Acabamos de ver como usar Java equals y hashcode

Spring Data Custom Query o consultas personalizadas con Spring Data es uno de los conceptos más habituales que aparecen cuando uno empieza a trabajar con profundidad con este framework. Para generar una consulta personalizada con Spring Data debemos usar la anotación @Query vamos a ver un ejemplo usando la clase Libro:

```
package com.arquitecturajava.data1;



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

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

@Entity
@Table(name="Libros")
public class Libro {

  @Override
  public String toString() {
    return "Libro [isbn=" + isbn + "," +titulo+","+ autor+"]";
  }

  @Id
  private String isbn;
  private String titulo;
```

```
private String autor;
private double precio;
private Date fecha;
public String getIsbn() {
  return isbn;
}
public void setIsbn(String isbn) {
  this.isbn = isbn;
}
public String getTitulo() {
  return titulo;
}
public void setTitulo(String titulo) {
  this.titulo = titulo;
}
public String getAutor() {
  return autor;
}
public void setAutor(String autor) {
  this.autor = autor;
}
public double getPrecio() {
  return precio;
}
public void setPrecio(double precio) {
  this.precio = precio;
}
public Date getFecha() {
  return fecha;
}
public void setFecha(Date fecha) {
```

```
  this.fecha = fecha;
}
public Libro() {
  super();
}

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

```
    return true;
  }
  public Libro(String isbn, String titulo, String autor, double
precio, Date fecha) {
    super();
    this.isbn = isbn;
    this.titulo = titulo;
    this.autor = autor;
    this.precio = precio;
    this.fecha = fecha;
  }
}
```

Normalmente cuando queremos hacer una consulta usamos alguno de los métodos que vienen por defecto a nivel de Repositorio de Spring Data usando NamedMethods .

```
public interface LibroRepository extends CrudRepository<Libro, String>
{

  List<Libro> findByTitulo(String titulo);
  List<Libro> findByAutor(String autor);

}
```

De esta forma se pueden realizar búsquedas relativamente rápidas y sencillas.

Spring Data Custom Query

Sin embargo hay situaciones en las que necesitamos métodos muy específicos y nos puede venir bien usar la anotación de @Query que nos permite diseñar el método a medida a través de JPA.

```
package com.arquitecturajava.data1;

import java.util.Date;
import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.PagingAndSortingRepository;

public interface LibroRepository extends CrudRepository<Libro, String>
{

  List<Libro> findByTitulo(String titulo);
  List<Libro> findByAutor(String autor);
  @Query ("select l from Libro l where l.precio>20")
  List<Libro> findByCaros();

  }
```

Como se puede ver en este caso estamos diseñando una consulta a medida . De la misma manera que podemos diseñar una consulta a medida usando JPA podemos generarla usando SQL plano y utilizando una consulta nativa contra la base de datos

```
@Query ("select * from Libros where precio>20", native=true)
List<Libro> findAllCaros();
```

Ambas consultas nos devolverán todos los libros cuyo precio es mayor que 20 . Ejecutamos una prueba unitaria

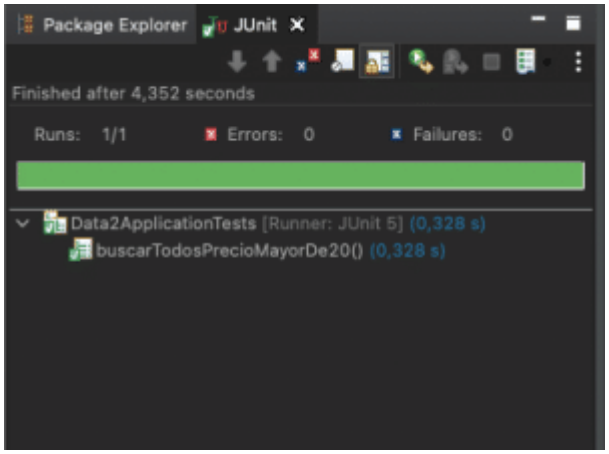
```
@Test
```

```
void buscarTodosPrecioMayorDe20() {

  List<Libro> lista=repositorio.findByCaros();
  assertThat(lista, hasItems(new Libro("7")));
  assertThat(lista, hasItems(new Libro("8")));

}
```

La prueba se ejecutará sin ningún problema con los datos precargados:



Otros artículos relacionados

1. Interface Segregation Principle y Spring Data
2. Spring Testing y el manejo de JUnit
3. Introducción a Spring Data y JPA
4. Spring Data