Optional class filter(..) method

In previous posts we saw how to [avoid Null Pointer Exception](http://data-structure-learning.blogspot.com/2015/07/avoid-null-pointer-using-optional-class.html) using Optional<T> class, [Optional<T> class introduction](http://data-structure-learning.blogspot.com/2015/07/optional-class-introduction.html), Optional<T> class [of(), ofNullable() and empty()](http://data-structure-learning.blogspot.com/2015/07/optional-class-of-ofnullable-empty.html) method and [Optional<T> class ifPresent() method](http://data-structure-learning.blogspot.com/2015/07/optional-class-ifpresent-method.html).

In this post we will explore filter method of Optional class. Filter criteria depends on the Predicate interface. I have written [several posts](http://data-structure-learning.blogspot.com/p/1.html) on Predicate interface. It is prerequisite for understanding this method.

First let us see the code of filter method then we will see how to use it.

Code for filter method

|  |
| --- |
| **public** Optional<T> filter(Predicate<? **super** T> predicate) {  Objects.*requireNonNull*(predicate);  **if** (!isPresent())  **return** **this**;  **else**  **return** predicate.test(value) ? **this** : *empty*();  }  First it makes sure that predicate is not null. If predicate is null then it will throw NullPointerException.  Then, if the value is present and value matches the given predicate then return this instance of Optional<T>. Else it returns the empty instance. |

Now let us use filter method for our problem statement. The problems statement is write a method that will pick an element that starts with some prefix.

List<String> days = **new** ArrayList<String>(Arrays.*asList*("Monday",

"Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"));

String prefix = "Wed";

**public** **static** **void** filterDemo(**final** List<String> days, **final** String prefix){

**final** Optional<String> found=days.stream()

.filter(day -> day.startsWith(prefix))

.findFirst();

System.***out***.println(found);// Optional[Wednesday]

**final** Optional<String> foundFilter = found.filter(day -> day.contains("day"));

System.***out***.println(foundFilter);// Optional[Wednesday]

**final** Optional<String> notFoundFilter = found.filter(day -> day.contains("xyz"));

System.***out***.println(notFoundFilter);// Optional.empty

}

So we just wrote a method for finding first day in List of days that contains a prefix. Remember this filter filter(day -> day.startsWith(prefix)) method used is of list and not of Optional class.

**final** Optional<String> foundFilter = found.filter(day -> day.contains("day")) This line of code uses filter method of Optional<T> class. It checks whether the value in Optional<T> object contains the CharSequence "day". String class implements CharSequence interface. This passes the test and prints Optional[Wednesday].

**final** Optional<String> notFoundFilter = found.filter(day -> day.contains("xyz")); This line of code also uses filter method of Optional<T> class. It checks whether the value in Optional<T> object contains the CharSequence "xyz". String class implements CharSequence interface. This **does not passes** the test and prints Optional.empty.