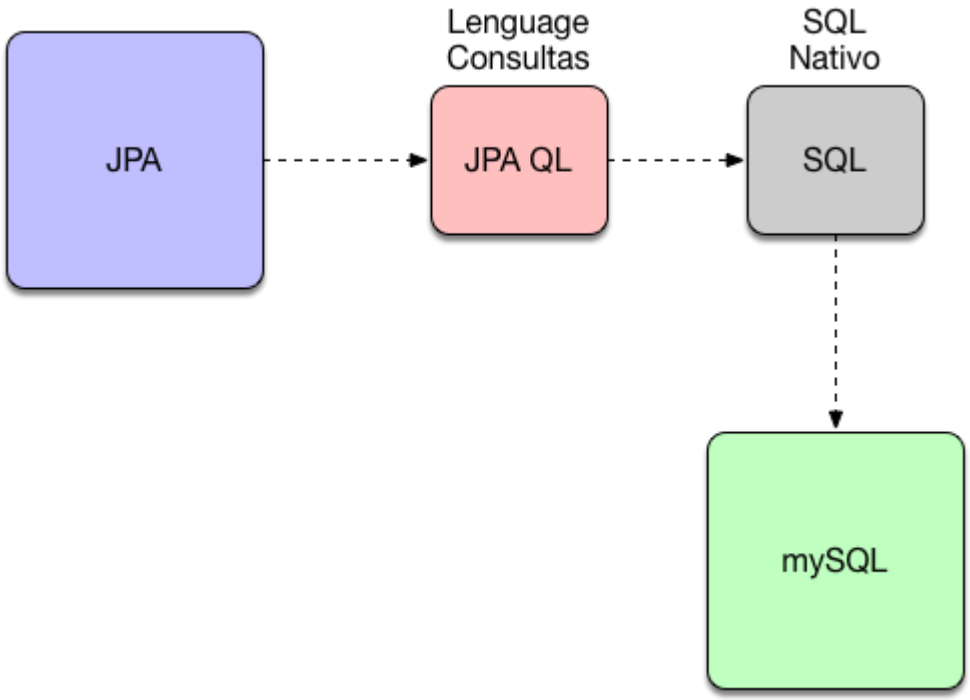
Usar Java LinQ nos puede parecer extraño ya que es una tecnología que nace con .NET framework . .NET nos permite usar programación funcional con los frameworks y soluciones de persistencia realizando una validación de las consultas en tiempo de compilación. ¿Existe alguna solución similar en Java? . La respuesta a esta pregunta es peculiar. Java 8 es posterior a JPA 2.0. Por lo tanto aunque Java soporta programación funcional , JPA en estos momentos no tiene integración con ella.



Existen dos opciones, esperar a Java 9 o usar una librería adicional que nos aporte la funcionalidad. En este caso vamos a usar JinQ como librería para añadir programación funcional a JPA y sus consultas. El primer paso es configurar las dependencias de Maven e incluir JPA y JlinQ.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <groupId>com.arquitecturajava</groupId>
        <artifactId>linqjava</artifactId>
        <version>0.0.1-SNAPSHOT</version>

        <dependencies>

                <dependency>
                        <groupId>org.jinq</groupId>
                        <artifactId>jinq-jpa</artifactId>
                        <version>1.8.19</version>
                </dependency>
                <dependency>
                        <groupId>org.jinq</groupId>
                        <artifactId>api</artifactId>
                        <version>1.8.19</version>
                </dependency>
                <dependency>
<groupId>org.hibernate.javax.persistence</groupId>
                        <artifactId>hibernate-jpa-2.1-api</artifactId>
                        <version>1.0.0.Final</version>
                </dependency>
                <dependency>
                        <groupId>org.hibernate</groupId>
                        <artifactId>hibernate-core</artifactId>
                        <version>5.2.10.Final</version>
                </dependency>
```

```
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.6</version>
		</dependency>
	</dependencies>

</project>
```

El siguiente paso es definir el fichero de persistence.xml :

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"
version="2.0">

	<persistence-unit name="arquitecturajava">
		<properties>
		<property name = "hibernate.show_sql" value = "true"
/>
			<property name="hibernate.dialect"
value="org.hibernate.dialect.MySQLDialect" />
			<property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver" />
			<property name="javax.persistence.jdbc.user"
value="root" />
			<property
name="javax.persistence.jdbc.password" value="mysql" />
			<property name="javax.persistence.jdbc.url"
```

```
value="jdbc:mysql://localhost:3306/pruebas" />
                </properties>
        </persistence-unit>

</persistence>
```

Es momento de construir el código Java :

```
package com.arquitecturajava;

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class Persona {

        @Id
        private String nombre;
        private String apellidos;
        private int edad;
        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;
	}
}
```

Por último diseñamos el programa principal que se conecta a la base de datos y nos selecciona los apellidos de “pepe”.

```
package com.arquitecturajava;

import java.util.List;

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

public class Principal {

	public static void main(String[] args) {
		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("arquitecturajava");
		EntityManager em = emf.createEntityManager();
		TypedQuery<Persona> consulta=em.createQuery("select p
```

```
from Persona p where p.nombre='pepe'",Persona.class);
                List<Persona> lista=consulta.getResultList();
                for(Persona p: lista) {
                        System.out.println(p.getApellidos());
                }
        }

}
```

Todo funciona correctamente y vemos el apellido de pepe imprimirse en la consola.

```
may 17, 2017 12:09:19 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator i
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.nombre as nombre1_0_, persona0_.apellidos as apellido2_0_
perez
```

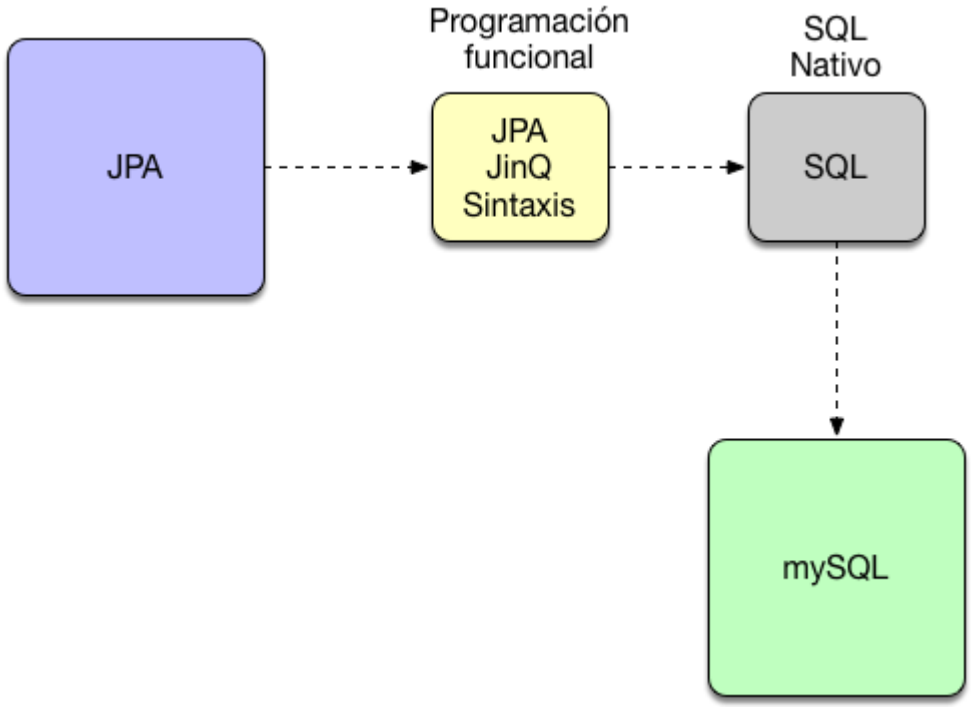
Java LinQ con la libreria JinQ

Sin embargo sería mucho más cómodo trabajar con una tecnología tipo LinQ que diseñe las consultas apoyándose en programación funcional. El compilador nos avisara de cualquier error de sintaxis .Es momento de ver el mismo ejemplo pero usando Java LinQ con JinQ como librería de apoyo.

```
import java.util.List;

import javax.persistence.EntityManager;
```

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

import org.jinq.jpa.JinqJPAStreamProvider;
import org.jinq.orm.stream.JinqStream;

public class Principal2 {

	public static void main(String[] args) {

		EntityManagerFactory emf =
Persistence.createEntityManagerFactory("arquitecturajava");
		EntityManager em = emf.createEntityManager();

		JinqJPAStreamProvider streams = new
JinqJPAStreamProvider(emf);
		JinqStream<Persona> personas = streams.streamAll(em,
Persona.class);
		List<String>
lista=personas.where(p->p.getNombre().equals("pepe")).select(p->p.getA
pellidos()).toList();
		for(String apellidos:lista) {
			System.out.println(apellidos);
		}
	}

}
```

Acabamos de diseñar al consulta con programación funcional utilizando las claúsulas Select y Where de JinQ .



El resultado es muy similar.

```
INFO: HHH10001003: Autocommit mode: false
may 17, 2017 12:13:45 PM org.hibernate.engine.jdbc.connections.internal.PooledConnections <init>
INFO: HHH000115: Hibernate connection pool size: 20 (min=1)
may 17, 2017 12:13:46 PM org.hibernate.dialect.Dialect <init>
INFO: HHH000400: Using dialect: org.hibernate.dialect.MySQLDialect
may 17, 2017 12:13:46 PM org.hibernate.engine.jdbc.env.internal.LobCreatorBuilderImpl useContextualLobCreation
INFO: HHH000423: Disabling contextual LOB creation as JDBC driver reported JDBC version [3] less than 4
may 17, 2017 12:13:46 PM org.hibernate.hql.internal.QueryTranslatorFactoryInitiator initiateService
INFO: HHH000397: Using ASTQueryTranslatorFactory
Hibernate: select persona0_.apellidos as col_0_0_ from Persona persona0_ where persona0_.nombre='pepe' limit ?
perez
```

Java comprobará en tiempo de compilación que la sintaxis es correcta y diseñara la consulta SQL más apropiada.

Otros artículos relacionados:

1. Un ejemplo de JPA Entity Graph
2. Introducción a Spring Data y JPA

3. JPA Entity Listener