## Exception Handling

בכל אפליקציה קיימות שגיאות. ככל שאפליקציה גדולה יותר, כך גדלה הסבירות שנמצא בה כמות שגיאות גדולה יותר. שגיאות הן הבעיה המרכזית של עולם התכנות, תיקונן גוזל משאבים כספיים ואנושיים רבים.

עלות תיקון השגיאות עולה ביחס ישיר לשלב הפיתוח שבו נתגלתה השגיאה.

ניתן לסווג את השגיאות למספר סוגים שונות:

**שגיאות קומפילציה** - אלו שגיאות שקל מאוד לזהותן ולתקנן. הקומפיילר הוא זה שמזהה את שגיאות הקומפילציה ומתריע עליהן.

**שגיאות לוגיות** - שגיאות שקשה מאוד לאתרן ולעיתים מסובך מאוד לתקנן. המאפיין של שגיאת אלו הוא שהאפליקציה עוברת קומפילציה, מתנהלת כרגיל, אולם התוצאה שמתקבלת או אחת התוצאות הן שגויות. לעיתים קורה ששגיאות אלו אינן מתגלות בזמן, אולם גורמות נזקים כבדים לארגון.

**שגיאות זמן ריצה** - שגיאות אלו אינן מתגלות בזמן הקומפילציה. האפליקציה מתנהלת כרגיל ואף התוצאות מדויקות, אולם במקרים מאוד מסוימים האפליקציה קורסת. הקריסה יכולה להתרחש בעקבות הקלדת נתונים, שמירה לקובץ, ניסיון כושל להקצאה דינמית, כשל במערכת חומרה כמו מודם, כרטיס רשת או Sound Blaster, התחברות ל- Data Base וכדומה.

תקלות אלו הגורמות לקריסת האפליקציה מכונות חריגים.

במקרים רבים לא ניתן למנוע היווצרות של מקרים חריגים. לא ניתן לקבוע האם ומתי ייגמר למשתמש הזיכרון, לא ניתן לחזות מתי תתרחש תקלת חומרה אשר תגרום לאפליקציה לקרוס, לא ניתן למנוע מהמשתמש לטעות בהקלדת נתונים.

לכן ניתן לומר בכמעט וודאות שהשאלה היא לא **האם** אותם החריגות יתרחשו, אלא **מתי** הן אכן יתרחשו. השאלה היא כיצד נתכונן לקראתם, כיצד נהפוך את קוד האפליקציה לקוד חסין (Robust

Code) אשר יוכל להתמודד בהצלחה עם תקלות אלו.

למזלנו חלק גדול מהבעיות הגורמות לקריסת האפליקציה ניתנות לחיזוי מראש.

מתכנתים ומנתחי מערכות מנוסים מסוגלים לזהות את אותם מצבים מסוכנים העלולים לגרום להתרסקויות ולהתכונן אליהם כראוי.

כאשר מתרחש מצב חריג, ניתן לפעול במספר דרכים:

* לא לטפל ולגרום לקריסת האפליקציה - ברור שמצב זה אינו אופטימלי, בכל מקרה לא נרצה שהאפליקציה תקרוס. מדובר באמינות שלנו כמפתחים, נעשה כל שביכולתנו למנוע מצב של קריסה.
* הודעה למשתמש על התקלה, ניסיון לגבות את כל הנתונים וסיום מכובד של היישום. במקרים רבים זו תהיה דרך הטיפול הטובה ביותר (בעיות חומרה למשל).
* הודעה למשתמש על מהות התקלה, גיבוי הנתונים, מתן אפשרות למשתמש לנסות להתגבר בכוחות עצמו על הבעיה, במידה והתיקון הצליח, ניתן להמשיך באפליקציה .
* האפליקציה בעצמה מנסה להתגבר על הבעיה אשר גרמה לחריגה מבלי להפריע למשתמש.

מנגנון הטיפול בחריגות ב- C# מוגדר מארבע מילות מפתח (Key Word):

**try** - מגדירה בלוק (block) של קוד מוגן. משמעות הקוד המוגן, שבמידה ויתגלה מקרה חריג, יעשה ניסיון לטפל בו באמצעות catch אחד או יותר, או לפחות להתייחס אליו מבלי שהאפליקציה תקרוס.

כל גילוי של מקרה חריג יגרום לסיום מיידי של הבלוק.

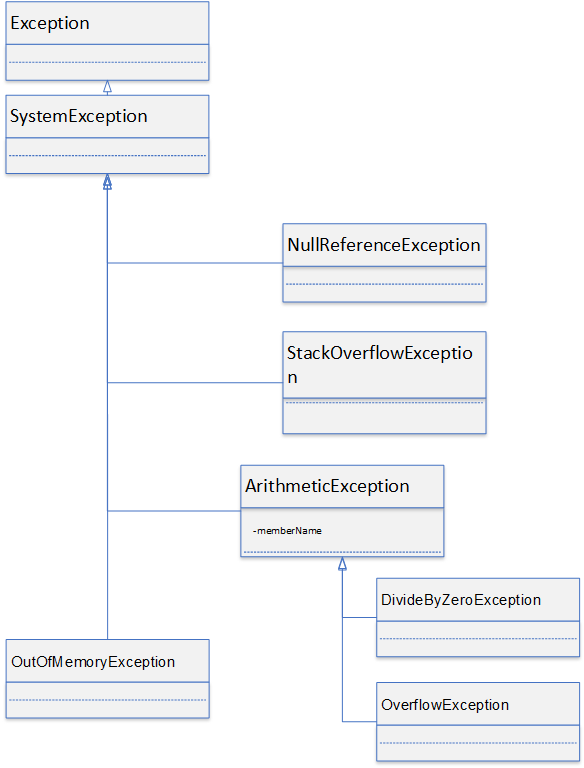
**catch** - מילת המפתח המגדירה מסגרת של טיפול בחריגה שנתגלתה. catch מגדיר קטע קוד התחום בבלוק המכונה event handler. מטרת קטע קוד זה הנה ניסיון לטפל במקרה החריג שנתגלה בתוך בלוק ה-try.

בסיום בלוק try חייבים להגדיר בלוק catch אחד או יותר (לטיפול במספר סוגי חריגות). כל בלוק catch מקבל ייחוס של המחלקה Exception או של אחת מנגזרותיה.

**throw** - גורם להעברת הטיפול במקרה החריג למתודה אשר הפעילה את המתודה בה נתגלה המקרה החריג, (כלומר אם מתודה/פונקציה A קראה למתודה/פונקציה B, ובפונקציה B התרחש אירוע חריג, הפקודה throw תעביר את הטיפול באירוע החריג חזרה לפונקציה A). גם מתודה זו (A בדוגמא) מסוגלת להעביר את הטיפול בחריגה למתודה שהפעילה אותה וחוזר חלילה. ניתן לבצע זאת באמצעות throw.

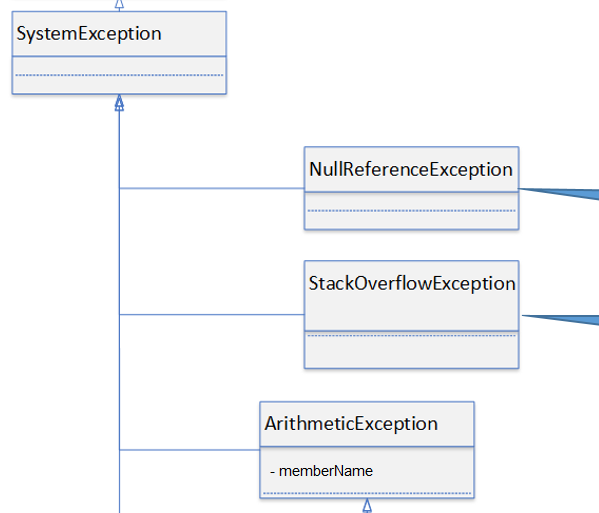
**finally** - קטע קוד התחום גם הוא בתוך בלוק. מובטח שקטע קוד זה תמיד יבוצע, אם נתפסה חריגה ואם לאו. קטע קוד זה יופעל אף אם הטיפול במקרה החריג מסיים את האפליקציה (Application.Exit()).

.NET Framework מגדירה היררכית מחלקות לטיפול במקרים חריגים. מחלקת הבסיס של הטיפול בחריגים הנה המחלקה Exception, ממנה נגזרות מספר רב של מחלקות המטפלות בחריגות מוגדרות.



מחלקת הבסיס של המחלקות המטפלות בחריגים.

מייצגת חריגה כללית.



חריגה המתקבלת כאשר נתייחס לאובייקט שהייחוס שלו שווה ל- null.

ניסיון לקרוא למתודה או להכניס משתנים למחסנית כאשר היא מלאה.

ניסיון לחלק ב-0 יעורר חריגה זו.

גלישה מהתחום המקסימלי או המינימלי של משתנה בתוך בלוק checked.

חריגה המתגלה כאשר אין מספיק זיכרון להמשיך בביצוע התוכנית.

בנוסף למחלקות המיוצגות באיור מוגדרות עוד עשרות רבות של מחלקות המטפלות בחריגות.

**בלוק checked**

תוצאת חישוב עלולה לחרוג מהתחום המוגדר, המקסימלי או המינימלי, שמשתנה יכול להחזיק, לדוגמה:

…

byte num1 = 200;

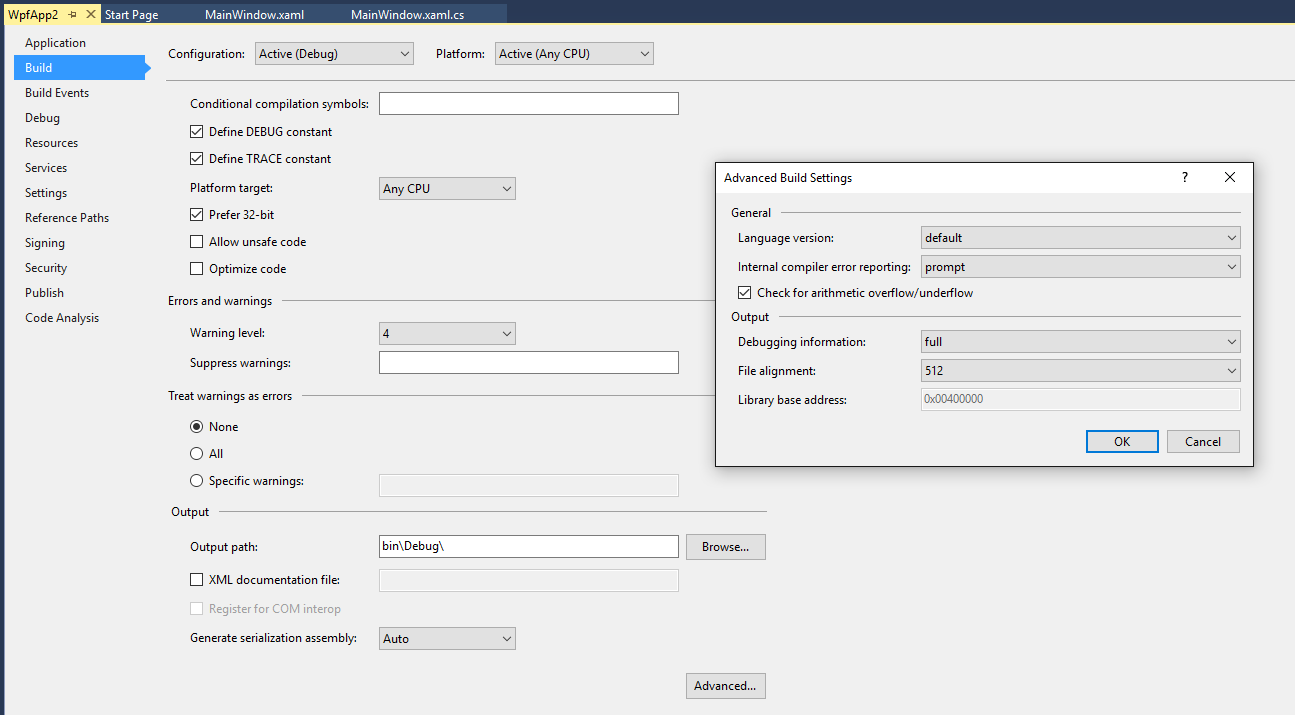
byte num2 = 100;

byte num3 = (byte)(num1 + num2);

מכיוון שטווח הערכים החוקי של משתנה מטיפוס byte הוא 0 עד 255 ברור שמתרחשת גלישה , כברירת מחדל שפת C# מאפשרת ביצוע גלישה (התוצאה תהיה 44), היא מטפלת בכך באופן שקוף.

אולם, במקרה הצורך, ניתן לשנות התנהגות זו ולגרום להעלאת חריגה במקרה של גלישה.

על מנת לשנות התנהגות ברירת מחדל זו ברמת האפליקציה, נוכל לבצע זאת בחלוןProject Properties:



לעיתים נרצה לשנות התנהגות זו רק במקטעים מסוימים בקוד. נבצע זאת בעזרת הגדרת בלוק (Block) checked.

לדוגמה:

…

byte num1 = 200;

byte num2 = 100;

byte num3 = 0;

**checked**

**{**

**num3= (byte)(num1 + num2);**

**}**

בדוגמה זו הניסיון לאחסן במשתנה num3 ערך הגדול מטווח ערכיו החוקי יעורר חריגה (OverflowException).

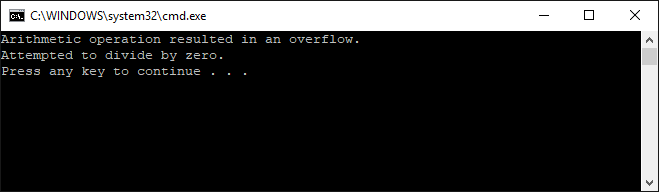
הערה - ניתן לשלב בתוך בלוק checked בלוק unchecked, בלוק זה יאפשר ביצוע גלישה.

דוגמה:

הגדרת בלוק try .

|  |
| --- |
| byte Num = byte.MaxValue;  הגדרת בלוק try .  try  {  checked  הגדרת בלוק checked המטפל בחריגה אם נתגלתה.  {  Num++;  Console.WriteLine("Num = {0}", Num);  }  }  catch (OverflowException e)  {  Console.WriteLine(e.Message);  }  int Num2 = 10;  int Num3 = 0;  int Div;  try  {  Div = Num2 / Num3;  חלוקה ב-0  }  catch (DivideByZeroException e)  {  Console.WriteLine(e.Message);  יתבצע רק אם התרחשה חריגה  } |

פלט:



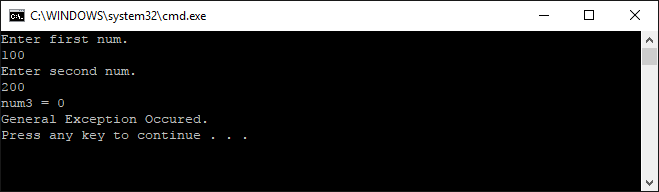
כאשר מתגלית חריגה בתוך בלוק try ואין בלוק catch מתאים, האפליקציה תקרוס.

במקרים רבים קטע קוד טומן בחובו סכנה למספר חריגות שונות. במקרים אלו, C# מאפשרת להגדיר מספר בלוקי catch, כל אחד מהם יתמחה בטיפול בחריגה אחרת. בנוסף ניתן ולעיתים אף רצוי להגדיר catch כללי, המטפל בכל סוגי החריגות, אותו Generic catch יוגדר תמיד אחרון.

דוגמה:

|  |
| --- |
| short num1 = 0;  short num2 = 0;  short num3 = 0;  string tmp;  try  {  Console.WriteLine("Enter first num.");  tmp = Console.ReadLine();  num1 = short.Parse(tmp);  Console.WriteLine("Enter second num.");  tmp = Console.ReadLine();  num2 = short.Parse(tmp);  num3 = (short)(num1 / num2);  Console.WriteLine("num3 = {0}", num3);  tmp = "Moshe";  הגדרת מספר בלוקי catch אשר יטפלו במקרים שונים של חריגות.  num3 = short.Parse(tmp);  }  catch (DivideByZeroException e)  {  Console.WriteLine(e.Message);  }  catch (OverflowException e)  {  Console.WriteLine(e.Message);  Generic catch  }  catch  {  Console.WriteLine("General Exception Occured.");  } |

פלט:



העלאת חריגה עלולה ליצור בעיה רצינית, הבעיה יכולה להיווצר כאשר בסוף הבלוק נכתוב קטע קוד אשר ישחרר משאבים שהוגדרו בתוך הבלוק. הבעיה שתיווצר היא שאם התרחשות חריגה ביצוע הקוד מסתיים, הביצוע עובר לבלוק catch המתאים והמשאבים לא ישוחררו.

על מנת לפתור בעיה חמורה זו נגדיר, במקרים אלו, בלוק finally.

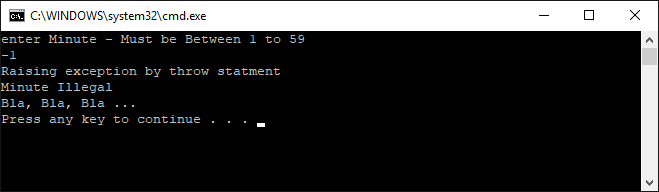
בלוק finally יבוצע תמיד, אם תתרחש חריגה ואם לאו.

לדוגמה:

|  |
| --- |
| int Minute;  string tmp;  Console.WriteLine("enter Minute - Must be Between 1 to 59");  tmp = Console.ReadLine();  Minute = int.Parse(tmp);  try  {  העברת הטיפול בחריגה למתודה הקודמת  if (Minute < 1 || Minute > 59)  {  throw new Exception("Minute Illegal");  }  Console.WriteLine("Minute = {0}", Minute);  }  catch (Exception e)  {  Console.WriteLine("Raising exception by throw statment");  Console.WriteLine(e.Message);  }  finally  {  Console.WriteLine("Bla, Bla, Bla ...");  } |

בלוק Finally

פלט:



לעיתים אין למחלקה את הכלים ואת היכולת להתמודד בהצלחה עם מקרים חריגים מסוימים. במקרים אלו C# מאפשרת להעביר את הטיפול באותם החריגים למתודות אשר הפעילו את המתודה אשר בה התגלתה החריגה.

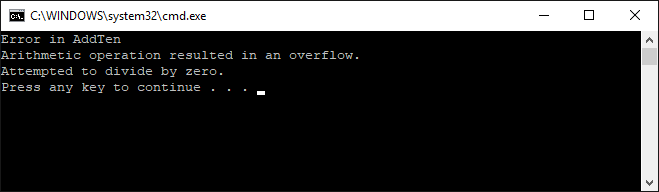
העברת האחריות מתבצעת באמצעות מילת המפתח throw .

לדוגמה:

|  |
| --- |
| class MyMath  {  public static byte AddTen(byte num)  {  try  {  checked  {  num += 10;  }  }  catch  {  Console.WriteLine("Error in AddTen");  throw;  העברת המשך הטיפול בחריגה למתודה המפעילה.  }  return num;  }  public static float Divide(float n1, float n2)  {  העלאה יזומה של חריגה  if (n2 == 0)  throw new DivideByZeroException();  return n1 / n2;  }  }  class Program  {  static void Main(string[] args)  {  byte num = byte.MaxValue;  try  {  Console.WriteLine(MyMath.AddTen(num));  }  catch (Exception e)  {  Console.WriteLine(e.Message);  }  try  {  Console.WriteLine(MyMath.Divide(12.34f, 0));  }  catch (DivideByZeroException e)  {  Console.WriteLine(e.Message);  }  }  }  } |

באמצעות throw המתודה AddTen מעבירה את המשך הטיפול בחריגה למתודה שהפעילה אותה. החריגה המועברת היא אותה החריגה שנתקבלה, במקרה דנן, תועבר למתודה Main החריגה OverflowException. אם Main לא תגדיר מקרה של טיפול בחריגה זו האפליקציה תקרוס, משום שהטיפול בחריגה לא הושלם.

פלט:



.NET Framework מגדירה עשרות מחלקות המתמחות במקרים חריגים. כל אותן המחלקות יורשות, כאמור, את המחלקה Exception. ככלל ניתן לקבוע שעדיף לנצל את אותן המחלקות.

עושר המחלקות המצוי בהיררכיה של Exception בהחלט מספיק על מנת להתמודד עם רוב רובם של המצבים החריגים.

לעיתים יותר נדירות, נאלץ לספק מענה לתרחישים אשר אין להן מענה בהיררכית Exception. במקרים אלו נתכנן מחלקה שתירש את Exception ואשר תיתן מענה לבעיה המיוחדת והנדירה שנוצרה.

על מנת לעורר חריגה מהמחלקה המותאמת נשתמש בפקודה throw.

המחלקה Exception מספקת מספר מאפיינים ציבוריים וירטואליים אשר ניתן לספק להן מימוש חדש במחלקה הנגזרת.

לדוגמה:

בניית מחלקת חריגים

|  |
| --- |
| class CustomException : Exception  {  private string help\_link = "www.mycomany.co.il";  public override string Message  {  מימוש מאפיינים וירטואליים המוגדרים במחלקה Exception.  מימוש מאפיינים וירטואליים המוגדרים במחלקה Exception.  get  {  return "My Custom Message";  }  }  public override string HelpLink  {  get  {  return help\_link;  }  set  {  העלאת חריגה מותאמת  יצירת קובץ Log במקרה של חריגה  help\_link = "www.mycomany.co.il";  }  }  public void SaveToLog()  {  StreamWriter sw = new StreamWriter(@"MyLog.txt", true);  sw.WriteLine(DateTime.Now.ToString());  sw.Close();  }  }  class Program  {  static void Main(string[] args)  {  int num1 = 10;  int num2 = 0;  try  {  if (num2 == 0)  throw new CustomException();  Console.WriteLine(num1 / num2);  }  catch (CustomException e)  טיפול בחריגה מותאמת  {  Console.WriteLine(e.Message);  Console.WriteLine(e.HelpLink);  e.SaveToLog();  }  }  } |

פלט:

