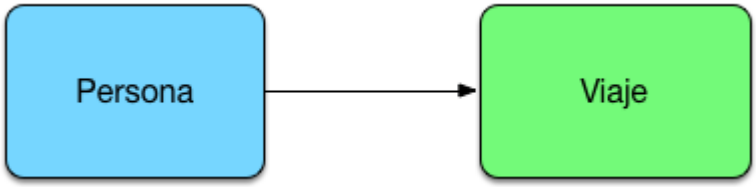
El uso de Java 8 FlatMap es algo que en muchas ocasiones cuesta entender . La programación funcional en Java 8 esta empezando y para la mayor parte de la gente es algo muy nuevo. Vamos a crear un ejemplo sencillo de flatMap, partiremos de dos clases relacionadas Personas y Viajes.



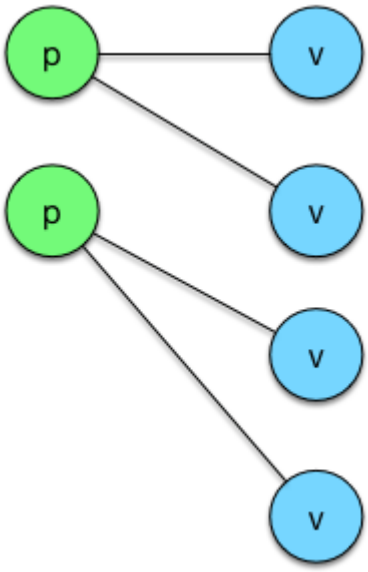
Una persona realiza varios viajes.

```
import java.util.List;
public class Persona {
  private String nombre;
  private List < Viaje > lista = new ArrayList < Viaje > ();
  public String getNombre() {
    return nombre;
  }
  public void addViaje(Viaje v) {
    lista.add(v);
  }
  public List < Viaje > getLista() {
    return lista;
  }
```

```
  public Persona(String nombre) {
    super();
    this.nombre = nombre;
  }
  public void setNombre(String nombre) {
    this.nombre = nombre;
  }
}
```

```
package com.arquitecturajava;

public class Viaje {
  private String pais;

  public Viaje(String pais) {
    super();
    this.pais = pais;
  }

  public String getPais() {
    return pais;
  }

  public void setPais(String pais) {
    this.pais = pais;
  }
}
```

La estructura es anidada.

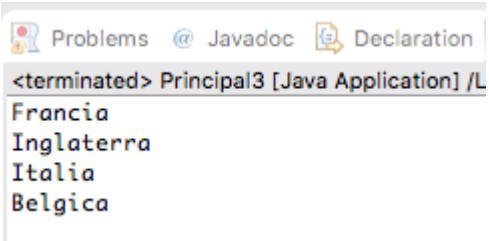


Es momento de crear y recorrer la estructura en código:

```
package com.arquitecturajava;

import java.util.ArrayList;
import java.util.List;

public class Principal {

  public static void main(String[] args) {

    Persona p = new Persona("pedro");
    Viaje v = new Viaje("Francia");
    Viaje v2 = new Viaje("Inglaterra");
    p.addViaje(v);
    p.addViaje(v2);
    Persona p1 = new Persona("gema");
    Viaje v3 = new Viaje("Italia");
    Viaje v4 = new Viaje("Belgica");
```

```
    p1.addViaje(v3);
    p1.addViaje(v4);

    List < Persona > lista = new ArrayList < Persona > ();
    lista.add(p);
    lista.add(p1);

    for (Persona persona: lista) {

      for (Viaje viaje: persona.getLista()) {

        System.out.println(viaje.getPais());
      }
    }

  }

}
```

La consola nos mostrará:



El problema es que en ningún caso hemos usado programación funcional para recorrer la lista, simplemente dos bucles anidados.

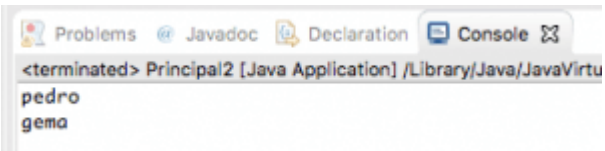
Usando Java 8 Map

El primer paso va a ser usar la función map para que a través de programación funcional

nos imprima los nombres de las personas.


```
 lista.stream().map(persona -> persona.getNombre()).forEach(new
Consumer < String > () {

   @Override
   public void accept(String s) {
     System.out.println(s);

   }

 });
```

Hemos convertido la lista en un Stream y a través de map() hemos convertido el Stream de Personas en un Stream de “Strings” que imprime los nombres.



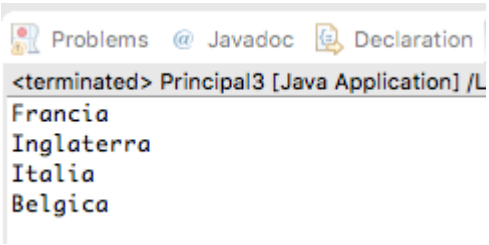
Lamentablemente no es lo que queríamos ya que necesitamos imprimir el nombre de los países.

Usando Java 8 FlatMap

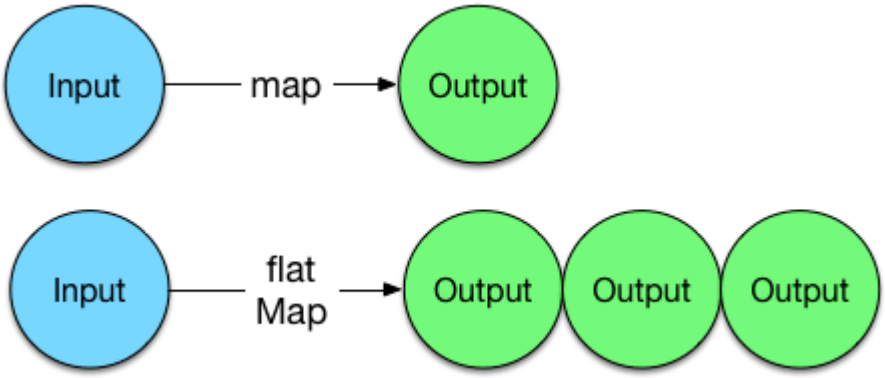
Para conseguir la lista de países debemos operar de otra manera y usar la función flatMap.

```
lista.stream().map(persona - > persona.getLista())
  .flatMap(viajes -> viajes.stream())
  .forEach(new Consumer < Viaje > () {
    @Override
    public void accept(Viaje t) {

      System.out.println(t.getPais());
    }

  });
```

El resultado es :



Ahora bien ,¿Cómo funciona exactamente flatMap? . FlatMap es una función que recibe una entrada y devuelve varias salidas para esa entrada . Esa es la diferencia con Map que tiene una entrada y devuelve una única salida. En este caso hemos convertido el array en un stream.



Podemos todavía optimizar algo más nuestro código y volver a usar la función map.

```
lista.stream().map(persona -> persona.getLista())
  .flatMap(viajes -> viajes.stream())
  .map(viaje -> viaje.getPais()).forEach(System.out::println);
```

El resultado será idéntico:

```
Problems @ Javadoc Declaration
<terminated> Principal3 [Java Application] /L
Francia
Inglaterra
Italia
Belgica
```

Java 8 FlatMap nos ha ayudado a eliminar todo tipo de bucle del programa.

Otros artículos relacionados:

- Java Lambda ,
- Java Reference Methods ,
- Java Streams