Reusing Predicates

In previous [posts](http://data-structure-learning.blogspot.com/p/functional-programming-in-java.html) of functional programming we saw several different things. We saw basics of [Lambda Calculus](http://data-structure-learning.blogspot.com/2015/05/lambda-calculas.html), [Functional Interfaces](http://data-structure-learning.blogspot.com/2015/06/functional-interfaces-java-8.html), [Iteration problems prior to Java 8](http://data-structure-learning.blogspot.com/2015/06/problems-with-normal-iterations-on.html) and [new way to iterate using forEach(..)](http://data-structure-learning.blogspot.com/2015/06/iterable-interfaces-foreach-method.html). Then we learnt to extend [forEach(..) method to transform the list](http://data-structure-learning.blogspot.com/2015/06/transforming-list-using-foreach.html). Then we learn about [Predicate Interface](http://data-structure-learning.blogspot.com/2015/06/predicate-functional-interface.html) and its [test](http://data-structure-learning.blogspot.com/2015/06/predicate-interface-for-filtering.html), [and](http://data-structure-learning.blogspot.com/2015/06/predicate-interface-and-method-joining.html), [negate](http://data-structure-learning.blogspot.com/2015/06/predicate-interface-negate-method.html) and [or](http://data-structure-learning.blogspot.com/2015/06/predicate-interface-or-method-short.html) methods.

Till now we have covered several topics of [Java 8](http://data-structure-learning.blogspot.com/p/functional-programming-in-java.html). But we have a long path [ahead](http://data-structure-learning.blogspot.com/p/main-page.html)☺.

In this post we will learn about reusing the predicates. Predicates are good thing to write but we need to make sure that we use it properly.

Let us write predicate for finding the total count of elements in list greater than some number.

Let us define 3 lists first.

List<String> names=**new** ArrayList<String>

(Arrays.*asList*("Robb","Ramsey","Ned","Sansa"));

List<String> places=**new** ArrayList<String>

(Arrays.*asList*("San Jose","Cupertino","Meeren"));

List<String> companies=**new** ArrayList<String>

(Arrays.*asList*("Paypal","Microsoft","Google"));

Now we will use predicate to find the total count in each list of String length greater than some number.

**long** namesCount = names.stream()

.filter(name -> name.length() > 4)

.count();

**long** placesCount = places.stream()

.filter(place -> place.length() > 6)

.count();

**long** companiesCount = companies.stream()

.filter(company -> company.length() > 5)

.count();

Now, let us print the output.

System.***out***.println("namesCount "+namesCount);

System.***out***.println("placesCount "+placesCount);

System.***out***.println("companiesCount "+companiesCount);

Output:

namesCount 2

placesCount 2

companiesCount 3

It works fine but see what we did for getting this values. First, we hardcoded the values on right hand side of length of String. Then we wrote the same piece of code 3 different times which is almost same. If we need to change things in future then we need to change the code which is last thing you want to do.

So we will write a method that will take one parameter as int.

**public** **static** Predicate<String> lengthChecker(**int** length){

Predicate<String> predicate = str -> str.length() > length;

**return** predicate;

}

Looking at this method we know that it can accept the length as parameter. Now our value is not hard coded.

Now we can change our methods

**long** namesCount = names.stream()

.filter(*lengthChecker*(4))

.count();

**long** placesCount = places.stream()

.filter(*lengthChecker*(6))

.count();

**long** companiesCount = companies.stream()

.filter(*lengthChecker*(5))

.count();

System.***out***.println("namesCount "+namesCount);

System.***out***.println("placesCount "+placesCount);

System.***out***.println("companiesCount "+companiesCount);

Output:

namesCount 2

placesCount 2

companiesCount 3

So we just learnt how to use Predicate repeatedly without breaking the code or modifying it in future.

In next post we will how to use Function Interface which accepts one argument and produces result. You can assume that we will input the String and then we will produce a result using Predicate interface.