Instituto tecnológico superior de jerez

Ingeniería en sistemas computacionales

Programación Lógica y Funcional

Modelo de programación funcional

Mapa conceptual y cuestionario

Docente: M.T.I. Salvador Acevedo Sandoval

Alumna: Viviana Michel Chávez Juárez

Numero de control: 14070013

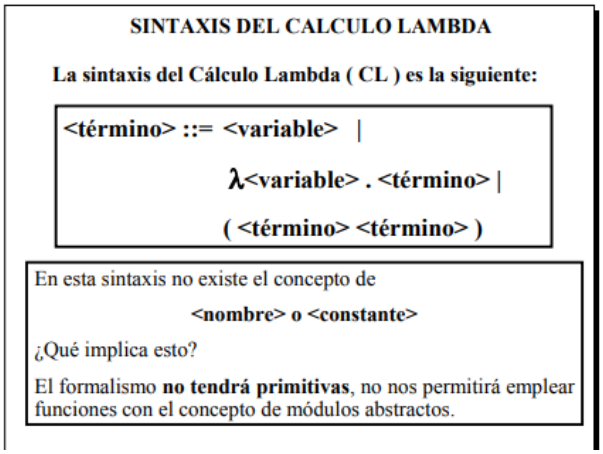
Fecha: 20/03/2020

Cuestionario

Lambda Expression. Java8

1. **Matemáticamente que es el cálculo lambda.**
   Es un sistema formal de lógica matemática para expresar la computación basada en función de la abstracción y la aplicación mediante la variable de unión y de sustitución.
2. **¿Que son las functional interfaces en java?**
   Una interfaz con un solo método abstracto
3. **¿Cuáles son las 6 interfaces funcionales del paquete java.util.function contains six basic functional interfaces?**
   - Functional Interfaces y Consumers
   - Java 8 Functions:El siguiente tipo de interface son los que habitualmente **se llaman Functions** reciben uno o más parámetros y retornan un resultado
   - Suppliers:Este caso es algo más complejo ya que se trata **de una función que no recibe parámetro alguna y devuelve un resultado**. Nos puede parecer algo absurdo de entrada ya que una función debe recibir algo. Sin embargo hay a veces que necesitamos **generar un nuevo contenido prácticamente desde la nada.**
   - Predicates: Los interfaces de tipo predicado son una especialización de los Functions ya que reciben un parámetro y devuelven un valor booleano.
   - Operators: El uso de Java 8 Operators es otro caso específico de los Function interfaces . Se trata de interfaces funcionales que **reciben un tipo de parámetro y devuelven un resultado que es del mismo tipo.**
4. **¿Que son las expresiones lambda?**
   *En el ámbito de la programación, una **expresión lambda**, también denominada función lambda, función literal o función anónima; es una subrutina definida que no está enlazada a un identificador.*
   Las **expresiones lambda** suelen utilizarse para lo siguiente:
   - Como argumentos que son pasados a otras funciones de orden superior.
   - Para construir el resultado de una función de orden superior que necesita retornar una función.

5. **Sintaxis de las expresiones lambda**



6. ¿Que son los streams y para qué sirven?

cLos Streams son una secuencia de elementos que soportan operaciones de agregación secuencial y paralela.

Stream se define como una secuencia de elementos que provienen de una fuente que soporta operaciones para el procesamiento de sus datos:

- De forma declarativa usando expresiones lambda.
- Permitiendo el posible encadenamiento de varias operaciones, con lo que se logra tener un código fácil de leer y con un objetivo claro.
- De forma secuencial o paralela (Fork/Join).

Las estructuras que soportan esta nueva API se encuentran en el paquete *java.util.stream* y en especial, la interface *java.util.stream.Stream* define un Stream.

La API nos permite realizar operaciones sobre colecciones de datos usando el modelo filtro/mapeo/reducción, en el cual se seleccionan los datos que se van a procesar (filtro), se convierten a otro tipo de dato (mapeo) y al final se obtiene el resultado deseado (reducción).

El siguiente ejemplo nos permite visualizar las tres (3) partes que componen un Stream:

- Fuente de información → La lista de transacciones
- Cero o más operaciones intermedias → Se puede apreciar la operación **filter** y la operación **mapToInt**, operaciones que serán explicadas más adelante en este mismo artículo.
- Operación terminal que produce un resultado o un efecto sobre los datos → Se puede apreciar la operación **sum**, la cual se explicará más adelante en este mismo artículo.

```
List transacciones = ...  int  sum = transacciones.stream().
filter(t ->  t.getProveedor().getCiudad().equals("Cali")).
mapToInt(Transaccion::getPrecio).
sum();
```

Ahora veamos algunas de las propiedades y características de un Stream:

- Las operaciones intermedias retornan otro Stream. Esto permite que podamos hacer un encadenamiento de operaciones .
- Las operaciones intermedias son encoladas hasta que una operación terminal sea invocada. Es muy importante tener esto claro, ninguna de las operaciones intermedias es ejecutada hasta que se invoca una operación terminal.
- Un Stream puede ser recorrido una sola vez, intentar recorrerlo de nuevo generará una excepción del tipo *IllegalStateException*.
- Al usar esta API estamos cambiando el paradigma **Iteración externa** vs. **Iteración interna**. En iteración interna, ésta sucede automáticamente permitiendo al desarrollador enfocarse en qué hacer con los datos y no en cómo hacerlo. Por otro lado, permite "esconder" complejidades y permite que la iteración secuencial o paralela sea transparente para el desarrollador.
- Imagina que los streams son una forma de iterar todos los elementos de la colección de una forma simple, y que además mientras los iteras puedes realizar operaciones sobre la colección de forma declarativa. Es decir, vamos a realizar operaciones pero centrándonos en el objetivo que buscamos, no en cómo implementar el algoritmo para que funcione. Veamos otro ejemplo más complejo.

Un Stream luce similar a una collection permitiendo además realizar operaciones directamente sobre el Stream. Cada operación devuelve un nuevo stream sobre el cual podemos seguir encadenando otras operaciones.

Funciones más relevantes de la clase Streams

Calsificacion:

- Streams
- Parallel Streams

Tipos de operaciones:

Intermedias

- Map:Altera los segmentos en base a una función y los devuelve alterados.
- Filter: filtra los elementos a partir de una condición de la función pasada como parámetro.
- Sorted: Ordena los elementos siguiendo la condición de la función como parámetro.
- Distinct: Retorna un stream con elementos no repetidos basándose en la comparación entre objetos con la función equals.
- Peek: Esta operación no realiza cambios sobe el stream y lo devuelve tal y como entra. Su propósito consiste en hacer debugging al ejecutar cualquier otra operación, ya que permite imprimir valores de los elementos del stream.
- Limit: Limita el número de elementos que tiene como salida el nuevo stream.

Terminales

- forEach: Realizar la acción de la función por parámetro para cada elemento.
- Collect: Ejecuta lo que denominan mutable reduction, que consiste en acumular los resultados en una collection a medida que los va procesando en la pipeline del stream.
- Reduce: Permite hacer una reducción de los resultados del stream,lo cual consiste en acumular el resultado en un resultado resumido de la entrada,por ejemplo,encontrar la suma de un stream de enteros. También otras operaciones que utilizan la operación reduce en brackground y son terminales, como sum();

Functional Interfaces

Functional interfaces en java

¿Que son?

Interfaz con un solo método abstracto.

Interfaces funcionales del paquete java.util.Function contains six basic functional interfaces

Functions

Recibe uno o mas parametros y retornan un resultado.

Suppliers

No recibe parámetros y devuelve un resultado.

Predicates

Reciben un parámetro y devuelven un valor booleano.

Operators

Reciben un tipo de parámetro y devuelven un resultado que es del mismo tipo.

Expresiones lambda

¿Que son?

Matemática

Sistema formal de lógica matemática para expresar la computación basada en función de la abstracción y la aplicación mediante la variable de unión y de sustitución.

Programación

¿Que es?

Subrutina definida que no está enlazada a un identificador.

Se utilizan

Como argumentos que son pasados a otras funciones de orden superior.

Para construir el resultado de una función de orden superior que necesita retornar una función.

Sintaxis

SINTAXIS DEL CALCULO LAMBDA

La sintaxis del Cálculo Lambda (CL) es la siguiente:

<término> ::= <variable> |

λ<variable> . <término> |

(<término> <término>)

En esta sintaxis no existe el concepto de

<nombre> o <constante>

¿Qué implica esto?

El formalismo **no tendrá primitivas**, no nos permitirá emplear funciones con el concepto de módulos abstractos.

Streams

¿Que son?

Son una secuencia de elementos que soportan operaciones de agregación secuencial y paralela.

Características

Puede ser recorrido una sola vez.

Las operaciones intermedias retornan otro Stream.

Ninguna de las operaciones intermedias es ejecutada hasta que se invoca una operación terminal.

¿ Para qué sirven ?

Permitiendo el posible encadenamiento de varias operaciones, con lo que se logra tener un código fácil de leer y con un objetivo claro

Funciones

Intermedias

Map

Altera los segmentos en base a una función y los devuelve alterados.

Filter

Filtra los elementos a partir de una condición de la función pasada como parámetro.

Sorted

Ordena los elementos siguiendo la condición de la función como parámetro

Distinct

Retorna un stream con elementos no repetidos basándose en la comparación entre objetos con la función equals.

Peek

Esta operación no realiza cambios sobe el stream y lo devuelve tal y como entra. Su propósito consiste en hacer debugging al ejecutar cualquier otra operación, ya que permite imprimir valores de los elementos del stream.

Limit

Limita el número de elementos que tiene como salida el nuevo stream.

Terminales

forEach

Permite hacer una reducción de los resultados del stream,lo cual consiste en acumular el resultado en un resultado resumido de la entrada.

Collect

Ejecuta lo que denominan mutable reduction, que consiste en acumular los resultados en una collection a medida que los va procesando en la pipeline del stream.

Reduce

Realizar la acción de la función por parámetro para cada elemento.

http://www.rinconmatematico.com/foros/index.php?action=dlattach;topic=19069.0;attach=4367

https://www.arquitecturajava.com/java-8-functional-interfaces-y-sus-tipos/

https://www.tokioschool.com/noticias/expresiones-lambda-uso-programacion-aplicaciones/

https://www.oracle.com/technetwork/es/articles/java/expresiones-lambda-api-stream-java-2737544-esa.html

https://www.oscarblancarteblog.com/2017/03/16/java-8-streams-2/

https://codingfactsblog.wordpress.com/2017/08/01/jugando-con-streams-y-predicates-en-java/

https://es.qwe.wiki/wiki/Lambda_calculus