Objektno orijentisano programiranje u realnom vremenu na jeziku C++
Beograd, 1996.
Objektno orijentisano programiranje u realnom vremenu na jeziku C++ 1
Deo I Objektno orijentisano programiranje i modelovanje
Objektno orijentisano programiranje u realnom vremenu na jeziku C++ 2
Uvod
*
Jezik C++ je objektno orijentisani programski jezik opšte namene. Veliki deo jezika C++ nasleđen je iz jezika C, pa C++ predstavlja (uz minimalne izuzetke) nadskup jezika C. ∗ Kurs uvodi u osnovne koncepte objektno orijentisanog programiranja i principe projektovanja objektno orijentisanih softverskih sistema, korišæenjem jezika C++ kao sredstva. ∗ Kurs je baziran na referencama [ARM] i [Miliæev95]. Knjiga [Miliæev95] predstavlja osnovu ovog kursa, a u ovom dokumentu se nalaze samo glavni izvodi. Kurs sadrži i najvažnije elemente jezika C.
Zašto OOP?
∗ Objektno orijentisano programiranje (Object Oriented Programming, OOP) je odgovor na tzv. krizu softvera. OOP pruža naèin za rešavanje (nekih) problema softverske proizvodnje. Softverska kriza je posledica sledeæih problema proizvodnje softvera: ∗ 1. Zahtevi korisnika su se drastièno poveæali. Za ovo su uglavnom "krivi" sami programeri: oni su korisnicima pokazali šta sve raèunari mogu, i da mogu mnogo više nego što korisnik može da zamisli. Kao odgovor, korisnici su poèeli da traže mnogo više, više nego što su programeri mogli da postignu. 2. Neophodno je poveæati produktivnost programera da bi se odgovorilo na zahteve korisnika. To je moguæe ostvariti najpre poveæanjem broja ljudi u timu. Konvencionalno programiranje je nametalo projektvanje softvera u modulima sa relativno jakom interakcijom, a jaka interakcija između delova softvera koga pravi mnogo ljudi stvara haos u projektovanju. 3. Produktivnost se može poveæati i tako što se neki delovi softvera, koji su ranije veæ negde korišæeni, mogu ponovo iskoristiti, bez mnogo ili imalo dorade. Laku ponovnu upotrebu koda (software reuse) tradicionalni naèin programiranja nije omoguæavao. 4. Poveæani su drastièno i troškovi održavanja. Potrebno je bilo naæi naèin da projektovani softver bude èitljiviji i lakši za nadgradnju i modifikovanje. Primer: èesto se dešava da ispravljanje jedne greške u programu generiše mnogo novih problema; potrebno je "lokalizovati" realizaciju nekog dela tako da se promene u realizaciji "ne šire" dalje po ostatku sistema. ∗ Tradicionalno programiranje nije moglo da odgovori na ove probleme, pa je nastala kriza proizvodnje softvera. Poveæane su režije koje prate proizvodnju programa. Zato je OOP došlo kao odgovor.
Šta daju OOP i C++ kao odgovor?
C++ je trenutno najpopularniji objektno orijentisani jezik. Osnovna rešenja koja pruža OOP, a C++ podržava su: 1. Apstrakcija tipova podataka (Abstract Data Types). Kao što u C-u ili nekom drugom jeziku postoje ugrađeni tipovi podataka (int, float, char, ...), u jeziku C++ korisnik može proizvoljno definisati svoje tipove i potpuno ravnopravno ih koristiti (complex, point, disk, printer, jabuka, bankovni_racun, klijent itd.). Korisnik može deklarisati proizvoljan broj promenljivih svog tipa i vršiti operacije nad njima (multiple instances, višestruke instance, pojave). 2. Enkapsulacija (encapsulation). Realizacija nekog tipa može (i treba) da se sakrije od ostatka sistema (od onih koji ga koriste). Treba korisnicima tipa precizno definisati samo šta se sa tipom može raditi, a naèin kako se to radi sakriva se od korisnika (definiše se interno). 3. Preklapanje operatora (operator overloading). Da bi korisnièki tipovi bili sasvim ravnopravni sa ugrađenim, i za njih se mogu definisati znaèenja operatora koji postoje u jeziku. Na primer, ako je korisnik definisao tip complex, može pisati c1+c2 ili c1*c2, ako su c1 i c2 promenljive tog tipa; ili, ako je r promenljiva tipa racun, onda r++ može da znaèi "dodaj (podrazumevanu) kamatu na raèun, a vrati njegovo staro stanje". 4. Nasleđivanje (inheritance). Pretpostavimo da je veæ formiran tip Printer koji ima operacije nalik na

print_line, line_feed, form_feed, goto_xy itd. i da je njegovim korišæenjem veæ realizovana velika količina softvera. Novost je da je firma nabavila i štampaèe koji imaju bogat skup stilova pisma i želja je da se oni ubuduæe iskoriste. Nepotrebno je ispoèetka praviti novi tip štampaèa ili prepravljati stari kôd. Dovoljno je kreirati novi tip PrinterWithFonts koji je "baš kao i obièan" štampaè, samo"još može da" menja stilove štampe. Novi tip æe naslediti sve osobine starog, ali æe još ponešto moæi da uradi. 5. Polimorfizam (polymorphism). Pošto je PrinterWithFonts veæ ionako Printer, nema razloga da ostatak programa ne "vidi" njega kao i obièan štampaè, sve dok mu nisu potrebne nove moguænosti štampaèa. Ranije napisani delovi programa koji koriste tip Printer ne moraju se uopšte prepravljati, oni æe jednako dobro raditi i sa novim tipom. Pod određenim uslovima, stari delovi ne moraju se èak ni ponovo prevoditi! Karakteristika da se novi tip "odaziva" na pravi naèin, iako ga je korisnik "pozvao" kao da je stari tip, naziva se polimorfizam.