educación IT
CODE YOUR FUTURE

# Bootcamp Java Developer

## Fase 1 - Java Analyst

Módulo 11

educación IT
CODE YOUR FUTURE

# Programación funcional

# Java 8

En 2014, Java lanza una versión que añade **novedades importantes al lenguaje** y divide el antes y después de esta versión. Las novedades más importantes que podemos destacar son:

- El agregado del **paquete *stream*.**
- La implementación del **paradigma de programación funcional**.

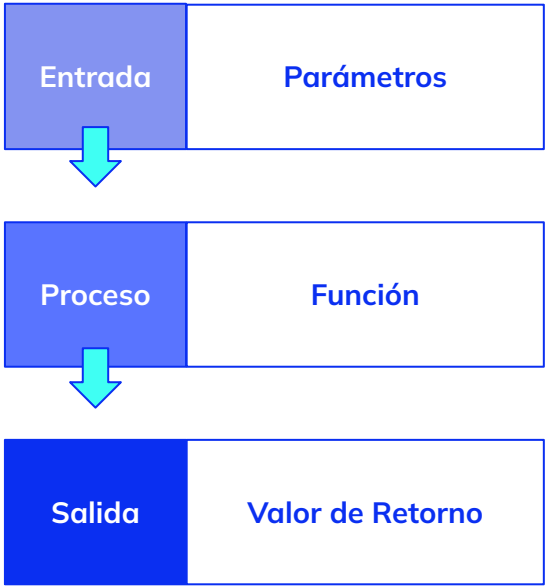
Ya habíamos hablado de los **métodos por defecto y estáticos en la interfaces** que se agregaron en esta versión y son muy importantes para las otras funcionalidades que vamos a ver en este apartado.

# Paradigma funcional

Recordemos que un paradigma no es más que una **forma de resolver un problema planteado desde un enfoque diferente**.

Este paradigma es **declarativo**, por lo que nos enfocaremos en **el qué** y no en cómo se está resolviendo el problema, es decir, expresaremos nuestra lógica sin describir controles de flujo; ciclos y/o condicionales.

La programación orientada objetos es imperativa que no es más que determinar el control de flujo (Ciclos y Condicionales) de nuestro software.



# Interfaz funcional

Son aquellas **interfaces que poseen si y solo si un solo método abstracto (sin cuerpo).**

Pueden poseer o no la notación **`@FunctionalInterface`**, que previene que otro desarrollador agregue más métodos abstractos generando errores en tiempo de compilación.

Adicionalmente, pueden tener métodos por defecto y estáticos sin restricciones.

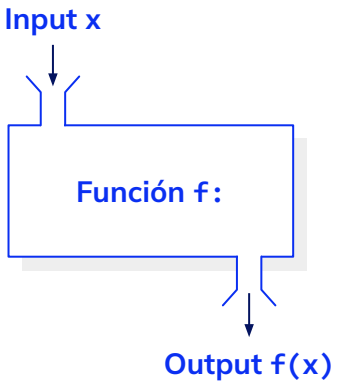
```java
@FunctionalInterface
public interface interfaz {

    retorno metodo(tipo parametros);

}
```

# Lambdas

Para poder implementar el paradigma funcional se agrega el concepto de ***lambdas*** que es un método anónimo. Básicamente, es un **método abstracto que sólo está definido en una interfaz (no necesita una clase) y esta debe ser una interfaz funcional**.



# Estructura Lambda

### Parámetros:

- 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 usar paréntesis.
- Está implícito el tipo de objeto que va a utilizar por lo que no es necesario declararlo en el parámetro.

### El operador lambda:

- **Operador flecha** (**->**) que separa la declaración de parámetros de la declaración del cuerpo de la función.

**Cuerpo:**

- Cuando el cuerpo de la expresión lambda tiene una **única línea** no es necesario utilizar las llaves y no se necesita especificar la sentencia `return` en el caso de que se 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 sentencia `return` en el caso de que la función deba devolver un valor.

# Ejemplo imperativo

```java
public interface CalculadoraTradicional {

    public static int sumar(int a, int b) {
        return a + b;
    }

    public static int restar(int a, int b) {
        return a - b;
    }

    public static int multiplicar(int a, int b) {
        return a * b;
    }

    public static int dividir(int a, int b) {
        return (b == 0 ? 0 : a / b);
    }
}
```

```java
int a = 4;
int b = 2;
int resultado;

resultado = CalculadoraTradicional.sumar(a, b);
System.out.println("Sumar (" + a + " , " + b + "): " + resultado);
resultado = CalculadoraTradicional.restar(a, b);
System.out.println("Restar (" + a + " , " + b + "): " + resultado);
resultado = CalculadoraTradicional.multiplicar(a, b);
System.out.println("Multiplicar (" + a + " , " + b + "): " + resultado);
resultado = CalculadoraTradicional.dividir(a, b);
System.out.println("Dividir (" + a + " , " + b + "): " + resultado);
```

# Ejemplo declarativo

A simple vista, parece lo mismo o hasta más complicado, pero la idea es poder **cambiar el comportamiento de los métodos en el momento que necesitemos sin declararlos previamente**.

En este ejemplo, cambiamos el comportamiento al método para hacer las operaciones que hicimos en el ejemplo imperativo y, de forma adicional, comportamiento necesario para que retorne el módulo. También podemos seguir agregando sin problemas.

```java
@FunctionalInterface
public interface CalculadoraFuncional {
    int operar(int a, int b);
}
```

```
int a = 4;
int b = 2;
int resultado;

CalculadoraFuncional operaracion;

operaracion = (numA, numB) -> numA + numB;
resultado = operaracion.operar(a, b);
System.out.println("Sumar (" + a + " , " + b + "): " + resultado);

operaracion = (numA, numB) -> numA - numB;
resultado = operaracion.operar(a, b);
System.out.println("Restar (" + a + " , " + b + "): " + resultado);

operaracion = (numA, numB) -> numA * numB;
resultado = operaracion.operar(a, b);
System.out.println("Multiplicar (" + a + " , " + b + "): " + resultado);

operaracion = (numA, numB) -> (numB == 0 ? 0 : numA / numB);
resultado = operaracion.operar(a, b);
System.out.println("Dividir (" + a + " , " + b + "): " + resultado);

operaracion = (numA, numB) -> numA % numB;
resultado = operaracion.operar(a, b);
System.out.println("Modulo (" + a + " , " + b + "): " + resultado);
```

educación IT
CODE YOUR FUTURE

# ¡Sigamos trabajando!