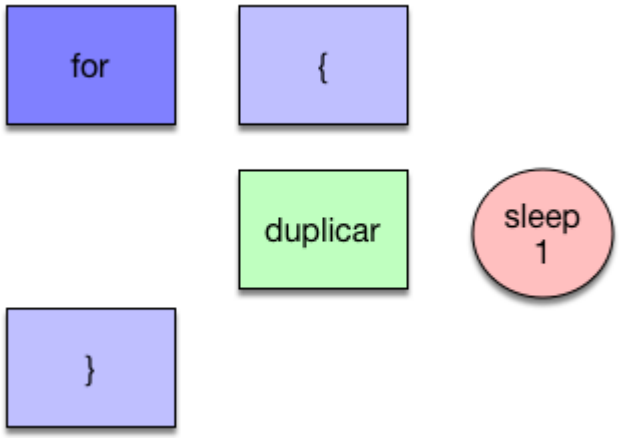
El concepto de Java Parallel Stream es un concepto sencillo de entender . En muchas ocasiones podemos tener un flujo de trabajo que necesitemos mejorar su rendimiento permitiendo su ejecución en paralelo a través de varios Threads . Esto es algo que dependiendo del código del programa hacerlo sin utilizar streams es complicado. Vamos a ver un ejemplo.

```
package com.arquitecturajava.streams.ejemplo013Paralelos;

public class Principal2 {

	public static void main(String[] args) {

		int total=0;

		long numero1=System.currentTimeMillis();

		for (int i=0;i&amp;amp;amp;lt;10;i++) {

			total+=duplicar(i);
		}

		long numero2=System.currentTimeMillis();

		System.out.println(numero2-numero1);
		System.out.println(total);
	}

	public static  int duplicar( int numero) {
```

```
                try {
                        Thread.sleep(1000);
                } catch (InterruptedException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

                return numero*2;
        }

}
```

Acabamos de construir un sencillo bucle for en el cual invocamos al método duplicar. Lo curioso de este método es que pone el thread a dormir durante un segundo.



Por lo tanto si realizamos 10 iteraciones el bucle tardará unos 10 segundos en ejecutarse.

Veamos el resultado por la consola:

Java Parallel Stream

Como esperábamos hemos tardado 10 segundos en ejecutar el código. ¿Podríamos tardar menos? . En principio si , si dividieramos la tarea en varios Threads . El problema es que no se nos ocurre como dividirla de una forma clara o por lo menos de una forma sencilla. Es aquí donde el manejo de streams es una ventaja. ¿Como podemos realizar la misma operación con Streams?.

```
package com.arquitecturajava.streams.ejemplo013Paralelos;

import java.util.stream.IntStream;

public class Principal {

	public static void main (String[] args) {

		long numero1=System.currentTimeMillis();

		IntStream lista= IntStream.range(1, 10);
		int total=lista.map(Principal::duplicar).sum();

		long numero2=System.currentTimeMillis();
		System.out.println(numero2-numero1);
		System.out.println(total);

	}
```

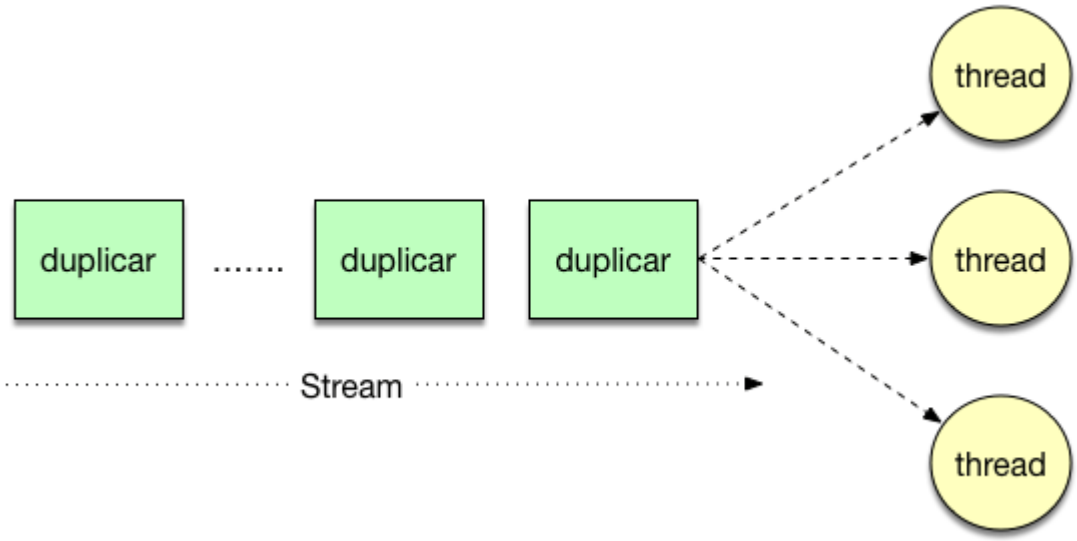
```
	public static  int duplicar( int numero) {

		try {
			Thread.sleep(1000);
		} catch (InterruptedException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

		return numero*2;
	}

}
```

El resultado por la consola es muy similar:

Acabamos de realizar la misma operación utilizando Streams

Java Parallel Stream y sus ventajas

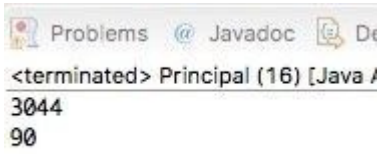
Sin embargo es aquí donde si podemos utilizar de una forma sencilla la programación en paralelo y modificar el número de Threads que se encargan de procesar este Stream.



Para ello nos es suficiente con modificar la siguiente linea y añadir el método parallels para que java lance un conjunto de hilos para realizar la tarea:

```
int total=lista.parallel().map(Principal::duplicar).sum();
```

El resultado mejora claramente:



Acabamos de procesar nuestro Stream utilizando varios Threads. Hemos construido nuestro primer ejemplo de Java Parallel Stream.

Acabamos de ver como usar Java equals y hashcode

Otros artículos relacionados:

1. Java Stream Sum y Business Objects
2. Java Stream File y manejo de ficheros

3. El concepto de Java infinite Stream
4. Java Streams