Sadržaj

Predgovor xxix

Deo I Jezik Java

Poglavlje 1 Istorija i evolucija Jave 3
Poreklo Jave 3
Rađanje savremenog programiranja: C 4
C++: sledeći korak 5
Sve je spremno za pojavu Jave 6
Nastanak Jave 6
Veza sa jezikom C# 7
Kako je Java uticala na internet 8
Java apleti 8
Bezbednost koda 8
Prenosivost koda 9
Magija Jave: bajt kôd 9
Prevazilaženje apleta 10
Servleti: Java na serverskoj strani 11
Pojmovi karakteristični za Javu 11
Jednostavno 11
Objektno orijentisano 11
Robusno 12
Višenitno 12
Nezavisno od platforme 12
Interpretirano i visokoefikasno 13
Distribuirano 13
Dinamičko 13
Evolucija Jave 13
Java SE 9 16
Kultura inovacija 17

Poglavlje 2 Pregled jezika Java 19
Objektno orijentisano programiranje 19
Dva modela 19
Apstrakcija 20
Tri principa objektno orijentisanog programiranja 20

Prvi jednostavan program 25
Unošenje programa 25
Kompajliranje programa 25
Detaljnije razmatranje prvog jednostavnog programa 26
Drugi kratak program 28
Dve upravljačke naredbe 29
Naredba if 29
Petlja for 31
Blokovi naredaba 32
Jezička pitanja 33
Beline 33
Identifikatori 33
Literali 34
Komentari 34
Separatori 34
Rezervisane reči 35
Javine biblioteke klasa 35
Poglavlje 3 Tipovi podataka, promenljive i nizovi 37
Java je strogo tipiziran jezik 37
Prosti tipovi 37
Celobrojni tipovi 38
byte 39
short 39
int 39
long 39
Brojevi s pokretnim zarezom 40
float 40
double 40
Znakovi 41
Logičke vrednosti 42
Detaljnije o literalima 43
Celobrojni literali 43
Literali za brojeve s pokretnim zarezom 44
Logički literali 45
Literali za pojedinačne znakove 45
Tekstualni literali 46
Promenljive 46
Deklarisanje promenljive 46
Dinamičko inicijalizovanje promenljivih 47
Opseg vidljivosti i životni vek promenljive 47
Automatska i eksplicitna konverzija tipova 49
Javina automatska konverzija tipova 50
Eksplicitna konverzija nekompatibilnih tipova 50
Automatsko unapređivanje tipova u izrazima 51
Pravila za unapređivanje tipova 52

Nizovi . . . 52
Jednodimenzionalni nizovi . . . 52
Višedimenzionalni nizovi . . . 55
Alternativna sintaksa za deklarisanje nizova . . . 59
Nekoliko reči o znakovnim nizovima . . . 59
Poglavlje 4 Operatori . . . 61
Aritmetički operatori . . . 61
Osnovni aritmetički operatori . . . 62
Operator ostatka deljenja (modulo) . . . 63
Kombinovani aritmetički operatori za dodeljivanje vrednosti . . . 63
Inkrementiranje i dekrementiranje . . . 64
Operatori za bitove . . . 65
Logički operatori nad pojedinačnim bitovima . . . 67
Pomeranje bitova ulevo . . . 69
Pomeranje bitova udesno . . . 70
Pomeranje udesno uz dopunjavanje nulama . . . 71
Dodeljivanje pomoću kombinovanih operatora nad bitovima . . . 72
Operatori za poređenje . . . 73
Operatori za logičke vrednosti . . . 74
Kratkospojni logički operatori . . . 75
Operator za dodeljivanje vrednosti . . . 76
Operator ?: . . . 76
Redosled prioriteta operatora . . . 77
Upotreba zagrada . . . 78
Poglavlje 5 Upravljačke naredbe . . . 79
Javine uslovne naredbe . . . 79
Naredba if . . . 79
Naredba switch . . . 82
Naredbe za iteracije . . . 86
Petlja while . . . 87
Petlja do-while . . . 88
Petlja for . . . 90
For-Each verzija petlje for . . . 94
Ugnežđene petlje . . . 98
Naredbe za skokove . . . 99
Upotreba naredbe break . . . 99
Upotreba naredbe continue . . . 103
Upotreba naredbe return . . . 104
Poglavlje 6 Uvod u klase . . . 107
Osnovni pojmovi o klasama . . . 107
Opšti oblik klase . . . 107
Primer jednostavne klase . . . 108
Deklarisanje objekata . . . 111
Detaljnije razmatranje operatora new . . . 112
Dodeljivanje reference na objekat promenljivoj . . . 112

Uvod u metode 113
Dodavanje metode klasi Kutija 114
Vraćanje vrednosti iz metode 115
Dodavanje metode koja ima parametre 117
Konstruktori 119
Konstruktori s parametrima 120
Rezervisana reč this 121
Skrivanje promenljivih instanci 122
Sakupljanje smeća 122
Klasa Stek 123
Poglavlje 7 Metode i klase izbliza 127
Preklapanje metoda 127
Preklapanje konstruktora 130
Upotreba objekata kao parametara 132
Detaljnije o prosleđivanju argumenata 134
Vraćanje objekata kao rezultata metoda 135
Rekurzija 136
Uvod u upravljanje pristupom članovima klase 138
Rezervisana reč static 142
Rezervisana reč final 143
Ponovo o nizovima 144
Ugnežđene i interne klase 145
Upoznavanje mogućnosti klase String 148
Zadavanje argumenata na komandnoj liniji 150
Vararg: Metode s promenljivim brojem argumenata 151
Poglavlje 8 Nasleđivanje 157
Osnove nasleđivanja 157
Pristup članovima klase i nasleđivanje 159
Praktičniji primer 160
Promenljiva tipa natklase može da referencira objekat čiji je tip potklasa 161
Rezervisana reč super 162
Upotreba rezervisane reči super za pozivanje konstruktora natklase 163
Drugi oblik upotrebe rezervisane reči super 166
Izrada višestepene hijerarhije 167
Kada se izvršavaju konstruktori 169
Redefinisanje metoda 170
Dinamičko razrešavanje metoda pri izvršavanju 173
Čemu redefinisanje metoda? 174
Primena redefinisanja metoda 174
Upotreba apstraktnih klasa 176
Modifikator final i nasleđivanje 179
Upotreba modifikatora final za sprečavanje redefinisanja 179
Upotreba modifikatora final za sprečavanje nasleđivanja 179
Klasa Object 180

Poglavlje 9 Paketi i interfejsi 181
Paketi 181
Definisanje paketa 182
Pronalaženje paketa i sistemska promenljiva CLASSPATH 182
Kratak primer paketa 183
Paketi i ograničavanje pristupa njihovim članovima 184
Primer upravljanja pristupom 185
Uvoženje paketa 188
Interfejsi 189
Definisanje interfejsa 190
Implementiranje interfejsa 191
Ugnežđeni interfejsi 193
Upotreba interfejsa 194
Promenljive u interfejsima 197
Interfejsi se mogu nasleđivati 199
Podrazumevane implementacije metoda interfejsa 200
Osnove upotrebe podrazumevanih implementacija metoda 201
Praktičniji primer 202
Pitanja višestrukog nasleđivanja 203
Statičke metode u interfejsu 204
Privatne metode interfejsa 204
Završne napomene u vezi s paketima i interfejsima 206
Poglavlje 10 Obrada izuzetaka 207
Osnove obrade izuzetaka 207
Tipovi izuzetaka 208
Nepresretnuti izuzeci 209
Upotreba naredaba try i catch 210
Prikazivanje opisa izuzetka 211
Više odredaba catch u istoj naredbi try 211
Ugnežđene naredbe try 213
Naredba throw 215
Odredba throws 216
Blok finally 217
Javini ugrađeni izuzeci 219
Izrada sopstvenih izuzetaka 220
Ulančani izuzeci 222
Tri dodatne mogućnosti izuzetaka 224
Primena izuzetaka 225
Poglavlje 11 Višenitno programiranje 227
Javin model programske niti 228
Prioritet izvršavanja programskih niti 229
Sinhronizovanje 229
Razmena poruka između niti 230
Klasa Thread i interfejs Runnable 230
Glavna programska nit 231

Pokretanje nove niti 232
Implementiranje interfejsa Runnable 233
Proširivanje klase Thread 234
Izbor rešenja. 235
Pravljenje više niti 236
Upotreba metoda isAlive() i join() 237
Prioritet izvršavanja niti 239
Sinhronizovanje. 240
Upotreba sinhronizovanih metoda 240
Naredba synchronized. 243
Komuniciranje između niti. 244
Uzajamna blokada 248
Suspendovanje, nastavljanje i zaustavljanje niti. 250
Utvrđivanje stanja niti 253
Upotreba fabričkih metoda za formiranje i pokretanje niti 254
Kada treba koristiti mogućnosti višenitnog rada. 255
Poglavlje 12 Nabrajanja, automatsko pakovanje i anotacije 257
Nabrajanja 257
Osnove nabrajanja 257
Metode values() i valueOf() 259
Javina nabrajanja su vrste klasa 261
Sva nabrajanja nasleđuju natklasu Enum 263
Još jedan primer nabrajanja 264
Omotači za proste tipove 266
Automatsko pakovanje 268
Automatsko pakovanje i metode 269
Automatsko pakovanje/raspakivanje u izrazima 270
Automatsko pakovanje/raspakivanje vrednosti tipa Boolean i Character 271
Automatsko pakovanje/raspakivanje sprečava greške. 272
Upozorenje 273
Anotacije 273
Osnovno o anotacijama. 273
Zadavanje pravila zadržavanja anotacije 274
Učitavanje anotacija pomoću refleksije u vreme izvršavanja koda . . 275
Interfejs AnnotatedElement 280
Upotreba podrazumevanih vrednosti. 280
Markirne anotacije. 281
Jednočlane anotacije 282
Ugrađene anotacije 283
Anotacije tipova 285
Ponavljanje anotacija 289
Neka ograničenja. 291
Poglavlje 13 Ulazno-izlazne operacije, naredba Try-with-Resources i druge teme . . . 293
Osnove ulazno-izlaznih operacija. 293
Tokovi 294
Binarni tokovi i znakovni tokovi 294

Učitavanje ulaznih podataka sa konzole 296
Učitavanje pojedinačnih znakova 297
Učitavanje znakovnih nizova 298
Ispisivanje podataka na konzoli 299
Klasa PrintWriter 300
Čitanje datoteka i upisivanje podataka u njih 301
Automatsko zatvaranje datoteka 306
Modifikatori transient i volatile 309
Operator instanceof 310
Modifikator strictfp 312
Upotreba mašinskog koda 312
Naredba assert 313
Opcije za uključivanje i isključivanje provere tvrdnji 315
Uvoženje statičkih članova 316
Pozivanje preklopljenih konstruktora pomoću rezervisane reči this() . . 318
Napomena u vezi s kompaktnim profilima API biblioteke 320
Poglavlje 14 Generički tipovi 321
Šta su generički tipovi? 322
Jednostavan primer s generičkim tipovima 322
Generički tipovi rade isključivo s referentnim tipovima 325
Generički tipovi se razlikuju po svojim argumentima tipa 326
Kako generički tipovi poboljšavaju bezbednost upotrebe tipova 326
Generička klasa s dva parametra tipa 328
Opšti oblik generičke klase 329
Ograničeni tipovi 329
Upotreba džokerskih argumenata 332
Ograničeni džokerski argumenti 334
Izrada generičke metode 339
Generički konstruktori 341
Generički interfejsi 342
Sirovi tipovi i stari kôd 344
Hijerarhije generičkih klasa 346
Upotreba generičke natklase 346
Generička potklasa 348
Poređenje tipova unutar generičke hijerarhije u veme izvršavanja koda 349
Eksplicitna konverzija tipova 351
Redefinisanje metoda u generičkoj klasi 351
Automatsko određivanje stvarnog tipa generičkog tipa 353
Čišćenje tipova 354
Metode za premošćavanje 354
Greške višeznačnosti 356
Neka ograničenja generičkih tipova 357
Ne mogu se praviti instance parametara tipa 357
Ograničenja za statičke članove 357
Ograničenja za generičke nizove 358
Ograničenje izuzetaka u generičkim klasama 359

Poglavlje 15 Lambda izrazi 361
Uvod u lambda izraze 361
Osnove lambda izraza 362
Fukcionalni interfejsi. 363
Nekoliko primera lambda izraza. 364
Blok lambda izrazi. 367
Generički fukcionalni interfejsi 368
Prosleđivanje lambda izraza kao argumenata 370
Lambda izrazi i izuzeci 372
Lambda izrazi i ograđivanje promenljivih 374
Reference na metode 375
Reference na statičke metode. 375
Referenca na metodu instance 376
Reference na metode generičkih tipova 379
Reference na konstruktor. 382
Unapred definisani fukcionalni interfejsi 386
Poglavlje 16 Moduli. 389
Osnove modula 389
Primer jednostavnog modula 390
Kompajlirajte i pokrenite prvi primer modula 394
Podrobniji uvid u naredbe requires i exports 395
Modul java.base i moduli platforme 396
Stariji kôd i bezimeni modul 397
Izvoženje u ciljni modul 398
Naredba requires transitive 399
Upotreba servisa 403
Osnove servisa i dobavljača servisa 403
Rezervisane reči koje se odnose na servise. 404
Primer modularnog servisa. 404
Grafovi modula 410
Tri specijalizovane mogućnosti modula. 411
Otvoreni moduli 411
Naredba opens 412
Naredba requires static 412
Uvod u alatku jlink i JAR datoteke modula. 412
Povezivanje datoteka u eksplodiranom direktorijumu 412
Povezivanje modularnih JAR datoteka. 413
JMOD datoteke 414
Nekoliko reči o slojevima i automatskim modulima. 414
Završne napomene u vezi s modulima. 414

Deo II Javina biblioteka

Poglavlje 17 Rad sa znakovnim nizovima 417
Konstruktori klase String 418
Dužina znakovnog niza. 420

Posebne operacije sa znakovnim nizovima 420
Znakovni literali 420
Nadovezivanje znakovnih nizova 420
Nadovezivanje podataka drugih tipova na znakovne nizove 421
Konverzija u znakovni oblik i metoda toString() 422
Izdvajanje znakova iz znakovnog niza 423
Metoda charAt() 423
Metoda getChars() 423
Poređenje znakovnih nizova 424
Metoda regionMatches() 425
Metode startsWith() i endsWith() 425
Metoda equals() i operator == 426
Metoda compareTo() 426
Pretraživanje znakovnih nizova 428
Menjanje znakovnih nizova 429
Metoda substring() 429
Metoda concat() 430
Metoda replace() 430
Metoda trim() 430
Konvertovanje podataka pomoću metode valueOf() 431
Pretvaranje velikih slova u mala (i obrnuto) 432
Spajanje znakovnih nizova 433
Ostale metode klase String 433
Klasa StringBuffer 435
Konstruktori klase StringBuffer 435
Metode length() i capacity() 435
Metoda ensureCapacity() 436
Metoda setLength() 436
Metode charAt() i setCharAt() 436
Metoda getChars() 437
Metoda append() 437
Metoda insert() 438
Metoda reverse() 438
Metode delete() i deleteCharAt() 439
Metoda replace() 439
Metoda substring() 440
Ostale metode klase StringBuffer 440
Klasa StringBuilder 441
Poglavlje 18 Paket java.lang 443
Omotači prostih tipova 444
Klasa Number 444
Klase Double i Float 444
Klase Byte, Short, Integer i Long 449
Klasa Character 458
Dopune klase Character radi podrške za Unicode kodne tačke 460
Klasa Boolean 462
Klasa Void 463

Klasa Process 463
Klasa Runtime 464
Upravljanje memorijom 465
Pokretanje drugih programa 467
Klasa Runtime.Version 468
Klasa ProcessBuilder 468
Klasa System 471
Merenje vremena izvršavanja programa pomoću metode currentTimeMillis() 473
Upotreba metode arraycopy() 473
Svojstva okruženja 474
Interfejs SystemLogger i klasa SystemLoggerFinder 474
Klasa Object 475
Upotreba metode clone() i interfejsa Cloneable 475
Klasa Class 477
Klasa ClassLoader 480
Klasa Math 480
Trigonometrijske funkcije 480
Eksponencijalne funkcije 481
Funkcije za zaokružavanje brojeva 481
Ostale metode klase Math 483
Klasa StrictMath 485
Klasa Compiler 485
Klase Thread i ThreadGroup i interfejs Runnable 485
Interfejs Runnable 485
Klasa Thread 485
Klasa ThreadGroup 488
Klase ThreadLocal i InheritableThreadLocal 491
Klasa Package 492
Klasa Module 493
Klasa ModuleLayer 494
Klasa RuntimePermission 494
Klasa Throwable 494
Klasa SecurityManager 494
Klasa StackTraceElement 494
Klasa StackWalker i interfejs StackWalker.StackFrame 495
Klasa Enum 495
Klasa ClassValue 496
Interfejs CharSequence 496
Interfejs Comparable 497
Interfejs Appendable 497
Interfejs Iterable 498
Interfejs Readable 498
Interfejs AutoCloseable 498
Interfejs Thread.UncaughtExceptionHandler 499
Potpaketi paketa java.lang 499
java.lang.annotation 499
java.lang.instrument 499

java.lang.invoke 499
java.lang.module 500
java.lang.management 500
java.lang.ref 500
java.lang.reflect 500
Poglavlje 19 Paket java.util, prvi deo: Javine kolekcije 501
Uvod u kolekcije 502
Interfejsi za rad s kolekcijama 503
Interfejs Collection 504
Interfejs List 506
Interfejs Set 508
Interfejs SortedSet 509
Interfejs NavigableSet 510
Interfejs Queue 511
Interfejs Deque 512
Klase za rad s kolekcijama 514
Klasa ArrayList 514
Klasa LinkedList 517
Klasa HashSet 519
Klasa LinkedHashSet 520
Klasa TreeSet 520
Klasa PriorityQueue 522
Klasa ArrayDeque 522
Klasa EnumSet 523
Pristupanje elementima kolekcije pomoću iteratora 524
Upotreba iteratora 526
For-each petlja kao alternativa iteratorima 527
Spliteratori 528
Smeštanje objekata korisničkih klasa u kolekcije 531
Interfejs RandomAccess 532
Rad s mapama 533
Interfejsi za rad s mapama 533
Klase za rad s mapama 540
Komparatori 544
Upotreba komparatora 547
Algoritmi za rad s elementima kolekcija 552
Klasa Arrays 558
Klase i interfejsi iz ranijih verzija Jave 563
Interfejs Enumeration 563
Klasa Vector 564
Klasa Stack 567
Klasa Dictionary 569
Klasa Hashtable 570
Klasa Properties 573
Upotreba metoda store() i load() 576
Zaključak o kolekcijama 578

Poglavlje 20 Paket java.util, drugi deo: ostale Javine alatke 579
Klasa StringTokenizer 579
Klasa BitSet 581
Klase Optional, OptionalDouble, OptionalInt i OptionalLong 584
Klasa Date 586
Klasa Calendar 588
Klasa GregorianCalendar 591
Klasa TimeZone 592
Klasa SimpleTimeZone 593
Klasa Locale 594
Klasa Random 596
Klase Timer i TimerTask 598
Klasa Currency 600
Klasa Formatter 601
Konstruktori klase Formatter 601
Metode klase Formatter 602
Osnovno o formatiranju 603
Formatiranje pojedinačnih znakova i znakovnih nizova 605
Formatiranje brojeva 605
Formatiranje datuma i vremena 606
Specifikatori %n i %% 608
Zadavanje minimalne širine polja 608
Zadavanje preciznosti 610
Upotreba indikatora formata 610
Poravnavanje rezultata 611
Indikatori razmak, +, 0 i (. 611
Indikator zarez 613
Indikator # 613
Zadavanje specifikatora velikim slovima 613
Upotreba indeksa argumenta 614
Zatvaranje instance tipa Formatter 615
Veza s Javinom metodom printf() 615
Klasa Scanner 615
Konstruktori klase Scanner 616
Osnove upotrebe klase Scanner 617
Primeri upotrebe klase Scanner 620
Zadavanje graničnika 624
Ostale metode klase Scanner 625
Klase ResourceBundle, ListResourceBundle i PropertyResourceBundle 626
Ostale pomoćne klase i interfejsi 631
Potpaketi paketa java.util 631
Potpaketi java.util.concurrent, java.util.concurrent.atomic i java.util.concurrent.locks 632
Paket java.util.function 632
Paket java.util.jar 634
Paket java.util.logging 634
Paket java.util.prefs 635

Paket java.util.regex . . . 635
Paket java.util.spi . . . 635
Paket java.util.stream . . . 635
Paket java.util.zip . . . 635
Poglavlje 21 Ulazno-izlazne operacije: paket java.io . . . 637
Javine klase i interfejsi za ulazno-izlazne operacije . . . 638
Klasa File . . . 638
Direktorijumi . . . 641
Interfejs FilenameFilter . . . 642
Alternativno rešenje pomoću metode listFiles() . . . 643
Izrada direktorijuma . . . 643
Interfejsi AutoCloseable, Closeable i Flushable . . . 644
U/I izuzeci . . . 644
Dva načina zatvaranja toka . . . 645
Klase za rad sa tokovima . . . 646
Binarni tokovi . . . 646
Klasa InputStream . . . 647
Klasa OutputStream . . . 648
Klasa FileInputStream . . . 648
Klasa FileOutputStream . . . 650
Klasa ByteArrayInputStream . . . 652
Klasa ByteArrayOutputStream . . . 654
Filtrirani binarni tokovi . . . 655
Baferovani binarni tokovi . . . 655
Klasa SequenceInputStream . . . 659
Klasa PrintStream . . . 660
Klase DataOutputStream i DataInputStream . . . 662
Klasa RandomAccessFile . . . 664
Znakovni tokovi . . . 665
Klasa Reader . . . 665
Klasa Writer . . . 666
Klasa FileReader . . . 667
Klasa FileWriter . . . 667
Klasa CharArrayReader . . . 668
Klasa BufferedReader . . . 670
Klasa BufferedWriter . . . 672
Klasa PushbackReader . . . 672
Klasa PrintWriter . . . 673
Klasa Console . . . 674
Serijalizovanje objekata . . . 676
Interfejs Serializable . . . 676
Interfejs Externalizable . . . 677
Interfejs ObjectOutput . . . 677
Klasa ObjectOutputStream . . . 677
Interfejs ObjectInput . . . 678
Klasa ObjectInputStream . . . 679
Primer serijalizovanja . . . 680
Prednosti tokova . . . 681

Poglavlje 22 Uvod u NIO 683
NIO Klase 683
Osnove sistema NIO 684
Baferi 684
Kanali 686
Skupovi znakova i selektori 687
Poboljšanja NIO 688
Interfejs Path 688
Klasa Files 689
Klasa Paths 692
Interfejsi za atribute datoteka 692
Klase FileSystem, FileSystems i FileStore 694
Upotreba sistema NIO 694
Upotreba NIO sistema za U/I operacije pomoću kanala 695
Upotreba NIO metoda za U/I operacije pomoću tokova 704
Upotreba NIO sistema za operacije s putanjama i sistemom datoteka 706
Poglavlje 23 Umrežavanje 715
Osnove umrežavanja 715
Klase i interfejsi za umrežavanje 716
Klasa InetAddress 717
Fabričke metode 717
Metode instance 718
Klase Inet4Address i Inet6Address 719
Klijentske TCP/IP utičnice 719
Klasa URL 722
Klasa URLConnection 724
Klasa HttpURLConnection 726
Klasa URI 728
Kolačići 728
Serverske TCP/IP utičnice 728
Datagrami 729
Klasa DatagramSocket 729
Klasa DatagramPacket 730
Primer razmene datagrama 731
Poglavlje 24 Obrada događaja 733
Dva mehanizma za obradu događaja 734
Model prosleđivanja događaja 734
Događaji 734
Izvori događaja 734
Prijemnici događaja 735
Klase za događaje 736
Klasa ActionEvent 737
Klasa AdjustmentEvent 738
Klasa ComponentEvent 738
Klasa ContainerEvent 739
Klasa FocusEvent 739

Klasa InputEvent 740
Klasa ItemEvent 741
Klasa KeyEvent 742
Klasa MouseEvent 742
Klasa MouseWheelEvent 744
Klasa TextEvent 745
Klasa WindowEvent 745
Izvori događaja 746
Interfejsi za prijemnike događaja 747
Interfejs ActionListener 747
Interfejs AdjustmentListener 748
Interfejs ComponentListener 748
Interfejs ContainerListener 748
Interfejs FocusListener 748
Interfejs ItemListener 748
Interfejs KeyListener 748
Interfejs MouseListener 749
Interfejs MouseMotionListener 749
Interfejs MouseWheelListener 749
Interfejs TextListener 749
Interfejs WindowFocusListener 749
Interfejs WindowListener 749
Upotreba modela prosleđivanja događaja 750
Nekoliko osnovnih AWT koncepata u vezi s grafičkim korisničkim interfejsom 750
Obrada događaja čiji je izvor miš 751
Obrada događaja čiji je izvor tastatura 754
Adapterske klase 757
Interne klase 759
Anonimne interne klase 761
Poglavlje 25 Uvod u AWT: rad s prozorima, grafikom i tekstom 763
AWT klase 764
Osnovno o prozorima 766
Klasa Component 766
Klasa Container 766
Klasa Panel 766
Klasa Window 767
Klasa Frame 767
Klasa Canvas 767
Rad s prozorima koji nasleđuju klasu Frame 767
Podešavanje veličine prozora 767
Skrivanje i prikazivanje prozora 768
Zadavanje teksta na naslovnoj traci prozora 768
Zatvaranje prozora tipa Frame 768
Metoda paint() 768
Prikazivanje kraćeg teksta u prozoru 768
Zadavanje boja prednjeg plana i pozadine 769

Zahtevanje ažuriranja sadržaja prozora 769
Izrada aplikacije koja radi s prozorima tipa Frame 771
Uvod u grafiku. 771
Crtanje linija 771
Crtanje pravougaonika 772
Crtanje elipsa i krugova. 772
Crtanje lukova 772
Crtanje poligona 773
Primer upotrebe metoda za crtanje grafičkih likova. 773
Promena veličine grafičkih objekata 774
Upotreba boja 776
Metode klase Color 776
Podešavanje tekuće boje grafičkog objekta 777
Program koji ilustruje rad s bojama 777
Podešavanje načina crtanja objekata 778
Rad s fontovima. 780
Utvrđivanje koji su sve fontovi na raspolaganju. 781
Izrada i biranje fonta 782
Pribavljanje podataka o fontu. 784
Doterivanje izgleda teksta pomoću klase FontMetrics 785
Poglavlje 26 AWT kontrole, raspoređivači i meniji 789
Osnovno o AWT kontrolama 790
Postavljanje i uklanjanje kontrola 790
Reagovanje na događaje u kontrolama 790
Izuzetak HeadlessException 791
Natpisi 791
Dugmad 792
Upotreba dugmadi 793
Polja za potvrđivanje 796
Upotreba polja za potvrđivanje. 797
Grupa polja za potvrđivanje 799
Kontrole za biranje ponuđenih opcija. 800
Upotreba padajuće liste 801
Liste 803
Rad sa listama. 804
Trake za pomeranje sadržaja 806
Upotreba traka za pomeranje 807
Polja za tekst 809
Upotreba polja za tekst 810
Višeredna polja za tekst. 811
Raspoređivači sadržaja u kontejneru 813
Raspoređivač FlowLayout 814
Raspoređivač BorderLayout 815
Upotreba umetaka. 816
Raspoređivač GridLayout 818
Raspoređivač CardLayout 819
Raspoređivač GridBagLayout. 822

Linije menija i meniji 827
Okviri za dijalog 831
Preporuke u vezi s redefinisanjem metode paint() 836
Poglavlje 27 Slike 839
Formati datoteka 839
Osnove rada sa slikama: izrada, učitavanje i prikazivanje 840
Izrada objekta tipa Image 840
Učitavanje slike 840
Prikazivanje slike 841
Dvostruko baferovanje 842
Interfejs ImageProducer 845
Klasa MemoryImageSource 845
Interfejs ImageConsumer 847
Klasa PixelGrabber 847
Klasa ImageFilter 849
Klasa CropImageFilter 850
Klasa RGBImageFilter 852
Dodatne klase za slike 862
Poglavlje 28 Alatke za paralelan rad 863
Paketi koji čine API za konkurentno programiranje 864
Paket java.util.concurrent 864
Paket java.util.concurrent.atomic 865
Paket java.util.concurrent.locks 865
Upotreba sinhronizacionih objekata 866
Semafor 866
Klasa CountDownLatch 871
Klasa CyclicBarrier 872
Klasa Exchanger 875
Klasa Phaser 877
Upotreba interfejsa Executor 884
Jednostavan primer upotrebe izvršioca 884
Upotreba interfejsa Callable i Future 886
Nabrajanje TimeUnit 888
Kolekcijske klase koje podržavaju paralelan rad 890
Blokade 890
Jedinične operacije 893
Paralelno programiranje pomoću Fork/Join Frameworka 894
Glavne klase Fork/Join Frameworka 895
Klasa ForkJoinTask<V> 895
Strategija “podeli pa vladaj” 898
Jednostavan prvi primer upotrebe klasa Fork/Join Frameworka 899
Uticaj nivoa paralelizma 902
Primer upotrebe klase RecursiveTask<V> 905
Asinhrono izvršavanje poslova 907
Prekidanje pokrenutog posla 907
Utvrđivanje tekućeg statusa pokrenutog posla 908
Ponovno pokretanje postojećeg posla 908

Dalje istraživanje 908
Neke dodatne mogućnosti koje pruža klasa ForkJoinPool 909
Nekoliko saveta o upotrebi elemenata Fork/Join Frameworka 910
Alatke za paralelan rad u poređenju s Javinim tradicionalnim pristupom 910
Poglavlje 29 API za rad s tokovima podataka 911
Osnove tokova 911
Interfejsi za tokove 912
Kako formirati tok 915
Jedostavan primer upotrebe toka 915
Reduktivne operacije 918
Upotreba paralelnih tokova 921
Preslikavanje elemenata jednog toka u drugi 923
Preslikavanje tokova u kolekcije 927
Iteratori i tokovi 930
Kombinovanje iteratora s tokom 930
Upotreba spliteratora 932
Dalje istraživanje API-ja za rad s tokovima 934
Poglavlje 30 Regularni izrazi i drugi paketi 935
Obrada regularnih izraza 935
Klasa Pattern 936
Klasa Matcher 936
Sintaksa regularnih izraza 937
Primer pronalaženja šablona 938
Dve opcije za pronalaženje poklapanja sa šablonom 943
Istražite mogućnosti regularnih izraza 943
Refleksija 943
Daljinsko pozivanje metoda (RMI) 947
Upotreba RMI u jednostavnoj klijent/server aplikaciji 947
Formatiranje datuma i vremena pomoću paketa java.text 951
Klasa DateFormat 951
Klasa SimpleDateFormat 953
API za rad s datumima i vremenima u paketu java.time 954
Osnove obrade datimu i vremena 955
Formatiranje datuma i vremena 956
Raščlanjivanje znakovnih vrednosti koje predstavljaju datume i vremena 959
Šta bi još trebalo istražiti u paketu java.time 960

Deo III Uvod u GKI programiranje pomoću Swinga

Poglavlje 31 Uvod u Swing 963
Koreni Swinga 963
Swing je sagrađen na temeljima AWT-a 964
Dve ključne odlike Swinga 964
Swingove komponente su lagane 964
Swing podržava zamenjiv izgled i osećaj 964
Veza sa arhitekturom MVC 965

Komponente i kontejneri 966
Komponente 966
Kontejneri 967
Okna kontejnera najvišeg nivoa 967
Paketi koji čine Swing 968
Jednostavna swing aplikacija 968
Obrada događaja 972
Iscrtavanje Swing komponente 975
Osnove iscrtavanja sadržaja 976
Izračunavanje veličine oblasti za iscrtavanje 977
Primer iscrtavanja sadržaja 977
Poglavlje 32 Upoznavanje Swinga 981
Natpis (klasa JLabel) i ikonica (klasa ImageIcon) 981
Polje za tekst (klasa JTextField) 983
Swing dugmad 985
Klasa JButton 985
Klasa JToggleButton 987
Polja za potvrđivanje 989
Radio-dugmad 991
Grupa kartica s jezičcima 993
Okno za pomeranje sadržaja 995
Liste 997
Kombinovano polje 1000
Hijerarhijska stabla 1002
Tabele 1005
Poglavlje 33 Uvod u Swingove menije 1009
Osnove menija 1009
Pregled klasa JMenuBar, JMenu i JMenuItem 1011
Klasa JMenuBar 1011
Klasa JMenu 1012
Klasa JMenuItem 1013
Izrada glavnog menija 1013
Dodavanje mnemonika i ubrzivača stavkama menija 1018
Dodavanje sličica i kratkih opisa stavkama menija 1020
Stavke menija s radio-dugmadima i poljima za potvrđivanje 1021
Izrada iskačućeg menija 1023
Izrada palete alatki 1026
Upotreba akcija 1028
Sastavljanje programa PrimerMenija u jednu celinu 1033
Dalje istraživanje Swinga 1040

Deo IV Uvod u GKI programiranje pomoću JavaFXa

Poglavlje 34 Uvod u GKI programiranje pomoću JavaFXa 1043
Osnovni koncepti u JavaFXu 1044
Paketi koji čine JavaFX 1044
Klase za pozornicu i scenu 1044
Čvorovi i grafovi scene 1045

Raspoređivanje sadržaja scene 1045
Klasa Application i metode životnog ciklusa aplikacije 1045
Pokretanje JavaFX aplikacije 1046
Osnovna struktura jedne JavaFX aplikacije 1046
Kompajliranje i izvršavanje JavaFX programa 1049
Nit u kojoj se izvršava aplikacija 1050
Jednostavna JavaFX kontrola: natpis 1050
Upotreba dugmadi i događaja 1052
Osnove JavaFX događaja 1052
Uvod u kontrolu tipa Button (dugme) 1053
Primer obrade događaja i upotrebe dugmeta 1054
Crtanje direktno na platno 1056
Poglavlje 35 JavaFX kontrole 1061
Upotreba klasa Image i ImageView 1061
Postavljanje slike na natpis 1063
Postavljanje slike na dugme 1066
Klasa ToggleButton (preklopno dugme) 1069
Klasa RadioButton (radio-dugme) 1071
Obrada događaja promene stanja grupe radio-dugmadi 1074
Alternativni način obrade radio-dugmadi 1075
Klasa CheckBox (polje za potvrđivanje) 1077
Klasa ListView (prikaz u obliku liste) 1081
Trake za pomeranje sadržaja liste ListView 1084
Biranje više stavki 1085
Klasa ComboBox (padajuća lista) 1086
Klasa TextField (polje za tekst) 1089
Klasa ScrollPane (okno za pomeranje sadržaja) 1091
Klasa TreeView (hijerarhijsko stablo) 1094
Efekti i transformacije 1099
Efekti 1099
Transformacije 1100
Primer upotrebe efekata i transformacija 1101
Dodavanje kratkih opisa 1104
Isključivanje kontrole 1105
Poglavlje 36 Uvod u JavaFX menije 1107
Osnove menija 1107
Pregled klasa MenuBar, Menu i MenuItem 1109
Klasa MenuBar 1109
Klasa Menu 1110
Klasa MenuItem 1110
Izrada glavnog menija 1111
Dodavanje mnemonika i ubrzivača stavkama menija 1116
Postavljanje slika na stavke menija 1118
Stavke menija s radio-dugmadima i poljima za potvrđivanje 1118
Izrada kontekstnog menija 1121
Izrada palete alatki 1124

Sastavljanje programa PrimerMenija u jednu celinu 1126
Dalje istraživanje JavaFXa 1132

Deo V Primena Jave

Poglavlje 37 Zrna Jave 1135
Šta je zrno Jave? 1135
Prednosti tehnologije zrna Jave 1136
Introspekcija 1136
Šabloni metoda za rad sa svojstvima zrna 1136
Šabloni metoda za obradu događaja zrna 1138
Šabloni za projektovanje metoda i naredbi 1138
Upotreba interfejsa BeanInfo 1138
Vezana i ograničena svojstva 1139
Čuvanje stanja 1139
Alatke za prilagođavanje zrna 1139
API zrna Jave 1140
Klasa Introspector 1142
Klasa PropertyDescriptor 1142
Klasa EventSetDescriptor 1142
Klasa MethodDescriptor 1142
Primer izrade zrna Jave 1142
Poglavlje 38 Servleti 1147
Osnove 1147
Životni ciklus servleta 1148
Mogućnosti za razvijanje servleta 1148
Upotreba Tomcata 1149
Jednostavan servlet 1150
Pisanje i kompajliranje izvornog koda servleta 1151
Pokretanje Tomcata 1151
Pokretanje čitača veba i izvršavanje servleta 1151
API Servlet 1152
Paket javax.servlet 1152
Interfejs Servlet 1153
Interfejs ServletConfig 1153
Interfejs ServletContext 1153
Interfejs ServletRequest 1154
Interfejs ServletResponse 1155
Klasa GenericServlet 1155
Klasa ServletInputStream 1155
Klasa ServletOutputStream 1156
Klase za izuzetke u servletu 1156
Učitavanje parametara koji su prosleđeni servletu 1156
Paket javax.servlet.http 1157
Interfejs HttpServletRequest 1158
Interfejs HttpServletResponse 1159
Interfejs HttpSession 1159
Klasa Cookie 1160
Klasa HttpServlet 1161

Obrada HTTP zahteva i odgovora 1162
Obrada HTTP zahteva tipa GET. 1162
Obrada HTTP zahteva tipa POST 1164
Upotreba kolačića 1165
Čuvanje tekućeg stanja HTTP sesije 1167

Deo VI Dodaci

Dodatak A Dokumentacioni komentari u Javi 1171
Markeri za alatku javadoc 1171
@author 1172
{@code} 1172
@deprecated 1173
{@docRoot} 1173
@exception. 1173
@hidden 1173
{@index} 1173
{@inheritDoc} 1173
{@link} 1174
{@linkplain} 1174
{@literal} 1174
@param 1174
@provides 1174
@return 1174
@see 1174
@serial 1175
@serialData. 1175
@serialField 1175
@since. 1175
@throws 1175
@uses 1175
{@value} 1176
@version 1176
Opšti oblik dokumentacionog komentara 1176
Rezultat programa javadoc 1176
Primer upotrebe dokumentacionih komentara 1176

Dodatak B Uvod u Java Web Start 1179
Šta je Java Web Start? 1179
Četiri ključna elementa Java Web Starta 1180
Java Web Start aplikacije zahtevaju JAR datoteku 1180
Java Web Start aplikacije su potpisane 1180
Java Web Start zahteva JNLP datoteku 1181
Veza koja upućuje na Java Web Start JNLP datoteku 1182
Eksperimentisanje s Java Web Start Using i lokalnim sistemom datoteka 1183
Izrada JAR datoteke za aplikaciju ToggleButtonDemo 1184
Izrada skladišta ključeva i potpisivanje datoteke ToggleButtonDemo.jar 1184

Izrada JNLP datoteke za aplikaciju ToggleButtonDemo. 1185
Izrada kratke HTML datoteke čije je ime StartTBD.html. 1186
Dodavanje datoteke ToggleButtonDemo.jnlp na listu Exception Site List u Java Control Panelu 1186
Pokretanje aplikacije ToggleButtonDemo iz čitača veba 1187
Pokretanje Java Web Start aplikacije pomoću alatke javaws. 1187
Upotreba Java Web Starta za aplete 1187

Dodatak C Uvod u JShell 1189
Osnove JShella. 1189
Pregledanje, ispravljanje i ponovno izvršavanje koda. 1191
Dodavanje metode 1192
Definisanje klase 1193
Upotreba interfejsa. 1194
Izračunavanje vrednosti izraza i upotreba ugrađenih promenljivih . . . 1195
Uvoženje paketa 1196
Izuzeci 1196
Još neke JShellove komande. 1196
Dalje proučavanje JShella. 1197

Dodatak D Osnove apleta 1199
Dve vrste apleta 1199
Osnove apleta 1200
Klasa Applet. 1201
Arhitektura apleta 1201
Skelet apleta 1201
Inicijalizovanje i završavanje apleta 1203
Swing apleti 1204

Spsiak termina korišćenih u knjizi 1263
Indeks 1263

Predgovor

Java je jedan najvažnijih i najviše korišćenih računarskh jezika, a tu odliku ima već dugi niz godina. Za razliku od nekih drugih računarskih jezika čiji uticaj je tokom vremena primetno opao, Java postaje sve jača. Java već od svog prvog izdanja izbila na čelo internet programiranja. Svaka naredna verzija je samo učvrstila na tom položaju. Danas je to i dalje prvi i najbolji izbor za razvijanje veb aplikacija. To je takođe moćni programskijezik opšte namene, pogodan za široku lepezu raznih namena. Jednostavno rečeno: veliki deo savremenog sveta radi na Java koda. Java je zaista toliko važna!

Ključni razlog uspeha Jave jeste njena prilagodljivost. Od svog početnog izdanja 1.0, Java se neprestano prilagođava promenama u programskom okruženju i promenama načina na koji programeri programiraju. Što je najvažnije, Java nije samo pratila trendove, nego ih je sama pokretala. Javina sposobnost prilagođavanja brzim promenama u svetu računarstva najjači je razlog zbog kojeg je Java bila i nastavlja da bude tako uspešna.

Nakon prvog izdanja 1996. godine, ova knjiga je imala i više drugih izdanja, gde je svako održavalo tekuće stanje evolucije Jave. Ovo je deseto izdanje, koje je bilo ažurirano za Java SE 9 (JDK 9). Zbog toga ovo izdanje knjige sadrži značajnu koiličinu novog gradiva zato što Java SE 9 uvodi u jezik Javu više novih mogućnosti. Najvažnija među njima su moduli, koji omogućavaju da zadate međusobne odnose i zavisnosti između delova koda koji čine jednu aplikaciju. Moduli mogu takođe da utiču na dostupnost pojedinih elemenata koda. Uvođenje modula predstavlja jednu od najznačajnijih promena jezika Java, kojem dodaju i nov element sintakse i deset novih rezervisanih reči. Moduli takođe značajno utiču na Java API biblioteku zato što su njeni paketi sada organizovai u module. Osim toga, zbog podrške za module, jeziku su dodate nove alatke, postojeće alatke su dopunjene i definisan je format datoteka. Zbog važnosti modula, posvećeno im je celo poglavlje 16.

Osim modula, JDK 9 donosi i više drugih mogućnosti. Jedna od najzanimljivijih jeste JShell. JShell je alatka koja pruža interaktivno okruženje u kojem možete lako eksperimentirati s odlomcima koda bez obaveze da napišete kompletan program. I početnici i iskusni profesionalci će otkriti da je JShell veoma koristan. Uvod u JShell naći ćete u dodatku C. Isto kao i prethodne verzije, JDK 9 sadrži veliki broj manjih ispravki i poboljšanja i samog jezika Java i njegovih API biblioteka. Zbog toga ćete u celoj knjizi nailaziti na ažurirano gradivo. Još jedna završna napomena: Java SE 9 čini zastarelim aplete i API za apleta. Iz tog razloga, više nisu razmatrani u glavnom delu knjige. Međutim, kratak uvod u aplete naći ćete u dodatku D.

Knjiga za sve programere

Ova knjiga je namenjena svim programerima, bez obzira na to da li ste početnik ili iskusan profesionalac. Početniku će biti posebno korisni mnogobrojni primeri i brižljivo osmišljena objašnjenja. Detaljno razmatranje Javinih naprednijih mogućnosti i bibloteka više će zanimati profesionalce. Za obe kategorije, knjiga će biti dugotrajan izvor informacija i praktična referenca.

Šta ćete naći u knjizi

Ova knjiga je detaljan vodič kroz jezik Java, gde opisuje njegovu sintaksu, rezervisane reči i osnovne principe programiranja. Razmatrani su i značajni delovi Javine API biblioteke. Knjiga je podeljena na pet delova, gde svaki obrađuje različit aspekt Javinog okruženja za programiranje.

Deo I predstavlja detaljan uvod u jezik Java. Počinje od osnova i razmatra elemente jezika kao što su tipovi podataka, operatori, upravljačke naredbe i klase. Zatim prelazi na nasleđivanje, pakete, interfejse, obradu izuzetaka i višenitni rad. Dalje, opisuje anotacije, nabrajanja, automatsko pakovanje i raspakivanje podataka, generičke klase i lambda izraze. Opisane su i U/I operacije. Završno poglavlje prvog dela obrađuje module. Kao što je već ranije pomenuto, moduli su najznačajnija nova mogućnost koju uvodi verzija Java SE 9.

Deo II razmatra ključne aspekte Javine standardne API biblioteke. Među obrađenim temama naći ćete rad sa znakovnim nizovima, U/I operacije, umrežavanje, standardne alatke, Collections Framework, AWT, obrada događaja, rad sa slikama, istovremenost (uključujući i Fork/Join Framework), regularni izrazi i biblioteka za rad s podacima pomoću tokova.

Deo III se sastoji od tri poglavlja koja opisuju Swing.

Deo IV se sastoji od tri poglavja koja predtavljaju JavaFX.

Deo V sadrži dva poglavja koja prikazuju primere upotrebe Jave. Prvo obrađuje zrna Jave (Java Beans). Drugo predstavlja uvod u servlete.

Zahvalnica

Želim da se posbeno zahvalim Patriku Notonu (Patrick Naughton), Džou O'Nilu (Joe O'Neil) i Deniju Kauardu (Danny Coward).

Patrik Noton je jedan tvoraca jezika Java. Pomogao mi je i pri pisanju prvog izdanja ove knjige. Na primer, između njegovih mnogih drugih doprinosa, Patrik je obezbedio materijal i za prvobitne verzije poglavlja 21, 23 i 27. Njegova pronicljivost, stručnost i energija znatno su doprinele uspehu knjige.

Tokom pripremanja drugog i trećeg izdanja ove knjige, Džo O'Nil je obezbedio početne verzije gradiva koje se sada načazi u poglavljima 30, 32, 37 i 38 ovog izdanja. Džo mi je pomogao pri pisanju više mojih knjiga, a njegov doprinos je uvek bio vrhunski.

Deni Kauard je tehnički urednik ovog izdanja knjige. Dani je sarađivao na više mojih knjiga, a njegovi saveti, oštroumnost i predlozi uvek su dragoceni i veoma cenjeni.

HERBERT ŠILT

Za dalje učenje

Java: Vodič za programere je vaša ulaznica u seriju knjiga o Java programiranju Herberta Šilta. Ovo su druge njegove knjige koje će vas možda zanimati:

Herb Schildt's Java Programming Cookbook

Java: A Beginner's Guide

Introducing JavaFX 8 Programming

Swing: A Beginner's Guide

The Art of Java

DEO

I

Jezik Java

POGLAVLJE 1
Istorija i evolucija Jave

POGLAVLJE 2
Pregled jezika Java

POGLAVLJE 3
Tipovi podataka, promenljive i nizovi

POGLAVLJE 4
Operatori

POGLAVLJE 5
Upravljačke naredbe

POGLAVLJE 6
Uvod u klase

POGLAVLJE 7
Metode i klase izbliza

POGLAVLJE 8
Nasleđivanje

POGLAVLJE 9
Paketi i interfejsi

POGLAVLJE 10
Obrada izuzetaka

POGLAVLJE 11
Višenitno programiranje

POGLAVLJE 12
Nabrajanja, automatsko pakovanje i anotacije

POGLAVLJE 13
Ulazno-izlazne operacije, naredba Try-with-Resources i druge teme

POGLAVLJE 14
Generički tipovi

POGLAVLJE 15
Lambda izrazi

POGLAVLJE 16
Moduli

POGLAVLJE

1 Istorija i evolucija Jave

Ako želite da potpuno razumete Javu, morate shvatiti razloge koji su uticali na njeno stvaranje, sile koje su je uobličile i elemente koje je dobila u nasleđe. Slično uspešnim programskim jezicima koji su joj prethodili, i Java je mešavina najboljih elemenata svog bogatog nasleđa, kombinovanih s novim koncepcijama neophodnim za njenu jedinstvenu misiju. Dok ćemo se u narednim poglavljima ove knjige baviti praktičnim aspektima Jave – njenom sintaksom, bibliotekama i primenama – u ovom poglavlju ćete saznati kako i zašto je Java nastala, šta je čini tako važnom i kako se menjala tokom godina.

Iako je postala nerazdvojivi deo mrežnog okruženja interneta, treba istaći da je Java na prvom mestu programski jezik. Programski jezici se usavršavaju i razvijaju iz dva osnovna razloga:

- da bi se prilagodili izmenjenom okruženju i novoj primeni
- da bi se u njih ugradila poboljšanja i novosti iz oblasti programiranja.

Kao što ćete se i sami uveriti, na razvoj Jave uticala su oba ova elementa, u približno jednakoj meri.

Poreklo Jave

Java je srodnik jezika C++, koji je direktan potomak jezika C. Veći deo svojih osobina Java je nasledila od ova dva jezika. Iz jezika C Java je preuzela sintaksu. Mnoge objektno orijentisane osobine Jave nastale su pod uticajem jezika C++. U stvari, više karakteristika koje definišu Javu potiču od njenih prethodnika ili su nastale kao odgovor na njih. Štaviše, nastanak Jave ima duboke korene u procesima poboljšavanja i prilagođavanja programskih jezika tokom poslednjih nekoliko decenija. Iz pomenutih razloga, u ovom odeljku ćemo prikazati redosled događaja i uticaja koji su doveli do nastanka Jave. Kao što ćete videti, svaka inovacija programskog jezika nastaje iz potrebe da se reši određeni temeljni problem koji nije bio rešiv u ranijim programskim jezicima. U tom pogledu ni Java nije izuzetak.

Rađanje savremenog programiranja: C

Pojava jezika C uzdrmala je računarski svet. Njegov uticaj ne treba potcenjivati, pošto je on iz osnova izmenio pristup programiranju i programerski način razmišljanja. Pojava jezika C direktan je rezultat potrebe za strukturiranim, efikasnim jezikom visokog nivoa koji može da zameni asemblerski kôd za pisanje sistemskih programa. Kao što možda i sami znate, kada se projektuje računarski jezik, često se prave kompromisi slični sledećim:

- lakoća upotrebe naspram moći jezika
- bezbednost naspram efikasnosti
- strogost primene pravila naspram proširivosti.

Pre C-a, programeri su obično morali da biraju jezik u kome je jedan skup osobina bio favorizovan u odnosu na drugi. Na primer, iako se FORTRAN može koristiti za pisanje prilično efikasnih programa za primenu u nauci, on nije najpogodniji za pisanje sistemskih programa. Zatim, premda se BASIC lako uči, on nije previše moćan, a nedostatak strukturnih elemenata čini ga nepogodnim za pisanje dugačkih programa. Asemblerom se mogu napraviti veoma efikasni programi, ali se on teško uči i teško efikasno koristi. Štaviše, otklanjanje grešaka u asemblerskom kodu može da bude veoma komplikovano.

Još je gore to što prvi računarski jezici, kao što su BASIC, COBOL i FORTRAN, nisu projektovani u skladu s principima strukturiranog programiranja. Umesto njih, oslanjali su se na naredbu GOTO kao na glavnu programsku kontrolu. Zbog toga su se programi pisani na ovim jezicima lako preobražavali u tzv. „špageti kôd„ – masu prepletenih skokova i uslovnih grananja koje gotovo niko nije mogao da razume. Iako su jezici kao što je Pascal strukturirani, njihova efikasnost nije bila osnovni cilj, a nedostajale su im i određene osobine koje bi ih učinile primenljivim za širok opseg programa. (Konkretno, ako uzmemo u obzir standardne dijalekte Pascala koji su u to vreme postojali, Pascal nije dolazio u obzir za pisanje sistemskih programa.)

Dakle, pre nastanka C-a, nijedan jezik nije uspeo da razreši suprotstavljene principe koji su osujećivali ranije napore. Pa ipak, potreba za takvim jezikom bila je urgentna. Početkom sedamdesetih godina prošlog veka, kada je računarska revolucija počela da se zahuktava, potražnja za softverom uveliko je prevazilazila mogućnosti programera da ga isporuče. U akademskim krugovima žustro su se lomila koplja u pokušajima da se stvori bolji programski jezik. Međutim, a to je možda i važnije, počeo je da se oseća i drugi uticaj. Računarski hardver se toliko raširio da je dosegao „kritičnu masu". Računari više nisu čuvani iza strogo zaključanih vrata. Po prvi put su programeri dobili naizgled neograničen pristup svojim mašinama. To je donelo slobodu u eksperimentisanju i istovremeno omogućilo programerima da počnu da prave sopstvene alatke. U predvečerje rađanja C-a, računarska pozornica je bila spremna za kvalitativan skok u oblasti programskih jezika.

Jezik C, koji je stvorio Denis Riči (Dennis Ritchie) i ugradio ga na mašinu DEC PDP-11 koja je radila pod UNIX-om, bio je rezultat razvojnog procesa započetog na starijem jeziku, BCPL, koji je razvio Martin Ričards (Martin Richards). BCPL je uticao na jezik, nazvan B, koji je stvorio Ken Tompson (Ken Thompson), a ovaj je zatim sedamdesetih godina doveo do razvoja jezika C. Tokom više godina, de facto standard jezika C bio je onaj koji se koristio na operativnom sistemu UNIX, a koji su Brajan Kernigan (Brian Kernighan) i Denis Riči opisali u knjizi *The C Programming Language* (Prentice-Hall, 1978). Jezik C je i formalno standardizovan decembra 1989, kada je Američki nacionalni institut za standardizaciju (American National Standards Institute, ANSI) usvojio standard za jezik C.

Mnogi smatraju da nastanak jezika C označava početak savremenog doba u računarskim jezicima. On je uspešno pomirio sukobljene principe koji su toliko mučili njegove prethodnike.

Tako je nastao moćan, efikasan, strukturiran jezik koji se srazmerno lako uči. On ima i drugi, gotovo neprimetan aspekt: C je programerski jezik. Pre pojave C-a, računarski jezici obično su nastajali kao rezultat akademskih eksperimenata ili pod uticajem birokratskih komiteta. C je u tom pogledu drugačiji. Njega su zamislili, izgradili i razvili pravi programeri, i zbog toga on odražava njihov pristup programiranju. C su izbrusili, isprobali i o njemu neprestano razmišljali ljudi koji su ga stvarno i koristili. Rezultat je bio jezik koji se programerima sviđao. Zaista, C je ubrzo privukao mnoge sledbenike koji su za njim žudeli s gotovo verskim zanosom. Zbog toga je on među programerima veoma brzo i sigurno našao svoje mesto. Ukratko, C je jezik koji su programeri stvorili za programere. Kao što ćete kasnije videti, Java je nasledila tu osobinu.

C++: sledeći korak

Krajem sedamdesetih i početkom osamdesetih godina dvadesetog veka, C je postao dominantan programski jezik, a i danas se masovno koristi. Pošto je C bio uspešan i upotrebljiv programski jezik, možda se pitate zašto je postojala potreba i za nečim drugim. Odgovor je u reči složenost. Kroz čitavu istoriju programiranja, sve složeniji programi uvek su zahtevali sve bolje načine za rešavanje te složenosti. C++ je bio odgovor na takve zahteve. Da biste bolje razumeli zašto je hvatanje u koštac sa složenošću programa bilo osnovni razlog za nastanak jezika C++, razmotrite tekst koji sledi.

Pristup programiranju se dramatično izmenio od onog doba kada su se pojavili računari. Recimo, na prvim računarima su se sve binarne mašinske instrukcije morale ručno uneti sa konzole. To je moglo da se radi sve dok programski kôd nije bio duži od nekoliko stotina instrukcija. Kako su programi rasli, ovo je postalo nepraktično, pa je smišljen asembler, jezik koji je programerima omogućio da rade na većim, složenijim programima koristeći simboličke predstave mašinskih instrukcija. Kako su programi i dalje rasli, smišljani su jezici visokog nivoa koji su programerima stavljali na raspolaganje sve više alatki neophodnih za borbu sa složenošću programa.

Prvi široko popularan jezik bio je, naravno, FORTRAN. Mada je to bio značajan korak u programiranju, teško da je podsticao pisanje jasnih i lako razumljivih programa. Šezdesete godine su iznedrile *strukturirano programiranje*. Slavu te metode proneli su jezici kao što je C. Strukturirani jezici su po prvi put omogućili programerima da prilično lako pišu umereno složene programe. Međutim, čak i uz metode strukturiranog programiranja, kada projekat dostigne određenu veličinu, njegova složenost u jednom trenutku prevaziđe mogućnosti programera. Početkom osamdesetih godina, u mnogim projektima princip strukturiranog programiranja nasilno je primenjivan preko svojih mogućnosti. Da bi se ovaj problem prevazišao, smišljen je nov način programiranja, tzv. *objektno orijentisano programiranje (OOP)*. O objektno orijentisanom programiranju detaljnije ćemo govoriti kasnije u knjizi, pa se sada zadovoljimo samo kratkom definicijom: OOP je metodologija programiranja koja pomaže pri organizovanju složenih programa koristeći nasleđivanje, kapsuliranje i polimorfizam.

Sve u svemu, iako je C jedan od najboljih svetskih programskih jezika, ni on nije bez ograničenja ako treba savladati složene situacije. Kada program premaši izvesnu veličinu, postaje tako složen da je teško pojmiti njegovu celinu. Mada je teško odrediti tačnu veličinu kada se to događa jer to zavisi i od prirode programa i od programera, uvek je postojala granica iznad koje je postajalo nemoguće upravljati programom. Nove mogućnosti koje je doneo jezik C++ učinile da ta prepreka nestane, a programeri bolje shvataju složenije programe i lakše njima upravljaju.

C++ je 1979. stvorio Bjarne Stroustrup dok je radio u kompaniji Bell Laboratories u Mari Hilu, Nju Džerzi. Stroustrup je novi jezik prvobitno nazvao „C sa klasama". Godine 1983. ime mu je promenjeno u C++. (U jeziku C, „++" označava operator uvećanja. Tako bi C++ znači-

lo „poboljšani C". – prim. rec.) C++ proširuje jezik C dodajući mu objektno orijentisane osobine. Pošto je C++ izgrađen na temelju jezika C, on obuhvata sve njegove osobine, svojstva i prednosti. To je osnovni razlog uspeha jezika C++, koji nije načinjen s namerom da se stvori potpuno nov programski jezik, već da se poboljša jedan koji se već pokazao veoma uspešnim.

Sve je spremno za pojavu Jave

Krajem osamdesetih i početkom devedesetih godina, objektno programiranje na jeziku C++ uzelo je maha. I zaista, za trenutak je izgledalo da su programeri konačno pronašli savršen programski jezik. Pošto C++ objedinjuje visoku efikasnost i stilske elemente C-a sa objektno orijentisanim pristupom, mogao se koristiti za pravljenje široke lepeze programa. Međutim, baš kao toliko puta u prošlosti, ponovo je narasla potreba da se računarski jezici pomaknu za još jedan stepenik u svojoj evoluciji. Za nekoliko godina, internet i veb poprimiće svoj sadašnji oblik, a ovaj događaj će začeti novu revoluciju na polju programiranja.

Nastanak Jave

Javu su prvobitno, 1991. godine, koncipirali Džejms Gosling (James Gosling), Patrik Noton (Patrick Naughton), Kris Vort (Chris Warth), Ed Frenk (Ed Frank) i Majk Šeridan (Mike Sheridan), iz korporacije Sun Microsystems, Inc. Trebalo je 18 meseci rada da bi se došlo do prve radne verzije. Ovaj jezik je prvobitno dobio ime „Oak", ali je ono 1995. godine promenjeno u „Java". U periodu od prve realizacije Oaka s jeseni 1992. i zvaničnog najavljivanja Jave u proleće 1995, još mnogo osoba je doprinelo uobličavanju i razvijanju ovog jezika. Bil Džoj (Bill Joy), Artur van Hof (Arthur van Hoff), Džonatan Pejn (Jonathan Payne), Frenk Jelin (Frank Yellin) i Tim Lindholm bili su glavni saradnici na usavršavanju prvobitnog prototipa.

Donekle iznenađuje to što glavni pokretač za razvoj Jave nije bio internet! Osnovni motiv je bila potreba za jezikom nezavisnim od računarske platforme (tj. neutralnim prema arhitekturi), koji bi se mogao koristiti za pisanje softvera namenjenog različitim elektronskim uređajima u domaćinstvu, kao što su mikrotalasne pećnice i daljinski upravljači. Možda i sami pogađate da se kao kontroleri takvih uređaja koriste različiti tipovi procesora. Problem sa jezicima C i C++ (kao i s većinom drugih jezika), leži u tome što se pri prevođenju u binarni oblik oni moraju usmeriti na određeni procesor. Iako se program pisan na jeziku C++ može prevesti za bilo koji procesor, za to je potreban potpun prevodilac (kompajler) namenjen konkretnom ciljnom procesoru. Teškoća je to što su kompajleri skupi i sporo se prave. Očigledno je bilo potrebno lakše i jeftinije rešenje. U pokušaju da dođu do njega, Gosling i ostali počeli su da rade na prenosivom jeziku koji ne zavisi od platforme i čiji bi kôd mogao da se izvršava na različitim procesorima i u različitim okruženjima. Ti napori su konačno doveli do rađanja Jave.

Otprilike u vreme kada su razrađivani detalji novog programskog jezika, pojavio se drugi, bez sumnje značajniji uticaj koji je odigrao ključnu ulogu u budućnosti Jave. Taj činilac bio je, naravno, World Wide Web. Da se veb nije uobličio približno u isto vreme kada i Java, taj programski jezik – upotrebljiv, ali složen – možda bi se koristio samo za programiranje kućne elektronike. Međutim, pojavom veba, Java se probila u prvi plan računarskih jezika, jer su i za veb bili potrebni prenosivi programi.

Većina programera vrlo rano nauči da su prenosivi programi, iako poželjni, prilično neostvarljivi. Iako je potraga za efikasnim, prenosivim (nezavisnim od platforme) programima stara gotovo koliko i samo programiranje, preči problemi su je potisnuli u drugi plan. Štaviše, pošto se u to vreme većina korisnika računara svrstavala u tri konkurentska tabora: Intel, Macintosh i UNIX, najveći broj programera se zadržao unutar svojih utvrđenih granica i potreba za preno-

sivim kodom izgubila je svoj primarni značaj. Međutim, sa razvojem interneta i veba, stari problem prenosivosti ponovo se pojavio u punoj snazi. Na kraju krajeva, internet predstavlja raznolik, distribuiran univerzum ispunjen mnogim vrstama računara, operativnih sistema i procesora. Iako su na Internet priključene mnoge vrste platformi, korisnici bi želeli da na svima može da se izvršava isti program. Ono što je jednom bilo neugodan, ali malo značajan problem, postalo je problem koji je pod hitno trebalo rešiti.

Godine 1993, članovima tima za razvoj Jave postalo je jasno da se problem prenosivosti, s kojim su se često sretali pri programiranju koda za kontrolere elektronskih uređaja, pojavljuje i pri pisanju koda za internet. U stvari, isti problem zbog koga je Java prvobitno smišljena, pojavio se i na internetu, samo u velikom obimu. Zbog toga je fokus primene Jave premešten s kućne elektronike na programiranje za internet. Dakle, premda je želja za dobijanjem programskog jezika koji ne zavisi od platforme bila početni motiv, za uspeh Jave na visokom nivou zaslužniji je internet.

Pomenuli smo ranije da Java većinu svojih osobina duguje jezicima C i C++. To nije slučajno. Autori Jave su dobro znali da će poznata sintaksa jezika C i preslikavanje objektno orijentisanih osobina jezika C++ privući mnoge iskusne C/C++ programere. Osim očiglednih sličnosti, Javi i jezicima C i C++ zajedničke su i one osobine koje su ova dva jezika učinile uspešnim. Kao prvo, Javu su oblikovali, isprobavali i poboljšavali profesionalni programeri. To je jezik utemeljen na potrebama i iskustvu osoba koje su ga stvorile. Zbog toga je Java i jezik programera. Drugo, Java je harmoničan i logički dosledan jezik. Treće, izuzev ograničenja koje postavlja internet, Java omogućuje programeru potpunu kontrolu. Ako programirate na pravi način, to će se pokazati u vašem programu. A pokazaće se i ako ste loš programer. Ukratko, Java nije jezik koji štiti nevešte početnike, nego jezik za profesionalne programere.

Zbog sličnosti između Jave i jezika C++, nameće se misao da je Java samo „C++ u verziji za internet". Grdno ćete pogrešiti ako prihvatite tu ideju. Java se od jezika C++ znatno razlikuje – i praktično i po filozofiji. Mada je tačno da je Java nastala pod uticajem jezika C++, ona nije njegova poboljšana verzija. Na primer, Java nije ni uzlazno ni silazno kompatibilna sa jezikom C++. Naravno, sličnosti su znatne i, ako ste programirali na jeziku C++, osećate se kao „kod kuće" i s Javom. Još nešto: Java nije napravljena da zameni C++ već da reši određen skup problema. C++ je stvoren da reši drugačiji skup problema. Oba jezika će uporedo postojati još mnogo godina.

Kao što smo pomenuli na početku poglavlja, programski jezici se razvijaju iz dva razloga: da bi se prilagodili promenama okruženja i da bi prihvatili novine u oblasti programiranja. Promena okruženja koja je izazvala pojavu Jave bila je potreba za programima koji rade nezavisno od platforme, namenjenim za distribuciju širom interneta. Međutim, Java omogućava i nov način pristupa pisanju programa. Na primer, Java je poboljšala i unapredila koncepciju objektne orijentisanosti koju je koristio i C++, a donela je integrisanu podršku za višenitni rad i ugrađenu biblioteku koja omogućava jednostavan pristup internetu. Ali, uzevši sve u obzir, Java nije postala tako izuzetan jezik zbog pojedinih mogućnosti koje pruža, nego zato što je kao celina bila savršen odgovor na pojavu novog i veoma distribuiranog računarskog sveta. Za programiranje na internetu, Java predstavlja ono što je bio C za sistemsko programiranje: revolucionarnu snagu koja je izmenila svet.

Veza sa jezikom C#

Doseg i moć Jave još uvek se osećaju u svetu razvoja računarskih jezika. Dobar deo njenih inovativnih karakteristika, konstrukcija i koncepata postao je obavezan sastavni deo svih novih jezika. Uspeh Jave je toliki da ga je nemoguće ignorisati.

Jezik C# možda je najvažniji primer Javinog uticaja, jer joj je veoma blizak. Microsoft je razvio C# kao podršku za svoju tehnologiju .NET Framework. Oba jezika imaju istu opštu sintaksu, podržavaju distribuirano programiranje i koriste isti model objekata. Naravno, između njih postoje i neke razlike, ali im je opšti izgled i utisak koji ostavljaju veoma sličan. „Ukrštanje" jezika Java i C# do sada je najjači dokaz da je Java promenila način razmišljanja o programskim jezicima i način njihove upotrebe.

Kako je Java uticala na internet

Internet je pomogao da se Java probije u prve redove programiranja, a Java je, za uzvrat, jako uticala na internet. Osim opšteg pojednostavljivanja veb programiranja, Java je uvela novu vrstu programa koja radi u mrežnom okruženju i koja se zove aplet. Aplet je promenio način na koji korisnici vide sadržaj objavljen na internetu. Java je takođe donela rešenje za neke od najnezgodnijih problema u oblasti prenosivosti i bezbednosti softvera na internetu. Te elemente detaljnije ćemo razmotriti u narednim odeljcima.

Java apleti

U vreme nastajanja Jave, jedan od njenih najuzbudljivijih mogućnosti bio je aplet. *Aplet* je posebna vrsta Java programa namenjena distribuciji preko interneta i automatskom izvršavanju u čitačima veba (engl. *web browsers*) koji podržavaju Javu. Aplet se preuzima s mreže, bez intervencije korisnika. Ako korisnik mišem pritisne hipervezu koja sadrži aplet, čitač veba automatski preuzima i izvršava taj aplet. Apleti su kratki programi koji se najčešće koriste za prikazivanje podataka koji stižu od servera, za obradu podataka koje unosi korisnik ili za izradu jednostavnih funkcija (kao što je kalkulator iznosa kamate) i izvršavaju se lokalno, a ne na serveru. Ukratko, aplet omogućava da se deo funkcionalnosti premesti sa servera na klijenta.

Uvođenje apleta bilo je važan događaj u to vreme zato što je proširilo svet objekata koji se mogu slobodno kretati kiberprostorom. Postoje dve veoma opšte kategorije objekata koji se razmenjuju između servera i klijenta: pasivni podaci i dinamički, aktivni programi. Na primer, kada čitate poruke koje ste dobili e-poštom, pregledate pasivne podatke. Čak i kad preuzmete program, kôd programa je i dalje samo grupa pasivnih podataka dok ga ne izvršite. Za razliku od toga, aplet je dinamički izvršiv program. Uprkos tome što je takav program aktivan na klijentskom računaru, njegovo izvršavanje pokreće server.

U ranim danima Jave, apleti su bili ključni deo programiranja na jeziku Java. Oni su ilustrovali moć i prednosti upotrebe Jave, dodavali uzbudljivu novu dimenziju veb stranicama i omogućavali programerima da istražuju celokupan opseg onoga što je bilo izvodljivo pomoću Jave. Mada se danas apleti i dalje koriste, tokom vremena postali su manje važni. Kao što će biti objašnjeno, od verzije JDK 9 nadalje, aplete polako zamenjuju drugi mehanizmi koji pružaju alterativne načine za isporučivanje dinamičkih, aktivnih programa putem veba.

Bezbednost koda

Koliko god da su korisni programi koji se dinamički izvršavaju u mrežnom okruženju, oni mogu doneti i ozbiljne probleme u oblastima bezbednosti i prenosivosti. Razume se, mora se obavezno sprečiti da program koji klijentski računar automatski preuzima i izvršava čini štetu. Osim toga, takav program treba da bude u stanju da se izvršava u različitim okruženjima i pod različitim operativnim sistemima.

Kao što ćete videti u detaljnijem razmatranju koje sledi, Java rešava navedene probleme efikasno i elegantno. Sada ćemo ih detaljnije razmotriti, počev od problema bezbednosti koda.

Verovatno znate da kad god s mreže preuzimate neki „normalan“ program, preuzimate i rizik, jer kôd koji preuzimate može sadržati virus, trojanac ili drugu vrstu štetnog koda. Suština problema leži u činjenici da zlonamerni kôd uspeva da prouzrokuje štetu zato što neovlašćeno pristupa sistemskim resursima. Na primer, virus može da prikupi važne privatne podatke, kao što su brojevi kreditnih kartica, stanja na bankovnim računima i lozinke za njih, tako što pretraži sadržaj sistema datoteka na lokalnom računaru. Da bi Java omogućila bezbedno preuzimanje i izvršavanje programa na klijentskom računaru, bilo je neophodno sprečiti da ti programi pokrenu tu vrstu napada.

Java pruža tu zaštitu tako što omgućava da aplikaciju ograničite na Javino izvršno okruženje i ne dozvolite joj da pristupa drugim delovima računara. (Videćete u nastavku teksta kako se to postiže.) Mogućnost preuzimanja programa uz visok stepen poverenja da neće biti načinjena šteta možda je jedan od najinovativnijih aspekata Jave.

Prenosivost koda

Prenosivost je veoma važan aspekt interneta jer na njemu postoje najrazličitije vrste računara i operativnih sistema. Ako dati Java program treba da radi na praktično svakom računaru na internetu, neophodno je omogućiti da se taj program izvršava na različitim sistemima. Drugim rečima, potrebno je da istu aplikaciju preuzima i izvršava široka lepeza različitih procesora, operativnih sistema i čitača veba. Pošto nije praktično da se za svaku vrstu računara piše i koristi različita verzija aplikacije, *isti* kôd mora da radi na *svim* računarima. Zbog toga je bio potreban način za generisanje prenosivog izvršnog koda. Kao što ćete uskoro videti, isti mehanizam koji pruža bezbednost omogućava i prenosivost.

Magija Jave: bajt kôd

Ono što Javi omogućuje da reši upravo pomenute probleme bezbednosti i prenosivosti jeste činjenica da kompajler jezika Java ne generiše izvršni kôd već tzv. bajt kôd. Bajt kôd (engl. *bytecode*) jeste visokooptimizovan skup instrukcija koji u trenutku izvršavanja programa tumači (interpretira) Javin izvršni sistem, poznat kao *Javina Virtuelna Mašina* (Java Virtual Machine, JVM). U suštini, prvobitna JVM bila je zamišljena kao *interpretator bajt koda*. To će vas možda iznenaditi, pošto se zbog postizanja boljih performansi mnogi savremeni jezici direktno prevode u izvršni kôd. Međutim, činjenica da Java programe izvršava JVM, pomaže da se reše glavni problemi s preuzimanjem programa sa interneta. Evo i zašto.

Java program preveden u bajt kôd mnogo se lakše izvršava u različitim okruženjima zato što je za različite platforme potrebno napraviti samo različite virtuelne mašine. Kada se u određeni sistem jednom ugradi odgovarajući paket za izvršavanje, na tom sistemu će moći da se pokrene svaki Java program. Iako se Javine virtuelne mašine razlikuju na različitim platformama, sve one razumeju isti Javin bajt kôd. Kada bi se Java programi direktno prevodili u izvršni kôd odgovarajućih mašina, onda bi morale postojati različite verzije programa za svaki procesor priključen na internet. Ovo, naravno, nije pogodno rešenje. Iz rečenog proizlazi da je izvršavanje bajt koda pomoću JVM-a najjednostavniji način pravljenja uistinu prenosivih programa.

Činjenica da Java programe izvršava JVM čini ih i bezbednijim. Pošto svime upravlja JVM, upravlja i izvršavanjem programa. Zato je JVM u stanju da za program formira ograničeno okruženje u kojem se program izvršava i koje se zove *karantin* (engl. *sandbox*), čime se programu sprečava neograničen pristup u sistem. Bezbednost je još povećana i određenim ograničenjima u samom jeziku Java.

Kada se program prevodi u neki međuoblik i zatim interpretira pomoću virtuelne mašine, on se po pravilu izvršava sporije nego da je direktno preveden u izvršni kôd. U Javi ta razlika u vremenu izvršavanja nije tako velika. Pošto je bajt kôd u velikoj meri optimizovan, njegova upotreba omogućuje Javinoj virtuelnoj mašini da programe izvršava mnogo brže nego što biste očekivali.

Mada je Java prvobitno zamišljena kao interpretiran jezik, s tehničkog aspekta nema nikakvih prepreka da se, radi ubrzanja rada, bajt kôd odmah prevodi u izvršni kôd za određeni ciljni računar. Zbog toga je nedugo posle objavljivanja Jave uvedena tehnologija HotSpot. HotSpot sadrži JIT (Just In Time) kompajler za bajt kôd. Kada je JIT kompajler ugrađen u JVM, odabrani delovi bajt koda prevode se u izvršni kôd u realnom vremenu – deo po deo, po potrebi. Treba razumeti da da se ceo Java program ne kompajlira odmah u izvršni kôd. Umesto toga, Java kompajlira kôd prema potrebama, tokom izvršavanja. Osim toga, ne kompajlira se ceo bajt kôd, nego samo oni njegovi delovi koji će bitno ubrzati izvršavanje. Preostali deo koda se i dalje samo interpretira. Međutim, čak i ovim načinom „kompajliranja samo kad zatreba“ još uvek se postižu bolje performanse pri izvršavanju koda. Bezbednost i prenosivost programa ne smanjuju se kada se na bajt kôd primeni ovakvo dinamičko prevođenje, jer izvršnim okruženjem i dalje upravlja JVM.

Još nešto: od verzije JDK 9 nadalje, izabrana Java okruženja imaće i *kompajler unapred*, koji se može iskoristiti za kompajliranje bajt koda u mašinski kôd pre izvršavanja na JVM, umesto u hodu. Kompajliranje bajt koda unapred je specijalizovana mogućnost koje ne zamenjuje opisani Javin tradicionalni način. Osim toga, kompajliranje unapred uvodi više ograničenja. Tri primera: u vreme pisanja ove knjige, kompajliranje unapred je predviđeno samo za eksperimentalne namene i izvodljivo samo za 64-bitne Linux verzije Jave, a tako unapred kompajliran kôd mora se izvršavati samo na istom (ili na slično konfigurisanom) sistemu na kom je kôd bio i kompajliran. To znači da unapred kompajliran kôd umanjuje prenosivost. Zbog visoko specijalizovane prirode kompajliranja unapred, u ovoj knjizi ga nećemo dalje razmatrati.

Prevazilaženje apleta

Kao što je već objašnjeno, u ranim danima Jave apleti su bili ključni deo programiranja na jeziku Java. Ne samo što su dodavali uzbudljivost veb stranici, nego su bili i vrlo jasno uočljiv deo Jave, koji je pojačavao njenu privlačnost. Međutim, za rad apleta u čitaču veba bio je neophodan i Java priključni dodatak. To je značilo da je čitač veba morao da podržava rad apleta. Ali u novije vreme u čitačima veba opada podrška za upotrebu Java priključnih dodataka. Jednostavno rečeno, bez podrške u čitaču veba, apleti su neupotrebljivi. Iz tog razloga, od verzije JDK 9 nadalje, apleti se u Javi više ne preporučuju. U jeziku Java, nešto što se ne preporučuje znači da je to i dalje na raspolaganju, ali se smatra zastarelim. U nekom budućem izdanju jezika, nepreporučljiva (odnosno izastarela) mogućnost može biti u potpunosti izbačena. Iz tog razloga, zastarele mogućnosti ne bi trebalo ugrađivati u nov kôd.

Postoje razne alternative za aplete, od kojih je verovatno najvažnija Java Web Start. Java Web Start omogućava da se aplikacija dinamički preuzima sa veb stranice. Razlika je u tome što se sada ta aplikacija izvršava samostalno, a ne unutar čitača veba, pa joj zbog toga nije potreban Java dodatak u čitaču. Java Web Start je mehanizam za distribuiranje aplikacija koji je upotrebljiv za mnoge vrste Java programa. Mada detaljno razmatranje strategija distribuiranja apliakcija izlazi izvan opsega ove knjige, zbog važnosti te teme, u dodatku B naći ćete kratak uvod u Java Web Start.

Servleti: Java na serverskoj strani

Kôd na klijentskoj strani je samo polovina jednačine klijent/server. Nedugo posle objavljivanja prvobitne verzije, postalo je očigledno da bi Java bila korisna i na serverskoj strani. Rezultat je bio *servlet*. Servlet je kratak program koji se izvršava na serveru.

Servleti se koriste za generisanje dinamičkog sadržaja koji se zatim šalje klijentu. Recimo, na stranici veb prodavnice servlet može učitavati cenu određenog artikla iz baze podataka. Podatak o ceni zatim se koristi za dinamičko generisanje veb stranice koja se šalje čitaču. Mada je dinamički generisan sadržaj na raspolaganju pomoću mehanizama kao što je CGI (Common Gateway Interface), servlet pruža više prednosti, među kojima su i bolje performanse.

Pošto se servleti (kao i svi drugi Java programi) prevode u bajt kôd koji izvršava JVM, postiže se veoma visok stepen prenosivosti. To znači da se isti servlet može koristiti u velikom broju različitih serverskih okruženja. Jedina obaveza je da server podržava JVM i kontejner za servlet.

Pojmovi karakteristični za Javu

Nijedna priča o istoriji Jave nije potpuna ako se ne pomenu izrazi koji su karakteristični za nju. Mada su osnovni činioci koji su uticali na pojavu Jave prenosivost i bezbednost, postoje i drugi činioci koji su odigrali važnu ulogu u oblikovanju Jave. Sledeća lista pojmova, koju je sastavio autorski tim Jave, ukazuje na osnovne pravce razmišljanja:

- jednostavno
- bezbedno
- prenosivo
- objektno orijentisano
- robusno
- višenitno
- nezavisno od platforme
- interpretirano
- visokoefikasno
- distribuirano
- dinamičko

Dva pojma – bezbedno i prenosivo – već smo objasnili. Pogledajmo šta znače ostali.

Jednostavno

Java je koncipirana tako da programeri mogu lako da je nauče i efikasno koriste. Pod uslovom da imate određeno iskustvo u programiranju, neće vam biti teško da ovladate Javom. Ako već poznajete osnovne pojmove objektno orijentisanog programiranja, učenje Jave će vam biti još lakše. Najbolje je ako ste iskusan programer na jeziku C++, jer ćete veoma lako preći na Javu. Pošto je Java nasledila sintaksu jezika C i C++, kao i mnoge objektno orijentisane osobine jezika C++, učenje Jave većini programera ne stvara teškoće.

Objektno orijentisano

Iako je Java nastala pod uticajem svojih prethodnika, nije bilo predviđeno da njen izvorni kôd bude kompatibilan sa izvornim kodom bilo kog drugog jezika. Zato su autori Jave imali potpuno odrešene ruke. Jedan rezultat koji je odavde proizišao jeste čist, upotrebljiv i pragmatičan tretman objekata. Slobodno pozajmljujući principe iz mnogih originalnih objektnih sof-

tverskih okruženja koja su nastala tokom minulih decenija, Java je uspela da održi ravnotežu između puritanskog stava „sve je objekat“ i pragmatičnog pravila „ne smetaj mi“. Model objekta u Javi jednostavan je i lako se proširuje, dok se prosti tipovi, kao što su celi brojevi, ne obrađuju kao objekti jer je tako mnogo efikasnije.

Robusno

Veb okruženje sa više platformi postavlja izuzetne programske zahteve jer program mora da se pouzdano izvršava na različitim sistemima. Zbog toga je pri projektovanju Jave dat prioritet sposobnosti da se napravi robustan program. U cilju postizanja pouzdanosti rada programa, Java vas ograničava u nekoliko ključnih područja, terajući vas da programske greške ispravite u ranoj fazi. Istovremeno, Java vas oslobađa briga o mnogim najčešćim programskim greškama. Pošto je Java strogo tipiziran jezik, ona proverava vaš kôd u trenutku njegovog kompajliranja. Međutim, ona ga proverava i u trenutku izvršavanja. U stvari, mnoge neuhvatljive greške – one koje se pojavljuju u situacijama koje je teško simulirati – u Javi se ne mogu ni napraviti. Glavna prednost Jave je to što možete da predvidite kako će se ono što ste napisali ponašati u različitim uslovima.

Da biste bolje razumeli robusnost Jave, setite se dva glavna razloga otkazivanja programa: greške pri upravljanju memorijom i loša obrada izuzetaka (tj. grešaka koje nastaju tokom izvršavanja programa). U tradicionalnom programskom okruženju, upravljanje memorijom može da bude težak i zapetljan posao. Na primer, u jezicima C i C++ programer često mora da ručno zauzima i oslobađa svu dinamički dodeljenu memoriju. To ponekad dovodi do problema, jer programeri zaboravljaju da oslobode prethodno dodeljenu memoriju ili, što je gore, pokušavaju da oslobode memoriju koju drugi deo njihovog koda još uvek koristi. Java otklanja te probleme jer dodeljuje i oslobađa memoriju umesto vas. (U stvari, memorija se oslobađa potpuno automatski, pošto Java obezbeđuje „sakupljanje smeća“, tj. uklanjanje iz sistema objekata koji se više ne koriste.) U tradicionalnom okruženju izuzeci često nastaju pri deljenju nulom, ili kada se ne pronađe odgovarajuća datoteka, i oni se moraju razrešiti rogobatnom, teško čitljivom konstrukcijom. Java je i ovde od pomoći jer obezbeđuje objektno orijentisanu obradu izuzetaka. U dobro napisanom Java programu, sve greške koje nastanu u vreme izvršavanja mogu se obraditi istim tim programom i tome treba težiti.

Višenitno

Java je projektovana tako da izađe u susret realnim zahtevima pravljenja interaktivnih mrežnih programa. Da bi se to postiglo, Java podržava višenitno programiranje – varijantu programiranja koja omogućuje da vaš program istovremeno radi više stvari. Javin izvršni sistem ima elegantno i potpuno rešenje za sinhronizovanje više procesa, koje omogućuje da projektujete interaktivne sisteme što glatko rade. Javin lako primenljiv pristup istovremenom radu omogućava da više razmišljate o specifičnom ponašanju programa, umesto da se bavite podsistemom za višeprogramski rad.

Nezavisno od platforme

Osnovni problem projektanata Jave bio je kako obezbediti trajan i prenosiv kôd. U vreme stvaranja Jave, jedan od najozbiljnijih problema s kojim su se suočavali programeri bio je sledeći: ako su napisali program koji radi danas, niko nije mogao da garantuje da će on raditi i sutra, čak i na istoj mašini. Operativni sistemi se neprestano poboljšavaju, poboljšavaju se i procesori, a kada se sve kombinuje s promenama u osnovnim resursima sistema, može se dogoditi da pro-

gram više ne radi kako treba. Zbog toga su projektanti Jave morali da donesu nekoliko teških odluka o samom jeziku i o Javinoj virtuelnoj mašini. Njihov slogan je bio: „Napiši jednom, izvršavaj bilo gde, bilo kada". Taj cilj je najvećim delom i postignut.

Interpretirano i visokoefikasno

Kao što je ranije objašnjeno, Java omogućava pisanje programa za više platformi tako što prevodi izvorni kôd u međuproizvod, zvan bajt kôd. Takav kôd se može izvršavati na svakom sistemu koji ima Javinu virtuelnu mašinu. Najveći broj prethodnih pokušaja da se napravi kôd koji će raditi nezavisno od platforme ostvarivao je to na račun performansi. Kao što smo već pomenuli, Javin bajt kôd je pažljivo optimizovan tako da se po potrebi, pomoću JIT kompajlera, lako prevodi u mašinski kôd konkretnog računara, čime se postižu visoke performanse. Javini izvršni sistemi koji obezbeđuju ovakvo prevođenje njime nimalo ne umanjuju nezavisnost koda od platforme.

Distribuirano

Java je posebno namenjena distribuiranom internet okruženju jer lako rukuje protokolima TCP/IP. U stvari, pristupanje resursu pomoću URL-a u osnovi se ne razlikuje od pristupanja datoteci. Java podržava i *daljinsko pozivanje metoda* (Remote Method Invocation, RMI). Dakle, program može da zahteva izvršavanje procedura koje se nalaze na bilo kojoj mrežnoj adresi.

Dinamičko

Java programi sadrže znatne količine podataka o tipu koji se koriste za proveravanje i razrešavanje pristupa objektima u trenutku izvršavanja. Time je omogućeno dinamičko povezivanje koda na pouzdan i efikasan način. To je od ključnog značaja za robusnost okruženja apleta, u kojem se delići bajt koda mogu dinamički ažurirati na sistemu koji izvršava program.

Evolucija Jave

Već je prvo izdanje Jave izazvalo revoluciju, ali njime nije prestalo doba brzih, inovativnih promena Jave. Za razliku od većine drugih softverskih sistema koji se poboljšavaju korak po korak, Java je nastavila da se intenzivno razvija. Ubrzo posle objavljivanja Jave 1.0, pojavila se verzija 1.1. Osobine uvedene s Javom 1.1 mnogo su značajnije nego što bi se moglo pretpostaviti na osnovu broja verzije. U nju je dodato mnogo novih biblioteka, redefinisani su načini na koje apleti obrađuju događaje i iznova su konfigurisane mnoge osobine biblioteka iz verzije 1.0. U ovoj verziji takođe su u drugi plan (kao zastarele) potisnute mnoge osobine Jave 1.0. Na taj način, Java 1.1 je unela neke nove atribute u prvobitnu specifikaciju i istovremeno neke odbacila.

Sledeće veće proširenje Jave bila je verzija Java 2, gde brojka 2 označava drugu generaciju. Pojava Jave 2 predstavlja prelomni događaj koji obeležava početak „modernog doba" ovog jezika. Prvo izdanje Jave 2 nosilo je broj verzije 1.2. Možda vam je to čudno. Objašnjenje je to što se ovaj broj prvobitno odnosio na verziju Javinih biblioteka, a zatim je prihvaćen kao oznaka celog izdanja. Od izdavanja Jave 2, firma Sun je prepakovala Javu, nazvala je J2SE (Java 2 Platform Standard Edition) i brojeve verzija počela da primenjuje na taj proizvod.

Java 2 uvodi niz novih mogućnosti, među kojima su Swing i Collections Framework, poboljšanu Javinu virtuelnu mašinu i razne poboljšane programske alatke. Iz Jave 2 neki elementi su i odbačeni. To se najviše odnosi na klasu **Thread** u kojoj su metode **suspend()**, **resume()** i **stop()** označene kao zastarele.

J2SE 1.3 bilo je prvo veliko poboljšanje originalne verzije Jave 2. Njime je najvećim delom poboljšana postojeća funkcionalnost i bolje „zategnuto" razvojno okruženje. Programi pisani na verzijama 1.2 i 1.3 su u načelu međusobno kompatibilni na nivou izvornog koda. Iako verzija 1.3 sadrži manji broj izmena od prethodna tri izdanja, ona je svakako važna.

Novo obogaćenje Java je doživela u izdanju J2SE 1.4. Ono sadrži više važnih nadogradnji, poboljšanja i dodataka. Primera radi, dodati su nova rezervisana reč **assert**, ulančani izuzeci (engl. *chained exceptions*) i podsistem za kanalske U/I operacije. Izmenjene su i kolekcije i klase za umrežavanje. Uz to, posvuda je učinjeno više malih izmena. Uprkos značajnom broju novih karakteristika, verzija 1.4 zadržala je gotovo stoprocentnu kompatibilnost izvornog koda s prethodnim verzijama.

Sledeće izdanje Jave bilo je J2SE 5 i donelo je revolucionarne promene. Za razliku od većine prethodnih nadogradnji Jave, koje su donosile važna, ali nesuštinska poboljšanja, J2SE 5 iz osnova proširuje doseg, moć i oblast važenja tog jezika. Od pojavljivanja Jave pre deset godina nije bilo tako važnog, niti toliko željno očekivanog izdanja Jave.

Da biste shvatili veličinu izmena koje je J2SE 5 donela Javi, razmotrite sledeći spisak njenih glavnih novih mogućnosti:

- generički tipovi (engl. *generics*)
- anotacije (engl. *annotations*)
- automatsko pakovanje (engl. *autoboxing*) i automatsko raspakivanje (engl. *auto-unboxing*)
- nabrajanja (engl. *enumerations*)
- poboljšana petlja **for,** u stilu for–each
- metode s promenljivim brojem argumenata (vararg)
- uvoženje statičkih članova (engl. *static import*)
- U/I operacije s formatiranim podacima (engl. *formatted I/O*)
- skup klasa za konkurentno programiranje (engl. *concurrency utilities*)

Ovo nije spisak beznačajnih izmena niti delimičnih nadogradnji. Svaka stavka spiska predstavljala je značajan dodatak jeziku Java. Jedan deo stavki, kao što su generički tipovi, poboljšana petlja **for** i metode s promenljivim brojem argumenata, uvode nove elemente sintakse. Druge stavke, poput automatskog pakovanja i raspakivanja, menjaju semantiku jezika. Metapodaci dodaju programiranju novu dimenziju. U svakom slučaju, uticaj tiih dodataka prevazilazi njihove neposredne efekte. Oni menjaju sam karakter Jave.

Važnost novih mogućnosti odražava se u broju verzije „5". Bilo bi normalno da je sledeći broj verzije za Javu bio 1.5. Međutim, izmene i nove mogućnosti toliko su značajne da prelazak sa 1.4 na 1.5 naprosto nije mogao da izrazi veličinu promene. Sun je povećao broj verzije na 5 da bi naglasio važnost događaja, pa je ta verzija nazvana J2SE 5, a razvojni komplet – JDK 5. Međutim, da bi se održala doslednost, Sun je odlučio da 1.5 upotrebljava kao svoj interni broj verzije, koji se takođe naziva broj *razvojne verzije*. Dakle, 5 je eksterni, a 1.5 interni broj verzije. Broj „5" u JSE 5 nazvan je broj proizvodne verzije.

Sledeće izdanje Jave nazvano je Java SE 6. Kompanija Sun je još jednom odlučla da izmeni ime platforme Java. Prvo, obratite pažnju na to da je izostavljeno „2". Platforma je dobila ime *Java SE*, a puno zvanično ime proizvoda bilo je *Java Platform, Standard Edition 6*. Java razvojni komplet dobio je ime JDK 6. Kao i u verziji J2SE 5, broj 6 u naslovu Java SE 6 predstavlja broj verzije proizvoda. Interni razvojni broj je 1.6.

Java SE 6 je nadograđena na osnovicu J2SE 5, koju poboljšava i proširuje. U verziji SE 6, samom jeziku nisu dodate važnije mogućnosti, ali su proširene biblioteke API funkcija, uneto je nekoliko novih paketa i poboljšanja izvršnog okruženja. Tokom njenog dugačkog život-

nog ciklusa (prema Java terminologiji) uvedeno je više izmena. Verzija Java SE 6 uglavnom je poslužila da još više učvrsti napredak koji je donela verzija J2SE 5.

Sledeće izdanje Jave nazvano je Java SE 7, razvojni komplet se zove JDK 7 a interni broj verzije je 1.7. Java SE 7 je bilo prvo važnije izdanje Jave nakon što je kompaniju Sun Microsystems preuzeo Oracle. Verzija Java SE 7 je uvela mnoge nove mogućnosti, među kojima su i značajna proširenja jezika i biblioteka API funkcija. Dodate su i nadogradnje Java izvršnog okruženja radi podrške za jezike koji nisu Java, ali su za Java programere najzanimljivije bile dopune samog jezika i pratećih biblioteka.

Nove mogućnosti jezika razvijene su u okviru *projekta Coin*. Svrha projekta Coin bila je da identifikuje brojne manje izmene koje bi bile ugrađene u JDK 7. Mada se te nove mogućnosti bile smatrane kao „manje značajne", one su ipak prilično značajno uticale na kôd na koji se odnose. U stvari, za mnoge programere te izmene su svakako bile najvažnije nove mogućnosti koje je uvela Java SE 7. To su bile sledeće:

- Naredbom **switch** odsad može da upravlja i tip **String**.
- Binarni celobrojni literali.
- Znak za podvlaku u numeričkim literalima.
- Proširena naredba **try**, nazvana ***try****-with-resources*, podržava automatsko upravljanje resursima. (Na primer, tokovi se sada mogu zatvarati automatski kada više nisu potrebni.)
- Prepoznavanje tipa (pomoću operatora diamond) pri konstruisanju generičke instance.
- Poboljšana obrada izuzetaka kada dva ili više njih treba da presretne ista naredba **catch** (višestruki catch) i bolja provera tipa u izuzecima koji se ponovo izazivaju.
- Mada nije u pitanju izmena sintakse, poboljšana su upozorenja koja kompajler generiše za neke metode s promenljivim brojem argumenata, a imate i veći stepen kontrole nad tim upozorenjima.

Kao što vidite, uprkos tome što se mogućnosti koje je uveo projekat Coin ne smatraju tako značajnim izmenama jezika, koristi koje su one donele su znatno veće nego što bi se zaključilo po izrazu „manje značajan". Pre svega, naredba **try**-*with-resources* suštinski je izmenila način pisanja koda koji radi s tokovima. Osim toga, mogućnost da tip **String** upravlja naredbom **switch** bilo odavno je očekivano poboljšanje koje je u mnogim slučajevima pojednostavilo programski kôd.

Java SE 7 je donela i više dopuna Java API biblioteka. Dve najvažnije su proširenja NIO Frameworka i dodavanje Fork/Join Frameworka. NIO (koji se prvobitno zvao New I/O) dodat je Javi u verziji 1.4. Međutim, izmene koje su predložene za Javu SE 7 suštinski proširuju njegove mogućnosti. Te izmene su toliko značajne da se često koristi izraz NIO.2.

Biblioteka Fork/Join Framework pruža važnu podršku za *paralelno programiranje*. Paralelno programiranje je uobičajeno ime za tehnike koje omogućavaju efikasno korišćenje računara u koje je ugrađeno više od jednog procesora, uključujući i sisteme s više jezgara. Prednost okruženja s više jezgara jeste mogućnost značajnog poboljšavanja performansi programa. Biblioteka Fork/Join Framework je omogućila paralelno programiranje tako što:

- pojednostavljuje definisanje i pokretanje poslova koji se mogu izvršavati istovremeno.
- automatski radi s više procesora (kada ih ima).

To znači da pomoću biblioteke Fork/Join Framework lako možete praviti skalabilne aplikacije koje automatski koriste sve procesore koji su na raspolaganju u radnom okruženju apli-

kacije. Razume se, nisu baš svi algoritmi pogodni za paralelizovanje, ali se značajno može ubrzati izvršavanje onih koji jesu.

Sledeće izdanje Jave bilo je Java SE 8, a razvojni paket je dobio ime JDK 8. Interni broj verzije je 1.8. JDK 8 je bio značajna nadogradnja jezika Java zbog uvođenja nove mogućnosti koja je imala dalekosežne posledice: *lambda izraz*. Uticaj lambda izraza je bio i nastaviće da bude, značajan jer je promenio i način na koji su se konceptualizovala programska rešenja i nalin na koji se pisao Java kôd. Kao što je detaljno opisano u poglavlju 15, lambda izrazi dodaju jeziku Java mogućnosti funkcionalnog programiranja. Tokom tog postupka, lambda izrazi mogu da pojednostave i umanje količinu programskog koda koji je potreban za izražavanja određenih konstrukata, kao što su neki tipovi anonimnih klasa. Zbog lambda izraza, u jezik su uvedeni i novi operator strelica (–>) i nov sintaksni element.

Uvođenje lambda izraza imalo je i dalekosežne efekte i na Java biblioteke, u koje su ugrađene nove mogućnosti da bi se iskoristili lambda izrazi. Jedna od najvažnijih je novi API za rad s tokovima, koji je upakovan u **java.util.stream**. API za rad s tokovima podržava operacije koje obrađuju podatke u obliku cevovoda, a optimizovan je za lambda izraze. Drugi nov paket bio je **java.util.function**, u kojem je definisano više funkcionalnih interfejsa, koji pružaju dodatnu podršku za lambda izraze. Druge nove mogućnosti u vezi s lambda izrazima naći ćete u biblioteci API-ja.

Još jedna mogućnost koja se odnosi na lambda izraze tiče se **interfejsa**. Počev od verzije JDK 8 pa nadalje, sada je moguće definisati podrazumevanu implementaciju za metodu koja je navedena u određenom interfejsu. Ako za tu metodu nije izričito definisana imlementacija, koristi se podrazumevana implementacija zadata u samom interfejsu. To omogućava interfejsima da slobodno evoluiraju tokom vremena zato što interfejsu možete naknadno dodati novu metodu a da to ne naruši već postojeći kôd. To takođe može da poboljša implementaciju interfejsa kad su podrazumevane implementacije metoda potrebne. Među mnogim drugim novim mogućnostima JDK-a 8 možemo navesti nov API za rad s datumima i vremenima, anotacije tipova i mogućnost primene paralelne obrade kada sortirate sadržaj niza. U JDK 8 je ugrađena i podrška za JavaFX, Javain najnoviji frjemvork za GUI aplikacije. Predviđeno je da JavaFX vremenom zameni Swing u većini projekata koji imaju GUI. U četvrtom delu ove knjige naći ćete uvod u JavaFX.

Java SE 9

Najnovije izdanje Jave je Java SE 9. Razvojni paket se zove JDK 9. U verziji JDK 9, interni broj verzije takođe je 9. JDK 9 predstavlja važno novo izdanje Jave, koje donosi značajna poboljšanja i samog jezika Java i pripadajućih biblioteka. Isto kao i verzije JDK 5 i JDK 8, JDK 9 utiče na jezik Javu i njegove API biblioteke na suštinski način.

Najvažniju novinu predstavjaju moduli, koji vam omogućavaju da u kodu aplikacije zadate međusobne veze i zavisnosti. Moduli dodaju i novu dimenziju Javinim mogućnostma za kontrolu pristupa. Zbog modula, u Javu su dodati nov sintaksni element i više novih ključnih reči. Osim toga, u JDK je dodata alatka **jlink**, koja programeru omogućava da napravi izvršivu sliku aplikacije koja sadrži samo neophodne module. Definisan je i nov tip datoteke, nazvan JMOD. Moduli imaju i značajan efekat na API biblioteku zato što, od verzije JDK 9 nadalje, paketi biblioteka su organizovani u module.

Uprkos tome što moduli predstavljaju značajno proširenje Jave, oni su konceptualno jednostavni laki za upotrebu. Osim toga, budući da je podržan i stariji kôd napisna pre pojave modula, module možete uključiti tokom postupka razvoja kad god vam to odgovara. Nema

potrebe da odmah menjate već postojeći kôd tako da u njega uključite module. Ukratko, moduli dodaju značajnu funkcionalnost bez menjanja same suštine Jave.

Osim modula, JDK 9 uvodi i mnoge druge nove mogućnosti. Jedna od najzanimljivijih je JShell, što je alatka koja podržava interaktivno eksperimentisanje s programom i učenje. (Uvod u JShell naći ćete u dodatku C.) Druga zanimljiva dopuna je podrška za privatne metode u interfejsu. Njihovovuvođenje dalje propiruje podršku za podrazumevane metode u interfejsima uvedenu u verziji JDK 8. JDK 9 dodaje mogućnost pretraživanja u alatki **javadoc** i novu oznaku **@index** za podršku tome. Kao i prethodna izdanja, JDK 9 sadrži više proširenja Javainih API biblioteka.

Opšte pravilo u svakom izdanju Jave je da nove mogućnosti privlače najviše pažnje. Međutim, postoji jedan (nekada) važan aspekt Jave koji je u verziji JDK 9 postao sporedan: apleti. Od verzije JDK 9 nadalje, apleti se više ne preporučuju u novim projektima. Kao što je već navedeno u prethodnom delu ovog poglavlja, zbog sve slabije podrške za aplete u čitačima veba (a i zbog drugih razloga), JDK 9 zapostavlja ceo API za aplete. U vreme pisanja ove knjige, za distribuiranje programa putem interneta preporučuje se upotreba Java Web Starta. Budući da se apleti polako odbacuju i ne preporučuju se u novom kodu, nećemo ih detaljnije razmatrati u ovoj knjizi. Međutim, u dodatku D naći ćete kratak uvod u aplete.

Zaključak ove analize jeste da JDK 9 nastavlja Javaino usmerenje na inovacije, čime obezbeđuje da Java ostaje živ i zanimljiv jezik kakav je svet programiranja navikao da očekuje. Ovo izdanje knjige dopunjeno je tako da obrađuje verziju Java SE 9 (JDK 9). Dodati su opisi mnogih novih mogućnosti, dopuna i proširenja.

Kultura inovacija

Java je od svojih početaka bila u središtu kulture inovacija. Njeno prvo izdanje promenilo je definiciju programiranja za internet. Javina virtuelna mašina (JVM) i bajt kôd izmenili su način razmišljanja o problemima bezbednosti i prenosivosti koda. Prenosiv programski kôd je učinio da veb oživli. Proces Javine zajednice (Java Community Process, JCP) promenio je način usvajanja novih ideja i njihovog prenošenja u jezik. Svet Jave nikada nije dugo stajao u mestu.

Java SE 9 je najnovije izdanje u Javinoj neprekidnoj i dinamičnoj istoriji.