El concepto de Java Stream Context , es difícil de entender en un primer momento . Sin embargo una vez que lo entendemos podremos simplificar el como trabajamos con Streams. Vamos a construir un ejemplo sencillo que nos ayude a entender como se usa el contexto en un Stream. Supongamos que disponemos de una lista de Compras realizadas.

```
package com.arquitecturajava.ejemplo1;

public class Compra {

	private int id;
	private double importe;

	public Compra(int id, double importe) {
		super();
		this.id = id;
		this.importe = importe;
	}
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	public double getImporte() {
		return importe;
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}

}
```

La compra dispone de dos propiedades id e importe , vamos a crear una lista de elementos con ella:

```
package com.arquitecturajava.ejemplo1;

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

public class Principal {

	public static void main(String[] args) {

		Compra c1= new Compra(1,100);
		Compra c2= new Compra(2,130);
		Compra c3= new Compra(3,130);
		List<Compra> misCompras= Arrays.asList(c1,c2,c3);

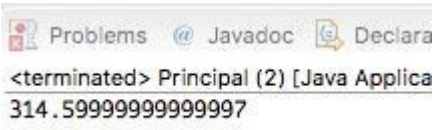
	}

}
```

El siguiente paso es recorrer la lista de compras y sumar aquellas que impuestos incluidos superen los 150 euros de gasto para calcular cual ha sido el importe de las compras más caras. La forma más directa de hacerlo es utilizando expresiones Lambda y Streams.

```
misCompras.stream().map(c->c.getImporte()*1.21).filter(i->i>150).reduc
e(Double::sum).ifPresent(System.out::println);
```

Esto nos imprimirá el resultado en la consola.



Ahora bien quizás lo podríamos haber hecho de otra forma y fusionar los conceptos de programación funcional y programación orientada a objeto a través del concepto de Java Stream Context.

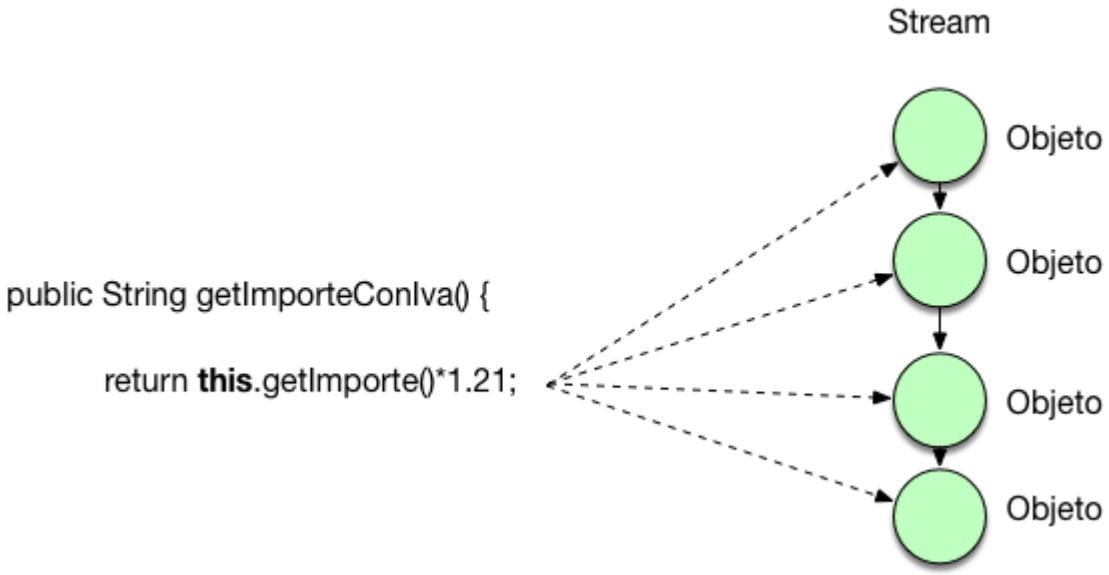
Java Stream Context

Lo más sencillo en este caso es añadir un nuevo método que nos calcule el IVA en la clase Compra.

```
public double getImporteIVA() {

                return this.importe * 1.21;
        }
```

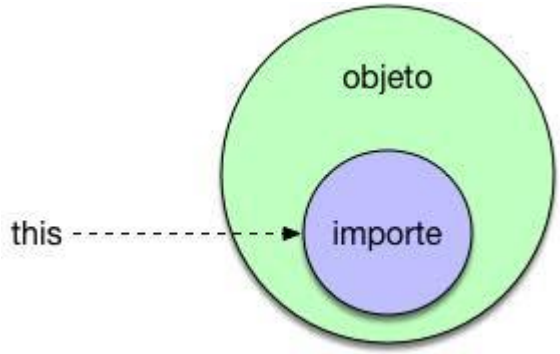
Ahora podremos cambiar nuestro Stream para que haga las cosas de una forma más compacta ya que aplica el contexto de nuestra lista de objetos.



Vamos a ver el código:

```
misCompras.stream().map(Compra::getImporteIVA).filter(i->i>150).reduce
(Double::sum).ifPresent(System.out::println);
```

Como vemos la sintaxis de Compra::getImporteIVA es mucho más natural.¿Cómo funciona si estamos invocando una función directamente que no tiene parámetros? . ¿Cómo se comporta la variable this?.



No hay ningún problema la variable this hará referencia al objeto que maneja el Stream y sus propiedades. De esa forma se trabaja más cómodo. No solo eso sino que podríamos haberlo construido de forma más natural añadiendo la siguiente función a la clase:

```
public static boolean esCaro (Compra c) {

                if (c.getImporteIVA()&gt;150) {
                        return true;
                }else {
                        return false;
                }
        }
}
```

Acabamos de diseñar un Predicate , que podemos aplicar sobre nuestro Stream y el código quedará muy sencillo.

```
misCompras.stream().filter(Compra::esCaro).map(Compra::getImporteIVA).
reduce(Double::sum).ifPresent(System.out::println);
```

El código es muy claro , acabamos de usar Java Stream contextos y Predicates para manejar una lista de compra de forma natural.

Otros artículos relacionados:

- Usando JQuery Context
- Tomcat context xml y su configuración
- Spring MVC static resources y su configuración
- ¿Que es un Bounded Context?
- Java Stream toList