Станка Матковић, Мијодраг Ђуришић

МАЛА ШКОЛА ОБЈЕКТНО ОРИЈЕНТИСАНОГ ПРОГРАМИРАЊА У ПРОГРАМСКОМ ЈЕЗИКУ C#

Трећи део

Наслеђивање – први део

Наслеђивање је још једна важна особина објектно оријентисаних програмских језика. Она је последица генерализације као метода за моделовање објеката.

Сетимо се примера наставног процеса из првог дела „*Мале школе ООП у програмском језику* C#“. У наставном процесу учествују ученици, професори, психолог, директор, ... Сви они имају неке заједничке карактеристике али и своје специфичности. Сви учесници у наставном процесу имају своје име, презиме, датум и место рођења, адресу становања, ... Генерализацијом долазимо до класе (назовимо је `Osoba`) која обједињује заједничке особине и заједничке функционалности свих посебних група које учествују у наставном процесу.

Свака група има и своје специфичне особине, као и специфично понашање у наставном процесу. За ученика су, поред карактеристика описаних класом `Osoba`, важне и информације о разреду који похађа, о предметима које слуша, о резултатима које постиже али и акције које изводи у наставном процесу (учи лекцију, ради писмени задатак, одговара, ...). Слично, за професора можемо уочити низ додатних карактеристика (предмете које предаје, године стажа, платни разред, ...) као и низ специфичних функционалности у наставном процесу (предаје лекцију, оцењује, ...). Операцију извођења посебних класа из опште (генерализоване) класе зовемо специјализација.

Класе добијене специјализацијом, осим што наслеђују све чланове (атрибуте и методе) полазне класе, дефинишу и нове, специфичне чланове. Полазну класу зовемо основна класа (родитељска класа, надкласа), а класу која је наслеђује зовемо изведена класа (подкласа).

У изведеној класи дефинишемо само атрибуте и методе специфичне за ту класу (евентуално предефинишемо методе основне класе) али њени објекти наслеђују и све чланове основне класе. Изведена класа проширује, а у неким случајевима и прецизира основну класу.

У дефиницији изведене класе после навођења имена класе наводимо ‘:’ а затим следи име основне класе из које је изведена.

class <imeIzvedeneKlase>:<imeOsnovneKlase>
{
opis / definicija članova klase
}

У програмском језику C# класа може наследити само једну основну класу, односно не може настати као специјализација две или више класа. Такође, изведена класа не може наследити класу која садржи модификатор `sealed`. Ако желимо да дефинишемо класу из које се не могу изводити друге класе модификатор `sealed` наводимо у заглављу класе испред службене речи `class`.

Важно је напоменути да у изведеној класи, без обзира што наслеђује све чланове основне класе, не можемо приступити приватним члановима основне класе. Да се не би нарушила енкапсулација атрибути основне класе не треба да буду јавни а ако су приватни онда им се не може приступити из изведене класе. Зато постоји и трећи ниво приступа члановима класе – заштићени (енгл. *protected*). Заштићени чланови класе доступни су у класи у којој су дефинисани и у свим класама изведеним из те класе, а изван њих нису. Према томе, из изведене класе може се приступити јавним и заштићеним члановима основне класе, али не и приватним.

Да би био креиран објекат изведене класе мора се прво креирати његов основни, базни део са атрибутима дефинисаним у основној класи. За креирање објекта основне класе задужен је конструктор основне класе. Зато се у конструктору изведене класе имплицитно (аутоматски) позива конструктор без аргумената основне класе осим ако програмер експлицитно не наведе који конструктор основне класе позива приликом креирања објекта. Позив конструктора основне класе програмер реализује тако што после потписа конструктора изведене класе наведе две тачке (':') а затим службену реч `base` и у заградама редом параметре конструктора основне класе којег позивамо.

```
public class Osnovna {
    // ....
public Osnovna()
    {
// ....
    }
  public Osnovna(int x)
    {
// ....
    }
//...
}
public class Izvedena:Osnovna
{
    // ....
public Izvedena()
```

```
    {
//....
    }
public Izvedena(int a,int s):base(a)
    {
// ...
    }
//...
  }
```

Изведена класа је наслеђивањем добила све атрибуте и методе основне класе па објекат изведене класе садржи све чланове као и објекат основне класе. Зато се објекат изведене класе, без експлицитне конверзије, може доделити промењивој основне класе. Обрнуто није дозвољено јер објекат основне класе не садржи све елементе изведене класе. Важно је истаћи да преко промењиве основне класе којој смо доделили објекат изведене класе можемо позивати само чланове основне класе.

Ако су A и B промењиве, редом основне и изведене класе, дозвољена је додела $A = B$. После те доделе преко промењиве A можемо позивати само чланове основне класе. Није дозвољена додела $B = A$ јер је A промењива основне класе (садржи само чланове основне класе) па специфични чланови које би требало да има објекат изведене класе, B, не могу бити дефинисани том доделом.

У изведеној класи можемо, осим потпуно нових метода, дефинисати и методе које по имену, повратном типу и по аргументима (по типу, броју и редоследу аргумената) одговарају методима основне класе. Можемо рећи да ти методи у изведеној класи „крију“ одговарајуће методе основне класе. Скривеном члану основне класе у изведеној класи можемо приступити само ако испред имена члана додамо службену реч `base` и тачку (‘.’).

```
public class Osnovna
  {
// ....
    public void radi()
    {
    // ....
    }
  public string pisi()
    {
    // ....
    }
    //...
  }
public class Izvedena:Osnovna
  {
    // ....
```

```
new public void radi()
  {
  base.radi();
  //....
  }
public string pisi()
  {
  return base.pisi()+...;
  }
  //...
}
```

Ако у изведеној класи дефинишемо метод истог заглавља као у основној класи (са или без модификатора `new` испред заглавља) онда тај метод можемо позвати само као метод променљиве изведене класе (`new` се наводи да би се компајалеру ставило до знања да смо намерно, а не случајно, заклонили методу базне класе; ако се не наведе, компајлер упозорава програмера).

Izvedena B = new Izvedena();
Text=B.Pisi(); B.radi();//pozivaju se metodi klase Izvedena

Ако објекат декларишемо као објекат основне класе позивом метода ћемо приступати методу основне класе чак иако објекат креирамо као објекат изведене класе.

Osnovna A = new Izvedena();
A.radi();//poziva se metod klase Osnovna

Често се дешава да методе имају исту сврху али се различитио реализују у различитим изведеним класама. У класама изведеним из класе `Osoba` метод за приказ информација о особи различито се реализује јер у свакој класи имамо додатне информације, а метод за проверу да ли је особа старија од друге особе или метод за промену адресе особе у свим класама се реализују на исти начин. Да бисмо могли предефинисати метод у изведеној потребно је у основној класи прогласити метод виртуелним, навођењем службене речи `virtual` у заглављу метода, пре навођења повратног типа метода. У изведеној класи при дефиницији предефинисаног метода треба навести службену реч `override`.

```
  public class Osnovna
{
  // ....
  public virtual void radi()
    {
    // ....
    }
  public virtual string pisi()
    {
```

```
      // ....
      }
    //...
  }
public class Izvedena:Osnovna
  {
// ....
  public override void radi()
    {
      //....
    }
  //...
}
```

Позив виртуелног метода се врши на основу типа објекта који променљива садржи а не на основу типа променљиве.

Osnovna A = new Izvedena();
A.radi();//poziva se metod klase Izvedena

Предефинисање виртуелних метода у једној или више изведених класа може бити изостављено. Тада се за објекте тих класа позива одговарајући метод из класе која је у хијерархији класа најближа класи објекта.

Osnovna A = new Izvedena();
Text=A.pisi();//poziva se metod klase Osnovna

Избор виртуелног метода који се позива врши се у току извршавања, а не у току превођења, коришћењем технике динамичког повезивања.

Метод `ToString()` је виртуелни метод класе `object` која је основна класа за све класе реализоване у C#. Зато у свакој C# класи имамо могућност предефинисања овог метода. Ако га не предефинишемо, виртуелни метод класе `object` враћа пун назив класе у којој је објекат креиран.

Виртуелне методе омогућавају да променљива основне класе, у зависности од класе чији јој је објекат додељен, исти метод извршава на различите начине (коришћењем предефинисаног (енгл. *override*) метода из изведене класе којој објекат припада). Ту особину објектно оријентисаног програмирања зовемо полиморфизам (појављивање у више облика).

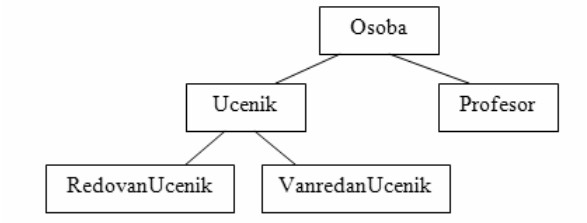
Osnovna A = new Osnovna();
A.radi();//poziva se metod klase Osnovna
A = new Izvedena();
A.radi();//poziva se metod klase Izvedena

Променљива A је променљива класе `Osnovna`. Када јој доделимо објекат исте класе позив метода `A.radi()` извршава виртуелни метод `radi()` из класе `Osnovna`. Када променљивој A доделимо објекат класе `Izvedena` истим позивом извршава се предефинисани (`override`) метод `radi()` из класе `Izvedena`.

Захваљујући полиморфизму можемо једноставно креирати апликације које раде на различитом хардверу и у различитим оперативним системима. Програмер може позивати одређене методе без осврtања на начин њихове реализације. Замислимо да из апликације треба штампати документ. Програмер ће при кодирању позвати одговарајући метод (рецимо `Print`) активног штампача без обзира који је модел у питању. Тај метод је различито реализован за сваки модел понаособ, али програмер не мора да размишља о начину реализације јер је сваки штампач објекат неке од изведених класа из основне класе (нпр. `Printer`). У основној класи је дефинисана виртуелна метода `Print` која је у изведеним класама предефинисана.

Пример 1. `Osoba`

У овом примеру навешћемо упрошћену реализацију основне класе `Osoba` и изведених класа `Ucenik` и `Profesor`, и једну једноставну апликацију као илустрацију претходног текста. Следећи дијаграм приказује хијерархију класа у примеру:



У реализацији наведених класа, која следи, коментарима су објашњени делови кода који се односе на наслеђивање.

```
public class Osoba//osnovna klasa
  {
string ime, prezime;
int brGodina;
string adresa;
public Osoba (string ime, string prezime,int brGodina, string adresa)
    {
this.ime = ime;
this.prezime = prezime;
this.brGodina = brGodina;
this.adresa = adresa;
    }
public Osoba()
    {
      ime = prezime =adresa="";
      brGodina = 0;
    }
public Osoba(Osoba X)
```

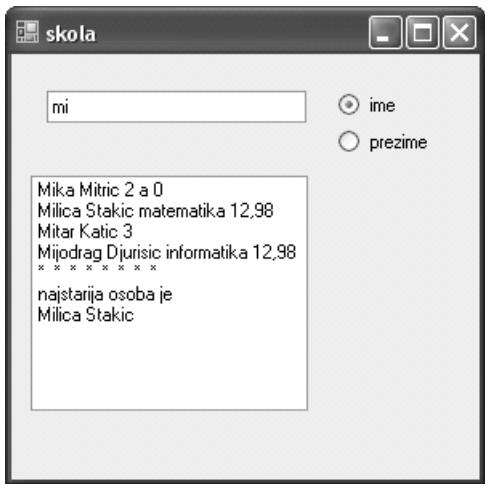
```
    {
      ime = X.ime;
      prezime = X.prezime;
      brGodina = X.brGodina;
      adresa = X.adresa;
    }
// virtuelna metoda
public virtual string informacije()
    {
return ime + " " + prezime;
    }
// nema potrebe da metodi koji slede budu virtuelni
//jer se jednako realizuju kod svih osoba
public bool starijaOd(Osoba B)
    {
return brGodina > B.brGodina;
    }
    public void promenaAdrese( string novaAdresa)
    {
      adresa = novaAdresa;
    }
  }
public class Ucenik :  Osoba
  {
int razred;
int []ocene;
public Ucenik():base()
// nije neophodno eksplicitno pozvati konstruktor bazne klase,
// jer ako ne pozovemo nijedan konstruktor bazne klase automatski se
// poziva podrazumevani konstruktor, tj. konstruktor bez argumenata
    {
      razred = 1;
      ocene=newint[15];
    }
public Ucenik(string uIme, string uPrezime, int uBrGod, string uAdresa, int uRaz)
      :  base(uIme, uPrezime,uBrGod,uAdresa)
//poziv odgovarajuceg baznog konstruktora
    {
      razred = uRaz;
      ocene = newint[15];
    }
public Ucenik(Ucenik U)
      :  base(U)
// base(U) poziv konstruktora kopije bazne klase
// bez obzira sto je parametar konstruktora kopije bazne klase promenljiva tipa
```

```
// Osoba, poziv konstruktora kopije klase Osobe sa argumentom U koji je objekat
// izvedene klase Ucenik je ispravno jer je dozvoljeno promenljivoj bazne
// klase dodeliti objekat izvedene klase
    {
      razred = U.razred;
      ocene=newint [U.ocene.Length];
for (int i = 0; i < U.ocene.Length; i++) ocene[i] = U.ocene[i];
    }
//predefinisanje viruelnog metoda
public override string informacije()
    {
      returnbase.  informacije ()+ " " + razred;
    //poziva se metod osnovne klase
    }
  }
public class RedovanUcenik :  Ucenik
  {
    char odeljenje;
    int brojIzostanaka; public RedovanUcenik():
    base()
      {
      odeee = ' ';
      brojIzostanaka = 0;
      }
  public RedovanUcenik(string rIme, string rPrezime, int rBrGod,
          string rAdresa, int rRaz,char rOdeljenje)
        :  base(rIme,rPrezime,rBrGod,rAdresa,rRaz)
      {
      odeljenje = rOdeljenje;
      brojIzostanaka =0;
      }
    public RedovanUcenik(RedovanUcenik r):  base(r)
        {
        odeljenje = r.odeljenje;
        brojIzostanaka = r.brojIzostanaka;
        }
    public override string informacije()
      {
        returnbase.  informacije ()+" "+odeljenje+" "+brojIzostanaka;
        //bazna klasa za klasu RedovanUcenik je klasa Ucenik
      }
    // Mozemo dodati metode karakteristicne za redovnog ucenika
    // na primer ocenjivanje ucenika, dodavanje izostanaka, izracunavanje proseka
  }
public class VanredanUcenik :  Ucenik
```

```
  {
    DateTime []datumPolagaa;
    public VanredanUcenik():  base()
      {
      datumPolaganja=new DateTime[15];
      }
    public VanredanUcenik(string vIme, string vPrezime, int vBrGod,
        string vAdresa, int vRaz)
    :  base(vIme, vPrezime,vBrGod, vAdresa, vRaz)
      {
      datumPolaganja=new DateTime[15];
      }
    public VanredanUcenik(VanredanUcenik V)
    :  base(V)
      {
      datumPolaganja = new DateTime[V.datumPolaganja.Length];
      for (int i = 0; i < V.datumPolaganja.Length; i++)
      datumPolaganja[i] = V.datumPolaganja[i];
      }
    // Mozemo dodati metode karakteristicne za vanrednog ucenika,
    // na primer unos ocene ucenika
  }
public class Profesor:Osoba
  {
    string predmet;
    float koeficijent;
    public Profesor()
      {
        predmet="";
        koeficijent=0;
      }
    public Profesor (string ime, string prezime, int brGod,
        string adresa,stringpredmet,float koeficijent)
:  base(ime, prezime, brGod,adresa)
      {
    this.predmet = predmet;
    this.koeficijent = koeficijent;
      }
    public Profesor (Profesor z) :  base(z)
      {
    this.predmet = z.predmet;
    this.koeficijent = z.koeficijent;
      }
    public override string informacije ()
      {
```

```
      returnbase.  informacije () + " " + predmet + " " + koeficijent;
    }
  // Mozemo dodati jos metoda karakteristicnih za profesora
  }
```

Следи једноставна апликација у којој користимо претходно реализоване класе. У фајлу `skola.txt` налазе се информације о учесницима наставног процеса у некој школи (највише 1000 учесника). За сваког учесника у једној линији је дата информација о врсти тог учесника (да ли је редован ученик, ванредан ученик или професор), а затим све потребне информације о њему (свака информација у посебној линији). Апликацијом је обезбеђен приказ информација о свим учесницима наставног процеса те школе чије име или презиме, у зависности од избора корисника, почиње датим стрингом. Обезбеђен је и испис имена и презимена најстарије особе међу приказаним особама.



Све учеснике наставног процеса, без обзира што припадају различитим класама, можемо објединити коришћењем низа a променљивих основне класе `Osoba`. Дозвољено је променљивој основне класе доделити објекат изведене класе, тако да сваком елементу низа аможемо доделити објекте класе `RedovanUcenik, VanredanUcenik, Profesor`.

```
    Osoba[] a = new Osoba[1000];
int n = 0;
Osoba citajOsobu(StreamReader sr)
    {
string vrsta = sr.ReadLine();
Osoba A;
string ime, prezime,adresa;
int brGod;
      ime = sr.ReadLine();
      prezime=sr.ReadLine();
```

```
      brGod=Convert.ToInt32(sr.ReadLine());
      adresa=sr.ReadLine();
if (vrsta == "redovan ucenik")
      A = new RedovanUcenik(ime, prezime, brGod, adresa,
        Convert.ToInt32(sr.ReadLine()), sr.ReadLine()[0]);
elseif (vrsta == "vanredan ucenik")
      A = new VanredanUcenik(ime, prezime, brGod,adresa,
        Convert.ToInt32(sr.ReadLine()));
else
      A = new Profesor(ime, prezime, brGod, adresa,
        sr.ReadLine(),Convert.ToSingle(sr.ReadLine()));
return A;
    }
private void Form1_Load(object sender, EventArgs e)
    {
StreamReader sr = new StreamReader("skola.txt");
Osoba max = null;
while (!sr.EndOfStream)
  {
      a[n] = citajOsobu(sr);
      lbSpisak.Items.Add(a[n].informacije());
    // metod informacije() je virtuelni i u svakoj izvedenoj klasi je predefinisan
    // pa se u zavisnosti od objekta koji se nalazi u promenljivoj a[n] poziva metod
    // informacije() iz odgovarajuće klase
if (max == null || a[n].starijaOd(max))
  //poziv metoda starijaOd definisanog u klasi Osoba
      max = a[n];
    n++;
  }
  sr.Close();
if (max != null)
    {
      lbSpisak.Items.Add("* * * * * * * *");
      lbSpisak.Items.Add("najstarija osoba je ");
      lbSpisak.Items.Add(max.Ime+ " "+max.Prezime);
    }
  }
private void tbIme_TextChanged(object sender, EventArgs e)
  {
string s = tbIme.Text.ToLower();
    lbSpisak.Items.Clear();
Osoba max = null;
for (int i = 0; i < n; i++)
    {
if((rbIme.Checked && a[i].Ime.ToLower().StartsWith(s))||
```

```
    (rbPrezime.Checked && a[i].Prezime.ToLower().StartsWith(s)))
  // koriscenje svojstava Ime i Prezime definisanih u klasi Osoba
    {
    lbSpisak.Items.Add(a[i].informacije());
    // poziv metoda informacije() iz odgovarajuce klase
if (max == null || a[i].starijaOd(max))
//poziv metoda starijaOd definisanog u klasi Osoba
        max = a[i];
    }
    }
if (max != null)
    {
    lbSpisak.Items.Add("* * * * * * * *");
    lbSpisak.Items.Add("najstarija osoba je ");
    lbSpisak.Items.Add(max.Ime + " " + max.Prezime);
    }
  }
```