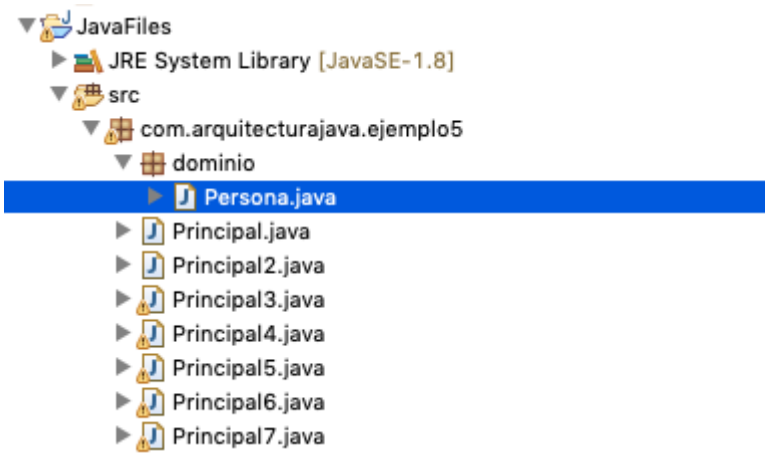
El uso de Java 8 Files Walk resulta a veces un poco complicado de entender cuando estamos hablando de manejo de ficheros y directorios en Java . ¿Para qué sirve el método Files.Walk de Java 8? . Vamos a echarlo un vistazo.Normalmente cuando nosotros queremos acceder a una estructura de carpetas y directorios usamos la clase File.

```
package com.arquitecturajava.ejemplo5;

import java.io.File;

public class Principal {

  public static void main(String[] args) {

    File carpeta = new File("./src");

    File[] lista = carpeta.listFiles();

    for (File fichero : lista) {

      imprimir(fichero);
    }
  }

  private static void imprimir(File fichero) {
    if (fichero.isDirectory()) {

      for (File ficheroHijo : fichero.listFiles()) {

        imprimir(ficheroHijo);
      }
```

```
    } else {

      System.out.println(fichero.getName());
    }
  }

}
```

Si tenemos una estructura de carpetas de este estilo:

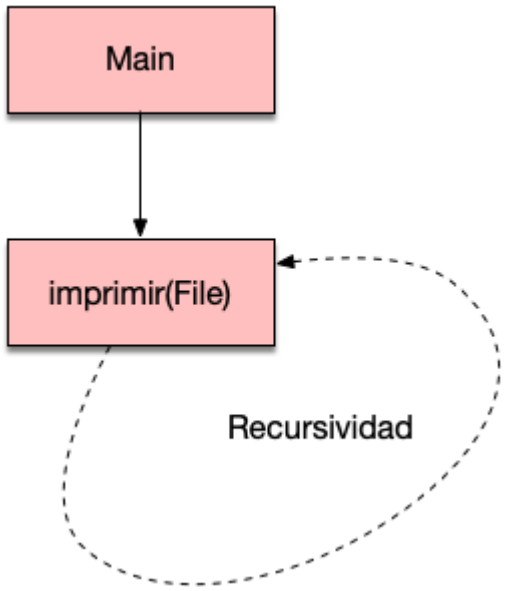


El resultado será algo como lo siguiente impreso por la consola:

```
Principal6.java
Principal7.java
Persona.java
Principal4.java
Principal5.java
Principal.java
Principal2.java
Principal3.java
```

Java Recursividad

El código que estamos construyendo hace uso de la recursividad y comprueba si un fichero es un directorio o no. En tal caso vuelve a llamar a la función imprimir de forma recursiva hasta que termina con todos. Cuando uno tiene experiencia programando este código resulta relativamente sencillo de entender. Ahora bien hay que reconocer que para una operación que parece sencilla tenemos mucho código que construir.



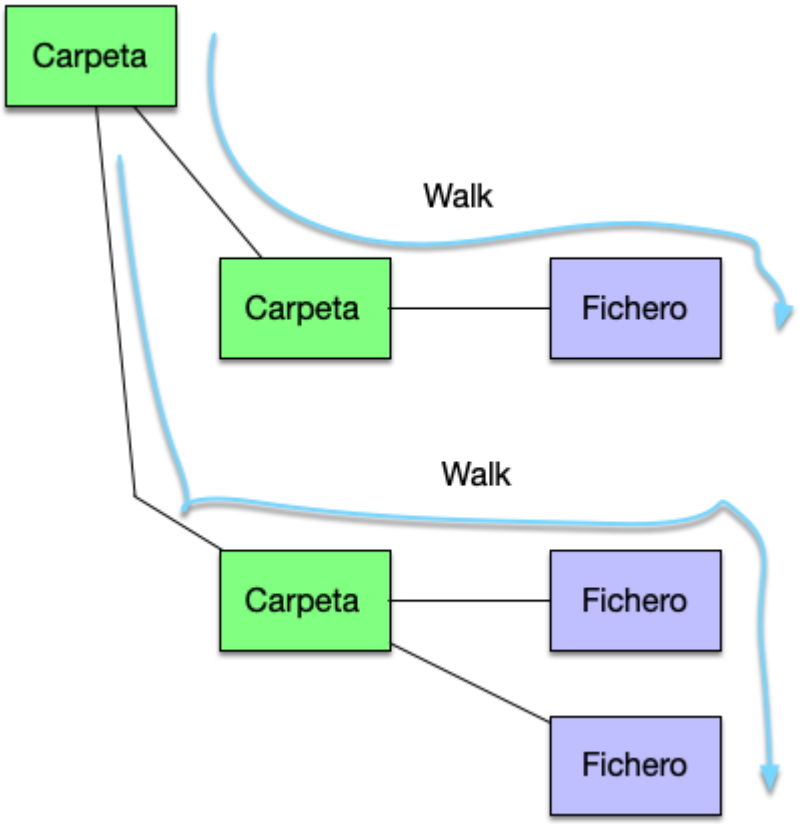
Java 8 Files Walk

Ahora bien este código existe prácticamente desde la versión 1.0 de Java. ¿No hay una opción de hacerlo de una forma más transparente y directa? . Eso es lo que aporta Java 8 con el manejo de Streams usando Java 8 Files Walk . Vamos a verlo.

```
package com.arquitecturajava.ejemplo5;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
```

```
import java.nio.file.Paths;
import java.util.stream.Stream;

public class Principal3 {

  public static void main(String[] args) {

    try {
      Stream<Path> miStream = Files.walk(Paths.get("./src"));
      miStream.forEach(System.out::println);

    } catch (IOException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
    }

  }

}
```

Este código es mucho más reducido que el anterior y es capaz de imprimir la información por la consola de una manera mucho más exacta que la versión previa la cual hemos simplificado para ganar claridad. ¿Cómo funciona exactamente? .



En este caso estamos construyendo un Streams sobre el directorio, ficheros y directorios hijos que tiene la carpeta actual . Devolviendo sucesivamente cada uno de los items con su path completo. De esta forma es muy sencillo recorrerlos ya que parece que se trata de un sencillo camino que va devolviendonos la estructura.

Los avances que contiene Java 8 son muy importantes a la hora de simplificar código clásico de alta complejidad. Java 8 Functional Interfaces y sus tipos

1. Java 8 Lambda Syntax ,simplificando nuestro código
2. Java List Directory en Java 8 con Streams

3. Oracle JavaDoc