

Forum:[Hack Forums](#)

Thread: [Easier form of generic type constraints?](#)

severos - June 19, 2017, 8:14 am

I'm writing a library for my personal testing, currently I'm writing the interfaces, and I've ended up with some pretty abstracted stuff like this:

PHP Code

```
<?php
```

```
//nice interface
```

```
public interface IUser<TKey>
{
    TKey ID { get; }
}
```

```
//things are getting a bit annoying in checks
```

```
public interface IUserManager<TUserKey, TUser>
    where TUser:IUser<TUserKey>
{
    TUser FindUser(TUserKey key);
}
```

```
//This is starting to get ugly and annoying doing it second time
```

```
public interface ISession<TUserKey, TUser, TSessionKey>
    where TUser : IUser<TUserKey>
{
    TSessionKey Key { get; }
    TUser User { get; }
}
```

```
//Now this is spaghetti interface....
```

```
internal interface ISessionManager<TUserKey, TUser, TSessionKey, TSession, IUserManager>
    where TUser :IUser<TUserKey>
    where TSession : ISession<TUserKey,TUser,TSessionKey>
    where IUserManager : IUserManager<TUserKey,TUser>
{

    TSession TryGetSession(TSessionKey keyy);

    TSession AddSession(TUser user);
}
```

So I was wondering, isn't there some way to make generics a bit easier? like for example the ISession, can't I just tell it to take whatever thing implementing whatever generic of IUser since I have no interest in their key type? Same for ISessionManager.

I'm pretty sure by the time I get this library done and start using it I'll be cursing my self for having to type like 6~7 generic types...

Thanks in advance.

Ninja - June 19, 2017, 8:30 am

My English isn't the best but Ill try to explain.

You don't always have to set the type on the class, you can set the type on the method as well.

```
Code public interface Whatever
{
    T DoStuff<T>();
}
```

severos - June 19, 2017, 8:55 am

(Jun 19, 2017 08:30 AM)Ninja Wrote: [\[CITE\]](#) My English isn't the best but Ill try to explain.

You don't always have to set the type on the class, you can set the type on the method as well.

```
Code public interface Whatever
{
    T DoStuff<T>();
}
```

I can't really see how that would help in this case, actually it might get things worse since I'll have to write types for each method and write the constraints for them as well... hope I didn't miss the point you were making there.

Ninja - June 19, 2017, 9:04 am

(Jun 19, 2017 08:55 AM)severos Wrote: [\[CITE\]](#) I can't really see how that would help in this case, actually it might get things worse since I'll have to write types for each method and write the constraints for them as well... hope I didn't miss the point you were making there.

That is not necessarily true. And it means that you don't have to instantiate a new class just because you input a different type. I used this when I was working with mapping objects with custom attributes.

After looking at it again, do you have to have interfaces for this? Can we have some background info to know what you are trying to achieve? Maybe an abstract class would work here.

severos - June 19, 2017, 10:51 am

(Jun 19, 2017 09:04 AM)Ninja Wrote: [\[CITE\]](#) That is not necessarily true. And it means that you don't have to instantiate a new class just because you input a different type. I used this when I was working with mapping objects with custom attributes.

After looking at it again, do you have to have interfaces for this? Can we have some background info to know what you are trying to achieve? Maybe an abstract class would work here.

The part I've posted up there is some custom auth & session, I'm interfacing it to allow using different implementations later without having to change a thing since Dependency Injection sees interfaces only, so whenever I pull a user schema and want to auth using it, I'll just implement the interface in it. same goes with different session managers, and different DB engines.

Ninja - June 19, 2017, 10:53 am

(Jun 19, 2017 10:51 AM)severos Wrote: [\[CITE\]](#) The part I've posted up there is some custom auth & session, I'm interfacing it to allow using different implementations later without having to change a thing since Dependency Injection sees interfaces only, so whenever I pull a user schema and want to auth using it, I'll just implement the interface in it. same goes with different session managers, and different DB engines.

You can make them abstract classes, so you can override the methods and stuff.

Also, assuming UserManager interacts with the database, you should use a pattern like the repository pattern, or an ORM like Entity Framework. This will make life a lot easier.

severos - June 19, 2017, 11:51 am

(Jun 19, 2017 10:53 AM)Ninja Wrote: [\[CITE\]](#) You can make them abstract classes, so you can override the methods and stuff.

Also, assuming UserManager interacts with the database, you should use a pattern like the repository pattern, or an ORM like Entity Framework. This will make life a lot easier.

I'll think about the abstract classes, but kinda feel like sticking with interfaces, as they give more options, specially if going to use multiple languages. As for the database part, I really like Entity Framework, and ORM in general as they certainly make life easier, however it's hard to generalize them cause I deal with various storage stuff like SQL, MongoDB, raw cloud storage, and some custom DBs for companies that aren't really standard.

Ninja - June 19, 2017, 2:29 pm

(Jun 19, 2017 11:51 AM)severos Wrote: [\[CITE\]](#) I'll think about the abstract classes, but kinda feel like sticking with interfaces, as they give more options, specially if going to use multiple languages. As for the database part, I really like Entity Framework, and ORM in general as they certainly make life easier, however it's hard to generalize them cause I deal with various storage stuff like SQL, MongoDB, raw cloud storage, and some custom DBs for companies that aren't really standard. Custom DBs? I am interested. Mind linking me some if you can?

And, I recommend using abstract classes for this (though I barely have an idea of what you are doing). Or a combination of both.

idb. - June 19, 2017, 3:08 pm

(Jun 19, 2017 02:29 PM)Ninja Wrote: [\[CITE\]](#) And, I recommend using abstract classes for this (though I barely have an idea of what you are doing). Or a combination of both. Why are you making recommendations when you barely have an idea of what is going on though? lol. In these instances is best to not say anything at all than to make suggestions that may not be the best solution for the issue.

Ninja - June 19, 2017, 3:47 pm

(Jun 19, 2017 03:08 PM)idb. Wrote: [\[CITE\]](#) Why are you making recommendations when you barely have an idea of what is going on though? lol. In these instances is best to not say anything at all than to make suggestions that may not be the best solution for the issue.

Why can't I? Maybe abstract classes is what he needs? I even told him I am not sure based on the information he provided.

BahNahNah - June 19, 2017, 3:56 pm

Try and use the generics that are there insted of adding new ones, like

Code public interface ISession<TUserKey, TUser, TSessionKey>

where TUser : IUser<TUserKey>

```
{
    TSessionKey Key { get; }
    TUser User { get; }
```

}can be

Code public interface ISession<TUserKey, TSessionKey> {

```
    TSessionKey Key { get; }
    IUser<TUserKey> User { get; }
}
```

this is what i got

Spoiler [\(Click to View\)](#)

severos - June 20, 2017, 12:30 am

(Jun 19, 2017 02:29 PM)Ninja Wrote: [\[CITE\]](#) Custom DBs? I am interested. Mind linking me some if you can?

And, I recommend using abstract classes for this (though I barely have an idea of what you are doing). Or a combination of both. Afraid I can't pass you links, they're not public, I deal with them when I do stuff for local companies around. And for the abstract class part, it'll end up kinda like the interface, just allows to add fields and implementations...

(Jun 19, 2017 03:56 PM)BahNahNah Wrote: [\[CITE\]](#) Try and use the generics that are there insted of adding new ones, like

```
Code public interface ISession<TUserKey, TUser, TSessionKey>
where TUser : IUser<TUserKey>
```

```
{
    TSessionKey Key { get; }
    TUser User { get; }
```

}can be

```
Code public interface ISession<TUserKey, TSessionKey> {
    TSessionKey Key { get; }
    IUser<TUserKey> User { get; }
}
```

this is what i got

Spoiler [\(Click to View\)](#)

Sounds like a nice suggestion

Rageline - June 20, 2017, 1:52 am

There are few undocumented C# keywords that you can use for generics ;)

severos - June 20, 2017, 3:34 am

(Jun 20, 2017 01:52 AM)Rageline Wrote: [\[CITE\]](#) There are few undocumented C# keywords that you can use for generics ;)

Did a fast search on that, found `__arglist`, `__refvalue`, `__reftype` but have no idea how they're related to my case here...

Ninja - June 20, 2017, 9:29 am

(Jun 20, 2017 01:52 AM)Rageline Wrote: [\[CITE\]](#) There are few undocumented C# keywords that you can use for generics ;)

Ah, some more knowledge spurted out by Rageline aka the grasshopper. I am still searching for the elusive grasshopper, and now you are assigning me the quest of finding the undocumented C# keywords you so vaguely described.

EDIT: Does it have to do with typecasting and using the default keyword? That is all Google gave me. Or did you make stuff up again?

See, you come in this section saying you know things no one knows, yet you never, ever, elaborate on them. And when asked what the things are or for sources you usually plain out ignore the questions.

EDIT 2:

By grasshopper algorithm do you maybe mean Boyer-Moore algorithm?

(Jun 20, 2017 03:34 AM)severos Wrote: [\[CITE\]](#) Did a fast search on that, found `__arglist`, `__refvalue`, `__reftype` but have no idea how they're related to my case here...

He tends to make stuff up and when asked to provide sources never does.

he just wants to sound smart. Read back his posts.