1

Uvod u Javu

Čitajući ovo poglavlje steći ćete najosnovniju predstavu o programskom jeziku Java. U ovoj fazi nije toliko bitno da razumete sve detalje koje ćemo ovde pomenuti zato što ćemo sve teme iz ovog poglavlja kasnije obraditi mnogo detaljnije. Ovde samo želimo da vam predstavimo opšte koncepte na kojima se zasnivaju teme narednih poglavlja, najčešće situacije u kojima se koriste Java programi, kao i vrste programa koji odgovaraju svakoj od tih situacija.

Dakle, u ovom poglavlju ćete naučiti:

- osnovne karakteristike jezika Java;
- kako Java programi rade na vašem računaru;
- zašto Java programi mogu da rade na različitim računarima;
- osnovne postavke objektno orijentisanog programiranja;
- kako izgleda jednostavan Java program i kako ga možete pokrenuti korišćenjem alata Java Development Kit;
- šta je HTML i kako ga koristiti da bi se Java program uključio u veb stranicu.

Šta je zapravo Java?

Java je inovativni programski jezik koji je postao nezaobilazan kada su u pitanju programi koji treba da se izvršavaju na različitim sistemima. Najpre, Java vam omogućava da pišete male programe, takozvane **aplete**. To su programi koje možete da ugradite u veb stranice kako biste obezbedili određenu funkcionalnost. Ugrađivanje izvršnog koda u veb stranice donosi neke vrlo interesantne mogućnosti. Umesto da se na njoj na uobičajen način pasivno prikažu tekst i grafički elementi, veb stranica može da bude interaktivna, i to na način koji vi izaberete. U nju možete da ubacite animacije, igre, obradu interaktivnih transakcija itd. – mogućnosti su praktično neograničene.

Naravno, ugrađivanje programskog koda u veb stranice donosi posebne bezbednosne zahteve. Kada pristupa stranici u koju je ugrađen Java kôd, korisnik Interneta mora biti siguran da taj kôd neće ugroziti rad njegovog računara ili oštetiti podatke koje čuva u njemu. To znači da izvršenje ugrađenog koda mora biti kontrolisano, čime se sprečava bilo kakvo neželjeno ugrožavanje vašeg računara i osigurava da svaki Java kôd napravljen sa lošim namerama bude efikasno zaustavljen. U okviru Jave postoje mere koje smanjuju mogućnost sličnih događaja vezanih za Java aplete.

Javina podrška za mrežne aplikacije i aplikacije bazirane na Internetu ne završava se na apletima. Primera radi, Java serverske strane (*Java Server Pages*, JSP) predstavlja moćno sredstvo za pravljenje serverskih aplikacija koje su u stanju da dinamički prave i HTML stranice i da ih zatim šalju po konkretnim zahtevima klijenata. Naravno, stranice koje generiše JSP mogu da sadrže i Java aplete.

Osim toga, Java omogućava i pravljenje široke lepeze programa koji se neizmenjeni mogu izvršavati na svakom računaru čiji operativni sistem podržava Java okruženje. Ovo se odnosi na većinu savremenih računara. Čak možete da napišete programe koji će funkcionisati i kao obični programi i kao apleti.

Poslednjih godina Java je značajno unapređena, naročito od pojave standarda Java 2. Dijapazon funkcija koje pruža standardna operativna Java značajno je povećan. Sada u okviru ovog programskog jezika postoje obimne mogućnosti za pravljenje aplikacija sa interaktivnim grafičkim korisničkim interfejsom (GUI), detaljnu obradu slika i programiranje grafičkih elemenata, kao i podrška za pristup relacionim bazama podataka i mrežnu komunikaciju sa udaljenim računarima. Praktično sve vrste aplikacija sada mogu uspešno da se programiraju u Javi i sve one odlikuju se potpunom prenosivošću.

Naravno, Java se još uvek razvija i raste. U čitavom nizu poboljšanja izdvojićemo verziju 1.4 koja je donela mogućnost čitanja i pisanja jezika XML. Java 5.0 koja se pojavila nakon verzije 1.4 donela je neke nove i važne jezičke mogućnosti, kao i značajna proširenja biblioteka klasa. Ovo su samo neke od tema o kojima ćemo govoriti u ovoj knjizi.

Karakteristike Jave

Verovatno najvažnija Javina karakteristika jeste činjenica da je dizajnirana tako da u startu bude mašinski nezavisna. Java programi mogu se neizmenjeni koristiti na svakoj platformi koja podržava ovaj programski jezik. Naravno, manji problemi su uvek mogući budući da u krajnjoj liniji uvek zavisite od načina na koji je Java implementirana na datom računaru. Međutim, Java programi su, svakako, suštinski u mnogo većoj meri prenosivi od programa pisanih u drugim programskim jezicima. Aplikacija napisana u Javi zahtevaće samo jedan skup iskaza izvornog koda, bez obzira na broj različitih platformi na kojima će se koristiti. U svakom drugom programskom jeziku vrlo često je neophodno da se izvorni kôd aplikacije prilagodi konkretnom računarskom okruženju, naročito ako ima zahtevan grafički korisnički interfejs. Prema tome, Java omogućava značajne uštede i vremena i resursa kada su u pitanju razvoj, podrška i održavanje osnovnih aplikacija na većem broju različitih softverskih i hardverskih platformi.

Sledeću veoma važnu karakteristiku Jave predstavlja njena **objektna orijentisanost**. Objektno orijentisan pristup programiranju takođe predstavlja suštinsku karakteristiku svih Java programa, tako da ćemo u nastavku ovog poglavlja razmatrati i šta to konkretno znači. Objektno orijentisani programi su razumljiviji i za njihovo održavanje i proširivanje potrebno je manje vremena nego za programe koji su pisani bez korišćenja objekata.

Osim toga što je objektno orijentisana, u Javi su izbegnute mnoge teškoće i komplikacije koje prate neke druge objektno orijentisane programske jezike. Zahvaljujući tome, Java je jednostavnija i za učenje i za korišćenje. Uopšteno govoreći, u Javi ne postoje zamke koje su karakteristične za druge programske jezike. Učenje je, zahvaljujući tome, brže, dok je za postizanje kompetentnosti i samopouzdanja potrebno mnogo manje stvarnog programiranja. Pored svega pomenutog, Java se i lakše testira.

Programski jezik Java odlikuje se i ugrađenom podrškom nacionalnih skupova znakova. Programi se u Javi jednako lako pišu za korišćenje u Grčkoj ili Japanu, kao i za zemlje u kojima je u upotrebi engleski jezik. Naravno, podrazumeva se da navedene jezike i dobro poznajete. Programi se u startu mogu napraviti tako da podržavaju nekoliko različitih svetskih jezika i automatski se prilagođavaju okruženju u kome se kôd izvršava.

Učenje Jave

Java nije težak programski jezik, ali je ipak potrebno uložiti dosta truda da bi se savladao. Prilično je kompaktan, ali i vrlo moćan. Da bi se efikasno programiralo u Javi, pored ostalog, potrebno je da razumete i biblioteke koje idu uz program, a one su već prilično obimne. Redosled kojim ćete u ovoj knjizi učiti kako Java funkcioniše i kako da je koristite brižljivo je strukturiran. Zahvaljujući tome, stručnost i programersko samopouzdanje sticaćete na relativno lak i bezbolan način. Koliko god je to bilo moguće, u poglavljima je izbegnuto korišćenje pojmova koje još niste naučili. To znači da vaše prve aplikacije neće imati grafički korisnički interfejs. Iako vas to verovatno privlači, uključivanje GUI u prve aplikacije bi bilo isto kao kada biste učili da plivate skočivši u bazen u onom delu koji je najdublji. Ali, ako biste u vodu ušli u najplićem dela bazena i prvo pokušali samo da se održite na površini, opasnosti da se udavite praktično ne bi bilo. Osim toga, sve su šanse da biste, uz ovakav pristup, posle izvesnog vremena bili solidan plivač.

Java programi

Kao što smo već rekli, postoje dve vrste programa koje možete da pišete u Javi: one koji se ugrađuju u veb stranice i zovu Java apleti i normalne nezavisne programe koje nazivamo Java aplikacije. Java aplikacije dalje mogu da se podele na konzolne aplikacije, koje podržavaju samo izlaz u vidu znakova na ekranu vašeg računara (na računaru pod operativnim sistemom Windows, to je komandna linija) i aplikacije sa prozorima, u kojima možete da napravite i koristite veći broj prozora. U aplikacijama sa prozorima koriste se mehanizmi uobičajeni za programe ove vrste kao što su meniji, palete alata, okviri za dijalog i slično.

Učeći osnove programskog jezika Java, kao ilustracije određenih mehanizama poslužiće nam konzolne aplikacije. Ove aplikacije imaju jednostavan ulaz i izlaz u vidu komandne linije. To će vam pomoći da se usredsredite na specifičnosti jezika i ne razmišljate o složenim problemima vezanim za kreiranje i upravljanje prozorima. Tek kada ovladate svim mogućnostima Java jezika, preći ćemo na aplikacije sa prozorima i na primere apleta.

Učenje Jave – šta vas čeka

Pre nego što bilo gde krenemo, uvek prethodno imamo ideju gde smo se zaputili i kojim ćemo putem ići. Zato pogledajmo kako će izgledati naše „putovanje" kroz Javu. Uopšteno govoreći, učeći Javu pomoću ove knjige proći ćete kroz pet etapa:

1. Prvu etapu predstavlja ovo poglavlje. U njemu je dat prikaz osnovnih postavki vezanih za strukturu Java programa i objašnjenje kako oni funkcionišu. Između ostalog, govorićemo o tome šta su objektno orijentisani programi i kako se izvršni program pravi iz izvorne datoteke sa Java kodom. Razjašnjavanje ovih koncepata na samom početku značajno će vam olakšati savladavanje Jave.
2. Zatim ćete naučiti kako se kombinuju iskazi, koja sredstva možete da koristite za upisivanje osnovnih podataka u program, kako da izvodite proračune i kako da donosite odluke koje se baziraju na rezultatima tih proračuna. Ovo su ona najosnovnija znanja koja su neophodna za prelazak na sledeći nivo.
3. U trećoj etapi bavićemo se **klasama** – njihovim definisanjem i korišćenjem. Klase opisuju strukturu objekata, što znači da ćete se ovde prvi put sresti sa objektnom orijentisanošću Jave. Na kraju ove etape imaćete najosnovnija znanja o tome kako funkcioniše programski jezik Java i biti spremni da ta znanja praktično upotrebite.
4. U četvrtoj etapi naučićete kako da aktivnosti koje vaši programi izvršavaju podelite na odvojene zadatke koji mogu da se izvršavaju simultano. Ovo je posebno važno ukoliko, primera radi, u jednu veb stranicu želite da ugradite nekoliko apleta, a ne želite da svaki od njih mora da čeka na završetak izvršenja onog prethodnog. Ili, možda ćete želeti da neka lepa animacija nastavi da se izvršava dok vi igrate neku igricu, a da su oba programa deo iste veb stranice.
5. U petoj etapi detaljno ćemo vam pokazati kako da implementirate aplikaciju ili aplet sa grafičkim korisničkim interfejsom i kako da, u ovom kontekstu, realizujete interakciju sa korisnikom. Ovde će mogućnosti Javinih biblioteka klasa doći do punog izražaja. Kada završite ovu etapu, bićete potpuno spremni da u Javi napišete kompletne aplikacije ili aplete.

Dakle, kada pročitate ovu knjigu, trebalo bi da budete solidan Java programer. Sve ostalo je već stvar iskustva.

Da bismo vam pokazali kako Java funkcioniše, u ovoj knjizi smo koristili zaokružene primere. Preporučujemo vam da sve ove primere napravite i isprobate, čak i one najjednostavnije, i da ih sve sami ukucate. Slobodno eksperimentišite. Ukoliko vam nešto nije sasvim jasno, slobodno izmenite primer i posmatrajte šta će se desiti. Najbolje bi bilo da sami napišete neki svoj primer. Ako niste sigurni kako funkcioniše neki aspekt Jave o kome smo već govorili, bez oklevanja ga isprobajte praktično. Ne zaboravite da se na greškama uči, i to veoma uspešno.

Java okruženje

Programi napisani u Javi mogu da se izvršavaju na najrazličitijim računarima i pod raznim operativnim sistemima. Vaši programi će jednako dobro raditi na PC računaru pod bilo kojom podržanom verzijom operativnog sistema Microsoft Windows, kao i na nekoj Linux ili Sun Solaris radnoj stanici. Ovo je moguće zato što se program napisan u Javi zapravo ne izvršava direktno na vašem računaru, već u okviru standardizovanog okruženja po imenu **Java 2 Platforma** koje se u vidu softvera implementira na čitavom nizu računara i operativnih sistema. Platforma Java ima dve komponente – softversku implementaciju hipotetičkog računara po imenu **Java virtuelna mašina** (*Java Virtual Machine*, JVM) i **Javin programski interfejs aplikacije** (*Java Application Programming Interface*, Java API) koji predstavlja skup softverskih komponenti koje obezbeđuju sve ono što vam je neophodno za pisanje zaokruženih interaktivnih aplikacija u Javi.

Java kompajler pretvara Javin izvorni kôd koji ste napisali u binarni program koji se sastoji od **bajtkodova**. Bajtkodovi predstavljaju mašinske instrukcije za Java virtuelnu mašinu.

Prilikom izvršavanja programa napisanog u Javi, program po imenu **Java interpreter** ispituje i dešifruje bajtkod, proverava njegovu autentičnost i bezbednost i zatim u okviru komponente Java virtuelne mašine izvršava akcije koje su naznačene bajtkodom. Java interpreter može da se izvršava samostalno ili u okviru pretraživača veba kao što su, na primer, Netscape Navigator, Mozilla ili Microsoft Internet Explorer u kojima se automatski poziva kad god veb stranica koja se prikazuje sadrži aplet.

S obzirom na to da se Java programi sastoje od bajtkoda, a ne od mašinskih instrukcija, oni su u potpunosti izolovani od konkretnog hardvera na kome se izvršavaju. Svaki računar na kome postoji Java okruženje može da izvršava vaš Java program jednako dobro kao i bilo koji drugi. Osim toga, Java interpreter se nalazi između vašeg programa i samog računara, tako da može da spreči izvršenje svih neovlašćenih akcija u okviru programa.

U prošlosti su fleksibilnost i bezbednost povlačile i sporije izvršenje Java programa. Interpretirani Java program se obično izvršavao deset puta sporije od ekvivalentnih programa kod kojih su upotrebljene nativne mašinske instrukcije. Međutim, u aktuelnim implementacijama Jave ove slabije performanse su u velikoj meri nadoknađene, pa u programima koji nisu računski zahtevni kao što su, na primer, oni koje ćete ugrađivati u veb stranice, nećete primetiti baš nikakvu razliku u performansama. Prilikom korišćenja platforme JVM, dela aktuelnog kompleta Java 2 Development Kit (JDK) koji možete da preuzmete sa veb lokacije kompanije Sun, postoji svega nekoliko situacija u kojima ćete primetiti pad performansi u odnosu na program kompajliran do nivoa nativnog mašinskog kôda.

Razvoj Java programa

Za korišćenje ove knjige potrebna vam je platforma Java 2 Platform, Standard Edition (J2SE), verzija 5.0 ili neka aktuelnija. Komplet JDK za čitav niz hardverskih platformi i operativnih sistema možete da preuzmete ili sa veb lokacije kompanije Sun `http://java.sun.com` (za Windows, Solaris ili Linux) ili sa lokacija na koje ćete biti upućeni sa ove. Komplet JDK koji ćete ovde koristiti možete da preuzmete sa adrese `http://java.sun.com/j2se`. Verzije kompleta Java Development Kit za Mac OS X možete da preuzmete sa adrese `http://devworld.apple.com/java/`.

Skrećemo vam pažnju na to da je platforma J2SE 5.0 nasledila verziju J2SE 1.4. Bilo bi logično da je verziju 1.4 nasledila verzija 1.5, ali je odlučeno da se, zbog značaja novih funkcija i sveukupne zrelosti proizvoda, nova verzija nazove 5.0. Kodna imena za module u verziji 5.0 i dalje koriste notaciju 1.5.0, koju ćete videti i u imenima direktorijuma i na nekoliko drugih mesta.

Jedan aspekt terminologije takođe nekada ume da zbuni – komplet Java Development Kit se nekada naziva baš tako – JDK ili Java Development Kit, a nekada SDK ili Software Development Kit. U verziji 5.0 koristi se termin JDK, dok se u verziji 1.4 koristio termin SDK. Prema tome, u pitanju je samo mala terminološka razlika, a značenje je praktično isto. Radi doslednosti, svaku verziju kompleta Java Development Kit u ovoj knjizi ćemo nazivati JDK.

Za pravljenje izvornih datoteka Java programa koje ćete koristiti u kompletu JDK, potreban vam je editor teksta. Dolazi u obzir svaki editor koji u sadržaj datoteke ne ugrađuje formatiranje. Postoji čitav niz podesnih besplatnih programa od kojih su neki baš i namenjeni Javi, tako da ćete bez većih problema pronaći onaj koji vam najviše odgovara. Nama je najviše odgovarao editor Jcreator. Postoje i potpuno besplatna verzija ovog programa, ali i komercijalna verzija koja je nešto bogatija funkcijama. Za početak vam je ova besplatna verzija sasvim dovoljna i možete je preuzeti sa adrese `http://www.jcreator.com`. Ukoliko želite da ispitate i neke druge editore, obavezno posetite veb lokaciju `http://www.download.com`.

Postoji nekoliko izuzetnih profesionalnih okruženja za razvijanje Java programa čiji su proizvođači kompanije Sun, Borland, Metrowerks i Symantec. U pitanju su okruženja u kojima su

pravljenje i editovanje Java izvornog koda, kao i kompajliranje programa i naknadno otklanjanje grešaka maksimalno olakšani. U rukama iskusnih programera ovo su veoma moćne alatke. Ipak, preporučujemo vam da se, učeći Javu pomoću ove knjige, naročito ako nemate zavidno programersko iskustvo, držite kompleta JDK kompanije Sun i nekog najjednostavnijeg editora za pravljenje izvornog koda. Verovatno se pitate zašto vam ne preporučujemo korišćenje alatki koje olakšavaju i ubrzavaju programiranje. Postoji više razloga. Najpre, profesionalni sistemi za razvoj programa kriju od vas mnogo toga što morate da znate ukoliko želite da savladate Javu. Zatim, ova okruženja koriste se za pravljenje složenih aplikacija sa mnogo koda i bolje je da se u fazi učenja te kompleksnosti klonite. Praktično sva komercijalna okruženja za razvoj Java programa imaju unapred ugrađene funkcije koje ubrzavaju programiranje. Ovo je izuzetno korisno za iskusne programere, ali predstavlja ozbiljnu smetnju u procesu učenja. Treba da vodite računa i o tome da su komercijalna okruženja nekada povezana sa konkretnom verzijom platforme Java 2. To znači da neke od najnovijih funkcija možda neće funkcionisati. Ukratko, profesionalna okruženja za programiranje u Javi namenjena su prvenstveno obučenim i iskusnim programerima, tako da vam preporučujemo da na njih pređete tek kada pročitate ovu knjigu.

Sa druge strane, ukoliko iz bilo kog razloga želite da radite u nekom komercijalnom sistemu za razvoj Java programa, a imate problema sa nekim primerom iz knjige, dati primer uvek možete da, korišćenjem komandne linije, isprobate i u okviru kompleta JDK. Po svoj prilici, primer će funkcionisati bez problema.

Instaliranje kompleta JDK

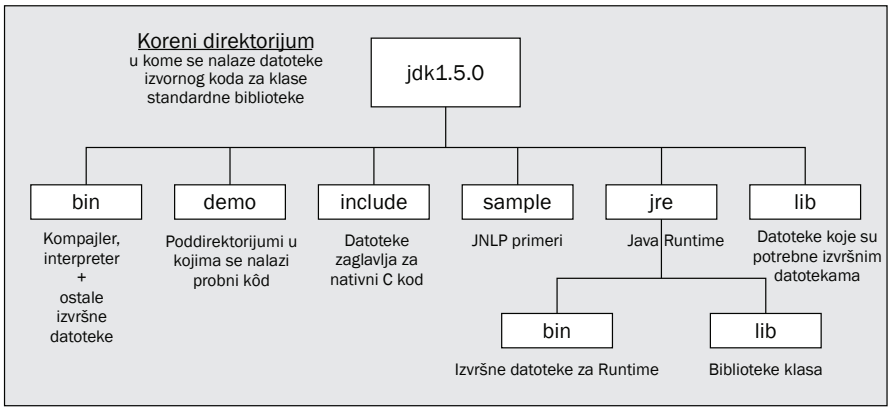
Na veb lokaciji kompanije Sun postoje detaljne instrukcije o postupku instaliranja kompleta JDK u okviru raznih operativnih sistema, tako da ovde nećemo ulaziti u sve varijacije ovog postupka. Ipak, trebalo bi da obratite pažnju na nekoliko stvari koje možda nećete primetiti u dokumentaciji postupka instaliranja.

Najpre, komplet JDK i njegova dokumentacija su potpuno nezavisni i instaliraju se posebno. Komplet JDK za operativni sistem Windows postoji u dve verzije – kao veb instalacija u kojoj se datoteke preuzimaju redom i kao kompletno preuzimanje `.exe` datoteke čijim se aktiviranjem pokreće celokupna instalacija. Dokumentaciju kompleta JDK čini veliki broj HTML hijerarhijski uređenih datoteka koje se distribuiraju u formi ZIP datoteke. Biće vam lakše da najpre instalirate JDK, pa tek onda i dokumentaciju. Ukoliko pod operativnim sistemom Windows komplet JDK instalirate na jedinicu C:, dobićete strukturu direktorijuma kao na slici 1-1.

Direktorijum `jdk1.5.0_n` sa slike 1-1 nekada se naziva i Javin koreni direktorijum, a nekada Javin „home" direktorijum. U imenu stvarnog korenog direktorijuma može se pojaviti i broj izdanja kompleta, što znači da bi ime direktorijuma moglo da glasi `jdk.5.0_n`, gde *n* predstavlja broj izdanja. Primera radi, u prvom izdanju za održavanje ovaj direktorijum zvao bi se `jdk.5.0_01`.

U direktorijumu `sample` nalaze se primeri aplikacija koje koriste protokol JNLP (Java Network Launching Protocol) koji se koristi za izvršavanje aplikacija ili apleta sa mrežnog servera bez potrebe korišćenja pretraživača ili preuzimanja i instaliranja koda.

Nema razloga da previše razmišljate o sadržaju ovih direktorijuma, pogotovo ne sada, na samom početku, ali bi trebalo da putanjama koje definišete u okviru svoje promenljive okruženja `PATH` dodate i putanju `jdk.5.0\bin`. Ovo će vam omogućiti da kompajler i interpreter aktivirate sa bilo kog mesta, bez prethodnog navođenja putanje. Ukoliko ste JDK instalirali na jedinici C:, onda će ova putanja glasiti `C:\jdk.5.0\bin`.



Slika 1-1

Samo jedno upozorenje – ukoliko ste prethodno instalirali neki komercijalni proizvod za programiranje u Javi, proverite da li je on možda izmenio vašu promenljivu okruženja `PATH` i u nju ubacio putanje do vlastitih Java izvršnih datoteka. Ako jeste, prilikom pokretanja Java kompajlera ili interpretera pojaviće se verzija datog komercijalnog proizvoda, a ne ona koju ste dobili uz JDK. Ovaj problem možete da rešite tako što ćete sami ukloniti neželjenu putanju ili putanje. Ukoliko iz nekog razloga ne želite da uklonite ove putanje komercijalnog proizvoda, prilikom svakog pokretanja kompajlera ili interpretera moraćete da navedete njihovu punu putanju. U direktorijumu `jre` nalaze se Java Runtime elementi koji se koriste prilikom izvršavanja Java programa. Klase Javine biblioteke čuvaju se u direktorijumu `jre\lib`. Njih, međutim, nećete moći da vidite pojedinačno zato što su upakovane u arhivu `rt.jar`. Ovde ništa ne dirajte. Java Runtime prilikom izvršenja vaših programa sam iz ove arhive uzima ono što mu je potrebno.

Promenljiva okruženja `CLASSPATH` predstavlja čest izvor problema i zabune za početnike u ovom programskom jeziku. Aktuelni komplet JDK ***NE ZAHTEVA*** definisanje ove promenljive, ali ako ju je već definisala neka druga verzija Jave ili nekog drugog sistema, ona će vam gotovo sigurno napraviti probleme. Komercijalni sistemi za programiranje u Javi i verzije kompleta JDK starije od 1.2 najčešće definišu promenljivu `CLASSPATH`. Zato obavezno proverite da li je ova promenljiva okruženja definisana na vašem sistemu. Ako jeste, a vi više ne koristite aplikaciju koja je ovu promenljivu instalirala, slobodno je obrišite. Sa druge strane, ukoliko baš morate da je zadržite – recimo da želite da zadržite sistem koji je definisao ili delite računar sa nekim kome je ona i dalje potrebna, prilikom svakog izvršavanja Java koda moraćete da privremeno definišete promenljivu `CLASSPATH` korišćenjem opcije komandne linije. U nastavku ovog poglavlja pokazaćemo vam kako se to radi.

Ukoliko želite da se i JDK dokumentacija instalira u hijerarhiju prikazanu na slici 1-1, raspakujte je u Java korenom direktorijumu. Ako ste komplet JDK instalirali na jedinici C:, to bi onda bila putanja `C:\jdk.5.0`. Nakon toga, u Java korenom direktorijumu pojaviće se i poddirektorijum `docs` u koji treba da instalirate datoteke ove dokumentacije. Kada budete želeli da koristite ovu dokumentaciju, jednostavno pokrenite datoteku `index.html` koja će se nalaziti u ovom poddirektorijumu..

Raspakivanje izvornog koda biblioteka klasa

Izvorni kôd biblioteka klasa nalazi se u arhivi `src.zip` koju ćete pronaći u Java korenom direktorijumu `jdk.5.0`. Kada savladate osnove programskog jezika Java, pretraživanje ovog izvornog koda biće vam od velike koristi. Čak i iskusnijim Java programerima ove izvorne datoteke mogu pomoći da shvate kako određene stvari funkcionišu ili zašto nešto ne funkcio-

niše. Izvorne datoteke možete da raspakujete iz arhive pomoću programa Winzip, pomoćnog programa JAR koji se dobija uz JDK ili bilo kog drugog programa koji je u stanju da raspakuje `.zip` arhive. Skrećemo vam pažnju na to da je u pitanju velika arhiva i da će taj postupak potrajati.

Nakon raspakivanja sadržaja arhive `src.zip` u Java korenom direktorijumu `\jdk1.5.0` pojaviće se poddirektorijum `src` u koji će se automatski instalirati izvorni kôd. Ukoliko želite da ga pregledate i u njemu pronađete neku klasu u bilo kom editoru teksta, samo otvorite `.java` datoteku koja vas zanima..

Kompajliranje Java programa

Javin izvorni kôd se uvek čuva u datotekama sa oznakom tipa `.java`. Kada napravite izvorni kôd za program i sačuvate ga u datoteci `.java`, potrebno je da se izvorni kôd obradi Java kompajlerom. Pre nego što upotrebite kompajler koji se dobija uz JDK, direktorijum u kome se nalazi vaš izvorni kôd učinite tekućim direktorijumom, a zatim zadajte sledeću komandu:

```
javac MyProgram.java
```

U ovom slučaju `javac` je ime Java kompajlera, a `MyProgram.java` ime izvorne datoteke programa. U ovoj komandi pretpostavlja se da se u tekućem direktorijumu nalazi vaša izvorna datoteka. Ukoliko ona nije tu, kompajler neće moći da je pronađe. Isto tako, ovde smo pretpostavili i to da izvorna datoteka odgovara definiciji programskog jezika Java iz aktuelne verzije kompleta JDK. Postoji i opcija komandne linije `–source` kojom možete da naznačite verziju Jave. Za JDK 5.0 komanda koju smo već naveli bi uz dodatak ovog parametra glasila:

```
javac -source 5 MyProgram.java
```

Skrećemo vam pažnju na to da kao vrednost opcije `–source` možete da koristite i 1.5 i u tom slučaju komanda izgleda ovako:

```
javac -source 1.5 MyProgram.java
```

U praksi možete da zanemarite ovu opciju komandne linije osim ukoliko ne želite da kompajlirate Java program koji je napisan u starijoj verziji kompleta JDK. Za kompajliranje koda koji je napisan u JDK 1.4, napisali biste sledeću komandu:

```
javac -source 1.4 oldSourceCode.java
```

Evo i jednostavnog programa koji možete da isprobate u kompajleru:

```
public class MyProgram {
 public static void main(String[] args) {
  System.out.println("Rome wasn't burned in a day!");
 }
}
```

Sve što ćete kao rezultat dobiti jeste ispisivanje jedne rečenice u komandnoj liniji. S obzirom na to da je poenta ovog programa samo isprobavanje kompajlera, sada nećemo objašnjavati kako on radi. Naravno, ovaj kôd morate da upišete upravo onako kako je naveden i snimite izvornu datoteku kao `MyProgram.java`. Ako pogrešite, kompajler će da signalizira grešku.

Ukoliko biste želeli da prevaziđete postojeću definiciju promenljive CLASSPATH – recimo, zato što ju je definisao neki drugi sistem za programiranje u Javi, komanda bi izgledala ovako:

```
javac -classpath . MyProgram.java
```

Iza reči –classpath dolazi vrednost ove promenljive, a u ovom slučaju stoji samo tačka. Na ovaj način definiše se samo putanja do tekućeg direktorijuma. To znači da će kompajler vašu izvornu datoteku (ili datoteke) potražiti u tekućem direktorijumu. Ukoliko zaboravite na ovu tačku, kompajler neće moći da pronađe izvorne datoteke u tekućem direktorijumu. U svakom slučaju, opciju –classpath. možete da ubacite bez ikakvog straha zato što ona nikako ne može da škodi.

Skrećemo vam pažnju na to da svoje izvorne datoteke ne treba da čuvate u strukturi direktorijuma kompleta JDK zato što vam to može napraviti probleme. Bolje je da izvorne kodove svojih programa čuvate u posebnim direktorijumima koje ste sami napravili.

Ako u vašem programu nema grešaka, kompajler će da generiše bajtkod program koji predstavlja ekvivalent vašeg izvornog koda. Bajtkod program se čuva kao datoteka istog imena kao izvorna datoteka, ali sa oznakom tipa .class. Javini izvršni moduli uvek se čuvaju u vidu datoteka sa oznakom tipa .class. Prema podrazumevanim parametrima, datoteka .class biće sačuvana u onom direktorijumu u kome se nalazi i izvorna datoteka.

Opcije komandne linije koje smo vam ovde prikazali samo su neke od opcija koje možete da koristite u kompajleru. Međutim, one su sasvim dovoljne za kompajliranje svih primera iz ove knjige. U okviru JDK dokumentacije postoji detaljan opis svih raspoloživih opcija kompajlera. Pored toga, navođenjem opcije –help možete da dobijete spisak standardnih opcija koje vam stoje na raspolaganju.

Ukoliko za razvijanje programa u Javi koristite neki drugi proizvod, prilikom kompajliranja programa ćete verovatno koristiti neki pristupačniji grafički korisnički interfejs u kome nećete morati da na ovaj način upisujete komande. Međutim, oznake tipa rezultujućih izvornih i klasnih datoteka svuda su iste.

Izvršavanje Java aplikacije

Da biste pokrenuli izvršavanje bajtkod programa u vidu datoteke .class pomoću Java interpretera koji je deo kompleta JDK, direktorijum u kome se nalazi datoteka .class učinite tekućim i zatim upišite komandu:

```
java -enableassertions MyProgram
```

Skrećemo vam pažnju na to da smo za identifikovanje programa koristili samo ime MyProgram, a ne ime datoteke MyProgram.class koju je generisao kompajler. Početnici često prave ovu grešku i, po analogiji sa operacijom kompajliranja, koriste ime datoteke. Kada biste imenu MyProgram dodali i oznaku tipa .class, vaš program se ne bi izvršio i dobili biste sledeću poruku o grešci:

```
Exception in thread "main" java.lang.NoClassDefFoundError: MyProgram/class
```

Dok kompajler očekuje da pronađe ime vaše izvorne datoteke, Java interpreteru potrebno je ime klase – u ovom slučaju MyProgram – a ne ime datoteke. Datoteka MyProgram.class sadrži i klasu MyProgram. Uskoro ćemo vam objasniti i šta je klasa.

Opcija –enableassertions neophodna je kod JDK 5.0 programa koji koriste *tvrdnje*. S obzirom na to da ćete tvrdnje koristiti čim naučite šta su one zapravo, bilo bi dobro da razvijete naviku stalnog korišćenja ove opcije. Ukoliko želite, opciju –enableassertions možete i da skratite u –ea.

Ukoliko želite da preskočite postojeću definiciju promenljive CLASSPATH, ova opcija je ista kao i kod kompajlera. I u kompajleru i u interpreteru možete da koristite skraćenicu – umesto –classpath upišite samo –cp. Evo kako bi ova komanda mogla da izgleda:

```
java -ea -cp . MyProgram
```

Kako bi izvršio vaš program, Java interpreter analizira i zatim izvršava bajtkod instrukcije. Implementacija Java Virtual Machine je identična na svim računarskim okruženjima koja podržavaju Javu, tako da možete biti sasvim sigurni u prenosivost svog programa. Kao što smo već rekli, vaš program će jednako dobro da radi na Unixovoj implementaciji Jave, kao i pod bilo kojim drugim operativnim sistemom (Microsoft Windows, Solaris, Linux, OS/2) koji podržava Javu. (Ipak vodite računa o tome koja je verzija Jave podržana. Neka okruženja kao što je, na primer, Macintosh malo kasne, tako da će se za njih implementacija Jave 2 pojaviti nešto kasnije nego za Windows ili Solaris.)

Izvršavanje apleta

Java kompajler koji je deo kompleta JDK, kompajlira i aplikacije i aplete. Međutim, aplet se ne izvršava na isti način kao i aplikacija. Da bi mogao da se pokrene, aplet se mora ugraditi u veb stranu. Nakon toga možete da ga pokrenete ili pomoću pretraživača u kome je uključena opcija Java 2 ili korišćenjem krajnje jednostavnog i ogoljenog pretraživača appletviewer koji se dobija uz JDK. Preporučujemo vam da za pregledanje apleta u fazi učenja koristite appletviewer. Ako vaš aplet ne bude funkcionisao, tačno ćete znati da li je problem u vašem kodu ili u integraciji sa pretraživačem.

Kompajlirani aplet koji ste ugradili u veb stranu i sačuvali u tekućem direktorijumu svog računara kao MyApplet.html možete da pokrenete sledećom komandom:

```
appletviewer MyApplet.html
```

Ali kako se aplet ubacuje u veb stranu?

Jezik HTML

Jezik za označavanje hiperteksta (*Hypertext Markup Language*, HTML) jeste jezik koji se koristi za definisanje veb strana. Kada veb stranu definišete kao HTML dokument, ona se čuva kao datoteka sa oznakom tipa .html. HTML dokument se sastoji od niza elemenata koji se identifikuju **tagovima**. Dokument počinje sa <html>, a završava se sa </html>. Ovi graničnici <html> i </html> predstavljaju tagove i svaki element HTML dokumenta nalaziće se unutar sličnog para tagova u uglastim zagradama. Tagovi elemenata ne razlikuju upotrebu malih i velikih slova, tako da ih možete koristiti bez ikakvih ograničenja, ali se prema konvenciji uvek koriste velika slova kako se ne bi mešali sa tekstom. Evo primera HTML dokumenta koji ima naslov i malo teksta:

```
<html>
 <head>
  <title>This is the title of the document</title>
 </head>
 <body>
```

```
    Ovde možete da stavite bilo kakav tekst. Telo dokumenta može da sadrži sve vrste
    ostalih HTML elemenata, uključujući i <B>Java aplete</B>. Skrećemo vam pažnju
    na to kako svaki element počinje tagom koji ga identifikuje, a završava se
    istim tim tagom kome je dodata jedna kosa crta. Parom tagova oko 'Java apleta'
    u prethodnoj rečenici postigli bismo ispisivanje teksta polucrnim slovima.
  </body>
</html>
```

Postoje dva elementa koji se mogu pojaviti direktno unutar elementa `<html>`. Kao što možete da vidite u prethodnom primeru, u pitanju su elementi `<head>` i `<body>`. Element `<head>` obezbeđuje informacije o dokumentu, a nije obavezno njegov deo. Tekst obuhvaćen tagovima `<title>`, koji je u ovom primeru u okviru elementa `<head>`, predstavljaće naslovnu liniju prozora u kom se prikazuje stranica.

Tagovi ostalih elemenata mogu se nalaziti u okviru elementa `<body>` i u pitanju su tagovi za naslove, liste, tabele, hiperveze ka drugim stranama i Java aplete. Postoje elementi za koje nije potrebno stavljati završni tag zato što se smatraju praznim. Takav je, na primer, kôd elementa `<hr/>` kojim se postavlja horizontalna linija celom širinom stranice. Ovim tagom možete da podelite stranu i odvojite jednu vrstu elemenata od druge.

Dodavanje apleta u HTML dokument

Za većinu parova tagova elemenata u početnom kodu možete da navedete **atribut elementa** koji definiše dodatne ili kvalifikujuće podatke o elementu. Ovako je Java aplet identifikovan u okviru taga `<applet>`. Sledi primer u kome ćete videti kako se Java aplet ugrađuje u HTML dokument:

```
<html>
 <head>
  <title> A Simple Program </title>
 </head>
 <body>
  <hr/>
  <applet code = "MyFirstApplet.class" width = 300 height = 200 >
  </applet>
  <hr/>
 </body>
</html>
```

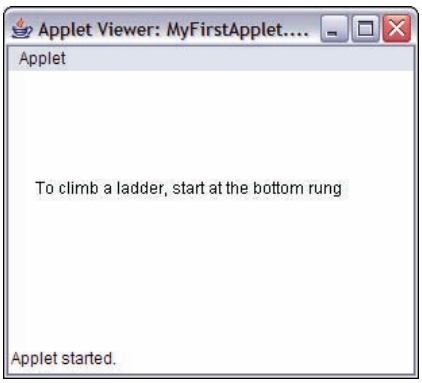
Dvema osenčenim linijama između tagova za horizontalne linije istakli smo to da se bajtkod apleta nalazi u okviru klase `MyFirstApplet.class`. Ime datoteke koja sadrži bajtkod apleta navodi se kao vrednost atributa `code` kontrolnog koda `<applet>`. Druga dva atributa – `width` i `height` definišu širinu i visinu dela ekrana koji će aplet da koristi prilikom svog izvršenja. Ovi parametri moraju se precizirati za svaki aplet. Evo kako izgleda Javin izvorni kôd jednostavnog apleta.

```
import javax.swing.JApplet;
import java.awt.Graphics;

public class MyFirstApplet extends JApplet {

 public void paint(Graphics g) {
  g.drawString("To climb a ladder, start at the bottom rung", 20, 90);
 }
}
```

Skrećemo vam pažnju na to da Java razlikuje upotrebu velikog i malog slova. Ukoliko reč `public` upišete sa velikim slovom `P`, program se neće kompajlirati. Kada pokrenete aplet, on će vam samo prikazati poruku na ekranu. Za sada nije mnogo bitno šta se dešava u pozadini prikazivanja ove poruke – ovim primerom smo samo želeli da vam ilustrujemo kako se aplet ugrađuje u HTML stranu. Ukoliko ovaj kôd kompajlirate i prethodnu specifikaciju HTML strane snimite kao `MyFirstApplet.html` u istom direktorijumu u kome je i kôd Java apleta, aplet ćete moći da pokrenete i pomoćnim programom `appletviewer`. Komanda je sledeća:

```
appletviewer MyFirstApplet.html
```

Nakon toga pojaviće se prozor poput ovoga na slici 1-2.



Slika 1-2

Ovaj prozor je prikazan u Internet Exploreru pod operativnim sistemom Windows XP i verovatno bi nešto drugačije izgledao u nekom drugom pretraživaču i pod nekim drugim operativnim sistemom. Budući da su visina i širina prozora u apletu navedeni u pikselima, njegove stvarne dimenzije zavisiće od rezolucije i veličine vašeg monitora.

S obzirom na to da JDK automatski instalira Javin dopunski modul, trebalo bi da ovaj primer funkcioniše bez ikakvih problema u Internet Exploreru. U slučaju da ne radi, iz menija Tools vašeg Internet Explorera izaberite stavku Internet Options. U kartici Advanced pronađite opciju „Use JRE v1.5.0 for <applet> (requires restart)" i proverite da li je izabrana. Za pretraživače Mozilla 1.*x* ili Netscape 7.*x* možete da u instalacionoj dokumentaciji kompleta JDK pronađete kako se ovaj Java dopunski modul naknadno uključuje.

Objektno orijentisano programiranje u Javi

Kao što smo na početku ovog poglavlja već istakli, Java je objektno orijentisan programski jezik. Prilikom korišćenja jezika koji nisu objektno orijentisani vi, u osnovi, rešenje svakog problema morate da iskažete pomoću brojeva i znakova – osnovnih tipova podataka kojima možete da manipulišete u jednom programskom jeziku. U objektno orijentisanim programskim jezicima stvari stoje malo drugačije. U njima i dalje možete da radite sa brojevima i znacima koji se nazivaju **osnovnim tipovima podataka**, ali i da definišete druge vrste entiteta koji su relevantni za konkretan problem koji želite da rešite. Dakle, u objektno orijentisanim jezicima za rešavanje problema koriste se entiteti ili objekti koji su povezani sa datim problemima. Ova činjenica će uticati na način strukturiranja programa, ali i na termine kojima se formuliše rešenje problema. Ukoliko se, primera radi, vaš problem tiče bejzbol igrača, u vašem Java programu će se po svemu sudeći naći i objekat po imenu `BaseballPlayer`; ako se bavite proizvodnjom voća u Kaliforniji, u vašem programu bi mogao da se nađe objekat `Oranges`.

Osim činjenice da imaju veoma razumnu strukturu, objektno orijentisani programi su obično mnogo razumljiviji.

U Javi je praktično sve definisano objektima. Ako se ranije niste bavili objektno orijentisanim programiranjem (ili jeste, ali ne u toj meri), ovo vam se na prvi pogled može učiniti zastrašujućim. Samo hrabro! Javini objekti su sasvim jednostavni – toliko jednostavni da ćemo odmah preći na njih kako biste što pre shvatili principe na kojima se ovaj programski jezik zasniva. U tom smislu, već od samog početka bićete na pravom putu.

Ipak, ovo ne znači da ćemo bilo šta preskočiti. Na ovom mestu ćemo se baviti, pre svega, osnovnim konceptima. Tamo gde je to potrebno, u tome će nam pomoći i segmenti Java koda. Sav kôd koji ćemo ovde koristiti u potpunosti ćemo objasniti u poglavljima koja slede. Sada samo pokušajte da razumete pojam objekata, a njihovi konkretni praktični detalji doći će kasnije na red.

Šta su zapravo objekti?

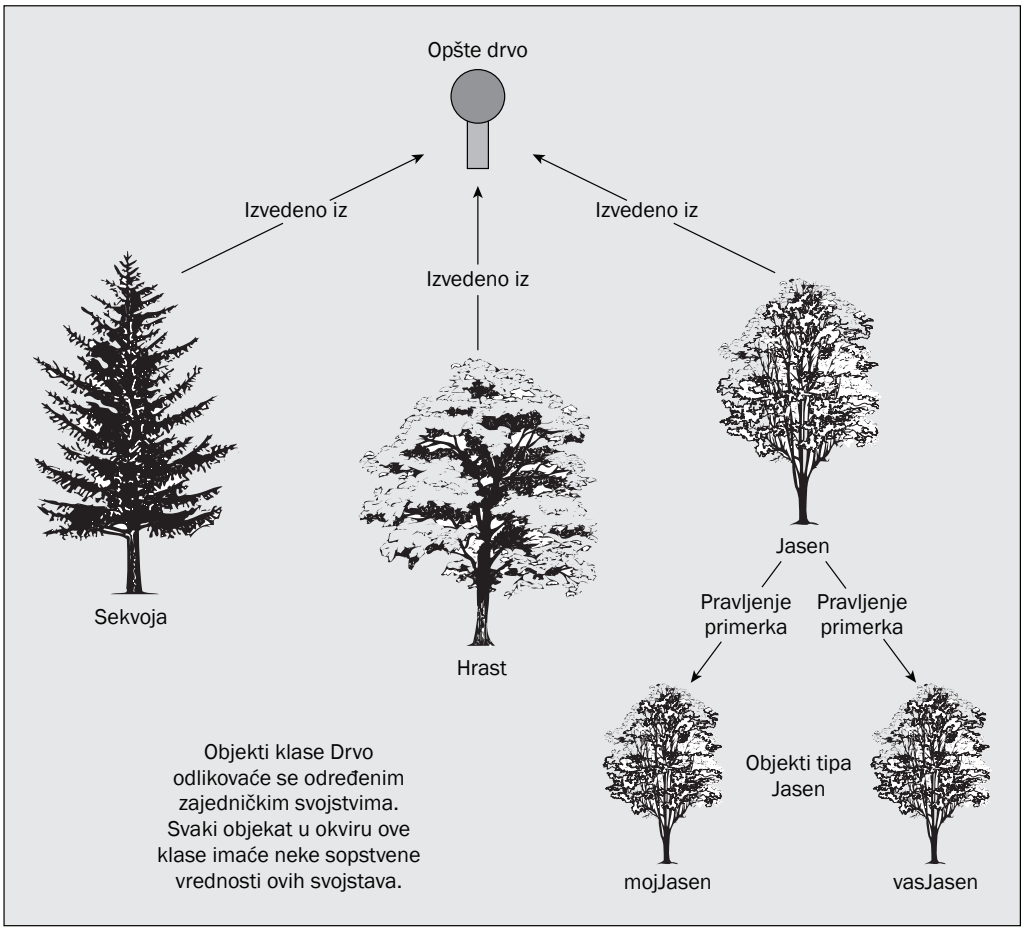
Objekt može da bude bilo šta. Objekti su svuda oko nas. Tako bi, na primer, `Drvo` mogao da predstavlja određenu klasu objekata – drveća u celini. Uopšteno govoreći, objekat `Drvo` je prilično apstraktan koncept – iako se svako drvo uklapa u ovu kategoriju, bolje je imati konkretnije objekte za vrste drveća. Tako bi hrast u mom dvorištu bio označen sa `mojHrast`, jasen u vašem dvorištu sa `onoProkletoDrvo`, a čuvena ogromna sekvoja u SAD, koja je pod zaštitom države, sa `generalSherman` – i svi oni predstavljaju konkretne primere drveta. Potklase objekta `Tree` u ovom kontekstu su `Hrast`, `Jasen` i `Sekvoja`. Skrećemo vam pažnju na to da već ovde lagano ulazimo u terminologiju – **klasa** je termin koji se odnosi na specifikaciju skupa objekata koji imaju zajednička svojstva. Na slici 1-3 prikazane su neke klase drveća, kao i njihovi međusobni odnosi.

Dakle, klasa je specifikacija ili shematski plan – u formi segmenta programskog koda – koja definiše kakva je struktura određene vrste objekta. Potklasa je klasa koja nasleđuje sva svojstva roditeljske klase, ali ima i neke posebnosti. Klase `Hrast` i `Jasen` koje pripadaju klasi `Drvo` moraju imati karakteristike ove opšte klase inače ne bi mogle da joj pripadaju. Međutim, svaka potklasa klase `Drvo`, kao što je, na primer, klasa `Hrast`, ima i neke posebnosti kojima se razlikuje od ostalih tipova klase `Drvo`.

Naravno, klasu ćete definisati tako da se uklopi u ono što želite da učinite sa nekim kontekstom u aplikaciji. Ovde nema isključivosti. U našem trivijalnom problemu specifikacija klase `Drvo` mogla bi da obuhvati samo ime vrste i njenu visinu. Ali, ukoliko biste se bavili botanikom, vaš problem bi mogao da zahteva postojanje mnogo složenije klase, ili skupa klasa, koje bi morale da obuhvate i veliki broj drugih osobina drveća.

Za svaki objekat koji će vaš program da upotrebi mora negde da postoji odgovarajuća definicija klase u koju se uklapaju objekti tog tipa. Ovo pravilo važi i u Javi, ali i u svim drugim objektno orijentisanim programskim jezicima. Koncepcija klasa u programiranju identična je sa klasifikovanjem stvari u stvarnom svetu. To je pogodan i ustanovljen način za grupisanje stvari.

Primerak klase jeste tehnički termin za postojeći objekat neke klase. Klasa `Jasen` predstavlja specifikaciju tipa objekta, a vaš objekat `vasJasen` napravljen je prema toj specifikaciji. Dakle, objekat `vasJasen` bio bi primerak klase `Jasen`. Čim definišete klasu, možete da kreirate objekte ili primerke date klase. Ova konstatacija dovodi nas do sledećeg bitnog pitanja. Po čemu se objekat jedne klase (recimo, `Jasen`) razlikuje od objekta koji pripada nekoj drugoj klasi (na primer, `Sekvoja`)? Drugim rečima, kojim se to informacijama definiše klasa?



Slika 1-3

Šta definiše klasu objekata?

Verovatno ste već sami pogodili kako glasi odgovor na ovo pitanje. Definicija klase identifikuje sve parametre koji definišu objekat date klase. Ovakva definicija nam je za sada dovoljna. Možda bi neko drugi definisao klasu većim ili manjim skupom parametara koji definišu istu vrstu objekta – sve zavisi od toga šta zapravo želite da uradite sa datom klasom. Vi odlučujete koje ćete aspekte objekata upotrebiti za definisanje date klase objekata, a birate ih u zavisnosti od vrste problema za koji su vam objekti te klase potrebni. Zamislimo sada neku konkretnu klasu objekata.

Kada biste, primera radi, definisali klasu `Hat`, u njenoj definiciji biste mogli da koristite samo dva parametra. Tip kape biste mogli da definišete nizom znakova – `"Fedora"` ili `"Bezbol kapa"`, a veličinu numeričkom vrednošću. Parametri koji definišu objekat neke klase nazivaju se **promenljive primerka**, **atributi** klase ili **polja** klase. Promenljive primerka mogu da budu osnovni tipovi podataka kao što su brojevi, ali isto tako i objekti drugih klasa. Primera radi, naziv objekta `Hat` mogao bi da bude definisan tipom `String` – gde klasa `String` definiše objekte koji se sastoje od niza znakova.

Naravno, postoje i mnoge druge osobine kojima biste mogli da upotpunite definiciju objekta `Hat` – atribut `boja` mogao bi da bude drugi niz znakova, na primer, `"Plava"`. Prilikom definisanja klase, opredeljujete se za onaj skup atributa koji ispunjava vaše zahteve. U terminologiji objektno orijentisanog programiranja, ovo se naziva **apstrahovanje podataka** zato što iz čitavog niza mogućnosti jednog tipičnog objekta vi izdvajate one atribute koje želite da koristite.

U Javi bi definicija klase `Hat` izgledala ovako:

```
class Hat {
 // Ovde se nalazi ono što detaljno definiše klasu.
 // Ovde bismo mogli da definišemo naziv kape, njenu veličinu,
 // možda boju i sve drugo što nam je potrebno.
}
```

Naziv klase dolazi iza reči `class`, a detalji njene definicije između velikih zagrada.

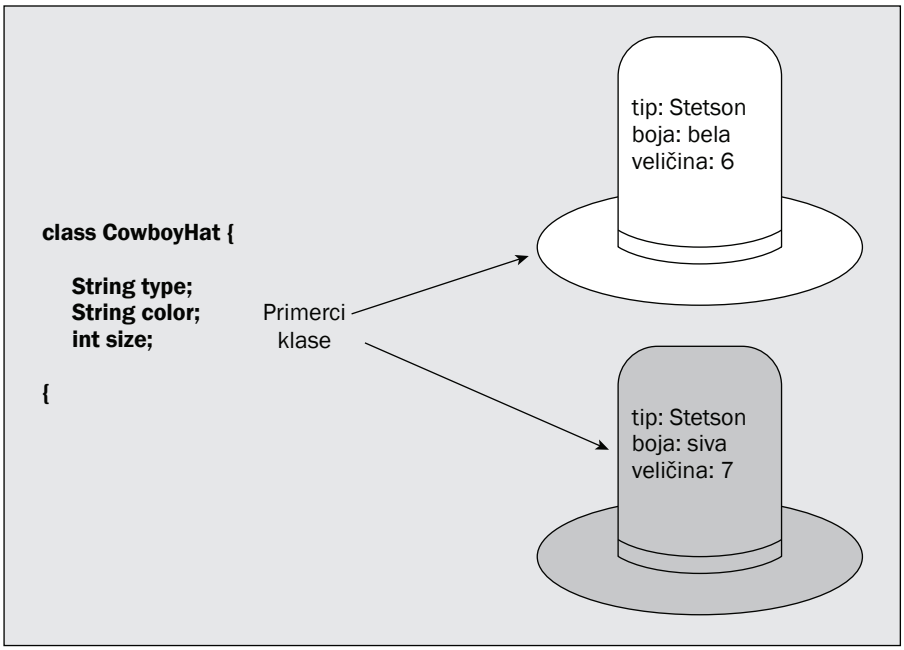
Zbog svog posebnog značenja, reč *class* u programskom jeziku Java naziva se *ključna reč* i njeno korišćenje rezervisano je samo za ovaj kontekst. U Javi postoji mnogo drugih ključnih reči sa kojima ćete se vremenom upoznati. Zapamtite da se nijedna od njih ne sme koristiti u neke druge svrhe.

Ovde nećemo ulaziti u detalje definisanja klase `Hat` zato što vam to trenutno nije neophodno. Redovi između zagrada u našem prethodnom primeru nisu kôd, već **komentari programa**, budući da počinju sa dve kose crte. Kompajler u Java programima ignoriše sve što se u nekom redu nalazi iza ove dve kose crte, tako da ih možete koristiti za ostavljanje objašnjenja. Uopšteno govoreći, što više korisnih komentara dodate u svoj program, tim bolje. U Javi postoje i drugi načini za pisanje komentara, o čemu ćemo govoriti u poglavlju 2.

Svaki objekat vaše klase imaće konkretan skup definisanih vrednosti koje ga karakterišu. Primera radi, mogli biste da imate objekat tipa `CowboyHat` koji bi bio definisan vrednostima `"Stetson"` za tip, `"Belo"` za boju i brojem `7` za veličinu (slika 1-4).

Iako su objekti tipa `CowboyHat` sa slike 1-4 definisani skupom od tri vrednosti za koje ne biste očekivali da se menjaju u datom primerku, uopšteno govoreći, vrednosti parametara koji definišu objekte ne moraju biti fiksirane. Bilo bi logično da su vrednosti `tip` i `veličina` datog objekta `CowboyHat` fiksirane budući da šeširi obično ne menjaju svoju veličinu – bar to ne čine kada je suvo. Ali, tu bi mogli da postoje i neki drugi atributi (slika 1-5).

Na primer, mogao bi da postoji i parametar `vlasnik` u kome bi bilo zapisano ime vlasnika. Ova vrednost bi onda mogla da se promeni kad god bi šešir prodajom, ili na neki drugi način, promenio vlasnika. Isto tako, mogli bismo da imamo i parametar `hatOn` koji bi nam govorio da li je vlasnik stavio šešir na glavu (vrednost `tačno`) ili nije (vrednost `netačno`).



Slika 1-4

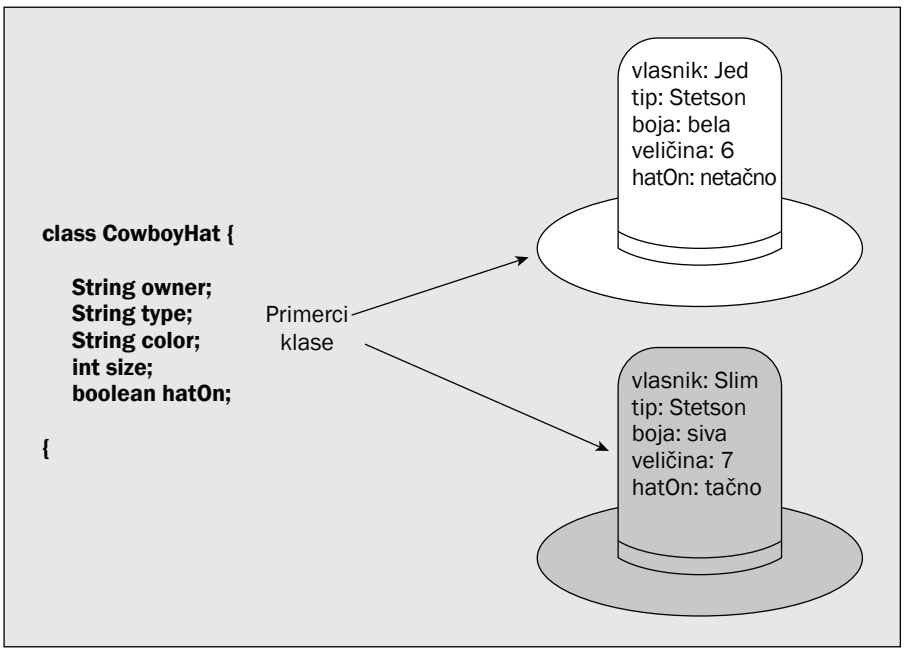
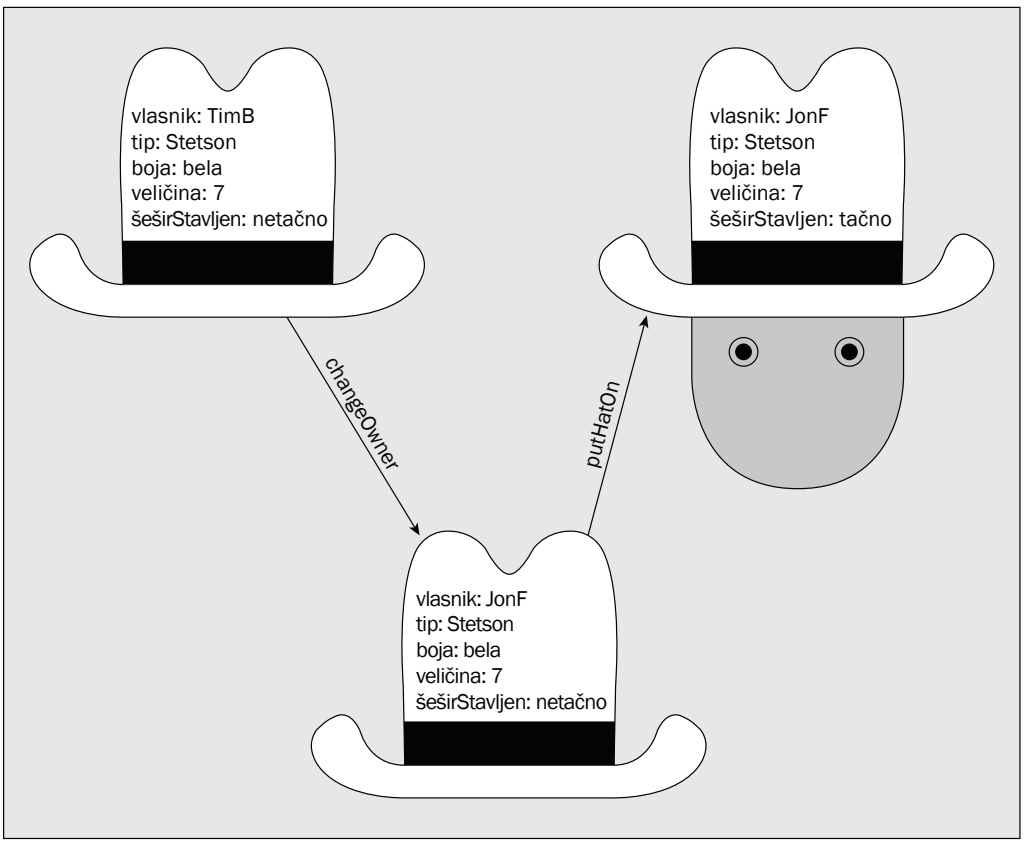


Figure 1-5

Operacije sa objektima

Suprotno onome što biste mogli da zaključite gledajući sliku 1-5, objekat neke klase nije samo skup različitih podataka. Osim parametara koji karakterišu neki objekat, u okviru klase je precizirano šta možete da učinite sa njenim objektom – odnosno, definisane su operacije koje je moguće izvesti sa objektima date klase. Naravno, da biste u programu imali bilo kakvu korist od objekata, najpre morate da odlučite na koji način želite da ih koristite. U tom smislu, važno je znati o kojoj je vrsti objekta reč, koje atribute on sadrži i šta vi nameravate sa njim da učinite.

U slučaju klase `CowboyHat` sa slike 1-5, mogli biste da imate operacije koje biste nazvali `putHatOn` i `takeHatOff`, čije bi značenje bilo u skladu sa njihovim nazivima i koje imaju smisla kod objekata `CowboyHat`. Ove operacije bi kod konkretnog objekta `CowboyHat` mogle da definišu vrednost `hatOn`. Da biste odredili da li je vaš `CowboyHat` stavljen ili nije, samo treba da proverite ovu vrednost. Isto tako, mogli biste imati i operaciju `changeOwner` kojom biste definisali promenljivu u okviru koje se evidentira ime aktuelnog vlasnika šešira. Na slici 1-6 sa objektom `CowboyHat` redom smo izveli dve operacije.



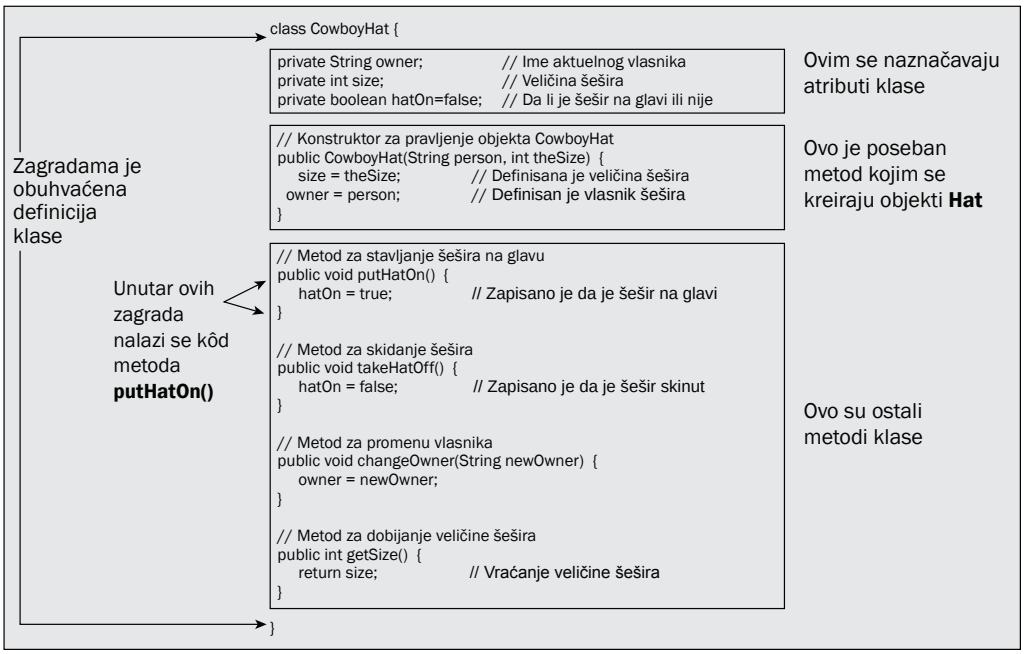
Slika 1-6

Naravno, za svaku vrstu objekta možete imati one operacije koje vi želite. Kada biste za objekte `Hat` želeli da dodate operaciju `ShootHoleIn`, ni to ne bi bio nikakav problem. Treba samo da definišete šta data operacija čini sa objektom.

Možda se u ovom trenutku pitate kako se definiše operacija za neku klasu objekata. Kao što ćete uskoro i videti, to se svodi na pravljenje segmenta koda po imenu **metod** koji se identifikuje imenom koje mu dodelite. Metodima možete da prosledite razne podatke – cele brojeve, brojeve u pokretnom zarezu, nizove znakova ili objekte klasa. Sve ove podatke obrađuje kôd datog metoda. Metod i sam može da kao rezultat vrati podatke. Izvođenje operacija sa objektima svodi se na *izvršavanje* metoda koji definiše te operacije.

Naravno, na određenu klasu možete da primenite samo one operacije koje su u okviru nje već definisane. To znači da upotrebljivost i fleksibilnost klase zavise od toga koliko ste je promišljeno definisali. U poglavlju 5 ćemo se vratiti na ovu temu.

Da biste bili u stanju da prepoznate definiciju klase, razmotrićemo kako izgleda jedna potpuna definicija. Kôd klase `CowboyHat` o kojoj smo već govorili mogao bi da se ilustruje kao na slici 1-7.



Slika 1-7

Ovaj kôd ćemo sačuvati kao datoteku po imenu `CowboyHat.java`. Ime datoteke koje sadrži definiciju klase uvek je isto kao ime klase, dok se oznakom tipa `.java` identifikuje Javin izvorni kôd.

Kôd za definiciju klase nalazi se unutar vitičastih zagrada koje idu iza identifikacije klase (slika 1-7). Kôd svakog metoda klase takođe se nalazi unutar zagrada. Klasa ima tri promenljive primerka – `owner`, `size` i `hatOn`, a ova poslednja se uvek inicijalizuje vrednošću `false`. Svaki objekat koji se kreira u skladu sa specifikacijom ove klase imaće svoju nezavisnu kopiju svake od ovih promenljivih. To znači da će svaki objekat imati sopstvene vrednosti za vlasnika, veličinu šešira i to da li je šešir na glavi ili nije. Da bi kôd bio kraći, u ovoj verziji klase smo malo ograničili parametar `type`.

Ključnom rečju `private` koja je primenjena na svaku promenljivu primerka osigurava se to da samo kôd u okvirima metoda klase može da pristupi ovim vrednostima direktno i promeni ih. Metodi klase takođe mogu da se navedu uz korišćenje ključne reči `private`. Sprečavanje pristupa nekim članovima klase spolja jeste veoma važna funkcija. Na ovaj način se unutrašnjost klase štiti od menjanja ili drugih vidova neispravnog korišćenja. Nekome ko iz nekog drugog programa koristi vašu klasu dozvolićete da pristupi samo onim delovima vaše klase za koje vi smatrate da je to uputno. Zahvaljujući tome, vi možete da izmenite unutrašnje funkcionisanje klase, a da se to ne odrazi na ostale programe koji tu klasu koriste. Možete da izmenite sve ono što ste unutar klase označili ključnom rečju `private`, pa čak i sam kôd unutar bilo kog javnog metoda. Jedino ne smete da menjate ime metoda i brojeve i tipove vrednosti koji se tom metodu predaju ili se od njega dobijaju.

Naša klasa `CowboyHat` ima pet metoda, što znači da sa objektom `CowboyHat` možete da učinite pet različitih stvari. Jedan od njih je i specijalni metod **konstruktor** koji kreira objekat `CowboyHat` – ovaj metod ima isto ime kao klasa. Svim onim što se nalazi unutar zagrada koje idu iza imena konstruktora, navode se podaci koji se predaju ovom metodu prilikom njegovog izvršavanja – odnosno, kreiranja objekta `CowboyHat`.

U praksi biste morali da definišete još nekoliko drugih metoda da bi ova klasa bila upotrebljiva. Primera radi, bio bi vam potreban i metod za poređenje `CowboyHat` objekata kako biste videli koji je od njih veći. Međutim, u ovom trenutku samo želimo da kod vas stvorimo predstavu o tome kako kôd izgleda. Detalji nam ovde nisu važni jer ćemo se na njih vratiti u poglavlju 5.

Programski iskazi u Javi

Kao što ste videli na primeru klase `CowboyHat`, kôd svakog metoda neke klase stoji unutar zagrada, a sastoji se od **programskih iskaza**. Znak tačka-zarez označava kraj svakog programskog iskaza. Ukoliko je potrebno, u Javi jedan iskaz može da obuhvati i nekoliko redova koda, jer se kraj svakog od njih određuje znakom tačka-zarez, a ne krajem reda. Evo jednog Javinog programskog iskaza:

```
hatOn = false;
```

Ako želite, ovo biste mogli da napišete i ovako:

```
hatOn =
     false;
```

U iskaze možete da ubacite i razmake i tabulatorske znake i oni se mogu prostirati kroz nekoliko redova čime se postiže bolja čitkost, naročito u jako dugim iskazima. Ipak, postoje i neka razumna ograničenja. Recimo, ne možete da stavite znak za razmak usred imena nekog primerka. Kada biste napisali `hat On`, kompajler bi to pročitao kao dve reči.

Enkapsuliranje

U ovom odeljku ćemo vam predstaviti još jedan termin kojim kasnije možete da impresionirate svoje prijatelje. U pitanju je **enkapsuliranje**. Taj termin se odnosi na postupak sakrivanja podataka i metoda unutar objekta. Ovo sakrivanje se postiže tako što se ove stavke u definiciji klase navedu uz ključnu reč `private`. U klasi `CowboyHat` promenljive primerka `owner`, `type` i `hatOn` su enkapsulirane. Njima je moguće pristupiti jedino putem metoda koji su definisani u okviru klase. Prema tome, vrednosti koje ove promenljive sadrže mogu da se izmene jedino pozivom metoda koji je za to zadužen. Mogućnost ovakvog enkapsuliranja članova klase veoma je bitna za bezbednost i integritet objekata klase. Možda ćete u nekom programu imati klasu sa podacima koji mogu da prime samo neke konkretne vrednosti. Sakrivanje podataka članova i obavezno korišćenje metoda za njihovo definisanje ili menjanje, pružiće vam garancije u pogledu valjanosti definisanih vrednosti.

Nešto ranije smo već pomenuli veliku prednost enkapsuliranja – mogućnost sakrivanja implementacije klase. Ograničavanje pristupa članovima klase pružiće vam slobodu da unutrašnjost date klase menjate, a da se to ne odražava na programe koji je koriste. Dok god su spoljne karakteristike metoda koji se pozivaju izvan klase neizmenjene, interni kôd možete da izmenite kako god vi to želite.

U konkretnom objektu – primerku klase `CowboyHat` enkapsulirane su promenljive `owner`, `size`, kao i `hatOn`, koje pokazuju status šešira. Spolja je moguće pristupiti samo konstruktoru i metodima `putHatOn()`, `takeHatOff()`, `changeOwner()` i `getSize()`.

Kad god u tekstu budemo pozivali na neke metode, koristićemo zagrade kako biste mogli da ih razlikujete od drugih pojmova koji imaju imena. Neke od primera ove vrste već ste videli u prethodnom pasusu. S obzirom na to da u definiciji i u programima metodi uvek imaju zagrade, sasvim je logično da ih na isti način navodimo i u tekstu.

Klase i tipovi podataka

U programiranju se u osnovi bavimo načinima na koje se razni tipovi podataka obrađuju ili transformišu. Budući da klase definišu tipove objekata sa kojima će program da radi, na definisanje klase možete da gledate isto kao i na definisanje tipa podataka. Prema tome, klasa `Hat` je tip podatka kao što je to i klasa `Tree` i svaka druga klasa koju ćete definisati. U Javi postoji i biblioteka standardnih klasa koje će vam pružiti čitav niz programerskih alatki i funkcija. Dakle, u najvećem broju slučajeva u vašim programima pisanim u Javi dešavaće se obrađivanje, manipulisanje i transformisanje objekata raznih klasa.

U Javi postoje i neki jednostavni tipovi podataka koji nisu klase, a nazivaju se **osnovni tipovi podataka**. O njima ćemo nešto detaljnije govoriti u narednom poglavlju, a sada ćemo reći samo toliko da su u pitanju tipovi za numeričke vrednosti kao što su, recimo, 99 ili 3,75, zatim

za jednostavne znake kao što su *A* ili *?*, ili za logičke vrednosti kao što su `true` ili `false`. U Javi postoje i klase koje odgovaraju svakom od osnovnih tipova podataka iz razloga o kojima ćemo govoriti nešto kasnije. Primera radi, u Javi postoji klasa `Integer` koja definiše objekte koji enkapsuliraju cele brojeve. Svaki entitet vaših Java programa koji ne pripada osnovnim tipovima biće objekat neke klase – ili klase koju ste vi definisali, ili one koja je bila deo Java okruženja ili nekog drugog programerskog paketa.

Klase i potklase

Mnogi skupovi objekata koje biste mogli da definišete u okviru klase mogli bi se dalje podeliti na specijalizovane podskupove koji su takođe predstavljeni klasama. Jedna od karakteristika Jave upravo je i ta mogućnost definisanja jedne klase kao specijalizovane verzije neke druge. Tako je, zapravo, i u stvarnosti. Uvek postoji više načina na koje biste mogli da podelite kolač – ili šumu. Klasa `Četinar` bi, primera radi, bila potklasa klase `Drvo`. To znači da klasa `Četinar` ima sve promenljive primerka i metode klase `Drvo`, ali i neke dodatne koje je čine posebnom. Klasa `Četinar` je **potklasa** klase `Drvo`, dok je klasa `Drvo` **natklasa** klase `Četinar`.

Zbog toga što smo prilikom njenog definisanja za polaznu tačku upotrebili klasu `Drvo`, za klasu `Četinar` ćemo reći da je **izvedena** iz klase `Drvo` i da od nje **nasleđuje** sve atribute.

Prednosti korišćenja objekata

Kao što smo već rekli, objektno orijentisani programi napisani su korišćenjem objekata koji su specifični za probleme o kojima je reč. Primera radi, u simulatoru flipera mogli biste da definišete i upotrebite objekte `Table`, `Ball`, `Flipper` i `Bumper`. Ovakva koncepcija je izuzetno pogodna zato što olakšava razvoj i proširivanje programa i čini njihov kôd razumljivijim. U okviru Jave postoji čitav niz standardnih klasa koje će vam pomoći u razvoju programa, a uvek možete da napravite i svoje opšte klase koje ćete koristiti u onim oblastima koje vas posebno interesuju.

S obzirom na to da u objektu postoje i metodi koji dejstvuju na njega i podaci koji ga definišu, greške u programiranju su mnogo ređe ukoliko se koriste objekti. Vaši objektno orijentisani Java programi bi morali da budu pouzdaniji od svojih ekvivalenata u proceduralnim programskim jezicima. Pravljenje objektno orijentisanih programa traje duže zato što moramo da osmislimo i napravimo klase, ali pisanje samog koda kod njih nekada ide čak i mnogo brže nego kod proceduralnih programskih jezika. Osim toga, objektno orijentisani programi se lakše održavaju i proširuju.

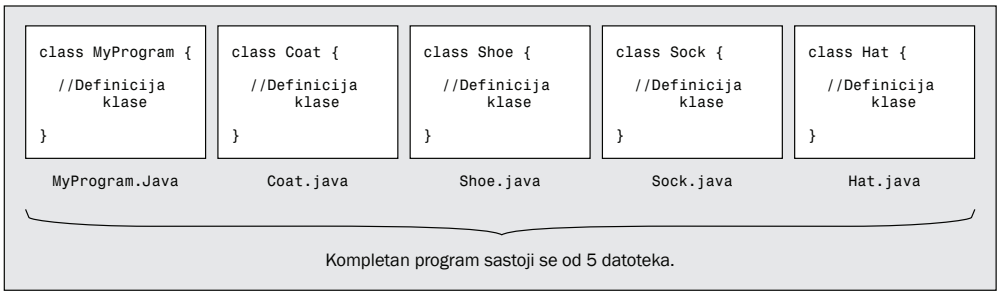
Struktura Java programa

Pogledajmo sada kakva je struktura programa napisanih u Javi:

- Program napisan u Javi uvek se sastoji od jedne ili više klasa.
- Kodovi svake klase obično se nalaze u posebnim datotekama, a svakoj od ovih datoteka morate dati isto ime kao i klasi koju njom definišete.
- Izvorna Javina datoteka mora imati oznaku tipa `.java`.

Prema tome, datoteka u kojoj je definisana klasa `Hat` zvaće se `Hat.java`, a datoteka u kojoj je definisana klasa `BaseballPlayer` – `BaseballPlayer.java`.

Na slici 1-8 ilustrovali smo tipičan program koji se sastoji od nekoliko datoteka.



Slika 1-8

Ovaj program nam je očigledno povezan sa odevanjem, jer četiri od pet klasa predstavljaju odevne predmete. Svaka izvorna datoteka sadrži definiciju klase i sve datoteke koje čine program čuvaju se u istom direktorijumu. U izvornim datotekama vaših programa nalazi se sav kôd koji ste vi napisali, ali to ne znači da je tu apsolutno sve što je ugrađeno u vaš program. U vašim programima biće i koda iz **standardne javina biblioteke klasa** i zato ćemo sada razmotriti o čemu je tu zapravo reč.

Javina biblioteka klasa

U Javi, biblioteka predstavlja skup klasa, najčešće srodnih mogućnosti, koje možete da koristite u svojim programima. Javina biblioteka klasa donosi vam čitav niz pogodnosti. Bez nekih od njih vaši programi ne bi ni funkcionisali, dok vam neke druge jednostavno značajno olakšavaju posao. S obzirom na sveobuhvatnost standardne biblioteke klasa, ovde nećemo previše ulaziti u detalje, ali ćete videti kako se primenjuju mnoge od njihovih mogućnosti.

Budući da biblioteka klasa u osnovi predstavlja skup klasa, ona se i čuva u vidu skupova datoteka od kojih svaka sadrži definiciju jedne klase. Ove klase su grupisane u povezane skupove koji se nazivaju **paketi** i svaki od njih se čuva u posebnom direktorijumu. Jedna klasa može da pristupi bilo kojoj drugoj klasi iz istog paketa, dok joj klase iz drugih paketa mogu, ali i ne moraju biti dostupne. Na ovu temu ćemo se vratiti u poglavlju 5.

Ime paketa bazira se na putanji do direktorijuma u kome se čuvaju klase koje datom paketu pripadaju. Primera radi, klase iz paketa `java.lang` čuvaju se u direktorijumu `java\lang` (ili, `java/lang` pod Unixom). Ova putanja povezana je sa konkretnim direktorijumom koji je automatski poznat Java runtime okruženju koje izvršava vaš kôd. Uvek možete da napravite i vlastite pakete u kojima će se nalaziti vaše klase koje želite da koristite u nekom drugom kontekstu ili one koje su na neki način povezane.

U okviru kompleta JDK postoji sve veći broj standardnih paketa – verovatno više od 100. Evo paketa koje ćete najčešće koristiti:

Naziv paketa	Opis
java.lang	Ove klase podržavaju osnovne jezičke funkcije, rukovanje nizovima i znakovnim nizovima. Klase iz ovog paketa, prema podrazumevanim parametrima, uvek stoje na raspolaganju vašim programima zato što se uvek automatski učitavaju sa njima.
java.io	Klase za ulazne i izlazne operacije sa podacima.
java.util	U ovom paketu nalaze se razne pomoćne klase kao što su, na primer, klase za rukovanje podacima koji se nalaze u kolekcijama, tj. grupama podataka.
javax.swing	Ove klase obezbeđuju komponente za pravljenje grafičkih korisničkih interfejsa (GUI) koje su fleksibilne i lake za korišćenje. Nazivaju se i Swing komponente.
java.awt	Klase iz ovog paketa obezbeđuju originalne GUI komponente (JDK 1.1), kao i neku najosnovniju podršku neophodnu Swing komponentama.
java.awt.geom	Ovim klasama definišu se dvodimenzionalni geometrijski oblici.
java.awt.event	Klase iz ovog paketa koriste se u implementaciji aplikacija sa prozorima za rukovanje događajima u okviru vaših programa. Događaji su, na primer, pomeranje miša, pritisak njegovog levog tastera ili izbor neke stavke iz nekog menija.

Kao što smo već napomenuli, u svojim programima prema podrazumevanim parametrima možete da koristite bilo koju klasu iz paketa `java.lang`. Za korišćenje klasa iz drugih paketa obično se koriste iskazi `import` u kojima se navode imena klasa koje su vam potrebne. To znači da klase možete da referencirate jednostavnim navođenjem njihovih imena. Kada ne biste koristili iskaz `import`, prilikom svakog korišćenja neke klase morali biste da navedete njeno puno kvalifikovano ime. Kao što ćete uskoro i videti, puno kvalifikovano ime obuhvata osnovno ime klase, kao i ime paketa u kome se ta klasa nalazi. Korišćenje punih naziva klasa bi vaš program učinilo nezgrapnim i svakako manje čitljivim. Pisanje takvih programa bi vam bilo osetno teže.

Pomoću iskaza `import` u svoj program možete da uvezete jednu ili sve klase nekog paketa. Dva iskaza `import`, koje ste videli na početku koda apleta koji smo vam pokazali na početku poglavlja, predstavljaju primere uvoza jedne klase. Prvi ovakav iskaz glasio je:

```
import javax.swing.JApplet;
```

Ovim iskazom uvozi se klasa po imenu `JApplet` koja je definisana u okviru paketa `javax.swing`. Formalno, ime ove klase nije `JApplet` – njeno puno kvalifikovano ime je `javax.swing.JApplet`. Nekvalifikovana imena možete da koristite samo kada u svoj program uvozite klasu ili ceo paket u kome se ona nalazi. Klase iz paketa mogu i da se referenciraju (bez uvoženja), ali se tada koristi puno kvalifikovano ime – u ovom slučaju, `javax.swing.JApplet`. Ukoliko želite, ovo možete da isprobate na datom apletu. Samo izbrišite dva iskaza `import` iz datoteke i u programu upotrebite puna kvalifikovana imena klasa. Kada ponovo kompajlirate ovaj program, trebalo bi da funkcioniše isto kao i ranije. Dakle, puno kvalifikovano ime klase čini ime paketa u kome je data klasa definisana iza koga slede tačka i ime koje je dato klasi u njenoj definiciji.

Iskazom koji sledi mogli biste da uvezete imena svih klasa u okviru paketa `java.swing`:

```
import javax.swing.*;
```

Zvezdicom koja se nalazi na kraju iskaza navodi se da treba uvesti sve klase. Kompajliranje vašeg koda biće brže ukoliko uvezete samo one klase koje se u kodu koriste. Ali, ako koristite mnogo klasa nekog paketa, možda će vam se više isplatiti da uvezete ceo taj paket. Time ćete sebi uštedeti trud oko kucanja velikog broja iskaza za uvoženje. Da bismo broj redova kodova sveli na minimum, mi smo se u ovoj knjizi upravo držali tog pristupa. Ipak, skrećemo vam pažnju i na to da uvoženje svih klasa nekog paketa donosi i određene rizike. Vrlo lako se može dogoditi da u njima postoje klase sa identičnim imenima koja ste vi dali svojim klasama, zbog čega bi u kompajliranju koda nastali veliki problemi.

U poglavlju 5 ćemo vam još detaljnije prikazati korišćenje iskaza import, a videćete i kako se prave i koriste. Osim toga, u ostatku knjige ćete vrlo često biti u situaciji da koristite standardne pakete klasa.

Kao što smo već rekli, standardne klase na vašem disku nisu sačuvane kao direktorijumi ili datoteke. One su upakovane u jednu komprimovanu datoteku po imenu `rt.jar` koja se nalazi u direktorijumu `jre/lib`. Ovaj direktorijum se automatski pravi prilikom instaliranja kompleta JDL. Datoteka .jar je **J**avina **ar**hiva – komprimovana arhiva sa Javinim klasama. Standardne klase koje će biti potrebne vašim programima učitavaju se automatski iz datoteke `rt.jar`, tako da nećete imati nikakav neposredan kontakt sa njom.

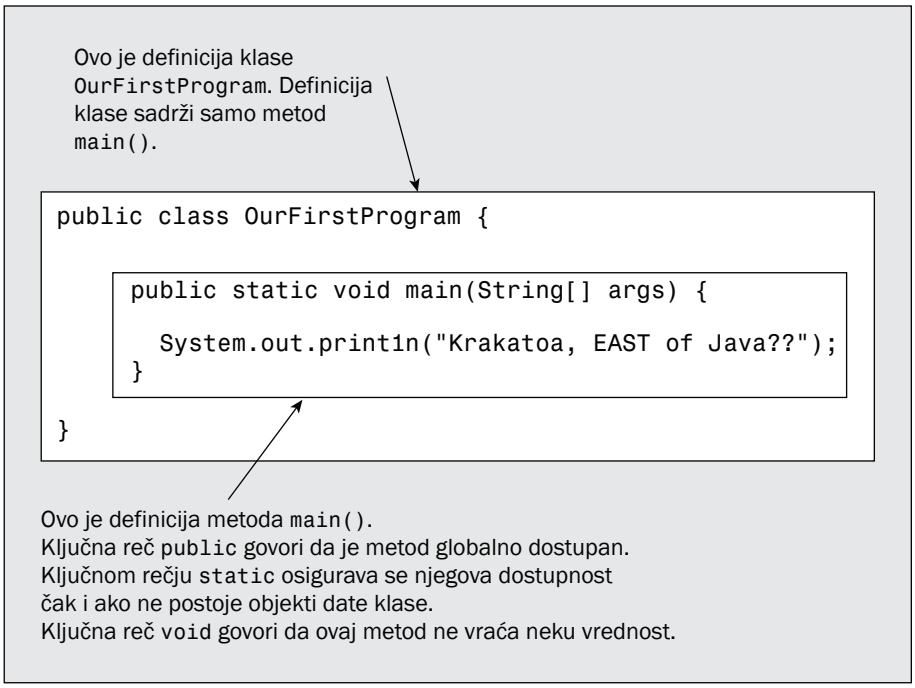
Java aplikacije

U svakoj Java aplikaciji postoji klasa koja definiše metod po imenu `main()`. Ova klasa imaće ime koje ćete koristiti kao argument za Java interpreter prilikom pokretanja aplikacije. Klasu možete da nazovete kako god želite, ali metod koji se prvi izvršava u nekoj aplikaciji uvek se naziva `main()`. Nakon pokretanja Java aplikacije, metod `main()` obično prouzrokuje izvršenje metoda koji pripadaju drugim klasama, dok se najjednostavnija moguća Java aplikacija sastoji samo od jedne klase koja ima samo metod `main()`. Kao što ćete videti u produžetku, metod `main()` koristi se u konkretnom i nepromenljivom obliku. Ukoliko se odstupi od ovog oblika, Javin interpreter neće prepoznati ovaj metod kao metod od koga počinje izvršenje.

Na primeru jednog Javinog programa možemo videti kako ovaj metod funkcioniše. Kôd programa treba da upišete u nekom editoru za običan tekst, ili to učinite u okviru editora svog Java razvojnog sistema. Kada upišete sav kôd, sačuvajte ga kao datoteku pod onim imenom koje ste koristili za klasu i sa oznakom tipa `.java`. U ovom primeru ime datoteke glasiće `OurFirstProgram.java`. Kôd ovog programa prikazan je na slici 1-9.

Program se sastoji od definicije klase koju smo nazvali `OurFirstProgram`. U definiciji klase postoji samo jedan metod – metod `main()`. Prvi red definicije metoda `main()` uvek ima sledeći oblik:

```
public static void main(String[] args)
```



Slika 1-9

Kôd ovog metoda stoji između para velikih zagrada. Ova verzija metoda ima samo jedan izvršivi iskaz:

```
System.out.println("Krakatoa, EAST of Java??");
```

Šta, u stvari, ovaj iskaz čini? Analizirajmo ga sleva nadesno.

- `System` je ime standardne klase koja sadrži objekte koji enkapsuliraju U/I uređaje vašeg sistema – tastaturu za ulaz komandne linije i ekran za izlaz komandne linije. Klasa `System` se nalazi u okviru paketa `java.lang`, tako da joj uvek možete pristupiti korišćenjem prostog imena `System`.
- Objekat `out` u okviru klase `System` predstavlja standardni izlazni tok – to je komandna linija na vašem ekranu. Član `out`, zapravo, predstavlja posebnu vrstu člana klase `System`. Poput metoda `main()` naše klase `OurFirstProgram`, i on je statičan. To znači da `out` postoji čak i kada ne postoje objekti tipa `System` (nešto više o ovome u poglavljima koja slede). Korišćenjem imena klase `System` iza kojega stoji tačka, a zatim ime metoda `out` – `System.out` – referencira se član `out`.
- Segment u desnom kraju iskaza, `println("Krakatoa, EAST of Java??")`, poziva metod `println()` koji pripada objektu `out`, što dovodi do prikazivanja teksta na vašem ekranu – onog teksta koji se nalazi između zagrada kod metoda `println`. Ovim smo prikazali jedan od načina za pozivanje metoda klase – korišćenje imena objekta iza koga sledi tačka i ime metoda. Ono što se nalazi između zagrada koje slede nakon imena metoda

jesu informacije koje se prosleđuju metodu prilikom njegovog izvršenja. Kao što smo već rekli, za `println()` to je tekst za koji želimo da se prikaže u okviru komandne linije.

Da bi sve bilo kompletno, ključne reči `public`, `static` *i* `void` *koje se pojavljuju u okviru definicije metoda ukratko su objašnjene u komentarima programskog koda. Ako vam i dalje nisu najjasniji, ne treba da brinete zato što ćemo ih mnogo detaljnije opisati u poglavlju 5.*

Ovaj program možete da kompajlirate u JDK kompajleru korišćenjem komande:

```
javac –source 5 OurFirstProgram.java
```

Isto to možete da učinite i navođenjem opcije `–classpath`:

```
javac –classpath . OurFirstProgram.java
```

Ukoliko se program nije kompajlirao, negde postoji greška. Evo mogućih izvora problema:

- Zaboravili ste da u okviru promenljive okruženja `PATH` navedete putanju do direktorijuma `jdk1.5.0\bin` ili to niste učinili tačno. Zbog toga vaš operativni sistem nije uspeo da pronađe `javac` kompajler koji se nalazi u tom direktorijumu.
- Pogrešili ste prilikom kucanja koda programa. Podsećamo vas da se u Javi razlikuju mala i velika slova, što znači da `OurfirstProgram` nije isto što i `OurFirstProgram`. Pored toga, u nazivu klase ne sme biti razmaka. Kada ustanovi grešku, kompajler najčešće odredi i broj reda u kome je greška utvrđena. Posebno vodite računa da ne pomešate broj `0` i malo slovo `o`, ili broj `1` i malo slovo `l`. Znaci kao što su tačke, zarezi ili tačke-zarezi jako su bitni za kôd i moraju se uvek nalaziti na pravim mestima. Male `()`, srednje `[]` i velike zagrade `{}` uvek idu u paru i ne mogu se međusobno zamenjivati.
- Ime izvorne datoteke mora da se poklapa sa imenom klase. I najmanja razlika između njihovih imena rezultiraće greškom. Ova datoteka mora da ima oznaku tipa `.java`.

Kada se program kompajlira, možete da ga pokrenete komandom:

```
java –ea OurFirstProgram
```

Opcija `–ea` nije neophodna zato što u ovom programu nema tvrdnji, ali je bolje da steknete naviku da ih stavljate u program kako to ne biste zaboravili kada bude bilo bitno. Ukoliko je potrebno, možete da navedete i opciju `–classpath`:

```
java –ea –classpath . OurFirstProgram
```

Ukoliko pretpostavimo da je izvorna datoteka uspešno kompajlirana i da ste naveli putanju do direktorijuma `jdk1.5.0\bin`, nemogućnost izvršavanja programa vrlo često je posledica tipografskih grešaka u imenu klase. Još jedan od mogućih uzroka greške bio bi da umesto imena klase `OurFirstProgram` napišete `OurFirstProgram.class`.

Kada ga pokrenete, program će na ekranu ispisati sledeći tekst:

```
Krakatoa, EAST of Java??
```

Java i Unicode

Programiranje podrške za jezike u kojima postoje znaci koji se razlikuju od latiničnog skupa znakova uvek je predstavljalo veliki problem. Postoji mnogo 8-bitnih skupova znakova koji su definisani za razne svetske jezike, ali ukoliko želite da u istom kontekstu kombinujete latinične i ćirilične znakove, situacija postaje prilično komplikovana. Kada biste želeli da tu dodate i podršku za japanski jezik, to jednostavno ne bi bilo moguće zato što vam u okviru 8-bitnog skupa znakova na raspolaganju stoji svega 256 različitih kodova, što znači da ne biste imali dovoljno kodnih znakova. Unicode je standardni skup znakova koji je trebalo da omogući kodiranje znakova neophodnih za većinu svetskih jezika. U njemu se znaci predstavljaju 16-bitnim kodom (to znači da svaki znak zauzima 2 bajta) i tih 16 bitova omogućava do 65 535 kodnih znakova različitih od nule. Ovako veliki broj raspoloživih kodnih znakova dozvoljava da se svakom značajnijem nacionalnom skupu znakova dodeli vlastiti skup kodova. Jedan takav skup znakova je, primera radi, Kanji koji se koristi u japanskom jeziku i koji zahteva hiljade kodnih znakova. To, međutim, nije sve. Unicode podržava tri forme kodiranja koje omogućavaju predstavljanje miliona dodatnih znakova.

Kao što ćete videti u poglavlju 2, Javin izvorni kôd je u Unicode znacima. Svi komentari, identifikatori (drugim rečima, imena – pročitajte poglavlje 2) i literali znakova i znakovnih nizova mogu da koriste sve znake Unicode skupa koji predstavljaju slova. Java i interno podržava Unicode za predstavljanje znakova i nizova, čime je dobijen radni okvir za sveobuhvatnu međunarodnu podršku u programima. Uobičajeni ASCII skup koji vam je, verovatno, već poznat poklapa se sa prvih 128 znakova Unicode skupa. Ako se zanemari to da na svaki znak obično odlaze 2 bajta, vi u potpunosti možete da ignorišete činjenicu da radite sa Unicode znacima osim, naravno, ukoliko u startu želite da napravite aplikaciju koja podržava više svetskih jezika.

Rekli smo da na svaki Unicode znak obično odlaze dva bajta zato što Java podržava Unicode 4.0 koji omogućava i korišćenje 32-bitnih znakova po imenu **surogati**. Ako mislite da je skup od 65 535 znakova koje možete da predstavite sa 16 bitova dovoljan, varate se. U jezicima Dalekog istoka kao što su japanski, korejski ili kineski postoji više od 70 000 ideograma, tako da se surogati koriste za predstavljanje znakova koji ne postoje u osnovnom višejezičnom skupu koji se definiše 16-bitnim znacima.

Zaključak

U ovom poglavlju prikazali smo vam osnovne karakteristike programskog jezika Java. Videli ste kako se u Javi postiže prenosivost, a zatim smo vam predstavili i osnovne elemente objektno orijentisanog programiranja. Ukoliko vam nešto o čemu smo do sada pričali nije baš najjasnije, ne brinite. Sve što smo pominjali u ovom poglavlju ponovićemo znatno detaljnije u nastavku knjige.

U ovom poglavlju naučili ste sledeće:

- Java apleti su programi koji se ugrađuju u HTML dokumente, dok su Java aplikacije zasebni programi. Java aplikacije mogu biti programi konzole koji podržavaju isključivo tekstualni izlaz na ekranu, ali i aplikacije sa prozorima i grafičkim korisničkim interfejsom.
- Programi napisani u Javi su po svojoj suštini objektno orijentisani.
- Java izvorni kôd čuva se u vidu datoteka sa oznakom tipa `.java`.

- Programi napisani u Javi kompajliraju se u bajtkod koji predstavlja instrukcije za komponentu Java Virtual Machine, koja je identična na svim računarima na kojima je implementirana, čime se obezbeđuje prenosivost Java programa.
- Java objektni kôd čuva se u vidu datoteka sa oznakom tipa `.class`.
- Programe napisane u Javi izvršava Java interpreter koji analizira bajtkodove i izvršava operacije koje su u njima naznačene.
- Okruženje Java Development Kit (JDK) podržava kompilaciju i izvršenje Java aplikacija i apleta.

Izvori

Izvorni kôd za primere iz ove knjige možete da preuzmete sa adrese `http://www.wrox.com`.

Osim samog izvornog koda, u nekim primerima se preuzimaju i pomoćne datoteke kao što su, na primer, `.gif` datoteke sa ikonama. Osim toga, na ovoj lokaciji pronaći ćete i rešenja vežbanja koja se nalaze na kraju većine poglavlja.