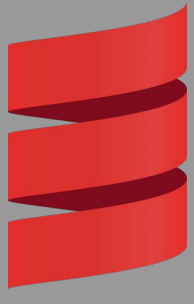
Programación **Funcional** en **Java SE**

Introducción a la **programación funcional**

¿Qué es la **programación funcional?**


ERLANG

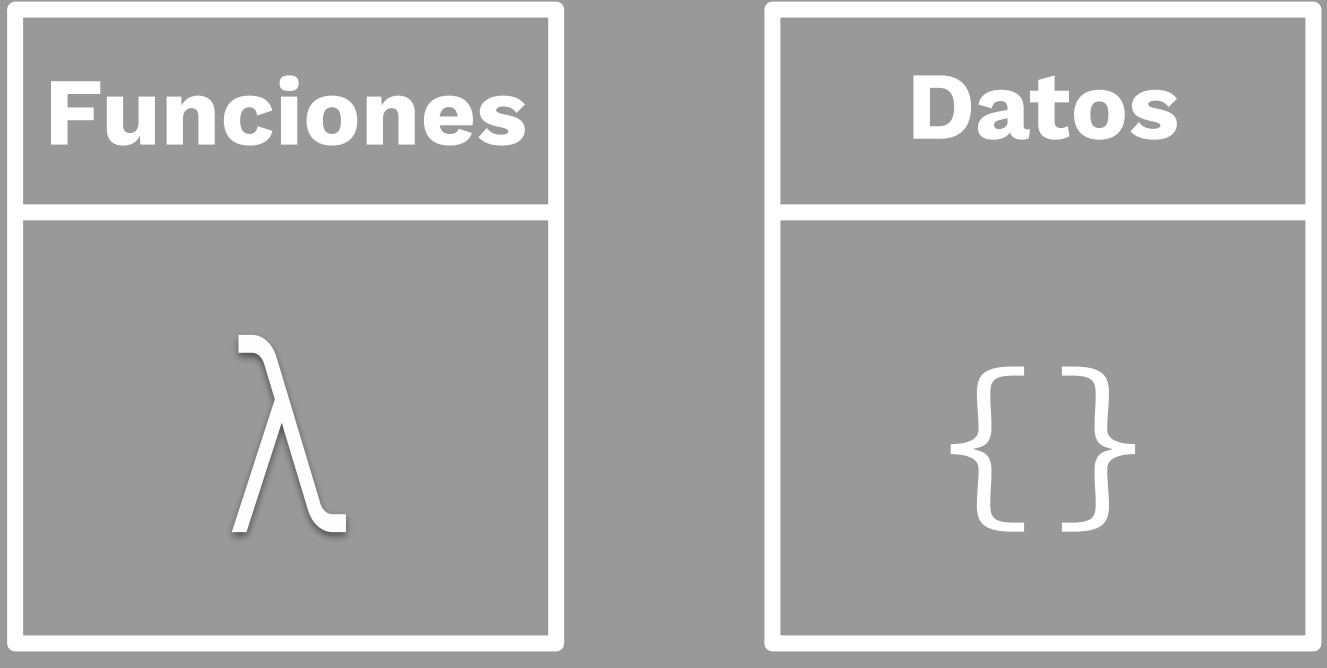



JS


Java™

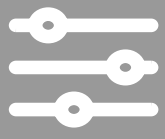

ERLANG


JS


Funciones
λ
Datos
{ }

Legibilidad

Testing

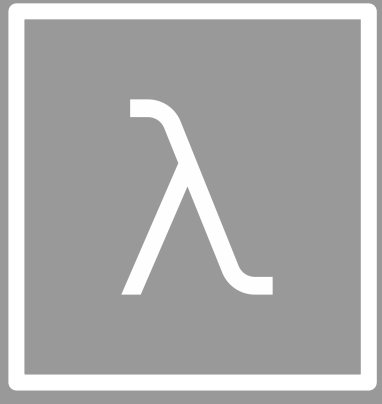


Concurrencia

Comportamientos
más definidos

Menos manejo
de estados

No hay que instalar
nada adicional



Funciones

¿Qué es una función?

X	Y
x1	y1
x2	y2
x3	y3
…	…

Un tipo de dato u objeto que toma un valor *X* y genera un valor *Y*.

Idealmente, por cada X siempre genera una Y.

- Es una **serie de pasos parametrizados.**
- Puede o no **devolver un resultado.**
- Se puede **definir, almacenar o declarar bajo demanda** (como cualquier otro tipo).

$f(x) = x^2 + 5$

```
double parabola(double x) {
    return Math.pow(x, 2) +
5;
}
```

Podemos definir funciones con respecto a otras funciones.

```
esPar(x) =
!esNon(x)
```

Podemos definir funciones con respecto a si mismas (recursividad).

```
factorial(x) = if x <= 2 :
x
                  else : x *
factorial(x-1)
```

Podemos definir funciones que tomen otras funciones como parámetros.

$$f(x, g(x)) = x^2 * g(x)$$

Ciudadanos de primera clase

Tomarlos como parámetros o retornos

```
int
foo(Function
param)

Function
bar(int x)
```

Cómo definir tipos

```
Function x;

Function sum =
…
```

Definirlos bajo demanda

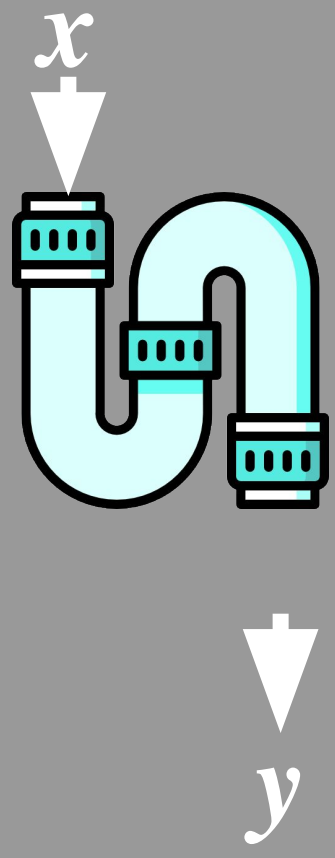
```
//Definiendo un String
funWithString(“Hi”)

// Una funcion (por ahora)
plotWithFunction(x -> x * 2)
```

No se obtiene de una variable o algún otro lado

Funciones “Puras” (Pure functions)

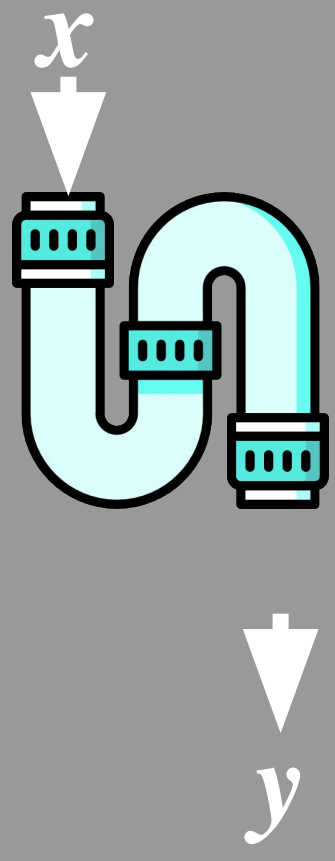
Funciones puras

- ✅ Generan el mismo resultado para el mismo parámetro
- ✅
  - `sum(5, 3)`
    `//Siempre`
    `será 8`



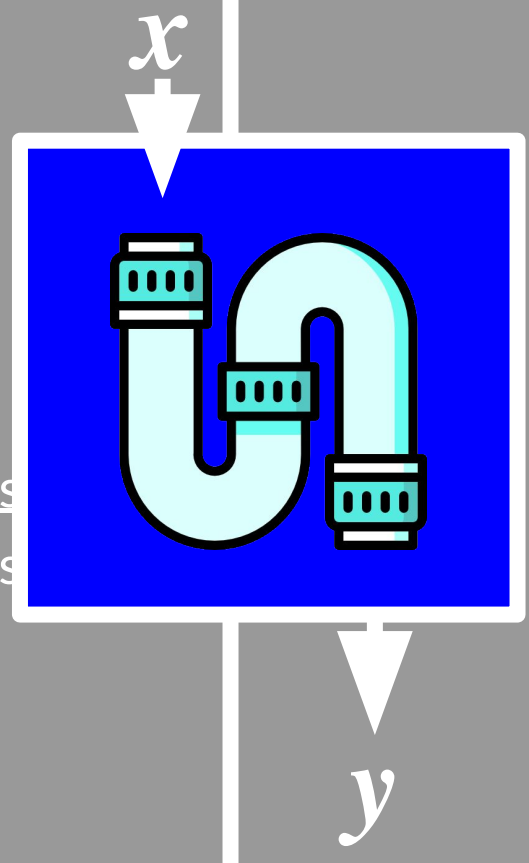
Funciones puras

- ✅ Funcionan en aislamiento (no dependen ni las afecta un contexto)
- ✅ Son determinísticas



Funciones puras

- ✅ Por ejemplo: $f(x) = x^3$
  - ⓧ No genera valores aleatorios
    - ✅ $f(3) \rightarrow$ siempre será 27
  - ⓧ No cambia el valor de x
  - ⓧ No genera efectos secundarios
    - ⓧ No cambia una base de datos
    - ⓧ No crea un archivo
    - ⓧ No modifica el sistema



Una función pura en Java:

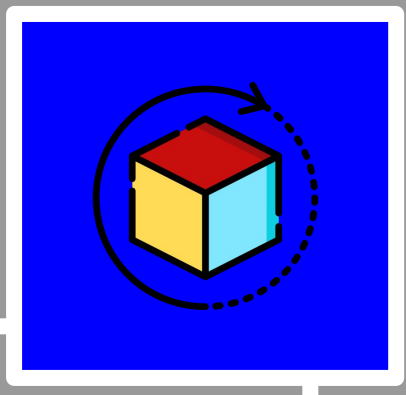
```java
boolean hasAvailableFound(double balance) {
    return balance > 0.0;
}
```

A las funciones que no son puras se les conoce como **funciones impuras.** Las reglas dictan que:

Funcion	Puede invocar: **Pura**	Puede invocar: **Impura**
Pura	✓	✗
Impura	✓	✓

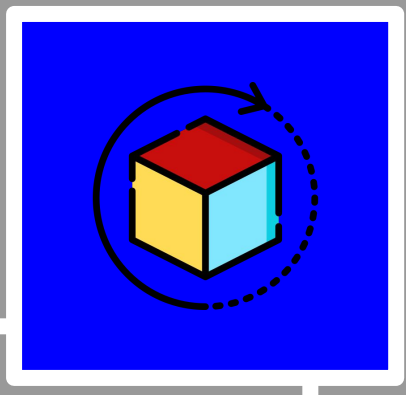
Efectos secundarios

“Todo cambio observable desde fuera del sistema es un efecto secundario”

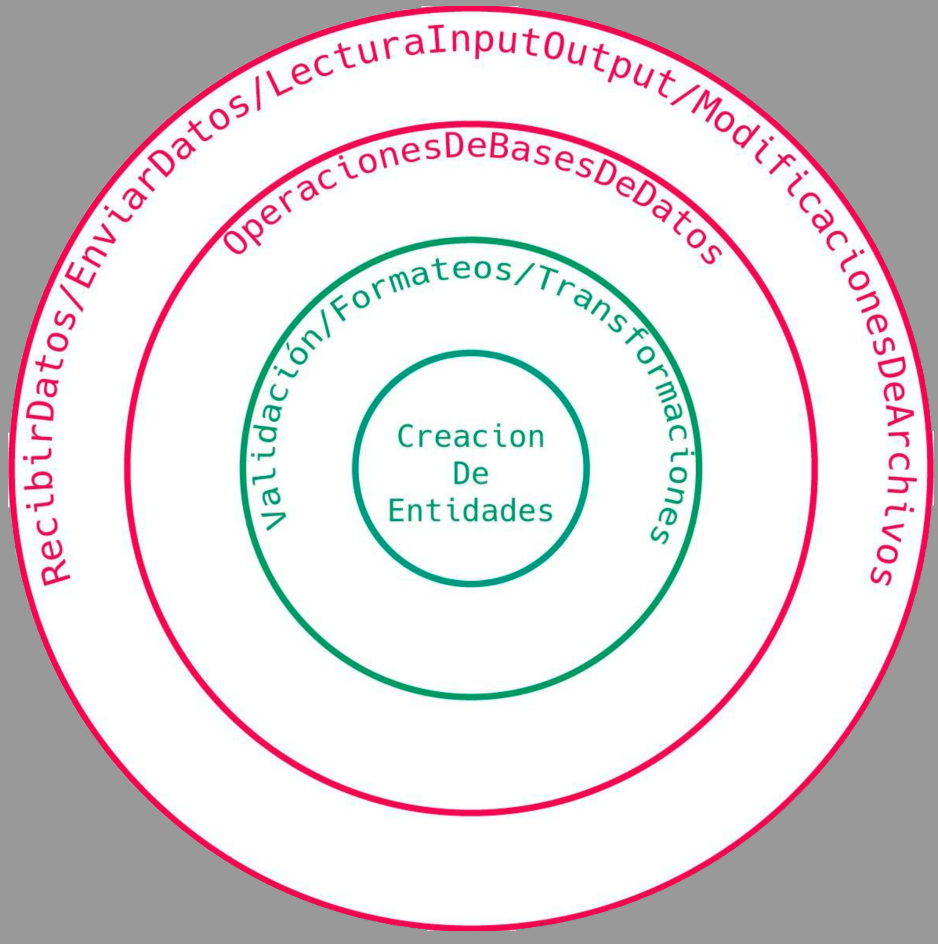


- Leer/Crear/modificar/borrar archivos
- Leer/Escribir una base de datos
- Enviar/Recibir una llamada de red
- Alterar un objeto/variable usada por otras funciones

Efectos secundarios



- ¡Pero podemos reducir/ controlar cuando suceden!
- ¿Por qué evitar los efectos secundarios?
- Los efectos secundarios son inevitables...



Funciones Puras

Funciones Impuras

Función Impura llama a…

Función Pura llama a…

Funciones de orden mayor (high order functions)

Una función de orden mayor cumple al menos con una de estas características:

- Toma otra función como parámetro
- Retorna una función como resultado

Toma una función como parámetro:

```
int foo(Function param)
```

Retorna una función como resultado:

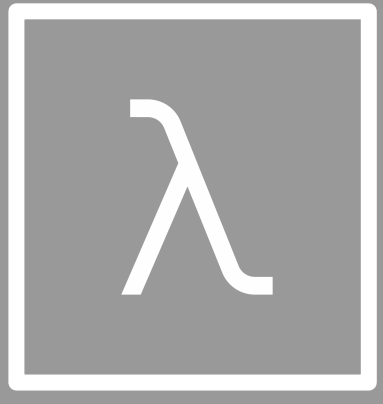
```
Function bar(int x)
```

O ambas...

```
Function baz(Function f)
```

Ventajas

- Pasar comportamiento
- Compartir un medio de comunicación (callbacks)
- Compartir lógica/reglas



Funciones lambda

- Parten de un concepto matemático de los años 30 (Alonzo Church)
- Son funciones anónimas
- ¯_(ツ)_/¯

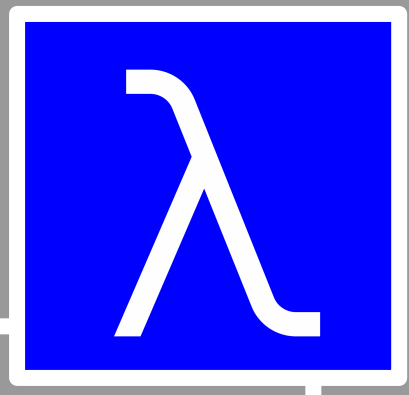
Funcion:

- Tiene un nombre
  - `Function baz = …`
  - `int foo(…)`

Lambda:

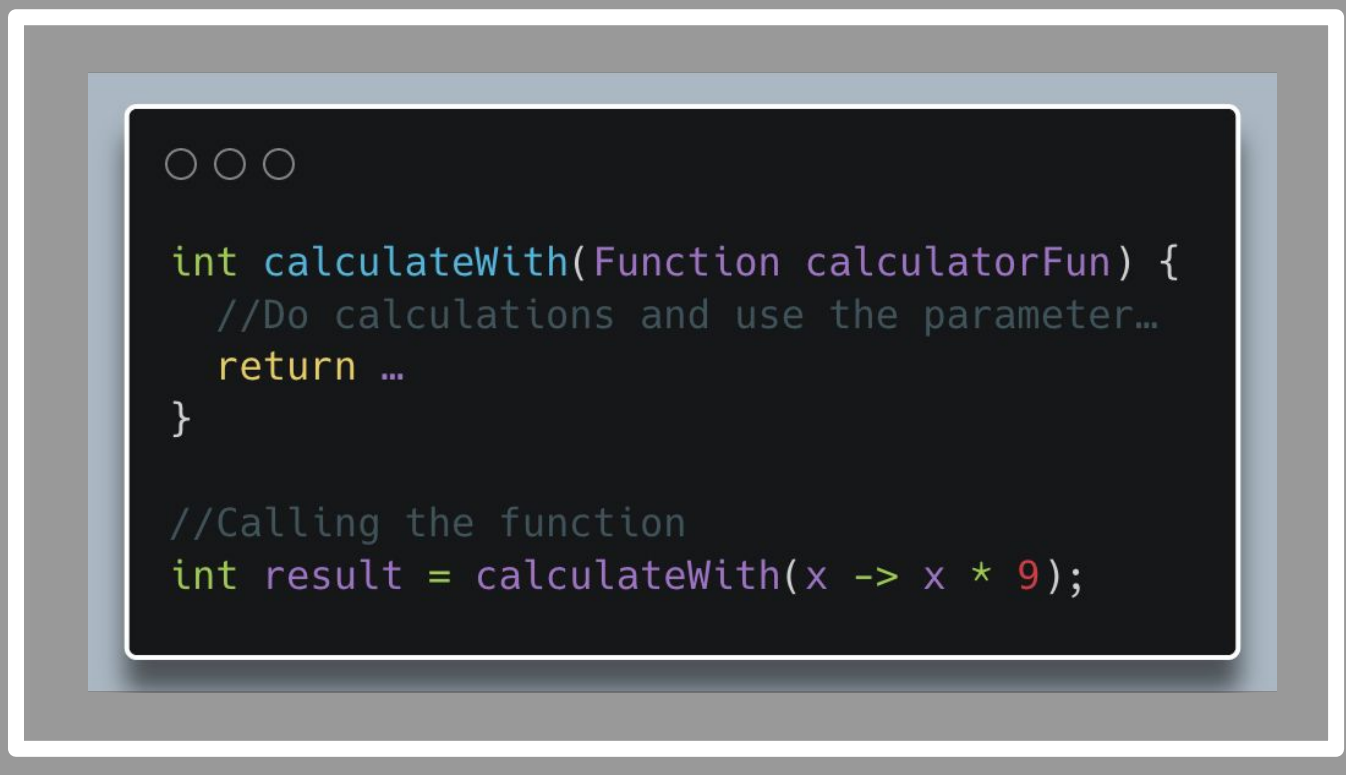
- No tiene nombre
  - `x -> …`

¿Por qué usarlas?



- Es un comportamiento de uso único
- Una regla que solo se requiere en un lugar
- Es una función extremadamente simple

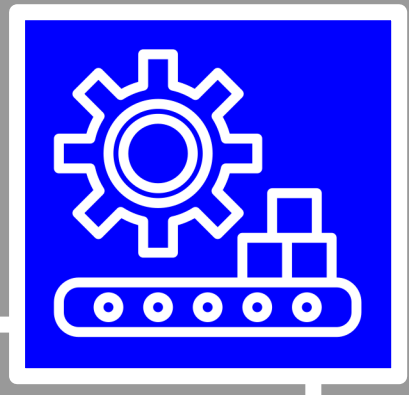
Una lambda sigue siendo una función



```
int calculateWith(Function calculatorFun) {
  //Do calculations and use the parameter…
  return …
}

//Calling the function
int result = calculateWith(x -> x * 9);
```

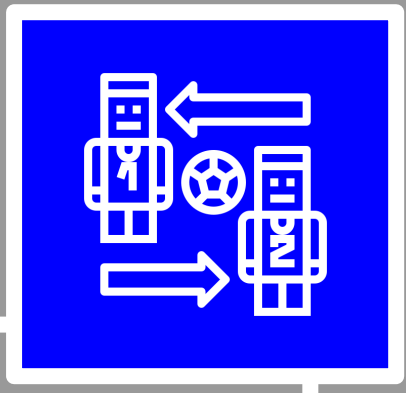
Inmutabilidad

Ventajas



- Una vez creado no se puede alterar
- Facilita crear funciones puras
- Facilita usar threads/concurrencia

Desventajas



- Nueva Instancia por cada set de modificaciones
- Requiere especial atención al diseño
- Objetos mutables fuera de nuestro alcance