

El uso de `takeWhile` y `dropWhile` con Java 9 poco a poco se irá extendiendo . Con la llegada de Java 8 y el uso de streams muchas cosas han cambiado en la programación cotidiana de Java. Sin embargo quedaban algunas casuísticas que cubrir en cuanto a operadores. Java 9 añade dos operadores importantes `takeWhile` y `dropWhile` que permiten filtrar un conjunto de elementos mientras una condición determinada se cumpla o no se cumpla .Vamos a ver unos ejemplos sencillos partiendo de una lista sencilla y recorriéndola como Stream.

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;

public class Principal {

    public static void main(String[] args) {
        List<String> lista=
Arrays.asList("hola","que","tal","estas","hoy","aqui");
        lista.forEach(System.out::println);

    }

}
```

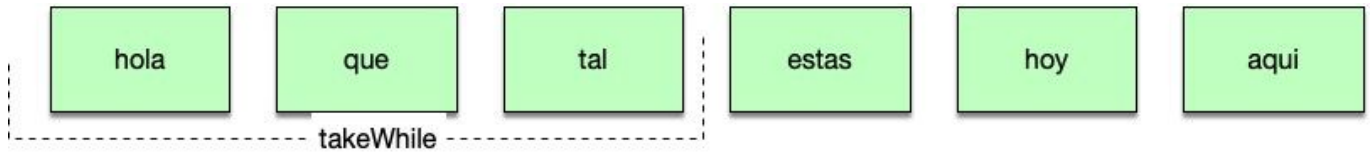
El resultado lo podemos ver en la consola:

```
hola  
que  
tal  
estas  
hoy  
aqui
```

Hasta aquí todo correcto pero hay a veces que queremos seleccionar un conjunto de elementos mientras se cumpla una condición para ello podríamos hacer lo siguiente utilizando el operador `takeWhile`:

```
package com.arquitecturajava;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class Principal2 {  
  
    public static void main(String[] args) {  
        List<String> lista=  
Arrays.asList("hola","que","tal","estas","hoy","aqui");  
lista.stream().takeWhile((s)->s.length()<5).forEach(System.out::println);  
  
    }  
  
}
```

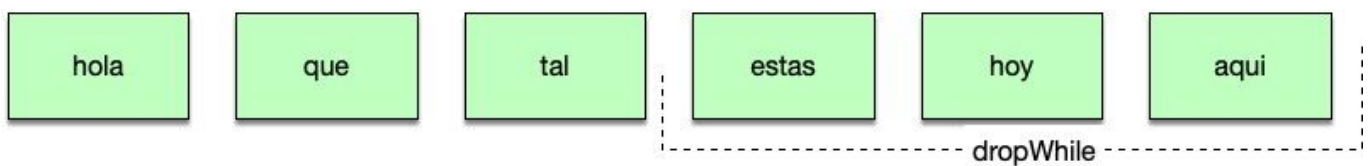
En este caso se seleccionan todos los elementos mientras que se cumpla la condición.



La consola los mostrará:

```
hola
que
tal
```

De igual forma podemos realizar la operación contraria y usar dropWhile.



La consola muestra el resultado:

```
estas
hoy
aqui
```

Utilizando dropWhile y takeWhile

Acabamos de ver cómo funcionan ambos métodos pero a veces es difícil entender en que casuísticas podemos utilizarlos de forma clara. Vamos a ver un par de ejemplos , en el primero vamos a usar el método iterate de los Streams para construir un bucle for con ellas usando takeWhile.

```
package com.arquitecturajava;

import java.util.stream.IntStream;

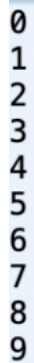
public class Principal3 {

    public static void main(String[] args) {
        IntStream.iterate(0,
            i->i+1).takeWhile(i->i<10).forEach(System.out::println)

    }

}
```

El resultado será:



```
0
1
2
3
4
5
6
7
8
9
```

Otra de las opciones muy comunes es combinar ambos para realizar una selección concreta de elementos . Imagenemos que disponemos de un fichero de texto con el siguiente contenido:

```
texto1
```

texto2

zona destacada

texto3

texto4

Queremos imprimir por la consola el contenido dentro de los “**” la zona destacada. Con takeWhile y dropWhile es instantáneo.

```
package com.arquitecturajava;
```

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.stream.Stream;
```

```
public class Principal7 {
```

```
    public static void main(String[] args) {
        String nombreFichero = "hola.txt";
```

```
        try (Stream<String> stream =
Files.lines(Paths.get(nombreFichero))) {
```

```
            stream
                .dropWhile(s->!s.contains("**"))
                .skip(1)
                .takeWhile(s->!s.contains("**"))
                .forEach(System.out::println);
```

```

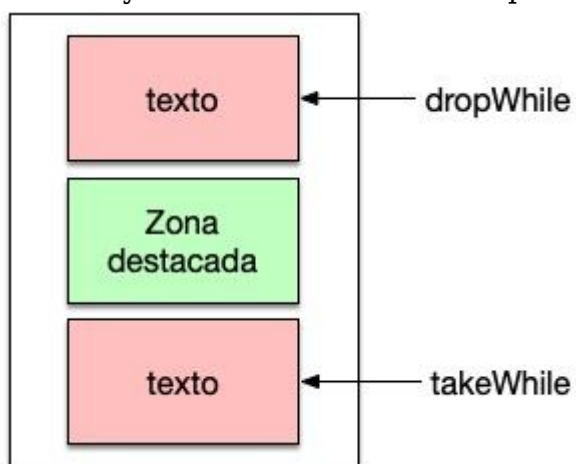
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}

```

En este caso el programa elimina todas la líneas hasta llegar a los “*” lee las líneas dentro de ellos y elimina todas las líneas que estén a continuación de ellos.



El resultado lo vemos en la consola:

```

zona destacada

```

El uso de takeWhile y dropWhile es importante en el manejo de java Streams.

Otros artículos relacionados:

1. [Java Stream String y Java 8](#)

2. [Java 8 Lambda Expressions \(I\)](#)
3. [Java Parallel Stream y rendimiento](#)
4. [Java Streams](#)