Objektno orijentisana paradigma

(sa implementacijom u programskom jeziku C#)

Objektno-orijentisana paradigma je bazirana na konceptu objekta koji se opisuje svojstvima i svojim ponašanjem. Svojstva objekata jesu strukture podataka koje se čuvaju u promenljivama i nazivaju se atributi, dok se ponašanje opisuje metodama, tj. procedurama za obradu podataka.

Proceduralna paradigma je uvela nekoliko dobrih apstraktnih koncepata kao što su potprogrami (apstrakcija kontrole) i strukture podataka (apstrakcija podataka), ali se vremenom pokazalo da to nije dovoljno i da su neophodne dodatne apstrakcije. Ovde će biti spomenute samo neke osnovne ideje OOP-a .

Klasa predstavlja strukturu podataka koju treba posmatrati kao **novi tip**.

Objekat je instanca klase i definiše se kao entitet koji je sposoban da čuva svoja stanja i koji okolini stavlja na raspolagaje skup operacija preko kojih se tim stanjima pristupa.

Objekat karakteriše njegov:
- **IDENTITET** - razlikovanje objekata među sobom,
-**PONAŠANJE** - dato preko **skupa metoda** koje sadrži objekat
-**STANJE** - varijable (promenljive) vezane za objekat

*// **stanja** odnose na **varijable**, a **ponašanje** na **metode***

Direktan pristup podacima je nepotreban i nepoželjan !!!
-podatke treba "začauriti" (enkapsulirati)!

Postoje dva bitna aspekta **enkapsulacije**:
1. objedinjavanje podataka i funkcija u jedinstven entitet (**preko klasa**) ,
2. kontrola mogućnosti pristupa članovima entiteta (**preko modifikatora pristupa**).

Enkapsulacija omogućava kontrolu korišćenja objekta (objekat se može koristiti isključivo preko javnih metoda/svojstava).

Instanciranje klasa

- vrši se pomoću opreatora **NEW**:

Pravougaonik alfa = new Pravougaonik();

Konstruktor je specijalna metoda koja:
- ima isti naziv kao i naziv klase,
- nema nikakvu povratnu vrednost.

```
public class Pravougaonik
{
        private float duzina;
        private float sirina;
        public Pravougaonik() // podrazumevani konstruktor
        {
                duzina = 0;
                sirina = 0;
```

```
        }
// konstruktor sa parametrima
        public Pravougaonik( float duz, float sir)
        {
                duzina = duz;
                sirina = sir;
```

Operator new vrši alokaciju memorije za objekat klase Pravougaonik, a metoda (konstruktor) Pravougaonik () kreira objekat.

Šta je ključna reč **this**?

- upotrebom ključne reči **this**, implicitno se pokazuje na tekući objekat/parametar

(kod statičkih metoda se ne prosleđuje ovaj dodatni parametar, jer se ne pozivaju pomoću objekata)

Nasleđivanje

Nasleđivanje predstavlja jedan od koncepata Objekto Orijentisanog Programiranja (OOP).

-omogućava da se na osnovu postojeće klase izvede nova klasa.

Izvedena klasa nasleđuje sve članove bazne klase.

// bazna klasa je super klasa ili nadređena klasa.

// izvedena klasa je podklasa ili podređena klasa (klasa koja nasleđuje baznu klasu).

Treba razlikovati nasleđivanje klasa od nasleđivanja interfejsa. Klasa koja nasleđuje interfejs treba da implementira sve navedene metode interfejsa.

- Nasleđivanje klasa može biti DIREKTNO I INDEREKTNO.

Indirektno, svaka klasa nasleđuje klasu Object.

- Jednu baznu klasu može da nasledi više izvedenih klasa (broj izvedenih klasa je neograničen).

- U programskim jezicima, nasleđivanje može biti i jednostruko i višestruko

-***C# podržava samo jednostruko nasleđivanje !***

Grupa klasa koje su povezane nasleđivanjem formiraju strukturu koja se naziva hijerarhija klasa.

– klase na višim nivoima su opštije (koncept generalizacije), dok su one na nižim nivoima u hijerarhiji specifičnije (koncept specijalizacije).

Dubina hijerarhije predstavlja broj nivoa nasleđivanja.

Primer: Roditelj-dete, pristaniste-brod-sidro

Polimorfizam

Pošto pričamo o nasleđivanju, treba predstaviti i koncept polimorfizma.

–polimorfizam predstavlja sposobnost promenljive da referencira objekte različitih tipova i da automatski poziva odgovarajuću metodu objekta koji referencira.

// Polimorfizam se zasniva na ideji da metoda koja je deklarisana u osnovnoj klasi može da se implementira na više različitih načina u različitim izvedenim klasama.

Polimorfizam se realizuje preko virtualnih metoda. Za deklaraciju virtuelnih metoda se koristi ključna reč virtual.

- virtualne metode se mogu reimplementirati u izvedenim klasama (kod njih se koristi ključna reč override).

Virtuelna i reimplementirana (overrajdovana Mr. Green) metoda moraju biti identične, tj. moraju imati:

isti naziv,isti modifikator pristupa,isti tip rezultata,iste tipove parametara.

Ukoliko ne želimo da drugi programeri, nasleđjuju neku od klasa, koje smo napravili to možemo učiniti pomoću ključne reči **sealed** , a takve klase se nazivaju ***zapečaćene klase.***

Zapečaćena (sealed) klasa ne može da se koristi kao bazna klasa bilo koje druge klase.

Interfejsi

Interfejs predstavlja "ugovor" kojim se garantuje da će se klasa, koja je nasledila taj interfejs, ponašati na određeni način.

Dakle, klasa garantuje da podržava metode, svojstva (properties), događaje (events) i indeksere nekog interfejsa!

Sintaksno:

-interfejs je klasa koja sadrži samo apstraktne metode.

Kada klasa implementira interfejs ona mora da implementira sve njene metode !

Moguće je da jedna klasa implementira više interfejsa.

(podsetimo se, višestruko nasleđivanje klasa nije dozvoljeno u C#, ali jeste višestruko nasleđivanje interfejsa).

Ukoliko želimo da proverimo da li objekat implementira interfejs učinićemo to na jedan od sledeća dva načina:

A) preko IS operatora:

npr. (kocka is ITelo)

-vraća true ako se kocka može kastovati u ITelo.

B) preko AS operatora:

npr. (ITelo telo = kocka as ITelo)

- ovaj način kombinuje IS i operacije kastovanja tako što prvo testira da li je kastovanje moguće a zatim kompletira kastovanje ako je moguće. Ukoliko kastovanje nije moguće operator AS vraća povratnu vrednost null.

//dakle, u ovom slučaju se provera vrši na indirektan način!

Apstraktna klasa je klasa koja se može naslediti, ali se ne može instacirati.

Može posedovati apstraktne metode, koje nemaju svoju implementaciju, ali ih klase naslednice (ukoliko nisu apstraktne) moraju implementirati korišćenjem ključne reči “override”.

Takođe, apstraktna klasa može imati metode koje imaju implementaciju (za razliku od interfejsa), i koje u svom telu mogu koristiti apstraktne metode.

U osnovi, interface i apstraktna klasa su slični, ali ipak se koriste na različite načine i za različite namene, te se mogu izvući određene smernice kada koristiti jedno a kada drugo.

Ukoliko naše klase nasleđuju apstraktnu klasu, u mogućnosti smo da na jednostavan način imamo ponovnu upotrebu koda (code-reuse). Određene funkcionalnosti možemo imati deljenje unutar apstraktne klase i u mogućnosti smo pružiti podrazumevanu (default) implementaciju.

S druge strane, interfejsi popunjavaju nedostatak korišcenja višestrukog nasljeđivanja u nekim jezicima – klasa može implementirati koliko je potrebno interfejsa, a naslediti samo jednu klasu. Ovde nećemo uzimati u obzir mogućnost korišcenja višestrukog nasleđivanja, pošto je prisutna tendencija onemogućavanja istog u novijim jezicima, a poznati su problemi korištenja višestrukog nasljeđivanja u jezicima koji ga podržavaju – stoga se ono ionako ne preporučuje.

Interfejs ne pruža mogućnost korišcenja zajedničke baze koda – on je jednostavno ugovor koji propisuje šta klase koje ga implementiraju moraju podržati, ali na svakoj klasi ostaje da za sebe implementira datu funkcionalnost. Zbog toga, može izgledati praktičnije kada god imamo potrebu za deljenjem koda, koristiti apstraktnu klasu, ali to nije tako. Nasleđivanje, ako se koristi neispravno, može rezultirati vrlo lošim dizajnom.

Code-reuse nije dovoljan razlog za preferiranje apstraktne klase nad interfejsom.

S druge strane, korišcenje interfejsa uvodi dodatni problem. Kako isti ne podržavaju mogućnost podrazumevane implementacije, svaka promena interfejsa, zahteva promenu postojeće implementacije. Ako dodamo novi metod na interfejs, sve klase koje su izvedene iz tog interfejsa moraju implementirati taj metod. To može biti poseban problem kod frameworka. Recimo da imate klijente koji koriste vaš framework, i implementiraju određeni interfejs(e). Oni moraju izmeniti/kompajlirati/redeployati sve klase koje su izvedene iz tog interfejsa kad god dodate novi metod na interfejs. Kod apstraktnih klasa to možete izbeći praveći virtuelni metod i pružajući podrazumevanu implementaciju. Iz svega ovoga može se zaključiti da apstraktne klase i interfejsi, ponaosob, imaju svoje prednosti i nedostake u određenim okolnostima. Interfejsi imaju svoju namenu, ali ne mogu izbaciti potrebu za nasleđivanjem, kao osnovnim konceptom OOP-a.

Kada onda koristiti apstraktnu klasu, a kada interfejs?

Ako između dve apstrakcije imamo jasnu i čistu klasifikacionu “je” (is a) vezu, odnosno, ako možemo reći da “B je A”, onda B treba da nasleđuje iz A, i A je kandidat za apstraktnu baznu klasu. Primer: pas je sisar.

Ako pak, želimo označiti da neki objekat ima određenu mogućnost (ability), koristićemo interfejs, kao atributsku “je” vezu. Pri tome, više konceptualno nepovezanih klasa može koristiti isti interfejs. Primer, ptica može leteti i avion može leteti. Ni u kom slučaju oni ne trebaju biti izvedeni iz neke zajedničke klase. U ovom slučaju, kandidat za interfejs je mogućnost letenja. Označava zajedničku osobinu među pomenutim klasama, iako su klase međusobno nepovezane (ptica je životinja, avion je vozilo). Ove klase mogu imati i proizvoljan broj drugih interfejsa, a vrlo vjerovatno će imati kao bazu – apstraktnu klasu životinja i vozila. Bitno je da će klijent moći posmatrati ove obe klase kao “Flyable” i polimorfno pozivati na njima metod Fly(), a pri tome ga ne treba zanimati da li se radi o ptici ili avionu. Svaka od njih ponaosob će implementirati svoj način letenja. U mnogo slučajeva za implementaciju je uputno koristiti kompoziciju, posebno ako postoje zajednički implementacijski detalji.

Sada cemo uraditi nekoliko primera:

Zadatak oblici

1.Kreirati klasu Oblik sa atributima

− Centar oblika

− Boja oblika

i metodama

− void transliraji(int dx, int dy) za translaciju oblika za vrednost dx i dy

− void pomerCentari(int a, int b) za postavljanje centra oblika u tacku (a,b)

− void nacrtajOblik(Graphics g)

− float povrsina()

− float obim()

− bool pripada(int x, int y)

2. Kreirati klase koje nasledjuju Oblik:

− Krug, sa atributom poluprecnik i override metodama nacrtaj, povrsina, obim.

− JednakostranicniTrougao, sa atributom duzinaStranice i override metodama nacrtaj, povrsina, obim. (jedna stranica trougla je paralelna horizontalnoj ivici Graphics polja)

− Pravougaonik sa atributima duzine stranica a i b, override metodama nacrtaj, povrsina, obim. (stranice paralelene ivicama Graphics polja)

3. Kreirati klasu Kvadrat koja nasledjuju klasu Pravougaonik

Klasa antene

1. Napisati apstraktnu klasu Antena koja sadrži podatke o anteni:

dva cela broja (p,q) kojima je definisana poziciju antene i dva cela broja a,b čije značenje zavisi od vrste antene.

Obezbediti:

- konstruktor, konstruktor kopije

- metod koji vraca oznaku antene ('e'- elipticna, 'l'- linearna, 'p' - pravougaona)

- metod bool uDomenu(int x,int y) kojim se proverava da li je tačka (x,y) u domenu antene

- metod int oblast() kojim se određuje broj tačaka koje su u domenu antene

- metod za crtanje antene u datoj grafici olovkom date debljine

- metod ToString (oznaka antene (p,q) a b)

2. Napisati klase EAntena, PAntena, LAntena kao izvedene klase iz klase Antena.

EAntena (p,q,a, b) pokriva sve tačke elipse tj. sve tačke (x,y) za koje je

(x-p)2/a2+(y-q)2/b2<=1

PAntena (p,q,a,b) pokriva sve tačke koje su u unutrašnjosti pravougaonika stranica 2*a i 2*b čiji je centar (p,q) tj. sve tačke (x,y) za koje je

Ix-pI<=aIy-qI<=b

LAntena(p,q,a,b) pokriva sve tačke koje se nalaze na duži čija je sredina tačka(p,q) a jedan kraj tačka (a,b) tj. sve tačke (x,y) koje pripadaj pravoj

(y-q)*(a-p)=(x-p)*(b-q)

3. Napisati klasu SkupAntena za rad sa skupom antena na nekom području.

Obezbediti konstruktor na osnovu broja antena, svojstvo za vraćanje broja antena, za vraćanje antene datog rednog broja. Obezediti metod za dodavanje antene u skup, vraćanje antene koja ima najviše tačaka u domenu, određivanje skupa antena u čijim domenima je data tačka (x,y), crtanje u datoj grafici.

Malo slozeniji primer koji obuhvata sve:

Projekat “Samouslužni automat“

Nije lako naći lep primer na kome bi se mogli ilustrovati pojmovi objektno orjentisanog programiranja. Neke oblasti nisu poznate učenicima, pa bi uvođenje u tu oblast odnelo previše vremena pre projektovanja. Samouslužni automat je stvar sa kojom se većina susretala. Možemo se poslužiti slikom ili crtežom, da bismo konkretizovali i olakšali razmišljanje.

U pripremnoj fazi (1-2 časa) možemo se baviti Apstrakcijom. Automat ima mnogo osobina: dimenzije, težina, potrošnja energije, broj polica, broj pozicija po polici, količina novca,... Neke od njih su nam bitne i pomoću njih možemo da opišemo stanje automata. U saradnji sa učenicima možemo ih nabrojati:

Broj polica (u prikazanom primeru: 5 polica)

Broj pozicija na polici (u prikazanom primeru: 6)

Vrstu artikala na pozicijama (voda Rosa, Coca cola, Čips, ...)

Količinu artikala na poziciji (ne vidi se na crtežu, ali je bitna osobina)

Količina novca u automatu (da bi se vratio kusur)

Kasnije se možemo vratiti i detaljnije grupisati osobine, već prema potrebama.

Automat ima neko ponašanje i njega možemo opisati akcijama koje automat može da izvede. Najlakše ih je nabrojati, ako napravimo scenario dešavanja:

1. Ubacimo novac
2. Izaberemo artikal i uzmemo ga
3. Uzmemo kusur

Daljom analizom ustanovićemo da postoje neke „nevidljive“ radnje bez kojih čitava priča ne bi mogla da se realizuje:

1. Vraćanje kusura nije jednostavna operacija, jer se pojavljuje problem (vrlo realan i sa njime se svakog dana susrećemo u prodavnici) postojanja odgovarajućeg broja potrebnih novčanica. Zato se prva i treća tačka moraju vezati za novčanice u odgovarajućim apoenima.

2. Artikal možemo uzeti samo ako ga ima na poziciju koju smo izabrali. Znači, neko mora automat da napuni, pre korišćenja.

Artikal ima neke svoje bitne i nebitne osobine. Mi ćemo izabrati koje su nama bitne:

a. Naziv (Čips, Rosa, ...)

b. Pakovanje (50 g, 0,5l, ,03l, ...)

c. Rok trajanja!

d. Cena (Kako bismo kasnije ilustrovali da geteri ne moraju samo da vrate vrednost atributa, možemo uvesti popust koji važi u nekim uslovima – nekog dana

u nedelji, na primer. Takođe, popust možemo definisati na nivou artikla, police ili celog automata)

Sada već možemo da uvodimo OOP termine:

Automat je klasa koja ima svoje stanje (opisano atributima) i ponašanje (opisano metodama).

Artikal je, takođe klasa. Kasa (deo koji prima novčanice i vraća kusur), takođe može da se izdvoji u posebnu klasu i da nam koristi u drugim projektima: apoteka, biletarnica, ...

Ono što stalno treba imati na umu: objekat valja praviti tako da bude autonomna celina koja sa ostalima komunicira preko poruka (pozivi metoda).

Tako ćemo postići da jednu klasu možemo koristiti u više projekata.

Ovako izgleda kod za projekat automat:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections;

namespace ConsoleApplication7
{
   public class artikal
   {
```

```
    public string naziv;
    public DateTime rok_trajanja;
    public int cena;
    public string index;
    public int popust;
    public int dan_popusta;
    int Datum = (int)DateTime.Now.DayOfWeek;

    public int get_cena() {

        if (Datum==0) {Datum=7;}
        if (Datum == this.dan_popusta) { return cena * (100 - popust) / 100; }
        else { }
        return cena;
    }

    public artikal(string n, DateTime d, int c, string i, int _popust, int
_danpopusta) {
        naziv = n;
        rok_trajanja = d;
        cena = c;
        index = i;
        popust = _popust;
        dan_popusta = _danpopusta;
    }
```

```
    }

    public class Automat
    {
        public int credit=0;
        Queue[] polica = new Queue[5];




        public void dodaj_artikl(artikal roba, int gde)
        {
            polica[gde].Enqueue(roba);
        }

        public void prikazi() {
            for (int i = 0; i < 5; i++)
            {
                Console.WriteLine("\nPolica " + i+":");
                foreach (artikal a in polica[i]) {
                    Console.WriteLine(a.index + " " + a.naziv + " - " +
a.get_cena().ToString() + " dinara");
                }
            }
        }
```

```
struct pare
{
   public int novcanica;
   public int kolicina;
}

pare[] parice = new pare[5];

public void daj_novac(int un)
{
   bool da = false;
   for (int i = 0; i < 5; i++)
   {
      if (parice[i].novcanica == un)
      {
         parice[i].kolicina++;
         da = true;
      }
   }
   if (da == false) { Console.WriteLine("Vrednost novcanice nije podrzana"); }
}

public void prodaj(int odakle)
{
```

```
        try
        {
            artikal prodat = (artikal)polica[odakle].Peek();



            if (prodat.cena > get_credit()) { Console.WriteLine("Nemate dovoljno
kredita za ovaj proizvod!"); }
            else
            {
                credit -= prodat.get_cena();
                Console.WriteLine("Proizvod " + prodat.naziv + " uspesno prodat po
ceni od " + prodat.get_cena() + "!");
                polica[odakle].Dequeue();
            }
        }
        catch (InvalidOperationException) {
            Console.WriteLine("Polica je prazna!");
        }
    }


    public bool ima(string ind) {
        for (int i = 0; i < 5; i++)
        {
            foreach (artikal a in polica[i])
            {
```

```
            if (a.index == ind)
            {
                return true;
                break;
            }
        }
    }
    return false;
}

int unos
{
    get { return unos; }
    set { unos = value; }
}



public Automat()
{
    parice[0].novcanica = 5;
    parice[1].novcanica = 10;
    parice[2].novcanica = 20;
    parice[3].novcanica = 50;
```

```
        parice[4].novcanica = 100;

        for (int i = 0; i < 5; i++)
        {
            parice[i].kolicina = 0;
        }

        for (int i = 0; i < 5; i++)
        {
            polica[i] = new Queue();
        }
    }

    public int get_credit(){
        return parice[0].kolicina * parice[0].novcanica + parice[1].kolicina *
parice[1].novcanica + parice[2].kolicina * parice[2].novcanica + parice[3].kolicina *
parice[3].novcanica + parice[4].kolicina * parice[4].novcanica;
    }

    public void daj_kusur() {
        int k100=0;
        int k50=0;
        int k20=0;
        int k10=0;
        int k5=0;
```

```
        while (credit >= 5) {
            if (credit >= 100) { credit = credit - 100; parice[4].kolicina--; k100++;}
            if ((credit < 100) && (credit >= 50)) { credit = credit - 50;
parice[3].kolicina--; k50++;}
            if ((credit < 50) && (credit >= 20)) { credit = credit - 20; parice[2].kolicina-
-; k20++;}
            if ((credit < 20) && (credit >= 10)) { credit = credit - 10; parice[1].kolicina-
-; k10++; }
            if ((credit < 10) && (credit >= 5)) { credit = credit - 5; parice[0].kolicina--;
k5++;}
        }
        Console.WriteLine(k100.ToString() + " " + k50.ToString() + " " +
k20.ToString() + " " + k10.ToString() + " " + k5.ToString() + " ");
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      String input="";
      Automat levi = new Automat();
      levi.daj_novac(100);
      levi.daj_novac(100);
      levi.daj_novac(100);
      levi.daj_novac(100);
```

```
        levi.credit = levi.get_credit();
        artikal Caprice = new artikal("Grcke Caprice",
DateTime.Parse("10/10/2020"), 210, "10",20, 4);
        artikal Cookies = new artikal("Grcki Cookies",
DateTime.Parse("14/10/2020"), 320, "11",20,4);
        levi.dodaj_artikl(Caprice,0);
        levi.dodaj_artikl(Cookies, 1);
        Console.WriteLine("Dobrodosli u Automat Simulator. Imate " +
levi.credit.ToString() + " dinara.  Izaberite proizvod:");
        levi.prikazi();

        while (input != "end")
        {
           Console.WriteLine("Ukucajte index police da biste kupili artikal.");
           input = Console.ReadLine();
           if (input == "end")
           {
              break;
           }
           levi.prodaj(int.Parse(input));
           Console.WriteLine("Kredit: "+levi.credit);

        }
        levi.daj_kusur();
```

```
        }
    }
}
```