Expresiones Lambdas

Pablo Borrego Gutiérrez

14 de Mayo de 2018

Expresiones Lambdas DEFINICIÓN

Función anónima, no está asociada a un identificador

Expresiones Lambdas DEFINICIÓN

Función anónima, no está asociada a un identificador

Función típica

```
public void tomarRebujito(Double cantidad) {
      System.out.println(
            String.format("Tomando %s rebujitos", cantidad)
      );
}
```

Expresiones Lambdas DEFINICIÓN

Función anónima, no está asociada a un identificador

Función típica

```
public void tomarRebujito(Double cantidad) {
      System.out.println(
            String.format("Tomando %s rebujitos", cantidad)
      );
}
```

Función lambda

```
(cantidad) -> System.out.println(String.format("Tomando %s rebujitos", cantidad));
```

Expresiones Lambdas DEFINICIÓN

Función anónima, no está asociada a un identificador

Función típica

```
public void tomarRebujito(Double cantidad) {
      System.out.println(
            String.format("Tomando %s rebujitos", cantidad)
      );
}
```

Función lambda

```
(cantidad) -> System.out.println(String.format("Tomando %s rebujitos", cantidad));
```

Tipos de Lambdas en la JDK 8 de Java:

```
// Sin parámetros
() -> System.out.println("Hello, world.")
// Con un parámetro (este ejemplo es una función de identidad).
a -> a
// Con una expresión
(a, b) -> a + b
// Con un bloque de código
(a, b) -> { return a + b; }
// Con múltiples afirmaciones en el cuerpo lambda. Necesita un bloque de código.
// Este ejemplo también incluye dos expresiones lambda anidadas (la primera también es una clausura).
(id, defaultPrice) -> {
  Optional<Product> product = productList.stream().filter(p -> p.getId() == id).findFirst();
  return product.map(p -> p.getPrice()).orElse(defaultPrice);
}
```

Ejemplo 1 PROGRAMACIÓN CONCURRENTE

```
executor.submit(new Runnable() {
	@Override
	public void run() {
		try { Thread.sleep(random(2000)); } catch (InterruptedException e) {}
		System.out.println(String.format("Ejecutandome en el hilo [%s]", Thread.currentThread().getName()));
	}
});
```

Ejemplo 1 PROGRAMACIÓN CONCURRENTE

```
executor.submit(new Runnable() {
	@Override
	public void run() {
		try { Thread.sleep(random(2000)); } catch (InterruptedException e) {}
		System.out.println(String.format("Ejecutandome en el hilo [%s]", Thread.currentThread().getName()));
	}
});
```

```
executor.submit(() -> {
	try { Thread.sleep(random(2000)); } catch (InterruptedException e) {}
	System.out.println(String.format("Ejecutandome en el hilo [%s]", Thread.currentThread().getName()));
});
```

Ejemplo 1.2 ANDROID

```
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something here
    }
});
```

Ejemplo 1.2 ANDROID

```
mButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        // do something here
    }
});
```

```
mButton.setOnClickListener((View v) -> {
    // do something here
});
```

Expresiones Lambdas STREAM API

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

- Se usan expresiones lambdas.
- Es posible concatenar diferentes operaciones entre sí.
- Se puede ejecutar de forma secuencial o paralela.

Algunas operaciones

- AllMatch
- AnyMatch
- Collect
- Count
- Filter
- FindFirst
- ForEach
- Map
- Sorted

Ejemplos

```
Boolean bool = ...AllMatch(lp -> lp.getPrecio > 5);
Boolean bool = ...AnyMatch(lp -> lp.getPrecio > 5);
List<LineaPedido> lp = ...collect(Collectors.toList());
Long total = ...count();
List<LineaPedido> lp = ...filter(lp -> lp.getPrecio > 5)...
Optional<LineaPedido> lp = ...findFirst();
pedidos.forEach(lp -> System.out.println(lp));
pedidos.forEach(lp -> System.out.println(lp));
List<LineaPedido> lp = ...sorted()...
```

Ejemplo 2 API STREAM

```
public class LineaPedido {

        ...

        private String producto;

        private Integer cantidad;

        private Double precio;

        ...
}
```

Ejemplo 3 TIEMPO DE EJECUCIÓN MÉTODO

```
public static Double calcularTotal(List<LineaPedidoDto> pedidos) {
      return pedidos.stream().mapToDouble(lineaPedido -> lineaPedido.getPrecio() * lineaPedido.getCantidad()).sum();
}
```

Ejemplo 3 TIEMPO DE EJECUCIÓN MÉTODO

```
public static Double calcularTotal(List<LineaPedidoDto> pedidos) {
	return pedidos.stream().mapToDouble(lineaPedido -> lineaPedido.getPrecio() * lineaPedido.getCantidad()).sum();
}
```

```
public static Double avgExecutionTime(Runnable function) {
	Long TOTAL_EXECUTIONS = 1000000L;
	Double totalTimeExecution = 0D;
	for(int i = 0; i < TOTAL_EXECUTIONS; ++i) {
		Long tiempoInicio = System.nanoTime();
		function.run();
		totalTimeExecution += (System.nanoTime() - tiempoInicio);
	}
	return Double.valueOf(totalTimeExecution/TOTAL_EXECUTIONS)/1000;
}
```

Ejemplo 3 TIEMPO DE EJECUCIÓN MÉTODO

```
public static Double calcularTotal(List<LineaPedidoDto> pedidos) {
	return pedidos.stream().mapToDouble(lineaPedido -> lineaPedido.getPrecio() * lineaPedido.getCantidad()).sum();
}
```

```
public static Double avgExecutionTime(Runnable function) {
	Long TOTAL_EXECUTIONS = 1000000L;
	Double totalTimeExecution = 0D;
	for(int i = 0; i < TOTAL_EXECUTIONS; ++i) {
		Long tiempoInicio = System.nanoTime();
		function.run();
		totalTimeExecution += (System.nanoTime() - tiempoInicio);
	}
	return Double.valueOf(totalTimeExecution/TOTAL_EXECUTIONS)/1000;
}
```

```
System.out.println(String.format("Total: %s ms", avgExecutionTime(() -> calcularTotal(pedidos))));
```

Gracias

¿Alguna pregunta?