Tarea Independiente 11/08/2025

David Núñez Franco

August 13, 2025

Inventario de Conceptos Claves

- Lambda y el cálculo lambda de Church
- Tipos en tiempo de compilación
- Tipos en tiempo de ejecución
- Lambdas en Java
- Método versus lambda en Java
- El rol del Typer en Java
- Lenguaje estáticamente tipado versus dinámicamente tipado
- AST (equivalente al árbol de expresión de Estructuras Discretas)
- Tablas de símbolos
- Typer de Java
- lambda versus método en Java
- Tipo genérico
- Método como parámetro de tipo genérico
- Extra: preprocesador, macros y templates en C++

Ejercicio 1

a) Considere esta función add en JS:

```
1  const add = (f, g) => x => f(x) + g(x)
```

Pruebe que da lo mismo add(add(f, g), h) que add(f, add(g, h)) es decir, add es asociativa.

b) Construya una lambda zero tal que, add() = 0, add(zero, f) dé igual que add(f, zero) y dé igual que f para cualquier lambda f.

c) Construya una función max_function(f, g) que cumpla: max_function(f,g)(x) = max(f(x), g(x)) para todo x de tipo number. dode max(x, y) calcula el máximo entre cualesquiera números x y y.

Solución punto 1.a

Ejercicio demostrado en el cuaderno de práctica.

Solución punto 1.b

Dicho problema pide construir una función zero que sea el elemento neutro de la operación add. Debe cumplir que para cualquier funcion f:

- add(zero, f) = f
- add(f, zero) = f

Además, add(zero, zero) produce siempre 0 + 0 = 0

```
// DEFINIMOS ADD COMO LAMBDA
const add = (f, g) => (x) => f(x) + g(x);

// DEFINIMOS ZERO COMO LAMBDA. RETORNA CERO
const zero = (x) => 0;

// EJEMPLO DE FUNCIONES
const f = (x) => 2 * x;
const g = (x) => x + 1;

// PROBAMOS
console.log(add(zero, f)(5)); // resultado es 10 = 2 * 5
console.log(add(f, zero)(5)); // resultado es 10 = 2 * 5
console.log(add(zero, zero)(5)); // resultado es 0
```

Listing 1: Solucion en JavaScript

Solución punto 1.c

La función max_function devuelve el mayor de los números que recibe. Luego, se aplica a los valores calculados por f, g para el mismo x. El resuktado será una nueva función que, dado un número, siempre entrega el mayor de f(x) y g(x).

```javascript
// max_function retorna una funcion que calcula el maximo de f(x) y g(x)
const max_function_with_math = (f, g) => (x) => Math.max(f(x), g(x));
const max_function_without_math = (f, g) => (x) => f(x) >= g(x) ? f(x) : g(x);

// construimos algunas funciones
const f = (x) => 2 * x;
const g = (x) => x + 5;

// probamos
console.log(max_function_with_math(f, g)(2)); // resultado es 7, Math.max(4,7)
console.log(max_function_with_math(f, g)(10)); // resultado es 20, Math.max(20,15)
```

Listing 2: Solución en JavaScript

Ejercicio 2

Haga lo equivalente que 1 pero en Java.

Solución punto 2.b

```java
import java.util.function.Function;

public class Ejercicio2B
{
    // add: suma de funciones
    public static Function<Integer, Integer> add(Function<Integer, Integer> f, Function<Integer, Integer> g) {
        return x -> f.apply(x) + g.apply(x);
    }

    // zero: elemento neutro
    public static Function<Integer, Integer> zero = x -> 0;

    public static void main(String[] args) {
        // creamos las funciones
        Function<Integer, Integer> f = x -> 2 * x;
        Function<Integer, Integer> g = x -> x + 1;

        // prueba
        System.out.println(add(zero, f).apply(5)); // 10
```

```
        System.out.println(add(f, zero).apply(5)); // 10
        System.out.println(add(zero, zero).apply(5)); // 0
    }
}
```

Listing 3: Solución en Java

Solución punto 2.c

```
import java.util.function.Function;

public class Ejercicio2C {
    // max_function utilizando Math.max
    public static Function<Integer, Integer>
       maxFunctionWithMath(Function<Integer, Integer> f,
       Function<Integer, Integer> g) {
        return x -> Math.max(f.apply(x), g.apply(x));
    }

    // max_function sin Math.max
    public static Function<Integer, Integer>
       maxFunctionWithoutMath(Function<Integer, Integer> f,
       Function<Integer, Integer> g) {
        return x -> f.apply(x) >= g.apply(x) ? f.apply(x) :
           g.apply(x);
    }

    public static void main(String[] args) {
        // construimos las funciones
        Function<Integer, Integer> f = x -> 2 * x;
        Function<Integer, Integer> g = x -> x + 5;

        // pruebas maxFunctionWithMath
        System.out.println(maxFunctionWithMath(f, g).apply
           (2));   // 7
        System.out.println(maxFunctionWithMath(f, g).apply
           (10));  // 20

        // prueba maxFunctionWithoutMath
        System.out.println(maxFunctionWithoutMath(f, g).
           apply(2));   // 7
        System.out.println(maxFunctionWithoutMath(f, g).
           apply(10));  // 20
    }
}
```

Listing 4: Solución en Java

Ejercicio 3

Traduzca a Java el siguiente código

```
template <typename T> T max(T a, T b) {
    return (a > b) ? a : b;
}
int main() {
    int x = 5, y = 10;
    std::cout << max(x, y) << std::endl;
    double a = 666.5, b = 665.8;
    std::cout << max(a, b) << std::endl;
    return 0;
}
```

Solución

Primero, debemos entender el problema. Utilizaremos genéricos con restricciones.

`<T extends Comparable<? super T>>` exige que el tipo T implemente Comparable, accediendo al método compareTo para establecer un orden.

`a.compareTo(b)` devuelve

- valor positivo si a mayor que b
- 0 si son iguales
- valor negativo si a menor que b

```
public class main {
    /*T extends Comparable<? super T> permite que T use
       compareTo, para compararse con su tipo o super-tipos.
       */
    public static <T extends Comparable<? super T>> T max(T
       a, T b) {
        /*compareTo devuelve:
        * 1. Un numero positivo si a > b
        * 2. Un cero si a == b
        * 3. Un numero negativo si a < b*/
        return (a.compareTo(b) > 0) ? a : b;
    }

    public static void main(String[] args) {
        Integer x = 5, y = 10;
        System.out.println(max(x, y));

        Double a = 666.5, b = 665.8;
        System.out.println(max(a, b));
    }
```

```
18          }
```

Listing 5: Solución en Java, utilizando genéricos con restricciones

Ejercicio 4

Investigue los términos aplicables a lenguajes de programación "fuertemente tipado" versus "débilmente tipado". Esté en capacidad de explicarlo con ejemplos.

Fuertemente Tipado vs Débilmente Tipado

- **Fuertemente tipado**: El lenguaje no permite usar valores de tipos incompatibles, sin conversión explícita. Las operaciones inválidas entre tipos fallan en compilación o en tiempo de ejecución.
- **Débilmente tipado**: El lenguaje aplica conversiones implícitas (coerción) incluso entre tipos distintos, lo que puede dar resultados inesperados.

Ejemplos

Fuertemente tipado

```
1           int x = 5;
2           String y = "10";
3           int z = x + y; // ERROR DE COMPILACION
```

Listing 6: Java

```
1           5 + "10" # TypeError: unsupported operand type(s)
```

Listing 7: Python, tipado dinámico pero fuerte

Débilmente tipado

```
1           5 + "10" // "510" (concatena)
2           "5" - 1 // 4 (convierte "5" a numero)
3           [] == 0 // true
```

Listing 8: JavaScript

Relación con tipado estático/dinámico

- **Tipado fuerte/débil**: Se refiere a cuánta coerción implícita permite el lenguaje.
- **Tipado estático/dinámico**: Se refiere a si el tipo se verifica en compilación o en ejecución.

Lenguaje	**Tipado**	**Comprobación**
Java	Fuerte	Estático
Python	Fuerte	Dinámico
JavaScript	Débil	Dinámico
C	Débil	Estático

Table 1: Comparación de tipado y comprobación en distintos lenguajes