

Ejemplos prácticos de Expresiones Lambda en Java 8



📅 26 noviembre, 2016 (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8/>) 🚩 Elivar Largo (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/author/elargo/>)

Ejemplos prácticos de Expresiones Lambda en Java 8

Hola que tal, en un tutorial anterior habíamos visto cuales son los Entendiendo las Expresiones Lambda paso a paso (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/entendiendo-paso-a-paso-las-expresiones-lambda-en-javascript/>), esta vez vamos hacerlo un poco más práctico, haciendo ejemplos prácticos de expresiones Lambda en Java 8.

En este tutorial combinaremos el **API Stream** que junto con las expresiones Lambda forman una combinación extremadamente poderosa como se menciona en el artículo Processing Data with Java SE 8 Streams (<https://web.archive.org/web/20161201070414/http://www.oracle.com/technetwork/articles/java/ma14-java-se-8-streams-2177177.html>).

2177646.html).

Antes de empezar vale la pena recordar la sintaxis de las expresiones Lambda.

Las expresiones Lambda están compuestas de la siguiente manera:

1. Los **parámetros o argumentos** que pueden ir o no dentro de paréntesis, esto depende de cuantos existan, si hay más de uno, deben ir obligatoriamente dentro de paréntesis y separados por comas.
2. Una flecha a la derecha **->**.
3. El **cuerpo** que puede ser una expresión y que va dentro de llaves si existe más de una sentencia.

De forma general una expresión lambda se puede definir de la siguiente manera::

```
1 (argumentos)->{cuerpo}
2 (int a, int b) -> a >b;
3 (int a, int b) -> System.out.println(a + b); return a + b; }
```

Para realizar los ejemplos sería bueno que leas la entrada Entendiendo las Expresiones Lambda paso a paso (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/entendiendo-paso-a-paso-las-expresiones-lambda-en-java/>), para evitar confusiones.

API Stream

A través del API Stream podemos trabajar sobre colecciones como si estuviéramos realizando sentencias SQL pero de una manera limpia y clara, evitando bucles y algoritmos que ralentizan los programas e incluso hacen que el código se torne inmanejable.

Partes que componen un Stream

Existen 3 partes que componen un Stream que de manera general serían:

1. Un Stream funciona a partir de una lista o colección, que también se la conoce como la **fuente de donde obtienen información**.
2. **Operaciones intermedias** como por ejemplo el método filter, que permite hacer una selección a partir de un predicado.

3. **Operaciones terminales**, como por ejemplo los métodos max, min, forEach, findFirst etc.

De manera que la sintaxis puede quedar de la siguiente manera:

```
1 listaAlumnos.stream.filter(expresion_lambda).forEach(Sysout.out::println);
```

Para el ejemplo vamos a crear una clase Alumno.java, que va permitir guardar un alumno con sus nombres, apellidos, curso que tomó, nota del curso, etc.

```
1 package com.ecodeup.lambda;
2
3 public class Alumno {
4     private int id;
5     private String cedula;
6     private String nombres;
7     private String apellidos;
8     private String nombreCurso;
9     private double nota;
10    private int edad;
11
12    public Alumno() {
13    }
14
15
16    public Alumno(int id, String cedula, String nombres, String apellidos, String nombreCurso, double nota, int edad) {
17        this.id = id;
18        this.cedula = cedula;
19        this.nombres = nombres;
20        this.apellidos = apellidos;
21        this.nombreCurso = nombreCurso;
22        this.nota = nota;
23        this.edad = edad;
24    }
25
26    public int getId() {
27        return id;
28    }
29
30    public void setId(int id) {
31        this.id = id;
32    }
33
34    public String getCedula() {
35        return cedula;
36    }
37
38    public void setCedula(String cedula) {
39        this.cedula = cedula;
40    }
41
42    public String getNombres() {
43        return nombres;
44    }
45
46    public void setNombres(String nombres) {
47        this.nombres = nombres;
```

```
48     }
49
50     public String getApellidos() {
51         return apellidos;
52     }
53
54     public void setApellidos(String apellidos) {
55         this.apellidos = apellidos;
56     }
57
58     public String getNombreCurso() {
59         return nombreCurso;
60     }
61
62     public void setNombreCurso(String nombreCurso) {
63         this.nombreCurso = nombreCurso;
64     }
65
66     public double getNota() {
67         return nota;
68     }
69
70     public void setNota(double nota) {
71         this.nota = nota;
72     }
73
74     public int getEdad() {
75         return edad;
76     }
77
78     public void setEdad(int edad) {
79         this.edad = edad;
80     }
81
82     @Override
83     public String toString() {
84         return id+" | "+cedula+" | "+nombres+" | "+apellidos+" | Curso: "+nombreCurso+" | Nota: "+nota+" | Edad: "+edad;
85     }
86 }
```

Ahora en la clase DemoLambda.java creamos una lista de tipo Alumno, sobre esta lista realizaremos varias consultas tipo SQL que se suele realizar comúnmente, por ejemplo obtener **el estudiante con mayor nota**, o el que **tomó el curso de Java**, entre otro tipo de consultas. Para esto escribimos el código siguiente:

```

1 List<Alumno> listaAlumnos = new ArrayList<>();
2
3 listaAlumnos.add(new Alumno(1, "1717213183", "Javier Ignacio", "Molina Cano", "Java 8", 7, 28));
4 listaAlumnos.add(new Alumno(2, "1717456218", "Lillian Eugenia", "Gómez Álvarez", "Java 8", 10, 33));
5 listaAlumnos.add(new Alumno(3, "1717328901", "Sixto Naranjoe", "Marín", "Java 8", 8.6, 15));
6 listaAlumnos.add(new Alumno(4, "1717567128", "Gerardo Emilio", "Duque Gutiérrez", "Java 8", 10, 13));
7 listaAlumnos.add(new Alumno(5, "1717902145", "Jhony Alberto", "Sáenz Hurtado", "Java 8", 9.5, 15));
8 listaAlumnos.add(new Alumno(6, "1717678456", "Germán Antonio", "Lotero Upegui", "Java 8", 8, 34));
9 listaAlumnos.add(new Alumno(7, "1102156732", "Oscar Dario", "Murillo González", "Java 8", 8, 32));
10 listaAlumnos.add(new Alumno(8, "1103421907", "Augusto Osorno", "Palacio Martínez", "PHP", 9.5, 17));
11 listaAlumnos.add(new Alumno(9, "1717297015", "César Oswaldo", "Alzate Agudelo", "Java 8", 8, 26));
12 listaAlumnos.add(new Alumno(10, "1717912056", "Gloria Amparo", "González Castaño", "PHP", 10, 28));
13 listaAlumnos.add(new Alumno(11, "1717912058", "Jorge León", "Ruiz Ruiz", "Python", 8, 22));
14 listaAlumnos.add(new Alumno(12, "1717912985", "John Jairo", "Duque García", "Java Script", 9.4, 32));
15 listaAlumnos.add(new Alumno(13, "1717913851", "Julio Cesar", "González Castaño", "C Sharp", 10, 22));
16 listaAlumnos.add(new Alumno(14, "1717986531", "Gloria Amparo", "Rodas Monsalve", "Ruby", 7, 18));
17 listaAlumnos.add(new Alumno(15, "1717975232", "Gabriel Jaime", "Jiménez Gómez", "Java Script", 10, 18));

```

En el primer ejemplo vamos a obtener todos los alumnos de la lista:

```

1 System.out.println("*** Lista de Alumnos ***");
2     listaAlumnos.stream().forEach(a->System.out.println(a));
3     //o
4     listaAlumnos.stream().filter(a -> true).forEach(a -> System.out.println(a));

```

Lo podemos hacer de dos formas, la primera simplemente utilizando foreach a partir del stream y en el segundo caso además del método foreach utilizando el método filter con el predicado **a->true**.

A continuación vamos a imprimir todos aquellos alumnos cuyo nombre empieza con el carácter 'L' o 'G', esto se valida utilizando una **operación intermedia** con filter.

```

1 System.out.println("\n*** Alumnos cuyo nombre empiezan con el carácter L u G ***");
2 listaAlumnos.stream().
3     filter(c -> c.getApellidos().charAt(0) == 'L' || c.getApellidos().charAt(0) == 'G')
4         .forEach(c -> System.out.println(c));

```

Con el método count() que vendría siendo una **operación terminal** (mencionada anteriormente) y con la que devolvemos la longitud de la lista:

```

1 System.out.println("\n**** Número de Alumnos ***");
2     // número de elementos en la lista
3     System.out.println(listaAlumnos.stream().count());

```

Obtener los alumnos con notas mayores a 9 y que el curso sea PHP:

```

1 System.out.println("\n***** Alumnos con nota mayor a 9 y que sean del curso PHP ***");
2     // alumnos con notas mayores a 9
3     listaAlumnos.stream()
4         .filter(a -> a.getNota() > 9 && a.getNombreCurso().equals("PHP"))
5             .forEach(p -> System.out.println(p));

```

Imprimir los 2 alumnos de la lista con el método limit(numero_elementos):

```

1 System.out.println("\n***** Imprimir los 2 primeros Alumnos de la lista ***");
2     listaAlumnos.stream().limit(2).forEach(a -> System.out.println(a));

```

Obtener el alumno que tiene la menor edad:

```

1 System.out.println("\n***** Imprimir el alumno con menor edad ***");
2     // minimo por edad
3     System.out.println(listaAlumnos.stream().min((a1, a2) -> a1.getEdad() - a2.getEdad()));

```

Obtener el alumno que tiene la mayor edad:

```

1 System.out.println("\n***** Imprimir el alumno con mayor edad ***");
2     // maximo por edad
3     System.out.println(listaAlumnos.stream().max((a1, a2) -> a1.getEdad() - a2.getEdad()));

```

Obtener el primer alumno de la lista con el método findFirst():

```

1 System.out.println("\n***** Encontrar el primer Alumno***");
2     // encontrar el primer registro
3     System.out.println(listaAlumnos.stream().findFirst());

```

Obtener los alumnos cuyos nombres de curso terminen con el carácter 't':

```

1 System.out.println("\n***** Alumnos en los que los nombres de los cursos (lenguajes) que terminan en t ***");
2     listaAlumnos.stream().filter(a -> a.getNombreCurso().endsWith("t")).forEach(System.out::println);

```

Obtener los alumnos cuyos nombres de curso **contengan** el carácter 'a':

```

1 System.out.println("\n***** Alumnos que tienen un curso en el que el nombre contienen la A ***");
2     listaAlumnos.stream().filter(a -> a.getNombreCurso().contains("a")).forEach(System.out::println);

```

Obtener los alumnos cuya longitud de nombres sea mayor a 10:

```

1 System.out.println("\n**** Alumnos en que su tamaño de su nombre es mayor a 10 caracteres ***");
2     listaAlumnos.stream().filter(a -> a.getNombres().length() > 10).forEach(System.out::println);

```

Obtener los alumnos, cuyo nombre empiece con el carácter 'P' y la longitud de su nombre sea <= a 6:

```

1 // combinar predicados
2     System.out.println("\n**** Combinación de predicados ***");
3     Predicate<Alumno> empiezaConJ = a -> a.getNombreCurso().startsWith("P");
4     Predicate<Alumno> longitud = a -> a.getNombreCurso().length() <= 6;
5     // Obtiene los alumnos en los cuales el nombre del curso empieza con el
6     // carácter 'P' y la longitud sea <= a 6
7     listaAlumnos.stream().filter(empiezaConJ.and(longitud)).forEach(System.out::println);

```

Hasta ahora hemos mostrado por consola todas las consultas realizadas, pues bien si queremos asignar dicha consulta a una nueva lista debemos hacer lo siguiente:

```

1 List<Alumno> nuevaLista= listaAlumnos.stream().filter(a -> a.getNombreCurso().contains("a")).collect(Collectors.toList());
2     nuevaLista.forEach(System.out::println);

```

Estos son algunos de todos los ejemplos que se puede encontrar utilizando las Expresiones Lambda y el API Stream, existen más métodos con los cuales se puede hacer cosas más complejas y que espero abordarlos en otro entrada.

Espero que hayas aprendido algunos de los métodos donde se utilizan las expresiones lambda y el API Stream. Nos vemos en la próxima entrada.

Compartir en:  [https://web.archive.org/web/20161201070414/http://www.facebook.com/sharer.php?
u=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8/](https://web.archive.org/web/20161201070414/http://www.facebook.com/sharer.php?u=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8/)

 <https://web.archive.org/web/20161201070414/https://plus.google.com/share?url=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8/>

 [https://web.archive.org/web/20161201070414/http://twitter.com/share?
url=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8&text=Ejemplos+pr%C3%A1cticos+de+Expresiones+Lambda+en+Java+8+](https://web.archive.org/web/20161201070414/http://twitter.com/share?url=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8&text=Ejemplos+pr%C3%A1cticos+de+Expresiones+Lambda+en+Java+8+)

 <https://web.archive.org/web/20161201070414/http://www.linkedin.com/shareArticle?mini=true&url=https://www.ecodeup.com/ejemplos-practicos-de-expresiones-lambda-en-java-8/>



Te gusta la Programación Web, Java y PHP?

Suscríbete ahora y recibe los mejores contenidos sobre Programación con Java y PHP en tu correo.

Nombre

Email

SUSCRIBIRME

Tus datos estarán protegidos y 100% libre de Spam

Elivar Largo

(<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/author/elargo/>)



(<https://web.archive.org/web/20161201070414/https://www.youtube.com/channel/UCViAh->

(<https://web.archive.org/web/20161201070414/https://ec.linkedin.com/in/elivar->
largo-">https://web.archive.org/web/20161201070414/https://ec.linkedin.com/in/elivar-
largo/

(<https://web.archive.org/web/20161201070414/http://facebook.com/elivarlargo>)

Programador. Blogger casi a tiempo completo y con unas cuantas características de Emprendedor. Apasionado por el Desarrollo Web y la Programación con Java.

Programación Java SE (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/category/programacion-java/>) API Stream (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/tag/api-stream/>), Ejemplos Lambda en Java (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/tag/ejemplos-lambda-en-java/>), Expresiones Lambda en Java (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/tag/expresiones-lambda-en-java/>)

Navegación de entradas

Cómo enviar parámetros a un Servlet desde una vista utilizando Post y Get (<https://web.archive.org/web/20161201070414/http://www.ecodeup.com/enviar-parametros-servlet-desde-una-vista-utilizando-post-get/>)

Deja un Comentario

Comentario

Nombre

Email

Sitio Web

Publicar Comentario

