Asynchronous Programming

A teraz programowanie asynchroniczne. Dlaczego tego potrzebujemy. Jest bardzo dobry powód, dla którego powinieneś wiedzieć coś o tym. Rozważ następujący przypadek pisania programu, który zajmuje się bardzo dużymi plikami lub komunikuje się z bazami danych lub po prostu wykonuje zadanie, którego wykonanie zajmuje dużo czasu. Gdy zaczniesz pracę z plikiem lub bazą danych, czy czymkolwiek podobnym, to będziesz mieć opóźnienia ponieważ te zadania zajmują dużo czasu.
A to sprawi, że Twoja aplikacja nie będzie odpowiadać.
W takich przypadkach bardzo dobrze jest wiedzieć, jak zastosować proste programowanie asynchroniczne, przynajmniej aby wyświetlić komunikat na konsoli i nie powodować, że nasza aplikacja nie odpowiada.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

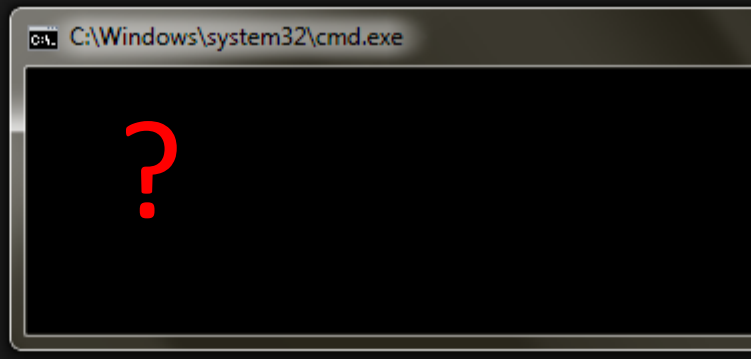
Debug | Any CPU | Rozpocznij

Eksplorator serwera | Przybornik

EntryPoint.cs

AsynchronousProgramming | AsynchronousProgramming.EntryPoint | Main()

```
1   using System;
2   using System.Threading.Tasks;
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              ConcatenateChars();
11              Console.WriteLine("Completed!");
12          }
13
14          public static string ConcatenateChars()
15          {
16              string concatenatedString = string.Empty;
17
18              for (int i = 0; i < 400000; i++)
19              {
20                  concatenatedString += 'a';
21              }
22
23              return concatenatedString;
24          }
25      }
26  }
27
28
29
30
31
32
```



Zacznijmy od bardzo prostego przykładu.
Stwórzmy metodę, która łączy bardzo długi ciąg.
To zajmuje tyle czasu.
W ramach tej metody utworzymy nowy łańcuch połączony.
Stworzymy pętlę, aby połączyć te napisy postać setki tysięcy razy w ten sposób.
Jeśli użyjemy tej metody teraz, a potem chcemy od razu wydrukować wiadomość, to efekt jest taki, że przez bardzo długi czas nie zobaczymy efektu działania programu.
Obecnie nasza metoda nie działa w tle.
Działa i po zakończeniu metody otrzymujemy komunikat.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug Any CPU Rozpocznij

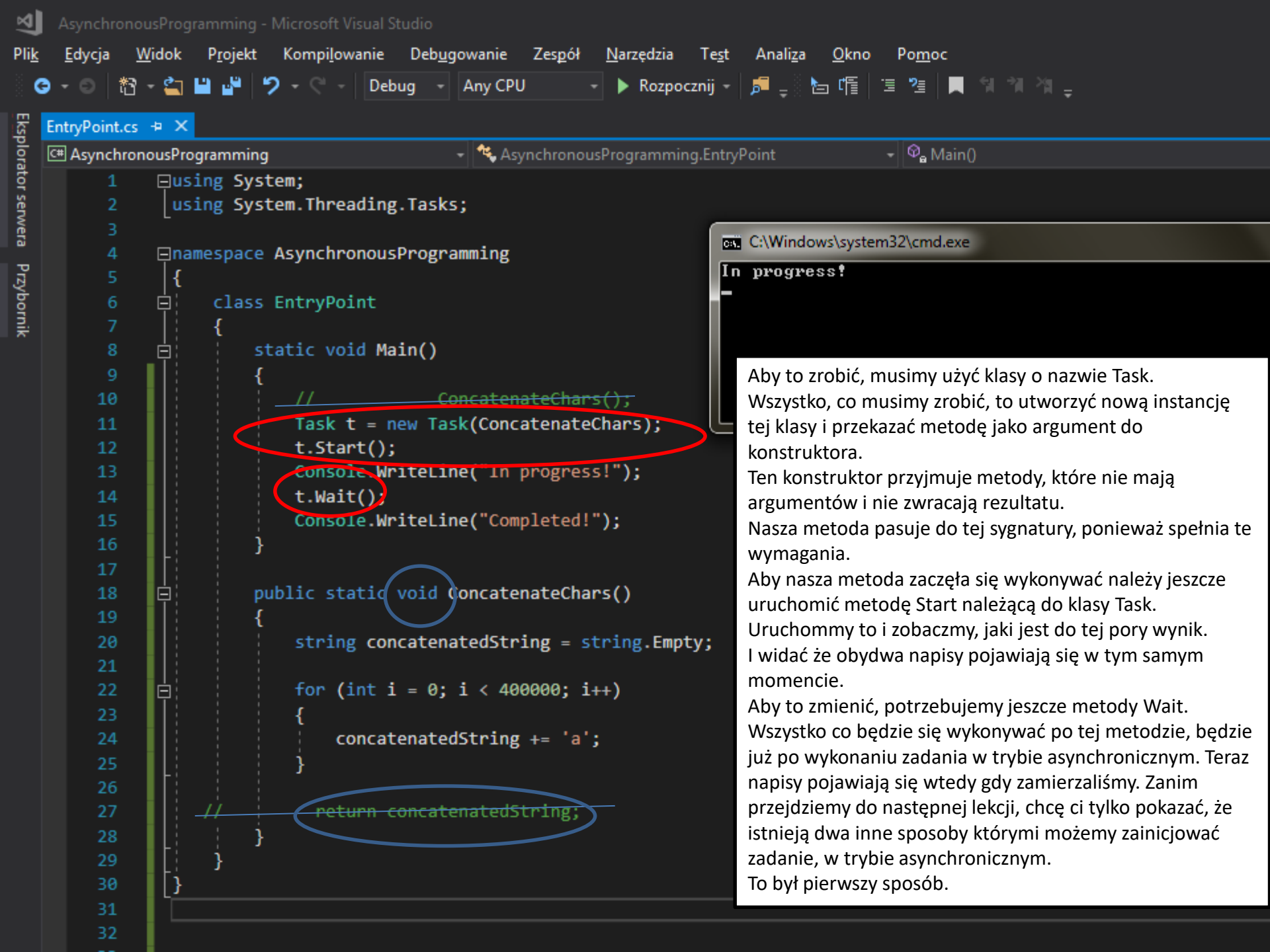
Eksplorator serwera Przybornik

EntryPoint.cs*

AsynchronousProgramming AsynchronousProgramming.EntryPoint Main()

```
1   using System;
2   using System.Threading.Tasks;
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              ConcatenateChars();
11              Console.WriteLine("In progress!"); // how to do this
12              Console.WriteLine("Completed!");
13          }
14
15          public static string ConcatenateChars()
16          {
17              string concatenatedString = string.Empty;
18
19              for (int i = 0; i < 400000; i++)
20              {
21                  concatenatedString += 'a';
22              }
23
24              return concatenatedString;
25          }
26      }
27  }
28
29
30
31
32
```

Co jeśli chcemy wyświetlić komunikat w toku? Ta metoda wciąż działa, więc jeśli chcemy napisać, że jest w toku, a następnie ją ukończyć, to oczywiście nie zadziała tak jak powinna i zobaczymy obie wiadomości jednocześnie.
Chcemy jednak, abyśmy zostali powiadomieni o tym, że zadanie jest w toku.


AsynchronousProgramming - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
Rozpocznij
EntryPoint.cs
AsynchronousProgramming
AsynchronousProgramming.EntryPoint
Main()
Eksplorator serwera
Przybornik
1 using System;
2 using System.Threading.Tasks;
3
4 namespace AsynchronousProgramming
5 {
6 class EntryPoint
7 {
8 static void Main()
9 {
10 // ConcatenateChars();
11 Task t = new Task(ConcatenateChars);
12 t.Start();
13 Console.WriteLine("In progress!");
14 t.Wait();
15 Console.WriteLine("Completed!");
16 }
17
18 public static void ConcatenateChars()
19 {
20 string concatenatedString = string.Empty;
21
22 for (int i = 0; i < 400000; i++)
23 {
24 concatenatedString += 'a';
25 }
26
27 // return concatenatedString;
28 }
29 }
30 }
31
32
C:\Windows\system32\cmd.exe
In progress!
Aby to zrobić, musimy użyć klasy o nazwie Task.
Wszystko, co musimy zrobić, to utworzyć nową instancję tej klasy i przekazać metodę jako argument do konstruktora.
Ten konstruktor przyjmuje metody, które nie mają argumentów i nie zwracają rezultatu.
Nasza metoda pasuje do tej sygnatury, ponieważ spełnia te wymagania.
Aby nasza metoda zaczęła się wykonywać należy jeszcze uruchomić metodę Start należącą do klasy Task.
Uruchommy to i zobaczmy, jaki jest do tej pory wynik.
I widać że obydwa napisy pojawiają się w tym samym momencie.
Aby to zmienić, potrzebujemy jeszcze metody Wait.
Wszystko co będzie się wykonywać po tej metodzie, będzie już po wykonaniu zadania w trybie asynchronicznym. Teraz napisy pojawiają się wtedy gdy zamierzaliśmy. Zanim przejdziemy do następnej lekcji, chcę ci tylko pokazać, że istnieją dwa inne sposoby którymi możemy zainicjować zadanie, w trybie asynchronicznym.
To był pierwszy sposób.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug Any CPU Rozpocznij

Eksplorator serwera Przybornik

EntryPoint.cs

AsynchronousProgramming AsynchronousProgramming.EntryPoint Main()

```
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              //              Task t = new Task(ConcatenateChars);
11              //              t.Start();
12              Task t = Task.Factory.StartNew(ConcatenateChars);
13              Console.WriteLine("In progress!");
14              t.Wait();
15              Console.WriteLine("Completed!");
16          }
17
18          public static void ConcatenateChars()
19          {
20              string concatenatedString = string.Empty;
21
22              for (int i = 0; i < 400000; i++)
23              {
24                  concatenatedString += 'a';
25              }
26          }
27      }
28  }
29
30
31
32
33
34
```

second method

Drugi sposób to skorzystanie z fabryki zadań. Więc nie musimy już tworzyć nowego zadania za pomocą new i uruchamiać konstruktora a następnie metody Start. Po prostu uruchamiamy nowe zadanie w fabryce zadań. Możemy zobaczyć, że działa ono dokładnie tak samo, jak pierwszym sposobem. Tutaj również można użyć metody Wait.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug | Any CPU | Rozpocznij

EntryPoint.cs

AsynchronousProgramming | AsynchronousProgramming.EntryPoint | Main()

```
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              //              Task t = new Task(ConcatenateChars);
11              //              t.Start();
12              //              Task t = Task.Factory.StartNew(ConcatenateChars);
13              Task t = Task.Run(new Action(ConcatenateChars));
14              Console.WriteLine("In progress!");
15              t.Wait();
16              Console.WriteLine("Completed!");
17          }
18
19          public static void ConcatenateChars()
20          {
21              string concatenatedString = string.Empty;
22
23              for (int i = 0; i < 400000; i++)
24              {
25                  concatenatedString += 'a';
26              }
27          }
28      }
29  }
30
31
32
33
34
```

third method

Trzecim sposobem jest wykorzystanie akcji. Więc metoda Run i new Action z parametrem który jest metodą, którą chcemy uruchomić. W dalszej części będziemy korzystać z drugiego sposobu czyli fabryki zadań.

Using Task on methods with arguments

W porządku, udało ci się przejść przez asynchroniczne wywoływanie metod, ale nie jest to zbyt przydatny sposób.
Nie otrzymujemy żadnych wyników i nie jesteśmy w stanie użyć żadnych argumentów.
Powiedzmy, że chcemy zmienić ilość konkatenacji - zrobimy to, tworząc argumenty wejściowe dla metody.
Więc zróbmy to.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug | Any CPU | Rozpocznij

EntryPoint.cs

AsynchronousProgramming | AsynchronousProgramming.EntryPoint | Main()

```
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              Task t = Task.Factory.StartNew(() =>
11              {
12                  ConcatenateChars('a', 400000);
13              });
14              Console.WriteLine("In progress!");
15              t.Wait();
16              Console.WriteLine("Completed!");
17          }
18
19          public static void ConcatenateChars(char charToConcatenate, int count)
20          {
21              string concatenatedString = string.Empty;
22
23              for (int i = 0; i < count; i++)
24              {
25                  concatenatedString += charToConcatenate;
26              }
27          }
28      }
29  }
30
31
32
33
34
```

C:\Windows\system32\cmd.exe

```
In progress!
```

Jeżeli utworzymy metodę z dwoma parametrami. Pierwszy parametr wskazuje znak, który będziemy dodawać, a drugi parametr ustali ile razy ma się wykonać ta operacja. Otrzymamy wtedy błąd. Zasadniczo oznacza to, że nasza metoda nie odpowiada już sygnaturze akcji, która jest wymagana do użycia. Aby ominąć ten problem, możemy użyć wyrażenia Lambda i nazwiemy tę metodę anonimową metodą. Więc stworzymy normalną metodę, która będzie w stanie wywołać inną metodę. Wygląda to mniej więcej tak. Używamy pustych nawiasów, ponieważ nie musimy mieć żadnych argumentów wejściowych, aby zadowolić delegata Action

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug Any CPU Rozpocznij

Eksplorator serwera Przybornik

EntryPoint.cs

AsynchronousProgramming AsynchronousProgramming.EntryPoint Main()

```
3
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              Task t = Task.Factory.StartNew(() => ConcatenateChars('a', 400000));
11              Console.WriteLine("In progress!");
12              t.Wait();
13              Console.WriteLine("Completed!");
14          }
15
16          public static void ConcatenateChars(char charToConcatenate, int count)
17          {
18              string concatenatedString = string.Empty;
19
20              for (int i = 0; i < count; i++)
21              {
22                  concatenatedString += charToConcatenate;
23              }
24          }
25      }
26  }
27
28
29
30
31
32
33
34
```

Ponieważ mamy tutaj tylko jeden wiersz kodu, po prostu stworzyliśmy metodę, która pozwala nam wywoływać naszą drugą metodę bez ograniczeń.
Nie musimy używać nawiasów klamrowych, więc możemy to zrobić w jednej linii. I będzie nadal działać.

Using Task on methods with return type

Nauczyliśmy się, jak wywoływać metodę asynchronicznie i jak korzystać z metody, która ma konkretne argumenty wywołania. Kolejnym logicznym krokiem jest dla nas odzyskanie niektórych danych z tej metody.

AsynchronousProgramming - Microsoft Visual Studio

Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug | Any CPU | Rozpocznij

EntryPoint.cs

AsynchronousProgramming | AsynchronousProgramming.EntryPoint | Main()

```
1  using System;
2  using System.Threading.Tasks;
3
4  namespace AsynchronousProgramming
5  {
6      class EntryPoint
7      {
8          static void Main()
9          {
10             Task<string> t = Task.Factory.StartNew(() => ConcatenateChars('a', 400000));
11             Console.WriteLine("In progress!");
12             t.Wait();
13             Console.WriteLine("Completed!");
14             Console.WriteLine("The length of the result is " + t.Result.Length);
15         }
16
17         public static string ConcatenateChars(char charToConcatenate, int count)
18         {
19             string concatenatedString = string.Empty;
20
21             for (int i = 0; i < count; i++)
22             {
23                 concatenatedString += charToConcatenate;
24             }
25             return concatenatedString;
26         }
27     }
28 }
29
30
31
32
```

C:\Windows\system32\cmd.exe

```
In progress!
Completed!
The length of the result is 400000
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Jeśli jest to metoda, która coś zwraca. Np - ciąg znaków, który chcemy później użyć w naszym kodzie. Musimy sprawić, by nasza metoda zwracała ciąg znaków. Wymaga to również, aby nasze zadanie miało ciąg znaków typu generycznego, ponieważ zadanie, które nie zawiera typów, jest zadaniem który jest używany w przypadku metod, które są typu void. Jeśli uruchomimy teraz kod, nadal będzie działał, ale nie będziemy mieli dostępu do danych wyjściowych metody. Tak więc mamy wynik, ale jeszcze nie potrafimy z niego skorzystać. Więc kiedy masz metodę, która zwraca wynik, jedyne, co musisz zrobić, to wykonać zadanie i odczytać właściwość Result z obiektu klasy Task. Możemy go odczytać dopiero po zakończeniu wykonywania metody, czyli po metodzie Wait.

The async and await keywords

Do tego momentu masz już wszystkie elementy składowe, dzięki którym twoje metody będą asynchroniczne.
Wiesz, jak wywoływać metody, przekazywać argumenty i uzyskiwać wyniki.
W tej części wykładu poznasz słowa kluczowe, które po prostu ułatwią ci życie w tworzeniu asynchronicznego kodu.
Zróbmy to i zobaczmy, dokąd nas to zaprowadzi.

Użyjmy tutaj słowa kluczowego async w deklaracji naszej metody, zobaczymy, że wystąpił błąd - metody asynchroniczne muszą być void. Utwórzmy więc nowe zadanie, które będzie platformą do asynchronicznego wywołania naszej metody. Teraz musimy wypełnić anonimową metodę. Możesz zobaczyć tutaj, że po prostu używamy tej samej metody. Po prostu zamierzamy przenieść ten kod do naszej fabryki zadań. Pamiętaj więc, że musimy zachować słowo kluczowe return, ponieważ chcemy zwrócić ciąg znaków. Ponieważ chcemy złapać nasz wynik dopiero po ukończeniu działania metody - wykorzystamy do tego słowo kluczowe await. Zasadniczo utworzyliśmy coś w rodzaju bańki.

```
4   namespace AsynchronousProgramming
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              int count = 200000;
11              char charToConcatenate = '1';
12              Task<string> t = ConcatenateCharsAsync(charToConcatenate, count);
13              Console.WriteLine("In progress");
14              Console.WriteLine("The length of the result is " + t.Result.Length);
15              string text = ConcatenateChars(charToConcatenate, count);
16              Console.WriteLine("Synchronously - The length: " + text.Length);
17          }
18
19          public static string ConcatenateChars(char charToConcatenate, int count)
20          {
21              string concatenatedString = string.Empty;
22              for (int i = 0; i < count; i++)
23              {
24                  concatenatedString += charToConcatenate;
25              }
26              return concatenatedString;
27          }
28
29          public async static Task<string> ConcatenateCharsAsync(char charToConcatenate, int count)
30          {
31              return await Task<string>.Factory.StartNew(() =>
32              {
33                  return ConcatenateChars(charToConcatenate, count);
34              });
35          }
```

```
C:\Windows\system32\cmd.exe
In progress
The length of the result is 200000
Synchronously - The length: 200000
Aby kontynuować, naciśnij dowolny klawisz . . .
```

asynchronously

synchronously

Możemy wywołać naszą metodę w sposób asynchroniczny i synchroniczny. Jak widać w czasie działania postaci asynchronicznej możemy wykonywać inne zadania. Jednakże w momencie gdy zamierzamy odczytać właściwość Result zadanie oczekuje do ukończenia obliczeń. Jak widać uzyskane wyniki są takie same.