<out T> vs <T> in Generics

153

46

What is the difference between `<out T>` and `<T>`? For example:

```
public interface IExample<out T>
{
    ...
}
```

vs.

```
public interface IExample<T>
{
    ...
}
```

c# generics covariance

edited Apr 18 '18 at 23:43

asked Jun 8 '12 at 23:05
Cole Johnson
5,459 13 40 61

1 Good example would be IObservable<T> and IObserver<T>, defined in system ns in mscorlib. public interface IObservable<out T>, and public interface IObserver<in T>. Similarly, IEnumerator<out T>, IEnumerable<out T> – VivekDev Feb 6 '16 at 1:00

5 Answers

The `out` keyword in generics is used to denote that the type T in the interface is covariant. See Covariance and contravariance for

✓

```
IEnumerable<string> strings = new List<string>();
IEnumerable<object> objects = strings;
```

The second line above would fail if this wasn't covariant, even though logically it should work, since string derives from object. Before variance in generic interfaces was added to C# and VB.NET (in .NET 4 with VS 2010), this was a compile time error.

After .NET 4, `IEnumerable<T>` was marked covariant, and became `IEnumerable<out T>` . Since `IEnumerable<out T>` only uses the elements within it, and never adds/changes them, it's safe for it to treat an enumerable collection of strings as an enumerable collection of objects, which means it's *covariant*.

This wouldn't work with a type like `IList<T>` , since `IList<T>` has an `Add` method. Suppose this would be allowed:

```
IList<string> strings = new List<string>();
IList<object> objects = strings;  // NOTE: Fails at compile time
```

You could then call:

```
objects.Add(new Image()); // This should work, since IList<object> should let us add
**any** object
```

This would, of course, fail - so `IList<T>` can't be marked covariant.

There is also, btw, an option for `in` - which is used by things like comparison interfaces. `IComparer<in T>` , for example, works the opposite way. You can use a concrete `IComparer<Foo>` directly as an `IComparer<Bar>` if `Bar` is a subclass of `Foo` , because the `IComparer<in T>` interface is *contravariant*.

edited Sep 9 '16 at 10:53
Aurélien Gasser ♦
2,527 1 15 22

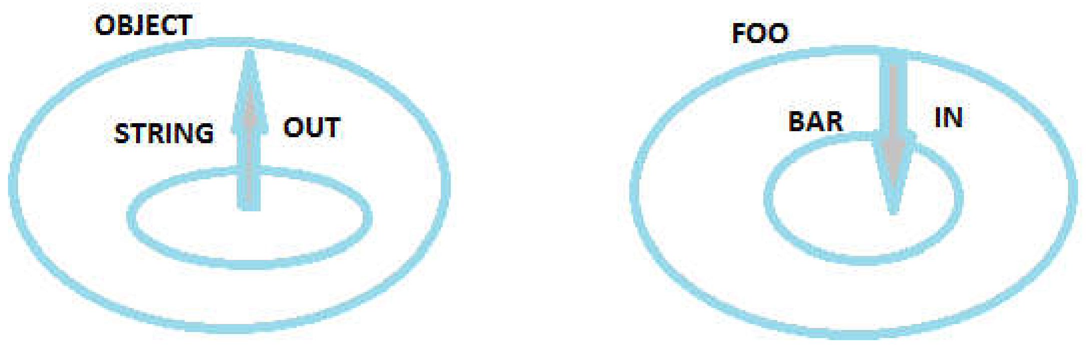
answered Jun 8 '12 at 23:11
Reed Copsey
478k 60 1002 1286

4 @ColeJohnson Because `Image` is an abstract class ;) You can do `new List<object>() { Image.FromFile("test.jpg") };` with no problems, or you can do `new List<object>() { new Bitmap("test.jpg") };` as well. The problem with yours is that `new Image()` isn't allowed (you can't do `var img = new Image();` either) – Reed Copsey Aug 20 '12 at 16:28

50

For remembering easily the usage of `in` and `out` keyword (also covariance and contravariance), we can image inheritance as wrapping:

```
String : Object
Bar : Foo
```



edited Aug 6 '17 at 4:14
shA.t
13.4k 4 39 75

answered Jan 9 '15 at 9:39
o0omycomputero0o
1,485 1 19 29

1 This makes it so clear. – antiduh Aug 6 '17 at 4:18

10 Isn't this the wrong way around? Contravariance = in = allows less derived types to be used in place of more derived. / Covariance = out = allows more derived types to be used in place of less derived. Personally, looking at your diagram, I read it as the opposite of the that. – Sam Shiles Aug 24 '17 at 7:19

consider,

```
interface ISkinned<T> {}
```

and the functions,

```
void Peel(ISkinned<Fruit> skinned) { }

void Peel(ICovariantSkinned<Fruit> skinned) { }
```

The function that accepts `ICovariantSkinned<Fruit>` will be able to accept `ICovariantSkinned<Fruit>` or `ICovariantSkinned<Bananna>` because `ICovariantSkinned<T>` is a covariant interface and `Banana` is a type of `Fruit` ,

the function that accepts `ISkinned<Fruit>` will only be able to accept `ISkinned<Fruit>` .

edited Sep 26 '16 at 12:46

answered Dec 18 '13 at 14:34
Jodrell
27.3k 3 59 104

28

" `out T` " means that type `T` is "covariant". That restricts `T` to appear only as a returned (outbound) value in methods of the generic class, interface or method. The implication is that you can cast the type/interface/method to an equivalent with a super-type of `T` . E.g. `ICovariant<out Dog>` can be cast to `ICovariant<Animal>` .

edited Jan 6 '16 at 11:18
shA.t
13.4k 4 39 75

answered Jun 8 '12 at 23:16
James World
23.4k 5 70 101

5 I didn't realize that `out` enforces that `T` can be returned only, until I read this answer. The whole concept makes more sense now! – MarioDS Sep 21 '15 at 10:09

From the link you posted....

> For more information, see Covariance and Contravariance (C# and Visual Basic). http://msdn.microsoft.com/en-us/library/ee207183.aspx

answered Jun 8 '12 at 23:11

Brad Cunningham
5,270 1 28 37