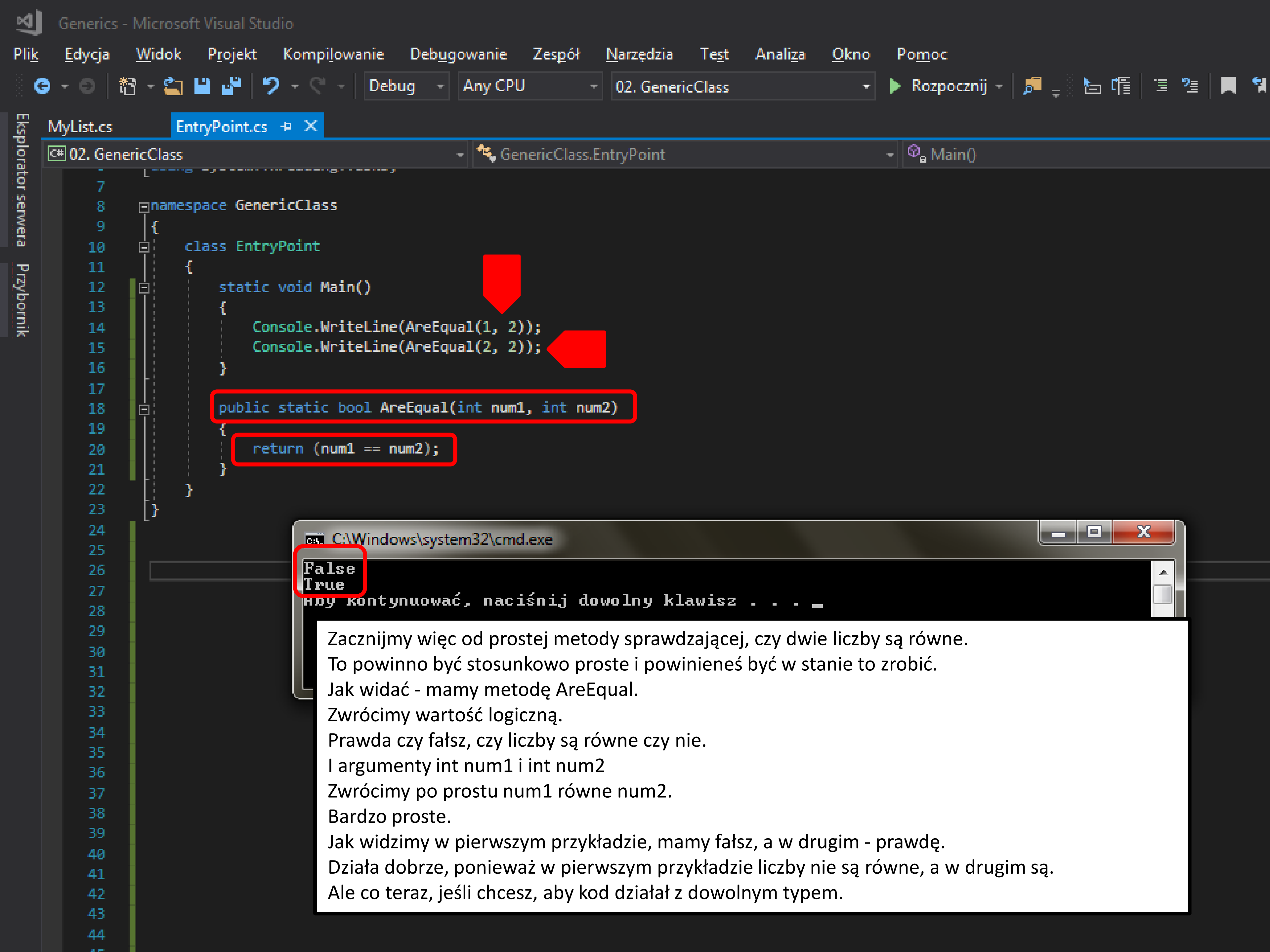
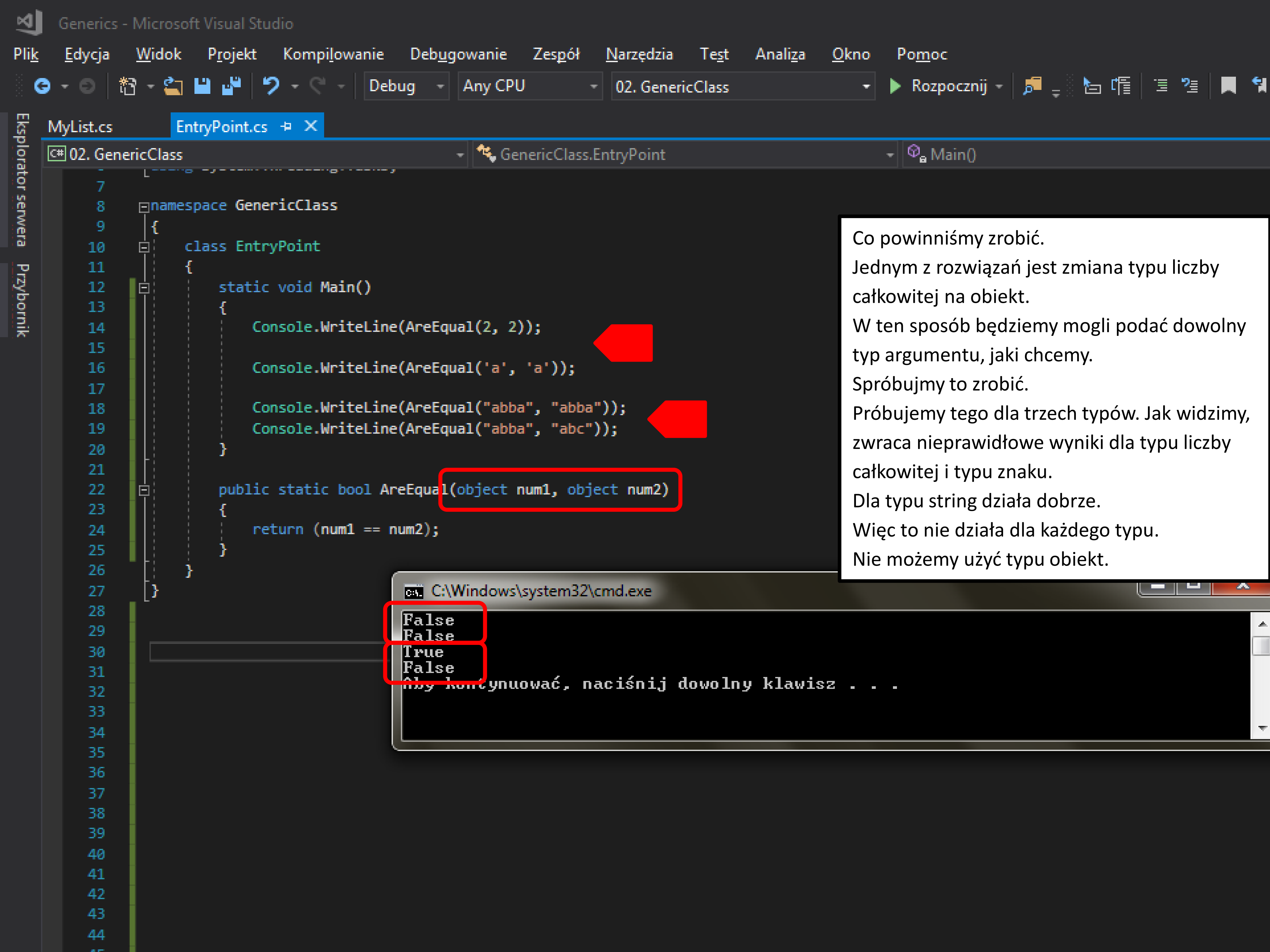
Generic methods

W końcu jesteśmy w metodach generycznych (ogólnych).
Z początku wygląda to na skomplikowane.
Ale w rzeczywistości tak nie jest.
Oczywiście ogólnie, generyka oznacza, że dotyczy więcej niż jednego typu.
Tak więc metoda lub klasa, która jest generyczna - może działać zarówno dla typu liczb całkowitych, jak i typu string oraz dla dowolnego innego typu.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug Any CPU 02. GenericClass Rozpocznij
MyList.cs EntryPoint.cs
02. GenericClass GenericClass.EntryPoint Main()
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(1, 2));
Console.WriteLine(AreEqual(2, 2));
}
public static bool AreEqual(int num1, int num2)
{
return (num1 == num2);
}
}
}
C:\Windows\system32\cmd.exe
False
True
Aby kontynuować, naciśnij dowolny klawisz . . .
Zacznijmy więc od prostej metody sprawdzającej, czy dwie liczby są równe.
To powinno być stosunkowo proste i powinieneś być w stanie to zrobić.
Jak widać - mamy metodę AreEqual.
Zwrócimy wartość logiczną.
Prawda czy fałsz, czy liczby są równe czy nie.
I argumenty int num1 i int num2
Zwrócimy po prostu num1 równe num2.
Bardzo proste.
Jak widzimy w pierwszym przykładzie, mamy fałsz, a w drugim - prawdę.
Działa dobrze, ponieważ w pierwszym przykładzie liczby nie są równe, a w drugim są.
Ale co teraz, jeśli chcesz, aby kod działał z dowolnym typem.



Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(2, 2));
Console.WriteLine(AreEqual('a', 'a'));
Console.WriteLine(AreEqual("abba", "abba"));
Console.WriteLine(AreEqual("abba", "abc"));
}
public static bool AreEqual(object num1, object num2)
{
return (num1 == num2);
}
}
}
C:\Windows\system32\cmd.exe
False
False
True
False
Aby kontynuować, naciśnij dowolny klawisz . . .
Co powinniśmy zrobić.
Jednym z rozwiązań jest zmiana typu liczby całkowitej na obiekt.
W ten sposób będziemy mogli podać dowolny typ argumentu, jaki chcemy.
Spróbujmy to zrobić.
Próbujemy tego dla trzech typów. Jak widzimy, zwraca nieprawidłowe wyniki dla typu liczby całkowitej i typu znaku.
Dla typu string działa dobrze.
Więc to nie działa dla każdego typu.
Nie możemy użyć typu obiekt.

Generics - Microsoft Visual Studio

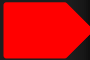
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc

Debug | Any CPU | 02. GenericClass | Rozpocznij

MyList.cs | EntryPoint.cs

02. GenericClass | GenericClass.EntryPoint | Main()

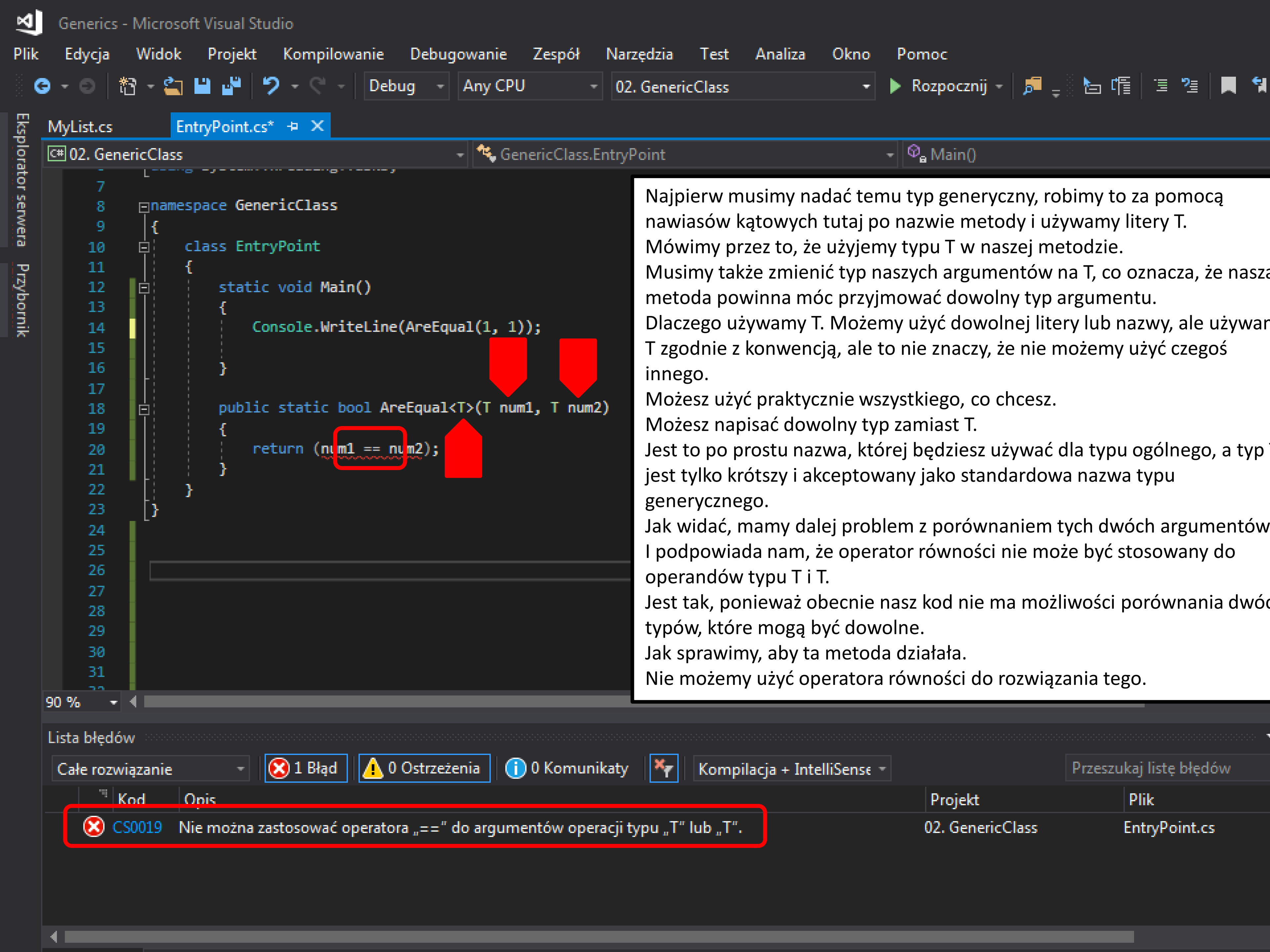
```
namespace GenericClass
{
    class EntryPoint
    {
        static void Main()
        {
            Console.WriteLine(AreEqual(1, "abc"));

        }

        public static bool AreEqual(object num1, object num2)
        {
            return (num1 == num2);
        }
    }
}
```

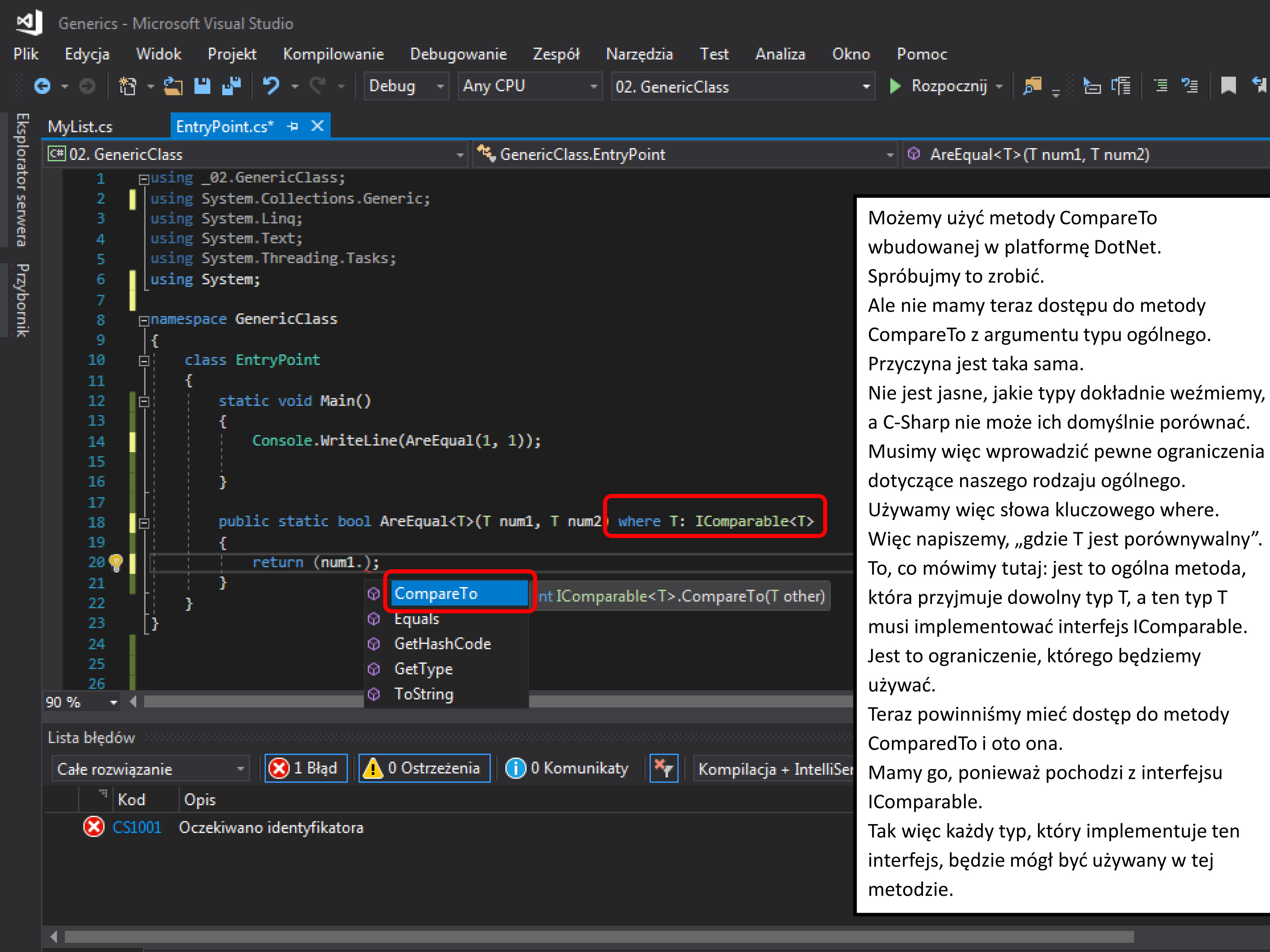


C:\Windows\system32\cmd.exe

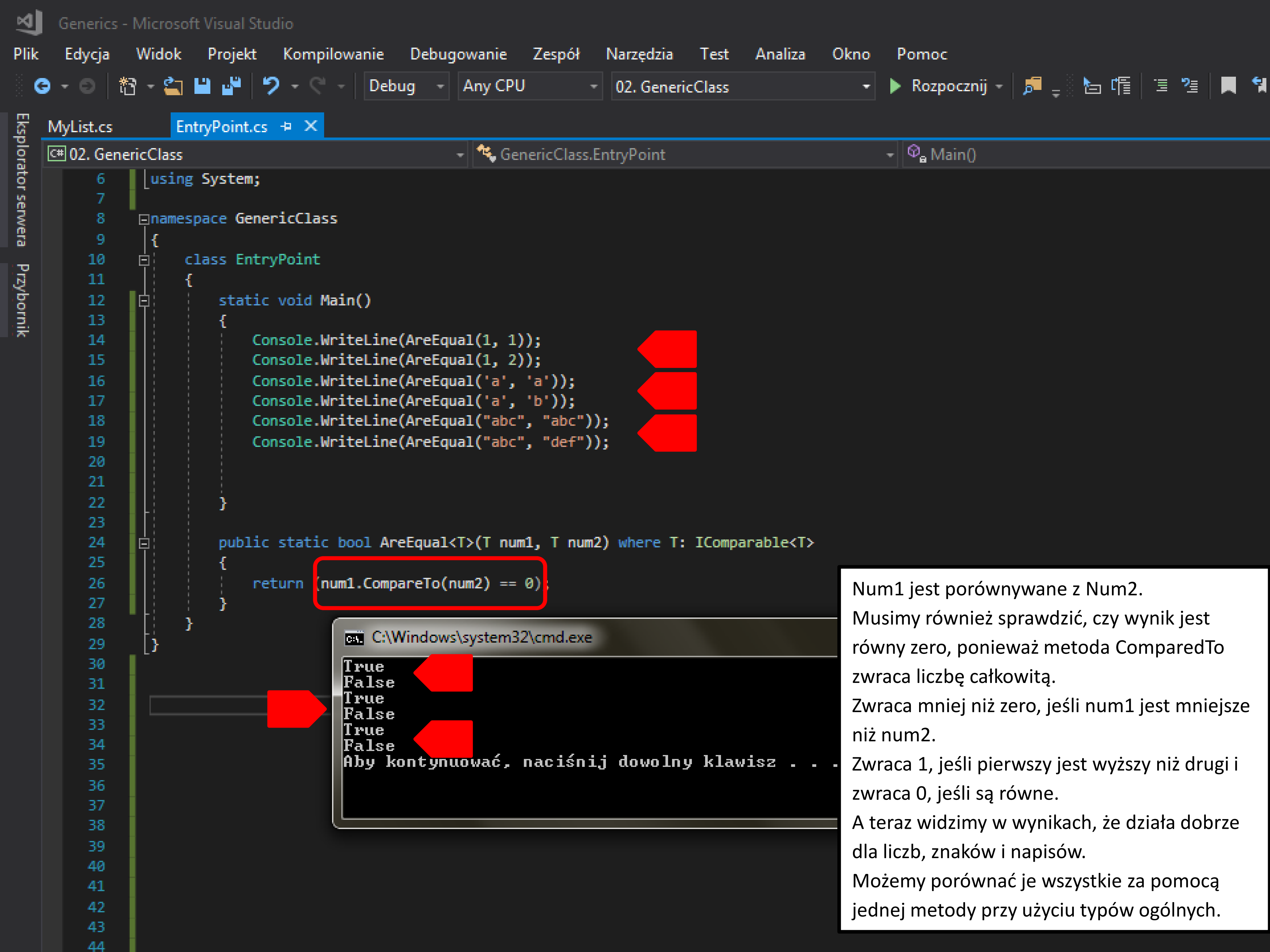
```
False
Aby kontynuować, naciśnij dowolny klawisz . . .
```

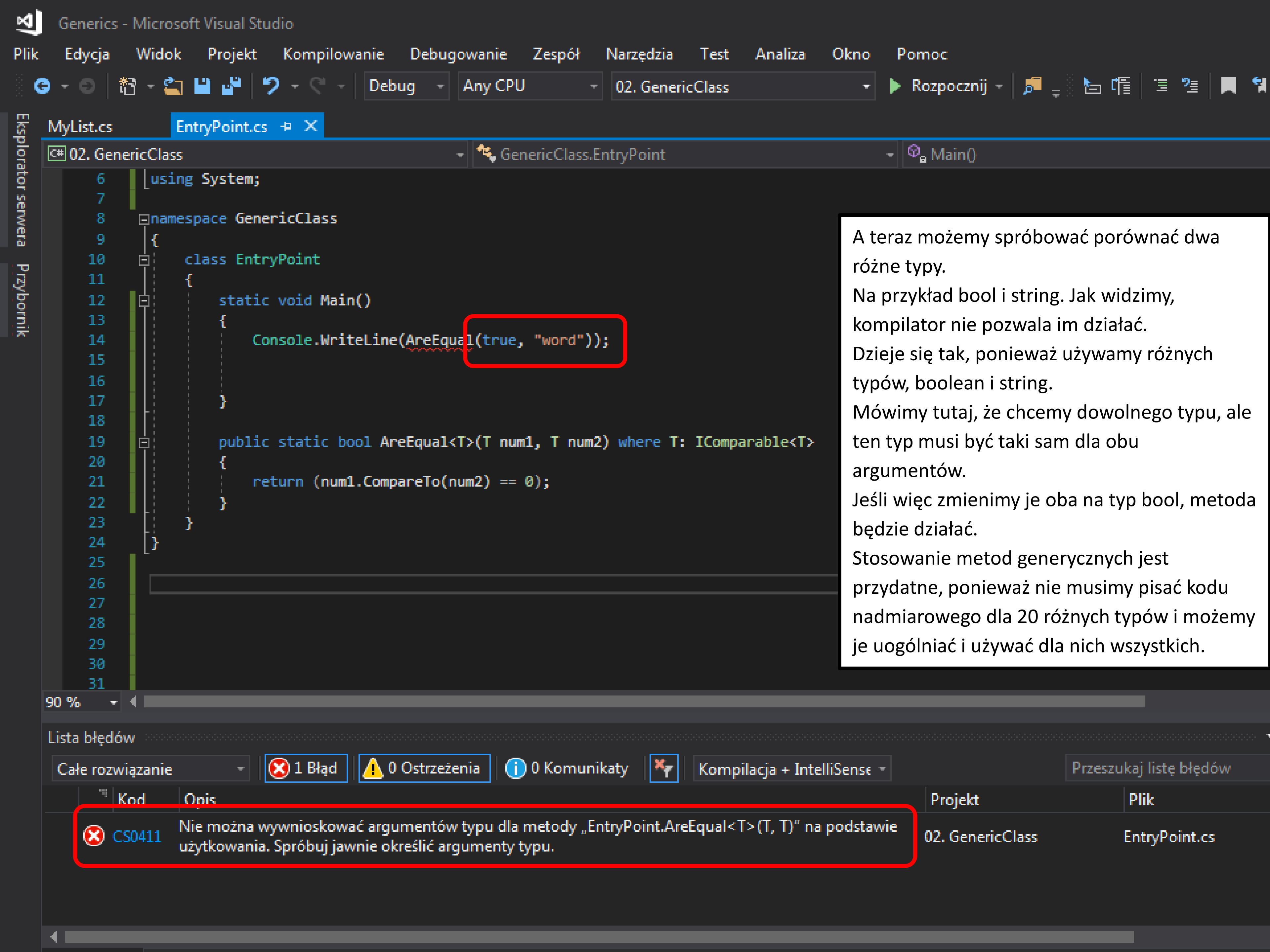
I jak widzimy. Kiedy próbujemy porównać dwa różne typy.
Na przykład liczba i napis - metoda próbuje go rozwiązać.
Kompilator nie może rozpoznać, że istnieją dwa różne typy - niekompatybilne i nieporównywalne.
Nie można ich porównywać, ponieważ liczby są liczbami a napisy lub napisami.
Nie ma możliwości ich porównania.
Musimy więc wprowadzić kilka zmian w naszej metodzie, aby działała z dowolnym typem i działała tak jak powinna.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs*
02. GenericClass
GenericClass.EntryPoint
Main()
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(1, 1));
}
public static bool AreEqual<T>(T num1, T num2)
{
return (num1 == num2);
}
}
}
Najpierw musimy nadać temu typ generyczny, robimy to za pomocą
nawiasów kątowych tutaj po nazwie metody i używamy litery T.
Mówimy przez to, że użyjemy typu T w naszej metodzie.
Musimy także zmienić typ naszych argumentów na T, co oznacza, że nasza
metoda powinna móc przyjmować dowolny typ argumentu.
Dlaczego używamy T. Możemy użyć dowolnej litery lub nazwy, ale używam
T zgodnie z konwencją, ale to nie znaczy, że nie możemy użyć czegoś
innego.
Możesz użyć praktycznie wszystkiego, co chcesz.
Możesz napisać dowolny typ zamiast T.
Jest to po prostu nazwa, której będziesz używać dla typu ogólnego, a typ
jest tylko krótszy i akceptowany jako standardowa nazwa typu
generycznego.
Jak widać, mamy dalej problem z porównaniem tych dwóch argumentów
I podpowiada nam, że operator równości nie może być stosowany do
operandów typu T i T.
Jest tak, ponieważ obecnie nasz kod nie ma możliwości porównania dwóch
typów, które mogą być dowolne.
Jak sprawimy, aby ta metoda działała.
Nie możemy użyć operatora równości do rozwiązania tego.
90 %
Lista błędów
Całe rozwiązanie
1 Błąd
0 Ostrzeżenia
0 Komunikaty
Kompilacja + IntelliSense
Przeszukaj listę błędów
Kod Opis Projekt Plik
CS0019 Nie można zastosować operatora „==" do argumentów operacji typu „T" lub „T". 02. GenericClass EntryPoint.cs



Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs*
02. GenericClass
GenericClass.EntryPoint
AreEqual<T>(T num1, T num2)
using _02.GenericClass;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(1, 1));
}
public static bool AreEqual<T>(T num1, T num2) where T: IComparable<T>
{
return (num1.);
}
}
}
CompareTo
Equals
GetHashCode
GetType
ToString
int IComparable<T>.CompareTo(T other)
90 %
Lista błędów
Całe rozwiązanie
1 Błąd
0 Ostrzeżenia
0 Komunikaty
Kompilacja + IntelliSen
Kod
Opis
CS1001
Oczekiwano identyfikatora
Eksplorator serwera
Przybornik
Możemy użyć metody CompareTo wbudowanej w platformę DotNet.
Spróbujmy to zrobić.
Ale nie mamy teraz dostępu do metody CompareTo z argumentu typu ogólnego.
Przyczyna jest taka sama.
Nie jest jasne, jakie typy dokładnie weźmiemy, a C-Sharp nie może ich domyślnie porównać.
Musimy więc wprowadzić pewne ograniczenia dotyczące naszego rodzaju ogólnego.
Używamy więc słowa kluczowego where.
Więc napiszemy, „gdzie T jest porównywalny”.
To, co mówimy tutaj: jest to ogólna metoda, która przyjmuje dowolny typ T, a ten typ T musi implementować interfejs IComparable.
Jest to ograniczenie, którego będziemy używać.
Teraz powinniśmy mieć dostęp do metody ComparedTo i oto ona.
Mamy go, ponieważ pochodzi z interfejsu IComparable.
Tak więc każdy typ, który implementuje ten interfejs, będzie mógł być używany w tej metodzie.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(1, 1));
Console.WriteLine(AreEqual(1, 2));
Console.WriteLine(AreEqual('a', 'a'));
Console.WriteLine(AreEqual('a', 'b'));
Console.WriteLine(AreEqual("abc", "abc"));
Console.WriteLine(AreEqual("abc", "def"));
}
public static bool AreEqual<T>(T num1, T num2) where T: IComparable<T>
{
return (num1.CompareTo(num2) == 0);
}
}
}
C:\Windows\system32\cmd.exe
True
False
True
False
True
False
Aby kontynuować, naciśnij dowolny klawisz . . .
Num1 jest porównywane z Num2.
Musimy również sprawdzić, czy wynik jest równy zero, ponieważ metoda ComparedTo zwraca liczbę całkowitą.
Zwraca mniej niż zero, jeśli num1 jest mniejsze niż num2.
Zwraca 1, jeśli pierwszy jest wyższy niż drugi i zwraca 0, jeśli są równe.
A teraz widzimy w wynikach, że działa dobrze dla liczb, znaków i napisów.
Możemy porównać je wszystkie za pomocą jednej metody przy użyciu typów ogólnych.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug Any CPU 02. GenericClass Rozpocznij
MyList.cs EntryPoint.cs
02. GenericClass GenericClass.EntryPoint Main()
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(true, "word"));
}
public static bool AreEqual<T>(T num1, T num2) where T: IComparable<T>
{
return (num1.CompareTo(num2) == 0);
}
}
}
A teraz możemy spróbować porównać dwa różne typy.
Na przykład bool i string. Jak widzimy, kompilator nie pozwala im działać.
Dzieje się tak, ponieważ używamy różnych typów, boolean i string.
Mówimy tutaj, że chcemy dowolnego typu, ale ten typ musi być taki sam dla obu argumentów.
Jeśli więc zmienimy je oba na typ bool, metoda będzie działać.
Stosowanie metod generycznych jest przydatne, ponieważ nie musimy pisać kodu nadmiarowego dla 20 różnych typów i możemy je uogólniać i używać dla nich wszystkich.
90 %
Lista błędów
Całe rozwiązanie 1 Błąd 0 Ostrzeżenia 0 Komunikaty Kompilacja + IntelliSense Przeszukaj listę błędów
Kod Opis Projekt Plik
CS0411 Nie można wywnioskować argumentów typu dla metody „EntryPoint.AreEqual<T>(T, T)" na podstawie użytkowania. Spróbuj jawnie określić argumenty typu. 02. GenericClass EntryPoint.cs
Eksplorator serwera
Przybornik

Generic methods for sorting collections

Dobrze, spróbujmy innego przykładu tutaj.

Generics - Microsoft Visual Studio

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

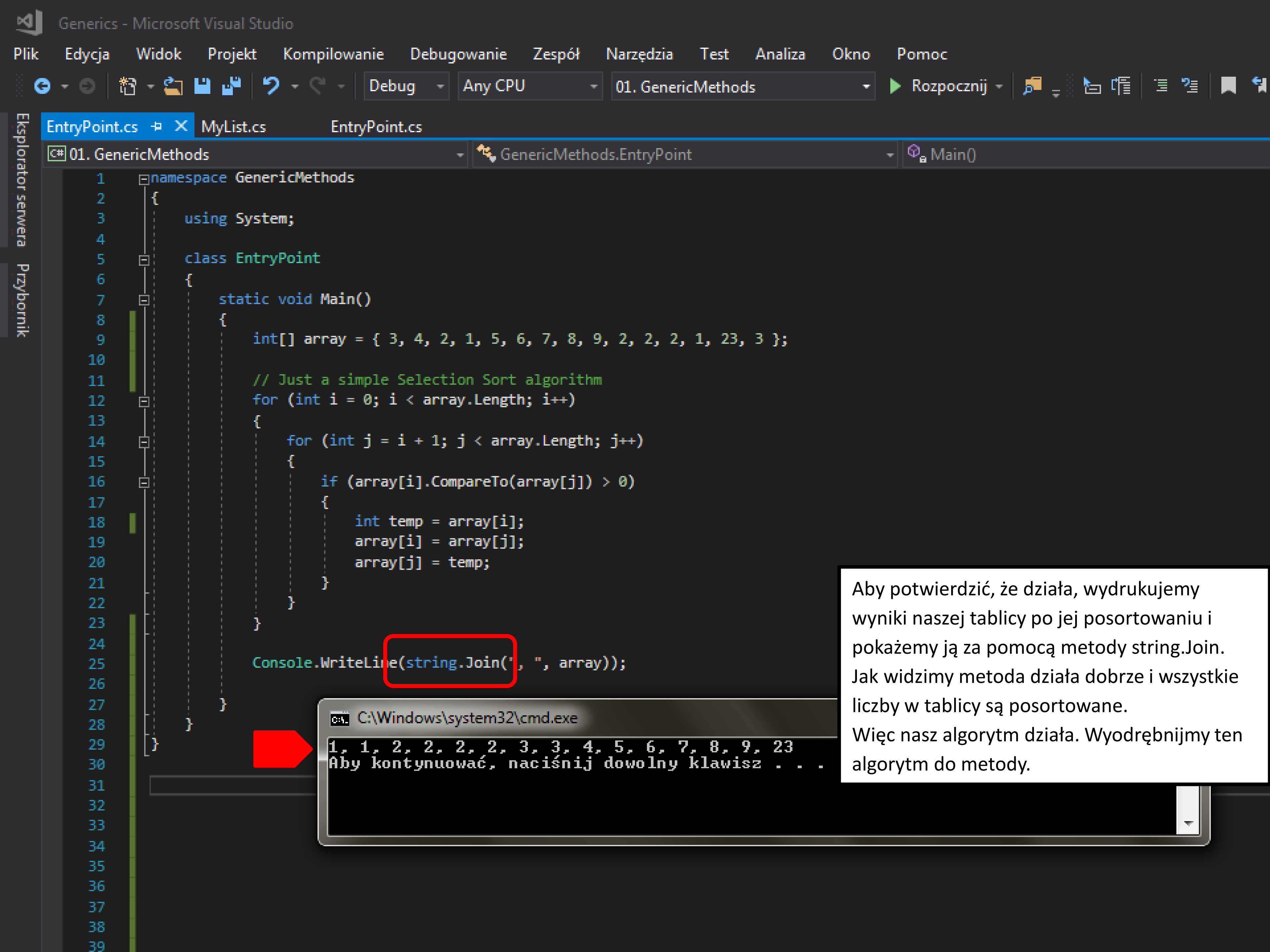
Debug | Any CPU | 01. GenericMethods | Rozpocznij

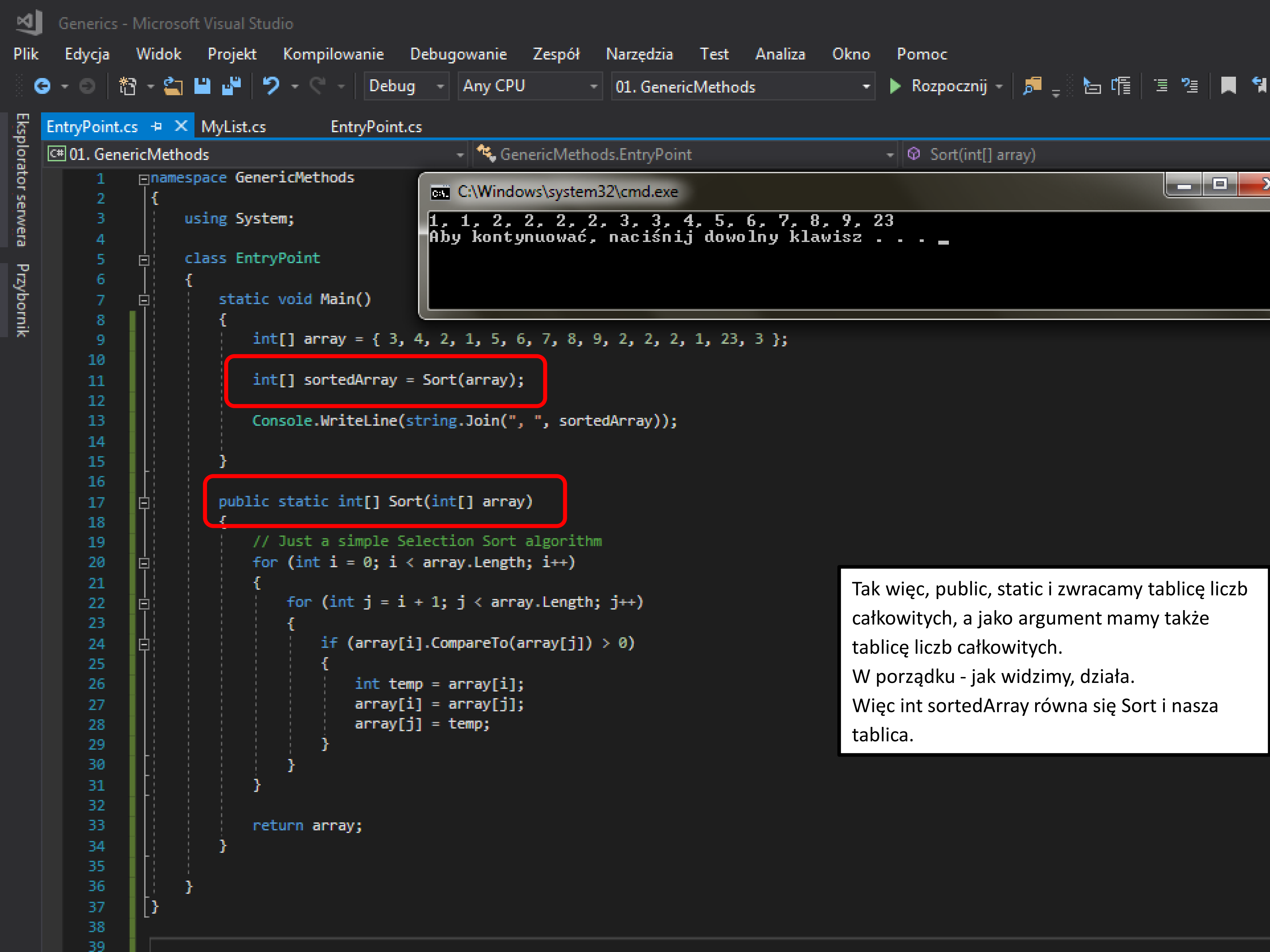
EntryPoint.cs | MyList.cs | EntryPoint.cs

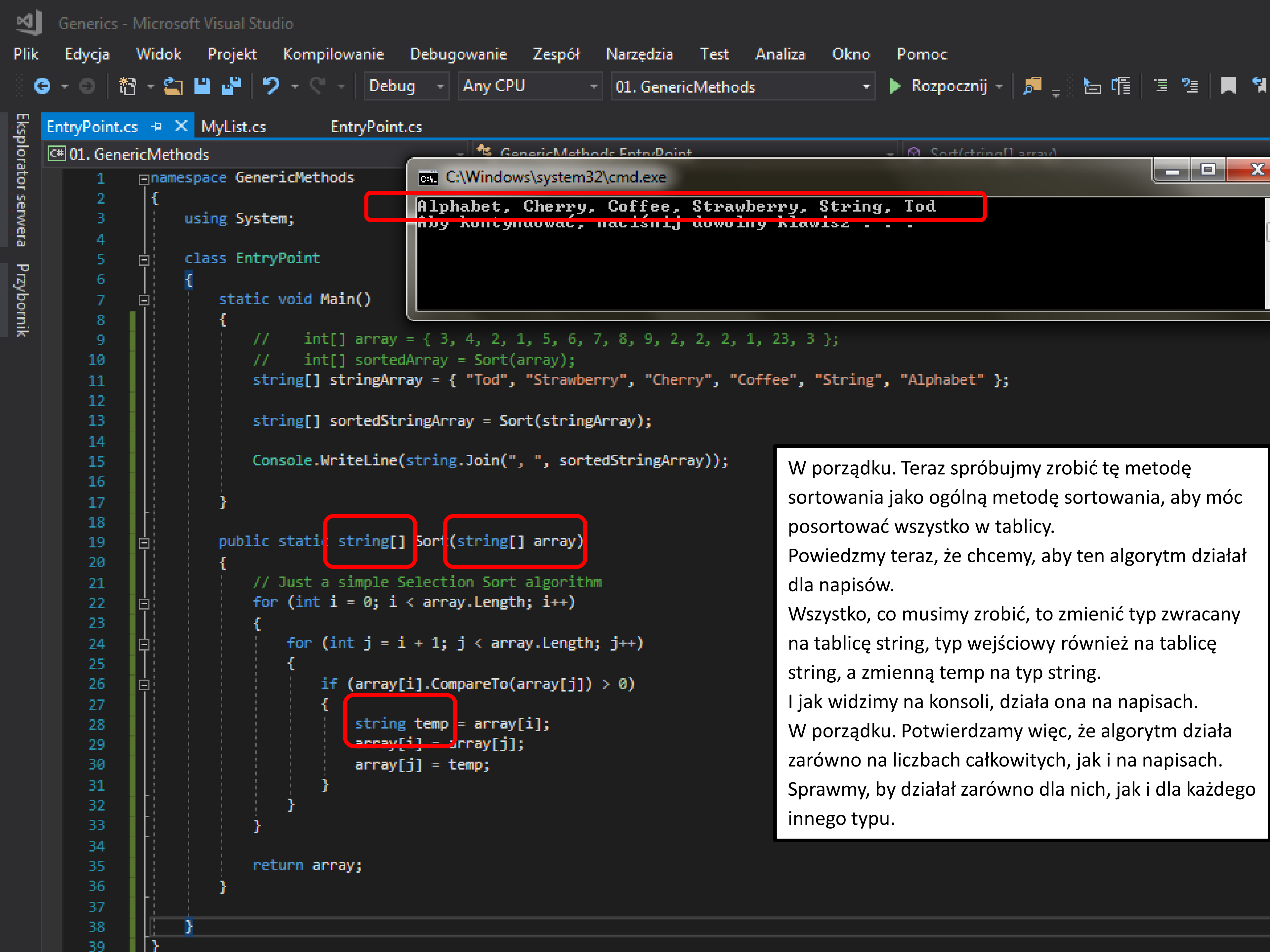
01. GenericMethods | GenericMethods.EntryPoint | Main()

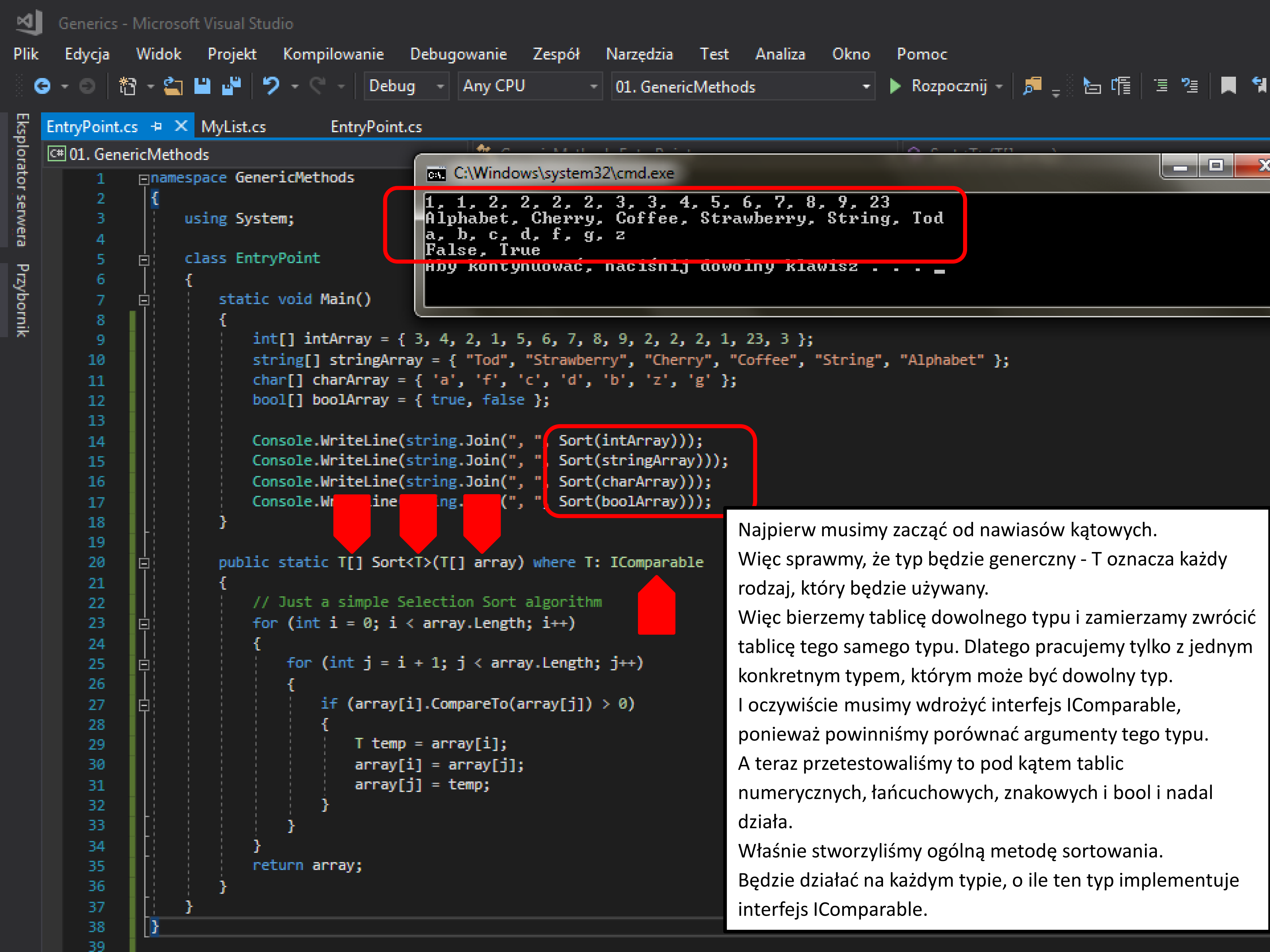
```
namespace GenericMethods
{
    using System;

    class EntryPoint
    {
        static void Main()
        {
            int[] array = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };

            // Just a simple Selection Sort algorithm
            for (int i = 0; i < array.Length; i++)
            {
                for (int j = i + 1; j < array.Length; j++)
                {
                    if (array[i].CompareTo(array[j]) > 0)
                    {
                        int temp = array[i];
                        array[i] = array[j];
                        array[j] = temp;
                    }
                }
            }
        }
    }
}
```

Mamy implementację algorytmu sortowania przez wybieranie.
To po prostu sortowanie tablicy.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. GenericMethods
Rozpocznij
EntryPoint.cs
MyList.cs
EntryPoint.cs
01. GenericMethods
GenericMethods.EntryPoint
Main()
Eksplorator serwera
Przybornik
namespace GenericMethods
{
using System;
class EntryPoint
{
static void Main()
{
int[] array = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };
// Just a simple Selection Sort algorithm
for (int i = 0; i < array.Length; i++)
{
for (int j = i + 1; j < array.Length; j++)
{
if (array[i].CompareTo(array[j]) > 0)
{
int temp = array[i];
array[i] = array[j];
array[j] = temp;
}
}
}
Console.WriteLine(string.Join(", ", array));
}
}
}
C:\Windows\system32\cmd.exe
1, 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 23
Aby kontynuować, naciśnij dowolny klawisz . . .
Aby potwierdzić, że działa, wydrukujemy wyniki naszej tablicy po jej posortowaniu i pokażemy ją za pomocą metody string.Join.
Jak widzimy metoda działa dobrze i wszystkie liczby w tablicy są posortowane.
Więc nasz algorytm działa. Wyodrębnijmy ten algorytm do metody.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. GenericMethods
Rozpocznij
EntryPoint.cs
MyList.cs
EntryPoint.cs
01. GenericMethods
GenericMethods.EntryPoint
Sort(int[] array)
Eksplorator serwera
Przybornik
C:\Windows\system32\cmd.exe
1, 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 23
Aby kontynuować, naciśnij dowolny klawisz . . .
namespace GenericMethods
{
using System;
class EntryPoint
{
static void Main()
{
int[] array = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };
int[] sortedArray = Sort(array);
Console.WriteLine(string.Join(", ", sortedArray));
}
public static int[] Sort(int[] array)
{
// Just a simple Selection Sort algorithm
for (int i = 0; i < array.Length; i++)
{
for (int j = i + 1; j < array.Length; j++)
{
if (array[i].CompareTo(array[j]) > 0)
{
int temp = array[i];
array[i] = array[j];
array[j] = temp;
}
}
}
return array;
}
}
}
Tak więc, public, static i zwracamy tablicę liczb całkowitych, a jako argument mamy także tablicę liczb całkowitych.
W porządku - jak widzimy, działa.
Więc int sortedArray równa się Sort i nasza tablica.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug Any CPU 01. GenericMethods Rozpocznij
EntryPoint.cs MyList.cs EntryPoint.cs
01. GenericMethods
Eksplorator serwera Przybornik
C:\Windows\system32\cmd.exe
Alphabet, Cherry, Coffee, Strawberry, String, Tod
namespace GenericMethods
{
using System;
class EntryPoint
{
static void Main()
{
// int[] array = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };
// int[] sortedArray = Sort(array);
string[] stringArray = { "Tod", "Strawberry", "Cherry", "Coffee", "String", "Alphabet" };
string[] sortedStringArray = Sort(stringArray);
Console.WriteLine(string.Join(", ", sortedStringArray));
}
public static string[] Sort(string[] array)
{
// Just a simple Selection Sort algorithm
for (int i = 0; i < array.Length; i++)
{
for (int j = i + 1; j < array.Length; j++)
{
if (array[i].CompareTo(array[j]) > 0)
{
string temp = array[i];
array[j] = temp;
}
}
}
return array;
}
}
}
W porządku. Teraz spróbujmy zrobić tę metodę sortowania jako ogólną metodę sortowania, aby móc posortować wszystko w tablicy.
Powiedzmy teraz, że chcemy, aby ten algorytm działał dla napisów.
Wszystko, co musimy zrobić, to zmienić typ zwracany na tablicę string, typ wejściowy również na tablicę string, a zmienną temp na typ string.
I jak widzimy na konsoli, działa ona na napisach.
W porządku. Potwierdzamy więc, że algorytm działa zarówno na liczbach całkowitych, jak i na napisach.
Sprawmy, by działał zarówno dla nich, jak i dla każdego innego typu.

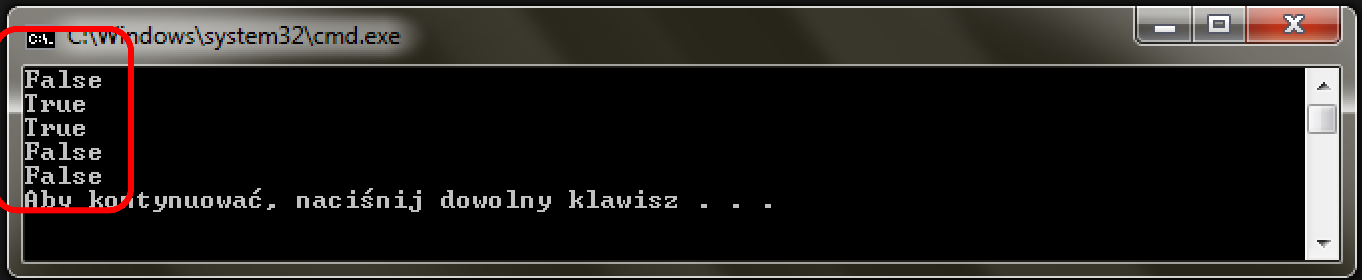


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. GenericMethods
Rozpocznij
EntryPoint.cs
MyList.cs
EntryPoint.cs
01. GenericMethods
Eksplorator serwera
Przybornik
C:\Windows\system32\cmd.exe
1, 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 23
Alphabet, Cherry, Coffee, Strawberry, String, Tod
a, b, c, d, f, g, z
False, True
Aby kontynuować, naciśnij dowolny klawisz . . .
namespace GenericMethods
{
using System;
class EntryPoint
{
static void Main()
{
int[] intArray = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };
string[] stringArray = { "Tod", "Strawberry", "Cherry", "Coffee", "String", "Alphabet" };
char[] charArray = { 'a', 'f', 'c', 'd', 'b', 'z', 'g' };
bool[] boolArray = { true, false };
Console.WriteLine(string.Join(", ", Sort(intArray)));
Console.WriteLine(string.Join(", ", Sort(stringArray)));
Console.WriteLine(string.Join(", ", Sort(charArray)));
Console.WriteLine(string.Join(", ", Sort(boolArray)));
}
public static T[] Sort<T>(T[] array) where T: IComparable
{
// Just a simple Selection Sort algorithm
for (int i = 0; i < array.Length; i++)
{
for (int j = i + 1; j < array.Length; j++)
{
if (array[i].CompareTo(array[j]) > 0)
{
T temp = array[i];
array[i] = array[j];
array[j] = temp;
}
}
}
return array;
}
}
}
Najpierw musimy zacząć od nawiasów kątowych.
Więc sprawmy, że typ będzie generczny - T oznacza każdy rodzaj, który będzie używany.
Więc bierzemy tablicę dowolnego typu i zamierzamy zwrócić tablicę tego samego typu. Dlatego pracujemy tylko z jednym konkretnym typem, którym może być dowolny typ.
I oczywiście musimy wdrożyć interfejs IComparable, ponieważ powinniśmy porównać argumenty tego typu.
A teraz przetestowaliśmy to pod kątem tablic numerycznych, łańcuchowych, znakowych i bool i nadal działa.
Właśnie stworzyliśmy ogólną metodę sortowania.
Będzie działać na każdym typie, o ile ten typ implementuje interfejs IComparable.

Implementing the IComparable interface in a class

W porządku, stworzyliśmy dwie ogólne metody: AreEqual i Sort, które działają na każdym typie.
Ale do tej pory próbowaliśmy tylko z liczbami całkowitymi i ciągami znaków oraz kilkoma prostymi typami.
A co gdybyśmy mieli własny niestandardowy typ lub klasę.

```
namespace GenericMethods
{
    using System;

    class EntryPoint
    {
        static void Main()
        {
            Console.WriteLine(AreEqual(3, 5));
            Console.WriteLine(AreEqual(5, 5));
            Console.WriteLine(AreEqual("abc", "abc"));
            Console.WriteLine(AreEqual(true, false));
            Console.WriteLine(AreEqual('a', 'b'));
        }

        public static bool AreEqual<T>(T input1, T input2) where T : IComparable<T>
        {
            return input1.CompareTo(input2) == 0;
        }

    }
}
```



Użyjmy ogólnej metody AreEqual. Jak widać, działa dla liczb, napisów, bool i znaków.
A teraz stwórzmy nową klasę - klasa osoba.

Generics - Microsoft Visual Studio

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

Debug | Any CPU | 01. GenericMethods | Rozpocznij

Person.cs* | EntryPoint.cs | MyList.cs | EntryPoint.cs

01. GenericMethods | GenericMethods.Person | Age

```
6      {
7          static void Main()
8          {
9              Console.WriteLine(AreEqual(3, 5));
10             Console.WriteLine(AreEqual(5, 5));
11             Console.WriteLine(AreEqual("abc", "abc"));
12             Console.WriteLine(AreEqual(true, false));
13             Console.WriteLine(AreEqual('a', 'b'));
14
15             Person p1 = new Person() { Age = 15 };
16             Person p2 = new Person() { Age = 15 };
17
18             Console.WriteLine(AreEqual(p1, p2));
19         }
20
21         public static bool AreEqual<T>(T input1, T input2) where T : IComparable<T>
22         {
23             return input1.CompareTo(input2) == 0;
24         }
25
26     }
27
28     public class Person
29     {
30         public int Age { get; set; }
31     }
```

Ta klasa będzie miała tylko jedną właściwość, jaką jest Wiek.
Oczywiście w rzeczywistości może mieć wiele innych właściwości, ale w tej chwili ma tylko jedną.
Więc teraz postaramy się utworzyć dwie osoby.
Osoba pierwsza i osoba druga.
A jeśli spróbujemy je porównać, to nie zadziała.
To nie zadziała, ponieważ nie ma niejawnej metody ich porównania.
Zasadniczo powiemy, że te argumenty po prostu wpisują typ person, który obecnie nie implementuje interfejsu IComparable.
Dlatego nie działa.
Aby to naprawić, musimy zaimplementować metodę IComparable w naszej klasie.

90 %

Lista błędów

Całe rozwiązanie | 1 Błąd | 0 Ostrzeżenia | 0 Komunikaty | Kompilacja + IntelliSense | Przeszukaj listę błędów

Kod	Opis	Projekt	Plik	W...	Stan pominięcia
CS0311	Nie można użyć typu „GenericMethods.Person" jako parametru typu „T" w typie ogólnym lub metodzie „EntryPoint.AreEqual<T>(T, T)". Brak niejawnej konwersji odwołania z typu „GenericMethods.Person" na „System.IComparable<GenericMethods.Person>".	01. GenericMethods	EntryPoint.cs	18	Aktywne

Generics - Microsoft Visual Studio

Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs

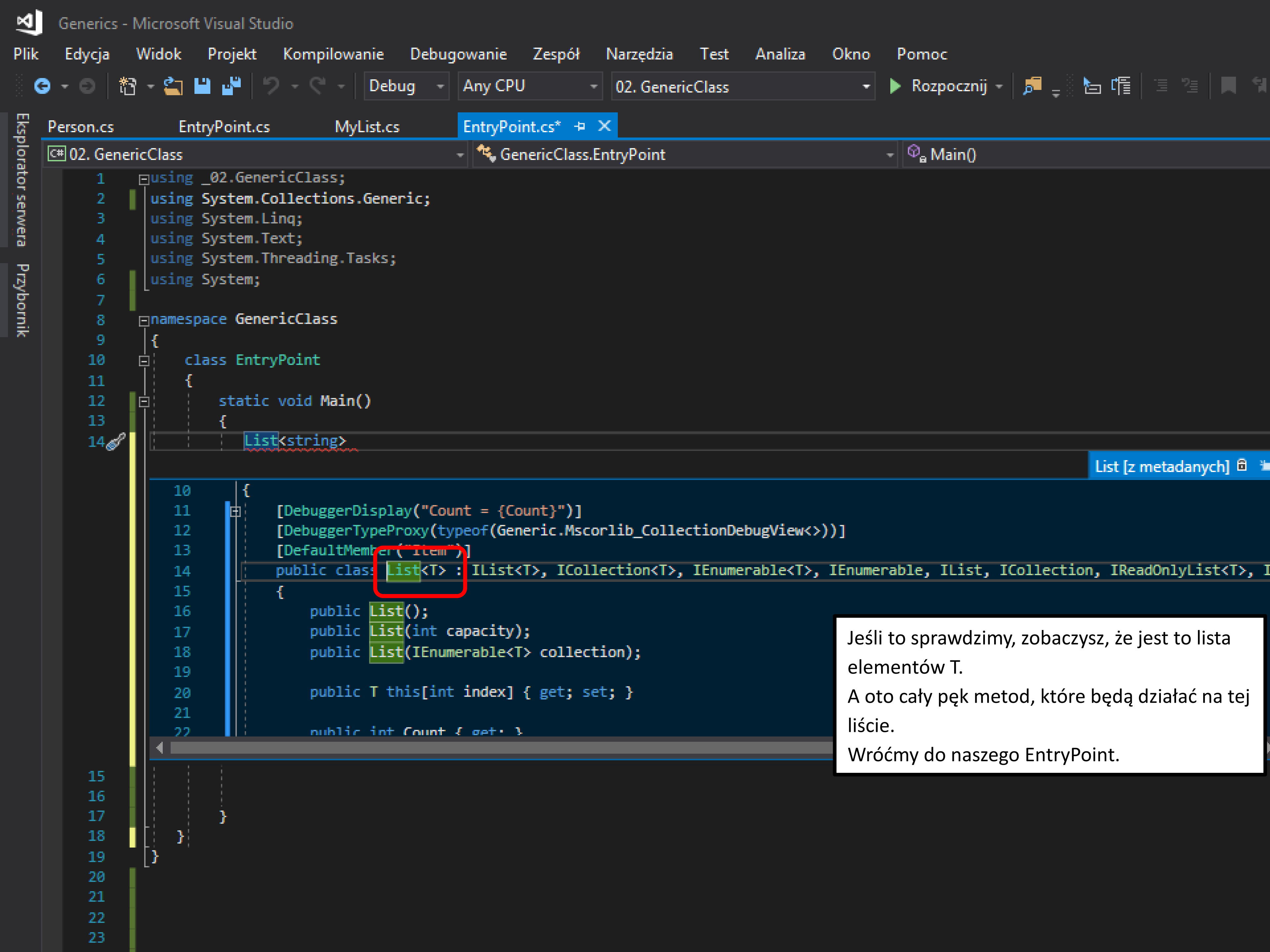
01. GenericMethods | GenericMethods.EntryPoint | Main()

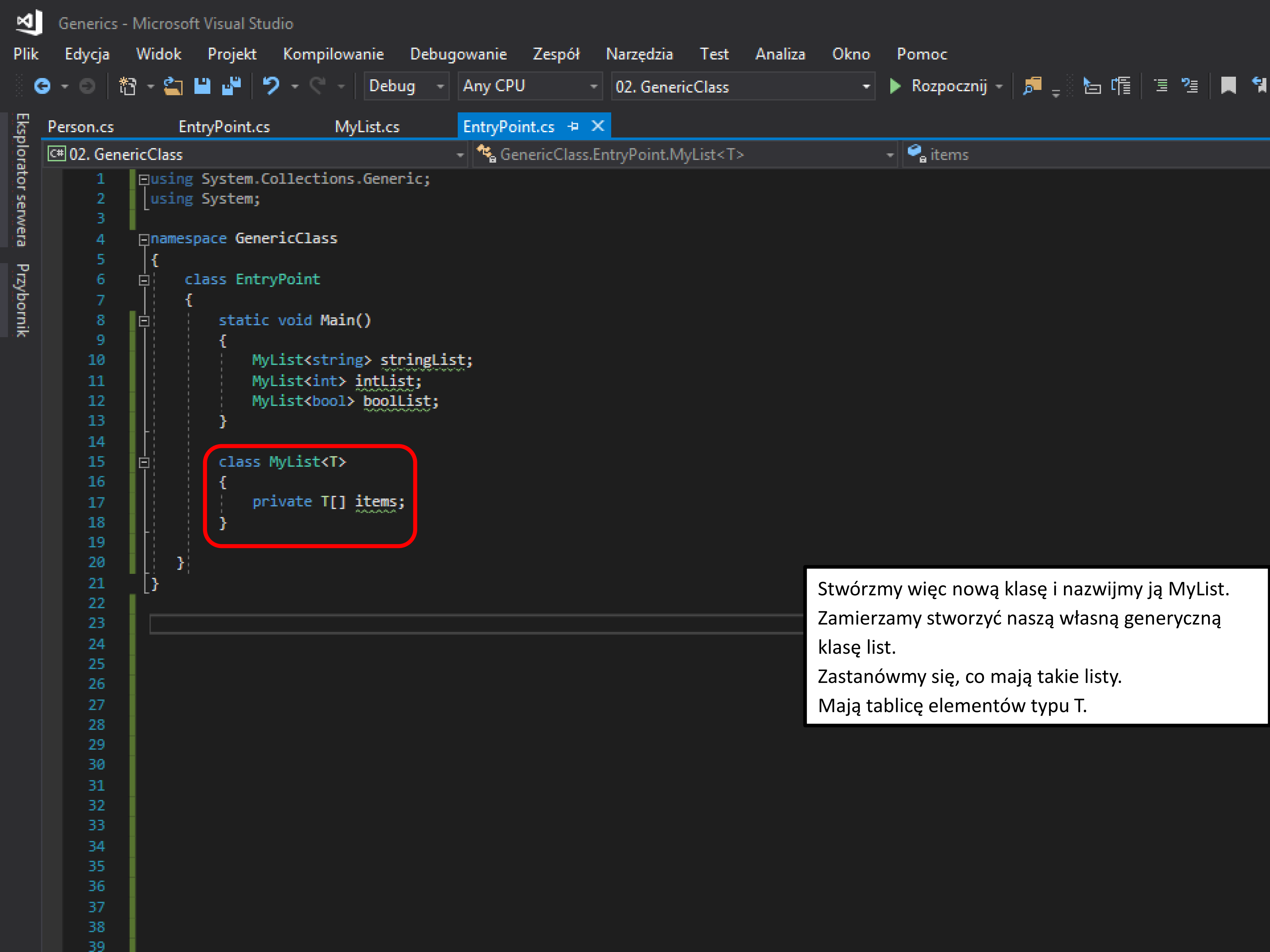
```csharp
class EntryPoint
{
    static void Main()
    {
        Person p1 = new Person() { Age = 15 };
        Person p2 = new Person() { Age = 15 };

        Console.WriteLine(AreEqual(p1, p2));
        Console.WriteLine(AreEqual(new Person() { Age = 17 }, new Person() { Age = 21 }));
    }

    public static bool AreEqual<T>(T input1, T input2) where T : IComparable<T>
    {
        return input1.CompareTo(input2) == 0;
    }

}

public class Person : IComparable<Person>
{
    public int Age { get; set; }
    public int CompareTo(Person other)
    {
        if (this.Age < other.Age)
        {
            return -1;
        }
        else if (this.Age == other.Age)
        {
            return 0;
        }
        else
        {
            return 1;
        }
    }
}
```

Robimy to za pomocą interfejsu Icomparable na nazwie naszej klasy.
Jedyne, czego wymaga od nas, to wdrożenie sposobu porównywania dwóch osób.
Jak więc porównamy dwie różne osoby.
Cóż, jedyną właściwością jest Wiek.
Tak właśnie zrobimy, aby porównać dwie osoby.
Porównamy je według wieku.
Jeśli pierwszy wiek jest mniejszy niż drugi, wtedy pierwsza osoba jest mniejsza niż druga osoba. Gdy wiek jest równy - ludzie są równi.
I jak widać, nie mamy już błędu, ponieważ zaimplementowaliśmy wymóg metody generycznej.
W pierwszym przykładzie osoby są równe, ponieważ ich wiek jest równy. Wynikiem jest true.
W drugim przykładzie wiek nie jest równy, więc ludzie nie są równi. Wynikiem jest więc false.

Generic classes

Mamy więc nie tylko metody generyczne, ale także klasy generyczne i istnieje jedna szczególna klasa ogólna, która była używana wielokrotnie.
Tą klasą jest klasa List.
Pamiętaj, że po napisaniu listy zawsze masz nawiasy kątowe, aby określić, jaki typ będzie miała lista. Możemy napisać listę z liczbą całkowitą lub ciągiem znaków lub dowolną inną zawartością listy. To idealny przykład klasy generycznej.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs*
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
using _02.GenericClass;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
List<string>
List [z metadanych]
{
[DebuggerDisplay("Count = {Count}")]
[DebuggerTypeProxy(typeof(Generic.Mscorlib_CollectionDebugView<>))]
public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>, IEnumerable, IList, ICollection, IReadOnlyList<T>,
{
public List();
public List(int capacity);
public List(IEnumerable<T> collection);
public T this[int index] { get; set; }
}
}
}
Jeśli to sprawdzimy, zobaczysz, że jest to lista elementów T.
A oto cały pęk metod, które będą działać na tej liście.
Wróćmy do naszego EntryPoint.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint.MyList<T>
items
Eksplorator serwera
Przybornik
using System.Collections.Generic;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
MyList<string> stringList;
MyList<int> intList;
MyList<bool> boolList;
}
class MyList<T>
{
private T[] items;
}
}
}
Stwórzmy więc nową klasę i nazwijmy ją MyList.
Zamierzamy stworzyć naszą własną generyczną klasę list.
Zastanówmy się, co mają takie listy.
Mają tablicę elementów typu T.

Generics - Microsoft Visual Studio

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

Debug | Any CPU | 02. GenericClass | Rozpocznij

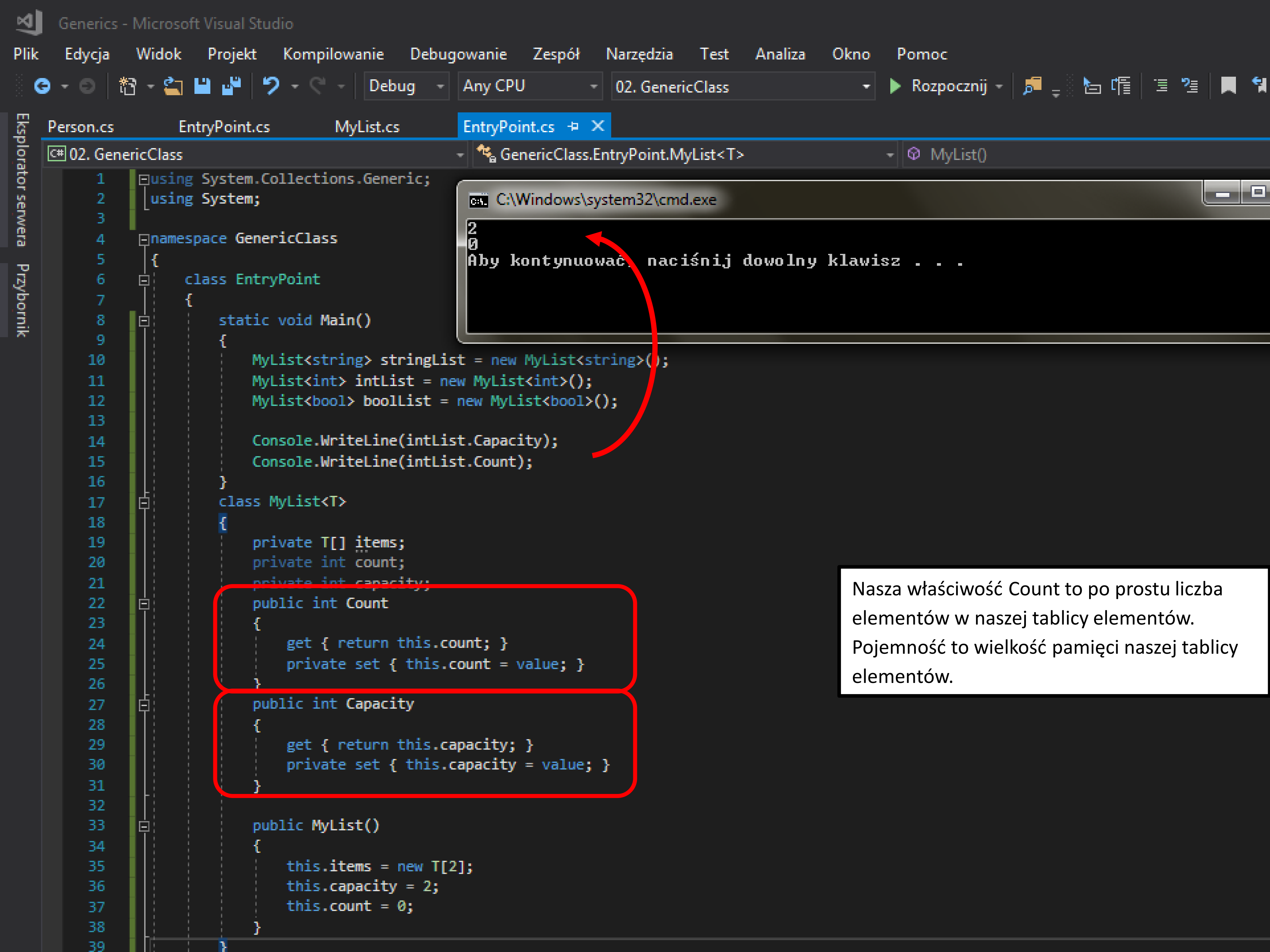
Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs*

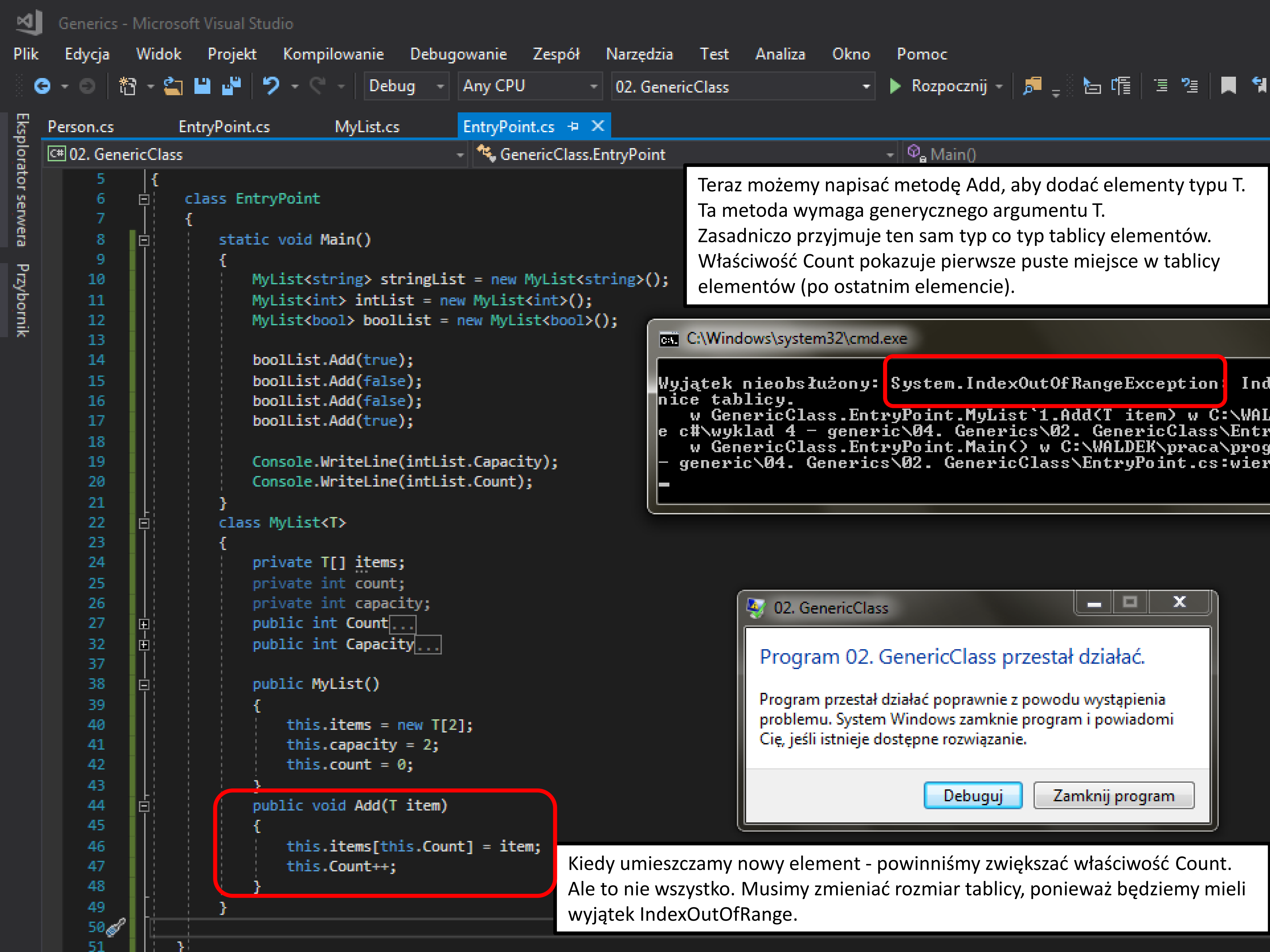
02. GenericClass | GenericClass.EntryPoint.MyList<T> | MyList()

Eksplorator serwera | Przybornik

```
using System.Collections.Generic;
using System;

namespace GenericClass
{
    class EntryPoint
    {
        static void Main()
        {
            MyList<string> stringList;
            MyList<int> intList;
            MyList<bool> boolList;
        }

        class MyList<T>
        {
            private T[] items;
            private int count;
            private int capacity;

            public int Count { get; private set; }
            public int Capacity { get; private set; }

            public MyList()
            {
                this.items = new T[2];
                this.capacity = 2;
                this.count = 0;
            }

        }
    }
}
```

Mają też właściwość count, która zawiera liczbę elementów na liście.
Musimy ustawić ją jako prywatną, ponieważ nikt nie musi uzyskiwać dostępu do tej tablicy.
Mamy również capacity dla naszej listy.
Potrzebujemy również konstruktor.
Więc co będziemy mieć w tym konstruktorze?
Musimy zainicjować naszą tablicę oraz właściwości Count i Capacity wartościami początkowymi.


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint.MyList<T>
MyList()
Eksplorator serwera
Przybornik
C:\Windows\system32\cmd.exe
2
0
Aby kontynuować, naciśnij dowolny klawisz . . .
using System.Collections.Generic;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
MyList<string> stringList = new MyList<string>();
MyList<int> intList = new MyList<int>();
MyList<bool> boolList = new MyList<bool>();
Console.WriteLine(intList.Capacity);
Console.WriteLine(intList.Count);
}
class MyList<T>
{
private T[] items;
private int count;
private int capacity;
public int Count
{
get { return this.count; }
private set { this.count = value; }
}
public int Capacity
{
get { return this.capacity; }
private set { this.capacity = value; }
}
public MyList()
{
this.items = new T[2];
this.capacity = 2;
this.count = 0;
}
}
Nasza właściwość Count to po prostu liczba elementów w naszej tablicy elementów. Pojemność to wielkość pamięci naszej tablicy elementów.



Teraz możemy napisać metodę Add, aby dodać elementy typu T.
Ta metoda wymaga generycznego argumentu T.
Zasadniczo przyjmuje ten sam typ co typ tablicy elementów.
Właściwość Count pokazuje pierwsze puste miejsce w tablicy elementów (po ostatnim elemencie).
class EntryPoint
static void Main()
MyList<string> stringList = new MyList<string>();
MyList<int> intList = new MyList<int>();
MyList<bool> boolList = new MyList<bool>();
boolList.Add(true);
boolList.Add(false);
boolList.Add(false);
boolList.Add(true);
Console.WriteLine(intList.Capacity);
Console.WriteLine(intList.Count);
class MyList<T>
private T[] items;
private int count;
private int capacity;
public int Count
public int Capacity
public MyList()
this.items = new T[2];
this.capacity = 2;
this.count = 0;
public void Add(T item)
this.items[this.Count] = item;
this.Count++;
C:\Windows\system32\cmd.exe
Wyjątek nieobsłużony: System.IndexOutOfRangeException
nice tablicy.
w GenericClass.EntryPoint.MyList`1.Add(T item)
w GenericClass.EntryPoint.Main()
02. GenericClass
Program 02. GenericClass przestał działać.
Program przestał działać poprawnie z powodu wystąpienia problemu. System Windows zamknie program i powiadomi Cię, jeśli istnieje dostępne rozwiązanie.
Debuguj
Zamknij program
Kiedy umieszczamy nowy element - powinniśmy zwiększać właściwość Count.
Ale to nie wszystko. Musimy zmieniać rozmiar tablicy, ponieważ będziemy mieli wyjątek IndexOutOfRange.

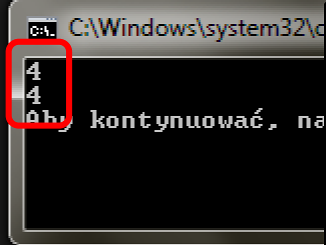
Generics - Microsoft Visual Studio

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

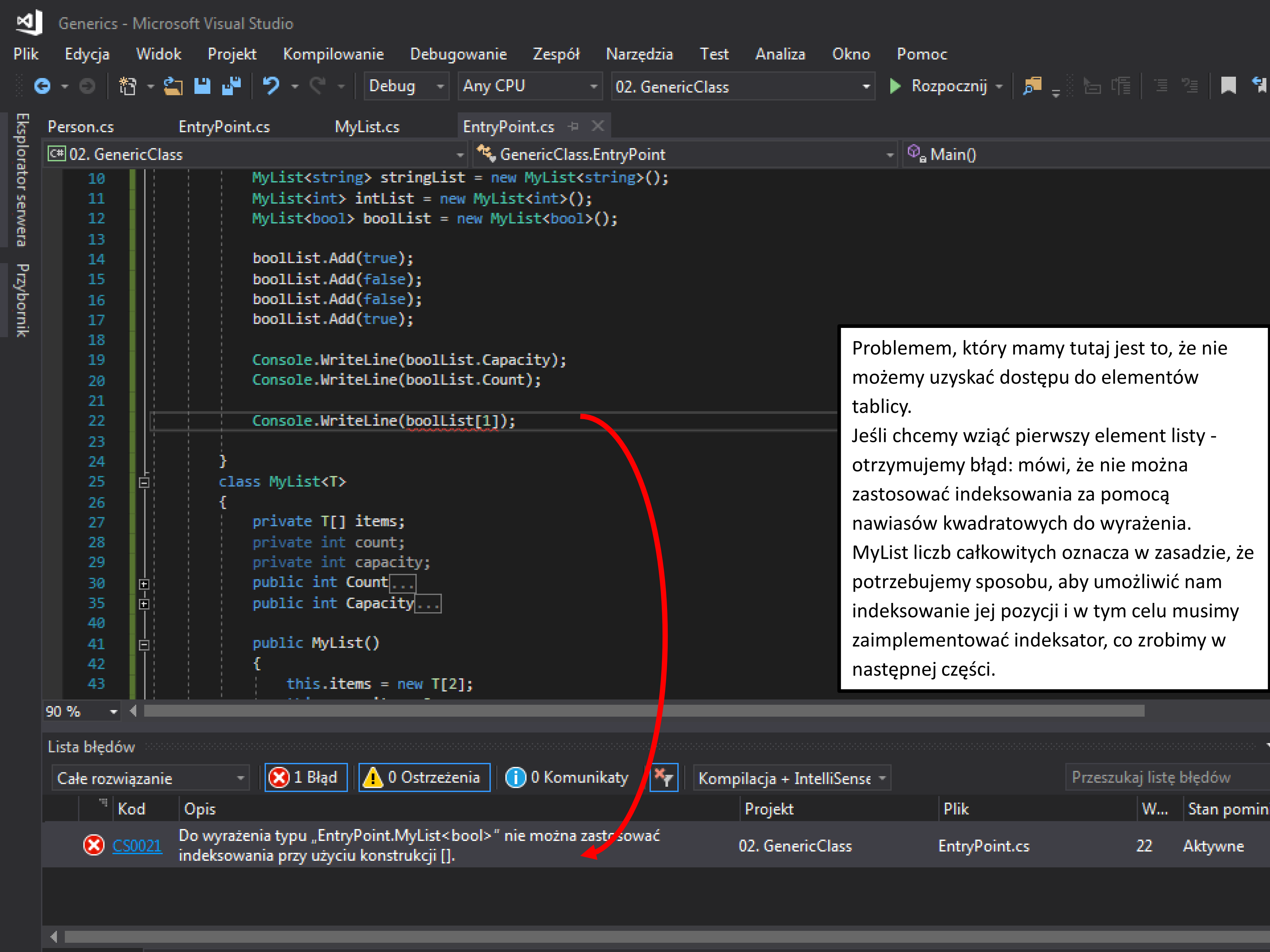
Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs

02. GenericClass | GenericClass.EntryPoint | Main()

```
10            MyList<string> stringList = new MyList<string>();
11            MyList<int> intList = new MyList<int>();
12            MyList<bool> boolList = new MyList<bool>();
13
14            boolList.Add(true);
15            boolList.Add(false);
16            boolList.Add(false);
17            boolList.Add(true);
18
19            Console.WriteLine(boolList.Capacity);
20            Console.WriteLine(boolList.Count);
21        }
22        class MyList<T>
23        {
24            private T[] items;
25            private int count;
26            private int capacity;
27            public int Count...
32            public int Capacity...
37
38            public MyList()
39            {
40                this.items = new T[2];
41                this.capacity = 2;
42                this.count = 0;
43            }
44            public void Add(T item)
45            {
46                if (this.capacity == this.count)
47                {
48                    T[] clone = (T[])items.Clone();
49                    this.capacity *= 2;
50                    this.items = new T[this.capacity];
51                    Array.Copy(clone, 0, this.items, 0, clone.Length);
52                }
53                this.items[this.Count] = item;
54                this.Count++;
55            }
56        }
```

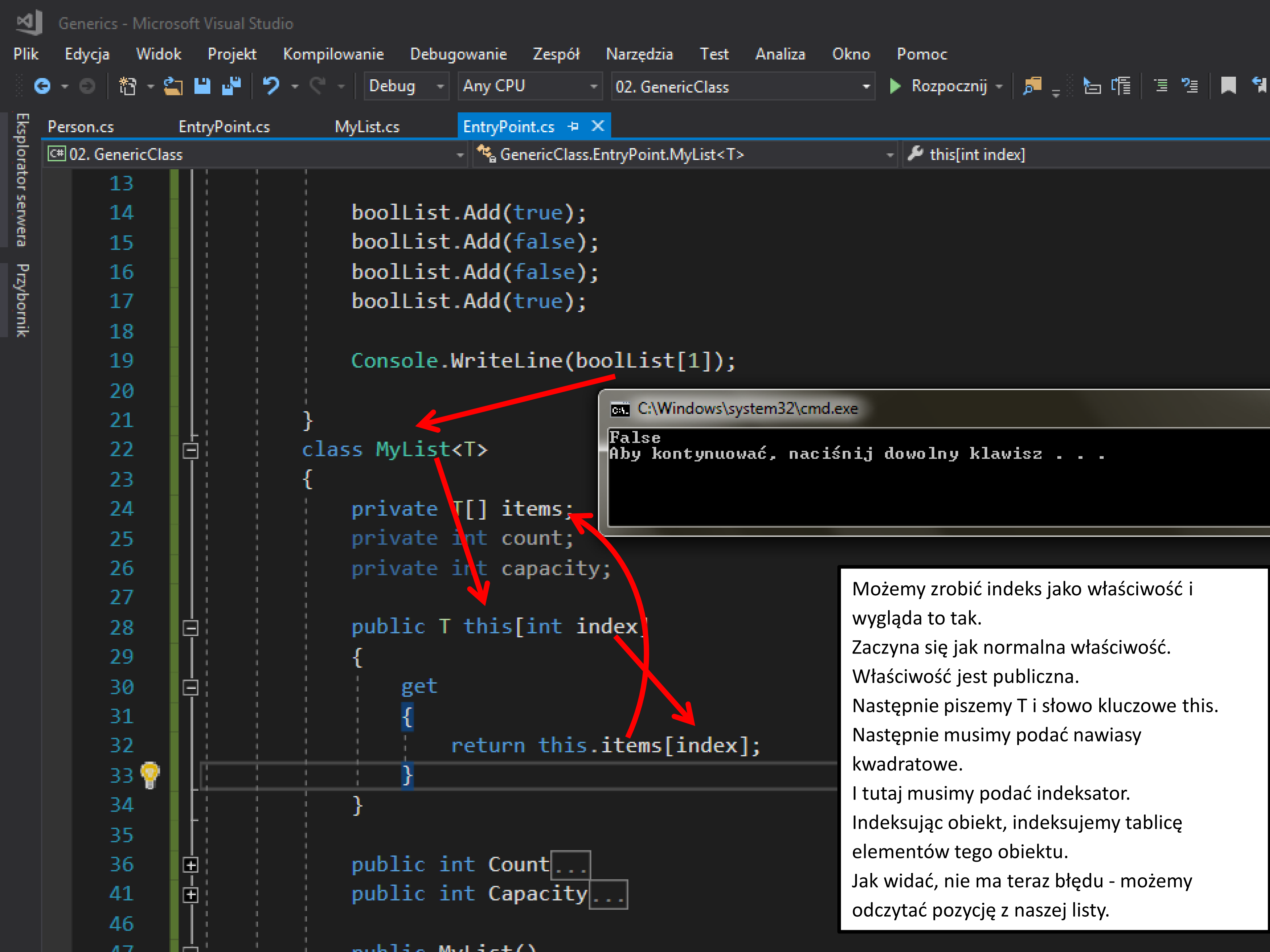


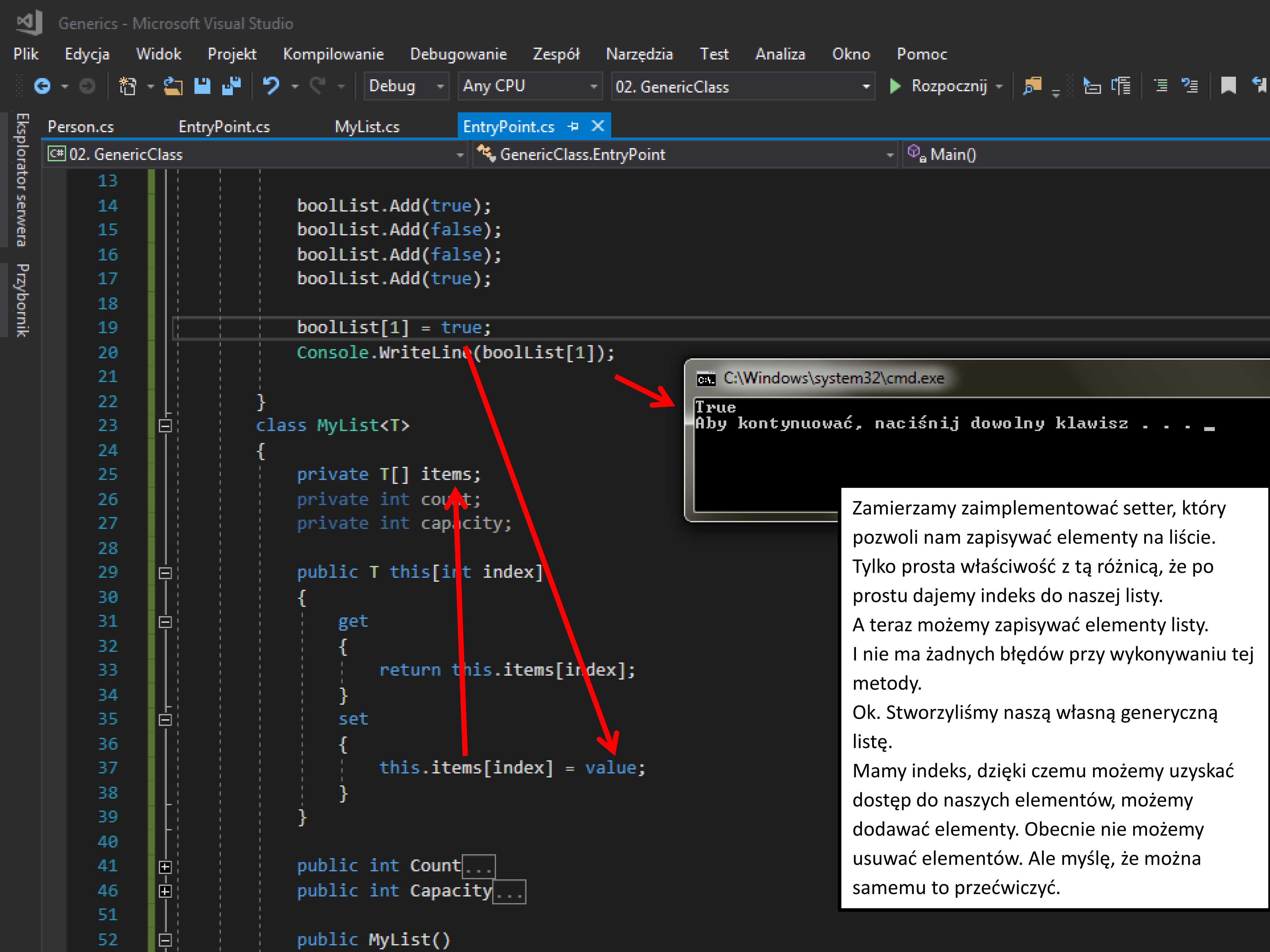
Oczywiście zanim dodamy coś do tablicy, musimy sprawdzić, czy w tablicy jest miejsce, ponieważ inicjujemy tablicę tylko z 2 miejscami dla elementów.
Możemy to zrobić w następujący sposób:
Więc jeśli Count są równe Capacity, pomnożymy to przez 2.
Stworzymy nową tablicę, która sklonuje naszą obecną tablicę.
Następnie podwoimy pojemność.
Tak więc ta maksymalna pojemność jest pomnożona przez dwa, a my stworzymy nową większą tablicę i skopiujemy poprzednią tablicę do nowej większej.
Teraz dodajemy niektóre elementy.
I sprawdzamy, jaka jest pojemność i Liczba elementów.
Jak widać, działa.
Mamy Capacity cztery i Count cztery. I to jest poprawne.



Generics - Microsoft Visual Studio
Plik
Edycja
Widok
Projekt
Kompilowanie
Debugowanie
Zespół
Narzędzia
Test
Analiza
Okno
Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
MyList<string> stringList = new MyList<string>();
MyList<int> intList = new MyList<int>();
MyList<bool> boolList = new MyList<bool>();
boolList.Add(true);
boolList.Add(false);
boolList.Add(false);
boolList.Add(true);
Console.WriteLine(boolList.Capacity);
Console.WriteLine(boolList.Count);
Console.WriteLine(boolList[1]);
}
class MyList<T>
{
private T[] items;
private int count;
private int capacity;
public int Count
public int Capacity
public MyList()
{
this.items = new T[2];
Problemem, który mamy tutaj jest to, że nie możemy uzyskać dostępu do elementów tablicy.
Jeśli chcemy wziąć pierwszy element listy - otrzymujemy błąd: mówi, że nie można zastosować indeksowania za pomocą nawiasów kwadratowych do wyrażenia.
MyList liczb całkowitych oznacza w zasadzie, że potrzebujemy sposobu, aby umożliwić nam indeksowanie jej pozycji i w tym celu musimy zaimplementować indeksator, co zrobimy w następnej części.
90 %
Lista błędów
Całe rozwiązanie
1 Błąd
0 Ostrzeżenia
0 Komunikaty
Kompilacja + IntelliSense
Przeszukaj listę błędów
Kod
Opis
Projekt
Plik
W...
Stan pomin
CS0021
Do wyrażenia typu „EntryPoint.MyList<bool>” nie można zastosować indeksowania przy użyciu konstrukcji [].
02. GenericClass
EntryPoint.cs
22
Aktywne

Generic classes - indexers


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint.MyList<T>
this[int index]
Eksplorator serwera
Przybornik
boolList.Add(true);
boolList.Add(false);
boolList.Add(false);
boolList.Add(true);
Console.WriteLine(boolList[1]);
}
class MyList<T>
{
private T[] items;
private int count;
private int capacity;
public T this[int index]
{
get
{
return this.items[index];
}
}
public int Count...
public int Capacity...
C:\Windows\system32\cmd.exe
False
Aby kontynuować, naciśnij dowolny klawisz . . .
Możemy zrobić indeks jako właściwość i wygląda to tak.
Zaczyna się jak normalna właściwość.
Właściwość jest publiczna.
Następnie piszemy T i słowo kluczowe this.
Następnie musimy podać nawiasy kwadratowe.
I tutaj musimy podać indeksator.
Indeksując obiekt, indeksujemy tablicę elementów tego obiektu.
Jak widać, nie ma teraz błędu - możemy odczytać pozycję z naszej listy.



Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
boolList.Add(true);
boolList.Add(false);
boolList.Add(false);
boolList.Add(true);
boolList[1] = true;
Console.WriteLine(boolList[1]);
}
class MyList<T>
{
private T[] items;
private int count;
private int capacity;
public T this[int index]
{
get
{
return this.items[index];
}
set
{
this.items[index] = value;
}
}
public int Count...
public int Capacity...
public MyList()
C:\Windows\system32\cmd.exe
True
Aby kontynuować, naciśnij dowolny klawisz . . .
Zamierzamy zaimplementować setter, który pozwoli nam zapisywać elementy na liście.
Tylko prosta właściwość z tą różnicą, że po prostu dajemy indeks do naszej listy.
A teraz możemy zapisywać elementy listy.
I nie ma żadnych błędów przy wykonywaniu tej metody.
Ok. Stworzyliśmy naszą własną generyczną listę.
Mamy indeks, dzięki czemu możemy uzyskać dostęp do naszych elementów, możemy dodawać elementy. Obecnie nie możemy usuwać elementów. Ale myślę, że można samemu to przećwiczyć.

Overloading mathematical operators

Teraz będziemy przeciążać operator matematyczny.

Generics - Microsoft Visual Studio

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

Debug | Any CPU | 02. GenericClass | Rozpocznij

Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs

02. GenericClass | GenericClass.EntryPoint | Main()

```
4   namespace GenericClass
5   {
6       class EntryPoint
7       {
8           static void Main()
9           {
10              MyList<int> firstIntList = new MyList<int>();
11              firstIntList.Add(5);
12              firstIntList.Add(6);
13              firstIntList.Add(3);
14
15              MyList<int> secondIntList = new MyList<int>();
16              secondIntList.Add(5);
17              secondIntList.Add(2);
18              secondIntList.Add(4);
19
20              MyList<int> sumIntList = firstIntList + secondIntList;
21          }
22
23          class MyList<T>
24          {
25              private T[] items;
```

Chcemy dodać dwie listy tego samego typu. Mamy dwie listy typów całkowitych i używamy operatora +, aby je dodać.
Ale teraz kompilator nie wie, co należy zrobić dla tego operatora. Możesz to zrobić z twoją każdą niestandardową klasą.

109 %

Lista błędów

Całe rozwiązanie | 1 Błąd | 0 Ostrzeżenia | 0 Komunikaty | Kompilacja + IntelliSense | Przeszukaj listę błędów

	Kod	Opis	Projekt	Plik	W...	Stan pomin
	CS0019	Nie można zastosować operatora „+" do argumentów operacji typu „EntryPoint.MyList<int>" lub „EntryPoint.MyList<int>".	02. GenericClass	EntryPoint.cs	20	Aktywne

Generics - Microsoft Visual Studio

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

Debug | Any CPU | 02. GenericClass | Rozpocznij

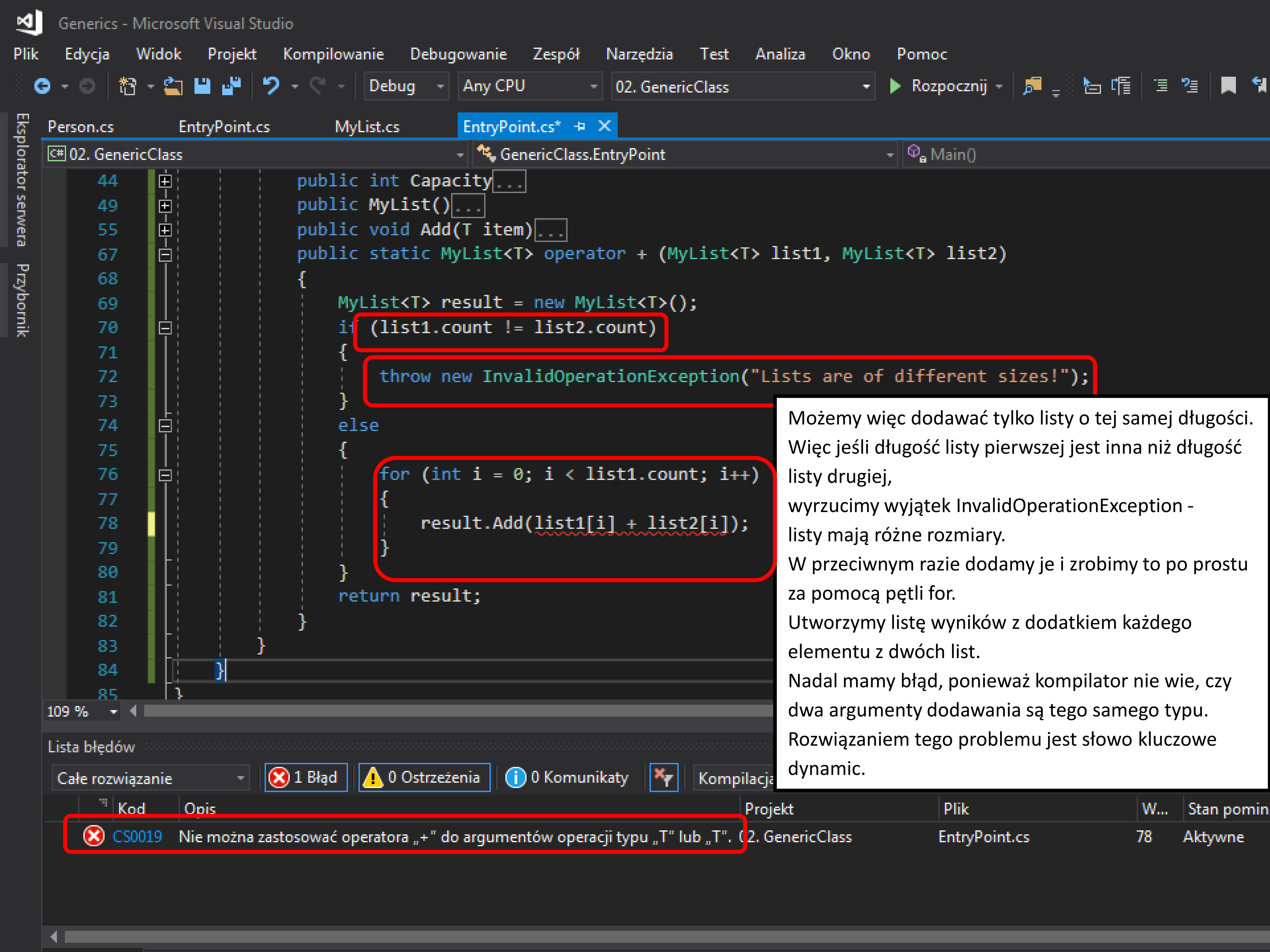
Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs

02. GenericClass | GenericClass.EntryPoint

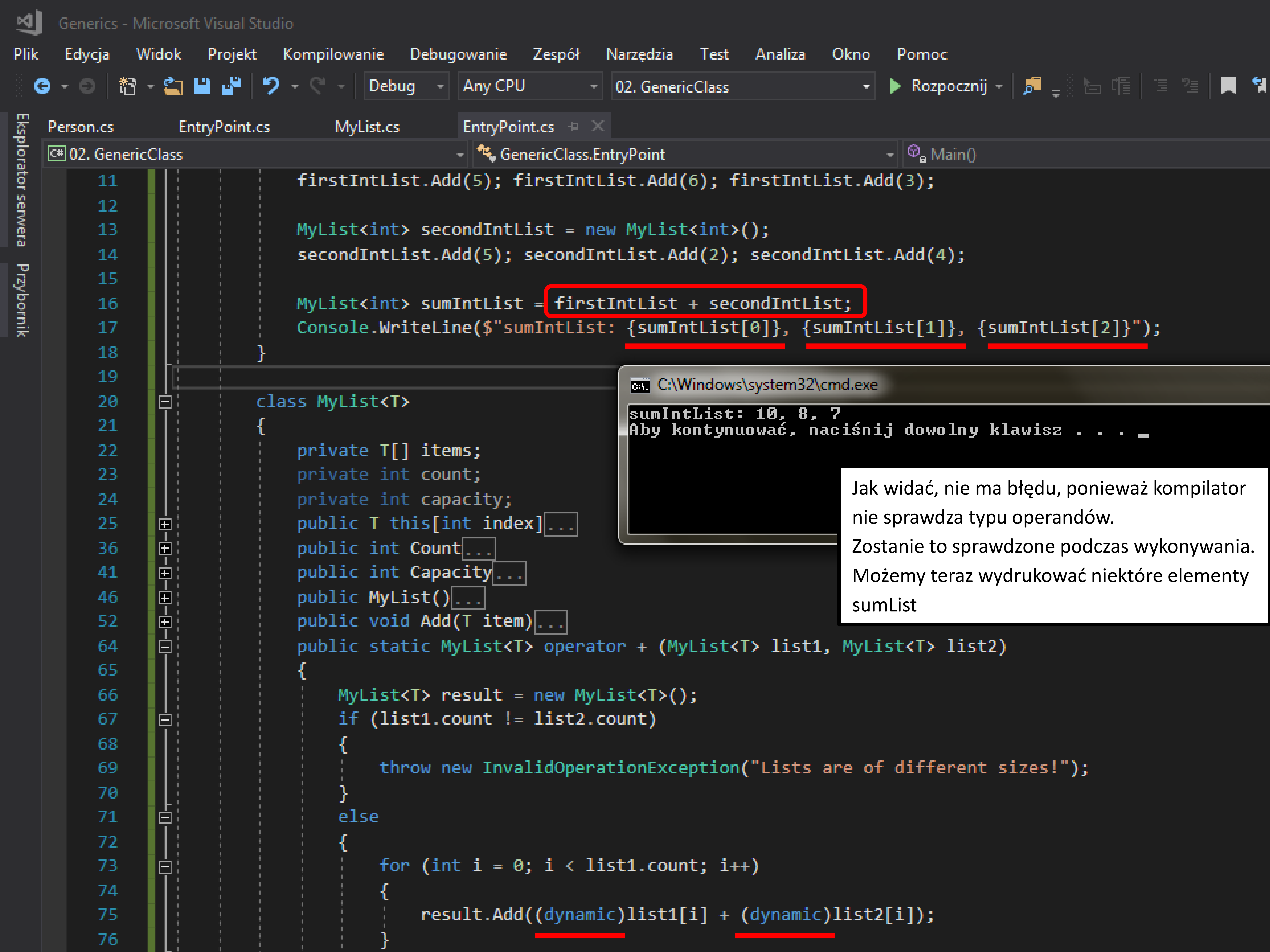
Eksplorator serwera | Przybornik

```
10            MyList<int> firstIntList = new MyList<int>();
11            firstIntList.Add(5);
12            firstIntList.Add(6);
13            firstIntList.Add(3);
14
15            MyList<int> secondIntList = new MyList<int>();
16            secondIntList.Add(5);
17            secondIntList.Add(2);
18            secondIntList.Add(4);
19
20            MyList<int> sumIntList = firstIntList + secondIntList;
21        }
22
23        class MyList<T>
24        {
25            private T[] items;
26            private int count;
27            private int capacity;
28            public T this[int index]...
39            public int Count...
44            public int Capacity...
49            public MyList()...
55            public void Add(T item)...
67            public static MyList<T> operator + (MyList<T> list1, MyList<T> list2)
68            {
69                MyList<T> result = new MyList<T>();
70                return result;
71            }
72        }
73    }
74 }
75
```

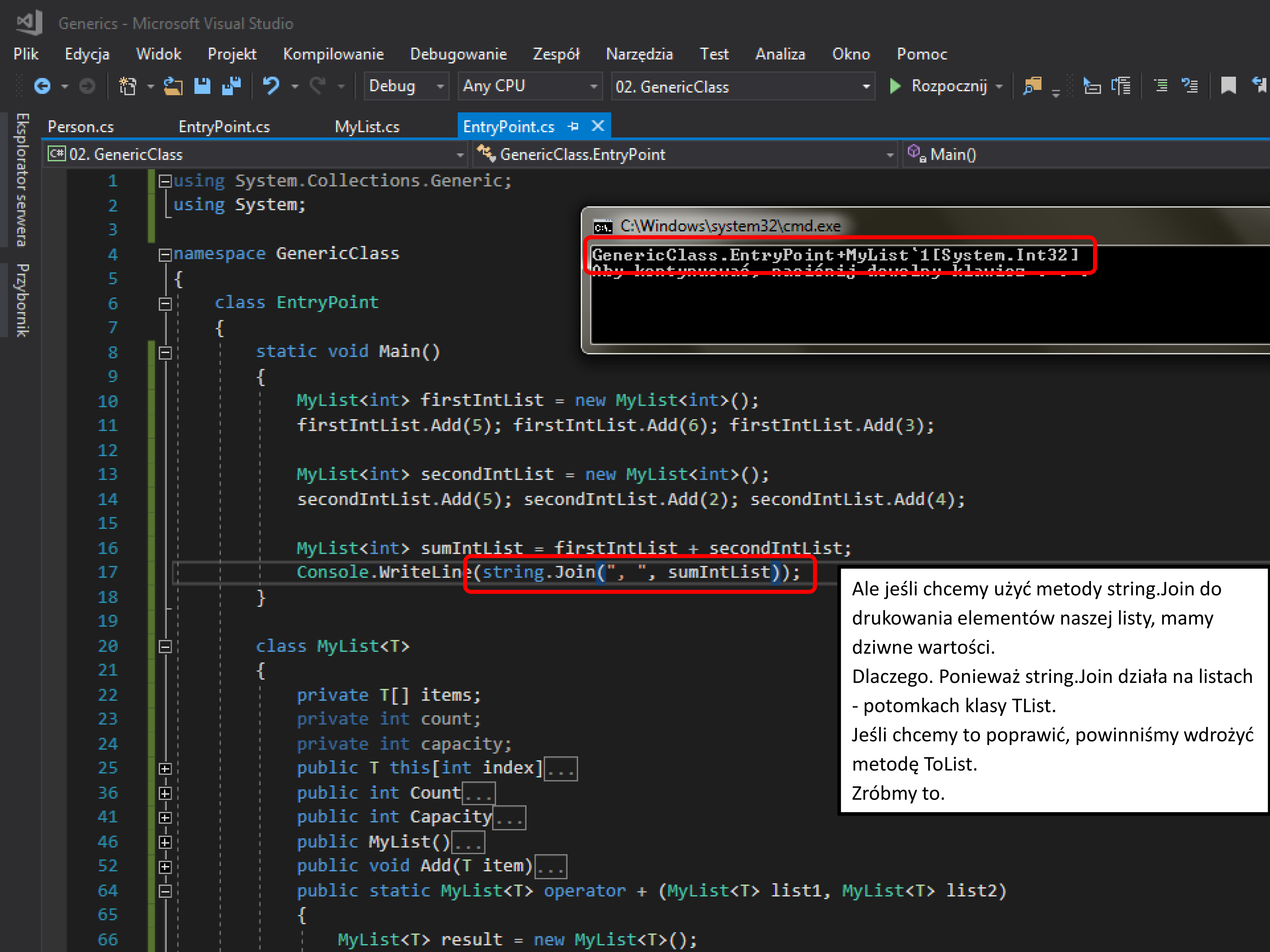
Możemy porównywać nasze obiekty, ale możemy również wykonywać na nich operacje matematyczne takie jak dodawanie, mnożenie, dzielenie i odejmowanie.
Zróbmy to z klasą Lista.
Nauczmy C-Sharpa, jak dodawać dwie listy.
Mamy firstIntegerList i secondIntegerList. Są to trzy elementy, a trzecia lista to suma dwóch poprzednich list.
Obecnie nie możemy tego zrobić, ponieważ C-Sharp nie wie, jak dodać te dwa obiekty.
Przeciążymy operator matematyczny dodawania. Musimy to zrobić publicznie, statycznie i zwrócić rodzaj tej operacji.
Zwracamy po prostu sumę wszystkich liczb całkowitych z listy.



Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs*
Eksplorator serwera
Przybornik
02. GenericClass
GenericClass.EntryPoint
Main()
44 public int Capacity...
49 public MyList()...
55 public void Add(T item)...
67 public static MyList<T> operator + (MyList<T> list1, MyList<T> list2)
68 {
69 MyList<T> result = new MyList<T>();
70 if (list1.count != list2.count)
71 {
72 throw new InvalidOperationException("Lists are of different sizes!");
73 }
74 else
75 {
76 for (int i = 0; i < list1.count; i++)
77 {
78 result.Add(list1[i] + list2[i]);
79 }
80 }
81 return result;
82 }
83 }
84 }
85 }
Możemy więc dodawać tylko listy o tej samej długości.
Więc jeśli długość listy pierwszej jest inna niż długość listy drugiej,
wyrzucimy wyjątek InvalidOperationException - listy mają różne rozmiary.
W przeciwnym razie dodamy je i zrobimy to po prostu za pomocą pętli for.
Utworzymy listę wyników z dodatkiem każdego elementu z dwóch list.
Nadal mamy błąd, ponieważ kompilator nie wie, czy dwa argumenty dodawania są tego samego typu.
Rozwiązaniem tego problemu jest słowo kluczowe dynamic.
109 %
Lista błędów
Całe rozwiązanie
1 Błąd
0 Ostrzeżenia
0 Komunikaty
Kompilacja
Kod Opis Projekt Plik W... Stan pomin
CS0019 Nie można zastosować operatora „+" do argumentów operacji typu „T" lub „T". 02. GenericClass EntryPoint.cs 78 Aktywne



sumIntList: 10, 8, 7
Aby kontynuować, naciśnij dowolny klawisz . . .
Jak widać, nie ma błędu, ponieważ kompilator nie sprawdza typu operandów.
Zostanie to sprawdzone podczas wykonywania.
Możemy teraz wydrukować niektóre elementy sumList



Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System.Collections.Generic;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
MyList<int> firstIntList = new MyList<int>();
firstIntList.Add(5); firstIntList.Add(6); firstIntList.Add(3);
MyList<int> secondIntList = new MyList<int>();
secondIntList.Add(5); secondIntList.Add(2); secondIntList.Add(4);
MyList<int> sumIntList = firstIntList + secondIntList;
Console.WriteLine(string.Join(", ", sumIntList));
}
class MyList<T>
{
private T[] items;
private int count;
private int capacity;
public T this[int index]...
public int Count...
public int Capacity...
public MyList()...
public void Add(T item)...
public static MyList<T> operator + (MyList<T> list1, MyList<T> list2)
{
MyList<T> result = new MyList<T>();
C:\Windows\system32\cmd.exe
GenericClass.EntryPoint+MyList`1[System.Int32]
Ale jeśli chcemy użyć metody string.Join do drukowania elementów naszej listy, mamy dziwne wartości.
Dlaczego. Ponieważ string.Join działa na listach - potomkach klasy TList.
Jeśli chcemy to poprawić, powinniśmy wdrożyć metodę ToList.
Zróbmy to.

Generics - Microsoft Visual Studio

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

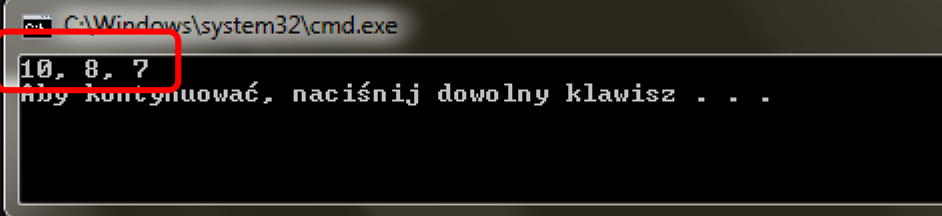
Debug | Any CPU | 02. GenericClass | Rozpocznij

Person.cs | EntryPoint.cs | MyList.cs | EntryPoint.cs

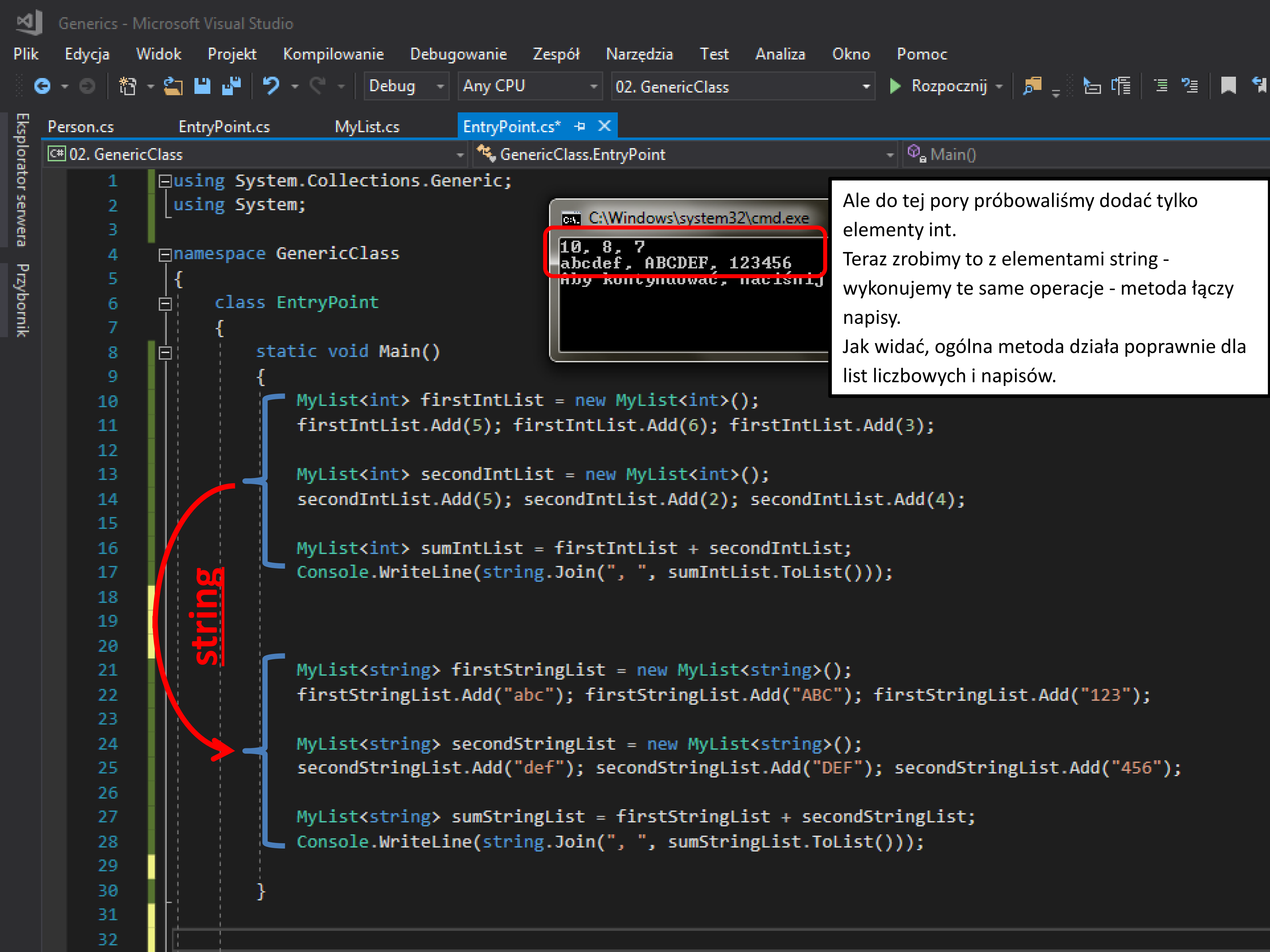
02. GenericClass | GenericClass.EntryPoint.MyList<T> | ToList()

Eksplorator serwera | Przybornik

```
9         {
10            MyList<int> firstIntList = new MyList<int>();
11            firstIntList.Add(5); firstIntList.Add(6); firstIntList.Add(3);
12
13            MyList<int> secondIntList = new MyList<int>();
14            secondIntList.Add(5); secondIntList.Add(2); secondIntList.Add(4);
15
16            MyList<int> sumIntList = firstIntList + secondIntList;
17            Console.WriteLine(string.Join(", ", sumIntList.ToList()));
18        }
19
20        class MyList<T>
21        {
22            private T[] items;
23            private int count;
24            private int capacity;
25            public T this[int index]...
36            public int Count...
41            public int Capacity...
46            public MyList()...
52            public void Add(T item)...
64            public static MyList<T> operator + (MyList<T> list1, MyList<T> list2)...
80            public List<T> ToList()
81            {
82                List<T> list = new List<T>();
83                for(int i = 0; i < count; i++)
84                {
85                    list.Add(items[i]);
86                }
87                return list;
88            }
89        }
```



Po prostu piszemy generyczną metodę ToList - tworzymy nowy obiekt List, a następnie dodajemy wszystkie elementy z naszej Listy, a następnie zwracamy wynik.
Teraz możemy użyć metody string.Join, aby pobrać wszystkie elementy z naszej listy.
Widać, że mamy listę wyników, w której wszystkie elementy są sumą elementów z poprzednich list.
5 + 5 = 10; sześć i dwa to osiem, a 3 plus 4 to 7;


Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs*
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System.Collections.Generic;
using System;
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
MyList<int> firstIntList = new MyList<int>();
firstIntList.Add(5); firstIntList.Add(6); firstIntList.Add(3);
MyList<int> secondIntList = new MyList<int>();
secondIntList.Add(5); secondIntList.Add(2); secondIntList.Add(4);
MyList<int> sumIntList = firstIntList + secondIntList;
Console.WriteLine(string.Join(", ", sumIntList.ToList()));
MyList<string> firstStringList = new MyList<string>();
firstStringList.Add("abc"); firstStringList.Add("ABC"); firstStringList.Add("123");
MyList<string> secondStringList = new MyList<string>();
secondStringList.Add("def"); secondStringList.Add("DEF"); secondStringList.Add("456");
MyList<string> sumStringList = firstStringList + secondStringList;
Console.WriteLine(string.Join(", ", sumStringList.ToList()));
}
string
C:\Windows\system32\cmd.exe
10, 8, 7
abcdef, ABCDEF, 123456
Ale do tej pory próbowaliśmy dodać tylko elementy int.
Teraz zrobimy to z elementami string - wykonujemy te same operacje - metoda łączy napisy.
Jak widać, ogólna metoda działa poprawnie dla list liczbowych i napisów.