El concepto de JPA Stream es un concepto relativamente nuevo y viene con JPA 2.2 es decir no todo el mundo lo tiene disponible en su servidor de aplicaciones. Como su nombre indica permite devolver un Stream a partir de una consulta de JPA algo que en principio nos puede parecer curioso. Vamos a ver un ejemplo partiendo del concepto de Cliente como objeto de negocio.

```
package com.arquitecturajava.jpa.bo;

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

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQuery;
import javax.persistence.OneToMany;
import javax.persistence.Table;

@Entity
@Table(name="Clientes")

public class Cliente {
        @Id
        private String dni;
        private String nombre;
        private String apellidos;
        private int edad;
        private int telefono;
        @OneToMany(mappedBy="cliente")
        private List<Factura> listaFacturas= new ArrayList<Factura>();
```

```
	public List<Factura> getListaFacturas() {
		return listaFacturas;
	}


	public void setListaFacturas(List<Factura> listaFacturas) {
		this.listaFacturas = listaFacturas;
	}


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


	public Cliente() {
		super();
	}


	public String getDni() {
```

```
		return dni;
	}
	public void setDni(String dni) {
		this.dni = dni;
	}
	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 int getTelefono() {
		return telefono;
	}
	public void setTelefono(int telefono) {
		this.telefono = telefono;
	}
}
```

Estamos ante una clase muy sencilla de manejar y queremos seleccionar una listado de clientes utilizando JPA. La solución es trivial.

```
package com.arquitecturajava.jpa;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.TypedQuery;

import com.arquitecturajava.jpa.bo.Cliente;

public class Principal21Stream {

        public static void main(String[] args) {

                EntityManagerFactory emf;

                emf = Persistence.createEntityManagerFactory("jpa");

                EntityManager em = emf.createEntityManager();
                TypedQuery<Cliente> consulta = em.createQuery("select
c from Cliente c", Cliente.class);

                List<Cliente> lista = consulta.getResultList();

                for (Cliente c : lista) {
```

```
                        System.out.printf(" %s %s %s %s %n",
c.getDni(), c.getNombre(), c.getApellidos(), c.getEdad());

                }

                em.close();

        }

}
```

Si ejecutamos el programa nos imprimirá la lista de clientes que tenemos con sus datos:

```
124 manuel alvarez 50
234 pedro perez 20
456 gema blanco 20
567 vanesa perez 20
667 mar lopez 50
789 alfonso diaz 40
```

Podemos a partir de ahora realizar la misma operación utilizando un Stream de JPA.

```
package com.arquitecturajava.jpa;

import java.util.stream.Stream;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
```

```
import javax.persistence.TypedQuery;

import com.arquitecturajava.jpa.bo.Cliente;

public class Principal22Stream {

	public static void main(String[] args) {

		EntityManagerFactory emf;
		emf=Persistence.createEntityManagerFactory("jpa");
		EntityManager em= emf.createEntityManager();
		TypedQuery<Cliente> consulta=em.createQuery("select c
from Cliente c",Cliente.class);
		Stream<Cliente> miStream= consulta.getResultStream();
		miStream.forEach((c)-> {
			System.out.printf(" %s %s %s %s %n",
c.getDni(), c.getNombre(), c.getApellidos(), c.getEdad());

		});

		em.close();
	}

}
```

El resultado en consola es el mismo:

```
124 manuel alvarez 50
234 pedro perez 20
456 gema blanco 20
567 vanesa perez 20
667 mar lopez 50
789 alfonso diaz 40
```

No parece que ganemos nada simplemente es otra forma de abordar el problema , o eso parece en un primer momento.

JPA Stream y posibilidades

Recordemos que podemos utilizar un Stream para filtrar los datos y transformarlos una opción podría ser:

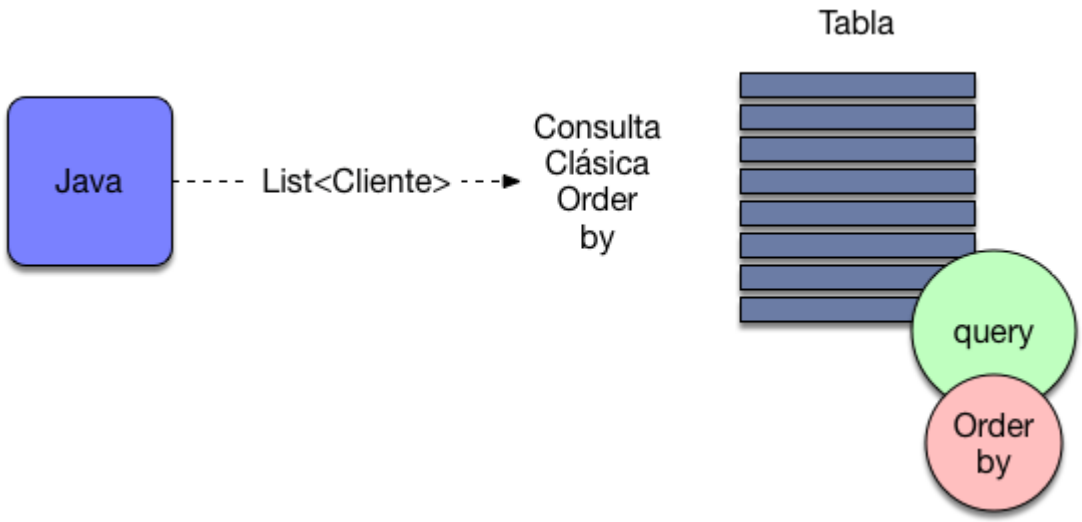
```
Stream<Cliente> miStream= consulta.getResultStream();
miStream
.filter((c)->c.getNombre().startsWith("m"))
.map((c)->c.getApellidos()).forEach(System.out::println);
```

En este caso nos quedamos con los apellidos de las personas cuyo nombre empieza por la letra m.

```
[EL Info]:
[EL Fine]:
alvarez
lopez
```

La verdad es que no es mucha ganancia ya que esto se podría haber realizado desde la base

de datos directamente de una forma mucho más eficiente. Sin embargo hay situaciones en las que las bases de datos no son tan flexibles porque están sobrecargadas. Por ejemplo podríamos seleccionar los datos de la tabla y devolverlos ordenados por un campo que no es indice. Hay situaciones en las cuales las bases de datos ya están muy exprimidas y te aconsejan no usar claúsulas order by.



En este caso si podríamos usar los Streams y seleccionar los datos de forma plana para a posteriori ordenarlos con un Stream.

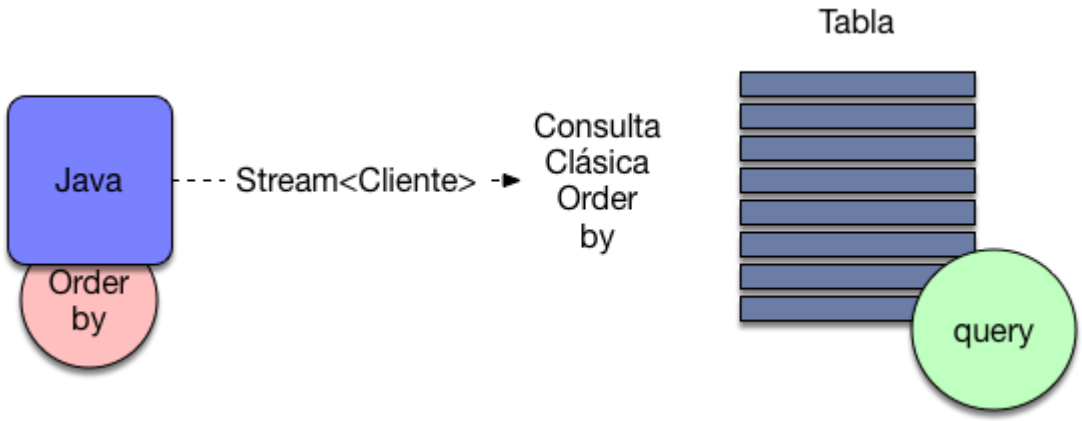
```
Stream<Cliente> miStream= consulta.getResultStream();
miStream.sorted((c1,c2)->c1.getApellidos().compareTo(c2.getApellidos()
)).forEach((c)-> {
                        System.out.printf(" %s %s %s %s %n",
c.getDni(), c.getNombre(), c.getApellidos(), c.getEdad());

                });
```

El resultado será:

```
124 manuel alvarez 50
456 gema blanco 20
789 alfonso diaz 40
667 mar lopez 50
234 pedro perez 20
567 vanesa perez 20
```

Acabamos de ordenar la lista descargando de esfuerzo a la base de datos utilizando un JPA Stream.



No solo eso sino que podríamos incluso pedirle al programa Java que paralelize esa ordenación con la siguiente consulta.

```
miStream.parallel().sorted((c1,c2)->c1.getApellidos().compareTo(c2.get
Apellidos())).forEach((c)-> {
                System.out.printf(" %s %s %s %s %n", c.getDni(),
c.getNombre(), c.getApellidos(), c.getEdad());
});
```

Lo cual aceleraría el rendimiento de la ordenación a nivel de Java.

Otros artículos relacionados:

1. Java Generic Repository y JPA
2. JPA Criteria API , un enfoque diferente
3. JPA Database Schema y automatización
4. JPA DTO (Data Transfer Object) y JPQL

Otros artículos externos:

- JPA Wiki