Izjeme, razhroščevanje

Varovalni bloki

Razhroščevanje

Varovalni bloki

- Še tako popolni programi imajo lahko napake
  - Nepravilni uporabnikovi vnosi
  - Deljenje z 0
  - Neobstoječa datoteka
  - Indeks izven obsega

- Rešitev: mehanizem izjem – **exceptions**
  - Če pri delovanju programa pride do napake, se sproži tako imenovana izjema
  - izjemo lahko programsko prestrežemo in napišemo ustrezno kodo, kako naj program nadaljuje v tem primeru.

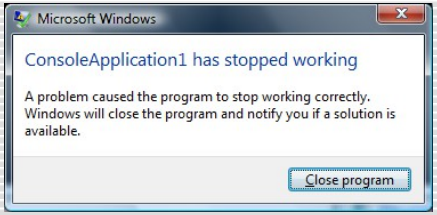
Primer izjeme

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

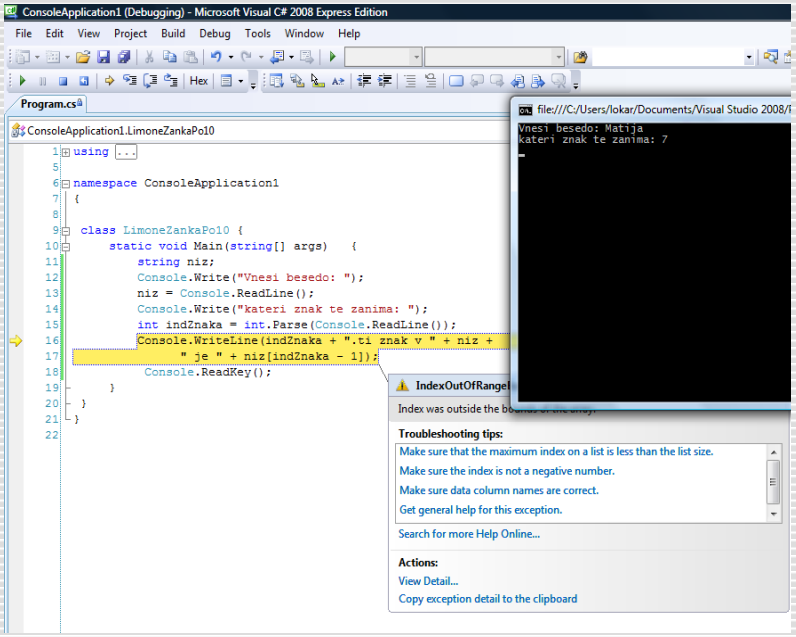
```
static void Main(string[] args)
{
        string niz;
        Console.Write("Vnesi besedo: ");
        niz = Console.ReadLine();
        Console.Write("kateri znak te zanima: ");
        int indZnaka = int.Parse(Console.ReadLine());
        Console.WriteLine(indZnaka + ".ti znak v " +
    niz +"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

Primer izjeme

- Če vpišemo napačen (na primer prevelik) indeks, program neha delovati.
  - Če smo poganjali program neposredno (.exe), dobimo le obvestilo operacijskega sistema, npr takole



  - Če pa program poženemo znotraj razvojnega okolja, nas okolje postavi v "problematično" vrstico in javi, da je prišlo do **izjeme** *IndexOutOfBounds*.



Poskus rešitve s pomočjo kode

- Kodo lahko spremenimo tako, da preverimo ustreznost podatkov in ugotovimo 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.

Prava rešitev – Varovalni blok

- Dele programa, ki so "kritični", obdamo z varovalnim blokom, npr. takole

```
try
{
   //kritični del programa: to so stavki, kjer lahko
   //pride do napake
}
catch
{
   //lovilni del, namenjen obdelavi napake
}
```

Če v varovalnem bloku (**try**) pride do napake, se izvede lovilni del (**catch**), kjer lahko ob napakah ustrezno reagiramo.

Delitev varovalnih blokov

- Za obdelavo izjem pozna **C#** naslednje bloke:
  - blok za obravnavo prekinitev **try…catch …** ,
  - večkratni varovalni blok **try … catch … catch …** ,
  - brezpogojni varovalni blok **try … catch … finally …**

Blok za obravnavo prekinitev

- Kodo, ki bi jo sicer napisali v delu programa ali pa npr. v neki metodi, zapišemo v varovalnem bloku **try** .
- Drugi del začenja besedica **catch** in v bloku zapišemo enega ali več stavkov za obdelavo izjem oz. napak (t.i. ***catch* handlers**).

- Če kateri koli 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 (lahko pa seveda le napišemo, da gre za napako!).

Blok za obravnavo prekinitev - primer

```
bool napaka=true;
while (napaka)
{
    try  // običajna programska koda
    {
        Console.Write("Prvo število: ");
        int levo = int.Parse(Console.ReadLine());
        Console.Write("Drugo število: ");
        int desno = int.Parse(Console.ReadLine());
        double rezultat = levo / desno;
        Console.WriteLine(rezultat);
        napaka = false;
    }
    catch // koda za obdelavo napake
    {
        Console.WriteLine("Ponovno vnesi podatke");
        napaka = true;
    }
}
```

Blok za obravnavo prekinitev - primer

```
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");
```

Večkratni varovalni blok - informativno

- Različne napake seveda proizvedejo različne vrste izjem.
  - Pri deljenju se lahko zgodi, da je drugi operand enak 0. Izjema, ki se pri tem zgodi, se imenuje **DivideByZeroException**.
  - Napaka pri pretvarjanju v drug podatkovni tip se imenuje **FormatException**

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");
```

Brezpogojni varovalni blok - informativno

- 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**.

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

finally
{
    // blok, ki se izvede vedno, ne glede  na napako
}
```

Brezpogojni varovalni blok: primer

- Od uporabnika zahtevamo vnos decimalnega števila, V primeru njegovega napačnega vnosa (npr. da je namesto cifer vtipkal nek znak), pa bi radi s programom nadaljevali, a namesto uporabnikovega vnosa želimo uporabiti kar privzeto vrednost števila!

```
Console.Write("Vnesi decimalno število: ");
double stevilo=10; //Privzeta vrednost spremenljivke število
try
{
    // ReadLine vrača string, zato uporabimo pretvorbo v tip double
    stevilo = Convert.ToDouble(Console.ReadLine());
}
catch  // catch del se izvede le če pride do napake
{
    // obvestilo o napačnem vnosu, spremenljivka stevilo bo zaradi tega
    // ohranila začetno vrednost 10
    Console.WriteLine("Napačen vnos števila, vrednost ostane privzeta (10)");
}
finally // izvede se v vsakem primeru
{
    //V primeru napačnega vnosa bo izpis enak:   Število = 10
    Console.Write(“Število = "+ stevilo);
}
```

Razhroščevanje - debugging

- Vsak program lahko zaženemo brez prekinitve (*Ctrl + F5*), s klikom na *Debug* → *Start Debugging* (oziroma *F5*), ali pa s klikom na zeleni gumb *Start Debugging* v orodjarni. **Program pa lahko izvajamo tudi stavek za stavkom – koračno**.

  Pri koračnem izvajanju programa so na voljo opcije

  - *Debug* → *Step Into* (F11) - ob vsakem klicu poljubne lastne metode se poglobimo še v njegovo kodo.
  - *Debug* → *Step Over* (F10) – premikanje preko klicev metod na nov stavek (ne zanima nas notranjost neke metode, torej gremo kar preko!).
  - *Run To Cursor* (Ctrl+F10) – za preskok nezanimivih delov kode.
  - *Debug* → *Step Out* (Shift + F10) – za prekinitev razhroščevanja znotraj trenutne metode; metoda se brez prekinitev izvede do konca.

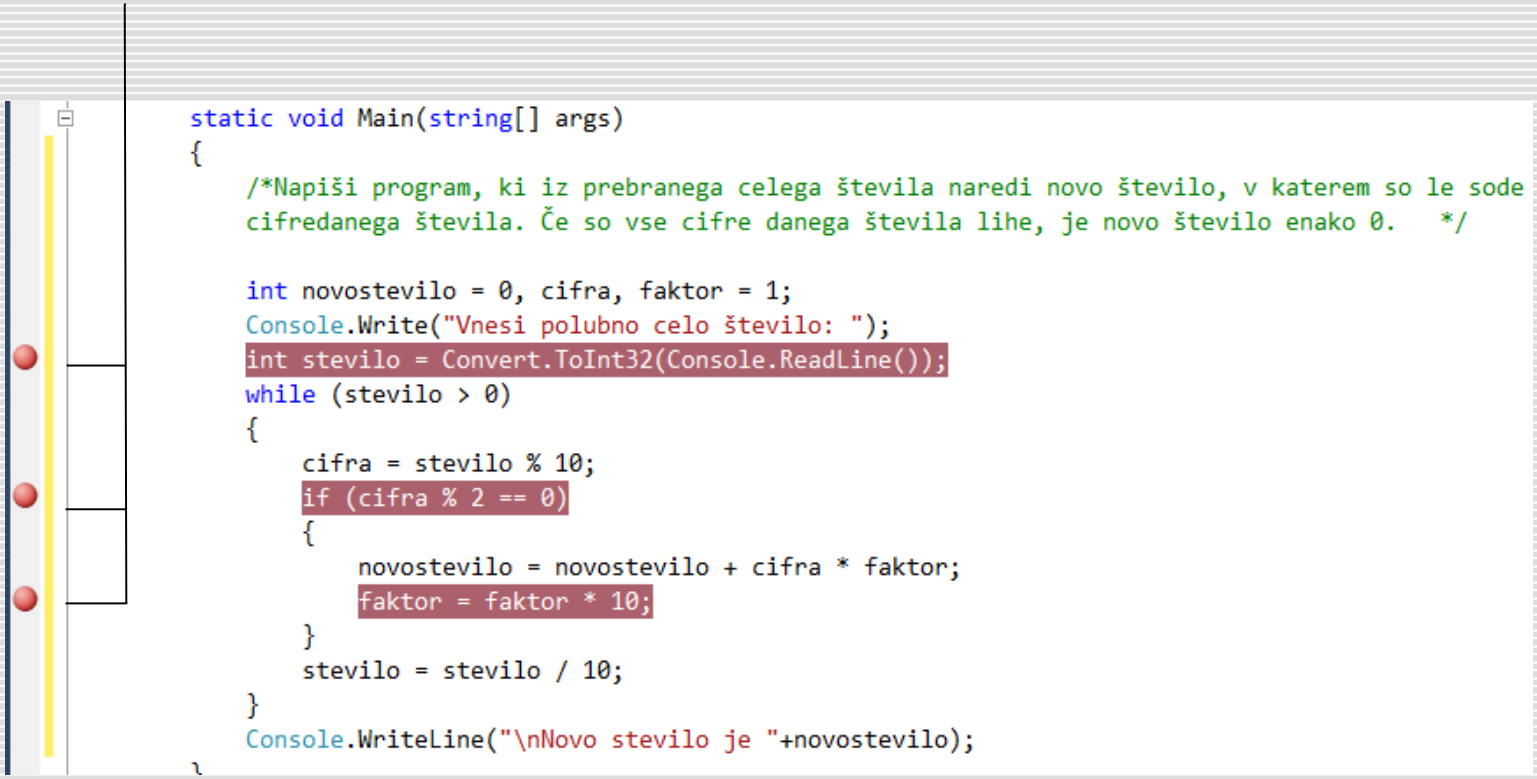
Prekinitvene točke

- Prekinitvene točke lahko postavimo ali zbrišemo na tri načine. Najprej izberemo vrstico, kjer želimo nastaviti prekinitveno točko, nato pa izberemo eno od treh možnosti:
  - Debug → Toggle Breakpoint
  - Tipka **F9**
  - Klik z miško v levem robu ustrezne vrstice s kodo

- Po končanem nastavljanju prekinitvenih točk program poženemo in v njem nemoteno delamo. Ko pridemo do prekinitvene točke, se izvajanje programa ustavi. Program je še vedno v pomnilniku, le delovanje je ustavljeno za toliko časa, da bomo pregledali vrednosti spremenljivk, oziroma raziskali del programa ki ne dela tako kot smo pričakovali.

Prekinitvene točke

- Na ekranu vidimo prekinitveno točko kot rdečo piko na levem robu okna s kodo, celotna vrstica pa je pobarvana rdeče.

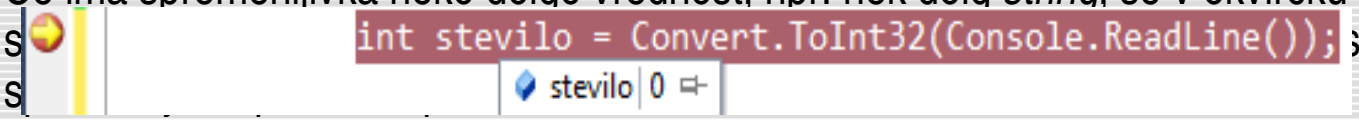


```
static void Main(string[] args)
{
    /*Napiši program, ki iz prebranega celega števila naredi novo število, v katerem so le sode
    cifredanega števila. Če so vse cifre danega števila lihe, je novo število enako 0.   */

    int novostevilo = 0, cifra, faktor = 1;
    Console.Write("Vnesi polubno celo število: ");
    int stevilo = Convert.ToInt32(Console.ReadLine());
    while (stevilo > 0)
    {
        cifra = stevilo % 10;
        if (cifra % 2 == 0)
        {
            novostevilo = novostevilo + cifra * faktor;
            faktor = faktor * 10;
        }
        stevilo = stevilo / 10;
    }
    Console.WriteLine("\nNovo stevilo je "+novostevilo);
}
```

Razhroščevanje – dodatne možnosti

- Dodatne možnosti za spremljanje poteka programa (samo nekatere)
  - Dodatne prekinitvene točke lahko poljubno nastavljamo kar med samim razhroščevanjem. Postopek je enak kot pri začetnem nastavljanju prekinitev.
  - Med razhroščevanjem lahko popravljamo kodo in nadaljujemo z razhroščevanjem ne da bi projekt ponovno zagnali.
  - *Visualizer*: med razhroščevanjem se lahko postavimo na poljubno spremenljivko, in pod njo se pokaže vrstica z vrednostjo te spremenljivke.

  - Če ima spremenljivka neko dolgo vrednost, npr. nek dolg *string*, se v okvirčku pod s 

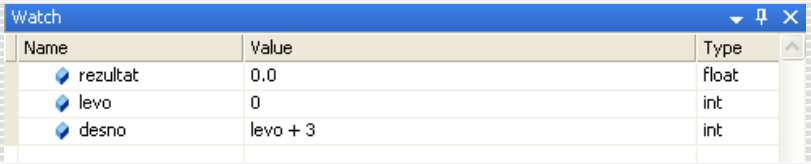
st
s

Razhroščevanje – dodatne možnosti

- Med razhroščevanjem se lahko v tekoči vrstici postavimo na neko spremenljivko in ji poljubno spremenimo vrednost.

- Med samim razhroščevanjem lahko v okno *Watch* (okno *Watch* odpremo med delovanjem programa s klikom na *Debug* → *Windows* → *Watch*) potegnemo (*drag & drop*) poljubno spremenljivko in v tem oknu nato spreminjamo vrednosti spremenljivk. Opcija je dvosmerna (kar naredimo v *Watch* oknu se odseva v programu in obratno!!!).

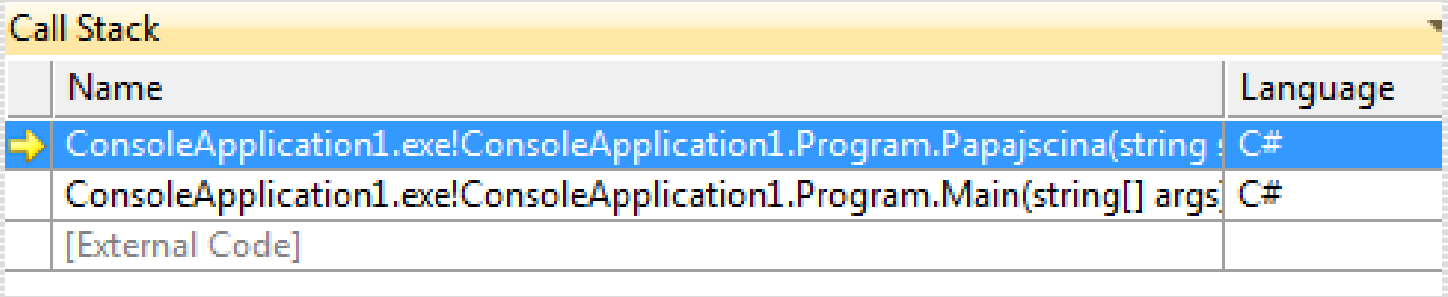
- V *Watch* okno lahko pišemo izraze. V oknu *Watch* se postavimo na določeno spremenljivko, kliknemo desni miškin gumb, izberemo opcijo *Edit value* in nato v oknu zapišemo ustrezen izraz.



Razhroščevanje – dodatne možnosti

- V *Watch* oknu imamo na voljo še nekaj možnosti. Če kliknemo z levo miškino tipko kamorkoli v to okno, lahko nato z desnim klikom miške odpremo meni za kopiranje, lepljenje, urejanje, dodajanje, brisanje ene ali brisanje vseh prekinitev.

- V oknu *Call Stack* (*Debug* → *Windows* → *Call Stack*) so navedeni vsi klici metod, ki so bile izvedene preden je prišlo do tekoče prekinitve (breakpointa)



Razhroščevanje – dodatne možnosti

- Okno *Immediate* (*Debug* → *Windows* → *Immediate*) se uporablja za ugotavljanje trenutne vrednosti spremenljivk, vrednosti izrazov oz. izvajanju stavkov, ki jih želimo izvesti kar med samim razhroščevanjem. Če hočemo izvedeti za trenutno vrednost neke spremenljivke, moramo v tekoči vrstici okna *Immediate* najprej obvezno zapisati znak **>** , nato znak **?** in potem še ime spremenljivke (lahko pa tudi zapišemo poljuben izraz) in končno stisnemo tipko <*Enter*>. V naslednji vrstici se nam prikaže ustrezna vrednost. (namesto znaka **?** lahko napišemo tudi *Debug.Print*).