1 SEDMICA

Programski jezik Java

1. Osnovne napomene o programskom jeziku Java
2. Osnove programiranja
3. Rad sa objektima
4. Liste, programska logika i programske petlje
5. Kreiranje klasa i metoda
6. Paketi, interfejsi i druge funkcionalnosti klasa
7. Izuzeci, potvrde i niti

Dan 1

Osnovne napomene o programskom jeziku Java

ONO ŠTO SE ŽELELO POSTIĆI PROGRAMSKIM JEZIKOM JAVA, A ŠTO JE NAJVEĆIM DELOM I OSTVARENO, PREDSTAVLJA POVEZIVANJE VELIKOG BROJA RAZLIČITIH DOMENA, TAKO DA SE OMOGUĆAVA FUNKCIONISANJE APLIKACIONOG SERVERA I FUNKCIONISANJE MOBILNOG TELEFONA, KREIRANJE NAUČNIH PROGRAMA, RAZVOJ SOFTVERA, IMPLEMENTACIJA MEĐUPLANETARNE NAVIGACIJE I JOŠ MNOGO ŠTOŠTA...

— JAMES GOSLING, KREATOR PROGRAMSKOG JEZIKA JAVA, U INTERVJUU ZA POTREBE WEB STRANICE SEARCHWEBSERVICES.COM

Kada je 1995. godine "Sun Microsystems" prvi put javno predstavio programski jezik Java, to je bio inventivan alat za Web, koji je imao puno potencijalnih primena.

Reč "potencijal" je kompliment koji ima vrlo ograničen rok upotrebe. Pre ili kasnije, potencijal mora da se realizuje, ili se umesto te reči upotrebljavaju neke druge, kao što su "razočarenje", "škart" i "najveće razočarenje".

Ukoliko budete želeli da razvijete svoje veštine u dvadesetjednodnevnom podučavanju u okviru petog izdanja knjige "Naučite Javu 6 za 21 dan", bićete u dobroj poziciji da prosudite da li je ovaj programski jezik ispunio očekivanja iz prethodne dekade.

Osim toga, postaćete Java programer sa dosta potencijala.

Programski jezik Java

U sedmoj verziji programski jezik Java je ispunio očekivanja u vezi sa njegovim pojavljivanjem. Više od 3,5 miliona programera je naučilo ovaj programski jezik, koji se koristi na mestima kao što su NASA, IBM i Kaiser Permanente i u okviru Apache projekta. Ovaj programski jezik postao je standardan u velikom broju departmana u vezi sa računarskom tehnikom širom sveta. U početku je korišćen za kreiranje jednostavnih programa u okviru web stranica, a danas se primenjuje u velikom broju slučajeva, između ostalog, i za

- web servere
- relacione baze podataka
- orbitalne teleskope
- personalne digitalnie asistenate
- mobilne telefone

Iako je programski jezik Java i dalje veoma koristan za web programere koji pokušavaju da "ožive" svoje stranice i kreiraju web aplikacije, on danas ima mnogo širu primenu. Java je danas veoma popularan programski jezik opšte namene.

Istorija programskog jezika

Priča o razvoju programskog jezika Java je danas dosta poznata. James Gosling i drugi projektanti u kompaniji "Sun" su polovinom devedesetih godina prošlog veka bili angažovani u realizaciji projekta posvećenog razvoju interaktivne televizije. Tada je Gosling bio prilično frustriran što koriste C++, objektno-orijentisani programski jezik, koji je razvio Bjarne Stroustrup u AT&T Bell laboratorijama 10 godina ranije, kao proširenje programskog jezika C.

Gosling je vredno radio i kreirao novi programski jezik, koji je bio pogodan za projekat na kome je radio i u okviru koga je eliminisao i neke elemente programskog jezika C++, koji su ga "izluđivali".

Pokušaj razvoja interaktivne televizije u kompaniji "Sun" je propao, ali je rezultat razvoja novog programskog jezika mogao da se primeni na medijumu koji je postajao popularan u to doba - na Webu.

"Sun" je objavio Javu u jesen 1995. godine. Iako je najveći broj funkcionalnosti ovog jezika bio mnogo jednostavniji nego u programskom jeziku C++ (što je slučaj i danas), Java programi, koji su se nazivali *apleti*, mogli su da se izvršavaju kao deo web stranica u Netscape Navigator čitaču weba.

Ova funkcionalnost (prvo interaktivno programiranje raspoloživo na Webu) pomoglo je u reklamiranju Jave i privuklo je nekoliko stotina hiljada programera u prvih šest meseci postojanja novog programskog jezika.

Čak i nakon što Java više nije bila nov programski jezik, korist od primene ovog programskog jezika je postala potpuno jasna, a programeri su i dalje bili privučeni njome. Danas postoji više profesionalnih programera koji koriste programski jezik Java nego onih koji koriste programski jezik C++.

Uvod u programski jezik Java

Java je objektno-orijentisani, nezavisan od platforme, bezbedan programski jezik, koji je projektovan tako da ga je jednostavnije naučiti od C++-a, a teže zloupotrebiti od C-a i C++-a.

Objektno-orijentisano programiranje (OOP) je metodologija razvoja softvera u kojoj se program konceptualizuje pomoću grupe objekata koji zajedno funkcionišu. Objekti se kreiraju korišćenjem šablona koji se nazivaju klase i sadrže podatke i naredbe koje su neophodne za korišćenje tih podataka. Programski jezik Java je u potpunosti objektno-orijentisan, što ćete imati priliku da vidite u toku ove lekcije kada budete kreirali svoju prvu klasu i koristili je za kreiranje objekta.

Platformska nezavisnost je mogućnost programa da se izvršava bez modifikacija u okviru različitih radnih okruženja. Java programi se prevode u format koji se naziva *bajtkod*, koji u okviru bilo kog operativnog sistema može da izvršava bilo koji softver ili uređaj koji sadrži interpretator programskog jezika Java. Vi možete na Windows Vista mašini kreirati Java program koji će se izvršavati na Linux web serveru, na Apple Mac mašini koja koristi OS X operativni sistem ili na Palm personalnom digitalnom asistentu. Ukoliko platforma sadrži interpretator programskog jezika Java, može da se koristi bajtkod.

Java je projektovana tako da bude jednostavnija od programskog jezika C++, i to pre svega zbog sledećeg:

- U okviru programskog jezika Java automatski se obavlja alokacija i dealokacija memorije, čime se programeri oslobađaju dosadnog i složenog posla.
- Java ne sadrži pokazivače, moćnu funkcionalnost koju koriste prevashodno iskusni programeri, a prilikom čijeg korišćenja može veoma lako doći do grešaka.
- Java implementira samo koncept jednostrukog nasleđivanja prilikom objektno-orijentisanog programiranja.
- Nedostatak pokazivača i prisustvo automatskog upravljanja memorijom su dva ključna elementa bezbednosti u slučaju programskog jezika Java. Detaljniji prikaz istorije programskog jezika Java i prednosti korišćenja ovog jezika možete pronaći u članku "Izbor Java programskog jezika" na pratećem CD-u.

Izbor razvojnog okruženja

Pošto ste upoznali programski jezik Java, pravi je trenutak da neke koncepte praktično isprobate i kreirate svoj prvi Java program.

Ukoliko budete obradili sve lekcije u ovoj knjizi, prilično dobro ćete upoznati sve mogućnosti programskog jezika Java, uključujući i primenu grafike, rad sa datotekama, razvoj web aplikacija, obradu Extensible Markup Language (XML) datoteka i povezivanje sa bazom podataka. Kreiraćete programe koji se izvršavaju u okviru web stranica i one koji se izvršavaju na personalnim računarima, web serverima i drugim okruženjima.

Da biste mogli da počnete da programirate, neophodno je da na svom računaru imate odgovarajući softver, koji možete koristiti za ažuriranje, prevođenje i izvršavanje Java programa, i to onaj softver koji omogućava razvoj programa korišćenjem najnovije verzije programskog jezika - Jave 6.

Nekoliko veoma popularnih razvojnih okruženja podržavaju Java 6 programski jezik, kao što su, između ostalih, JBuilder, IntelliJ IDEA i Eclipse.

Sva prethodno navedena okruženja preporučuju Java programeri, ali ukoliko učite istovremeno korišćenje ovih alata i programski jezik Java, to može biti prilično naporan posao. Najveći broj integrisanih razvojnih okruženja namenjen je, pre svega, veoma iskusnim programerima, koji žele da budu što produktivniji, a ne ljudima koji upoznaju novi programski jezik.

Zbog toga, ukoliko niste dovoljno upoznali razvojno okruženje pre nego što ste nabavili ovu knjigu, trebalo bi da koristite najjednostavniji alat za razvoj Java aplikacija - Java Development Kit, koji je besplatan i koji možete preuzeti sa Java web stranice kompanije "Sun" (http://java.sun.com).

Uvek kada kompanija "Sun" objavi novu verziju programskog jezika Java, objavljuje se na web stranici i odgovarajući alat za tu verziju. Trenutno izdanje je Java Development Kit Version 6.

Jednostavnosti radi, programski jezik o kome govorimo u ovoj knjizi nazivaćemo Java, a alat za razvoj JDK. Alat se negde može naći i pod nazivom Java Development Kit 6.

Ukoliko budete koristili JDL za kreiranje probnih programa u ovoj knjizi, dodatne informacije potražite u Dodatku A, "Korišćenje JDK-a (Java Development Kit)". U tom dodatku objašnjen je način na koji se preuzima i instalira alat, odnosno način na koji se koristi u procesu kreiranja Java programa.

Nakon što nabavite razvojni alat koji podržava programski jezik Java 6 i instalirate ga na svom računaru, spremni ste da počnete učenje ovog programskog jezika.

Objektno-orijentisano programiranje

Najveći izazov za novog Java programera je istovremeno učenje objektno-orijentisanih koncepata i programskog jezika Java.

Iako to može da deluje prilično obeshrabrujuće ukoliko ne poznajete ovaj stil programiranja, naučićete koncepte objektno-orijentisanog programiranja, učeći programski jezik Java. Ne postoji drugi način da primenjujete ovaj programski jezik. Objektno-orijentisano programiranje je pristup razvoju računarskih programa koji imitira način na koji su objekti definisani u realnom svetu.

Primenjujući ovaj stil razvoja, možete da kreirate programe čiji se kod može više puta koristiti, programe koji su pouzdani i razumljivi.

Da biste ovo spoznali, neophodno je prvo da naučite na koji način Java implementira principe objektno-orijentisanog programiranja. Sledeće teme razmatramo u prvom delu ove knjige:

- organizovanje programa pomoću elemenata koji se nazivaju klase
- upoznavanje postupka na koji se kreiraju objekti na osnovu klasa
- definisanje klase pomoću dva aspekta njene strukture: načina na koji treba da se ponaša i atributa koje treba da sadrži
- međusobno povezivanje klasa, tako da jedna klasa nasleđuje funkcionalnasti neke druge klase
- povezivanje klasa korišćenjem paketa i interfejsa

Ukoliko ste već upoznali koncepte objektno-orijentisanog programiranja, poznat Vam je veliki deo materijala u ovom poglavlju. Čak i u slučaju da samo prelistate uvodni materijal, trebalo bi da kreirate probni program da biste stekli neko isukstvo u proceu kreiranja, prevođenja i izvršavanja Java programa.

Postoji nekoliko različitih načina za konceptualizovanje računarskog programa. Jedan od tih načina je da program tretirate kao seriju instrukcija koje se izvršavaju u sekvenci; to se obično naziva *proceduralno programiranje*. Mnogi programeri su počeli tako što su učili neki proceduralni programski jezik, kao što je BASIC.

Proceduralni jezici odslikavaju način na koji računar izvršava instrukcije, tako da se programi kreiraju na način na koji računar funkcioniše. Prilikom korišćenja proceduralnih jezika programer prvo mora da nauči da razloži problem u obliku serije jednostavnih koraka.

U okviru objektno-orijentisanog programiranja se računarski program razmatra na sasvim drugačiji način, pri čemu je suštinski značajan, zapravo, zadatak zbog koga koristite računar, a ne način na koji računar izvršava sam zadatak.

U okviru objektno-orijentisanog programiranja računarski program se tretira kao niz objekata koji sarađuju u cilju izvršavanja određenog zadatka. Svaki objekat je poseban deo programa, koji interaguje sa drugim delovima programa na specifičan, kontrolisan način.

Kao primer iz realnog života, razmotrimo stereo sistem. Najveći broj stereo sistema sastavljen je od velikog broja različitih objekata, koji se najčešće nazivaju komponente, kao što su:

- zvučnici koji reprodukuju zvuke srednje i visoke učestalosti
- posebna komponenta (Subwoofer) koja reprodukuje basove
- prijemnik koji prima signale radio stanica
- CD plejer koji čita zvučne podatke sa CD-a.

Sve prethodno navedene komponente su projektovane tako da interaguju jedne sa drugima korišćenjem standardnih ulaznih i izlaznih konektora. Čak i u slučaju da ste kupili zvučnike, subwoofer, primenik i CD plejer različitih proizvođača, možete ih kombinovati i formirati stereo sistem, naravno u slučaju da poseduju standardne konektore.

U slučaju objektno-orijentisanog programiranja važi potpuno identičan princip: svoj program kreirate kombinujući novokreirane objekte i postojeće objekte, primenjujući prilikom povezivanja odgovarajuće standardne postupke. Svaki objekat ima specifičnu ulogu u samom programu.

Objekat je nezavistan element računarskog programa, koji predstavlja grupu povezanih funkcionalnosti i koji je projektovan tako da izvršava specifičan zadatak.

Objekti i klase

Objektno-orijentisano programiranje je kreirano uz pretpostavku da su u fizičkom svetu objekti sastavljeni od različitih tipova manjih objekata.

Međutim, mogućnost kombinovanja objekata je samo jedan aspekt objektno-orijentisanog programiranja. Još jedna značajna funkcionalnost je upotreba klasa.

Klasa je šablon koji se koristi za kreiranje objekta. Svi objekti kreirani korišćenjem jedne klase imaju identične funkcionalnosti.

Klase obuhvataju sve funkcionalnosti konkretnog skupa objekata. Kada pišete program korišćenjem nekog objektno-orijentisanog programskog jezika, ne morate da definišete pojedinačne objekte. Umesto toga, definišete klase koje ćete upotrebljavati za kreiranje tih pojedinačnih objekata.

Na primer, možete kreirati klasu `Modem`, koja opisuje funkcionalnosti računarskih modema. Najveći broj modema poseduje sledeće funkcionalnosti:

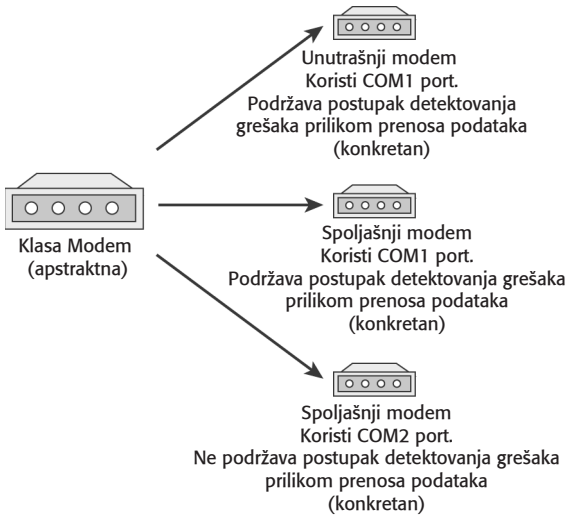
- Povezuju se na serijski port računara.
- Šalju i primaju informacije.
- Pozivaju odgovarajuće telefonske brojeve.

Klasa `Modem` služi kao apstraktan model za opisivanje koncepta modema. Da biste imali nešto konrektno što možete da koristite u okviru programa, morate da koristite `Modem` klasu za kreiranje konkretnih objekata. Proces kreiranja objekta na osnovu klase naziva se *instanciranje*, pošto se sami objekti često nazivaju i instance klase.

Klasu `Modem` možete koristiti za kreiranje velikog broja `Modem` objekata u svom programu, a svaki od tih objekata može da ima različite funkcionalnosti, kao što su:

- Neki modemi su unutrašnji, a neki spoljašnji.
- Neki modemi koriste COM1 port, a neki drugi COM2 port.
- Neki imaju implementiran postupak detektovanja grešaka prilikom prenosa podataka, a drugi nemaju.

Čak i u slučaju prethodno navedenih razlika, dva objekta klase `Modem` još uvek su dovoljno slična jedan drugom. Na slici 1.1 prikazani su klasa `Modem` i nekoliko objekata koji su kreirani na osnovu te klase (ona je korišćena kao šablon za kreiranje).



Slika 1.1
Klasa Modem i nekoliko objekta klase Modem

Sledi još jedan primer: koristeći programski jezik Java, možete da kreirate klasu koja predstavlja sve komandne tastere - polja koja možete da pritisnete, a koja se nalaze u okviru prozora, dijaloga i drugih delova grafičkog korisničkog interfejsa Vašeg programa.

Prilikom kreiranja `CommandButton` klase, možete da definišete sledeće funkcionalnosti:

- tekst koji se prikazuje na tasteru
- veličinu tastera
- aspekte prikazivanja - na primer, da li postoje trodimenzionalne senke.

`CommandButton` klasa može da definiše i način na koji se ponaša taster, utvrđujući pri tom sledeće:

- da li je neophodno taster pritisnuti jednom ili dva puta
- da li treba ignorisati neprekidno pritiskanje tastera
- šta treba uraditi nakon uspešnog pritiskanja tastera

Nakon što definišete `CommandButton` klasu, možete da kreirate instance tog tastera - drugim rečima, objekata `CommandButton` klase. Objekti imaju sve osnovne funkcionalnosti tastera koje definiše sama klasa, ali svaki objekat može da ima različite vrednosti i neznatno drugačije ponašanje, zavisno od toga šta objekat treba da radi.

Kada kreirate `CommandButton` klasu, ne morate da iznova pišete kod za svaki komandni taster koji želite da koristite u svojim programima. Pored toga, možete da više puta koristite `CommandButton` klasu da biste kreirali različite tipove tastera koji su Vam neophodni u tom ili nekom drugom programu.

NAPOMENA Jedna od standardnih klasa programskog jezika Java `javax.swing.JButton` obuhvata sve funkcionalnosti hipotetičke `CommandButton` klase i još mnogo štošta. Imaćete priliku da koristite ovu klasu u toku 9. dana, "Upotreba Swinga".

Kada kreirate Java program, Vi, u stvari, projektujete i konstruišete skup klasa. Prilikom izvršavanja programa objekti se instanciraju na osnovu klasa, a zatim se koriste prema potrebi. Vaš zadatak kao Java programera je da kreirate odgovarajući skup klasa da biste omogućili da program izvršava ono za šta je predviđen.

Srećom, ne morate da krenete "od nule". Programski jezik Java sadrži nekoliko hiljada klasa koje implementiraju mnoge funkcionalnosti koje su Vam neophodne. Ove klase sačinjavaju biblioteku Java klasa i instaliraju se zajedno sa razvojnim alatom, kao što je JDK alat.

Kada se govori o korišćenju programskog jezika Java, zapravo se misli na korišćenje prethodno pomenute biblioteke klasa i nekih standardnih ključnih reči i operatora koje prepoznaju programski prevodioci Java jezika.

Biblioteka klasa sadrži veliki broj metoda koji izvršavaju matematičke operacije, obradu teksta i implementiraju grafičke elemente programa, korisničku interakciju i mrežnu komunikaciju. Korišćenje ovih klasa se ne razlikuje od korišćenja Java klasa koje sami kreirate.

U toku razvoja Java programa možete da kreirate ceo skup novih klasa koje formiraju posebnu biblioteku klasa, koju možete da koristite prilikom razvoja drugih programa.

Višestruko korišćenje programskog koda je jedna od osnovnih prednosti objektno-orijentisanog programiranja.

Atributi i ponašanje

Java klasa sadrži dva različita tipa informacija: atribute i ponašanje.

Oba ova tipa nalaze se u `VolcanoRobot` projektu (danas ćete ga implementirati) koji sadrži klasu. Ovaj projekat, koji predstavlja jednostavnu simulaciju vozila za istraživanje vulkana, kreiran je po uzoru na robot Dante II koji koristi NASA u okviru Telerobotics Research programa u procesu istraživanja unutrašnjosti vulkanskih kratera.

Atributi klase objekata

Atributi su podaci na osnovu kojih se razlikuju objekti. Oni se mogu upotrebljavati za utvrđivanje pojave, stanja i drugih kvaliteta objekata koji pripadaju jednoj klasi.

Vozilo za istraživanje unutrašnjosti vulkanskih kratera može da sadrži sledeće atribute:

- **status** - istraživanje, pomeranje, vraćanje na prvobitnu poziciju
- **brzina** - brzina izražena u miljama na sat
- **temperatura** - temperatura izražena u farenhajtima

U okviru klase, atributi su definisani pomoću promenljivih - pomoću onoga što se u računarskom programu koristi za čuvanje informacija. Promenljive instance su atributi sa vrednošću koja se razlikuje u slučaju različitih objekata.

Promenljiva instance definiše atribut jednog konkretnog objekta. Klasa objekta definiše o kom tipu atributa je reč, a svaka instanca čuva odgovarajuću vrednost tog atributa. Promenljive instance se nazivaju i *promenljive objekta*.

Za svaki atribut klase "vezana" je jedna odgovarajuća promenljiva. Atribut objekta menjate tako što promenite vrednost odgovarajuće promenljive.

Na primer, klasa `VolcanoRobot` definiše `speed` promenljivu instance. Ovo mora biti promenljiva instance, pošto se svaki robot kreće drugačijom brzinom. Vrednost `speed` promenljive instance robota može da se menja, tako da se robot pomera brže ili sporije.

Promenljivim instance mogu se dodeljivati vrednosti prilikom kreiranja objekta, pa da te vrednosti ostanu konstantne u toku "života" objekta. Osim toga, promenljivim instance se mogu menjati vrednosti u toku izvršavanja programa.

U slučaju nekih drugih promenljivih ima više smisla da postoji jedna vrednost koju dele svi objekti određene klase. Ovakvi atributi se nazivaju promenljive klase.

Promenljiva klase definiše atribut cele klase. Ona je definisana na nivou klase i svih njenih instanci, tako da se čuva samo jedna vrednost, nezavisno od broja objekata posmatrane klase koje kreirate.

Primer promenljive klase u slučaju `VolcanoRobot` klase mogla bi biti promenljiva koja beleži trenutno vreme. Ukoliko bismo za beleženje tekućeg vremena kreirali promenljivu instance, svaki objekat bi imao različitu vrednost te promenljive, što bi dovelo do problema u slučaju da roboti treba da izvršavaju neke akcije u saradnji sa drugim robotima.

Upotrebom promenljive klase rešava se prethodni problem, pošto u tom slučaju svi objekti klase automatski dele istu vrednost. Svaki objekat VolcanoRobot klase u tom slučaju može da pristupa konkretnoj promenljivoj.

Ponašanje klase objekata

Ponašanje ukazuje na sve ono što što može da se izvrši nad objektima te klase i drugih klasa. Ono podrazumeva sve ono što može da se koristi za menjanje atributa jednog objekta, za prijem informacija od drugih objekata i za slanje poruka drugim objektima, u kojima se zahteva izvršavanje određenih zadataka.

U slučaju vozila za istraživanje unutrašnjosti vulkanskih kratera može se definisati sledeće ponašanje:

- provera trenutne temperature
- početak istraživanja
- izveštavanje o karakteristikama trenutne lokacije

Ponašanje se u slučaju klase objekata implmentira korišćenjem metoda.

Metodi su grupe povezanih naredbi u klasi, koje izvršavaju specifičan zadatak. Koriste se za izvršavanje specifičnih zadataka nad objektima klase kojoj pripadaju i nad drugim objektima, a mogu se uporediti sa onim što obavljaju funkcije i opšti potprogrami u drugim programskim jezicima.

Objekti međusobno komuniciraju korišćenjem metoda. Klasa ili objekat može da poziva metode druge klase ili objekta u velikom broju slučajeva, kao što su:

- izveštavanje drugog objekta o nekoj promeni
- slanje poruke drugom objektu da nešto promeni
- zahtevanje da drugi objekat nešto uradi

Na primer, klasa koja definiše vozilo za istraživanje unutrašnjosti vulkanskih kratera može da sadrži metode za izveštavanje drugih vozila o trenutnoj lokaciji, a samim tim i za izbegavanje sudara, a jedan robot može da zahteva da se drugi robot zaustavi da bi se omogućilo mimoilaženje.

Kao što postoje promenljive instance i promenljive klase, postoje i metodi instance i metodi klase. *Metodi instance*, koji se obično nazivaju, jednostavno, *metodi*, koriste se kada se radi sa odgovarajućim objektom klase. Ukoliko metod obavlja izmenu pojedinačnog objekta, on mora biti metod instance. *Metodi klase* se primenjuju na samu klasu.

Kreiranje klase

Da biste videli klase, objekte, atribute i ponašanje u akciji, neophodno je da kreirate `VolcanoRobot` klasu, da kreirate objekte te klase i da radite sa objektima u okviru programa.

NAPOMENA Glavni cilj u ovom projektu je razmatranje objektno-orijentisanog programiranja. Nešto više o sintaksi programskog jezika Java naučićete u drugom poglavlju.

Da biste počeli kreiranje klase, otvorite editor teksta koji ćete koristiti za kreiranje Java programa i kreirajte novu datoteku. Unesite tekst prikazan u listingu 1.1, a zatim snimite tako kreiranu datoteku pod nazivom `VolcanoRobot.java` u direktorijumu koji koristite za čuvanje programa datih u ovoj knjizi.

Listing 1.1: Celokupan sadržaj VolcanoRobot.java datoteke

```
 1: class VolcanoRobot {
 2:     String status;
 3:     int speed;
 4:     float temperature;
 5:
 6:     void checkTemperature() {
 7:         if (temperature > 660) {
 8:             status ="povratak kuci";
 9:             speed = 5;
10:         }
11:     }
12:
```

nastavlja se

Listing 1.1: Celokupan sadržaj VolcanoRobot.java datoteke *nastavak*

```
13:     void showAttributes() {
14:         System.out.println("Status: " ++status);
15:         System.out.println("Brzina: " ++speed);
16:         System.out.println("Temperatura: " ++temperature);
17:     }
18: }
```

Naredba `klase` u liniji 1 listinga 1.1 definiše naziv `VolcanoRobot` klase. Sve ono što se nalazi između otvorene velike zagrade ("{") u liniji 1 i zatvorene velike zagrade ("}") u liniji 18 predstavlja sadržaj ove klase.

`VolcanoRobot` klasa sadrži tri promenljive instance i dva metoda instance. Promenljive instance su definisane u linijama 2-4:

```
String status;
int speed;
float temperature;
```

Promenljive instance nose naziv `status`, `speed` i `temperature`. Svaka od njih beleži različiti tip informacija:

- Promenljiva `status` sadrži `String` objekat, odnosno grupu slova, brojeva, znakova interpunkcije i drugih karaktera.
- Promenljiva `speed` je tipa `int`, odnosno sadrži celobrojnu vrednost.
- Promenljiva `temperature` je tipa `float`, odnosno sadrži realnu vrednost u pokretnoj zapeti.

`String` objekti se kreiraju na osnovu `String` klase, koja predstavlja deo Java biblioteke klasa, a možete je koristiti u bilo kom programu.

SAVET Primetićete na osnovu korišćenja klase `String` u ovom programu da klasa može koristiti objekte kao promenljive instance.

Prvi metod instance u `VolcanoRobot` klasi definisan je u linijama 6-11 i sledećeg je oblika:

```
void checkTemperature() {
    if (temperature > 660) {
        status = "povratak kuci";
        speed = 5;
    }
}
```

Metodi su definisani na sličan način kao i sama klasa. Oni počinju naredbom koja definiše naziv metoda, tip informacije koju generiše metod i parametre koje on koristi.

Metod `checkTemperature()` je definisan u delu koda između linija 6 i 11 listinga 1.1. Ovaj metod može da se primeni nad `VolcanoRobot` objektom da bi bila očitana temperatura.

Ovaj metod proverava da li `temperature` promenljiva instance objekta ima vrednost veću od 660. Ukoliko je ovaj uslov ispunjen, menjaju se druge dve promenljive instance na sledeći način:

- Promenljiva instance `status` dobija vrednost stringa "povratak kuci", čime se ukazuje da je temperatura suviše visoka, pa robot mora da se vrati nazad u bazu.
- Promenljiva instance speed dobija vrednost 5 (pretpostavlja se da je to najveća moguća brzina kojom robot može da se kreće).

Drugi metod instance `showAttributes()` definisan je u linijama 13 -17 na sledeći način:

```
void showAttributes() {
    System.out.println("Status: " ++status);
    System.out.println("Brzina:" ++speed);
    System.out.println("Temperatura: " ++temperature);
}
```

Ovaj metod poziva `System.out.println()` metod za prikazivanje vrednosti tri promenljive instance, zajedno sa odgovarajućim propratnim tekstom koji objašnjava značenje svake od prikazanih vrednosti.

Nakon što unesete neophodan izvorni kod, snimite datoteku. U ovom trenutku nije neophodno da je prevodite.

Izvršavanje programa

Čak i da ste uspešno preveli `VolcanoRobot` klasu, sa njom ne možete da uradite ništa. Klasa koju ste kreirali definiše šta `VolcanoRobot` objekat treba da radi u slučaju da se koristi u okviru nekog drugog programa, ali se pri tom ne kreira ni jedan objekat same klase.

Postoje dva načina korišćenja `VolcanoRobot` klase:

- Kreirajte poseban Java program koji će korisititi klasu.
- U `VolcanoRobot` klasu dodajte specijalan metod klase, koji se naziva `main()`, tako da klasa može da se izvršava kao aplikacija, a zatim upotrebite objekte `VolcanoRobot` klase u okviru tog metoda.

Prva opcija će se primenjivati u ovoj vežbi. U listingu 1.2 prikazan je izvorni kod za `VolcanoApplication` klasu, Java klasu koja kreira objekat `VolcanoRobot` klase, definiše vrednosti promenljivih instance i poziva metode klase.

Listing 1.2: Celokupan sadržaj VolcanoApplication.java datoteke

```
 1: class VolcanoApplication {
 2:     public static void main(String[] arguments) {
 3:         VolcanoRobot dante = new VolcanoRobot();
 4:         dante.status = "istrazivanje";
 5:         dante.speed = 2;
 6:         dante.temperature = 510;
 7:
 8:         dante.showAttributes();
 9:         System.out.println("Povecati brzinu na 3.");
10:         dante.speed = 3;
11:         dante.showAttributes();
12:         System.out.println("Promeniti temperaturu na 670.");
13:         dante.temperature = 670;
14:         dante.showAttributes();
15:         System.out.println("Proveriti temperaturu.");
16:         dante.checkTemperature();
17:         dante.showAttributes();
18:     }
19: }
```

Snimite datoteku pod nazivom `VolcanoApplication.java`, a zatim prevedite program.

Ukoliko koristite JDK, da biste preveli program, treba da uradite sledeće: Pređite u komandnu liniju, ili otvorite komandni prozor, pristupite direktorijumu u kome se nalazi `VolcanoApplication.java` datoteka, a zatim prevedite program ukucavanjem sledećeg teksta u komandnoj liniji:

```
javac VolcanoApplication.java
```

Java kompilator kreira `VolcanoApplication.class` datoteku, koja sadrži bajtkod koji može izvršavati interpretator programskog jezika Java. Kompilator isti postupak primenjuje i u slučaju `VolcanoRobot.java` klase, ukoliko je to neophodno, pošto se ta klasa koristi u ovoj aplikaciji.

SAVET Ukoliko dođe do nekih problema prilikom prevođenja ili izvršavanja bilo kog programa iz ove knjige korišćenjem JDK alata, kopije datoteka sa izvornim kodom i ostalih datoteka iz ove knjige možete pronaći na zvaničnoj prezentaciji knjige na adresi: http://www.java21days.com.

Nakon što uspešno prevedete program, startujte ga.

U slučaju da koristite JDK alat, `VolcanoApplication` program startujete tako što pristupite direktorijumu koji sadrži `VolcanoRobot.class` i `VolcanoApplication.class` datoteke i unesete u komandnu liniju sledeću komandu:

```
java VolcanoApplication
```

Nakon što startujete `VolcanoApplication` aplikaciju, rezultat izvršavanja bi trebalo da bude sledećeg oblika:

```
Status: istrazivanje
Brzina: 2
Temperatura: 510.0
Povecati brzinu na 3.
Status: istrazivanje
Brzina: 3
Temperatura: 510.0
Promeniti temperaturu na 670.
Status: istrazivanje
Brzina: 3
Temperatura: 670.0
Proveriti temperaturu.
Status: povratak kuci
Brzina: 5
Temperatura: 670.0
```

Ukoliko detaljnije analizirate listing 1.2, videćete da `main()` metod klase sadrži sledeće:

- **Linija 2** - Kreira se `main()` metod i dodeljuje mu se naziv. Svaki `main()` metod ima isti format, što ćete videti u okviru Poglavlja 5, "Kreiranje klasa i metoda". Za sada, najznačajnije je da uočite ključnu reč `static`, koja ukazuje da je reč o metodu klase koji dele svi objekti `VolcanoRobot` klase.
- **Linija 3** - Novi objekat `VolcanoRobot` klase kreira se korišćenjem klase kao šablona za kreiranje objekata. Novi objekat naziva se `dante`.
- **Linije 4-6** - Tri promenljive instance `dante` objekta dobijaju vrednosti: promenljivoj instance `status` dodeljuje se vrednost "istrazivanje", promenljivoj instance `speed` dodeljuje se vrednost `2`, a promenljivoj instance `temperature` dodeljuje se vrednost `510`.
- **Linija 8** - U okviru ove linije i nekoliko linija koje slede poziva se `showAttributes()` metod `dante` objekta. Ovaj metod prikazuje trenutne vrednosti promenljivih `instance status`, `speed` i `temperature`.
- **Linija 9** - U ovoj liniji i u nekoliko linija koje slede naredba `System.out.println`() se koristi za prikazivanje teksta u okviru zagrada.
- **Linija 10** - Promenljiva instance `speed` dobija vrednost `3`.
- **Linija 13** - Promenljiva instance `temperature` dobija vrednost `670`.
- **Linija 16** - Poziva se metod `checkTemperature()` objekta `dante`. On proverava da li `temperature` promenljiva instance ima vrednost koja je veća od 600. Ukoliko je uslov ispunjen, `status` i `speed` promenljive instance dobijaju nove vrednosti.

Organizovanje klasa i ponašanje klase

Uvodna razmatranja u vezi sa objekto-orijenetisanim programiranjem u programskom jeziku Java nisu potpuna ukoliko se ne razmotre još tri koncepta: nasleđivanje, interfejsi i paketi.

Ova tri koncepta predstavljaju mehanizme za organizovanje klasa i ponašanja klase.

Nasleđivanje

Nasleđivanje je jedan od suštinskih koncepata objektno-orijentisanog programiranja i od presudnog je uticaja na način na koji projektujete i pišete sopstvene Java klase.

Ono predstavlja mehanizam koji omogućava da jedna klasa nasleđuje celokupno ponašanje i sve atribute neke druge klase.

Nasleđivanjem jedna klasa u potpunosti preuzima funkcionalnosti postojeće klase. Zbog toga, morate da definišete ono što novu klasu čini drugačijom od postojeće.

Primenom nasleđivanja, sve klase (one koje kreirate i one koje se nalaze u Java biblioteci klasa i u nekim drugim bibliotekama klasa) uređene su u okviru odgovarajuće striktne hijerarhije.

Klasa koja nasleđuje neku drugu klasu naziva se *izvedena klasa* (potklasa). Klasa koja se nasleđuje naziva se *natklasa* (superklasa).

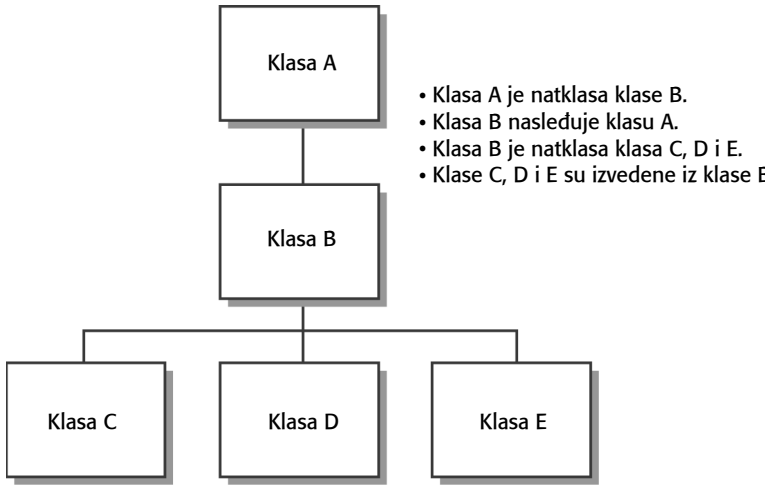
Klasa može da ima samo jednu natklasu, ali svaka klasa može da ima neograničeni broj izvedenih klasa. Izvedene klase nasleđuje sve atribute i ponašanje klasa koje nasleđuju.

To praktično znači da u slučaju da neka natklasa poseduje ponašanje i atribute koji su neophodni Vašoj klasi, ne morate da preddefinišete ili kopirate kod osnovne klase da biste u svojoj klasi definisali određeno ponašanje i atribute. Vaša klasa automatski nasleđuje ponašanje i atribute od osnovne klase, ta klasa preuzima odgovarajuće ponašanje i atribute od neke druge klase i tako redom u okviru odgovarajuće hijerarhije klasa. Vaša klasa postaje kombinacija sopstvenih funkcionalnosti i svih funkcionalnosti klasa koje se nalaze iznad nje u okviru posmatrane hijerarhije klasa.

Ova situacija je slična načinu na koji nasleđujete osobine od svojih roditelja, kao što su visina, boja kose, odnosno sklonost ka korišćenju putera ili banana. Vaši roditelji su neke svoje osobine nasledili od svojih roditelja, koji su ih nasledili od svojih roditelja i tako dalje sve do Rajskog vrta, Velikog praska ili ... [*unesite na ovom mestu ono u šta verujete da je bilo početak svega*].

Na slici 1.2 prikazan je način na koji su klase uređene u okviru hijerarhije klasa.

Na vrhu hijerarhije Java klasa je `Object`; sve klase nasleđuju ovu natklasu. `Object` klasa je najopštija u hijerarhiji - ona definiše ponašanje koje nasleđuju sve ostale klase Java biblioteke klasa.



Slika 1.2
Hijerarhija klasa

U okviru hijerarhije klasa svaka klasa koja se nalazi niže u hijerarhiji ima specifičniju namenu. Hijerarhija klasa definiše apstraktne koncepte na vrhu hijerarhije. Ovi konpcepti postaju konkretniji u okviru klasa koje se nalaze niže u okviru same hijerarhije klasa.

Često ćete prilikom kreiranja nove klase u programskom jeziku Java želeti da uključite sve funkcionalnosti neke postojeće klase, uz određene modifikacije koje definišu specifičnosti Vaše klase. Na primer, možda ćete želeti da kreirate varijantu `CommandButton` klase tako da se reprodukuje odgovarajući zvuk prilikom pritiskanja tastera.

Da biste preuzeli sve funkcionalnosti `CommandButton` klase, a da pri tom ne morate samostalno da je kreirate, možete da definišete da je Vaša klasa nasleđuje. Vaša klasa će u tom slučaju automatski naslediti ponašanje i atribute koji su definisani u `CommandButton` klasi, kao i u svim klasama koje ona nasleđuje. Sve što treba da uradite je da realizujete nove metode i atribute koje će Vašu klasu učiniti različitom u odnosu na samu `CommandButton` klasu. Definisanje potklasa podrazumeva kreiranje novih klasa na osnovu razlika između tih klasa i njihovih natklasa.

Definisanje *potklase* je kreiranje nove klase koja nasleđuje postojeću klasu. Jedini zadatak u ovom slučaju je utvrđivanje razlika u ponašanju i postojećim atributima između novokreirane klase i njoj odgovarajuće natklase.

Ukoliko Vaša klasa definiše potpuno novo ponašanje i ne predstavlja klasu izvedenu na osnovu neke postojeće klase, možete u tom slučaju direktno da nasledite klasu `Object`. Time je omogućeno da svoju klasu uklopite u hijerarhiju postojećih Java klasa. Ukoliko kreirate definiciju klase koja ne poseduje natklasu, Java prevodilac podrazumeva da klasa nasleđuje direktno `Object` klasu. Prilikom kreiranja `VolcanoRobot` klase koju ste kreirali u toku ovog poglavlja nije eksplicitno definisana natklasa, tako da je reč o klasi izvedenoj direktno iz `Object` klase.

Kreiranje hijerarhije klasa

Ukoliko kreirate veliki broj klasa, ima smisla da svoje klase nasleđujete iz postojećih klasa u okviru hijerarhije, odnosno da kreirate hijerarhiju klasa zbog sledećih razloga:

- Opšte funkcionalnosti, koje su zajedničke za veliki broj klasa, mogu se postaviti u određenu natklasu, čime se omogućava upotreba tih funkcionalnosti u svim klasama koje nasleđuju tu klasu u okviru hijerahrije klasa.
- Izmene neke natklase automatski se odslikavaju na sve potklase, njihove potklase i tako redom u okviru hijerarhije. Ne postoji potreba da se menja ili ponovo prevodi bilo koja klasa koja se nalazi na nižem nivou u okviru hijerarhije; klase koje se nalaze na nižem nivou u okviru hijerarhije primaju neophodne informacije procesom nasleđivanja.

Na primer, zamislite da ste kreirali Java klasu koja implementira sve funkcionalnosti vozila za istraživanje unutrašnjosti vulkanskih kratera.

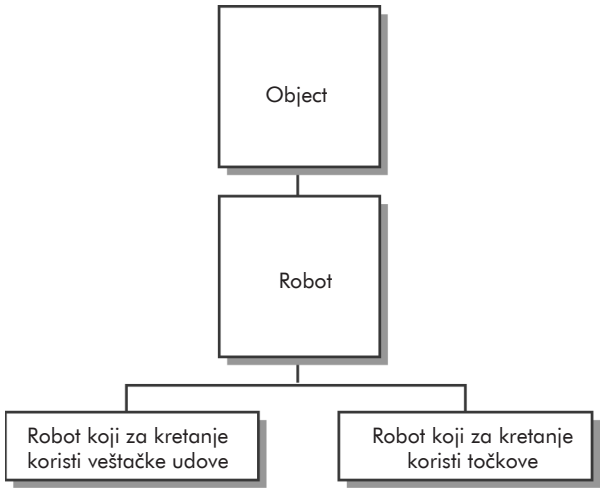
`VolcanoRobot` klasa je kreirana i uspešno funkcioniše. Sada želite da kreirate Java klasu koja se naziva MarsRobot.

Ova dva tipa robota poseduju iste funkcionalnosti - to su istraživački roboti, koji funkcionišu u neprijatnom okruženju i u njemu vrše istraživanja. Vaš prvi potez može biti otvaranje `VolcanoRobot.java` datoteke koja sadrži izvorni kod i kopiranje velikog dela te datoteke u novu datoteku sa izvornim kodom pod nazivom `MarsRobot.java`.

Bolje rešenje je da otkrijete zajedničke funkcionalnosti `MarsRobot` i `VolcanoRobot` klasa, pa da ih organizujete u okviru opšte hijerarhije klasa. Možete imati dosta posla u slučaju da imate samo klase `VolcanoRobot` i `MarsRobot`. Šta će se dogoditi u slučaju da želite da dodate još klasa, kao što su `MoonRobot`, `UnderseaRobot` i `DesertRobot`? Implementiranjem opšteg ponašanja u jednoj ili više natklasa značajno redukujete posao koji treba da obavite.

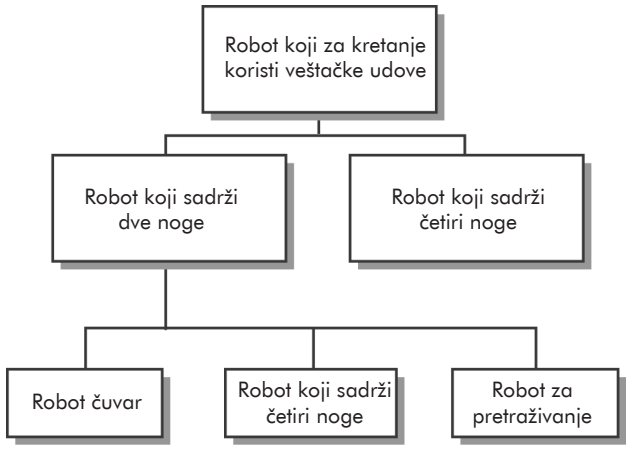
Da biste projektovali hijerarhiju klasa koja može da se primenjuje u tom slučaju, počnite od vrha, od `klase Object`, osnovne klase svih Java klasa. Najopštija klasa u kojoj vozila za istraživanje unutrašnjosti vulkanskih kratera mogu da se nalaze može da se naziva `Robot`. U opštem slučaju, `Robot` može da se definiše kao uređaj za istraživanje, koji poseduje sopstvenu kontrolu. U toj klasi definišete samo ponašanje koje nešto kvalifikuje da bude uređaj, da poseduje sopstvenu kontrolu i da je projektovano za potrebe istraživanja.

Klasa `Robot` može da sadrži dve potklase: `WalkingRobot` i `DrivingRobot`. Očigledna razlika između ove dve klase je da prvi tip robota za kretanje koristi veštačke udove, a drugi točkove. Ponašanje robota koji hoda može da uključi ulaganje napora za podizanje nečega, saginjanje, trčanje i slično. Roboti koji prilikom kretanja upotrebljavaju točkove mogu se ponašati na drugi način. Na slici 1.3 prikazano je ono što smo do sada uradili.



Slika 1.3 Osnovna hijerarhija klasa u slučaju robota

Vi i dalje možete da razvijate ovu hijerarhiju. U slučaju `WalkingRobot` klase moguće je uvesti još nekoliko novih klasa, kao što su: `ScienceRobot`, `GuardRobot`, `SearchRobot` i tako dalje. Kao alternativu, možete da istaknete neke funkcionalnosti i da uvedete klase `TwoLegged` i `FourLegged` robota, sa različitim ponašanjem za svakog od njih (videti sliku 1.4).



Slika 1.4 Robot koji za kretanje koristi veštačke udove, koji sadrže dve, odnosno četiri noge

Konačno, hijerarhija je kreirana i Vi možete da nađete mesto u toj hijerarhiji za `VolcanoRobot` klasu. Vaša klasa može biti potklasa klase `ScienceRobot`, koja je potklasa klase `WalkingRobot`, koja je potklasa klase `Robot`, a ova potonja je izvedena iz klase `Object`.

U hijerarhiji klasa vrednosti kao što su status, temperatura ili brzina treba postaviti na onom mestu u hijerarhiji gde je to najprirodnije. Pošto svi roboti imaju potrebu da sadrže informacije o temperaturi okruženja u kome se nalaze, ima smisla definisati `temperaturu` kao promenljivu instance klase `Robot`. Sve izvedene klase će takođe sadržati tu promenljivu instance. Zapamtite da je neophodno da definišete ponašanje ili atribut samo na jednom mestu u hijerarhiji klasa - to ponašanje ili atribut nasleđuju sve izvedene klase.

NAPOMENA Projektovanje efikasne hijerarhije klasa zahteva dosta planiranja i revizija. Kako budete pokušavali da definišete odgovarajuće atribute i ponašanje u hijerarhiji klasa, verovatno ćete pronalaziti razloge da neke klase pomerate na druga mesta u samoj hijerarhiji. Cilj pomeranja je redukovanje broja funkcionalnosti koje se ponavljaju u okviru hijerarhije klasa.

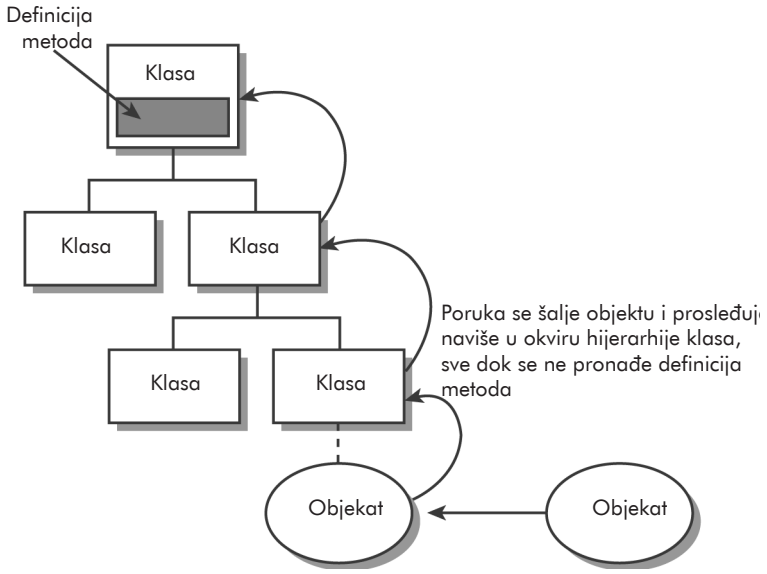
Nasleđivanje u akciji

Nasleđivanje u programskom jeziku Java funkcioniše mnogo jednostavnije nego u realnom svetu. U Javi ne postoji ništa što ispunjava korisnikove želje ili što može da ga učini srećnim na bilo koji način.

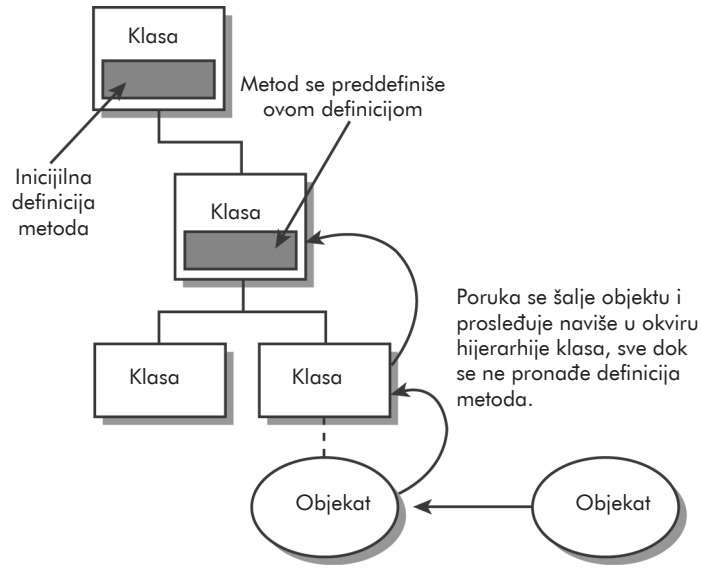
Nakon što kreirate novi objekat, Java prati sve promenljive definisane za njega i sve promenljive definisane u natklasi posmatrane klase. Zahvaljujući tome, sve klase se kombinuju u cilju formiranja šablona postojećeg objekta, a svaki objekat sadrži informacije koje opisuju odgovarajuću situaciju u kojoj se nalazi.

Metodi funkcionišu na isti način - novi objekat ima pristup svim nazivima metoda klase i njene natklase. Ovo se utvrđuje dinamički prilikom korišćenja metoda u toku izvršavanja programa. Ukoliko izvršite metod konkretnog objekta, interpretator programskog jezika Java prvo proverava da li je u klasi tog objekta definisan taj metod. U slučaju da metod nije pronađen u klasi objekta, interpretator pretražuje natklasu posmatrane klase i tako redom, sve dok ne pronađe definiciju metoda koji ste upotrebili. To je prikazano na slici 1.5.

Okolnosti se malo komplikuju u slučaju da neka izvedena klasa definiše metod koji se po nazivu i nekim drugim aspektima poklapa sa metodom odgovarajuće natklase. U tom slučaju, koristiće se onaj metod čija je definicija metoda prva pronađena (krenuvši odozdo naviše u hijerarhiji klasa). Zbog toga, možete da u izvedenoj klasi kreirate metod koji sprečava korišćenje metoda natklase. Da biste to uradili, metod u izvedenoj klasi mora da ima isti naziv, povratni tip i argumente kao i metod u natklasi. Ova procedura se naziva *preddefinisanje*, odnosno *preopterećenje* (videti sliku 1.6).



Slika 1.5
Način lociranja metoda u hijerarhiji klasa



Slika 1.6
Preddefinisanje (preopterećenje) metoda

Jednostruko i višestruko nasleđivanje

Način nasleđivanja koji se primenjuje u programskom jeziku Java naziva se *jednostruko nasleđivanje*, zato što Java klasa može da nasledi samo jednu natklasu (iako se iz svake klase može izvesti veći broj klasa).

U drugim objektno-orijentisanim programskim jezicima, kao što je programski jezik C++, klase mogu nasleđivati više od jedne natklase, što znači da nasleđuju kombinovane promenljive i metode svih tih natklasa. Ovaj postupak nasleđivanja naziva se *višestruko nasleđivanje* i predstavlja postupak za kreiranje klasa koje mogu da sadrže bukvalno bilo kakvo ponašanje. Međutim, ovim postupkom se značajno komplikuje postupak definisanja klasa i kreiranja koda klasa. U programskom jeziku Java postupak nasleđivanja je značajno pojednostavljen time što je omogućeno samo jednostruko nasleđivanje.

Interfejsi

Jednostruko nasleđivanje omogućava da se jednostavnije razume proces implementiranja i projektovanja veza između klasa i funkcionalnosti ovih klasa. Međutim, to može biti i značajno ograničenje, posebno kada imate identično ponašanje koje je neophodno duplirati u različitim delovima hijerarhije klasa. Programski jezik Java omogućava da se problem u vezi sa deljenim ponašanjem reši korišćenjem interfejsa.

Interfejs je kolekcija metoda koji ukazuju da klasa poseduje neko ponašanje pored onoga koje nasleđuje od natklase. Metodi uključeni u interfejs ne definišu samo ponašanje; taj zadatak ostavljen je klasama koje implementiraju sam interefejs.

Na primer, `Comparable` interfejs sadrži metod koji poredi sva objekta iste klase da bi se utvrdilo koji od njih treba da se pojavi prvi u uređenoj listi. Svaka klasa koja implementira ovaj interfejs može da definiše redosled u kome su uređeni objekti te klase. Ovo ponašanje ne bi bilo moguće realizovati da ne postoje interfejsi.

O interjfesima ćete naučiti nešto više u okviru šestog poglavlja.

Paketi

Paketi u programskom jeziku Java predstavljaju sredstvo pomoću koga se mogu grupisati srodne klase i interfejsi. Oni omogućavaju da grupišete klase tako da se mogu koristiti samo u slučaju da su potrebne, a pomoću njih eliminišete potencijalne konflikte koji se mogu javiti u slučaju naziva klasa u različitim grupama klasa.

Prema definiciji, Java klase imaju mogućnost pristupa samo klasama koje se nalaze u `java.lang` paketu, koji obezbeđuje osnovne funkcionalnosti jezika, kao što je rukovanje stringovima. Da biste upotrebljavali klase koje se nalaze u nekim drugim paketima, neophodno je da eksplicitno navedete naziv paketa u kome se nalazi klasa ili da uvezete klase u svoju izvornu datoteku.

Da biste koristili klasu koja se nalazi u okviru nekog paketa, neophodno je da upotrebite njen pun naziv. Na primer, pošto se klasa `Color` nalazi u `java.awt` paketu, klasu Color biste u svom programu koristili u obliku `java.awt.Color`.

Kratak pregled

Ukoliko ste se danas prvi put susreli sa objektno-orijentisanim programiranjem, verovatno Vam je sve ovo delovalo prilično teoretski i ne preterano zanimljivo.

Ukoliko ste se prvi put suočili sa konceptima u vezi sa objektno-orijentisanim programiranjem i odgovarajućom terminologijom, možda ćete se pomalo zabrinuti da nećete moći da narednih 20 dana na adekvatan način propratite lekcije u vezi sa programskim jezikom Java.

U ovom trenutku bi trebalo da posedujete osnovna znanja o klasama, objektima, atributima i ponašanju i šta su promenljive i metodi instanci. Već u sledećem poglavlju ćete upotrebljavati upravo promenljive i metode instanci.

Drugi koncepti objektno-orijentisanog programiranja, kao što su nasleđivanje i paketi, biće detaljnije obrađeni u poglavljima koja slede. Objektno-orijentisano programiranje ćete primenjivati u svim preostalim poglavljima ove knjige. Nakon prve nedelje (nakon prvih sedam poglavlja), imaćete praktično iskustvo u vezi sa radom sa objektima, klasama, nasleđivnajem i svim drugim aspektima metodologije.

Pitanja i odgovori

P **Gledano sa praktične strane, metodi su, zapravo, funkcije koje su definisane u okviru klase. Ukoliko izgledaju i ponašaju se kao funkcije, zbog čega se drugačije nazivaju?**

O U nekim objektno-orijentisanim programskim jezicima se i nazivaju funkcije (u programskom jeziku C++ govori se o funkcijama članicama). U drugim objektno-orijentisanim jezicima definiše se razlika između funkcija koje se nalaze u telu klase ili objekta i funkcija koje se nalaze van tela klase ili objekta, pa se te funkcije drugačije nazivaju, zato što je značajno razumeti način na koji se svaka od njih izvršava. Pošto je u nekim programskim jezicima bitno definisati tu razliku i pošto je pojam *metod* danas ustaljen u terminologiji "vezanoj" za objektno-orijentisane jezike i koncepte, u programskom jeziku Java se koristi pojam metod.

P **Koja je razlika između promenljivih i metoda instance i njima odgovarajućih promenljivih i metoda klase?**

O Skoro sve što budete koristili u svojim Java programima su neke instance (koje se nazivaju i objekti klase), a ne klase. Međutim, neka ponašanja i atributi imaju više smisla ukoliko se primenjuju na samu klasu, a ne na pojedinačni objekat.

Na primer, `Math` klasa, koja se nalazi u `java.lang` paketu, sadrži promenljivu klase `PI`, koja definiše približnu vrednost broja pi. Ova vrednost se ne menja, tako da ne postoji potreba da svaki pojedinačni objekat klase sadrži posebnu kopiju promenljive PI. Sa druge strane, svaki objekat klase `String` sadrži metod koji se naziva `length()`, čiji je zadatak da odredi broj karaktera posmatranog stringa.

Ova vrednost može da se razlikuje za svaki pojedinačni objekat klase `String`, tako da mora da bude metod instance, a ne klase.

Kviz

Pregled današnjeg materijala ćemo obaviti pomoću kviza koji se sastoji od tri pitanja.

Pitanja

1. Kako se na drugi način naziva klasa?
   - **a.** objekat
   - **b.** šablon
   - **c.** instanca
2. Kada kreirate izvedenu klasu, šta morate da definišete za nju?
   - **a.** Sve se automatski definiše.
   - **b.** ono što tu klasu razlikuje od odgovarajuće klase koju nasleđuje
   - **c.** sve što je "vezano" za samu klasu
3. Šta predstavlja metod instance jedne klase?
   - **a.** atribute posmatrane klase
   - **b.** ponašanje posmatrane klase
   - **c.** ponašanje jednog objekta kreiranog na osnovu te klase

Odgovori

1.b. Klasa je apstrkatan šablon koji se upotrebljava za kreiranje objekata koji su slični jedni drugima.

2.b. Definišete sve ono što izvedenu klasu razlikuje u odnosu na natklasu. Ono što je slično već je definisano samim postupkom nasleđivanja. Odgovor pod a. je tehnički korektan, ali u slučaju da je izvedena klasa identična klasi koju nasleđuje, ne postoji uopšte potreba da se kreira odgovarajuća izvedena klasa.

3.c. Metodi instance "vezani" su za specifično ponašanje objekta, a metodi klase za ponašanje svih objekata koji pripadaju odgovarajućoj klasi.

Pitanja u vezi sa dobijanjem odgovarajućih sertifikata

Pitanja koja slede mogu se naći u testovima za dobijanje sertifikata o poznavanju programskog jezika Java. Odgovore dajte bez gledanja u materijal prikazan u ovom poglavlju.

Koji je od narednih iskaza tačan?

a. Svi objekti koji se kreiraju na osnovu jedne klase moraju biti identični.

b. Svi objekti koji se kreiraju na osnovu jedne klase mogu biti međusobno različiti.

c. Objekat nasleđuje atribute i ponašanje klase koja se koristi za njegovo kreiranje.

d. Klasa nasleđuje atribute i ponašanje od izvedene klase.

Odgovor na ovo pitanje možete pronaći na zvaničnoj web stranici ove knjige na adresi http://www.java21days.com. Pristupite stranici posvećenoj prvom poglavlju, pa pritisnite link Certification Practice.

Vežbe

Da biste proširili svoje znanje o temama koje su razmatrane u ovom poglavlju, pokušajte da uradite sledeće vežbe:

1. U `main()` metodu `VolcanoRobot` klase kreirajte drugi objekat klase `VolcanoRobot` i nazovite da `virgil`, definišite vrednosti promenljivih ove instance, a zatim prikažite tako definisane vrednosti.
2. Krierajte hijerarhiju nasleđivanja u slučaju šahovskih figura. Odlučite gde treba da se u okviru hijerarhije nalaze sledeće promenljvie instance: `color` (boja), `startingPosition` (početna pozicija), `forwardMovement` (pomeranje unapred) i `sideMovement` (pomeranje u stranu).

Tamo gde to ima smisla rešenja vežbi su data na zvaničnoj web stranici ove knjige na adresi http://www.java21days.com.