Станка Матковић, Мијодраг Ђуришић

МАЛА ШКОЛА ОБЈЕКТНО ОРИЈЕНТИСАНОГ ПРОГРАМИРАЊА У ПРОГРАМСКОМ ЈЕЗИКУ C#

Други део

Оператори

Као што смо нагласили у претходном чланку, методом описујемо понашање објекта у одређеној ситуацији и под одређеним условима, одређујемо нове вредности на основу особина које објекат поседује али и описујемо одређене акције међу објектима које за резултат имају или нови објекат или неку вредност.

Лако се може направити аналогија између оваквих акција и математичких операција које дефинишемо као пресликавање скупа уређених n-торки у неки скуп објеката или вредности. У складу са тим у већини објектно оријентисаних програмских језика постоји могућност дефинисања оператора, пре свега да би се што једноставније позивали и што ефектније користили такви методи.

У C# не можемо дефинисати нове операторе али у свакој класи можемо предефинисати велики број постојећих. Предефинисањем оператора у класи додељујемо оператору ново значење које се примењује када се оператор позива за објекте те класе. Понашање истог оператора за друге типове објеката се не мења. У табели која следи наведени су оператори у C# који се најчешће предефинишу.

Унитарни оператори	Бинарни оператори
$+, -, !, \sim, ++, --$	$+, -, *, /, \%, \&, \mid, \hat{}, <<, >>$ $==, !=, <, >, <=, >=$

Предефинисањем оператора не можемо променити арност (број операнада), приоритет и асоцијативност (груписање) оператора.

У општем случају оператор се записује на следећи начин:

```
public static <povratni tip> operator <operacijski znak>(<lista parametara>)
    <telo operatora>
```

Службена реч `static` означава да је оператор статички члан класе, што значи да није као остали („не статик“) чланови везан за објекат `this` већ за саму класу. Када је неки атрибут класе `static` онда постоји само једна копија тог

атрибута коју деле сви објекти те класе. Када је метод класе `static` он није везан за конкретни објекат класе, односно не може користити `this`. Статичке методе се користе да израчунају одређене вредности а не да промене стање објекта (`Math` класа у C# садржи статичке методе `Sqrt`, `Sin`, `Cos`, ...). Статичке чланове класе можемо користити без креирања објекта.

Оператори су `static` јер иако се примењују на објектима класе нису њихов саставни део нити њихова функционалност већ су функционалност класе. У неким, старијим програмским језицима оператори се везују за објекте али је начин на који су оператори реализовани у C# практичнији и природнији.

Код оператора `<povratni tip>` може бити било који тип података дефинисан у C#, системски или од стране корисника. Њиме дефинишемо тип резултата операције реализоване оператором. Повратни тип не може бити `void` јер оператор мора имати повратну вредност.

Службена реч `operator` заједно са операцијским знаком заправо представља име методе. Скуп могућих вредности за `<operacijski znak>` дат је претходно наведеном табелом. Уколико изабрани операцијски знак представља унарну операцију, `<lista parametara>` садржи само један параметар а уколико представља бинарну операцију, `<lista parametara>` садржи два параметра међусобно одвојена зарезом. У свакој `<lista parametara>` бар један од параметара мора бити објекат класе чији је оператор члан.

У класи Razlomak можемо дефинисати више оператора. Наводимо исправна заглавља неких од њих:

- `public static Razlomak operator +(Razlomak a, Razlomak b)`
  оператор за одређивање збира разломака a и b
- `public static Razlomak operator *(Razlomak a, Razlomak b)`
  оператор за одређивање производа разломака a и b
- `public static Razlomak operator +(Razlomak a, int b)`
  оператор за одређивање збира разломка a и целог броја b
- `public static Razlomak operator -(Razlomak a)`
  оператор за одређивање разломка супротног знака од разломка a
- `public static Razlomak operator ~(Razlomak a)`
  оператор за одређивање реципрочне вредности разломка a
- `public static Razlomak operator -(Razlomak a, Razlomak b)`
  оператор за одређивање разлике разломака a и b
- `public static bool operator >(Razlomak a, Razlomak b)`
  оператор за проверу да ли је разломак a већи од разломка b

У телу оператора мора постојати једна или више команди `return <izraz>` где је `<izraz>` произвољан израз типа `<povratni tip>`.

Унарни оператори се позивају префиксно, `<operacijski znak><objekat klase>`. Оператори ++ и −− имају и постфиксну нотацију па се могу позивати и постфиксно, `<objekat klase><operacijski znak>`. Бинарни оператори се позивају инфиксно `<stvarni parametar><operacijski znak><stvarni parametar>`.

У следећем сегменту команди C# наводимо исправне позиве оператора класе `Razlomak` чија су заглавља претходно наведена.

```
Razlomak x = new Razlomak(3, 4);
Razlomak y = new Razlomak(-12, 5);
Razlomak z;
z = x + y;
z = x + 5;
z = ~x;
z = -y;
if (x > y)
  z = x  y*z + 3;
else
  x += y;
```

Предефинисањем бинарних оператора $+, -, *, /, \%, \&, |, \hat{}, <<, >>$ аутоматски су предефинисани и оператор $+=, -=, *=, /=, \%=, \&=, |=, \hat{}=, <<=, >>=$.

Оператори за поређење морају бити предефинисани у пару. Ако предефинишемо оператор == онда морамо предефинисати и оператор !=. Слично и за операторе $<$, $>$ и $<=$, $>=$.

Класа Razlomak

Употребу оператора илуструјемо на примеру класе `Razlomak`, јер су операције које се изводе над скупом разломака (рационалних бројева) свима блиске а ниједан програмски језик нема уграђен тип података који представља разломак.

Дефиншимо бинарни оператор + којим реализујемо операцију сабирања два `Razlomka`. Резултат је, у складу са правилима сабирања рационалних бројева, такође `Razlomak`.

```
public static Razlomak operator +(Razlomak a, Razlomak b)
  {
    Razlomak r = new Razlomak();
    r.imenilac = a.imenilac * b.imenilac;
    r.brojilac = a.brojilac * b.imenilac + b.brojilac * a.imenilac;
    r.skrati();
    return r;
  }
```

Операције не мењају операнде већ креирају резултат у зависности од вредности операнада. Зато на почетку методе креирамо нови `Razlomak` r који ће

представљати резултат, а његове атрибуте постављамо у складу са математичким правилима сабирања разломака. Ради једноставније реализације при сабирању не тражимо НЗС именилаца већ операнде доводимо на заједнички именилац који представља производ полазних именилаца, па извршимо потребно сабирање. Затим `Razlomak` r доводимо у нескратив облик, дељењем бројиоца и имениоца њиховим НЗД, што је реализовано методом `skrati`.

Ако у класи `Razlomak` напишемо конструктор који на основу датог бројиоца и имениоца иницијализује атрибуте објекта тако да он представља прави разломак (бројилац и именилац узајамно прости бројеви) оператор + можемо дефинисати на следећи начин.

```
public static Razlomak operator +(Razlomak a, Razlomak b)
  {
    return new
    Razlomak(a.brojilac * b.imenilac + b.brojilac * a.imenilac,
    a.imenilac * b.imenilac);
  }
```

Оператори као и све друге методе могу се преоптеретити, тако да можемо писати више оператора истог имена са различитом листом параметара. Претходно дефинисаним оператором реализована је операција сабирања два разломка. Често је потребно и сабирање објекта класе `Razlomak` и целог броја, па можемо дефинисати оператор + са параметрима редом `Razlomak, int` као и оператор + са параметрима редом `int, Razlomak`. Приликом реализације метода `operator +(Razlomak a, int b)` креирамо нови разломак $b/1$ од целог броја b, а затим коришћењем оператора + за сабирање два разломка саберемо тај разломак и разломак a.

```
public static Razlomak operator +(Razlomak a, int b)
  {
    return a + new Razlomak(b,1);
  }
```

Коришћењем претходно дефинисаног оператора + можемо дефинисати и сабирање целог броја и објекта класе `Razlomak` на следећи начин.

```
public static Razlomak operator +(int b,Razlomak a)
  {
    return a + b;
  }
```

Слично можемо дефинисати и остале операторе за рад са рационалним бројевима, одузимање, множење, дељење. И те операторе морамо преоптеретити на сличан начин ако желимо да омогућимо извођење операција између разломака и целих бројева.

Дељење два разломка можемо реализовати као множење разломка његовом реципрочном вредношћу. Зато је потребно дефинисати оператор за одређивање реципрочне вредности, то је унарни оператор који једноставно реализујемо:

```
public static Razlomak operator ~(Razlomak a)
  {
    return new Razlomak(a.imenilac, a.brojilac);
  }
```

Помоћу оператора можемо омогућити и конверзију објеката класе (типа података) коју дефинишемо у објекте друге класе или у податке основног типа као и обрнуто.

Конверзија се дефинише као унарни оператор. Име тог оператора је име типа у који се врши конверзија параметра оператора. Параметар оператора или тип у који се врши конверзија, али не оба, морају бити из класе у којој дефинишемо конверзију.

Конверзија може бити имплицитна или експлицитна. Приликом рачунања вредности израза да би сви подаци у изразу били истог типа врши се аутоматски њихова имплицитна конверзија (без експлицитног навођење од стране програмера). Имплицитном конверзијом не долази до губитка информација па се може безбедно позивати и без знања програмера.

Експлицитну конверзију програмер захтева додатним кодом, коришћењем оператора `cast`. Приликом експлицитне конверзије може доћи до губитка информација.

Заглавља имплицитног и експлицитног оператора конверзије у C# изгледају овако:

```
public static implicit operator TipRezultata(TipParametra parametar)
public static explicit operator TipRezultata(TipParametra parametar)
```

Сваки цео број можемо тумачити као рационалан број па је могуће дефинисати имплицитну конверзију из податка типа `int` у објекат класе `Razlomak`.

```
public static implicit operator Razlomak(int i)
  {
    return new Razlomak(i,1);
  }
```

С друге стране, за сваки разломак можемо одредити његову тачну или приближну децималну вредност, што значчи да при конверзији из разломка у реалан број може доћи до губитка информација. Зато је природно дефинисати експлицитну конверзију из класе `Razlomak` у основни тип `double`.

```
public static explicit operator double(Razlomak r)
  {
    return (double)r.brojilac / r.imenilac;
  }
```

Дефинисањем оператора имплицитне конверзије губи се потреба за преоптерећеним операторима којима омогућавамо извођење операција између објеката класе `Razlomak` и целих бројева. У следећем сегменту наредби C# приказана је

употреба имплицитног и експлицитног оператора конверзије. У наредби `c = a + 5` приликом рачунања вредности израза цео број 5 се аутоматски (имплицитном конверзијом) преводи у разломак па се извршава операција сабирања између два разломка. У наредби `x = (double)a + 5` коришћењем `cast` оператора (`double`) `Razlomak a` се конвертује експлицитном конверзијом у реалан број па се врши операција сабирања између реалних бројева.

```
Razlomak a=new Razlomak(3,4), b=new Razlomak(textBox1.Text), c;
double x;
c = a + 5;
x = (double)a + 5;
b = a / 4 - 3;
x = (double)(a + b);
```

Ради бољег сагледавања свих компоненти класе `Razlomak` наводимо целу класу.

```
public class Razlomak
  {
    #region Atributi
    int imenilac;
    int brojilac;
    # endregion

    #region Konstruktori
    public Razlomak()
    {
      imenilac = 1;
      brojilac = 0;
    }
    public Razlomak(int br)
    {
      brojilac = br;
      imenilac = 1;
    }
    public Razlomak(int br, int im)
    {
      if (im = = 0)
        throw new Exception("Greska:  imenilac0");
      brojilac = br;
      imenilac = im;
      if (imenilac < 0)
      {
        brojilac=-brojilac;
        imenilac=-imenilac;
      }
      skrati();
```

```
    }
    public Razlomak(string s)
    {
      int p = s.IndexOf('/');
      if (p != -1)
      {
        brojilac = Convert.ToInt32(s.Substring(0, p));
        imenilac = Convert.ToInt32(s.Substring(p + 1));
        if (imenilac = = 0) throw new Exception("Greska:  imenilac 0");
        this.skrati();
      }
      else
      {
        brojilac = Convert.ToInt32(s);
        imenilac = 1;
      }
    }
    private static int nzd(int a, int b)
    {
      if (b = = 0)
        return a;
      return
        nzd(b, a % b);
    }
    private void skrati()
    {
      int p = nzd(Math.Abs(brojilac), Math.Abs(imenilac));
      imenilac /= p;
      brojilac /= p;
    }
    #endregion

    #region Operatori
    public static Razlomak operator +(Razlomak a, Razlomak b)
    {
      return new Razlomak(a.brojilac * b.imenilac + b.brojilac * a.imenilac, a.imenilac
* b.imenilac);
    }
    public static Razlomak operator -(Razlomak a)
    {
      return new Razlomak(-a.brojilac, a.imenilac);
    }
    public static Razlomak operator -(Razlomak a, Razlomak b)
    {
      return a + (-b);
    }
```

```
  public static Razlomak operator *(Razlomak a, Razlomak b)
  {
    return new Razlomak(a.brojilac * b.brojilac, a.imenilac * b.imenilac);
  }
  public static Razlomak operator (Razlomak a)
  {
    return new Razlomak(a.imenilac, a.brojilac);
  }
  public static Razlomak operator /(Razlomak a, Razlomak b)
  {
    return a * b;
  }
  public static explicit operator double(Razlomak r)
  {
    return (double)r.brojilac / r.imenilac;
  }
  public static implicit operator Razlomak(int i)
  {
    return new Razlomak(i, 1);
  }
  public static bool operator >(Razlomak A, Razlomak B)
  {
    return A.brojilac * B.imenilac > B.brojilac * A.imenilac;
  }
  public static bool operator <(Razlomak A, Razlomak B)
  {
    return A.brojilac * B.imenilac < B.brojilac * A.imenilac;
  }
  #endregion

  public string UString()
  {
    if (brojilac = = 0)
      return "0";
    else
      if (imenilac = = 1)
        return brojilac + "";
      else
        return brojilac + "/" + imenilac;
  }
}
```

Класе Vreme, Datum, VremenskiTrenutak

Нека је класа `Vreme` дефинисана као у претходном чланку (један атрибут, секунде, одговарајући конструктори, својства, ...). Ако посматрамо објекат те

класе A који представља време трајања филма и објекат исте класе B који представља време трајања филмског журнала и реклама, сабирањем та два времена се може добити време трајања биоскопске пројекције, C. Ако бисмо написали оператор + за класу `Vreme` могли бисмо време C да одредимо следећим позивом тог оператора:

$$C = A + B;$$

Оператор сабирања можемо дефинисати на следећи начин:

```
public static Vreme operator+(Vreme A, Vreme B)
{
    return new Vreme(A.sekunde+B.sekunde);
}
```

На сличан начин можемо дефинисати и операторе:

- `public static Vreme operator +(Vreme a,int b)`
  оператор за одређивање збира времена a и броја секунди b
- `public static Vreme operator -(Vreme a, Vreme b)`
  оператор за одређивање апсолутне разлике времена a и b
- `public static bool operator >(Vreme a, Vreme b)`
  оператор за проверу да ли време a траје дуже од времена b
- `public static bool operator <(Vreme a, Vreme b)`
  оператор за проверу да ли време a траје краће од времена b

Класу `Datum` можемо реализовати са три целобројна атрибута (`dan, mesec, godina`) са одговарајућим својствима и конструкторима:

```
public class Datum
  {
    #region AtributiISvojstva
    private int dan, mesec, godina;

    public int Dan
    {
    get { return dan; }
    }

    public int Mesec
    {
      get { return mesec; }
    }

    public int Godina
    {
      get { return godina; }
    }

    #endregion
```

```
    #region Konstruktori

    public Datum()
    {
      dan = 1;
      mesec = 1;
      godina = 2010;
    }

    public Datum(Datum D)
    {
      godina = D.godina;
      mesec = D.mesec;
      dan = D.dan;
    }

    public Datum(string s)
    {
      dan = Convert.ToInt32(s.Substring(0,s.IndexOf('.')));
      s=s.Remove(0, s.IndexOf('.')  + 1);
      mesec = Convert.ToInt32(s.Substring(0, s.IndexOf('.')));
      godina = Convert.ToInt32(s.Substring(s.IndexOf('.')  + 1));
    }

    public Datum(int d, int m, int g)
    {
      godina = g;
      mesec = m;
      dan = d;
    }
    #endregion
.  .  .
  }
```

У овако реализованој класи можемо дефинисати разноврсне операторе. Један од интересантнијих оператора је унарни оператор који обезбеђује прелазак у следећи дан, односно одређивање сутрашњег датума. Реализоваћемо га као `operator++`:

```
public static Datum operator ++(Datum D)
  {
    int d = D.dan + 1;
    int m = D.mesec;
    int g = D.godina;
  //  статичким методом brDana(m,g)
  //  одређујемо број дана у месецу m године g
    if (d > brDana(m, g))
    {
```

```
      d = 1;
      m++;
      if (m == 13)
      {
        m = 1;
        g++;
      }
    }
    return new Datum(d, m, g);
  }
```

Овај оператор предефинише познати оператор увећавања за 1 из програмског језика C па је и његово коришћење у складу са тим. Наиме, овај оператор као и њему сродни оператор `--` може се позивати и у префиксној и у постфиксној нотацији. Он при сваком позиву мења вредност параметра додељујући му повратну вредност али је вредност самог израза `<parametar>++` једнака старој вредности параметра, док је вредност израза `++<parametar>` једнака новој вредности.

Можемо приметити да у класи `Datum` нема смисла дефинисати оператор којим се сабирају два објекта класе `Datum` али се може дефинисати оператор којим се на објекат класе `Datum` може додати цео број дана. Такође, од датума можемо одузимати известан цео број дана али се може одредити и број дана између два датума.

- `public static Datum operator +(Datum a,int b)`
- `public static Datrum operator -(Datrum a, int b)`
- `public static int operator -(Datum a, Datum b)`

Реализацију ових оператора препуштамо читаоцу.

Смислено је и предефинисати операторе поређења два датума како би се утврдило који је датум пре а који после (оператори `<` и `>`) као и да ли су два датума иста, односно различита (`==` и `!=`). Предефинисањем оператора `==` губимо његово уобичајено значење поређења референци два објекта и дефинишемо ново. У класи `Datum` операторе `==` и `!=` можемо дефинисати тако да пореде датуме по вредности атрибута.

```
public static bool operator==(Datum a, Datum b)
  {
    return a.godina==b.godina && a.mesec==b.mesec && a.dan==b.dan;
  }
public static bool operator!=(Datum a, Datum b)
  {
    return !(a==b);
  }
```

Када говоримо о било ком догађају међу најзначајнијим информацијама су информације које говоре о томе када се тај догађај одвија. Како бисмо такве информације прецизно записали неопходно је да имамо податке и о датуму и о

времену. Можемо дефинисати класу `VremenskiTrenutak` чије објекте описују атрибути D, типа `Datum` и V, типа `Vreme`. Ова класа је нарочито интересантна због оператора које можемо реализовати у њој.

- `public static VremenskiTrenutak operator +(VremenskiTrenutak a, Vreme b)`

  одређивање тренутка који је за време b после тренутка a
- `public static Vreme operator -(VremenskiTrenutak a, VremenskiTrenutak b)`

  одређивање времена између два тренутка
- `public static VremenskiTrenutak operator -( VremenskiTrenutak a, Vreme b)`

  одређивање тренутка који је за време b пре тренутка a
- `public static bool operator <( VremenskiTrenutak a, VremenskiTrenutak b)`
- `public static bool operator >(VremenskiTrenutak a, VremenskiTrenutak b)`
- `public static bool operator ==( VremenskiTrenutak a, VremenskiTrenutak b)`
- `public static bool operator !=( VremenskiTrenutak a, VremenskiTrenutak b)`

Класе `Vreme`, `Datum` и `VremenskiTrenutak` су врло употребљиве у великом броју апликација али и као атрибути бројних сложенијих класа. Рецимо да је дефинисана класа `Manifestacija` која за атрибуте има назив, место одржавања, почетак (типа `VremenskiTrenutak`) и дужину трајања (типа `Vreme`). Нека је задат распоред корисника са временом боравка у различитим местима. Можемо креирати апликацију којом се на основу доступних информација може предложити кориснику које све манифестације може да посети, тако да не поремети свој распоред. При креирању такве апликације дефинисани оператори би допринели једноставности реализације.

Предлажемо читаоцу да у потпуности реализује ове класе са свим наведеним, али и ненаведеним, а смисленим операторима као и што више апликација у којима се корисно могу употребити.

Класа Polinom

Илуструјмо дефинисање оператора у класи која као атрибут има низ. Посматрајмо класу за рад са полиномима. Полином можемо описати степеном и низом коефицијената, или низом монома код којих су коефицијенти различити од нуле. Ради једноставније реализације у наведеној класи полином је описан степеном (n) и низом коефицијената (k) тако да i-ти елемент низа k (`k[i]`) представља коефицијент уз i-ти степен полинома. У класи `Polinom` можемо дефинисати операторе (`+, -, *, /`) за извођење основних аритметичких операција над објектима класе.

```
public class Polinom
  {
    float[] k;//  низ коефицијената полинома
    int n; //  степен полинома
    //  подразумевани конструктор за формирање нула полинома
    public Polinom()
    {
      n = 0;
      k = new float[1];
      k[0] = 0;
    }
    //  конструктор којим се формира полином степена n
    //  са коефицијентима једнаким 0
    public Polinom(int n)
    {
      this.n = n;
      k = new float[n+1];
      for(int i=0;i<=n;i++) k[i] = 0;
    }
    //  приватни индексер уведен ради лаксег приступа коефицијентима полинома
    private float this[int i]
    {
      get { return k[i]; }
      set { k[i] = value; }
    }
    public static Polinom operator +(Polinom a, Polinom b)
    {
      Polinom c = new Polinom(Math.Max(a.n, b.n));
      for (int i = 0; i <= c.n; i++)
    {
      //c.k[i] = 0;  ово је већ урађено конструктором
      if (i <= a.n) c[i] += a[i]; // c.k[i] += a.k[i];
      if (i <= b.n) c[i] += b[i];
    }
    int n1=c.n;
    while (n1 > 0 && c[n1] = = 0) n1--;
    c.n = n1;
    return c;
  }
public static Polinom operator *(Polinom a, Polinom b)
  {
    Polinom c = new Polinom(a.n + b.n);
    for (int i = 0; i <= a.n; i++)
      for (int j = 0; j <= b.n; j++)
        c[i + j] += a[i] * b[j];
```

```
    return c;
  }
}
```

Наводимо неколико интересантних класа на којима можемо илустровати операторе:

- класа за рад са комплексним бројевима (обезбедити и графички приказ објекта класе). У оквиру те класе могу се реализовати оператори за сабирање, одузимање, множење, дељење два комплексна броја, одређивање конјугованог комплексног броја (унарни оператор нпр. ~), одређивање модула комплексног броја, оператор имплицитне конверзије из реалног у комплексни број.
- класа за рад са векторима (обезбедити и графички приказ објеката класе)
- класа којом реализујемо аритметику великих бројева (бројева са произвољно много цифара). Могу се постепено развијати класе за рад са великим бројевима полазећи од природних бројева, затим развој класе за рад са целим бројевима и на крају са великим реалним бројевима. Велики цео број можемо описати знаком и низом цифара (знак и апсолутна вредност броја), а можемо и низом цифара који представља запис броја у потпуном комплементу основе 10. У овим класама природно је реализовати све аритметичке операторе као и операторе поређења.
- класа за рад са скуповима (реализовати операторе унија, пресек, разлика).

Математичка гимназија, Краљице Наталије 37, Београд