

Objektno orijentisano programiranje 2

Jezik C#



Primer programa

- Program se sastoji od definicije tipova unutar odgovarajućih prostora imena

- Primer:

```
using System;
public class Pozdrav{
    public static void Main(){
        Console.WriteLine("Zdravo");
    }
}
```

- Razlike u odnosu na *Javu*:

- obavezno eksplicitno korišćenje prostora imena `System`
 - za razliku od paketa `java.lang` čiji se tipovi automatski uvoze
- `Console` je klasa za rad sa standardnim tekstualnim izlazom (konzolom)
 - `WriteLine` je statički metod za pisanje linije na standardnom izlazu
 - u *Javi* `System.out.println()` – gde je `out` statički objekat klase `System`
- ime glavnog programa je `Main`, sa velikim `M` i argumenti nisu neophodni

Glavni program

- Postoje 4 potpisa metoda koji predstavlja glavni program:

```
static void Main()  
static void Main(String[] argumenti)  
static int Main()  
static int Main(String[] argumenti)
```

- Modifikator `public` nije neophodan
 - ulazna tačka je uvek pristupačna za CLR
- Rezultat tipa `int` predstavlja statusni kod završetka
 - kod se vraća izvršnom okruženju
 - `Main` metodi koji vraćaju `void` – podrazumevano vraćaju 0
- Jedna klasa može imati samo jedan `Main` metod
- Ako više od jedne klase sadrži `Main` metod
 - prilikom prevođenja se pomoću `/main` opcije određuje ulazna tačka aplikacije
 - kao parametar se navodi klasa čiji se `Main` koristi

Komentari

- Podržane su sledeće vrste komentara:
 - višelinijski komentari `/* . . . */`
 - komentari u jednoj liniji `//`
 - dokumentacioni višelinijski komentar `/** . . . */`
 - dokumentacioni komentar u jednoj liniji `///`
- Dokumentacioni komentari se izdvajaju tokom prevođenja
 - potrebna opcija prevođenja `/doc`
 - komentari formiraju XML fajl
 - proizvedeni XML fajl se može transformisati u razne formate
 - XSL alat → HTML

Leksički stil

- Mala i velika slova se razlikuju
 - ali se ne preporučuje razlikovanje identifikatora samo na osnovu razlike u veličini slova
 - razlog: kombinovanje tipova sa tipovima definisanim u jezicima koji ne razlikuju mala i velika slova
- Izvorni kod se piše
 - na kodnoj stranici Latin-1 ili
 - na UTF-8 (Unicode Transformation Format)
 - Unicode znak se kodira sa 1-4 okteta
 - 1 oktet za ASCII znake
 - 2 okteta za posebne latinične znake, ćirilicu, grčki, hebrejski, arapski,...
 - 3 okteta za kineski, japanski, koreanski,...

Identifikatori i ključne reči

- Mora početi slovom ili znakom `_`, znak `$` nije dozvoljen (kao u Javi)
- Ostatak identifikatora – slova ili cifre (Unicode znaci) i znak `_`
- Ključne reči se mogu koristiti kao identifikatori dodavanjem znaka `@` ispred
- Veliki broj ključnih reči zajednički sa Javom
- Ne koriste se:
`import`, `package`, `boolean`, `extends`, `implements`,
`instanceof`, `super`, `final`, `native`, `strictfp`,
`synchronized`, `transient`, `assert`, `throws`
- Nove:
`using`, `namespace`, `bool`, `sbyte`, `ushort`, `uint`, `ulong`,
`decimal`, `delegate`, `event`, `object`, `operator`, `string`,
`struct`, `params`, `out`, `ref`, `as`, `is`, `typeof`, `base`,
`explicit`, `implicit`, `internal`, `sealed`, `override`,
`virtual`, `readonly`, `extern`, `lock`, `foreach`, `in`, `checked`,
`unchecked`, `fixed`, `sizeof`, `stackalloc`, `unsafe`

Operatori

- U Javi postoji 44 operatora, a u C# postoji 48 operatora (+5 nebezbednih)
- C# omogućava i preklapanje operatora (slično kao C++)
- Ne koriste se operatori iz Java: `>>>`, `>>>=`, `instanceof`
- Novi bezbedni: `true`, `false`, `is`, `as`, `typeof`, `checked`, `unchecked`
- Novi nebezbedni (preuzeti iz C++): `sizeof`, `*`, `->`, `[]`, `&`
- Logički `true` i `false` su i operatori i literali
 - kao operatori se mogu preklapati
- Operator `is` je ekvivalent `instanceof` operatora u Javi
- Operator `as` vrši konverziju tipa (*cast*) ali vraća `null` ako ne uspe, ne baca izuzetak
 - na primer: `string s = refObj as string;`
- Operator `typeof` vraća objekat tipa `System.Type`
 - ekvivalent za statički metod `Class.forName` iz Java
- Operatori za kontrolu bacanja izuzetaka na prekoračenje: `checked`, `unchecked`
- Delegatske operacije: `+` (dodavanje) i `-` (uklanjanje) delegata
- Operator `sizeof`: vraća broj zauzetih bajtova za vrednosni tip (na steku)
- Operacije sa pokazivačima: `*`, `->`, `[]`, `&`

Prostori imena

- Koncept preuzet iz C++, alternativa Javinim paketima
- Organizacija tipova u logički hijerarhijsku strukturu globalno jedinstvenih imena
- Članovi:
 - klase, strukture, nabranja, interfejsi, delegati i prostori imena
- Za spoljašnji prostor imena se može koristiti hijerarhijska notacija sa tačkom:
`namespace Rs.Ac.Bg.Etf.MojProstor {...}`
- Za unutrašnje prostore imena nije dozvoljena notacija sa tačkom:
`namespace NajvisiNivo{
 namespace DrugiNivo{
 namespace TreciNivo{...}}}
namespace NajvisiNivo{
 namespace DrugiNivo.TreciNivo{...}} //GRESKA`
- Ako se ne navede prostor imena – članovi pripadaju globalnom prostoru
- Prostori imena implicitno imaju javni pristup
 - nisu dozvoljeni modifikatori za pravo pristupa

Korišćenje prostora imena

- Korišćenje tipova iz prostora imena – deklaracija `using`
 - `using prostor//` koriscenje svih tipova iz prostora
 - `using prostor.tip //` koriscenje jednog tipa
- Ako se koristi više prostora imena sa istim imenima tipova
 - potrebna puna kvalifikacija
- C# omogućava nadimke (aliase) za punu kvalifikaciju imena:

```
using Moje=Rs.Ac.Bg.Etf.MojProstor;
Moje.X x;
using MojeY=Rs.Ac.Bg.Etf.MojProstor.Y;
MojeY y;
```
- Doseg deklaracije `using` je unutar prostora imena u kojem je data
 - različiti prostori imena unutar jednog fajla mogu da koriste različite skupove prostora imena
 - svaki prostor imena može da koristi svoje posebne aliase

Naredbe

- Većina je zajedničkih za Javu i C#
- Sve naredbe se završavaju terminacionim simbolom ;
- Prazna naredba ima samo terminacioni simbol (i komentar – dobra praksa)
- Blokovi naredbi se pišu u vitičastim zagradama { }
- Labele se pišu ispred naredbi i završavaju se simbolom :
 - doseg labele – unutar bloka u kojem je deklarirana i ugnježdenih blokova
- U Javi dozvoljeni samo `break` i `continue` skokovi na labele, a u C# ponovo i `goto`, dok `break` i `continue` nemaju labele
- Naredbe `if...else` `if...else`, `while`, `do` i `for` su iste kao u Javi
- Razlike su u `switch` konstrukciji, skokovima i `foreach` petlji

Selekcija switch

- U početku C# podržavao širi opseg tipova izraza za selekciju od Java
 - ugrađeni celobrojni tipovi, nabrojanja, znakovni, niske (stringovi)
- U aktuelnoj verziji Java, praktično je isti opseg, pri čemu Java podržava i:
 - Character, Byte, Short, Integer
- C# ne podržava propadanje iz jedne grane u narednu
 - ako se grana ne završava sa goto ili break – greška u prevođenju
 - sa goto je moguće skočiti čak na drugu granu iste switch konstrukcije
- C# podržava da se za više vrednosti izraza izvršava ista grana
- Primer:

```
public void Metod(string boja){
    switch(boja.ToLower()){
        case "zuta": case "crvena": MetodA(); goto default;
        case "plava": MetodB(); break;
        case "zelena": MetodC(); goto case "crvena";
        default: MetodX(); break;
    }
}
```

Naredbe skoka

- Naredba `return` se ponaša isto kao u Javi
- Naredbe `break` i `continue`
 - ne mogu imati kao parametar labelu na koju se skače
- Naredba `goto`
 - predstavlja bezuslovni skok na naredbu sa zadatom labelom
 - može se skakati unapred ili unazad ali `goto` mora biti u doseg labela
 - to omogućava skokove unutar bloka i iskakanja iz ugnežđenih blokova
 - nije moguće uskakanje u neki blok
 - iskakanje iz `try` ili `catch` bloka izaziva izvršenje `finally` bloka
 - iskakanje iz `finally` bloka izaziva grešku u fazi prevođenja

Petlja foreach

- Nova konstrukcija namenjena iteriranju kroz elemente kolekcije
- Kolekcija mora da implementira interfejs `System. IEnumerable`
 - nizovi implementiraju `System. IEnumerable`, pa se kroz njih može iterirati
- Sintaksa:
`foreach(<tip> <identifikator> in <kolekcija>) <naredba>`
- Indeks iteracije (iterator) je promenljiva koja se može samo čitati u petlji
- Iterator mora biti istog tipa kao element kolekcije
- Primer:
iteriranje kroz niz stringova uz ispisivanje svakog elementa na konzoli

```
string[] dani=new string[]  
{ "Ponedljak", "Utorak", "Sreda", "Cetvrtak", "Petak" };  
foreach (string s in dani)System.Console.WriteLine(s);
```

Naredba zaključavanja

- Ključna reč `lock` odgovara ključnoj reči `synchronized` u Javi
- U C# se `lock` ne može koristiti kao modifikator metoda
 - može se koristiti samo kao naredba
- Primer:

```
public void Metod(){
    lock (NekiObjekat){
        // naredbe koje zahtevaju sinhronizovani
        // pristup objektu NekiObjekat
    }
}
```

Naredbe za kontrolu prekoračenja

- Prekoračenje nastupa
 - kada je rezultat celobrojne aritmetičke operacije veći (+) ili manji (-) od odredišta
- Na Javi se prekoračenje ne kontroliše: odsecaju se najviši bitovi (efekat arit. kruga)
- Na primer:

```
short x=32767; x++; // x== -32768
```
- Na C# se koriste ključne reči `checked` i `unchecked` za kontrolu prekoračenja
- Ove ključne reči se koriste i kao operatori i kao naredbe
- Primeri operacije i naredbe sa kontrolisanim prekoračenjem:

```
int b = checked (x*5);  
checked { int p=x++; int q=x*5; int r=p*q; }
```
- Ako se otkrije prekoračenje
 - u vreme prevođenja – javlja se greška
 - u vreme izvršenja – baca se `System.OverflowException`
- Korišćenje ključne reči `unchecked` sprečava proveru prekoračenja
- Ako se ne koriste ove ključne reči sledeća pravila se podrazumevju:
 - za konstantne vrednosti primenjuje se kontrola prekoračenja u vreme prevođenja
 - za promenljive vrednosti se dopušta prekoračenje (vrši se odsecanje najviših bita)

Naredba using

- Ključna reč se koristi i kao direktiva i kao naredba (nisu povezane)
- Naredba se koristi
 - za eksplicitno upravljanje oslobađanjem resursa koje je neki objekat alocirao
- Klase i strukture kojima upravlja naredba `using`
 - moraju da implementiraju `IDisposable`
 - interfejs `IDisposable` deklariše samo metod `Dispose()`
 - metod `Dispose()` obavlja oslobađanje neupravljanih (*unmanaged*) resursa
 - resursi su npr. fontovi, mrežne konekcije, ...
- Primer:

```
using(Klasa x=new Klasa(), Struktura y=new Struktura())
{x.M();y.M();}
```

 - na kraju bloka se automatski poziva `Dispose()` metod za `x` i za `y`
 - Klasa i Struktura moraju da implementiraju `IDisposable`