Laslo Kraus

PROGRAMSKI JEZIK

C++

sa rešenim zadacima

(C++14)

AKADEMSKA MISAO
Beograd, 2016

Laslo Kraus

PROGRAMSKI JEZIK C++ SA REŠENIM ZADACIMA
Deseto izdanje

Recenzenti
Dr Igor Tartalja
Dr Đorđe Đurđević

Izdavač
AKADEMSKA MISAO
Bulevar kralja Aleksandra 73, Beograd

Lektor
Anđelka Kovačević

Dizajn naslovne strane
Zorica Marković, akademski slikar

Štampa
Planeta print, Beograd

Tiraž
500 primeraka

ISBN 978-86-7466-582-4

Dušanki i Katarini

Predgovor

Ova knjiga predstavlja udžbenik za programski jezik *C++* namenjen širokom krugu čitalaca. Knjigu mogu da koriste i ljudi s relativno malo programerskog iskustva, ali je vrlo korisna i za profesionalne programere kojima jezik *C++* nije osnovni programski jezik u profesionalnoj delatnosti.

Jezik *C++* je izrastao iz jezika *C*. Ova knjiga je nastavak autorove knjige *Programski jezik C sa rešenim zadacima* ([8]). Podrazumeva se da čitalac zna osnovne elemente jezika *C* i da ima izvesno iskustvo u programiranju na jeziku *C*.

Programski jezik *C++* izložen je u obimu koji može da zadovoljava i naprednije neprofesionalne programere. Od sadržaja prateće standardne biblioteke klasa i funkcija prikazani su samo važniji delovi.

Jedini način da se nauči neki programski jezik jeste da se pišu programi na njemu. Ova knjiga je u potpunosti podređena tom osnovnom načelu.

Uvodno poglavlje 1 sadrži informacije korisne pre svega za početnike: osnovne pojmove objektno-orijentisanog programiranja, komande za obradu programa na jeziku *C++* na računarima u tekstualnom i grafičkom okruženju i kratak pregled jezika *C* kao podsetnik na ono što bi trebalo da je poznato odranije.

U poglavlju 2 prikazana su manja proširenja jezika *C* koja čine da jezik postane bolji jezik *C*. Deo tih izmena je uveden radi otklanjanja nekih nekonzistentnosti jezika *C*, drugi deo uvodi i nove mogućnosti. U novine spadaju upućivači koji omogućavaju prenos podataka u potprograme pomoću adrese na način koji se razlikuje od korišćenja pokazivača, uvođenje operatora za dinamičko dodeljivanje memorije i operatora za ulaz i izlaz podataka uz primenu konverzija, mogućnost umetanja manjih funkcija u kôd, kao i mogućnost definisanja više funkcija s istim imenima pod uslovom da imaju parametre različitih tipova. Za razvoj velikih programskih sistema uvedena je i mogućnost razbijanja celokupnog programa na više prostora imena koji čine odvojene dosege identifikatora.

Ono što posebno obeležava jezik *C++* je uvođenje klasa sa svim pratećim konceptima. Klase su tipovi podataka u pravom smislu reči, jer pored definisanja mogućih vrednosti podataka klasnih tipova, mogu da definišu i operacije koje mogu da se primenjuju na te podatke. U jeziku *C++* posebna pažnja posvećena je stvaranju i uništavanju objekata klasnih tipova. Klase mogu da se definišu tako da nijedan novi objekat (podatak tipa klase) ne može da ostane neinicijalizovan i da svaki objekat bude propisno uništen kada više nije potreban. U poglavlju 3 izloženi su osnovni pojmovi vezani za klase.

Jezik *C++* omogućava da se standardnim operatorima dodeljuju posebna tumačenja za slučajeve kada su operandi objekti klasnih tipova. To se postiže definisanjem odgovarajućih operatorskih funkcija, kojima je posvećeno poglavlje 4.

Grupe objekata u svakodnevnom životu mogu da se dele na podgrupe. Na primer, vozila mogu da se podele na automobile, avione i brodove. Ako zajedničke osobine grupe objekata mogu da se opišu nekom klasom, klase za opis osobina podgrupa mogu da se obrazuju izvođenjem iz klase zajedničkih osobina. Izvedene klase obrađene su u poglavlju 5.

Jezik *C++* omogućava da rukovanje izuzecima (greškama) vremenski ne opterećuje program sve dok se izuzetak ne pojavi. Kada se izuzetak pojavi, kontrola se automatski predaje jednom od rukovalaca izuzecima koje je definisao programer. Način rukovanja izuzecima prikazan je u poglavlju 6.

Generičke funkcije i klase omogućavaju automatsko generisanje funkcija ili klasa kada istu obradu treba primeniti na podatke različitih tipova. Na primer, algoritam za uređivanje niza objekata po nekom kriterijumu ne zavisi od tipa objekata koji se uređuju. Opisivanje šablona generičkih funkcija i klasa predmet je poglavlja 7.

Funkcijske klase su klase čiji objekti mogu da se koriste kao da su funkcije. U poglavlju 8 obrađeni su lambda izrazi koji omogućavaju jednostavno i sažeto definisanje funkcijskih klasa.

Kao dopuna jeziku C++ postoji velika biblioteka standardnih klasa i funkcija za rad sa zbirkama podataka (nizovi, liste, skupovi, ...), tekstovima, kompleksnim brojevima itd. U poglavlju 9 prikazan je deo te biblioteke.

Ulaz i izlaz podataka nije deo jezika *C++* već se realizuje standardnim bibliotečkim klasama. U poglavlju 10 prikazan je deo tih klasa koji je dovoljan i za relativno složene ulazne i izlazne operacije, uključujući i rad s datotekama različitih struktura.

Jezik C++ vrlo je složen. Nisu svi detalji neophodni svakome, a naročito ne početnicima. Odeljci, ponegde cela poglavlja, u knjizi koji mogu da se preskoče u prvom čitanju, bilo zbog složenosti, bilo zbog manjeg značaja, obeleženi su sa ∆. Odeljci se, uglavnom, mogu preskakati nezavisno. Ako se preskoči jedan odeljak, to ne znači da mora da se preskoči i neki kasniji, obeleženi odeljak.

Izuzetak je odeljak 2.2.11.3 – „Upućivači na *dvrednosti* ∆”. On uvodi pojam koji kasnije omogućava pisanje efikasnijih programa. Ako se preskoči, u nastavku treba preskočiti odeljke, pasuse ili čak samo reči u rečenicama koji se odnose na pojmove *„upućivač na* dvrednost*”*, *„premeštajući konstruktor”*, *„premeštajuća dodela vrednosti”* i *„polimorfna kopija premeštanjem”*. Delovi teksta koje treba preskočiti obeleženi su sivom podlogom. Na taj način ne gubi se ništa u funkcionalnosti programa. Jedino će se neke radnje izvršavati manje efikasno. Potpuni programi u knjizi će ispravno raditi i ako se zatamnjeni delovi izostave.

Rukovodeći se već istaknutim načelom o učenju jezika, ova knjiga, pored manjih primera u toku izlaganja, na kraju svakog poglavlja sadrži nekoliko rešenih zadataka sa detaljnim objašnjinjima. Više zadataka može da se nađe u autorovoj zbirci *Rešeni zadaci iz programskog jezika C++* (videti [7]). Kroz zadatke u ovoj knjizi i u zbirci skreće se pažnja na neke specifičnosti jezika *C++* i daju ideje za elegantno i efikasno rešavanje nekih problema.

Ova knjiga je više nego udžbenik za programski jezik *C++*. Kroz rešene zadatke prikazane su i primene osnovnih elemenata objektno-orijentisanog programiranja, a na primerima koji se uopšteno sreću u računarskoj tehnici: obradi redova, stekova, listi, tekstova,

datoteka itd. Posebna pažnja posvećena je i inženjerskim aspektima programiranja. Nije dovoljno da program rešava zadati problem, već treba da bude i „lep”, razumljiv, da se lako održava, da troši što manje memorije i da bude što efikasniji.

Svi programi koji se nalaze u ovoj knjizi potpuni su u smislu da mogu da se izvršavaju na računaru. Provereni su na nekoliko računara korišćenjem nekoliko različitih prevodilaca za jezik *C++*.

Izvorni tekstovi svih programa iz ove knjige mogu da se preuzmu sa Interneta, sa adrese `home.etf.rs/~kraus/knjige/`. Na istoj adresi objaviće se ispravke eventualnih, naknadno otkrivenih grešaka u knjizi. Svoja zapažanja čitaoci mogu da upute autoru elektronskom poštom na adresu `kraus@etf.rs`.

Laslo Kraus

Beograd, *decembar 2015.*

Sadržaj

1 Uvod

1.1 O programskom jeziku *C++*

Programski jezik *C* jeste jezik opšte namene, srednjeg nivoa, koji omogućava dosta intiman kontakt s hardverom računara. Poseduje strukturirane tipove podataka i upravljačke strukture (složene naredbe) što je karakteristika viših programskih jezika. S druge strane podržava manipulaciju bitovima, korišćenje procesorskih registara, pristup podacima pomoću adrese i operatore orijentisane ka hardveru računara. Ovo su karakteristike nižih programskih jezika kao što su simbolički mašinski jezici. Sintaksa jezika omogućava koncizno izražavanje i pisanje strukturiranih programa.

Programski jezik *C* projektovao je Denis Riči (*Dennis Ritchie)* 1972. godine u Belovim laboratorijama (*Bell Laboratories*). Osnovni cilj bio je sastavljanje jezika nezavisnog od računara, s karakteristikama viših programskih jezika, koji će moći da zameni simboličke mašinske (*assembler*-ske) jezike koji su, i te kako, zavisni od računara.

Dugi niz godina osnovna definicija jezika *C* bio je Referentni priručnik (*Reference Manual*) u sastavu prvog izdanja knjige *The C Programming Language* čiji su autori Brajan V. Kernigan (*Brian W. Kernighan*) i Denis M. Riči ([6]). Varijanta jezika *C* koja je opisana u toj knjizi danas se naziva *K&R C*.

Prvi zvaničan standard za jezik *C*, takozvani *ANSI C*, izdao je Američki nacionalni institut za standarde (*American National Standards Institute*) 1989. Ta verzija jezika *C* poznata je i pod imenom *C89*.

Vremenom je jezik *C* razvijan dodavanjem novih mogućnosti koje su se ozvaničavale izdavanjem novih standarda. Važeći međunarodni standard za jezik C u vreme pisanja ove knjige je iz 2011. godine poznat je pod imenom *C11*.

Pojavom novih tehnika u programiranju, prvenstveno pojavom objektno-orijentisanog programiranja, osetila se potreba za novim mogućnostima jezika *C*. Tako su 1980. godine dodate klase, provera i konverzije tipova argumenata prilikom pozivanja funkcija i još neke druge novine. Tako dobijeni jezik nazivao se *C sa klasama*.

Glavnu novinu predstavlja mogućnost definisanja novih tipova podataka u pravom smislu te reči. Dok strukture (**`struct`**) u jeziku *C* definišu samo moguće vrednosti za definisane podatke, novouvedene klase (**`class`**) definišu i moguće operacije nad tim podacima.

Štaviše, moguće je da jedine operacije nad podacima date klase budu samo one koje su predviđene definicijom te klase.

Posle daljeg proširivanja, na prvom mestu dodavanjem virtuelnih funkcija i preopterećivanja operatora, jezik je 1983/84. godine konačno dobio današnje ime *C++*. Ime treba da sugeriše da se ne radi o novom jeziku, već o proširivanju jezika *C* (++ je u jeziku *C* operator povećavanja!). Preko 95% jezika *C* usvojeno je bez izmena i u jeziku *C++*. Promenjeni su samo detalji koji su morali da budu promenjeni radi obezbeđivanja konzistentnosti novih koncepcija u jeziku *C++*. Te izmene odnose se na vrlo suptilne detalje koji dolaze do izražaja samo kod krajnje profesionalnog programiranja.

U nedostatku zvaničnog standarda, kao osnovna definicija jezika u početku se koristila knjiga *The C++ Programming Language* ([3]), čiji je autor *Bjarne Stroustrup*. On se smatra i autorom samog jezika *C++*.

Kasnije su dodate nove mogućnosti kao što su višestruko nasleđivanje, apstraktne klase, mehanizmi za sastavljanje generičkih klasa i za rukovanje izuzecima (obradu grešaka). Osnovnu definiciju jezika *C++* s početka 1991. godine predstavlja knjiga *The Annotated C++ Reference Manual* ([4]) čiji su autori *Margaret A. Ellis* i *Bjarne Stroustrup*. To je istovremeno bio i jedan od osnovnih dokumenata grupe za izradu međunarodnog standarda za jezik *C++*.

Međunarodni (*ISO*) standard za jezik *C++* usvojen je 1998. godine i poznat je pod imenom *C++98*. Tim standardom jezik *C++* je znatno proširen. Dodata je mogućnost provere tipa objekata za vreme izvršavanja programa, detaljnije su razrađene generičke funkcije i klase i definisana je bogata biblioteka gotovih klasa i funkcija za često korišćene obrade.

Drugo izdanje standarda, koje je publikovano 2003. godine, sadržalo je ispravke mana koje su u međuvremenu otkrivene u prvom izdanju. Taj standard je poznat pod imenom *C++03*.

Sredinom 2011. godine objavljeno je treće izdanje standarda koji je poznat pod imenom *C++11*. Ovim standardom unete su znatne izmene u sâm jezik, a i standardna biblioteka je znatno proširena. Posebno mogu da se istaknu lambda izrazi, kao i podrška za regularne izraze, atomičnu obradu i rad sa nitima.

Važeći međunarodni standard za jezik *C++* u vreme pisanja ove knjige jeste četvrto izdanje koje je izdato u decembru 2014. godine i poznato je pod imenom *C++14*. Ovaj standard je uglavnom samo doradio nove elemente jezika koji su uvedeni u standardu *C++11*.

U nastavku ove knjige standard *C++14* skraćeno se naziva *Standard*.

U prvom izdanju standarda trudilo se da jezik *C++98* bude u što većoj meri proširenje jezika *C89*. Drugim rečima, skoro sve što važi u jeziku *C89* važi i u jeziku *C++98*. Slično je bilo i u drugom izdanju: jezici *C99* i *C++03* su u velikoj meri kompatibilni. S tim da su u standardu *C99* preuzeti neki elementi iz standarda *C++98*.

Nažalost u trećem izdanju standarda ova dva jezika, *C11* i *C++11*, došlo je do narušavanja kompatibilnosti, čak i po nekim elementarnim pitanjima. Istina, neke novine jezika *C11* mogu da se unose u sledeće izdanje standarda jezika *C++*, ali postoje i takva odstupanja koja neće moći nikad da se otklone. Zbog toga jezik *C++* više nije nadskup jezika *C*, što je bila prvobitna ideja projektanata jezika *C++*!

Jezik *C++* danas je jedan od najmoćnijih jezika za objektno-orijentisano programiranje.

Operativni sistem *UNIX* predstavlja prirodno okruženje za jezik *C++*, kao što je to slučaj i sa jezikom *C*. Ali, s obzirom na to da se radi o jeziku opšte namene, nudi se i pod drugim operativnim sistemima, kao što je *Windows* na danas vrlo popularnim ličnim računarima.

1.2 Uvod u objektno-orijentisano programiranje

U svakom računarskom programu mogu da se uoče dve grupe elemenata: naredbe i podaci. Naredbe određuju šta se radi, a podaci čime se radi. Organizacija programa može da bude orijentisana ka jednoj od ove dve grupe elemenata.

Klasičan stil programiranja okrenut je prema postupcima i naziva se **proceduralno programiranje** (*procedural programming*). Po tom modelu program se sastoji od niza uzastopnih koraka. Logičke celine koraka mogu da se ostvaruju u obliku modula koji se nazivaju potprogrami, procedure ili funkcije. Složena obrada ostvaruje se kombinovanjem takvih modula.

Mana ovakvog pristupa programiranju je veliki stepen povezanosti delova složenog sistema. To otežava održavanje i unapređivanje sistema. Da bi se unele nove mogućnosti često je potrebno preraditi vrlo veliki deo već gotovih delova programa.

Dva programska jezika koji podržavaju proceduralno programiranje i koji su u današnje vreme vrlo rašireni jesu *Pascal* i *C*.

Savremen stil programiranja okrenut je prema podacima i naziva se **objektno-orijentisano programiranje** (*OOP – object-oriented programming*). Po tom modelu program se sastoji od **objekata** (*objects*) koji imaju neka moguća stanja i ponašanja. Stanja predstavljaju vrednosti objekata, koje vremenom mogu da se menjaju. Ponašanja predstavljaju pravila menjanja stanja, reakcije na uticaje okoline i načine uticanja na okolinu. Celokupna obrada ostvaruje se u obliku međusobnih delovanja objekata u programu.

Svi objekti u programu grupišu se po svojim osobinama. Grupe objekata sa sličnim osobinama čine **klase** (*classes*). Objekti iste klase imaju ista moguća stanja i ista ponašanja. Objekti date klase predstavljaju pojedinačne primerke svojih klasa.

Pojam klase i objekta najlakše se shvata kroz primere iz svakodnevnog života. *Pas* (klasa) je životinja sa svima poznatim opštim osobinama i ponašanjem. *Lesi* (objekat) je jedan tačno određeni pas s konkretnim osobinama (boja, težina, starost) i ponašanjem koje se uklapa u ponašanje svih pasa. Drugi primer: *automobil* je opšte ime (klasa) za sva prevozna sredstva te vrste, dok je *automobil BG 172AX* jedan tačno određeni automobil (objekat).

Klase su tipovi podataka prema definiciji tipova podataka u programiranju, jer određuju moguće vrednosti svojih primeraka i moguće radnje nad tim primercima. Po tome, objekti su podaci klasnih tipova. Za razliku od prostih tipova podataka, kao što su celi brojevi ili realni brojevi, klase su složeni tipovi podataka.

Stanja objekata predstavljaju se podacima unutar objekata koji se nazivaju **polja** (*fields*). Polja mogu da budu podaci prostih i složenih tipova. U proste tipove podataka spadaju celi brojevi, realni brojevi itd., a u složene tipove nizovi i klase.

Ponašanje objekata ostvaruje se postupcima unutar klasa koji se nazivaju **metode** (*methods*). Metode odgovaraju funkcijama i procedurama u klasičnim programskim jezicima.

Polja i metode klasa zajedničkim imenom nazivaju se **članovi** (*members*) klasa.

Prednosti objektno-orijentisanog programiranja ogledaju se u tome što je pri menjanju mogućnosti programskog sistema potrebno prerađivati samo mali deo već gotovog programa. Ako se prošire mogućnosti neke klase nije potrebno promeniti i deo programa koji koristi tu klasu. Naravno, pod uslovom da se ništa ne promeni u načinu korišćenja mogućnosti klase koje su postojale i pre promene. Takođe, ako se u programski sistem uvode nove klase, deo programa koji koristi samo stare klase ne treba promeniti.

Najviše zastupljeni jezici za objektno-orijentisano programiranje su *C++*, *Java* i, u novije vreme, *C#*.

Objektno-orijentisano programiranje zasniva se na pet osnovnih principa: apstrakciji, učaurivanju, nasleđivanju, polimorfizmu i ponovnom korišćenju koda.

1.2.1 Apstrakcija

Pod apstrakcijom (*abstraction*) podrazumeva se zanemarivanje nebitnih detalja složenih objekata, u zavisnosti od trenutnih potreba.

Na primer, da bi se koristio televizor nije potrebno poznavati od čega se sastoji i kako radi televizor. Za korisnika je dovoljno da zna za postojanje same kutije televizora i daljinskog upravljača, da zna da priključi izvor napajanja i antenu, da stavi bateriju u daljinski upravljač i da rukuje televizorom pomoću dugmadi na samom aparatu i na daljinskom upravljaču. Majstoru, pak, koji treba da popravlja televizor, potrebno je da poznaje i komponente koje sadrži televizor.

Drugi primer: da bi se čovek učlanio u biblioteku, od svih njegovih osobina, potrebni su samo lični podaci (ime, adresa, broj lične karte itd.), a pri korišćenju lifta, njegova težina.

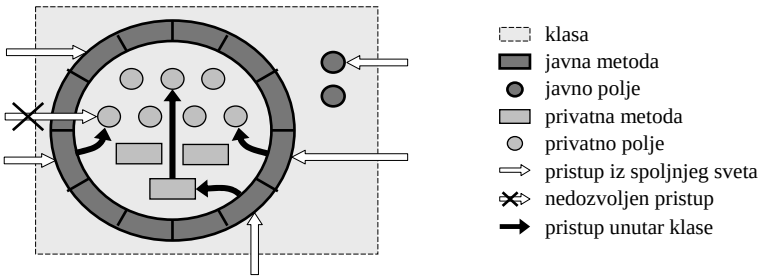
U programiranju apstrakcija se ogleda u odabiru parametara pri modelovanju predmeta ili pojmova koji su bitni za planiranu obradu.

1.2.2 Učaurivanje

Učaurivanje (*encapsulation*) u objektno-orijentisanom programiranju predstavlja mehanizam kojim se podaci i postupci u klasama štite od nepoželjnih uticaja okoline.

Na primer, televizor sadrži puno komponenata koje nisu pristupačne korisniku. Kutija uređaja čini zaštitnu površ. Vezu između korisnika i unutrašnjosti čine dugmad za upravljanje i ekran. Preko dugmadi korisnik utiče na rad televizora, a preko ekrana dobija informacije od televizora.

Učaurivanje u programiranju ostvaruje se podelom članova klase na javne i privatne (slika 1.1). **Javni članovi** (*public members*) mogu slobodno da se koriste u bilo kom delu programa, dok su **privatni članovi** (*private members*) nedostupni izvan klase. Polja su najčešće privatna dok su metode većinom javne. Do privatnih polja moguće je doći samo posredstvom javnih metoda, koje mogu da obezbede da sve promene stanja (vrednosti polja) budu u skladu s definicijom klase. Za javna polja ne može da se sprovede kontrola ispravnog korišćenja, pa njih treba maksimalno izbegavati.



***Slika 1.1** – Javni i privatni članovi klase*

Privatne metode, mada su u manjini u odnosu na javne, mogu da budu korisne. One mogu da obezbede pomoćne radnje za potrebe javnih metoda.

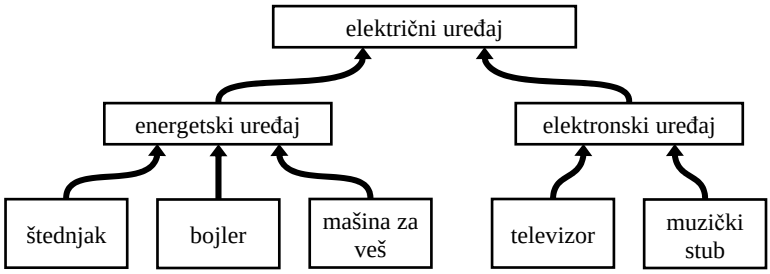
Kao primer može se uočiti klasa kalendarskih datuma. Za polja su potrebna tri cela broja koji predstavljaju dan, mesec i godinu. Ako su polja javna ne može da se obezbedi da se u objektu ne pojavi trojka celih brojeva koji ne predstavljaju ispravan datum (na primer, 31.02.2013, pri čemu je svaka komponenta za sebe unutar dozvoljenih granica za datume – drastičniji je slučaj kada je neka komponenta izvan dozvoljenih granica: 55.23.8034). Ako su polja privatna njihovo menjanje moguće je samo pomoću javnih metoda. One mogu da proveravaju da li su željene nove vrednosti ispravne i da odbiju promenu stanja objekta ako nisu.

1.2.3 Nasleđivanje

Već je rečeno da klase predstavljaju grupe srodnih objekata. Objekti unutar klase često mogu da se grupišu, prema nekim detaljnijim osobinama, u specifičnije klase. Specifičnija klasa naziva se **potklasa** (*subclass*) opštije **natklasa** (*superclass*).

Na primer, električni uređaji u domaćinstvu mogu da se podele na energetske i na elektronske uređaje. Energetski uređaji mogu da se podele na štednjake, bojlere, mašine za veš itd. Elektronski uređaji mogu da se podele na televizore, muzičke stubove itd.

Ova podela na sve specifičnije podgrupe može još da se nastavi, čime se izgrađuje proizvoljno složen hijerarhijski sistem klasa (slika 1.2). Strelice na slici okrenute su od potklasa ka natklasama, pošto se za potklasu zna koja je njena natklasa. Na primer, televizor je elektronski uređaj koji je, pak, električni uređaj. Obrnuto, ako se razmatra pojam elektronskog uređaja, ne može da se zaključi koje sve vrste takvih uređaja postoje. Štaviše, broj podvrsta elektronskih uređaja vremenom može i da se širi.



***Slika 1.2** – Hijerarhija klasa*

Zajednička karakteristika za sve vrste električnih uređaja je da troše električnu energiju. Energetski uređaji, osim činjenice da troše električnu energiju, imaju zajedničku osobinu da troše mnogo energije. Štednjaci, osim toga što troše mnogo električne energije, imaju grejne ploče i rerne. Treba uočiti da specifičnije grupe uređaja imaju sve osobine opštije grupe i još neke dodatne specifične osobine.

Pošto potklasa, osim svojih specifičnih osobina, poseduje sve osobine svoje natklase, kaže se da potklasa **nasleđuje** (*inherits*) svoju natklasu.

U objektno-orijentisanom programiranju nasleđivanje znači da u potklasi postoje sva polja i metode natklase, kao i neka dodatna polja i metode specifične za potklasu. Dodatna

polja obezbeđuju dodatna moguća stanja, a metode dodatna moguća ponašanja objekata potklase u odnosu na objekte natklase.

Na primer, geometrijske figure mogu da budu određenih boja. Krugovi, kao geometrijske figure, dodatno su okarakterisani svojim poluprečnicima, a pravougaonici dužinama stranica. Svaki krug, kao objekat klase krugova, ima svoju boju i dužinu poluprečnika. Slično tome, svaki pravougaonik, kao objekat klase pravougaonika, ima svoju boju i dužine dve stranice.

1.2.4 Polimorfizam

Pod polimorfizmom (*polymorphism*) u objektno-orijentisanom programiranju podrazumeva se sposobnost programa da se prilagođava tipu podataka koji se upravo obrađuju. Na primer, ako se obrađuje skup geometrijskih figura i treba izračunavati površine tih figura. Površina različitih vrsta figura, kao što su krugovi, pravougaonici i trouglovi, računa se na različite načine, pomoću različitih formula. Polimorfizam obezbeđuje da u programu ne treba za svaku pojedinačnu figuru ispitivati kojoj vrsti pripada i primeniti odgovarajuću formulu. Dovoljno je zatražiti da se izračuna površina date figure, a prepoznavanje vrste figure i primena odgovarajuće formule dešavaju se automatski.

Drugačije rečeno, pri obradi geometrijskih figura treba da postoji klasa figura u kojoj je predviđeno izračunavanje površine figure. Za svaku konkretnu vrstu figura treba da postoji potklasa, a u njoj metoda za izračunavanje površine te vrste figura prema odgovarajućoj formuli a na osnovu dimenzija koje su polja klase. Kada se zatraži izračunavanje površine neke figure koristiće se metoda iz potklase kojoj ta figura pripada.

Treba još uočiti da se u zajedničkoj natklasi figura izračunavanje površine i ne može nikako sprovesti. Dok se ne zna o kakvoj vrsti figura se zapravo radi, ne može da se uradi ništa što bi imalo smisla. Ovakve situacije nisu retke u objektno-orijentisanom programiranju.

Metoda za izračunavanje u natklasi je tada prazna, odnosno, stručno rečeno, ona je **apstraktna metoda** (*abstract method*). I za samu natklasu se tada kaže da je **apstraktna klasa** (*abstract class*), jer nema smisla stvarati objekte tipa takve klase.

U posmatranom primeru geometrijska figura je apstraktan pojam, jer ne mogu da postoje geometrijske figure kao takve, već samo krugovi, pravougaonici i trouglovi. Pojam geometrijska figura služi samo zato da se objedinjeno iskažu neke zajedničke osobine svih vrsta figura. Na primer, da im se može izračunavati obim i površina.

1.2.5 Ponovno korišćenje koda

Radi što efikasnijeg programiranja poželjno je da se jednom napisani programski delovi mogu iskoristiti više puta, i to bez ponovnog pisanja.

U klasičnom, proceduralnom programiranju, to se svodi na upotrebu biblioteka funkcija opšte namene u više programa.

Taj vid ponovnog korišćenja koda postoji i u objektno-orijentisanom programiranju, pri čemu se tu prave biblioteke klasa opšte namene.

Nasleđivanje predstavlja drugi vid ponovnog korišćenja koda. Metode napisane u natklasi nasleđuju se u potklasama, pa ih ne treba ponovo pisati. U potklasama je potrebno napisati samo metode koje obezbeđuju dodatne funkcionalnosti u odnosu na natklasu.