Is this C# extension method impure and if so, bad code?

Asked 15 years, 9 months ago Modified 11 years, 8 months ago Viewed 2k times

4

I'm learning a bit about function programming, and I'm wondering:

1) If my `ForEach` extension method is pure? The way I'm calling it seems violate the "don't mess with the object getting passed in", right?

```
public static void ForEach<T>(this IEnumerable<T> source, Action<T> action)
{
  foreach ( var item in source )
     action(item);
}


static void Main(string[] args)
{
    List<Cat> cats = new List<Cat>()
    {
        new Cat{ Purring=true,Name="Marcus",Age=10},
        new Cat{ Purring=false, Name="Fuzzbucket",Age=25 },
        new Cat{ Purring=false, Name="Beanhead",Age=9 },
        new Cat{Purring=true,Name="Doofus",Age=3}
    };


    cats.Where(x=>x.Purring==true).ForEach(x =>
    {
        Console.WriteLine("{0} is a purring cat... purr!", x.Name);
    });

    // *************************************************
    //  Does this code make the extension method impure?
    // *************************************************
    cats.Where(x => x.Purring == false).ForEach(x =>
    {
        x.Purring = true; // purr,baby
    });

    // all the cats now purr
    cats.Where(x=>x.Purring==true).ForEach(x =>
    {
        Console.WriteLine("{0} is a purring cat... purr!", x.Name);
    });
}

public class Cat {
        public bool Purring;
        public string Name;
        public int Age;
}
```

2) If it is impure, is it bad code? I personally think it makes cleaner looking code than the old `foreach ( var item in items) { blah; }`, but I worry that since it might be impure, it could make a mess.

3) Would it be bad code if it returned `IEnumerable<T>` instead of `void`? I'd say as long as it is impure, yes it would be very bad code as it would encourage chaining something that would modify the chain. For example, is this bad code?

```
// possibly bad extension
public static IEnumerable<T> ForEach<T>(this IEnumerable<T> source, Action<T>
action)
{
    foreach ( var item in source )
        action(item);

    return source;

}
```

c# functional-programming extension-methods

Share

Improve this question

Follow

edited Apr 15, 2013 at 3:11

community wiki
5 revs, 2 users 100%
Cory R. King

4 Answers

Sorted by: Highest score (default)

12

Impurity doesn't necesarily mean bad code. Many people find it easy and useful to use side effects to solve a problem. The key is first knowing how to do it in a pure way, so you'll know when impurity is appropriate :).

.NET doesn't have the concept of purity in the type system, so a "pure" method that takes in arbitrary delegates can always be impure, depending on how it's called. For instance, "Where", aka "filter", would usually be considered a pure function, since it doesn't modify its arguments or modify global state.

But, there's nothing stopping you from putting such code inside the argument to Where. For example:

```
things.Where(x => { Console.WriteLine("um?");
                    return true; })
      .Count();
```

So that's definately an impure usage of Where. Enumerables can do whatever they want as they iterate.

Is your code bad? No. Using a foreach loop is just as "impure" -- you're still modifying the source objects. I write code like that all the time. Chain together some selects, filters, etc., then execute a ForEach on it to invoke some work. You're right, it's cleaner and easier.

Example: ObservableCollection. It has no AddRange method for some reason. So, if I want to add a bunch of things to it, what do I do?

```
foreach(var x in things.Where(y => y.Foo > 0)) { collection.Add(x)); }
```

or

```
things.Where(x => x.Foo > 0).ForEach(collection.Add);
```

I prefer the second one. At a minimum, I don't see how it can be construed as being worse than the first way.

When is it bad code? When it does side effecting code in a place that's not expected. This is the case for my first example using Where. And even then, there are times when the scope is very limited and the usage is clear.

Chaining ForEach

I've written code that does things like that. To avoid confusion, I would give it another name. The main confusion is "is this immediately evaluated or lazy?". ForEach implies that it'll go execute a loop right away. But something returning an IEnumerable implies that the items will be processed as needed. So I'd suggest giving it another name ("Process", "ModifySeq", "OnEach"... something like that), and making it lazy:

```
public static IEnumerable<T> OnEach(this IEnumerable<T> src, Action<T> f) {
  foreach(var x in src) {
    f(x);
    yield return x;
  }
}
```

Share Improve this answer Follow

answered Mar 28, 2009 at 23:33

MichaelGG

10k ● 1 ● 41 ● 82

6

It is not pure, as it can call impure methods. I think by typical definitions, purity is a transitive closure - a function is pure only if all the functions it calls (directly or indirectly) are also pure, or if the effects of those functions are encapsulated (e.g. they only mutate a non-escaping local variable).

Share Improve this answer Follow

answered Mar 28, 2009 at 22:19
Brian
119k ● 17 ● 243 ● 304

Basically, any public .NET method that calls a function argument is impure, becuase it could call impure code? – MichaelGG Mar 28, 2009 at 23:18

One can imagine a type system where you can be 'generic' with respect to 'purity', and so G(Func<T> f) could have G be pure-when-f-is-pure and impure-when-f-is-impure. – Brian Mar 29, 2009 at 0:26

But until we have it in the type system, is it not useful to be able to distinguish HoF that are pure if their params aren't versus ones that simply aren't? – MichaelGG Mar 29, 2009 at 20:53

I'm not sure what you're asking... "until we have it in the type system", there isn't even a 'Pure' attribute yet, much less a type system for it; we're already discussing futuristic fantasies, so nothing is 'useful' per se, this is all armchair-language-design-speculation. – Brian Mar 29, 2009 at 21:32

In any case, I just upvoted your answer, as it's better than mine and more on-topic. The higher-order bit is, rather than focus on definitions and type systems, write good code that minimizes surprising/unexpected effects and interactions. – Brian Mar 29, 2009 at 21:40

4

Yes, it's not pure, but that's kind of a moot point as it's not even a function.

As the method doesn't return anything, the only option for it to do anything at all is to either affect the objects that you are sending in, or affecting something unrelated (like writing to the console window).

Edit:
To answer your third question; yes, that is bad code, as it seems to be doing something that it doesn't. The method returns a collection so it seems to be pure, but as it just returns the collection that was sent in, it's actually not any more pure than the first version. To make any sense the method should take a `Func<T,T>` delegate to use as conversion, and return a collection of the converted items:

```
public static IEnumerable<T> ForEach<T>(this IEnumerable<T> source, Func<T,T>
converter) {
   foreach (T item in source) {
      yield return converter(item);
   }
}
```

It's of course still up to the converter function if the extension call is pure. If it doesn't make a copy of the input item but just changes it and returns it, the call is still not pure.

Share

Improve this answer

Follow

edited Mar 28, 2009 at 22:40

answered Mar 28, 2009 at 22:24

Guffa
700k ● 110 ● 750 ● 1k

@Noldorin: No, that's not what I mean. It doesn't have a return value, so it's not a function.
– Guffa Mar 29, 2009 at 2:04

3

Indeed, because your lambda expression contains an assignment, the function is now by definition impure. Whether the assignment is related to one of the arguments or another object defined outside the current function is irrelevant... A function must have no side-effects whatsoever in order to be called *pure*. See Wikipedia for a more precise (though quite straightforward) definition, which details the two conditions a function must satisfy to be deemed *pure* (having no side-effects is one of them). I believe lambda expressions are typically meant to be used as pure functions (at least I would imagine they were originally studied as such from the mathematical perspective), though clearly C# isn't stringent about this, where purely functional languages are. So it's probably not bad pratice, though it's definitely worth being away that such a function is impure.

Share

Improve this answer

Follow

edited Mar 28, 2009 at 22:41

Judah Gabriel Himango
60k ● 39 ● 161 ● 215

answered Mar 28, 2009 at 22:20

Noldorin
147k ● 56 ● 272 ● 307