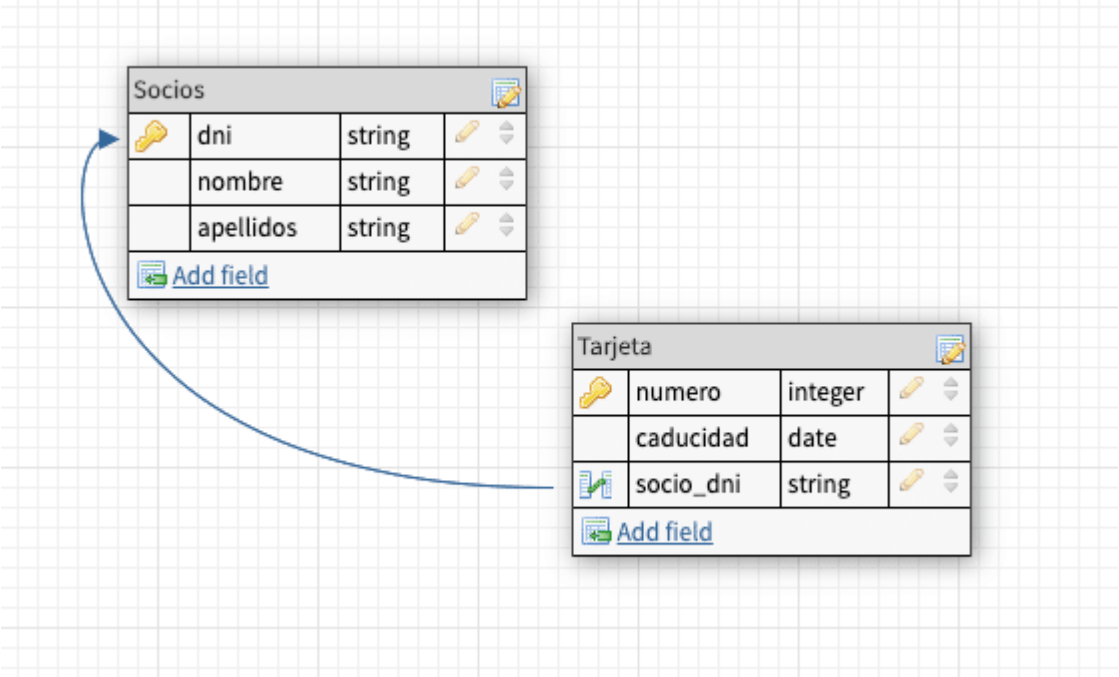
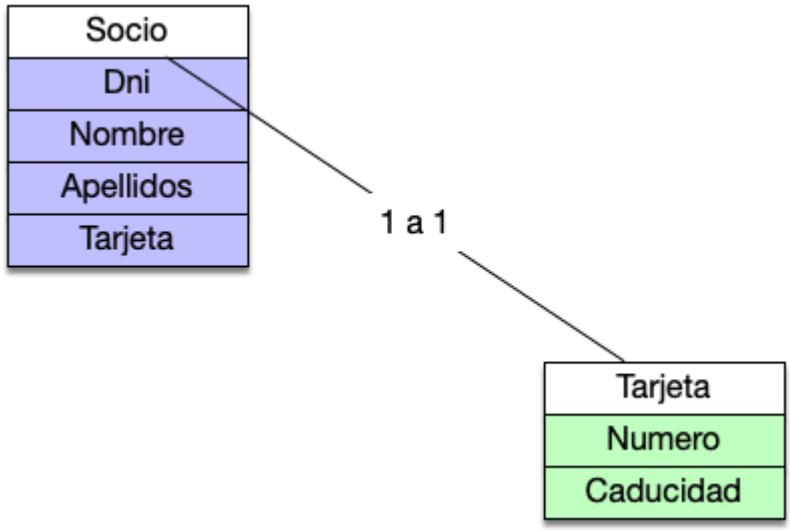
JPA @OneToOne , en muchas ocasiones nos encontramos trabajando con JPA y construyendo relaciones @OneToMany @ManyToOne ya que son las más habituales pero nos olvidamos de las relaciones @OneToOne que aunque no son las más habituales existen y tienen casuísticas bastante comunes . Imaginémonos un gimnasio en el cual cada Socio tiene una tarjeta que le permite el acceso . Estamos ante una relación 1 a 1 ya que cada socio tiene asignada su tarjeta y cada tarjeta pertenece a un Socio.



Claro en un modelo entidad relación esta relación no queda tan clara ya que siempre vendrá definida por una clave externa. En el modelo Entidad relación deberíamos apoyarnos en el uso de indices para forzarla. Eso es algo que no ocurre en un modelo de dominio en el cual la relación 1 a 1 esta claramente diferenciada.



JPA @OneToOne

Vamos a construir la relación con Java Persistence API . Lo primero que necesitamos es definir el conjunto de dependencias de Maven en nuestro proyecto.

```
<dependencies>

		<!--
https://mvnrepository.com/artifact/org.hibernate.javax.persistence/hib
ernate-jpa-2.1-api -->
		<dependency>
<groupId>org.hibernate.javax.persistence</groupId>
			<artifactId>hibernate-jpa-2.1-api</artifactId>
			<version>1.0.2.Final</version>
		</dependency>
		<!--
https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.4.12.Final</version>
```

```
		</dependency>
		<!--
https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>6.0.6</version>
		</dependency>
```

Una vez tenemos las dependencias de Maven claras es momento de definir el fichero de persistence.xml encargado de realizarla conexión a la base de datos

```
  <persistence-unit name="biblioteca" transaction-
type="RESOURCE_LOCAL">
	<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.url"
value="jdbc:mysql://localhost/biblioteca" />
		<property name="javax.persistence.jdbc.user" value="root"
/>
		<property name="javax.persistence.jdbc.password" value=""
/>
	</properties>
    </persistence-unit>
</persistence>
```

Ya disponemos de lo más básico es cuestión de definir ahora las relaciones a nivel de entidades. La primera clase que vamos a ver es la clase Socio esta incluirá una referencia a

la tarjeta utilizando la anotación @OneToOne.

Veamos el código:

```
package com.arquitecturajava.dominio;

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

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

@Entity
@Table(name="socios")
public class Socio {
        @Id
        private String dni;
        private String nombre;
        private String apellidos;
        @OneToOne(mappedBy="socio")
        private Tarjeta tarjeta;
        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 Tarjeta getTarjeta() {
        return tarjeta;
}
public void setTarjeta(Tarjeta tarjeta) {
        this.tarjeta = tarjeta;
}
public Socio(String dni, String nombre, String apellidos) {
        super();
        this.dni = dni;
        this.nombre = nombre;
        this.apellidos = apellidos;
}
public Socio() {
        super();
```

```
	}
	public Socio(String dni) {
		super();
		this.dni = dni;
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + ((dni == null) ? 0 :
dni.hashCode());
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		Socio other = (Socio) obj;
		if (dni == null) {
			if (other.dni != null)
				return false;
		} else if (!dni.equals(other.dni))
			return false;
		return true;
	}
}
```

Es momento de construir la otra parte de la relación que como vimos a nivel de tablas es la

de Tarjeta y contiene la clave foranea.

```
package com.arquitecturajava.dominio;

import java.util.Date;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.OneToOne;
import javax.persistence.Table;
@Entity
@Table(name="tarjetas")
public class Tarjeta {
	@Id
	private int numero;
	private Date caducidad;
	@OneToOne
	@JoinColumn(name="socios_dni")
	private Socio socio;
	public int getNumero() {
		return numero;
	}
	public void setNumero(int numero) {
		this.numero = numero;
	}
	public Date getCaducidad() {
		return caducidad;
	}
	public void setCaducidad(Date caducidad) {
		this.caducidad = caducidad;
```

```
        }
        public Tarjeta(int numero, Date caducidad) {
                super();
                this.numero = numero;
                this.caducidad = caducidad;
        }
        public Socio getSocio() {
                return socio;
        }
        public void setSocio(Socio socio) {
                this.socio = socio;
        }
        public Tarjeta() {
                super();
        }
        public Tarjeta(int numero) {
                super();
                this.numero = numero;
        }
}
```

Aquí podemos observar como a través de JoinColumn se genera la relación.

Acabamos de construir una relación JPA @OneToOne entre dos entidades de dominio. Estas relaciones nos puedan parecer en muchos momentos escasas . Pero son más habituales de

lo que uno cree por ejemplo Paciente y Historial Médico , o incluso más complejas como una Persona esta casada con otra Persona . En este caso estaríamos hablando de una relación OneToOne recursiva.

Otros artículos relacionados

- JPA @OneToMany
- JPA Fetch JOIN
- JPA @ManyToOne
- Curso JPA

Cecilio Álvarez Caules
Cecilio Álvarez Caules Oracle Java Certified Architect