

1001 TIPŮ A TRIKŮ PRO

jazyk Java

Bogdan Kiszka



CD obsahuje
všechny zdrojové
kódy z knihy a instalace
vývojových prostředí
Eclipse a NetBeans



Nejužitečnější programátorské fenty

- ▲ Návodů pro 2D grafiku, ovládací prvky, dialogy, tisk
- ▲ Tvorba dokonalého uživatelského rozhraní
- ▲ Recepty pro práci s multimédií
- ▲ Aplikace pro mobilní zařízení
- ▲ Nové možnosti JDK 6

C P R E S S

Bogdan Kiszka

1001 tipů a triků pro jazyk Java

**Computer Press
Brno
2012**

1001 tipů a triků pro jazyk Java

Bogdan Kiszka

Obálka: Martin Sodomka

Odpovědný redaktor: Martin Herodek

Technický redaktor: Jiří Matoušek

Objednávky knih:

<http://knihy.cpress.cz>

www.albatrosmedia.cz

eshop@albatrosmedia.cz

bezplatná linka 800 555 513

ISBN 978-80-251-2467-3

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 16 509.

© Albatros Media a. s. Všechna práva vyhrazena. Žádná část této publikace nesmí být kopírována a rozmnožována za účelem rozšiřování v jakékoli formě či jakýmkoli způsobem bez písemného souhlasu vydavatele.

Dotisk 1. vydání


ALBATROS MEDIA a.s.

Stručný obsah

Úvodem	33
Vznik prvního objektu a prvního programu	39
Práce s třídami a objekty	47
Dialogy a formuláře	69
Fokus alias ohnisko uživatelského vstupu	107
Písma použitá v aplikaci	117
Ovládací prvky, jejich rozvržení a změny vzhledu	119
Obsluha událostí a posluchači	143
Kreslení 2D	151
Práce s rastrovými obrázky	163
Práce s obrazovkou	185
Možnosti zobrazení textu	191
Formátování textu	199
Hledání v textu	209
Regulární výrazy	213
Převody a kódování řetězců	233
Lokalizace uživatelského rozhraní	243
XML	249
Manipulace se soubory	273
Nová rozhraní pro vstup a výstup	291
Komprese a dekomprese	303
Internet	307
Pracujeme na straně serveru	317
Sokety	345
Distribované systémy	355
Správa paměti	365
Souběžné zpracování	371

Tisk	379
Soubory zásad a správce zabezpečení	391
Certifikáty, digitální podpisy, elektronické klíče a šifrování	399
Přehrávání zvuků, zvukových souborů a sekvencí MIDI	431
Java Media Framework (JMF)	447
Ovladače JDBC	457
Databáze a práce s daty	459
Databáze a výsledné datové sady	471
Java ME a NetBeans	481
Midlety	489
Databáze v mobilním zařízení	505
Mobile Media API (JSR 135 API)	509
Konfigurace a instalace mobilních aplikací	513
Bezdrátové technologie	517
Rejstřík	527

Obsah

Úvodem 33

Jak číst tuto knihu	37
Zdrojové kódy	38
Doprovodné CD	38
Konvence použité v knize	38

Vznik prvního objektu a prvního programu 39

1	Co je to třída?	39
2	Nejjednodušší nová třída	39
3	Třída s vlastními daty	39
4	Manipulace s objekty a s jejich daty	40
5	Určení jedinečnosti metody	40
6	Modifikátory viditelnosti	41
7	Překrývání implicitní viditelnosti členu třídy	41
8	Klíčové slovo static	41
9	Nejjednodušší třída jako samostatný program	42
10	Jak program uložit	42
11	Překlad programu	42
12	Jak spustit připravený program v jazyce Java	43
13	Nejjednodušší odezva prvního programu	43
14	Jednoduchá aplikace s argumenty	43
15	Okamžité ukončení aplikace	44
16	Jak zjistit, že se aplikace chystá ukončit svůj běh	44
17	Komentáře v kódu	44
18	S objekty se manipuluje pomocí odkazů	45
19	Všechny objekty musíte vytvořit	45
20	Objekty nemusíte mazat	46
21	Doba platnosti objektů	46
22	Mazání objektů z paměti	46
23	Explicitní ukončení platnosti objektu	46

Práce s třídami a objekty 47

24	Přirozený jazyk v pojmenování	47
25	Čeština v identifikátorech	47

26	Konvence pojmenování	47
27	Význam názvů identifikátorů	47
28	Co je při volbě názvu nejdůležitější	48
29	Rozlišujte mezi názvy typů a objektů	48
30	Nepoužívejte v názvech číslice	48
31	Nepoužívejte názvy bez souvislosti s obsahem třídy nebo objektu	48
32	Styl pojmenování v jazyce Java	49
33	Návrhové vzory pojmenování	49
34	Viditelnost a jedinečnost názvů	49
35	Umístění nových tříd v balíčcích	50
36	Informace o typu objektu 1	50
37	Informace o typu objektu 2	50
38	Informace o typu objektu 3	50
39	Název třídy objektu	50
40	Odkud byla načtena třída	51
41	Umístění tříd načtených pomocí systémového zaváděče	52
42	Zjištění viditelnosti objektů libovolného typu	52
43	Zjištění viditelnosti datové složky	52
44	Zjištění předka testovaného typu	53
45	Jak zjistit, zda má třída předka?	53
46	Je to třída nebo rozhraní?	53
47	Výpis bazových rozhraní daného rozhraní	54
48	Seznam rozhraní implementovaných danou třídou	54
49	Jak zjistit umístění třídy v balíčku	54
50	Zjišťování názvů členských objektů	55
51	Nalezení metod pomocí objektů typu Method	55
52	Volání metod pomocí objektů typu Method	56
53	Jak lze pomocí objektu typu Class získat datové složky libovolného typu?	56
54	Nastavení nebo načtení hodnoty datové složky pomocí objektu typu Field	56
55	Zjištění informací o konstruktorech třídy	57
56	Tvorba nového objektu pomocí instance typu Constructor	57
57	Zaručená inicializace datových složek objektu	57
58	Parametrizovaná inicializace objektu	58
59	Implicitní a explicitní konstruktory	59
60	Konstruktor a argumenty	59
61	Konstruktor v jazyce Java	59
62	Konstruktory a metody. Je to totéž?	60
63	Konstruktor nesmí mít žádný návratový typ	60
64	Proč nelze deklarovat konstruktor jako konečný?	60
65	Zaručená inicializace pomocí konstruktoru	60

66	Volání konstruktorů z konstruktorů	61
67	Metoda finalize()	62
68	Úklid pomocí metody finalize()	62
69	Metoda finalize() není destruktorka	62
70	V jazyce Java nejsou destruktory	63
71	Objekty nemusí být vymazány z paměti	63
72	Singleton – jediný objekt daného typu v celém programu	63
73	Zpřístupnění tříd – import	64
74	Vlastnosti a přístupové metody	64
75	Klíčové slovo final	65
76	Klíčové slovo return	65
77	Překrývání metod předka	65
78	Přetížené metody	65
79	Rozdíl mezi proměnnou CLASSPATH a příkazem import	65
80	Načtení třídy neuvedené v proměnné CLASSPATH	65
81	Indexované vlastnosti	66
82	Přiřazení hodnoty instanci typu Object	66
83	Tvorba duplikátů	67
84	Zapouzdření primitivního typu do objektově orientovaného reprezentanta	67
85	Změna typu hodnoty	68
86	Přetypování datových typů	68
87	Přetypování primitivních typů	68
88	Zvláštnost datového typu Boolean	68
	Dialogy a formuláře	69
89	Návrh dialogů a formulářů v IDE	69
90	Hlavní okna	72
91	Rozdíl mezi třídami Frame a Canvas	73
92	Tvorba a zobrazení oken	73
93	Zavření okna	74
94	Ukončení aplikace pomocí systémového tlačítka Zavřít	74
95	Ukrytí hlavního okna pomocí systémového tlačítka Zavřít	75
96	Zobrazení okna uprostřed obrazovky	75
97	Nastavení maximálních rozměrů okna	75
98	Zákaz změn rozměrů hlavního okna aplikace	76
99	Změna ikony okna	76
100	Změna ikony okna (2)	76
101	Odstranění záhlaví okna	77
102	Exkluzivní celoobrazovkový režim	77
103	Celoobrazovkový režim pomocí běžných oken	78

104	Celoobrazovkový režim a změna rozměrů okna	78
105	Oprávnění pro celoobrazovkový režim apletů	78
106	Okno na celou obrazovku bez okrajů a titulku	78
107	Okno na celou obrazovku	78
108	Diagnostika možností průhledných a tvarovaných oken	78
109	Průhledná a poloprůhledná okna	79
110	Okna s průhledným obsahem	81
111	Okna různých tvarů	82
112	Okno s odleskem	84
113	Minimalizace a maximalizace hlavního okna aplikace	89
114	Jak určit, zda je okno minimalizováno nebo maximalizováno	91
115	Jak určit, zda je okno otevřeno nebo zavřeno	91
116	Jak používat předdefinované dialogy	92
117	Jednoduché předdefinované dialogy	92
118	Informační dialog s vlastní ikonou	93
119	Vlastní text na tlačítkách předdefinovaných dialogů	94
120	Dialog, kdy uživatel musí vybrat některou z možností	95
121	Nemodální dialog	97
122	Modalita dialogů	97
123	Implementace dialogů s různým rozsahem modality	98
124	Uživatelský vstup získaný z dialogu	99
125	Změna titulku dialogu pomocí komponenty TitleBorder	100
126	Výběr souboru pomocí standardního dialogu	100
127	Výběr adresáře pomocí standardního dialogu	100
128	MDI – vnitřní okna	101
129	Pravidla užívání vnitřních oken v rozhraní MDI	102
130	Obsluha událostí v dokumentech aplikace MDI	103
131	Manipulace se všemi okny aplikace	104
132	Rychlejší přetahování oken v aplikacích MDI	105
133	Minimalizace blikání při překreslování (1)	105
134	Minimalizace blikání při překreslování (2)	105

Fokus alias ohnisko uživatelského vstupu **107**

135	Definice kláves pro změnu ohniska vstupu v celé aplikaci	107
136	Klávesy pro přesun ohniska uživatelského vstupu	108
137	Ověřování textového pole při přesunu ohniska vstupu na jinou komponentu	108
138	Jak odebrat ohnisko vstupu aktuální aplikaci	109
139	Pořadí komponent uvnitř okna	109
140	Přesun výběru na další nebo předchozí komponentu I.	110
141	Přesun výběru na další nebo předchozí komponentu II.	111

142	Sledování změny výběru komponent v celé aplikaci	112
143	Výběr komponenty ihned po zobrazení okna	113
144	Změna ohniska uživatelského vstupu	114
145	Zákaz výběru objektu	114
146	Zákaz výběru okna	114
147	Určujeme komponentu, na kterou bude přesunuto ohnisko vstupu	115
148	Určujeme vybraný objekt nebo vybrané okno	115
149	Určujeme, zda je ztráta ohniska vstupu dočasná nebo trvalá	116
150	Nastavení typu ukazatele myši pro vybranou komponentu	116

Písma použitá v aplikaci 117

151	Seznam všech dostupných příbuzných písem	117
152	Více písem v aplikaci	117
153	Zobrazení odstavce textu	117
154	Zobrazení textu různými styly	118
155	Výpočet šířky textu v pixelech před jeho zobrazením	118
156	Tvar grafického objektu na základě obrysu textu	118

Ovládací prvky, jejich rozvržení a změny vzhledu 119

157	Elegantní rozvržení textových polí s popisky – rozvržení typu <code>SpringLayout</code>	119
158	Dialog s kartami – rozvržení typu <code>CardLayout</code>	121
159	Dialog Najít – rozvržení typu <code>GroupLayout</code>	122
160	Umístění grafických prvků v mřížce – rozvržení typu <code>GridBagLayout</code>	124
161	Zobrazení komponent s menší zobrazovací plochou než jsou možnosti příslušného okna	125
162	Jak používat motivy v Javě	125
163	Implicitní motiv uživatelského rozhraní	126
164	Nastavení motivu uživatelského rozhraní na příkazovém řádku	126
165	Nastavení motivu uživatelského rozhraní pomocí souboru vlastností	126
166	Nastavení motivu uživatelského rozhraní v kódu	126
167	Úprava vlastností motivu	127
168	Integrace vzhledu aplikací do hostitelského operačního systému	128
169	Změna motivu po spuštění aplikace	129
170	Změna systémových vlastností	129
171	Změna implicitního písma aplikace	130
172	Změna barvy ovládacích prvků ve vybraném vzhledu	131
173	Počestění standardních dialogů	131
174	Počestění standardních dialogů (2)	132
175	Změna vzhledu, zobrazení a hasnutí plovoucích popisků	133
176	Vlastní motivy grafického uživatelského rozhraní – Synth	133
177	Vlastní motivy grafického uživatelského rozhraní – Synth deklarativně	134

178	Nesměšujte instance tříd knihovny AWT s instancemi tříd knihovny Swing	136
179	Rozdíly mezi knihovnami AWT a Swing	136
180	Vykreslování "lehkých" komponent knihovny Swing	136
181	Kontejnery nejvyšší úrovně	137
182	Vodící linky ve stromové struktuře	137
183	Zákaz klepnutí pravým tlačítkem ve stromové struktuře	137
184	Podpora ikony v oznamovací oblasti hlavního panelu	138
185	Přístup do oznamovací oblasti	138
186	Práce s oznamovací oblastí (třída SystemTray)	138
187	Implementace ikon v oznamovací oblasti	140
188	Změna ikony v oznamovací oblasti	140
189	Změna popisku ikony v oznamovací oblasti	140
190	Změna rozměrů obrázku použitého v oznamovací oblasti	141
191	Bublinová nápověda v oznamovací oblasti	141

Obsluha událostí a posluchači **143**

192	Systém zasílání zpráv v jazyce Java	143
193	Obsluha událostí tlačítek a nabídek	144
194	Obsluha klepnutí tlačítka myši	145
195	Obsluha pohybu ukazatele myši	145
196	Obsluha stisku kláves	145
197	Obsluha událostí pomocí anonymní třídy	146
198	Jak rozpoznat klepnutí myši, poklepání nebo "trojklik"?	146
199	Čekání na událost typu PropertyChange	147
200	Čekání na změnu vlastnosti, která má právo změnu odmítnout	147
201	Úklid posluchačů z paměti	148
202	Generování klepnutí tlačítka myši	148
203	Přesun ukazatele myši na obrazovce	148
204	Simulace stisku klávesy nebo tlačítka myši	148
205	Způsob určení viditelnosti, přesunutí nebo změny rozměrů komponenty	149

Kreslení 2D **151**

206	Nejjednodušší kreslení na povrch okna	151
207	Kreslení primitivních tvarů – bod	152
208	Vzdálenost mezi dvěma body	152
209	Kreslení primitivních tvarů – úsečka	152
210	Kreslení primitivních tvarů – kvadratická křivka	153
211	Kreslení primitivních tvarů – kubická křivka	153
212	Kreslení primitivních tvarů – obdélník	153
213	Kreslení primitivních tvarů – elipsa	154

214	Kreslení primitivních tvarů – oblouk	154
215	Změna tloušťky pera	154
216	Definice okrajů primitivního grafického objektu	155
217	Kombinujeme různé tvary	155
218	Kreslení a vyplňování objektů	155
219	Kreslení barevného přechodu	156
220	Kreslíme kruhový diagram (graf)	157
221	Nastavení barvy	159
222	Výřez nakresleného tvaru	159
223	Výřez vymezený textem	160
224	Převod textového názvu barvy na odpovídající hodnotu	161
225	Tvorba nových tvarů pomocí čar a křivek	161
226	Úprava měřítka, oseknutí, překlopení nebo otočení tvaru	162

Práce s rastrovými obrázky

163

227	Obrázek ve stupních šedé	163
228	Převod barevného obrázku v paměti na stupně šedé	163
229	Zaostření obrázku	163
230	Rozmazání obrázku	164
231	Tvorba reliéfu z rastrového obrázku	164
232	Úprava obrázku: Měřítka, ořezání, překlopení, otočení.	164
233	Jak zesvětlit nebo ztmavit obrázek	165
234	Průhledné pozadí obrázku	165
235	Dotaz na množství volné zrychlené paměti použitelné pro práci s obrázkem	166
236	Filtrování složek RGB v obrázku	166
237	Komprese souboru JPEG	167
238	Kreslení obrázku v paměti	168
239	Tvorba a kreslení obrázků v paměti	169
240	Úprava měřítka, oseknutí, překlopení a otočení obrázku v paměti	171
241	Práce s pixely v obrázku načteném do paměti	171
242	Průhledné pixely v obrázku	172
243	Zjištění hodnoty průhledného pixelu nebo počtu barev použitých v obrázku ve formátu GIF	172
244	Překlopení obrázku	173
245	Převod objektu typu BufferedImage na objekt typu Image	173
246	Převod objektu typu Image na objekt typu BufferedImage	173
247	Načtení barevného modelu obrázku	175
248	Třídy a metody pro práci s barevnými modely	175
249	Načtení obrázku nebo ikony ze souboru	176
250	Načtení obrázku ze souboru, vstupního proudu nebo z adresy URL	176
251	Načtení vybrané části obrázku	177

252	Oložení generované grafiky do souboru ve formátu PNG nebo JPEG	177
253	Výpis všech grafických formátů, které lze načítat a do nichž lze ukládat	178
254	Určení formátu obrázku v souboru	179
255	Způsob zjištění, zda lze formát obrázku číst nebo použít k zápisu	180
256	Rozpohybování pole obrázků	182
257	Zachycení snímku obrazovky	183

Práce s obrazovkou **185**

258	Rozlišení obrazovky	185
259	Rozměry obrazovky	185
260	Načtení dostupných rozměrů obrazovky, obnovovací frekvence a kvality (počtu) barev	185
261	Načtení rozměrů obrazovky	185
262	Způsob zjištění aktuální obnovovací frekvence a kvality barev obrazovky	186
263	Nastavení rozměrů obrazovky, obnovovací frekvence a kvality barev	186
264	Počet dostupných obrazovek	187
265	Lepší výkon v režimu celé obrazovky	188

Možnosti zobrazení textu **191**

266	Vykreslení prostého textu	191
267	Přirozenější proložení znaků	191
268	Podtržení textu	192
269	Přeškrtnutí textu	192
270	Změna barvy textu	193
271	Změna atributu textu u části textu	193
272	Rozměry textu v nakresleném objektu	194
273	Změna orientace textu v nakreslených objektech	195
274	Změna orientace textu v nakreslených objektech	195
275	Změna velikosti a typu písma v nakreslených objektech	196
276	Paprskovité zobrazení textu	197

Formátování textu **199**

277	Formátování a zpracování času	199
278	Formátování a zpracování času v češtině	199
279	Vlastní formátovací vzory pro čas	200
280	Formátování zpráv obsahujících čas	201
281	Formátování a zpracování kalendářních dat	202
282	Vlastní formáty kalendářních dat	202
283	Formátování zpráv obsahující datum	203
284	Formátovací vzory pro kalendářní data	204

285	Formátování zpráv obsahujících čísla	204
286	Formátování čísel v souladu s místním nastavením	205
287	Formátování měny v souladu s místním nastavením	205
288	Formátování procent v souladu s místním nastavením	206
289	Formátování čísel v exponenciálním zápisu	206
290	Minimální počet číslic vlevo i vpravo od desetinné čárky v exponenciálním formátu	207
291	Formátování čísel vlastním formátem	207
292	Chyby při užívání modifikačních metod třídy String	208

Hledání v textu

209

293	Hledání podřetězce v řetězci	209
294	Hledání podřetězce v řetězci	209
295	Hledání zalomení řádků v řetězcích Unicode	209
296	Hledání znaku nebo podřetězce v řetězci	210
297	Nalezení znaku, části slova nebo slovního spojení	210
298	Nahrazení znaku, části slova nebo slovního spojení	210
299	Hledání a nahrazení textu	210
300	Porovnání řetězců bez ohledu na místní a jazykové nastavení	211
301	Porovnávání textových řetězců	211
302	Jak ověřit pořadí řetězců?	211
303	Jak ověřit pořadí dvou řetězců bez ohledu na velikost písmen?	212
304	Porovnání řetězce s objektem typu StringBuffer	212
305	Procházení textu po znacích	212

Regulární výrazy

213

306	Vyhledávání pomocí regulárních výrazů	213
307	Ukázka hledání pomocí regulárních výrazů	213
308	Funkce „Najít a nahradit“ pomocí regulárních výrazů	214
309	Nahrazení textu proměnnými řetězci	214
310	Zachycení skupin znaků pomocí regulárního výrazu	215
311	Užití zachyceného textu ve vzoru regulárního výrazu	216
312	Užití zachyceného textu ve vzoru nahrazujícího textu	217
313	Aplikace regulárních výrazů na obsah souboru	217
314	Převod řetězce na tokeny	218
315	Zpracování dat oddělovaných určitým znakem	219
316	Rozdělení textu na odstavce pomocí regulárního výrazu	220
317	Čtení odstavců pomocí regulárních výrazů	220
318	Čtení řádků pomocí regulárního výrazu	221
319	Filtrování řádků ze vstupního proudu	221
320	Filtrování vstupu pomocí regulárních výrazů	222

321	Indexování textu	224
322	Komentáře v regulárním výrazu	225
323	Nalezení konců řádku pomocí regulárního výrazu	226
324	Nalezení textu bez ohledu na konce řádků	226
325	„Nenasyté“ vyhledávání pomocí regulárních výrazů	227
326	Shoduje se nalezený řetězec se vzorem?	228
327	Odstranění konce řádku z objektu typu String	228
328	Odstranění zdvojených mezer	228
329	Překlad regulárního výrazu s více příznaky	229
330	Skupiny vzorů, které nezachycují text	229
331	Záměny řídicích znaků v regulárních výrazech	230
332	Změna v rozlišování malých a velkých písmen	230

Převody a kódování řetězců 233

333	Určování typu znaku	233
334	Převod řetězce na velká nebo malá písmena	233
335	Převod znaků mezi kódováním Unicode a UTF-8	233
336	Znakové bloky v řetězcích Unicode	234
337	Převod čísel na textové řetězce	234
338	Převod hodnoty primitivního datového typu na řetězec	234
339	Rozklad textu na jednotlivá slova	235
340	Stanovení hranic slova v řetězci Unicode	235
341	Stanovení hranic věty v řetězci Unicode	235
342	Stanovení hranice znaku v řetězci Unicode	236
343	Načtení seznamu hodnot oddělených tabulátory	236
344	Rozklad zdrojového kódu v Javě na tokeny	236
345	Je řetězec platným identifikátorem v jazyce Java?	238
346	Sestavení řetězce	238
347	Ukládání řetězců v objektu typu ByteBuffer	239
348	Načítání dat v požadovaném kódování	240
349	Převod bajtového pole na řetězec v kódování Base64	240
350	Převod dat kódovaných pro přenosy v síti WWW	240
351	Zápis dat ve vybraném kódování	241
352	Výpis všech dostupných převaděčů mezi kódováním Unicode a jinými znakovými sadami	241
353	Převody mezi Unicode a jinými znakovými sadami	242

Lokalizace uživatelského rozhraní 243

354	Výpis všech dostupných místních nastavení	243
355	Tvorba národního prostředí	243
356	Lokalizace uživatelského rozhraní	243

357	Priorita zdrojových souborů jazykového nastavení	245
358	Priorita jazykového nastavení aplikace	245
359	Jazykové nastavení aplikace	245
360	Znaky Unicode v souborech jazykových prostředků	246
361	Zpětný převod z kódování ASCII do UNICODE	246
362	Kódování souborů jazykových prostředků	246
363	Zjištění implicitního jazykového nastavení	247
364	Vkládání znaků rozšířené abecedy do instance třídy JTextField	247

XML

249

365	Standard kódování souborů XML	249
366	Tvorba modelu DOM ze souboru XML	249
367	Dotaz na prvek modelu DOM na základě ID	250
368	Dotaz na kořenový prvek dokumentu DOM	250
369	Editace textu v uzlech typu CDATA, Comment a Text	251
370	Hodnota znakové entity v modelu DOM	252
371	Jak získat z objektu typu DOM Document pouze text	252
372	Komentáře v dokumentu DOM	253
373	Kopírování podstromu uzlů v modelu DOM	253
374	Kopírování podstromu uzlů z jednoho objektu typu Document modelu DOM do jiného	254
375	Načtení a změna atributu prvku v dokumentu DOM	254
376	Nový uzel v dokumentu DOM	255
377	Oddíl CDATA v dokumentu DOM	256
378	Odebrání uzlu z dokumentu DOM	257
379	Odstanění všech atributů vybraného prvku v dokumentu DOM	258
380	Procházení uzlů v dokumentu DOM	258
381	Procházení uzlů v objektu Document modelu DOM	258
382	Přidání textového uzlu do dokumentu DOM	259
383	Přidávání a odebrání atributů prvků v modelu DOM	259
384	Přidávání instrukcí zpracování do dokumentu DOM	260
385	Relativní dotazy na uzly v objektech typu Document modelu DOM	261
386	Rozdělení textového uzlu v dokumentu DOM	261
387	Slučování textových uzlů v dokumentu DOM	262
388	Tvorba prázdného dokumentu modelu DOM	263
389	Ukládání objektů typu DOM Document do souboru XML	263
390	Výpis všech atributů prvku v dokumentu DOM	263
391	Vytvoření deklarace DOCTYPE při ukládání souboru XML	264
392	Změna názvu prvku v dokumentu DOM	264
393	Převod fragmentu XML na fragment DOM	265
394	Ignorujeme komentáře v souboru XML	266

395	Jak zabránit rozvinutí znakových entit při zpracování souboru XML	266
396	Převod uzlů CDATA na textové uzly	267
397	Explicitní nebo implicitní atribut prvku	267
398	Ošetření chyb při zpracování souboru XML	267
399	Pracujeme s jazykem XPath	268
400	XPath: Dotaz na kořenový prvek	269
401	XPath: Základní dotazy	269
402	XPath: Pořadí načítaných prvků	270
403	XPath: Základní dotazy na atributy prvků	271
404	XPath: Rozlišování velkých a malých písmen	271
405	Transformace dokumentu XML pomocí stylů XSL	272
406	Transformace dokumentu XML pomocí stylu XSL do objektu typu DOM Document	272

Manipulace se soubory **273**

407	Výpis souborů v adresáři	273
408	Obsah adresáře	273
409	Tvorba souboru	274
410	Tvorba nových souborů	274
411	Výpis kořenového adresáře	274
412	Odkazují se dvě cesty na stejný soubor?	274
413	Výpis všech kořenových adresářů	275
414	Aktuální pracovní adresář	275
415	Nadřazené adresáře v souborové cestě	275
416	Soubor nebo na adresář?	276
417	Dotaz na existenci souboru nebo adresáře	276
418	Seznam souborů a podadresářů	276
419	Filtrovaný seznam souborů a podadresářů	277
420	Seznam souborů jako objektů File	277
421	Seznam podadresářů	277
422	Procházení souborů a podadresářů	277
423	Zpracování všech podadresářů	278
424	Zpracování všech souborů ve všech podadresářích	278
425	Tvorba dočasných souborů	278
426	Odstranění souboru ze souborového systému	279
427	Odstranění souboru ze souborového systému	279
428	Velikost souboru	279
429	Čtení nebo změna značky poslední změny souboru či adresáře	279
430	Tvorba adresáře	280
431	Odstranění adresáře	280
432	Odstranění prázdného adresáře	280

433	Odstranění adresáře, který není prázdný	280
434	Přejmenování souboru nebo adresáře	281
435	Přesunutí souboru nebo adresáře	281
436	Kopírované soubory s obrázky jsou poškozeny	282
437	Relativní souborové cesty	282
438	Sestavení souborové cesty	282
439	Převody mezi souborovou cestou a adresou URL	282
440	Převody adresy URL na souborovou cestu	283
441	Přesměrování výstupu metody <code>println()</code> do souboru	283
442	Přesměrování standardního a chybového výstupu	283
443	Přesměrování výstupního proudu <code>System.err</code> do souboru	284
444	Načtení textu ze standardního vstupu	284
445	Čtení textu ze souboru	285
446	Zápis textu do souboru	285
447	Připojení textu k existujícímu souboru	285
448	Odstranění obsahu textového souboru	285
449	Načtení obsahu souboru do bajtového pole	286
450	Práce se soubory v režimu náhodného čtení a zápisu	286
451	Rozdělení souboru	287
452	Vynucení aktualizace souboru na pevném disku	288
453	Serializace objektů a jejich členských proměnných	288
454	Uložení objektu do souboru	289

Nová rozhraní pro vstup a výstup

291

455	Co je to NIO?	291
456	Tvorba datového proudu ze souborového kanálu	291
457	Zápis do souborového kanálu	292
458	Čtení ze souborového kanálu	292
459	Kopírování obsahu jednoho souboru do jiného	293
460	Tvorba souborového zámku	293
461	Tvorba sdíleného souborového zámku	294
462	Tvorba souboru mapovaného do paměti pro čtení	294
463	Tvorba souboru mapovaného do paměti pro čtení a zápis	294
464	Tvorba soukromého souboru mapovaného do paměti	295
465	Tvorba bajtové vyrovnávací paměti	295
466	Převod mezi objekty typu <code>ByteBuffer</code> a <code>byte</code>	295
467	Tvorba pohledů na typ <code>ByteBuffer</code>	296
468	Ukládání dat z objektů typu <code>ByteBuffer</code> do souboru	296
469	Dotaz na kapacitu vyrovnávací paměti	297
470	Dotaz na aktuální pozici v objektu <code>ByteBuffer</code>	297

471	Vkládání dat do objektů typu ByteBuffer	297
472	Nebajtové typy v objektech typu ByteBuffer	298
473	Dotaz na bajty z objektů typu ByteBuffer	298
474	Pořadí bajtů v objektu typu ByteBuffer	299
475	Pracujeme s přímou vyrovnávací pamětí	299
476	Trvalé změny v mapované vyrovnávací paměti	300
477	Tvorba výstupního proudu z vyrovnávací paměti	300
478	Tvorba vstupního proudu z vyrovnávací paměti	301

Komprese a dekomprese 303

479	Tvorba archivu ZIP	303
480	Komprese pole bajtů	303
481	Komprese souboru do formátu GZIP	304
482	Dekomprese pole bajtů	304
483	Extrahování souboru ve formátu GZIP	305
484	Extrahování souboru z archivu ZIP	305
485	Načtení obsahu archivu ZIP	306
486	Výpočet kontrolního součtu pro pole bajtů	306
487	Výpočet kontrolního součtu pro soubor	306

Internet 307

488	Reprezentace adres IP v jazyce Java	307
489	Adresa IP a název hostitele lokálního počítače	307
490	Adresa IP vybraného hostitele	307
491	Název hostitele dané adresy IP	308
492	Testování síťových aplikací bez připojení k síti	308
493	Tvorba objektu typu URL	308
494	Zpracování adresy URL	308
495	Adresa URL s odkazem na aktivní místo v dokumentu	309
496	Dotazy na archiv JAR prostřednictvím objektu typu URL	309
497	Načítání záhlaví odpovědi z připojení HTTP	310
498	Načtení obrázku z adresy URL	310
499	Načtení textu adresy URL	310
500	Odeslání požadavku POST pomocí objektu typu URL	311
501	Práce s třídou URL	311
502	Přetypování z URL na URI	312
503	Přístup k adrese URL chráněné heslem	312
504	Zákaz automatického přesměrování	313
505	Načítání souborů cookie z připojení HTTP	314
506	Odeslání souboru cookie na server HTTP	315

	Pracujeme na straně serveru	317
507	JSP versus servlet	317
508	Rozsah platnosti na stránkách JSP	317
509	Dotaz na adresu klienta	317
510	Jednoduchá stránka JSP	317
511	Základní kostra servletu	318
512	Čeština na stránkách HTML z kontejneru JSP	319
513	Čeština na stránkách JSP	319
514	Čeština na stránkách JSP	319
515	Nastavení kódování parametrů požadavku HTTP	319
516	Překódování parametrů požadavku HTTP z Latin 1	320
517	Komentáře na stránkách JSP	320
518	Sdílení stránek JSP	320
519	Umístění servletů	320
520	Zahrnutí souboru na stránce JSP	321
521	Zahrnutí souboru na stránce JSP	321
522	Zahrnutí další stránky na stránce JSP	321
523	Předávání parametrů další stránce	322
524	Předávání požadavků HTTP na stránky JSP dalším stránkám	322
525	Spouštění kódu na stránkách JSP	322
526	Přesměrování na stránce JSP	323
527	Přenesení webového serveru	323
528	Parametry požadavku na stránce JSP	323
529	Chybová hlášení na stránkách JSP	324
530	Zákaz tvorby uživatelské relace	324
531	Dotaz na hlavičku požadavku	324
532	Dotaz na parametr požadavku	325
533	Dotaz na záhlaví požadavku HTTP	326
534	Podmíněný obsah na stránce JSP – příkaz if	327
535	Podmíněný obsah na stránce JSP – příkaz switch	327
536	Dynamický obsah na stránkách JSP	327
537	Nedovolte uživateli opakovaně odeslat stránku JSP	328
538	Zjištění adresy URL hostitele prostřednictvím servletu	328
539	Zjištění adresy URL hostitele prostřednictvím servletu	328
540	Zjištění adresy URI hostitele prostřednictvím servletu	329
541	Protokolování v servletu	329
542	Předkompilování stránky JSP	330
543	Odeslání obrázku pomocí servletu	330
544	Transakce na stránkách JSP	330
545	Tvorba dokumentu XML ze stránky JSP	331

546	Ukázka jednoduchého filtru	331
547	Ukládání dat na stránkách JSP	332
548	Uložení dat prostřednictvím servletu	332
549	Dotaz na všechny atributy s platností požadavku	333
550	Dotaz na všechny atributy s platností relace	333
551	Dotaz na všechny atributy s platností aplikace	333
552	Jednoduchý objekt typu JavaBeans	334
553	Objekt JavaBeans na stránce JSP	334
554	Vícislovné hodnoty na stránkách JSP z kódu	335
555	Vícislovné hodnoty na stránkách JSP deklarativně	336
556	Zamezte souběžným požadavkům na servlet	336
557	Inicializační parametry servletu v kódu	337
558	Inicializační parametry servletu deklarativně	337
559	Zpracování požadavku HEAD	338
560	Nastavení prostředí pro tvorbu vlastních značek JSTL	338
561	Deklarace knihoven JSTL na stránce JSP	339
562	Aplikace jazyka výrazů JSTL	339
563	Použití příkazů JSTL s formátovacími značkami HTML	339
564	Dotaz na parametr požadavku pomocí JSTL	340
565	Ukládání dat pomocí JSTL	340
566	Zobrazení dat uložených pomocí JSTL	341
567	Podmíněná tvorba výstupu s knihovnou JSTL: if	341
568	Podmíněná tvorba výstupu s knihovnou JSTL: choose	342
569	Jednoduchá uživatelsky definovaná značka	342
570	Způsob užití uživatelsky definované značky	343
571	Hodnota atributu URI v direktivě taglib	344

Sokety

345

572	Tvorba serverového soketu	345
573	Tvorba klientského soketu bez časového omezení	345
574	Tvorba klientského soketu s časovým limitem	345
575	Čtení textu z objektu typu Socket	346
576	Zápis textu do objektu typu Socket	346
577	Odeslání datagramu	346
578	Příjem datagramu	347
579	Příjem dat ve skupině vícesměrového vysílání	347
580	Odesílání dat skupině odběratelů pomocí vícesměrového rozesílání	347
581	Připojení ke skupině vícesměrového vysílání	348
582	Odeslání požadavku POST prostřednictvím soketu	348
583	Problémy s třídou Socket	348

584	Tvorba neblokujícího serverového socketu	349
585	Tvorba neblokujícího socketu	349
586	Čtení dat ze socketového kanálu	349
587	Zápis do socketového kanálu	350
588	Čekání na připojení pomocí serverového socketového kanálu	350
589	Řízení neblokujících serverových socketů pomocí objektů typu Selector	351
590	Řízení neblokujících socketů pomocí objektů typu Selector	351
591	Ošetření síťové komunikace pomocí neblokujících socketových kanálů	352
592	Způsob zjištění, zda vzdálený počítač zavřel připojení	353
593	Tvorba klientského socketu SSL	353
594	Tvorba serverového socketu SSL	354

Distribuované systémy

355

595	Komunikace mezi dvěma stroji JVM na jednom počítači	355
596	Posílání odkazů prostřednictvím socketů	355
597	RMI versus sokety	355
598	Volání systémových funkcí pomocí skriptů jazyka Perl	355
599	Překlad kódu v jazyce IDL	356
600	Implementace objektu proxy	356
601	Tvorba objektu proxy	357
602	Spouštění registru RMI	357
603	Spouštění názvového serveru	357
604	Definice a export přenositelných vzdálených objektů	357
605	Definice a export vzdáleného objektu	358
606	Načtení návratové hodnoty vzdálené metody	358
607	Předávání argumentů vzdálené metodě	360
608	Vyhledání vzdáleného objektu a volání jeho metod	361
609	Vyhledání vzdáleného objektu a volání jeho metod	362
610	Vyvolávání výjimek ze vzdálených metod	362
611	Tvorba a mazání vnořeného kontextu názvové služby (Naming Service)	363
612	Tvorba počátečního kontextu pro názvovou službu (Naming Service)	363
613	Užití adresy URL jako názvu počátečního kontextu	363
614	Vyhledání objektu pomocí názvové služby	364
615	Přidávání, náhrada, odstraňování a přejmenování vazeb v názvové službě	364
616	Dotaz na úplný název objektu	364
617	Výpis obsahu názvové služby	364

Správa paměti

365

618	Mazání nepotřebných objektů	365
619	Platnost objektů	365

620	Je třeba určit, kdy bude objekt nepotřebný	366
621	Je třeba určit, kdy bude objekt vymazán z paměti	366
622	Explicitní uvolnění neplatných objektů z paměti	366
623	Explicitní uvolnění neplatných objektů z paměti	367
624	Explicitní uvolnění neplatných objektů z paměti	367
625	Dotaz na velikost dynamické paměti (haldy) v bajtech	367
626	Dotaz na volnou paměť v dynamické oblasti	367
627	Dotaz na maximální velikost haldy v bajtech	367
628	Dotaz na velikost obsazené paměti	367
629	Jak zachovat objekt, dokud je dostatek paměti	367
630	K čemu je vlastnictví objektu v systémové schránce?	368
631	Jak určit, zda je položka stále v systémové schránce	368
632	Třída pro vkládání obrázků do systémové schránky	368
633	Vkládání obrázků do systémové schránky	369
634	Výběr obrázků ze systémové schránky	369
635	Vložení textu do systémové schránky	370
636	Výběr textu ze systémové schránky	370

Souběžné zpracování

371

637	Tvorba pracovního vlákna odvozením od bazové třídy	371
638	Tvorba pracovního vlákna implementací rozhraní	371
639	Zastavení vlákna	372
640	Zachycení okamžiku ukončení běhu vlákna	372
641	Určení okamžiku, kdy dojde k ukončení běhu vlákna	373
642	Čekání na ukončení běhu vlákna	373
643	Pozastavení aktuálního vlákna	373
644	Nepoužívejte metody suspend() a resume()	373
645	Korektní pozastavení běhu vlákna	374
646	Implementace fronty pracovních vláken	374
647	Seskupování vláken	375
648	Jak vyhledat kořenovou skupinu vláken	375
649	Rekurzivní procházení všech vláken ve skupině	375
650	Výpis všech spuštěných vláken	376
651	Určujeme, zda vlákno drží synchronizační zámek	376
652	Ukončení aplikace se spuštěnými vlákny	377
653	Zcela bezpečná třída pro užití ve vláknech	377
654	Třída částečně bezpečná pro užití ve vláknech	378
655	Zastavení všech vláken po stisku Ctrl+C	378

Tisk	379
656 Tisk textu	379
657 Tisk do souboru	379
658 Tisk hlavního okna aplikace	379
659 Implementace rozhraní Printable	380
660 Kostra jednoduchého tiskového programu	380
661 Zobrazení dialogu Tisk	380
662 Zobrazení dialogu Vzhled stránky	381
663 Dotaz na rozměry celé stránky	381
664 Dotaz na rozměry tisknutelné oblasti	381
665 Nastavení počtu kopií tiskové úlohy	381
666 Nastavení orientace tištěné stránky	382
667 Nastavení orientace tištěné stránky	382
668 Tisk stránek v různých formátech	382
669 Dotaz na implicitní tiskovou službu	383
670 Nalezení všech dostupných tiskových služeb	383
671 Nalezení tiskových služeb s podporou určitého formátu	383
672 Nalezení tiskárny	383
673 Nalezení tiskárny s možností barevného tisku	383
674 Vyhledání tiskových služeb pro datové toky	384
675 Tiskové služby pro datové toky v určitém formátu	384
676 Tvorba tiskové služby pro datové toky	384
677 Atributy tiskové služby	384
678 Atributy tiskové úlohy podporované tiskovou službou	385
679 Čekání na změnu stavu tiskové úlohy	385
680 Čekání na změny atributu tiskové úlohy	386
681 Čekání na změny stavů tiskových služeb	386
682 Dotaz na implicitní hodnotu atributu tiskové úlohy	387
683 Dotaz na možné hodnoty atributu tiskové úlohy	387
684 Implementace hlídače dokončení tiskové úlohy	388
685 Způsob užití hlídače dokončení tiskové úlohy	388
686 Zrušení tiskové úlohy	389
687 Kostra programu využívajícího tiskové služby	390
688 Tisk datového proudu do souboru	390
 Soubory zásad a správce zabezpečení	 391
689 Používejte správce zabezpečení	391
690 Deklarativní aktivace správce zabezpečení	391
691 Konfigurace ochrany souborů	392

692	Kdy se zapíná správce zabezpečení	392
693	Udělování oprávnění a ochrana souborového systému	392
694	Nastavte vždy jen minimální oprávnění	393
695	Ochrana systémových vlastností pomocí oprávnění	394
696	Udělení oprávnění na základě podpisu autora	394
697	Udělení oprávnění více třídám na základě podpisu	395
698	Udělení kombinovaného oprávnění na základě podpisu	395
699	Adresy URL v souborech zásad	395
700	Kombinujte různé typy oprávnění	395
701	Jak zjistit provázanost oprávnění?	396
702	Rozvinutí systémových proměnných v souborech zásad	396
703	Správa souborů zásad	396
704	Tvorba nového souboru zásad	397
705	Dotaz na oprávnění pro adresu URL	398
706	Dotaz na oprávnění pro vybraný adresář	398
707	Výpis všech udělených oprávnění	398

Certifikáty, digitální podpisy, elektronické klíče a šifrování

399

708	Certifikát zabezpečení	399
709	Digitální podpis I.	399
710	Digitální podpis II.	399
711	Cesta k certifikátu příslušného serveru SSL	399
712	Dotaz na rozlišovací názvy předmětu a vydavatele certifikátu X509	400
713	Export certifikátu do souboru	401
714	Export certifikátu do souboru pomocí nástroje keytool	401
715	Import veřejného klíče ze souboru certifikátu	401
716	Import veřejného klíče ze souboru certifikátu	402
717	Import veřejného klíče ze souboru certifikátu do nového úložiště certifikátů	402
718	Jak upravit soubor zásad pro ověření identity ve vhodném úložišti certifikátů	402
719	Načtení certifikátu z úložiště certifikátů	403
720	Ověření validity certifikační cesty	403
721	Převod certifikátů X509 z typu javax na typ java	404
722	Převod certifikátů X509 z typu java na typ javax	404
723	Tvorba certifikační cesty	404
724	Přidání certifikátu do úložiště certifikátů	405
725	Výpis certifikátů nejdůvěryhodnějších certifikačních úřadů z úložiště certifikátů	405
726	Výpis všech aliasů v úložišti certifikátů	406
727	Výpis všech aliasů v úložišti certifikátů pomocí nástroje keytool	406

728	Výpis všech dostupných algoritmů pro ověřování validity certifikační cesty	406
729	Výpis všech dostupných formátů certifikátů	407
730	Načtení a uložení souboru v apletu	407
731	Veřejný a soukromý klíč	408
732	Dotaz na bajty generovaného páru klíčů	408
733	Dotaz na bajty generovaného symetrického klíče	409
734	Dotaz na parametry DSA pro dvojici klíčů	409
735	Dotaz na přihlašovací jméno aktuálně přihlášeného uživatele	410
736	Generování bezpečného náhodného čísla	411
737	Načtení dvojice klíčů z úložiště certifikátů	411
738	Jak ověřit oprávnění pro přístup k adresáři	411
739	Ověření podpisu bajtů ve vyrovnávací paměti	412
740	Ověření podepsaného objektu v jazyce Java	412
741	Konfigurace přihlášení k aplikaci	412
742	Ověřování uživatele při přihlášení k aplikaci	413
743	Řízení přístupu k objektu	413
744	Tisk trasovacích zpráv systému zabezpečení	414
745	Tvorba podpisu	414
746	Tvorba vlastního typu oprávnění	414
747	Příklady užití vlastního typu oprávnění	415
748	Výpis všech oprávnění udělených načtené třídě	416
749	Zákaz ověřování validity při připojení HTTPS	416
750	Tvorba páru šifrovacích klíčů s vlastním podpisem	417
751	Tvorba seznamu parametrů pro algoritmus Diffie-Hellman Key Agreement (DH)	417
752	Tvorba soukromého a veřejného klíče DH	418
753	Tvorba symetrického klíče DES	418
754	Tvorba symetrického klíče Blowfish	418
755	Tvorba symetrického klíče Triple DES	418
756	Tvorba symetrického klíče MAC pro algoritmus SHA1	419
757	Tvorba symetrického klíče MAC pro algoritmus MD5	419
758	Tvorba konspektu zprávy algoritmem MAC k ověření integrity zprávy	419
759	Tvorba šifrovaného výpisu zprávy algoritmem MD5	419
760	Tvorba soukromého a veřejného klíče RSA	420
761	Tvorba soukromého a veřejného klíče DSA	420
762	Tvorba klíčů na základě parametrů DSA	420
763	Generování tajného klíče pomocí algoritmu Diffie-Hellman Key Agreement (DH)	421
764	Šifrování objektů algoritmem DES	422
765	Třída pro šifrování algoritmem DES	422
766	Příklad užití třídy DesEncrypter	423
767	Třída pro šifrování algoritmem DES na základě hesla	423

768	Příklad užití třídy DesEncrypterPass	424
769	Šifrování souboru nebo proudu algoritmem DES	424
770	Příklad užití třídy DesFileEncrypter	425
771	Převod 56bitové hodnoty na klíč algoritmem DES	426
772	Výpis všech dostupných typů kryptografických služeb	427
773	Výpis všech implementací daného typu kryptografické služby	427
774	Výpis všech dostupných generátorů soukromých nebo veřejných klíčů	428
775	Výpis všech dostupných generátorů symetrických klíčů	428
776	Výpis všech dostupných podpisových algoritmů	428
777	Seznam všech dostupných bezpečných generátorů náhodných čísel	429
778	Seznam všech dostupných šifrovacích a dešifrovacích algoritmů	429
779	Výpis všech dostupných algoritmů pro šifrování zpráv	429

Přehrávání zvuků, zvukových souborů a sekvencí MIDI

431

780	Jednoduchý zvukový signál	431
781	Aplikační rozhraní Java Sound	431
782	JMF (Java Media Framework)	431
783	Digitálně vzorkovaná zvuková data v Javě	431
784	Formátovaná zvuková data	432
785	Datové formáty	432
786	Souborové formáty	432
787	Čtení a zápis zvuků	433
788	Program neskončí po ukončení metody main()	433
789	Diagnostika přítomnosti zvukového subsystému Java Sound	433
790	Načtení zvukového souboru z místního disku	433
791	Načtení zvukového souboru ze sítě	433
792	Jak načíst zvukový soubor bez hlavičky?	434
793	Formát zvukového souboru	434
794	Souborový formát zvukového souboru	434
795	Jak zjistit formáty podporované zvukovým systémem?	435
796	Úprava formátu zvukového souboru pro přehrávání v Javě	435
797	Přehrání zvukových souborů	436
798	Přehrávání souborů MP3 pomocí Java Sound API	436
799	Nepřetržité přehrávání zvukového souboru	436
800	Jak se dovíte, že jsou k dispozici další data pro zápis nebo čtení	437
801	Proč jsou metody getFramePosition() a getMicrosecondPosition() tak nepřesné?	437
802	Rozdíl mezi metodami isActive() a isRunning()	437
803	Nedostatek paměti při přehrávání zvukových souborů o velikosti větší než 5 MB	437

804	Postupné přehrávání zvukové stopy z datového proudu	438
805	Délka zvukové stopy	438
806	Délka zvukového souboru	438
807	Pozice zvukové stopy	439
808	Nastavení hlasitosti přehrávání zvukové stopy	439
809	Pouštění mono-proudu jen do jednoho stereo-kanálu	439
810	Událost ukončení přehrávání zvukového souboru	440
811	Co je to sekvence MIDI	440
812	Zvukové banky MIDI	441
813	Která zvuková banka je implicitní, když je jich v počítači více?	441
814	Načtení implicitní zvukové banky	441
815	Načtení zvukové banky z konkrétního souboru	441
816	Zjištění základních údajů o zvukové bance	442
817	Zobrazení seznamu nástrojů dostupných ve zvukové bance	442
818	Proč metoda <code>getAvailableInstruments()</code> vrací prázdné pole?	442
819	Chyba „MIDI OUT transmitter not available“	443
820	Načtení sekvence MIDI z místního disku	443
821	Načtení sekvence MIDI ze sítě	443
822	Souborový formát sekvence ve formátu MIDI	443
823	Přehrávání sekvence ve formátu MIDI	444
824	Přehrávání sekvence ve formátu MIDI z datového proudu	444
825	Délka sekvence ve formátu MIDI	445
826	Pozice sekvenceru MIDI	445
827	Nastavení hlasitosti přehrávání sekvence ve formátu MIDI	445
828	Opakované přehrávání sekvence MIDI v JDK 1.4-	446
829	Opakované přehrávání sekvence MIDI v JDK 1.5+	446
830	Korektní ukončení přehrávání sekvence MIDI	446

Java Media Framework (JMF) 447

831	Rozdíl mezi JMF a rozhraním Java Sound API	447
832	Diagnostika prostředí pro tvorbu aplikací s podporou multimédií	447
833	Diagnostika prostředí pro tvorbu aplikací s podporou multimédií	447
834	Jednoduchý způsob ověření, zda multimediální soubor půjde přehrát ve vaší aplikaci	448
835	Adresa souboru s multimediálním obsahem	448
836	Tvorba objektu přehrávače multimediálního obsahu	448
837	Přehrávání multimediálního souboru	449
838	V jednoduchosti je síla: Přehrávač souborů MP3	449
839	Zobrazení videa pomocí lehkých komponent knihovny JFC/Swing	449
840	Univerzální panel pro zobrazení přehrávače médií v okně	450
841	Mini Media Player	451

842	MDI Media Player	452
843	Dotaz na aktuální snímek	454
844	Dotaz na délku filmu v sekundách	455
845	Dotaz na počet snímků ve filmu	455
846	Převinutí multimediálního souboru	455
847	Převíjení filmu v přehrávači	456

Ovladače JDBC 457

848	Získání ovladačů JDBC pro příslušné databáze	457
849	Načtení ovladače JDBC	457
850	Ovladač použitý pro databázové připojení	457
851	Výpis všech načtených ovladačů JDBC	458
852	Výpis parametrů pro tvorbu připojení JDBC	458

Databáze a práce s daty 459

853	Dotaz na záznamy z databázové tabulky	459
854	Připojení k databázi ODBC a výpis obsahu tabulky	459
855	Tvorba nové databázové tabulky	460
856	Vložení nového záznamu do tabulky	460
857	Vložení záznamu do tabulky pomocí výsledné sady	460
858	Vymazání všech záznamů z databázové tabulky	460
859	Vymazání celé tabulky z databáze	461
860	Vymazání záznamu z tabulky pomocí výsledné sady	461
861	Vymazání vybraného záznamu z databázové tabulky	461
862	Vymazání vybraného záznamu z databázové tabulky	462
863	Aktualizace záznamu v databázové tabulce	462
864	Ovlivnění počtu záznamů načítaných z databáze	462
865	Potřebujeme zjistit počet záznamů v tabulce	463
866	Potvrzení nebo vrácení aktualizace databáze	463
867	Uložení binárních dat v databázové tabulce	463
868	Načtení binárních dat uložených v databázové tabulce	464
869	Načtení binárních dat pomocí objektu typu Blob	464
870	Zpracování výjimek při práci s databázemi SQL	464
871	Varovné zprávy databázového serveru	465
872	Zástupné znaky v příkazech jazyka SQL	466
873	Výpis detailů o připojené databázi	467
874	Dotaz na maximální délku názvu tabulky v databázi	467
875	Zjišťujeme, jaké funkce má databáze pro datum a čas	467
876	Zjišťujeme, jaké funkce má databáze pro práci s řetězci	467
877	Zjišťujeme, jaké funkce má databáze pro práci s čísly	468

878	Zjišťujeme, zda databáze podporuje transakce	468
879	Jaké systémové funkce obsahuje připojená databáze	468
880	Dotaz na dostupné typy SQL připojené databáze	468
881	Jaké uložené procedury obsahuje připojená databáze	469
882	Žádost o seznam všech tabulek v databázi	469
883	Dotaz na všechna klíčová slova připojené databáze	469
884	Dotaz na možnost užití dávkového zpracování příkazů	470
885	Dotaz na podporu dynamických výsledných sad	470

Databáze a výsledné datové sady 471

886	Tvorba dynamické obousměrné výsledné sady	471
887	Jak zjistit, zda lze výslednou sadu aktualizovat	471
888	Tvorba obousměrných výsledných datových sad	471
889	Jak zjistit, zda je výsledná sada obousměrná	472
890	Je vybraný záznam ve výsledné sadě první?	472
891	Zjištění pozice vybraného záznamu v obousměrné výsledné sadě	472
892	Je vybraný záznam ve výsledné sadě poslední?	473
893	Procházení záznamů v obousměrné výsledné sadě	473
894	Názvy sloupců ve výsledné sadě	474
895	Načítání dat z výsledné sady	474
896	Hodnoty NULL v datech výsledné sady	475
897	Zjišťujeme počet záznamů ve výsledné sadě	476
898	Efektivnější zjištění počtu záznamů ve výsledné sadě	476
899	Jaké výsledné datové sady lze používat v připojené databázi	476
900	Aktualizace záznamu v databázové tabulce	477
901	Obnovení záznamu v dynamické výsledné sadě	477
902	Jak stornovat nechtěnou aktualizaci dat	478
903	Připojení k databázím MySQL	478
904	Připojení k databázím Oracle	479
905	Připojení k databázím SQL Server	479

Java ME a NetBeans 481

906	Mikroedice jazyka Java	481
907	CLDC (Connected Limited Device Configuration)	481
908	MIDP (Mobile Information Device Profile)	481
909	CDC (Connected Device Configuration)	481
910	Jak vyvíjet aplikace pro mobilní zařízení v integrovaném vývojovém prostředí NetBeans	482
911	Visual Mobile Designer	482
912	Práce ve vizuálním návrháři	482
913	Práce s emulátory	484

914	Umístění úložiště emulátoru	484
915	Velikost úložiště emulátoru	484
916	Jak přidat kontakty do databáze emulátoru	485
917	Tvorba kontaktů pro testování	485
918	Ovládání emulátoru	486
919	Pozastavení aplikace spuštěné v emulátoru	486
920	Úprava výkonu emulátoru	486
921	Seznam emulátorů podporovaných v IDE NetBeans	487
922	Přidání nové platformy emulátoru do NetBeans	487
923	Změna implicitního zařízení emulátoru	488
924	Nastavení platformy CLDC/MIDP	488
925	Samostatné spuštění emulátoru	488

Midlety

489

926	Základní kostra midletu	489
927	Jednoduchý midlet	489
928	Jak na úvodní obrazovky v aplikacích typu Java ME	490
929	Úvodní obrazovka midletu	491
930	Neznámá velikost displeje	491
931	Využití celého displeje telefonu	492
932	Midlet s úvodní obrazovkou	493
933	Prohlížeč souborů v mobilním zařízení	494
934	Personal Information Management – PIM	495
935	Synchronizace kontaktů, kalendáře a seznamu úkolů	495
936	Tvorba kontaktů programově	495
937	Prohlížeč kontaktů	496
938	Tvorba vyčkávací stránky pro úlohy spouštěné na pozadí	497
939	Připojení k síti	498
940	Přihlašovací obrazovka	499
941	Vlastní program pro posílání zpráv SMS	500
942	Co je to formát SVG	501
943	Příklad jazyka SVG	501
944	Zobrazení vektorového obrázku	502
945	Otevření vektorové animace SVG	503

Databáze v mobilním zařízení

505

946	Jak ukládat data v mobilním zařízení	505
947	Citlivá osobní data v mobilním zařízení	505
948	Jak ukládat data do mobilní databáze	505
949	Jak číst data z mobilní databáze	506

950	Zobrazení záznamů z mobilní databáze	506
951	Jak třídit záznamy v mobilní databázi	507
952	Jak vyvolat reakci na změny v databázi	507
953	Jak vymazat záznam z mobilní databáze	507
954	Jak vymazat celou databázi aplikace	508
955	Jak zjistit názvy všech databází aplikace	508
956	Jak zjistit velikost databáze	508
957	Jak zjistit, o kolik lze velikost databáze ještě zvětšit	508

Mobile Media API (JSR 135 API) 509

958	Hlavní objekty pro zpracování multimédií	509
959	Generování tónů	509
960	Jak určit kmitočet a výšku tónu	509
961	Tvorba přehrávačů	509
962	Typy mediálního obsahu	510
963	Adresování různých typů médií	510
964	Seznam podporovaných typů mediálních souborů	511
965	Seznam podporovaných protokolů	511
966	Určení typu mediálního souboru z adresy URL	511
967	Zachycení zvuku v mobilním zařízení	511
968	Přehrávání nahrávky	512
969	Nastavení hlasitosti přehrávání	512

Konfigurace a instalace mobilních aplikací 513

970	Vývoj aplikací pro různé typy zařízení pomocí konfigurací projektu	513
971	Vývoj aplikací pro různé typy zařízení pomocí atributů typu Ability	513
972	Vývoj aplikací pro různé typy zařízení pomocí direktiv preprocesoru	514
973	Spouštění více konfigurací najednou ve vývojovém prostředí NetBeans	514
974	Jak připravit aplikaci s pomocí vývojového prostředí NetBeans	514
975	Jak připravit aplikaci vlastními silami	514
976	Jak dostat aplikaci do mobilního telefonu	515
977	Jak dostat aplikaci do mobilního telefonu protokolem WAP	515
978	Nastavení typů MIME na serveru HTTP, jenž nabízí vaše midlety	516
979	Proč je lepší v deskriptoru JAD uvádět absolutní adresu URL souboru JAR	516

Bezdrátové technologie 517

980	Spojení platformy J2ME a technologie Bluetooth	517
981	Standard JSR-82	517
982	Minimální požadavky technologie Bluetooth a J2ME	517

983	Co je to BCC	518
984	Možnosti aplikačního rozhraní JSR-82	518
985	Bluetooth 2.2/3.0 = 24 Mb/s	518
986	Inicializace zásobníku Bluetooth	518
987	Informace o lokálním zařízení	519
988	Informace o vzdáleném zařízení	519
989	Vyhledávání zařízení	520
990	Vyhledávání služeb	520
991	Registrace služby Bluetooth	520
992	Komunikace	521
993	Emulace sériového portu	521
994	Server sériového připojení Bluetooth	522
995	Klient sériového připojení Bluetooth	522
996	Formát adresy URL pro spojení se zařízením standardu Bluetooth	523
997	Není třeba stále vyhledávat dostupná zařízení	523
998	Chyba ve specifikaci	523
999	Server typu Bluetooth	524
1000	Klient typu Bluetooth	525
1001	Posílání dat sítí GPRS	526

Rejstřík **527**

Úvodem

Devadesát procent projektu spotřebuje devadesát procent vašeho času. Zbývajících deset procent projektu spotřebuje dalších devadesát procent času. Abyste svůj čas strávený na projektech vyvíjených v jazyce Java mohli využít co nejefektivněji, dostává se vám do ruky nové vydání knihy z řady 1001 tipů a triků. Jistě jste se mnohokrát setkali s tím, že chcete-li vyřešit nějaký problém, nemůžete ani za nic zjistit, jak byste vlastně měli postupovat. Pochopitelně až do doby, kdy už takové řešení potřebovat nebudete. A pravděpodobnost, že svůj problém vyřešíte pomocí nápovědy, je... však to znáte sami. Efektivní využití času a energie programátora je základem úspěchu jakéhokoli softwarového projektu. Proto v této knize přinášíme řadu rad a receptů, jak vyřešit dílčí problémy, s nimiž se většina z nás čas od času potká. Právě skutečnost, že se s určitými úlohami nesetkáváme každodenně, způsobuje, že na jejich vyřešení trávíme více času, než bychom chtěli.

Možná jste počítačovým samoukem, který dostal práci v menší firmě proto, že zná rozdíl mezi objektem a třídou, a očekává, že zde najde odpověď na všechny své dotazy. Možná se ale jen zajímáte o specifika jazyka Java a chcete se o něm dovědět něco víc, nebo se dokonce v programování v Javě chcete stát specialistou. Nebo snad v tomto oboru platíte za uznávaného odborníka, který si chce jen doplnit něco málo, co mu ještě uniklo.

Hledáte-li jako pomocníka knihu, jež vám v řadě situací a problémů navrhne možná řešení, pak jste určité na správné adrese.

Pravděpodobně zde neobjevíte nové horizonty, můžete si však ušetřit spoustu drahocenného času místo vymýšlení toho, co už někdo vymyslel před vámi. Tato kniha vám může například posloužit jako záchytný bod v zoufalé situaci, kdy je potřeba rychle vyřešit poměrně snadný problém, který má ovšem tu nepříjemnou zvláštnost, že jste se jím dosud nikdy vážně nezabývali, nebo jste se jím zabývali pouze okrajově. Nejste-li chodící encyklopedií, musíte logicky čas od času vyhledat pomoc. Určitě zde nenajdete odpovědi na všechny své otázky. Tato kniha se vám snaží pouze ukázat, že každý problém lze vyřešit několika způsoby. Vybrat si však musíte sami.

Pokud tyto řádky čtete ještě v knihkupectví, máte dvě možnosti: Buď si knihu koupíte, nebo ji vrátíte do regálu. Není to učebnice programovacího jazyka. Koncepce knihy ale od čtenáře určité programátorské zkušenosti vyžaduje – to se nedá popřít. Tipy a triky budou logicky srozumitelnější zkušenějším programátorům. Příklady v sobě nesou rovněž velmi cenné informace, jež jsou užitečné i pro začínající programátory.

Toto vydání je vůči tomu původnímu obohaceno o tipy určené pro práci s verzí JDK 6. Tato verze jazyka umožňuje mnohem pohodlnější programování zejména uživatelského rozhraní, multimédií nebo aplikací pro mobilní zařízení.

Považujte, prosím, knihu *1001 tipů a triků pro Javu* za určitý druh zaváděcího programu, do něhož jste vstoupili a do něhož byly zadány různé vstupní informace od nejjednodušších apletů po nepoměrně složitější postupy používané například při zpracování multimédií – zkrátka mnohé z toho, co je často v dnešním programátorském světě potřeba. Představte si, že vás tento zaváděcí program napojí na dokovací stanici, k níž je připojeno přes tisíc konektorů,

které vás spojí s dalšími obzory, za nimiž se skrývá tajuplný svět, ve kterém se jazyk Java používá k předávání myšlenek nejen počítačům, ale i dalším lidem.

Chci upozornit čtenáře, že všechny informace uvedené v této knize jsou šířeny bez záruky. Přes všechny snahy autora i nakladatelství o výstižnost sdělení i přesnost uváděného kódu je možné, že v určitých konkrétních případech některé z tipů nebudou fungovat tak, jak je uváděno v příslušných pokynech. Je dokonce velmi pravděpodobné, že ani důsledný přístup autora a dvojí redakční korektury nezajistí při zpracování rozsáhlého množství informací naprosté vymýcení nebo výjimečné nezavlečení chyb nebo chybiček do knihy. Stejně jako u jiných knih všude na světě si jich bohužel všimnete až vy, čtenáři. Odhalíte-li nějakou, pošlete, prosím, zprávu o ní spolu s návrhem její opravy redakci nakladatelství CPress na adresu knihy@cpress.cz.

Všechny odůvodněné opravy budeme moci vystavit na webu <http://knihy.cpress.cz> v dokumentu Errata a samozřejmě text opravit v případném dotisku nebo novém vydání knihy.

Kromě toho je zde ještě další okolnost. Vzhledem k tomu, že se publikované tipy soustředí vždy na princip či zvláštnost daného řešení, jde často jen o úryvky kódu, nikoli o kompletní aplikace s ošetřením všech okolností, které mohou vykonávání kódu ovlivnit. Nebylo proto účelné v omezených příkladech zachytit všechny možné důsledky, kterými vás může daný kód zasazený do vaší aplikace či daného prostředí překvapit. Kvůli úspoře cenného místa neobsahují ukázky kódu většinou ani příklady ošetření možných výjimek. Doporučované postupy je nutno v příslušných aplikacích podrobit důslednému testování, které by mělo odhalit možné skryté důsledky vyplývající z konkrétní konfigurace hostitelského systému. Z tohoto důvodu nenesou autor ani nakladatelství žádnou odpovědnost za jakékoli ztráty nebo škody způsobené přímo nebo nepřímo informacemi uvedenými v této knize.

Vážení čtenáři,

držíte v ruce 2. vydání sbírky tipů a triků pro programátory v jazyce Java. Společně s redakcí věříme, že vás mohou povzbudit na strastiplné cestě za poznáním tajů práce v tomto moderním programovacím jazyce.

Knihy je pomyslně rozdělená do devíti tematických celků. Tipy, recepty, návody a rady jsou přesto uspořádány tak, aby na sebe volně navazovaly. Čtenář tak teoreticky může svůj program postupně rozvíjet řadou po sobě následujících návodů.

Základem je objekt

Milí čtenáři, víte, co je to objekt?

Jistě víte, jaký je význam slova „objekt“ mimo programátorské prostředí. Objektem může být neidentifikovaný létající objekt známý také jako UFO, může jím být objekt vaší touhy, může jím být také rekreační objekt v Beskydech nebo na Šumavě. Může jím ale být také tužka, lidský vlas nebo také klika u dveří. Objekt je obvykle vnímán jako obecné vyjádření spřízněných pojmů jako předmět, věc nebo těleso. Objektem je však také jakákoli stavba či budova. V určitých případech může jít o více objektů, budov, předmětů nebo těles, pokud tyto tvoří jeden celek. V programování jde však o jeden ze základních stavebních prvků. V Javě dokonce o stavební prvek stěžejní.

Proto se první dvě kapitoly tipů zabývají především možnostmi řešení úskalí při tvorbě objektů. Začátečníci, pokročilí a snad i znalci zde najdou recepty na řešení problémů při práci s třídami, objekty, s uspořádáním objektů do balíčků. Protože jde o první kapitoly, neměli by-

chom zapomenout ani na úplné nováčky a právě pro ně je zde několik receptů na překlad prvních programů.

Grafické uživatelské rozhraní

Nejběžnější uživatelské rozhraní umožňuje ovládat počítač pomocí interaktivních grafických ovládacích prvků. Na monitoru nebo displeji počítače či jiného inteligentního zařízení jsou zobrazena okna, v nichž programy zobrazují svůj výstup. Uživatel v tomto prostředí používá ke komunikaci s programy klávesnici, myš a grafické ovládací prvky, jako jsou menu, ikony, tlačítka, posuvníky, formuláře a podobně, což jsou základní prvky pro interakci programu s uživatelem.

Grafické uživatelské rozhraní, o němž je řeč, využívá možnosti počítače ke snazšímu užívání počítačových programů. Je-li grafické uživatelské rozhraní navrženo dobře, nemusí se uživatel učit složité příkazové jazyky a syntaxe. Grafické uživatelské rozhraní však nelze zbytečně přeceňovat, neboť mnozí uživatelé upřednostňují rozhraní příkazového řádku pro jeho vysokou efektivitu, zejména jsou-li důvěrně obeznámeni s příkazovou syntaxí.

Protože grafické uživatelské rozhraní je fasádou většiny moderních programů, hodnotí mnozí uživatelé kvalitu programu právě podle něho. Vývojáři se logicky snaží tomuto požadavku vyhovět. Autoři jazyka Java na zmiňované snahy reagovali v nové verzi jazyka (Java 6) a rozšířili základní knihovny o mnoho velmi užitečných tříd.

Čtenář proto v dalších osmi kapitolách najde například recepty na tvorbu průhledných oken, oken různých tvarů, na práci s ikonami v oznamovací oblasti hlavního panelu. Také tipy pro práci s 2D grafikou nabízejí několik možností, jak při tvorbě neotřelého designu realizovat poměrně snadno vlastní grafické návrhy. Pokud vás mimo jiné zajímá, jak vytvářet ovládací prvky, formuláře, jak pracovat v režimu celé obrazovky, jak měnit atributy nebo směr písma, určité v tomto souboru kapitol najdete také něco pro sebe.

Manipulace s textem

Text je přirozeným způsobem komunikace. Ve všech systémech je prakticky stejný. Rozumějí mu lidé a v dalších sedmi kapitolách zjistíte, že při troše snahy mu mohou porozumět také vaše programy. Někdy se vám může hodit, že vaše aplikace bude schopna zpracovat nebo dokonce upravit malé nebo velké množství textu. Když už aplikace takové schopnosti má, je už jen na vás a na zadání, zda aplikace bude s textem pracovat automaticky, nebo bude interaktivně plnit požadavky uživatelů.

Programy v jazyce Java byly dlouhou dobu považovány za příliš „textové“. V těchto kapitolách se dovíte, že zpracování textu nemusí nutně znamenat pouze zobrazení textového souboru ve formátu prostého textu nebo zpracování konfiguračního souboru či vstupu příkazového řádku.

Java poskytuje dostatek nástrojů k formátování textu na špičkové úrovni. Příprava zdrojových kódů není možná nejintuitivnější, ale výsledky mohou opravdu stát za to. Běžní uživatelé totiž většinou dávají přece jen přednost úpravám textu v grafickém prostředí.

V těchto kapitolách najdete mimo jiné recepty na prohledávání textu, vyhledávání znaků, slov a slovních spojení. Ti, kdo potřebují rychle zjistit, jak správně naformátovat číslo, datum nebo čas, zde najdou přesně to, co potřebují. Sem jsem zařadil také příklady práce s kódováním textu.

Manipulace se soubory

Souborový systém je označení pro způsob organizace dat ve formě souborů tak, aby bylo možné je snadno najít a přistupovat k nim. Jazyk Java obsahuje velmi účinné a efektivní nástroje pro správu souborového systému a manipulaci s uloženými daty. Další tři kapitoly obsahují řadu účinných a zároveň poměrně jednoduchých technik, díky nimž pro vás budou úkony spojené s manipulací se soubory procházkou růžovým sadem.

Práce v síti

Další logickou zastávkou jsou čtyři kapitoly tipů zaměřené na práci se sítí, na distribuované systémy, Internet a v návaznosti na to na stránky JSP. Sokety a přesměrování jsou další souborové abstrakce, ale zároveň velmi preferované síťové prvky, které se vám při každodenní práci mohou velmi dobře hodit.

Systém

Pokračováním, ale také logickým završením předchozích sedmi kapitol je soubor pěti kapitol pojednávajících o systémových oblastech, jako jsou třeba správa paměti, možnosti souběžného zpracování (procesy a podprocesy), možnosti spouštění aplikací v různých operačních systémech a zabezpečení.

Multimédia

Do dvou následujících kapitol jsem zařadil tipy a triky pro práci se zvukem a obrazem.

Od počátku 90. let minulého století se označení multimediální aplikace nebo multimediální software začalo používat pro aplikace, které využívaly kombinace textových, obrazových, zvukových či animovaných nebo filmových dat. Jelikož jsou dnes multimediální nejen prakticky všechny osobní počítače, ale také mobilní a další inteligentní zařízení, je tvorba aplikací tohoto typu trendem soudobého vývoje.

V řadě případů jsou totiž animace a zvuk tím pravým kořením dodávajícím aplikacím punc originality. Využitím potenciálu multimédií váš projekt může dostat zcela jiný rozměr.

Práce s databází

V dalších třech kapitolách se detailně věnujeme jedné z klíčových technologií. Jde o technologii JDBC, která poskytuje základní rozhraní pro unifikovaný přístup k databázím. Díky tomu nemusíte znát různá API jednotlivých databází. Naučíte-li se pracovat s jednotným rozhraním JDBC, můžete je pak používat pro přístup do libovolného databázového systému dostupného prostřednictvím ovladače JDBC.

Rozhraní JDBC navíc není určeno jen pro přístup k relačním databázím, ale k libovolnému formátu dat, ukládanému ve „sloupcové podobě“, což mohou být i sešity tabulkových kalkulátorů, textové soubory apod.

Mobilní zařízení

V dnešní době je pro každého naprostou samozřejmostí mít mobilní telefon. Mobilní komunikace je odvětví spotřební elektroniky, které prochází velmi rychlým vývojem. Jedním z posledních hitů je použití jazyka Java právě v mobilních telefonech. Ale nejen v nich, nýbrž i v ostatních mobilních zařízeních, například v PDA. Tato implementace otevírá ce-

lou řadu možností, jak rozšířit funkce mobilního zařízení podle potřeb uživatele. Mobilní zařízení se tak stává přenosnou kapesní alternativou počítače, která sice nemá takové možnosti jako klasické PC, ale jak ukazuje vývoj midletů, k dostání je stále více aplikací napsaných v J2ME, jež jsou odvozeny od „plnokrevných“ aplikací známých z PC.

V posledních kapitolách této knihy se dovíte mnohé o tom, jak aplikace pro mobilní zařízení vytvářet, ladit a testovat, ale také leccos o vizuálním vývojovém prostředí, v němž je vývoj takových aplikací výrazně jednodušší.

Jak číst tuto knihu

Tato kniha byla navržena takovým způsobem, aby ji bylo možno číst najednou od první do poslední strany, nebo podle vybraných témat. Chcete-li se dostat ke specifickým programátorským tipům, nalistujte příslušné téma v obsahu nebo v rejstříku vyhledejte klíčový odkaz na témata, která budete považovat za zajímavá.

V této knize si přijdou na své nejen profesionálové, ale i domácí kutilové. Věříme, že každý, kdo bude chtít proniknout do neznámých tajů světa bez hranic, najde v této knize správné ukazatele na své cestě za pravdou.

Každý tip, trik, recept nebo návod je v knize označen pomocí jedné ze tří ikon naznačujících orientační úroveň znalostí čtenáře, pro něhož je daný tip určen především. Kniha je vhodná pro všechny skupiny programátorů a své si v ní najdou jak začátečníci, tak pokročilí vývojáři.

Skuteční znalci zase mohou kromě tipů a triků popisujících spíše pokročilé možnosti jazyka Java k tvorbě různých zvláštních efektů využít knihu například jako referenční příručku.



začátečník

Takto označený tip je určen především čtenářům s minimálními znalostmi jazyka Java, kteří se s jazykem teprve seznamují.



pokročilý

Takto označený tip je určen hlavně čtenářům se základními znalostmi jazyka Java, které rozšiřuje.



znalec

Takto označený tip je určen zejména čtenářům s velmi dobrou znalostí jazyka a obvykle popisuje velmi pokročilé a důmyslné postupy.

Zdrojové kódy

Zdrojové kódy všech tipů a triků, jež obsahují zdrojový kód, najdete na doprovodném disku v textové podobě. Každý takový textový soubor je označen číslem příslušného tipu. V souboru je odkaz na tuto knihu a pak už zdrojový kód, jehož aplikace při řešení konkrétního problému je tak výrazně jednodušší, než kdybyste vše museli opisovat ručně.

Doprovodné CD

Doprovodný disk obsahuje kromě zdrojových kódů také řadu odkazů na užitečné stránky a také několik užitečných nástrojů, jež vám programování v jazyce Java výrazně usnadní nebo alespoň zpříjemní.

Najdete na něm také instalační programy dvou vývojových prostředí: Eclipse a NetBeans.

Konvence použité v knize

- *Kurzíva* – Kurzívou v knize budeme vyznačovat cesty, názvy souborů, programů a internetové adresy (adresy URL i názvy domén). Stejně budeme označovat i nové pojmy, které budeme definovat.
- *Neproporcionální písmo* – Toto písmo použijeme pro výstup z programů a pro názvy a klíčová slova v příkladech.
- *Neproporcionální písmo s kurzívou* – Toto písmo použijeme pro volitelné parametry či prvky v okamžiku, kdy budeme popisovat syntaxi příkazu.
- **Neproporcionální písmo vyznačené tučně** – Toto písmo použijeme pro příkazy, které by měly být psány doslovně a také pro zvýraznění ve zdrojovém kódu programu či konfiguračních souborech.

Vznik prvního objektu a prvního programu

1 Co je to třída?



začátečník

Je-li všechno v Javě považováno za objekt, co vlastně určuje, jak bude vypadat ta která třída objektů a jak se bude chovat? Co stanovuje typ objektu? V jazyce Java, stejně jako v mnoha dalších objektově orientovaných jazycích, se k definici typu objektu používá klíčové slovo `class`, které má význam oznamovací věty: Toto je deklarace nového typu objektu. Sděluje celému světu: „Tímto vám oznamuji, jak bude vypadat nová třída objektů.“ Třída obvykle definuje vlastnosti, metody a události, s nimiž její instance později pracují.

2 Nejjednodušší nová třída



začátečník

Deklarace nové třídy, potažmo nového datového typu, je velmi jednoduchá.

```
class UkázkováTřída { /* sem vložíme definici těla třídy */ }
```

Nyní máme zcela nový datový typ. Program může obsahovat libovolný počet objektů tohoto typu. Ba, i program samotný může být typu `UkázkováTřída`.

V tomto příkladu obsahuje tělo třídy `UkázkováTřída` pouze komentář. To znamená, že s objektem tohoto typu toho nyní asi moc nesvedete. Dokud pro tento typ nedefinujete žádné metody, nemůžete mu vlastně přikázat vůbec nic.

3 Třída s vlastními daty



začátečník

Při definici lze do nového typu (třídy objektů) vložit dva typy prvků: datové složky a metody. Datová složka je objektem libovolného typu, s nímž lze komunikovat prostřednictvím odkazu. Datové složky slouží objektu k ukládání vlastních soukromých dat. Tato data objekt s jinými objekty nesdílí:

```
public class UkázkováTřída {  
    int celéČíslo;  
    float desetinnéČíslo;  
    boolean logickáHodnota;  
}
```

Jakmile vytvoříte nový odkaz na objekt, můžete do objektu ukládat požadované informace:

```
UkázkováTřída ukázka = new UkázkováTřída();  
ukázka.celéČíslo = 10;  
ukázka.desetinnéČíslo = 6.7f;  
ukázka.logickáHodnota = true;
```


4 Manipulace s objekty a s jejich daty



Objekty našeho ukázkového typu mohou prozatím sloužit pouze jako datové kontejnery, protože jsme doposud nedefinovali žádné metody, jimiž bychom mohli s daty jednotlivých objektů nějak manipulovat.

Každá metoda musí být v Javě postavena pomocí čtyř základních stavebních kamenů, jimiž jsou název, argumenty, typ návratové hodnoty a konečně tělo metody, které obsahuje aplikační logiku. Následující příklad znázorňuje jednoduchou metodu, která neobsahuje argumenty, ale informuje o hodnotě uložené v logické proměnné `logickáHodnota`:

```
boolean JeNastaveno() { return logickáHodnota; }
```

Následující metoda změní hodnotu celočíselné proměnné a vrátí novou hodnotu:

```
public class UkázkováTřída {
    //...
    int přidejHodnotu(int přidat) {
        celéČíslo = celéČíslo + přidat;
        return celéČíslo;
    }
    //...
}
```

Metody však nemusejí vracet žádnou hodnotu. Jako návratový typ se v takovém případě používá speciální klíčové slovo `void`.

```
public class UkázkováTřída {
    //...
    void zvýšitHodnotu() {
        celéČíslo ++;
        return;
    }
    //...
}
```

Je-li typem návratové hodnoty speciální typ `void`, používá se klíčové slovo `return` pouze k ukončení metody. Není třeba jej uvádět, pokud běh programu dospěl ke konci metody, ale v zájmu zachování konzistence programovacího stylu doporučuji klíčové slovo `return` používat vždy. Metodu lze ukončit v libovolném místě, tj. i kdyby za příkazem `return` následovaly další řádky kódu. Jestliže však má metoda vrátit hodnotu, překladač si vynutí, abyste před ukončením metody vrátili hodnotu odpovídajícího typu, bez ohledu na to, z jakého místa jejího těla budete metodu ukončovat.

5 Určení jedinečnosti metody



Jedinečnost metody určuje signatura. V jazyce Java je signatura utvářena názvem, seznamem argumentů a datovými typy argumentů. Návratová hodnota není součástí signatury. Znamená to tedy, že dvě metody definované pro daný typ objektů se lišíci, pouze návratovým typem, budou překladačem považovány za totožné.

6 Modifikátory viditelnosti



V předchozích dílech jsme použili klíčové slovo `public`. Jedná se o modifikátor viditelnosti, jenž nám sděluje, že je daná třída, metoda či proměnná veřejná, tedy viditelná odkudkoli ze zdrojového kódu. Pokud proměnnou či metodu definujete pomocí klíčového slova `private` jako soukromou, bude přístupná pouze z jiných metod téhož objektu.

Dalšími modifikátory viditelnosti jsou `protected`, `private protected` a implicitní modifikátor viditelnosti (není-li uvedeno nic).

7 Překrývání implicitní viditelnosti členu třídy



Ukázka využití tříd z balíčku `java.lang.reflect`.

Objekty v jazyce Java implicitně vynucují přístup podle pravidel stanovených zásadami jazyka Java. To znamená, že implicitně nelze zvenčí používat přímo žádnou soukromou datovou složku. Chcete-li tato nastavení obejít, použijte metodu `setAccessible()` příslušného objektu. Má to však háček. Váš program nemusí mít oprávnění potřebná k volání metod `setAccessible()`. Pokud je nemá, dojde k výjimce typu `SecurityException`.

```
datováSložka.setAccessible(true);  
konstruktor.setAccessible(true);  
metoda.setAccessible(true);
```

8 Klíčové slovo static



Nový objekt určitého typu lze obvykle používat až po vytvoření pomocí klíčového slova `new`. Právě to je okamžik, kdy je vytvořeno soukromé úložiště, s nímž lze pomocí metod objektu manipulovat. K užití běžných nestatických metod a dat musíte tedy mít nejprve konkrétní objekt, který použijete jako klíč k požadované operaci na konkrétním objektu.

Pokud ovšem chcete, aby určitou informaci sdílely všechny objekty určitého typu, bez ohledu na počet použitých instancí, nebo chcete-li používat metodu, která není přidružená k žádnému konkrétnímu objektu daného typu, jednoduše proto, abyste ji mohli používat, i kdyby nebyl doposud vytvořen žádný objekt příslušného typu, použijte k definici takového prvku klíčové slovo `static`. Je-li datová složka nebo metoda statická, znamená to, že není vázána na žádný objekt daného typu. Statická data lze upravovat nebo statické metody volat, aniž byste museli vytvořit jakoukoli instanci příslušného typu. To v podstatě znamená, že taková data a takové metody lze označit jako data třídy nebo metody třídy.

Chcete-li datovou složku nebo metodu změnit na statickou, vložte před příslušnou definicí klíčové slovo `static`. V následujícím příkladu vidíte definici statického datového členu a jeho inicializaci:

```
class UkázkováTřída {  
    static int statickáHodnota = 2008;  
}
```

Bez ohledu na to, kolik bude v programu objektů typu `UkázkováTřída`, bude program pracovat pouze s jedinou hodnotou `UkázkováTřída.statickáHodnota` a všechny tyto objekty ponesou stejnou hodnotu 2008.

Možnost volání statických metod bez nutnosti vytvoření objektu je podstatou definice metody `main()`, která je vstupním bodem ke spuštění aplikace.

9 Nejjednodušší třída jako samostatný program



K tomu, aby objekt určitého typu (nebo také instance určité třídy) mohl být použit jako program, stačí doplnit jen nepatrně předchozí deklaraci. Typ musí být veřejný a musí obsahovat alespoň veřejnou statickou metodu `main()`, která očekává argument typu `String[]`, čili pole řetězců.

```
public class UkázkováTřída {
    public static void main(String[] args) {
        // Sem umístíte logiku své aplikace.
    }
}
```

Kdyby typ nebyl veřejný nebo kdyby neobsahoval veřejnou statickou metodu `main()`, neměl by k němu virtuální stroj jazyka Java, jenž je v hostitelském počítači odpovědný za spouštění programů napsaných v jazyce Java, přístup. Tudíž by ho nemohl spustit.

V tomto příkladu obsahuje tělo metody `main()` opět pouze komentář (i když tentokrát jiného typu).

10 Jak program uložit



V jazyce Java se uložení programu řídí několika specifickými pravidly. Soubor, v němž je program uložen, musí být pojmenován stejně jako v něm uložený typ, který obsahuje hlavní metodu aplikace – statickou metodu `main()`. To znamená, že typ `UkázkováTřída` je třeba uložit do souboru `UkázkováTřída.java`. Při tvorbě názvů souborů je třeba dodržovat psaní velkých a malých písmen, neboť virtuální stroj jazyka Java považuje například názvy `UkázkováTřída`, `ukázkováTřída` nebo `UkázkováTřída` za zcela odlišné identifikátory.

11 Překlad programu



Deklarace typu neboli také třídy objektů je uložena ve zdrojovém souboru, jenž může být programátorem kdykoli podle potřeby editován. Nabízí se však otázka, jak přimět počítač, aby námi připravený zdrojový kód spustil a vykonal tak požadovanou sadu programátorských instrukcí. Abyste mohli přeložit a spustit jakýkoli program napsaný v jazyce Java, musíte nejprve nainstalovat vývojové prostředí jazyka Java. To najdete zdarma na serveru `java.sun.com`, kde najdete veškeré informace a odkazy, jež vám stažení a instalaci prostředí usnadní. Po dokončení instalace prostředí JDK, upravte systémovou proměnnou `path` tak, aby systém nalezl překladač `javac.exe`, který slouží k překladu zdrojových souborů na bajtový kód jazyka Java. Je-li vývojové prostředí ve vašem počítači instalováno a příkazové prostředí zná cestu ke zmíněné aplikaci, stačí v pracovním adresáři (tj. v adresáři, v němž máte uložen zdrojový soubor `UkázkováTřída.java`) zadat následující příkaz:

```
javac UkázkováTřída.java
```

Překladač vytvoří nový soubor s příponou `.class` (např. soubor `UkázkováTřída.class`), jenž obsahuje spustitelný bajtový kód jazyka Java.

Pokud se však při překladu zobrazí jakékoli chybové hlášení, zjistěte jeho příčinu, opravte problém a teprve pak se vraťte k překladu programu.

12 Jak spustit připravený program v jazyce Java



začátečník

Spuštění takto připraveného programu je již zcela jednoduché. Stačí na příkazovém řádku zadat následující příkaz:

```
java UkázkováTřída
```

Co se ale stane, až program spustíme? Po spuštění naší první objektové aplikace se nestane nic pozorovatelného, protože jsme v hlavní metodě objektu aplikace nespecifikovali žádný pozorovatelný úkon.

13 Nejjednodušší odezva prvního programu



začátečník

Nahraďte ve výše uvedeném příkladu následující komentář:

```
// Sem umístíte logiku své aplikace.
```

příkazem:

```
System.out.println("Nazdar, světe!");
```

Nyní stačí znovu zdrojový kód na příkazovém řádku přeložit pomocí překladače `javac` a spustit pomocí příkazového prostředí `java`. Výsledek se dostaví v podobě vypsaného pozdravu.

14 Jednoduchá aplikace s argumenty



začátečník

Ukázka využití tříd z balíčku `java.lang`.

Argumenty aplikace jsou dostupné prostřednictvím pole `args`. Tato aplikace jednoduše vypíše větu „Nazdar, světe!“. Pokud při spuštění aplikace uvedete nějaké argumenty, aplikace je ještě před závěrečným pozdravem vypíše na obrazovku.

```
public class PrvniAplikace {
    public static void main(String[] args) {
        // Zpracování argumentů získaných z příkazového řádku.
        for (int i=0; i<args.length; i++) {
            // Zde můžete zpracovat prvek args[i].
            // Tato aplikace získané argumenty pouze vypíše na obrazovku.
            System.out.println(args[i]);
        }
        // Výpis obligátního pozdravu na obrazovku.
        System.out.println("Nazdar, světe!");
    }
}
```

Zdrojový kód uložte do souboru `PrvniAplikace.java` a přeložte do bajtového kódu:

```
javac PrvniAplikace.java
```

Výsledný program spustíte následujícím příkazem:

```
> java PrvniAplikace <argument1> <argument2>
```

15 Okamžité ukončení aplikace



Ukázka využití tříd z balíčku `java.lang`.

Aplikaci lze ukončit metodou `System.exit()`, přičemž této metodě můžete předat argument určující, zda během vykonávání programu k chybě došlo nebo nedošlo.

```
// K chybám nedošlo.
int errorCode = 0;
```

```
// Došlo k chybě.
errorCode = -1;
```

```
// Ukončení běhu aplikace.
System.exit(errorCode);
```

16 Jak zjistit, že se aplikace chystá ukončit svůj běh



Ukázka využití tříd z balíčku `java.lang`.

Je-li aplikace ukončena normálně, spustí všechna registrovaná ukončovací vlákna, počká na jejich dokončení a nakonec ukončí svůj běh. Aplikaci lze ukončit voláním metody `System.exit()`. K ukončení aplikace dojde rovněž po dokončení všech pracovních vláken. Aplikaci lze ukončit také uživatelsky, a sice stisknutím kombinace kláves `Ctrl+C`.

```
// Registrace ukončovacího vlákna.
Runtime.getRuntime().addShutdownHook(new Thread() {
    // Tato metoda se volá během ukončování běhu aplikace.
    public void run() {
        // Vykonat ukončovací operace...
    }
});
```

Abnormální ukončení běhu aplikace mají obvykle na svědomí závažné chyby ve virtuálním stroji jazyka Java nebo v přirozených knihovnách. V takových případech výše uvedený kód ukončovacího vlákna neproběhne.

17 Komentáře v kódu



V Javě lze používat dva typy komentářů. Zprvke tradiční komentář ve stylu jazyka C, který byl také zděděn jazykem C++. Tyto komentáře začínají značkou `/*` a končí značkou `*/`. Někteří programátoři začínají každý řádek takového komentáře symbolem `*`, takže komentář může vypadat takto:

```
/* To je komentář,  
 * který je rozložen  
 * na několik řádků.  
 */
```

Překladač vše, co se nachází mezi značkami `/*` a `*/`, ignoruje. Proto předchozí a následující zápis je v podstatě totožný:

```
/* To je komentář, který je rozložen  
   na několik řádků. */
```

Druhým typem komentáře je jednořádkový komentář, jenž začíná značkou `//` a pokračuje až do konce řádku. Používá se poměrně často, protože stačí pouze dvakrát stisknout stejnou klávesu. Navíc se nemusíte starat o ukončení komentáře:

```
// Toto je jednořádkový komentář.
```

18 S objekty se manipuluje pomocí odkazů



Máte-li vytvořený nový datový typ, znamená to, že systém tříd jazyka Java je nyní bohatší. Od této chvíle totiž můžete nový datový typ používat stejně jako jakýkoli jiný předdefinovaný typ.

Chcete-li použít například proměnnou typu `String` (čili odkaz na textovou proměnnou), vytvoříte v programu příslušný typový odkaz:

```
String text;
```

Obdobným způsobem můžete použít i nový datový typ:

```
UkázkováTřída nt;
```

Nyní jste však vytvořili pouze odkaz, nikoli samotný objekt. Pokud se v tomto okamžiku rozhodnete odeslat odkazu `nt` nějakou zprávu, vygeneruje systém chybu (v době provádění), protože odkaz ještě není připojen k žádnému objektu (není zde ještě žádný televizní přijímač). Bezpečnější praktikou je vždy inicializovat odkaz ihned při jeho vytvoření. K vytvoření konkrétního objektu nového typu použijte klíčové slovo `new` stejně jako u všech předdefinovaných tříd:

```
UkázkováTřída nt = new UkázkováTřída();
```

19 Všechny objekty musíte vytvořit



Když vytváříte nějaký odkaz (proměnnou), chcete k němu také pravděpodobně připojit nový objekt. V Javě se to dělá zpravidla pomocí klíčového slova `new`. Toto klíčové slovo sděluje systému: „Vytvoř mi nový objekt tohoto typu.“ V následujícím příkladu tedy systému sdělujeme, aby vytvořil odkaz na uvedený textový řetězec:

```
String s = new String("nový text");
```

Zjednodušeně však lze odkaz inicializovat přímo textovou konstantou. Správná inicializace nového objektu proběhne pomocí implicitního konstruktoru:

```
String s = "nový text";
```

20 Objekty nemusíte mazat



začátečník

Ve většině programovacích jazyků musí programátoři stanovit vhodnou životnost proměnných. Obvykle řeší otázku, jak dlouho je třeba ponechat proměnnou v paměti? Když je tedy proměnnou třeba vymazat, kdy by k tomu mělo dojít? Zmatek kolem životnosti proměnných může vést ke vzniku mnoha chyb. Java ovšem významně zjednodušuje tento problém tím, že za vás vymazání všech nepotřebných objektů zajistí.

21 Doba platnosti objektů



začátečník

Objekty nemají v jazyce Java stejnou životnost jako primitivní datové typy. Vytvoříte-li objekt jazyka Java pomocí klíčového slova `new`, zůstane v paměti i po vykročení z rozsahu výchozího oboru platnosti. Použijete-li následující zápis, bude odkaz `s` vymazán z paměti ihned po ukončení oboru platnosti:

```
{  
    String s = new String("řetězec");  
} /* konec platnosti odkazu "s" */
```

Přesto je však odkazovaný objekt typu `String` stále v paměti počítače. V této ukázce už ale neexistuje žádný způsob, jímž by bylo možné přístup k objektu získat, neboť platnost jeho jediného odkazu skončila.

22 Mazání objektů z paměti



začátečník

Předchozí příklad vyvolává zajímavou otázku. Jestliže Java ponechává osiřelé objekty v paměti, co jim brání, aby nakonec zcela zaplnily paměť a zastavily běh programu? Odpověď je jednoduchá: automatický správce paměti neboli garbage collector. V Javě jsou totiž objekty z paměti vymazány automaticky, jakmile skončí platnost posledního odkazu na ně, a tudíž je nelze dále používat. Nemusíte se tedy starat o uvolňování paměti. Tuto činnost zajišťuje právě automatický správce paměti. Ten si pamatuje počet odkazů na každý objekt. Jakmile počet odkazů klesne na nulu, je objekt označen k vymazání. V okamžiku, kdy systém bude mít k takové operaci čas, bude objekt z paměti vymazán.

23 Explicitní ukončení platnosti objektu



pokročilý

Pokud ovšem chcete, aby objekt z nějakého důvodu přestal používat systémové prostředky (soubory, síťová spojení), umístěte příslušný kód do metody `finalize()`:

```
protected void finalize() { /* Ukončovací kód. */ }
```

Práce s třídami a objekty

24 Přirozený jazyk v pojmenování



K pojmenování balíčků, tříd, objektů (proměnných) a metod se snažte používat jeden přirozený jazyk. To platí zejména pro vícejazyčné projekty.

25 Čeština v identifikátorech



K pojmenování tříd jsme v předchozích příkladech použili české znaky. V Javě, stejně jako v ostatních moderních programovacích jazycích, lze totiž k tvorbě srozumitelných identifikátorů používat diakritiku. Použití diakritiky v těchto případech je věcí diskuse. V pokročilých projektech, zejména když na projektu pracuje více programátorů nebo dokonce programátorů různých národností, se k tvorbě identifikátorů obvykle používá angličtina. Začínajícím programátorům však doporučuji tvorbu srozumitelných českých identifikátorů, neboť srozumitelné identifikátory zjednodušují orientaci uvnitř zdrojového kódu. Zejména, jde-li o kód ukázkový nebo studijní. Protože ale tato kniha není určena jen začínajícím programátorům, přejdeme u pokročilejších tipů v pojmenování identifikátorů k angličtině.

26 Konvence pojmenování



Neexistují žádná přísná pravidla, která by určovala, kdy musíte zavést konvenci pojmenování. Jednotná konvence se vám ale rozhodně vyplatí, pracuje-li na projektu více vývojářů, chcete-li úpravy nebo údržbu svého kódu předat jinému vývojáři (což je velmi časté), je-li program tak rozsáhlý, že si nemůžete všechno zapamatovat a musíte o programu uvažovat po částech, nebo je-li vývoj programu dlouhodobý, takže se může klidně stát, že jej na několik týdnů nebo měsíců odložíte, než se k němu opět vrátíte.

27 Význam názvů identifikátorů



Význam volby dobrých názvů balíčků, tříd, proměnných a obecně různých identifikátorů je pro efektivní programování zcela nesporný, ale také velkou měrou opomíjený. Mnohé publikace obsahují pouze několik frází o konvencích a je pak na vás, abyste se o sebe postarali sami. Názvy tříd (ale také proměnných) nelze volit jako jména domácích mazlíčků, tedy podle toho, zda vám znějí roztomile nebo že vám něco připomínají. Takový domácí mazlíček a jeho jméno, to jsou dvě samostatné entity. Ovšem třída a její název jsou v podstatě totéž. To znamená, že kvalita identifikátoru závisí ve velké míře na jejím názvu.

```
měsíčníSoučet = novénákupy + DaňZProdeje( novénákupy );
```

```
zůstatek =
```

```
zůstatek + PoplatekZaZpoždění( idZákazníka, zůstatek ) + měsíčníSoučet;
```


28 Co je při volbě názvu nejdůležitější



začátečník

Při pojmenování tříd a objektů je nejdůležitějším pravidlem vyčerpávající, ale zároveň co nejstručnější popis pojmu, který třída nebo objekt zastupuje. Účinnou technikou pro tvorbu dobrých názvů je výpis všech slov vyjadřujících zastoupenou entitu. Nejlepším názvem se často jeví pouhé uvedení těchto slov. Tato slova jsou nejsrozumitelnější, protože neobsahují tajemné a nesrozumitelné zkratky. Navíc jsou obvykle jednoznačná. Protože jde o úplný popis entity, nebudeme si třídu nebo objekt plést s ničím jiným. Kromě toho si název snadno zapamatujeme, protože název je shodný s účelem.

```
class ČtyřnohýSavecSParožím { /* sem vložíme definici těla třídy */ }
```

29 Rozlišujte mezi názvy typů a objektů



začátečník

Shoda mezi názvy typů a objektů může být ošidná. Existuje několik standardů, viz následující příklady. Pokuste se název objektu od názvu typu odlišit užitím konkrétnějšího názvu:

```
Pomůcka pracovníPomůcka;  
PracovníNástroj odbornýPracovníNástroj;
```

Při tomto způsobu pojmenování totiž musíte význam jednotlivých objektů velmi dobře promyslet. Ve většině případů je výsledkem úvahy o konkrétním názvu objektu srozumitelnější kód. Občas ale může být název `pomůcka` skutečně obecným objektem typu `Pomůcka`. Takže v takových případech byste museli vytvářet zbytečně názvy jako `obecnáPomůcka`, což programu na srozumitelnosti možná nepřidá. Takže tento tip použijte s náležitou rozvahou.

30 Nepoužívejte v názvech číslice



začátečník

Jsou-li číslice v názvu významné, měli byste raději místo samostatných objektů použít pole objektů. Ovšem pokud pole použít nelze, jsou číslice ještě méně vhodné. Neměli byste například používat názvy typu `objekt1` a `objekt2`. Téměř vždy byste se měli pokusit o nalezení jiného způsobu odlišení dvou objektů než pomocí číslic `1` a `2` na konci názvu.

31 Nepoužívejte názvy bez souvislosti s obsahem třídy nebo objektu



začátečník

Chcete-li zajistit, aby se ve vašem programu nikdo nevyznal, používejte názvy jako `Boo`, `Foo`, `BlaBlaBla` apod. Chcete-li však, abyste se v programu i vy sami orientovali, i když zdrojový kód neuvídíte třeba rok, tak pro označení tříd a objektů nepoužívejte jména kamarádů a kamarádek, ani oblíbených nápojů, pokud ovšem nevytváříte program například pro míchání nápojů. Ale ani v takovém případě nezapomínejte, že se váš vkus může po určité době změnit, takže vždy volte raději obecnější názvy typu `NealkoholickýNápoj` nebo `oblíbenéNealko` (jde-li o konkrétní instanci příslušné třídy nápojů).

32 Styl pojmenování v jazyce Java



začátečník

- Identifikátory `i` a `j` se používají jako celočíselné indexy v cyklech.
- Konstanty se píše `POUZE_VELKÝMI_PÍSMENY` a slova v názvech jsou oddělena podtržítky.
- Názvy tříd a rozhraní se píše malými i velkými písmeny, přičemž velkými písmeny začínají všechna dílčí slova – tedy i to první, viz `NázevTřídy` nebo `Rozhraní`.
- Názvy proměnných a metod začínají malým písmenem, ale v ostatních ohledech se řídí stejnou konvencí jako názvy tříd a rozhraní. Název proměnné lze zapsat takto: `názevProměnné` nebo `Rutiny`.
- Podtržítko se jako oddělovač slov nepoužívá. Výjimkou jsou názvy, jež se píše všemi velkými písmeny.
- Předpony `get` a `set` se používají pro přístupové metody (accessor methods).
- Názvy balíčků se píše malými písmeny: `cz.cpress.java.tipy`.

Nikdy ale nezapomínejte na to, že spotřebitel vašeho kódu bude muset všechny názvy (tedy i ty dlouhé) zapsat ve svém kódu také. Takže s ním mějte soucit a snažte se o přiměřeně stručné názvy.

33 Návrhové vzory pojmenování



začátečník

Při zjišťování jednoduchých vlastností komponenty se vyhledávají tyto typy metod:

```
public <Typ vlastnosti> get<Název vlastnosti>();
public void set<Název vlastnosti>(<Typ vlastnosti> a);
```

Pokud jsou nalezeny obě tyto metody, je zjištěno, že komponenta obsahuje atribut `<Název vlastnosti>` s typem `<Typ vlastnosti>`. Pokud je atribut typu `boolean`, jsou povoleny i metody tohoto typu:

```
public boolean is<Název vlastnosti>();
public void set<Název vlastnosti>(boolean a);
```

Indexované atributy mohou být reprezentovány těmito typy metod:

```
public <Typ vlastnosti>[] get<Název vlastnosti>();
public void set<Název vlastnosti>(<Typ vlastnosti>[] a);
public <Typ vlastnosti> get<Název vlastnosti>(int index);
public void set<Název vlastnosti>(int index, <Typ vlastnosti> a);
```

Při zjišťování zpráv, které daná komponenta může zasílat, se hledají tyto typy metod:

```
public void add<Typ posluchače>(<Typ posluchače> a);
public void remove<Typ posluchače>(<Typ posluchače> a);
```

Může-li v metodě `add<Typ posluchače>(<Typ posluchače> a)` vzniknout výjimka typu `java.util.TooManyListenersException`, může mít komponenta jednoho posluchače.

34 Viditelnost a jedinečnost názvů



začátečník

V každém programovacím jazyce se setkáte s problémy správy názvů. Jak budete rozlišovat mezi dvěma identifikátory, použijete-li nějaký název v rámci jednoho modulu programu a jiný programátor použije stejný název v rámci jiného modulu téhož programu?

V jazyce Java se těmto konfliktům můžete vyhnout novým přístupem. K vytváření jedinečných názvů knihoven použijte podobného přístupu jako v případě domén sítě Internet. Je-li vaše doména `www.jannovak.cz`, můžete balíček svých nástrojů pojmenovat `cz.jannovak.utility`. Každá vaše třída tak bude mít jedinečný název ve světovém měřítku.

35 Umístění nových tříd v balíčcích



V předchozích příkladech jsme pro maximální zjednodušení vytvořili novou třídu objektů v takzvaném bezejmenném balíčku. V Javě se totiž všechny třídy umísťují právě do balíčků. Balíček je v Javě skupina souvisejících tříd. Balíčky můžete vytvářet sami, ale Java samotná má k dispozici mnoho předdefinovaných balíčků, jako jsou třeba `java.lang`, `java.util` nebo `java.awt`. Pokud nový typ (novou třídu) explicitně neumístíte do nějakého balíčku, říká se, že je automaticky v implicitním bezejmenném balíčku. Název typu je tedy v programu dostupný bez nutnosti uvádět název balíčku.

Použití typu z implicitního bezejmenného balíčku:

```
UkázkováTřída obecnáUkázkováTřída;
```

Kdybychom ovšem pro tuto třídu vytvořili zvláštní balíček s názvem `cz.cpress.java.tipy`, museli bychom použít třídu takto:

```
cz.cpress.java.tipy.UkázkováTřída obecnáUkázkováTřída;
```

36 Informace o typu objektu 1



Ukázka využití tříd z balíčku `java.lang`.

Zjištění třídy objektu za běhu programu může být za určitých okolností velmi užitečné:

```
Class cls = object.getClass();
```

37 Informace o typu objektu 2



Ukázka využití tříd z balíčku `java.lang`.

Statická metoda `forName` vrací objekt typu `Class` přidružený k třídě nebo rozhraní určitého názvu. Takto získáme objekt zastupující typ `String`.

```
Class cls = Class.forName("java.lang.String");
```

38 Informace o typu objektu 3



Ukázka využití tříd z balíčku `java.lang`.

Následujícím způsobem také lze získat objekt zastupující typ `String`.

```
Class cls = java.lang.String.class;
```

39 Název třídy objektu



Ukázka využití tříd z balíčku `java.lang`.

Potřebujete-li zjistit název datového typu, můžete použít jeden z následujících dotazů.

```
// Dotaz na úplný název třídy.
Class cls = java.lang.String.class;
String name = cls.getName();           // java.lang.String

// Dotaz na úplný název vnitřní třídy.
cls = java.util.Map.Entry.class;
name = cls.getName();                   // java.util.Map$Entry

// Dotaz na neúplný název třídy.
cls = java.util.Map.Entry.class;
name = cls.getName();
if (name.lastIndexOf('.') > 0) {
    name = name.substring(name.lastIndexOf('.')+1); // Map$Entry
}
// Znak $ lze převést na tečku.
name = name.replace('$', '.');          // Map.Entry

// Dotaz na název primitivního typu.
name = int.class.getName();             // int

// Dotaz na název pole.
name = boolean[].class.getName();       // [Z
name = byte[].class.getName();          // [B
name = char[].class.getName();          // [C
name = short[].class.getName();         // [S
name = int[].class.getName();           // [I
name = long[].class.getName();          // [J
name = float[].class.getName();         // [F
name = double[].class.getName();        // [D
name = String[].class.getName();        // [Ljava.lang.String;
name = int[][].class.getName();         // [[I

// Dotaz na název hodnoty void.
cls = Void.TYPE;
name = cls.getName();                   // void
```

40 Odkud byla načtena třída



Ukázka využití tříd z balíčku `java.lang`.

Existuje mnoho případů, kdy je třeba přesně určit, odkud byla třída načtena. Chcete-li najít umístění definice příslušné třídy, postupujte takto:

```
// Dotaz na umístění dané třídy.
Class cls = this.getClass();
ProtectionDomain pDomain = cls.getProtectionDomain();
CodeSource cSource = pDomain.getCodeSource();
URL loc = cSource.getLocation();
```

41 Umístění tříd načtených pomocí systémového zaváděče



Umístění tříd načtených pomocí systémového zaváděče tříd nelze určit stejným způsobem, protože zdrojový kód třídy je pak vyhodnocen jako hodnota `null`. Jedinou další metodou je užití přepínače `-verbose` příkazu `java`. Tento přepínač způsobí, že JVM vypíše zprávu při každém načtení třídy.

```
> java -verbose MojeApplikace
```

Takto by mohl vypadat i váš výstup:

```
[Opened c:\jdk1.6\jre\lib\rt.jar]
[Opened c:\jdk1.6\jre\lib\sunrsasign.jar]
[Opened c:\jdk1.6\jre\lib\jsse.jar]
[Opened c:\jdk1.6\jre\lib\jce.jar]
[Opened c:\jdk1.6\jre\lib\charsets.jar]
[Loaded java.lang.Object from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.io.Serializable from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.Comparable from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.CharSequence from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.String from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.Class from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.Cloneable from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.ClassLoader from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.System from c:\jdk1.6\jre\lib\rt.jar]
[Loaded java.lang.Throwable from c:\jdk1.6\jre\lib\rt.jar]
```

42 Zjištění viditelnosti objektů libovolného typu



Ukázka využití tříd z balíčku `java.lang.reflect`.

Modifikátory umožňují zjistit o objektu typu `Class` mnohé zajímavé informace – například viditelnost objektu.

```
int mods = cls.getModifiers();
if (Modifier.isPublic(mods)) {
    // Tato třída je veřejná.
}
```

43 Zjištění viditelnosti datové složky



Ukázka využití tříd z balíčku `java.lang.reflect`.

Třídy `Field`, `Constructor` a `Method` jsou potomky třídy `Member`.

```
// Modifikátory datové složky.
int mods = member.getModifiers();
if (Modifier.isPublic(mods)) {
    // Tento člen je veřejný.
}
```

44 Zjištění předka testovaného typu



Ukázka využití tříd z balíčku `java.lang`.

Informace o předcích typu (třídy) jsou někdy velmi důležité. Proto se také na ně můžete zeptat následujícím způsobem:

```
Object o = new String();
Class sup = o.getClass().getSuperclass(); // java.lang.Object

// Objekt nemá žádného předka.
o = new Object();
sup = o.getClass().getSuperclass();      // null

// Přestože typem objektu o2 je rozhraní,
// vrací metoda getSuperclass() báзовou třídu objektu.
Runnable o2 = new Runnable() {
    public void run() {
    }
};
sup = o2.getClass().getSuperclass();     // java.lang.Object
```

45 Jak zjistit, zda má třída předka?



Ukázka využití tříd z balíčku `java.lang`.

Každá třída by měla mít předka. Tak je tomu u všech tříd v jazyce Java. Jak je tomu ale u třídy `Class`?

```
Class cls = java.lang.String.class;
Class sup = cls.getSuperclass(); // java.lang.Object

cls = java.lang.Object.class;
sup = cls.getSuperclass();      // null

// Předkem rozhraní je vždy hodnota null.
cls = java.lang.Cloneable.class;
sup = cls.getSuperclass();     // null

// Předkem primitivních typů je rovněž vždy hodnota null.
cls = int.class;
sup = cls.getSuperclass();     // null
```

46 Je to třída, nebo rozhraní?



Ukázka využití tříd z balíčku `java.lang`.

Potřebujete-li zjistit, zda daný objekt typu `Class` reprezentuje třídu, nebo rozhraní, použijte metodu `isInterface()`.

```
Class cls = java.lang.String.class;
boolean isClass = !cls.isInterface();
```

```
cls = java.lang.Cloneable.class;
isClass = !cls.isInterface();
```

47 Výpis bazových rozhraní daného rozhraní



Ukázka využití tříd z balíčku `java.lang`.

V určitých případech je vhodné zjistit, jaká bazová rozhraní byla použita k tvorbě té které třídy. Použijte k tomuto účelu metodu `getInterfaces()`.

```
Class cls = java.util.List.class;
Class[] intfs = cls.getInterfaces(); // java.util.Collection
```

48 Seznam rozhraní implementovaných danou třídou



Ukázka využití tříd z balíčku `java.lang`.

Potřebujete-li zjistit, jaká rozhraní jsou implementována v dané třídě, použijte následující postup:

```
Class cls = java.lang.String.class;
Class[] intfs = cls.getInterfaces();
// [java.lang.Comparable, java.lang.CharSequence, java.io.Serializable]

// Dotaz na rozhraní primitivního typu vrátí prázdné pole.
cls = int.class;
intfs = cls.getInterfaces(); // []
```

49 Jak zjistit umístění třídy v balíčku



Ukázka využití tříd z balíčku `java.lang`.

Potřebujete-li zjistit, ve kterém balíčku je příslušná třída uložena, postupujte takto:

```
Class cls = java.lang.String.class;
Package pkg = cls.getPackage();
String name = pkg.getName(); // java.lang
// Metoda getPackage() vrací hodnotu null
// u všech tříd v implicitním balíčku.
cls = MojeTrida.class;
pkg = cls.getPackage(); // null

// Metoda getPackage() vrací hodnotu null
// u všech primitivních typů a u pole.
pkg = int.class.getPackage(); // null
pkg = int[].class.getPackage(); // null
```

50 Zjišťování názvů členských objektů



Ukázka využití tříd z balíčku `java.lang.reflect`.

Na příkladu třídy `String` ukážeme, jak lze získat úplný i částečný název objektu.

```
Class cls = java.lang.String.class;
Method method = cls.getMethods()[0];
Field field = cls.getFields()[0];
Constructor constructor = cls.getConstructors()[0];
String name;
// Úplné názvy.
name = cls.getName();           // java.lang.String
name = cls.getName() + "."
        + field.getName();      // java.lang.String.CASE_INSENSITIVE_ORDER
name = constructor.getName();   // java.lang.String
name = cls.getName()+"."+method.getName(); // java.lang.String.hashCode

// Neúplné názvy.
name = cls.getName().substring(cls.getPackage().getName().length()+1);
name = field.getName();         // CASE_INSENSITIVE_ORDER
name = constructor.getName().substring(
        cls.getPackage().getName().length()+1); // String
name = method.getName();        // hashCode
```

51 Nalezení metod pomocí objektů typu `Method`



Ukázka využití tříd z balíčku `java.lang.reflect`.

Existuje hned několik způsobů, jak pomocí objektu typu `Method` zjistit a použít metody vybraného typu, což názorně předvedeme na příkladu třídy `String`.

```
Class cls = java.lang.String.class;

// Takto lze získat seznam všech deklarovaných metod.
Method[] methods = cls.getDeclaredMethods();

// Seznam všech veřejných metod: deklarovaných a zděděných.
methods = cls.getMethods();
for (int i=0; i<methods.length; i++) {
    Class returnType = methods[i].getReturnType();
    Class[] paramTypes = methods[i].getParameterTypes();
    zpracovat(methods[i]);
}

// Takto lze získat konkrétní objekt typu Method.
// V tomto příkladu získáme metodu String.substring(int).
try {
    Method method = cls.getMethod("substring", new Class[] {int.class});
    zpracovat(method);
} catch (NoSuchMethodException e) {}
```


52 Volání metod pomocí objektů typu Method



Ukázka využití tříd z balíčku `java.lang.reflect`.

Metodu libovolného objektu lze volat prostřednictvím metody `invoke()` definované ve třídě `Method`.

```
Object result = method.invoke(object,  
    new Object[] {argument1, argument2, ..., argumentN});
```

53 Jak lze pomocí objektu typu Class získat datové složky libovolného typu?



Ukázka využití tříd z balíčku `java.lang.reflect`.

Existují celkem tři způsoby, jak pomocí objektu typu `Field` definovaného v objektu typu `Class` získat datové složky libovolného typu. V tomto příkladu zjistíme požadované informace o datových složkách třídy `Point`.

```
Class cls = java.awt.Point.class;
```

```
// Jeden způsob předpokládá tvorbu seznamu všech
```

```
// deklarovaných datových složek.
```

```
Field[] fields = cls.getDeclaredFields();
```

```
// Dalším způsobem je tvorba všech veřejných datových složek
```

```
// - deklarovaných i zděděných.
```

```
fields = cls.getFields();
```

```
for (int i=0; i<fields.length; i++) {
```

```
    Class type = fields[i].getType();
```

```
    zpracovat(fields[i]);
```

```
}
```

```
// Načíst lze rovněž vybraný objekt typu Field.
```

```
// V tomto příkladu načteme datovou složku java.awt.Point.x.
```

```
try {
```

```
    Field field = cls.getField("x");
```

```
    zpracovat(field);
```

```
} catch (NoSuchFieldException e) {}
```

54 Nastavení nebo načtení hodnoty datové složky pomocí objektu typu Field



Ukázka využití tříd z balíčku `java.lang.reflect`.

V tomto příkladu předpokládáme, že datová složka je typu `int`.

```
try {
```

```
    // Zjištění hodnoty.
```

```
    field.getInt(object);
```

```
// Změna hodnoty.  
field.setInt(object, 123);  
  
// Načtení hodnoty statické datové složky.  
field.getInt(null);  
  
// Změna hodnoty statické datové složky.  
field.setInt(null, 123);  
} catch (IllegalAccessException e) {}
```

55 Zjištění informací o konstruktorech třídy



Ukázka využití tříd z balíčku `java.lang.reflect`.

Existují dva způsoby, jak zjistit konstruktor objektu pomocí metod instance typu `Class`.

```
// Načtením seznamu všech objektů typu Constructor.  
Constructor[] cons = cls.getDeclaredConstructors();  
for (int i=0; i<cons.length; i++) {  
    Class[] paramTypes = cons[i].getParameterTypes();  
    zpracovat(cons[i]);  
}  
  
// Načtením konkrétního objektu typu Constructor.  
// V tomto příkladu načteme konstruktor java.awt.Point(int, int).  
try {  
    Constructor con = java.awt.Point.class.getConstructor(  
        new Class[]{int.class, int.class});  
    zpracovat(con);  
} catch (NoSuchMethodException e) {}
```

56 Tvorba nového objektu pomocí instance typu Constructor



Ukázka využití tříd z balíčku `java.lang.reflect`.

Nyní vytvoříme nový objekt typu `Point` pomocí konstruktoru `Point(int, int)`.

```
java.awt.Point obj = (java.awt.Point)con.newInstance(  
    new Object[]{new Integer(123), new Integer(123)});
```

57 Zaručená inicializace datových složek objektu



Mnozí programátoři vytvářejí v každé třídě metodu `initialize()`. Z názvu vyplývá, že tuto metodu používají k inicializaci datových složek před prvním použitím objektu. Inicializovat datové úložiště objektu je nezbytné, ale existence takové metody znamená ovšem také nemalé riziko, že uživatel objektu tuto metodu zavolat zapomene. V jazyce Java se takovému riziku můžete vyhnout inicializací každého nového objektu tím, že definujete konstruktor. Vlastní-li třída konstruktor, zavolá jej Java ihned po vytvoření objektu, a to

ještě dříve, než s ním může uživatel cokoli udělat. Inicializace objektů a jeho datových složek je tedy zaručena.

V jazyce Java bylo při definici konstruktorů použito řešení z jazyka C++: název konstruktoru je shodný s názvem třídy. Překladač automaticky zajistí, aby tato „metoda“ byla volána automaticky při inicializaci každého nového objektu příslušného typu.

```
class UkázkováTřída {
    UkázkováTřída() { // To je konstruktor
        System.out.println("Inicializace třídy UkázkováTřída.");
    }
}

public class Program {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new UkázkováTřída();
    }
}
```

Po vytvoření nového objektu typu `UkázkováTřída` je automaticky vytvořeno úložiště a vykonán konstruktor. Tím je zaručeno, že objekt bude správně inicializován ještě dříve, než s ním začnete pracovat.

Konstruktory odstraňují mnoho problémů a významně zjednodušují výsledný programový kód. Výsledný kód je navíc mnohem odolnější chybám ze zbytečných opomenutí. V příkladu není žádné explicitní volání nějaké inicializační metody ve stylu `initialize()`, která by byla od definice koncepčně oddělena. V jazyce Java jsou definice a inicializace sjednoceny pojmy – jeden bez druhého prostě nemůže existovat. Tak by tomu mělo být v každém dobrém objektovém návrhu.

58 Parametrizovaná inicializace objektu



Konstruktory mohou přijímat, stejně jako ostatní metody, argumenty, které umožňují přesněji specifikovat, jak má být požadovaný objekt vytvořen. Předchozí příklad můžete snadno změnit tak, aby konstruktor přijímal argument:

```
class UkázkováTřída {
    UkázkováTřída(int i) { // To je konstruktor
        System.out.println("Inicializace třídy UkázkováTřída." + i);
    }
}

public class Program{
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new UkázkováTřída(i);
    }
}
```

Argumenty konstruktoru vám tedy umožní poskytovat parametry inicializace objektu.

59 Implicitní a explicitní konstruktory



Bude-li `UkázkováTřída(int)` jediný konstruktor příslušného typu, překladač vám nedovolí vytvořit objekt typu `UkázkováTřída` žádným jiným způsobem. Překladač v jazyce Java zastává názor, že si definujete jakýkoli vlastní konstruktor, nebo mu řekněte, aby pro vás jeden vytvořil. Pokud jste pro svůj typ nespecifikovali žádný konstruktor, překladač automaticky doplní definici typu o tzv. **implicitní** konstruktor. Pokud ale definujete jakýkoli konstruktor s argumenty, už nesmíte počítat s tím, že příklad jako ten následující bude fungovat. Překladač v takovém případě ohlásí, že nemůže najít žádný konstruktor, jenž by odpovídal tomuto zápisu. Má-li typ vlastní explicitně definovaný konstruktor, překladač uvažuje takto: „Programátor definoval vlastní konstruktor, takže asi ví, co dělá. Pokud nevytvořil žádný implicitní konstruktor, znamená to asi, že jej chce vynechat.“

```
class UkázkováTřída {
    UkázkováTřída(int i) { // To je konstruktor
        System.out.println("Inicializace třídy UkázkováTřída." + i);
    }
}

public class Program {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new UkázkováTřída();
    }
}
```

60 Konstruktor a argumenty



Konstruktory se od ostatních metod ničím neliší, a proto mohou také přijímat argumenty, které umožňují přesněji specifikovat, jak má být požadovaný objekt vytvořen. Pokud například třída `Strom` obsahuje konstruktor, který přijímá jeden argument datového typu celé číslo, udávající výšku stromu, mohli byste vytvořit objekt `Strom` následujícím příkazem:

```
Strom s = new Strom(4); // Vytvoříme 4metrový strom.
```

61 Konstruktor v jazyce Java



Konstruktory mají stejný název jako třída objektu, nemají žádnou návratovou hodnotu a mohou nepovinně mít libovolný počet argumentů.

Konstruktor je poněkud nezvyklým typem „metody“, protože nevrací žádnou hodnotu. Tím se významně liší od prázdné návratové hodnoty `void`, používané v definici metod, jež nevracejí nic. U metod totiž vždy máte možnost rozhodnout se, že nějakou hodnotu přece jen chcete volajícímu kódu vrátit. Konstruktory však nevracejí nic a nelze to nijak změnit. Pokud tedy u konstruktoru návratovou hodnotu přece jen použijete, bude ho překladač považovat za běžnou metodu, nikoli za konstruktor.

62 Konstruktory a metody. Je to totéž?



začátečník

Konstruktory však nejsou metodami, přestože jsou tak často prezentovány (obvykle se říká, že konstruktory jsou speciálními metodami). Voláním metody `Class.getMethods()` získáte pole instancí typu `Method`, zatímco volání metody `Class.getConstructors()` vrací pole instancí typu `Constructor`. Z toho vyplývá, že třídy `Constructor` a `Method` nelze zaměňovat (nejsou odvozeny jedna od druhé). Obě ovšem implementují rozhraní `Member`.

Konstruktor je jedinečným prvkem (i když jich třída obsahuje několik). Má stejný název jako třída, od metod se liší svou deklarací a dokonce nemá stejnou syntaxi jako metoda. Metodu z konstruktoru volat lze, ale naopak to možné není.

63 Konstruktor nesmí mít žádný návratový typ



začátečník

Podívejte se na následující příklad:

```
class Start {
    public void Start() {
        System.out.println("Konstruktor START");
    }
}

public class Test {
    public static void main(String[] args) {
        Start s = new Start();
    }
}
```

Problémem je definice návratového typu `void` v deklaraci konstruktoru. Kvůli tomu nevznikl konstruktor, nýbrž normální metoda, která má shodu okolností stejný název jako hostitelská třída. Odstraňte definici návratového typu a vše bude v pořádku.

64 Proč nelze deklarovat konstruktor jako konečný?



začátečník

Zajímá vás, proč nelze konstruktory deklarovat jako konečné? Klíčové slovo `final` (konečný) se používá u metod, které nelze v odvozených třídách překrývat. Vzhledem k tomu, že se konstruktory nedědí, a tedy nikdy nebudete mít možnost je překrývat, nemá klíčové slovo `final` v případě konstruktoru žádný praktický význam.

65 Zaručená inicializace pomocí konstruktoru



pokročilý

Představte si, že pro každou novou třídu vytváříte novou metodu nazvanou `initialize()`. Název napovídá, že tuto metodu je nutno zavolat ještě před prvním použitím objektu tohoto typu. Znamená to bohužel také i fakt, že uživatel třídy nesmí za žádných okolností zapomenout tuto metodu zavolat. V jazyce Java existuje lepší řešení – užití konstruktoru třídy. Název konstruktoru je vždy přesně shodný s názvem třídy. Konstruktor je volán automaticky při inicializaci každého nového objektu dané třídy. Vyzkoušejte si to:

```
class Start {
    Start() { // Tak to je konstruktor.
        System.out.println("Tvorba objektu typu Start.");
    }
}

public class Test {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new Start();
    }
}
```

66 Volání konstruktorů z konstruktorů



Chcete-li, aby měla třída několik konstruktorů, zajistě budete chtít, abyste v rámci minimalizace duplicit mohli volat jeden konstruktor z těla jiného konstruktoru. Využijte k tomuto účelu klíčové slovo `this`.

Klíčové slovo `this` má obvykle význam „tento objekt“ nebo „aktuální objekt“ a obsahuje odkaz na aktuální objekt. Uvedete-li však za ním seznam argumentů konstruktoru, využijete jeho jinou schopnost – explicitní volání konstruktoru, jehož seznam argumentů se přesně shoduje s příslušným voláním. Takto jednoduše můžete volat všechny zbývající konstruktory dané třídy:

```
class UkázkováTřída {
    int počet = 0;
    String text = new String("null");

    UkázkováTřída(int i) { // To je konstruktor
        System.out.println("Inicializace pomocí celého čísla." + i);
        počet += i;
    }

    UkázkováTřída(String informace) {
        System.out.println("Inicializace pomocí řetězce." + informace);
        text = informace;
    }

    UkázkováTřída(String informace, int i) {
        System.out.println("Inicializace pomocí řetězce a celého čísla.");
        this(i);
        text = informace;
    }
}
```

```

    UkázkováTřída() {
        this("ahoj", 2);
        System.out.println("Implicitní konstruktor.");
    }
}

public class Program {
    public static void main(String[] args) {
        for(int i = 0; i < 10; i++)
            new UkázkováTřída();
    }
}

```

67 Metoda finalize()



začátečník

Každá třída obsahuje metodu nazvanou `finalize()`, která je volána při konečném vymazání objektu z operační paměti. Objekty však nejsou v jazyce Java mazány z paměti explicitně běžícím programem. O jejich vymazání se stará automatický správce paměti (garbage collector). Speciální metoda `finalize()` tedy bude zavolána. Otázkou je kdy? Proto byste metodu `finalize()` neměli používat například k explicitnímu snižování počtu odkazů na objekty určitého typu, ale pouze k uzavření vázaných systémových prostředků, jako jsou síťová připojení, otevřené soubory apod.

68 Úklid pomocí metody finalize()



pokročilý

Předpokládejme, že si váš objekt rezervuje určitou část paměti bez klíčového slova `new`. Automatická správa paměti jazyka Java ví, jak uvolnit paměť rezervovanou pomocí klíčového slova `new`. Nebude si však vědět rady, jak uvolnit paměť rezervovanou jinak. Pro takové případy nabízí Java metodu nazvanou `finalize()`, kterou můžete definovat pro každou svou třídu. Jak ji používat? V okamžiku, kdy automatická správa paměti chce již neplatný objekt uvolnit z paměti, zavolá nejprve jeho metodu `finalize()`. Paměť rezervovanou pro tento objekt klíčovým slovem `new` uvolní až po následném testu kolekce neplatných dat. Budete-li tedy používat metodu `finalize()`, neunikne vám ani jeden bajt paměti.

69 Metoda finalize() není destruktork



pokročilý

Nezaměňujte však metodu `finalize()` s destruktorem. Destruktor je volán při každém vymazání objektu. Je však nesmírně důležité, abyste si uvědomili, že na rozdíl od jazyka C++ nemusí v jazyce Java být objekty vždy automatickou správou paměti zachyceny. Nebo jinak: zachycení neplatných objektů a dat ještě nemusí nutně vést k jejich vymazání z paměti.

70 V jazyce Java nejsou destruktory



Budete-li si to pamatovat, nebudete mít s úklidem žádné problémy. Musí-li být něco uklizen, když už to nepotřebujete, musíte to uklidit vědomě sami. Java totiž neobsahuje destruktory nebo jemu podobný mechanismus. K úklidu musíte použít běžnou metodu. Pokud objekt něco vypisuje na obrazovku a vy nechcete, aby to na obrazovce zůstalo navždy, musíte například do metody `finalize()` vložit kód, jenž se postará o vymazání obrazovky. Pak bude po zachycení již neplatného objektu automatickou správou paměti vymazán nejprve text z obrazovky a teprve pak bude vymazán objekt z paměti.

71 Objekty nemusí být vymazány z paměti



Neplatné objekty nemusí být automatickou správou paměti uklizeny! Objekty mohou trvale zůstat v paměti, protože váš běžící program ještě nevyčerpal dostupnou paměť počítače. K uvolnění paměti může dojít až po ukončení programu. Obecně je to dobré řešení, ale může také znamenat velké zatížení hostitelského systému. Chcete-li úklid urychlit, použijte metodu:

```
System.gc();
```

Metoda `System.gc()` se používá k vynucení úklidu paměti a chcete-li zkrátit dobu ladění, používejte ji zejména při návrhu nových aplikací.

72 Singleton – jediný objekt daného typu v celém programu



Singleton je třída, která vytváří statickou instanci inicializovanou pouze jednou při prvním použití.

```
// První instance se vytvoří při prvním volání funkce getInstance()
// příklad použití:
// Singleton nameSingleton = Singleton.getInstance();
// String mojeJméno = nameSingleton.getName();
```

```
public class Singleton {

    // v JVM bude existovat pouze jedna instance
    private static Singleton instance = null;
    private Singleton1() {}

    public static synchronized Singleton getInstance() {
        if (instance == null) {
            instance = new Singleton();
        }
        return instance;
    }
}
```



```
public static String getName() {  
    String name = "Frodo";  
    return name;  
}  
}
```

73 Zpřístupnění tříd – import



začátečník

Kdykoli chcete použít ve svém programu některou z předdefinovaných tříd, musí překladač vědět, kde ji má hledat. Samozřejmě – třída může existovat také ve stejném zdrojovém souboru, z něhož je volána. V tomto případě ji můžete používat přímo. Jak ale budete pracovat s třídou umístěnou v jiném souboru nebo jmenném prostoru? Importujte ji do programu pomocí direktivy `import`.

```
import java.util.ArrayList;
```

Knihovna `java.util` obsahuje mnoho užitečných tříd. Klidně se může stát, že jich budete chtít používat více, aniž byste je museli výslovně deklarovat. Jak je importovat všechny? Příkazem:

```
import java.util.*;
```

74 Vlastnosti a přístupové metody



začátečník

Každý objekt je vybaven určitou množinou vlastností. Určité vlastnosti lze definovat jednoduše pomocí veřejných datových složek. Tento postup se však nedoporučuje, protože se do budoucna zbavíte možnosti eventuální změny implementace. V případě, že chcete implementaci vlastnosti ukrýt a ponechat si do budoucna možnost změn, ukryjte data za přístupovými metodami. Názvy přístupových metod by měly začínat předponami `get` a `set`, za něž připojíte název příslušné vlastnosti. Je-li vlastností objektu hmotnost, budete mít přístupové metody `setWeight()` a `getWeight()`:

```
public class SampleClass {  
    // chráněná datová složka, dostupná pouze v odvozených potomcích.  
    protected Weight weight;  
  
    // Vracení hodnoty vlastnosti.  
    public Weight getWeight() {  
        return weight;  
    }  
  
    // Nastavení hodnoty vlastnosti.  
    public void setWeight(Weight weight) {  
        this.weight = weight;  
    }  
}
```

75 Klíčové slovo final



Pokud v jazyce Java vložíte před signaturu metody klíčové slovo `final`, zajistíte, že žádný z potomků dané třídy už implementaci této metody nepřekryje.

76 Klíčové slovo return



Klíčové slovo `return` má dva významy: určuje návratovou hodnotu metody (pokud metoda nevrací prázdnou hodnotu (`void`)), ale také přerušuje a následně ukončuje metodu.

77 Překrývání metod předka



Aby mohl potomek překrýt (předefinovat) jakoukoli metodu svého předchůdce (předka), musí poskytovat metodu s naprosto stejnou signaturou, jakou má příslušná metoda jeho předchůdce. Signaturu metody tvoří název operace, typ návratové hodnoty a typy všech stejně seřazených argumentů. Názvy argumentů nejsou důležité, protože slouží pouze jako pohodlný způsob odkazu na specifický argument v těle metody. Názvy argumentů tedy nejsou součástí signatury metody. Změnou názvů argumentů ke změně signatury metody nedojde.

Avšak pozor! V jazyce Java sice není návratový typ metody součástí signatury metody, ale přesto jej při překrývání změnit nesmíte. Překryjete-li tedy metodu předka metodou potomka, která se liší pouze návratovým typem, nevznikne nová metoda, ale překladač jednoduše ohlásí chybu!

78 Přetížené metody



Jak překladač jazyka Java rozlišuje mezi stejnojmennými metodami? Na základě jedinečného seznamu argumentů a jejich datových typů. Stejnojmenné metody s různými argumenty se nazývají metodami přetíženými.

79 Rozdíl mezi proměnnou CLASSPATH a příkazem import



Proměnná `CLASSPATH` je proměnnou operačního systému, která sděluje virtuálnímu stroji jazyka Java, kde má hledat příslušné třídy. Příkaz `import` umožňuje vyhledat konkrétní třídu mezi třídami specifikovanými proměnnou `CLASSPATH`. Z toho vyplývá, že importovaný balíček musí být nalezen prostřednictvím proměnné `CLASSPATH`.

80 Načtení třídy neuvedené v proměnné CLASSPATH



Ukázka využití tříd z balíčku `java.lang`.

K načtení třídy z libovolného adresáře lze použít třídu `URLClassLoader`.

```
// Vytvořte objekt typu File obsahující odkaz na kořenový adresář
// obsahující soubor třídy
File soubor = new File("c:\\vlastnitridy\\");
```

```

try {
    // Přetypování objektu z typu File na URL.
    URL url = soubor.toURL();           // file:/c:/vlastnitridy/
    URL[] urls = new URL[]{url};

    // Tvorba nového zavadače tříd pro daný adresář.
    ClassLoader cl = new URLClassLoader(urls);

    // Načtení třídy. Soubor MojeTrida.class by měl být umístěn v adresáři
    // file:/c:/vlastnitridy/com/mojefirma
    Class cls = cl.loadClass("com.mojefirma.MojeTrida");
} catch (MalformedURLException e) {
} catch (ClassNotFoundException e) {}

```

81 Indexované vlastnosti



Chcete-li jako vlastnost objektu použít indexovanou proměnnou nebo pole, lze použít nejen metody pro přístup na celou proměnnou, ale i jednotlivé prvky indexované proměnné. Například pokud bude vlastností pole `Dimension[] dimensions`, lze použít tyto přístupové metody:

```

Dimension[] getDimensions();
void setDimensions(Dimension[] dimensions);
Dimension getDimension(int index);
void setDimension(int index, Dimension dimension);

```

82 Přiřazení hodnoty instanci typu Object



Ukázka využití tříd z balíčku `java.util`.

Příklad ukazuje, jak přidružit hodnotu k libovolnému objektu. Tato technika vyžaduje uložení objektu a přidružené hodnoty jako položky ve tvaru `klíč=hodnota` v objektu typu `IdentityHashMap`. Objekt typu `HashMap` k tomuto účelu použít nelze, protože kdyby byly dva objekty považovány metodou `Object.equals()` za rovné, byl by uložen jen jeden z nich.

```

// Vytvořte mapu.
Map objMap = new IdentityHashMap();

// Přidejte do mapy objekt a hodnotu.
Object o1 = new Integer(123);
Object o2 = new Integer(123);
objMap.put(o1, "první");
objMap.put(o2, "druhá");

// Teď můžete načíst hodnotu přidruženou k objektu.
Object v1 = objMap.get(o1);    // první
Object v2 = objMap.get(o2);    // druhá

```

83 Tvorba duplikátů



Ukázka využití tříd z balíčku `java.lang`.

Existují případy, kdy je třeba vytvořit duplikát pracovního objektu, jenž ovšem nebude sdílet společná data s originálem.

Předpokládejme, že máte tuto třídu:

```
class VlastniClass implements Cloneable {
    public VlastniClass() {}
    public Object clone() {
        Cloneable duplikat = new VlastniClass();
        // Inicializace duplikátu
        return duplikat;
    }
}
```

Následující ukázky ukazují tvorbu duplikátů:

```
VlastniClass vlastniObject = new VlastniClass();
VlastniClass vlastniObjectClone = (VlastniClass)vlastniObject.clone();
```

Pole vždy umožňuje tvorbu duplikátů:

```
int[] ints = new int[]{123, 234};
int[] intsClone = (int[])ints.clone();
```

84 Zapouzdření primitivního typu do objektově orientovaného reprezentanta



Ukázka využití tříd z balíčku `java.lang`.

V jazyce Java existuje osm primitivních typů, jež nejsou objekty – `boolean`, `byte`, `char`, `short`, `int`, `long`, `float` a `double`. V určitých situacích si však s nimi nevystačíte, neboť z různých důvodů může implementace vyžadovat užití objektových místo primitivních typů. Například třídy kolekce, jako jsou `Map` a `Set`, pracují výhradně s objekty. Primitivní typ je pak třeba zapouzdřit do jiného objektu. Pro každý primitivní typ však existuje speciální zapouzdřující objekt. V tomto příkladu ukážeme způsob, jak zapouzdřit hodnotu primitivního typu do zapouzdřujícího objektu a jak následně hodnotu primitivního typu použít.

```
// Vytvořte zapouzdřující objekt pro každý primitivní typ.
Boolean refBoolean = new Boolean(true);
Byte refByte = new Byte((byte)123);
Character refChar = new Character('x');
Short refShort = new Short((short)123);
Integer refInt = new Integer(123);
Long refLong = new Long(123L);
Float refFloat = new Float(12.3F);
Double refDouble = new Double(12.3D);
```

```
// Nyní můžete hodnoty načíst zpět.  
boolean bool = refBoolean.booleanValue();  
byte b = refByte.byteValue();  
char c = refChar.charValue();  
short s = refShort.shortValue();  
int i = refInt.intValue();  
long l = refLong.longValue();  
float f = refFloat.floatValue();  
double d = refDouble.doubleValue();
```

85 Změna typu hodnoty



začátečník

Ukázka využití tříd z balíčku `java.lang`.

Chcete-li změnit datový typ určité hodnoty, umístěte požadovaný datový typ do závorek (včetně všech modifikátorů) nalevo od konvertované hodnoty

```
int i = 200;  
long l = (long)i;  
long l2 = (long)200;
```

Přetypovat lze jak proměnné, tak i číselné hodnoty. V obou případech je přetypování zbytečné, neboť překladač automaticky povýší datový typ `int` na datový typ `long`. Přesto však raději tato explicitní přetypování používejte, protože váš kód pak bude mnohem čitelnější.

86 Přetypování datových typů



začátečník

Datové typy jako takové přetypování neumožňují. Chcete-li změnit jednu třídu na jinou, musíte k tomu mít připraveny speciální metody. Objekty lze navíc přetypovávat jen v rámci jedné rodiny; například `Dub` lze přetypovat na `Strom` a naopak, nelze jej však přetypovat na cizí typ, jako je třeba `Kočka`.

87 Přetypování primitivních typů



začátečník

Java umožňuje přetypovat data na libovolný primitivní datový typ kromě datového typu `boolean`, který jako jediný neumožňuje přetypování vůbec.

88 Zvláštnost datového typu Boolean



začátečník

Java neumožňuje používat čísla jako datový typ `boolean`. Chcete-li v booleovském testu jako `if(x)` používat nebooleovské hodnoty, musíte je nejprve převést na booleovský výraz jako `if(x != 0)`.

Dialogy a formuláře

89 Návrh dialogů a formulářů v IDE

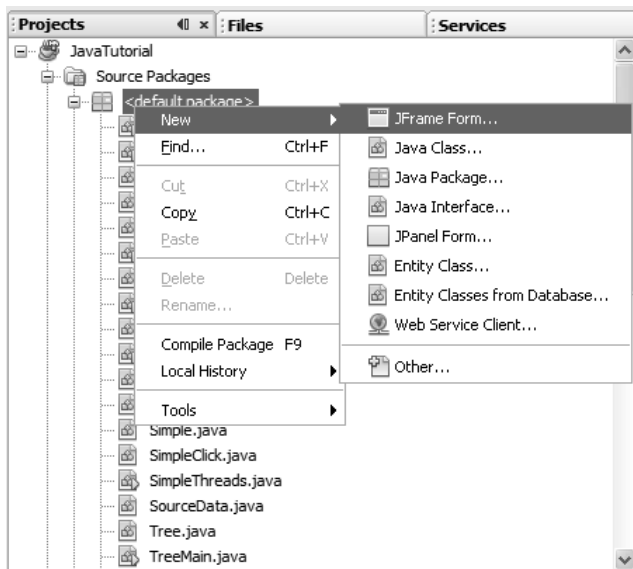


začátečník

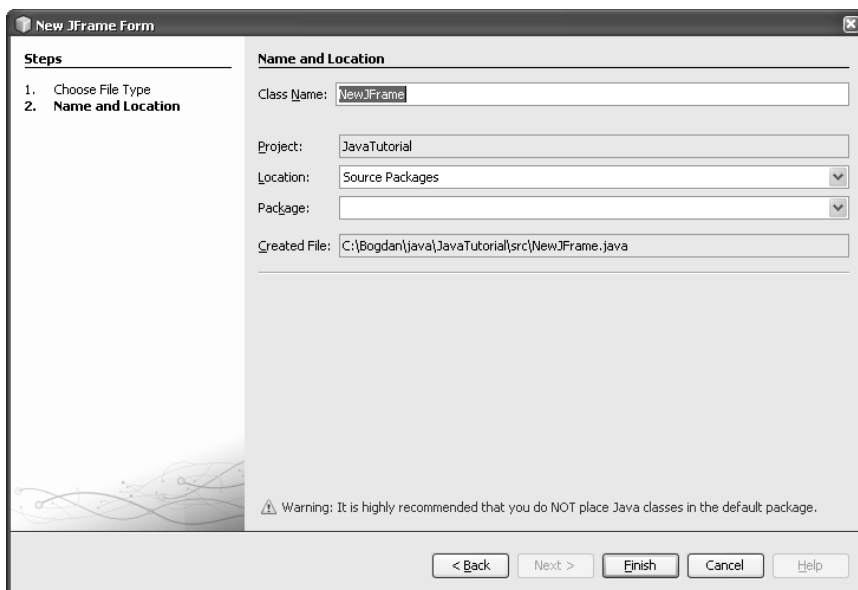
Formuláře a dialogy už dnes nemusíte vytvářet pouze v řádkovém rozhraní. Moderní vývojová prostředí pro jazyk Java umožňují vizuální návrh uživatelského rozhraní aplikace, včetně návrhu oken, formulářů a dialogů.

Používáte-li například vývojové prostředí NetBeans, můžete definici nového formuláře (potomka třídy `javax.swing.JFrame`) navrhnout vizuálně takto:

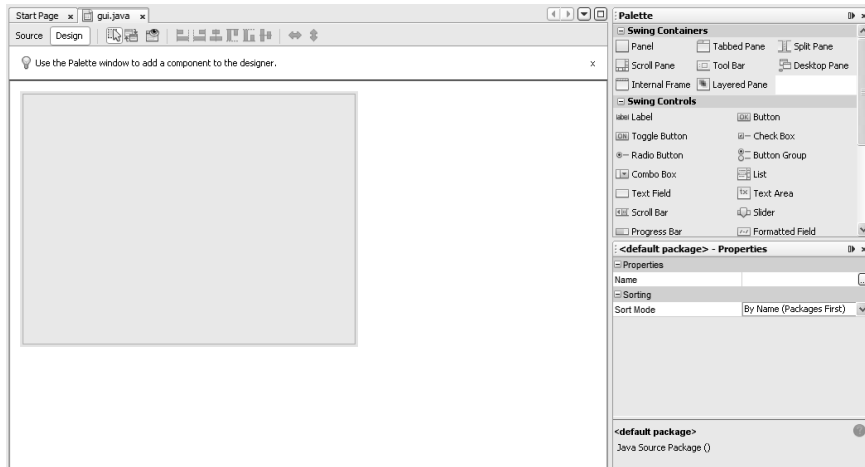
1. V podokně projektu klepněte pravým tlačítkem myši na uzlu požadovaného balíčku (na obrázku je použit implicitní balíček) a z místní nabídky nejprve vyberte položku **New** a pak **JFrame Form...**



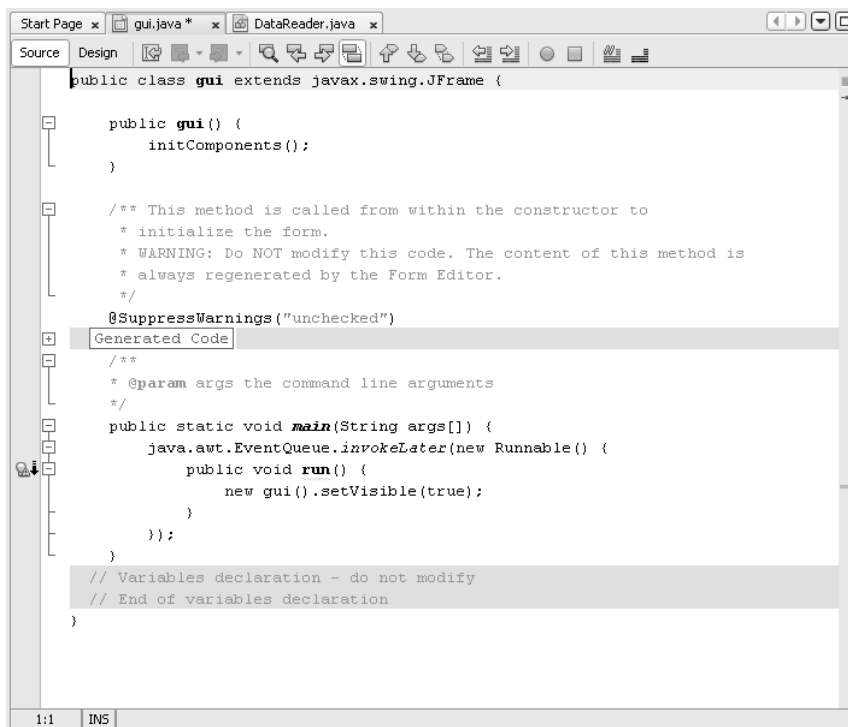
2. V zobrazeném průvodci zadejte název nové třídy a vyberte její umístění:



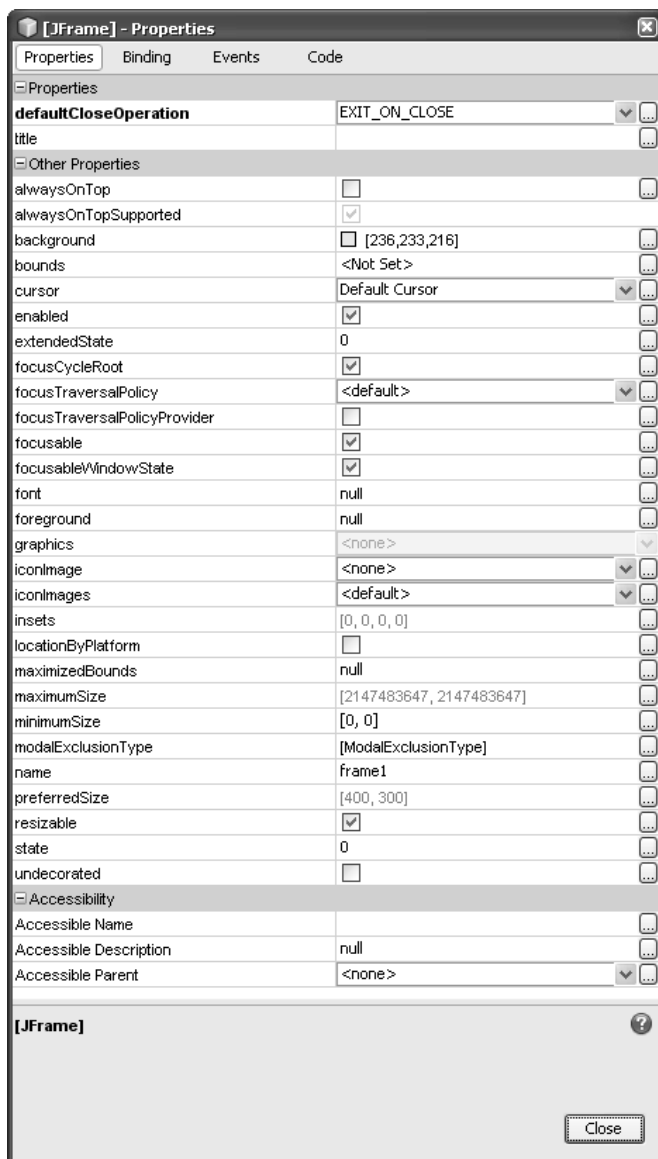
3. Výsledek pak spatříte v podokně návrhu:



4. Když se přepnete do podokna kódu, můžete pohodlně upravit kód nové třídy a přitom v oddílu `Generated Code` najdete veškerý kód nezbytný k inicializaci formuláře.



5. Mnoho užitečných vlastností formuláře pak můžete nastavit pomocí okna vlastností, jež zobrazíte po klepnutí pravým tlačítkem myši na ploše formuláře a po výběru položky **Properties**.



90 Hlavní okna



Hlavní okna jsou okna s titulkem a s okrajem. Rozměry hlavního okna zahrnují také oblast vymezenou pro okraje. Rozměry oblasti využitě pro okraje lze získat pomocí metody `getInsets()`. Jelikož je oblast obsazená okrajem zahrnuta do celkových rozměrů okna, okraj ve skutečnosti zakrývá část okna a ubírá z oblasti dostupné pro překreslování a zob-

razování komponent. Oblast pro umístování komponent je vymezena body `insets.left`, `insets.top`, šířka - (`insets.left` + `insets.right`) a výška - (`insets.top` + `insets.bottom`).

Instance třídy `JFrame` jsou okna s dekoracemi, jako jsou okraje, titulek, systémová nabídka, ikony pro minimalizaci, maximalizaci a obnovení okna. Aplikace s grafickým uživatelským rozhraním obsahují alespoň jedno takové okno. Hlavní okna lze využívat také v `apletech`.

Chcete-li vytvořit závislé okno (například takové, které bude skryto při minimalizaci nadřazeného okna), použijte instanci třídy `JDialog`.

Chcete-li vytvořit okno, jež bude zobrazeno uvnitř jiného okna, použijte instanci třídy `JInternalFrame`. Nadřazené okno v takových případech bývá instancí třídy `JDesktopPane`, jež je potomkem třídy `JLayeredPane`, obsahující API pro správu kolekce překryvných vnitřních oken.

91 Rozdíl mezi třídami `Frame` a `Canvas`



Ukázka využití tříd z balíčků `java.awt` a `javax.swing`

Instance třídy `Frame` (nebo v knihovně `Swing` `JFrame`) reprezentují okna s okraji, s tlačítky **Minimalizovat**, **Maximalizovat**, **Zavřít** a nemohou obsahovat další prvky, jako jsou řádek nabídek, tlačítka, panely apod. Instance třídy `Canvas` (nebo v knihovně `Swing` `JCanvas`) je ovládacím prvkem uživatelského rozhraní (vkládaným například do okna), na nějž lze kreslit.

92 Tvorba a zobrazení oken



Jako `frame` (frejm) se obvykle označuje okno, které zpravidla obsahuje titulek, systémová tlačítka pro minimalizaci, maximalizaci a obnovení okna a systémovou nabídku s ikonou.

Následující obrázek a kód ukazují nejjednodušší hlavní okno.



```
import java.awt.*;
import javax.swing.*;

public class Frame1
{
    public static void main(String[] args) {
        // Tvorba instance okna, s níž budeme dále pracovat.
        JFrame frame = new JFrame("Okno");

        // Nepovinná definice toho, co se stane, až zavřeme okno.
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Tvorba komponent a jejich přidání na plochu okna.
        // (V tomto případě jsme vytvořili prázdný popisek.)
        frame.getContentPane().add(
            new JLabel("Popisek", JLabel.CENTER), BorderLayout.CENTER);
    }
}
```

```
// Nastavení velikosti okna.
// Tato metoda upraví velikost okna tak, aby byly zobrazeny
// všechny vložené komponenty.
frame.pack();

// Zobrazení okna
frame.setVisible(true);
}
}
```

93 Zavření okna



začátečník

Při zavření okna dojde obvykle pouze k ukrytí okna, tj. že okno zmizí z povrchu obrazovky. Přestože je neviditelné, stále existuje a program je může kdykoli zobrazit (zviditelnit). Chcete-li, aby se okno při zavření chovalo jinak, musíte registrovat posluchač, který bude očekávat na událost zavření okna. Můžete také reakci na zavření okna definovat pomocí metody `setDefaultCloseOperation()`. Můžete ale rovněž použít oba přístupy zároveň.

Metoda `setDefaultCloseOperation()` očekává argument, jehož pomocí můžete metodě předat jednu z následujících hodnot:

- `DO_NOTHING_ON_CLOSE` – nestane se nic (program by měl použít posluchač okna, který vykoná požadovanou akci ve své metodě `windowClosing()`),
- `HIDE_ON_CLOSE` – ukryje okno (okno lze opět v programu zobrazit),
- `DISPOSE_ON_CLOSE` – ukryje okno a odstraní všechny prostředky, které byly k tvorbě okna použity (okno nelze v programu dále používat; pokud jde o jediné okno aplikace, má stejný význam jako hodnota `EXIT_ON_CLOSE`),
- `EXIT_ON_CLOSE` – ukončí aplikaci voláním `System.exit(0)`. Tento postup se doporučuje pouze u aplikací. Je-li použit v apletu, dojde k výjimce typu `SecurityException`.

94 Ukončení aplikace pomocí systémového tlačítka Zavřít



pokročilý

Ukázka využití tříd z balíčku `java.awt`.

Implicitně se po klepnutí na systémové tlačítko **Zavřít** nestane nic. Aplikace běží dál. V tomto příkladu ukážeme, jak po této události běh aplikace ukončit.

```
Frame frame = new Frame();

// Přidejte posluchače události Zavřít.
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent evt) {
        // Ukončení aplikace.
        System.exit(0);
    }
});
```

95 Ukrytí hlavního okna pomocí systémového tlačítka Zavřít



pokročilý

Ukázka využití tříd z balíčku java.awt.

Implicitně se po klepnutí na systémové tlačítko **Zavřít** nestane nic. V tomto příkladu ukážeme, jak po této události hlavní okno ukrýt.

```
Frame frame = new Frame();

// Přidání posluchače události Zavřít.
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent evt) {
        Frame frame = (Frame)evt.getSource();

        // Ukrýt okno.
        frame.setVisible(false);

        // Je-li okno nepotřebné, můžeme se ho zbavit.
        frame.dispose();
    }
});
```

96 Zobrazení okna uprostřed obrazovky



pokročilý

Ukázka využití tříd z balíčku java.awt.

Dialogy a formuláře se implicitně v GUI zobrazují v levém horním okně obrazovky. Chcete-li, aby se dialog nebo formulář zobrazil uprostřed, použijte tento kód:

```
private void center() {
    Dimension screenDim = Toolkit.getDefaultToolkit().getScreenSize();
    setLocation(((screenDim.width - getSize().width) / 2),
        ((screenDim.height - getSize().height) / 2));
}
```

Tento kód můžete volat kdykoli po vykonání metody `pack()`, `setSize()` nebo `setBounds()`, ale před voláním metody `setVisible()`.

97 Nastavení maximálních rozměrů okna



pokročilý

Ukázka využití tříd z balíčku java.awt.

Maximalizované hlavní okno aplikace implicitně vyplní celou plochu obrazovky. V tomto příkladu ukážeme, jak lze upravit rozměry a umístění maximalizovaného okna.

```
Frame frame = new Frame();

// Specifikace umístění a rozměrů maximalizovaného okna.
int x = 100;
```

```
int y = 100;
int width = 300;
int height = 300;
Rectangle bounds = new Rectangle(x, y, width, height);
```

```
// Nastavení maximalizovaných okrajů.
frame.setMaximizedBounds(bounds);
```

98 Zákaz změn rozměrů hlavního okna aplikace



pokročilý

Ukázka využití tříd z balíčku `java.awt`.

Rozměry hlavního okna aplikace lze implicitně měnit. Avšak například pomocí metody `setResizable(false)` tomu lze velmi úspěšně zabránit.

```
Frame frame = new Frame();
frame.setResizable(false);
```

```
// Následující metodou zjistíme aktuální možnosti změny rozměrů okna.
boolean resizable = frame.isResizable();
```

99 Změna ikony okna



pokročilý

Ukázka využití tříd z balíčku `javax.swing`.

Chcete-li změnit ikonu u systémové nabídky okna nebo dialogu, použijte jednoduchou metodu `setIconImage()`:

```
JFrame frame = new JFrame("Okno");
java.net.URL imgURL = SimpleFrame.class.getResource("background.jpg");
if (imgURL != null) {
    frame.setIconImage(new ImageIcon(imgURL).getImage());
}
```

100 Změna ikony okna (2)



pokročilý

Ukázka využití tříd z balíčku `java.awt`.

Hlavní ikona aplikace neboli ikona na hlavním okně aplikace je ikona, která se zobrazí v levém rohu titulku hlavního okna a v tlačítku na hlavním panelu systému Windows. Standardně je použit šálek s kouřící kávou. Vy však můžete velmi snadno zobrazit libovolnou vlastní ikonu.

```
// Nejprve vytvoříme hlavní okno aplikace.
String title = "Titulek okna";
Frame frame = new Frame(title);
```

```
// Nastavení ikony.
Image icon = Toolkit.getDefaultToolkit().getImage("ikona.gif");
frame.setIconImage(icon);
```

101 Odstranění záhlaví okna



Ukázka využití tříd z balíčku `java.awt`.

Titulek, stejně jako další ozdoby, lze z hlavního okna odstranit. Takové okno pak vypadá a chová se jako kontejner typu `Window`. Dekorace lze z okna odebrat pouze v případě, že okno není zobrazeno!

```
Frame frame = new Frame();
frame.setUndecorated(true);

// Touto metodou zjistíte stav zobrazení dekorací.
boolean undecorated = frame.isUndecorated();
```

102 Exkluzivní celoobrazovkový režim



Programátoři, kteří mají zkušenosti s rozhraním Microsoft DirectX API, se s pojmem exkluzivní celoobrazovkový režim setkali. Pro ostatní to však obvykle bývá nový pojem. Od verze J2SE 1.4 je to však velmi efektivní sada nástrojů, jež umožňují programátorům obejít systém oken a kreslit přímo na obrazovku.

Exkluzivní celoobrazovkový režim lze používat skrze objekt typu `java.awt.GraphicsDevice`. Seznam obrazovkových zařízení (v systémech s jedním nebo několika monitory) získáte pomocí metody `getScreenDevices()` volané pro objekt `java.awt.GraphicsEnvironment`. V případě primární (výchozí) obrazovky (jediné obrazovky v systému) můžete volat metodu `getDefaultScreenDevice()`.

Jakmile máte k dispozici grafické zařízení, můžete volat následující metody:

```
public boolean isFullscreenSupported()
public void setFullscreenWindow(Window w)
```

První metodou zjistíte, zda je celoobrazovkový režim v hostitelském systému dostupný. Pokud dostupný není, je lepší použít systém oken s hlavním oknem o rozměrech shodných s rozměry obrazovky. Druhá metoda zahájí práci s určeným oknem v exkluzivním celoobrazovkovém režimu. Je-li tento režim podporován, okno je umístěno na pozici (0, 0) a má rozměry celé obrazovky. Celoobrazovkový režim lze ukončit voláním metody `setFullscreenWindow()` s argumentem `null`:

```
GraphicsDevice device;
Window okno;
try {
    device.setFullscreenWindow(okno);
} finally {
    device.setFullscreenWindow(null);
}
```

103 Celoobrazovkový režim pomocí běžných oken



U většiny aplikací, jež mají být používány v celoobrazovkovém režimu, je lepší využít okna bez dekorací. V takovém případě vypněte dekorace hlavního okna nebo dialogu pomocí metody `setUndecorated()`.

104 Celoobrazovkový režim a změna rozměrů okna



Používáte-li ve své aplikaci celoobrazovkový režim, nezapomeňte, že hlavní okno aplikace nesmí umožňovat změnu rozměrů, protože to může způsobit nepředvídatelné (a pravděpodobně také nebezpečné) chování.

105 Oprávnění pro celoobrazovkový režim apletů



Chcete-li v apletu použít exkluzivní celoobrazovkový režim, musí uživatel z bezpečnostních důvodů udělit apletu oprávnění `fullScreenExclusive`.

106 Okno na celou obrazovku bez okrajů a titulku



Ukázka využití tříd z balíčku `javax.swing`.

Chcete *zobrazit* instanci typu `JFrame` na celé obrazovce ihned po spuštění programu, bez okrajů a bez titulku? Použijte raději třídu `JWindow`, jejíž instance lze upravit tak, aby se nezobrazily ani okraje, ani titulek.

107 Okno na celou obrazovku



Ukázka využití tříd z balíčku `java.awt`.

Okno vyplní celou obrazovku po vykonání následujícího příkazu:

```
setBounds(GraphicsEnvironment.getLocalGraphicsEnvironment().  
    getDefaultScreenDevice().getDefaultConfiguration().getBounds());
```

108 Diagnostika možností průhledných a tvarovaných oken



Ukázka využití tříd z balíčku `com.sun.awt.AWTUtilities`.

Protože balíček `com.sun.awt.AWTUtilities`, který budeme používat v následujících příkladech, je považován za proprietární (může být v budoucnu změněn nebo dokonce zcela odstraněn), je třeba před užitím speciálních funkcí pro tvorbu oken různých tvarů nebo průhledných či poloprůhledných oken ověřit, zda jsou v hostitelském systému k dispozici všechny nezbytné funkce.

Testovat můžete takto:

```
import java.awt.*;

public class TestOken {

    static void DiagnostikaUtilitAWT() {
        System.out.println("Výpis možností systému:\n");
        GraphicsEnvironment ge =
            GraphicsEnvironment.getLocalGraphicsEnvironment();
        GraphicsDevice[] gs = ge.getScreenDevices();
        System.out.println("\tLze vytvářet okna různých tvarů: "
            + com.sun.awt.AWTUtilities.isTranslucencySupported(
                com.sun.awt.AWTUtilities.Translucency.PERPIXEL_TRANSPARENT));
        System.out.println("\tLze vytvářet průhledná okna: "
            + com.sun.awt.AWTUtilities.isTranslucencySupported(
                com.sun.awt.AWTUtilities.Translucency.TRANSLUCENT));
        System.out.println("\tLze vytvářet průhledná okna různých tvarů: "
            + com.sun.awt.AWTUtilities.isTranslucencySupported(
                com.sun.awt.AWTUtilities.Translucency.PERPIXEL_TRANSLUCENT));

        for (int j = 0; j < gs.length; j++) {
            GraphicsDevice gd = gs[j];
            GraphicsConfiguration[] gc = gd.getConfigurations();
            for (int i = 0; i < gc.length; i++) {
                System.out.println("zařízení " + j + ", konfigurace " + i);
                System.out.println("\tUmožňuje průhlednost: "
                    + com.sun.awt.AWTUtilities.isTranslucencyCapable(gc[i]));
            }
        }
    }

    public static void main(String[] args) {
        TestOken.DiagnostikaUtilitAWT();
    }
}
```

109 Průhledná a polopřůhledná okna



Ukázka využití tříd z balíčku `com.sun.awt.AWTUtilities`.

Chcete-li, aby uživatel měl možnost skrze okno vidět, co je za ním (zejména, je-li okno používáno jako informační panel s menší důležitostí), můžete nastavit jeho průhlednost. Míru průhlednosti nastavte v procentech.

V příkladu je využit speciální balíček `com.sun.awt.AWTUtilities`. Jsou-li v aplikaci využity funkce tohoto balíčku, zobrazí překladač následující varování:

```
warning: com.sun.awt.AWTUtilities is Sun proprietary API and may be
         removed in a future release com.sun.awt.AWTUtilities
         ^
```


Společnost SUN tímto způsobem varuje, že toto speciální rozhraní API může být v budoucnu odstraněno.

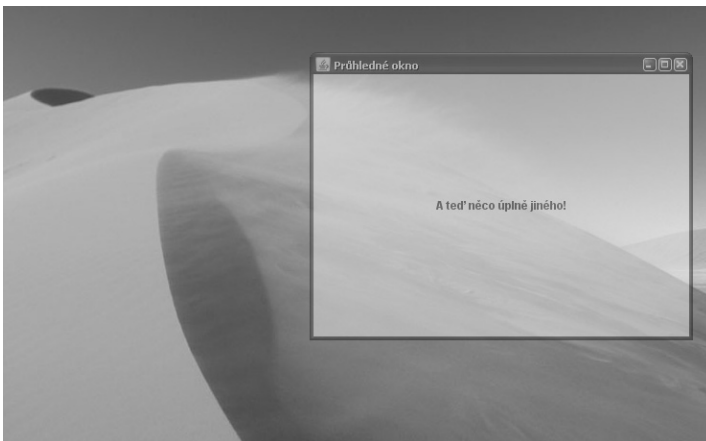
```
import java.awt.*;
import java.awt.geom.Ellipse2D;

import javax.swing.*;

public class TranslucentWindow extends JFrame {
    public TranslucentWindow() {
        super("Průhledné okno");
        this.setLayout(new BorderLayout());
        this.add(BorderLayout.CENTER,
            new JLabel("A teď něco úplně jiného!", JLabel.CENTER));

        this.setSize(new Dimension(400, 300));
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

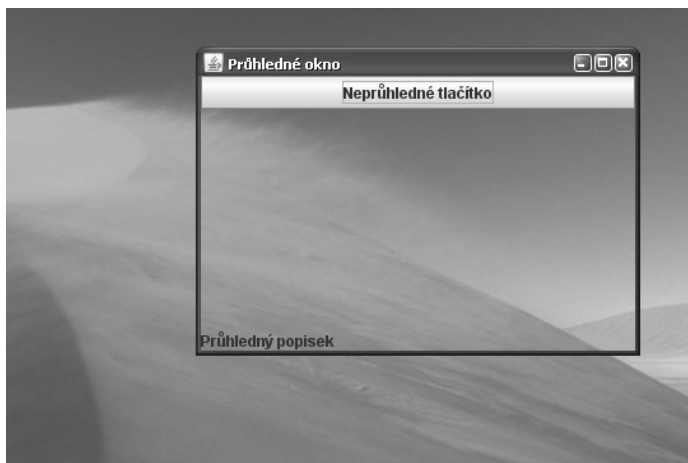
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Window w = new TranslucentWindow();
                w.setVisible(true);
                com.sun.awt.AWTUtilities.setWindowOpacity(w, 0.5f);
            }
        });
    }
}
```



110 Okna s průhledným obsahem



Vývojáři často musí vyřešit požadavek na průhledné okno. Koncepce některých aplikací je dokonce leckdy založena na tom, že část okna je průhledná a část neprůhledná. Tento příklad ukazuje, jak jednoduchým způsobem část okna nastavit jako průhlednou. Příklad neřeší překreslování při změně velikosti nebo při přesunu.



```
import java.awt.*;
import javax.swing.*;

class TransparentBackground extends JComponent {

    private JFrame frame;
    private Image background;

    public TransparentBackground(JFrame frame) {
        this.frame = frame;
        updateBackground();
    }

    public void updateBackground() {
        try {
            Robot rbt = new Robot();
            Toolkit tk = Toolkit.getDefaultToolkit();
            Dimension dim = tk.getScreenSize();
            background = rbt.createScreenCapture(
                new Rectangle(0, 0, (int) dim.getWidth(), (int) dim.getHeight()));
        } catch (Exception ex) {
            System.out.print(ex.toString());
            ex.printStackTrace();
        }
    }
}
```

```

    public void paintComponent(Graphics g) {
        Point pos = this.getLocationOnScreen();
        Point offset = new Point(-pos.x, -pos.y);
        g.drawImage(background, offset.x, offset.y, null);
    }
}

public class TransparentWindow extends JFrame {
    public TransparentWindow() {
        super("Průhledné okno");
        TransparentBackground bg = new TransparentBackground(this);
        bg.setLayout(new BorderLayout());
        JButton button = new JButton("Neprůhledné tlačítko");
        bg.add("North", button);
        JLabel label = new JLabel("Průhledný popisek");
        bg.add("South", label);

        this.getContentPane().add("Center", bg);
        this.pack();
        this.setSize(200, 200);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    public static void main(String args[]) {
        new TransparentWindow().setVisible(true);
    }
}

```

111 Okna různých tvarů



Ukázka využití tříd z balíčku `com.sun.awt.AWTUtilities`.

Chcete-li vytvořit okna jiná než ve tvaru obdélníka v jazyce Java, můžete využít skryté možnosti a speciálního balíčku `com.sun.awt.AWTUtilities`. Použijete-li v aplikaci funkci tohoto balíčku, zobrazí překladač následující varování:

```

warning: com.sun.awt.AWTUtilities is Sun proprietary API and may be
         removed in a future release com.sun.awt.AWTUtilities

```

Znamená to, že toto speciální rozhraní API může být v budoucnu společností SUN odstraněno.

Nesmírně důležité je však užití metody `setDefaultLookAndFeelDecorated` definované v třídě `JFrame`. Pokud dekorace formuláře nepovolíte, úspěch v podobě elipsy se bohužel nedostaví.

```

import java.awt.*;
import java.awt.geom.Ellipse2D;

```

```
import javax.swing.*;

public class ShapedWindow extends JFrame {
    public ShapedWindow() {
        this.setLayout(new BorderLayout());
        this.add(BorderLayout.CENTER,
            new JLabel("Oválné okno", JLabel.CENTER));

        this.setSize(new Dimension(400, 300));
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

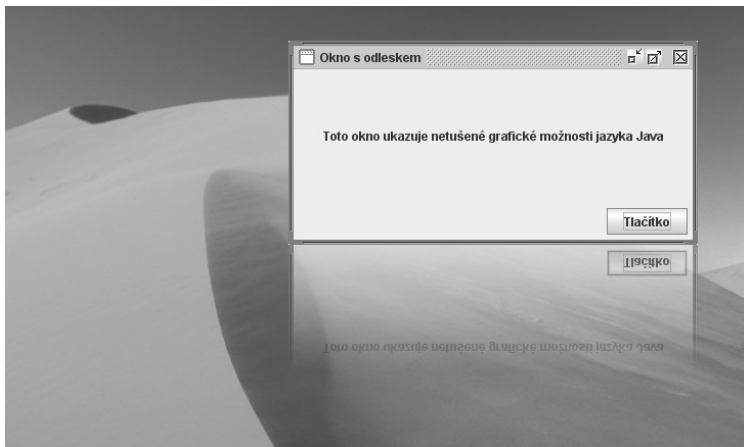
    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Window w = new ShapedWindow();
                w.setVisible(true);
                com.sun.awt.AWTUtilities.setWindowShape(w,
                    new Ellipse2D.Double(5, 40, w.getWidth() - 10,
                        w.getHeight() - 50));
            }
        });
    }
}
```



112 Okno s odleskem



Chcete-li vytvořit působivé funkční uživatelské rozhraní s vodním odleskem, které znáte například z úvodních stránek různých webů nebo ze systému Windows Vista, můžete k dalšímu bádání využít následující třídu, jejíž implementace zaručuje automatickou tvorbu zrcadlového odrazu pod aktivním oknem.



Odraz okna je nejprve sestaven v paměti a teprve pak je vykreslen na displeji. Díky tomu nedochází k blikání ani jiným nežádoucím vedlejším efektům ani při tažení okna myší.

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;

import javax.swing.*;
import javax.swing.border.Border;

/**
 * Okno, které zobrazuje zrcadlový odlesk pod spodním okrajem.
 * Využívá částečně kód publikovaný v knize Filthy Rich Clients
 * autorů Romaina Guye a Cheta Haase.
 */
public class JReflectionFrame extends JFrame {
    private BufferedImage contentBuffer;
    private BufferedImage reflectionBuffer;

    private Graphics2D contentGraphics;
    private Graphics2D reflectionGraphics;

    private GradientPaint alphaMask;

    private float length = 0.65f;
    private float opacity = 0.75f;
```

```
private Window reflection;
private JPanel reflectionPanel;

public JReflectionFrame(String title) {
    super(title);
    reflection = new JWindow();
    reflectionPanel = new JPanel() {
        @Override
        protected void paintComponent(Graphics g) {
            // vykreslí odraz hlavního okna
            paintReflection(g);
        }
    };
    // Nejprve je třeba označit panel tak, aby nevyužíval
    // dvojitou vyrovnávací paměť a aby nebyl neprůhledný.
    reflectionPanel.setDoubleBuffered(false);
    reflectionPanel.setOpaque(false);

    reflection.setLayout(new BorderLayout());
    reflection.add(reflectionPanel, BorderLayout.CENTER);

    this.addComponentListener(new ComponentAdapter() {
        @Override
        public void componentHidden(ComponentEvent e) {
            reflection.setVisible(false);
        }

        @Override
        public void componentMoved(ComponentEvent e) {
            // aktualizace pozice odrazu...
            reflection.setLocation(getX(), getY() + getHeight());
        }

        @Override
        public void componentResized(ComponentEvent e) {
            // aktualizace rozměrů a pozice odrazu...
            reflection.setSize(getWidth(), getHeight());
            reflection.setLocation(getX(), getY() + getHeight());
        }

        @Override
        public void componentShown(ComponentEvent e) {
            reflection.setVisible(true);

            // Je-li odraz neprůhledný, označit průhlednost po pixelech...
            if (com.sun.awt.AWTUtilities.isWindowOpaque(reflection)) {
                com.sun.awt.AWTUtilities.setWindowOpaque(reflection, false);
            }
        }
    });
}
```

```

    }
}
});

this.addWindowListener(new WindowAdapter() {
    @Override
    public void windowActivated(WindowEvent e) {
        // vynucení zobrazení odrazu.
        reflection.setAlwaysOnTop(true);
        reflection.setAlwaysOnTop(false);
    }
});

// inicializace rozměrů a pozice odrazu.
reflection.setSize(getSize());
reflection.setLocation(getX(), getY() + getHeight());
reflection.setVisible(true);

// Instalace vlastního správce překreslování, aby byl odraz
// aktualizován, kdykoli se změní obsah hlavního okna.
RepaintManager.setCurrentManager(new ReflectionRepaintManager());
}

@Override
protected JRootPane createRootPane() {
    return new JRootPane() {
        @Override
        public void paint(Graphics g) {
            paintContent(g);
        }

        private void paintContent(Graphics g) {
            if (contentBuffer == null
                || contentBuffer.getWidth() != getWidth()
                || contentBuffer.getHeight() != getHeight()) {
                if (contentBuffer != null) {
                    contentBuffer.flush();
                    contentGraphics.dispose();
                }

                GraphicsConfiguration gc = ((Graphics2D) g)
                    .getDeviceConfiguration();
                contentBuffer = gc.createCompatibleImage(getWidth(),
                    getHeight(), Transparency.TRANSLUCENT);
                contentGraphics = contentBuffer.createGraphics();
            }

            // vykreslení obsahu na obrázku v paměti...

```

```
Graphics2D g2 = contentGraphics;
g2.clipRect(getX(), getY(), getWidth(), getHeight());

g2.setComposite(AlphaComposite.Clear);
Rectangle clip = g.getClipBounds();
g2.fillRect(clip.x, clip.y, clip.width, clip.height);
g2.setComposite(AlphaComposite.SrcOver);

g2.setColor(g.getColor());
g2.setFont(g.getFont());
super.paint(g2);

// překreslení obrázku z paměti na obrazovku...
g.drawImage(contentBuffer, 0, 0, null);
}
};
}

// Vykreslení odrazu hlavního okna.
public void paintReflection(Graphics g) {
    JRootPane rootPane = this.getRootPane();
    Border rootPaneBorder = rootPane.getBorder();
    int width = rootPane.getWidth();
    int height = (int) (rootPane.getHeight() * length);
    createReflection(g, width, height);

    Graphics2D g2 = (Graphics2D) g.create();
    g2.scale(1.0, -1.0);
    g2.drawImage(reflectionBuffer, 0, -height
        + rootPaneBorder.getBorderInsets(rootPane).bottom, null);
    g2.dispose();
}

// Tvorba odrazu hlavního okna.
private void createReflection(Graphics g, int width, int height) {
    JRootPane rootPane = this.getRootPane();
    if (reflectionBuffer == null || reflectionBuffer.getWidth() != width
        || reflectionBuffer.getHeight() != height) {
        if (reflectionBuffer != null) {
            reflectionBuffer.flush();
            reflectionGraphics.dispose();
        }

        GraphicsConfiguration gc = ((Graphics2D) g)
            .getDeviceConfiguration();
        reflectionBuffer = gc.createCompatibleImage(getWidth(),
            getHeight(), Transparency.TRANSLUCENT);
    }
}
```



```

        reflectionGraphics = reflectionBuffer.createGraphics();

        alphaMask = new GradientPaint(0.0f, 0.0f, new Color(0.0f, 0.0f,
            0.0f, 0.0f), 0.0f, height, new Color(0.0f, 0.0f, 0.0f,
            opacity), true);
    }

    int yOffset = rootPane.getHeight() - height;
    Rectangle clip = g.getClipBounds();

    Graphics2D g2 = (Graphics2D) reflectionGraphics.create();
    g2.setClip(clip.x, clip.y - yOffset, clip.width, clip.height);

    g2.setComposite(AlphaComposite.Clear);
    g2.fillRect(clip.x, clip.y - yOffset, clip.width, clip.height);
    g2.fillRect(0, 0, getWidth(), getHeight());
    g2.setComposite(AlphaComposite.SrcOver);

    g2.translate(0, -yOffset);
    g2.drawImage(contentBuffer, 0, 0, null);
    g2.translate(0, yOffset);

    g2.setComposite(AlphaComposite.DstIn);
    g2.setPaint(alphaMask);
    g2.fillRect(clip.x, clip.y - yOffset, clip.width, clip.height);

    g2.dispose();
}

private class ReflectionRepaintManager extends RepaintManager {
    @Override
    public void addDirtyRegion(JComponent c, int x, int y, int w, int h) {
        Window win = SwingUtilities.getWindowAncestor(c);
        if (win instanceof JReflectionFrame) {
            // Označíme celý hlavní panel k překreslení.
            JRootPane rp = ((JReflectionFrame) win).getRootPane();
            super.addDirtyRegion(rp, 0, 0, rp.getWidth(), rp.getHeight());

            // volání reflection.repaint() nepřekreslí panel
            reflectionPanel.repaint();
        } else {
            super.addDirtyRegion(c, x, y, w, h);
        }
    }
}
}

```

Instanci třídy `JReflectionFrame` vytvoříte například takto:

```

import java.awt.*;
import javax.swing.*;

public class SimpleFrame extends JFrame {
    public SimpleFrame() {
        super("Okno s odleskem");
        this.setLayout(new BorderLayout());
        this.add(new JLabel(
            "Toto okno ukazuje netušené grafické možnosti jazyka Java",
            JLabel.CENTER));
        JPanel bottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        bottom.add(new JButton("Tlačítko"));
        this.add(bottom, BorderLayout.SOUTH);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setSize(400, 200);
        this.setLocationRelativeTo(null);
    }

    public static void main(String[] args) {
        JFrame.setDefaultLookAndFeelDecorated(true);
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                Window w = new SimpleFrame();
                w.setVisible(true);
            }
        });
    }
}

```

V předchozím příkladu (a v několika dalších příkladech také) jsme použili velmi důležitý příkaz:

```

JFrame.setDefaultLookAndFeelDecorated(true);

```

Metodu `setDefaultLookAndFeelDecorated()` je třeba volat vždy *před* tvorbou oken, jejichž dekorace chcete následnými příkazy ovlivnit. Hodnota, kterou této metodě předáte, bude v programu použita pro všechny následně vytvořené objekty typu `JFrame`. Chcete-li změnit užití systémových dekorací oken, zavolejte na příslušném místě opět tuto metodu a předejte jí hodnotu `false`. Některé motivy neumožňují využívat dekorace implementované do knihoven jazyka Java. V takových případech program použije systémové dekorace oken.

113 Minimalizace a maximalizace hlavního okna aplikace



Ukázka využití tříd z balíčku `java.awt`.

V tomto příkladu implementujeme metody pro minimalizaci, obnovení a maximalizaci hlavního okna aplikace. Za normálních okolností byste metodu `Frame.setExtendedState` s holým argumentem `Frame.ICONIFIED` neměli volat vůbec, protože tím můžete nežá-

doucím způsobem změnit příznak maximalizace. Mnohem lepší je konstantu stavu `Frame.ICONIFIED` kombinovat s aktuálním příznakem maximalizace.

```
// Tato metoda minimalizuje okno a neovlivní příznak maximalizace.
```

```
public void iconify(Frame frame) {  
    int state = frame.getExtendedState();
```

```
    // Nastaví příznak minimalizace.  
    state |= Frame.ICONIFIED;
```

```
    // Minimalizuje okno.  
    frame.setExtendedState(state);  
}
```

```
// Tato metoda obnoví okno a nezmění příznak maximalizace.
```

```
public void deiconify(Frame frame) {  
    int state = frame.getExtendedState();
```

```
    // Vyčistí příznak minimalizace.  
    state &= ~Frame.ICONIFIED;
```

```
    // Obnoví okno.  
    frame.setExtendedState(state);  
}
```

```
// Tato metoda minimalizuje okno a nezmění příznak minimalizace.
```

```
public void minimize(Frame frame) {  
    int state = frame.getExtendedState();
```

```
    // Vyčistí příznak maximalizace.  
    state &= ~Frame.MAXIMIZED_BOTH;
```

```
    // Maximalizuje okno.  
    frame.setExtendedState(state);  
}
```

```
// Tato metoda minimalizuje okno a nezmění příznak minimalizace.
```

```
public void maximize(Frame frame) {  
    int state = frame.getExtendedState();
```

```
    // Nastaví příznak maximalizace.  
    state |= Frame.MAXIMIZED_BOTH;
```

```
    // Maximalizuje okno.  
    frame.setExtendedState(state);  
}
```

114 Jak určit, zda je okno minimalizováno nebo maximalizováno



pokročilý

Ukázka využití tříd z balíčku java.awt.

Potřebujete-li zjistit, zda je okno minimalizováno nebo maximalizováno, registrujte posluchače událostí okna a překryjte metodu `windowStateChanged()`.

```
Frame frame = new Frame();

// Registrace posluchače události okna.
frame.addWindowStateListener(new WindowAdapter() {
    public void windowStateChanged(WindowEvent evt) {
        int oldState = evt.getOldState();
        int newState = evt.getNewState();

        if ((oldState & Frame.ICONIFIED) == 0
            && (newState & Frame.ICONIFIED) != 0) {
            // Okno bylo minimalizováno.
        } else if ((oldState & Frame.ICONIFIED) != 0
            && (newState & Frame.ICONIFIED) == 0) {
            // Okno bylo obnoveno.
        }

        if ((oldState & Frame.MAXIMIZED_BOTH) == 0
            && (newState & Frame.MAXIMIZED_BOTH) != 0) {
            // Okno bylo maximalizováno.
        } else if ((oldState & Frame.MAXIMIZED_BOTH) != 0
            && (newState & Frame.MAXIMIZED_BOTH) == 0) {
            // Okno bylo minimalizováno.
        }
    }
});
```

115 Jak určit, zda je okno otevřeno nebo zavřeno



pokročilý

Ukázka využití tříd z balíčku java.awt.

Potřebujete-li zjistit, zda je okno otevřeno nebo zavřeno, registrujte posluchače událostí okna a překryjte metody `windowOpened()`, `windowClosing()` a `windowClosed()`.

```
Frame frame = new Frame();

// Registrace posluchače pro dané okno.
frame.addWindowListener(new WindowAdapter() {
    // Tato metoda se volá při otevření okna.
    public void windowOpened(WindowEvent evt) {
        Frame frame = (Frame)evt.getSource();
    }
});
```

```
// Tato metoda se volá po klepnutí na systémové tlačítko Zavřít.
public void windowClosing(WindowEvent evt) {
    Frame frame = (Frame)evt.getSource();
}

// Tato metoda se volá po zavření okna
public void windowClosed(WindowEvent evt) {
    Frame frame = (Frame)evt.getSource();
}
});
```

116 Jak používat předdefinované dialogy

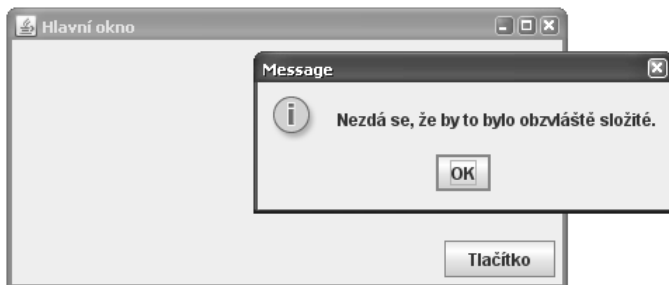


Dialog je okno, které obsahuje poznámku nebo dočasnou informaci a jež je nezávislé na hlavním okně (formuláři) typu `Frame` nebo `JFrame`. Většina dialogů slouží k zobrazení chybové zprávy nebo varování. Pomocí dialogů ovšem lze zobrazit také obrázky, stromové struktury (například adresářovou strukturu) – tedy v podstatě cokoli, co lze zobrazit také v hlavním okně.

Knihovna Swing obsahuje několik komponent, jež výrazně usnadňují tvorbu a zobrazení informačních dialogů. Pro jednoduché standardní dialogy používejte třídu `JOptionPane`. Třída `ProgressMonitor` umožňuje vytvořit dialog s ukazatelem průběhu časově náročnějších operací. K zobrazení standardních dialogů, jež znáte také z mnoha jiných aplikací, slouží třídy `JColorChooser` nebo `JFileChooser`. K zobrazení dialogu **Tisk** použijte funkci balíčku `Printing`. K zobrazení vlastního dialogu použijte třídu `JDialog`.

Následující kód postačí k zobrazení jednoduché informační zprávy:

```
JFrame frame = new JFrame("Hlavní okno");
...
JOptionPane.showMessageDialog(frame,
    "Nezdá se, že by to bylo obzvláště složité.");
```



117 Jednoduché předdefinované dialogy



Většinu modálních dialogů zobrazíte pomocí metod `showXxxDialog()` definovaných v třídě `JOptionPane`. Je-li dialog volaný z dokumentu aplikace MDI, použijte některou z metod `showInternalXxxDialog()`.

Nejčastěji budete asi potřebovat metody `showMessageDialog()` a `showOptionDialog()`. První zobrazí jednoduchou zprávu s jedním tlačítkem určeným k zavření dialogu. Druhý vám umožní definovat nejen znění zprávy, ale také počet a obsah tlačítek. Hodit se vám ale může také metoda `showConfirmDialog()`, která umožní autorovi programu přimět uživatele k vědomému potvrzení důležité akce. Dialog zobrazený pomocí této metody obsahuje standardní tlačítka **Yes** a **No**, případně jejich lokalizované ekvivalenty (pokud explicitně použijete motiv Windows, nikoli výchozí motiv Javy). Poslední metodou k zobrazení dialogu je `showInputDialog()`, do níž uživatel může interaktivně zadat požadovaný údaj prostřednictvím textového pole, či rozevíracího nebo běžného seznamu.

```
// dialog bez ikony (standardní jednoduchý dialog)
JOptionPane.showMessageDialog(frame,
    "Používání dialogů v jazyce Java je nadmíru snadné.",
    "Zpráva bez ikony",
    JOptionPane.PLAIN_MESSAGE);

// informační dialog
JOptionPane.showMessageDialog(frame,
    "Používání dialogů v jazyce Java je nadmíru snadné.",
    "Informační dialog",
    JOptionPane.INFORMATION_MESSAGE);

// dotaz na uživatele
JOptionPane.showMessageDialog(frame,
    "Myslíte si, že tento dialog je vhodný k dotazu?",
    "Dotazovací dialog",
    JOptionPane.QUESTION_MESSAGE);

// oznámení chyby
JOptionPane.showMessageDialog(frame,
    "Používání dialogů v jazyce Java je nadmíru snadné.",
    "Oznámení chyby",
    JOptionPane.ERROR_MESSAGE);

// varování
JOptionPane.showMessageDialog(frame,
    "Používání dialogů v jazyce Java je nadmíru snadné.",
    "Varování",
    JOptionPane.WARNING_MESSAGE);
```

118 Informační dialog s vlastní ikonou



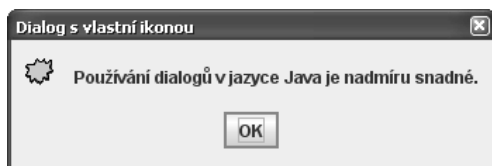
znalec

Pomocí metody `showMessageDialog()` můžete zobrazit ale také dialog s vlastní ikonou.

```
java.net.URL cesta = DialogDemo.class.getResource(path);
ImageIcon ikona = new ImageIcon(cesta);
```

```
JOptionPane.showMessageDialog(frame,
    "Používání dialogů v jazyce Java je nadmíru snadné.",
```

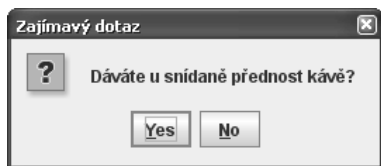
```
"Dialog s vlastní ikonou",
OptionPane.INFORMATION_MESSAGE,
ikona);
```



119 Vlastní text na tlačítkách předdefinovaných dialogů



Pomocí metody `showOptionDialog()` můžete vytvořit dialog, jehož tlačítka, ikony a text budou plně ve vaší režii. Pokud se rozhodnete použít dialog s předdefinovanou ikonou a s předdefinovanými tlačítky, bude vypadat takto:



Zpracování uživatelských rozhodnutí získaných pomocí předdefinovaných dialogů, v nichž jste použili tlačítka s vlastními popisky, se od zpracování jiných předdefinovaných dialogů neliší. Zapamatujte si, že jste změnili pouze popisky tlačítek. Jejich definice se nezměnila.

```
int n = JOptionPane.showConfirmDialog(frame,
    "Dáváte u snídani přednost kávě?",
    "Zajímavý dotaz",
    JOptionPane.YES_NO_OPTION);
if (n == JOptionPane.YES_OPTION) {
    // Káva je to pravé ořechové.
} else if (n == JOptionPane.NO_OPTION) {
    // Takže raději něco jiného?
} else {
    // Takže jsme se odpovědi nedočkali...
}
```

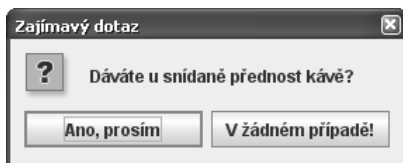
Mnohem lepší je nabídnout uživateli mnohem intuitivnější odpověď. Všimněte si, že kromě změny textu tlačítek je způsob volání dialogu stejný.

```
Object[] options = {"Ano, prosím", "V žádném případě!"};
int n = JOptionPane.showOptionDialog(frame,
    "Dáváte u snídani přednost kávě?",
    "Zajímavý dotaz",
    JOptionPane.YES_NO_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[0]);
```

```

if (n == JOptionPane.YES_OPTION) {
    // Káva je i v tomto případě to pravé ořechové.
} else if (n == JOptionPane.NO_OPTION) {
    // Takže i nyní raději něco jiného?
} else {
    // Takže jsme se odpovědi opět nedočkali...
}

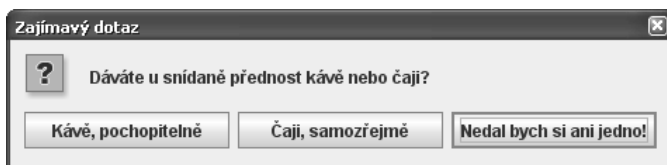
```



```

Object[] options = {"Kávě, pochopitelně",
    "Čaji, samozřejmě",
    "Nedal bych si ani jedno!"};
int n = JOptionPane.showOptionDialog(frame,
    "Dáváte u snídaně přednost kávě, nebo čaji?",
    "Zajímavý dotaz",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[2]);
if (n == JOptionPane.YES_OPTION) {
    // Káva je i v tomto případě to pravé ořechové.
} else if (n == JOptionPane.NO_OPTION) {
    // Takže nyní víme, že jste čajoví.
} else if (n == JOptionPane.CANCEL_OPTION) {
    // Takže ani kávu, ani čaj?
} else {
    // Takže jsme se odpovědi opět nedočkali...
}

```



120 Dialog, kdy uživatel musí vybrat některou z možností



V předchozím příkladu jste určitě zaznamenali, že uživatel měl možnost nabízené možnosti ignorovat a zavřít dialog pomocí tlačítka **Zavřít** umístěného v pravém horním rohu dialogu.

Existují však situace, v nichž by ošetření takové únikové varianty bylo příliš složité. Použijte proto možnost dialogu, v němž si uživatel některou z nabízených možností vybrat musí, ať už chce nebo ne. Jinak totiž takový dialog zavřít nelze:

```
final JOptionPane optionPane = new JOptionPane(
    "Tento dialog lze zavřít pouze pomocí jednoho\n"
    + "ze dvou nabízených tlačítek.\n\n"
    + "Co vy na to?",
    JOptionPane.QUESTION_MESSAGE,
    JOptionPane.YES_NO_OPTION);

final JDialog dialog = new JDialog(frame, "Klepněte na tlačítko", true);
dialog.setContentPane(optionPane);
dialog.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
dialog.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent we) {
        setLabel("A uživatel si nedá říct a nedá.");
    }
});
optionPane.addPropertyChangeListener(
    new PropertyChangeListener() {
        public void propertyChange(PropertyChangeEvent e) {
            String prop = e.getPropertyName();

            if (dialog.isVisible() && (e.getSource() == optionPane)
                && (JOptionPane.VALUE_PROPERTY.equals(prop))) {
                // Chcete-li něco ověřit před zavřením dialogu,
                // udělejte to zde.
                dialog.setVisible(false);
            }
        }
    });
dialog.pack();
dialog.setLocationRelativeTo(frame);
dialog.setVisible(true);

int value = ((Integer)optionPane.getValue()).intValue();
if (value == JOptionPane.YES_OPTION) {
    // V pořádku.
} else if (value == JOptionPane.NO_OPTION) {
    // Pokuste se zavřít dialog pomocí ovládacích prvků v proužku titulku.
} else {
    // Dialog byl zavřen klávesou Esc.
}
```

Pokud nechcete ověřovat vstup uživatele, můžete k zobrazení použít metodu `showInputDialog()`.

121 Nemodální dialog



Všechny dialogy jsou závislé na nadřazeném objektu typu `Frame` nebo `JFrame`. Zanikne-li nadřazený objekt, zaniknou také všechny v něm vytvořené dialogy. Je-li nadřazené okno minimalizováno, zmizí z obrazovky také s ním související dialogy, jež se na obrazovce objeví zpět, jakmile nadřazené okno obnovíte nebo maximalizujete. Tuto charakteristiku dědí objekty typu `JDialog` od třídy `Dialog` definované v knihovně AWT.

Dialog může být modální nebo nemodální. Modální dialogy pozastaví běh programu, dokud uživatel nevykoná požadovanou akci. Pomocí třídy `JOptionPane` lze vytvořit pouze modální dialogy. Chcete-li vytvořit dialog nemodální, musíte použít třídu `JDialog` přímo.

V situacích, v nichž lze použít pouze nemodální dialog, vám může pomoci následující kód:

```
final JDialog dialog = new JDialog(frame, "Nemodální Dialog");

JLabel label = new JLabel("Toto je nemodální dialog.\n"
    + "Těchto dialogů můžete zobrazit klidně více\n"
    + "a přesto neztratíte možnost pracovat s nadřazeným hlavním oknem.");
label.setHorizontalAlignment(JLabel.CENTER);

JButton closeButton = new JButton("Zavřít");
closeButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        dialog.setVisible(false);
        dialog.dispose();
    }
});

JPanel closePanel = new JPanel();
closePanel.setLayout(new BoxLayout(closePanel, BoxLayout.LINE_AXIS));
closePanel.add(Box.createHorizontalGlue());
closePanel.add(closeButton);
closePanel.setBorder(BorderFactory.createEmptyBorder(0,0,5,5));
JPanel contentPane = new JPanel(new BorderLayout());
contentPane.add(label, BorderLayout.CENTER);
contentPane.add(closePanel, BorderLayout.PAGE_END);
contentPane.setOpaque(true);
dialog.setContentPane(contentPane);

// Zobrazení nemodálního dialogu.
dialog.setSize(new Dimension(300, 150));
dialog.setLocationRelativeTo(frame);
dialog.setVisible(true);
```

122 Modalita dialogů



V J2SE 6 se konečně podařilo vyřešit problémy kolem modality dialogů, jež sužovaly dřívější verze Javy. Nový model modality umožňuje vývojáři určit rozsah aplikace, pro nějž je dialog modální. V J2SE 6 lze využívat následující typy modality dialogů: