Objektno programiranje III

Dostop do stanj objektov
Statična polja in metode
Lastnost/Property
Dedovanje

Razred – dostop do stanj objektov

- Možnost ***public***, da neposredno dostopamo do stanj/lastnosti objekta v “pravih programih” ni najboljša!
  - Ne moremo izvajati nobene kontrole nad pravilnostjo podatkov o objektu
  - Celoten program je bolj podvržen napakam
  - Težava pri kasnejši spremembi načina predstavitve podatkov o objektu
- Ideja:objekt sam naj poskrbi, da bo v pravilnem stanju.
  - Objekt je v bistvu “črna škatla”
  - Uporabnikom preprečimo “kukanje” v zgradbo objektov
  - Če bo hoteli dostopati do lastnosti objekta, bodo morali uporabljati metode (v teh metodah pa bo poskrbljen za kontrolo nad podatki)
  - Uporabnika pravzaprav ne zanima, kako so podatki predstavljeni, zanima ga le stanje podatkov

Metode za dostop do stanj objekta

- Za dostop do stanj objekta potrebujemo dve vrsti metod
  - Metode za dostop do stanj (za dostop do podatkov v objektu) – pogosto jim rečemo tudi “get” metode
  - Metode za nastavljanje stanj (za spreminjanje podatkov o objektu) – pogosto jim rečemo tudi “set” metode

Dostop do stanj objekta

- Ponovimo razloge, zakaj je dostop do podatkov v objektu preko metod boljši kot neposredni dostop:
  - Možnost kontrole pravilnosti!
  - Možnost kasnejše spremembe načina predstavitve (hranjenja podatkov).
  - Možnost oblikovanja pogleda na podatke:
    - Podatke uporabniku posredujemo drugače, kot jih hranimo. Tako denimo v razredu *Datum* mesec hranimo kot število, proti uporabniku pa je zadeva videti, kot bi uporabljali besedni opis meseca.
    - Dostop do določenih lastnosti lahko omejimo:
      - Npr. spol lahko nastavimo le, ko naredimo objekt (kasneje pa uporabnik spola sploh ne more spremeniti, saj se ne spreminja ... če odmislimo kakšne operacije, določene vrste živali ... seveda)
      - Hranimo lahko tudi določene podatke, ki jih uporabnik sploh ne potrebuje ..

Zgled – razred Clan

- Za primer vzemimo razred Clan in ustvarjanje novega člana

```
public class Clan
{
    public string ime;
    public string priimek;
    public int leto_vpisa;
    public string vpisna_st;
}
static void Main(string[] args)
{
    Clan novClan = new Clan();//Nov objekt razreda Clan
    novClan.ime = "Katarina";
    novClan.leto_vpisa = 208;
}
```

Ker so vsi podatki javni, smo za leto vpisa lahko vnesli nemogoč podatek 208 – bodisi zavestno, bodisi gre za tipkarsko napako.

Zgled – popravljen razred Clan

- Možnost napake pri določanju leta vpisa odpravimo tako, da napišemo objektno metodo, ki preverja, če je podatek smiselen, polje *leto_vpisa* pa označimo kot ***private***
  Objektna metoda (znotraj razreda Clan) za nastavljanje leta vpisa bi izgledala npr. takole:

```
public int NastaviLetoVpisa(int leto)
{
   //preverjanje pravilnosti vnesene letnice vpisa
  if ((1900 <= leto) && (leto <= 2020))
  {
    this.leto_vpisa = leto;//nastavimo novo leto vpisa
  }
    this.leto_vpisa=0;//za leto vpisa nastavimo privzeto
   vrednost
}
```

Zgled – razred Clan in novi objekti

- Ustvarimo nekaj novih objektov razreda Clan

```
Clan novClan = new Clan();
novClan.ime = "Katarina";
novClan.NastaviLetoVpisa(208)//leto vpisa bo 0

Clan novClan1 = new Clan();
novClan.ime = "Anja";
novClan1.NastaviLetoVpisa(-2000)//leto vpisa bo 0

Clan novClan2 = new Clan();
novClan2.ime = "Maja";
novClan2.NastaviLetoVpisa(2000)//leto vpisa bo 2000
```

Dostop do stanj

- V C# imamo torejmožnost, da dostop do spremenljivk lahko nadziramo. Poznamo 4 načine dostopa:
  - public
  - private
  - protected
  - internal

  Zadnjih dveh zaenkrat ne bomo uporabljali (2. letnik – dedovanje!!!).

  - ***Public***: Če je način dostopa nastavljen na *public*, to pomeni, da do lastnosti (komponent, spremnljivk, polj ...) lahko dostopajo vsi, od kjerkoli (iz katerihkoli datotek (razredov)) in sicer z

    ```
    ime_objekta.lastnost
    ```
  - ***Private:*** do lastnosti ne more dostopati nihče, razen metod znotraj razreda, ki poskrbijo za npravilne nastavitve stanj in še za pridobivanje vrednosti stanj **.**

Vaja

- Napišimo razred *Tocka*, ki naj ima dve zasebni polji (koordinati točke), privzeti konstruktor, ki obe koordinati postavi na 0, ter konstruktor z dvema parametroma, s katerima inicializiramo koordinati nove točke. Napišimo še metodo, ki izračuna in vrne razdaljo točke od središča koordinatnega sistema.

Statična polja in metode

- Statična polja in metode lahko uporabljamo ne da bi tvorili objekte – uporabljamo in kličemo jih torej neposredno nad razredom
  - Uporabljamo jih za različne konstante, za enolične identifikacije (ko naj ima npr. objekt svojo serijsko številko in so te številke zaporedne), vedno, ko je določen podatek skupen za vse objekte tega razreda
  - Kako vemo, ali naj bo komponenta (polje ali pa metoda) statična ali objektna? Statične komponente so tiste, ki so skupne za celoten razred – niso posebna lastnost enega (ali nekaj) primerkov objektov tega razreda. Lahko so zasebne (*private*) ali pa javne (*public*), odvisno pač od željenega načina dostopa. Če jih želimo skriti pred uporabniki, bo način dostopa *private*.

Zgled – statično polje

```
class Test
{
    public static double statičnoPolje; //Statično polje
    public double običajnoPolje;
}
static void Main(string[] args)
{
    //Zato, da uporabimo statično polje NE POTREBUJEMO OBJEKTA!!!
    Test.statičnoPolje=100;//nastavljanje vrednosti statič. polja
    //za nastavljanje vrednosti običajnega polja potrebujemo objekt
    Test novObjekt = new Test();
    novObjekt.običajnoPolje = 100;

    //od tu naprej je uporaba podobna npr.:
    Test.statičnoPolje = Test.statičnoPolje + 10;
    novObjekt.običajnoPolje = novObjekt.običajnoPolje+10;
}
```

Lastnost/Property

- Vrednosti polj smo objektom prirejali takole
  - Neposredno (če so bila javna)
  - S pomočjo konstruktorja (če smo ga napisali)
  - S pomočjo objektnih metod
- Ostaja pa še tretji način, ki je namenjen izkušenim programerjem – za vsako polje lahko definiramo ustrezno lastnost (*property*), s pomočjo katere dostopamo do posameznega polja, ali pa z njeno pomočjo prirejamo (nastavljamo) vrednosti polja.
  - Lastnosti v razredih torej uporabljamo za inicializacijo oziroma dostop do polj razreda (objektov)
  - Lastnost (*property*) je nekakšen križanec med spremenljivko in metodo.
  - Branje in izpis (dostop) vrednosti je znotraj lastnosti realizirana s pomočjo rezerviranih besed **get** in **set**: *get* mora vrniti vrednost, ki mora biti istega tipa kot lastnost (seveda pa mora biti lastnost istega tipa kot polje, kateremu je namenjena), v *set* pa s pomočjo implicitnega parametra **value** lastnosti priredimo (nastavimo) vrednost.

Zgled - property

```
class Artikel
{
    public string naziv;
    private double cena;
    public Artikel(string naziv, double cena)
    {
        this.naziv = naziv;
        this.cena = cena;
    }
    public double Cena  //lastnost oz. Property
    {
        get
        { return cena; }
        set
        { cena = value; }
    }
    public void Izpis() //objektna meoda
    {
        Console.WriteLine("Naziv artikla: " + naziv + ", cena: " + cena);
    }
}
static void Main(string[] args)
{
    Artikel A1 = new Artikel("Zvezek", 5.45);
    A1.naziv="Zvezek - A4"; //Ker je naziv artikla javno polje ga lahko spremenimo
    A1.Cena=6.77; //polje cena je zasebno, zato ga lahko spremenimo preko lastnosti Cena
    A1.Izpis(); //Izpis podatkov objekta A1
}
```

Vaje

- Potrebujemo razred, ki bo hranil podatke o objektih tipa *Instrukcije* s komponentama *predmet* (zasebno polje) in *ure* - število opravljenih ur(zasebno polje). Razred naj ima tudi konstruktor in ustrezno lastnost. Na osnovi razreda *Instrukcije* napišimo še testni program, ki bo kreiral dva objekta razreda *Instrukcije*.
- Sestavi razred *Pacient*, ki ima tri komponente: *ime* in *priimek* naj bosta obe *public*, *krvna_skupina* pa *private*, vse tri pa tipa string. Napiši metodo, ki vse tri komponente nastavi na "NI PODATKOV". Napiši metodo, ki sprejme vse tri podatke in ustrezno nastavi komponente. Z metodo *public string ToString()* naj se izpišejo podatki o pacientu (ime, priimek, krvna skupina). Ker je *krvna_skupina* zasebno/private polje, napiši še ustrezno *lastnost/property* ali pa metodo, ki sprejme podatek o krvni skupini in to vrednost priredi polju krvna_skupina.

Vaje

- Napiši razred *Kosarka*, za spremljanje košarkaške tekme. Voditi moraš število prekrškov za vsakega tekmovalca (12 igralcev), število doseženih točk (posebej 1 točka, 2 točki in 3 točke), ter metodo za izpis statistike tekme. Doseganje košev in prekrškov realiziraj preko metod *ZadelProstiMet*(), *ZadelZa2Tocki*, *ZadelZa3Tocke* in *Prekrsek*.

Dedovanje

- Pomen dedovanja

  - Dedovanje (Inheritance) je ključni koncept objektno orientiranega programiranja
  - Dedovanje je orodje, s katerim se izognemo ponavljanju pri definiranju različnih razredov, ki pa imajo več ali manj značilnosti skupnih.
  - Smisel in pomen dedovanja je v tem, da iz že zgrajenih razredov skušamo zgraditi bolj kompleksne
  - Primer iz biologije: pojem sesalec (konji in kiti – oboji dihajo zrak in imajo žive mladiče, a oboji imajo tudi svoje posebnosti).

Dedovanje - sintaksa

- Bazični razredi in izpeljani razredi
  - Sintaksa, ki jo uporabimo za deklaracijo, s katero želimo povedati, da razred podeduje nek drug razred, je takale:

```
class IzpeljaniRazred : BazičniRazred
{
. . .
}
```

Izpeljani razred deduje od bazičnega razreda

Zgled: dedovanje razreda Tocka

- Dedovanje razreda Tocka

```
class Tocka //bazični razred
{
    public Tocka(int x, int y) //konstruktor
    {
        //telo konstruktorja
    }
    //telo razreda Tocka
}
class Tocka3D : Tocka ////razred Tocka3D podeduje razred Tocka
{
    public Toca3D(int z)
    :base(x,y) //klic bazičnega konstruktorja Tocka(x,y)
    {
    //telo konstruktorja Tocka3D
    }
    //telo razreda Tocka3D
}
```