Akademija tehničko-vaspitačkih strukovnih studija

.NET tehnologije

Prof. dr Zoran Veličković, dipl. inž. el.

2022/23.

Prof. dr Zoran Veličković, dipl. inž. el.

.NET tehnologije

Obrada izuzetaka u C#-u

(13)

Sadržaj

Pojam izuzetaka

- Iz **OO** programiranja znamo da **IZUZECI** (engl. *exceptions*) predstavljaju **EFIKASAN NAČIN** za **OBRADU GREŠAKA** i drugih **NEDOZVOLJENIH** STANJA koja nastaju pri **IZVRŠENJU** programa.
- Pod **IZUZETKOM** se u .NET-u podrazumeva **OBJEKAT** koji **KAPSULIRA INFORMACIJUE** o neprihvatljivim pojavama u programu.
- Ako se uklone **SVE PROGRAMERSKE GREŠKE**, ipak se mogu desiti situacije koje se **NE MOGU** izbeći!
- Primer opisane situacije je pokušaj otvaranja **NEPOSTOJEĆEG FAJLA** (primer urađen na laboratorijskoj vežbi) ili **NEDOSTATAK MEMORIJSKOG PROSTORA**.
- Dakle, ove situacije se nazivaju **IZUZECI** i **NIJE IH MOGUĆE SPREČITI**, ali se mogu **OBRADITI** i tako omogućiti da se izvršavanje programa **NE PREKINE** nasilno.
- Na pojavu ovakvih situacija, kaže se da program "**BACA**" (podiže) **IZUZETAK** čijom obradom se može razrešiti problem.

Obrada izuzetaka

- Na pojavu izuzetaka, **ZAUSTAVLJA** se **IZVRŠENJE TEKUĆE** programske **INSTRUKCIJE** i poziva se odgovarajuća **PROCEDURA** (metoda) **ZA OBRADU IZUZETKA**.
- Ako u tekućoj metodi **NIJE PREDVIĐENA** obrada nastalog izuzetka, onda će CLR "**ODMOTATI STEK**" da bi se **PRONAŠAO** odgovarajuću proceduru za obradu prekida.
- Pojam "**ODMOTAVANJA STEKA**" podrazumeva pretraživanje **UNAZAD** (u steku) za **ODGOVARAJUĆOM METODOM** kojom bi se obradio izuzetak.
- Dakle, **PROCEDURA ZA OBRADU IZUZETAKA** treba da obradi izuzetak i obezbediti **NASTAVAK IZVRŠENJA PROGRAMA**.
- Naredba "**`catch`**" obezbeđuje **IMPLEMENTACIJU** procedure za obradu izuzetaka.
- Deo koda koji se mora izvršiti iako se pojavio izuzetak, stavlja se u blok "**`finally`**".
- Za ove potrebe obrade izuzetaka koristi se klasa **`System.Exception`** ili klase izvedene iz nje.

Naredba throw

- Imenski prostor "**System**" sadrži sledeće primere izuzetaka: **ArgumentNullException**, **InvalidCastException**, **OverflowException**, ...
- Sistem podiže (kaže se baca) izuzetak da bi ukazao na **NEPREDVIĐENO STANJE** neke klase.
- **ISTOVREMENO** se formira objekt koji **OPISUJE IZUZETAK** i njegovo podizanje se obavlja primenom naredbom "**throw**" , operatorom "**new**" i pozivom konstruktora izuzetka:

```
throw new System.Exception();
```

- Kada se **BACI IZUZETAK**, a u tekućoj metodi nema odgovarajuće procedure, **CLR ODMOTAVA STEK** i pregleda **METODE POZIVAOCE** dok ne pronađe odgovarajuću koja može obraditi izuzetak.
- Ako **CLR** pregleda **SVE FUNKCIJE** (unazad) sve do metode **Main**, i ako ne nađe odgovarajuću metodu – **PREKIDA PROGRAM**!

Primer korišćenja naredbe `throw` (1)

```
....
 { using System;
    public class Test
    {
        public static void Main( )
        {
            Console.WriteLine("Enter Main...");
            Test t = new Test();
            t.Func1();
            Console.WriteLine("Exit Main...");
        }
        public void Func1()
        {
            Console.WriteLine("Enter Func1...");
            Func2();
            Console.WriteLine("Exit Func1...");
        }
```

Formiranje objekta **t** tipa **Test**

Poziv funkcije **Func1()** objekta **t**

Samo štampa tekst

Poziv funkcije **Func2()**

Primer korišćenja naredbe `throw` (2)

```
public void Func2()
        {
            Console.WriteLine("Enter Func2...");
            throw new System.Exception();
            Console.WriteLine("Exit Func2...");
        }
    }
}
```

Unutar funkcija **Func2()** se baca izuzetak.

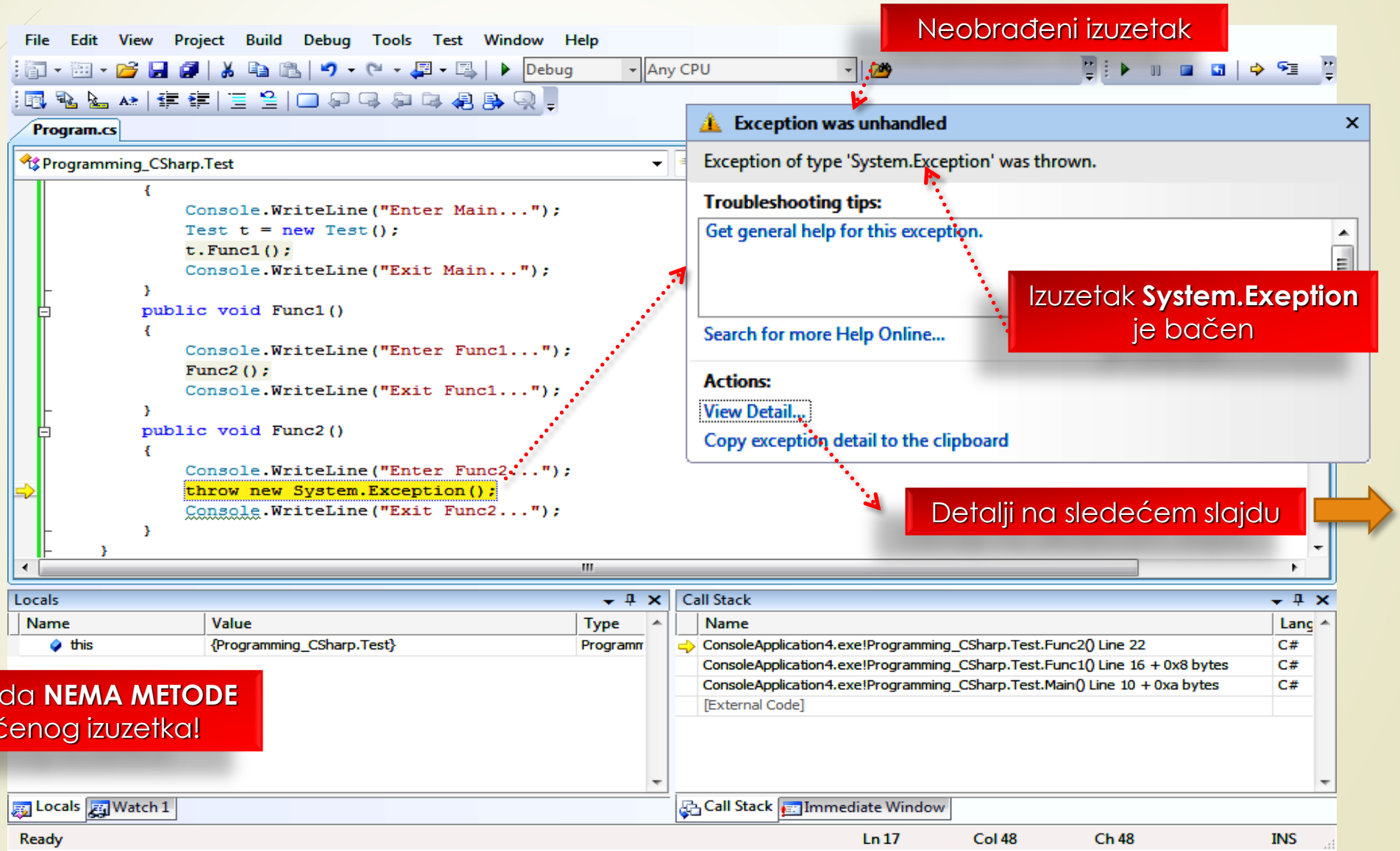
Bacanje izuzetka. Izazvaće se **PROBLEM** ako **nema** metode za obradu izuzetaka!

Zašto nema štampanja izlaska iz **Func1** i **Func2**?

Da li ima metode koja će obraditi izuzetak? Očigledno ne!

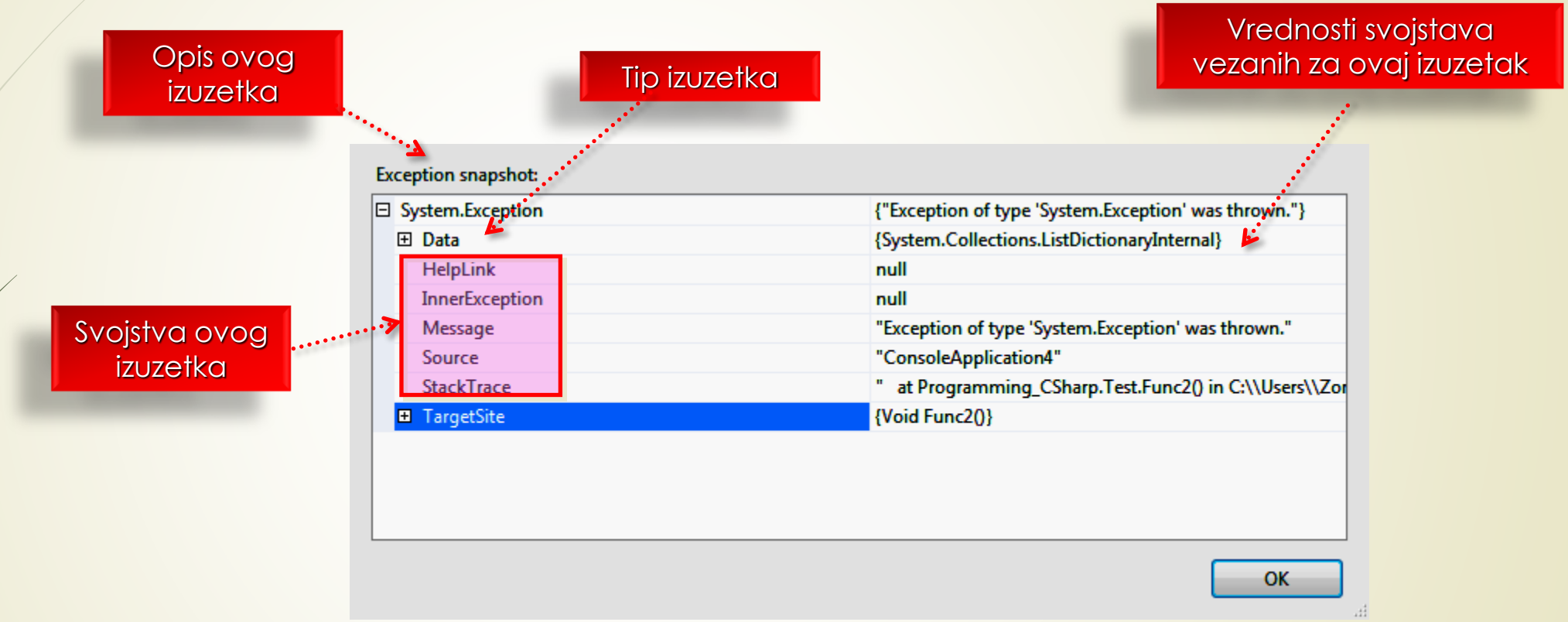
```
Output:
Enter Main...
Enter Func1...
Enter Func2...
```

Nobrađeni izuzetak u VS



VS je prepoznao da **NEMA METODE** za obradu bačenog izuzetka!

Detalji o bačenom izuzetku u VS



- Uopšte o SVOJSTVIMA izuzetaka nešto kasnije na ovom času.

Naredba `catch`

- Procedura za obradu izuzetaka se poziva (formira) rezervisanom rečju "**`catch`**".
- Naredba "**`throw`**" se izvršava u **BLOKU** "**`try`**".
- Prethodnom primeru je pridodata programska struktura **`tray-catch`** u cilju **HVATANJA I OBRADE** izuzetka.
- Uobičajeno je da se potencijalno opasna naredba(e) (one koje mogu izazvati izuzetke) stavlja(ju) u **BLOK `try`**.
- Posle bloka **`try`**, postavljaju se naredbe **`catch`**.
- **TIP IZUZETKA** koji se hvata specificira se u naredbi **`catch`**.
- Ako **NEMA SPECIFIKACIJE TIPA**, onda je **`catch`** naredba **GENERIČKA** – hvata **SVE TIPOVE IZUZETAKA**!
- Pogledajte izmenjeni kod metode **`Func2()`**. Šta se dobija ovom izmenom?

Naredbe try-catch

```
public void Func2()
    {
       Console.WriteLine( "Enter Func2..." );
    try
       {
          Console.WriteLine("Entering try block...");
             throw new System.Exception();
          Console.WriteLine("Exiting try block...");
       }
    catch
       {
          Console.WriteLine("Exception caught and handled.");
       }
       Console.WriteLine("Exit Func2...");
    }
```

```
Output:
Enter Main...
Enter Func1...
Enter Func2...
Entering try block...
Exception caught and
handled.
Exit Func2...
Exit Func1...
Exit Main...
```

U **try** bloku se postavljaju naredbe koje mogu izazvati izuzetke.

Procedura za obradu izuzetaka se "hvata" rezervisanom rečju **catch**.

Često korišćeni izuzeci

- U C# je definisan veliki skup **SISTEMSKIH IZUZETAKA**.
- Najčešće korišćeni tipovi izuzetaka sa njihovim opisom su prikazani u sledećoj tabeli:

IZUZETAK	OPIS
`ArithmeticException`	Osnovna klasa izuzetaka, bacaju ih aritmetički operatori.
`DivideByZeroException`	Bacaju se pri pokušaju deljenja nulom.
`IndexOutOfRangeException`	Bacaju se pri pokušaju pristupa nepostojećem elementu niza.
`InvalidCastException`	Baca se pri pokušaju eksplicitne konverzije, konverzija nije moguća.
`OutOfMemoryException`	Baca se pri neuspelom pokušaju alociranja – rezervisanja memorije operatorom **new**.
`StackOverflowExceptin`	Baca se kada stek poziva ima previše poziva metoda koje tek treba kompajlirati.

Namenski izuzeci

- **IZUZETKE** treba koristiti tako da kada se desi problem, **ODMAH** treba ponuditi **NAČIN NJIHOVOG REŠAVANJA**, a naročito:
  - kad ponestane memorije;
  - kada nema fajla koji se traži.
- Treba ispisati bar **PORUKU O GREŠCI** (kako bi korisnik bio obavešten o neregularnoj situaciji).
- Pored **GENERIČKIH** "`catch`" **NAREDBI** mogu se formirati i **NAMENSKE** "`catch`" **NAREDBE**.
- Shodno tome, mogu se formirati i **NAMENSKI IZUZECI**!
- Dozvoljeno je formiranje naredbi `catch` kojima se obrađuju samo **ODGOVARAJUĆI TIPOVI IZUZETAKA**.
- Biće pokazani primeri izuzetaka koji se javljaju prilikom **POKUŠAJA DELJENJA NULOM**.
- Takođe, biće razmatran pokušaj deljenja **nule** sa **nulom**, kao **SPECIJALAN SLUČAJ**.

Namenski `catch` (1)

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion

namespace SpecifyingCaughtException
{
    public class Test
    {
        public static void Main()
        {
            Test t = new Test();
            t.TestFunc();
        }
```

Formiranje klase **`Test`** koja poziva svoju metodu **`TestFunc()`**

Namenski `catch` (2)

Generalno deljenje dva broja može proizvesti problem!

```
public void TestFunc()
      {
    try
        {
            double a = 5;
            double b = 0;
            Console.WriteLine( "{0} / {1} = {2}", a, b, DoDivide(a, b));
        }
    catch ( System.DivideByZeroException)
        {
            Console.WriteLine("DivideByZeroException caught!");
        }
    catch ( System.ArithmeticException)
        {
            Console.WriteLine("ArithmeticException caught!");
        }
```

Pokušaj deljenja nulom može izazvati izuzetak – **OBRADI GA**.

1

Hvataj prvo najčešći tip izuzetaka pri deljenju

2

Namenski `catch` (3)

Izuzeci **opšte namene** se hvataju tek na kraju

```
catch
        {
            Console.WriteLine("Unknown exception caught" );
        }
    }

// deli samo ako je to dozvoljeno, u suprotnom baci izuzetak
    public double DoDivide(double a, double b)
        {
            if (b == 0)
                throw new System.DivideByZeroException();
            if (a == 0)
                throw new System.ArithmeticException();
            return a / b;
        }
    }
}
```

Bacanje dva tipa izuzetka iz metode **DoDivide()**

```
Output:
DivideByZeroException caught!
```

Naredba `finally` (1)

- **DIREKTAN SKOK** na metodu za obradu izuzetaka ponekad može izazvati probleme! Kada?
- Ako program koristi neki resurs (otvara fajl) i on **OSTANE ZAUZET** zbog skoka na metode za obradu izuzetaka eto problema! Koje vrste?
- Da bi se specificirala akcija koja se **MORA** obaviti **PRE POZIVA METODE** za obradu izuzetaka, specificira se rezervisanom rečju "**`finally`**".
- Ova akcija se specificira u programskom bloku "**`finally`**".
- Šta više, programski kod u bloku **`finally`** se izvršava **BEZ OBZIRA** da li je izuzetak bačen ili ne!

Naredba `finally` (2)

```
public void TestFunc()
      {
       try
         {
            Console.WriteLine("Ovde se otvara fajl");
            double a = 5;
            double b = 0;
            Console.WriteLine("{0}/{1} = {2}", a, b, DoDivide(a, b));
            Console.WriteLine("Ova linija može ali i ne mora biti štampana");
         }
```

Pokušaj deljenja može izazvati izuzetak – treba ga obarditi.

try blok sa potencijano "opasnom" naredbom.

Naredba `finally` (3)

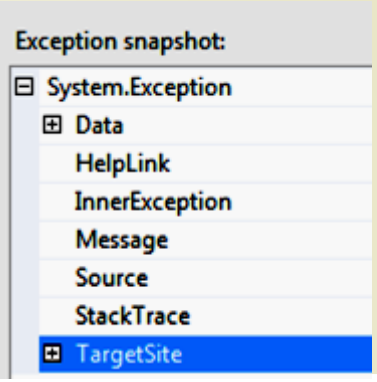
```
// najčešći tip izuzetka
    catch (System.DivideByZeroException)
    {
        Console.WriteLine("DivideByZeroException caught!");
    }
    catch
    {
        Console.WriteLine( "Unknown exception caught");
    }

    finally
    {
        Console.WriteLine("Ovde zatvori već otvoren fajl.");
    }
}
```

`catch` naredba sa **NAMENSKIM** izuzetkom

`catch` naredba sa **GENERALIZOVANIM** izuzetkom

OBAVEZNO IZVRŠAVANJE bloka **`finally`** bez obzira na to da li je izuzetak bačen ili ne

Objekt tipa `Exception` (1)



- U C# klasa **`System.Exception`** se koristi za **OPIS IZUZETAKA**.
- Klasa **`System.Exception`** ima veliki broj **SVOJSTAVA**, evo nekih:

SVOJSTVO	ZNAČENJE
`Message`	Svojstvo **`Message`** predstavlja informacije o izuzetku i samo se može **ČITATI**. Vrednost svojstva **`Message`** se definiše prilikom bacanja izuzetka i prosleđuje se kao **argument** konstruktoru izuzetka.
`HelpLink`	Svojstvo **`HelpLink`** predstavlja **URL** vezu do datoteke za pomoć povezanu sa izuzetkom.
`StackTrace`	Svojstvo **`StackTrace`** je samo za **ČITANJE** i sadrži informaciju o steku poziva za poruku o grešci.
`Source`	Svojstvo **`Source`** sadrži **ime programa** koje koji je doveo do izuzetka.
`TargetSite`	Svojstvo **`TagretSite`** sadrži **ime metode** iz koje je bačen izuzetak.

Primena svojstva `System.Exception` (1)

```
...
namespace Programming_CSharp
{
    using System;
    public class Test
{
    public static void Main()
    {
        Test t = new Test();
        t.TestFunc();
    }

// pokušaj deljenja dva broja
// opsluživanje mogućih izuzetaka
```

Korišćenje svojstava klase **`System.Expection`**

Primena svojstva System.Exception (2)

```
public void TestFunc()
{
try
    {
        Console.WriteLine( "Open file here");
        double a = 12;
        double b = 0;
        Console.WriteLine("{0} / {1} = {2}", a, b, DoDivide(a, b));
        Console.WriteLine("This line may or may not print");
    }
catch (System.DivideByZeroException e)
  {
        Console.WriteLine("\nDivideByZeroException! Msg: {0}", e.Message);
        Console.WriteLine("\nHelpLink: {0}", e.HelpLink);
        Console.WriteLine("\nHere's a stack trace: {0}\n", e.StackTrace);
    }
```

Svojstvo **Message** za objekt **e** formiran izuzetkom

Svojstvo **HelpLink** treba dodliti **PRE** ispaljivanja izuzetka.

Svojstvo **StackTrace** za objekt **e**

Primena svojstva System.Exception (3)

```
catch
    {
        Console.WriteLine("Unknown exception caught");
    }
finally
    {
        Console.WriteLine ("Close file here.");
    }
} // kraj Test

// deli ako je to moguće!
public double DoDivide(double a, double b)
{
  if (b == 0)
   {
        DivideByZeroException e = new DivideByZeroException();
        e.HelpLink ="http://www.libertyassociates.com";
        throw e;
   }
```

Svojstvo **Message** nije zadato, te se poziva STANDARDNA PORUKA ovog izuzetka: **Attempted to devide by zero**.

Formiranje objekta **e** tipa **DivideByZeroException**.

Svojstvo **HelpLink** se dodeljuje pre ispaljivanja izuzetka.

Bacanje izuzetka

Primena svojstva System.Exception (3)

```
if (a == 0)
        throw new ArithmeticException();
        return a/b;
          }
    }
}
```

Bacanje idrugog zuzetka

Output:

```
Open file here
DivideByZeroException! Msg: Attempted to divide by zero.
HelpLink: http://www.libertyassociates.com
Here's a stack trace:
at Programming_CSharp.Test.DoDivide(Double a, Double b)
in c:\...exception06.cs:line 56
at Programming_CSharp.Test.TestFunc( )
in...exception06.cs:line 22
Close file here.
```

Korisnički izuzeci

- **UGRAĐENE TIPOVE IZUZETAKA** prepoznaje **CLR** i povezuje sa porukama o grešci.
- Ovi ugrađeni tipovi izuzetaka su **ČESTO DOVOLJNI** da se prikažu detaljne informacije o bačenom izuzetku.
- Ako postoje situacije u kojima je zgodnije pozvati različite procedure za obradu izuzetaka, onda, treba kreirati **SOPSTVENE – KORISNIČKE IZUZETKE**.
- **SOPSTVENI – KORISNIČKI IZUZECI** podrazumevaju i korisničke metode za obradu izuzetaka.
- Sopstveni izuzetak se izvodi iz klase:

```
System.ApplicationException
```

- Na ovaj način kreirani izuzetak se „hvata“ **`catch`** naredbom u kojoj se specificira korisnički izuzetak.
- Primer korisničkog izuzetka koji se baca u slučaju kada je **imenilac jednak nuli** je dat u nastavku.

https://resources.oreilly.com/examples/9780596006990/tree/master/ProgCSharp4eSourceR5

Korisnički izuzeci (1)

```
#region Using directives
using System;
using System.Collections.Generic;
using System.Text;
#endregion

namespace CustomExceptions
{
    public class MyCustomException : System.ApplicationException
    {
        public MyCustomException(string message) : base(message)
        {

        }
    }
```

Korisničko izuzetak **MyCustomException**

Nasleđivanje klase **ApplicationException**

Konstruktor izuzetka

string argument koji se prosleđuje svojoj klasi.

Isto što i **this**, koristi se za pristup članu **osnovne** klase iz **izvedene** klase.

Korisnički izuzeci (2)

```
public class Test
{
    public static void Main()
    {
        Test t = new Test();
        t.TestFunc();
    }

// pokušaj deljenja , obraditi moguće greške
    public void TestFunc()
    {
        try
        {
            Console.WriteLine("Ode se otvara fajl za čitanje");
            double a = 0;
            double b = 5;
            Console.WriteLine("{0} / {1} = {2}", a, b, DoDivide(a, b));
            Console.WriteLine("Ova linija koda se može ali i ne mora izvršiti");
        }
```

Poziv metode za obavljanje deljenja

Zašto?

Korisnički izuzeci (3)

```
// prvo hvataj najčešći tip izuzetaka
        catch (System.DivideByZeroException e)
         {
           Console.WriteLine("\nDivideByZeroException! Msg: {0}", e.Message);
           Console.WriteLine("\nHelpLink: {0}\n", e.HelpLink);
         }
        catch (MyCustomException e)
          {
            Console.WriteLine("\nMyCustomException! Msg: {0}", e.Message);
            Console.WriteLine("\nHelpLink: {0}\n", e.HelpLink);
          }
        catch
          {
            Console.WriteLine("Unknown exception caught" );
          }
        finally
          {
            Console.WriteLine("Zatvori fajl na ovom mestu.");
          }
    }
```

Hvataj prvo **DivideByZeroException**

Potom - korisnički **MyCustomException**

Na kraju – bilo koji - nepoznati izuzetak

Korisnički izuzeci (4)

```
// deli ako je to dozvoljeno
 public double DoDivide(double a, double b)
 {
    if (b == 0)
     {
       DivideByZeroException e = new DivideByZeroException();
       e.HelpLink = "http://www.korisnicki_sajt.com";
       throw e;
     }
    if (a == 0)
     {
       MyCustomException e = new MyCustomException("Imenilac jednak 0!");
       e.HelpLink ="http://www. korisnicki_sajt.com/Broilac_nula.htm";
       throw e;
     }
    return a / b;
 }
}
}
```

Definiši svojstva izuzetka **DivideByZeroException**

Baci izuzetak **DivideByZeroException**

Baci izuzetak **MyCustomException**

Definiši svojstva izuzetka **MyCustomException**

Ili obavi deljenje