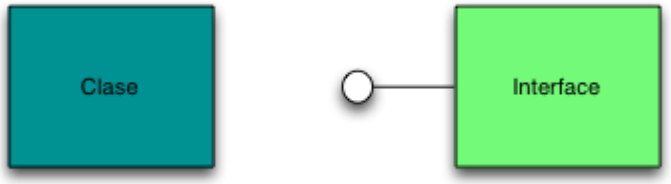
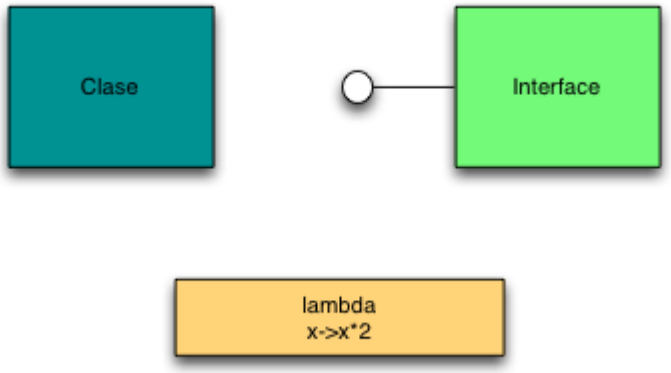
Java 8 reference method es uno de los conceptos que llegan con las nuevas expresiones lambda que vienen en el JDK 8 ,vamos a explicarlo paso a paso. Hasta este momento en Java existían dos conceptos fundamentales, las clases y los interfaces.



Con la llega de Java 8 aparecen las expresiones lambda:



Estas expresiones nos permiten una mayor reutilización de nuestro código ya que en vez de reutilizar clases podemos simplemente reutilizar funciones . Estas se convierten por decirlo de alguna manera en la unidad mínima de código. Para que Java pueda integrar estos conceptos hay que asumir que hay que realizar concesiones ya que Java solo permite el manejo de clases e interfaces. Así pues las expresiones lambda vienen a ser interfaces que únicamente soportan un método abstracto . En Java 8 existen muchos de estos interfaces predefinidos. Para abordar nuestro ejemplo nos vamos a apoyar en el interface Consumer<T> que tiene un único método que se denomina accept el cual recibe un parámetro y no devuelve nada. Es uno de los interfaces funcionales más sencillos.

```
import java.util.function.Consumer;

public class Principal {

    public static void main(String[] args) {

        Consumer<String> consumidor = (x) -> System.out.println(x);
        consumidor.accept("hola");

    }

}
```

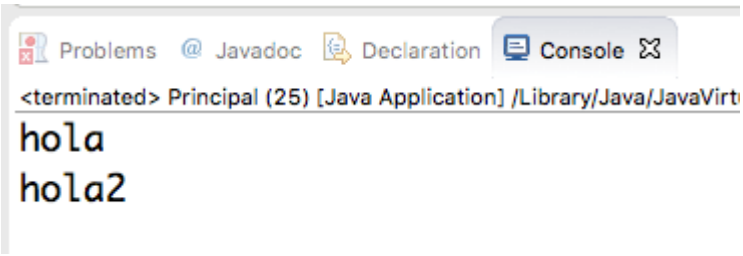
En este caso hemos asignado al interface funcional Consumer una expresión lambda que encaja con él . Recordemos que el interface recibe un parámetro y no devuelve nada, el resultado imprimirá hola por la consola:

Podemos querer tener una mayor flexibilidad en nuestro código y apoyarnos en un método que nos permite variar la expresión lambda y el texto que imprimimos.

```
import java.util.function.Consumer;

public class Principal {

    public static void main(String[] args) {

        Consumer<String> consumidor = (x) -> System.out.println(x);
        consumidor.accept("hola");
```

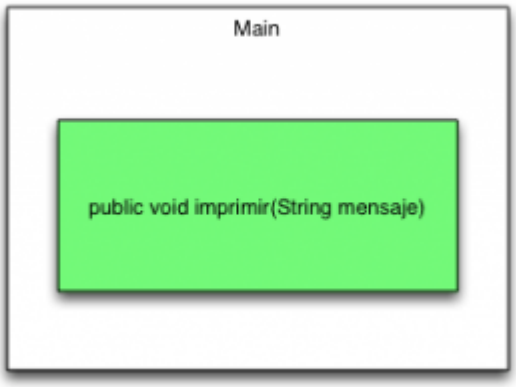
```
        procesar((x) -> System.out.println(x), "hola2");
    }

    public static <T> void procesar(Consumer<T> expresion, T mensaje)
{

        expresion.accept(mensaje);
    }
}
```

El resultado será similar:



Java 8 Reference Method

Ahora bien las expresiones lambda nos permiten añadir más flexibilidad al código . Por ejemplo podemos disponer de otro método imprimir en la misma clase que cumpla con el patrón de la expresión lambda (reciba un parámetro y no devuelva nada) .

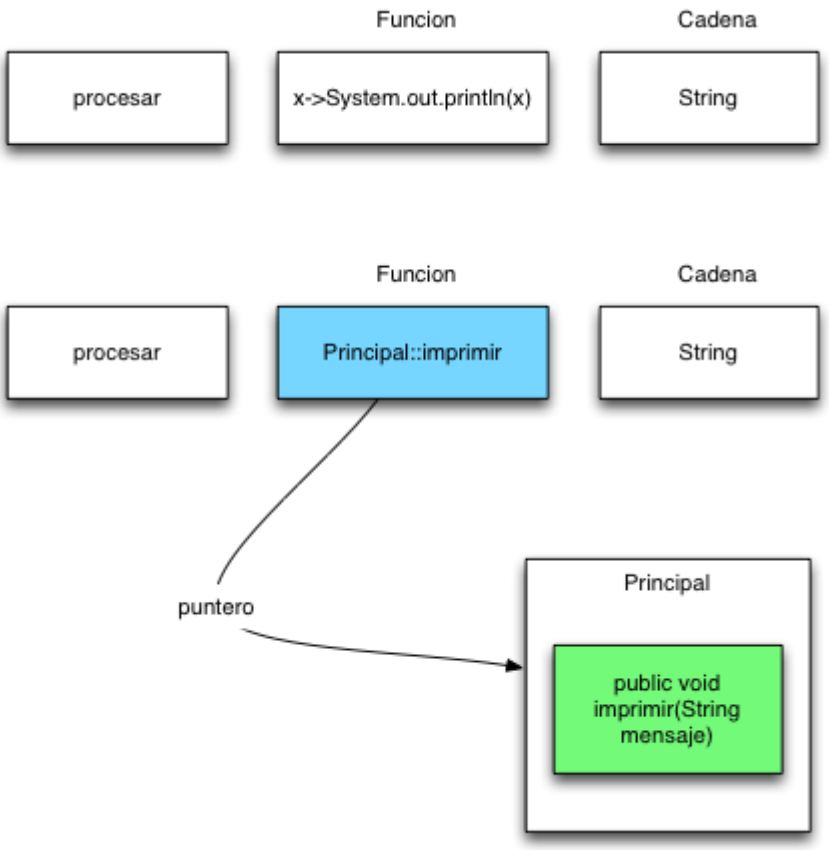


Podemos utilizar la sintaxis de Java 8 reference method para invocar al método procesar apoyándonos en esta nueva función.

```
package com.arquitecturajava.lambdareferencias;

import java.util.function.Consumer;

public class Principal {

    public static void main(String[] args) {

        Consumer<String> consumidor = (x) -> System.out.println(x);
        consumidor.accept("hola");

        procesar((x) -> System.out.println(x), "hola2");
        procesar(Principal::imprimir, "hola3");

    }

    public static <T> void procesar(Consumer<T> expresion, T mensaje)
{

        expresion.accept(mensaje);
    }

    public static void imprimir(String mensaje) {

        System.out.println("---------");
        System.out.println(mensaje);
        System.out.println("---------");
    }
```

```
}
```

Como se puede observar hemos usado la sintaxis Principal::imprimir para pasar como referencia una función que se encuentra en nuestro propio programa.



La consola mostrará:

```
hola
hola2
---------
hola3
---------
```

Esto aportará una gran flexibilidad al método ya que si por ejemplo disponemos de otra clase que disponga de un método que reciba un parámetro y no devuelva nada podremos

usarla también:

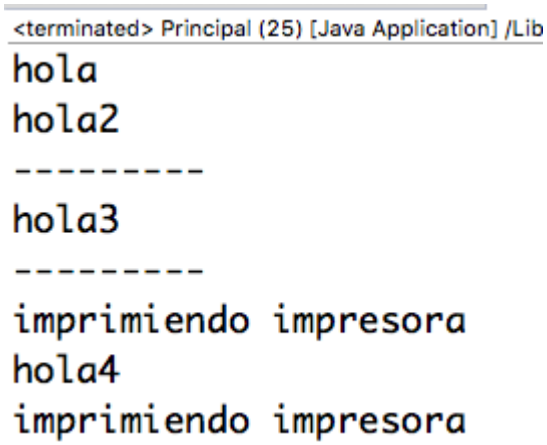
```
package com.arquitecturajava.lambdareferencias;

import java.util.function.Consumer;

public class Principal {

    public static void main(String[] args) {

        Consumer<String> consumidor = (x) -> System.out.println(x);
        consumidor.accept("hola");

        procesar((x) -> System.out.println(x), "hola2");
        procesar(Principal::imprimir, "hola3");

    }

    public static <T> void procesar(Consumer<T> expresion, T mensaje)
{

        expresion.accept(mensaje);
    }

    public static void imprimir(String mensaje) {

        System.out.println("---------");
        System.out.println(mensaje);
        System.out.println("---------");
    }
}
```

En este caso hemos creado la clase Impresora , podemos utilizar su método imprimir como si fuera una referencia a una expresión lambda.

```
package com.arquitecturajava.lambdareferencias;

import java.util.function.Consumer;

public class Principal {

    public static void main(String[] args) {

        Consumer<String> consumidor = (x) -> System.out.println(x);
        consumidor.accept("hola");

        procesar((x) -> System.out.println(x), "hola2");
        procesar(Principal::imprimir, "hola3");
        Impresora i = new Impresora();
        procesar(i::imprimir, "hola4");
    }

    public static <T> void procesar(Consumer<T> expresion, T mensaje)
{

        expresion.accept(mensaje);
    }

    public static void imprimir(String mensaje) {

        System.out.println("---------");
        System.out.println(mensaje);
        System.out.println("---------");
```

```
    }
}
```

El programa Principal imprimirá la funcionalidad de la impresora:



Los Java 8 reference method son parte del core de las expresiones lambda y su uso nos permitirá aumentar la flexibilidad de nuestro código.

Otros artículos relacionados:

- Introducción a Lambda
- ¿Qué es un Java Lambda?
- Java Streams
- Java Lambda WorkFlows
- Oracle Java Lambda