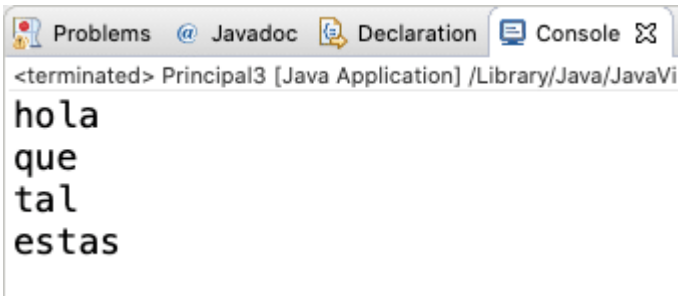
Tabla de Contenidos

El uso de Java 9 Factory Methods nos ayudará a trabajar de una forma más cómoda con la inicialización de colecciones . En la mayoría de los casos cuando tenemos que trabajar con colecciones siempre hacemos operaciones como las siguientes a la hora de inicializarlas.

```
package com.arquitecturajava;

import java.util.ArrayList;

public class Principal2 {

	public static void main(String[] args) {
		ArrayList<String> lista= new ArrayList<>();
		lista.add("hola");
		lista.add("que");
		lista.add("tal");
		lista.add("estas");
		for (String cadena : lista) {
			System.out.println(cadena);
		}

	}

}
```

Este tipo de operaciones es realmente tediosa pero nos permiten construir la lista y recorrerla de forma sencilla , como podemos observar la lista se presenta en la consola

correctamente.



Para solventar este problema se puede usar la clase Arrays y su método asList() que nos permite simplificar el código:

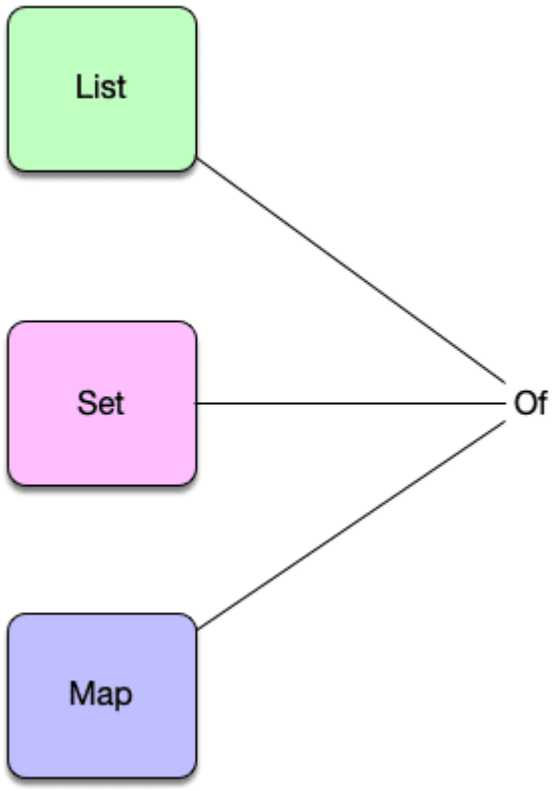
```
package com.arquitecturajava;

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

public class Principal {

	public static void main(String[] args) {
		List<String> lista= new ArrayList<>();
		lista= Arrays.asList("hola","que","tal","estas");
		for (String cadena : lista) {
			System.out.println(cadena);
		}

	}

}
```

Sin embargo esto solo nos sirve para las listas y existen otros tipos de colecciones que se

usan muy habitualmente como son Set y Map .

Java 9 Factory Methods

Para solventar estos problemas a partir existen los Java 9 Factory Methods que a partir de la versión 9 del JDK están disponibles



.Así pues si queremos hacer uso de ellos e inicializar las colecciones de una forma sencilla y rápida podemos hacer lo siguiente:

```
package com.arquitecturajava;

import java.util.List;

public class Principal3 {
```

```
        public static void main(String[] args) {
                List<String> lista=
List.of("hola","que","tal","estas");
                for (String cadena : lista) {
                        System.out.println(cadena);
                }

        }

}
```

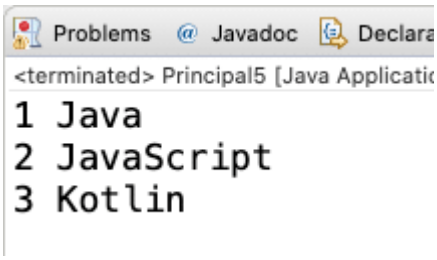
Utilizamos el método of del interface List y automáticamente podremos construir una lista. Del mismo modo podemos aplicar esta solución a los Sets

```
package com.arquitecturajava;

import java.util.Set;

public class Principal4 {

        public static void main(String[] args) {
                Set<String> lista= Set.of("hola","que","tal","estas");
                for (String cadena : lista) {
                        System.out.println(cadena);
                }

        }

}
```

O mejor aún lo podemos aplicar a los Mapas asignando de forma sucesiva claves y valores a estos.

```
package com.arquitecturajava;

import java.util.Map;


public class Principal5 {

	public static void main(String[] args) {
		Map<Integer,String>
mapa=Map.of(1,"Java",2,"JavaScript",3,"Kotlin");
		for (Map.Entry<Integer, String> claveValor :
mapa.entrySet()) {
			System.out.println(claveValor.getKey()+ "
"+claveValor.getValue());
		}

	}

}
```

De esta manera cuando recorremos el mapa podemos ver las parejas de clave valor que contiene.



Otras opciones

Las capacidades de Java 9 nos ayudan sobre manera pero hay mucha gente que en estos

momentos esta en versiones de Java 8 .Recordemos que en Java 8 también disponemos del método Stream.of que nos permite generar los elementos de forma sencilla pero no tan directa como Java 9.

```
package com.arquitecturajava;

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

public class Principal6 {

	public static void main(String[] args) {
		List<String> lista=
Stream.of("hola","que","tal","estas").collect(Collectors.toList());
		for (String cadena : lista) {
			System.out.println(cadena);
		}

	}

}
```

Como podemos observar en este caso necesitamos hacer uso de collectors para que todo encaje de forma relativamente sencilla a nivel de código.

Otros artículos relacionados

1. Java Stream Filter
2. Java Stream map
3. Java Stream reduce
4. Java 9