Programski jezik C#

Izjeme

Matija Lokar in Srečo Uranič

V 0.8

november 2008

Predgovor

Omenjeno gradivo pokriva delo z izjemami v C# .

Gradivo vsekakor ni dokončano in predstavlja delovno različico. V njem so zagotovo napake (upava, da čimmanj), za katere se vnaprej opravičujeva. Da bo lažje spremljati spremembe, obstaja razdelek Zgodovina sprememb, kamor bova vpisovala spremembe med eno in drugo različico. Tako bo nekomu, ki si je prenesel starejšo različico, lažje ugotoviti, kaj je bilo v novi različici spremenjeno.

Matija Lokar in Srečo Uranič

Kranj, november 2008

Zgodovina sprememb

14. 11. 2008: Različica V0.8 – le prenos Sašovega gradiva. Premisliti je še potrebno kaj in kako.

1. 12. 2008: Različica V0.81 – potrebno je še dodati kako vajo, zgled, ... Manjka še oblikovanje.

KAZALO

Varovalni bloki (osnove) – obravnava napak in izjem (Exceptions) ... 6

Blok za obravnavo prekinitev: try ... catch 8

Večkratni varovalni blok za obravnavo prekinitev try ... catch ... catch 10

Brezpogojni varovalni blok try ... finally 11

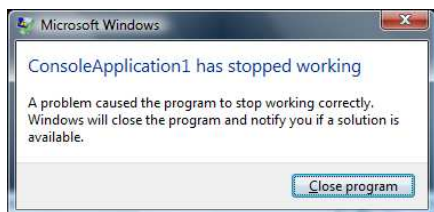
Varovalni bloki (osnove) – obravnava napak in izjem (Exceptions)

Tako kot vsakdanjem življenju se tudi pri delovanju programov pojavljajo napake. Zelo neprijetno je, če se program sredi delovanja "sesuje" (neha delovati), ker na primer programer ni predvidel, da lahko v določenih izjemnih primerih pride do "čudne" situacije. Na primer, da se uporabi negativni indeks, ali pa da datoteka, kamor naj bi se zapisali rezultati, ne obstaja, da uporabnik kot število, s katerim naj se deli, po pomoti vnese 0 in podobno. Dobro je, če imamo možnost, da v primeru napak program ustrezno reagiramo, pa če drugega ne, izpiše prijazno obvestilo, da je žal prišlo do take in take napake in da bo zaradi tega program nehal z delovanjem.
C#, podobno kot drugi sodobni programski jeziki, pozna tako imenovan mehanizem **izjem** (**exceptions**). Če pri delovanju programa pride do napake, se sproži tako imenovana izjema. To izjemo lahko programsko prestrežemo in napišemo ustrezno kodo, kako naj program nadaljuje v tem primeru.

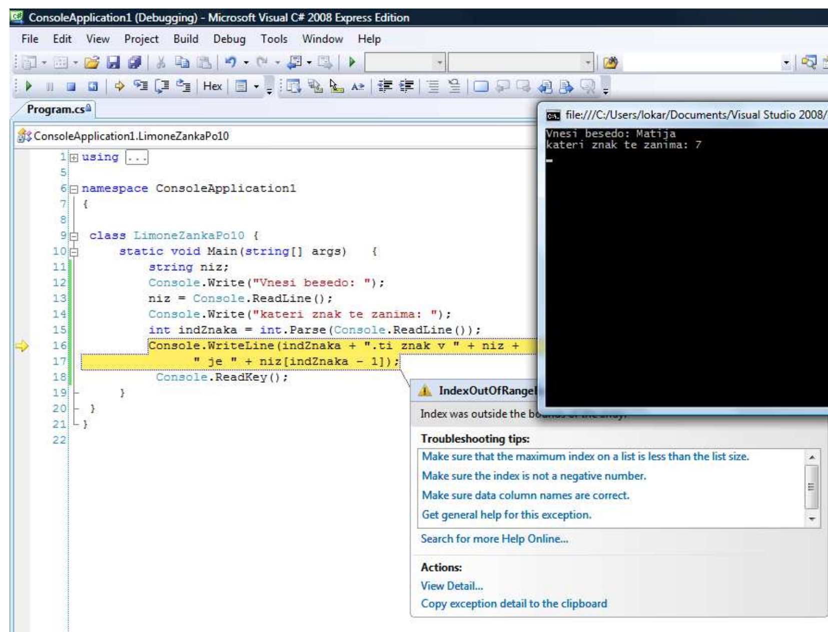
Poglejmo si primer, Napisali smo program, ki prebere niz in uporabniku omogoči, da vnese indeks znaka, ki ga zanima.

```
static void Main(string[] args)   {
    string niz;
    Console.Write("Vnesi besedo: ");
    niz = Console.ReadLine();
    Console.Write("kateri znak te zanima: ");
    int indZnaka = int.Parse(Console.ReadLine());
    Console.WriteLine(indZnaka + ".ti znak v " + niz +
          " je " + niz[indZnaka - 1]);
}
```

Če program poženemo in vpišemo neapačen (na primer prevelik) indeks, program neha delovati. Če smo poganjali program neposredno (torej EXE), dobimo le obvestilo operacijskega sistema. V primeru OS Vista je to npr. takole



Če pa program poženemo znotraj razvojnega okolja Visual C# 2008 Express Edition, nas okolje postavi v "problematično" vrstico in javi, da je prišlo do izjeme IndexOutOfBounds.



Sedaj bi lahko kodo spremenili tako, da bi preverili ustreznost podatkov in ugotovili ali lahko pride do napake.

```
...
int indZnaka = int.Parse(Console.ReadLine());
if ((0 <= indZnaka) && (indZnaka <= niz.Length())){
     Console.WriteLine(indZnaka + ".ti znak v " + niz +
           " je " + niz[indZnaka - 1]);
} else {
    Console.WriteLine(niz + " nima " + indZnaka
        + "tega znaka");
}
```

Vendar se na ta način tok programa in koda za obdelavo napak prepletata. Poleg tega lahko včasih pride tudi do napak, ki jih je težko uspešno preprečiti s preverjanjem določenih pogojev ali pa so ti zelo zapleteni. V našem primeru denimo, lahko do napake pride tudi, če uporabnik kot indeks znaka vnese nekaj, kar ni celo število.

Zato je ideja izjem v tem, da dele programa, ki so "kritični" obdamo z varovalnim blokom. Če v tem varovalnem bloku pride do napake, se izvede lovilni del, kjer lahko ob napakah ustrezno reagiramo.

Idejna rešitev za obdelavo izjem je torej v tem, da ločimo kodo, ki predstavlja tok programa in kodo za obdelavo napak. Na ta način postaneta obe kodi lažji za razumevanje, saj se ne prepletata.

Za obdelavo izjem pozna **C#** naslednje bloke:

- blok za obravnavo prekinitev **try…catch …**,

- večkratni varovalni blok **try … catch … catch …**
- brezpogojni varovalni blok **try … finally …** .

Blok za obravnavo prekinitev: try … catch …

Kodo, ki bi jo sicer napisali v delu programa ali pa npr. v neki metodi, zapišemo v varovalnem bloku **try** (blok je vsak del kode napisane med oklepajema { in }). Drugi del začenja besedica **catch** in v bloku zapišemo enega ali več stavkov za obdelavo izjem oz. napak (t.i. ***catch* handlers**). Če katerikoli stavek znotraj bloka **try** povzroči izjemo (če pride do napake), se normalni tok izvajanja programa prekine (normalni tok izvajanja pomeni, da se posamezni stavki izvajajo od leve proti desni, stavki pa se izvajajo eden za drugim, od vrha do dna), program pa se nadaljuje v bloku **catch**, v katerem pa moramo seveda napako ustrezno obdelati.

Primer:

```
while (napaka) {
 try
 {
   int levo = int.Parse(Console.ReadLine());
   int desno = int.Parse(Console.ReadLine());
   double rezultat = levo / desno;
   Console.WriteLine(rezultat);
   napaka = false;
 }
 catch
 {
   Console.WriteLine("Ponovno vnesi podatke");
   napaka = true;
 }
```

običajna programska koda (try blok)

koda za obdelavo napake (catch blok)

Metoda `int.Parse` skuša napraviti ustrezni pretvorbi iz niza v celo število, pri čemer pa seveda lahko pride do napake (npr. če uporabnik vnese znak, ki je različen od znakov '0' do '9'). Do napake lahko pride tudi v stavku, kjer je operacija za deljenje, saj se lahko zgodi, da je drugi operand enak 0. Varovalni blok take napake prestreže in izvede se stavek (oz. stavki) v bloku **catch**.

Če v stavkih znotraj bloka **try** pride do napake (izjeme), se program na tem mestu ne "sesuje". Ostali stavki znotraj bloka **try** se ne izvedejo. Začnejo pa se izvajati stavki za obdelavo izjem (napak), ki so znotraj bloka **catch**. Če pa do napake v bloku **try** ne pride, se stavki v bloku **catch** NE izvedejo nikoli!

Varovalni blok je torej zgrajen takole:

```
try
```

```
{
   // stavki, kjer lahko pride do napake
}
catch
{
   // obdelava napake . . .
}
```

Jezik C# ponuja tudi možnost, da tudi ugotovimo, do kakšne izjeme je prišlo in potem reagiramo glede na vrsto izjeme.

Primer:

```
try
{
   int levo = int.Parse(Console.ReadLine());
   int desno = int.Parse(Console.ReadLine());
   double rezultat = levo / desno;
   Console.WriteLine(rezultat);
}
catch (System.Exception napaka)
{
   // Vrsto napake izpišemo na zaslonu
   Console.WriteLine(napaka.Message);
}
```

} običajna programska koda

} koda za obdelavo napake

V zgornjem primeru je uporabljena kar **splošna** metoda za prestrezanje napake (`System.Exception`); le-ta se odzove/odkrije na vsako izjemo (napako), tudi tako, ki se je morda sploh ne zavedamo. V takem primeru lahko blok **catch** uporabimo tudi brez parametrov takole:

```
try
{
   // stavki, kjer lahko pride do napake
}
catch
{
   // obdelava napake . . .
}
```

Seveda pa obstajajo tudi metode za obdelavo posameznih vrst napak: **`System.FormatException`** (to metodo uporabimo na primer za obdelavo uporabnikovih napačnih vnosov podatkov), **`System.DivideByZeroException`**, **`System.EArgument.Exception`**, …) . Vsaka od teh metod vrne ustrezno obvestilo o napaki, ki ga lahko izpišemo v konzolnem. Obvestilo o napaki je v tem primeru v angleščini; v kolikor pa želimo uporabnikom ponuditi prijazna obvestila o napaki, jih je potrebno napisati posebej, npr. takole:

Izjeme lahko prožimo tudi sami programsko. Vendar se v začetnem spoznavanju jezika C# z vsem tem ne bomo ubadali.

Zgled, ki smo ga navedli na začetku, dajmo v varovalni blok.

```
static void Main(string[] args)
{
    string niz;
```

```
        Console.Write("Vnesi besedo: ");
        niz = Console.ReadLine();
        try
        {
            Console.Write("kateri znak te zanima: ");
            int indZnaka = int.Parse(Console.ReadLine());
            Console.WriteLine(indZnaka + ".ti znak v " + niz +
                      " je " + niz[indZnaka - 1]);
        }
        catch
        {
          Console.WriteLine("Bodisi nisi vnesel števila " +
                 "ali pa je to število preveliko/premajhno");
        }
    }
```

Sedaj se naš program uspešno zaključi ne glede na to, ali vnesemo napačen indeks

```
file:///C:/Users/lokar/Documents/Visual Studio 2008/Projects/ConsoleApplication1/C.
Vnesi besedo: matija
kateri znak te zanima: 9
Bodisi nisi vnesel števila ali pa je to število preveliko/premajhno
```

ali pa za indeks ne sploh vnesemo števila

```
file:///C:/Users/lokar/Documents/Visual Studio 2008/Projects/ConsoleApplication1/C..
Vnesi besedo: Matija
kateri znak te zanima: ii
Bodisi nisi vnesel števila ali pa je to število preveliko/premajhno
```

Večkratni varovalni blok za obravnavo prekinitev try … catch … catch …

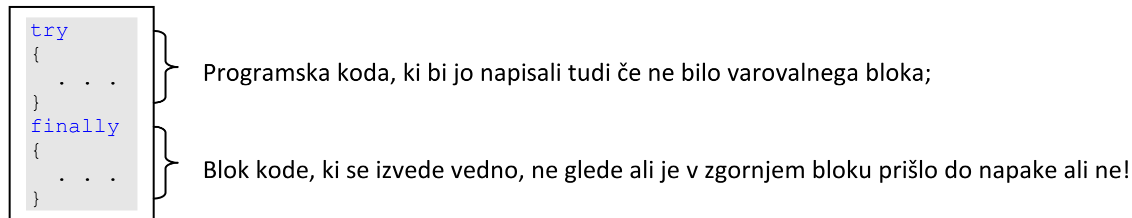
Različne napake seveda proizvedejo različne vrste izjem. Recimo, da imamo operacijo deljenja, pri kateri se lahko zgodi, da je drugi operand enak 0. Izjema, ki se pri tem zgodi, se imenuje **DivideByZeroException**. V takem primeru lahko napišemo večkratni **catch** blok, enega za drugim. Najprej ujamemo najnižji razred izjem, nazadnje pa najvišjega:

```
try
{
   int levo = int.Parse(Console.ReadLine());
   int desno = int.Parse(Console.ReadLine());
   double rezultat = levo / desno;
   Console.WriteLine(rezultat);
}
catch (System.FormatException napaka)
{
   Console.WriteLine("Napaka pri pretvarjanju podatkov!!!");
}
catch (System.DivideByZeroException napaka)
{
   Console.WriteLine("Napaka pri deljenju z 0");
}
```

Seznam vseh možnih izjem dobimo, če v **Debug** meniju kliknemo opcijo **Exceptions**.

Brezpogojni varovalni blok try … finally …

Stavki v bloku **catch** se izvedejo samo ob prekinitvi (napaki), sicer pa se ne izvedejo. Kadar pa za del kode želimo, da se izvede v vsakem primeru, uporabimo brezpogojni varovalni blok **finally**.



V praksi pogosto uporabljamo tudi kombinacijo obeh metod: blok za obravnavo prekinitev vgnezdimo v brezpogojni varovalni blok:

```
try
{
   // stavki, kjer lahko pride do napake
}
catch
{
   // obdelava napake
}
finally
{
    // blok, ki se izvede vedno, ne glede  na napako
}
```

Vaja:

```
//Varovalne bloke lahko tudi gnezdimo – na ta način lahko npr. uporabnika učinkovito
seznanjamo z napakami pri vnašanju podatkov!
class Narocilo
{
    public string CustomerName;
    public DateTime datumNarocila;
    public DateTime casNarocila;
    public int SteviloEnot;
}

static void Main(string[] args)
{
   Narocilo nar1 = new Narocilo();
   Console.Write("Stranka:  ");
   string stranka = Console.ReadLine();
   try
   {
      Console.Write("Vnesi datum naročila(mm/dd/yyyy):  ");
      nar1.datumNarocila=Convert.ToDateTime(Console.ReadLine());
      try
      {
         Console.Write("Vnesi čas naročila(hh:mm AM/PM): ");
         nar1.casNarocila = Convert.ToDateTime(Console.ReadLine());
         try
         {
             Console.Write("Število enot: ");
             nar1.casNarocila = Convert.ToDateTime(Console.ReadLine());
         }
         catch
         {
             Console.WriteLine("Nepravilen vnos števila enot!");
         }
      }
```

```
        catch
        {
            Console.WriteLine("Nepravilen vnos ure naročila!");
        }
    }
    catch (FormatException)
    {
        Console.WriteLine("Nepravilen vnos datuma");
    }
}
```