

El concepto de Java Collector Join viene a solventar un problema clásico cuando trabajamos con listas de elementos . En muchas ocasiones disponemos de una lista de objetos y necesitamos construir una cadena de texto que contenga parte de la información de esta lista y que además la delimite de alguna forma. Para entenderlo mejor vamos a construir un ejemplo apoyándonos en la clase Persona.

```
package com.arquitecturajava;
```

```
public class Persona {
```

```
    private String nombre;
```

```
    private String apellidos;
```

```
    private int edad;
```

```
    public String getNombre() {  
        return nombre;  
    }
```

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

```
    public String getApellidos() {  
        return apellidos;  
    }
```

```
    public void setApellidos(String apellidos) {  
        this.apellidos = apellidos;  
    }
```

```
    public int getEdad() {  
        return edad;  
    }
```

```
    public void setEdad(int edad) {
```

```

        this.edad = edad;
    }
    public Persona(String nombre, String apellidos, int edad) {
        super();
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.edad = edad;
    }
}

```

Java List y StringBuilder

Vamos a crearnos un programa que construya una lista de personas y las recorra para obtener sus nombres . Estos nombres deberán ser introducidos en una cadena de texto e impresos delimitados por “(“. El código no es complejo de construir:

```

package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;

public class Principal {

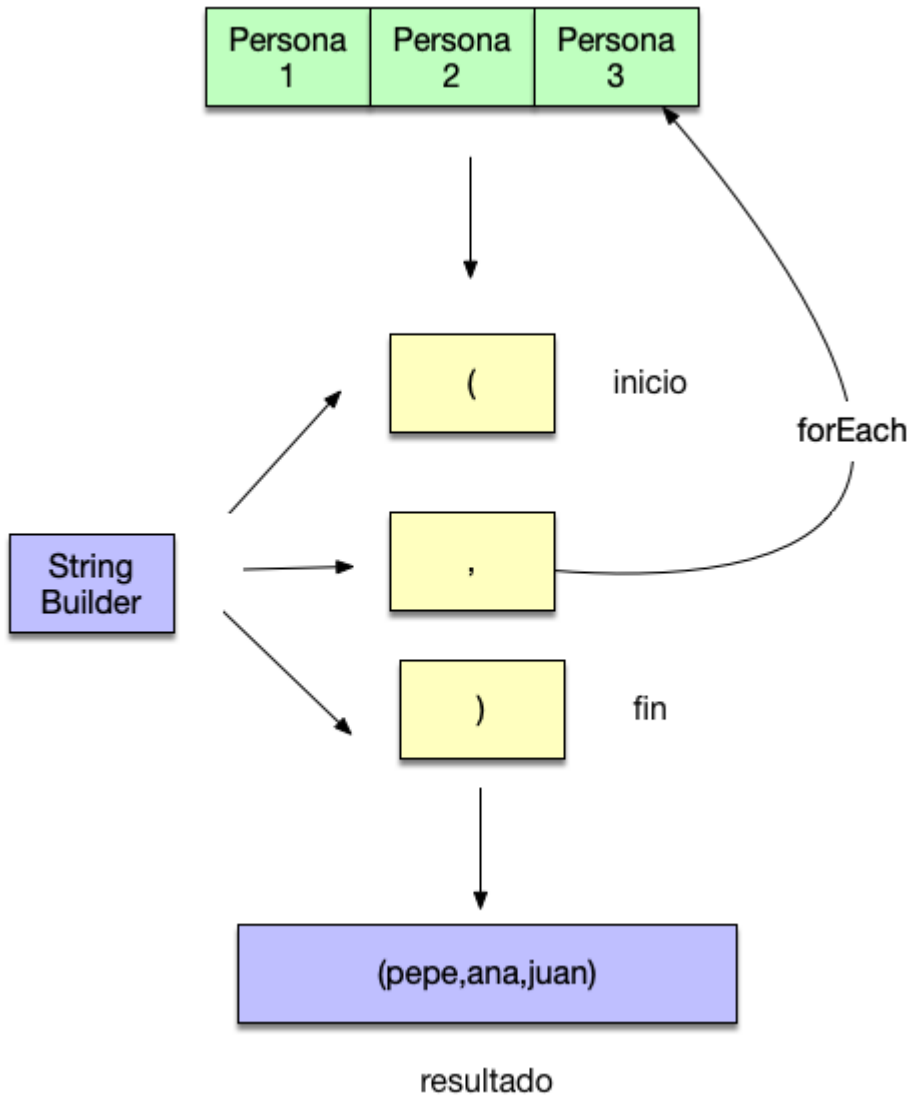
    public static void main(String[] args) {
        Persona p1= new Persona("pepe", "perez", 30);
        Persona p2= new Persona("juan", "perez", 50);
        Persona p3= new Persona("ana", "perez", 40);
        List<Persona> lista= Arrays.asList(p1,p2,p3);
        StringBuilder nuevaCadena= new StringBuilder("(");
    }
}

```

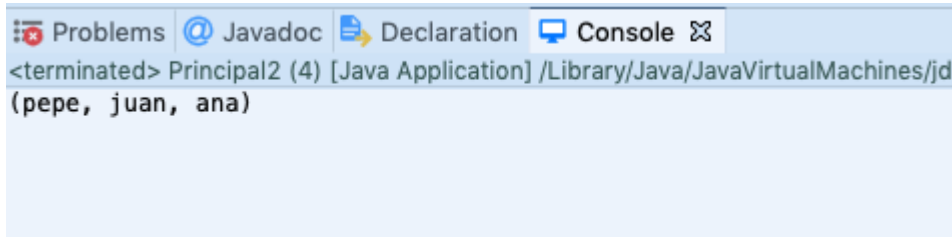
```
        for(Persona p:lista) {
            if(nuevaCadena.length()>1) {
                nuevaCadena.append(",");
            }
            String nombre=p.getNombre();
            nuevaCadena.append(nombre);
        }
        nuevaCadena.append(")");

        System.out.println(nuevaCadena.toString());
    }
}
```

En este caso podemos darnos cuenta cómo usamos un `StringBuilder` y vamos construyendo la cadena paso a paso teniendo en cuenta cuál es el carácter de inicio de bloque "(" así como el carácter de finalización ")". En medio de esto tenemos una estructura clásica de `forEach` que se encarga de recorrer la lista de `Personas`. El código es completamente válido y funcional .



Podemos ver su resultado en la consola



```
<terminated> Principal2 (4) [Java Application] /Library/Java/JavaVirtualMachines/jd
(pepe, juan, ana)
```

¿Se puede hacer de otra manera en Java 8? . La realidad es que SI , utilizando streams ,vamos a verlo

Java Collector Join

Podemos realizar la misma operación trabajando con el concepto de Stream y utilizando un **Java Stream Collector** de esta forma el código quedará mucho más compacto y sencillo de entender . En este caso utilizaremos un Collector denominado `joinings` que nos permite recoger una lista de valores y convertirlos en una cadena asignando delimitadores.

```
package com.arquitecturajava;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

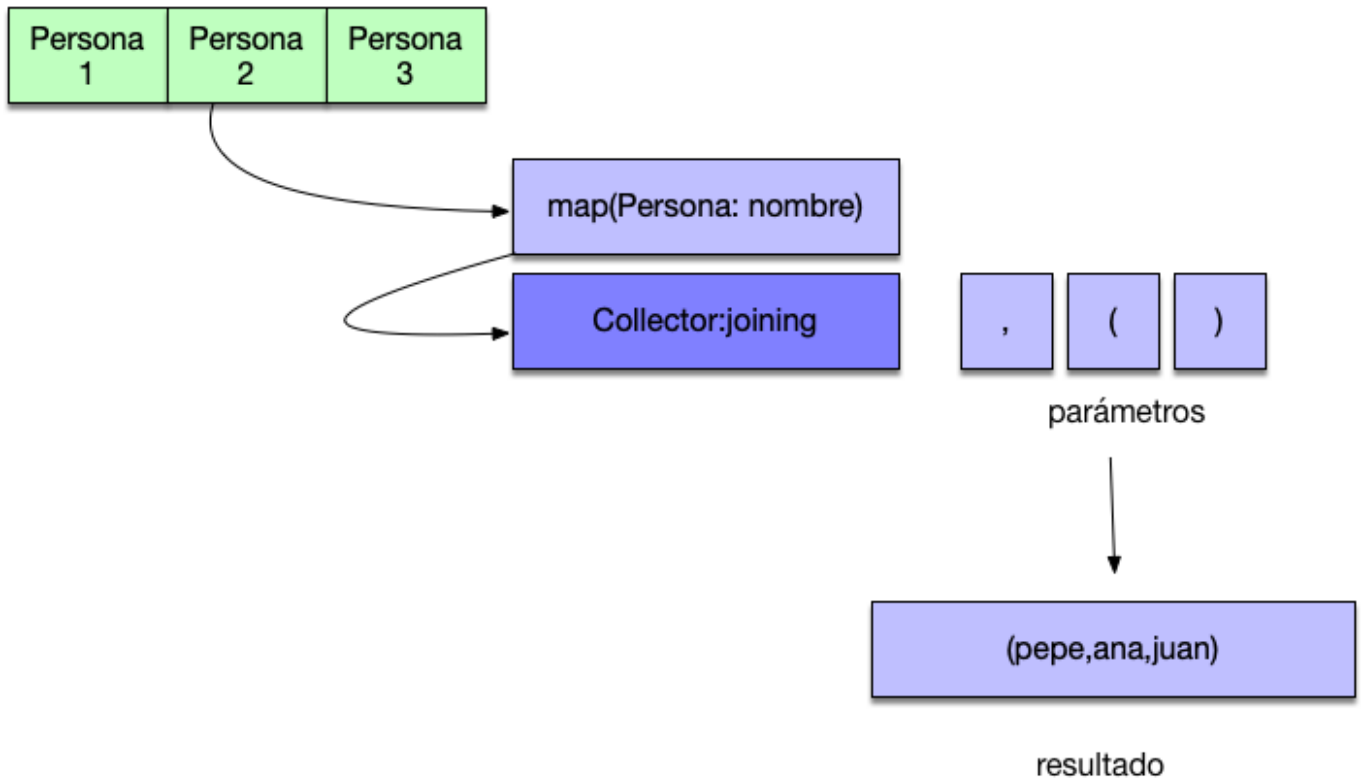
public class Principal2 {

    public static void main(String[] args) {
        Persona p1= new Persona("pepe", "perez", 30);
        Persona p2= new Persona("juan", "perez", 50);
        Persona p3= new Persona("ana", "perez", 40);
        List<Persona> lista= Arrays.asList(p1,p2,p3);
```

```
String nuevaCadena =
    lista.stream()
        .map(Persona::getNombre)
        .collect(Collectors.joining(",
", "(" , ")"));
System.out.println(nuevaCadena);
}

}
```

En este caso lo que hemos hecho es usar la función `map` para transformar la lista de objetos en una lista de Strings con los nombres. Una vez que tenemos esta lista de Streams nos queda realizar la operación de `collect` y agruparlos todos utilizando tres parámetros el primero es el divisor entre elementos `","` . El segundo es el `"("` de inicio y el último el `")"` de fin.



El resultado será idéntico

```
Problems Javadoc Declaration Console
<terminated> Principal2 (4) [Java Application] /Library/Java/JavaVirtualMachines/jd
(pepe, juan, ana)
```

Aprendamos a usar mas el concepto de Java Collector que siempre es útil:

Otros artículos relacionados

1. [Java Stream String y Java 8](#)
2. [Java Stream peek funcionamiento y logging](#)
3. [Java Stream forEach y colecciones](#)

4. Java Stream en Oracle