Osnove objektno orijentiranog programiranja

Stvaranje vlastitih klasa

Definiranje klase, varijable članice i modifikatori pristupa

- Da bi definirana klasa bila dostupna i izvan našeg programa piše se modifikator pristupa public

```
public class Ucenik
{
string ime;
public string prezime;
}
```

- I varijable članice mogu imati modifikator pristupa public.
- Dostupni modifikatori su: private, protected i public

Metode

- Objekti izvode metode i one su vezane uz njih.
- I metode imaju modifikatore pristupa kojeg slijedi tip metode, njeno ime i zatim parametri u zagradama (s tipom i imenom odvojeni zarezima).

```
class Ucenik
{
public string ime;
public string prezime;
private DateTime vrijemeInstanciranja=DateTime.Now;
public void IspisiDetalje()
{
Console.Writeln(„Ime: {0}\nPrezime: {1}\nVrijeme instanciranja: {2}”, this.ime, this.prezime, this.vrijemeInstanciranja); //za konzolnu aplikaciju
}
}
```

```
public string DohvatiDetalje()
{
return „Ime: „+this.ime+”\nPrezime: „+this.prrezime+
„\nVrijeme instanciranja: „+this.vrijemeInstanciranja;
}//i za aplikaciju sa grafičkim korisničkim sučeljem
```

Instanciranje objekata

- Novi objekt klase Ucenik može se instancirati pomoću operatora new: Ucenik u=new Ucenik();
- Pri instanciranju novog objekta se u dinamičkoj memoriji (hrpi) zauzima potreban prostor za spremanje vrijednosti svih varijabli članica i one se inicijaliziraju (na vrijednost praznog stringa i sl.).
- Nakon instanciranja može se preko imena objekta pristupati javno dostupnim članovima:

```
u.ime=„Hrvoje”;
u.prezime=„Horvat”;
Console.WriteLine(u.IspisiDetalje());
```

Konstruktori

- Pri instanciranju objekta pozivom operatora new poziva se konstruktor klase.
- Konstruktor je metoda koja se zove isto kao i klasa, nema posebno naveden tip i vraća novi objekt te klase. Može i ne mora sadržavati i dodatne naredbe.
- Podrazumijevani (default) konstruktor nema parametre i ne mora se u klasi eksplicitno navoditi.
- public Ucenik(){...}
- Kad se pri instanciranju novog objekta pomoću konstruktora trebaju inicijalizirati neke ili sve varijable članice na konkretne vrijednosti definira se parametrizirani konstruktor kojem se kao parametri prosljeđuju potrebne vrijednosti.
- public Ucenik(string ime, string prezime){this.ime=ime; this.prezime=prezime;}//ključna riječ this predstavlja objekt koji će biti instanciran
- (primjer 148, konzolno i grafičko sučelje)

Statički članovi i članovi instance

- Članovi instance primjenjuju se tek nakon što se iz klase instancira objekt.
- Statički članovi pozivaju se preko imena klase (npr. klasa Console).
- Za definiranje neke varijable ili metode kao statičkog člana klase upotrebljava se ključna riječ static i navodi ispred imena tipa u definiciji tog člana, npr. public static int brojInstanci;
- Vrijednost ovakve statičke varijable članice moguće je izvana dohvaćati bez instanciranja novog objekta izravno preko imena klase, npr. Console.WriteLine(Ucenik.brojInstanci);
- (primjer 150, konzolno)

Svojstva i učahurivanje (enkapsulacija)

- Svojstva su članovi klase koji dopuštaju vanjski pristup varijabli članici kroz posebno oblikovano sučelje koje se sastoji od dvije metode: get (metoda za dohvaćanje varijable članice) i set (metoda za postavljanje vrijednosti varijabli članici)
- Varijable članice definiraju se kao privatne, a zatim se učahuruju (encapsulate) u svojstvo te se njihovim vrijednostima pristupa preko imena svojstva koje je tipa public:

```
private string ime;
public string Ime
{
get{return ime;}
set{ime=value;}
}
```

Enkapsulacija

- Kad netko izvan klase dohvaća vrijednost svojstva, tada se poziva njegov pristupnik get u koje se samo prosljeđuje vrijednost privatne varijable članice
- Pokuša li netko izvana postaviti novu vrijednost svojstvu, tada se zapravo poziva metoda set u kojoj se nova vrijednost dodjeljuje varijabli članici, ključna riječ value predstavlja vrijednost koja se izvana dodjeljuje svojstvu
- Na ovaj je način stvarna vrijednost uvijek sačuvana u privatnoj varijabli članici, a metode get i set omogućuju joj pristup izvana
- Get i set metode mogu implementirati i provjere prije nego vrijednost bude vraćena odnosno dodijeljena varijabli članici
- (primjer 152)

Svojstva samo za čitanje

- Objekt se sam o sebi može brinuti kako ne bi došao u nedopušteno stanje: sve potrebno definirano je unutar same klase i ako netko izvana nepravilno rukuje ili pokuša dodijeliti neodgovarajuću vrijednost, klasa izbacuje grešku (Exception) i program se prekida (osim ako greška ne bude uhvaćena i obrađena u try-catch bloku)
- Svojstvo samo za čitanje posjeduje samo pristupnik get.
- (primjer 155)
- Kad se svojstvo definira na podrazumijevani način (bez dodavanja provjere unutar get ili set bloka), možemo se koristiti skraćenim zapisom za njegovo definiranje: public tip ImeSvojstva {get; set;}

Događaji

- Bit grafičkog korisničkog sučelja je obrada događaja – u Windows aplikacijama program većinu vremena čeka korisnikov klik na neku kontrolu na formi i zatim izvršava odgovor na taj događaj
- Windows Forms kontrole imaju već definirane očekivane događaje

Delegati i događaji

- Primjer: click događaj uzrokuje izvođenje metode tipa void (sa dva parametra) unutar klase Form. Prvi parametar je sender (tipa object), a drugi e (tipa EventArgs) i prema tome se stvaraju sve metode koje obrađuju događaje u .NET-u
- Akcija za pokretanje click događaja inicirana je unutar klase (npr. Button), a na nju odgovara metoda definirana u klasi Form – poziv je u jednoj klasi, a izvršavanje u drugoj – to se izvodi preko delegata
- U C# događaji se definiraju pomoću delegata (referentnih tipova koji učahuruju metode)
-

Definiranje događaja u klasi

- Za definiranje događaja u klasi najprije moramo definirati njegov pripadajući delegat
- Delegat specificira tip metode koja će se primjenjivati za obradu tog događaja: uobičajeno je da su metode za obradu svih događaja tipa void i imaju dva prije spomenuta parametra
- Definiramo delegat: `public delegate void ImeDelegata (object sender, EventArgs e);`
- Preko delegata definiramo događaj: `public event ImeDelegata ImeDogađaja;`
- Događaj je tipa ImeDelegata - delegat opisuje tip metode
- Slijedi da na događaj mogu odgovarati samo takve metode

Pozivanje događaja u klasi

- Događaj deklariran kao član klase treba pozvati u točno određenom trenutku – poziva se vanjska metoda definirana za obrađivanje tog događaja
- Prije poziva događaja, potrebno je provjeriti da li je on uopće implementiran (da li mu je izvan klase definirana metoda za obradu) pa je ispravan poziv događaja u klasi:

```
if (this.ImeDogađaja != null)
this.ImeDogađaja(this, new EventArgs());
```

- Ako je događaj implementiran, dodijeljena mu je neka metoda za obradu pa u pozivu događaja kao prvi parametar prosljeđujemo sam objekt koji je događaj „podignuo” (this). Drugi se parametar upotrebljava za obradu složenijih događaja.
- Vrijednost null upotrebljava se kod referentnih tipova (objekata, delegata) za označavanje prazne reference (varijabla referentnog tipa ima vrijednost null ako nije inicijalizirana).

Implementiranje događaja u glavnom programu

- Prilikom instanciranja objekta u glavnom programu, njegov se događaj implementira:

```
static void Main(string[] args) {
ImeKlase obj = new ImeKlase();
obj.ImeDogađaja += new ImeKlase.ImeDelegata
(obj_ImeDogađaja);
}
static void obj_ImeDogađaja(object sender, EventArgs e){}
```

- Odmah nakon instanciranja objekta, pomoću operatora += potrebno je povezati događaj s novom instancom njegovog delegata u čiji je konstruktor prosljeđeno ime metode koja će događaj obraditi. Ta metoda mora zadovoljavati dogovor (imati tip i parametre navedene u definiciji delegata).
- Pri implementaciji događaja u glavnom programu, VS pomaže tako da kod dizajniranja Windows formi metodu za obradu automatski kreira dizajner.
- Kad se implementira događaj u kodu: nakon što se navede ime događaja i operator +=, dvaput treba pritisnuti tipku TAB i VS će automatski dovršiti naredbu i kreirati metodu za obradu.
- Primjer str. 159 Primjer upotrebe događaja

Nasljeđivanje

- Potreba da se u više klasa nalaze ista svojstva – definiramo zajedničku baznu klasu iz koje se izvode ostale i nasljeđuju njena svojstva
- U definiciji izvedene klase nasljeđivanje se označava dvotočkom nakon koje slijedi ime bazne klase, npr. class Ucenik : Osoba...
- Primjer (162)

Skrivanje (hiding) metoda

- Ako metodu u izvedenoj klasi trebamo definirati na drugačiji način od onog u baznoj klasi, navodi se npr. public **new** string DohvatiDetalje()... čime se skriva istoimena metoda bazne klase

Premošćivanje (overriding) metoda

- Kad treba predefinirati metodu implementiranu u nekoj .NET klasi
- Npr. može se premostiti (zaobići) unaprijed definirana funkcionalnost naslijeđena iz bazne klase navođenjem: public override string ToString(){return this.Ime+” „+this.Prezime;}
- Primjer (166)
- Ključna riječ base predstavlja objekt bazne klase, pa ju moramo promijeniti ako želimo da metoda radi nešto drugo...
- Console.WriteLine uzima kao parametar bilo koji tip, pa se može koristiti i objekt na koji onda implicitno primijeni poziv metode ToString

Polimorfizam

- Pri upotrebi nasljeđivanja pomoću ključne riječi base može se dohvatiti član definiran u baznoj klasi iako je metoda premošćena u izvedenoj klasi. Na taj je način uvijek dostupna i bazna implementacija metode.
- Može se reći da premošćena metoda tako ima dva primijenjiva oblika.
- Mogućnost primjene više oblika tipova, bez obzira na njihove pojedinosti, zovemo polimorfizam.
- Primjer polimorfizma (168)