

Začínáme programovat v jazyku **JAVA**

- » Nepředpokládá žádné předchozí znalosti programování
- » Neomezuje se na výuku kódování, ale učí, čtenáře programovat podle moderních zásad a metodik
- » Probírá všechny důležité konstrukce jazyka včetně těch, které začátečnické učebnice přeskakují
- » Všechny probírané konstrukce vysvětluje na příkladech
- » Pro výklad složitějších programových konstrukcí používá syntaktické diagramy



edice
začínáme s ...

Začínáme programovat v jazyku JAVA

**RUDOLF PECINOVSKÝ
JARMILA PAVLÍČKOVÁ**

GRADA Publishing



Upozornění pro čtenáře a uživatele této knihy

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

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

Rudolf Pecinovský, Jarmila Pavlíčková

Začínáme programovat v jazyku Java

Vydala Grada Publishing, a.s.

U Průhonu 22, Praha 7

obchod@grada.cz, www.grada.cz

tel.: +420 234 264 401

jako svou 8318. publikaci

Odpovědný redaktor Petr Somogyi

Fotografie na obálce Depositphotos/gdolgikh

Grafická úprava a sazba Rudolf Pecinovský

Počet stran 400

První vydání, Praha 2021

Vytiskla TISKÁRNA V RÁJI, s.r.o, Pardubice

© Grada Publishing, a.s., 2021

Cover Design © Grada Publishing, a. s., 2021

Cover Photo © Depositphotos/gdolgikh

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

ISBN 978-80-271-4641-3 (ePub)

ISBN 978-80-271-4640-6 (pdf)

ISBN 978-80-271-3062-7 (print)

Všem, kteří se chtějí něco naučit

Stručný obsah

Úvod	19
Část A Superzáklady	27
1 Předehra.....	28
2 Prostředí JShell	40
3 Zadávání hodnot.....	51
4 Proměnné a výrazy.....	65
Část B Začínáme programovat	79
5 Práce s objekty	80
6 Robot Karel a jeho svět, knihovny, balíčky	90
7 Definice metod.....	103
8 Opakování – cykly.....	120
9 Rozhodování.....	130
Část C Objektově orientované programování	143
10 Základy definice třídy.....	144
11 Stručný úvod do BlueJ	159
12 Uspořádání kódu	169
13 Programátorská dokumentace.....	180
14 Rozhraní a interfejs	189
15 Dědění interfejsů	200
16 Návrhové vzory.....	214
17 Prohlubujeme znalosti	229
18 Lambda-výrazy, funkční interfejsy a generické typy.....	243
19 Dědění implementace.....	261
20 Abstraktní třídy	276

Část D Knihovny **293**

21 Speciální datové typy.....	294
22 Výjimky.....	312
23 Kontejnery	325
24 Pole.....	339
25 Interní datové typy	352
26 Datovody.....	361
27 Čtení a ukládání dat	372
28 Vytvoření aplikace.....	388

Literatura 394

Rejstřík 396

Část E Přílohy **401**

A Příprava spuštění JShell pod Windows.....	402
B Význam cizích slov	407
C Česko-americký slovník.....	409

Část F SEZNAMY **411**

Seznam výpisů programů.....	412
Seznam obrázků	416
Seznam tabulek	418
Seznam odboček – podšeděných bloků	419

Podrobný obsah

Úvod	19
Komu je kniha určena	20
Koncepce výkladu	20
Potřebné vybavení	21
Doprovodné programy	22
Použité typografické konvence	22
Odbočka – podšeděný blok	24
Zpětná vazba	24
Část A Superzáklady	27
1 Předehra	28
1.1 Trocha historie	28
Co je to program	29
Postupná změna priorit	29
Vývoj metodik programování	31
OOP a OFP	32
1.2 Překladače, interprety, platformy	32
Operační systém a platforma	33
Programovací jazyky	33
Způsoby zpracování programu	33
Překládaný program	34
Interpretovaný program	34
Porovnání	34
Hybridní programy	34
1.3 Java a její zvláštnosti	35
Java je jazyk i platforma	35
1.4 Vývojové nástroje	36
Základní vývojářská sada	36
IDE	37
JShell	37
BlueJ	37
NetBeans	38
IntelliJ IDEA	38
Eclipse	38
Visual Studio Code	39
Shrnutí	39
2 Prostředí JShell	40
2.1 Prostředí JShell	40
Spuštění programu JShell	40
2.2 Úryvky (snippets)	42
Použití proměnných	43
Identifikace úryvků	43

Terminologie: výrazy, příkazy, deklarace, definice	43
Středník.....	44
Více objektů na řádku, zavlečené chyby	44
2.3 Příkazy prostředí JShell	45
Vyloučení úryvku: /drop	46
Přehled aktivních úryvků: /list.....	46
Přehled všech úryvků: /list -all	47
Uložení aktivních úryvků: /save <file>	48
Uložení všech zadanych úryvků: /save -all <file>	48
Uložení dosavadního průběhu seance: /save -history <file>.....	48
Načtení skriptu: /open <file>	48
Ukončení seance: /exit.....	49
Restart: /reset.....	49
Znovuzavedení: /reload -restore.....	49
Natavení startovního skriptu: /set -start <file>.....	49
Nápověda: /?	50
Spuštění editoru: /edit.....	50
Nastavení běhového prostředí: /env.....	50
Další informace	50
3 Zadávání hodnot.....	51
3.1 Datové typy	51
Primitivní datové typy.....	52
Objektové datové typy.....	53
Odkazy na objekty	54
3.2 Komentáře.....	54
3.3 Zadávání celočíselných hodnot	55
3.4 Zadávání reálných hodnot	56
Číslo v exponentovém tvaru	56
3.5 Zadávání znaků.....	57
3.6 Prázdný odkaz null	59
3.7 Hodnoty typu String.....	59
Textové bloky	62
3.8 Zadávání logických hodnot	63
3.9 Literály	64
4 Proměnné a výrazy.....	65
4.1 Pravidla pro tvorbu identifikátorů	65
Používání znaku \$	66
4.2 Deklarace proměnných	66
4.3 Terminologie	68
4.4 Operace přiřazení =	69
Asociativita.....	69
4.5 Aritmetické operace	70
Sčítání	70
Odčítání.....	70
Násobení	70
Dělení	70
4.6 Přetypování	71
4.7 Volání metod	72
4.8 Závorky	73
4.9 Složené přiřazení	73
4.10 Inkrementační a dekrementační operátory	74
4.11 Porovnávací operátory	76
4.12 Příkazy versus výrazy	77

Část B Začínáme programovat 79

5 Práce s objekty	80
5.1 Nejprve trocha teorie	80
Principy OOP	80
Objekty	81
Atributy	81
Zprávy	82
Metody	82
Třídy a jejich instance	83
Interní datové typy	84
Třída jako datový typ	84
Třída jako objekt	84
Kontejner	85
Terminologie objektových datových typů	85
5.2 Práce s atributy	86
5.3 Volání metody	86
5.4 Konstruktory a tovární metody	88
Tovární metoda	88
Typické vytvoření instance	89
Pokračování	89
6 Robot Karel a jeho svět, knihovny, balíčky	90
6.1 Robot Karel a jeho svět	90
Vytváření	90
Historie robota Karla	91
Akce	91
Testy	92
Zrychlování	93
Reakce na zavření okna	94
6.2 Knihovny	96
6.3 Velké programy a jejich problémy	97
6.4 Balíčky	98
Kořenový balíček a strom balíčků	98
6.5 Import objektových typů	98
Příklad	99
Import všech typů z daného balíčku – hvězdičkový import	101
6.6 Zakázaný balíček java	102
7 Definice metod	103
7.1 Zásada DRY	103
7.2 Definice a volání metody	104
Povinný středník	104
Příklad jednoduché metody pro robota	104
7.3 Blok příkazů	106
7.4 Metody s parametry	106
Parametry	106
Argumenty	107
Metody robota versus metody prostředí JShell	108
7.5 Metody s více parametry	108
Tisk na standardní výstup	109
7.6 Lokální proměnné	110
Proměnné lokální v bloku	110
7.7 Přetěžování metod	110
7.8 Ještě jednou robot Karel a jeho svět	111
Třídy RobotWorld a RobotWindow	112
Přetížené tovární metody tříd RobotWorld a RobotWindow	112

RobotWorld newWorld()	112
RobotWorld newWorld(int rows, int cols)	112
RobotWorld newWorld(String... world)	112
Společné vlastnosti	112
static void destroyWorld()	113
void destroy()	113
static RobotWorld getWorld()	113
Přetížené konstruktory třídy Karel	113
Karel(int row, int col, Direction dir)	113
Karel(int row, int col, Direction dir, Color color)	113
Karel()	113
7.9 Metody vracějící hodnotu	114
7.10 Přehled aktuálně definovaných metod v JShell	116
7.11 Logické výrazy	116
Příklad	117
8 Opakování – cykly	120
8.1 Cykly	120
8.2 Cyklus s počáteční podmínkou – cyklus while	121
8.3 Cyklus s koncovou podmínkou – cyklus do...while	123
8.4 Cyklus s parametrem – cyklus for	124
Příklad	126
putNMarkers(Karel, int)	126
putIncreasingMarkers(Karel)	127
markers(Karel)	127
Test	128
8.5 Nekonečný cyklus	128
8.6 Cyklus s podmínkou uprostřed	129
9 Rozhodování	130
9.1 Jednoduchý podmíněný příkaz	130
9.2 Úplný podmíněný příkaz	132
9.3 Podmíněný výraz	133
9.4 Složený podmíněný příkaz	133
9.5 Přepínač – příkaz/výraz switch	136
Pravidla	136
Přepínací výraz – výraz switch	139
9.6 Cyklus s podmínkou uprostřed – příkaz break	140

Část C Objektově orientované programování **143**

10 Základy definice třídy	144
10.1 Vytváříme vlastní třídu	144
Bílé znaky a uspořádání programu	145
10.2 Viditelnost tříd a jejich členů: public, private	145
10.3 Výměna knihoven – knihovna tvarů	146
IO	146
Canvas	146
NamedColor	147
Direction	147
Ellipse, Rectangle, Triangle	147
10.4 Konstruktory	147
Podrobnosti o konstruktorech	149
10.5 Definice atributů	149

Účel atributů.....	150
10.6 Modifikátor <code>final</code>	150
10.7 Konstruktory a parametr <code>this</code>	151
Kvalifikace parametrem <code>this</code>	152
10.8 Magické hodnoty	152
10.9 Modifikátor <code>static</code>	153
Definice třídy počítající své instance	154
10.10 Zděděné metody, metoda <code>toString()</code>	155
10.11 Výsledná definice a test.....	155
Upravená definice.....	155
Test upravené definice.....	157
11 Stručný úvod do BlueJ	159
11.1 Instalace BlueJ.....	159
11.2 Projekty a BlueJ.....	160
Vyhledání a otevření projektu	160
11.3 Aplikační okno BlueJ	161
Okna balíčků	161
11.4 Diagram tříd.....	163
11.5 Překlad a jeho pravidla.....	163
11.6 Manipulace s třídami v diagramu	164
11.7 Otevření okna jiného balíčku.....	167
11.8 Práce se zdrojovými soubory.....	167
12 Uspořádání kódu	169
12.1 Uspořádání projektů v Javě	169
Uspořádání projektu 67_Java.....	169
Nové umístění zdrojů	170
12.2 Samostatně definované třídy.....	170
12.3 Příkazy <code>package</code> a <code>import</code>	170
12.4 Zapouzdření a skrývání implementace	172
12.5 Entity programu	173
12.6 Rozhraní versus implementace.....	173
Signatura versus kontrakt	174
12.7 Atributy versus vlastnosti; přístupové metody	174
Možné kombinace a výhody.....	175
Pravidla pro názvy.....	175
12.8 Uspořádání jednotlivých prvků v těle třídy	176
Motivace pro zavedení některých zásad	176
Kódování	176
Co dříve: atributy nebo metody?	177
Statické a instanční členy	177
Veřejné, neveřejné, soukromé.....	177
Uvození oddílů.....	177
12.9 Prázdná standardní třída.....	178
13 Programátorská dokumentace.....	180
13.1 Komentáře a dokumentace.....	180
Proč psát srozumitelné programy	180
Odsazování	182
Dokumentační komentáře	182
13.2 Pomocné značky pro tvorbu dokumentace	183
13.3 Dokumentace balíčku.....	184
13.4 Ukázka dokumentované třídy	184
13.5 Zobrazení dokumentace.....	186
Obsah a uspořádání dokumentace.....	186

Dokumentace entit vyšší úrovně	186
Dokumentace datových typů	187
Vyhledávání	187
Zavržené (nedoporučované) entity	187
13.6 Zakomentování a odkomentování části programu	188
14 Rozhraní a interfejs	189
14.1 Motivace	189
14.2 Rozhraní versus interfejs versus interface	190
Interfejs a jeho instance	190
14.3 Použití v programu	191
Knihovni balíček eu.pedu.b67.canvas_2	192
14.4 Použití interfejsu na příkladu	193
Počáteční úvahy	193
14.5 Definice interfejsu	196
14.6 Implementace interfejsu třídou	196
Anotace @Override	196
Ověření funkcionality	198
15 Dědění interfejsů	200
15.1 Problém více rolí	200
15.2 Současná implementace více interfejsů	201
Realizace v kódu	202
15.3 Dědění interfejsů	203
Trocha teorie o dědění	203
Dědění a přetypování	204
Aplikace dědění interfejsů na balíček canvas_3 – balíček canvas_4	205
ICanvasPaintable	206
IResizable, IMovable	206
IChangeable	206
IModular	206
IShape	207
Multishape	207
15.4 Deklarace rodičů interfejsu v kódu	207
15.5 Vlastnosti interfejsů na příkladu Multishape	207
Podrobnosti o třídě Multishape	208
Mnohotvar se skládá z kopií	208
15.6 Příklad: definice vozidla	209
Kontrakt	209
Test vytvořeného vozidla	212
Problém plátna	213
16 Návrhové vzory	214
16.1 Úvod do návrhových vzorů	214
16.2 Přehled vzorů, s nimiž jsme se již setkali	216
Knihovni třída (Utility class)	216
Statická tovární metoda (Static factory method)	216
Jedináček (Singleton)	217
Výčtový typ (Enumerated type)	217
Multiton, originál	217
Služebník (Servant)	218
Prázdný objekt (Null Object)	218
Prototyp (Prototype)	219
16.3 Teorie k nové koncepci kreslení	220
Návrhový vzor Prostředník (Mediator)	220
Inverze závislostí	221
Návrhový vzor Pozorovatel (Observer), hollywoodský princip	222

16.4 Správce plátna – CanvasManager	223
Baliček canvasmanager.....	224
Import klíčových tříd knihovny	225
16.5 Nová verze třídy Robot	226
17 Prohlubujeme znalosti	229
17.1 Statický import.....	229
17.2 Další členy interfejsů.....	230
Statické konstanty.....	230
Statické metody	232
Implicitní metody	232
Soukromé metody	233
17.3 Návrhový vzor Adaptér (Adapter)	233
17.4 Návrhový vzor Přepravka.....	234
Třídy typu záznam (record).....	235
Záznamy v používané knihovně.....	236
17.5 Návrhový vzor Šablonová metoda	238
Příklad.....	239
17.6 Návrhový vzor Abstraktní továrna	239
Motivace	239
Návrhový vzor Tovární metoda.....	240
Co je abstraktní továrna.....	240
Použití.....	240
Tovární třída převádí konstruktory a statické členy na instanční	241
17.7 Shrnující úloha.....	242
18 Lambda-výrazy, funkční interfejsy a generické typy.....	243
18.1 Nezávislé opakování zadané akce	243
Možná řešení.....	244
18.2 Syntaxe lambda-výrazů.....	244
Aplikace na světlo	245
18.3 Lambda-výrazy a lokální proměnné.....	247
Zásobník návratových adres – ZNA	248
18.4 Lambda-výrazy zastupující metody	249
18.5 Funkční interfejsy	250
18.6 Generické datové typy	250
Motivace	250
Syntaxe zadávání a používání generických typů	251
Specifika generických typů Javy	251
Omezení typových parametrů	252
Příklad: Interval.....	252
Typové parametry s více předky.....	252
Potomci a předci generických typů.....	253
18.7 Funkční interfejsy – pokračování.....	254
Demonstrace použití	255
18.8 Lambda-výrazy nelze přetypovat na Object přímo.....	258
19 Dědění implementace.....	261
19.1 Tři druhy dědění.....	261
Přirozené (nativní) dědění	261
Dědění typu	262
Dědění implementace.....	262
LSP – Liskov Substitution Principle	263
Shrnutí	263
19.2 Základy dědění tříd	263
Univerzální (pra)rodič Object.....	264
Instance třídy Object jako parametr či návratová hodnota.....	264

19.3 Co dědíme od třídy Object.....	264
Class-objekt.....	265
Úplný název tříd v JShell.....	266
Přehled veřejných členů zděděných od třídy Object.....	266
19.4 Definice třídy s předkem – Square	267
Rodičovský podobjekt.....	268
Volání rodičovského konstruktoru	269
19.5 Přebíjení metod.....	270
19.6 Skrytá nebezpečí: úprava třídy Square	272
19.7 Virtuální versus konečné metody	274
19.8 Zakrývání statických metod.....	274
Metody interfejsů se nezakrývají.....	275
19.9 Jediný implementační předek	275
20 Abstraktní třídy.....	276
20.1 Abstraktní třídy a jejich role v dědické hierarchii.....	276
Definice abstraktních tříd a metod	278
public abstract class AX { ... }	278
public abstract void method();	278
Experimenty s abstraktní třídou.....	278
20.2 Účel abstraktních tříd	280
20.3 Zavedení abstraktních tříd do projektu	280
20.4 Návrhový vzor Stav	281
20.5 Aplikace na příklad s vozidlem.....	282
Hlavní třída vozidla – třída Robot4_4	283
Stavově nezávislé metody	286
Stavově závislé metody.....	286
Společný rodič jednosměrných tříd – třída ARobot1_4.....	286
Statická tovární metoda	286
Datové členy a konstruktory	288
Stavově závislé metody	288
Jednostavové (jednosměrné) třídy	288
Test.....	290

Část D Knihovny

293

21 Speciální datové typy.....	294
21.1 Třída Object	294
21.2 Odkazové a hodnotové datové typy	295
21.3 Metody equals(Object) a hashCode()	296
Metody equals(Object)	296
Metoda hashCode()	297
21.4 Proměnné a neměnné hodnotové typy.....	297
21.5 Primitivní a obalové typy	299
21.6 Třída String	299
Možné problémy při práci s některými znaky	300
Interfejs java.lang.CharSequence	300
Problémy s kódováním znaků	301
Třídy StringBuilder a StringBuffer	302
21.7 Výčtové typy.....	303
Složitější příklad: Direction	304
21.8 Třída Throwable	306
21.9 Třída Thread	306
21.10 Třída java.util.Optional<T> & spol.....	306

Motivace	306
Alternativní řešení	307
21.11 Třída java.util.Random	308
int nextInt() long nextLong() int nextInt(int bound)	308
double nextDouble()	308
double nextGaussian()	308
DoubleStream doubles() IntStream ints() LongStream longs()	308
DoubleStream doubles() IntStream ints() LongStream longs()	309
21.12 Interfejs java.lang.Comparable<T>	309
21.13 Interfejs java.util.Comparator<T>	310
22 Výjimky	312
22.1 Co to jsou výjimky	312
22.2 Nejdůležitější výjimky	313
22.3 Vyhození výjimky	314
Reakce systému na vyhození výjimky	314
22.4 Výjimky a nedosažitelný kód	316
22.5 Hierarchie dědění výjimek	316
22.6 Zachycení vyhozené výjimky	318
22.7 Společný úklid – blok finally	319
22.8 Definice vlastních výjimek	321
22.9 Kontrolované výjimky	322
22.10 Převedení kontrolované výjimky na nekontrolovanou	323
23 Kontejnery	325
23.1 Co je to kontejner v Javě	325
23.2 Kategorizace kontejnerů	326
23.3 Knihovna kolekcí	326
Collections – kolekce	327
Set – množina	328
SortedSet – uspořádaná množina	328
List – seznam	328
Queue – fronta	328
Deque – oboustranná fronta	328
Map – Mapa	329
Collections	329
23.4 Deklarujte typy co nejobecněji	329
23.5 Collection<E> – kolekce	330
boolean add(E o)	330
boolean addAll(Collection<? extends E> c)	330
void clear()	330
boolean contains(Object o)	330
boolean containsAll(Collection<?> c)	330
boolean isEmpty()	330
Iterator<E> iterator()	330
boolean remove(Object o)	330
boolean removeAll(Collection<?> c)	331
boolean retainAll(Collection<?> c)	331
int size()	331
23.6 Dvojtečkový cyklus for(;)	331
23.7 Množiny – Set<E>	331
23.8 Seznamy – List<E>	333
Pořadí prvků	333
Příklad: seřazení prvků v seznamu	335

23.9 Slovníky a mapy – Map<K,V>.....	337
Pohledy	338
Set<K> keySet()	338
Collection<V> values().....	338
Set<Map.Entry<K,V>> entrySet()	338
24 Pole.....	339
24.1 Představení	339
24.2 Atributy a metody polí.....	340
24.3 Pole jako kontejner.....	340
Pole odkazů na objekty	340
Pole hodnot primitivních typů	341
Hlídání mezi polí.....	343
Inicializace polí v deklaraci	343
Inicializace vytvářeného pole	344
Neinicializovaná pole objektových typů	345
24.4 Vícerozměrná pole.....	346
Obdélníková pole	346
Neobdélníková pole	347
Inicializace vícerozměrného pole.....	348
24.5 Metody s proměnným počtem argumentů	348
24.6 Pole, kolekce a moderní programování.....	349
24.7 Závěrečný příklad	349
25 Interní datové typy	352
25.1 Přehled.....	352
Terminologie	352
Společné charakteristiky.....	353
Použití	353
Jedna hladina soukromí.....	354
25.2 Globální interní (členské) datové typy	354
25.3 Vnořené datové typy	355
25.4 Vnitřní třídy.....	356
25.5 Lokální třídy	356
Pojmenované lokální třídy	357
Anonymní třídy.....	357
25.6 Návrhový vzor Iterátor.....	358
Interfejsy java.util.Iterator<E> a java.lang.Iterable<E>	358
boolean hasNext()	359
E next()	359
void remove().....	359
Použitelnost cyklu for(:).....	360
Vnější (sekvenční) a vnitřní (dávkové) iterátory	360
26 Datovody.....	361
26.1 Analogie.....	361
26.2 Druhy operací.....	362
26.3 Princip práce datovodu	363
26.4 Specifické vlastnosti	363
26.5 Datové typy související s datovody	364
26.6 Vytváření datovodů	364
Kolekce	365
Pole.....	365
Interfejs java.util.stream.Stream	366
26.7 Koncepční příklad.....	366
26.8 Operace s daty v datovodu.....	367

Koncové operace.....	367
Průběžné operace.....	367
Částečně průběžné operace.....	368
26.9 Kolektory	368
static <T> Collector<T,?,List<T>> toList()	369
static <T> Collector<T,?,Set<T>> toSet()	369
static <T,K,U> Collector<T,?,Map<K,U>> toMap(Function<? super T,? extends K> keyMapper, Function<? super T,? extends U> valueMapper)	369
Příklad.....	369
27 Čtení a ukládání dat	372
27.1 Konceptce čtení a ukládání dat	372
27.2 Soubory: bleskové opakování.....	373
Soubor, souborový systém, cesta.....	373
Relativní, absolutní a kanonická cesta.....	374
Substituované disky ve Windows.....	374
27.3 Třída java.io.File.....	375
Instanční metody	375
Metody vracející cestu či soubor	375
Zjišťovací metody	376
Vytváření souborů a složek	376
Odstraňování souborů a složek	376
Zjišťování obsahu složek	376
27.4 Návrhový vzor Dekorátor	378
Motivace	378
Princip funkce.....	379
27.5 Rozdělení datových proudů.....	380
27.6 Zápis textů.....	381
Splachování a zavírání proudů	382
Přidávání dat na konec existujícího souboru	382
Příklad.....	383
27.7 Čtení textů	385
Příklad.....	386
27.8 Třída java.util.Scanner	387
28 Vytvoření aplikace.....	388
Prohlížení obsahu JAR-souborů.....	388
28.1 JAR-soubor	389
28.2 Demonstrační aplikace.....	389
28.3 Hlavní třída aplikace.....	390
Vytvoření souboru JAR s aplikací	390
28.4 Spuštění aplikace	392
28.5 Soubor MANIFEST_MF	393

Literatura	394
-------------------------	------------

Rejstřík	396
-----------------------	------------

Úvod

Otevíráte čtvrté, zcela přepracované vydání knížky, která vás chce naučit programovat moderním, objektově orientovaným stylem. Stylem, jímž se v dnešní době vyvíjí drtivá většina klíčových aplikací. Oproti předchozím vydáním je kniha od základů přepracovaná. Stejně jako v předchozím vydání je výklad rozdělen do dvou dílů.

První díl (ten právě čtete) se soustředí především na výklad základních syntaktických konstrukcí a architektonických principů, které by si měl čtenář osvojit předtím, než se pustí do kódování složitějších projektů. Jeho cílem je, aby čtenáři přešly tyto principy co nejdříve do krve.

Plánovaný druhý díl pak získané návyky prohloubí a seznámí čtenáře s řadou dalších programátorských technik a prohloubí znalosti standardní knihovny. Postupně se v něm navrhuje jednoduchá, ale na druhou stranu netriviální aplikace. V průběhu tohoto návrhu kniha vysvětluje hlubší souvislosti, na něž v běžných učebnicích již nezbyvá místo, a doplní čtenářovy znalosti zásad návrhu aplikací.

Kniha je výsledkem mnohaletého experimentování s tím, jak co nejlépe učit *soudobé* programování. Vyzkoušeli jsme si výuku programování snad na všech typech zařízení. Učili jsme v zájmových kroužcích na základní škole, na střední škole, na univerzitách i v rozšiřujících kurzech pro profesionální programátory.

Ve svých kurzech neustále zjišťujeme, že střední i vysoké školy opouští řada programátorů, kteří sice programují v nějakém „moderním“ jazyce, ale neumějí programovat moderně. Absolvované kurzy je sice naučily navržený program zakódovat, ale nenaučily je netriviální program samostatně navrhnout. Naučily je používat nejrůznější frameworky, ale oni tyto frameworky používají většinou mechanicky, bez pochopení základních principů, na jejichž základě jsou navrženy.

Řada vysokých škol se v informatických oborech snaží naučit své studenty zakódovat program v řadě nejrůznějších jazyků. Neučí je však styl programování, ale především syntaxi a sémantiku probíraných jazyků. Důsledkem jsou pak nestabilní a těžko udržovatelné aplikace.

Ve svých učebnicích se to pokoušíme napravit. Reakce čtenářů prozatím naznačují, že se nám to snad alespoň částečně daří.

Tato učebnice je již podle svého názvu určena pro naprosté začátečníky: máme-li dodržet akceptovatelný rozsah, musíme probrat opravdu jen základy. Pro zájemce proto chystáme učebnici pro mírně pokročilé, která se nesoustředí na kód, ale bude se věnovat spíše návrhu. Představili bychom v ní další zásady moderního programování, včetně např. automatizovaného testování, a v závěru bychom navrhli jednoduchou, avšak netriviální aplikaci od úplného počátku. Vše záleží na zájmu potenciálních čtenářů.

Komu je kniha určena

Tato kniha je určena především těm, kteří ještě nikdy neprogramovali, anebo se je to sice někdo snažil naučit, ale oni už vše zase zapomněli. Knihu jsme se snažili napsat tak, aby ji mohl použít bystrý středoškolák. Nepředpokládá žádné předběžné znalosti a dovednosti kromě základů práce s počítačem. Jejím cílem je předat čtenáři základní znalosti a naučit ho dovednosti potřebné k vytváření jednoduchých aplikací. Osvojené základy mu pak umožní, aby v případě hlubšího zájmu o programování v jazyku *Java* pokračoval některou z učebnic určených pro mírně pokročilé programátory – nejlépe samozřejmě chystaným druhým dílem a referenční příručkou [\[19\]](#) či její následovnicí.

Koncepce výkladu

Musíme vás upozornit na to, že kniha, kterou držíte v ruce, se od běžných učebnic poněkud liší. Ostatní učebnice jsou totiž většinou především učebnicemi nějakého programovacího jazyka. Jejich autoři se proto ve svém výkladu soustředí hlavně na výklad vlastností popisovaného jazyka a jeho knihoven. Bohužel se v nich ale nedozvíte skoro nic o tom, jak při návrhu programů přemýšlet, aby vás nezaskočily náhlé změny zadání, kterými je současné programování pověstné. Takového učebnice proto nevychovávají návrháře, kteří by uměli program navrhnout, ale pouze kodéry, kteří umějí zanalyzované zadání zakódovat.



Tady si neodpustíme vzpomínku na jednoho studenta, který na konci semestru vysvětloval, že už profesionálně programuje, takže si myslel, že kurz hravě zvládne. Když jsme na začátku semestru probírali naprosté základy, řekl si, že to je pouhé hraní a že si počká, až začneme doopravdy programovat, a pak chybějící body rychle dožene. Když se k nám však v polovině semestru připojil, zjistil, že řadu konstrukcí, které běžně používáme, vůbec nechápe. Nezbylo nám, než mu zopakovat, že mezi prostým používáním objektových konstrukcí a návrhem objektových programů je velký rozdíl a že řada účastníků přichází do našich kurzů právě proto, aby se tento jiný způsob programátorského myšlení naučila.

Vypadá to, jako když autoři předpokládají, že se při čtení jejich knihy naučíte programovat nějak sami od sebe – obdobně, jako se to museli naučit oni. Zkušenosti s programátory, kteří navštěvují naše kurzy ve firmě či na univerzitě, však ukazují, že tohoto výsledku bývá dosaženo jen zřídka.¹ Většina z nich zná poměrně dobře konstrukce nějakého objektově

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

orientovaného programovacího jazyka a základy často používaných frameworků. Bohužel, skoro nikdo z nich v něm neumí objektivně programovat, neumí přepnout na objektivní způsob uvažování.

Pro jistotu proto varuji: toto není učebnice jazyka *Java*, toto je učebnice objektivního programování v *Javě*. My se primárně nebudeme učit, jak program zapsat, ale jak jej navrhnout. Cílem není vychovat z čtenářů kodéry v *Javě*, ale připravit je na to, aby z nich mohli vyrůst schopní architekti. Nebudeme vás proto učit specialitám použitého programovacího jazyka (ty jsou podrobně popsány v knize [19]), ale pokusíme se vás naučit efektivně navrhovat a vytvářet spolehlivé a snadno udržovatelné programy. Jinými slovy: chceme vás naučit dovednostem, které budete používat, ať už budete programovat v jakémkoliv objektivně orientovaném jazyku. Jazyky přicházejí a odcházejí. Základní programátorské techniky a způsob myšlení však žijí daleko déle než jazyky, se kterými byly zavedeny.

Díky tomu, že se místo na programovací jazyk soustředíme spíše na vlastní programování, vznikl v knize prostor pro výklad řady programátorských zásad a technik, o nichž se klasické učebnice vůbec nezmiňují, a to často ani učebnice pro zkušené programátory. Tyto zásady a techniky se většinou přednášejí až v nadstavbových kurzech, které učí vytvářet programy tak, aby s vámi mohly růst a aby vás nezaskočily rychle se měnící požadavky zákazníků (a připravte se na to, že tyto měnící se požadavky vás potkají i v případě, kdy jste zákazníkem sami sobě). Přitom vůbec nejde o techniky složité, které by začátečník nezvládl pochopit. Ostatní učebnice je pomíjejí pouze proto, že nesouvisejí přímo se syntaxí programovacího jazyka, ale týkají se obecného programování.

Tato učebnice je naopak vysvětluje od samého počátku výkladu, protože víme, že byste si je měli osvojit co nejdříve. Ti, kteří se nejprve učí kódovat, a teprve následně se dozvídají, jak lze návrh programu zefektivnit, bývají příliš v zajetí svých zkušeností s kódováním a při návrhu programů se pak zbytečně silně soustředí na některé nepodstatné detaily.

Potřebné vybavení

K vývoji programů budete potřebovat vývojovou sadu JDK, kterou můžete stáhnout na adrese <https://www.oracle.com/java/technologies/javase-downloads.html>. Potřebujete ale sadu JDK 17 nebo mladší. Na starších verzích *Javy* by naše programy nemusely pracovat. Počítejte s tím, že instalační soubory zabírají zhruba 160 MB a po instalaci bude sada zabírat přes 300 MB.

K vývojové sadě je vhodné si stáhnout a nainstalovat i dokumentaci. Pokud ale neopouštíte web, tak můžete využívat i její trvalou instalaci na webu. Musíte se však smířit s tím, že ji seženete pouze anglicky. ZIP soubor s dokumentací zabírá asi 50 MB a po rozbalení bude zabírat zhruba 460 MB.

Pro úspěšný vývoj programů potřebujete ještě vhodný vývojový nástroj. S těmi nejpoužívanějšími vás stručně seznámíme v podkapitole [1.4 Vývojové nástroje](#) na straně [36](#).

Doprovodné programy

Text knihy je prostoupen řadou doprovodných programů. Budete-li si je chtít spustit a ověřit jejich funkci, potřebujete je nejprve stáhnout. Najdete je na stránce knihy na adrese² http://knihy.pecinovsky.cz/67_java_nz. Měli byste zde najít příslušné soubory i jejich stručný popis.

Názvy zdrojových souborů s demonstračními skripty začínají vždy písmenem **s** následovaným dvojmístným číslem kapitoly. Následují-li dvě podtržítka a text (např. **s02__Prostředí_JShell**), jedná se o soubor příkazů zadaných v průběhu kapitoly. Je-li za číslem malé písmeno následované jedním podtržítkem (např. **s01a_script**), jedná se o samostatný skript.

Soubory s příkazy zadávanými v průběhu kapitoly se vyskytují ve dvou podobách se stejným názvem, ale odlišnou příponou. Soubory s příponou **jsh** jsou zdrojové soubory pro program *JShell*, soubory s příponou **jdoc** obsahují výpisy programů nebo záznamy komunikace s prostředím i s čísly řádků uvedenými v knize. Tyto soubory vám mají usnadnit sledování rozboru některých programů, abyste při něm nemuseli neustále listovat mezi rozbohem a rozebíraným výpisem. Jsou určeny k tomu, abyste si je otevřeli v samostatném okně, anebo si je vytiskli.

Doprovodné programy si umístíte, kam uznáte za vhodné. Jak napovídají adresy souborů v jejich úvodních komentářích, máme je rozbalené do složek nazvaných **67_JSH** a **67_WRD**, které jsou na substituovaném disku **K:**.

Použité typografické konvence

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

Termíny	První výskyt nějakého termínu a další texty, které chceme zvýraznit, vysazujeme tučně .
Název	Názvy firem a jejich produktů jsou vysázeny <i>kurzivou</i> . Kurzivou jsou také názvy kapitol, podkapitol a oddílů, na které v textu odkazujeme.
Citace	Texty, které si můžete přečíst na displeji, např. názvy polí v dialogových oknech či názvy příkazů v nabídkách, vysazujeme tučným bezpatkovým písmem .

² Číslicí 67 v adrese se nevzrušujte, jedná se pouze o interní označení pořadí vytvářené knihy, protože bychom v nich jinak bloudili.

- Odkaz** Celá kniha je prošípikovaná křížovými odkazy na související pasáže. Ne-ní-li odkazovaný objekt (kapitola, obrázek, výpis programu, ...) na stejné stránce nebo na některé ze sousedních stránek, je pro čtenáře tištěné verze doplněn o číslo stránky, na níž se nachází. Čtenářům elektronické verze stačí, když na něj klepnou, a použitý prohlížeč by je měl na odkazovaný objekt ihned přenést.
- Adresa** Názvy souborů a internetové adresy jsou obyčejným bezpatkovým písmem.
- Program** Identifikátory a další části programů zmíněné v běžném textu vysazujeme **neproporcionálním písmem**, které je v elektronických verzích pro zvýraznění tmavě červené.
- metoda(?)** Při odkazech na metody budeme v závorkách za názvem metody vždy uvádět seznam typů jejich parametrů – např. `equals(Object)`. Nebude-li v danou chvíli jasné, jaké má zmiňovaná metoda parametry, budeme do závorek psát otazník – např. `metoda(?)`.
- zadání** Ve výpisech programů a odpovědi počítače texty zadávané autorem, vysazeny tučně (v elektronických verzích pro zvýraznění modré).

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



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



Otevřená schránka s dopisy označuje informace o projektu, s nímž budeme v dalším textu pracovat, nebo v němž najdete vzorové řešení aplikující probranou látku. Příslušný projekt získáte pomocí generátoru projektů popsaného výše.



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



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



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



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



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



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



Brýle označují tzv. „poznámky pro štouraly“, v nichž se vás snažíme seznámit s některými zajímavými vlastnostmi probírané konstrukce nebo upozorňujeme na některé souvislosti, které však nejsou k pochopení látky nezbytné.

Odbočka – podšeděný blok

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

Zpětná vazba

Přes veškeré úsilí, které jsme knize věnovali, nemůžeme vyloučit, že v textu či doprovodných příkladech zůstaly skryté nějaké chyby. Předem se za ně omlouváme. Objevíte-li proto v knize nějakou chybu nebo budete-li mít návrh na nějaké její vylepšení, pošlete prosím e-mail s předmětem [67 JavaNZ DOTAZ](mailto:67_JavaNZ_DOTAZ) na adresu rudolf@pecinovsky.cz. Na stránku knihy http://knihy.pecinovsky.cz/67_java_nz se pak pokusíme co nejdříve zanést příslušná errata s opravou, kterou pak zapracujeme do případného dalšího vydání.

Tento e-mail pošlete i v případě, bude-li vám někde připadat text nepříliš srozumitelný nebo budete-li mít nějaký dotaz, ať už k vykládané látce či použitému vývojovému prostředí. Bude-li se dotaz týkat něčeho obecnějšího, zveřejníme na stránce knihy odpověď i pro ostatní, které by mohl obdobný dotaz napadnout za pár dní, anebo jsou natolik ostýchaví, že si netroufnou sami se zeptat.

Dopředu se omlouváme, že vzhledem k velkému pracovnímu zatížení občas odpovídáme na e-maily až se značným zpožděním.

Část A

Superzáklady

Tato část je určena pro čtenáře, kteří se s programováním doposud nikdy nesetkali. Přináší proto řadu základních informací, jež byste měli znát před tím, než začnete doopravdy programovat. Nejprve vás uvede do světa programování a prozradí jeho základní vlastnosti a současné trendy. Poté vám velmi stručně představí nejpoužívanější nástroje pro vývoj programů. Pak poněkud podrobněji představí prostředí *JShell*, které bude po většinu výkladu použito k tomu, abychom získali okamžitou odpověď na zadané programové obraty. Budete tak moci od začátku zkoušet probírané konstrukce, aniž byste museli používat něco, co jsme ještě neprobrali. Na to naváže výklad o datových typech a zadávání jednoduchých hodnot. Část končí představením proměnných a výkladem nejpoužívanějších výrazů a operací.

Kapitola 1

Přehled



Co se v kapitole naučíte

V této kapitole se seznámíte s nástroji, které budete potřebovat při studiu dalších částí knihy. Nejprve vám prozradí něco o historii a současných trendech v programování a seznámí vás s programovacím jazykem a platformou *Java*. Na závěr vám velice stručně představí nejpoužívanější vývojové nástroje.

1.1 Trocha historie

Historie počítačů (tj. strojů, u nichž je postup výpočtu řízen programem) a programování se začala psát již v devatenáctém století. Charles Babbage tehdy dostal zakázku od anglického námořnictva na vylepšení svého počítacího stroje pro výpočet navigačních tabulek, které se hemžily chybami. V roce 1837 dokončil návrh počítacího stroje, který byl řízený programem zadávaným na děrných štítcích. Zbytek života se pak věnoval jeho konstrukci. Stroj byl mechanický a měl být poháněn parním strojem, ale z programátorského hlediska již umožňoval značnou část operací, kterými se honosí současné počítače.

První funkční počítač postavil až v roce 1936 v Německu Konrád Zuse. Nabídl jej armádě, ale tehdejší německá generalita prohlásila, že takový nesmysl nebude armáda nikdy potřebovat. Bůh ví, kam by se vývoj ubíral, kdyby generálové nebyli tak krátkozrací a počítač pomáhal německým inženýrům za druhé světové války při výpočtech.

Další počítače se objevily na konci druhé světové války v USA a ve Velké Británii. Nejslavnějším z nich byl ENIAC, který byl vyroben v roce 1944 a byl prvním čistě elektronickým počítačem (ostatní ještě používaly relé³ či dokonce mechanické prvky). Skutečné počítačové (a tím i programátorské) orgie však vypukly až v padesátých

³ Relé je elektromechanický prvek, kde průchod proudu cívkou zapříčiní sepnutí nebo rozpojení kontaktů. Rychlá relé dokázala sepnout i 100krát za sekundu – to byla také maximální rychlost tehdejších počítačů.

letech, kdy se začaly počítače vyrábět sériově a počítačová věda (computer science) se zabydlela na všech univerzitách.

Problémem prvních počítačů byla jejich vysoká poruchovost. Antonín Svoboda tehdy přišel s řadou konstrukčních vylepšení umožňujících vytvořit z nespolehlivých součástek spolehlivý počítač.⁴ Konstruktéři se postupně naučili vytvářet vysoce spolehlivé počítače. Nejslabším článkem celého komplexu se tak stával program.

Co je to program

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

Cílovým procesorem nemusí být vždy počítač. Oblíbeným příkladem programů jsou např. kuchařské předpisy. V předpočítačových dobách zaměstnávaly některé instituce (např. armáda) velké skupiny počtářů,⁵ jež na mechanických kalkulačkách řešily podle zadaných programů výpočty, které dnes předhazujeme počítači.

Na programu je důležité, že musí být napsán v nějakém programovacím jazyku, kterému rozumí programátor. Napsaný program je pak jiným programem (překladačem) převeden do „jazyka“, kterému rozumí počítač.

Vybrat jazyk pro kuchařský předpis je jednoduché, vybrat jazyk pro počítač je mnohem složitější. Vlastnosti použitého jazyka totiž naprosto zásadně ovlivňují jak rychlost vývoje programu, tak i rychlost a kvalitu výsledných programů. Proto také prošly programovací jazyky i metodiky jejich používání celou řadou revolučních změn.

Postupná změna priorit

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

⁴ V tehdejší době (tj. v padesátých letech) byla naše republika na špičce světového vývoje počítačů. Svobodovy myšlenky byly později uplatněny v počítačích řídících americké kosmické lodě. Jenže to už bylo v době, kdy byl z naší republiky vypuzen, emigroval do USA a tam působil jako špičkový počítačový odborník.

⁵ Přiznejme si, že to tehdy byly většinou počtářky, protože muži dělají při práci podobného druhu příliš mnoho chyb. Takovéto „lidské počítače“ pomáhaly např. s vývojem atomové pumy či prvními lety do vesmíru.

Ještě větším problém nastal v okamžiku, kdy si uživatelé program oblíbili a chtěli po programátorech, aby do něj doplnili další funkce nebo aby upravili vlastnosti, které jim nevyhovovaly. Začalo se hovořit o softwarové krizi.

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

V současné době se začíná v některých oblastech prosazovat i *funkcionální programování*, které se soustřeďuje na efektivní vyřešení některých částí projektů, ale základní paradigma, na němž stojí opravdu rozsáhlé projekty, je nadále objektové.



Termínem *paradigma* označujeme celkový přístup k řešení problému a základní programovací styl. Nejpopulárnější programovací jazyky současné doby (*Java* patří mezi ně) jsou navrženy tak, aby umožňovaly vývoj programů podle několika paradigmat.

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

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

Prakticky každý program zaznamená během svého života řadu změn. Požadavky zákazníka na to, co má program umět, se většinou průběžně mění, a program je proto nutné průběžně upravovat, rozšiřovat a vylepšovat. Celé současné programování je proto vedeno snahou psát programy nejenom tak, aby pracovaly efektivně, tj. rychle a s minimální spotřebou různých zdrojů (operační paměť, prostor na disku, kapacita sítě atd.), ale aby je také bylo možné kdykoliv jednoduše upravit a vylepšit.

Předchozí zásady krásně shrnul Martin Fowler ve své knize *Refactoring* ([5], [6]):

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

Mějte při tvorbě svých programů tuto zásadu neustále na paměti.

Vývoj metodik programování

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

První revolucí bylo zavedení vyšších programovacích jazyků, jež programátory osvobodily od zapisování programů v podobě, která přesně odpovídala použitým instrukcím použitého počítače, a zavedly vyjadřování, které bylo daleko bližší zvyklostem lidí. To pak umožnilo vědcům a technikům nečekat s každou prkotinou na programátora a naprogramovat si spoustu jednodušších výpočtů sami.

Zavedení vyšších programovacích jazyků umožnilo tvorbu složitějších programů. Vyvíjené programy byly zanedlouho tak složité, že se v nich programátoři přestali orientovat. Modulární programování ukazovalo, že rychlost vývoje i kvalitu výsledného programu zvýšíme, když podle starého římského hesla *divide et impera* (rozděl a panuj) vhodně rozdělíme velký projekt do sady menších, rozumně samostatných modulů.

Produktivita programátorské práce se zvýšila, ale záhy se začalo ukazovat, že programátoři mají problémy nejenom s rozchozením obřích programů, ale i jednotlivých modulů. V roce 1968 publikoval E. W. Dijkstra článek,⁶ který kritizoval v té době převažující způsob psaní programů a prosazoval tzv. *strukturované programování*. Jeho článek rozpoutal obrovské diskuse, které krásně charakterizoval jiný článek nazvaný *Real Programmers Don't Use Pascal*⁷ a publikovaný v červnu 1983 v časopise *Datamation* (v češtině vyšel pod názvem *Opravdoví programátoři nepoužívají Pascal*⁸ a vřele vám doporučujeme, abyste si jej přečetli).

Strukturované programování se soustředilo na kód a ukazovalo, že dalšího zvýšení produktivity vývoje i kvality výsledných programů dosáhneme dodržením několika jednoduchých zásad při vlastním psaní kódu. Jeho hlasatelé předváděli, že opuštěním nejrůznějších fint, kterými se programátoři snažili o optimalizaci kódu, a maximálním zpřehledněním vytvářeného programu dosáhneme nejenom vyšší produktivity programátora, ale v mnoha případech i vyšší efektivity výsledného programu.

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

⁶ Dijkstra, E. W. "Letters to the editor: go to statement considered harmful". Communications of the ACM 11 (3): 147–148. DOI:10.1145/362929.362947. ISSN 0001-0782.

⁷ Najdete jej např. na adrese <http://www.webcitation.org/659yh1oSh>, ve Wikipedii pak má vlastní heslo [Real Programmers Don't Use Pascal](#).

⁸ Na české wiki zatím heslo nemá, ale po zadání hesla *opravdoví programátoři* vrátí Google řadu odkazů – např. <http://www.logix.cz/michal/humornik/Pojidaci.Kolacu.xp>.

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

Objektově orientované programování (v dalším textu budeme používat zkratku OOP) se proto obrátilo do vyšších hladin abstrakce a ukázalo, že vhodným zvýšením abstrakce dokážeme rychle a spolehlivě navrhovat a vyvíjet ještě větší a komplikovanější projekty. Studie z konce osmdesátých let dokonce ukázaly, že programy obsahující více než cca 100 000 příkazů již není v lidských silách naprogramovat s akceptovatelnou trojicí (*spolehlivost, doba vývoje, cena*) bez použití OOP.

Jak už jsme řekli, poslední dobou se vedle OOP prosazuje i funkcionální programování. To sice neaspiruje na pozici základního paradigmatu při návrhu rozsáhlých systémů, ale umožňuje efektivně navrhnout některé jejich části.

OOP a OFP

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

OOP je považováno za hlavní proud současného programování. Všechny rozšířené moderní programovací jazyky se honosí tím, že umožňují navrhovat programy objektově. Podpora OOP byla postupně doplněna i do těch jazyků, které vznikly dávno před tím, než se začalo o nějakém OOP hovořit. Prakticky žádný z nově vznikajících jazyků aspirujících na široké použití si již nedovolí nepodporovat OOP.

Funkcionální programování pak přichází s některými zásadami, jejichž dodržení usnadňuje návrh programů, u nichž je vhodné jejich řešení kvůli zvýšení rychlosti rozdělit mezi více procesorů. Jeho použití má své výhody i v některých dalších oblastech. I s jeho základy vás v této učebnici seznámíme. Hlavní paradigma současnosti bychom tak mohli označit za *objektově-funkcionální programování* – OFP.

1.2 Překladače, interprety, platformy

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

Operační systém a platforma

Operační systém je sada programů, jejímž úkolem je zařídit, aby počítač co nejlépe sloužil zadanému účelu. Operační systémy osobních počítačů se snaží poskytnout co největší komfort a funkčnost jak lidským uživatelům, tak programům, které operační systém nebo tito uživatelé spouští. (Teď nehodnotíme, jak se jim to daří.)

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

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

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

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

Nejrozšířenější platformou HWOS současných osobních počítačů je operační systém *Windows* na počítačích s procesory kompatibilními s procesorem *Intel Pentium*, které však mají i verze běžící na počítačích s jinými procesory. Další vám známé platformy budou nejspíš operační systémy z rodiny *Linux*, mezi něž svým způsobem patří i operační systém *MacOS* běžící na počítačích *Apple*. V našem přehledu bychom neměli zapomínat ani na platformu *Android* určenou pro mobilní telefony a tablety, které však svým výkonem občas překonají leckterý osobní počítač.

Programovací jazyky

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

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

Způsoby zpracování programu

Abychom měli z vytvořených programů nějaký užitek, musíme je spustit. K tomu se používá několik přístupů.

Překládaný program

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

Interpretovaný program

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

Porovnání

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

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

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

Hybridní programy

Vedle těchto základních druhů programů existují ještě hybridní programy, které jsou současně překládané i interpretované a které se snaží sloučit výhody obou skupin. Hybridní program se nejprve přeloží do jakéhosi mezijazyka, který je vymyšlen tak, aby jej bylo možné co nejrychleji interpretovat. Takto přeložený program je potom interpretován speciálním interpretem často označovaným jako **virtuální stroj**.⁹

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

⁹ Virtuální stroj se mu říká proto, že se vůči programu v mezijazyku chová obdobně, jako se chová procesor vůči programu v čistém strojovém kódu.

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



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

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

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

1.3 Java a její zvláštnosti

O splnění zásad objektově orientovaného programování se snaží tzv. objektově orientované jazyky. Na počátku tohoto století mezi nimi získal největší popularitu programovací jazyk *Java*, který se narodil v roce 1995 a hned po svém vzniku zaznamenal velký ohlas. Ještě v průběhu devadesátých let se stal nejpoužívanějším programovacím jazykem a svoji oblíbenost si od té doby stále udržuje. V oblasti návrhu rozsáhlých systémů dnes nemá konkurenci. V současné době v něm vyvíjí (alespoň podle firmy Oracle) přes 9 miliónů programátorů.

Java je jazyk i platforma

Jedním z klíčových záměrů autorů Javy bylo vytvořit nejenom jazyk, ale celou platformu. Programům napsaným v *Javě* pak bude jedno, na jaké platformě z rodiny HWOS běží, protože nad touto platformou bude platforma *Javy*, a ta bude z hlediska dané aplikace na všech počítačích stejná, nezávisle na podkladové platformě HWOS.

Platformu *Javy* realizuje výše zmíněný virtuální stroj spolu se základní knihovnou nejpoužívanějších funkcí.

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

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

Marketingové oddělení firmy Sun se při uvádění Javy rozhodlo pojmenovat jazyk i platformu stejně (obáváme se, že ani netušili, že jsou to dvě různé věci). Z toho ale vycházela řada nedorozumění. Musíme vždy rozlišovat, kdy hovoříme o platformě a kdy o jazyku.

Na platformě Java běží nepřehledné množství nejrůznějších programovacích jazyků.¹⁰ Jazyk Java je pouze jedním z nich. Programy napsané v různých jazycích jsou však většinou schopny komunikovat a spolupracovat, takže můžete část svého programu napsat např. v Javě a druhou část v nějakém jazyku, který je pro daný úkol výhodnější (velmi populární je např. jazyk *Groovy*).

1.4 Vývojové nástroje

Při vývoji programů se používají nejrůznější nástroje, které mají programátorům (a nejen jim) maximálně ulehčit práci.

Základní vývojářská sada

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

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

¹⁰ Ty nejznámější najdete např. ve Wikipedii pod heslem [List of JVM languages](#), ale skutečný počet je mnohem větší a obnáší několik set jazyků a jejich verzí.

Vy se chcete naučit pomocí této učebnice programovat, budete tedy potřebovat JDK (doporučuji vám pořídit si verzi **17** či vyšší). Uživatelům, kteří pak budou spouštět vaše programy, stačí JRE. V dalším textu budu předpokládat, že máte JDK nainstalováno podle pokynů v pasáži [Potřebné vybavení](#) na straně [21](#).

IDE

S JDK při vývoji programů teoreticky vystačíte (některé učebnice ani nic jiného nepoužívají), nutí vás však starat se o řadu konfiguračních detailů, které odvádějí vaši pozornost od vlastního programování. Převážná většina programátorů proto používá programy označované jako IDE, což je zkratka z anglického *Integrated Development Environment* – integrované vývojové prostředí. Ty za ně vyřeší konfigurační detaily a navíc jim pomohou i v řadě dalších oblastí.

Problémem těch „dokonalejších“ bývá, že zvládnutí daného vývojového prostředí je často náročnější než zvládnutí použitého programovacího jazyka. Při následné práci se ale vynaložená námaha bohatě vrátí, protože dobré IDE udělá řadu věcí za vás a v řadě dalších vás povede „za ručičku“ nebo vám alespoň bude dobře napovídat.

Vývojových prostředí pro *Javu* existuje celá řada.¹¹ Některá jsou určena spíše začátečníkům, jiná jsou optimalizovaná pro profesionální programátory. Práce s těmi profesionálními je sice poněkud náročnější, ale pokud překonáte počáteční stadium, kdy vám bude připadat, že zvládnutí daného IDE je těžší než zvládnutí programovacího jazyka, tak se vám vyvinuté úsilí při pozdějším vývoji složitějších programů vrátí.

Pojďme si nyní alespoň stručně představit ta nejrozšířenější IDE.

JShell

JShell není plnohodnotné vývojové prostředí. Je to prostředí určené spíše pro experimenty a rychlé ověřování některých nápadů. Jeho hlavní výhodou je, že je součástí standardní instalace, takže je ihned k dispozici.

Druhou výhodou tohoto prostředí je, že netrvá na tom, abyste zadali celý program, ale je ochotné zpracovat i pouhé úryvky. Proto jej budeme používat u značné části AHA-příkladů demonstrujících funkci aktuálně probíraných konstrukcí. Podrobněji vám toto prostředí představíme v následující kapitole.

BlueJ¹²

BlueJ je vývojové prostředí navržené speciálně pro podporu výuky ve vstupních kurzech programování. Je maximálně jednoduché a názorné, takže si jej v pohodě osvojíte za půl hodiny. Jeho velkou výhodou je, že zobrazuje program primárně prostřednictvím diagramu tříd, což vám pomůže pochopit celou řadu zásad moderního programování a jejich aplikaci ve vlastních programech.

¹¹ Jejich stručný (i když ne kompletní) přehled najdete např. ve Wikipedii na adrese https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Java.

¹² Prostor BlueJ stáhnete na <https://bluej.org/>.

Nevýhodou BlueJ je, že málo napovídá (uživatel proto musí řadu operací realizovat „ručně“) a nepodporuje žádné náročnější operace.

K tomuto vývojovému prostředí přejdeme ve chvíli, kdy začneme vytvářet samostatné programy. Důvodem je hlavně to, že práci s BlueJ můžeme vyložit na několika stránkách, kdežto výklad práce s následně uvedenými vývojovými prostředími by vydal na samostatnou publikaci. Jejich plnohodnotné ovládnutí, po němž už na vás nebudou chystat žádné podrazy, je pro mnohé začátečníky náročnější než ovládnutí základů jazyka.

Pro mnohé studenty je výhodné i to, že vývojové prostředí *BlueJ* je lokalizované do češtiny a slovenštiny.

NetBeans¹³

Zárodek tohoto prostředí vznikl v roce 1996 jako závěrečná práce na MFF UK a jeho vývojové centrum sídlí stále v Praze. Po členité historii, kdy se několikrát měnili vlastníci, je v současné době vyvíjeno v rámci nadace *Apache Software Foundation*.

IDE *NetBeans* je nejstarší z „velké trojky“ profesionálních vývojových prostředí. Jeho popularita je poměrně stabilní a používá je 10–15 % vývojářů. Recenze o něm říkají, že z těch opravdu profesionálních je pro začínající programátory nejsnáze zvládnutelné, aniž by to nějak omezovalo jeho schopnosti. Kromě toho mu dávají přednost ti, kteří potřebují budovat velké systémy a nejsou ochotni za vývojové nástroje platit.

IntelliJ IDEA¹⁴

Toto IDE vydala v lednu 2001 pražská firma *JetBrains*. Přestože zpočátku nabízela pouze placenou verzi, získalo si mezi profesionálními programátory velkou oblibu. Když ale v srpnu 2013 vyšla *Community edition*, kterou bylo možné legálně stáhnout a používat zdarma, začala popularita tohoto prostředí raketově stoupat, takže je dnes používá přes 60 % vývojářů, přičemž placenou verzi používá okolo 50 % (zbylých 10 % jsou studenti a začínající programátoři).

Nevýhodou tohoto prostředí je, že je poněkud těžší jej plně zvládnout. Navíc jeho inteligence a nápověda občas začátečníkům překáží a mate je. Budete-li se však programování věnovat déle, vynaložená námaha se jistě vrátí.

Kromě toho lze s *Community edition* vyvíjet jen středně velké programy a pro vývoj opravdu velkých programů si musíte zaplatit verzi *Ultimate*. Její uživatelé ale tvrdí, že se jim vynaložené prostředky bohatě vrací.

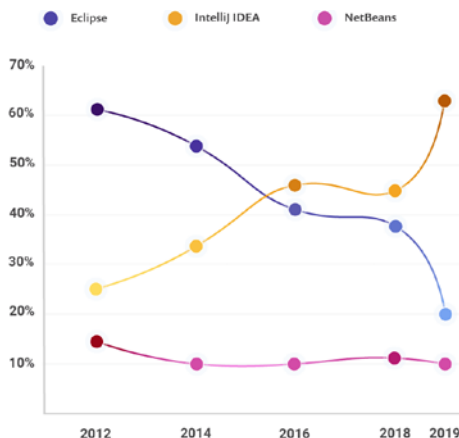
Eclipse¹⁵

Poslední IDE z „velké trojky“. Narodilo se v srpnu 2001 pod patronací firmy IBM. Protože bylo zdarma a IBM se podařilo vytvořit velkou koalici firem, které je pomáhaly zdokonalovat, používalo je zanedlouho přes 60 % vývojářů v *Javě*. V posledním desetiletí ale začala jeho popularita klesat, takže je v současné době používá přibližně 20 % vývojářů a jejich podíl dále klesá (viz graf na obrázku 1.1. převzatý ze studie [23]).

¹³ Prostedí *NetBeans* stáhnete na <https://netbeans.apache.org/download/index.html>.

¹⁴ Prostedí *IntelliJ IDEA* stáhnete na <https://www.jetbrains.com/idea/download/>.

¹⁵ Prostedí *Eclipse* stáhnete na <https://www.eclipse.org/downloads/>.



Obrázek 1.1:
Průběh popularity IDE z „velké trojky“

Visual Studio Code¹⁶

V roce 2015 vydal *Microsoft* vývojové prostředí *Visual Studio Code*, které bylo (mohli bychom říci, že k všeobecnému překvapení) pod otevřenou licencí MIT, takže se zveřejněným a všeobecně dostupným zdrojovým kódem. Toto prostředí bylo primárně určeno pro vývoj programů pro .NET, ale rychle se začaly objevovat pluginy pro další jazyky a platformy, mezi jinými i pro Javu. Rychle získávalo oblibu mezi studenty a dnes patří k velmi rozšířeným obecně použitelným vývojovým prostředím.

Shrnutí

Jak už jsme naznačili, pro demonstraci látky budeme většinou používat prostředí *JShell*. Když pak v části [C](#) začneme definovat celé třídy, tak bude jedno, které vývojové prostředí použijete, protože zdrojové kódy jsou stejné.

Pro první kroky bychom naprostým začátečníkům doporučovali používat prostředí *BlueJ*, protože je nejjednodušší a jeho zvládnutí nebude odvádět pozornost od vlastního návrhu programu. Navíc je to zdaleka nejefektivnější nástroj pro rychlý návrh a zobrazení základní architektury prostřednictvím diagramu tříd, pro což jej budeme v této učebnici využívat především.

Ve chvíli, kdy se vám bude zdát, že už jste si základy návrhu objektově orientovaných programů osvojili, a budete rozhodnutí se programování dále věnovat, můžete přejít na některé z prostředí „velké trojky“ nebo na *Visual Studio Code*.

Kterému z nich dáte přednost, necháme na vás. Obecně radíme studentům, že budou-li mít kamaráda, kterým jim některé prostředí doporučí s tím, že jim pomůže s jeho zvládnutím a řešením případných problémů, tak ať jej poslechnou. Pokud takového kamaráda nemají, budou pro start asi nejvhodnější *NetBeans*.

¹⁶ Prostředí *Visual Studio Code* lze stáhnout na <https://github.com/Microsoft/vscode>.

Kapitola 2

Prostředí JShell



Co se v kapitole naučíte

Tato kapitola vás stručně seznámí s prostředím *JShell*, které budeme používat pro demonstraci základních konstrukcí jazyka a které je vynikajícím nástrojem pro nejrůznější experimenty. Dopředu se omlouváme, že při výkladu použijeme pár termínů, které vysvětlíme až v následujících kapitolách.

2.1 Prostředí JShell

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



Zde vám představím jen nejzákladnější vlastnosti prostředí *JShell*. Koho vlastnosti tohoto prostředí zaujmou a chtěl by je pro své experimenty používat častěji, tomu doporučíme publikaci [\[17\]](#) nazvanou [Java 9 – JShell](#).

Spuštění programu JShell

Prostředí *JShell* spusťte příkazem

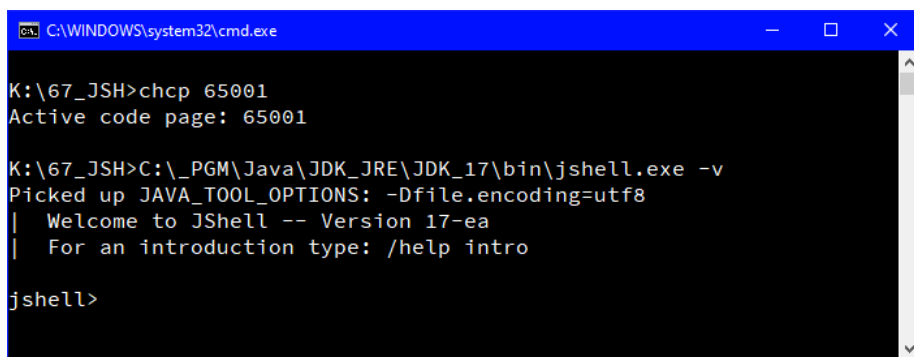
```
jshell -v
```

kde argumentem **-v** žádáte *JShell*, aby nám o všech provedených akcích referoval co nejpodrobněji. Až budete zkušenější, můžete tento argument vynechat, případně jej nahradit jiným, kterým naopak žádáte o stručnější odpovědi. Podrobnosti najdete v publikaci [\[17\]](#).



Pracujete-li pod *Windows*, bude spuštění maličko náročnější, protože *Windows* mají pro různé skupinky států nastavená různá kódování. Všechny doprovodné programy k tomuto kurzu ale budou kódovány v celosvětově jednotném kódování UTF-8. Návod, jak vše připravit pro spuštění pod *Windows*, najdete v příloze [A Příprava spuštění JShell pod Windows](#) na webové stránce knihy.

Po spuštění programu se otevře konzolové okno podobné oknu na obrázku [2.1](#). Lišit se budou pouze v cestách k aktuální složce vypisovaných na počátku prvního a čtvrtého řádku a v cestě k programu `jshell.exe` ve čtvrtém řádku. Uživatelům systémů *Linux* a *MacOS* budou chybět první tři řádky (nastavení kódové stránky, oznámení nastavené stránky a oddělující prázdný řádek) a řádek s oznámením o nastaveném kódování.



```
C:\WINDOWS\system32\cmd.exe

K:\67_JSH>chcp 65001
Active code page: 65001

K:\67_JSH>C:\_PGM\Java\JDK_JRE\JDK_17\bin\jshell.exe -v
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=utf8
| Welcome to JShell -- Version 17-ea
| For an introduction type: /help intro

jshell>
```

Obrázek 2.1:

Konzolové okno otevřené po spuštění dávky `!_JShell.bat` na mém počítači

V dalším textu již nebudeme ukazovat výpisy z okna ve formě obrázků, ale ve formě klasických výpisů programů s číslovanými řádky, abychom se mohli na čísla řádků snáze odvolávat. Obsah okna na obrázku [2.1](#) je ve výpisu [2.1](#).

Výpis 2.1: *Obsah konzolového okna otevřeného po spuštění dávky `!_JShell.bat` na mém počítači*

```
1 K:\67_JSH>chcp 65001
2 Active code page: 65001
3
4 K:\67_JSH>C:\_PGM\Java\JDK_JRE\JDK_17\bin\jshell.exe -v
5 Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=utf8
6 | Welcome to JShell -- Version 17-ea
7 | For an introduction type: /help intro
8
9 jshell>
```

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

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

2.2 Úryvky (snippets)

Výrazy (anglicky *expressions*), **příkazy** (anglicky *statements*), **deklarace** (anglicky *declarations*) a **definice** (anglicky *definitions*) jazyka *Java*, které za výzvu napíšete, se hromadě označují jako **úryvky** (anglicky *snippets*). Kdykoliv napíšete nějaký úryvek, prostředí *JShell* jej ihned vyhodnotí a uloží výsledek. Ve výpisu [2.2](#) najdete zadání dále popsanych úryvků spolu s odpověďmi prostředí.

Výpis 2.2: První tři pokusné úryvky

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

Zkuste napsat jednoduchý aritmetický výraz, např. `6+5` (řádek [1](#) výpisu), a potvrdit jej stiskem ENTER. Jak už bylo řečeno, prostředí *JShell* výraz spočítá a na dalším řádku oznámí jak výsledek, tak jeho uložení do pomocné proměnné nazvané `$1`.



Úplným začátečníkům prozradím, že **proměnná** (anglicky *variable*) je místo v paměti, kam si můžeme uložit nějakou hodnotu pro budoucí použití. Proměnných může být v programu více, a proto má každá přidělené svoje jméno. Když pak později chceme uloženou hodnotu využít, odvoláme se na ni jménem příslušné proměnné a virtuální stroj dosadí na dané místo hodnotu, jež je v dané proměnné uložena. O proměnných si budeme podrobněji povídat v kapitole [4 Proměnné a výrazy](#) na straně [65](#).