2	Koncepce lambda-výrazů	495
26.3	Funkční interfejsy	496
26.4	Syntaxe lambda-výrazů	498
	Jednoduchý příklad	499
	Lambda-výrazy zastupující metody	500
	Lambda-výraz zastupující konstruktor	501
26.5	Použití lokálních proměnných z okolního bloku	503
26.6	Shrnutí	504
27	Anotace	505
27.1	Co jsou anotace	505
27.2	Označování deklarací anotacemi	506
27.3	Kde všude můžeme anotace použít	508
	Anotování balíčků	509
	Anotování parametru this	510

27.4	Anotace ve standardní knihovně	510
	Standardní anotace v balíčku <code>java.lang</code>	511
	<code>@Deprecated</code>	511
	<code>@Override</code>	511
	<code>@SuppressWarnings</code>	511
	<code>@SafeVarargs</code>	512
	<code>@FunctionalInterface</code>	512
	Standardní anotace v balíčku <code>javax.annotation</code>	512
	Metaanotace	512
	<code>@Documented</code>	513
	<code>@Inherited</code>	513
	<code>@Repeatable</code>	513
	<code>@Retention</code>	514
	<code>@Target</code>	514
27.5	Syntaxe definice anotací	515
	Jednoduchá značkovácí anotace	517
27.6	Získávání informací o anotacích za běhu programu	518
27.7	Shrnutí	518
28	Vlákna a paralelní procesy	519
28.1	Paralelní provádění více činností	519
	Kooperativní plánování	520
	Preemptivní plánování	520
	Použité plánování	520
28.2	Vlákna a jejich stavy	520
28.3	Sdílení zdrojů	522
28.4	Kritické sekce a monitory	523
28.5	Synchronizace	523
28.6	Uvolnění kritické sekce	524
28.7	Jemnější způsoby synchronizace	525
	Modifikátor <code>volatile</code>	525
	Atomické objekty	525
28.8	Shrnutí	525
29	Moduly	526
29.1	Motivace	526
	Problémy předchozích verzí Javy	527
	Cíle projektu Jigsaw	528
	Dosažené výhody	529
29.2	Srovnání instalace Javy 8 a Javy 9	529
29.3	Modul × Balíček	530
29.4	Soubor <code>module-info.java</code>	531
	Syntaktický diagram deklarace modulu	532
	Název modulu	533
	Direktiva <code>requires</code>	533
	Direktiva <code>exports</code>	534
	Direktiva <code>opens</code> a modifikátor <code>open</code>	534
	Direktiva <code>uses</code>	535
	Direktiva <code>provides</code>	535
29.5	Modulární JAR-soubor	535
29.6	Proměnná <code>modulepath</code>	535
29.7	Vytvoření modulární aplikace	536
	Vytvoření projektu	536
	Definice modulu	537
	Přidání zdrojových souborů	539
	Odstraňování chyb z nepokrytých závislostí	539

	Přidání modulu eu.pedu.lib17w.geom	542
	Přidání modulu eu.pedu.lib17w.canvas	543
	Přidání modulu eu.pedu.lib17w.canvasmanager	543
	Závěrečná podoba deklarací modulů	544
	JAR-soubor s více moduly	545
29.8	Klasifikace modulů	545
	Běžný modul (normal module)	546
	Otevřený modul (open module)	546
	Automatický modul (automatic module)	546
	Vlastnosti automatických modulů	548
	Nepojmenované moduly (unnamed modules)	548
29.9	Moduly a platforma JShell	549
29.10	Shrnutí	550
	Literatura	551
	Rejstřík	552

Seznam výpisů programů

Výpis 1.1:	Obsah konzolového okna otevřeného po spuštění dávky J9_JShell.bat na mém počítači	43
Výpis 1.2:	První tři pokusné úryvky	44
Výpis 1.3:	Více úryvků na jednom řádku	46
Výpis 1.4:	Výpisy úryvků příkazem /list	48
Výpis 1.5:	Výpisy úryvků příkazem /list -all	48
Výpis 1.6:	Výpis aktivních proměnných a všech proměnných	50
Výpis 1.7:	Ukázky použití komentářů	54
Výpis 1.8:	Příkaz editace a výpis úryvků po editaci v okně na obrázku 1.3	56
Výpis 2.1:	Literály typu boolean	62
Výpis 2.2:	Literály typu int v různých číselných soustavách	65
Výpis 2.3:	Literály typu long	66
Výpis 2.4:	Různé způsoby zadání literálů typu double	68
Výpis 2.5:	Literály typu float	69
Výpis 2.6:	Literály typu char	72
Výpis 2.7:	Literály typu String	74
Výpis 2.8:	Nestandardní hodnoty reálných typů	77
Výpis 3.1:	Deklarace proměnných základních typů	84
Výpis 3.2:	Seznam deklarovaných proměnných	85
Výpis 3.3:	Deklarace více proměnných jediným příkazem	86
Výpis 3.4:	Redeklarace proměnné	88
Výpis 3.5:	Deklarace s přiřazením počáteční hodnoty	89
Výpis 4.1:	Použití přiřazení jako výrazu	93
Výpis 4.2:	Sčítání řetězců	95
Výpis 4.3:	Způsoby zpracování dělení	96
Výpis 4.4:	Operátor zbytku po dělení	97
Výpis 4.5:	Implicitní přetypování a jeho možnosti	100
Výpis 4.6:	Explicitní přetypování hodnot primitivních typů	104
Výpis 4.7:	Explicitní přetypování hodnot objektových typů	106
Výpis 4.8:	Specifika číselných typů	108
Výpis 5.1:	Definice a následné zavolání metody doubleText()	114
Výpis 5.2:	Definice a použití metody println(Object)	116
Výpis 5.3:	Definice a použití metody printPerson(String, String, String, int)	117
Výpis 5.4:	Definice a použití metody hypotenuse(double, double)	118
Výpis 5.5:	Použití přetížených verzí metod	119
Výpis 5.6:	Lokální proměnné a jejich použití	120
Výpis 5.7:	Lokální proměnné a jejich použití	122
Výpis 5.8:	Metody s proměnným počtem argumentů	124

Výpis 5.9:	Výpis aktivních metod.....	126
Výpis 6.1:	Porovnávání textových řetězců (stringů)	131
Výpis 6.2:	Porovnávání textových řetězců (stringů)	132
Výpis 6.3:	Podmíněné (zkrácené) a nepodmíněné logické operátory	135
Výpis 6.4:	Přípravné úryvky.....	137
Výpis 6.5:	Bitové logické operace.....	138
Výpis 6.6:	Bitové posunové operace	139
Výpis 6.7:	Složené přiřazovací operátory	141
Výpis 6.8:	Ternární operátor.....	143
Výpis 6.9:	Definice metody equals(double, double) porovnávající reálná čísla	144
Výpis 6.10:	Operátor instanceof	145
Výpis 6.11:	Operátor new.....	147
Výpis 6.12:	Priorita operátorů.....	150
Výpis 7.1:	Deklarace a inicializace polí	154
Výpis 7.2:	Přiřazení hodnoty poli a jejich přetypování	156
Výpis 7.3:	Zjišťování délky pole.....	157
Výpis 7.4:	Přístup k prvkům pole.....	159
Výpis 7.5:	Obdélníková pole.....	161
Výpis 7.6:	Zubaté pole.....	162
Výpis 7.7:	Proměnný počet argumentů	164
Výpis 8.1:	Jednoduchý podmíněný příkaz.....	167
Výpis 8.2:	Blok příkazů.....	168
Výpis 8.3:	Vnořování bloků a používání lokálních proměnných	171
Výpis 8.4:	Volání metody definované ve výpisu 8.3	172
Výpis 8.5:	Úplný podmíněný příkaz.....	173
Výpis 8.6:	Zanořování příkazů při výběru z více možností	175
Výpis 8.7:	Doporučené formátování při výběru z více možností	175
Výpis 8.8:	Volání metody sqrt(double, double, double) řešící kvadratickou rovnici	175
Výpis 8.9:	Definice metody sqrt(double, double, double) řešící kvadratickou rovnici.....	176
Výpis 8.10:	Koncepce přepínače – příkazu switch	178
Výpis 8.11:	Ukázka použití přepínače (příkazu switch) v definici metody.....	179
Výpis 8.12:	Volání metody využívající přepínač.....	179
Výpis 9.1:	Ukázka použití cyklu s ukončovací podmínkou (cyklu do-while) – definice metody clockTest_do()	183
Výpis 9.2:	Ukázka použití cyklu s počáteční podmínkou (cyklu while) – definice metody clockTest_while(int)	184
Výpis 9.3:	Ukázka použití cyklu s počáteční podmínkou (cyklu for) – definice metody clockTest_for(int).....	187
Výpis 9.4:	Definice metody variantArguments(int...) demonstrující zpracování proměnného počtu argumentů a klíčové vlastnosti cyklu for.....	189
Výpis 9.5:	Ukázka použití dvojtečkového cyklu for	190
Výpis 9.6:	Zanořování cyklů, srovnání klasického a dvojtečkového cyklu for	192
Výpis 9.7:	Ukázka použití cyklu for s prázdným tělem.....	193
Výpis 9.8:	Definice metody endlessLoop() s nekonečným cyklem.....	194
Výpis 9.9:	Definice metody clockTest_break(int) demonstrující použití cyklu s podmínkou uprostřed.....	196
Výpis 9.10:	Metoda labeledBreak() demonstrující použití příkazu break s návěštím.....	197
Výpis 9.11:	Metoda labeledContinue() demonstrující funkci příkazu continue.....	198

Výpis 9.12:	Demonstrace funkce metody <code>labeledContinue()</code>	199
Výpis 9.13:	Definice metody <code>stackOverflow(int)</code> vedoucí k přetečení zásobníku	201
Výpis 9.14:	Definice metody <code>factorial(int)</code> demonstrující rekurzivní volání	201
Výpis 11.1:	Definice třídy <code>Empty</code>	224
Výpis 11.2:	Definice třídy <code>Empty</code> s explicitně definovaným konstruktorem	226
Výpis 11.3:	Přehled dosavadních úryvků této lekce – celá definice třídy je jeden úryvek	227
Výpis 11.4:	Definice třídy <code>AllMembers</code>	229
Výpis 11.5:	Vytváření instancí třídy <code>AllMembers</code>	231
Výpis 11.6:	Zasílání zpráv instancím třídy <code>AllMembers</code>	233
Výpis 11.7:	Definice třídy <code>MC</code> s trojicí konstruktorů	235
Výpis 11.8:	Vytváření instancí třídy <code>MC</code>	237
Výpis 11.9:	Autoboxing – automatické převody hodnot primitivních typů na instance příslušných obalových typů a zpět	243
Výpis 11.10:	Převody textových řetězců na hodnoty primitivních typů	244
Výpis 12.1:	Definice metody <code>main(String[])</code> ve třídě <code>Main</code>	255
Výpis 12.2:	Konec výpisu v panelu Output po prvním použití příkazu Clean and Build	255
Výpis 12.3:	Konec výpisu v panelu Output po prvním použití příkazu Clean and Build	258
Výpis 12.4:	Dvě spuštění aplikace s různými sadami argumentů	259
Výpis 13.1:	Definice pomocné třídy <code>Canvas</code> v balíčku <code>eu.pedu.b57_j9lang.c13_packages</code>	280
Výpis 13.2:	Nastavení proměnné <code>classpath</code> a vytvoření okna plátna, tj. instance třídy <code>Canvas</code>	283
Výpis 13.3:	Reakce na násilné uzavření aplikačního okna plátna a následně zadání příkazu <code>/reload -restore</code>	284
Výpis 14.1:	Demonstrace využití značek v dokumentačních komentářích	291
Výpis 14.2:	Obsah souboru <code>package-info.java</code> s dokumentačním komentářem balíčku <code>eu.pedu.b57_j9lang.k14_javadoc</code>	292
Výpis 14.3:	Šablona zdrojového kódu tříd s řádkovými komentáři označujícími sekce	296
Výpis 15.1:	Definice interfejsu <code>IInterfaceTypical</code>	300
Výpis 15.2:	Definice a použití interfejsu <code>IInterfaceTypical</code> v <code>JShell</code>	301
Výpis 15.3:	Definice demonstračního interfejsu <code>IInterfaceFull</code> obsahujícího všechny povolené typy členů	304
Výpis 15.4:	Demonstrace kolize děděných metod a její řešení	311
Výpis 15.5:	Specifikace předka, jehož metodu volám	312
Výpis 15.6:	Možné problémy při dědění interfejsů	313
Výpis 15.7:	Použití dvojtečkového cyklu <code>for</code> (cyklu <code>for-each</code>) pro iterovatelné objekty	315
Výpis 16.1:	Definice třídy <code>This</code>	320
Výpis 16.2:	Definice třídy <code>Constructors</code>	321
Výpis 17.1:	Definice třídy <code>Square</code> a prověření její funkčnosti	336
Výpis 17.2:	Definice třídy <code>Square</code> s upravenou verzí konstrukturu	340
Výpis 17.3:	Definice třídy <code>Square</code> se čtyřmi konstruktory	341
Výpis 17.4:	Definice třídy <code>Granddaughter</code>	345
Výpis 17.5:	Definice metody <code>constructorReport(Object, Class)</code> ve třídě <code>Reports</code>	346
Výpis 17.6:	Vytváření instancí tříd v dědické hierarchii	349
Výpis 18.1:	Definice chráněných metod ve třídě <code>Mother</code>	355
Výpis 18.2:	Definice metody <code>parentCall()</code> ve třídě <code>OutsidePackage</code>	355
Výpis 18.3:	Nekomentované definice interfejsů <code>IMother</code> a <code>IDAughter</code> a <code>IGranddaughter</code>	360

Výpis 18.4:	Nekomentovaná definice třídy TestIHiding.....	361
Výpis 18.5:	Nekomentovaná definice třídy TestAttrHiding	364
Výpis 19.1:	Reakce vnučky na volání virtuálních metod definovaných již v matce	373
Výpis 19.2:	Přebíjecí definice metody copy() a třídě Square	375
Výpis 19.3:	Nabídnutá definice metody setSize(int,int) ve třídě Square.....	376
Výpis 19.4:	Upravená definice metody setSize(int,int) ve třídě Square.....	377
Výpis 19.5:	Definice testovací hlavní metody ve třídě Square	377
Výpis 19.6:	Počátek chybového hlášení zobrazeného po spuštění třídy Square	377
Výpis 19.7:	Definice implicitní metody setSize(int) v interfejsu IResizable	378
Výpis 19.8:	Chybové hlášení zobrazené po spuštění opravené třídy Square	379
Výpis 20.1:	Definice abstraktní třídy Abstract.....	384
Výpis 20.2:	Definice třídy Concrete jako potomka třídy Abstract.....	384
Výpis 20.3:	Doplnění třídy Abstract o abstraktní metodu	385
Výpis 20.4:	Definice třídy Concrete s doplněnou implementací zděděné metody.....	386
Výpis 20.5:	Definice metod toString() a forToString() ve třídě ANamed	391
Výpis 20.6:	Definice metody forToString() ve třídě AMovable.....	391
Výpis 20.7:	Definice metody forToString() ve třídě AChangeable.....	392
Výpis 20.8:	Definice metody forToString() ve třídě Ellipse.....	392
Výpis 21.1:	Ukázka vyhození výjimky v prostředí JShell.....	399
Výpis 21.2:	Definice metody cleanUpFirst() oddělující vytvoření a vyhození výjimky.....	400
Výpis 21.3:	Metoda unreachable() demonstrující nedosažitelnost kódu za vyhozením výjimky.....	401
Výpis 21.4:	Demonstrace zachycení vyhozené výjimky.....	405
Výpis 21.5:	Metoda recursion4(int) ve třídě ExceptionCatching demonstrující použití bloku finally.....	408
Výpis 21.6:	Reakce prostředí JShell po zadání příkazu catchTest(4);	410
Výpis 21.7:	Definice výjimek CheckedException a UncheckedException	411
Výpis 21.8:	Metody demonstrující práci s kontrolovanými a nekontrolovanými výjimkami.....	412
Výpis 21.9:	Definice metody conversion() převádějící kontrolovanou výjimku na nekontrolovanou	413
Výpis 21.10:	Demonstrace výpisu obsahu zásobníku výjimek znajících svého původce.....	414
Výpis 21.11:	Chybové hlášení po spuštění třídy Exceptions z výpisu 21.10	415
Výpis 21.12:	Definice třídy AssertDemo pro demonstraci funkce příkazu assert	418
Výpis 21.13:	Definice metody getCallerName(int,int) ve třídě CallerReporter	420
Výpis 22.1:	Definice třídy ObjectCrate2, jejíž instance představují univerzální přepravky pro dvě hodnoty; místa, kde byl použit typ Object jako náhražka předem neznámých datových typů atributů, jsou zvýrazněna podbarvením	423
Výpis 22.2:	Definice testovací třídy ObjectCrate2Test; místa, kde je nutno získanou hodnotu přetypovat, jsou zvýrazněna podbarvením.....	424
Výpis 22.3:	Definice generické třídy GenericCrate2; místa, kde jsou deklarovány a následně použity typové parametry, jsou zvýrazněna podbarvením	427
Výpis 22.4:	Definice testovací třídy GenericCrate2Test; místa, kde jsou zadány typové argumenty, jsou zvýrazněna podbarvením	430
Výpis 22.5:	Definice testovací třídy GenericCrate2Test2.....	433
Výpis 22.6:	První dvě varovná hlášení překladače při překladu třídy.....	435
Výpis 22.7:	Definice generické metody assignContent(GenericCrate2<T1,T2>, GenericCrate2<T1, T2>) ve třídě GenericMethods	437

Výpis 22.8:	Definice testovací metody <code>testAssignContent()</code> demonstrující ve třídě <code>GenericMethods</code> použití generické metody <code>assignContent</code>	438
Výpis 23.1:	Definice tříd <code>ListTest1</code> a <code>ListTest2</code>	444
Výpis 23.2:	Definice tříd <code>ParentChild</code> a <code>ChildParent</code>	444
Výpis 23.3:	Definice třídy a zpětně přeložené verze jejího překladu.....	445
Výpis 23.4:	Definice třídy <code>UpperStringList</code>	447
Výpis 23.5:	Definice přemosťovací metody.....	448
Výpis 23.6:	Definice třídy <code>Prohibited</code>	449
Výpis 23.7:	Definice polí instancí třídy <code>ArrayList</code>	451
Výpis 23.8:	Definice metody používající proměnný počet argumentů parametrizovaného typu.....	452
Výpis 23.9:	Možné nekorektní operace.....	452
Výpis 23.10:	Vytvoření pole typového parametru.....	453
Výpis 23.11:	Definice generické třídy <code>Crate12</code>	454
Výpis 23.12:	Definice generické třídy <code>Crate11</code>	455
Výpis 23.13:	Definice tříd <code>Parent</code> , <code>Child</code> a <code>IntervalD</code>	456
Výpis 23.14:	Upravená (ale špatně) definice třídy <code>Child</code>	457
Výpis 23.15:	Definice třídy <code>Interval<T extends Comparable<? super T>></code>	460
Výpis 24.1:	Začátek definice třídy <code>SparseArray</code> – definice atributů a metod; definice pokračuje v následujících výpisech.....	471
Výpis 24.2:	Pokračování definice třídy <code>SparseArray</code> – definice vnořené přepravky <code>Element</code>	473
Výpis 24.3:	Pokračování definice třídy <code>SparseArray</code> – definice vnitřní třídy <code>SAIterator</code>	474
Výpis 24.4:	Definice demonstrační třídy <code>SparseArrayTest</code>	475
Výpis 24.5:	Při volání metody <code>Arrays.setAll(?)</code> je druhý argument zadáván jako instance anonymní třídy.....	478
Výpis 25.1:	Zpětný překlad bajtkódu třídy <code>Season</code>	481
Výpis 25.2:	Definice metody <code>activity_1(Season)</code> ve třídě <code>SeasonUtil</code>	486
Výpis 25.3:	Definice metody <code>activity_2(Season)</code> ve třídě <code>SeasonUtil</code>	486
Výpis 25.4:	Definice metody <code>enumerate()</code> ve třídě <code>SeasonUtil</code>	487
Výpis 25.5:	Výsek z definice třídy <code>Direction8</code> – 1. část s deklaracemi atributů třídy.....	488
Výpis 25.6:	Výsek z definice třídy <code>Direction8</code> – 2. část s deklaracemi atributů instancí a konstruktorem instancí.....	490
Výpis 25.7:	Výseky z definice třídy <code>Direction8</code> – 3. část s definicí konstrukturu třídy.....	491
Výpis 25.8:	Definice třídy <code>Operation</code> a demonstrace jejího použití.....	492
Výpis 25.9:	Zjišťování mateřské třídy instancí výčtových typů.....	493
Výpis 26.1:	Funkční interfejsy v balíčku <code>java.util.function</code>	497
Výpis 26.2:	Důležité funkční interfejsy z jiných balíčků.....	498
Výpis 26.3:	Demonstrace použití lambda-výrazu na aplikaci vybrané operace.....	499
Výpis 26.4:	Rozdílné vnímání signatury metody při kvalifikaci třídou a instancí.....	501
Výpis 26.5:	Definice testovacích metod světél s nastavitelným tvarem.....	502
Výpis 26.6:	Definice třídy <code>JumpingButton</code>	504
Výpis 28.1:	Deklarace modulu <code>eu.pedu.lib17w.util</code>	533
Výpis 28.2:	Deklarace jednotlivých modulů projektu B57_Lib_Modular	544
Výpis 28.3:	Nastavení proměnné <code>modulepath</code> a používaných modulů v prostředí <code>JShell</code>	549

Seznam obrázků

Obrázek 1.1: Konzolové okno otevřené po spuštění dávky J9_JShell.bat na mém počítači	43
Obrázek 1.2: Okno zabudovaného editoru otevřené po zadání příkazu /edit 1 \$2 4	55
Obrázek 1.3: Okno zabudovaného editoru otevřené po zadání příkazu /edit 1 \$2 4	56
Obrázek 2.1: Syntaktický diagram zápisu čísla v různých číselných soustavách	64
Obrázek 2.2: Syntaktický diagram zápisu čísla typu double	69
Obrázek 3.1: Syntaktický diagram deklarace proměnné	90
Obrázek 5.1: Podoba definic metod v editoru	123
Obrázek 5.2: Syntaktický diagram definice metody	127
Obrázek 8.1: Vývojový diagram podmíněného příkazu	166
Obrázek 8.2: Vývojový diagram úplného podmíněného příkazu	173
Obrázek 8.3: Syntaktický diagram obecného podmíněného příkazu	173
Obrázek 8.4: Vývojový diagram složeného podmíněného příkazu	174
Obrázek 8.5: Syntaktický diagram přepínače – příkazu switch	176
Obrázek 8.6: Vývojový diagram přepínače – příkazu switch	177
Obrázek 9.1: Vývojový diagram obecného cyklu	181
Obrázek 9.2: Vývojový diagram cyklu s ukončovací podmínkou (cyklu do-while)	182
Obrázek 9.3: Syntaktický diagram cyklu s ukončovací podmínkou (cyklu do-while)	182
Obrázek 9.4: Syntaktický diagram cyklu s počáteční podmínkou (cyklu while)	183
Obrázek 9.5: Vývojový diagram cyklu s počáteční podmínkou (cyklu while)	184
Obrázek 9.6: Vývojový diagram cyklu s parametrem (cyklu for)	185
Obrázek 9.7: Syntaktický diagram cyklu s parametrem (cyklu for)	185
Obrázek 9.8: Syntaktický diagram „dvojtečkového“ cyklu for („cyklu „for each“)	190
Obrázek 9.9: Vývojový diagram cyklu s podmínkou uprostřed	195
Obrázek 10.1: Základní prvky používané v diagramu tříd jazyka UML	223
Obrázek 12.1: První ze série dialogových oken pro vytvoření nového projektu	251
Obrázek 12.2: První ze série dialogových oken pro vytvoření nového projektu	252
Obrázek 12.3: Vytvoření nové třídy	252
Obrázek 12.4: Zadání názvu a umístění vytvářené třídy	253
Obrázek 12.5: Zobrazení vytvořené třídy	254
Obrázek 12.6: Dialogové okno Properties	257
Obrázek 12.7: Syntaktický diagram příkazů import a import static	258
Obrázek 12.8: Nastavení parametru překladače pro podrobná varovná hlášení	260
Obrázek 13.1: Stromová struktura balíčků modulu java.base	266
Obrázek 13.2: Zadání nově vytvářené balíčku eu.pedu.b57_j9lang.c13_packages	268
Obrázek 13.3: Aplikační okno po zkopírování třídy Main do balíčku	269
Obrázek 13.4: Aplikační okno po zkopírování třídy Main do balíčku	270
Obrázek 13.5: Ručička ukazuje na stavovém řádku na informaci o tom, že aplikace stále běží	272

Obrázek 13.6: Syntaktický diagram příkazů <code>import</code> a <code>import static</code>	276
Obrázek 13.7: Panel projektů a dvě podoby zobrazení seznamu balíčků projektu B57_Lib_Pkg	277
Obrázek 13.8: Okno vlastností projektu 13P_Packages při zadání připojovaných zdrojů	278
Obrázek 13.9: Dialogové okno umožňující zadat připojovaný projekt	279
Obrázek 13.10: Dialogové okno umožňující zadat připojovaný projekt	280
Obrázek 14.1: Nastavení parametrů pro javadoc v okně Properties projektu.....	293
Obrázek 14.2: Okno prohlížeče zobrazující dokumentaci projektu	294
Obrázek 15.1: Architektura části knihovny z projektu B57_Lib_Pkg	308
Obrázek 17.1: Zanořování rodičovských podobjektů	337
Obrázek 18.1: Upozornění NetBeans na zakrytí rodičovského atributu a nabídka řešení	363
Obrázek 19.1: Dialogové okno s metodami, které je možno přebít	376
Obrázek 19.2: UML diagram části dědické hierarchie třídy Square	378
Obrázek 20.1: Architektura balíčku <code>eu.pedu.lib17w.geom</code>	389
Obrázek 21.1: Hierarchie dědění výjimek	403
Obrázek 21.2: Syntaktický diagram bloku <code>try ... catch</code>	406
Obrázek 21.3: Syntaktický diagram příkazu <code>assert</code>	417
Obrázek 21.4: Zadání parametru povolujícího provádění příkazu <code>assert</code>	419
Obrázek 23.1: Vztah předek potomek hodnot typových argumentů se nepřenáší na instance generického datového typu	458
Obrázek 24.1: Rozdělení interních datových typů	467
Obrázek 27.1: Syntaktický diagram definice anotace	516
Obrázek 28.1: Stavby vláken a přechody mezi nimi	521
Obrázek 28.1: Odchytky struktury složek osmé a deváté verze Javy	530
Obrázek 28.2: Závislosti modulů demonstrační knihovny na jiných modulech	531
Obrázek 28.3: Syntaktický diagram deklarace modulu	532
Obrázek 28.4: Dialogové okno New Project	537
Obrázek 28.5: Dialogové okno New Java Modular Application	537
Obrázek 28.6: Vytvoření nového modulu	538
Obrázek 28.7: Dialogové okno New Module pro zadání náavu a umístění vytvářeného modulu.....	538
Obrázek 28.8: Panel Projects po vytvoření pseudokořenového balíčku	539
Obrázek 28.9: Využití pole SEARCH k vyhledání dokumentace zadaného typu	540
Obrázek 28.10: Zjištění modulu, v němž se nachází daná třída	541
Obrázek 28.11: Šest typů modulů.....	545

Seznam odboček – podšeděných bloků

Historie rozložení České (QWERTY).....	34
Odbočka – podšeděný blok.....	37
Historická vsuvka – číselné soustavy	63
Názvy skupin bitů.....	64
Mělké a hluboké kopie objektů.....	245
Dědění implementace od více rodičů	338
Provedení akce před příkazem <code>this()</code> nebo <code>super()</code>	344
Staticky × dynamicky typované jazyky	366
Design by Contract	417
Reflexe.....	450

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 objektivě 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 nezbyvá 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 třídu 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 efektivitu 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 obzvláště 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 *Java* 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**, v které najdete podsekcí **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 *Java* 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_i9lang. 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 rozbořem 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.

- 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 jejich produktů vysazuji *kurzivou*. Kurzivou vysazuji také názvy kapitol, podkapitol a oddílů, na které se v textu odkazují.

[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 vysazují obyčejným bezpatkovým písmem.
- Program** Identifikátory a další části programů zmíněné v běžném textu vysazují **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 šfouraly“, 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](mailto: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 nabízejí 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 okna¹. 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 editoru² 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.

kteří 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.

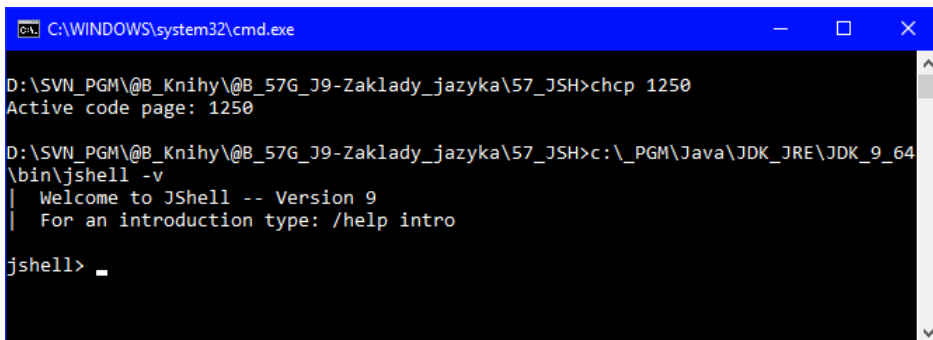
Po spuštění

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 oddělovací 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.



```
C:\WINDOWS\system32\cmd.exe
D:\SVN_PGM\@B_Knihy\@B_57G_J9-Zaklady_jazyka\57_JSH>chcp 1250
Active code page: 1250

D:\SVN_PGM\@B_Knihy\@B_57G_J9-Zaklady_jazyka\57_JSH>c:\_PGM\Java\JDK_JRE\JDK_9_64
\bin\jshell -v
| Welcome to JShell -- Version 9
| For an introduction type: /help intro

jshell> _
```

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 popsanych 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 nemusí 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 přidáním 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`) spuštěného 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*

```
1 jshell> /list
2
3   1 : 6+5
4   2 : 6 +
5     7+
6     9
7   3 : $1+$2
8   4 : $1+$3;
9   5 : $1+$4
10
11 jshell> /drop 2
12 | dropped variable $2
13
14 jshell> /list
15
16   1 : 6+5
17   3 : $1+$2
18   4 : $1+$3;
19   5 : $1+$4
20
21 jshell>
```

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

```
1 jshell> /list -all
2
3   s1 : import java.io.*;
4   s2 : import java.math.*;
5   s3 : import java.net.*;
6   s4 : import java.nio.file.*;
7   s5 : import java.util.*;
8   s6 : import java.util.concurrent.*;
9   s7 : import java.util.function.*;
10  s8 : import java.util.prefs.*;
11  s9 : import java.util.regex.*;
12  s10 : import java.util.stream.*;
13   1 : 6+5
14   2 : 6 +
15     7+
16     9
17   3 : $1+$2
18  e1 : $1+$3 $1+$4
19   4 : $1+$3;
20   5 : $1+$4
21
22 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 počkat⁴. 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

```

1 jshell> /vars
2 |   int $1 = 11
3 |   int $3 = 33
4 |   int $4 = 44
5 |   int $5 = 55
6
7 jshell> /vars -all
8 |   int $1 = 11
9 |   int $2 = (not-active)
10 |  int $3 = 33
11 |  int $4 = 44
12 |  int $5 = 55
13
14 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 načteli nějaký soubor, v němž je připraven užitečný kód, který v dané kapitole použiju. Budete-li chtít zkusit 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:

- mezer (`'\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

```

1 jshell> //Řádkový komentář
2
3 jshell> /* Několikařádkový blokový komentář
4   ...> /* Vložení nové otevírací závorky neodstartuje vnořený komentář
5   ...> vše se ukončí zapsáním zavírací závorky. */
6
7 jshell> $1 + $5 //Komentář jako součást úryvku
8 $6 ==> 66
9 | created scratch variable $6 : int
10
11 jshell> /list
12
13 1 : 6+5
14 3 : $1+$2
15 4 : $1+$3;
16 5 : $1+$4
17 6 : $1 + $5 //Komentář jako součást úryvku
18
19 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 zkratk, 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

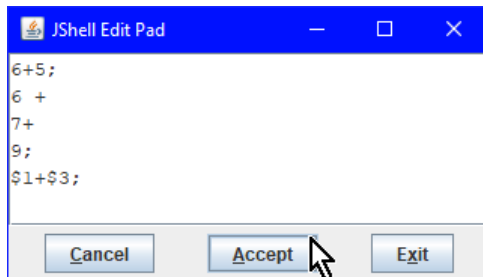
```
/edit <ID>
```

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 změnil 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.

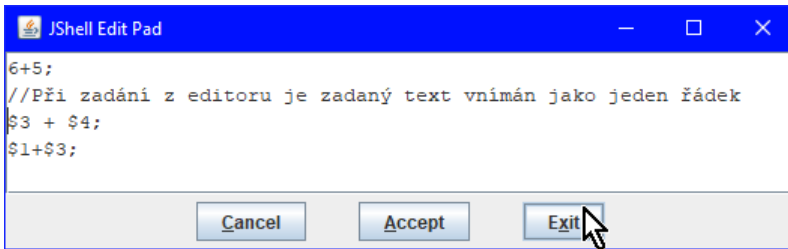
- Tlačítko **Accept**, které na obrázku [1.2](#) stiskává myš, použijete ve chvíli, kdy si nejste jisti tím, že vaše zadání je bezchybné a chcete ho nejprve prověřit. Vaše zadání se pak předá k vyhodnocení, vy si v konzolovém okně můžete prohlédnout výsledek a pokračovat v editaci.

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

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 obdrženy 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

Případá-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`.