POGLAVLJE 1

Uvod

OVA knjiga će vam pomoći da efikasnije koristite programski jezik Java™ i njegove osnovne biblioteke, `java.lang`, `java.util` i, u manjoj meri, `java.io`. U knjizi su mestimično objašnjene i druge biblioteke, ali se ona ne bavi programiranjem grafičkih aplikacija niti primenom API interfejsa.

Knjiga se sastoji od 57 saveta, od kojih svaki predstavlja po jedno pravilo. Pravila sadrže iskustva za koja najbolji i najiskusniji programeri smatraju da su korisna. Ti saveti su grupisani u devet poglavlja, od kojih se svako bavi širokim aspektom projektovanja softvera. Ovu knjigu nije neophodno čitati od korica do korica: svaki savet je manje-više nezavisan. Saveti u velikoj meri upućuju na druge savete tako da ih možete čitati proizvoljnim redosledom.

Većina saveta je praktično prikazana primerima programa. Osnovna karakteristika ove knjige je da sadrži primere koda koji ilustruju mnoge *projektne obrasce* i tehnike. Neki od njih su stari, poput obrasca Singleton (2. savet) a neki su novi, poput obrazaca Finalizer Guardian (6. savet) i opreznog korišćenja metode `readResolve` (57. savet). Zaseban indeks olakšava pristup ovim obrascima i tehnikama (213. stranica). Gde je odgovaralo, navedena je i referenca u ovoj oblasti [Gamma95].

Mnogi saveti sadrže jedan ili više programa koji prikazuju neke tehnike koje u praksi treba izbegavati. Takvi primeri, poznati i kao *antiobrasci* (engl. *antipatterns*), jasno su obeleženi komentarom poput „`// Nikad ne radite ovo!`“ U svakom slučaju, savet objašnjava zašto je primer loš i predlaže alternativan pristup.

Ova knjiga nije namenjena početnicima: da biste je razumeli, treba da poznajete programski jezik Java. Ako to nije slučaj, pročitajte neku uvodnu knjigu o Javi [Arnold00, Campione00]. Iako knjiga treba da bude pristupačna svakome ko poznaje programiranje na Javi, biće zanimljiva čak i naprednijim programerima.

Većina pravila u ovoj knjizi potiče iz nekoliko osnovnih principa. Najznačajniji su jasnoća i jednostavnost. Korisnika modula nikad ne bi trebalo da iznenadi način na koji se model ponaša. Moduli bi trebalo da budu što manji. (U ovoj knjizi, pojam *modul* se odnosi na bilo koju komponentu softvera koja se može koristiti više puta – od

pojedinačne metode do složenih sistema koji sadrže više paketa.) Kôd je bolje ponovo upotrebiti nego ga kopirati. Zavisnosti između modula treba smanjiti na najmanji mogući nivo. Greške treba otkrivati što ranije po njihovom pravljenju, a idealno je u vreme prevođenja koda.

Iako pravila predstavljena u ovoj knjizi nisu uvek primenljiva, ona ipak opisuju najbolja programerska rešenja za većinu slučajeva. Ne morate slepo pratiti ova pravila, već ih slobodno i prekršite s vremena na vreme ako imate dobar razlog za to. Ovladavanje umetnošću programiranja, kao i u slučaju drugih disciplina, sastoji se, na prvom mestu, od poznavanja pravila, a zatim od prepoznavanja trenutka u kome ih treba prekršiti.

Ova knjiga se uglavnom ne bavi performansama. Ona se bavi pisanjem programa koji su jasni, ispravni, upotrebljivi, veliki, fleksibilni i lako se održavaju. Ako sve to postignete, najčešće je relativno jednostavno postići i dobre performanse (37. savet). Neki saveti razmatraju zahteve koji se odnose na performanse, a neki sadrže i statističke podatke o performansama. Te podatke, označene frazom „Na mom računaru“, trebalo bi shvatiti kao približne, u najboljem slučaju.

Ako to nešto znači, moj računar je stari, kućni 400 MHz Pentium® II sa 128 MB RAM memorije, koji koristi Sunovu verziju 1.3 standardnog razvojnog paketa (SDK) za Microsoft Windows NT® 4.0. Ovaj SDK sadrži i Sunovu klijentsku virtuelnu mašinu Java HotSpot™, vrhunsku realizaciju JVM-a predviđenu za korisnike.

Razmatrajući programski jezik Java i njegove biblioteke, ponekad je neophodno pozvati se na određene verzije izvršnog okruženja. Sažetosti radi, u ovoj knjizi se radije koriste „inženjerskih brojevi verzija“ nego zvanični brojevi verzija. Tabela 1.1 prikazuje odnose između imena verzija i inženjerskih brojeva verzija.

Tabela 1.1: Verzije Java platformi

Zvanično ime verzije	Inženjerski broj verzije
JDK 1.1.x / JRE 1.1.x	1.1
Java 2 Platform, Standard Edition, v 1.2	1.2
Java 2 Platform, Standard Edition, v 1.3	1.3
Java 2 Platform, Standard Edition, v 1.4	1.4

Dok se osobine uvedene u verziji 1.4 razmatraju u nekim savetima, u primerima programa, sa malim brojem izuzetaka, te osobine se ne koriste. Primeri su testirani u verziji 1.3. Većina primera, ako ne i svi, trebalo bi da rade bez izmena i u verziji 1.2.

Primeri su uglavnom celoviti, ali je akcenat stavljen pre na njihovu čitljivost nego na celovitost. U primerima se često koriste klase iz paketa `java.util` i `java.io`. Da biste preveli te primere, možda ćete morati da dodate jednu od ovih, ili obe, važne naredbe za uvoz:

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

Slične šablonske naredbe takođe su izostavljene. Na Web adresi knjige, `http://java.sun.com/docs/books/effective`, nalaze se pune verzije svih primera koje možete prevesti i pokrenuti.

U većem delu knjige koriste se tehnički termini na način na koji su definisani u *specifikaciji jezika Java 2 [JLS]*. Nekoliko termina zaslužuju da budu posebno pomenuti. Programski jezik podržava četiri tipa: *interfejs*, *klase*, *nizove* (engl. *arrays*) i *proste tipove* (engl. *primitives*). Prva tri tipa su poznati kao *referentni tipovi*. Instance klasa i nizova su *objekti*; vrednosti prostih tipova nisu objekti. *Članovi* klase se sastoje od *polja*, *metoda*, *klasa članova* (engl. *member classes*) i *interfejsa članova* (engl. *member interfaces*). *Potpis metode* (engl. *method's signature*) sastoji se od njenog imena i tipova formalnih parametara; potpis metode *ne* sadrži tip rezultata.

U ovoj knjizi se nekoliko termina koristi drugačije nego u specifikaciji jezika. Na primer, *nasleđivanje* (engl. *inheritance*) koristim kao sinonim za *izradu potklasa* (engl. *subclassing*). Umesto da se termin nasleđivanje koristi za interfejse, u ovoj knjizi se samo navodi da klasa *realizuje* interfejs ili da se jedan interfejs *proširuje* na drugi. Da bismo opisali nivo pristupa koji se primenjuje u slučaju da nijedan nije naveden, u ovoj knjizi se koristi opisni termin *privatno u paketu* (engl. *package-private*) umesto tehnički neprihvatljivog termina *podrazumevani pristup* (engl. *default access*) [JLS, 6.6.1].

U ovoj knjizi se koristi nekoliko tehničkih termina koji nisu definisani u specifikaciji jezika. Termin *izvezeni API*, ili jednostavno *API*, odnosi se na klase, interfejse, konstruktore, članove i serijske oblike na osnovu kojih programer pristupa klasama, interfejsu ili paketu. Termin *API*, skraćeno od *application programming interface (interfejs za programiranje aplikacija)*, koristi se umesto termina *interfejs* kako bi se izbegle dvosmislenosti zbog interfejsa u programskom jeziku. Programer koji piše program korišćenjem nekog API-ja naziva se *korisnik* API-ja. Klasa u čijoj se primeni koristi API je *klijent* tog API-ja.

Klase, interfejsi, konstruktori, članovi i serijalizovani oblici zajednički se nazivaju *elementi API-ja*. Izvezeni API se sastoji od elemenata API-ja kojima se može pristupiti izvan paketa koji definiše API. To su oni elementi API-ja koje može koristiti bilo koji klijent i koje autor API-ja podržava. To su takođe elementi za koje korisnički program Javadoc standardno generiše dokumentaciju. Uopšteno govoreći, izvezeni (javni) API paketa sastoji se od javnih i zaštićenih članova i konstruktora svake javne klase ili interfejsa u paketu.