

Objektno orijentisano programiranje 1

Uvod



Motivacija

- Velika složenost savremenih programskih sistema
- Trendovi softverske industrije:
 - u pravcu sve složenijih programskih sistema
- Projektovanje i implementacija, a naročito održavanje, složenih programskih sistema je izuzetno skupo
- Pod održavanjem se podrazumeva:
 - otklanjanje grešaka i
 - funkcionalno proširivanje sistema
- Osnovni razlog visoke cene:
 - slaba reupotrebljivost programskog koda

Korišćenje postojećeg koda

- Korišćenje postojećeg koda u tradicionalnim jezicima:
 - na nivou biblioteka funkcija
- Funkcionalno proširenje aplikacije
 - menja se struktura podataka i funkcije koje rade nad tim podacima
 - dva rešenja za staru funkciju koja više ne odgovara:
 - pisati novu funkciju (stari kod poziva staru, novi kod – novu)
 - mora da se vodi računa koja funkcija se odakle poziva – skloni grešci
 - menjati postojeću funkciju
 - već testirani kod postaje ponovo sklon greškama

Cilj OO jezika i OO programiranja

- Jedan od osnovnih ciljeva OO jezika:
 - bolja podrška za reupotrebu postojećeg koda
- Važna karakteristika OO jezika: podrška za izvođenje novih tipova iz postojećih
 - prilikom izvođenja, novi tip:
 - nasleđuje svojstva osnovnog tipa i
 - dodaje neka specifična svojstva
 - podatak izvedenog tipa može zameniti podatak osnovnog
- Stari programski kod se ne mora menjati i ponovo testirati
 - nova funkcionalnost se lako dodaje

Elementi OO stila programiranja

- Apstrakcija
 - prepoznavanje bitnih svojstava skupova sličnih objekata i opisivanje zajedničkom klasom
- Kapsulacija implementacije
 - reprezentacije stanja i realizacije ponašanja
- Objavljivanje interfejsa
 - kroz "potpise" metoda (ime i lista argumenata)
- Izvođenje novih tipova (klasa)
 - uz nasleđivanje svih elemenata osnovne klase i
 - dodavanje specifičnih elemenata
- Dinamičko povezivanje i polimorfizam

Evolucija programskih jezika

- U evoluciji programskih jezika - tri značajna napretka:
 - Apstrakcija izraza (rane '50.) FORTRAN
Registri mašine su postali skriveni za programera
 - Apstrakcija kontrole (rane '60.) Algol60
Strukturiran tok kontrole programa
 - Apstrakcija podataka (rane '70.) Pascal
Razdvajanje detalja prezentacije podataka od apstraktnih operacija koje se definišu nad podacima (npr. tip nabiranja)
- Još neki koncepti koji su otvorili put OO jezicima:
 - zasebno prevođenje modula (FORTRAN, C, Ada)
 - razdvajanje specifikacije (interfejsa) od implementacije (Ada)
 - koncept klase: apstrakcija objekta i kapsulacija podataka i pridruženih operacija (Simula67).

Istorijat razvoja jezika C++

- 1972. **C** – *D. Ritchie* u *Bell* laboratorijama
- 1980. "**C sa klasama**" za simulacije vođene događajima
- 1983. **C++** *B. Stroustrup* u *AT&T Bell* laboratorijama
- 1989. ANSI standard za C
- 1997. ANSI/ISO prihvaćen finalni draft standarda za C++
- 1998. publikovan C++ standard ISO/IEC 14882:1998
- 2003. revizija C++ standarda ISO/IEC 14882:2003
- 2011. novi C++ standard ISO/IEC 14882:2011
- 2014. aktuelni standard ISO/IEC 14882:2014
- Postoje već draftovi standarda 2017 i 2020
- Telo za standardizaciju:
 - JTC1/SC22/WG21 - The C++ Standards Committee

Motiv za razvoj C++

- Dekomponovanje kompleksnih softverskih sistema na prirodan način radi bržeg razvoja i lakšeg održavanja
- Aspekti koje je (po Stroustrupu) trebalo da ispuni jezik:
 - da bude dovoljno blizak mašini – lako upravljanje arhitekturom
 - ispunjava jezik C koji je iskorišćen kao osnova za jezik C++
 - jezik C++ je praktično nadskup jezika C (ima odstupanja)
 - ime jezika C++ je simbolično (to je inkrementirani C)
 - da bude dovoljno blizak problemu koji se rešava
 - da se ideje mogu izraziti neposredno i jezgrovito
 - inspiracije: Simula67 (klase), Algol (preklapanje operatora)
- Spontani razvoj:
nikada nije postojao "odbor za projektovanje C++"

Pregled – klase i objekti (1)

- OO jezici proširuju pojam konvencionalnog tipa podataka uvođenjem pojma *klase*
- Klase su apstrakcije zajedničkih atributa i zajedničkog ponašanja jednog skupa srodnih objekata
- Klase sadrže:
 - podatke članove (atributi ili polja – definišu stanje) i
 - funkcije članice (metodi – definišu ponašanje)
- Pristupačnost određenim članovima klase (npr. javni i privatni) deklariraju programeri

Pregled – klase i objekti (2)

- Klasa ima svoj interfejs i svoju implementaciju
 - implementaciju klase čine:
 - privatni podaci članovi i
 - definicije (zaglavlje sa telom) funkcija
 - interfejs klase čine:
 - javni podaci članovi (ne preporučuje se) i
 - deklaracije (zaglavlja) javnih funkcija
- Primerci odgovarajućih klasa nazivaju se objektima
- Objekat je određen stanjem, ponašanjem i identitetom

Primer: klasa za opis publikacija

```
class Publikacija {
public:
    virtual void unos();
    virtual void ispis();
private:
    int id;
    char naslov[150];
};

#include <iostream>
using namespace std;
void Publikacija::unos()
    {cout<<"Id? "; cin>>id;      cout<<"Naslov? "; cin>>naslov; }
void Publikacija::ispis()
    {cout<<"Id: "<<id<<"", Naslov: "<<naslov;}
```

Pregled - konstruktori i destruktori

- Prilikom kreiranja i uništavanja objekata pogodno je da se obave izvesne aktivnosti
 - pri kreiranju objekta treba dovesti objekat u početno stanje inicijalizacijom njegovih članova
 - pri uništavanju složenih objekata delovi treba da se uništavaju automatski pri uništenju celine
- Konstruktori su specifične funkcije članice koje se automatski izvršavaju pri kreiranju objekata
- Destruktori su specifične funkcije članice koje se automatski izvršavaju pri uništavanju objekata

Primer: klasa za opis publikacija (2)

```
class Publikacija {  
public:  
    Publikacija();           // podrazumevani konstruktor  
    Publikacija(int, char*); // konstruktor sa dva argumenta  
    //...  
};  
  
#include <cstring>  
Publikacija::Publikacija(){} // prazno telo konstruktora  
Publikacija::Publikacija(int i, char* n)  
    {id=i;strcpy(naslov,n);}  
void main ()  
    {Publikacija knjiga1(1,"Alhemicar"), knjiga2;}
```

Pregled - izvođenje i nasleđivanje

- Iz opštije klase moguće je izvoditi više specifičnih klasa
- Na primer: iz klase geometrijskih tela mogu se izvoditi klase sfere, kvadra, prizme, valjka, itd.
- Izvedene klase nasleđuju attribute i metode osnovne klase i dodaju nove attribute i metode
- Objekti izvedenih klasa:
 - specijalne vrste objekata osnovne klase
 - oni mogu zameniti u izrazima objekat osnovne klase
- Nasleđeni metodi se mogu redefinisati

Pregled - polimorfizam

- Ako se u osnovnoj klasi funkcija proglašava virtuelnom – na nju se primenjuje dinamičko vezivanje
- Dinamičko (ili kasno) vezivanje funkcija
 - adresa se ne određuje u vreme prevođenja/povezivanja
 - poziv se vezuje za funkciju u vreme izvršenja
- Polimorfizam:
 - ponašanje objekta na koji ukazuje pokazivač ne zavisi samo od tipa pokazivača, već od tipa pokazanog objekta
- Polimorfizam omogućava jedinstvenu obradu objekata osnovne i izvedenih klasa

Primer: klasa za opis publikacija (3)

```
class Casopis: public Publikacija {
public:
    Casopis();
    Casopis(int, int, int, char*);
    void unos(); //redefinicija funkcije Publikacija::unos()
    void ispis(); //redefinicija funkcije Publikacija::ispis()
private:
    int broj, god;
};

...

void ispisi_sve(Publikacija *p[], int n){
    for (int i=0; i<n; i++) { p[i]->ispis(); cout<<endl; }
}
```


Klasifikacija objektnih jezika

- *Objektno-bazirani* jezici podržavaju
 - apstrakciju, kapsulaciju i modularnost
 - primeri: Ada83, Visual Basic V6
- *Objektno-orijentisani* jezici dodatno podržavaju
 - princip nasleđivanja
 - primeri: Simula, Smalltalk, Eiffel, Ada95, C++, Java, Visual Basic.NET, C#

Pregled - obrada izuzetnih situacija

- Često se u praksi pojavljaju izuzetne situacije pri izvršenju programa
- Primeri ovakvih situacija su otvaranje nepostojeće datoteke, prekoračenje opsega indeksa i sl.
- U tradicionalnim jezicima izuzetne situacije se obrađuju u pojedinim granama selekcija
 - kada se u nekoj funkciji pojavi izuzetak funkcija vraća neku vrednost koja signalizira izuzetak
 - ovako vraćena vrednost se analizira u kodu koji je pozvao funkciju i preduzimaju se akcije
- Kod postaje prilično nepregledan – obrada izuzetnih situacija zamagljuje osnovnu obradu

Pregled - izuzeci u OO jezicima

- Mehanizam obrade izuzetaka nije svojstven samo OO jezicima (npr. Ada83 podržava izuzetke)
- Iako mehanizam nije svojstven samo OO jezicima – dobro se uklapa u objektnu koncepciju
 - u C++ izuzetak je objekat koji se "baca" (*throw*) u trenucima kada se pojavi izuzetna situacija
 - delovi koda se pišu tako da se "pokuša" (*try*) neka obrada
 - ako se pojavi izuzetak pokušana obrada se automatski prekida i prelazi se na obradu izuzetka
 - izuzetak se "hvata" (*catch*) ukoliko se dogodi i prepoznaje na osnovu klase objekta izuzetka

Primer: klasa za opis publikacija (4)

```
class LosBroj{
    int broj;
public:
    LosBroj(int i){broj=i;}
    void poruka() {cout<<"Nedozvoljen broj! "<<broj<<endl;}
};

void Casopis::unos() {
    Publikacija::unos();
    try {
        cout<<"Broj? "; cin>>broj;
        if (broj<1 || broj>12) throw LosBroj(broj);
        cout<<"Godiste? "; cin>>god;
    }
    catch (LosBroj izuzetak) { izuzetak.poruka(); broj=0; god=0; }
}
```

Pregled - preklapanje operatora

- Koncept nije nužno OO, ali se dobro uklapa u OO paradigmu
- Standardni jezički operatori se mogu preklopiti novim definicijama
- Preklopljeni operator definiše operaciju nad korisničkim (klasnim) tipom podataka
- Piše se funkcija `operator<simbol> (<parametri>)`
- Ne mogu se preklopiti svi operatori, a za neke važe specijalna pravila
- Na primer: uvodi se klasa `Clanak` i operatorska funkcija `Casopis::operator+=(Clanak)`
 - časopis predstavlja zbirku članaka
 - operator `+=` dodaje novi članak časopisu

Pregled - šabloni

- Određene obrade ne zavise od tipa podataka:
 - stek i kružni bafer implementiraju LIFO, odnosno FIFO, protokol bez obzira na tip elementa
- Pogodno je da se klase i funkcije mogu pisati generički, parametrizovano tipovima podataka
- Takve generičke klase i funkcije nazivaju se u jeziku C++ šablonima (*templates*)
- Iz šablona se generišu stvarne klase, odnosno funkcije, za konkretne tipove (stvarne parametre)
- Generički mehanizam je u potpunosti statički
 - zamena parametara je u vreme prevođenja

Primer: klasa za opis publikacija (5)

```
template <class T> class SablonPublikacija{
    T id;                // identifikator je proizvoljnog tipa
    //...
};
void main(){
    SablonPublikacija<int> knjiga; // generisana klasa
    //...
}
```