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, nikolastojevic@gmail.com

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 imenovnja 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 deklarisane 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 deklarisana 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 naglašeno 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 klasa se podrazumeva private, dok kod struktura se podrazumeva public (što je jedna od osnovnih razlika između struktura i klasa 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 ce 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 jos 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 "+đok 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 destruktor. 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 destruktor, ali ne mogu da imaju virtuelni konstruktor.

3 Objective C

Proces definisanja klasa se malo razlikuje kod jezika Objective C. Obavlja se u dve sekcije koje se označavaju sa ključnom reči **@interface** i **@implementation**, gde se vrši deklaracija i implementacija metoda klase. Obe sekcije se završavaju ključnom reči **@end**. Svaka klasa je izvedena iz super klase NSObject, čiji konstruktor init je podrazumevani konstruktor i on se predefiniše da odgovara konkretnoj klasi. Moguće je kreirati i svoje konstruktore sa proizvoljnim argumentima,slično ostalim programskim jezicima i mogu se proizvoljno imenovati.. Detaljnije

o ovome i ključnim rečima u dokumentaciji [1]. U primeru (3) vidimo definisanje klase zaposleni (detaljnije 3.1).

```
@interface Employee : NSObject {
    @public int age;
    double salary;
}
@property(nonatomic, readwrite) double salary;  // Property
- (void)display;
@end

@implementation Employee
@synthesize salary;
- (void)display {
    NSLog(@"Employee salary is %f", salary);
}
@end

@interface Driver : Employee {
    NSString* truck;
}
- (id)initWithTruck:(NSString*)model;
@end

@implementation Driver
- (id)initWithTruck:(NSString*)model {
    truck = model;
    return self;
}
- (void)display {
    NSLog(@"Driver salary is %f", salary);
}
@end

int main(int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    Employee *empl = [[Driver alloc]initWithTruck:@"Mercedes"];
    empl.salary = 5.0;  empl->age = 33;
    [empl display];
    [pool drain];
    return 0;
}}
```

Listing 3: Primer koda u Objective C jeziku

3.1 Enkapsulacija

Atributi klase su automatski privatni, što pogoduje enkapsulaciji. Privatnim svojstvima se omogućuje pristup izvan klase preko ključne reči **@property**, uz implementaciju pristupnih metoda. Olakšanje, navođenjem atributa uz ključnu reč **@synthesize**, automatski se generišu metode pristupa. Primer (3) u main funkciji, prikazuje kako pozivamo pristupne metode pomoću operatora (.), dok se javnim atributima pristupa preko operatora (->).

3.2 Nasleđivanje

Ovaj koncept označavamo sa (:) i imenom klase koju nasleđujemo. Primer (3), klasa vozač nasleđuje klasu zaposleni .Postoji nasleđivanje u više nivoa i hijerarhijsko. Ako želimo iz izvedene klase da zovemo metode bazne, to radimo referisanjem na baznu klasu pomoću ključne reči **super**.

3.3 Polimorfizam

Višestruka upotrebljivost koda u vidu preopterećenosti (eng. ***overloading***) nije omogućeno u Objective C jeziku, tako da se metodi moraju različito imenovati[2]. Koncept važnosti metoda(eng. ***overriding***) postoji, i prikazan je u primeru (3). Gde klase zaposleni i vozač imaju isti metod display i poziv [empl display] će izvršiti metod klase vozač, zato što promenljiva zaposleni empl referiše na objekat tipa vozač.

3.4 Apstrakcija

Jezik Objective C nema definisan koncept apstraktnih klasa[2], sličan efekat je moguće postići programerskom snalažljivošću i ne instancirati klasu koja bi trebalo biti apstraktna. Za interfejse koje ovde nazivamo protokoli, vezana je ključna reč **@protocol** jer je **interface** rezervisana za klase. Sekcija protokola se završava sa **@end** i može sadžati dve podoblasti **@required** za metode koji se obavezno moraju implementirati u klasi i **@optional** za metode čija je implementacija opciona.

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 (4) predstavlja deklaraciju klase zaposleni, koja ima svoje atribute i metode(detaljnije 4.1).

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

Listing 4: Primer deklarisanja klase sa enkapsulacijom i nasleđivanjem

4.1 Enkapsulacija

Ograničavanje pristupa internim podacima klase postižemo navođenjem ključne reči ***private*** ispred deklaracije promenljive u klasi. Ovo znači da se podacima može pristupiti isključivo iz deklarisane klase. Podacima

neophodnim za funkcionalnost programa obezbeđuje se pristup čitanja i menjanja (eng. *getters and setters*)[3] preko javnih metoda. U primeru koda (4), 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 [4]:

- 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 klasa) nije moguće, već se implementira preko interfejsa(detaljnije 4.4).

U primeru koda (4), klasa vozač nasleđuje klasu zaposleni.

4.3 Polimorfizam

Višestruka upotrebljivost koda za različite vrste objekata.

Pripadnost metoda objektu se obavlja u vreme izvršavanja(eng. *run time execution*) i predstavlja koncept važnosti metoda(eng. ***overriding***)[3]. U primeru koda 4, Marko.display(); pozvaće metod klase vozač.

Koncept prenatrpanosti metoda(eng. ***overloading***)[3], 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 [4], Marko.display(2); pozvaće metod display(int x) klase vozač.

4.4 Apstrakcija

Izdvajanje skupa metoda sa kojima spoljašnji korisnik komunicira, prema artiklu [4], vršimo pomoću apstraktnih klasa ili interfejsa.

Za apstraktne klase navodimo ključnu reč ***abstract***(kod 5). 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 5: 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 6) i zatim ime interfejsa(slično nasleđivanju). Prednost interfejsa[3] 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 6: 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 postojecih klasa.
- Enkapsulacija je kada se grupa od povezanih metoda, svojstava I ostalih clanova tretira kao jedan isti objekat.
- Apstrakcija je proces kod koga programer krije sve osim relevantnih podataka o datom objektu u cilju pojednostavljanja I povecanja efikasnosti.

```
1000 abstract class MobilePhone {
         public void Calling();
1002     public void SendSMS();
     }
1004 public class Nokia1400: MobilePhone {}
     public class Nokia2700: MobilePhone {
1006     public void FMRadio();
         public void MP3();
1008     public void Camera();
     }
1010 public class BlackBerry: MobilePhone {
         public void FMRadio();
1012     public void MP3();
         public void Camera();
1014     public void Recording();
         public void ReadAndSendEmails();
1016 }
```

Listing 7: Primer deklarisanja apstraktivne klase u C#-u

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

5.1 Nasledjivanje

Koristi sintaksu c++ za definisanje klasa.
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
- Odgovarajuce 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 podrske C#-a I Javine podrske za OOP su relativno male.

6 Ada

Deo za Ada.

7 Ruby

Podršku u programskom jeziku Ruby ilustrovaćemo primerom(8) 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.

```
class Employee
attr_accessor :name
def initialize(name)
  @name = name
  print()
end
def print
  puts "Employee: #{@name}."
end
end
class Driver < Employee
  def initialize(name)
    @name = name
    print()
  end
  private
  def print
    puts "Driver: #{@name}."
  end
end

emp = Employee.new("John")
drv = Driver.new("John")
```

Listing 8: 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 mogucnost ugrađenih metoda za pristup(eng. *accessor methods*). U primeru(8) **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(8) gde klasa *Driver* nasleđuje klasu *Employee*(primetiti da u klasi *Driver* nismo implementirali **attr_accessor** jer se nasledjuje). Nasleđivanje po nivoima i hijerarhijsko nasleđivanje je moguće dok višestruko nasleđivanje nije(više o tipovima nasleđivanja u 4.2).

7.3 Polimorfizam

Osnovni vid polimorfizma možemo postići nasleđivanjem tako sto ce razliciti objekti odgovoriti različito na iste metode. U primeru(8) u klasi *Driver* smo definisali metod *print* koji je istog naziva kao i nasleđeni metod čime postižemo da instanca klase odgovori različito na metod od instance roditeljske klase.

Drugačiji vid polimorfizma postižemo 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 apstakciju pomoću nasleđivanja gde rodjiteljska klasa definiše apstraktne metode koji pokrecu "NotImplementedError"grešku tako da se ne mogu instancirati, pa mora postojati dete klasa koja će pomocu gore opisanog polimorfizma(7.3) nasleđivanjem implementirati željene apstraktne metode. Metode koje su zajedničke implementiramo u roditeljskoj klasi.

8 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.

Literatura

[1] Object C apple documentation. on-line at: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/ObjectiveC.

[2] Gary Bennett, Brad Lees, and Mitchell Fisher. *Objective-C for Absolute Beginners: IPhone, iPad and Mac Programming Made Easy*. Apress, Berkely, CA, USA, 3rd edition, 2016.

[3] Cay S Horstmann. *Core Java SE 9 for the Impatient*. Addison-Wesley Professional, 2017.

[4] Aayushi Johari. Object Oriented Programming – Java OOPs Concepts With Examples, 2018. on-line at: https://www.edureka.co/blog/object-oriented-programming/.

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.