

14 Objektno orjentisano (OO) programiranje

Šta je OO programiranje?

Objektno orjentisano programiranje je metodologija programiranja kojom se modelira realan svet kao skup objekata i odnosa među objektima. Osnovni element u ovoj metodologiji su, znači, objekti koji se koriste za razvoj programa (softvera).

Pomislite o objektima koji nas okružuju – automobili, ptice, drveće, ljudi itd. Ima ih bezbroj svuda oko nas. Uobičajeno je da sve objekte koji nas okružuju klasifikujemo pa otuda i jedan od osnovnih pojmova kod OO metodologije – pojam klase (Class).

Klasa (Class)

Klasa je sastavljena od osobina objekta koje se izražavaju podacima koje ih opisuju (atributa) i akcija koje objekti mogu sprovoditi (metoda).

Uzmimo na primer automobil: podaci (atributi) koji karakterišu automobile su brzina, boja, broj sedišta, itd., a aktivnosti (metode) koje ovaj objekat može da obavlja su ubrzavanje, kočenje, promena stepena brzine (menjačem) itd.

Ili, uzmimo za primer objekte kao što su studenti. Atributi studenta su ime, prezime, broj indeksa, godina rođenja, godina studija, itd. Aktivnosti koje studenti izvršavaju su upis semestra, prijava ispita, polaganje ispita, itd.

Ili, recimo posmatrajmo automat za kafu. Kao attribute automata mozemo da uzmemo boju, listu artikala koji se nude, iznos novca u kasi, trenutnu upatu, itd. Metode kod automata mogu biti izbor artikla, ubacivanje novca, preuzimanje artikla, preuzimanje kusura, odustajanje od kupovine, itd.

Klasa opisuje sve objekte datog tipa, na taj način što definiše koji će podaci biti korišćeni za opis svakog pojedinačnog objekta. Zato je klasa apstraktna struktura bez pojedinačnih vredosti za podatke kojima se opisuju objekti.

Objekti (Object)

Objekta je jedna konkretizacija klase, odnosno objekat je jedan konkretan primerak sa konkretnim atributima koji ga razlikuju od drugih objekata iz iste klase. Na primer u klasi student, koju smo ranije opisali, može da sadrži objekat “Jovana Mirković” koja predstavlja jedan poseban slučaj objekta iz klase “Student”.

Razmena poruka (Message Passing)

Objekti ne postoje izolovano. Oni interaguju sa drugim objektima. U OO programiranju ova interakcija se ostvaruje preko poruka. Prenos poruka je proces u kome jedan objekat (pošiljalac-sender) šalje podatke drugom objektu (primaocu-receiver) ili traži od drugog objekta da aktivira neki svoj metod. Tako svaka poruka ima pošiljaoca i primaoca.

Ponašanje objekata (Behavior)

Ponašanje se odnosi na to kako objekti reaguju na poruke, to jest kako menjaju sopstveno stanje (atribute) i/ili aktiviraju svoje metode.

Pogledajmo sada kako se OOP koristi u C#.

Deklarisanje nove klase

Kada želimo da deklariramo (definišemo, opišemo) novu klasu, u C# to činimo uz pomoć sledeće sintakse:

```
class < Ime Klase >
{
    atributi...// podaci kojima se opisuju objekti iz klase

    metode...// kojima se opisuju akcije (funkcije) koje objekti izvršavaju...
}

```

Ovde je *class* ključna reč. Ime klase može biti bilo koje ime koje želimo da damo klasi, ali u skladu sa pravilima imenovanje klase

Pravila za davanje imena klasama

- Ime klase mora početi slovom iza koga može da sledi niz slova (A-Z, velika i/ili mala), cifre (0-9) i znak “donja povlaka” (_)
- Specijalna slova kao što su ? - + * /\ ! @ # \$ % ^ () [] { } , ; : . ne smeju se koristiti u imenu klase.
- Ime klase ne sme da bude neka od službenih reči C#, kao što su reči **using**, **public**, etc.

Konvencija za imenovanje klase

Sledeća konvencija nije obavezujuća, već predstavlja preporuku kako treba davati imena klasama.

- Ime klase treba da bude sa značenjem koje opisuje klasu, a najbolje je da bude neka imenica (na primer Student za klasu studenti, Automat za automat za kafu, i sl).
- Kod imenovanja uobičajene su dva slučaja: takozvani Pascal označavanje gde je prvo slovo u imenu veliko a sva ostala mala, ili Kamilje (Camel) označavanje gde je u imenu klase prvo slovo malo a samo prva slova iz svake naredne reči su velika Na primer:-
 - Pascal Case: KlasaJedan
 - Camel Case: klasaJedan

Primer deklaracije klase

```
class Student
{
    // Podaci - Atributi

    public string Ime;
    public int Starost;

    // Metode
    public void Prikazinaekranu()
    {
        Console.WriteLine("Name {0}\n Age {1}", Name, Age);
    }
}
```

Reč *public* označava da će atributi i metode koje imaju ovu reč biti dostupne i van same klase Student.

Deklarisanje i inicijalizacija objekta

Kada želimo da definišemo objekat iz neke klase, to radimo na način sličan deklarisanju varijabli, sa sledećom sintaksom:

```
<Ime klase> <Ime objekta>;
```

Primer:

Student S1;

Ali, pre nego što počnemo sa korišćenjem objekta, potrebno je da mu zadamo početne attribute (podatke), kroz proceduru koja se naziva inicijalizacija. Ta procedura uključuje dodelu memorije i druge osnovne procese za inicijalizaciju (konstrukciju) objekta. U C# koristimo reč *new* da označimo ovaj proces

Sintaksa je sledeća:-

Napomena: Konstruktori su posebne metode iz klase koje služe za inicijalizaciju objekta.

```
<Ime objekta> = new <Ime klase>;
```

Primer:

S1 = new Student();

Ova dva koraka (deklaracija i inicijalizacija) mogu da se obave u jednoj liniji koda kao što sledi:

```
Student S1 = new Student(); // Konstruktor Student()
```

Korišćenje objekata u OO programima

Za pristup atributima i metodama deklarisanog i inicijalizovanog objekta koristi se . (Dot) operator.

Na primer ako želimo da aktiviramo metod za prikaz na ekranu podataka objekta S1 iz klase student pisaćemo:

```
S1.Prikazinaekranu();
```

Atributima se može pristupati na sličan način, na primer:

```
S1.Ime = "Teofilovic Danijel";
```

promeniće ime objekta student S1.

Oslobađanje memorije

Da oslobodimo memoriju koju zauzima objekat S1 (koji nam više nije potreban) koristimo izraz:

```
S1 = null;
```

Evo kako bi sada mogao da izgleda jedan kompletan program koji koristi klasu Student.

```
using System;
```

```
namespace PrimerStudent
```

```
{  
    class Student  
    {  
        // Podaci - Atributi  
        public string Ime;  
        public int Starost;  
  
        // Metode  
        public void Prikazinaekranu()  
        {  
            Console.WriteLine("Ime i prezime {0}\n Godine starosti {1}", Ime, Starost);  
        }  
    }  
}
```

```

class Program
{
    static void Main(string[] args)
    {
        Student S1;           // Deklaracija
        S1 = new Student();   // Inicijalizacija

        Student S2 = new Student(); // Deklaracija & Instantiation

        // Davanje vrednosti atributima
        S1.Name = "Mihailo Petrovic";
        S1.Starost = 20;

        S1.Name = "Jelica Milic";
        S2.Starost = 21;

        // Aktiviranje metoda
        S1.Prikazinaekranu();
        S2.Prikazinaekranu ();
        Console.ReadLine();
    }
}

```

Prednosti korišćenja OO programiranja

- **Realistično modeliranje realnosti:** Korišćenje OO metodologije daje mogućnost boljeg i verodostojnijeg modeliranja procesa iz naše realnosti.
- **Višestruko korišćenje klasa:** Jedanput dizajnirana klasa može biti iskorišćena u raznim aplikacijama.
- **Olakšano održavanje softvera:** Korišćenjem OO metodologije značajno se olakšava održavanje softvera, jer je jednostavnija identifikacija modulča koje treba menjati.

Osnovna svojstva OO metodologije

Sada ćemo se ukratko baviti nekim osnovnim svojstvima OO metodologije.

Učaurivanje, enkapsulacija (*Encapsulation, data hiding*)

Enkapsulacija je jedan važan koncept u OO programiranju. Spoljni svet, kao je već rečeno, pristupa samo onim atributima i metodama koje su označene kao public. Programer koji koristi neku klasu ne mora da zna kako su atributi i metode u samoj klasi kodirane. Sve što je njemu potrebno je da poznaje intefejs preko kojeg pristupa metodama i atributima klase. Naravno, potrebno je da razume i značenje atributa i metoda kojima pristupa, ali ne mora da zna kako su same metode programirane. Na taj način je obezbeđeno da se greške koje se pojave u klasi otklanjaju u samoj klasi, bez potrebe za promenom onog dela programa koji tu klasu koristi.

Apstrakcija (*Abstraction*)

Apstrakcija je proces kojim se objekti iz realnog sveta modeliraju uprošćenom "slikom" napravljenom u klasi pomoću atributa i metoda. Recimo klasa automobil koju smo napred pomenuli ne opisuje sve moguće karakteristike automobola, već neke osobine zanemaruje (apstrahuje) i ne prikazuje ih kroz atribute i metode. Koji atributi i metodi će biti odabrani za prikaz u klasi zavisi od namera koje imamo u pogledu budućeg korišćenja klase u raznim aplikacijama. Pravilno odabran model realnog sveta može da pomogne da se razumeju rešenja problema koje treba rešiti kroz softversku aplikaciju.

Kompozicija (*Composition*)

Objekti mogu da interaguju na različite načine u nekom sistemu. U nekim slučajevima klase i objekti mogu biti tako međusobno povezani da svi zajedno čine kompleksan sistem. U primeru sa automobilom točkovi, paneli, motor, menjač itd. Mogu biti posmatrani kao posebne klase. Klasa automobil, u tom slučaju, predstavlja kompoziciju ovih posebnih klasa

Nasleđivanje (*Inheritance*)

Nasleđivanje Inheritance is an interesting object-oriented programming conceptomogućava da neka klasa (sub klasa) bude bazirana na drugoj klasi (super klasa) i da od nje nasledi svu funkcionalnost. Kroz dodatni kod (attribute metode) sub klasa može biti specijalizovana za posebne potrebe. Na primer, kod klase vozila (kao super klase) možemo kreirati sub klase automobili i motorcikli. Obe ove klase nasledile bi sve metode klase vozila, ali bi takođe mogle da imaju i svoje specijalizovane metode i attribute, kao što su metoda Nagnise() i atribut Ugaonaginjanja za motorcikle.

Polimorfizam (*Polymorphism*)

Polimorfizam je sposobnost objekta da menja ponašanje u zavisnosti od načina na koji se koristi. Polimorfizam u najjednostavnijem obliku se pojavljuje kao niz metoda sa istim imenom ali sa različitim parametrima. Ovaj oblik polimorfizma naziva se "overloading" (overloading), a sreće se i kod operatora . U zavisnosti od parametara koje koristimo biće pozvan onaj metod koji odgovara korišćenim parametrima. Polimorfizam ima i druge oblike o kojima ovde neće reći.

Modularnost (*Modularity*)

Ipored gore navedenih koncepata OO programiranje donosi i povećanje modularnosti programa. Individualne klase ili grupe povezanih klasa (biblioteke klasa) mogu se posmatrati kao moduli koji se mogu koristiti u raznim softverskim projektima. Na taj način se smanjuje potreba za ponovnim programiranjem sličnih zadataka, pa se tako redukuje ukupan napor potreban za razvoj sofvera.

Pitanja

1. Šta je klasa?
2. Šta je objekat?
3. Koji su osnovne komponente klase?
4. Šta je apstarkcija?
5. Šta je ućaurivanje?
6. Šta je nasleđivanje? Primer.
7. Šta je polimorfizam? Primer.
8. Šta je overloding metoda? Primer.
9. Šta je overloding operatora? Primer.
10. Koje su prednosti korišćenja OO metoda?
11. Kako se postiže modularsnot u OO programiranju?