VIŠELINIJSKI STRINGOVI

U Javi ako želimo da pišemo string u više linija to možemo uraditi na sledeći način:

```
String sql = "SELECT * " +
             "FROM Korisnik " +
             "WHERE ID = " + id ;
```

Medutim C# ima ugrađenu sintaksu za višelinijske stringove:

```
string sql = @"SELECT *
                FROM Korisnik
                WHERE ID = " + id ;
```

Ako string literal počne sa @ on se može prelamati u nove redove sve do prvog znaka ". U slučaju da se pojavi navodnik u stringu on se mora napisati u vidu dva uzastopna navodnika "".

SVOJSTVA

Svojstva (Properties) predstavljaju nove elemente u C# jeziku koji se koriste radi lakše kontrole pristupa polja klase. Primer klase Datum sa tri svojstva za dan, mesec i godinu je prikazan u sledećem primeru:

```
public class Datum
{
    public int Dan { get; set; }
    public int Mesec { get; set; }
    public int Godina { get; set; }
}
```

Kao što možemo videti svojstva su veoma slična poljima - ona predstavljaju podatke u okviru klase. U vitičastim zagradama je definisano da se ova dva polja mogu čitati (get) i postavljati (set). Medutim nad poljima možemo imati bolju kontrolu načina na koji se polja koriste. Na primer, možemo lako definisati da je postavljanje polja privatno dok je čitanje javno kao što je prikazano u sledećem primeru:

```
public class Datum
{
    public int Dan { get; private set; }
    public int Mesec { get; private set; }
    public int Godina { get; private set; }
}
```

Ovo je isto moguće uraditi u Javi, s tim što bi se napravile posebne **metode** (getter i setter) gde bi jedna bila javna a druga privatna. Pored toga, možemo lako povezati svojstva sa poljima i definisati šta treba uraditi kada se neko polje čita ili upisuje, eventualno staviti neke validacije prilikom upisa i slično. Ovo se lako može postići tako što definišemo blokove koda koji će biti izvršeni kad se postavi ili čita svojstvo.

PRIMER 1

```
public class Datum
{
     private int dan;

     public int Dan
     {
        get
        {
            return this.dan;
        }
```

```
            set
            {
                if ( 0 < value && value <= 31)
                    this.dan = value;
            }

        public int Mesec { get; private set; }
        public int Godina { get; private set; }
    }
```

U ovom primeru je definisano polje dan u kome se nalazi konkretna informacija. Ovo polje je privatno i niko ga ne može čitati ni postavljati direktno. Jedini način da se pristupi ovom polju je indirektno preko get i set operacija svojstva Dan. Kada neko pokuša da pročita svojstvo Dan, kod u get operaciji će samo vratiti vrednost promenljive dan. Kad neko pokuša da upiše vrednost u svojstvo Dan, umesto da se vrednost upiše, izvršiće se kod koji proverava da li je nova vrednost izmadu 1 i 31, i ako jeste biće upisana u promenljivu dan, inače se ništa neće desiti. Svojstva su elegantan način da enkapsulirate polja bez korišćenja posebnih metoda za pristup.

PRIMER 2

```
class VremenskiPeriod
{
    private double sekunde;

    public double Sati
    {
        get { return sekunde / 3600; }
        set { sekunde = value * 3600; }
    }

    public double Sekunde
    {
        get { return sekunde; }
        set { sekunde = value; }
    }
}


class Program
{
    static void Main()
    {
        VremenskiPeriod t = new VremenskiPeriod();

        // Assigning the Hours property causes the 'set' accessor to be called.
        t.Sati = 24;

        // Evaluating the Hours property causes the 'get' accessor to be called.
        System.Console.WriteLine("Vreme u satima: " + t.Sati);
        System.Console.WriteLine("Vreme u sekundama: " + t.Sekunde);

        t.Sekunde = 3600;
        System.Console.WriteLine("Vreme u satima: " + t.Sati);
        System.Console.WriteLine("Vreme u sekundama: " + t.Sekunde);
    }
}
```

Primer ilustruje klasu VremenskiPeriod koja vreme interno čuva u sekundama. Vreme se može setovati ili preuzeti preko dva svojstva: **Sekunde** ili **Sati**. Prilikom pozivanja željenog svojstva, u pozadini se izvršava kod koji vrši odgovarajući konverziju (pretvara sekunde u sate, pretvara sate u sekunde…).

PARCIJALNE KLASE

C# omogućava da programski kod klasa podelite u više fizičkih fajlova. Na primer možete napisati deo klase Datum u fajlu Datum.polja.cs kao u sledećem primeru:

```
public partial class Datum
{
     public int Dan { get; private set; }
     public int Mesec { get; private set; }
     public int Godina { get; private set; }
}
```

Jedina razlika je u tome što je dodat modifikator **partial** koji govori kompajleru da je ovo samo deo klase. Drugi deo klase koji sadrži metode možete staviti u drugi fajl na primer Datum.metode.cs kao u sledećem primeru:

```
public partial class Datum
{
   public string ToString() { return this.Dan + "/" + this.Mesec + "/" +
   this.Godina; }
}
```

Iako je ovaj deo klase u drugom fajlu, u njegovim metodama možete da koristite sva svojstva, polja i metode iz drugih parcijalnih klasa. Kada kompajler prevede kod, pronaći će sve parcijalne klase u istom imenu prostora i spojiti ih.

Parcijalne klase se veoma često koriste ako imate delove klase koji se automatski generišu i druge delove koji se pišu ručno. U tom slučaju se generisani kod stavlja u jedan fajl koji sadrži parcijalnu klasu, dok se kod koji je ručno dodat stavlja u drugi fajl. Kompajler će spojiti ova dva fajla.

NOVI TIPOVI ARGUMENATA METODA

Kada u Javi napravite funkciju koja prima neke parametre, svi prosti tipovi se predaju "po vrednosti" dok se objekti predaju po referenci. To znaci da ako metoda prihvata parametar tipa int ona ga ne može promeniti. Metoda može promeniti samo argument koji je nekakav objekat.

U C# jeziku su identična pravila uz razliku da se prosti tipovi mogu pozivati po referneci, a čak se mogu definisati kao izlazne promenljive. Samo je potrebno dodati modifikatore **ref** i **out** u potpisu metode. Primer takve metode je dat u sledećem listingu:

```
public void Funkcija(ref int x, out int y)
{
    x = (x + 1)/2;
    y = x*2;
}
```

Pošto je prvi broj pozvan po referenci, njegova vrednost se može čitati ali i upisivati. Izlazni parametar se može samo upisivati. Primer poziva ove metode je dat u sledećem kodu:

```
Funkcija(ref a, out b);
```

Naravno ovde postoji jedno ograničenje - argumenti funkcije ne mogu da budu konstante pošto se u njih vrednosti ne mogu upisati.

PRIMER 1

```
class Program
{
    public static void Funkcija(ref int x, out int y)
    {
        x = (x + 1) / 2;
        y = x * 2;
    }

    static void Main()
    {
        int x = 10;
        int y;

        Funkcija(ref x, out y);

        System.Console.WriteLine("x = " + x + ", y = " + y);
    }
}
```

POZIVI FUNKCIJA PO IMENIMA PARAMETARA

Jedana korisna stvar koja je uvedena u C# jeziku je mogućnost da pozivate funkcije tako što im predate argumente po imenu. Na primer ako posmatrate standardnu funkciju za stepenovanje Math.Pow koja ima dva argumenta x - osnovu i y - stepen. Možete je pozvati na neki od sledećin načina:

```
Math.Pow(2, 5);
Math.Pow(x:2, y:5);
Math.Pow(y:5, x:2);
```

Prvi poziv je standardan poziv gde su argumenti predati po poziciji. U drugom pozivu je eksplicitno sedifnisano da je x 2 a y 5. Kada ovako koristite poziv, možete i da obrnete pozicije argumenata kao u trećem primeru.

Zašto bi ovo koristili? Zamislite da imate funkciju koja ima dosta parametara od kojih su u nekim slučajevima većina nule ili null. Nekada bi vam bilo tečko da pronađete gde se nalazi peti parametar tako da ovom sintaksom dobijate čitljiviji kod.

NOVI MODIFIKATOR - INTERNAL

I C# i Java imaju standardne modifikatore klasa public, private i protected. C# uvodi jedan dodatni modifikator internal. Ako klasa ima internal modifikator mogu joj pristupiti klase koje se nalaze u istom dll ili exe fajlu. Ne postoji ekvivalent u Javi pošto tamo ne postoji koncept biblioteke.

ENUMERACIJE

Enumeracije predstavljaju grupe konstanti koje koristite u kodu. Primer jedne enumeracije je prikazan u sledećem listingu:

```
enum Dani {Pon, Uto, Sre, Cet, Pet, Sub, Ned};
```

U kodu možete koristiti ove konstante kao Dan.Uto ili Dan.Sub. Ovo je sintaksa koju podržavaju i Java i C#.

Podrazumevano ovo su int konstante koje počinju od broja 0. Naravno i ovo se može promeniti tako što možete definisati neki drugi tip kao što je je byte i podesite da konstante ne počnu od nule nego od neke druge početne vrednosti (npr. od jedinice). Primer je prikazan u sledećem listingu:

```
enum Dani: byte { Pon=1, Uto, Sre, Cet=, Pet, Sub, Ned};
```

AUTOMATSKO UTVRĐIVANJE TIPOVA

Ovo su standardni izrazi u Javi:

```
int x = 1;
Osoba o = new Osoba();
```

Ako malo bolje razmislite o ovim konstrukcijama možete da zaključite da ovde ponekad može da bude viška informacija. Na primer, odmah se vidi da je x ceo broj na osnovu broja 1 koji mu se dodeljuje. Isto tako, objekat o mora da bude tipa Osoba da bi mu se dodelio objekat iz konstruktora - zašto se onda taj tip ponavlja i u deklaraciji? C# uvodi ključnu rec **var** kojom ne morate da ponavljate ove informacije. Iako je ovo takođe i validan C# kod, često se piše ovako:

```
var x = 1;
var o = new Osoba();
```

Kada se stavi ključna rec **var**, kompajler će videti koji je tip izraza sa desne strane i taj tip će koristiti kao tip promenljive na levoj strani. Primetite da ovo ne znači da objekti nisu tipizirani - oni uvek imaju svoj tip samo se prepušta kompajleru da ga utvrdi na osnovu desne strane.

ANONIMNI TIPOVI

C# vam omogućava da koristite objekte čije klase nisu statički predefinisane u nekim fajlovima. Umesto toga, možete dinamički kreirati objekte tako što ćete im samo definisati svojstva. Kada se kreira objekat anonimnog tipa umesto standardnog konstruktora u obliku **new** <<ime tipa>> se koristi samo **new**. Primer korišćenja anonimnih tipova je prikazan u sledećem listingu:

```
var tacka = new { X = 4, Y = 7, Z = -2 };
Console.WriteLine( tacka.X );
```

U ovom primeru je dinamički kreiran objekat koji ima polja X, Y i Z. Takav objekat je dodeljen promenljivoj tačka. Medutim klasa tačka nije definisana nigde u kodu - ova klasa se dinamički kreira kada se kreira objekat ali se može koristiti kao i svaka druga klasa. Jedina razlika je to što se ne može koristiti njeno ime. Obratite pažnju da u ovom slucaju moramo da koristimo automatsko utvrdivanje tipova (sa **var** ključnom reči) kada deklarišemo promenljivu tačka. Pošto klasa koja definiše tačku ne postoji, jedini način je da se prepusti kompajleru da "zaključi" da treba da kreira novi tip za promenljivu tačka kada vidi šta je rezultat izraza na desnoj strani.