

OOP

Klase i objekti

3e - ssplode

OOP

Proceduralno programiranje – koristi varijable za spremanje podataka, fokusira se na procese/funkcije koje se javljaju u programu. Podaci i funkcije su odvojeni i različiti.

Objektno orijentirano programiranje – OOP je bazirano na objektima koji kapsuliraju podatke i funkcije koje manipuliraju podacima

OOP terminologija

- **Objekt** – softwerska cjelina koja objedinjuje podatke i funkcije koje manipuliraju podacima u pojedinačnoj jedinici
- **Atributi** – vrst podataka nekog objekta, uskladištene u varijablama članicama
- **Funkcije članice ili metode**: procedure, funkcije koje manipuliraju atributima neke klase

Što je klasa?

- Klasa je korisnički definiran tip podataka kojim se modeliraju objekti sličnih svojstava
- Karakteristika - uočavanje zajedničkih osobina objekta i njihovo grupiranje u klasu (apstrakcija)

Kreiranje i definiranje klase (format)

```
class Name  
{  
    izjave;  
    izjave;  
};
```

Zapaziti
tačka-
zapetu

- Klasa je opisana svojim:
 - **atributima** (podacima članovima)
 - **operacijama** (funkcijama članicama, metodama)
- Atributi najčešće predstavljaju unutrašnjost klase, tj. njenu realizaciju, a metode njen interfejs, odnosno ono što se može raditi sa njom

Primjer definicije klase

```
class Point { //definicija klase
    int x, y;      // atributi
public:          // ovde počinje interfejs
    // funkcije članice
    void setX(const int val);
    void setY(const int val);
    int getX() { return x; }
    int getY() { return y; }
};
Point ObjectName; //kreiranje objekta
```

Primjer

- Varijable članice (atributi)

```
int side;
```

- Funkcije članice

```
void setSide(int s)
```

```
{
```

```
    side = s;
```

```
}
```

```
int getSide()
```

```
{
```

```
    return side;
```

- Varijabla **side** objekta Square
- Funkcije objekta Square
setSide – određuje stranicu kvadrata
getSide – vraća stranicu kvadrata

Kontrola pristupa članovima klase

- Članovi (podaci ili metode) klase koji se nalaze ispred ključne riječi **private**: zaštićeni su od pristupa spolja (oni su sakriveni, kapsulirani)
- Da bi se dio klase učinio javnim, tj. da bi bio vidljiv u kodu izvan klase, navodi se ključna riječ **public**:
 - članovi iza ključne riječi **public**: dostupni su izvan klase i nazivaju se javnim članovima

Funkcije članice metode klase

- Zaštićenim (**private**) članovima klase obično se pristupa preko javnih (**public**) funkcija članica
- Kada **funkcija članica** koristi privatni podatak član svoje klase, pristupa mu direktno, bez korišćenja operatora .
- Funkcija članica se uvijek poziva za neki objekat svoje klase

```
char * Osoba:: koSi () {  
    return imePrezime;  
}
```

Specifikatori pristupa – public i private

- Koriste se za kontrolu pristupa članovima klase.
- Mogu biti navedeni **u bilo kojem redoslijedu** u klasi, mogu se pojaviti više puta u klasi
- Ako nisu određeni (engl. default), zadani su kao **private**

Specifikatori pristupa: Primjer

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

class Example {
    // private as default ...
public:
    // what follows is public until ...
private:
    // ... here, where we switch back to private ...
public:
    // ... and back to public.
};
```

Primjer klase 1

```
class Square {  
    private:  
        int side;  
    public:  
        void setSide(int s) {  
            side = s;  
        }  
        int getSide() {  
            return side;  
        }  
};
```



Primjer Klase 2

```
class Time {  
public:  
    void setTime(int, int, int);  
    void getTime(int&, int&, int&);  
    void printTime() const;  
    bool equalTime(const clockType&);  
private:  
    int hr;  
    int min;  
    int sec;  
};
```

Što je objekt?

- Objekt je primjer (instanca) klase
- Pošto je klasa tip, objekti se smatraju promjenjivima tog tipa u programu
- Objekti se deklariraju navođenjem imena klase iza kojeg slijede nazivi objekata razdvojeni zarezima
- Članovima klase pristupa se pomoću znaka točke (.)

Osoba profesor, student, direktor;
profesor.koSi();
direktor.koSi ():

Skup vrijednosti članova klase nekog objekta

profesor.imePrezime = "Petar Petrović"

profesor.godine = 35;

profesor.imePrezime = "Jovan Jovanović";

profesor.godine = 21;

Primjer klase, inline funkcija i objekata

```
#include <iostream>
using namespace std;
class temp {
private:
    int data1; float data2;
public:
    void int_data(int d){
        data1=d;
        cout<<"Number: "<<data1;
    }
    float float_data(){
        cout<<"\nEnter data: ";
        cin>>data2;
        return data2;
    }
}
```

```
int main(){
    temp obj1, obj2;
    obj1.int_data(12);
    cout<<"You entered "
        <<obj2.float_data();
    int k ;
    cin>>k;
    return 0;
}
```

Primjer inline (unutrašnje funkcije klase)

```
class Square
{
    private:
        int side;
    public:
        void setSide(int s)
        {
            side = s;
        }
        int getSide()
        {
            return side;
        }
};
```



Primjer 2 – inline funkcija

```
class Brojac {  
    int i;  
public:  
    Brojac (int x) { //inline konstruktor  
        i=x;  
    }  
    int broj () { //inline funkcija članica  
        return ++i;  
    }  
}
```

Konstruktor klasa

- Konstruktor klasa služi za inicijalizaciju objekata
- Konstruktor je funkcija članica koja ima isto imo kao i klasa, a nema povratni tip
- Može ali ne mora imati argumente
- Može se preklopiti, tj za istu klasu može se definirati više konstruktora koji se razlikuju po broju ili tipu argumenta

Implementacija konstruktora unutar klase

```
class Point {  
    int _x, _y;  
public:  
    Point() {  
        _x = _y = 0;  
    }  
    Point(const int x, const int y) {  
        _x = x;  
        _y = y;  
        .....  
    };
```

Implementacija konstruktora i funkcija članice van klase

```
Osoba::Osoba(char * ime, int god){  
    imePrezime =ime;  
    godine= god;  
}  
char * Osoba:: koSi () {  
    return imePrezime;  
}
```

:: - Operator dosega, povezuje ime klase s njenim članom

Konstruktori - primjer

Inline:

```
class Square
{
    ...
public:
    Square(int s)
    {
        side = s;
    }
    ...
};
```

Definicija van klase

```
Square(int); //prototip u dat u klasi
Square::Square(int s)
```

```
{           side = s;
}
```

Pojam Destruktora – svaka klasa može imati samo jedan destruktur

Kada neki objekt želimo uništiti, on se mora ukloniti iz memorije, tada koristimo destruktur koji je zadužen za oslobođanje svih resursa dodijeljenih objektu. Poziva se prije nego što se memorija zauzeta objektom oslobodi te se obavlja deinicijalizacija objekta. Destruktur mora imati isto ime kao i klasa, no ispred sebe ima znak tildu (~). Kao i konstruktor, nema povratni tip. Automatski se poziva u sljedećim situacijama:

- na kraju bloka za automatske objekte u kojem je objekt definiran
- za statičke i globalne objekte nakon izlaska iz funkcije *main ()*
- za dinamičke objekte prilikom uništenja dinamičkog objekta operatorom *delete*

Pokazni primjeri – slike sintakse

Deklaracija klase

```
1 class nazivKlase //zaglavljeklase
2 {
3
4     // tijelo klase
5
6 }
```

Slika 2.3. Sintaksa deklaracije klase

Pokazni primjeri – slike sintakse

Kreiranje objekta

Klase su predlošci prema kojima kreiramo objekte. Slika (Sl.2.4.) prikazuje sintaksu kreiranja objekta na primjeru, gdje se nazivom *Vozilo* definira klasa, a nazivom *Auto* definirao objekt koji pripada klasi *Vozilo*.

```
1 int main()
2 {
3     Vozilo Auto;
4
5     return 0;
6 }
7 }
```

Slika 2.4. Sintaksa kreiranja objekta

Pokazni primjeri – slike sintakse

Podatkovni članovi

Podatkovni i funkcijski članovi klase mogu biti statični ili ne statični, definirani su sa ili bez ključne riječi *static*. Ukoliko se varijabla klase deklarira kao statična, bez obzira koliko se objekata klase kreira, kreirati će se samo jedna kopija statičkog člana. Obrnuta je situacija kod ne statičnih deklariranih varijabli, koji se kreiraju za svaki objekt posebno. Slika (Sl.2.5.) prikazuje jednostavnu klasu sa dvije statične varijable koje se mogu koristiti za pohranu informacija o korisniku programa.

```
1 class podaciKorisnika {  
2  
3     static string Ime;  
4  
5     static int Godine;  
6  
7 };
```

Slika 2.5. Klasa pod nazivom *podaciKorisnika* sa statičkim varijablama

Pokazni primjeri – slike sintakse

Primjer klase

```
1 class podaciIgaca {  
2  
3     string Ime;  
4  
5     int Godine;  
6  
7 };
```

Slika 2.6. Klasa pod nazivom podaciKorisnika sa nestatičkim varijablama

Pokazni primjeri – slike sintakse

Funkcijski članovi

```
1 class Student{  
2  
3     private:  
4         string Ime;  
5  
6         double Test1, Test2, Test3;  
7  
8     public:  
9  
10    double Prosjek()  
11    {  
12        return (Test1 + Test2 + Test3) / 3;  
13    }  
14  
15};
```

Slika2.7. Klasa s nazivom *Student* koja sadrži podatkovne članove *Ime*, *Test1*, *Test2*, *Test3* i metodu *Prosjek*

Pokazni primjeri – slike sintakse

Prava pristupa

```
1 class pravaPristupa{  
2  
3     public:  
4         float e, f;  
5         void Funkcija1 (float brojac);  
6  
7     private:  
8         string g;  
9  
10    protected:  
11        int h;  
12        int Funkcija2();  
13  
14    };
```

Slika 2.8. Klasa s nazivom *pravaPristupa* i članovima različiti prava pristupa

```
1 class Trokut{
2
3     private:
4         int a, b;
5
6     public:
7         Trokut ()           //konstruktor sa zadanim parametrima
8     {
9         a = 0;
10        b = 0;
11    }
12    Trokut( int x, int y) //konstruktor sa parametrima
13    {
14        a = x;
15        b = y;
16    }
17    int Izracun()          //metoda koja racuna umnozak varijabli
18    {
19        return ( a * b);
20    }
21}
22
23 int main()
24 {
25
26    Trokut A;           //kreiranje objekta A sa konstruktorom bez parametara
27    Trokut B(5,4);      //kreiranje objekta B sa konstruktorom sa parametara
28    cout<<"Izracun objekta A:"<< A.Izracun() << endl;
29    cout<<"Izracun objekta B:"<< B.Izracun() << endl;
30
31    return 0;
32}
```

Rezultat izvođenja programa



Slika 2.10. Rezultat prevodenja izvornog koda

Vježbe

- Napisati kodove i izvršiti na računalu

Vježbe 1.

Pr.1

```
#include <iostream.h>
#include <math.h>
class Pravokutnik {

public:
    double visina;
    double sirina;
    double povrsina() {
        return visina * sirina;
    }
    double opseg() {
        return 2*(visina + sirina);
    }
    double dijagonalna() {
        return sqrt(pow(visina,2) + pow(sirina,2)); // zbog sqrt i pow -> include math.h
    }
};

void main() {

    Pravokutnik mali;

    mali.visina =5;
    mali.sirina = 10;

    cout<< " Površina pravokutnika je " << mali.povrsina() << endl;
    cout<< " Opseg pravokutnika je " << mali.opseg() << endl;
    cout<< " Dijagonalna pravokutnika je " << mali.dijagonalna() << endl;
}
```

Pr.2

```
#include <iostream.h>
#include <math.h>

class Krug {
public:
    double opseg(double r) {
        return 2*r*3.1415;
    }

    double povrsina(double r) {
        return pow(r,2)*3.1415;
    }
};

void main() {
    double r;
    Krug k;
    cout<<"Upiši r : ";
    cin>> r;

    cout<<"Opseg kruga je " << k.opseg(r) << endl;
    cout<<"Površina kruga je " << k.povrsina(r) << endl;
}
```

Pr. 2a

```
#include <iostream.h>

class Krug {
private:
    float r;
public:
    float povrsina();
    float opseg();
};

float Krug::povrsina() {
    return r*r*3.1415F;
}

float Krug::opseg() {
    return 2*r*3.1415F;
}
```

```
#include <iostream.h>

class Kocka {
private:
    float a;
public:
    Kocka() { // konstruktor - sluzi za inicijalizaciju var klase
        a=10;
    }
    Kocka(float aa) { // konstruktor sa parametrima koji se salju kod kreiranja klase
        a=aa;
    }
    ~Kocka() {} // destruktor - cisti memoriju koju je klasa zauzela

    float oplosje() {return 6*a*a;}
    float volumen() {return a*a*a;}
    void UpisiA(float aa) { a=aa; }

};

void main() {
    Kocka koc;
    cout<<"Oplosje kocke " <<koc.oplosje()<< endl;
    cout<<"Volumen kocke " <<koc.volumen()<< endl;

    float aa;
    cout<<"\nUpiši a: ";
    cin>> aa;
    Kocka koc2(aa);
    cout<<"Oplosje kocke " <<koc2.oplosje()<< endl;
    cout<<"Volumen kocke " <<koc2.volumen()<< endl;

    // dinamički definirane klase
    Kocka *dkoc = new Kocka();
    cout<<"Oplosje kocke " <<dkoc->oplosje()<< endl;
    cout<<"Volumen kocke " <<dkoc->volumen()<< endl;
    delete dkoc;
}
```

Pr 2.

```
#include <iostream.h>

class Kocka {
private:
    float a;
public:
    Kocka() { // konstruktor - sluzi za inicijalizaciju var klase
        a=10;
    }
    Kocka(float aa) { // konstruktor sa parametrima koji se salju kod kreiranja klase
        a=aa;
    }
    ~Kocka() {} // destruktor - cisti memoriju koju je klasa zauzela

    float oplosje() {return 6*a*a;}
    float volumen() {return a*a*a;}
    void UpisiA(float aa) { a=aa; }
};

ssploce
```

```
void main() {
    Kocka koc;
    cout<<"Oplosje kocke " <<koc.oplosje()<< endl;
    cout<<"Volumen kocke " <<koc.volumen()<< endl;

    float aa;
    cout<<"\nUpiši a: ";
    cin>> aa;
    Kocka koc2(aa);
    cout<<"Oplosje kocke " <<koc2.oplosje()<< endl;
    cout<<"Volumen kocke " <<koc2.volumen()<< endl;

    // dinamički definirane klase
    Kocka *dkoc = new Kocka();
    cout<<"Oplosje kocke " <<dkoc->oplosje()<< endl;
    cout<<"Volumen kocke " <<dkoc->volumen()<< endl;
    delete dkoc;

    // dinamički definirane klase sa konstruktorom sa parametrima

    cout<<"\nUpiši a: ";
    cin>> aa;
    Kocka *dkoc2= new Kocka(aa);
    cout<<"Oplosje kocke " <<dkoc2->oplosje()<< endl;
    cout<<"Volumen kocke " <<dkoc2->volumen()<< endl;
    delete dkoc2;
}
```