المحاضرة الثالثة

التعامل مع النصوص

نستخدم نوع البيانات String للعامل مع النصوص .

لتعريف string يمكن استخدام أحد الطرق التالية:

```
string S1="....النص المطلوب....";

string  s2 = new string(  مصفوفة من النوع char );

string s3 = new string(  مصفوفة من نوعChar, بداية النسخ, عدد المحارف);
string s4 = new string( محرف, عدد مرات التكرار للمحرف );
```

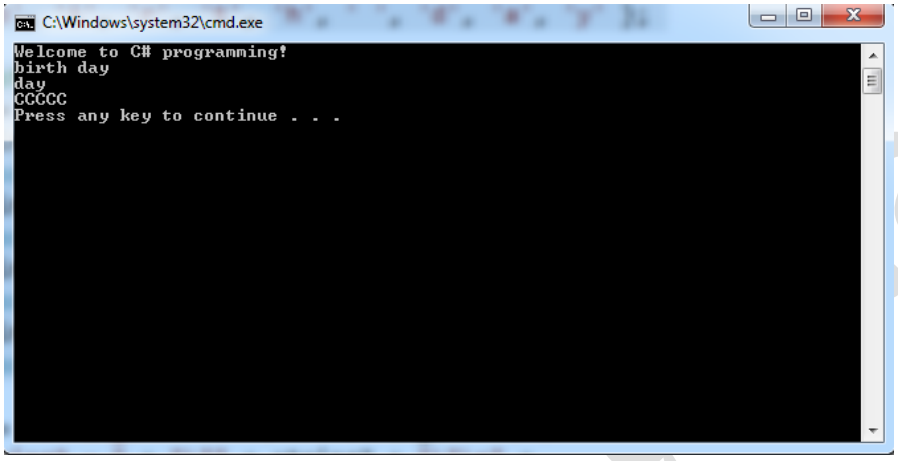
مثال:

```
string S1;
  string S2, string1, string2,
  string3, string4;

  char[] characterArray =
  { 'b', 'i', 'r', 't', 'h', ' ', 'd', 'a', 'y' };
  S2 = "Welcome to C# programming!";
  string1 = S2;
  Console.WriteLine(string1);
  string2 = new string( characterArray );
  Console.WriteLine(string2);
  string3 = new string( characterArray, 6, 3 );
  Console.WriteLine(string3);
```

```
string4 = new string( 'C', 5 );

Console.WriteLine(string4);
```



يمكن التعامل مع النوع String بنفس طريقة التعامل مع المصفوفات و يمكن التنقل بين الأحرف باستخدام الدليل .

مثال لإيجاد عدد مرات تكرار المحرف K في نص.

```
class Program
{
    static void Main(string[] args)
    {
      string S1; int j = 0;
      S1 = Console.ReadLine();
      for (int i = 0; i < S1.Length; i++)
          if (S1[i] == 'k')
              j++;
      Console.WriteLine(j);
    }
}
```

CopyTo: تستخدم لنسخ جزء من سلسلة إلى مصفوفة محارف أخرى:

مثال:

```
static void Main(string[] args)
{
    string S1 = "welcome";
   char[]s2=new char[10];
    S1.CopyTo(3, s2, 0, 4);
    Console.WriteLine(s2);
}
```

الخرج هو come

توابع هامة:

الدالة او الخاصية	الإستخدام
Length	تحدد طول النص
Compare	للمقارنة بين نصين
Contains	للبحث عن نص او حرف ضمن النص
EndWith	لمعرفة فيما اذا كان النص يبدأ او ينتهي بحرف او نص معين
IndexOf LastIndexOf	لمعرفة مكان وجود حرف او بداية نص معين ضمن النص ، سواء من البداية او من النهاية
Remove	حذف جزء معين من النص
Insert	ادراج نص داخل ال String
Replace	استبدال جزء من النص
Split	تقسيم النص حسب شيء معين إلى مصفوفة ، مثلاً تقسيم النص مع كل علامة (-) إلى مصفوفة جديدة
ToUpper	لتحويل حالة الاحرف بين capital و small

StartsWith: لاختبار السلسلة إذا كانت تبدأ بسلسلة محددة.

EndsWith لاختبار السلسلة إذا كانت تنتهي بسلسلة محددة.

IndexOf: للحصول على دليل محرف في سلسلة نصية.

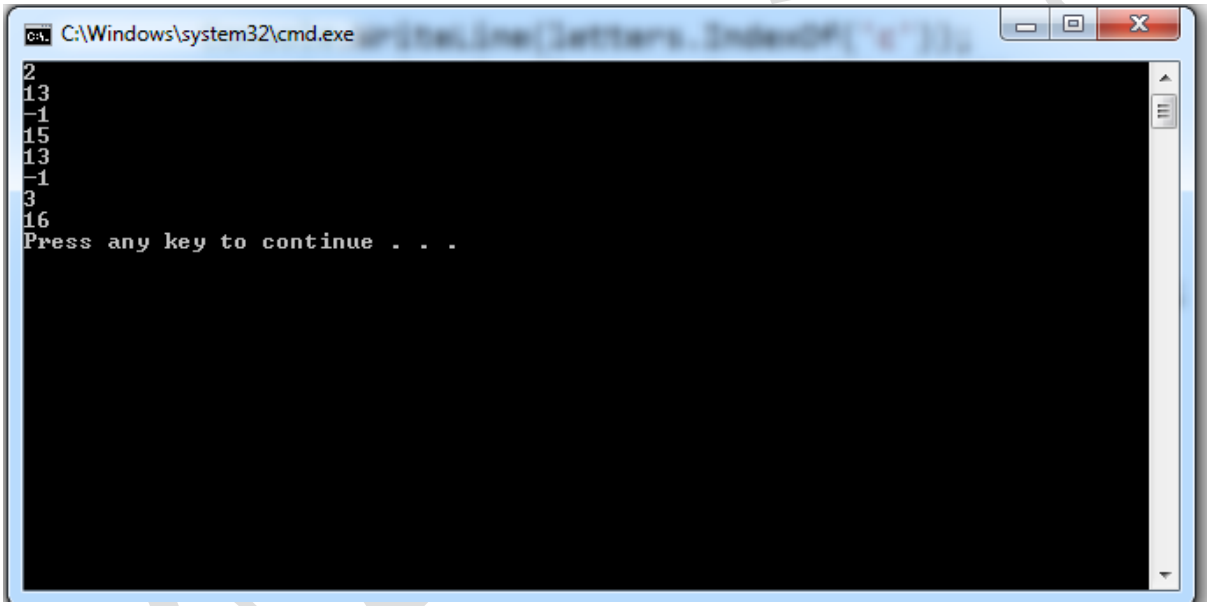
LastIndexOf للحصول على آخر دليل للمحرف في السلسلة.

```
string[] strings = { "started", "starting", "ended", "ending" };
for (int i = 0; i < strings.Length; i++)
    if (strings[i].StartsWith("st"))
        Console.WriteLine(strings[i]);// started-starting


for (int i = 0; i < strings.Length; i++)
    if (strings[i].EndsWith("ed"))
        Console.WriteLine(strings[i]);// started-ended
```

```
static void Main(string[] args)
{
string letters = "abcdefghijklmabcdefghijklm";
//البحث عن دليل المحرف c في السلسة
Console.WriteLine(letters.IndexOf('c'));
// البحث عن دليل المحرف a في السلسة بدءاً من الدليل١
Console.WriteLine(letters.IndexOf('a', 1));
//البحث عن دليل المحرف z في السلسة بدءاً من الدليل3 وبعدد ٥ محارف فقط
Console.WriteLine( letters.IndexOf('z', 3, 5));
// البحث عن آخر دليل للمحرف c في السلسة
Console.WriteLine( letters.LastIndexOf('c'));
// البحث عن آخر دليل للمحرف a في السلسة بدءاً من الدليل٢٥
Console.WriteLine(letters.LastIndexOf('a', 25));
//البحث عن آخر دليل للمحرف z في السلسة بدءاً من الدليل١٥ وبعدد ٥ محارف فقط
Console.WriteLine(letters.LastIndexOf('z', 15, 5));
```

```
//البحث عن دليل السلسلة def في السلسة
Console.WriteLine(  letters.IndexOf("def"));
//البحث عن دليل السلسلة def في السلسة  بدءاً من المحرف ٧
Console.WriteLine(letters.IndexOf("def", 7));
}
```



يمكن الحصول على جزء من سلسلة من مكان وبطول محدد باستخدام التابع Substring:

Substring(مكان الاقتطاع) تعيد سلسلة جديدة من المكان المحدد إلى نهاية السلسلة

Substring(عدد المحارف , مكان الاقتطاع) تعيد سلسلة جديدة من المكان المحدد حسب العدد المحدد

```
string originalString = "Welcome to c# programming";
Console.WriteLine(originalString.Substring(5)); //me to c# programming
Console.WriteLine(originalString.Substring(5, 3)); //me
```

لدمج النصوص نستخدم + أو نستخدم التابع Concate

```
string s1 = "hi ", s2 ="Students";
string s3 = String.Concat(s1, s2);
Console.WriteLine(s3);
```

```
string Test;
Test = "ABCDEFG";
Test += String.Concat('H', "I", "JKLM");
Console.WriteLine(Test);
```

- **لإيجاد طول السلسلة:**

```
string Test;
Test = "ABCDEFG";
Console.WriteLine(Test.Length);// result is 7
```

- **تستخدم الدالة** Split **لتقسيم محتويات النص إلى عناصر في مصفوفة اعتماداً على علامات يتم تحديدها مثل (, أو _ أو أي علامة أخرى).**

```
string StingVar = "hi, how are you, fine";

//تحويل السلسلة من نص إلى مصفوفة نصية كل عنصر فيها معرف واحد
string[] words = StingVar.Split(',');

// طباعة عدد عناصر المصفوفة
Console.WriteLine(words.Length);

// طباعة عناصر المصفوفة
for (int i = 0; i < words.Length; i++)
    Console.WriteLine(words[i]);
```

- **لمقارنة النصوص يمكن استخدام == إلا أنه من الأفضل استخدام الدالة** Equals.

```
string s1 = "hi ", s2 = "HI";
Console.WriteLine(s1.Equals(s2));
```

في حال التساوي يتم إعادة القيمة True وإلا تعاد القيمة False.

- **يمكن استخدام الدالة** CompareTo أيضاً(حسب الترتيب الأبجدي) .

نتيجة المقارنة هي عدد صحيح (١,٠,-١)في حال (أكبر، تساوي، أصغر).

```
string TestA;
TestA = "ABC";
Console.Write("Enter Your Text To Compare ( {0} ) : ", TestA);
string TestB = Console.ReadLine();
int Com = TestA.CompareTo(TestB);
if (Com == 0)
    Console.WriteLine("{0} = {1}", TestA, TestB);
else if (Com == 1)
    Console.WriteLine("{0} > {1}", TestA, TestB);
else if (Com == -1)
    Console.WriteLine("{0} < {1}", TestA, TestB);
```

- التابع Trim()**لحذف الفراغات** Spaceالموجود قبل وبعد الكلمة

```
string Test;
Test = "  ABCDEFG  ";
Test = Test.Trim();
```

- **يمكن استخدام التوابع** ToUpper و ToLower لجعل الحروف صغيرة أو كبيرة:

```
string Test;
Test = "AbCdEfG";
Console.WriteLine(Test.ToUpper() + "\n" + Test.ToLower());
```

- **يمكن استبدال حرف بحرف أو جملة بجملة ضمن السلسلة باستخدام التابع Replace**:

```
string originalString = "Welcome to c# programming";

Console.WriteLine(originalString.Replace('m','M'));
```

- **يمكن اضافة سلسلة فرعية ضمن سلسلة أخرى باستخدام الدالة Insert بعد عدد محدد من الحروف:**

```
string originalString = "Welcome to c# programming";

Console.WriteLine(originalString.Insert(8, "kkk "));//"Welcome kkk to c# programming
```

- **يمكن حذف جزء من سلسلة باستخدام الدالة Remove حسب عدد محدد:**

```
string originalString = "Welcome to c# programming";

Console.WriteLine(originalString.Remove(2,5)); //we to...
```

- **التابع contains للبحث عن سلسلة ضمن سلسلة :**

```
string letters = "abcdefghijklmabcdefghijklm";

Console.WriteLine( letters.Contains("jkl"));//True
Console.WriteLine(letters.Contains("jkl  "));//false
```

المحاضرة الرابعة

لا تخلو أيّ لغة برمجة محترمة من وسيلة لمعالجة الأخطاء. تحتوي لغة سي شارب على آليّة قويّة لمعالجة الأخطاء. تشبه هذه الآلية إلى حدّ ما تلك المستخدمة في لغة Java.

هناك نوعان أساسيّان من الأخطاء التي قد تنتج عن البرنامج: النوع الأوّل هي أخطاء أثناء التصميم، أي أثناء كتابة الشيفرة، وهي أخطاء يمكن اكتشافها بسهولة من خلال مترجم سي شارب الذي يتعرّف عليها ويعطيك وصفًا عنها، وقد يزوّدك أيضًا ببعض المقترحات للتخلّص منها. بالنسبة للنوع الثاني من الأخطاء فهي التي تنتج أثناء تنفيذ البرنامج runtime errors. توجد الكثير من الأسباب لحدوث مثل هذه الأخطاء. فمن القسمة على صفر إلى القراءة من ملف غير موجود أو حتى استخدام متغيّر مصرّح على أنّه من نوع مرجعيّ ولكن لم تتم تهيئته بعد بمرجع إلى كائن. عند حدوث مثل هذه الأخطاء ترمي بيئة التنفيذ المشتركة CLR استثناءً exceptionيحتوي على معلومات بخصوص الخطأ الذي حدث، فإذا كنت مستعدًّا لالتقاط هذا الاستثناء فستنجو، وإلّا سيتوقّف برنامجك عن العمل بصورة مفاجئة.

التقاط استثناء من خلال عبارة try-catch

إذا صادفتك عبارة برمجيّة تحتوي على عمليّة حسابيّة أو على استدعاء لتابع آخر، أو أيّ شيء قد يثير الريبة في نفسك، فمن الممكن مراقبتها أثناء تنفيذ البرنامج باستخدام عبارة try-catch. تتألّف هذه العبارة من قسمين: القسم الأوّل هو قسم المراقبة `try`، والقسم الثاني هو قسم الالتقاط `catch`. الشكل "الأبسط" لهذه العبارة هو التالي :

```
try
{
//عبارة برمجيّة مريبة
}
catch(Exception exp)
{
//هنا يُلتقط الاستثناء وتتمّ معالجته
}
```

يمكن أن يحوي قسم try على عبارات برمجيّة بقدر ما ترغب. عندما يصادف البرنامج أثناء التنفيذ خطأً ما، سيتوقّف التنفيذ عند العبارة التي سبّبت الخطأ، ثم ينتقل فورًا إلى قسم الالتقاط catch. لاحظ معي أنّ قسم catch سيُمرَّر إليه وسيط من الصنف Exception. الصنف Exception موجود ضمن نطاق الاسم System، وهو الصنف الأب لجميع الاستثناءات. إذ أنّ أي استثناء مهما كانت صفته (بما فيها الاستثناءات التي يمكنك أن تكتبها أنت) يجب أن ترث من هذا الصنف .

بعد حدوث الاستثناء والانتقال إلى قسم catch، سيحتوي الوسيط exp على معلومات حول مكان حدوث الاستثناء وسبب حدوثه، وغيرها من المعلومات التي قد تكون مفيدة لمستخدم البرنامج. تجدر الملاحظة بأنّ البرنامج لن يدخل إلى القسم catch أبدًا ما لم يحدث استثناء ضمن القسم try.

لنرى الآن البرنامج Lesson16_01 الذي سيعرّفنا على الاستثناءات بشكل عمليّ.
يحتوي هذا البرنامج البسيط على عبارة try-catch وحيدة سنعمل من خلالها على توليد خطأ بشكل مقصود أثناء التنفيذ، وسنتعلّم كيفيّة المعالجة .

```
using System;
namespace Lesson16_01
{
class Program
{
static void Main(string[] args)
{
int x = 5;
int y = 0;
int result;
12
            try
            {
                result = x / y;
            }
            catch(Exception exp)
```

```
            {
                Console.WriteLine("The following
error has occurred:");
                Console.WriteLine(exp.Message);
            }
            Console.WriteLine("Good Bye!");
        }
    }
}
```

من الواضح أنّ هذا البرنامج يتجّه لأن يجري عمليّة قسمة على صفر في السطر ضمن القسم try. عند وصول البرنامج إلى السطر وإجراء عمليّة القسمة هذه، سيتولّد استثناء يؤدّي إلى انتقال التنفيذ مباشرةً إلى قسم catch في السطر. في قسم catch يعرض البرنامج معلومات عن هذا الخطأ باستخدام الخاصيّة Message لكائن الحدث exp. في النهاية يعرض البرنامج في السطر رسالة توديعيّة للمستخدم .

نفّذ البرنامج لتحصل على الخرج التالي :

```
The following error has occurred:
Attempted to divide by zero.
Good Bye!
```

جرّب الآن تغيير قيمة المتغيّر y لتصبح ١ مثلًا وأعد تنفيذ البرنامج. ستلاحظ ظهور الرسالة التوديعيّة فقط على الشاشة، أي أنّه لم يحدث أي استثناء هذه المرّة. على كلّ الأحوال لا ينصح باستخدام معالجة الاستثناءات من أجل حالة القسمة على صفر في البرنامج السابق .

ملاحظة : في الواقع يمكن الاستغناء عن الوسيط الذي يمرّر إلى قسم catch بشكل كامل، وفي هذه الحالة لن يكون بإمكانك الحصول على معلومات حول الاستثناء المُلتقط. سيبدو شكل عبارة try-catch على الشكل التالي :

```
try
{
//عبارة برمجيّة مريبة
}
catch
{
//هنا يُلتقط الاستثناء وتتمّ معالجته
}
```

من الممكن استخدام هذا الأسلوب إذا كنّا على يقين حول طبيعة الخطأ الذي سيحدث .

عبارة try-catch أكثر تطورا

تصادفنا في بعض الأحيان حالات يكون من الضروري معها مراقبة أكثر من عبارة برمجيّة مريبة ضمن قسم try. لقد اتفقنا أنّه عند حدوث أيّ استثناء ضمن قسم try سينتقل التنفيذ إلى قسم catch. ولكن كيف سنميّز العبارة التي سبّبت هذا الاستثناء في قسم try؟

توجد العديد من الأصناف التي ترث من الصنف Exception والتي يُعتبر كلّ منها استثناءً مخصّصًا أكثر للمشكلة التي قد تحدث. فمثلًا كان من الممكن في البرنامج Lesson16_01 السابق أن نستخدم الصنف DivideByZeroException بدلًا من الصنف Exception في عبارة catch، وذلك لأنّه يرث (بشكل غير مباشر) من الصنف Exception، وسيعمل البرنامج كما هو متوقّع. ولكن في هذه الحالة لن تستطيع catch التقاط سوى الاستثناءات التي تنتج عن القسمة على صفر .

في كثير من الحالات قد تتسبّب العبارات البرمجيّة الموجودة في قسم try باستثناءات متنوّعة لا توجد علاقة فيما بينها. مما يفرض علينا استخدام الصنف Exception لكي نلتقط بشكل مؤكّد أي استثناء قد يصدر عنها، أو أن تساعدنا سي شارب في هذا الخصوص! في الحقيقة الخيار الثاني هو الأفضل وهو جاهز. يمكننا في الواقع إضافة أقسام catch أخرى بقدر ما نرغب بعد قسم try. سيوضّح البرنامج Lesson16_02 هذه الفكرة من خلال فتح ملف نصي ومحاولة قراءة محتوياته. لاحظ أنّنا سنستخدم هنا نطاق الاسم System.IO.

```
using System;
using System.IO;
namespace Lesson16_02
{
    class Program
    {
        static void Main(string[] args)
        {
            string contents;
            Try
            {
    using (StreamReader sr = new
StreamReader("myfile.txt"))
                {
            contents = sr.ReadLine();
                }
          Console.WriteLine("This file contains
{0} characters.", contents.Length);
```

```
            }
     catch(NullReferenceException nullExp)
            {
                Console.WriteLine("The file does
not contain any data.");
            }
  catch(FileNotFoundException notFoundExp)
            {
 Console.WriteLine("File: {0} not found!",
notFoundExp.FileName);
            }
        }
    }
}
```

يستخدم هذا البرنامج قسميّ .catchالقسم الأوّل (الأسطر من ٢١ إلى ٢٤) يلتقط استثناءً من النوع NullReferenceExceptionوهذا يحدث عند محاولة استدعاء تابع أو خاصيّة من متغيّر يحتوي على nullبدلًا من مرجع لكائن حقيقي. أمّا القسم الثاني (الأسطر من ٢٥ إلى ٢٨) فهو يلتقط استثناءً من النوع FileNotFoundExceptionوالذي يحدث عند محاولة القراءة من ملف غير موجود. نفّذ البرنامج السابق وستحصل على الرسالة التالية في الخرج :

```
File: C:\\Users\Husam\documents\visual studio
2015\Projects\Lesson16_02\Lesson16_02\bin\Debug\myfile
.txt not found!
```

وهذا طبيعي تمامًا لأنّني لم أنشئ الملف myFile.txtفي هذا المسار. لاحظ كيف يضيف الصنف FileNotFoundExceptionخاصيّة جديدة له وهي FileName) السطر) من النوع stringالتي تحوي مسار الملف مع اسمه .

أنشئ الآن الملف myFile.txtواتركه فارغًا، ثمّ ضعه ضمن نفس المجلّد الذي يحوي الملف التنفيذي للبرنامج (موجود ضمن bin\Debug\).

أعد تنفيذ البرنامج وستحصل على الرسالة التالي :

```
The file does not contain any data.
```

السبب في ظهور هذه الرسالة هو الاستثناء NullReferenceExceptionوذلك لأنّنا حاولنا الوصول إلى الخاصيّة (Lengthتعطينا عدد المحارف الموجودة ضمن متغيّر

نصي) من المتغيّر النصي contents رغم أنّه يحتوي على) nullتذكّر بأنّنا تركنا الملف myFile.txtفارغًا .(

اذهب إلى الملف myFile.txtالذي أنشأناه قبل قليل، واكتب بعض الكلمات ضمنه واحفظ الملف، ثمّ أعد تنفيذ البرنامج Lesson16_02من جديد. يجب الآن أن تحصل على رسالة تخبرك بعدد الأحرف التي كتبتها ضمن الملف .

ملاحظة :يجب الانتباه إلى ترتيب أقسام .catchفلو كان مثلًا أوّل قسم catchموجود بعد قسم tryيلتقط استثناءً من النوع Exceptionفعندها لن يستطيع أي قسم لاحق التقاط أي استثناء، لأنّ جميع الاستثناءات سيلتقطها هذا القسم الأوّل. السبب في ذلك أنّ الصنف Exceptionهو الأب العام لجميع أصناف الاستثناءات الأخرى، فيمكن له التقاطها .

عبارة try-catch-final

يمكن إضافة قسم أخير لعبارة try-catch اسمه .finalوكما يوحي اسمه، فهذا القسم يمكن له أن يحتوي على عبارات برمجيّة سيتمّ تنفيذها بعد أن يدخل البرنامج إلى القسم try العائد له. وذلك سواءً أحدث استثناء ضمن tryأم لم يحدث. تكمن فائدة وجود هذا القسم، في أنّه قد نواجه أحيانًا بعض الحالات التي تتطلّب إجراء بعض المهام عندما نفرغ من قسم try مثل إغلاق بعض المصادر المفتوحة، أو تحرير الذاكرة بشكل فوري وغيرها .الشرط الوحيد لاستخدام هذا القسم الاختياري هو أن يكون آخر قسم في عبارة .try-catchسنعدّل البرنامج Lesson16_02ليدعم القسم .finalانظر البرنامج .Lesson16_03

```
using System;
using System.IO;
namespace Lesson16_03
{
class Program
{
static void Main(string[] args)
{
string contents;

try
{
using (StreamReader sr =
new StreamReader("myfile.txt"))
{
```

```
contents =
sr.ReadLine();
}
Console.WriteLine("This
file contains {0} characters.", contents.Length);
}
catch(NullReferenceException
nullExp)
{
Console.WriteLine("The file
does not contain any data.");
}
catch(FileNotFoundException
notFoundExp)
{
Console.WriteLine("File:
{0} not found!", notFoundExp.FileName);
}
finally
{
Console.WriteLine("Good Bye!");
}
}
}
}
```

سواءً كان الملف myFile.txtموجودًا أم غير موجود، أو كان يحتوي على بيانات أم فارغاً، ستظهر العبارة Good Bye! على الشاشة .

ملاحظة :من الأفضل دومًا أن تحاول عدم استخدام عبارة try-catch وأن تستخدم عبارة if لاختبار الحالات التي تواجهك قبل تنفيذها. استخدم try-catch إذا كان ذلك ضروريًّا. والسبب في ذلك أنّ عمليّة معالجة الأخطاء بشكل عام تتطلّب المزيد من الموارد المخصّصة للبرنامج، مما قد يؤثّر على أداء البرنامج في حال تمّ استخدامها بشكل غير مدروس .

تمارين داعمة

تمرين ١

عدّل البرنامج Lesson16_02بحيث يمكن الاستغناء عن عبارة try-catch تمامًا .

)**تلميح** :ستحتاج إلى استخدام عبارتيّ `if` في هذه الحالة .(

تمرين ٢

لتكن لدينا الشيفرة التالية :

```
string input = Console.ReadLine();

int t = int.Parse(input);
```

تطلب الشيفرة السابقة من المستخدم أن يدخل عددًا على شكل نص لتعمل على تحويله إلى قيمة عدديّة باستخدام التابع `int.Parse`.

المطلوب هو إضافة عبارة try-catch إلى الشيفرة السابقة لمعالجة استثناء ممكن الحدوث في حال أدخل المستخدم قيمة مثل "u" وهي لا يمكن تحويلها إلى قيمة عدديّة كما هو واضح .

)**تلميح** :استخدام الاستثناء `FormatException` الذي يُعبّر عن هذه الحالة .(