Consumer Functional Interface

Represents an operation that accepts single input argument and returns no result. We have covered a post in which we iterated a List<String> using a default method forEach()of Iterable interface. [Click here](http://data-structure-learning.blogspot.com/2015/06/iterable-interfaces-foreach-method.html) to read about it.

|  |
| --- |
| **Consumer Interface Declaration**  **public** **interface** Consumer<T>  Consumer interface has single type *T*. |
| **accept() method**  **void** accept(T t);  This method is used to perform operation on given argument.  Let us take very simple example of printing a String on console.  Consumer<String> consume=(str) -> System.***out***.println(str);  consume.accept("Hello World");//Outputs Hello World  We will discuss accept() method in detail in a moment. |
| **andThen() method**  **default** Consumer<T> andThen(Consumer<? **super** T> after) {  Objects.*requireNonNull*(after);  **return** (T t) -> { accept(t); after.accept(t); };  }  This method returns the composed Consumer that performs, this operation followed by *after* operation. This method will throw the NullPointerException is *after Consumer* is null.  Below is the code for 2 consumers one for converting String to upper case and another to convert to lower case.  Consumer<String> upperCase = (str) -> System.***out***.println(str.toUpperCase());    Consumer<String> lowerCase = (str) -> System.***out***.println(str.toLowerCase());  upperCase.andThen(lowerCase).accept("AaBb");//First uppercase then lowercase.  Output  AABB  aabb |

Now let us understand how to iterate Collection using forEach() method. The forEach() method in Iterable interface accepts Consumer as parameter. Below is the code for forEach() method.

**default** **void** forEach(Consumer<? **super** T> action) {

Objects.*requireNonNull*(action);

**for** (T t : **this**) {

action.accept(t);

}

}

Let’s say we have a List<String> of languages.

List<String> languages = **new** ArrayList<String>(Arrays.*asList*("Java", "JavaScript", "C#", "Python"));

Now let us write forEach() method to print all the elements of List using Consumer as anonymous inner class syntax.

/\*\*

\* languages is list with following values.

       \* [Java, JavaScript, C#, Python]

       \* \*/

       languages.forEach(**new** Consumer<String>() {

**public** **void** accept(**final** String language) {

            System.***out***.println(language);

}

});

Yes this method does the following.

1. Accepts the parameter String language.
2. Prints it.
3. Repeat till list is empty

Here, accept method just prints the elements of the list. We can do much more but that’s for later.

But, no one likes the Anonymous inner class as the syntax is much clumsier and method looks verbose.

1. GOOD NEWS, you can use lambda operator here. Why? Because Consumer is Functional Interface.

languages.forEach((**final** String language) -> System.***out***.println(language));

1. We can drop the type of the parameter as we have only one method in Functional Interface on which we can apply lambda operator. So it will look like this.

languages.forEach((language) -> System.***out***.println(language));

1. If there is single parameter then we can eliminate the brackets outside the parameter too.

languages.forEach(language -> System.***out***.println(language));

1. Excellent, now let us make this more concise by using method reference. We will learn about this syntax later.

languages.forEach(System.***out***::println);

That’s all on Consumer Interface. ☺