**Iterable interface**   
Jakob Jenkov

The Java iterable interface (java.util.Iterable) represents a collection of objects which can be iterated.

The iterable interface is not the same as the iterator interface. the two interfaces are related but different.

The way the iterable interface is related to the ite rator interface is that an iterable is a source of an iterator /in other words from an iterable you can obtain an iterator.

Example of Iterable –-

1. At first I create a list add three elements to it and then
2. notice that because the list interface is a sub interface of collection it extends the collection interface and the collection interface extends the iterable interface that means that

I can actually cast the list to a collection **– Collection<String> collection = list;**

and I can cast the collection to an iterable-

**Iterable<String> iterable = collection;.**

1. Now I didn't have to go through these steps to get to the iterable .
2. I could just have cast a list to an Iterable directly or just down here where I'm using it I could have used the list directly this would have worked too.
3. once I have an instance of an object that implements the iterable interface. I can iterate the elements of that iterable in three different ways.
4. The first way is to use the built-in for each loop in Java.

so I write for and then I declare a

01:45

variable string element and then : and then I write the name of the iterable

01:50

now in this case it's this horrible declared here but I could have used the

01:55

list as I just explained you see the compiler doesn't complain now what

02:00

happens here is that for each element available in the iterable

02:06

for each iteration in the loop each element in this Israel will be bound to

02:13

this variable so we will loop through this loop one time

02:16

for each element in this iterable and bind each element to this variable and

02:23

then we can act on this variable inside of this loop that is the first way the

02:30

second way is to simply call the iterable dot iterator method which

02:36

returns a standard iterator and once we have an iterator

02:41

we can iterate it through using a standard while loop where recall while

02:47

iterator has next in a loop and inside of this while loop we called iterator

02:52

next method to obtain each of the elements available in the iterator and

02:57

the third way to iterate through the elements of an iterable is to call the

03:03

Israel dot for each method and as a parameter to the for each method I have

03:09

to pass a lambda expression at Java lambda expression and for each element

03:15

in the iterable this lambda expression gets called and the element each element

03:22

will be passed as parameter to the lambda expression and that means inside

03:25

this lambda expression body here I can operate on this element now as you can

03:34

see down here at the bottom it is also possible to obtain a splitter writer

03:39

from an iterable and the splitter rater is another component which can be used

03:44

to each rate the elements of an iterable but I will not cover the splitter rater

03:48

in more detail in this video now you know it exists you can go and find

03:53

information about it yourself there are actually several interfaces or and

03:59

classes in Java which implement the iterable interface now you can see here

04:04

I used list which is an interface itself which extends the iterable interface but

04:10

the same is true of a set I said now I need to use a hash set instead here now

04:16

as you can see yeah I haven't changed the variable name that doesn't matter

04:21

but you can see since set also extends the collection into

04:27

face and collection in extense the iterable interface you can actually also

04:33

iterate the elements of a set with the standard Java for each loop or you can

04:38

obtain an iterator or you can call the interval that for each on it let's just

04:47

quickly have a look at the iterable interface definition it's actually quite

04:52

simple and then later we will have a look at how to implement the iterable

04:55

interface yourself so that you can make your own collection classes iterable

05:02

with the standard java for each loop but first let's have a look at the iterable

05:07

interface I will just open up the interface definition now never mind all

05:13

the green text out here all the comments right so iterable has a method here

05:19

called iterator it has a method called for each we have already seen that in

05:24

use and then it has the splitter rate so a method which we have also seen in use

05:31

already notice that the splitter rates a method and the for each method our

05:38

default methods which means a you can actually have a code in them which is

05:45

normally not possible in in in an interface and B you do not actually have

05:49

to implement these two methods when you implement the interval interface if you

05:54

do not overwrite these methods do not implement this fridge and the splits

05:58

rater method your implementation inherits the default implementation

06:04

found in the iterable interface now let's have a look at an implementation

06:12

of the iterable interface as you can see it is just a regular java class which is

06:22

declared to implement the iterable interface and with the generic type

06:27

string so that you can iterate string elements contained inside of this Israel

06:35

input up yet now internally the iterable

06:43

in class here just keeps its elements in a standard list you are free to choose

06:49

how you want to keep your elements internally inside such an iterable

06:52

implementation but I've just used a list here for the sake of making the example

06:58

simpler and you can see this method here this add method is also just adding the

07:04

element directly to the list internally ok this add methods is not a method you

07:10

have to add or implement it's not part of the Israel interface I've just done

07:15

that so that later I want to show you how to use this class it is easy to add

07:19

elements to it now the iterable interface requires that you implement

07:26

this method called iterator and there are actually a few mistakes here right

07:31

first of all the generic type should be string because I have declared up here

07:37

that the class is implementing the eatable interface with the generic type

07:43

string so the iterator returned should have the same generic type second I

07:49

should not call this the iterator but of course these two elements dot iterator

07:54

right now it is correct now let's have a look at how it looks when we use this

08:02

Israel import lass I have prepared an example here first we create an instance

08:10

of this Israel input which I have just showed you the implementation of then

08:15

add two elements to it which will be stored in the list internally kept

08:20

internally in them might in the interval improve and then we can use the iterable

08:26

inside of a standard Java for each loop each of these elements added here will

08:32

be bound to this element variable for each iteration in the loop now let's run

08:38

this example and see that that is actually true as you can see the code

08:41

prints out these elements and as you can see we get printed out one and two which

08:46

where the elements added up here so you can see the iteration of the elements in

08:51

my iterable imple works obviously I could also obtain an iterator from my

08:59

iterable right because after all I have just implemented this method in my

09:06

eternal implementation now I could iterate through this iterator

09:11

with a standard while loop I can also obtain a splitter rater from the from my

09:21

iterable implementation because even though my implementation does not

09:27

explicitly implement the split eraser method the iterable interface comes with

09:33

a default implementation of the split aerator method which I then inherit my

09:37

iterable implementation inherits that default implementation and the same is

09:43

true for the for each method that is also inherited from the default

09:51

implementation found in the iterable interface even though my implementation

09:56

does not explicitly implement this method so I can call it pass a lambda

10:03

expression to it which will get called for each element in there iterable and

10:07

then I can print out the element now let me just remove these two lines here and

10:14

let's try to run this example and we expect of course to get two sets of

10:22

printouts of the elements like one two from the first iteration up here and

10:26

then again one two from the second iteration down here which was triggered

10:31

by the call to the for each method finally I just want to talk a little bit

10:36

about the iterable performance and if you look at this example here I create a

10:45

list I add two elements to it and then I iterate it with the standard for each

10:50

loop right now you to read through all the elements now what actually goes on

10:54

behind the scenes is this down here right so an iterator is obtained from

11:00

the iterable and then iterable or the iterator is looped

11:05

through with the standard while loop it doesn't look like that in your code but

11:09

that is actually what happens now from a performance perspective this is normally

11:15

not a big problem but if you are iterating through a list thousands of

11:19

times per second like I have done that sometimes a hundred thousand times per

11:22

second then for each time you need to eat right through the list you are

11:29

actually creating an iterator which is a new object right so you are creating a

11:34

lot of objects a lot of iterator object which will put some pressure on the

11:38

internal memory management in Java plus as well the garbage collector now so if

11:44

you really really really need maximum performance you can use a standard for

11:49

loop as I have shown you here right you start by creating a variable index

11:57

variable called I set it to zero and then another like a boundary a variable

12:01

called lists are called in which is set to the size of the list and then we each

12:06

rate you know as long as I is less than in as long as I is less than the size of

12:11

the list we we increment I right and then inside of this loop I can obtain

12:18

the each element in the list by calling lists that get I now this is actually

12:25

slightly faster than these two loops up here if you really really really need to

12:31

get that last bit of performance out of your dupe of your iteration but in in

12:37

most cases you won't need it now obviously this iteration here only works

12:45

for lists you would not be able to use this iteration with the sets or a map

12:51

because these two classes or interfaces do not have a get method where you can

12:57

get an object based on its index so it only works for lists but in case you

13:03

need performance while it's rating a list you can use this loop instead that

13:08

pretty much covers what you need to know about the Java iterable interface in

13:12

order to start using it remember to check out the description below the

13:16

video because

13:17

a link to a textual version of this tutorial as well as other related

13:21

tutorials and if you liked this video hit the like button and maybe subscribe

13:25

to this channel as well in case you want to watch more videos like this one