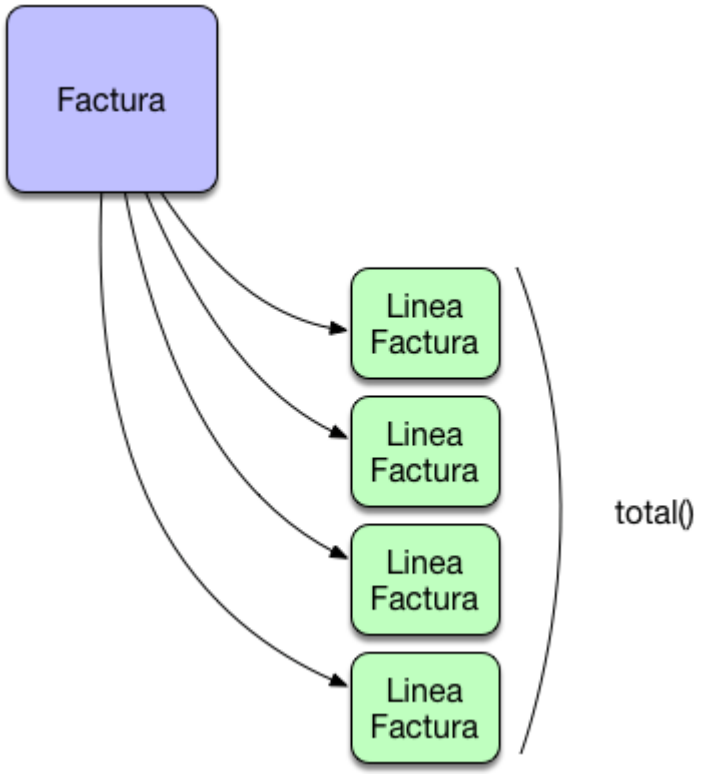
El uso de Java Stream sum nos lo encontramos muchas veces en nuestro código de Java para sumar un conjunto de elementos. Sin embargo a veces nos olvidamos que los Streams han venido para ser usados en la plataforma Java en general y se pueden aplicar a muchas situaciones. Vamos a ver un ejemplo en el cual los apliquemos a objetos de negocio. Para ello vamos a partir de los conceptos de Factura y Linea de Factura.




Estos conceptos están relacionados y es muy habitual que la clase Factura contenga un método total que nos calcule el importe total de todas las lineas. Veamos el código:

```
package com.arquitecturajava.ejemplo2;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;
```

```
import java.util.stream.Collectors;

public class Factura {

	private int numero;
	private String concepto;
	private List<LineaFactura> lineas = new
ArrayList<LineaFactura>();

	public void addLinea(LineaFactura lf) {
		lineas.add(lf);
	}
	public int getNumero() {
		return numero;
	}

	public void setNumero(int numero) {
		this.numero = numero;
	}

	public String getConcepto() {
		return concepto;
	}

	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}

	public List<LineaFactura> getLineas() {
		return lineas;
	}
```

```
	public void setLineas(List<LineaFactura> lineas) {
		this.lineas = lineas;
	}

	public double total() {

		double total = 0;
		for (LineaFactura l : lineas) {

			total += l.getImporte();
		}
		return total;
	}

}
```

```
package com.arquitecturajava.ejemplo1;

public class LineaFactura {

	private int numero;
	private String concepto;
	private double importe;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
```

```
	}
	public String getConcepto() {
		return concepto;
	}
	public void setConcepto(String concepto) {
		this.concepto = concepto;
	}
	public double getImporte() {
		return importe;
	}
	public void setImporte(double importe) {
		this.importe = importe;
	}
	public LineaFactura(int numero, String concepto, double
importe) {
		super();
		this.numero = numero;
		this.concepto = concepto;
		this.importe = importe;
	}
}
```

## Java Stream Sum

El código es correcto.¿Ahora bien podemos construir algo más flexible?. En muchas ocasiones cuando realizamos una compra y obtenemos la factura nos gusta mirar en que conceptos nos hemos gastado más dinero. Así pues podríamos usar Java Overload y sobrecargar el método total de la Factura para calcular el total apoyándonos en un Predicate.

```
public double total() {

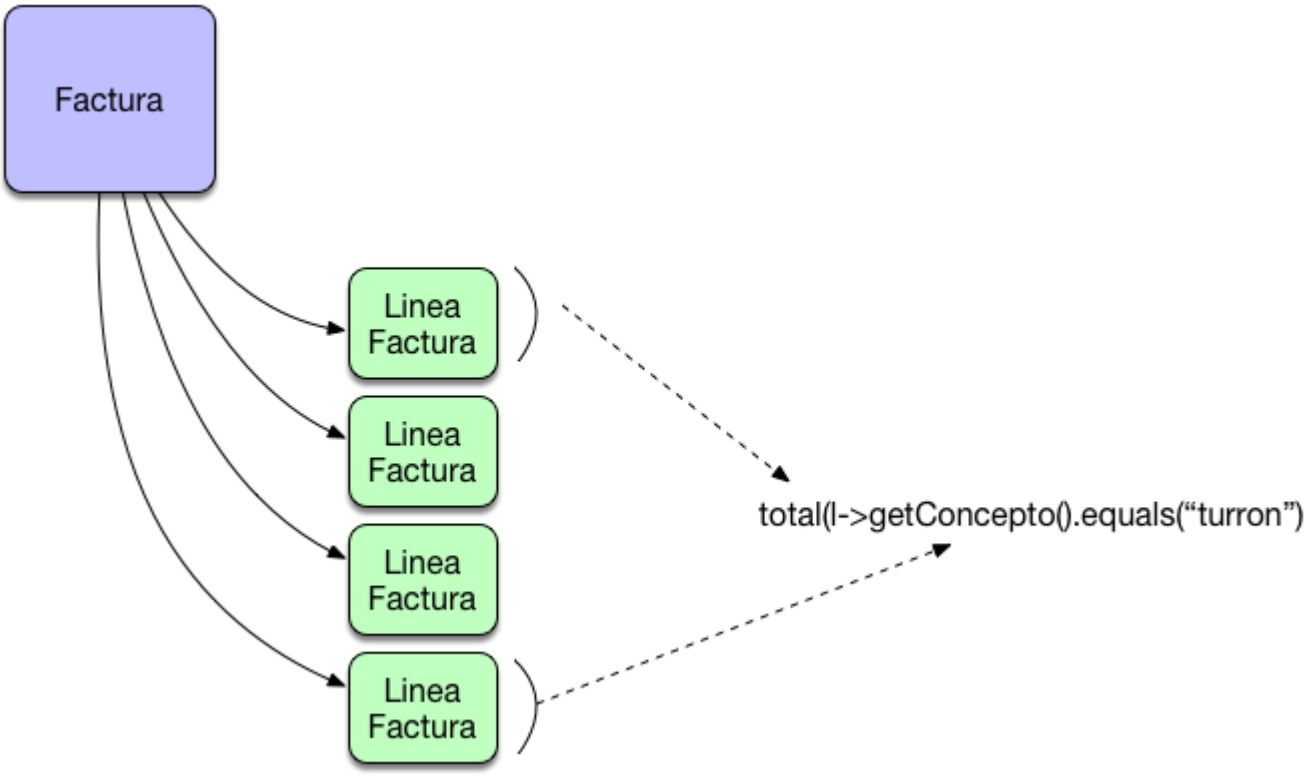
                double total = 0;
                for (LineaFactura l : lineas) {

                        total += l.getImporte();
                }
                return total;
        }

        public double total(Predicate<LineaFactura> plinea) {

                return lineas.stream().filter(plinea).mapToDouble(l ->
l.getImporte()).sum();
        }
```

Así ganamos en flexibilidad:




Veamos un programa main con el cual  obtener tanto el total como el total de un producto concreto.

```
package com.arquitecturajava.ejemplo2;

public class Principal {

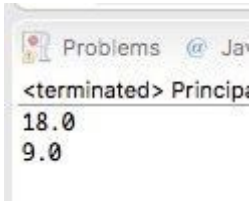
	public static void main(String[] args) {
		Factura f= new Factura();
		f.setNumero(1);
		f.setConcepto("miscompras");
		f.addLinea(new LineaFactura(1, "turron", 3));
		f.addLinea(new LineaFactura(1, "turron", 3));
```

```
                f.addLinea(new LineaFactura(1, "turron", 3));
                f.addLinea(new LineaFactura(1, "jamon", 5));
                f.addLinea(new LineaFactura(1, "jamon", 4));
                System.out.println(f.total());
System.out.println(f.total(l->l.getConcepto().equals("turron")));

        }

}
```

Acabamos de calcular el gasto total y el  que hemos hecho en turrón y vemos el resultado en la consola.




Otros artículos relacionados:

1. Java Stream map y estadísticas
2. Java 8 Lambda Expressions (I)
3. Java Stream Filter y Predicates
4. Procesamiento Streams