El uso de Java lambda inferred type es muy común cuando trabajamos con expresiones lambda pero a veces es difícil entender como funcionan. Vamos a construir unos ejemplos que nos ayuden a clarificar el concepto. Para ello vamos a partir de un interface Calculadora que define el concepto de operación para luego apoyarnos en expresiones lambda y utilizarlo.

```
public interface Calculadora {

     public double operacion(double numero1,double numero2);
}
```

Acabamos de definir un interface el cual recibe dos parámetros y nos devuelve un resultado. Podemos diseñar una clase que implemente este interface y sume los números.

```
package com.arquitecturajava.ejemplo1;

public class CalculadoraSuma implements Calculadora {

        @Override
        public double operacion(double numero1, double numero2) {
                return numero1+numero2;
        }

}
```

Construimos un programa principal y lo ejecutamos:

```
package com.arquitecturajava.ejemplo1;

public class Principal {

        public static void main(String[] args) {

                Calculadora c = new CalculadoraSuma();
                System.out.println(c.operacion(2, 2));

        }

}
```

El resultado será :

Utilizando Java Lambda inferred type

Podríamos realizar la misma operación utilizando una sencilla expresión lambda sin necesidad de construir una clase CalculadoraSuma:

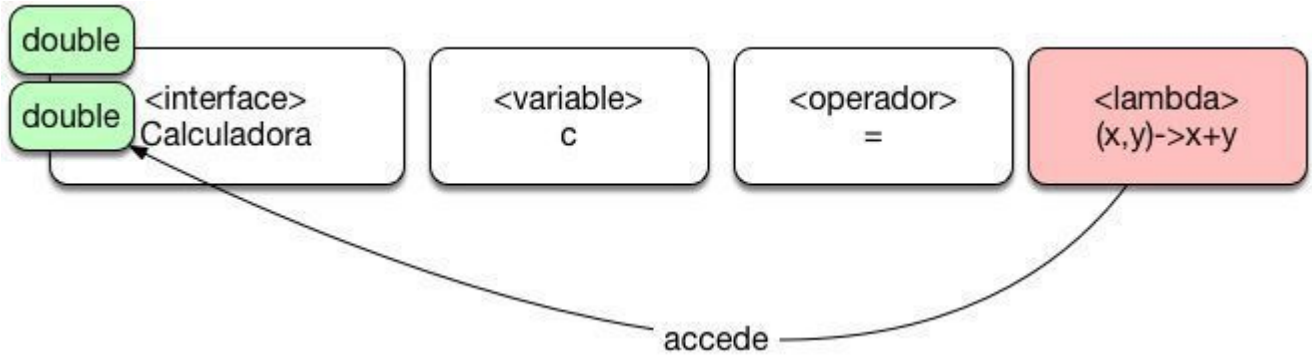
```
package com.arquitecturajava.ejemplo2;

public class Principal2a {

 public static void main(String[] args) {
```

```
 Calculadora c=(double x, double y)->x+y;

 System.out.println(c.operacion(2, 2));


 }

}
```

Todo funciona correctamente.,el problema para muchas personas viene cuando cambiamos la expresión lambda eliminando los tipos de parámetros:

```
package com.arquitecturajava.ejemplo2;

public class Principal2a {

 public static void main(String[] args) {

 Calculadora c=(x,y)->x+y;

 System.out.println(c.operacion(2, 2));


 }

}
```

En este caso no hemos definido x e y como tipos dobles ¿Cómo sabe Java que tipos debe asignar? . El compilador puede asignar los tipos apoyándose en el tipo de la variable c que es Calculadora ya que solo tiene un método operar que recibe dobles.



Se puede ver de forma más sencilla extrayendo el método:

```
package com.arquitecturajava.ejemplo2;

public class Principal2b {

 public static void main(String[] args) {

 imprimirResultado(2,(x, y) -> x + y);

 }

 private static void imprimirResultado(int numero,Calculadora c) {
 System.out.println(c.operacion(numero, numero));
 }

}
```

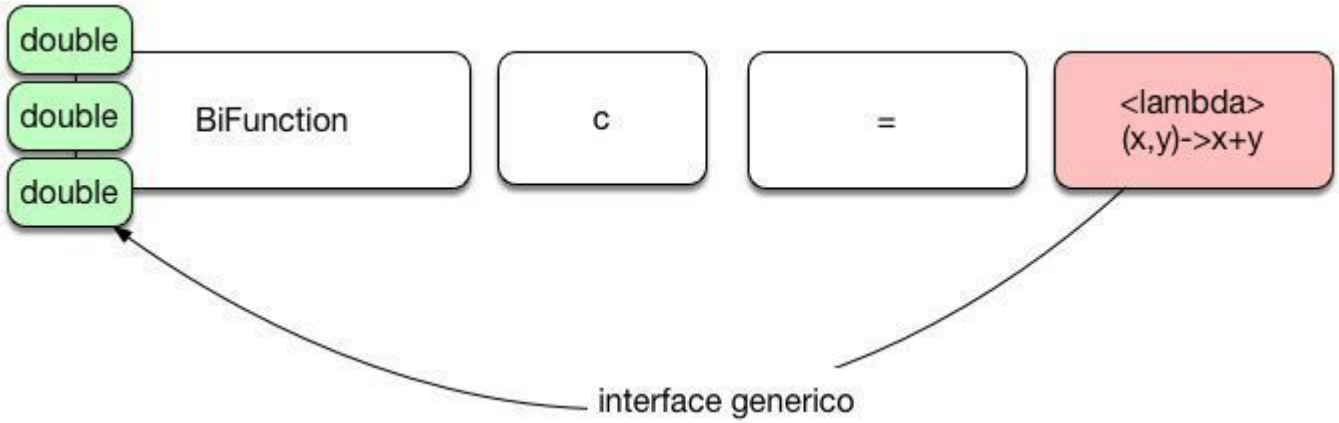
Recordemos que no hubiera hecho falta definir ni siquiera el interface ya que el package de java.util.function existen una serie de interfaces que nos permiten gestionar estas operaciones tan genéricas. Vamos a usar el interface genérico BiFunction.

```
package com.arquitecturajava.ejemplo2;

import java.util.function.BiFunction;

public class Principal3 {

 public static void main(String[] args) {


 imprimirResultado(2,(x,y)->x+y);
 }

 private static void imprimirResultado(double
numero,BiFunction<Double, Double, Double> c) {
 System.out.println(c.apply(numero,numero));
 }


}
```

El resultado es el mismo:

```
Problems  @ Javadoc
<terminated> Principal (4) [J
4.0
```

Eso sí, como hemos tenido que usar un interface genérico hay que definir los tipos de datos aceptados a nivel de interface:



El concepto de java lambda inferred type ayuda en Java 8 a simplificar el código:

Otros artículos relacionados:

1. Java 8 Lambda Expressions (I)
2. Utilizando un Java reference method
3. Java 8 Stream y workflows