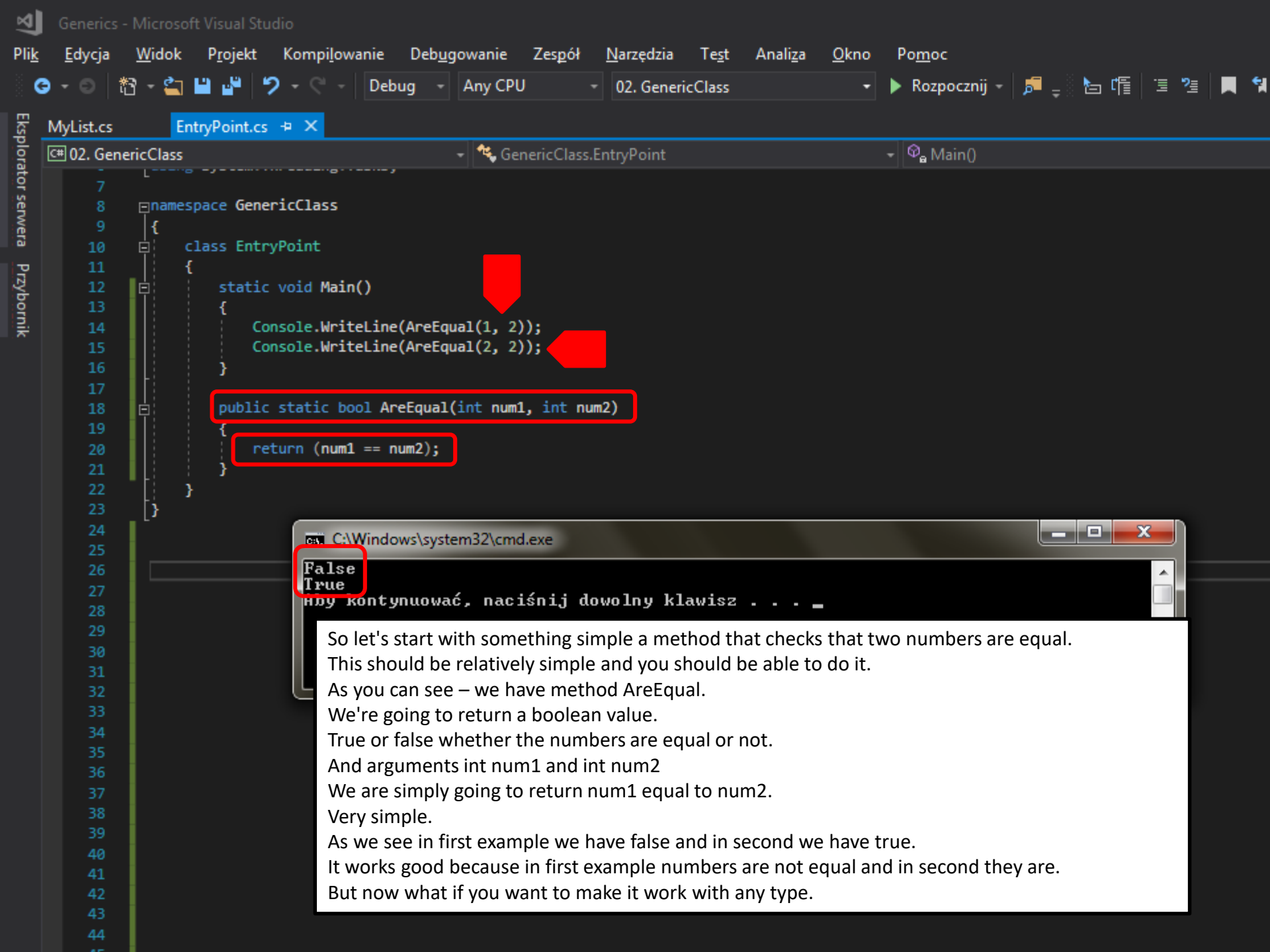
Generic methods

So we are finally in the generics.
At first they look complicated.
But in reality they really are not.
Generics of course meaning that they can apply to more than one type.
So a method or a class which is generic - can work both for integer type and for string type and for any other type.


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(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 . . .
So let's start with something simple a method that checks that two numbers are equal.
This should be relatively simple and you should be able to do it.
As you can see – we have method AreEqual.
We're going to return a boolean value.
True or false whether the numbers are equal or not.
And arguments int num1 and int num2
We are simply going to return num1 equal to num2.
Very simple.
As we see in first example we have false and in second we have true.
It works good because in first example numbers are not equal and in second they are.
But now what if you want to make it work with any type.

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

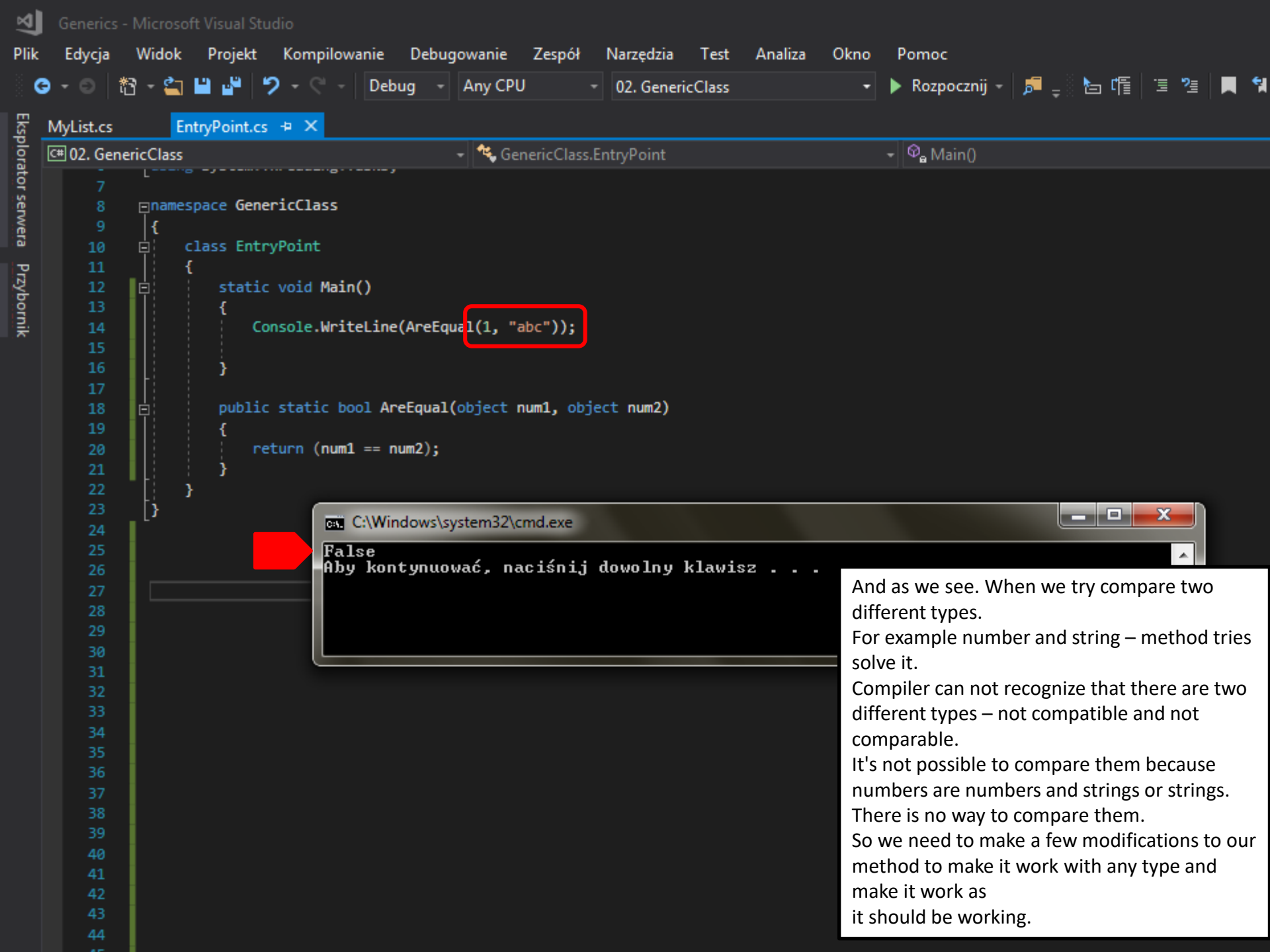
02. GenericClass | GenericClass.EntryPoint | Main()

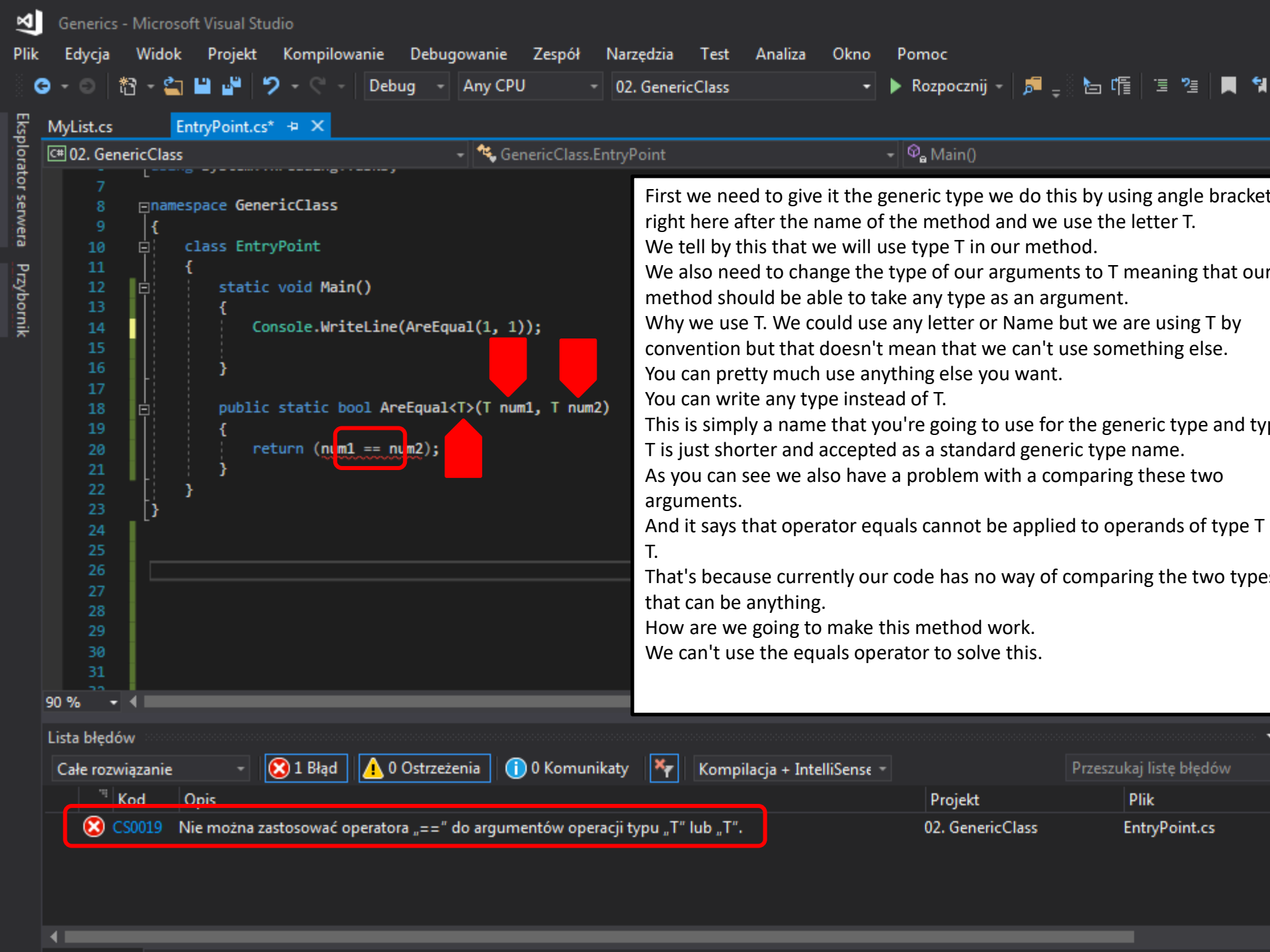
```
7
8   namespace GenericClass
9   {
10      class EntryPoint
11      {
12          static void Main()
13          {
14              Console.WriteLine(AreEqual(2, 2));
15
16              Console.WriteLine(AreEqual('a', 'a'));
17
18              Console.WriteLine(AreEqual("abba", "abba"));
19              Console.WriteLine(AreEqual("abba", "abc"));
20          }
21
22          public static bool AreEqual(object num1, object num2)
23          {
24              return (num1 == num2);
25          }
26      }
27  }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
```

What should we do.
Well one solution is to change the integer type to object.
That way we will be able to give with any type as an argument that we want.
Let's try do it.
We try it for three types. As we can see it returns wrong answers for integer type and for character type.
For string type it works good.
So it doesn't work for any types.
We can not use the object type.

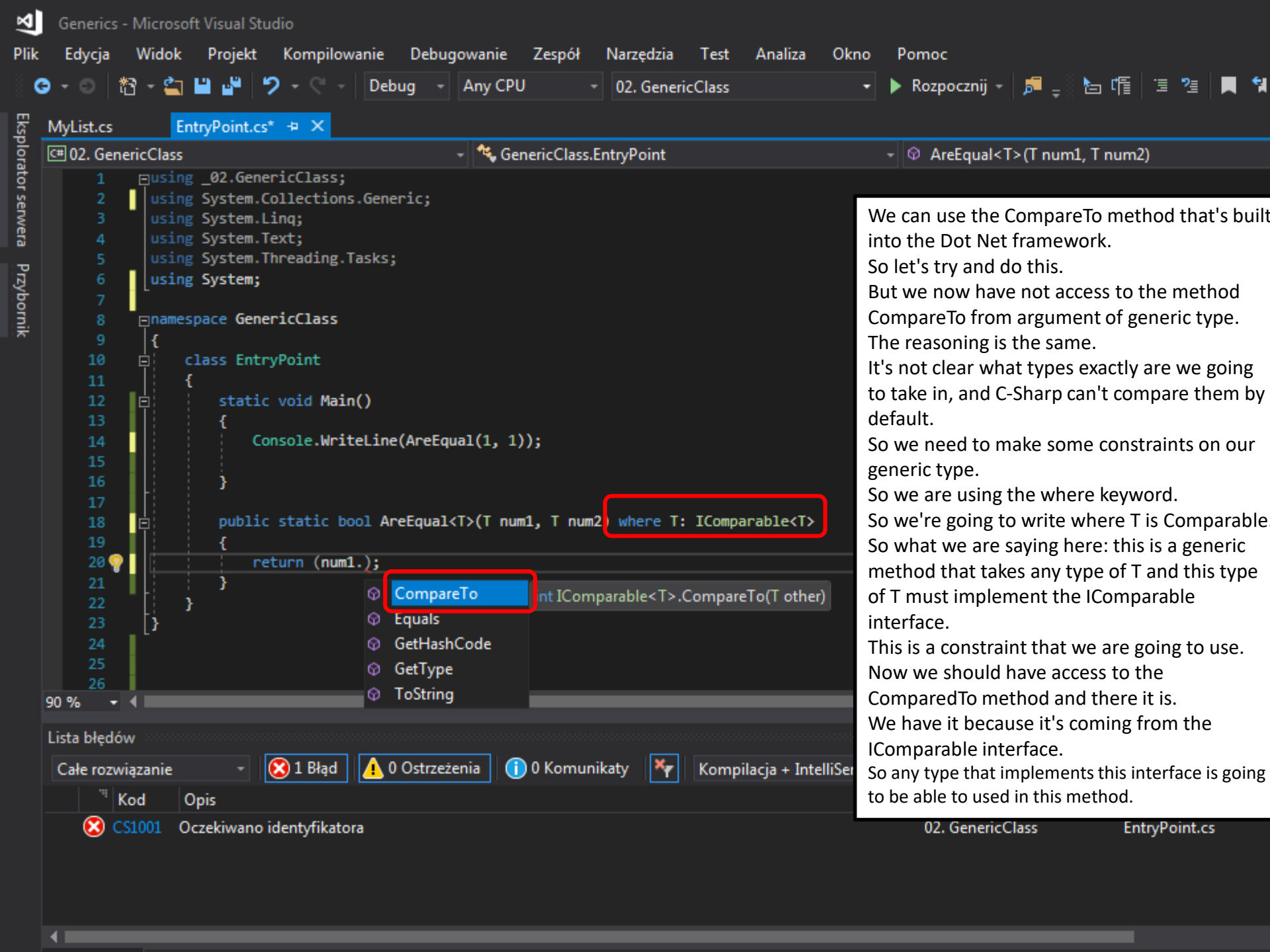
C:\Windows\system32\cmd.exe

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

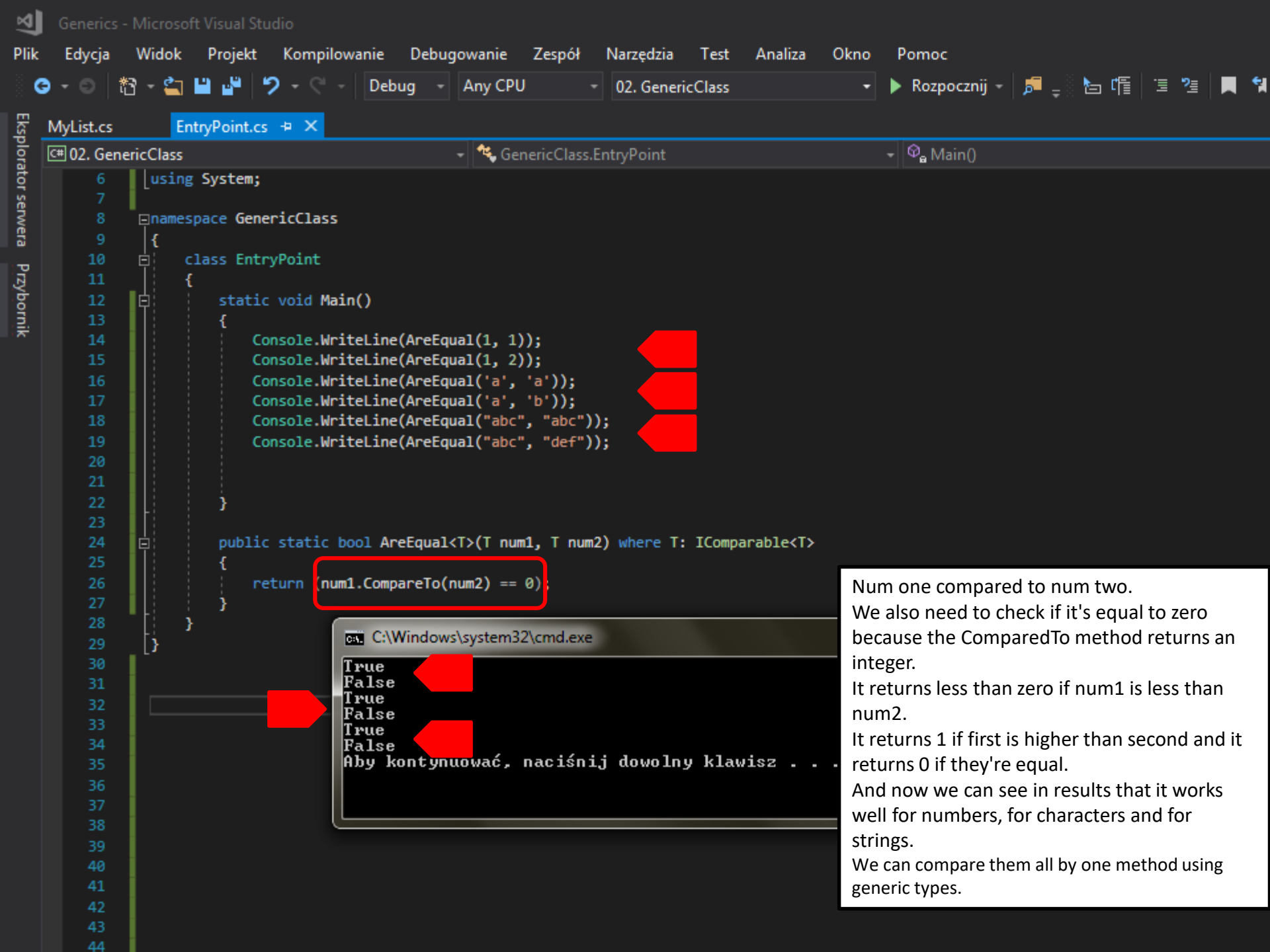

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
namespace GenericClass
{
class EntryPoint
{
static void Main()
{
Console.WriteLine(AreEqual(1, "abc"));
}
public static bool AreEqual(object num1, object num2)
{
return (num1 == num2);
}
}
}
C:\Windows\system32\cmd.exe
False
Aby kontynuować, naciśnij dowolny klawisz . . .
And as we see. When we try compare two different types.
For example number and string – method tries solve it.
Compiler can not recognize that there are two different types – not compatible and not comparable.
It's not possible to compare them because numbers are numbers and strings or strings. There is no way to compare them.
So we need to make a few modifications to our method to make it work with any type and make it work as
it should be working.

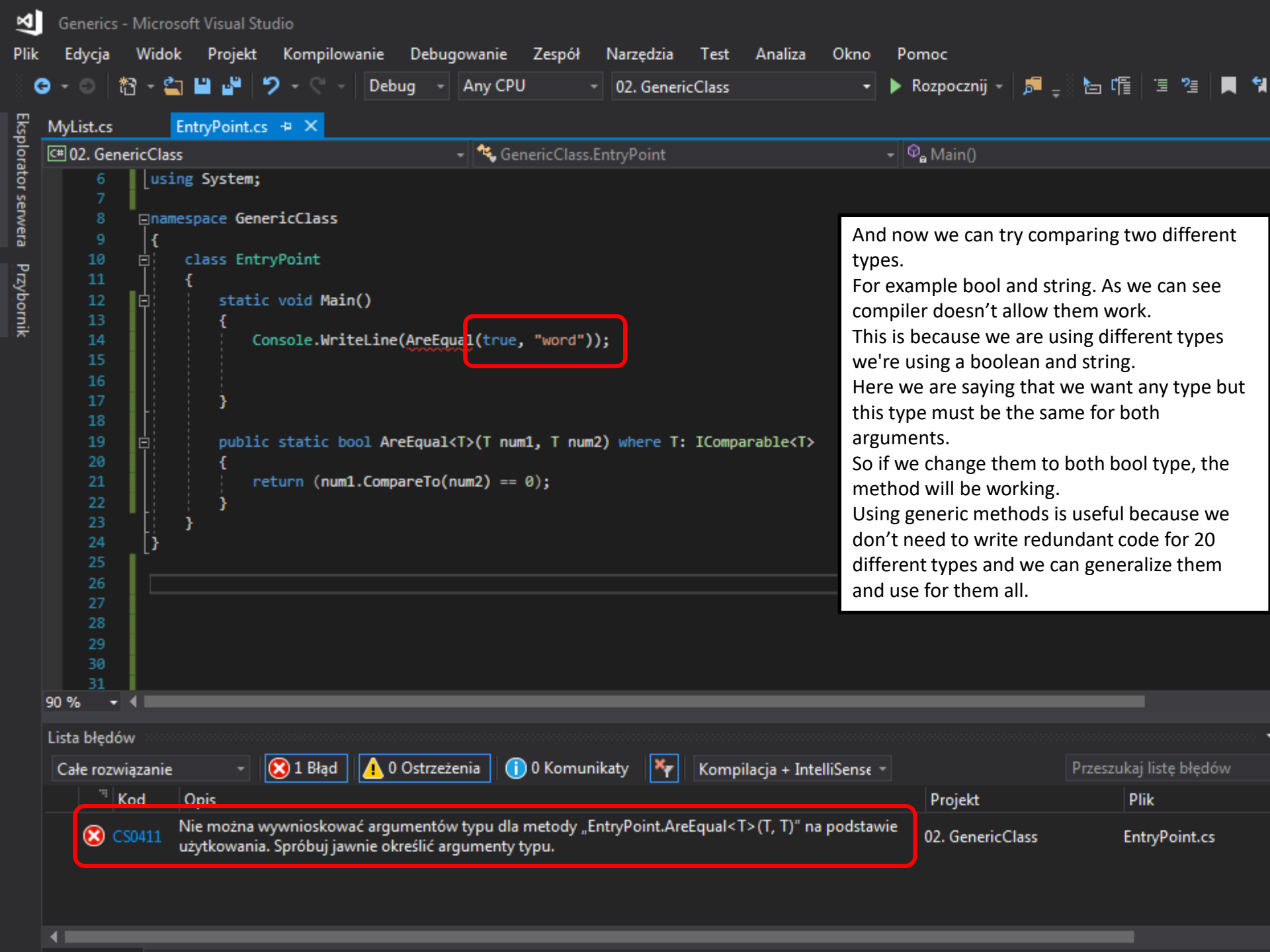


First we need to give it the generic type we do this by using angle bracket right here after the name of the method and we use the letter T.
We tell by this that we will use type T in our method.
We also need to change the type of our arguments to T meaning that our method should be able to take any type as an argument.
Why we use T. We could use any letter or Name but we are using T by convention but that doesn't mean that we can't use something else.
You can pretty much use anything else you want.
You can write any type instead of T.
This is simply a name that you're going to use for the generic type and typ
T is just shorter and accepted as a standard generic type name.
As you can see we also have a problem with a comparing these two arguments.
And it says that operator equals cannot be applied to operands of type T T.
That's because currently our code has no way of comparing the two types that can be anything.
How are we going to make this method work.
We can't use the equals operator to solve this.



We can use the CompareTo method that's built into the Dot Net framework.
So let's try and do this.
But we now have not access to the method CompareTo from argument of generic type.
The reasoning is the same.
It's not clear what types exactly are we going to take in, and C-Sharp can't compare them by default.
So we need to make some constraints on our generic type.
So we are using the where keyword.
So we're going to write where T is Comparable.
So what we are saying here: this is a generic method that takes any type of T and this type of T must implement the IComparable interface.
This is a constraint that we are going to use.
Now we should have access to the ComparedTo method and there it is.
We have it because it's coming from the IComparable interface.
So any type that implements this interface is going to be able to used in this method.

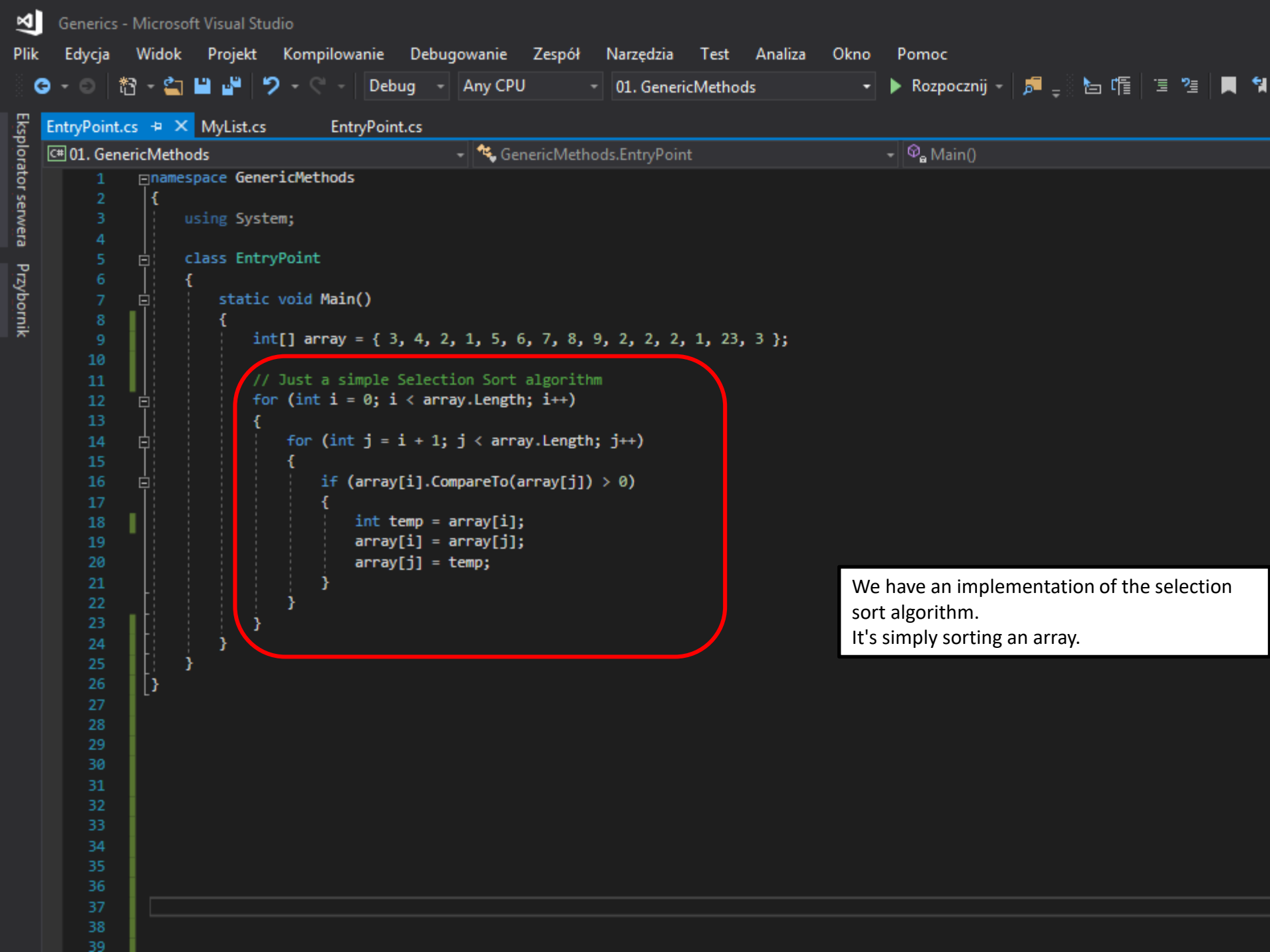

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
Eksplorator serwera
Przybornik
using System;
namespace GenericClass
{
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 . . .
Num one compared to num two.
We also need to check if it's equal to zero because the ComparedTo method returns an integer.
It returns less than zero if num1 is less than num2.
It returns 1 if first is higher than second and it returns 0 if they're equal.
And now we can see in results that it works well for numbers, for characters and for strings.
We can compare them all by one method using generic types.



And now we can try comparing two different types.
For example bool and string. As we can see compiler doesn't allow them work.
This is because we are using different types we're using a boolean and string.
Here we are saying that we want any type but this type must be the same for both arguments.
So if we change them to both bool type, the method will be working.
Using generic methods is useful because we don't need to write redundant code for 20 different types and we can generalize them and use for them all.

Generic methods for sorting collections

All right let's try another example here.


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;
}
}
}
}
}
}
We have an implementation of the selection sort algorithm.
It's simply sorting an array.

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

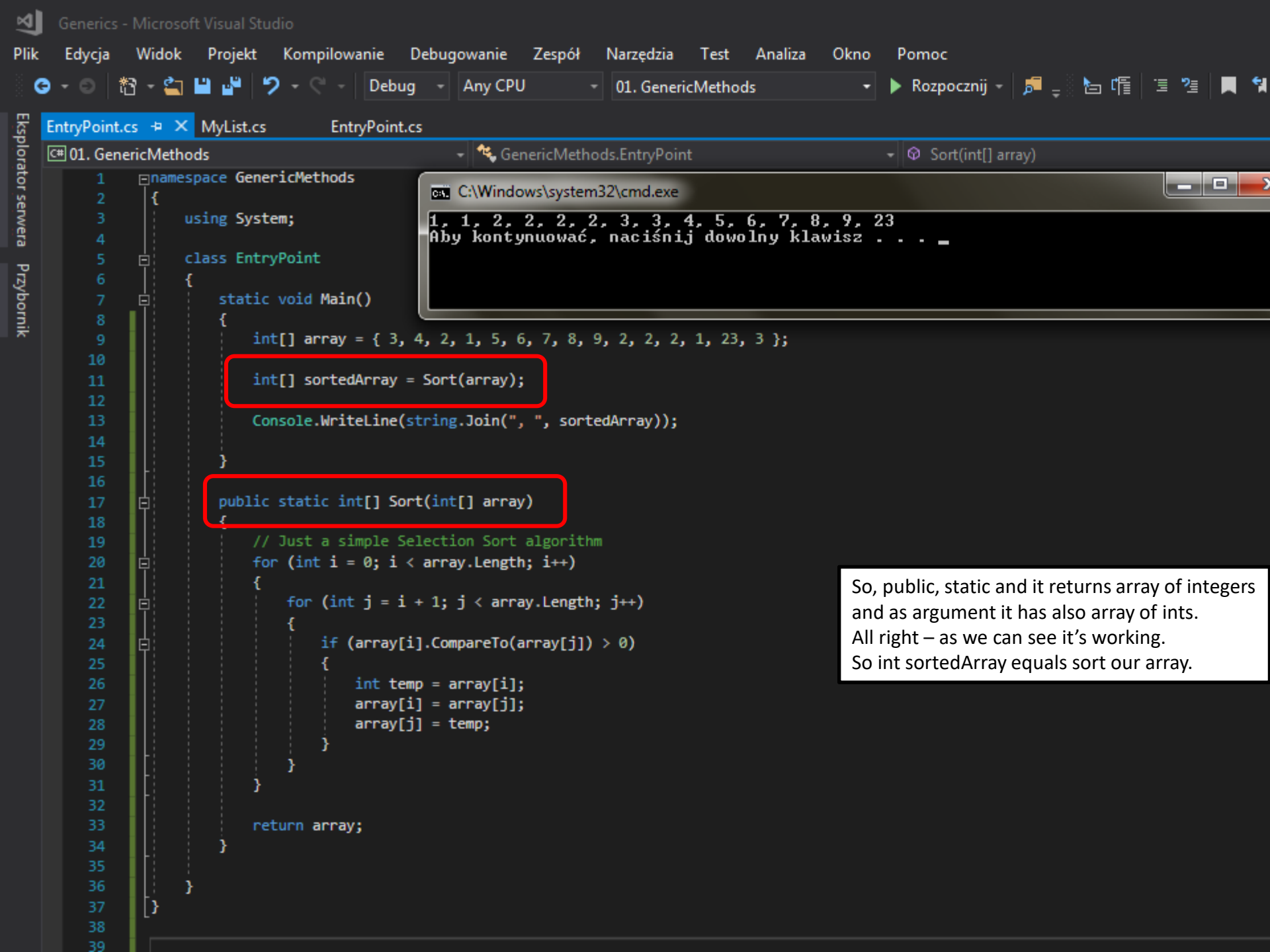
01. GenericMethods | GenericMethods.EntryPoint | Main()

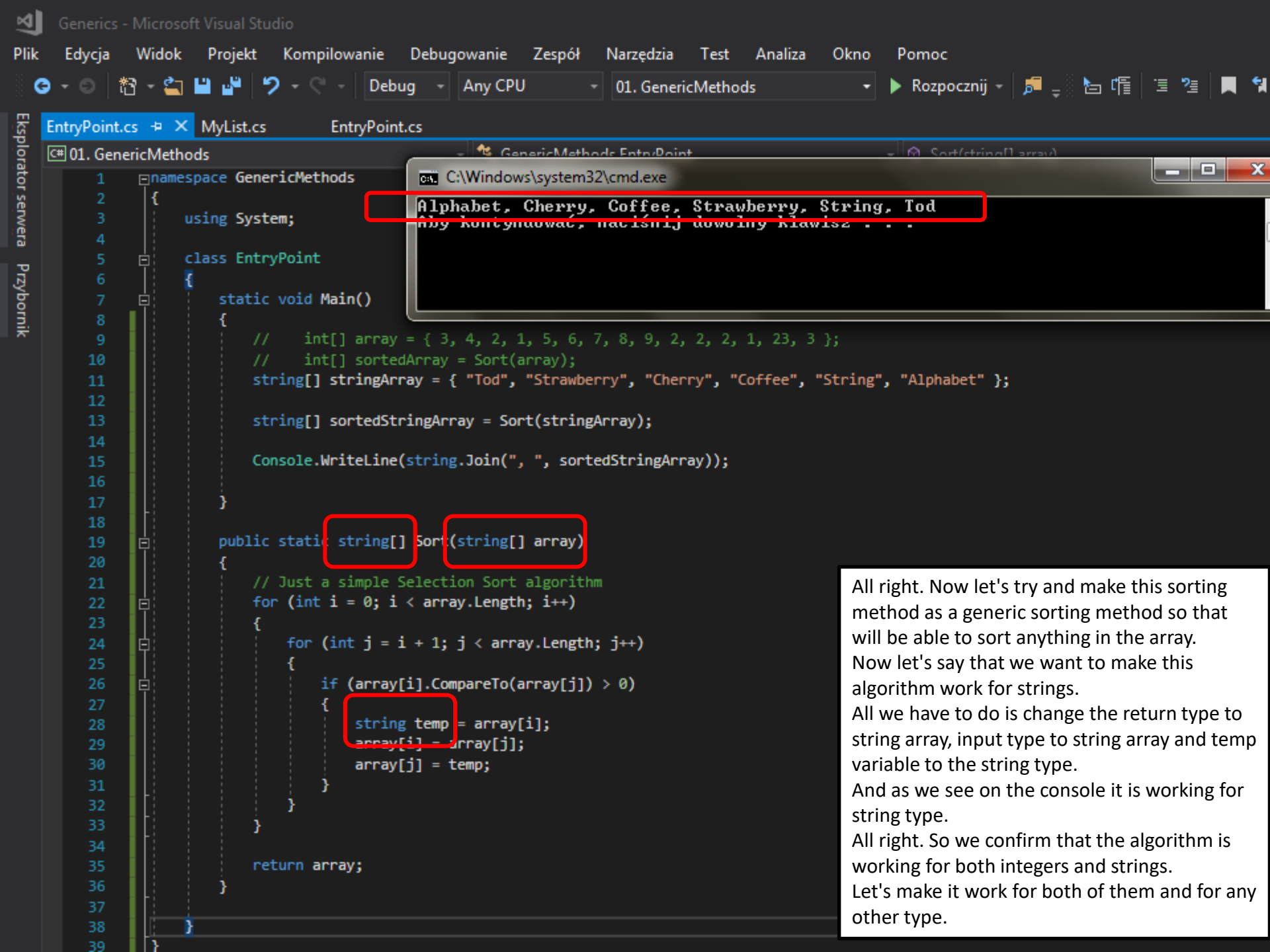
```
1   namespace GenericMethods
2   {
3       using System;
4
5       class EntryPoint
6       {
7           static void Main()
8           {
9               int[] array = { 3, 4, 2, 1, 5, 6, 7, 8, 9, 2, 2, 2, 1, 23, 3 };
10
11              // Just a simple Selection Sort algorithm
12              for (int i = 0; i < array.Length; i++)
13              {
14                  for (int j = i + 1; j < array.Length; j++)
15                  {
16                      if (array[i].CompareTo(array[j]) > 0)
17                      {
18                          int temp = array[i];
19                          array[i] = array[j];
20                          array[j] = temp;
21                      }
22                  }
23              }
24
25              Console.WriteLine(string.Join(", ", array));
26
27          }
28      }
29  }
30
31
32
33
34
35
36
37
38
39
```

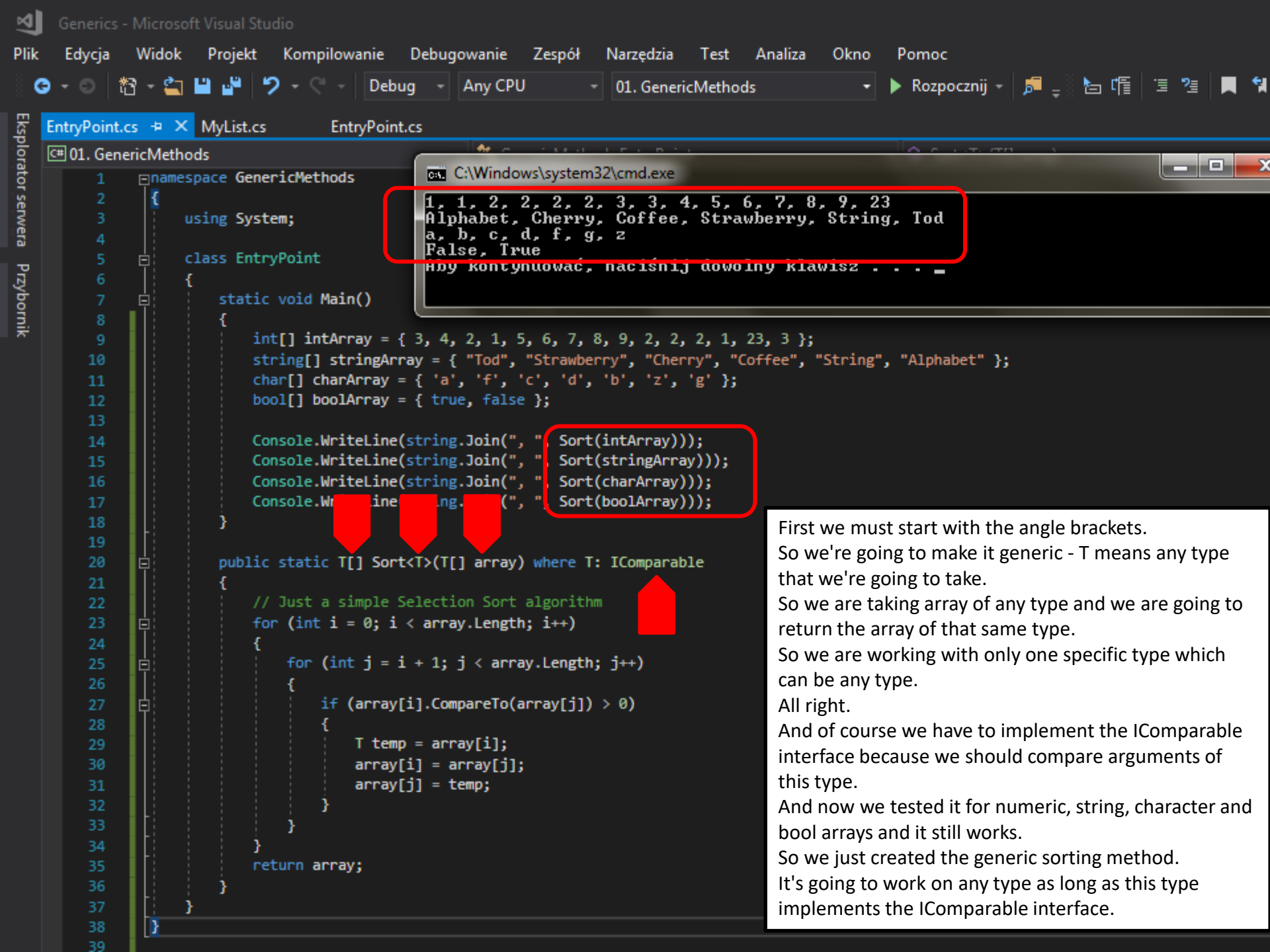
C:\Windows\system32\cmd.exe

```
1, 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 7, 8, 9, 23
Aby kontynuować, naciśnij dowolny klawisz . . .
```

To confirm that it's working, we're going to print the results of our array after it was sorted and we show it by using string.Join method. As we can see method works good and all numbers in array are sorted.
So our algorithm is working. Let's extract this algorithm into a method.


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;
}
}
}
So, public, static and it returns array of integers
and as argument it has also array of ints.
All right – as we can see it's working.
So int sortedArray equals sort our array.

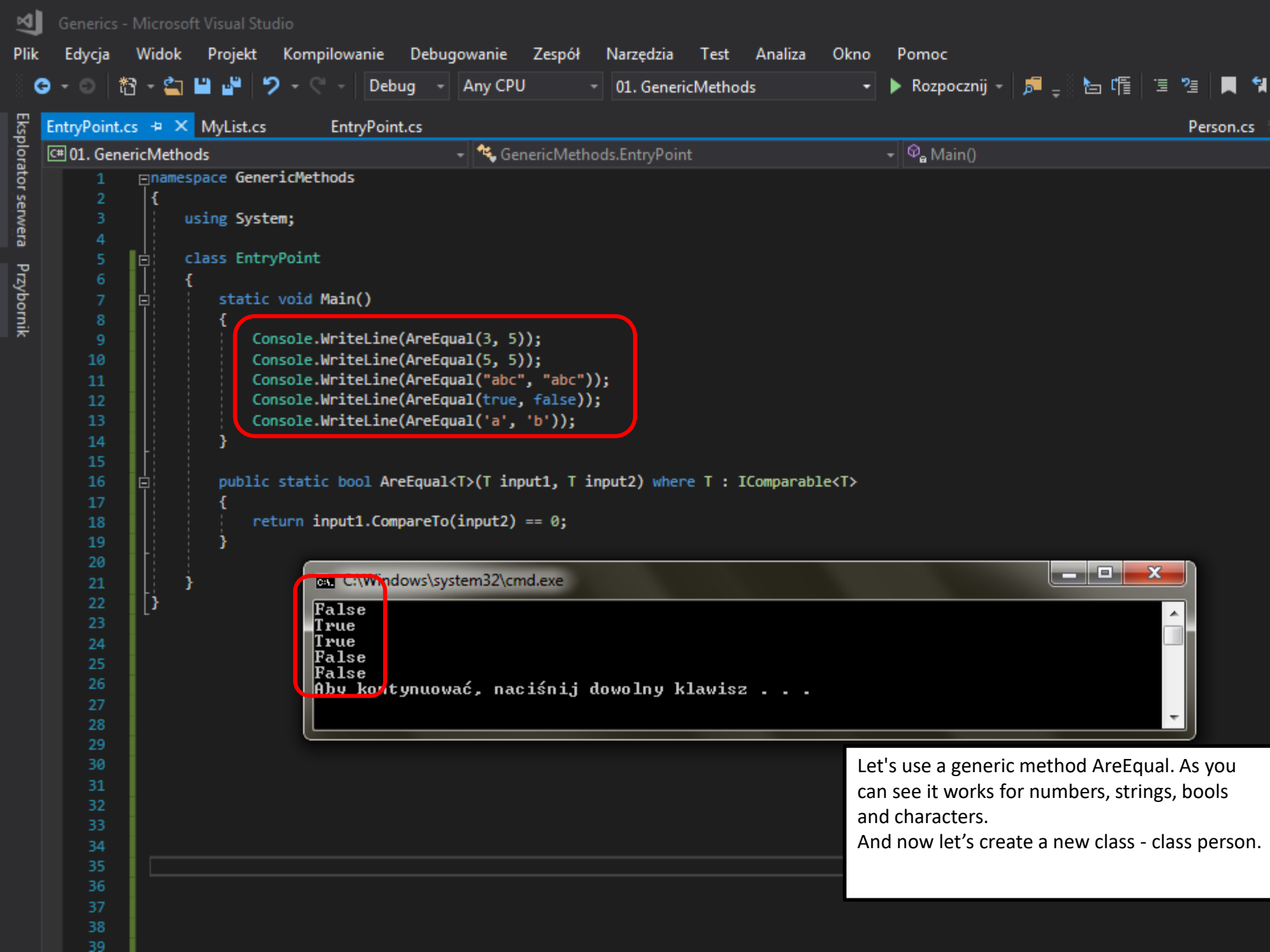

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. GenericMethods
Rozpocznij
EntryPoint.cs
MyList.cs
EntryPoint.cs
01. GenericMethods
C:\Windows\system32\cmd.exe
Alphabet, Cherry, Coffee, Strawberry, String, 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[i] = array[j];
array[j] = temp;
}
}
}
return array;
}
}
}
All right. Now let's try and make this sorting method as a generic sorting method so that will be able to sort anything in the array.
Now let's say that we want to make this algorithm work for strings.
All we have to do is change the return type to string array, input type to string array and temp variable to the string type.
And as we see on the console it is working for string type.
All right. So we confirm that the algorithm is working for both integers and strings.
Let's make it work for both of them and for any other type.



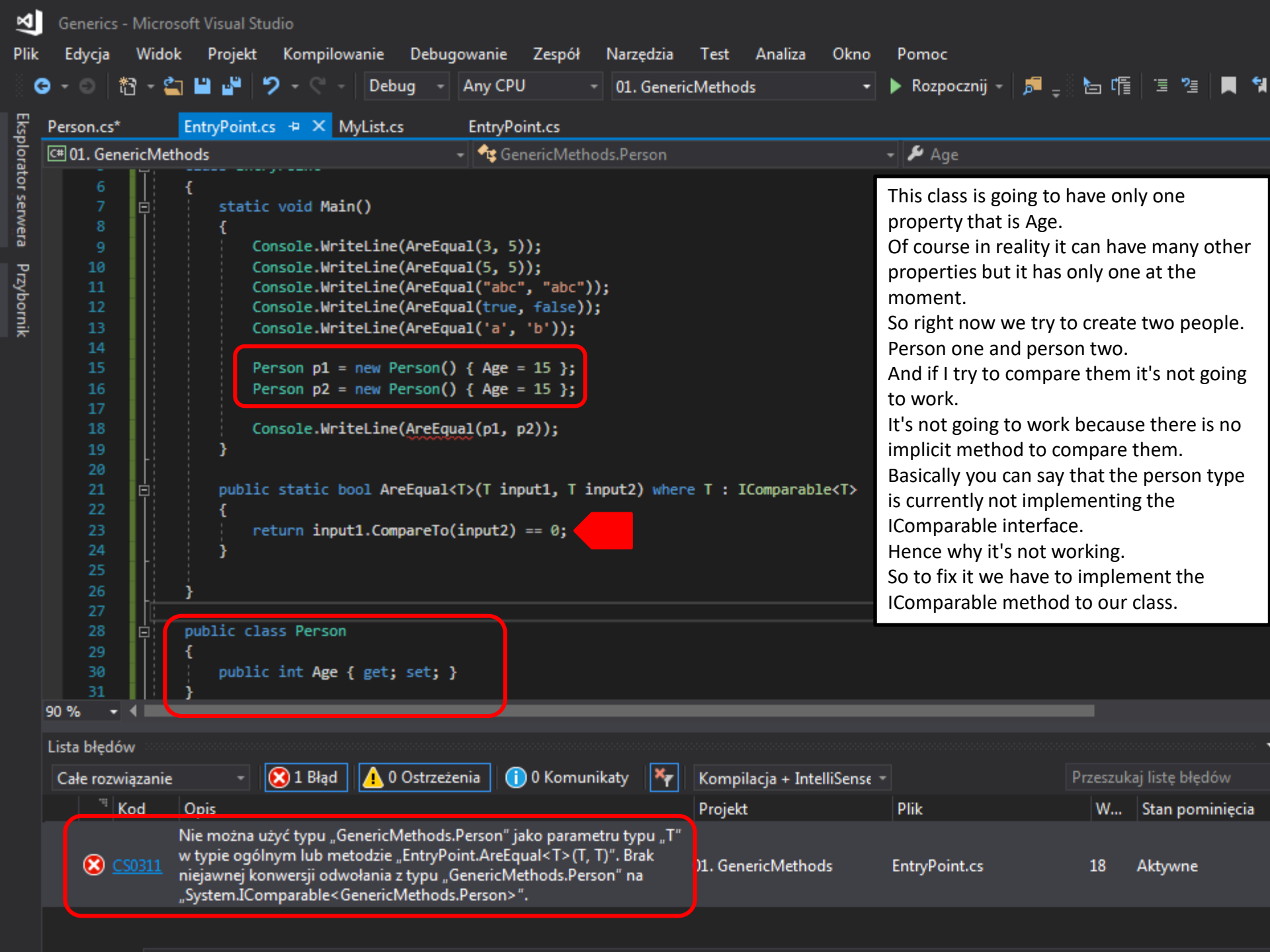
First we must start with the angle brackets.
So we're going to make it generic - T means any type that we're going to take.
So we are taking array of any type and we are going to return the array of that same type.
So we are working with only one specific type which can be any type.
All right.
And of course we have to implement the IComparable interface because we should compare arguments of this type.
And now we tested it for numeric, string, character and bool arrays and it still works.
So we just created the generic sorting method.
It's going to work on any type as long as this type implements the IComparable interface.

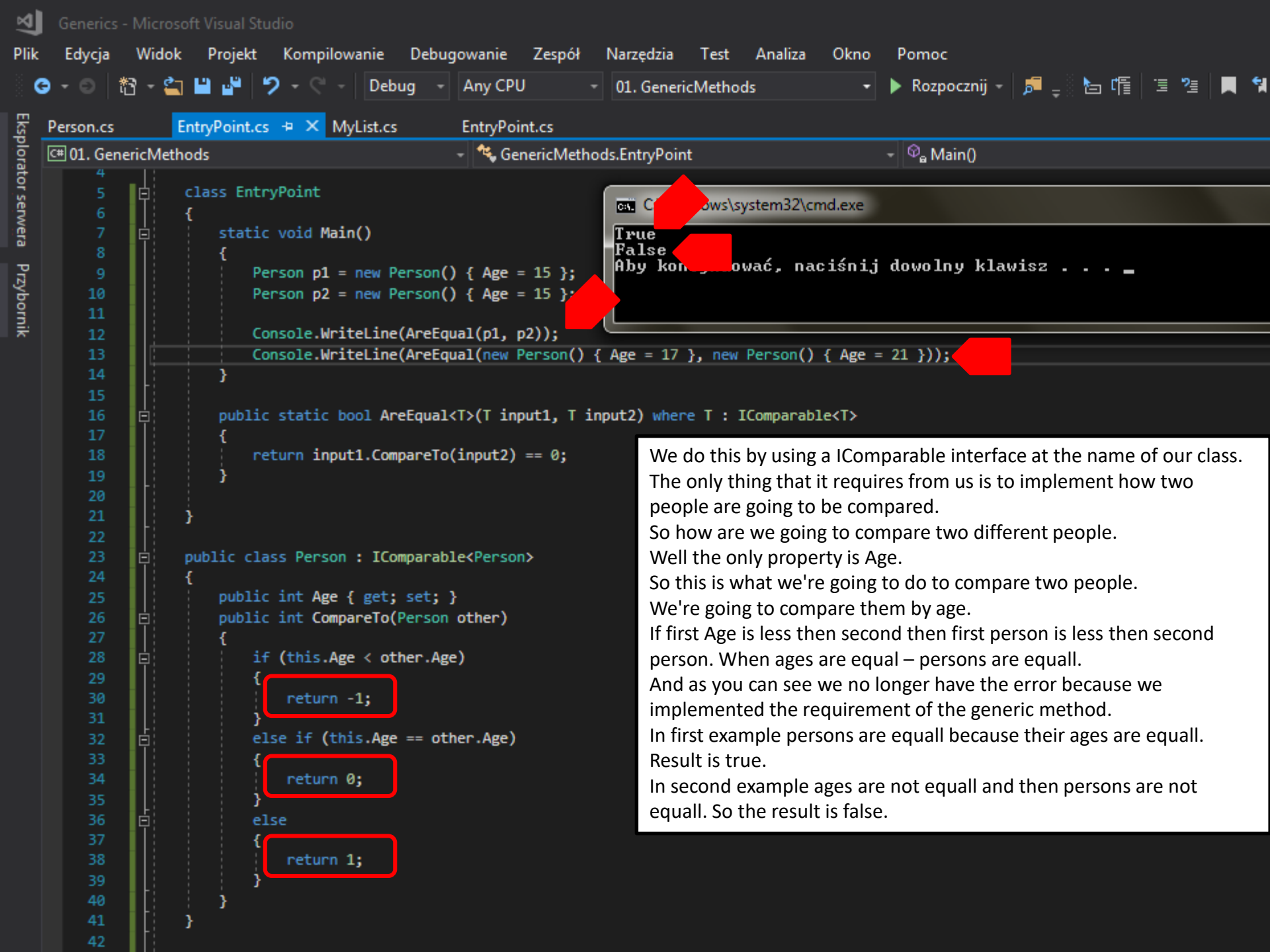
Implementing the IComparable interface in a class

All right we have created two generic methods: AreEqual and Sort that work on any type.
But so far we have tried it with integers and with strings and some simple types.
How about if we have our own custom type or a class.



Let's use a generic method AreEqual. As you can see it works for numbers, strings, bools and characters.
And now let’s create a new class - class person.

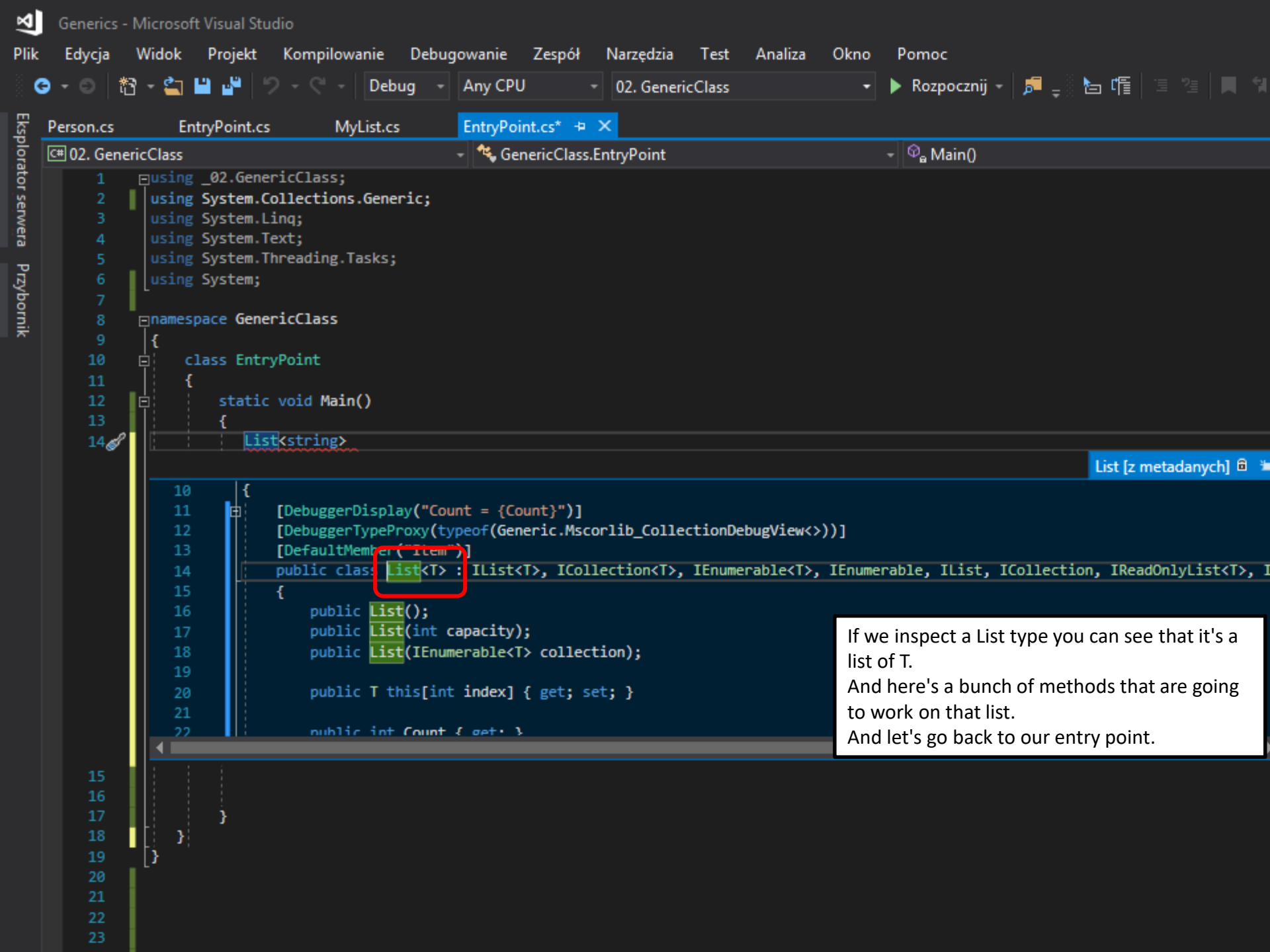

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
01. GenericMethods
Rozpocznij
Person.cs*
EntryPoint.cs
MyList.cs
EntryPoint.cs
01. GenericMethods
GenericMethods.Person
Age
Eksplorator serwera
Przybornik
6 {
7 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 }
90 %
This class is going to have only one property that is Age.
Of course in reality it can have many other properties but it has only one at the moment.
So right now we try to create two people.
Person one and person two.
And if I try to compare them it's not going to work.
It's not going to work because there is no implicit method to compare them.
Basically you can say that the person type is currently not implementing the IComparable interface.
Hence why it's not working.
So to fix it we have to implement the IComparable method to our class.
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

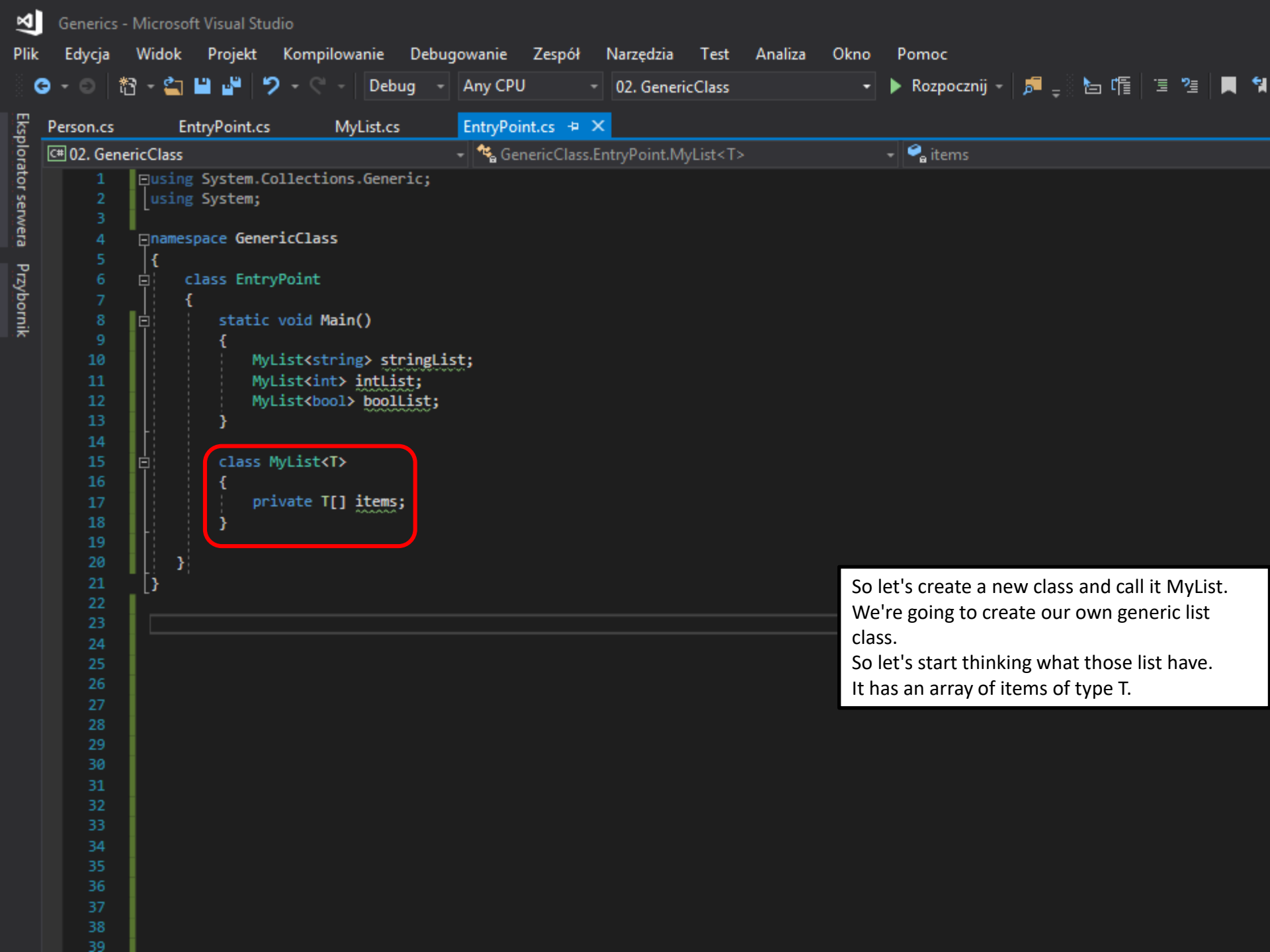


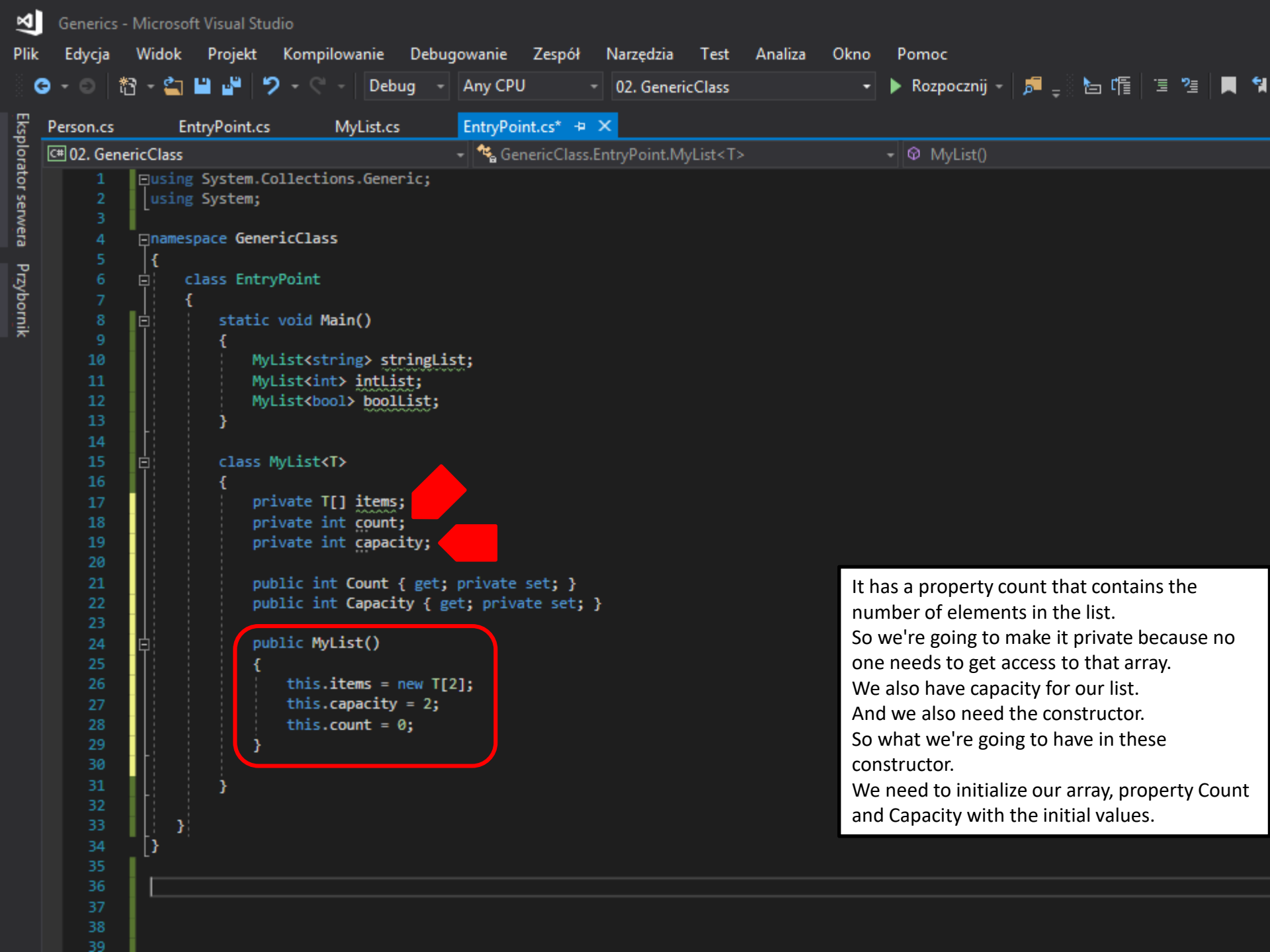
We do this by using a IComparable interface at the name of our class.
The only thing that it requires from us is to implement how two people are going to be compared.
So how are we going to compare two different people.
Well the only property is Age.
So this is what we're going to do to compare two people.
We're going to compare them by age.
If first Age is less then second then first person is less then second person. When ages are equal – persons are equall.
And as you can see we no longer have the error because we implemented the requirement of the generic method.
In first example persons are equall because their ages are equall.
Result is true.
In second example ages are not equall and then persons are not equall. So the result is false.

Generic classes

So we not only have generic methods but we also have generic classes and there is one particular generic class that we use over and over again.
This class is a List class.
Remember that after you write a list you have always angle brackets to deal with what type the list is going to have.
We can write list with integer or string or anything else that we may want the list contain.
This is the perfect example of a generic class.


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<>))]
[DefaultMember("Item")]
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; }
public int Count { get; }
If we inspect a List type you can see that it's a list of T.
And here's a bunch of methods that are going to work on that list.
And let's go back to our entry point.


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;
}
}
}
So let's create a new class and call it MyList.
We're going to create our own generic list class.
So let's start thinking what those list have.
It has an array of items of type 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
1 using System.Collections.Generic;
2 using System;
3
4 namespace GenericClass
5 {
6 class EntryPoint
7 {
8 static void Main()
9 {
10 MyList<string> stringList;
11 MyList<int> intList;
12 MyList<bool> boolList;
13 }
14
15 class MyList<T>
16 {
17 private T[] items;
18 private int count;
19 private int capacity;
20
21 public int Count { get; private set; }
22 public int Capacity { get; private set; }
23
24 public MyList()
25 {
26 this.items = new T[2];
27 this.capacity = 2;
28 this.count = 0;
29 }
30
31 }
32
33 }
34 }
35
36
37
38
39
It has a property count that contains the number of elements in the list.
So we're going to make it private because no one needs to get access to that array.
We also have capacity for our list.
And we also need the constructor.
So what we're going to have in these constructor.
We need to initialize our array, property Count and Capacity with the initial values.

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()

```
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;
            }
        }
```

C:\Windows\system32\cmd.exe

```
2
0
Aby kontynuować, naciśnij dowolny klawisz . . .
```

Our Count property is simply number of elements in our items array.
And Capacity is the size in memory of our items array.

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()

```
{
    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++;
            }
        }
    }
```

Now we can write Add method to add elements of the type T.
And this method needs to take generic argument T.
Basically it takes the same type as the type of the item array.
Property Count shows the first empty place in items array (after last element).

C:\Windows\system32\cmd.exe

```
Wyjątek nieobsłużony: System.IndexOutOfRangeException: Ind
nice tablicy.
   w GenericClass.EntryPoint.MyList`1.Add(T item) w C:\WAL
e c#\wyklad 4 - generic\04. Generics\02. GenericClass\Entr
   w GenericClass.EntryPoint.Main() w C:\WALDEK\praca\prog
- generic\04. Generics\02. GenericClass\EntryPoint.cs:wier
```

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

When we place the new element – we should increment the Count property.
But its not all.
We must resize the array because we will have IndexOutOfRange exception.

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

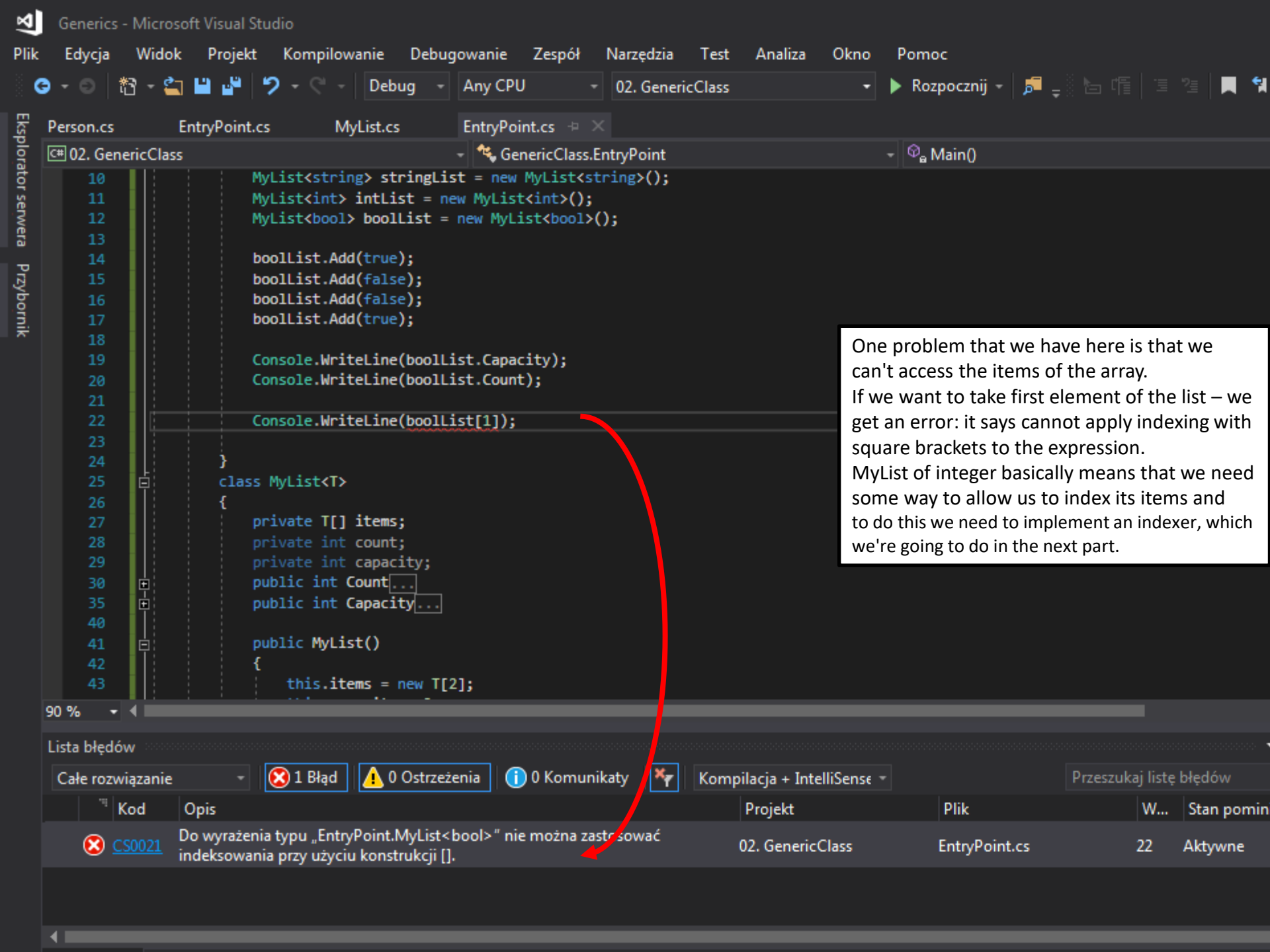
02. GenericClass | GenericClass.EntryPoint | Main()

```
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      }
```

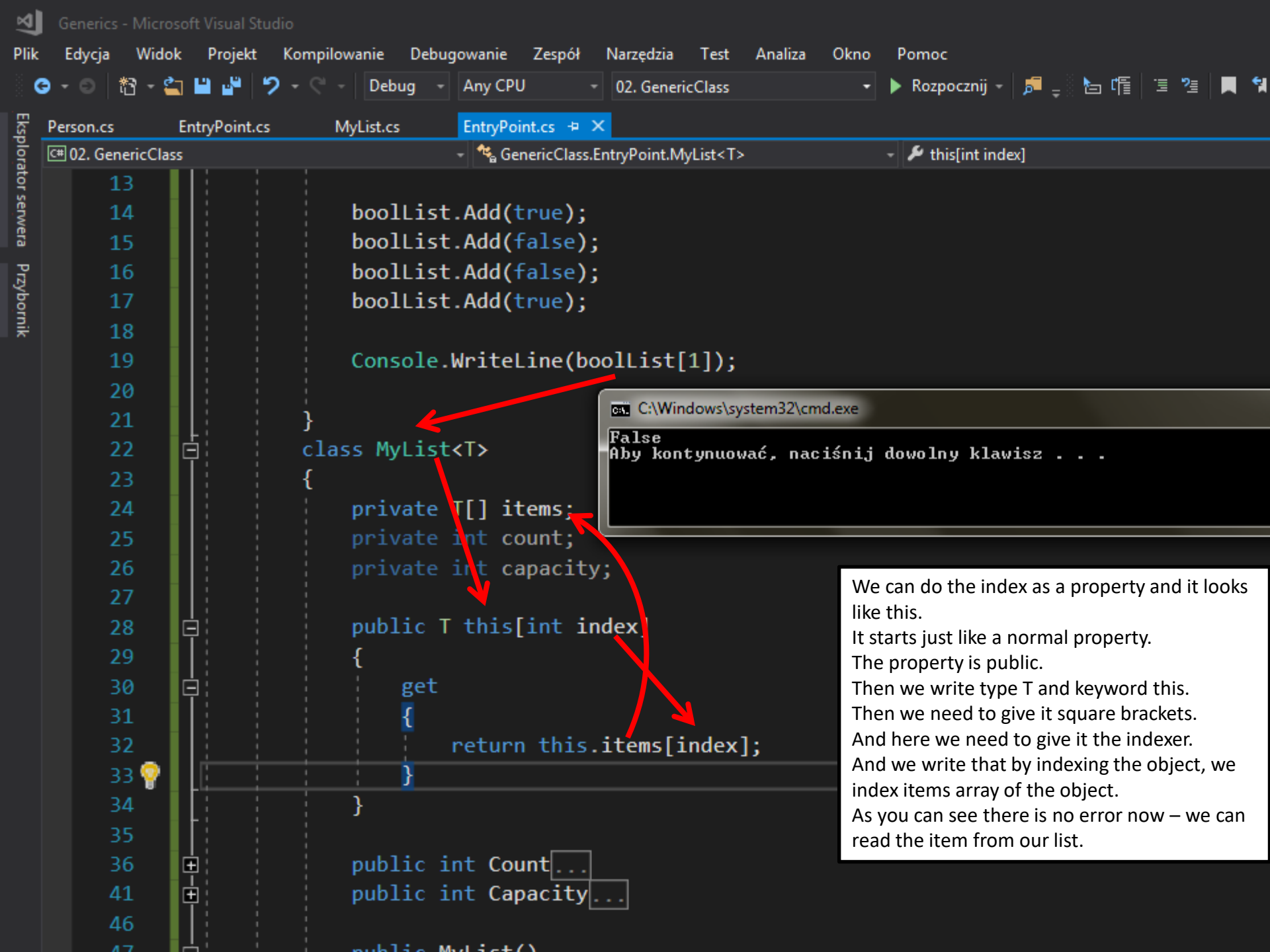
C:\Windows\system32\c

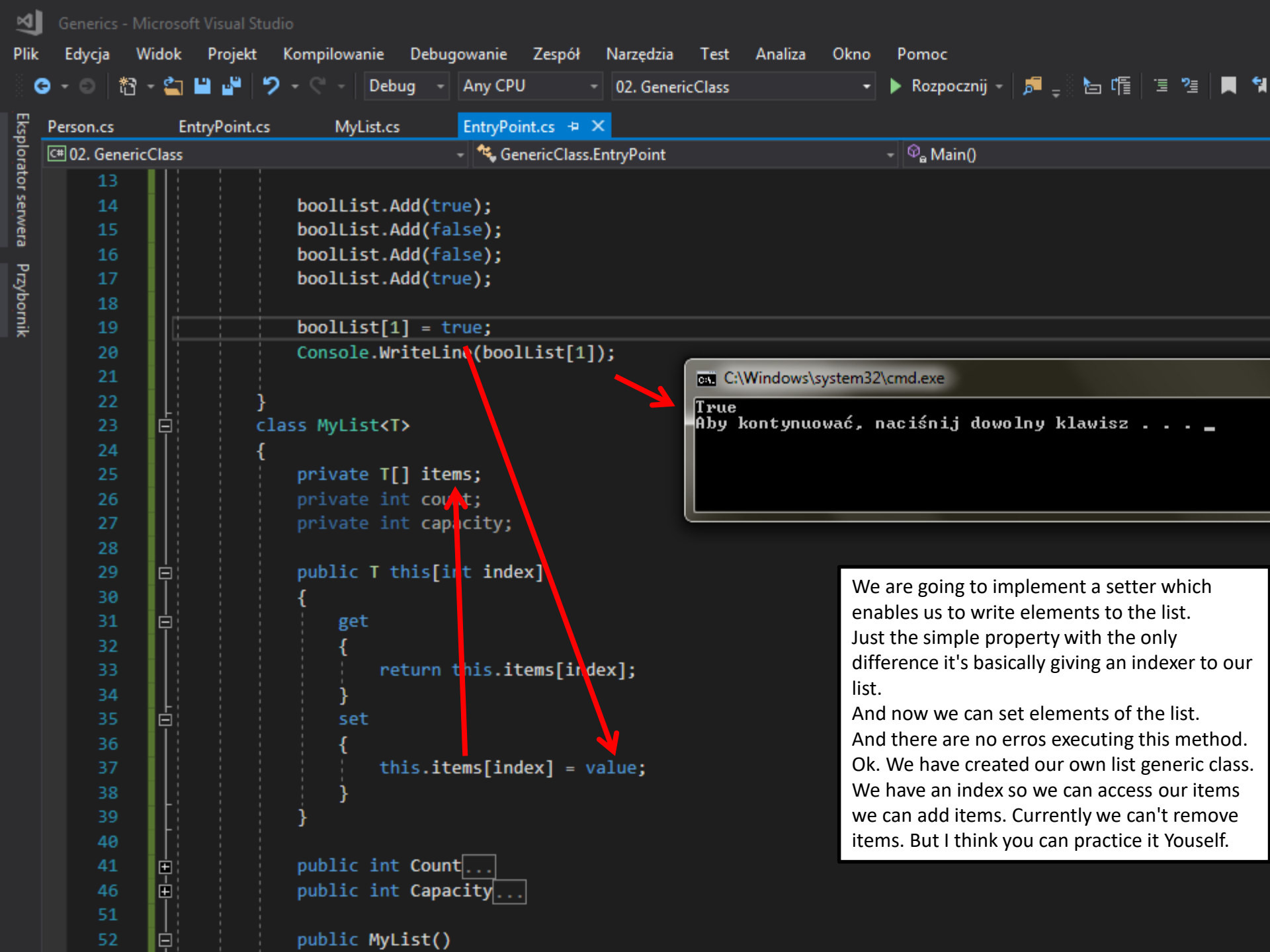
```
4
4
Dby kontynuować, na
```

Of course before we add anything to the array we must perform a check if the array has a space because we're initializing the array with 2 places for items only. We can do it in the following way:
So if the items are equal to the capacity we are going to multiply it by 2.
We are going to create a new array which is going to clone our current array.
Then we're going to double the capacity.
So this top capacity multiply by two and we are going to create new greater array and copy previous array to the new biger array.
Now we add some elements.
And we check what is the capacity and what is that count.
As you see it is working.
We have a capacity of four and count of four. And it is correct.


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()
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];
One problem that we have here is that we can't access the items of the array.
If we want to take first element of the list – we get an error: it says cannot apply indexing with square brackets to the expression.
MyList of integer basically means that we need some way to allow us to index its items and to do this we need to implement an indexer, which we're going to do in the next part.
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

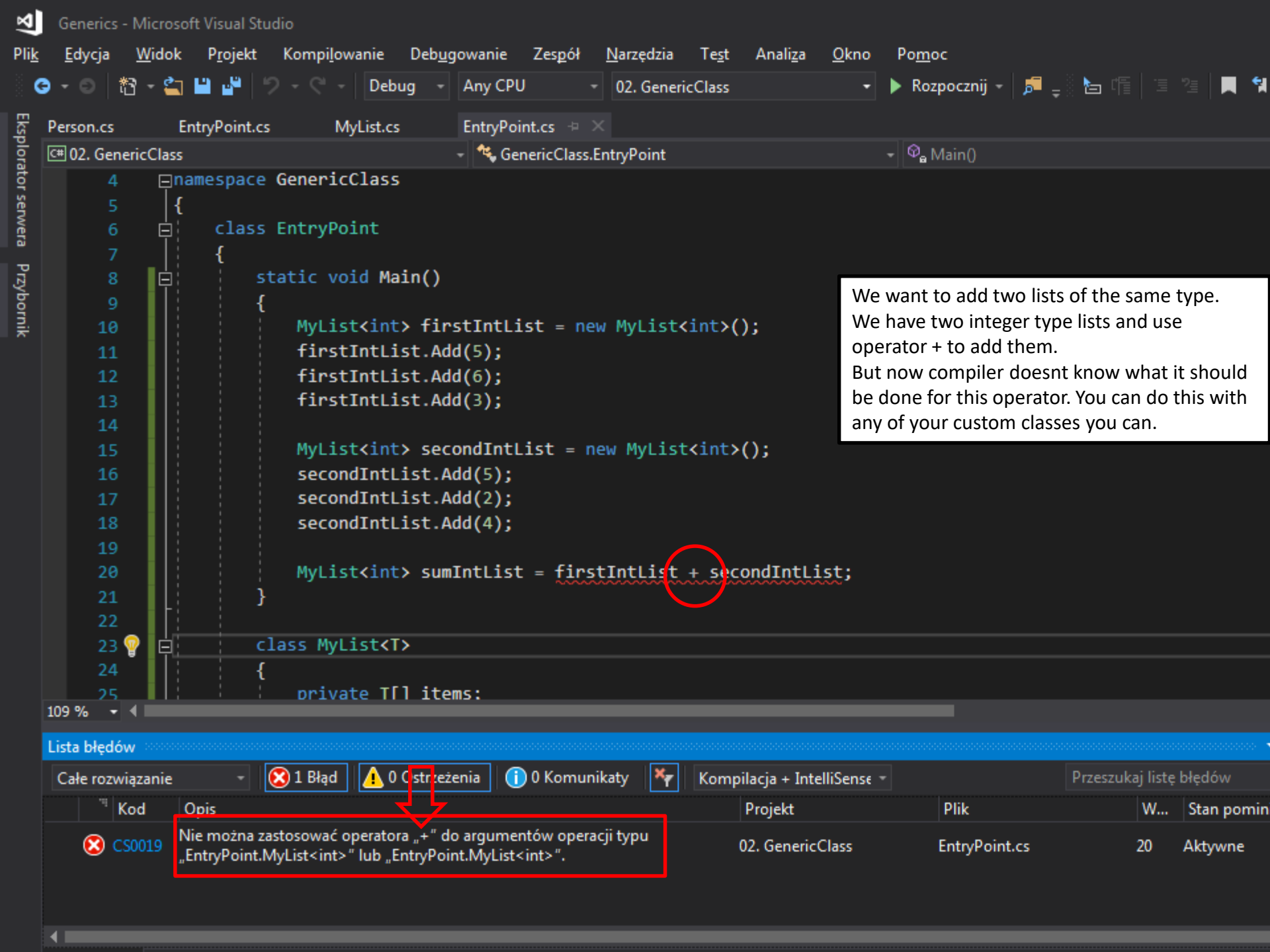
Generic classes - indexers

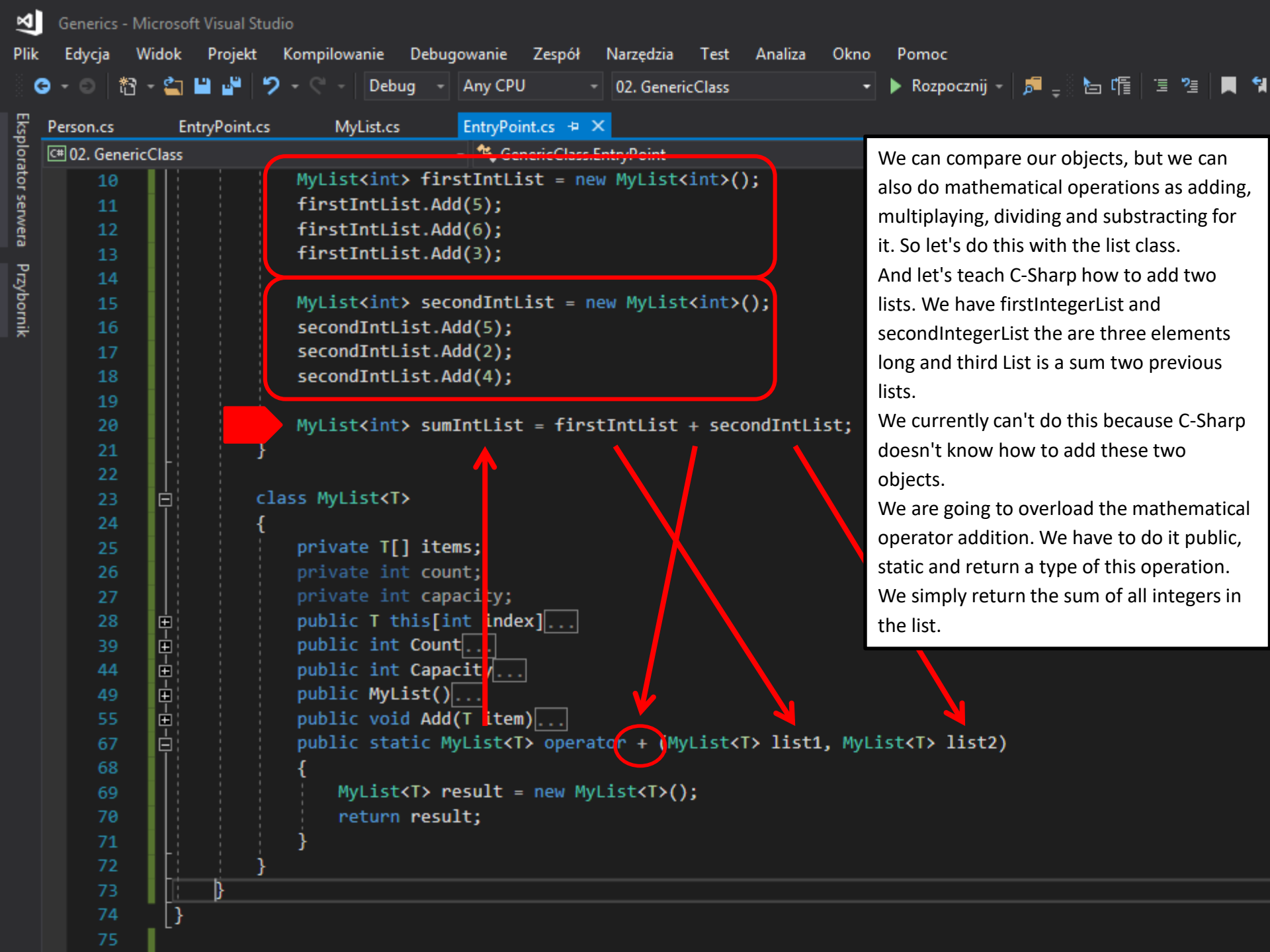

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint.MyList<T>
this[int index]
Eksplorator serwera
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 . . .
We can do the index as a property and it looks like this.
It starts just like a normal property.
The property is public.
Then we write type T and keyword this.
Then we need to give it square brackets.
And here we need to give it the indexer.
And we write that by indexing the object, we index items array of the object.
As you can see there is no error now – we can read the item from our list.

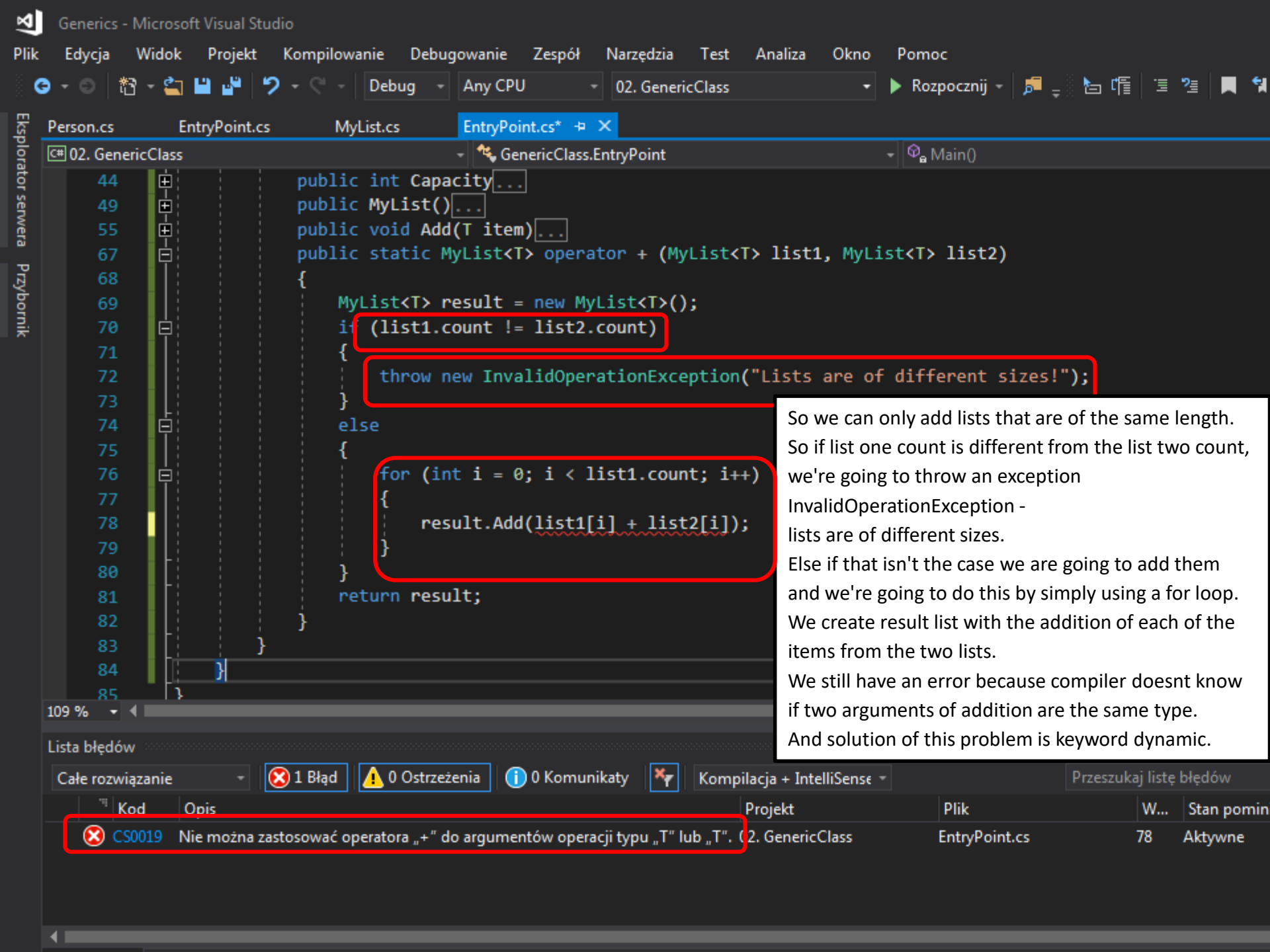

Generics - Microsoft Visual Studio
Plik Edycja Widok Projekt Kompilowanie Debugowanie Zespół Narzędzia Test Analiza Okno Pomoc
Debug
Any CPU
02. GenericClass
Rozpocznij
Person.cs
EntryPoint.cs
MyList.cs
EntryPoint.cs
02. GenericClass
GenericClass.EntryPoint
Main()
boolList.Add(true);
boolList.Add(false);
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 . . .
We are going to implement a setter which enables us to write elements to the list.
Just the simple property with the only difference it's basically giving an indexer to our list.
And now we can set elements of the list.
And there are no erros executing this method.
Ok. We have created our own list generic class.
We have an index so we can access our items we can add items. Currently we can't remove items. But I think you can practice it Youself.

Overloading mathematical operators

Now we are going to overload mathematical operator.


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
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;
We want to add two lists of the same type.
We have two integer type lists and use operator + to add them.
But now compiler doesnt know what it should be done for this operator. You can do this with any of your custom classes you can.
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
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;
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>();
return result;
We can compare our objects, but we can also do mathematical operations as adding, multiplaying, dividing and substracting for it. So let's do this with the list class.
And let's teach C-Sharp how to add two lists. We have firstIntegerList and secondIntegerList the are three elements long and third List is a sum two previous lists.
We currently can't do this because C-Sharp doesn't know how to add these two objects.
We are going to overload the mathematical operator addition. We have to do it public, static and return a type of this operation.
We simply return the sum of all integers in the list.


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()
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>();
if (list1.count != list2.count)
{
throw new InvalidOperationException("Lists are of different sizes!");
}
else
{
for (int i = 0; i < list1.count; i++)
{
result.Add(list1[i] + list2[i]);
}
}
return result;
}
}
}
109 %
So we can only add lists that are of the same length.
So if list one count is different from the list two count, we're going to throw an exception InvalidOperationException -
lists are of different sizes.
Else if that isn't the case we are going to add them and we're going to do this by simply using a for loop.
We create result list with the addition of each of the items from the two lists.
We still have an error because compiler doesnt know if two arguments of addition are the same type.
And solution of this problem is keyword dynamic.
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 „T" lub „T".
2. GenericClass
EntryPoint.cs
78
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

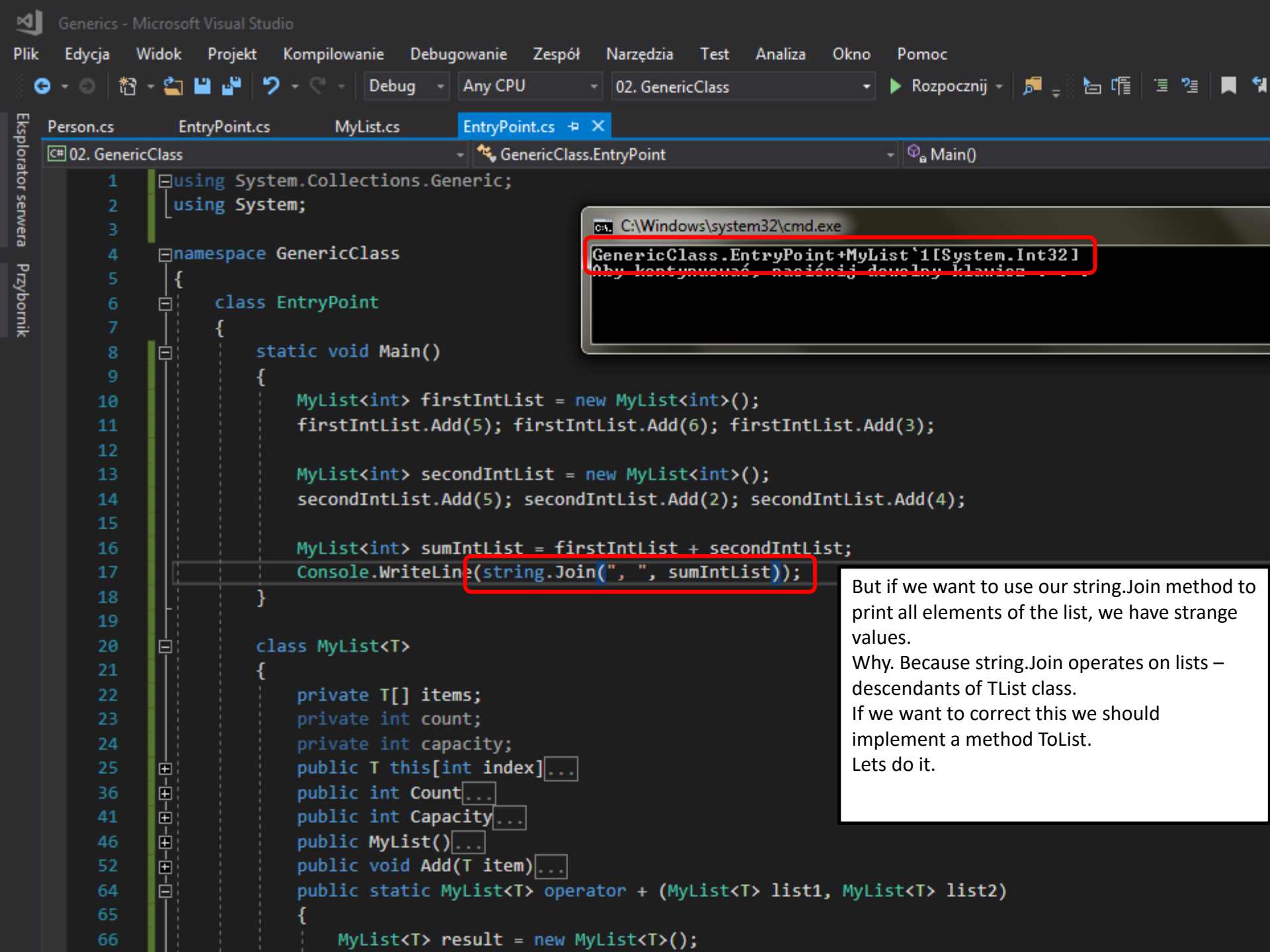
02. GenericClass | GenericClass.EntryPoint | Main()

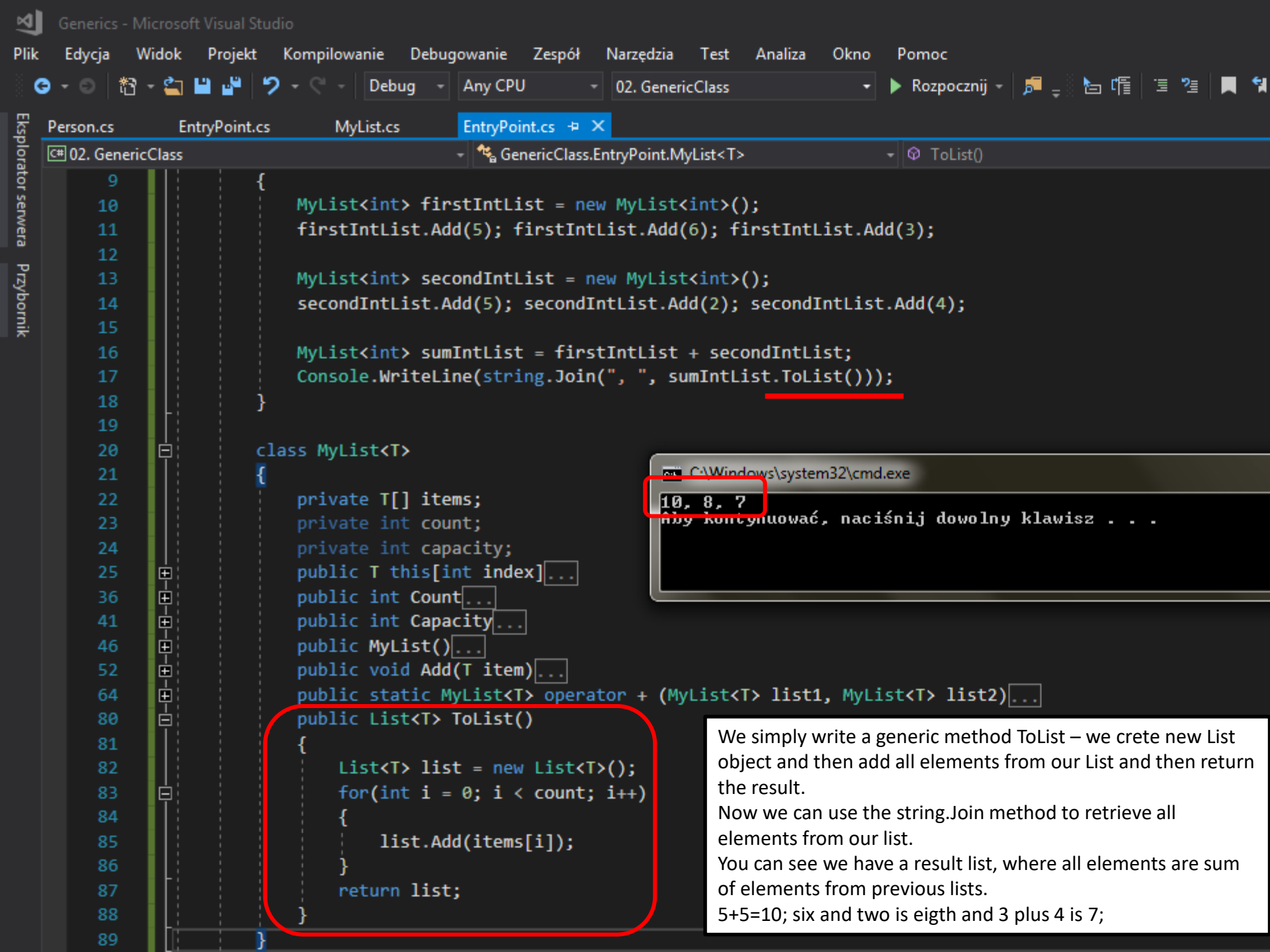
```
11          firstIntList.Add(5); firstIntList.Add(6); firstIntList.Add(3);
12
13          MyList<int> secondIntList = new MyList<int>();
14          secondIntList.Add(5); secondIntList.Add(2); secondIntList.Add(4);
15
16          MyList<int> sumIntList = firstIntList + secondIntList;
17          Console.WriteLine($"sumIntList: {sumIntList[0]}, {sumIntList[1]}, {sumIntList[2]}");
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)
65          {
66              MyList<T> result = new MyList<T>();
67              if (list1.count != list2.count)
68              {
69                  throw new InvalidOperationException("Lists are of different sizes!");
70              }
71              else
72              {
73                  for (int i = 0; i < list1.count; i++)
74                  {
75                      result.Add((dynamic)list1[i] + (dynamic)list2[i]);
76                  }
```

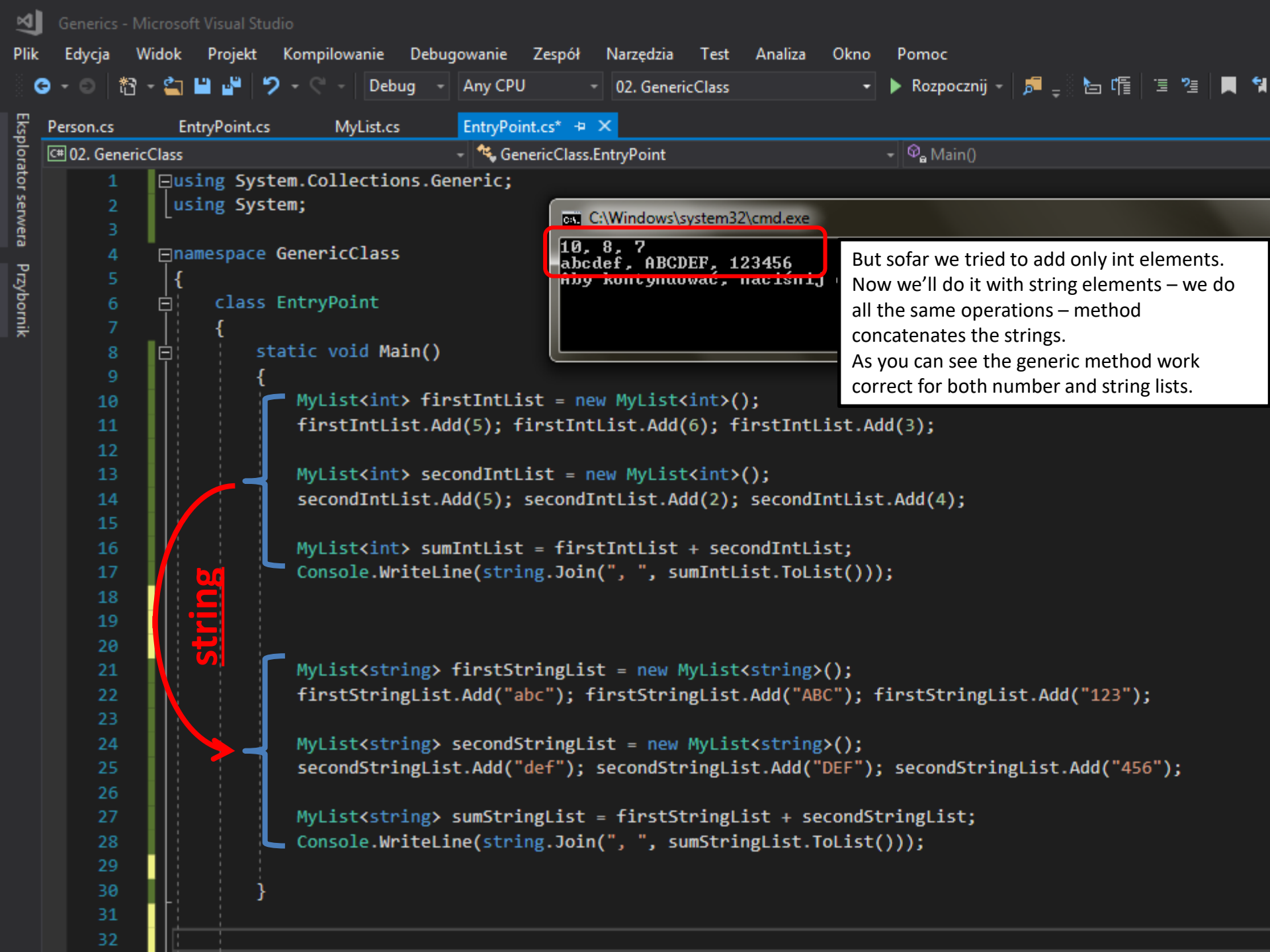
C:\Windows\system32\cmd.exe

```
sumIntList: 10, 8, 7
Aby kontynuować, naciśnij dowolny klawisz . . .
```

As you can see now it is no error because compiler doesn't check the type of operands. It will be checked during executing.
We can now print some elements of the 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()
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]
But if we want to use our string.Join method to print all elements of the list, we have strange values.
Why. Because string.Join operates on lists – descendants of TList class.
If we want to correct this we should implement a method ToList.
Lets do it.


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
{
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()));
}
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)...
public List<T> ToList()
{
List<T> list = new List<T>();
for(int i = 0; i < count; i++)
{
list.Add(items[i]);
}
return list;
}
}
C:\Windows\system32\cmd.exe
10, 8, 7
Aby kontynuować, naciśnij dowolny klawisz . . .
We simply write a generic method ToList – we crete new List object and then add all elements from our List and then return the result.
Now we can use the string.Join method to retrieve all elements from our list.
You can see we have a result list, where all elements are sum of elements from previous lists.
5+5=10; six and two is eigth and 3 plus 4 is 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
C:\Windows\system32\cmd.exe
10, 8, 7
abcdef, ABCDEF, 123456
But sofar we tried to add only int elements. Now we'll do it with string elements – we do all the same operations – method concatenates the strings.
As you can see the generic method work correct for both number and string lists.
string
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()));
}