

## Tema 13

# Tehnike efikasnog programiranja u jeziku C++ i uvod u OO modelovanje

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2



# Sadržaj

1. Uvod
2. Efikasnost poznatih algoritama
3. Metodologije razvoja softvera
4. Identifikacija klasa
5. Identifikacija veza između klasa
6. Primer projektovanja



# 1. Uvod

- Efikasnost programa
- Razvoj efikasnih algoritama
- Proces razvoja softvera



# Efikasnost programa

- Efikasni programi razvijaju se na osnovu **efikasnih algoritama**
- Poređenje algoritama, npr. linearnog i binarnog pretraživanja, može se izvršiti na osnovu testiranja, ali je zavisno od
  - hardverskih i softverskih osobina računara i trenutnih uslova konkurentnog izvršavanja
  - obima i osobina podataka, npr. da li su već sortirani
- Zbog toga se ocena efikasnosti algoritama vrši na osnovu njihovog asimptotskog ponašanja, funkcije rasta  $O(n)$
- Postoje **opšti metodi** dizajniranja efikasnih algoritama, kao što su npr. dinamičko programiranje, podela problema na potprobleme (*divide-and-conquer*) i sistematsko pretraživanje (*backtracking*)

# Razvoj efikasnih algoritama

- **Dizajn algoritama** predstavlja metod izgradnje matematičkog pristupa rešavanju problema, odnosno razvoju programa
- **Analiza algoritama** se bavi predviđanjem performansi algoritama
  - u praksi su identifikovani brojni obrasci (*design patterns*), šabloni metoda i načina upotrebe struktura podataka  
npr. šablon **dekorater** (*decorator pattern*) omogućava dodavanje ponašanja nekom objektu bez uticaja na ponašanje drugih objekata iste klase
- Jedan od najvažnijih aspekata dizajna algoritama je razvoj algoritama male vremenske složenosti  $O(n)$

# Proces razvoja softvera

- Softverski sistemi se kreiraju da postoje određeno vreme
- **Životni ciklus** softverskog sistema (*system life cycle*) može se podeliti u dve osnovne faze:
  - fazu *razvoja* (završava *isporukom*)
  - fazu *rada* i održavanja (završava *zastarevanjem*)
- Zastarevanje pokreće razvoj/nabavku nove verzije sistema i povlačenje iz upotrebe stare verzije (*pensioning*)
- **Proces razvoja** softvera je pristup izgradnji, isporuci i održavanju softvera
  - parcijalno uređeni niz koraka usmerenih ka cilju
  - cilj je efikasna i predvidiva isporuka softverskog sistema, koji zadovoljava postavljene zahteve

## 2. Efikasnost poznatih algoritama

1. Primeri elementarnih algoritama
2. Vremenska složenost binarnog pretraživanja
3. Vremenska složenost sortiranja selekcijom
4. Pronalaženje Fibonačijevih brojeva (dinamičko programiranje)
5. Poređenje opštih funkcija rasta



## 2.1 Primeri elementarnih algoritama

- Vremenska složenost ugnježđenih petlji

```
for (i=1; i<=n; i++) {  
    for (j=1; j<=i; j++) {  
        k = k + i + j;  
    }  
}
```

– spoljašnja  $n$  puta, unutrašnja  $\sum_{1..n} n$ ,  $T(n) = c \cdot n \cdot (n+1)/2 = O(n^2)$

- Vremenska složenost za izmenjenu unutrašnju petlju

```
for (i=1; i<=n; i++) {  
    for (j=1; j<=20; j++) {  
        k = k + i + j;  
    }  
}
```

– spoljašnja  $n$  puta, unutrašnja  $20 \cdot n$ ,  $T(n) = c \cdot n \cdot 20 = O(n)$



## 2.2 Vremenska složenost binarnog pretraživanja

- Binarno pretraživanje pronalazi zadani element *key* u sortiranoj listi od *n* elemenata
  - jedna iteracija petlje izvrši se za konstantno vreme *c*
  - algoritam na svakom koraku petlje eliminiše 1/2 elemenata, nakon dva poređenja
- Vremenska složenost algoritma je
$$T(n) = T(n/2) + c = T(n/2^2) + c + c$$
$$= \dots = T(n/2^k) + k \cdot c = T(1) + c \cdot \log n$$
$$= 1 + (\log n) + c = O(\log n)$$
- Algoritam ima *logaritamsku* vremensku složenost

```
int binarySearch (const int list[],
                  int key, int listSize) {
    int low = 0;
    int high = listSize - 1;

    while (high >= low) {
        int mid = (low + high)/2;
        if (key < list[mid])
            high = mid - 1;
        else if (key == list[mid])
            return mid; // pronađen
        else
            low = mid + 1;
    }
    return -low - 1; // nije pronađen
}
```

## 2.3 Vremenska složenost sortiranja selekcijom

- Sortiranje selekcijom, počev od prvog elementa, pronalazi minimalni element u preostalom delu liste i po potrebi ga zameni s prvim
- Ponavlja postupak od narednog elementa liste
  - broj poređenja je  $n-1$  u prvoj iteraciji,  $n-2$  u drugoj itd.
- Ukupan broj operacija je
$$T(n) = (n-1) + c + (n-2) + c + \dots + 2 + c + 1 + c = (n-1)(n-1+1)/2 + c \cdot (n-1) = n^2/2 - n/2 + c \cdot n - c = O(n^2)$$
- Algoritam ima kvadratnu vremensku složenost

```
void selectionSort(double list[], int lSize) {  
    for (int i = 0; i < lSize - 1; i++) {  
        // Pronaći min. vredn. u list[i..lSize-1]  
        double currMin = list[i];  
        int currMinInd = i;  
  
        for (int j = i + 1; j < listSize; j++) {  
            if (currMin > list[j]) {  
                currMin = list[j];  
                currMinInd = j;  
            }  
        }  
  
        // Ako je potrebno, zamena list[i] s  
        list[currMinInd]  
        if (currMinInd != i) {  
            list[currMinInd] = list[i];  
            list[i] = currMin;  
        }  
    }  
}
```

## 2.4 Pronalaženje Fibonačijevih brojeva (dinamičko programiranje)

- Rekurzivna verzija  
 $F(1) = 1, F(2) = 1$   
 $F(n) = F(n-1) + F(n-2)$   
ima složenost  $O(2^n)$
- Veliki nedostatak algoritma su redundantna računanja, jer za svaki poziv gde je  $n > 1$  generiše dva nova poziva, npr.
  - za `fib(4)`, poziva `fib(3)` i `fib(2)`
  - za `fib(3)`, poziva (ponovo!) `fib(2)` i `fib(1)`

```
// Računa Fibonačijev broj za  
// zadani n  
long fib(long n) {  
    if (n == 0) // početak  
        return 0;  
    else if (n == 1) // početak  
        return 1;  
    else // rekurzivni poziv za n>1  
        return fib(n-1) + fib(n-2);  
}
```

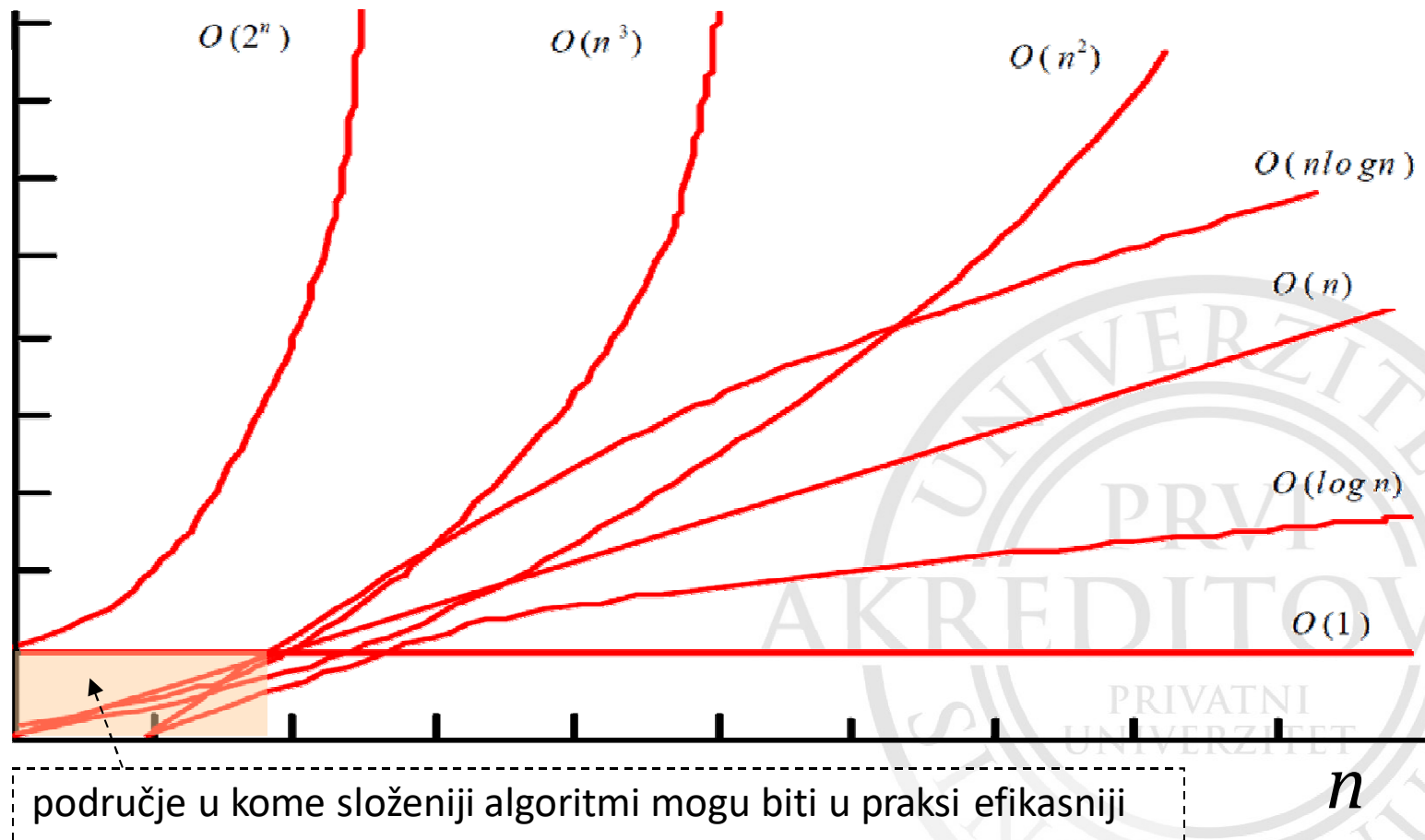
# Pronalaženje Fibonačijevih brojeva (dinamičko programiranje)

- Bolje rešenje je algoritam *dinamičkog programiranja* složenosti  $O(n)$ , koji *ne ponavlja* računanja na svakom koraku

```
// Računa Fibonačijev broj za zadani n
long fib(long n) {
    long f0 = 0; // fib(0)
    long f1 = 1; // fib(1)
    long f2 = 1; // fib(2)
    if (n == 0)
        return f0; // početak
    else if (n == 1)
        return f1; // početak
    else if (n == 2)
        return f2;
    // n>1
    for (int i = 3; i <= n; i++) {
        f0 = f1;
        f1 = f2;
        f2 = f0 + f1;
    }
    return f2;
}
```

## 2.5 Poređenje opštih funkcija rasta

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < \dots$$



## 3. Metodologije razvoja softvera

1. Objektno orijentisani razvoj softvera
2. Metodologija razvoja
3. Sekvencijalna metodologija
4. Metodologija RUP
5. Ekstremno programiranje



## 3.1 Objektno orijentisani razvoj softvera

- **Objektno orijentisani** pristup razvoju softvera, koji se koristi za složene softverske sisteme, obuhvata objektno orijentisanu *analizu i projektovanje*
  - **objektno orijentisana analiza** je proces usmeren na ispitivanje problema i zahteva, a ne na njihovo rešavanje  
npr. identifikuju se *objekti* od interesa, kao *Avion*, *Let* i *Pilot*
  - **objektno orijentisano projektovanje** je proces kreiranja konceptualnog rešenja koje zadovoljava postavljene zahteve, ali ne i njegovu implementaciju; u toku projektovanja definišu se *softverski objekti* i način njihove saradnje radi zadovoljenja zahteva  
npr. klasa *Avion* ima atribut *registarskiBroj* i metod *getIstorijaLetenja()*

# Faze razvoja softvera

- Razvoj softvera najčešće se posmatra kroz pet osnovnih *faza razvoja*:
  1. Analiza (*analysis*)
  2. Projektovanje (*design*)
  3. Implementacija (*implementation*)
  4. Testiranje (*testing*)
  5. Isporuka (*deployment*)
- Razvoj softvera ne teče sekvencijalno; obično se pojedine faze razvoja delimično preklapaju ili ponavljaju



# Projektovanje softvera

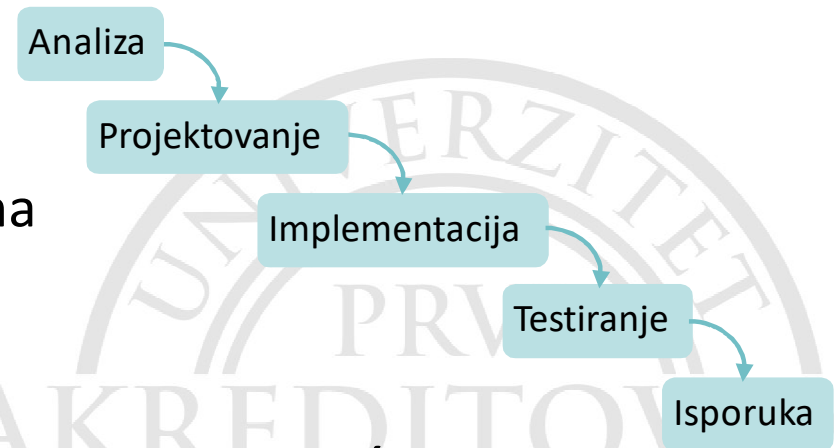
- Prilikom razvoja softvera kreira se *model sistema*, koji prikazuje softverski sistem s različitih aspekata, npr. posebnim vrstama UML dijagrama za
  - prikaz *strukture* softvera pomoću objekata, atributa, operacija i relacija (npr. dijagrami slučajeva korišćenja, klasa i paketa)
  - prikaz *ponašanja* softvera kroz prikaz saradnje objekata i promena njihovih unutrašnjih stanja (npr. dijagrami sekvenci, aktivnosti, stanja)
- Standard UML definiše 14 dijagrama u tri kategorije
- Softverski alati za projektovanje softvera omogućavaju kreiranje UML modela sistema i generisanje koda na osnovu UML dijagrama (samo strukture koda ili celog sistema)
  - npr. Microsoft *Visual Studio* i IBM *Rhapsody*

## 3.2 Metodologija razvoja

- **Metodologija razvoja** (*system development methodology*) je je skup aktivnosti, metoda, iskustava, preporuka i automatizovanih alata koji se koriste za razvoj i neprekidno usavršavanje softvera
- Postoje različite **klasične** (sekvencijalna), **novije** (iterativna, inkrementalna, kombinovane) i **agilne** metodologije
- Agilne metodologije
  - manje stroge metodologije, koje manje projekte i timove ne opterećuju birokratijom, najpoznatije su XP (*Extreme Programming*) i Scrum
  - podrazumevaju adaptivno planiranje, evolutivni razvoj i inkrementalnu isporuku softvera

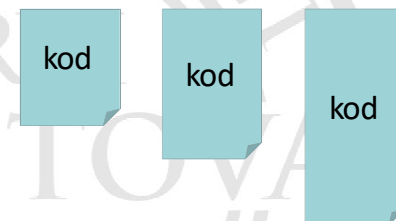
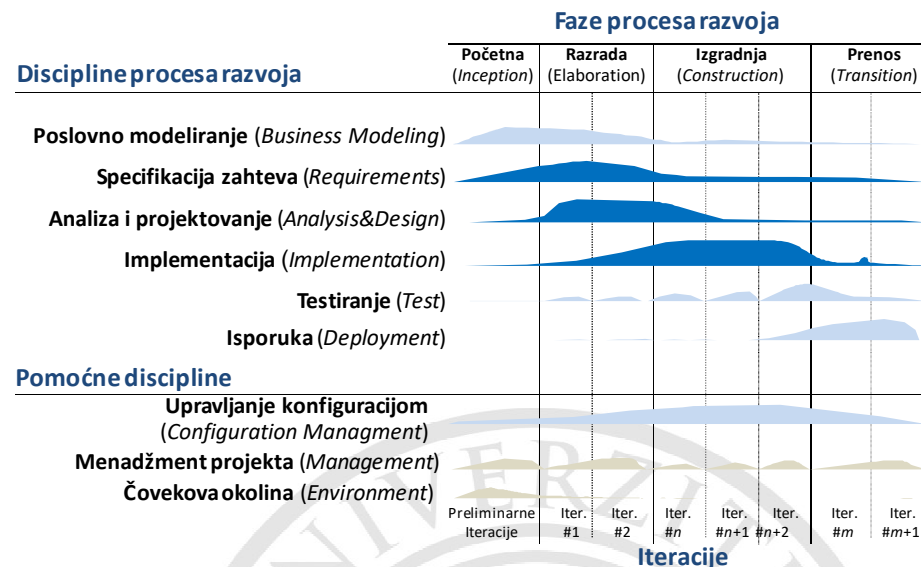
## 3.3 Sekvencijalna metodologija

- **Sekvencijalni** razvoj ili model vodopada (*waterfall*) je pristup razvoju gde se svaka faza razvoja završi u potpunosti, nakon čega se prelazi na sledeću fazu
- Metodologija nije adekvatna za sisteme velike kompleksnosti
  - sve veći obim aplikacija
  - veliki ili distribuirani timovi
  - povećana je tehnička složenost sistema
  - stalne novine u tehnologijama
  - produžava trajanje projekta
- Glavni problem ovog pristupa je što ne omogućava identifikovanje i umanjeње *rizika* u ranim fazama projekta



## 3.4 Metodologija RUP

- Konkretna objektno orijentisana metodologija *Rational Unified Process* koju je uvela kompanija *Rational* (kasnije IBM)
  - iterativno-inkrementalna
  - sve faze razvoja se realizuju kroz niz iteracija i inkremenata
  - cilj je što kvalitetniji rezultat u posmatranom vremenu
  - na kraju svake iteracije je *kôd*
  - metodologija ima sopstvene softverske alate



## 3.5 Ekstremno programiranje

- **Ekstremno programiranje** (XP) je *agilna* metodologija, koja podrazumeva i prihvata stalne promene sistema kao činjenicu
- Realizuje se kroz male ili srednje razvojne timove, koji rade u tesnoj *saradnji s korisnikom*
- Ne koristi se precizno planiranje, već se teži brzim, opipljivim rezultatima, koji se odmah predočavaju korisniku
- Elementi metodologije su specifične preporuke (*practices*)
  - realno planiranje, razvoj u malim koracima
  - programiranje u parovima (na smenu)
  - korisnik je dostupan članovima tima *celo vreme na lokaciji*
  - testiranje je *kontinualno* (vrše i programeri i korisnici)
  - kodiranje se vrši prema standardima (samodokumentujući kod)

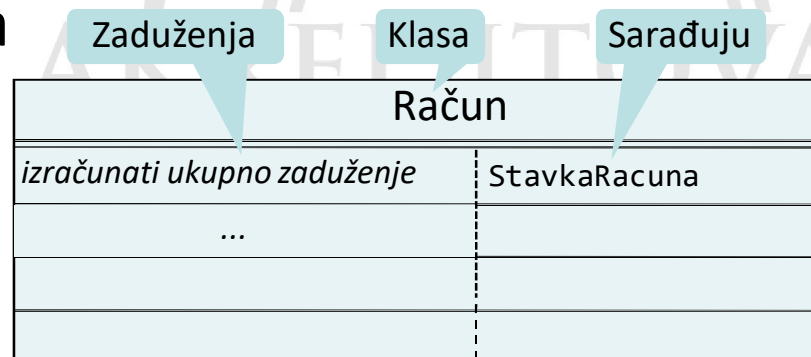
## 4. Identifikacija klasa

- Proces otkrivanja klasa
- Svojstva dobrog OO modela



# Proces otkrivanja klasa

- Osnovni problem objektno orijentisanog projektovanja je otkrivanje *entiteta* i ustanovljavanje njihovih *svojstava* i *zadataka*
  - koji će se predstaviti u obliku *klasa* i njihovih *atributa* i *metoda*
- Popularni naziv jedne neformalne tehnike evidentiranja klasa su CRC kartice (*Classes-Responsibilities-Collaborators*)
- Na osnovu *analize problema*, iz dokumentacije, upitnika, intervjuja, posmatranjem i sl., uočavaju se **klase**, njihovi **atributi** i **veze** s drugim klasama
  - tehnika: imenice, glagoli, pridevi iz opisa problema

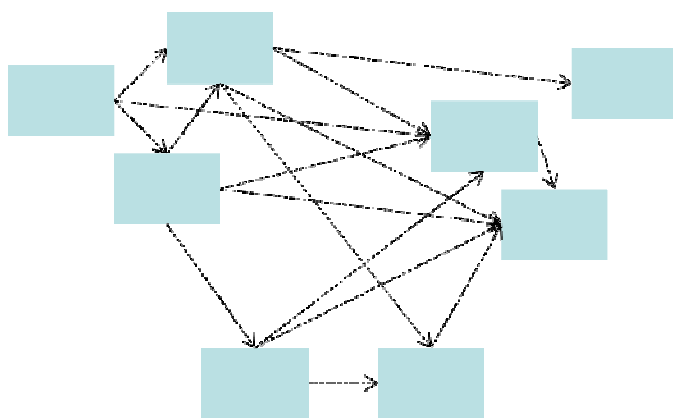


| Zaduženja                   | Klasa | Sarađuju     |
|-----------------------------|-------|--------------|
| izračunati ukupno zaduženje | Račun | StavkaRacuna |
| ...                         |       |              |
|                             |       |              |
|                             |       |              |

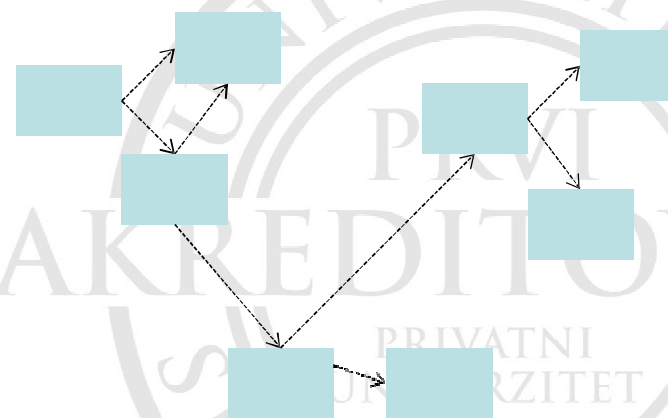


# Svojstva dobrog objektno orijentisanog modela

- Unutrašnja kohezija klasa (*cohesion*)
  - klasa treba da predstavlja jedan pojam iz domena problema, npr. iako je oblik točka automobila krug, točak nije geometrijski objekt
  - *sva svojstva* klase treba da se odnose na pojam koji klasa predstavlja
- Slaba međusobna povezanost klasa (*coupling*)
  - minimizacija veza između klasa (npr. zavisnosti)



Jaka povezanost klasa  
(*high coupling*)



Slaba povezanost klasa  
(*low coupling*)



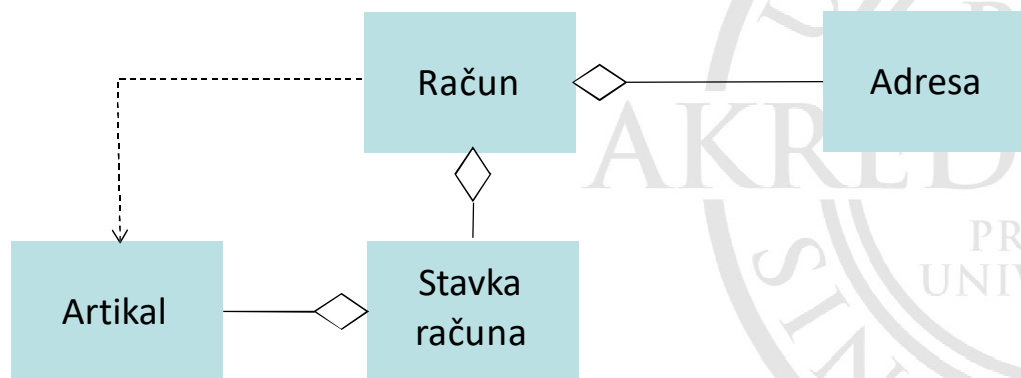
## 5. Identifikacija veza između objekata

- Vrste veza
- Prikaz veza u UML dijagramu klasa
- Implementacija agregacije



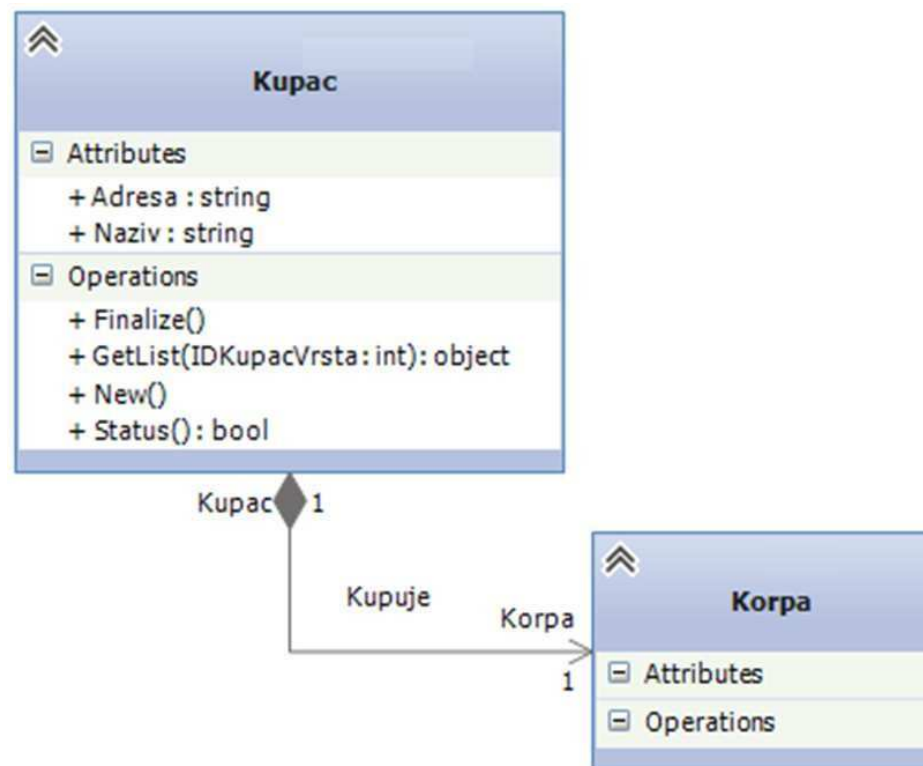
# Vrste veza

- Veza zavisnosti (*uses*, "koristi")
  - klasa je povezana s drugom klasom, ako neka njena funkcija član na neki način *koristi* objekt druge klase
- Veza nasleđivanja (*is-a*, "jeste" )
  - klasa je *specijalni slučaj* druge klase, nasleđuje njena svojstva
- Veza agregacije ili kompozicije (*has-a*, "ima")
  - klasa *sadrži* druge klase, koje su njeni delovi



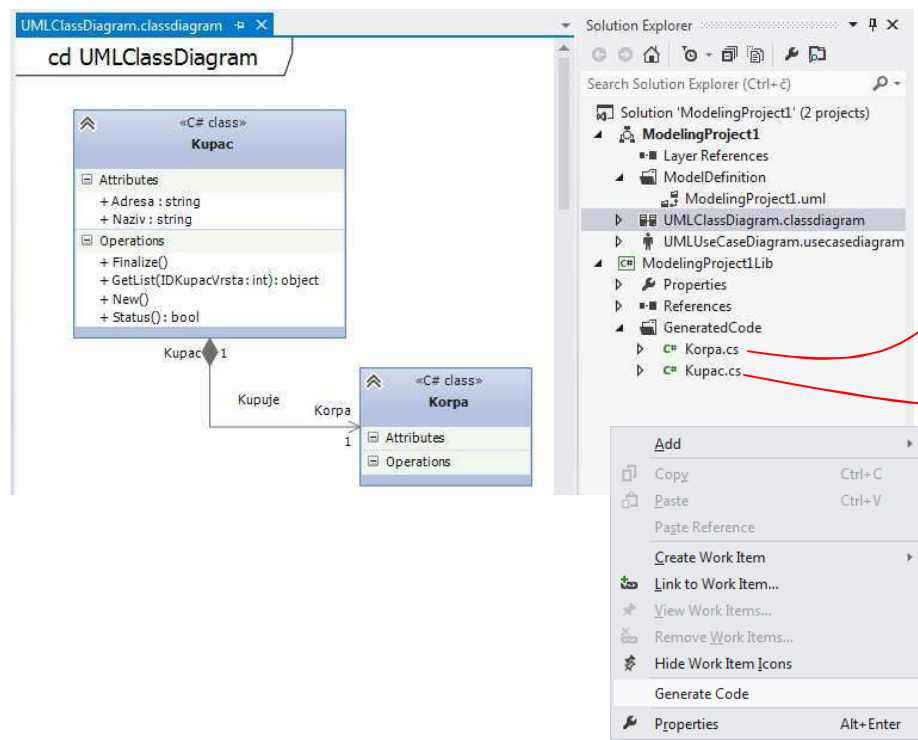
# Prikaz veza u UML dijagramu klasa

- Dijagram klasa je osnovni projektni dijagram za opis *struktura* sistema
- Modeli objekata i veze između objekata
  - klase (iz *Use Case* i dr.)
    - atributi (vidljivost, tip)
    - metodi (vidljivost, parametri)
  - veze (multiplikativnost)
- Dijagrami se razvijaju postepeno. U svakoj fazi je moguće *generisati kod*



# Primer: Generisanje kôda na osnovu dijagrama klasa (Visual Studio)

- Primer: UML definicija klasa Kupac i Korpa
- Generisana struktura koda:



```

1 1 //-----
2 2 // <auto-generated>
3 3 // This code was generated by a tool
4 4 // Changes to this file will be lost if the code is regenerated.
5 5 // </auto-generated>
6 6 //-----
7 7 using System;
8 8 using System.Collections.Generic;
9 9 using System.Linq;
10 10 using System.Text;
11 11
12 12 public class Kupac
13 13 {
14 14     public virtual string Naziv
15 15     {
16 16         get;
17 17         set;
18 18     }
19 19
20 20     public virtual string Adresa
21 21     {
22 22         get;
23 23         set;
24 24     }
25 25
26 26     public virtual Korpa Korpa
27 27     {
28 28         get;
29 29         set;
30 30     }
31 31
32 32     public virtual object GetList(int IDKupacVrsta)
33 33     {
34 34         throw new System.NotImplementedException();
35 35     }
36 36
37 37     public virtual void New()
38 38     {
39 39         throw new System.NotImplementedException();
40 40     }
41 41
42 42     public virtual bool Status()
43 43     {
44 44         throw new System.NotImplementedException();
45 45     }
46 46
47 47     public virtual void Finalize()
48 48     {
49 49         throw new System.NotImplementedException();
50 50     }
51 51
52 52
53 53 }
  
```

```

1 1 //-----
2 2 // <auto-generated>
3 3 // This code was generated by a tool
4 4 // Changes to this file will be lost if the code is regenerated.
5 5 // </auto-generated>
6 6 //-----
7 7 using System;
8 8 using System.Collections.Generic;
9 9 using System.Linq;
10 10 using System.Text;
11 11
12 12 public class Korpa
13 13 {
14 14 }
15 15
16 16
  
```

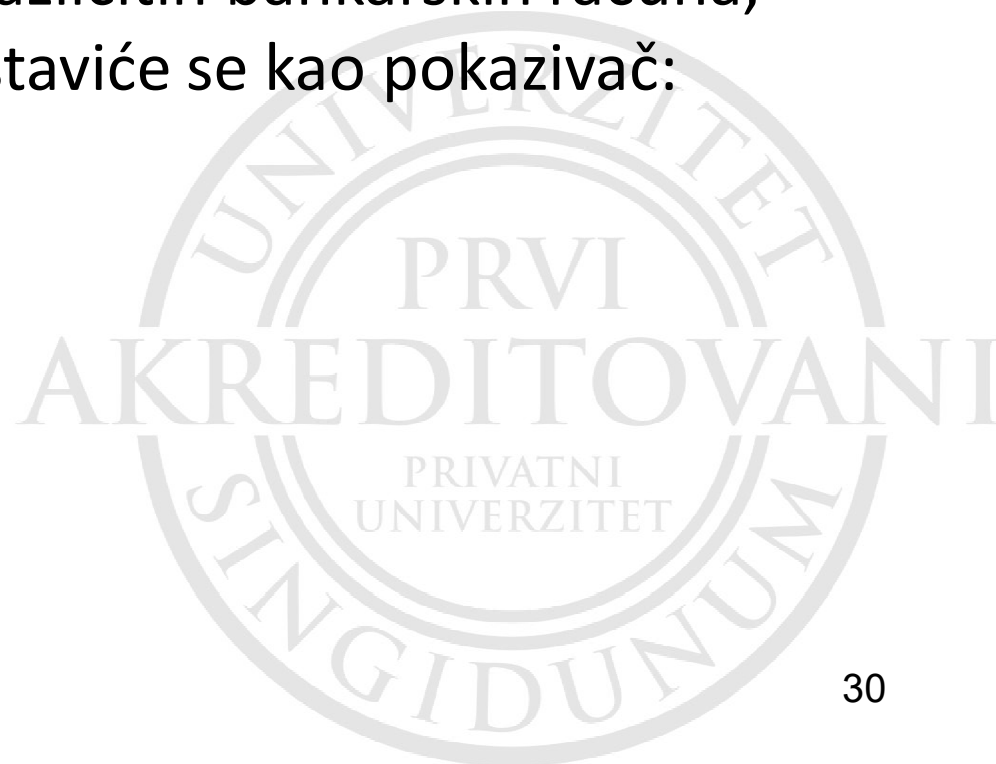
# Implementacija agregacije

- Agregacija se implementira kao *član podatak*, koji može biti promenljiva, vektor ili pokazivač
- Način implementacije zavisi od multiplikativnosti agregacije:
  - 1 : 1 - npr. svaki tekući račun ima jednog vlasnika
  - 1 : 0..1 - npr. svaki departman hotela ima 0 ili 1 recepcionera
  - 1 : \* - npr. svaka kompanija ima više zaposlenih
- Objekt tipa *vektor* može se koristiti za prikaz veze 1:više
- Umesto objekata koriste se *pokazivači*:
  - za veze 1 : 0..1
  - za polimorfne klase, gde povezuju objekt s drugim objektom, koji pripada osnovnoj ili izvedenoj klasi
  - za deljenje objekata (*object sharing*)

# Ilustracija: Bankarski račun

- Klasa `BankarskiRacun` sadrži objekt klase `Osoba`, koja je vlasnik računa i može se u programu predstaviti kao
  - objekt klase `Osoba` ili
  - pokazivač `Osoba*`
- Pošto osoba može da ima *više* različitih bankarskih računa, koji *dele* podatke o osobi, predstaviće se kao pokazivač:

```
class BankarskiRacun {  
    ...  
    private:  
        Osoba* vlasnik;  
};
```



# Ilustracija: Automobil

- Klasa **Automobil** povezana je s klasom **Tocak**
- Pošto automobil ima *više* točkova, mogu se predstaviti
  - vektorom objekata klase **Tocak** ili
  - pokazivačem na objekt klase **Tocak\***
- Pošto točak može biti deo samo *jednog* automobila, predstaviće se kao objekt

```
class Automobil {  
    ...  
    private:  
        vector<Tocak> tockovi;  
};
```



## 6. Primer projektovanja

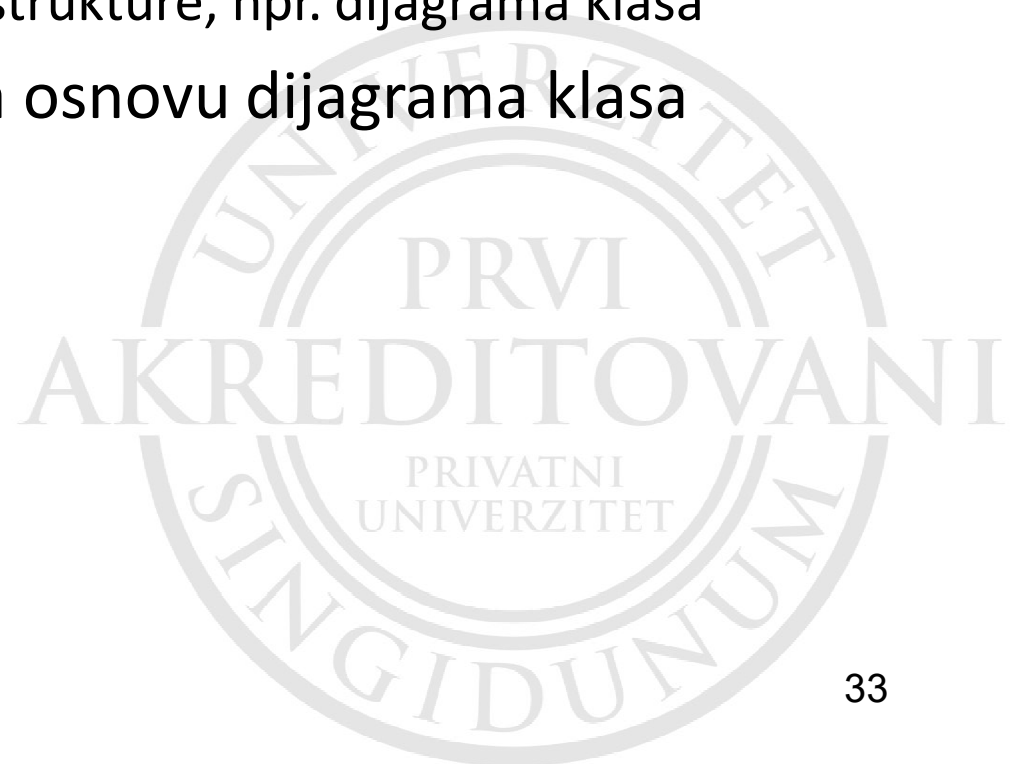
- Koraci projektovanja dela Veb aplikacije





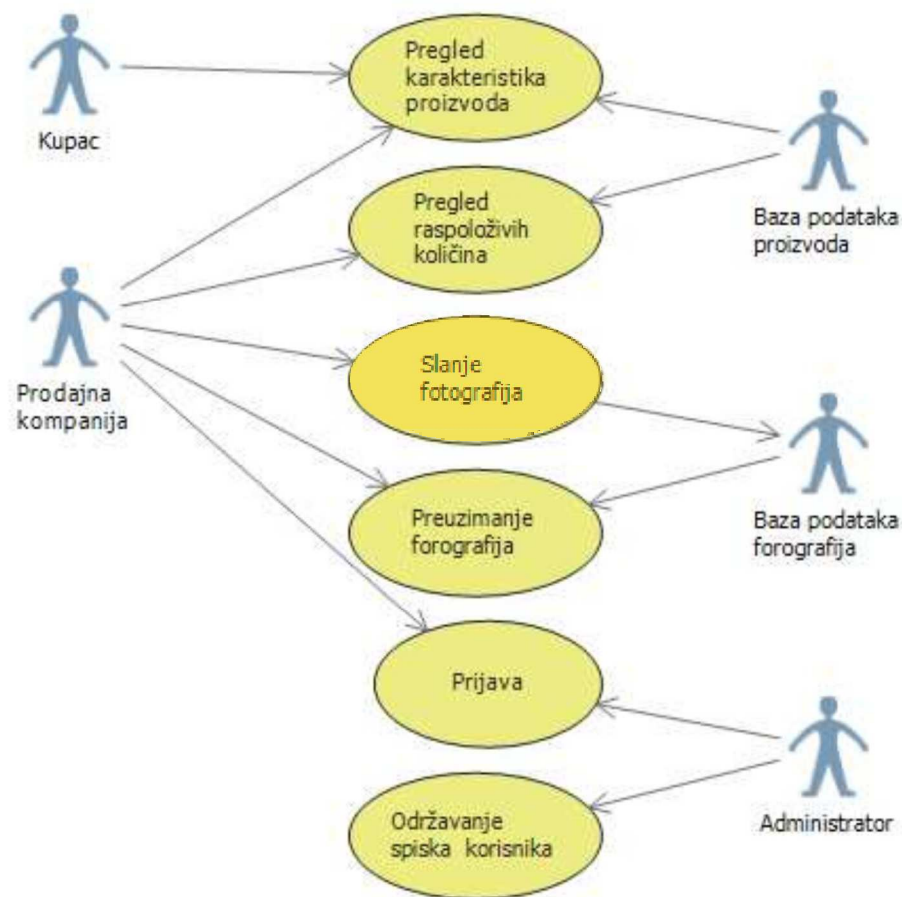
# Koraci projektovanja dela Veb aplikacije

1. Specifikacija korisničkih zahteva (*Software Requirements*)
  - deo specifikacije su UML dijagrami slučajeva korišćenja
2. Razrada slučajeva korišćenja (*Use Cases*)
  - ponašanje pomoću UML dijagrama ponašanja, npr. dijagrama sekvenci
  - struktura pomoću UML dijagrama strukture, npr. dijagrama klasa
3. Generisanje koda, uglavnom na osnovu dijagrama klasa



# Specifikacija korisničkih zahteva aplikacije (*Software Requirements*)

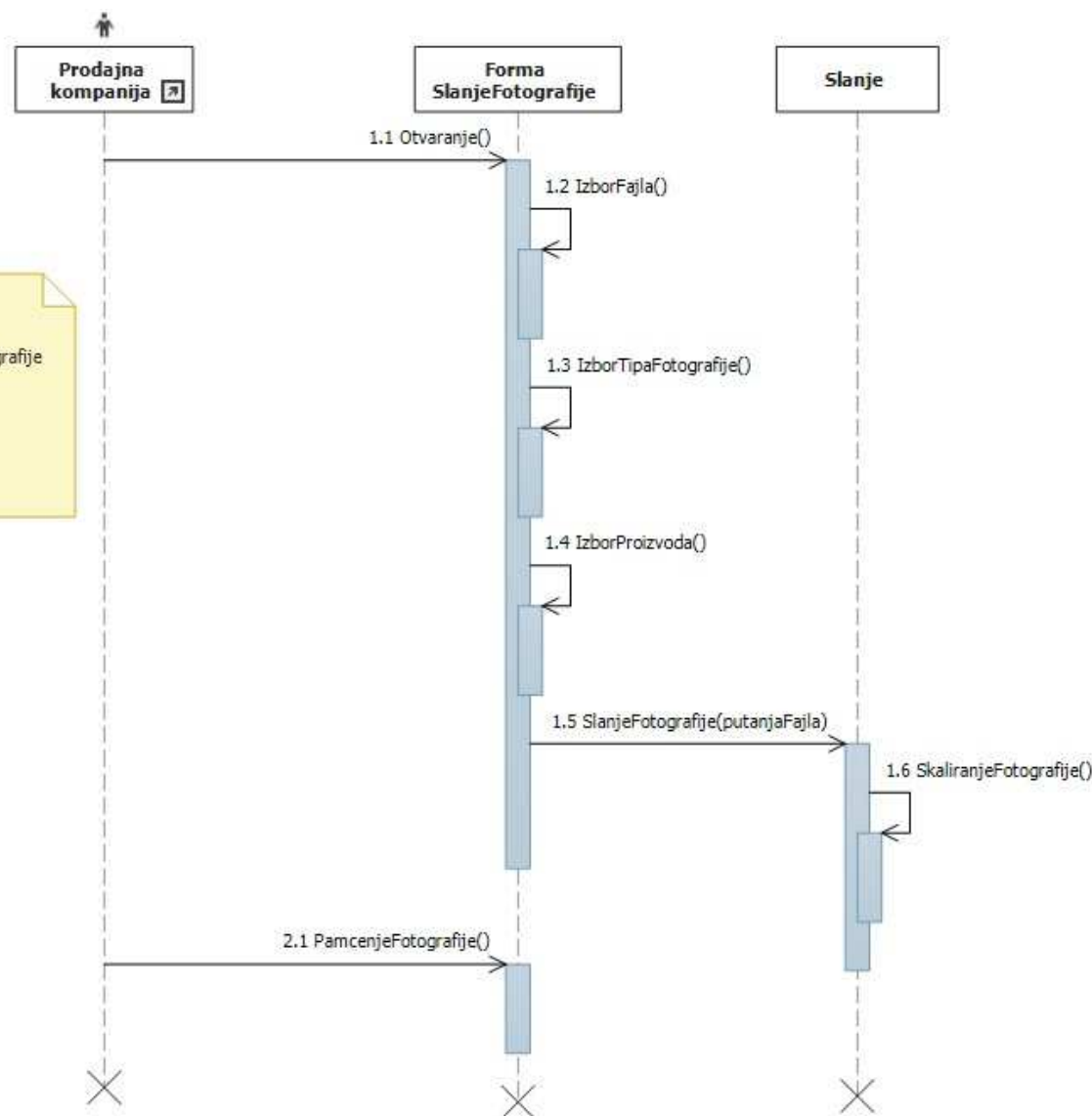
- Korisnički zahtevi se analiziraju kroz model slučajeva korišćenja (*Use Case*)
  - učesnici (akteri)
  - slučajevi korišćenja
  - veze između njih
- Dijagrami i njihove veze su samo pomoćna sredstva
  - *slučajevi korišćenja su tekstualni dokumenti - njihova izrada je pisanje teksta*



# Početni diagram sekvenci slučaja

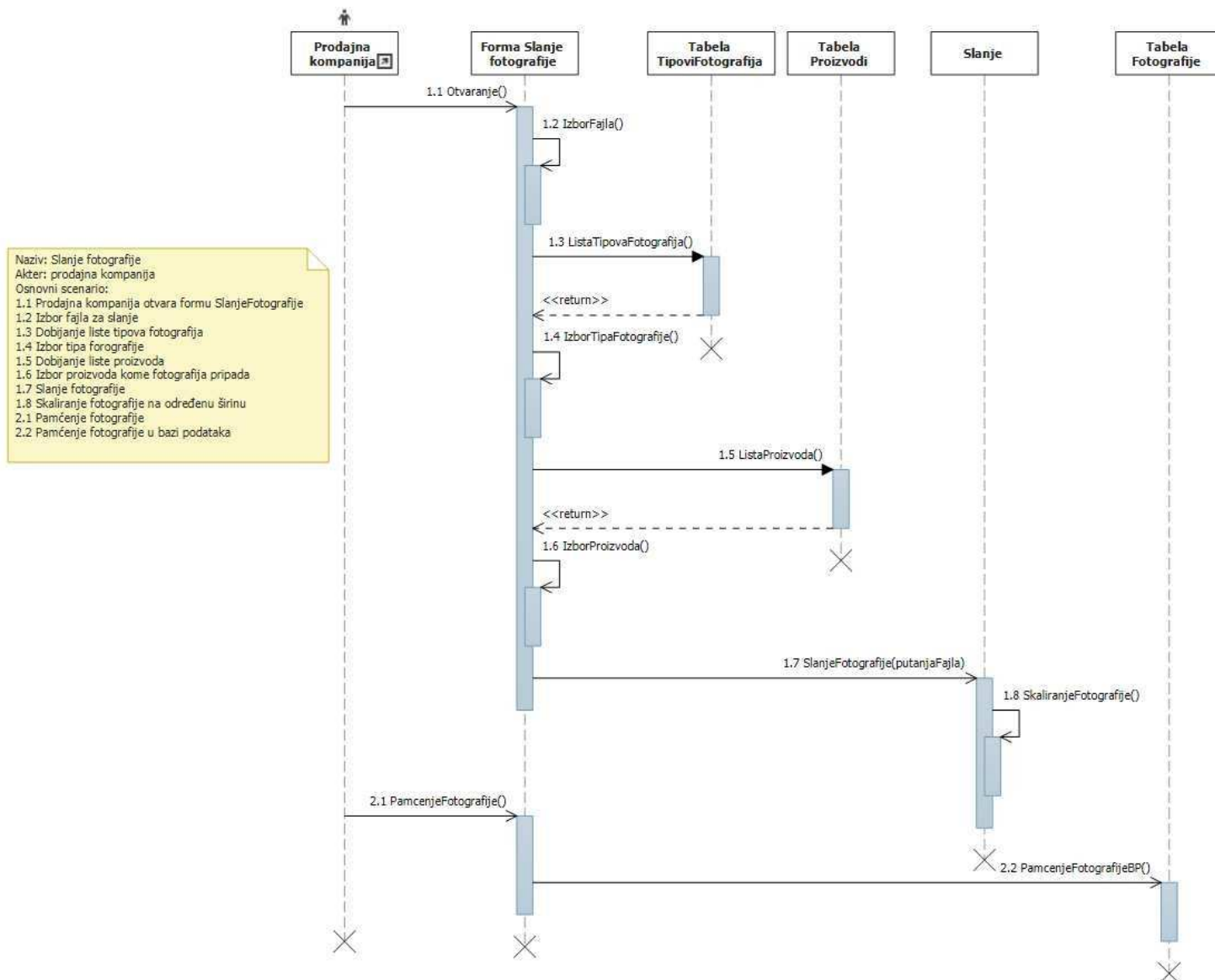
## *Slanje fotografija (upload)*

Naziv: Slanje fotografije  
 Akter: prodajna kompanija  
 Osnovni scenario:  
 1.1 Prodajna kompanija otvara formu SlanjeFotografije  
 1.2 Izbor fajla za slanje  
 1.3 Izbor tipa fotografije  
 1.4 Izbor proizvoda kome fotografija pripada  
 1.5 Prodajna kompanija šalje fotografiju  
 1.6 Skaliranje fotografije na određenu širinu  
 2.1 Pamćenje fotografije



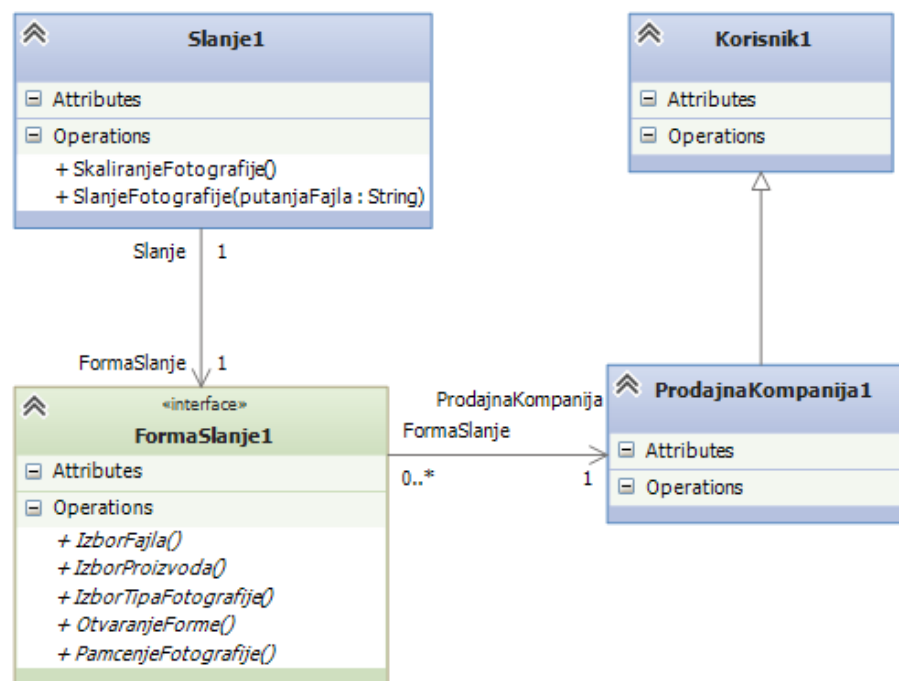
# Diagram sekvenci slučaja

## *Slanje fotografija (upload)*

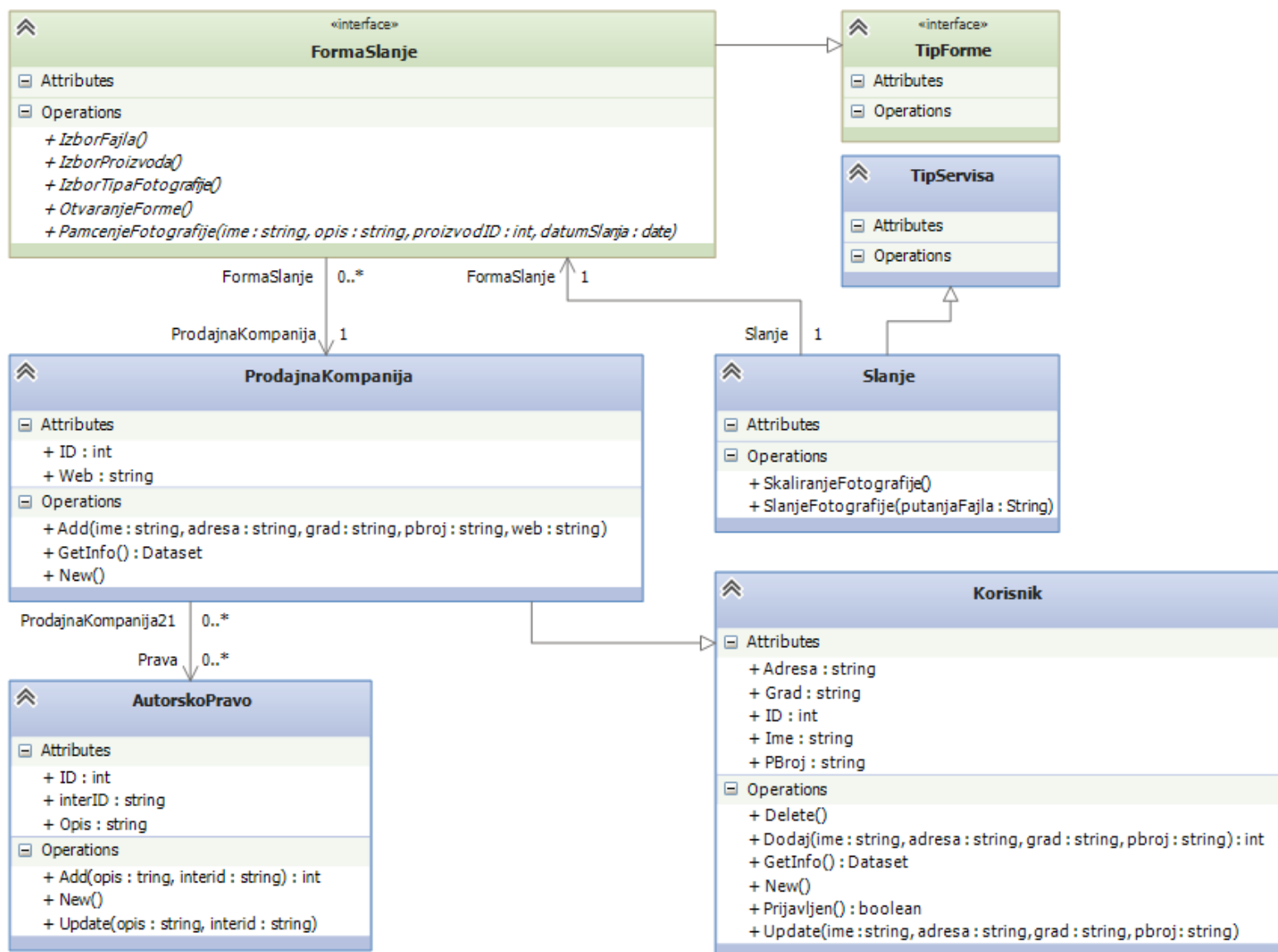


# Dijagram klasa (početni)

- Klase s trenutno poznatim elementima
  - Slanje (upload)
  - FormaSlanje
  - Korisnik
  - ProdajnaKompanija



# Dijagram klasa (konačna verzija)

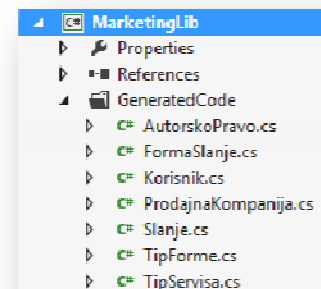


# Struktura generisanog koda (jezik se posebno bira: za Veb aplikaciju C#)

```

1  //-----
2  // <auto-generated>
3  //   This code was generated by a tool
4  //   Changes to this file will be lost if the code is regenerated.
5  // </auto-generated>
6  //-----
7  using System;
8  using System.Collections.Generic;
9  using System.Linq;
10 using System.Text;
11
12 public class AutorskoPravo
13 {
14     public virtual int ID
15     {
16         get;
17         set;
18     }
19
20     public virtual string Opis
21     {
22         get;
23         set;
24     }
25
26     public virtual string interID
27     {
28         get;
29         set;
30     }
31
32     public virtual void New()
33     {
34         throw new System.NotImplementedException();
35     }
36
37     public virtual int Add(string opis, string interid)
38     {
39         throw new System.NotImplementedException();
40     }
41
42     public virtual void Update(string opis, string interid)
43     {
44         throw new System.NotImplementedException();
45     }
46
47 }
48

```



Slanje

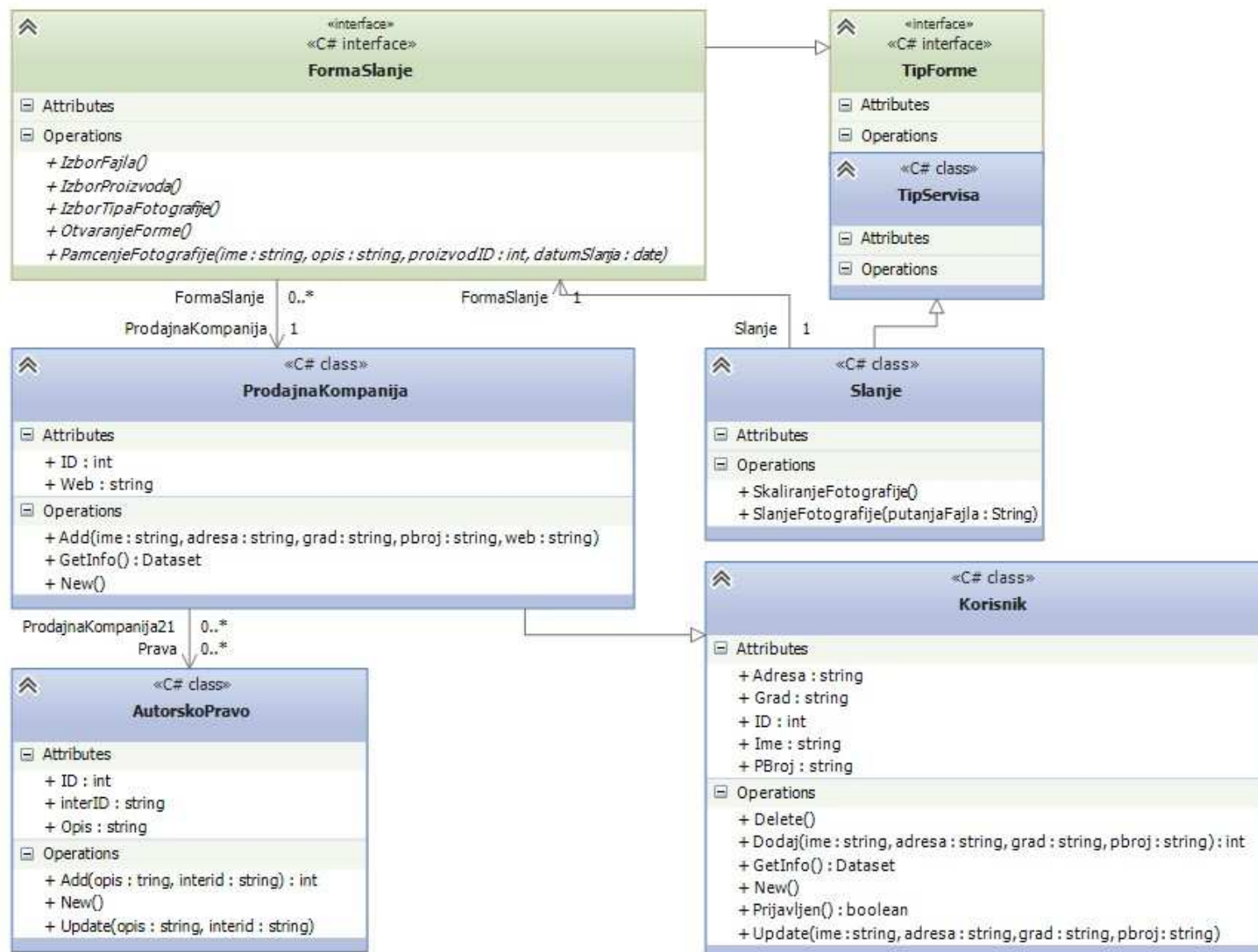
```

1  //-----
2  // <auto-generated>
3  //   This code was generated by a tool
4  //   Changes to this file will be lost if the code is regenerated.
5  // </auto-generated>
6  //-----
7  using System;
8  using System.Collections.Generic;
9  using System.Linq;
10 using System.Text;
11
12 public class Slanje : TipServisa
13 {
14     public virtual FormaSlanje FormaSlanje
15     {
16         get;
17         set;
18     }
19
20     public virtual void SlanjeFotografije(string putanjaFajla)
21     {
22         throw new System.NotImplementedException();
23     }
24
25     public virtual void SkaliranjeFotografije()
26     {
27         throw new System.NotImplementedException();
28     }
29
30 }
31

```



# *Dijagram klasa nakon generisanja koda*





# Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Liang D., *Introduction to Programming With C++*, Pearson Education, 2014
3. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
4. Deitel P, Deitel H., *C++ How to Program*, 9th Ed, Pearson Education, 2014
5. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
6. Horton I., *Beginning C++*, Apress, 2014
7. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
8. Veb izvori
  - <http://www.cplusplus.com/doc/tutorial/>
  - <http://www.learncpp.com/>
  - <http://www.stroustrup.com/>
9. Knjige i priručnici za *Visual Studio* 2010/2012/2013/2015/2017/2019

