



Argentina
programa
4.0



UNIVERSIDAD
TECNOLOGICA
NACIONAL

Interfaces funcionales y Expresiones Lambda

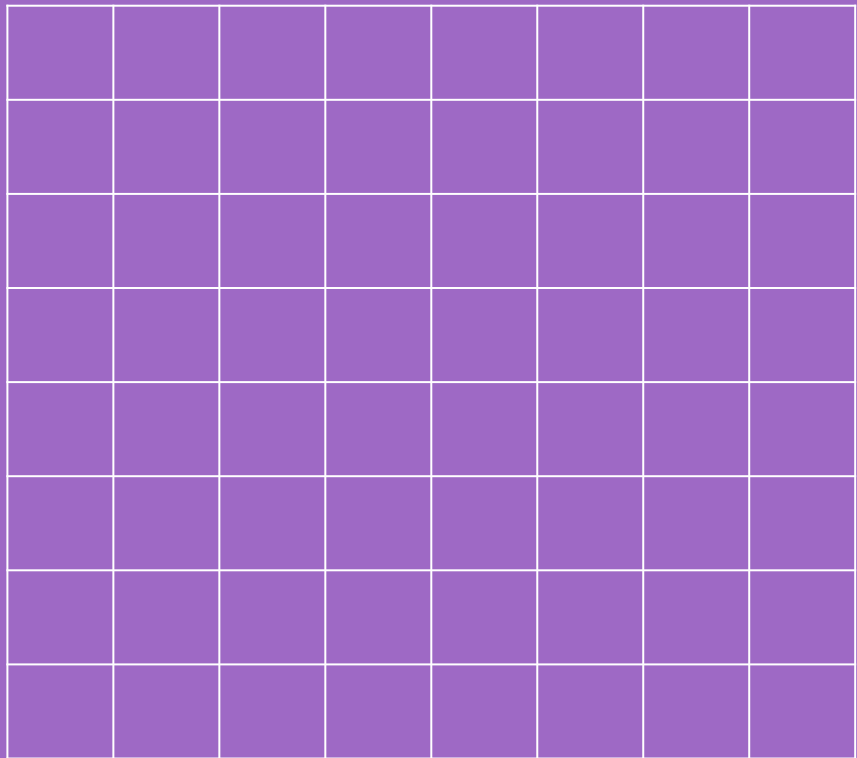
Etapas 2 - Desarrollador Java Intermedio

Agenda

- Interfaces funcionales
 - Concepto
 - Anotación `FunctionalInterface`
- Expresiones Lambda
 - Concepto
 - Expresión Lambda como variable
 - Expresión Lambda como parámetro



Interfaces funcionales



Concepto

Una interfaz funcional es aquella que declara un único método abstracto. Estas serán implementadas, como ya veremos más adelante, por expresiones lambda. Este único método abstracto define el propósito que tendrá la interfaz.

Un ejemplo de interfaz funcional:

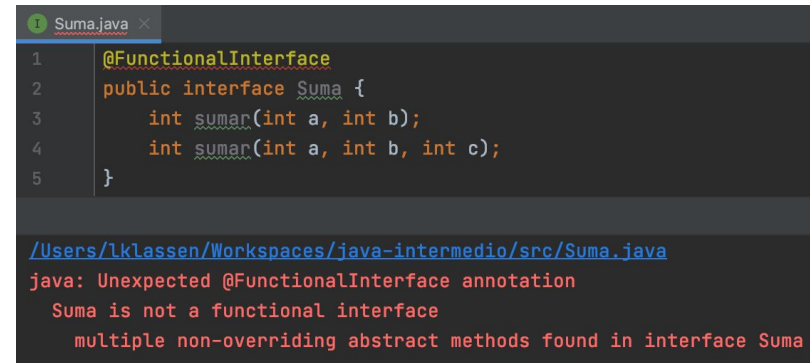
```
public interface Suma {  
  
    int sumar(int unNumero, int otroNumero);  
  
}
```

Anotación FunctionalInterface

Como vimos, la condición para cumplir con el concepto de interfaz funcional es que la misma tenga un único método abstracto. El compilador de Java, si no le indicamos de alguna forma, no sabrá que estamos queriendo definir una interfaz funcional. Entonces, si tenemos una interfaz que la queremos usar como funcional, pero accidentalmente se agrega más de un método abstracto, la aplicación compilará correctamente. Pero cuando se quiera hacer uso, en tiempo de ejecución, de la interfaz funcional, la aplicación se romperá.

```
@FunctionalInterface
public interface Suma {
    int sumar(int unNumero, int otroNumero);
}
```

Con el uso de la anotación, que se agrega en el ejemplo, se puede prevenir este error en tiempo de ejecución. De esta forma el compilador, al momento de compilar la aplicación, si la interfaz tuviera más de un método abstracto, la misma no compilará, como se puede apreciar en la imagen al costado.

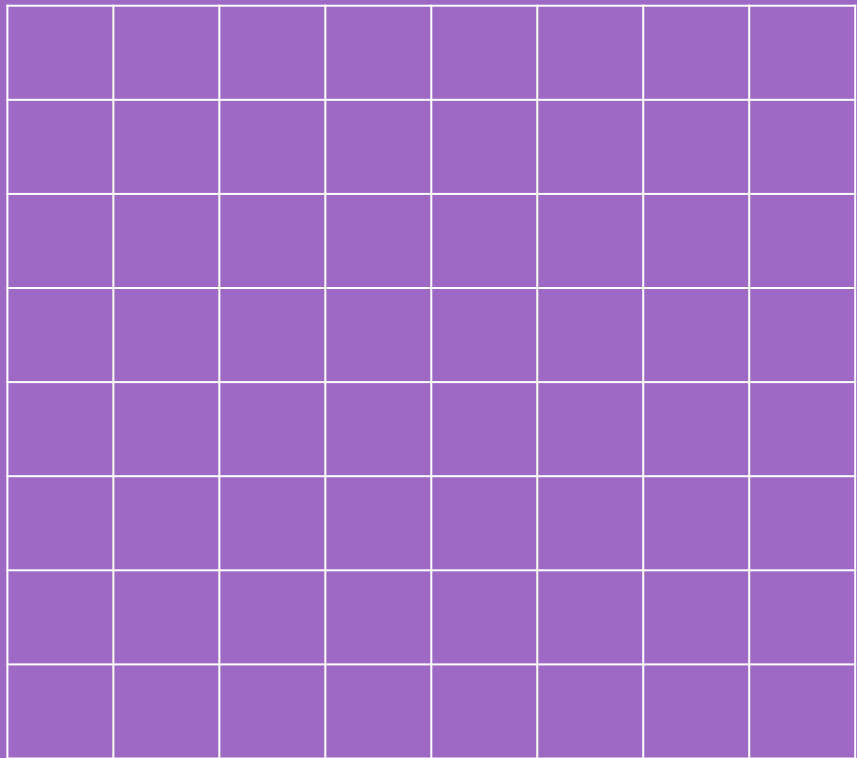


```
1  @FunctionalInterface
2  public interface Suma {
3      int sumar(int a, int b);
4      int sumar(int a, int b, int c);
5  }
```

[/Users/lklassen/Workspaces/java-intermedio/src/Suma.java](#)
java: Unexpected @FunctionalInterface annotation
Suma is not a functional interface
multiple non-overriding abstract methods found in interface Suma



Expresiones Lambda



Concepto

Una expresión lambda es una forma concisa de expresar un comportamiento que puede ser pasado como argumento a un método o asignado a una variable. Definen la implementación de una interfaz funcional.

```
(parámetros) -> { comportamiento }
```

Dónde parámetros es una lista de parámetros separados por coma y comportamiento es el bloque de código deseado.

También, pueden no recibir parámetros, en ese caso se representa:

```
() -> { comportamiento }
```

En el caso de que el comportamiento sea una única línea de código las llaves no son necesarias. Tomando el ejemplo de la interfaz funcional Suma, la expresión lambda que define la implementación de la misma podría ser:

```
(unNumero, otroNumero) -> unNumero + otroNumero;
```

Que es equivalente a:

```
(unNumero, otroNumero) -> {  
    return unNumero + otroNumero;  
}
```

Expresión Lambda como variable

Las expresiones Lambda pueden ser asignadas a variables. Siguiendo con el ejemplo de la interfaz funcional Suma, podemos declarar una variable con la expresión lambda mostrada en la diapositiva anterior de la siguiente manera:

```
Suma lambdaSuma = (unNumero, otroNumero) -> unNumero + otroNumero;
```

```
int unNumero = 3;
```

```
int otroNumero = 5;
```

```
int resultadoSuma = lambdaSuma.sumar(unNumero, otroNumero);
```

```
System.out.println(resultadoSuma);
```

Notar que se declara una variable “lambdaSuma” que es de tipo Suma. El valor de dicha variable es una expresión lambda que cumple con la firma del método abstracto de la interfaz funcional Suma. Recordar que la misma definía un método “sumar” el cual recibe como parámetros dos enteros y devuelve un entero. De esta forma estamos definiendo el comportamiento de nuestra interfaz funcional mediante una expresión Lambda.

Expresión Lambda como parámetro

Otra posibilidad es recibir expresiones Lambda como parámetros en nuestros métodos. Para entender este concepto definamos una nueva interfaz funcional:

```
@FunctionalInterface
public interface Operacion {
    int aplicar(int unNumero, int otroNumero);
}
```

Imaginemos un método que reciba como parámetros dos enteros y una Operacion:

```
public int calcular(int unNumero, int otroNumero, Operacion operacion) {
    return operacion.aplicar(unNumero, otroNumero);
}
```

Expresión Lambda como parámetro

Podríamos:

```
int unNumero = 10;  
int otroNumero = 5;
```

```
int suma = calcular(unNumero, otroNumero, (a, b) -> a + b);  
System.out.println("La suma de " + unNumero + " y " + otroNumero + " es: " + suma);
```

```
int resta = calcular(unNumero, otroNumero, (a, b) -> a - b);  
System.out.println("La resta de " + unNumero + " y " + otroNumero + " es: " + resta);
```

```
int producto = calcular(unNumero, otroNumero, (a, b) -> a * b);  
System.out.println("El producto de " + unNumero + " y " + otroNumero + " es: " + producto);
```

```
int cociente = calcular(unNumero, otroNumero, (a, b) -> a / b);  
System.out.println("El cociente de " + unNumero + " y " + otroNumero + " es: " + cociente);
```

Referencias

- <https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html>
- <https://stackabuse.com/guide-to-functional-interfaces-and-lambda-expressions-in-java/>

Gracias!