Tema 04

Objektno orijentisano programiranje u jeziku C++

Prof. dr Miodrag Živković

Tehnički fakultet

OBJEKTNO ORIJENTISANO PROGRAMIRANJE 2

Sadržaj

1. Uvod (podsetnik)
2. Klase i objekti u jeziku C++
3. Konstruktori i destruktor
4. Upotreba objekata
5. Klasa string
6. Nasleđivanje i izvedene klase u jeziku C++

1. Uvod (podsetnik)

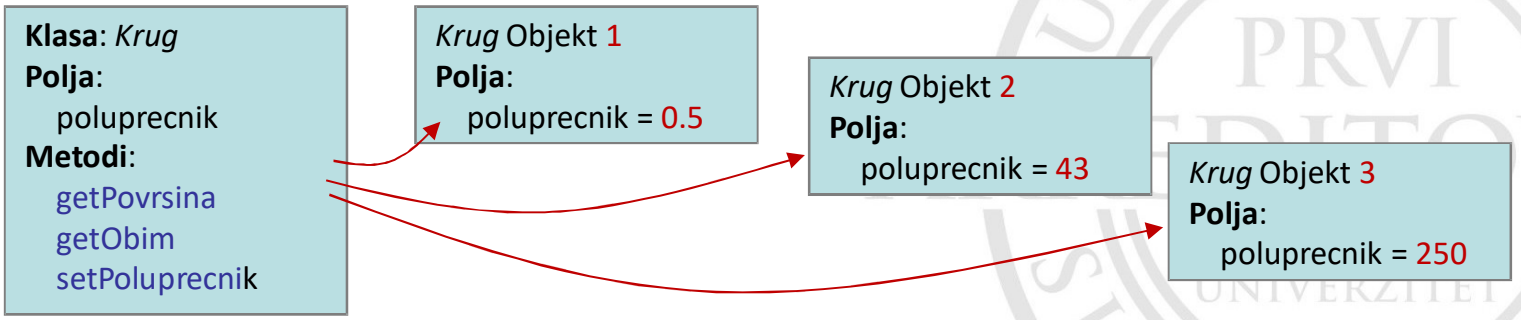
- Objektno orijentisani razvoj softvera
- Pojam objekta i klase

Objektno orijentisani razvoj softvera

- Objektno orijentisani pristup rešava mnoge probleme koji su svojstveni proceduralnim programiranju, gde su podaci i operacije su međusobno *razdvojeni*, tako da je neophodno slanje podataka metodima
- Objektno orijentisano programiranje smešta podatke i operacije koje se na njih odnose *zajedno*, u objektu
  - program se može posmatrati kao kolekcija *objekata* koji međusobno sarađuju
- Korišćenje objekata poboljšava višestruku upotrebljivost softvera i program čini lakšim za razvoj i održavanje

Pojam objekta i klase

- Klasa predstavlja opis objekata iste vrste, koji imaju slična svojstva: analogija *recepta* za kolač (klasa) i *kolača* (objekt)
  - korisnički definisana klasa koristi promenljive kao polja za smeštanje podataka i definiše metode za izvršavanje akcija
  - predstavlja obrazac (*template*), nacrt (*blueprint*) ili ugovor (*contract*)
- Objekt je primerak ili instanca (*instance*) klase
- Kreiranje instance klase se naziva instancijacija (*instantiation*)

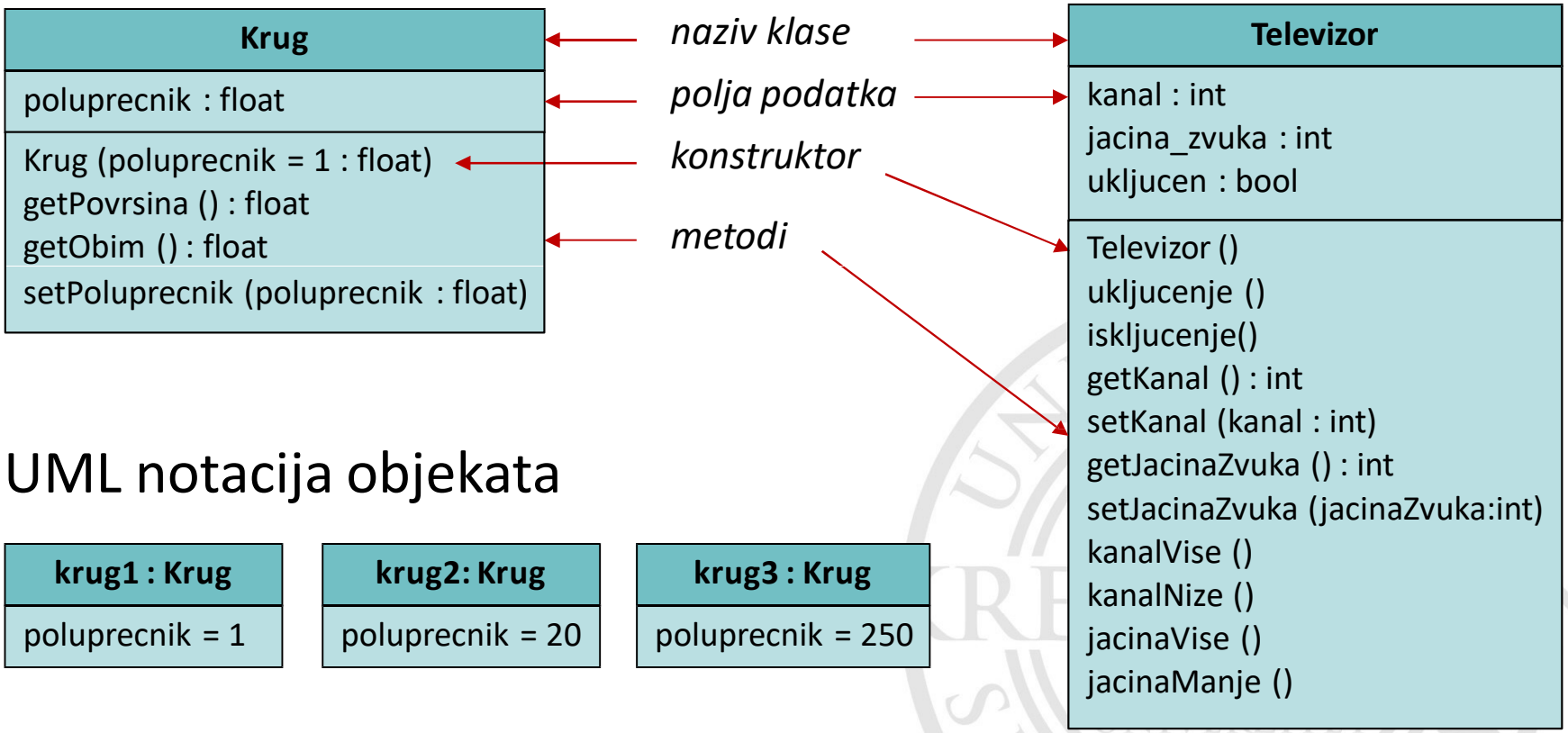


obrazac klase Krug

tri objekta klase Krug

Prikaz klasa u jeziku UML

- UML dijagrami klasa



- UML notacija objekata

2. Klase i objekti u jeziku C++

1. Deklarisanje klase
2. Enkapsuliranje klase
3. Definisanje funkcija članova klase
4. Pristup članovima klase
5. Ugrađene (inline) funkcije

2.1 Deklarisanje klase

- U jeziku C++ klasa se deklariše naredbom koja ima opšti oblik

```
class naziv {lista_članova} lista_objekata;
```

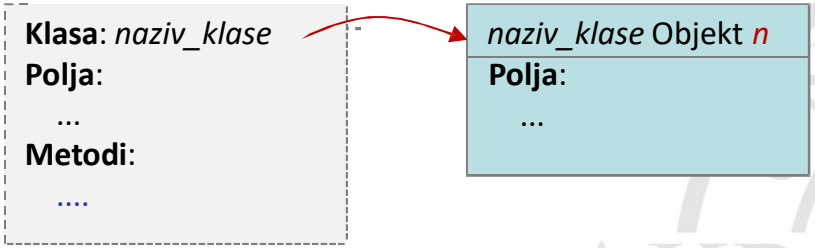
  - naziv klase je novi korisnički definisani tip podataka
  - lista članova je spisak članova klase - podataka (*member data*) i funkcija (*member functions*) .
    Članovi klase mogu biti *privatni* i *javni*.
    Privatnim članovima pristupaju samo drugi članovi klase, a javnim i drugi delovi programa. Podrazumevajući je privatni pristup; za javni pristup članovima se navodi ključna reč public
  - lista objekata (nije obavezna) je spisak naziva *objekata* novog tipa
- Načelno, klasa treba da predstavlja logičku celinu i grupiše samo logički povezane informacije

Deklarisanje promenljive tipa klase

- Klasa predstavlja novi tip podataka koji se može koristiti za kreiranje objekata - instanci klase
- Deklaracija promenljive tipa klase

  ```
  naziv_klase naziv_promenljive;
  ```

  prouzrokuje *kreiranje objekta* za koji se rezerviše memorijski prostor



- Članovima klase pristupa se pomoću naziva objekta i operatora tačka

  ```
  objekt.član;
  ```

2.2 Enkapsuliranje klase

- Apstrakcija klase je izdvajanje i predstavljanje samo bitnih svojstava objekata
- Enkapsuliranje je *skrivanje* od korisnika detalja načina objedinjavanja podataka i metoda u jednu celinu, odnosno implementacije klase
  - razdvajanje implementacije klase od njene upotrebe
  - članovi klase navedeni kao private zaštićeni su od pristupa izvan same klase (skriveni ili enkapsulirani)
  - u stvarnosti su detalji implementacije različitih pojava/sistema skriveni od korisnika; npr. *tehnički sistemi*, kao što su računari i uređaji u domaćinstvu, za koje korisnci znaju samo funkcije i način njihove upotrebe ili *bankarski krediti*, za koje korisnici znaju uslove, rokove i efekte, dok ih detalji procesa odobravanja i obrade kredita ne zanimaju

Primer: Upotreba klasa u programu (1/2)

```
# include <iostream>
using namespace std;

// Deklarisanje klase bez funkcija članova
class Vozilo {
  public:
    int brojMesta;
    int kapacitetRezervoara;
    int potrosnja;
};

int main() {
  // Deklarisanje promenljivih
  Vozilo kombi;
  Vozilo automobil;
  double autonomijaKombi, autonomijaAuto;
  // Dodela vrednosti podacima članovima objekta kombi
  kombi.brojMesta = 7;
  kombi.kapacitetRezervoara = 60;
  kombi.potrosnja = 7.;
```

obavezno na kraju naredbe deklarisanja klase

Primer: Upotreba klasa u programu (2/2)

```
  // Dodela vrednosti podacima članovima objekta automobil
  automobil.brojMesta = 5;
  automobil.kapacitetRezervoara = 50;
  automobil.potrosnja = 5.;

  autonomijaKombi = kombi.kapacitetRezervoara / kombi.potrosnja*100;
  autonomijaAuto = automobil.kapacitetRezervoara / automobil.potrosnja*100;

  cout << "U kombi staje " << kombi.brojMesta
       << " putnika. S punim rezervoarom prelazi "
       << autonomijaKombi << " kilometara." << endl;

  cout << "U automobil staje " << automobil.brojMesta
       << " putnika. S punim rezervoarom prelazi "
       << autonomijaAuto << " kilometara." << endl;
  return 0
}
```

Izvršavanje programa:

```
U kombi staje 7 putnika. S punim rezervoarom prelazi 800 kilometara.
U automobil staje 5 putnika. S punim rezervoarom prelazi 1000 kilometara.
```

2.3 Definisanje funkcija članova klase

- Funkcije članovi klase zadaju se pomoću *prototipa*, npr.

```
class Vozilo {
  public:
    int brojMesta;
    int kapacitetRezervoara;
    int potrosnja;
    int autonomija();  // funkcija član klase
}
```

- Implementacija funkcije člana navodi se zasebno, uz upotrebu operatora razrešenja dosega ::

```
// Implementacija funkcije člana: autonomija
int Vozilo::autonomija() {
  return kapacitetRezervoara / potrosnja*100; //direktan pristup čl.
}
```

2.4 Pristup članovima klase

- Članovi klase su podaci i metodi (funkcije), kojima se pristupa pomoću naziva objekta i operatora tačka , npr.

```
objekt.polje;
objekt.funkcija(...);
```

- Pristup članovima klase definiše se pomoću specifikacija pristupa public: i private:
- Podrazumeva se *privatni* pristup, samo za funkcije članove klase, koji je direktan, bez operatora tačka
- Ipak, dobra praksa programiranja je da se specifikcije pristupa navode *eksplicitno* i to prvo privatni, a zatim javni članovi klase

Primer: Upotreba javnih polja klase

```
# include <iostream>
using namespace std;

// Deklarisanje klase
class MojaKlasa {
  public: int i, j, k;
};

int main() {
  // Deklarisanje promenljivih
  MojaKlasa a, b;
  // Polja i, j, k su vidljiva
  a.i = 100;
  a.j = 4;
  a.k = a.i * a.j;
  b.k = 12;
  cout << a.k << " " << b.k;  // ispis: 400 12
}
```

2.5 Ugrađene (inline) funkcije

- Moguće je definisati kompletnu funkciju unutar deklaracije klase

```
class Brojac {
  private:
    int i;
  public:
    int broj { return ++i;}  // inline funkcija član
}
```

- Takva funkcija se naziva ugrađena (*inline*) funkcija. Prevodilac *ugrađuje kod* takve funkcije na mestima gde se ona poziva
- Ugrađena funkcija se može definisati i izvan deklaracije klase:

```
inline int f () { ... }
```

3. Konstruktori i destruktor

1. Pojam konstruktora i destruktora
2. Upotreba konstruktora i destruktora
3. Parametri konstruktora
4. Preklapanje konstruktora
5. Podrazumevani konstruktori
6. Konstruktor kopije

3.1 Pojam konstruktora i destruktora

- Konstruktor je funkcija član klase koja se automatski poziva prilikom kreiranja objekta, obično radi dodele početnih vrednosti članovima klase
- U jeziku C++ konstruktor je funkcija istog naziva kao i klasa, koja ne vraća rezultat i za koju tip rezultata nije definisan
- Sintaksa definicije konstruktora je:

```
naziv_klase () {
  // telo konstruktora
}
```

Pojam konstruktora i destruktora

- Destruktor je funkcija član klase koja se automatski poziva radi izvođenja različitih operacija prilikom uništavanja objekta
- Destruktor je funkcija naziva kao i klasa s dodatkom ~ ispred, koja ne vraća rezultat i za koju tip rezultata nije definisan
- Sintaksa definicije destruktora je:

```
~naziv_klase () {
  // telo destruktora
}
```

- Klasa ima samo jedan destruktor; ako se posebno ne navede, prevodilac će ga kreirati automatski

Primer: Definicija konstruktora i destruktora klase

```
# include <iostream>
using namespace std;

// Deklaracija klase
class Primer {
  public:
    Primer();   // prototip konstruktora
    ~Primer();  // prototip destruktora
};
// Definicija konstruktora i destruktora klase
Primer::Primer(){
  cout << "u konstruktoru" << endl;
}
Primer::~Primer(){
  cout << "u destruktoru" << endl;
}

int main() {
  Primer obj;   // pokreće konstruktor
  cout << "Program koji demostrira objekt" << endl;
  return 0;     // pokreće destruktor
}
```

Izvršavanje programa:

```
u konstruktoru
Program koji demonstrira objekt
u destruktoru
```

3.2 Upotreba konstruktora i destruktora

- U jeziku C++ konstruktori globalnih objekata pokreću se na početku programa, pre početka izvršavanja funkcije main(), po redosledu deklarisanja
- Deklaracija svake lokalne promenljive pokreće konstruktor lokalnog objekta
- Destruktori lokalnih objekata pokreću se u obrnutom redosledu od redosleda pokretanja konstruktora
- Destruktori globalnih objekata pokreću se nakon završetka funkcije main()

Primer: Redosled izvršavanja konstruktora i destruktora

```
# include <iostream>
using namespace std;

class MojaKlasa {
  public:
    int koji;
    MojaKlasa(int id);
    ~MojaKlasa();
} glob_obj1(1), glob_obj2(2);  // kreirana dva objekta

MojaKlasa::MojaKlasa(int id) {
  cout << "Inicijalizuje se " << id << endl;
  koji = id;
}

MojaKlasa:: ~ MojaKlasa() {
  cout << "Unistava se " << koji << endl;
}

int main() {
  MojaKlasa lokalni_obj1(3); // kreirana objekt 3
  cout << "Ovo nece biti prvi prikazani red." << endl;
  MojaKlasa lokalni_obj2(4); // kreirana objekt 4
  return 0;
}
```

Izvršavanje programa:

```
Inicijalizuje se 1
Inicijalizuje se 2
Inicijalizuje se 3
Ovo nece biti prvi prikazani red.
Inicijalizuje se 4
Unistava se 4
Unistava se 3
Unistava se 2
Unistava se 1
```

3.3 Parametri konstruktora

- Konstruktori su funkcije koje mogu imati parametre, koji prenose vrednosti za inicijalizaciju novog objekta, npr.

```
MojaKlasa obj(3, 5);
```

odgovara naredbi

```
MojaKlasa obj = mojaKlasa(3, 5);
```

- Parametri konstruktora mogu imati podrazumevane vrednosti
- Ako konstruktor ima samo jedan parametar, za dodelu vrednosti se može koristiti *inline* funkcija konstruktor

```
MojaKlasa(int param) { promenljiva = param; } // inline k.
int get_promenljiva() { return promenljiva; } // pristup
```

a vrednost se novom objektu dodeljuje kao

```
MojaKlasa obj = vrednost;
```

Primer: Parametri konstruktora

```
# include <iostream>
using namespace std;

class MojaKlasa {
  int a, b;
  public:
    // Inline konstruktor s parametrima
    MojaKlasa(int i, int j) {
       a = i;
       b = j;
    }
    void prikazi(){
      cout << a << " " << b << endl;
    }
};

int main() {
    MojaKlasa obj(3, 5);
    obj.prikazi();
    return 0;
}
```

Izvršavanje programa:

```
3 5
```

Primer: Podrazumevane vrednosti konstruktora

```
# include <iostream>
using namespace std;

class Osoba {
    char *imePrezime;
    int godine;
  public:
    Osoba(char *ime="Petar Petrovic", int godine=21); // podrazumevane vrednosti konstruktora
    void koJeOsoba();
};

// Konstruktor
Osoba::Osoba(char *i, int g) {
  imePrezime = i; godine = g;
}

void Osoba::koJeOsoba() {
  cout << imePrezime << endl;
}

int main() {
  Osoba o;        // Konstruktor s podrazumevanim vrednostima
  o.koJeOsoba()  // Ispisuje podrazumevanu vrednost Petar Petrovic
  return 0
}
```

Izvršavanje programa:

```
Petar Petrovic
```

3.4 Preklapanje konstruktora

- Prilagodljivost klasa različitim potrebama i tipovima objekata obezbeđuje se preklapanjem konstruktora
- Za svaki od mogućih načina kreiranja objekta obezbedi se poseban konstruktor
- Primer je unos datuma, koji se može zadati na dva načina:
  - u obliku tri cela broja (*dan*, *mesec*, *godina*) ili
  - kao datumski string *dd.mm.gggg*
- Klasa datum može da prihvati oba načina inicijalizacije ako se predvide dva (preklopljena) konstruktora

Primer: Preklapanje konstruktora

```
# include <iostream>
# include <cstdio>
using namespace std;

class Datum {
    int dan, mesec, godina;
  public:
    Datum(char *d);
    Datum(int d, int m, int g);
    void prikazi_datum();
};

// Inicijalizacija za vrednosti tipa string
Datum::Datum(char *d) {
  sscanf(d, %d.%d.%d", &dan, &mesec, &godina);
}

// Inicijalizacija za celobrojne vrednosti
Datum::Datum(int d, int m, int g) {
  dan = d; mesec = m; godina = g;
}
```

```
void Datum::prikazi_datum() {
  cout << dan << "." << mesec << "." <<
    godina << endl;
}

int main() {
  // Inicijalizacija datuma na dva načina
  Datum ob1(24, 3, 2020),
        ob2("24.3.2020");
  ob1.prikazi_datum();
  ob2.prikazi_datum();
  return 0;
}
```

Izvršavanje programa:

```
24.3.2020
24.3.2020
```

sscanf: čitanje podataka iz stringa *s*
Specifikacija formata (d - *cifra*) : %[*][*width*][*length*]*specifier*

3.5 Podrazumevani konstruktori

- Ako se klasa deklariše bez eksplicitno opisanog konstruktora, prevodilac će generisati podrazumevani (*default*) konstruktor, koji samo kreira objekt klase, *bez inicijalizacije* podataka članova klase
- Primer
  - deklaracija objekta klase bez argumenata iz prethodnog primera

```
Datum d;
```

izazvala bi grešku. Za ovu vrstu deklaracije objekta, u primeru bi klasi Datum trebalo dodati još jedan konstruktor, koji *nema argumente*

```
...
Datum(); // default konstruktor klase datum
...
```

3.6 Konstruktor kopije

- Konstruktor kopije (*copy constructor*) je posebna vrsta konstruktora za *inicijalizaciju* objekta drugim objektom
- Problem je što standardna dodela vrednosti

```
MojaKlasa B = A;// objekt B se kreira kao identična kopija A
```

kreira vernu kopiju objekta A, tako da objekt B koristi istu memoriju kao i objekt A i *iste pokazivače* na promenljive u memoriji koju će destruktor *jednog* od objekata osloboditi

- Da se to izbegne kreira se poseban konstruktor za kopije objekta, koji definiše nepromenljivu *referencu* na objekt

```
naziv_klase (const naziv_klase &objekt ...) {
  // telo konstruktora
}
```

Dodela vrednosti i inicijalizacija

- Dodela vrednosti objekta drugom objektu različito se tretira kad je u pitanju standardna dodela vrednosti i inicijalizacija
- Inicijalizacija može biti
  - inicijalizacija jednog objekta drugim
  - kreiranje privremenog objekta
  - kopija objekta kao argumenta funkcije
- Konstruktor kopije se koristi *samo za inicijalizaciju*, dok se u dodeli vrednosti ne koristi, npr.

```
MojaKlasa x=y; // inicijalizacija koristi konstruktor kopije
f(y);          // argument koristi konstruktor kopije
MojaKlasa a, b;
a = b;         // dodela ne koristi konstruktor kopije!
```

Ilustracija: Upotreba konstruktora kopije

```
class Osoba {
    char *ime;
    int godine;
  public:
    Osoba(char *i, int g) {          // konstruktor
      ime = new char[strlen(i)+1];
      strcpy(ime, i);
      godine = g;
    }
    Osoba(const Osoba &osoba) {      // konstruktor kopije
      ime = new char[strlen(osoba.ime)+1];
      strcpy(ime, osoba.ime);
      godine = osoba.godine;
    }
    ~Osoba() { delete[] ime; }       // destruktor
};
```

4. Upotreba objekata

1. Zajednički članovi klasa
2. Objekti kao parametri funkcije
3. Dodela vrednosti objekata
4. Pokazivači na objekte
5. Pokazivač this

4.1 Zajednički članovi klasa

- Deklaracija `static` ispred podatka člana klase znači da postoji samo jedna kopija tog podatka za sve kreirane objekte klase
- Takav podatak je *zajednički podatak član*, koji dele svi objekti klase
- Sve statičke promenljive klasa, odnosno zajednički podaci članovi, deklarišu se izvan klase i inicijalizuje na `0` pre kreiranja prvog objekta
- Statičke promenljive se koriste za pristup nekom deljenom resursu, npr. istom fajlu ili za praćenje ukupnog broja objekata neke klase

Primer: Upotreba statičkog podatka člana

```
# include <iostream>
using namespace std;

class Deljena {
    static int a;
    int b;
  public:
    void set(int i, int j) {a=i; b=j;}
    void prikazi();
};

int Deljena::a; // pristup statičkom
                // članu i bez objekta
void Deljena::prikazi() {
  cout << "Staticko a: " << a << endl;
  cout << "Nestaticko b: " << b <<
    endl;
}
```

```
int main() {
  Deljena x, y;
  x.set(1, 1);  // a poprima vrednost 1
  x.prikazi();
  y.set(2, 2);  // a poprima vrednost 2
  y.prikazi()
  // (statički) a je promenjen i za x
  x.prikazi()
  return 0;
}
```

Izvršavanje program a:

```
Staticko a: 1
Nestaticko b: 1
Staticko a: 2
Nestaticko b: 2
Staticko a: 2
Nestaticko b: 1
```

Primer: Upotreba statičkog podatka člana za praćenje ukupnog broja objekata

```
# include <iostream>
using namespace std;

class Brojac {
  public:
    static int broj;
    Brojac() { broj++; } // konstruktor
    ~Brojac() { broj--; } // destruktor
};

int Brojac::broj; // pristup statičkom
                  // članu bez objekta
void f() {
  Brojac temp; // konstruktor (++)
  cout << "Ukupno objekata: ";
  cout << Brojac::broj << endl;
  // temp se unistava nakon zavrsetka f
  // destruktor (--)
}
```

```
int main() {
  Brojac o1;      // konstruktor (++)
  cout << "Ukupno objekata: ";
  cout << Brojac::broj << endl;
  Brojac o2;     // konstruktor (++)
  f();
  cout << "Ukupno objekata: ";
  cout << Brojac::broj << endl;
  return 0;
}
```

Izvršavanje programa:

```
Ukupno objekata: 1
Ukupno objekata: 2
Ukupno objekata: 3
Ukupno objekata: 2
```

Funkcija kao zajednički član klase

- Funkcije se mogu deklarisati kao *statičke*
  - pristupaju samo statičkim poljima i mogu se koristiti za inicijalizaciju *privatnih* statičkih podataka članova klase
  - klasa može da ima dve verzije funkcije istog imena, *statičke* i *nestatičke*

```
# include <iostream>
using namespace std;

class Staticka {
  static int i;
  public:
    static int broj;
    static void init(int x) {i = x;}
    void prikazi() {cout << i << endl;}
};

int Staticka::i; // definiše se i

int main() {
  // Inicijalizacija statickih podataka
  // pre kreiranja objekata
  Staticka::init(100);
  Staticka x;   // kreiranje objekta x
  x.prikazi();  // prikazuje 100
  return 0;
}
```

4.2 Objekti kao parametri funkcije

- Objekti se mogu prosleđivati funkcijama *po vrednosti*, kada se kao argument prenosi *kopija* objekta (novi objekt)
- Prilikom kreiranja novog objekta ne pokreće se konstruktor (kreira se verna kopija objekta, bez inicijalizacije), ali se za uništavanja objekta pokreće destruktor da oslobodi memoriju
  - problem: npr. kada objekt koji se prenosi kao argument alocira i dealocira memoriju

    Memoriju može da oslobodi i destruktor kopije objekta i time originalni objekt učini neupotrebljivim

    Zbog toga se u takvim slučajevim se definiše poseban *konstruktor kopije*

Primer: Konstruktor i destruktor objekta argumenta funkcije

```
# include <iostream>
using namespace std;

class MojaKlasa {
    int i;
  public:
    MojaKlasa(int n);
    ~MojaKlasa();
    void set_i(int n) { i = n; }
    int get_i() { return i; }
};

MojaKlasa::mojaKlasa(int n) {
  i = n;
  cout << "Kreira se " << i << endl;
}

MojaKlasa::~mojaKlasa() {
 cout << "Unistava se " << i << endl;
}

void f(mojaKlasa obj);
```

```
int main() {
  MojaKlasa o(1);
  f(o);
  cout << "i u main(): ";
  cout << o.get_i() << endl;
  return 0;
}

void f(mojaKlasa obj) {
  obj.set_i(2);
  cout << "i u funkciji f(): "
       << obj.get_i() << endl;
}
```

Izvršavanje programa:

```
Kreira se 1
i u funkciji f(): 2
Unistava se 2
i u main(): 1
Unistava se 1
```

4.3 Dodela vrednosti objekata

- Objekti istog tipa mogu se dodeljivati jedan drugome naredbom dodele vrednosti
- Podaci objekta s desne strane kopiraju se u podatke objekta s leve strane
- Koji je rezultat izvršavanja programa na slici?

Izvršavanje programa:

```
# include <iostream>
using namespace std;

class MojaKlasa {
    int i;
  public:
    void set_i(int n) { i = n; }
    int get_i() { return i; }
};
int main() {
  MojaKlasa obj1, obj2;
  obj1.set_i(99);
  obj2 = obj1; // dodela objekta
  cout << "obj2.i: "
       << obj2.get_i() << endl;
  return 0;
}
```

4.4 Pokazivači na objekte

- Pristup objektima može se ostvariti preko *pokazivača*, kada se pristup članovima objekta ostvaruje pomoću *operatora* ->
- Npr. pristup funkciji članu se označava kao `p->funkcija(` , što je ekvivalentno notaciji `(*p).funkcija()`
  - zagrada je potrebna zbog višeg prioriteta operatora . od operatora *
- Operator adresiranja & koristi se i za objekte, kao i reference

```
# include <iostream>
using namespace std;

class MojaKlasa {
    int i;
  public:
    MojaKlasa(int j) { i = j; }
    int get_i() { return i; }
};

int main() {
  MojaKlasa obj(88), *p;
  p = &obj; // p pokazuje na obj
  cout << p->get_i() << endl;// 88
  return 0;
}
```

4.5 Pokazivač *this*

- Funkcije članovi klase pozivaju se za sve objekte neke klase
- Prilikom poziva, funkciji se implicitno šalje pokazivač na objekt za koji je pokrenuta, čiji je naziv this
- Uvek kad funkcija član klase koristi naziv nekog podatka člana, npr.

```
b = vrednost;
```

prevodilac automatski koristi pokazivač this da označi objekt u kome se nalazi promenljiva:

```
this->b = vrednost;
```

Primer: Implicitni pokazivač *this*

```
# include <iostream>
using namespace std;

class Pwr {
    double b;
    int e;
    double val;
  public:
    Pwr(double base, int exp);
    double get_pwr() { return val; }
};

Pwr::Pwr(double base, int exp) {
  b = base; // this->b = base;
  e = exp;  // this->e = exp;
  val = 1;  // this->val = 1;
  if (exp==0)
     return;
  for(; exp>0; exp--)
    val = val * b;
    // this->val = this->val * this->b;
}
```

```
int main() {
  Pwr x(4.0, 2), y(2.5, 1), z(5.7, 0);
  cout << x.get_pwr() << " ";
  cout << y.get_pwr() << " ";
  cout << z.get_pwr() << endl;
  return 0;
}
```

Izvršavanje programa:

```
16 2.5 1
```

5. Klasa string

- Upotreba klase string
- Inicijalizacija objekata klase string
- Funkcije klase string

Upotreba klase string

- Klasa string je ugrađena u jezik C++ preko programske biblioteke koja se aktivira pomoću datoteke zaglavlja <string>
  - objekti klase string koriste se kao objekti osnovnih klasa, npr.

```
string ime;                                    // deklarisanje
cout << "Unesite ime: ";
cin << ime;                                    // učitavanje
cout  << "Dobar dan, " << ime << endl; // ispis
```

  - učitavanje celog reda teksta klase string (s belinama) vrši se funkcijom `getlline(tok_objekt, string_objekt)`
  - poređenje stringova ove klase vrši se operatorima <, <=, >, >= i == . Konkatenacija stringova vrši se operatorima + i +=, npr.

```
string s1 = "ABC";
s2 = "DEF";
s3 = s1 + s2;
```

Inicijalizacija objekata klase string

`string adresa;`	Deklariše prazan string `adresa`
`string ime("Nikola Vukovic");`	Deklariše string objekat `ime` i inicijalizuje ga sa `"Nikola Vukovic"`
`string osoba2(osoba1)`	Deklariše string objekat `osoba1`, koji je kopija string objekta `osoba2`, pri čemu `osoba2` može da bude drugi string ili niz znakova
`string set1(set2, 5);`	Deklariše string objekat `set1` koji se inicijalizuje sa prvih pet znakova niza znakova `set2`
`string punRed ('z', 10);`	Deklariše string objekat `punRed` koji se inicijalizuje sa 10 znakova `'z'`
`string ime(punoIme, 0, 7);`	Deklariše string objekat `ime` koji se inicijalizuje podstringom stringa `punoIme`. Podstring ima 7 znakova i počinje od pozicije 0.

Funkcije klase string (1/2)

- Deo funkcija članica klase string naveden je je u tabeli

`str1.append(str2)`	Dodaje `str2` na `str1`. `str2` može biti string objekat ili niz znakova.
`str1.append(str2, x, n)`	`n` znakova objekta `str2` počev od pozicije `x` dodaju se objektu `str1`. Ako `str1` nema dovoljnu dužinu, kopiraće se onoliko znakova koliko može da stane.
`str1.append(str2, n);`	Prvih `n` znakova niza `str2` dodeljuju se `str1`
`str.append(n, 'z')`	`n` kopija znaka `'z'` dodeljuje se objektu `str`
`str1.assign(str2)`	Dodeljuje `str2` objektu `str1`. `str2` može da bude string objekat ili niz znakova.
`str1.assign(str2, x, n)`	`n` znakova objekta `str2` počev od pozicije `x` dodeljuje se `str1`. Ako `str1` nema dovoljnu dužinu, kopiraće se onoliko znakova koliko može da stane.
`str1.assign(str2, n)`	Prvih `n` znakova `str2` dodeljuje se objektu `str1`.
`str.assign(n, 'z')`	Dodeljuje `n` kopija znaka `'z'` objektu `str`.
`str.at(x)`	Vraća znak na poziciji `x` u objektu `str`.

Funkcije klase string (2/2)

`str.capacity()`	Vraća veličinu memorije koja je alocirana za string.
`str.clear()`	Briše sve znakove u stringu `str`.
`str1.compare(str2)`	Poredi stringove kao funkcija `strcmp` za C stringove, sa istim povratnim vrednostima. `str2` može da bude niz znakova ili string objekat.
`str1.compare(x, n, str2)`	Poredi stringove `str1` i `str2` počev od pozicije `x` narednih `n` znakova. Povratna vrednost je ista kao u funkciji `strcmp`. `str2` može da bude string objekat ili niz znakova.
`str1.copy(str2, x, n)`	Kopira n znakova niza znakova `str2` u `str1` počev od pozicije `x`. Ako `str2` nije dovoljno dugačak, funkcija kopira onoliko znakova koliko može da stane.
`str.data()`	Vraća niz znakova sa nulom na kraju, kao u `str`
`str.empty()`	Vraća `true` ako je `str` prazan.
`str.erase(x, n)`	Briše `n` znakova iz objekta `str` počev od pozicije `x`.

`str1.find(str2, x)`	Vraća prvu poziciju iza pozicije `x` gde se string `str2` nalazi u `str1`. `str2` može da bude string objekat ili niz znakova.
`str.find('z', x)`	Vraća prvu poziciju iza pozicije `x` na kojoj se znak `'z'` nalazi u `str1`
`str1.insert(x, str2)`	Umeće kopiju `str2` u `str1` počev pod pozicije `x`. `str2` može da bude string objekat ili niz znakova.
`str.insert(x, n, 'z')`	Umeće `'z'` `n` puta u `str` počev od pozicije `x`
`str.length()`	Vraća dužinu stringa `str`
`str1.replace(x, n, str2)`	Zamenjuje `n` znakova u `str1` počev od pozicije `x` znakovima iz string objekta `str2`
`str.size()`	Vraća dužinu stringa `str`
`str.substr(x, n)`	Vraća kopiju podstringa dugačkog `n` znakova koji počinje na poziciji `x` objekta `str`
`str1.swap(str2)`	Zamenjuje sadržaj `str1` sa `str2`

6. Nasleđivanje i izvedene klase

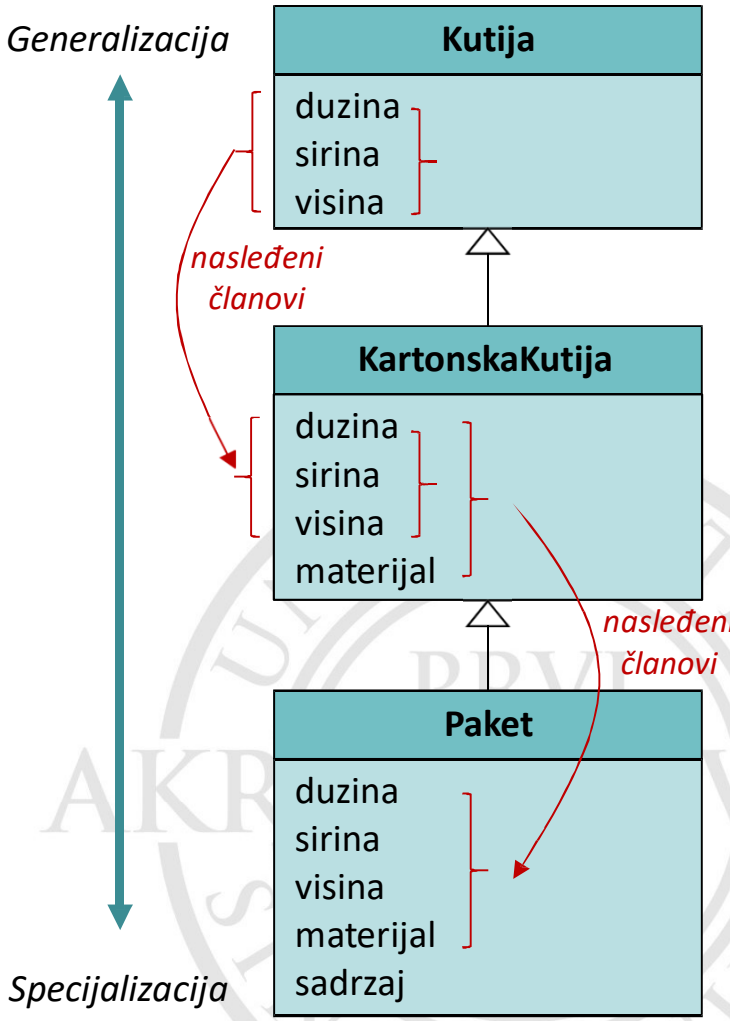
1. Nasleđivanje u klasama
2. Konstruktori u nasleđenoj klasi
3. Destruktori i nasleđivanje
4. Duplikati naziva članova klase
5. Višestruko nasleđivanje
6. Konverzija tipova povezanih klasa

6.1 Nasleđivanje u klasama

- Nasleđivanje je način da se nove klase kreiraju ponovnom upotrebom i proširenjem definicija postojećih klasa
- Klase modeliraju skupove entiteta koji imaju neke *zajedničke* osobine i ponašanje i mogu biti međusobno povezane na različite načine, kao i stvarni entiteti
- Objekti nekih klasa mogu biti sastavni elementi objekata drugih klasa ili mogu biti specijalni slučaj objekata drugih klasa, npr.
  - klase `Motor` i `Automobil` (motor je *sastavni deo* automobila)
  - klase `Pas` i `Sisar` (pas je *specijalni slučaj* sisara)

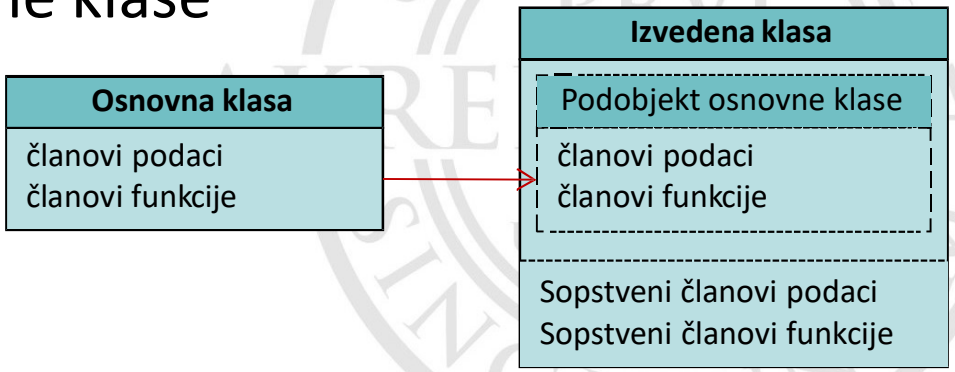
Hijerarhije klasa

- Između klasa može postojati višestruka veza generalizacije/specijalizacije (hijerarhija)
- Npr. klasa `KartonskaKutija` je *izvedena* iz osnovne klase `Kutija`, a klasa `Paket` iz osnovne klase `KartonskaKutija`
- U primeru su *izvedene* klase nasledile sve osobine *osnovnih* klasa



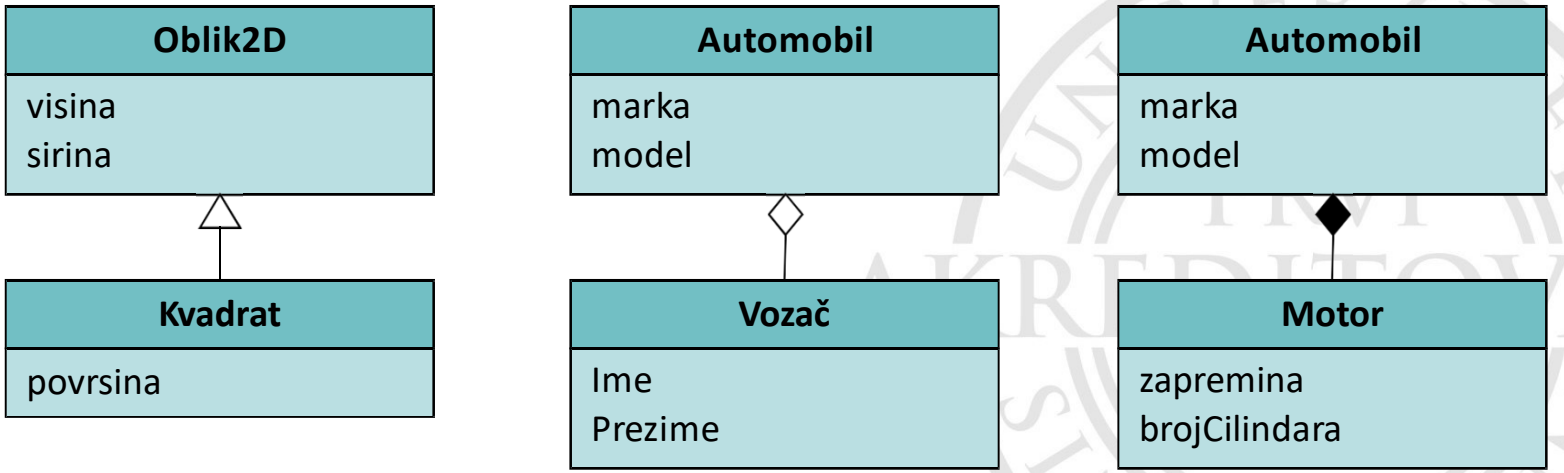
Izvedene klase

- Nasleđivanje omogućava novoj klasi da nasledi osobine neke druge postojeće klase
- Izvedena klasa je podklasa (*subclass*), a osnovna klasa je nadklasa (*superclass*)
- Nasleđivanje može biti *direktno* ili *indirektno*, preko druge izvedene klase
- Izvedena klasa sadrži sve članove podatke i (uz ograničenja) članove funkcije iz osnovne klase
- Izvedena klasa sadrži i sopstvene članove, podatke i funkcije



Nasleđivanje i agregacija

- Izvedena klasa može predstavljati npr. specijalni slučaj osnovne klase (*is-a*) ili može biti njen sastavni deo (*has-a*)
- Odgovarajuće veze između klasa nazivaju se *generalizacija* (Δ) i *agregacija* (◊) ili *kompozicija* (♦)
  - zavisno od toga da li se sastavni deo može izostaviti ili je obavezan



Izvođenje klase

- Sintaksa izvođenja klase je

```
class Izvedena_klasa : specifikacija Osnovna_klasa {
  // Telo izvedene klase
}
```

- *Specifikacija* izvođenja nije obavezna i može biti public, private ili protected
- Ako se izostavi, podrazumeva se specifikacija izvođenja *public*, koja označava da će svi javni članovi osnovne klase takođe biti javni u izvedenim klasama
- Izvedene klase omogućavaju lokalizaciju specifičnih svojstava nove klase u telu klase, čime se pojednostavljuje održavanje programa

Primer: Izvođenje klase (specijalizacija)

```
class Oblik2D {
  public:
    double visina;
    double sirina;
    void prikaziDimenzije() {
      cout << "sirina= " << sirina << "  visina= " << visina << endl;
    }
};

class Trougao: public Oblik2D {
  public:
    string vrsta;
    double povrsina() {
      return sirina * visina / 2;
    }
};
```

nasleđeni članovi:
visina, sirina, prikaziDimenzije()

sopstveni članovi

Zaštićeni članovi klase

- Podrazumevajuća specifikacija izvođenja public dozvoljava izvedenoj klasi pristup svim javnim članovima osnovne klase
- Pristup izvedene klase članovima osnovne klase može se *zabraniti* pomoću specifikacije izvođenja private
- Pristup izvedene klase članovima osnovne klase, uz istovremenu zaštitu od pristupa drugih klasa, može se omogućiti korišćenjem specifikacije izvođenja protected, npr.

```
class Kutija {
  protected:
    double duzina; double sirina; double visina;
  public:
  ...
}
```

Primer: Pristup zaštićenim članovima klase

```
class Osnovna {
  private:
    int privatni;
  protected:
    int zasticen;
  public:
    int javni;
};
class Izvedena: public Osnovna {
  public:
    void prikazi() {
      cout << zasticen << endl;
      cout << javni << endl;
      cout << privatni << endl;    // greška!
    }
};
```

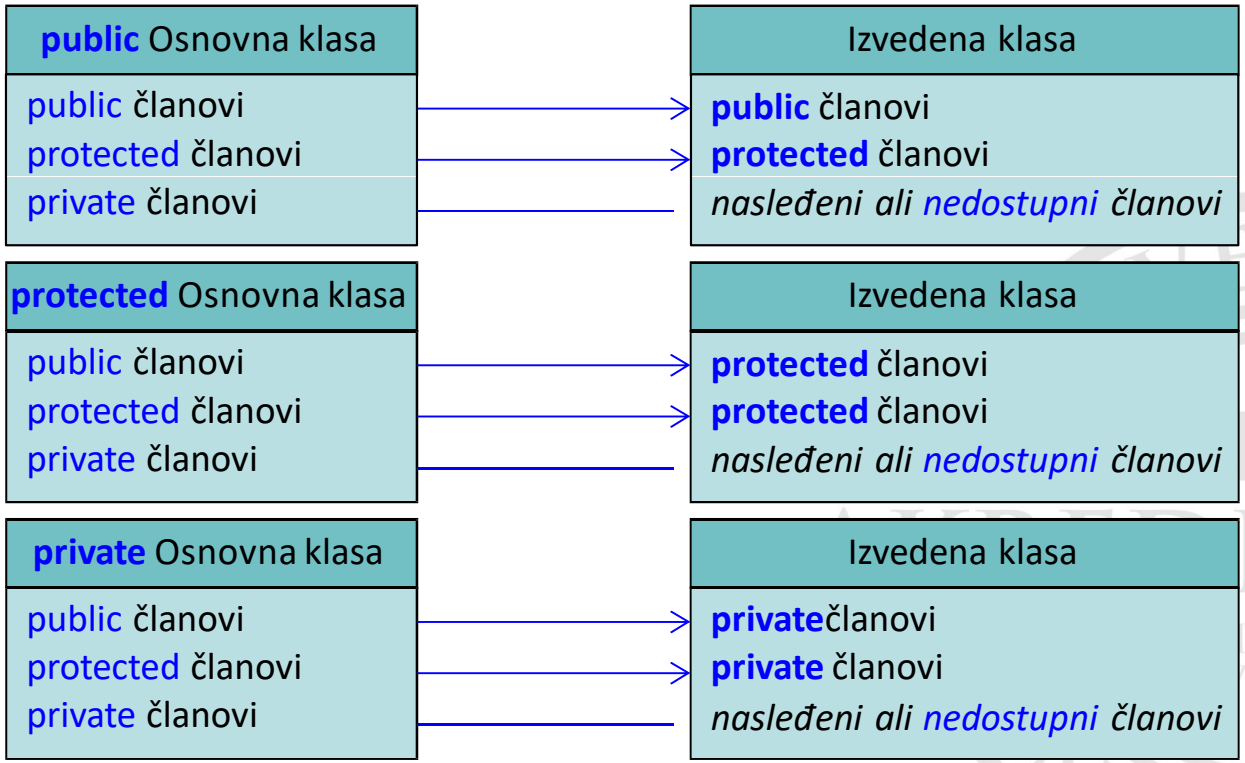
samo članovi osnovne i izvedene klase mogu da pristupe zaštićenim članovima klase

Specifikacija pristupa

- Izvođenjem klase dostupnost podataka u izvedenoj klasi može biti samo na istom ili nižem nivou
  - prilikom izvođenja klase iz osnovne specifikatorom pristupa public zaštićeni članovi ostaju zaštićeni i u izvedenoj klasi
  - kada se klasa izvodi specifikatorom private, zaštićeni članovi osnovne klase postaju private u izvedenoj klasi
- Javno izvođenje omogućava pristup javnim članovima osnovne klase i u izvedenoj klasi, tako da je izvedena klasa vrsta (*a kind of*) osnovne klase
- Privatno izvođenje skriva članove osnovne klase u izvedenoj klasi, tako da je izvedena klasa deo (*a part of*) osnovne klase, odnosno opisuje članstvo objekta ili *kompoziciju*

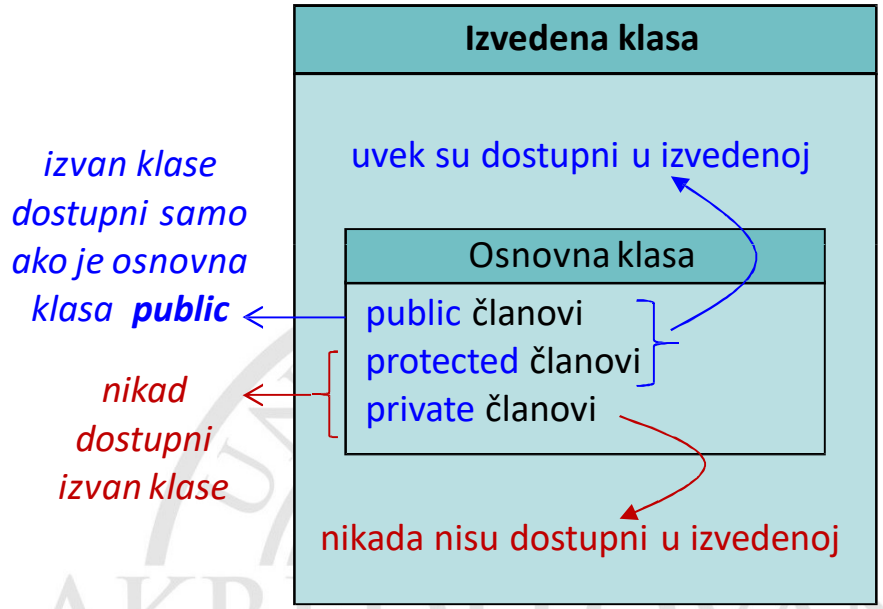
Pregled specifikacija pristupa

- Pristup članovima klase *istovremeno* zavisi od specifikacije pristupa u *osnovnoj* i *izvedenoj* klasi:



Izbor specifikacije pristupa u hijerarhijama klasa

- Javni (public) i zaštićeni (protected) članovi osnovne klase uz *specifikaciju* ...
  - …public ostaju *javni* i *zaštićeni* u izvedenoj klasi
  - …protected postaju *zaštićeni* članovi izvedene klase
  - …private postaju *privatni* članovi izvedene klase
- Privatni (private) članovi osnovne klase se nasleđuju, ali su tada *uvek nedostupni*



6.2 Konstruktori u nasleđenoj klasi

- Konstruktor izvedene klase
- Prenos argumenata konstruktoru osnovne klase
- Nasleđeni konstruktori

Konstruktor izvedene klase

- Konstruktor osnovne klase kreira deo objekta koji pripada osnovnoj klasi, a deo koji pripada izvedenoj klasi kreira konstruktor izvedene klase
- Kada su *u obe klase* definisani konstruktori, izvedena klasa mora eksplicitno da pokrene konstruktor osnovne klase, npr.

```
Naziv_izvedene_klase (lista_argumenata) :
  Konstruktor_osnovne_klase(lista_argumenata) {
    // Telo konstruktora izvedene klase
}
```

poziv konstruktora osnovne klase

- Kada *samo izvedena* klasa ima konstruktor, prilikom kreiranja objekta koristi se podrazumevani (*default*) konstruktor osnovne klase

Primer: Nasleđivanje klase koja *nema* definisan konstruktor (1/2)

```
#include <iostream>
#include <string>
using namespace std;

class Oblik2D {  // Klasa nema definisan konstruktor
  public:
    double visina;
    double sirina;
    void prikaziDimenzije() {
      cout << "sirina= " << sirina << "  visina= " << visina << endl;
    }
    double getVisina() const { return visina; }
    double getSirina() const { return sirina; }
    void setSirina(double s) { sirina = s; }
    void setVisina(double v) { visina = v; }
}
```

Primer: Nasleđivanje klase koja *nema* definisan konstruktor (2/2)

```
class Trougao: public Oblik2D { // Klasa ima definisan konstruktor
  public:
    string vrsta;
    double povrsina() { return sirina * visina / 2; }

    Trougao(string tip, double v, double s) {
      // Inicijalizacija osnovnog dela objekta (kreiranog automatski)
      setSirina(s);
      setVisina(v);

      // Inicijalizacija izvedenog dela objekta
      vrsta = tip;
    }

    void prikaziVrstu() {
    cout << "Trougao je>: " << vrsta << endl;    }
};
```

konstruktor izvedene klase

Prenos argumenata konstruktoru osnovne klase

- Konstruktor osnovne klase treba da inicijalizuje deo objekta koji pripada osnovnoj klasi na osnovu *argumenata* iz poziva
- Na osnovu prenesenih aktuelnih argumenata prevodilac bira odgovarajući konstruktor *osnovne* klase i *podataka članova*, a zatim konstruktor *izvedene* klase
- Prvo se poziva konstruktor *osnovne klase*, zatim konstruktori *podataka* članova i na kraju konstruktor *izvedene klase*
- Redosled pozivanja destruktora je obrnut

Primer: Nasleđivanje klase koja ima definisan konstruktor (1/2)

```
#include <iostream>
#include <string>
using namespace std;

class Oblik2D {
  private:
    double visina;
    double sirina;
  public:
    void prikaziDimenzije() {
      cout << "sirina= " << sirina << "  visina= " << visina << endl;
    }
    double getVisina() const { return visina; }
    double getSirina() const { return visina; }
    void setSirina(double s) { sirina = s; }
    void setVisina(double v) { visina = v; }
    // Konstruktor osnovne klase
    Oblik2D(double s, double v) { sirina = s; visina = v; }
};
```

Primer: Nasleđivanje klase koja ima definisan konstruktor (2/2)

```
class Trougao: public Oblik2D {
  public:
    string vrsta;
    double povrsina() { return getSirina() * getVisina() / 2; }

    Trougao(string tip, double s, double v) : Oblik2D (s,v) {
      // Inicijalizacija izvedenog dela
      vrsta = tip;
    }

    void prikaziVrstu() {
    cout << "Trougao je>: " << vrsta << endl;    }
};
int main() {
  Trougao t("jednakostranicni", 4.0, 4.0);
  t.prikaziDimenzije();
  t.prikaziVrstu();
  cout << "Povrsina trougla je: " << t.povrsina() << endl;
  return 0;
}
```

konstruktor izvedene klase pokreće konstruktor osnovne klase

```
sirina= 4  visina= 4
Trougao je>: jednakostranicni
Povrsina trougla je: 8
```

Nasleđeni konstruktori

- Izvedena klasa može da pokrene više oblika konstruktora osnovne klase, zavisno od prenesenih argumenata,npr.

```
Oblik2D::Oblik2D(double s, double v){ sirina=s; visina=v; }
Oblik2D::Oblik2D(double x){ sirina = visina = x; }
```

```
// Default konstruktor
Trougao::Trougao(){ vrsta = "nepoznat"; }
Trougao::Trougao(string tip, double v, double s) :
  Oblik2D(s, v) { vrsta = tip; }
// Konstruktor jednakostranicnog trougla
Trougao::Trougao(string tip, double x) :
  Oblik2D(x) { vrsta = "jednakostranicni"; }
```

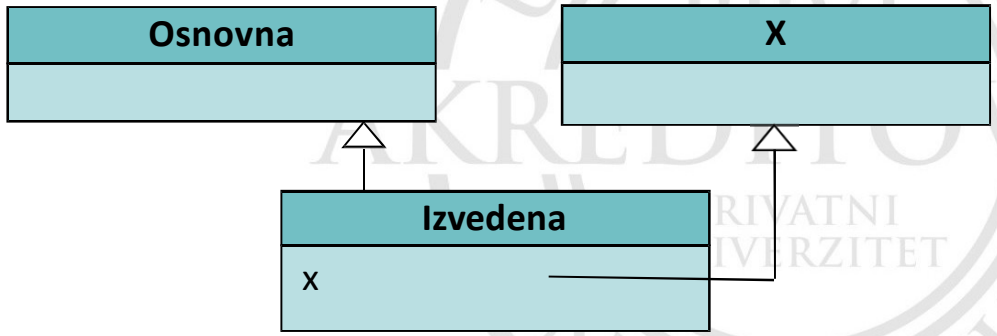
- Ako je neki parametar konstruktora osnovne klase obavezan, nasleđene klase ga moraju proslediti

6.3 Destruktori i nasleđivanje

- Brisanje objekta nasleđene klase
- Redosled pozivanja destruktora

Brisanje objekta nasleđene klase

- Uklanjanje objekta izvedene klase pokreće izvršavanje destruktora *osnovne* i destruktora *izvedene* klase
  - redosled pozivanja destruktora je obrnut od redosleda pozivanja konstruktora, odnosno redosleda izvođenja klasa: prvo se briše *izvedena* klasa, zatim *podaci članovi* i na kraju *osnovna* klasa, čime se obezbeđuje da u memoriji ne ostaju nepotpuni objekti čijim delovima se ne može pristupiti
- Primer: Kreiranje objekta izvedene klase koja ima član podatak x



Primer: Redosled pozivanja konstruktora i destruktora kod kreiranje objekta (1/2)

```
#include <iostream>
#include <string>
using namespace std;

class X {
 public:
   X() { cout << "konstruktor klase X" << endl; }
   ~X() { cout << "destruktor klase X" << endl; }
};

class Osnovna {
 public:
   Osnovna() { cout << "konstruktor osnovne klase" << endl; }
   ~Osnovna() { cout << "destruktor osnovne klase" << endl; }
};
```

Primer: Redosled pozivanja konstruktora i destruktora kod kreiranje objekta (2/2)

```
class Izvedena : public Osnovna {
  X x;    // ekvivalentno "class Izvedena: private X { ... };"
 public:
   Izvedena() { cout << "konstruktor izvedene klase" << endl; }
   ~Izvedena() { cout << "destruktor izvedene klase" << endl; }
};

int main() {
  // Kreiranje objekta izvedene klase (podobjekti Osnovna i X)
  Izvedena i;
  return 0;
}
```

```
konstruktor osnovne klase
konstruktor klase X
konstruktor izvedene klase
destruktor izvedene klase
destruktor klase X
destruktor osnovne klase
```

6.4 Duplikati naziva članova klase

- Duplikati naziva polja podataka
- Duplikati naziva funkcija članova

Duplikati naziva polja podataka

- Nazivi *polja podataka* članova osnovne i izvedene klase mogu biti isti i mogu se koristiti pomoću operatora razrešenja dosega, npr.

```
int Izvedena::ukupno() const {
  return vrednost + Osnovna::vrednost;
}
```

Duplikati naziva funkcija članova

- Nazivi *funkcija članova* osnovne i izvedene klase mogu biti isti, a koriste se u zavisnosti od njihovih *parametara*
- Ako su parametri *isti*, funkcije osnovne i izvedene klase se razlikuju pomoću *naziva klase*, npr.

```
Izvedena objekt;
objekt.Osnovna::funkcija();
```

- Ako se parametri funkcija *razlikuju*, funkcija član izvedene klase skriva funkciju člana osnovne klase istog naziva

  Tada se *u izvedenoj klasi* može definisati funkcija *novog* imena za pristup članu osnovne klase pomoću ključne reči *using*, npr.

```
using Osnovna::funkcija()  // koristi se osnovna
```

6.5 Višestruko nasleđivanje

- Osnovne klase
- Višeznačnost funkcija članova
- Ponovljeno nasleđivanje
- Virtuelne osnovne klase

Osnovne klase

- Nova klasa može istovremeno da nasledi *više osnovnih klasa*, tako što se u zaglavlju posebno navode *specifikacije* i *nazivi* svake od klasa, odvojene zarezima
- Svaki objekt izvedene klase nasleđuje *sve članove* osnovnih klasa, npr.

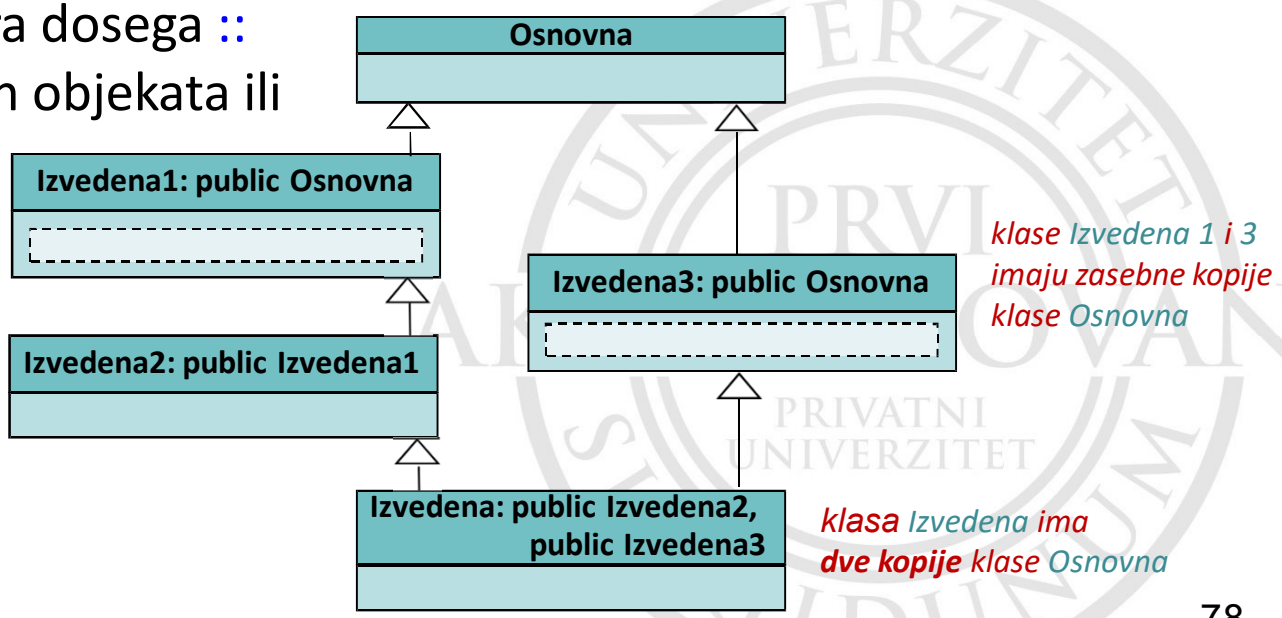
```
class Motocikl {
  // prva osnovna klasa
};
class VoziloSTriTocka {
  // druga osnovna klasa
};
class MotociklSPrikolicom: public VoziloSTriTocka, public Motocikl {
  // izvedena klasa
};
```

Višeznačnost funkcija članova

- Višestruko nasleđivanje može da dovede do dupliranja imena članova u različitim klasama
- Rešenja su
  1. promena naziva funkcija u klasama ili
  2. upotreba operatora dosega :: za pune nazive svih objekata

Ponovljeno nasleđivanje

- Višestruko nasleđivanje može da dovede do pojave više verzija osnovne klase u izvedenoj klasi, *direktno* i *indirektno* nasleđenih
- Rešenja su
  1. promena naziva funkcija u klasama ili
  2. upotreba operatora dosega :: za pune nazive svih objekata ili
  3. upotreba *virtuelnih klasa*



Virtuelne osnovne klase

- Virtuelne klase omogućavaju izbegavanje dupliranja osnovne klase u izvedenoj klasi specifikacijom virtual ispred naziva osnovne klase, npr.

```
class Izvedena1: public virtual Osnovna {
   ...
};
class Izvedena2: public virtual Osnovna {
   ...
};
```

- Na osnovu ove specifikacije prevodilac obezbeđuje da sve klase koje direktno ili indirektno nasleđuju osnovnu klasu naslede *samo jednu instancu* osnovne klase

6.6 Konverzija tipova povezanih klasa

- Svaki objekt izvedene klase ima podobjekt osnovne klase
- Konverzija objekta izvedene klase u objekte osnovne klase je automatska, tako što se uklone specifični elementi izvedene klase. Npr. objekt tipa Kartonska kutija

```
KartonskaKutija karton(30, 40, 50, "talasasti karton");
```

može se konvertovati u objekt tipa Kutija

```
Kutija kutija;
kutija = karton;
```

  - konverzija je moguća samo u smeru generalizacije: specifična klase se konvertuje u opštiju *odbacivanjem specifičnih elemenata* objekta, dok u obrnutom smeru konverzija nije moguća
  - kod višestrukog indirektnog nasleđivanja može se pojaviti neodređenost prilikom određivnja tipa u koji objekt treba konvertovati

Literatura

1. Branović I., *Osnove objektno orijentisanog programiranja: C++*, Univerzitet Singidunum, 2013
2. Stroustrup B., *The C++ Programming Language*, 4th Ed, Addison Wesley, 2013
3. Horton I., Van Weert P., *Beginning C++ 20*, 6th Edition, Apress, 2020
4. Horton I., *Beginning C++*, Apress, 2014
5. Horton I., *Beginning Visual C++ 2013*, Wox/John Wiley&Sons, 2014
6. Horton I., *Using the C++ Standard Template Libraries*, Apress, 2015
7. Horstmann C., *Big C++*, 2nd Ed, John Wiley&Sons, 2009
8. Web 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