Станка Матковић, Мијодраг Ђуришић

МАЛА ШКОЛА ОБЈЕКТНО ОРИЈЕНТИСАНОГ ПРОГРАМИРАЊА У ПРОГРАМСКОМ ЈЕЗИКУ C#

Први део

Класа, атрибути и методи

Основни појам у објектно оријентисаном програмирању је класа. Класа представља уопштење објеката који имају заједничке особине и функционалности. Особине описујемо атрибутима а функционалности методима.

Посматрајмо наставни процес у школи. Рецимо да у учионици 1 ученик IVа, Пера, ради задатак из математике по захтеву професора Лазе, а у учионици 2 професор Мика предаје свим ученицима одељења IVб нову лекцију из физике. Сасвим је могуће да сутрадан улоге професора буду промењене, па да професор Мика у IVа испитује или предаје нову лекцију из физике, а професор Лаза у IVб предаје нову лекцију из математике.

Већ у овом упрошћеном приказу наставног процеса можемо уочити основне *објекте* који учествују у њему (Пера, Мика, Лаза, IVа, IVб, математика, физика, учионица 1, ...). Лако се може уочити да Мика и Лаза координирају наставни процес тако што упознају ученике са новим градивом, врше проверу знања ученика, оцењују га, итд. Они имају и заједничке особине (име, стручна спрема, предмет који предају, године стажа). Самим тим они припадају једној од *класа* учесника у наставном процесу, класи Profesor. На сличан начин можемо уочити и класе Ucenik (Пера), Predmet (математика, физика), Odeljenje (IVа, IVб), ...

До класа долазимо полазећи од појединачних објеката. Посматрањем свих ученика уочавамо њихове заједничке особине. На пример, за све ученике је потребно пратити име, презиме, датум рођења, разред, одељење. Те особине називамо *атрибутима*. Такође, свим ученицима можемо придружити исте „акције“: ученик одговара и добија оцену, ученик мења одељење ... Те *акције* у оквиру класе називамо методима класе, и њима се описује функционалност објеката те класе.

Класу дефинишемо навођењем резервисане речи `class` иза које следи идентификатор класе (име класе). После имена, ако формирамо класу која наслеђује неку, претходно дефинисану класу, у заглављу наводимо ':' па име класе из које је изведена. Затим наводимо тело класе у коме у витичастим заградама {} дефинишемо чланове класе (атрибути и методи).

```
class <imeKlase> :<imePrethodnoDefinisaneKlase>
{
    opis / definicija članova klase
}
```

На пример, класу ученика можемо интуитивно дефинисати на следећи начин:

```
class Ucenik
{
  string ime, prezime;
  DateTime datumRodjenja;
  int razred;
  int SkolskaGodina;
  char odeljenje;
  Ocena [] ocene;
  double prosek()
  {
...
  }
  double prosek(Predmet P)
  {
...
  }
  void uci(Lekcija X)
  {
...
  }
}
```

Да бисмо направили објекат, употребљавамо израз облика

```
<ime objekta> =new <ime klase> () .
```

Коришћењем оператора `new` одваја се у динамичкој меморији простор за регистровање објекта класе `<ime klase>`. Овај оператор враћа адресу додељеног простора коју можемо доделити имену објекта:

```
Ucenik x=new Ucenik();
```

Атрибутима описујемо одређену особину објекта (име, презиме, датум рођења, разред, одељење). Најчешће, различити објекти исте класе имају различите вредности атрибута. Вредности атрибута дефинишу стање објекта. При опису атрибута морамо навести тип коме тај атрибут припада (целобројни, реални, знаковни, ...) и име атрибута. Приступ атрибутима објекта неке класе реализујемо на следећи начин:

```
imeObjekta.imeAtributa
```

На пример, ако је `x` објекат класе `Ucenik`, атрибуту `ime` приступамо навођењем `x.ime`.

Методом описујемо понашање објекта у одређеној ситуацији и под одређеним условима (`uci`), али и одређујемо нове вредности на основу особина које објекат поседује (`prosek`). На тај начин описујемо функционалност објекта.

Метод класе је именовани блок наредби који се састоји из заглавља и тела метода. У заглављу наводимо повратни тип (ако метод не производи вредност коју враћа, наводимо резервисану реч `void`), затим име метода за којим следи у малим заградама списак параметара метода. За сваки параметар наводи се тип коме тај параметар припада као и име параметра. После заглавља у витичастим заградама наводимо тело метода које се састоји из одговарајућих исказа програмског језика *C#*.

```
<povratni tip> <ime metoda>(<lista parametara>)
    { <telo metoda> }
```

Позив метода објекта у програмском језику *C#* реализујемо на следећи начин:

```
imeObjekta.imeMetoda(lista stvarnih parametara)
```

На пример, ако је `x` објекат класе `Ucenik`, метод `prosek` позивамо са `x.prosek()`, а метод `uci`, `x.uci(L)` где је L објекат класе `Lekcija`.

Уколико у телу метода користимо неки од атрибута или позивамо неки други метод те класе, не наводимо име објекта већ само име атрибута, односно методе. Уместо имена објекта можемо навести службену реч `this`. Објекат коме се обраћамо преко `this` је текући објекат.

Различити методи у једној класи могу имати исти назив али се морају разликовати по броју или типу параметара. Писање метода истог имена а различитих параметара назива се преклапање (`overloading`), преоптерећивање метода. У класи `Ucenik` метод просек смо преоптеретили: за ученика можемо рачунати просек свих закључених оцена (`prosek()`), а можемо и просек свих његових оцена из датог предмета (`prosek(Predmet P)`).

Својства

Једна од најзначајних карактеристика објектно оријентисаног програмирања је *енкапсулација* (затвореност, учауреље) објеката. Под енкапсулацијом објекта подразумевамо контролисан приступ елементима објекта. Неким елементима објекта могу приступити сви који тај објекат на било који начин користе. Те елементе називамо јавним (`public`) и они чине део преко кога објекат комуницира са другим објектима. Насупрот њима постоји део објекта који не користимо директно у комуникацији са другим објектима, да не би био изложен неовлашћеном приступу или промени. За такве елементе кажемо да су приватни (`private`).

Врло често постоји потреба да се атрибути заштите од неовлашћеног приступа или промене па су они углавном приватни чланови класе, док су методе у већини случајева јавне.

Посматрајмо класу `Vreme` којом ћемо описивати трајање одређених процеса. `Vreme` можемо дефинисати уз помоћ различитих атрибута на пример сатима, минутама и секундама, или данима, сатима, минутама и секундама и слично. Да бисмо што ефектније приказали различите елементе класе, време ћемо у примеру који следи дефинисати само једним атрибутом који представља број секунди.

```
public class Vreme
  {
    long sekunde; // ili Int64 sekunde;
    public long VratiSate() // broj sati sadrzanih u vremenu
    {
    return sekunde / 3600;
    }
    public long VratiMinute() // broj minuta sadrzanih u vremenu
    {
    return sekunde/60;
    }
    public long VratiSekunde()
    {
    return sekunde ;
    }
  }
```

Методи `VratiSate()` и `VratiMinute()` рачунају одређене информације па је разлог њиховог постојања интуитивно јасан. Поставља се питање зашто постоји метода `VratiSekunde()` када број секунди садржаних у времену описује атрибут `sekunde`. Атрибут `sekunde` је приватни члан класе па не можемо сазнати његову вредност ван ње. Зато уводимо јавни метод `VratiSekunde()` који ће нам омогућити приступ тој информацији. Овим методом добијамо информацију о броју секунди али је и даље не можемо променити (`read-only`).

Ради лакшег приступа приватним атрибутима класе у програмском језику *C#* можемо креирати својства (`property`) и она по потреби могу имати следеће компоненте:

- `get`, која на основу вредности атрибута одређује и враћа неку карактеристику објекта (коришћењем наредбе `return`)
- `set`, која на основу задате вредности (`value`) рачуна и поставља вредности атрибута.

Својство дефинишемо на следећи начин:

```
public <povratni tip svojstva> <ime svojstva>
{
    get { <telo get komponente> }
    set {<telo set komponente> }
}
```

Својства нису ни атрибути ни методе иако имају карактеристике и атрибута и метода. Као и атрибути својства немају параметре, а у њиховим компонентама,

као и у методама, може се навести произвољан низ наредби програмског језика *C#*. У `get` компоненти обавезно је навођење наредбе `return` којом се враћа израчуната вредност.

Својства се позивају навођењем имена објекта за којим следи име својства одвојено тачком. Ако се у изразу позив својства налази са леве стране оператора доделе, извршава се `set` компонента својства, а у супротном извршава се `get` компонента.

У класи `Vreme` можемо дефинисати својство `Sekunde` на следећи начин:

```
public long Sekunde
  {
    get { return sekunde; }
    set { sekunde = value; }
  }
```

Компонента `get` овог својства има еквивалентно дејство као и метода `VratiSekunde()`, а `set` компонента нам омогућава да постављамо вредности приватном атрибуту `sekunde`. Овако дефинисаним својством нарушавамо енкапсулацију јер је атрибут `sekunde` постао потпуно доступан. Самим тим могли смо дефинисати атрибут као јаван. У већини случајева потребно је да ван класе можемо прочитати вредност атрибута а да га не можемо мењати. Тада користимо својство код којег није дефинисана `set` компонента (`read only` својство, само за читање).

```
public long Sekunde
  {
    get { return sekunde; }
  }
```

Уколико је вредност атрибута ограниченог опсега, у `set` компоненти можемо извршити контролу вредности коју постављамо. На пример, ако класу `Vreme` описујемо атрибутима `sat`, `minut` и `sekund`, можемо проверавати да ли је вредност коју постављамо за атрибуте `minut` и `sekund` у дозвољеним границама (од 0 до 59).

Врло често постоје карактеристике објекта које зависе и могу се израчунати на основу вредности атрибута. Можемо дефинисати својства којима одређујемо вредност тих карактеристика.

Приметимо да објекте класе `Vreme` можемо описати и помоћу три цела броја који представљају број сати, минута (`minut<60`) и секунди (`sekund<60`). Дефинишимо својства `Sat`, `Minut` и `Sekund` која задовољавају једнакост:

```
    Sat*3600+Minut*60+Sekund=sekunde.

public long Sekund
{
  get { return sekunde%60; }
}
public long Minut
```

```
{
  get { return (sekunde / 60) % 60; }
}
public long Sat
{
  get { return sekunde / 3600; }
}
```

Ова својства нам омогућавају да прочитамо вредности `Sat`, `Minut` и `Sekund`. Било би природно да можемо и поставити њихову вредност. Промена вредности сваке од ових карактеристика повлачи и промену атрибута `sekunde`, па можемо реализовати и одговарајуће `set` компоненте.

```
public long Sekund
{
  get { return sekunde%60; }
  set { sekunde = sekunde-Sekund+value; }
}
public long Minut
{
  get { return (sekunde / 60) % 60; }
  set { sekunde = sekunde-Minut*60+value*60; }
}
public long Sat
{
  get { return sekunde / 3600; }
  set { sekunde = sekunde - Sat * 3600 + value * 3600; ; }
}
```

Индексери

У програмском језику $C\#$ можемо имати чланове класе које зовемо индексерима, који имају сличности са својствима али и својих специфичности.

Индексер нам омогућава да појединим карактеристикама објеката класе приступамо на ефикасан начин, навођењем имена објекта и у угластим заградама индекса који нас упућује на одговарајућу карактеристику. Као и својства, индексери имају `get` и `set` компоненту у којима се у зависности од вредности индекса одређује карактеристика објекта, односно постављају вредности атрибута. За разлику од својстава индексер има најмање један параметар а може имати и више параметара, па објекат добија више димензија. Индексер дефинишемо на следећи начин:

```
public <povratni tip indeksera> this[ <tip indeksa > <ime indeksa>, ...]
{
  get { <telo get komponente> }
  set {<telo set komponente> }
}
```

Индексеру програмер не задаје име јер он увек користи име `this`. Слично методама, индексери могу имати преклопљене верзије, које се морају разликовати по броју или по типу индекса.

Класу `Vreme` можемо проширити индексером који, у зависности од индекса, враћа `Sat`, `Minut` или `Sekund`. За индекс можемо изабрати: претходно дефинисани набројиви тип (`enum`) чије су могуће вредности `sat`, `minut` и `sekund`, или тип `char` тако да за вредности индекса редом '`h`', '`m`' и '`s`' враћа `Sat`, `Minut` и `Sekund`, или тип `string`. У наведеној реализацији индексера користимо претходно дефинисана својства. Овај индексер могли смо реализовати и без коришћења тих својстава и у том случају он би могао и заменити та својства. На сличан начин како је реализована `set` компонента својстава `Sat`, `Minut` и `Sekund` можемо реализовати `set` компоненту за сваки од наведених индексера.

```
public enum JedinicaVremena
  {
    sat,
    minut,
    sekund
  };
public long this[JedinicaVremena j]
  {
    get
      {
        if (j == JedinicaVremena.sat)
          return Sat; // return sekunde / 3600;
        if (j == JedinicaVremena.minut)
          return Minut; // return (sekunde / 60) % 60
        return Sekund;
      }
  }
public long this[char ch]
  {
    get
      {
        if (ch == 'h')
          return Sat;
        if (ch == 'm')
          return Minut;
        if (ch == 's')
          return Sekund;
        return sekunde;
      }
  }
```

Ако је `A` објекат класе `Vreme` вредност израза `A[JedinicaVremena.sat]` и `A['h']` је број сати садржаних у времену `A`.

Конструктори

У *C#*-у при декларацији неке инстанце класе, објекта, резервише се простор за адресу објекта (референцу) а сам објекат се креира коришћењем оператора `new`.

```
<ime klase> <ime objekta> = new <ime klase>(<lista parametara>) ;
```

Оператор `new` враћа адресу новог објекта која се додељује имену објекта.

Конструктор је метод који се позива при креирању објекта. Конструктори су саставни део сваке класе и носе њено име. Позивом конструктора објекат почиње свој живот. Класа може имати више конструктора који се разликују по листи параметара. Листа параметара најчешће садржи вредности којима иницијализујемо атрибуте објекта, а може бити и празна.

Ако програмер не напише ни један конструктор програмски преводилац ће направити подразумевани конструктор (конструктор без параметара, нумеричким типовима додељује `0`, логичким атрибутима додељује вредност `false`, референтне атрибуте поставља на `null`).

Ако пишемо више конструктора, како сви имају име класе, морају се разликовати по листи параметара, тј. по броју или типу параметара.

За класу `Vreme`, дефинисану у претходном делу чланка, можемо дефинисати следеће конструкторе:

- Конструктор без параметара који поставља атрибут `sekunde` на 0
```
public Vreme()
{
  sekunde = 0;
}
```
- Конструктор који за параметар има иницијални број секунди
```
public Vreme(long x)
{
  sekunde = x;
}
```
- Конструктор који за параметар има број сати и број минута
```
public Vreme(long h, long m)
{
  sekunde = 3600*h+60*m;
}
```
- Конструктор који за параметар има број сати, минута и секунди
```
public Vreme(long h, long m, long s)
{
  sekunde = 3600 * h + 60 * m + s;
}
```
- Конструктор који за параметар има `string` облика ‘‘`sati:minuti:sekunde`’’ (уколико се овом конструктору проследи другачији формат стринга он атрибут `sekunde` поставља на 0)

```
public Vreme(string s)
{
  try
    {
      long h = Convert.ToInt64(s.Substring(0, s.IndexOf(':')));
      long m = Convert.ToInt64(s.Substring(s.IndexOf(':')  + 1,
        s.LastIndexOf(':')  - s.IndexOf(':')  - 1));
      long sek = Convert.ToInt64(s.Substring(s.LastIndexOf(':')+1));
        sekunde = 3600 * h + 60 * m + sek;
    }
  catch
  {
    sekunde = 0;
  }
}
```

- Такозвани конструктор копије који за параметар има други објекат класе `Vreme`, `v`, `sekunde` поставља на вредност истог атрибута параметра `v`.

```
public Vreme(Vreme v)
{
  sekunde = v.sekunde;
}
```

Закључак

Ово је први у низу чланака који планирамо, којима ћемо покушати да професорима информатике приближимо објектно оријентисано програмирање и један савремени програмски језик који се, по нашим досадашњим искуствима, врло успешно може употребити у настави, у свим фазама учења програмирања.

У овом чланку смо вас упознали са основним елементима класе а планирано је да у наредна два обрадимо операторе и наслеђивање.

Предлажемо читаоцима да слично нашем примеру класе `Vreme` реализују и следеће класе:

- `TackaDekart` која представља тачку у Декартовом координатном систему за коју можете дефинисати својства `RastojanjeOdCentra`, `Kvadrant`
- `Krug`, задат са две дијаметрално супротне тачке, са својствима `Centar` и `Poluprecnik`
- `Vektor`, задат једном тачком у равни, са својствима `Intezitet`, `FazniUgao`

За све три класе могуће је реализовати и разноврсне конструкторе, као и индексере.

За крај, ево и једног сегмента кода показне апликације у коме се може уочити исправан начин позивања сваког од елемената класе коју смо кроз чланак обрадили:

```
//kreiranje objekta uz poziv podrazumevanog konstruktora
  Vreme T = new Vreme();
//koriscenje set komponente svojstava Sat, Minut, Sekund
  T.Sat = 12;
  T.Minut = 70;
  T.Sekund = 0;
//koriscenje get komponente indeksera sa enum indeksom
  Text=T[JedinicaVremena.sat]+" "+T[JedinicaVremena.minut];
//kreiranje objekta uz poziv konstruktora sa parametrom tipa string
  T = new Vreme("12:46:30");
//koriscenje get komponente svojstava Sat, Minut, Sekund
  label1.Text = T.Sat + " " + T.Minut + " " + T.Sekund;
//kreiranje objekta uz poziv konstruktora sa parametrom tipa string
  T = new Vreme("1:20:4");
//koriscenje get komponente indeksera sa char indeksom
  label1.Text += " " + T['h'] + " " + T['m'] + " " + T['s'];
```