

Podrška objektno orijentisanom programiranju u jezicima C++, Objective C, Java, C#, Ada i Ruby

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Katarina Popović, Dušan Pantelić, Dejan Bokić, Nikola Stojević
kontakt email prvog, pantelic.dusan@protonmail.com, trećeg, četvrtog autora

3. april 2019

Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da koristite!) kao i par tehničkih pomoćnih uputstava. Pročitajte tekst pažljivo jer on sadrži i važne informacije vezane za zahteve obima i karakteristika seminarskog rada.

Sadržaj

1 Uvod

Kada budete predavali seminarski rad, imenujete datoteke tako da sadrže redni broj teme, temu seminarskog rada, kao i prezimena članova grupe. Precizna uputstva na temu imenovanja će biti data na formi za predaju seminarskog rada. Predaja seminarskih radova biće isključivo preko veb forme, a NE slanjem mejla. Link na formu će biti dat u okviru obaveštenja na strani kursa. Vodite računa da prilikom predavanja seminarskog rada predate samo one fajlove koji su neophodni za ponovno generisanje pdf datoteke. To znači da pomoćne fajlove, kao što su .log, .out, .blg, .toc, .aux i slično, **ne treba predavati**.

2 C++

Jezici koji podržavaju sve 4 funkcionalnosti objektno orijentisane paradigme ali ne u potpunosti se obično nazivaju delimično objektno orijentisanim jezicima. Zbog sledećih karakteristika, C++ se smatra delimično objektno orijentisanim jezikom.

1. **Main funkcija je izvan klase :** U C++ može da se napiše validan, ispravan kod bez kreiranja ijednog objekta. Main funkcija je obavezna ali se ona nalazi izvan svake klase, što nije karakteristično za druge OOP jezike.
2. **Koncept globalne promenljive:** U C++ može da se kreira globalna promenljiva koja je dostupna svugde u kodu, dok u drugim OOP jezicima promenljive mogu biti deklarisanе samo u okviru klase gde mogu da se koriste modifikatori pristupa (private, protected, public).
3. **Postojanje friend funkcija:** Friend (prijateljska) funkcija može pristupiti privatnim poljima klase kojoj je deklarisanа kao prijateljska. Ovo je jedna veoma korisna karakteristika C++, ali i dalje narušava neke koncepte objektno orijentisane paradigme.

2.1 Enkapsulacija

U C++ ne mora eksplicitno za svaki atribut ili metod klase da bude nagašeno pravo pristupa, nego se prave sekcije, i na početku sekcije se stavi modifikator pristupa (podržani su private, public i protected, ne podržava package modifikator). Ukoliko se modifikator ne navede eksplicitno, kod klase se podrazumeva private, dok kod struktura se podrazumeva public (što je jedna od osnovnih razlika između struktura i klase u C++).

```
1000 class Employee {
      private:
1002     int salary;
      public:
1004     Employee(int e_salary) { salary = e_salary;}
      int getSalary(){ return salary;}
1006     void setSalary(int newSalary) { salary = newSalary;}
      void display() {
1008         std::cout << "Hello I'm the employee!" << std::endl;
      }
1010 };
```

Listing 1: Primer deklarisanja klase sa enkapsulacijom

2.2 Nasleđivanje

U jeziku C++ i nasleđivanje može biti private, protected ili public. Ukoliko je nasleđivanje public, to znači da sva nasleđena polja ostaju javna, ukoliko je protected, tada će sva public polja postati protected, a ukoliko je nasleđivanje private, to znači da će sva public i protected polja postati private. Takođe, za razliku od drugih OOP jezika, u C++ je podržano i višestruko nasleđivanje, gde jedna klasa može da nasledi više od jedne klase. Zbog problema koje višestruko nasleđivanje može da uvede, u C++ je uvedena još jedna ključna reč prilikom nasleđivanja-vritual, koja sprečava tzv. dijamant strukturu.

```
1000 class Driver: public Employee {
      private:
```

```

1002     std::string truck = "FAP";
public:
1004     Driver(int salary, std::string truck)
        : Employee(salary), truck(truck)
1006     {};
        void display() {
1008         std::cout << "My truck is " << truck << "!" << std::endl; };
    };

```

Listing 2: Primer nasleđivanja klasa u C++

2.3 Polimorfizam

U C++ su podržana dva osnovna tipa polimorfizma- **polimorfizam u vreme kompilacije** i **polimorfizam u vreme izvršavanja**. Prvi obezbeđuje preopterećenje metoda i operatora. Preopterećenje operatora je isto jedna od C++ specifičnih mogućnosti, gde možemo sami da definišemo ponašanje operatora npr "+" dok god ispunjava svoje osnovne karakteristike (da ima 2 argumenta). Drugi tip polimorfizma omogućuje premošćavanje metoda, tj. za metod se kaže da je premošten ukoliko izvedena klasa poseduje metod sa identičnim potpisom.

2.4 Apstrakcija

Za razliku od drugih OOP jezika, C++ ne poseduje ključnu rec *abstract*. Apstraktne klase se u C++ kreiraju tako što se napravi virtuelna metoda i u potpisu joj se dodeli 0. Takva klasa ne može biti instancirana, ali može biti napravljen pokazivač na nju. Takođe, apstraktna klasa može da ima konstruktor i destruktork. Klasa koje ne premosti virtuelnu metodu takođe postaje apstraktna klasa. Postoji nekoliko pravila koja moraju da važe za apstraktne klase:

1. Moraju biti proglašene javnim (inače potklasa ne može da ih premosti).
2. Virtuelne metode ne mogu biti static i ne mogu biti prijateljske metode neke druge klase.
3. Virtuelnim metodama se mora pristupati preko pokazivača na baznu klasu da bi se dobio pravi polimorfizam u vreme izvršavanja.
4. Potpis virtuelne metode mora biti identičan i u baznoj i izvedenoj klasi (povratna vrednost, tipovi argumenata, konstantnost argumenata,...).
5. Klase mogu imati virtuelni destruktork, ali ne mogu da imaju virtuelni konstruktor.

3 Objective C

Deo za Objective C.

4 Java

Objekte klasa instanciramo pomoću metoda konstruktora(nema povratni tip i uvek se zove isto kao i klasa) sa odgovarajućim argumentima. Ako ne definišemo konstruktor, automatski se generiše podrazumevani

konstruktor, koji je prazan i nema argumente. U slučaju da nema argumente, inicijalizuje objekat na podrazumevane vrednosti. Sledeći primer (??) predstavlja deklaraciju klase zaposleni, koja ima svoje atribute i metode(detalnije ??).

```

1000 public class Employee {
1001     private int salary;
1002     #this je referenca na tekuci objekat
1003     public Employee(int salary) { this.salary = salary;}
1004     public int getSalary(){ return salary;}
1005     public void setSalary(int newSalary) { salary = newSalary;}
1006     public void display() {
1007         System.out.println("Hello i'm employee!");
1008     }
1009     public static void main(String[] args) {
1010         Employee Marko = new Driver(600,"Mercedes");
1011         Marko.display();
1012     }
1013     class Driver extends Employee {
1014         String truck = "FAP";
1015         #super vrsi poziv konstruktora bazne klase
1016         public Driver(int salary,String truck) {
1017             super(salary); this.truck = truck;}
1018         public void display() {
1019             System.out.println("My truck is "+truck+"!");
1020         }
1021         public void display(String x) {
1022             System.out.println("My truck is "+truck+x+"!");
1023         }
1024     }

```

Listing 3: Primer deklarisanja klase sa enkapsulacijom i nasleđivanjem

4.1 Enkapsulacija

Ograničavanje pristupa internim podacima klase postizemo navođenjem ključne reči **private** ispred deklaracije promenljive u klasi. Ovo znači da se podacima može pristupiti isključivo iz deklarisanе klase. Podacima neophodnim za funkcionalnost programa obezbeđuje se pristup čitanja i menjanja (eng. *getters and setters*)[?] preko javnih metoda. U primeru koda (??), vrednosti privatnog atributa plata možemo pristupiti metodom `getSalary()` ili menjati sa `setSalary(newSalary)`.

4.2 Nasleđivanje

Za označavanje koristimo ključnu reč **extends**. Podela po artiklu [?]:

- Po nivoima, kada klasu A nasleđuje klasa B, a nju nasleđuje klasa C.
- Hijerarhijsko nasleđivanje, gde klase B i C nasleđuju klasu A.
- Višestruko nasleđivanje(nasleđivanje više klase) nije moguće, već se implementira preko interfejsa(detalnije ??).

U primeru koda (??), klasa vozač nasleđuje klasu zaposleni.

4.3 Polimorfizam

Višestruka upotrebljivost koda za različite vrste objekata.

Pripadnost metodu objektu se obavlja u vreme izvršavanja(eng. *run time execution*) i predstavlja koncept važnosti metoda(eng. **overriding**)[?]. U primeru koda ??, `Marko.display()`; pozvaće metod klase vozač.

Koncept prenatrpanosti metoda(eng. *overloading*)[?], određuje metode u vremenu kompajliranja(eng. *compile time*) na osnovu razlika u potpisu metode(različito ime metoda ili tipovi i broj parametara). U primeru koda [??], Marko.display(2); pozvaće metod display(int x) klase vozač.

4.4 Apstrakcija

Izdvajanje skupa metoda sa kojima spoljašnji korisnik komunicira, prema artiklu [?], vršimo pomoću apstraktnih klasa ili interfejsa.

Za apstraktne klase navodimo ključnu reč *abstract*(kod ??). Ne mogu se instancirati, ali može biti tip promenljive. Sadrže apstraktne metode(istom ključnom reči obeležavaju) koje treba da predefiniše neka podklasa.

```
1000 public abstract class Employee {  
      public abstract void display(); ...  
}
```

Listing 4: Apstraktna klasa

Interfejs predstavlja nacrt klase. Sadrži apstraktne, statične, podrazumevane metode(mogu se predefinisati u klasi) i statičke promenljive. Da implementiramo interfejs navodimo ključnu reč *implements*(kod ??) i zatim ime interfejsa(slično nasleđivanju). Prednost interfejsa[?] je da klasa može implementirati više interfejsa, dok može da nasleđuje samo jednu klasu.

```
1000 interface Employee {  
      public void display(); #podrazumevano apstraktna  
1002      default void work(){System.out.println("Working"); }  
      class Driver implements Employee{  
1004          public void display(){...}  
      }
```

Listing 5: Interfejs

5 C#

C# je jednostavan, moderan, objektno-orijentisan jezik, razvijen od strane Microsoft-a I odobren od strane ECMA-e(European Computer Manufacturers Association). Nudi punu podrsku objektno orijentisanom programiranju ukljucujuci nasledjivanje, enkapsulaciju, apstrakciju, I polimorfizam.

- Nasledjivanje je, kao sto samo ime kaze, mogucnost da “nasledjuje” metode I svojstva od postojećih klasa.
- Enkapsulacija je kada se grupa od povezanih metoda, svojstava I ostalih članova tretira kao jedan isti objekat.
- Apstrakcija je proces kod koga programer krije sve osim relevant-nih podataka o datom objektu u cilju pojednostavljanja I povecanja efikasnosti.

```
1000 abstract class MobilePhone {  
1001     public void Calling();  
1002     public void SendSMS();  
1003 }  
1004 public class Nokia1400: MobilePhone {}  
1005 public class Nokia2700: MobilePhone {  
1006     public void FMRadio();  
1007     public void MP3();  
1008     public void Camera();  
1009 }  
1010 public class BlackBerry: MobilePhone {  
1011     public void FMRadio();  
1012     public void MP3();  
1013     public void Camera();  
1014     public void Recording();  
1015     public void ReadAndSendEmails();  
1016 }
```

Listing 6: Primer deklarisanja apstraktivne klase u C#-u

Ne podrzava druge paradigme ali koristi svoje imperativne strukture. Veome je slicna podrška OOP-u kao kod Java programskog jezika takodje su iste I klase I strukture.

5.1 Nasledjivanje

Koristi sintaksu `c++` za definisanje klase. Nasledjena metoda od roditeljske klase moze biti zamenjena u izvedenoj klasi tako sto se definise kao `new(novo)`. Verzija roditeljske klase se I dalje moze zvati eksplicitnom sa prefiksom `base(baza): base.Draw()`.

5.2 Dinamicko vezivanje

Da bi se dozvolilo dinamicko vezivanje metoda pozivaju se metode:

- Bazicna klasa metode, oznacava se kao `virtual`
- Odgovarajuće metode u izvedenim klasama, oznacene su kao `override`
- Apstraktne metode, oznacene su kao `abstract` I moraju biti implementirane u svim podklasama.

Sve C# klase su izvedene od jedinog korena klase, Objekta.

5.3 Ugnjezdene klase

C# klasa koja je direktno ugnjezdena u ugnjezdene klase se ponasa kao staticka java ugnjezdena klasa.

C# ne podrzava ugnjezdene klase koje se ponasaju kao ne staticke java klase.

5.4 Evaluacija

C# je najskorije dizajniran OOP jezik na bazi C-a.

Razlika izmedju podrške C#-a I Javine podrške za OOP su relativno male.

6 Ada

Deo za Ada.

7 Ruby

Podršku u programskom jeziku Ruby ilustrovaćemo primerom(??) koji pokriva sve bitnije aspekte objektno orijentisanog programiranja. Standardni metod klase je **initialize**, on se poziva automatski prilikom kreiranja objekta i ponaša se skoro identično kao konstruktori u drugim programskim jezicima.

```
1000 class Employee
1001   attr_accessor :name
1002   def initialize(name)
1003     @name = name
1004     print()
1005   end
1006   def print
1007     puts "Employee: #{@name}."
1008   end
1009 end
1010 class Driver < Employee
1011   def initialize(name)
1012     @name = name
1013     print()
1014   end
1015   private
1016   def print
1017     puts "Driver: #{@name}."
1018   end
1019 end
1020
1021 emp = Employee.new("John")
1022 drv = Driver.new("John")
```

Listing 7: Primer objektno orijentisanog programiranja u jeziku Ruby.

7.1 Enkapsulacija

Kako u samom jeziku ne postoji mogućnost direktnog pristupa podacima unutar klase(podaci su privatni), njima možemo pristupiti jedino pomoću metoda klase. Svi metodi klase su javni, osim ako nije eksplicitno naznačeno drugačije ključnim rečima **public** **protected**, **private** neposredno pre definicije jednog ili više metoda. Ruby nam pruža mogućnost ugrađenih metoda za pristup(eng. *accessor methods*). U primeru(??) **attr_accessor** omogućava čitanje i menjanje vrednosti promenljivih klase. Pomocu **attr_reader** i **attr_writer** možemo pojedinačno dopustiti samo čitanje odnosno samo menjanje vrednosti promenljivih.

7.2 Nasleđivanje

Kada nakon imena klase u njenoj definicije dodamo znak < za kojim sledi ime već postojeće klase, dobijamo efekat nasleđivanja koji možemo videti u prethodnom primeru(??) gde klasa *Driver* nasleđuje klasu *Employee*(primetiti da u klasi *Driver* nismo implementirali **attr_accessor** jer se nasleđuje). Nasleđivanje po nivoima i hijerarhijsko nasleđivanje je moguće dok višestruko nasleđivanje nije(više o tipovima nasleđivanja u ??).

7.3 Polimorfizam

Osnovni vid polimorfizma možemo postići nasleđivanjem tako sto ce razliciti objekti odgovoriti razlicito na iste metode. U primeru(??) u klasi

Driver smo definisali metod *print* koji je istog naziva kao i nasleđeni metod čime postizemo da instanca klase odgovori različito na metod od instance roditeljske klase.

Drugačiji vid polimorfizma postizemo takozvanim "pačijim kucanjem" (eng. *duck typing*) u kojem nisu bitni tipovi objekata već skup istoimenih metoda koje poseduju. Za primer uzmimo klasu *Duck* koja poseduje metod *quack* i funkciju koja za argument uzima objekat tipa *Duck* i poziva metod *quack*. U tom slučaju funkciji možemo proslediti bilo koji objekat koji poseduje metod naziva *quack* (sa istim ili različitim ponašanjem metoda) i gledati na njega kao da je tipa *Duck* bez obzira što on to nije.

7.4 Apstrakcija

Ruby nema direktnu podršku za apstrakciju klasa ali se sličan efekat može dobiti korišćenjem biblioteke "abstract". Takođe je moguće implementirati apstrakciju pomoću nasleđivanja gde roditeljska klasa definiše apstraktne metode koji pokrecu "NotImplementedError" grešku tako da se ne mogu instancirati, pa mora postojati dete klasa koja će pomoću gore opisanog polimorfizma(??) nasleđivanjem implementirati željene apstraktne metode. Metode koje su zajedničke implementiramo u roditeljskoj klasi.

8 Osnovna uputstva

Vaš seminarski rad mora da sadrži najmanje jednu **sliku**, najmanje jednu **tabelu** i najmanje **sedam referenci** u spisku literature. Najmanje jedna slika treba da bude originalna i da predstavlja neke podatke koje ste Vi osmislili da treba da prezentujete u svom radu. Isto važi i za najmanje jednu tabelu. Od referenci, neophodno je imati bar jednu **knjigu**, bar jedan **naučni članak** iz odgovarajućeg časopisa i bar jednu adekvatnu **veb adresu**.

Dužina seminarskog rada treba da bude od 10 do 12 strana. Svakom prekoračenje ili potkoračenje biće kažnjeno sa odgovarajućim brojem poena. Eventualno, nakon strane 12, može se javiti samo tekst poglavlja **Dodatak** koji sadrži nekakav dodatni kôd, ali je svakako potrebno da rad može da se pročita i razume i bez čitanja tog dodatka.

Ко жели, може да пише рад ћирилицом. У том случају, неопходно је да су инсталирани одговарајући пакети: texlive-fonts-extra, texlive-latex-extra, texlive-lang-cyrillic, texlive-lang-other.

Nemojte koristiti stari način pisanja slova, tj ovo:

```
\v{s} i \v{c} i \'c ...
```

Koristite direktno naša slova:

```
š i č i ć ...
```

9 Engleski termini i citiranje

Na svakom mestu u tekstu naglasiti odakle tačno potiču informacije. Uz sve novouvedene termine u zagradi naglasiti od koje engleske reči termin potiče.

Naredni primeri ilustruju način uvođenja engleskih termina kao i citiranje. [?]

Primer 9.1 *Problem zaustavljanja (eng. halting problem) je neodlučiv [?].*

Primer 9.2 *Za prevođenje programa napisanih u programskom jeziku C može se koristiti GCC kompajler [?].*

Primer 9.3 *Da bi se ispitivala ispravnost softvera, najpre je potrebno precizno definisati njegovo ponašanje [?].*

Reference koje se koriste u ovom tekstu zadate su u datoteci *seminarski.bib*. Prevođenje u pdf format u Linux okruženju može se uraditi na sledeći način:

```
pdflatex TemaImePrezime.tex
bibtex TemaImePrezime.aux
pdflatex TemaImePrezime.tex
pdflatex TemaImePrezime.tex
```

Prvo latexovanje je neophodno da bi se generisao *.aux* fajl. *bibtex* proizvodi odgovarajući *.bbl* fajl koji se koristi za generisanje literature. Potrebna su dva prolaza (dva puta *pdflatex*) da bi se reference ubacile u tekst (tj da ne bi ostali znakovi pitanja umesto referenci). Dodavanjem novih referenci potrebno je ponoviti ceo postupak.

Broj naslova i podnaslova je proizvoljan. Neophodni su samo Uvod i Zaključak. Na poglavlja unutar teksta referisati se po potrebi.

Primer 9.4 *U odeljku ?? precizirani su osnovni pojmovi, dok su zaključci dati u odeljku ??.*

Još jednom da napomenem da nema razloga da pišete:

```
\v{s} i \v{c} i \c{ ...
```

Možete koristiti srpska slova

```
š i č i ć ...
```

10 Slike i tabele

Slike i tabele treba da budu u svom okruženju, sa odgovarajućim naslovima, obeležene labelom da koje omogućava referenciranje.

Primer 10.1 *Ovako se ubacuje slika. Obratiti pažnju da je dodato i*

```
\usepackage{graphicx}
```

Na svaku sliku neophodno je referisati se negde u tekstu. Na primer, na slici ?? prikazane su pande.

Primer 10.2 *I tabele treba da budu u svom okruženju, i na njih je neophodno referisati se u tekstu. Na primer, u tabeli ?? su prikazana različita poravnanja u tabelama.*

Tabela 1: Različita poravnanja u okviru iste tabele ne treba koristiti jer su nepregledna.

centralno poravnanje	levo poravnanje	desno poravnanje
a	b	c
d	e	f

11 Kôd i paket listings

Za ubacivanje koda koristite paket **listings**: https://en.wikibooks.org/wiki/LaTeX/Source_Code_Listings

Primer 11.1 *Primer ubacivanja koda za programski jezik Python dat je kroz listing ???. Za neki drugi programski jezik, treba podesiti odgovarajući programski jezik u okviru defnisanja stila.*

```
1000 # This program adds up integers in the command line
      import sys
1002 try:
      total = sum(int(arg) for arg in sys.argv[1:])
1004     print 'sum =', total
      except ValueError:
1006     print 'Please supply integer arguments'
```

Listing 8: Primer ubacivanja koda u tekst

12 Prvi naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

12.1 Prvi podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

12.2 Drugi podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

12.3 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

13 n-ti naslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

13.1 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

13.2 ... podnaslov

Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst. Ovde pišem tekst.

14 Zaključak

Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.
Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.
Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.
Ovde pišem zaključak. Ovde pišem zaključak. Ovde pišem zaključak.

A Dodatak

Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe. Ovde pišem dodatne stvari, ukoliko za time ima potrebe.