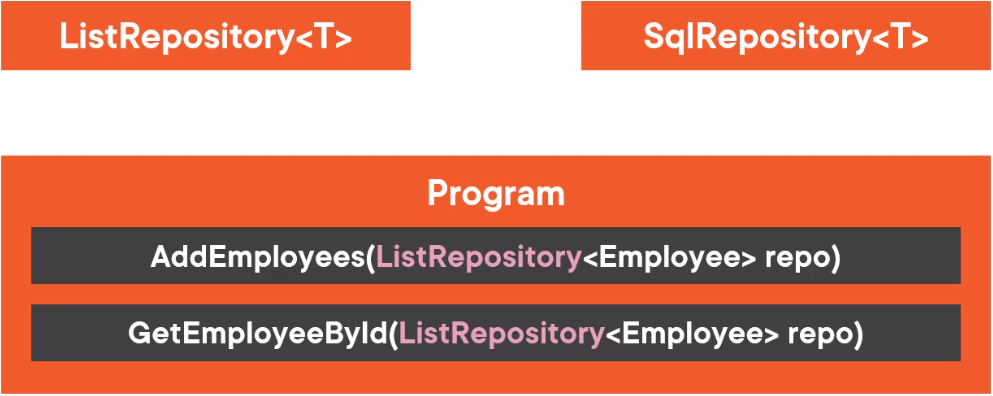
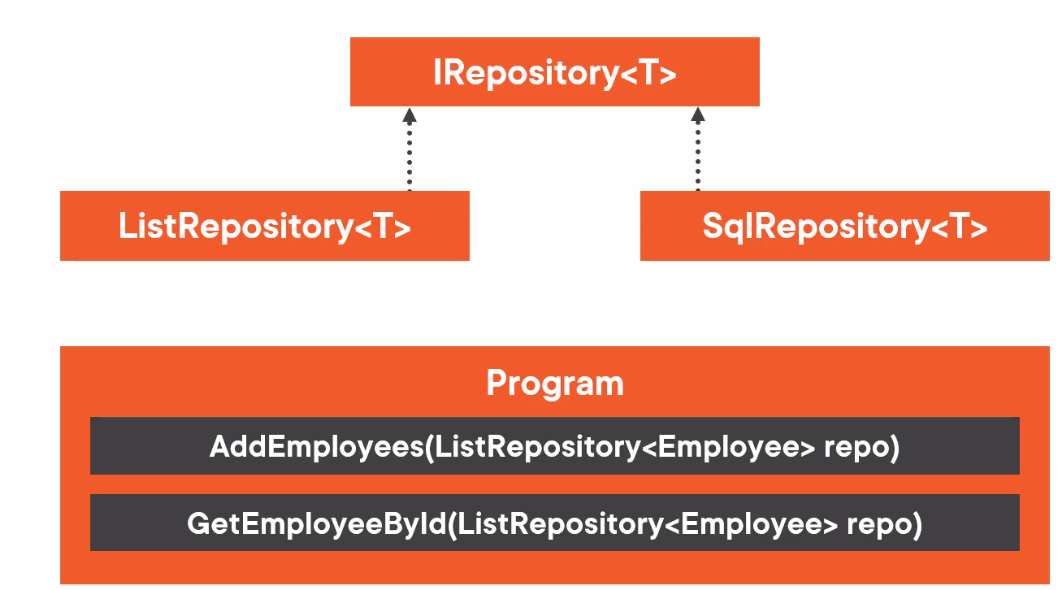
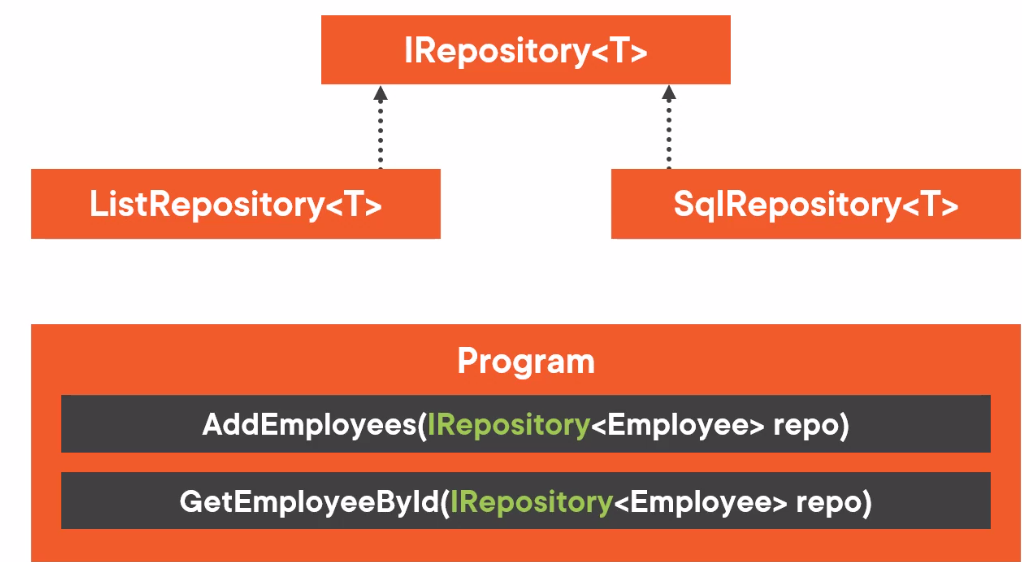
## Why use a generic Interface?

AddEmployee and GeEmployeeById are methods that have a ListRepository parameter. This means that you cannot pass a SqlRepository instance to these methods



There for we need to implement IRepositry<T> interface



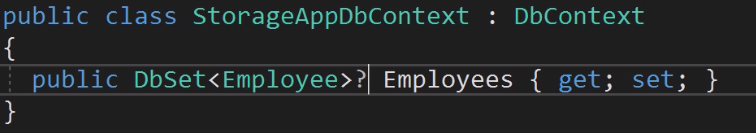
When implemented it can work with both, this is a software principle that we used here and it called Dependency inversion principle.

Dependency inversion principle.



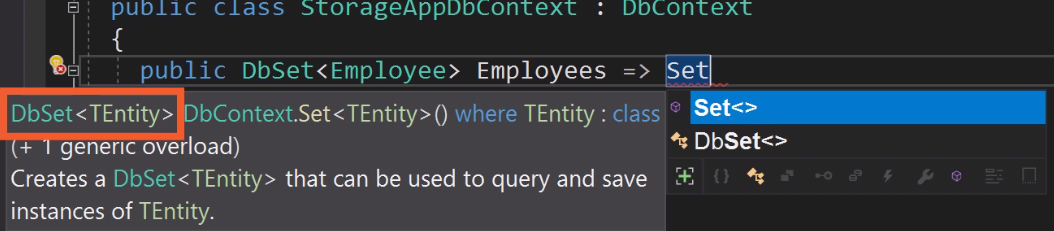
## Build a SqlRepository<T> Class \*

On the DbContext the DbSet is usually not null as it is initialized in the constructor of the DbContext base class.

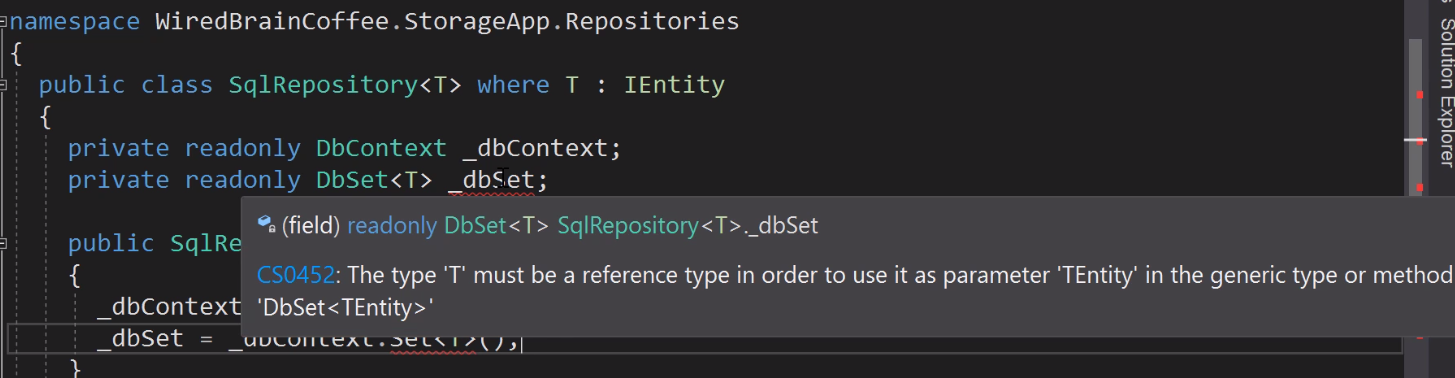


Instead of using? we can use an expression body and use the Generic set Method that is defined in the DbContext base class

And this method returns DbSet of <TEntity>



Since type T must be a reference type we must define the generic type constrain as class



## Conclusion

Now I have implemented a generic SqlRepository<T> class that uses an Entity framework DbContext to store the items.

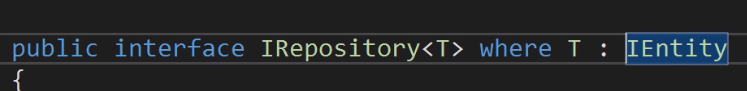
## Create a Generic Interface

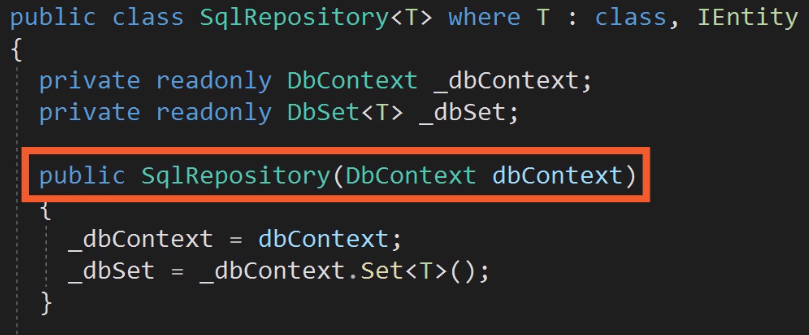
When you define type constraints like here



It means that generic classes that implement this interface also must define these types constrains for their generic type parameter

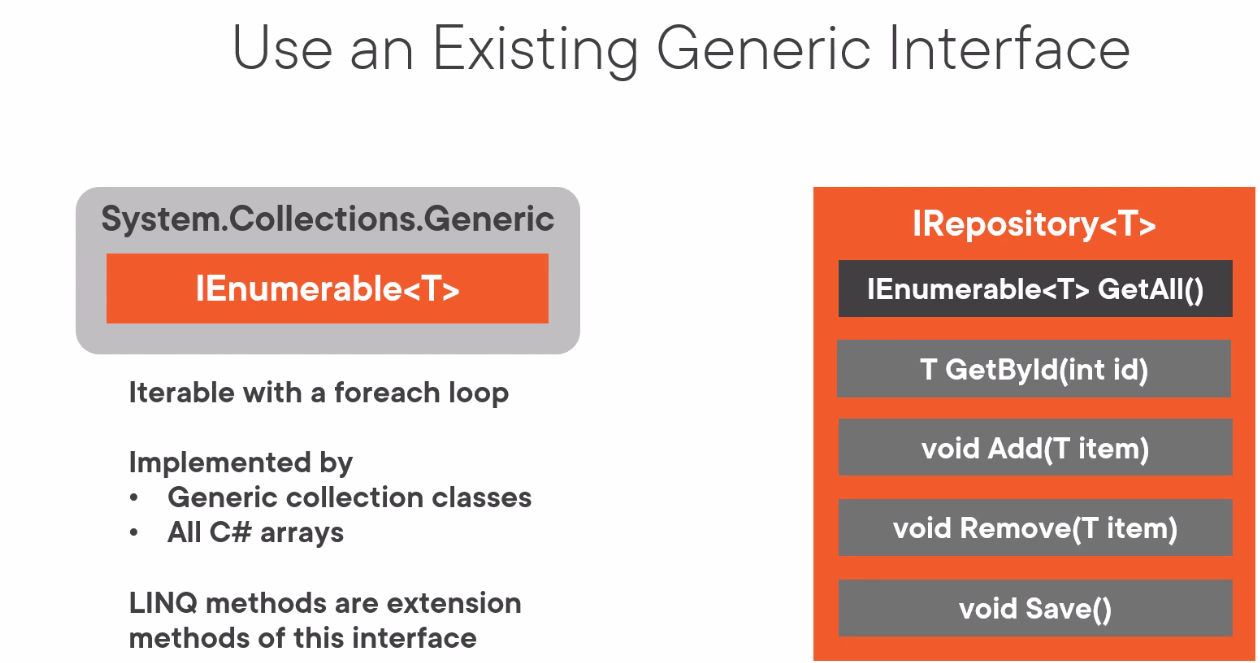


So, why having type constraints on interfaces as usually never as implementation. Keeping his IEntity type constrain means that you will get a compile error if you use a generic T type that is not of a type IEntity



## Use an Existing Generic Interface

IEnumerable <T> (read Ienummerable of t) is an existing generic interface in .NET from the namespae System.Collection.Generic , The interface is implemented by generic collections like List <T> stack <T>

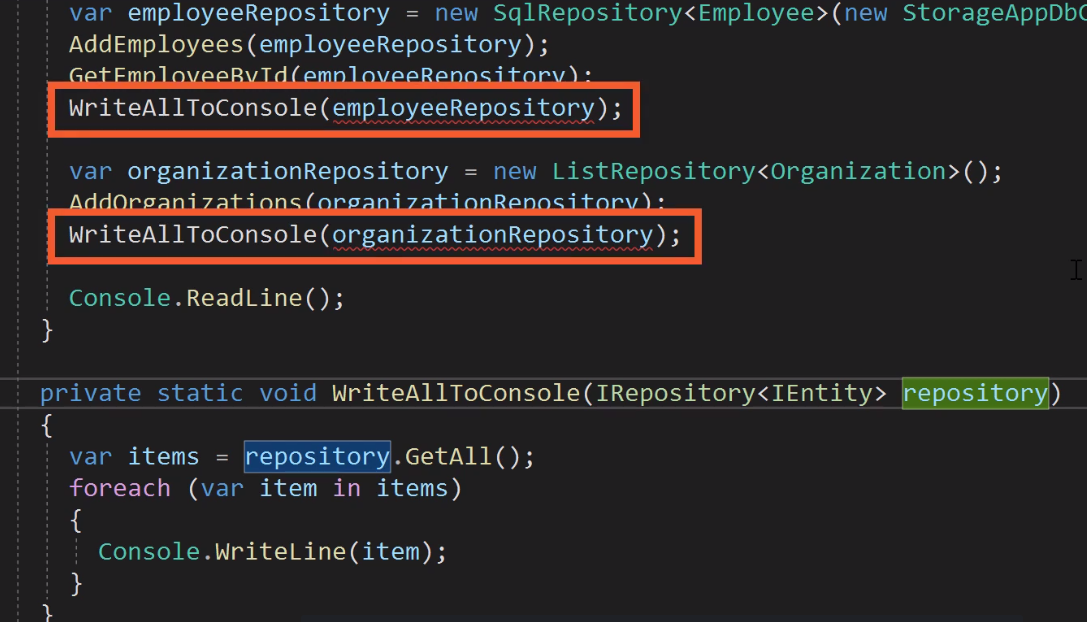


## Understanding Covariance \*

What if we want to use the GetAll method also with organization , Make IRepository<IEntity>

But not writeAllToConsole methods does not work

It because the writeAllToConsole parameter are by default invariant. This means they have to have exactly the same type as the type that you used on the class that implements the interface



It means, if you have for example …

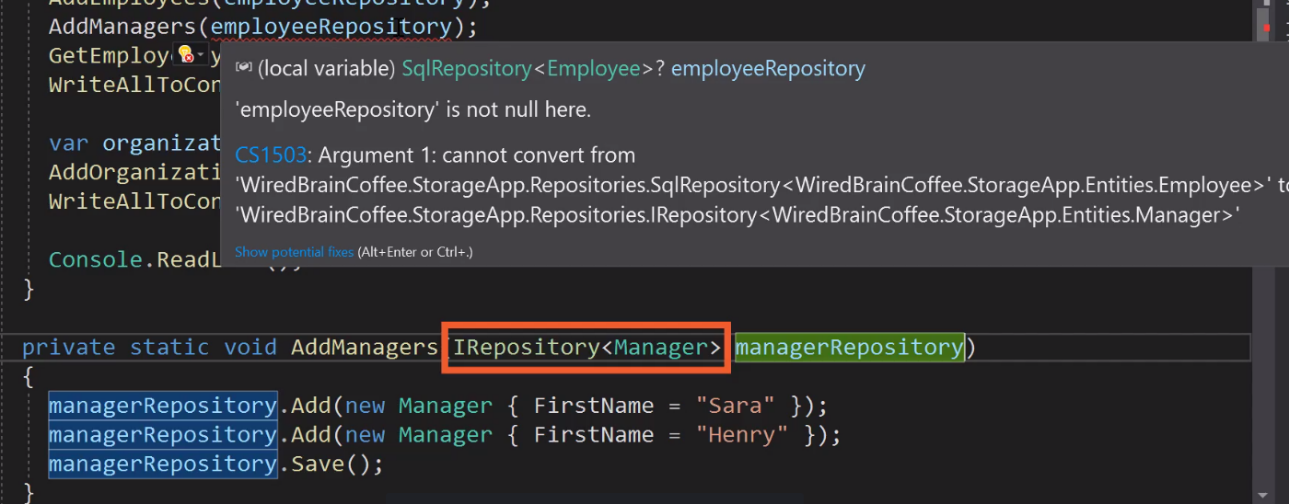
When you are getting all it does matter from where you are getting but if you are writing you want to be more specific to ether employee or organization

Covariant less specific type can be used

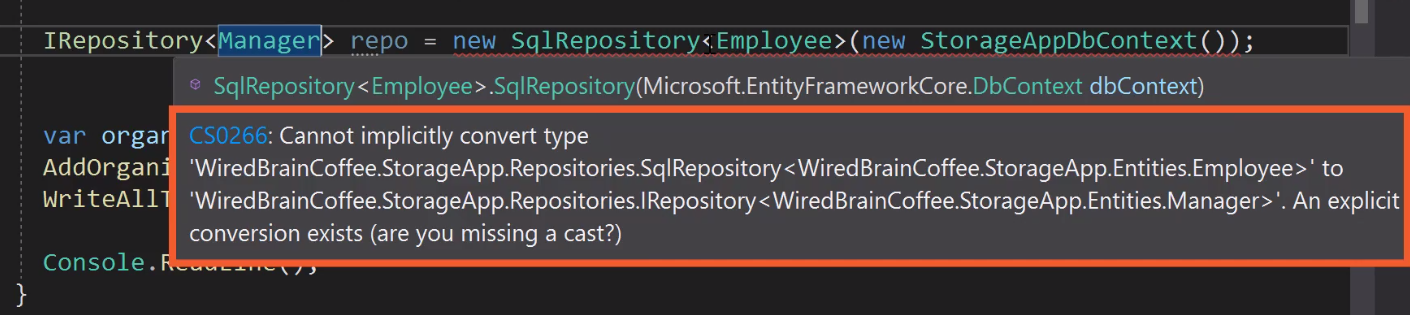


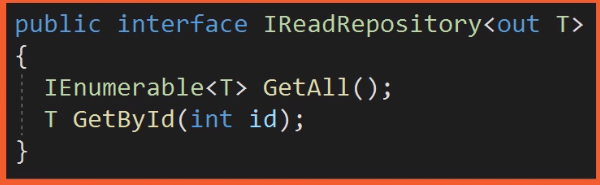
## Understanding Contravariance \*

We know that the generic type parameter of this IRepository interface is invariant

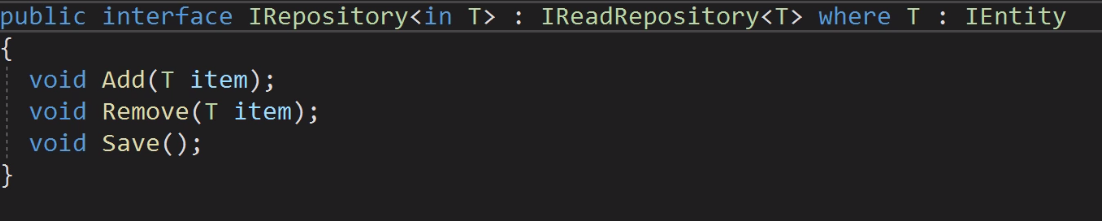


This is because manager is a more specific type than employee. But you can support this by defining the generic type parameter on the IRepository interface as contravariant

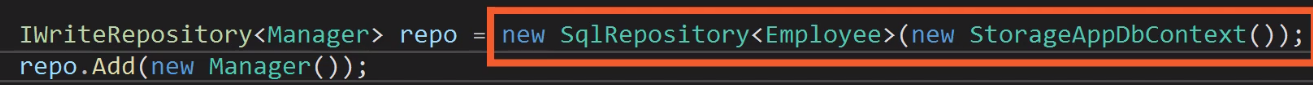


This means the type can be less specific on the interface

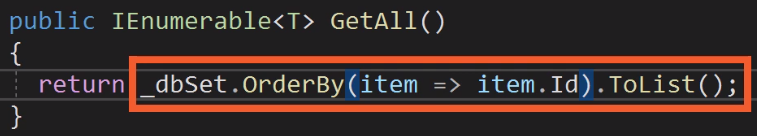
To create a contravariant type parameter. This works if the generic type is not used as a return value in the generic interface



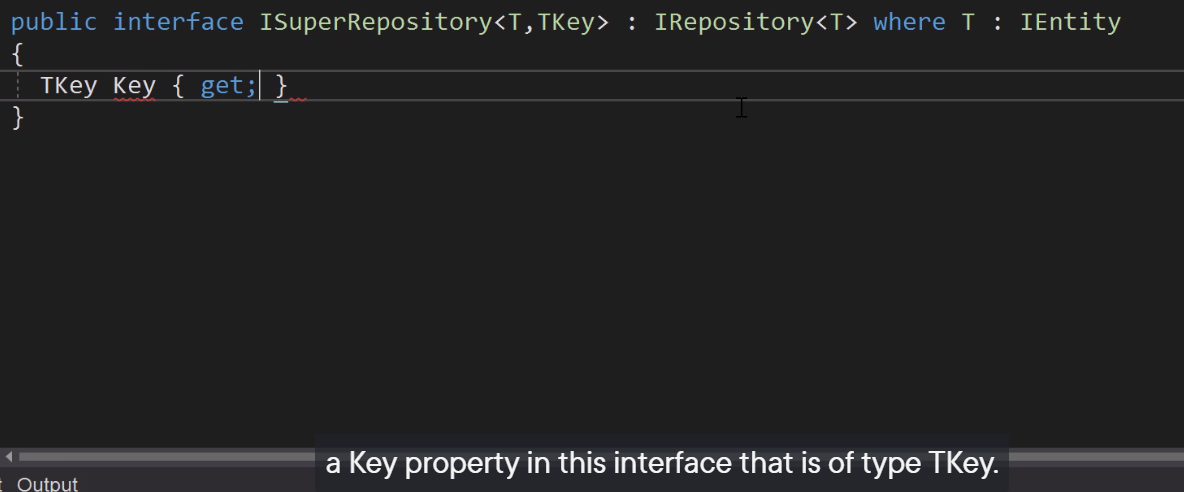
When you call the add method for example with a new manager. Manager is passed to the SqlRepository’s Add method that can handle employees

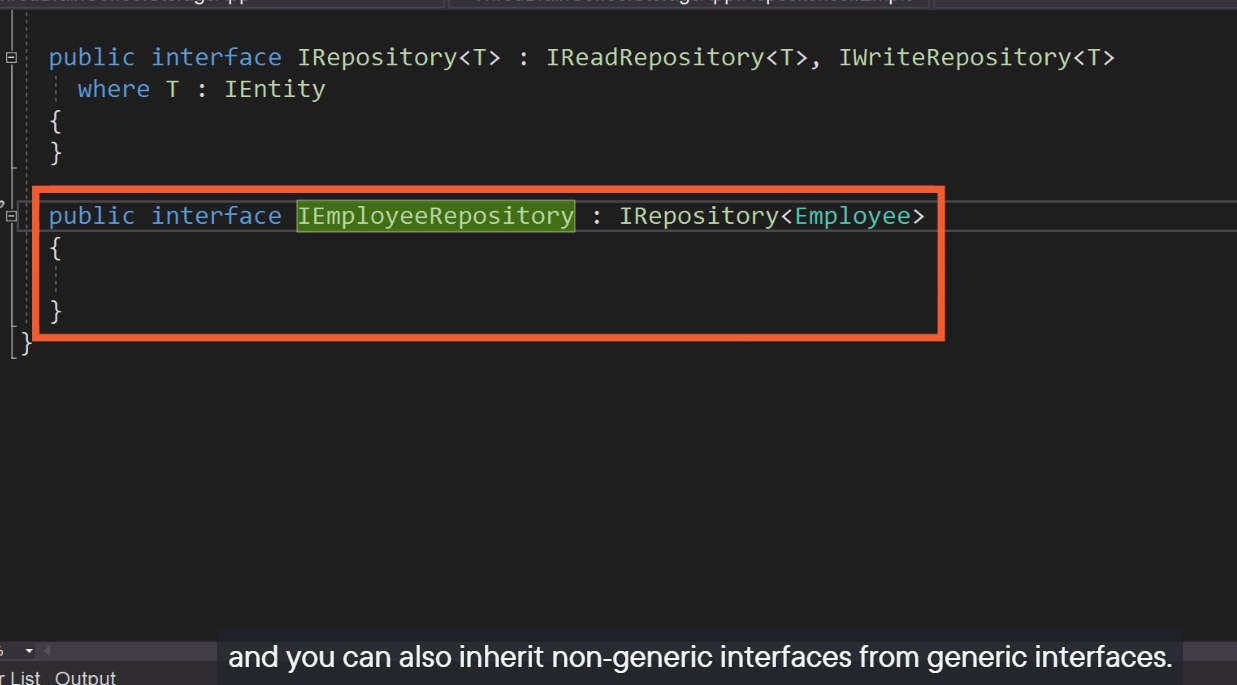


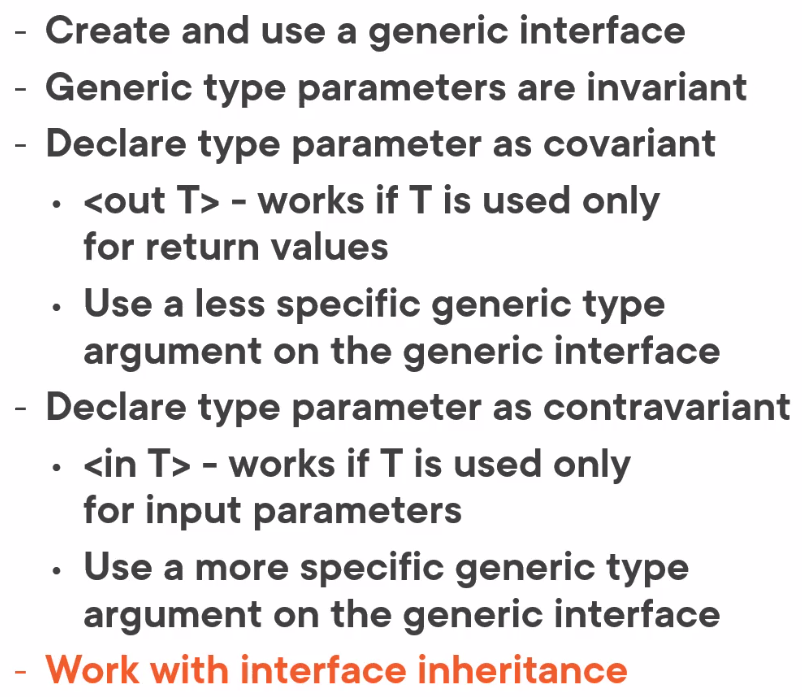
You will get the item ordered



Work with Interface Inheritance







## Definitions

Entity: a thing with distinct and independent existence.