

**FACULTAD DE INGENIERÍA DE SISTEMAS E INFORMATICA**

# ESTUDIOS GENERALES

# TEMA

EXPRESIONES LAMBDA Y STREAM EN JAVA 8

## CURSO DE:

PROGRAMACION I

**PROPÓSITO DEL TRABAJO: TRABAJO FINAL**

**PROFESOR: ERIC GUSTAVO CORONEL CASTILLO**

## AUTOR(ES):

Espinoza Ponciano Denis

Señas Sandoval Valeria

Ventocilla Gomero Fanny

## LOS OLIVOS – PERÚ

**2017**

**INTRODUCCIÓN**

Java 8 nos abre la puerta a la programación funcional con las expresiones lambda, también llamadas funciones anónimas, y la API Stream. Este tutorial no pretende entrar en profundidad en el tema si no ser un pequeño acercamiento a estas nuevas funcionalidades comparadas con la forma de realizarlas en programación imperativa. Para entrar más en profundidad, presentamos nuestra siguiente investigación.

**RESUMEN EJECUTIVO**

**Identificación del problema**

* ¿De qué forma podemos reducir código imperativo con la implementación de expresiones lambda y stream en java 8?

**Planteamiento de solución**

* Utilizando expresiones lambda y stream

**Cómo se va a implementar la solución**

* Usando La plataforma de desarrollo NetBeans

**Justificación de la investigación**

* Nuestra presente investigación está basada en la investigación de expresiones lambda y Stream en java 8, nosotros consideramos que es una herramienta bastante útil para el desarrollo de nuestra carrera la aplicación nos ayuda a desarrollar códigos de una manera más rápida y fácil por así decirlo nos brinda facilidades y el conocimiento de nuevas metodologías de aprendizaje de la programación computacional.

**OBJETIVO GENERAL**

* Implementar el uso de expresiones lambda y stream en java 8, para aplicarlo en futuros proyectos.

**OBJETIVOS ESPECÍFICOS**

* Mostraremos el uso de la api stream y la combinación ganadora lambdas + stream.
* Lograr la adición de expresiones lambda al lenguaje java 8.

**CAPITULO I**

**PROGRAMACIÓN FUNCIONAL**

**¿Qué es la Programación Funcional?**

Un programa funcional es una expresión simple que es ejecutada por evaluación de la expresión. La cuestión está en QUÉ va a ser computado, no en cómo va a serlo.

Otro lenguaje muy conocido, casi funcional es el lenguaje de consultas estándar de bases de datos, SQL. Una consulta SQL es una expresión con proyecciones, selecciones y uniones. Una consulta dice qué relación se debe computar sin decir cómo debe computarse. Además, la consulta puede ser evaluada en cualquier orden que sea conveniente

La programación funcional apareció como un paradigma independiente a principio de los sesenta.

Su creación es debida a las necesidades de los investigadores en el campo de la inteligencia artificial y en sus campos secundarios del cálculo simbólico, pruebas de teoremas, sistemas basados en reglas y procesamiento del lenguaje natural.

**Modelo Funcional**

El modelo funcional, tiene como objetivo la utilización de funciones matemáticas puras sin efectos.

El esquema del modelo funcional es similar al de una calculadora. Se establece una sesión interactiva entre sistema y usuario: el usuario introduce una expresión inicial y el sistema la evalúa mediante un proceso de reducción. En este procesos se utilizan las definiciones de funciones realizadas por el programador hasta obtener un valor no reducible. Estas necesidades no estaban cubiertas por los lenguajes imperativos de la época.

**EXPRESIONES LAMBDA EN JAVA 8**

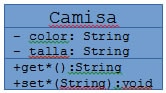
Las características más importantes de Java SE 8 son la adición de Expresiones Lambda y la API Stream. Con la adición de expresiones lambda podemos crear código más conciso y significativo, además de abrir la puerta hacia la programación funcional en Java, en donde las funciones juegan un papel fundamental. Por otro lado, la API Stream nos permite realizar operaciones de tipo filtro/mapeo/reducción sobre colecciones de datos de forma secuencial o paralela y que su implementación sea transparente para el desarrollador. Lambdas y Stream son una combinación muy poderosa que requiere un cambio de paradigma en la forma en la que hemos escrito código Java hasta el momento (Lopez, 2015).

**Funciones como entidades de primer nivel**

Uno de los conceptos de la programación funcional habla de que las funciones (métodos) sean definidas como entidades de primer nivel, es decir, que puedan aparecer en partes del código donde otras entidades de primer nivel, como valores primitivos u objetos, aparecen. Esto significa poder pasar funciones, en tiempo de ejecución, como valores de variables, valores de retorno o parámetros de otras funciones. Este es un concepto muy poderoso que se puede entender como la posibilidad de pasar comportamiento como valor y es precisamente lo que podemos lograr con la adición de expresiones lambda al lenguaje Java.

**Paso de comportamiento como valor/parámetro**

Para entender mejor el concepto de funciones como entidades de primer nivel, analicemos el siguiente caso con ayuda de la clase Camisa definida a continuación:



**Las expresiones lambda pueden clasificarse de la siguiente manera:**

* Consumidores.
* Proveedores.
* Funciones.
* Operadores Unarios.
* Operadores Binarios.
* Predicados.

Consumidores

Se trata de aquellas expresiones lambda que aceptan un solo valor y no devuelven valor alguno.

Proveedores

En este caso se trata de expresiones que no tienen parámetros pero devuelven un resultado.

Funciones

Aquellas expresiones que aceptan un argumento y devuelven un valor como resultado y cuyos tipos no tienen por qué ser iguales.

Predicados

Se trata de expresiones que aceptan un parámetro y devuelven un valor lógico.

Referencias a métodos

Las referencias a los métodos nos permiten reutilizar un método como expresión lambda. Para hacer uso de las referencias a métodos basta con utilizar la siguiente sintáxis: referenciaObjetivo:nombreDelMetodo.

Con las referencias a los métodos se ofrece una anotación más rápida para expresiones lambdas simples y existen 3 tipos diferentes:

Métodos estáticos.

Métodos de instancia de un tipo.

Métodos de instancia de un objeto existente.

**STREAM EN JAVA 8**

Stream es una nueva capa abstracta introducida en Java 8. Usando stream, puede procesar los datos de una forma declarativa similar a las sentencias SQL. Por ejemplo, considere la siguiente instrucción SQL:

SELECT max (salario), employee\_id, employee\_name FROM Empleado

La expresión SQL anterior devuelve automáticamente los detalles del empleado máximo asalariado, sin hacer ningún cálculo en el final del desarrollador. Usando el marco de colecciones en Java, un desarrollador tiene que usar bucles y hacer comprobaciones repetidas. Otra preocupación es la eficiencia; Como procesadores multi-núcleo están disponibles a gusto, un desarrollador de Java tiene que escribir el procesamiento de código paralelo que puede ser bastante propenso a errores.

Para resolver estos problemas, Java 8 introdujo el concepto de flujo que permite al desarrollador procesar los datos de forma declarativa y aprovechar la arquitectura multicore sin necesidad de escribir ningún código específico para ello.

**UNA EXPRESIÓN LAMBDA SE CARACTERIZA POR LA SIGUIENTE SINTAXIS:**

**Declaración de tipo opcional:** No es necesario declarar el tipo de parámetro. El compilador puede inferir lo mismo del valor del parámetro.

**Paréntesis opcional alrededor del parámetro:** No es necesario declarar un solo parámetro entre paréntesis. Para varios parámetros, se requieren paréntesis

**Dispositivos opcionales de rizo:** No es necesario utilizar rizadores en el cuerpo de la expresión si el cuerpo contiene una sola declaración.

**Opcional return keyword:** El compilador devuelve automáticamente el valor si el cuerpo tiene una sola expresión para devolver el valor. Las llaves curly se requieren para indicar que la expresión devuelve un valor.

**¿Qué es Stream?**

Stream representa una secuencia de objetos de una fuente, que admite operaciones agregadas. A continuación se presentan las características de una corriente -

Secuencia de elementos - Un flujo proporciona un conjunto de elementos de tipo específico de una manera secuencial. Una secuencia obtiene / calcula elementos bajo demanda. Nunca almacena los elementos.

**Origen -** El flujo toma colecciones, matrices o recursos de E / S como fuente de entrada.

Operaciones agregadas: Stream soporta operaciones agregadas como filtro, mapa, límite, reducción, búsqueda, coincidencia, etc.

Pipelining - La mayoría de las operaciones de flujo devuelven el flujo en sí para que su resultado pueda ser pipeline. Estas operaciones se denominan operaciones intermedias y su función es tomar entrada, procesarlas y devolver la salida al objetivo. El método collect () es una operación de terminal que normalmente está presente al final de la operación de pipelining para marcar el final del flujo.

Iteraciones automáticas - las operaciones de flujo realizan las iteraciones internamente sobre los elementos de origen proporcionados, en contraste con las colecciones en las que se requiere iteración explícita.

**Generación de flujos**

Con Java 8, la interfaz de la colección tiene dos métodos para generar un flujo -

Stream () - Devuelve un flujo secuencial considerando la colección como su fuente.

ParallelStream () - Devuelve un flujo paralelo considerando la colección como su origen. Stream representa una secuencia de elementos en la que se pueden realizar una o más operaciones. Las operaciones de flujo son intermedias o terminales. Mientras las operaciones de terminal devuelven un resultado de cierto tipo, las operaciones intermedias devuelven el flujo por sí mismo para que pueda encadenar múltiples llamadas de método en una fila. Los flujos se crean en una fuente, p. A java.util.Collection como listas o conjuntos (los mapas no son compatibles). Las operaciones de flujo se pueden ejecutar secuencialmente o en paralelo.

**VENTAJAS**

La ventaja de los streams es que pueden procesarse de forma serializada o paralela y proporcionan un estilo de operaciones más funcionales. Un flujo consiste en una fuente (una colección), varias operaciones intermedias (de filtrado o transformación) y una operación final que produce un resultado (suma, cuenta…). Los streams son lazy de modo que las operaciones solo se realizan cuando se llama a la operación final, también son eficientes no necesitando en algunos casos procesar todos los elementos del stream para devolver el resultado final.

La ventaja de las expresiones lambda tiene como las principales novedades que más se estaba echando de menos en Java de otros lenguajes como Groovy o Python.

**CONCLUSIONES**

En este trabajo de investigación no lo hemos enfocado en aprender con exactitud todos los entresijos de Java 8, sino más bien en servir de primer contacto a todos aquellos que quieran empezar a usar la programación funcional. Lo cierto es que hay otros lenguajes como Scala que ofrecen una programación funcional bastante más avanzada, pero como dije al principio, Java8 solo ha sido la primera puerta hacia la programación funcional.

Nos acerca a la programación funcional.

Hace nuestro código más preciso y legible, mejorando, en consecuencia, su mantenibilidad; Su utilización junto con la API Stream hace más fácil la ejecución concurrente de tareas.

**RECOMENDACIONES**

* Cuando se tiene un solo parámetro no es necesario utilizar los paréntesis.
* Cuando no se tienen parámetros, o cuando se tienen dos o más, es necesario utilizar paréntesis.
* Cuando el cuerpo de la expresión lambda tiene una única línea no es necesario utilizar las llaves y no necesitan especificar la cláusula return en el caso de que deban devolver valores.
* Cuando el cuerpo de la expresión lambda tiene más de una línea se hace necesario utilizar las llaves y es necesario incluir la cláusula return en el caso de que la función deba devolver un valor.

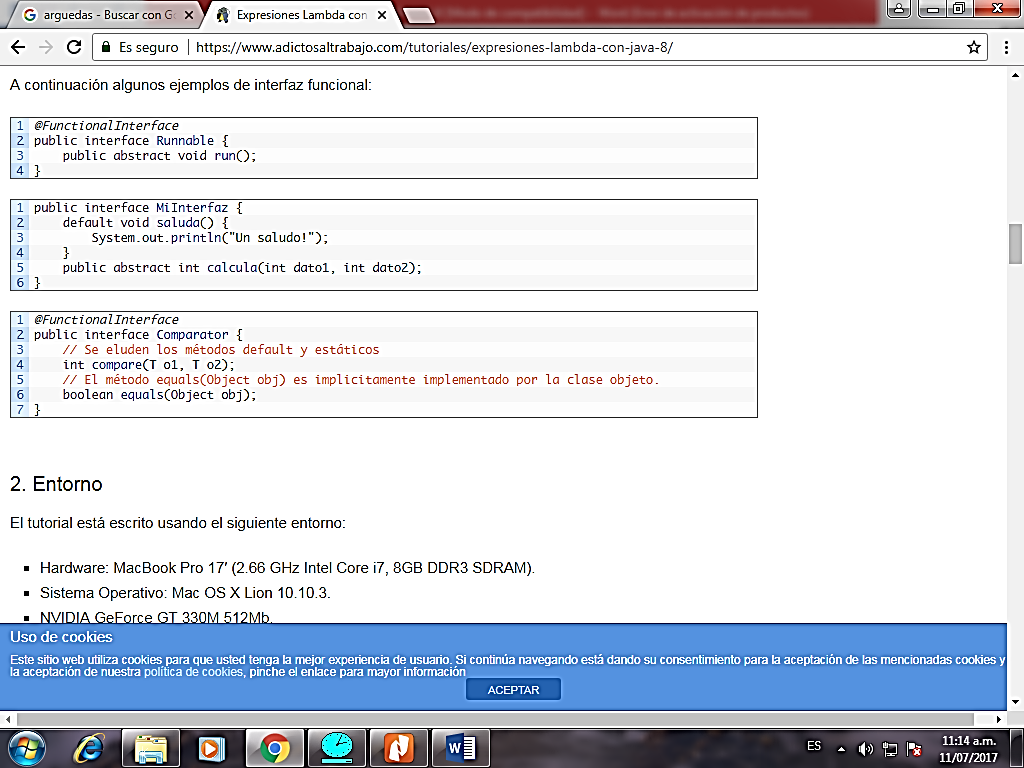
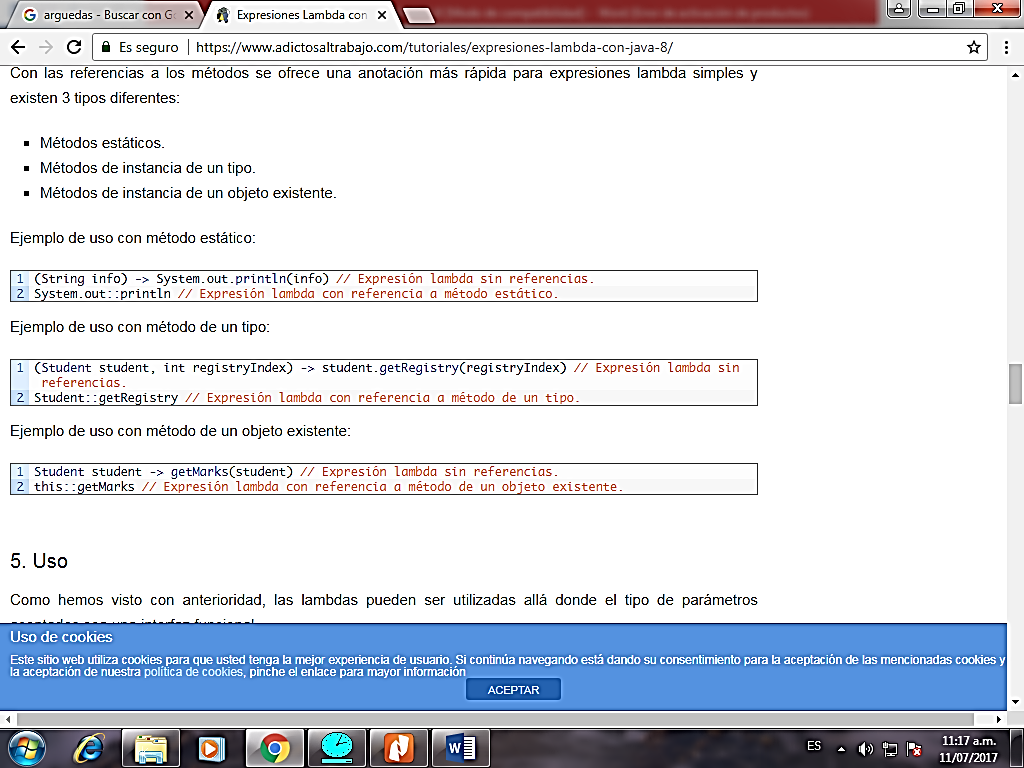
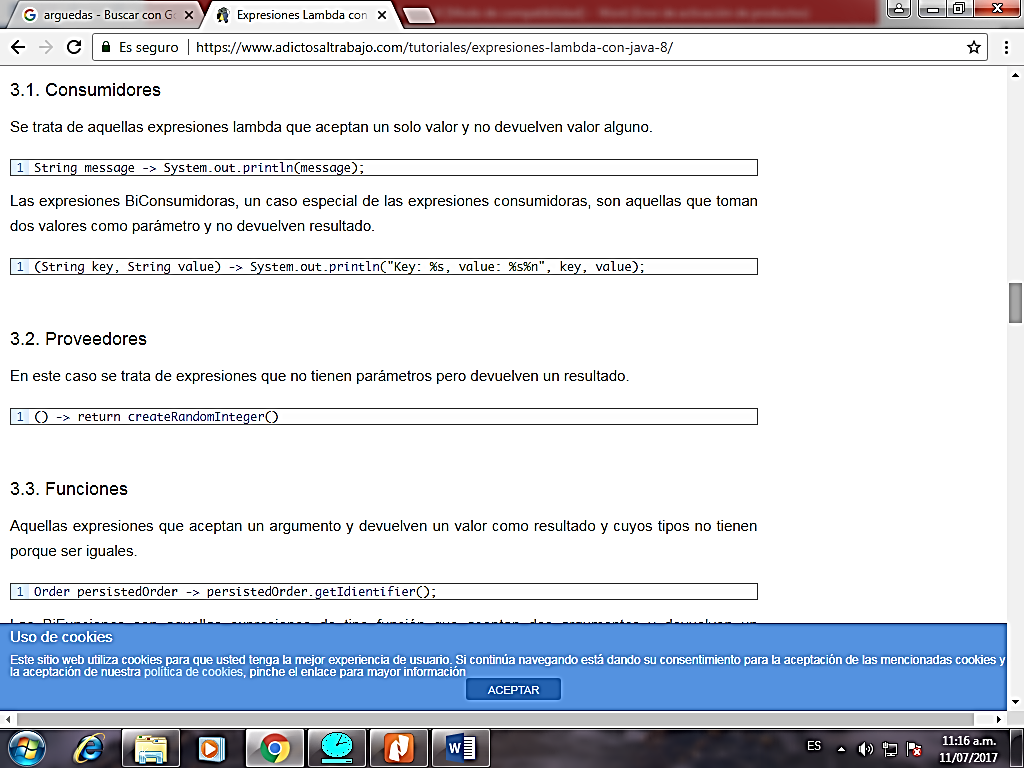
**REFERENCIAS BIBLIOGRÁFICAS**

Recuperado de: <http://winterbe.com/posts/2014/03/16/java-8-tutorial/>

Recuperado de: <https://www.adictosaltrabajo.com/tutoriales/uso-basico-de-java-8-stream-y-lambdas/>

Recuperado de: <https://www.adictosaltrabajo.com/tutoriales/expresiones-lambda-con-java-8/>

Rodríguez J., (2012) Expresiones Lambda con Java 8.

**ANEXOS**