

15 OOP i UML

Od proceduralnog ka OO modelu programiranja

Svi programski jezici podržavaju četiri sledeća koncepta:

- Sekvencu – kojom se izvršava niz instrukcija (naredbi, komandi)
- Selekciju – donošenje odluke (if..then...else, switch....case)
- Iteracije – ponavljanja, repeticije, petlje, ciklusa
- Apstrakcije – procesa kreiranja softvera kojim se omogućava parametrizacija rešenja čime se postiže da softver dobija generalan karakter

Upravo se na konceptu apstrakcije razlikuju proceduralni (kao što je C) i OO jezici (kao što su C++,C#,Java).

Apstrakcija kod proceduralnih jezika

Apstrakcija kod proceduralnih jezika se postiže korišćenjem potprograma (funkcija i procedura).

Funkcije služe za transformaciju eksternih podataka u neki finalni rezultat, kao na primer funkcija koja određuje srednju vrednost između dva broja:

```
float SrednjaVrednost(float a, float b) {  
    float result;  
    result = (a+b)/2;  
    return result;  
}
```

Na taj način se problem top-down (top-down) pristupom može podeliti u potprobleme (metoda poznata kao »podeli pa vladaj« - »divide and conquer«) koji se rešavaju kroz posebne procedure ili funkcije što se može izraziti na sledeći način:

$$F(G(x_1, x_2, \dots, x_n), H(y_1, y_2, \dots, y_m))$$

gde se funkcijom F rešava problem koji je podeljen na dva potproblema koji se rešavaju funkcijama G i H. Vrednosti x_1, x_2, \dots, x_n i y_1, y_2, \dots, y_m nazivaju se parametrima funkcija.

Prenos parametara

U jeziku C#, na primer, parametri mogu biti prenošeni na tri načina: po vrednosti, po referenci i kao izlazni parametri. Razmotrićemo samo prva dva načina (po vrednosti i po referenci)

Prenos parametara „po vrednosti“

Kod prenosa parametara kao vrednosti u potprogram (metodu) se prenosi vrednost koja se dalje u metodi koristi, ali se ne menja vrednost varijable koja je bila korišćena u pozivu potprograma (metode).

To je ilustrovano u sledećem primeru:

```
// PassingParams1.cs
using System;
class PassingValByVal
{
    static void SquareIt(int x)
    // The parameter x is passed by value.
    // Changes to x will not affect the original value of myInt.
    {
        x *= x;
        Console.WriteLine("The value inside the method: {0}", x);
    }
    public static void Main()
    {
        int myInt = 5;
        Console.WriteLine("The value before calling the method: {0}",
            myInt);
        SquareIt(myInt); // Passing myInt by value.
        Console.WriteLine("The value after calling the method: {0}",
            myInt);
    }
}
```

Output

```
The value before calling the method: 5
The value inside the method: 25
The value after calling the method: 5
```

Prenos parametara »po referenci«

Kod prenosa parametara »po referenci« u potprogram (metodu) se prenosi referenca (adresa) varijable koja se dalje u metodi koristi, pa se menja i vrednost varijable koja je bila korišćena u pozivu potprograma (metode).

To je ilustrovano u sledećem primeru:

```
// // PassingParams2.cs
using System;
class PassingValByRef
{
```

```

static void SquareIt(ref int x)
// The parameter x is passed by reference.
// Changes to x will affect the original value of myInt.
{
    x *= x;
    Console.WriteLine("The value inside the method: {0}", x);
}
public static void Main()
{
    int myInt = 5;
    Console.WriteLine("The value before calling the method: {0}",
        myInt);
    SquareIt(ref myInt); // Passing myInt by reference.
    Console.WriteLine("The value after calling the method: {0}",
        myInt);
}
}

```

Output

```

The value before calling the method: 5
The value inside the method: 25
The value after calling the method: 25

```

Apstrakcija u OO jezicima

Apstrakcija u OO jezicima se postiže kroz takozvane apstraktne tipove podataka (ADT – Abstract Data Type), koji sadrže istovremeno i podatke i operacije (metode) koji se nad njima mogu vršiti.

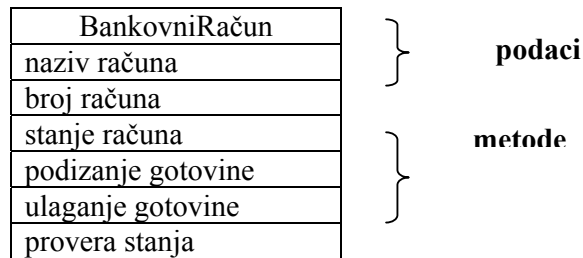
U OO jezicima apstrakcija se postiže preko programske konstrukcije koja se naziva *klasa*.

U OO terminologiji podaci se nazivaju *atributima* a operacije koje se nad njiva izvršavaju nazivaju se *metodama*.

Dakle klasa sadrži atribute i metode.

Klasa predstavlja generalizaciju (model) nekog entiteta (objekta) iz realnog sveta. Klasom se modeliraju podaci (atributi) i procedure (metode). Primeri entiteta koji se mogu modelirati klasama su: student, automobil, bankarski račun i sl.

Primer klase – Bankovni račun

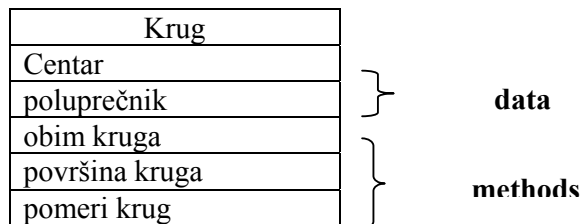


// Pseudokod za klasu Bankovni račun

```
class BankovniRacun {  
    string nazivRacuna;  
    long brojRacuna;  
    float stanjeRacuna;  
  
    podizanjeGotovine();  
    ulaganjeGotovine();  
    proveraStanja();  
} // Klasa Bankovni račun
```

Evo još jednog primera:

Klasa Krug



// Pseudokod za klasu Krug

```
class Krug {  
    double cetarX, centarY;  
    double poluprecnik;  
  
    povrsinaKrug();  
    obimKrug();  
    pomeriKrug();  
} // Klasa Krug
```

Objekat je jedan konkretan slučaj (instanca) klase – na primer Petrov račun u banci je jedan objekat tipa bankovni račun.

Objekat Jelenin račun je drugi objekat iz iste klase. Dakle objekti sadrže konkretne podatke dok klasa ne sadrži podatke. Klasa je samo okvir (templejt) za podatke – pokazuje koje će podatke objekat iz date klase imati.

Na primer Petrov račun može izgledati ovako:

nazivRacuna = Petar Petrovic

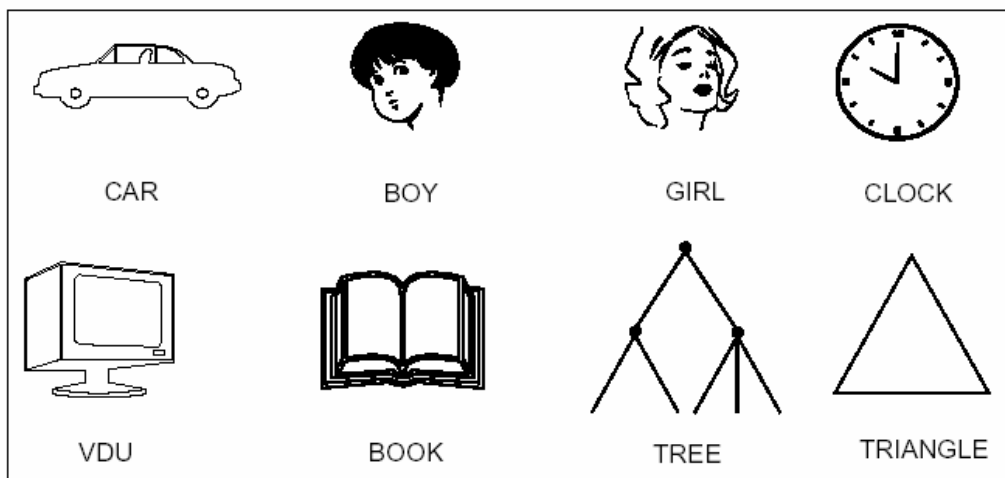
brojRacuna = 423838634543

stanjeRacuna = 150.850,00

Obično aplikacija koju želimo da razvijemo OO metodologijom sadrži veći broj objekata koji su međusobno povezani i u nekoj interakciji. Recimo, ako želimo da izgradimo informacioni sistem neke banke možemo imati sledeće klase: bankovni račun, korisnik usluga (mušterija), bankovni automat, krediti, itd.

Ili u slučaju trgovine, moguće klase su: skladišta, fakture, kupci, dobavljači, itd.

Objektno orjentisani sistem je skup međusobno interagujućih objekata kao što ilustruje sledeća slika.



Primer objekata

Klase su zapravo objekti sa istim atributima i ponašanjem

Na sledećoj slici su muskarci i žene predstavljeni jedinstvenom klasom osobe, putnički i teretni automobil klasom automobili, a trouglovi, šestouganci i rombovi klasom poligoni.

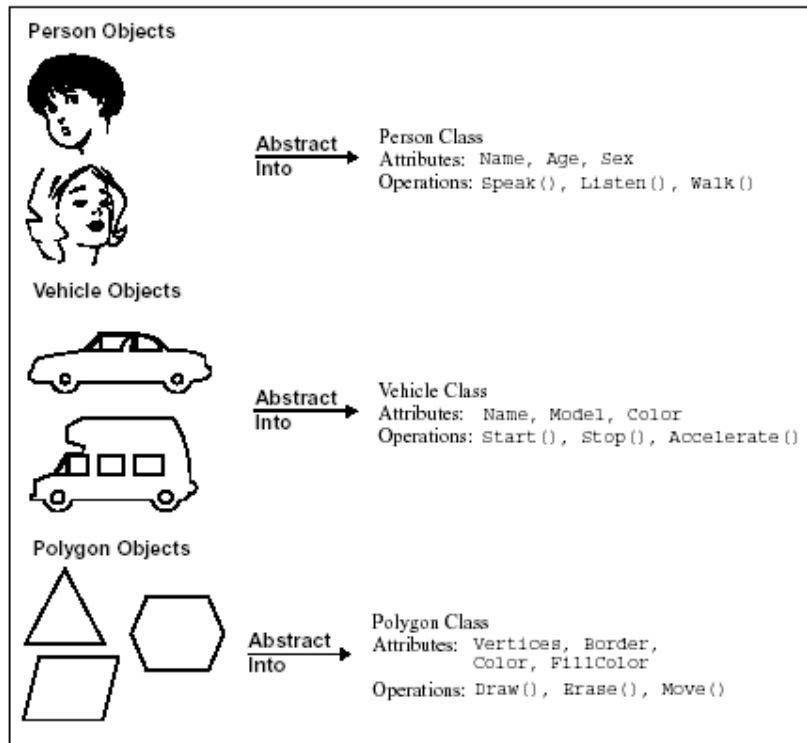


Figure 1.12: Objects and classes

Uvod u UML

Kao što smo kod proceduralnih metoda koristili grafičke simbole za prikazivanje algoritama, tako se i OO metodologiji nametnuo jedan standardni grafički način za prikazivanje klasa.

UML (Unified Modeling Language) je grafički jezik za modeliranje koji se koristi za prikaz različitih OO komponenti pri analizi, projektovanju i implementaciji OO sistema. UML je postao međunarodni standard. Njegovi autori su Grady Booch, Jim Rumbaugh and Ivar Jacobson.

Kratak istorijat razvoja UML-a

Istorijat razvoja UML je interesantan i sa aspekta ubrzanja širenja metodologija, a ne samo novih tehnoloških rešenja.

1994 – Grady Booch i Jim Rumbaugh su osnovali kompaniju Rational Rose sa novom idejama:

Grady Booch – sa Booch Metodom

Jim Rumbaugh – Tehnikom Objektnog Modeliranja

1995 – Ivar Jacobson se pridružio timu sa idejom:

Objekt Orjentsanog Softverskog Inženjerstva

Tako je nastala prva verzija UML – 0.8

1995 Object Management Group (OMG) – proglasila je UML standardom.

1996 – U proces su se uključile i druge kompanije.

Tekuća verzija je UML 2.0

Skraćeni opis UML-u i dobar tutorial možete pronaći na adresi

<http://www.smartdraw.com/tutorials/software-uml/uml.htm>

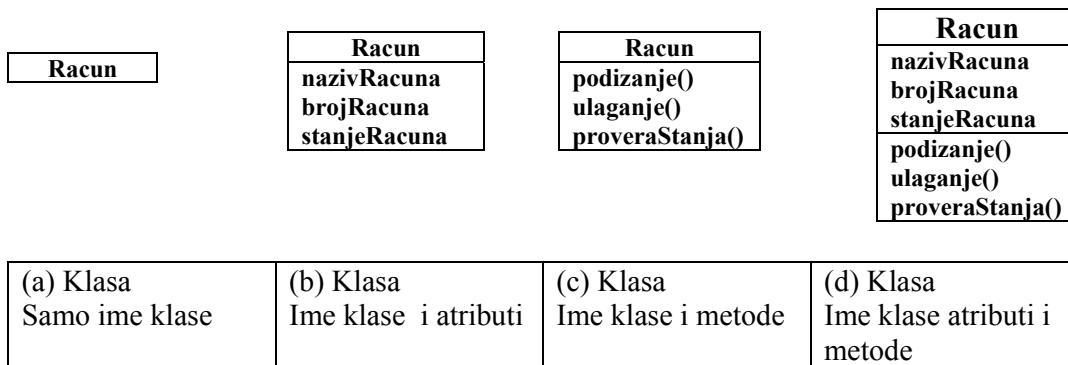
a detalje na: <http://www.uml.org>

Predstavljanje klasa pomoću UML-a

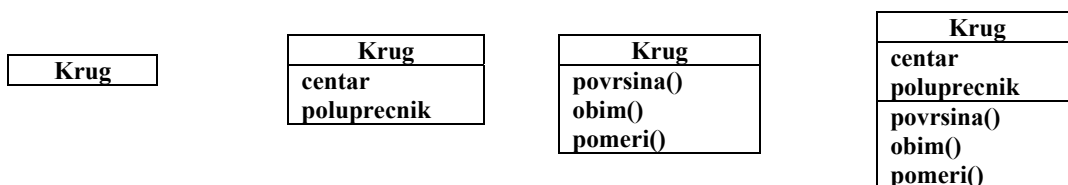
Klase se mogu predstaviti na jedan od sledećih načina u zavisnosti od nivoa detalja koji se želi prikazati.

Posmatrajmo ponovo naš primer bankovnog računa. Radi skracenog pisanja malo smo promenili nazive podataka i metoda.

Klasa Racun



Klasa Krug



Još o karakteristikama klasa

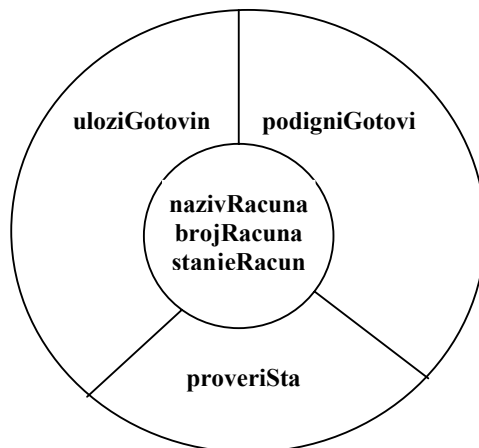
Učaurivanje (Encapsulation)

Sve informacije (atributi i metode) u OO sistemu su smeštene (sakrivene) unutar same klase (objekta).

Ovim informacija se može pristupiti samo preko metoda iz klase koje su stavljene na korišćenje spoljnjem svetu kao interfejs. Sama implementacija je sakrivena od korisnika.

Mogućnost da se kod klasa (objekata) od korisnika mogu sakriti detalji naziva se učaurivanjem (enkapsulacijom). Kao da je objekat (klasa) zatvoren u kapsulu, ne može se prići njegovim unutrašnjim delovima, već se objekat koristi samo pomoću javnih metoda koje jedino imaju "pravo" da pristupaju i menjaju podatke (attribute) objekta.

Primer učaurivanja



Samo metode `uloziGotovinu()`, `podigniGotovinu()` i `proveriStanje()` mogu pristupiti i modifikovati podatke `nazivRacuna`, `brojRacuna` i `stanjeRacuna`.

Pogledajmo sada kako pseudokod i UML dijagram međusobno korespondiraju.

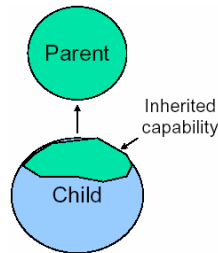
```
class Racun {  
    private String nazivRacuna;  
    private brojRacuna;  
    private float stanjeRacuna;  
  
    public podigniGotovinu();  
    public uloziGotovinu();  
    public proveriStanje();  
} // Klasa Racun
```

Racun
nazivRacuna brojRacuna stanjeRacuna
podigniGotovinu() uloziGotovinu() proveriStanje()

Nasleđivanje (Inheritance)

Nove klase mogu biti kreirane korišćenjem postojećih klasa. Tako se stvara odnos Roditelj-Dete (Parent-Child), gde roditelj predstavlja postojeću klasu (super klasu) a dete novu (izvedenu) klasu (pod klasu).

Podklasa (dete) nasleđuje svojstva od superklase (roditelja), što ilustruje sledeća slika:



Ponovo ćemo iskoristiti primer bankovnog računa da ilustrujemo nasleđivanje

Klasa Racun – je klasa roditelj (superklasa) koja ima sledeće atribute:
nazivRacuna, brojRacuna i stanjeRacuna

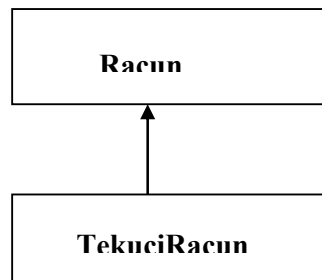
No, moguće je imati različite vrste računa u bankama – tekuće, žiro, itd.

Tako recimo, klasa TekuciRacun je nova klasa – dete (podklasa) koja pored gore navedenih atributa iz klase Racun (superklase) može imati i dodatne atribute:
brojIzdatihCekova i brojRealizovanihCekova

I sada, umesto da kreiramo novu klasu TekuciRacuni koja bi imala svih pet atributa, možemo da klasu TekuciRacuni definišemo kao podklasu klase Racun koja nasleđuje atribute nazivRacuna, brojRacuna i stanjeRacuna, kao i sve metode klase Racun.

Prikaz nasleđivanja UML dijagramima

TekuciRacun je podklasa superklase Racun. To se UML dijagramom iskazuje na sledeći način:



Drugim rečima, klasa TekuciRacun je izvedena iz klase Racun.

Nasleđivanje –Primer 2

Klasa Krug

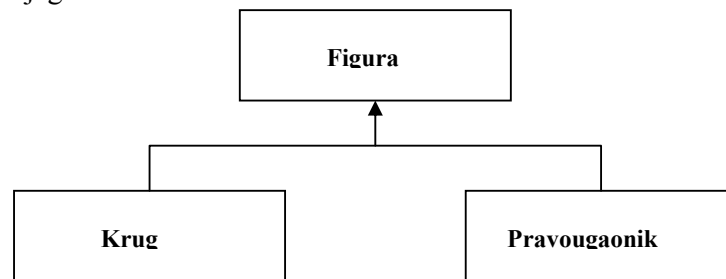
Klasa krug u našem primeru imala je atribute centar i poluprecnik.

Posmatrajmo sada klasu Pravougaonik. Ta klasa može imati atribut centar, ali poluprecnik nije prikladan atribut za ovu klasu.

Pravougaonik treba da ima atribute dužina i širina, ali oni nisu prikladni za krug. Metode površina, obim i pomeranje mogu biti primenjene i na krug i na pravougaonik (doduše sa različitim načinom izračunavanja površine i obima, ali o tome će biti reči kasnije).

Ali umesto da definišemo novu klasu Pravougaonik nezavisnu od klase Krug, mi ćemo definisati novu klasu Figura, tako što ćemo sve zajedničke atribute i metode kruga i pravougaonika smestiti u novu klasu. Onda ćemo Krug i Pravougaonik definisati kao podklase klase Figura.

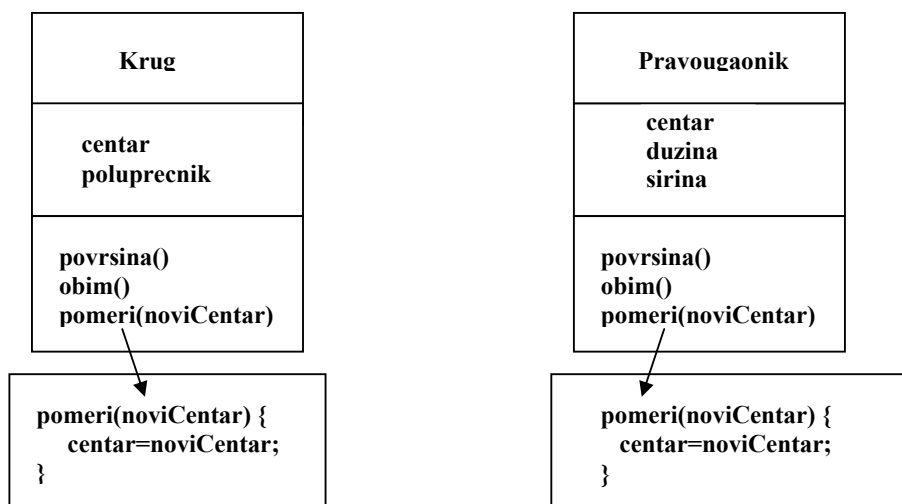
Prikažimo to UML dijagramom:



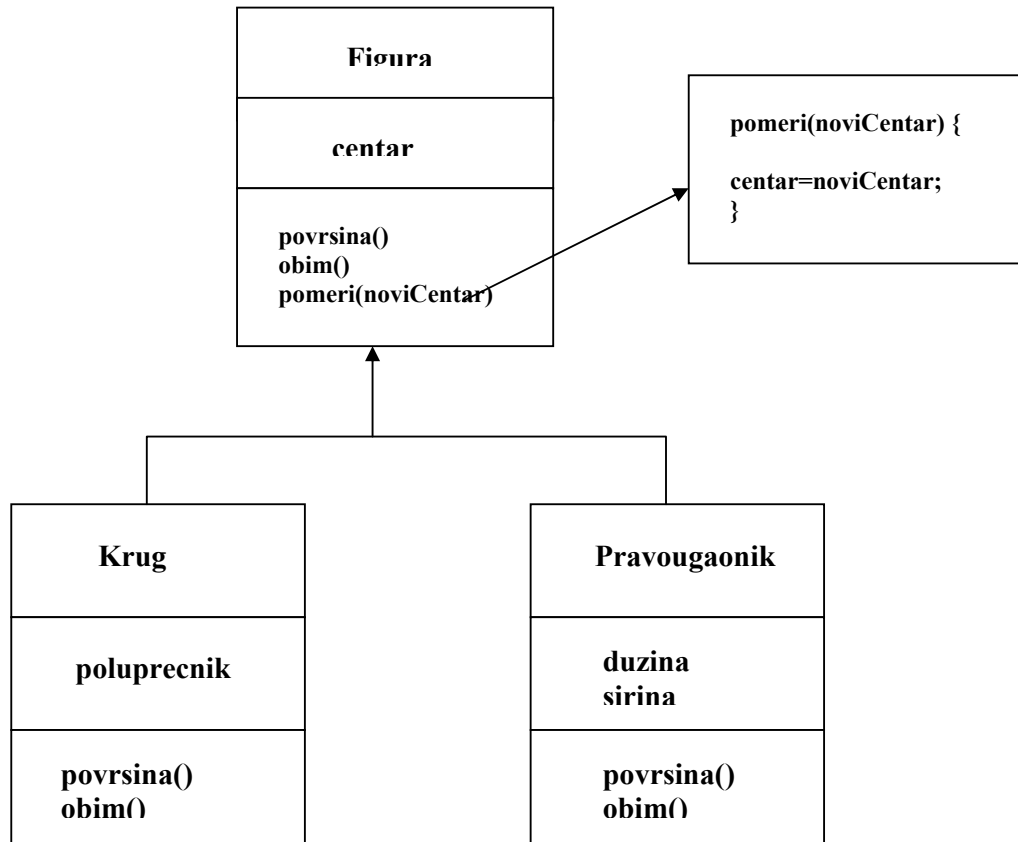
Ponovno korišćenje nasleđivanja - Reuse

Ako više klasa imaju zajedničke atribute i metode onda takve klase možemo sakupiti u jednu zajedničku klasu – superklasu (klasu roditelj). Time se postiže brži razvoj aplikacija jer se klase ne moraju nanovo programirati već se koriste već postojeće koje mogu biti sakupljene iz različitih izvora.

Primer: Pravougaonik i Krug imaju zajednički metod pomeranje jer je tada potrebno samo promeniti koordinate centra (to je atribut koji imaju i krug i pravougaonik).



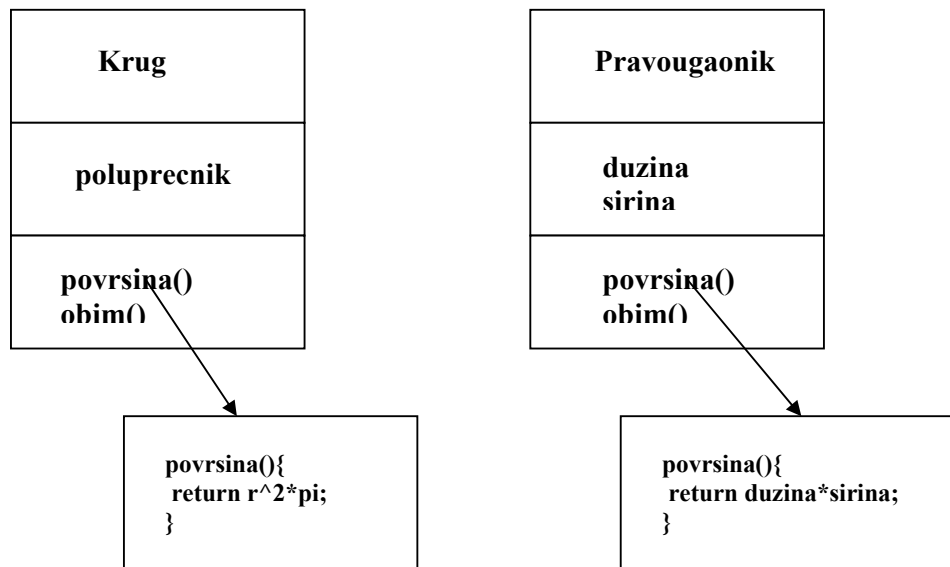
Sada bi mogli da definišemo novu klasu Figura koja će sadržati zajedničke atribute i zajedničke metode za klase Krug i Pravougaonik, što je ilustrovano na sledećoj slici:



Korišćenje nasleđivanja za specijalizacija

Subklasi (detetu) tokom nasleđivanja mogu biti pridodata nova svojstva. Taj postupak se naziva specijalizacijom klase.

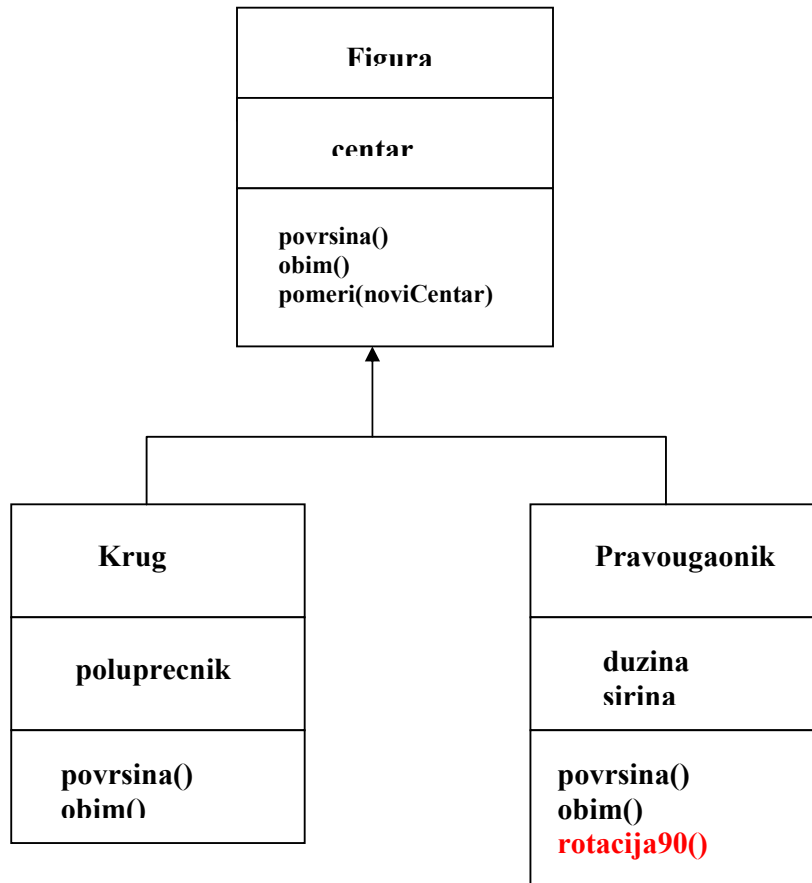
Na primer, metod **povrsina** je različit kod kruga i pravougaonika. Isto važi i za **obim**. Zato se površina i obim ponovo definišu (specijalizuju) u izvedenim klasama - **Krug** i **Pravougaonik**, kako je ilustrovano na sledećoj slici:



Korišćenje klase za proširenje funkcionalnosti postojeće klase

Klasa dete može imati i posebne metode koje nema klasa roditelj.

Na primer pravougaoniku možemo dodati novi metod koji kod kruga može biti besmislen – rotaciju za 90 stepeni, kako je to ilustrovano na sledećoj slici:



Dakle u ovom slučaju samo je dodata još jedna metoda klasi Pravougaonik, ali se ništa drugo nije promenilo.

Polimorfizam (Polymorphism)

Polimorfizam znači “više oblika”

U OOP polimorfizam i sam ima više značenja.

Polimorfizam omogućava da objekti, metode, operatori imaju različita značenja zavisno od načina kako im se prenose parametri.

Na primer u sledećem pseudokodu:

```
krugA = new Krug(); // Kreiranje novog objekta iz klase Krug
Figura figura = krugA; // kreiranje objekta figura iz klase figura, podklase Krug
Figura.povrsina(); // površina će biti izračunata prema metodi za krug

pravougaonikA = new Pravougaonik(); // Kreiranje novog objekta pravougaonika
figura= pravougaonikA;
Figura.povrsina(); // površina za metod pravougaonik će se koristiti
```

Overloading metoda(Method Overloading)

Jedan drugi slučaj korišćenja polimorfizma je overloading metoda, u kojem se istim imenom definiše više različitih metoda. Metode se međusobno razlikuju samo po parametrima koji im se pri pozivu prenose.

na primer neka imamo metodu inicijalizacija sa dve varijante prenosa parametara:

```
Metod 1 - inicijalizacija(int a)
Metod 2 - inicijalizacija(int a, int b)
```

Kada u programu pozivamo metodu, od toga koje parametre stavimo u pozivu zavisice koja od metoda (1 ili 2) će biti zapravo pozvan.

```
inicijalizacija(2) // Biće pozvana metoda 1
inicijalizacija(2,4) // Biće pozvana metoda 2
```

Overloading operatora (Operator Overloading)

Overloading-om operatora omogućava se davanje novog značenja operatorima kao što su +, -, *, /.

Na primer operator. + operator for Krug može biti definisan pa je izraz tipa

```
Krug c = c + 2;
```

postaje legitiman.

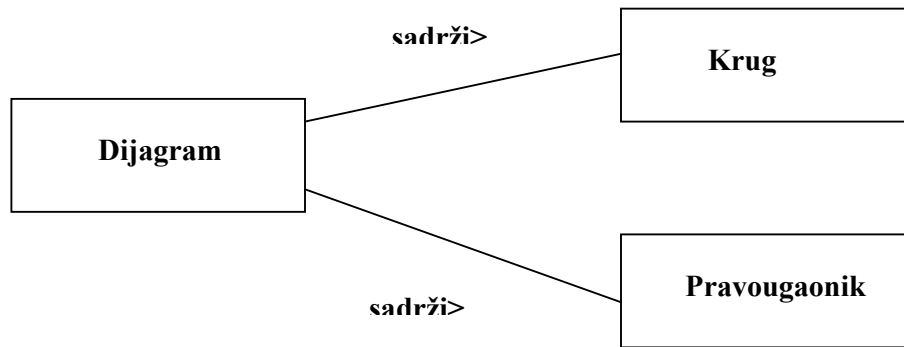
Ovo nije podržano u Java i C++ jezicima.

Asocijacija (Association)

Neka klasa može biti povezana sa drugim klasama pa je potrebno omogućiti komunikaciju među objektima iz različitih klasa.

U svhu povezivanja (asocijacije) klasa koriste se takozvani dijagrami klasa koji pokazuju međusobnu povezanost klasa i tip veze među njima.

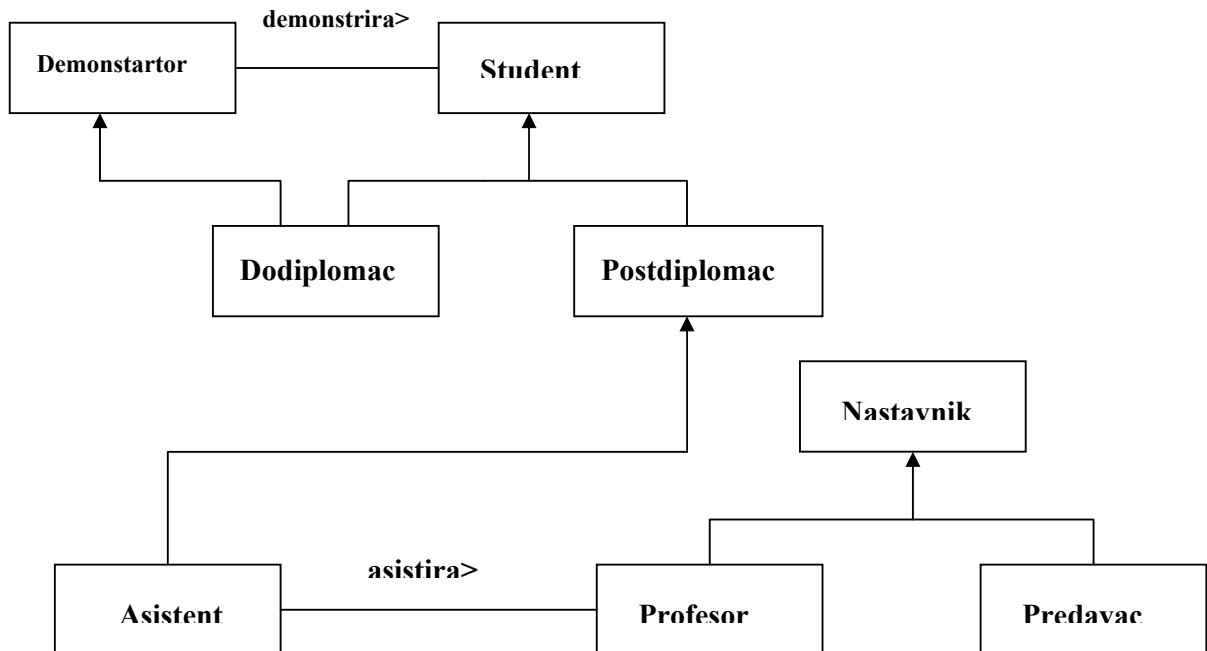
Na primer, Krug i Pravougaonik mogu biti deo klase Dijagram. Ovim se uspostavlja veza između klase Dijagram i klasa Krug i Pravougaonik kao što prikazuje sledeća alika:



Primer dijagrama klase

Nacrtati dijagram klase prema sledećem opisu:

Student može biti dodiplomac ili postdiplomac.
 Dodiplomac može biti demonstrator. Demonstrator vrši demonstracije studentima.
 Predavač i profesor su dva tipa nastavnika.
 Asistent je postdiplomac koji pomaže profesoru.



Pitanja

1. Koja su četiri osnovna koncepta programiranja?
2. Kako se princip apstrakcija izražava u proceduralnim jezicima?
3. Koji načini prenosa parametara se najčešće primenjuju u OO jezicima?
4. Koja je razlika između prenosa po vrednosti i prenosa po referenci?
5. Šta je UML? Primer.
6. Šta su dijagrami klase? Primer.