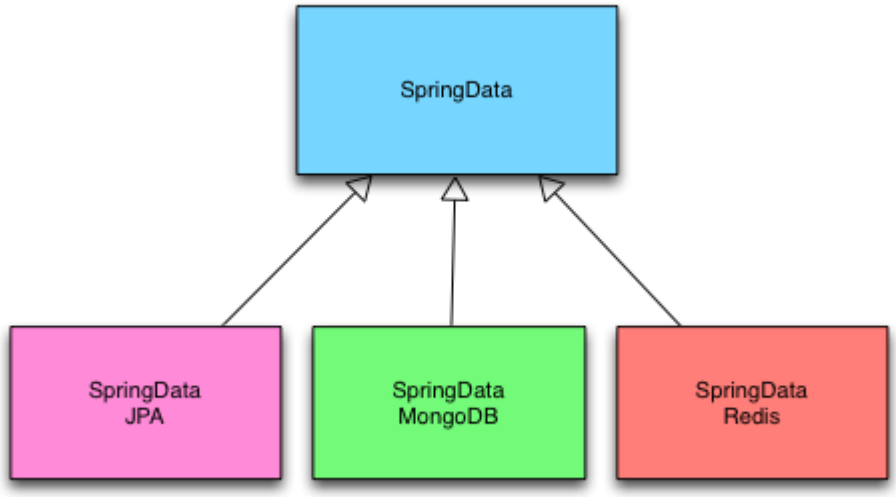
Acabamos de ver como usar Java equals y hashcode

Spring Data es uno de los frameworks que se encuentra dentro de la plataforma de Spring. Su objetivo es simplificar al desarrollador la persistencia de datos contra distintos repositorios de información .



Vamos a ver una introducción a este framework y como ayuda a simplificar nuestro trabajo. Nos apoyaremos en JPA para ello.

Spring Data y JPA

Es momento de configurar un proyecto utilizando JPA . Usaremos en un proyecto Maven creado a través de Spring Boot con todas sus dependencias , incluyendo las de Spring Data.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>springData</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>springData</name>
  <description>Demo project for Spring Boot</description>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.3.6.RELEASE</version>
    <relativePath />
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncod
ing>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
```

```
    </dependency>
    <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>5.1.39</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Visto el fichero de Maven, vamos a configurar los beans necesarios para acceder a una base de datos. En este caso nos vamos a centrar en un enfoque orientado a anotaciones puro.

```
package com.arquitecturajava;

import java.util.Properties;

import javax.sql.DataSource;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import
org.springframework.data.jpa.repository.config.EnableJpaRepositories;
```

```
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.jpa.JpaTransactionManager;
import
org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean;
import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;
import org.springframework.transaction.PlatformTransactionManager;
import
org.springframework.transaction.annotation.EnableTransactionManagement
;

@EnableJpaRepositories("com.arquitecturajava")
@EnableTransactionManagement
@ComponentScan("com.arquitecturajava")
public class SpringConfiguracion {

        @Bean
        public DataSource dataSource() {
                DriverManagerDataSource dataSource = new
DriverManagerDataSource();
dataSource.setDriverClassName("com.mysql.jdbc.Driver");
dataSource.setUrl("jdbc:mysql://localhost:3306/CursoJPA");
                dataSource.setUsername("root");
                dataSource.setPassword("mysql");
                return dataSource;
        }

        @Bean
        LocalContainerEntityManagerFactoryBean
entityManagerFactory(DataSource dataSource) {

                LocalContainerEntityManagerFactoryBean
```

```
entityManagerFactoryBean = new
LocalContainerEntityManagerFactoryBean();
                entityManagerFactoryBean.setDataSource(dataSource);
                entityManagerFactoryBean.setJpaVendorAdapter(new
HibernateJpaVendorAdapter());
entityManagerFactoryBean.setPackagesToScan("com.arquitecturajava");

                Properties jpaProperties = new Properties();

                jpaProperties.put("hibernate.dialect",
"org.hibernate.dialect.MySQLDialect");

entityManagerFactoryBean.setJpaProperties(jpaProperties);

                return entityManagerFactoryBean;

        }

        @Bean public PlatformTransactionManager transactionManager() {

                JpaTransactionManager txManager= new
JpaTransactionManager();
txManager.setEntityManagerFactory(entityManagerFactory(dataSource()).g
etObject());
                return txManager;

        }
}
```

Hemos utilizado la anotación @Bean para definir los diferentes beans, Datasource

EntityManagerFactory y TransactionManager. Es momento de crear una Entidad.

Spring Data y Entities

En este caso es bastante sencillo y basta con crear la clase Persona.

```
package com.arquitecturajava;

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

@Entity
public class Persona {

	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;
        }

        @Id
        private String dni;

        private String nombre;
        private String apellidos;

        public Persona(String dni, String nombre, String apellidos) {
                super();
                this.dni = dni;
                this.nombre = nombre;
                this.apellidos = apellidos;
        }

        public Persona() {
                super();
        }

}
```

El último paso es crear un interface de tipo Repositorio que nos permita gestionar las operaciones fundamentales:

```
import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
```

```
import com.arquitecturajava.Persona;

public interface PersonaRepository extends
JpaRepository&amp;amp;amp;lt;Persona,String&amp;amp;amp;gt;{
}
```

Se puede observar que el interface de repositorio para almacenar los objetos Persona extiende de JpaRepository. Esta es una clase que pertenece a SpringData. Es momento de usar Spring Framework , cargar los diferentes beans e invocar al método save del repositorio.

```
package com.arquitecturajava.repositories;
package com.arquitecturajava;

import org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.context.annotation.AnnotationConfigApplicationCont
ext;

import com.arquitecturajava.repositories.PersonaRepository;

@SpringBootApplication
public class SpringDataApplication {

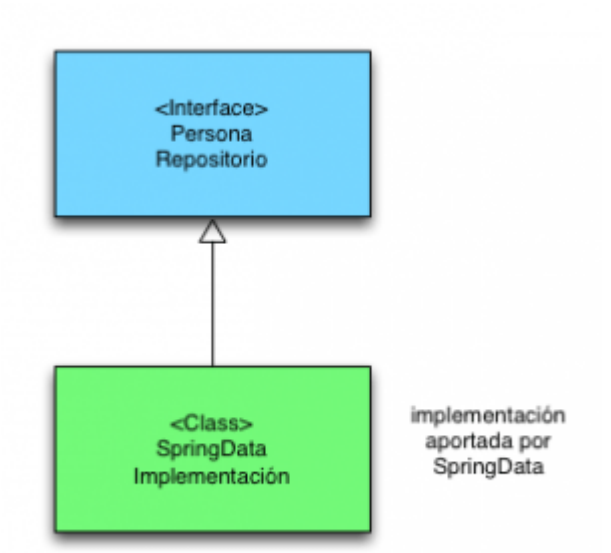
        public static void main(String[] args) {

                AnnotationConfigApplicationContext contexto=
                                new
```

```
AnnotationConfigApplicationContext(SpringConfiguracion.class);
                Persona p= new Persona("3","juan","gomez");
                PersonaRepository pRepositorio=
contexto.getBean(PersonaRepository.class);
                pRepositorio.save(p);

        }
}
```

Acabamos de almacenar una entidad en la base de datos. Todo ha sido muy sencillo. ¿En qué nos a ayudado Spring Data?. En mucho, no hemos tenido que implementar la clase de repositorio.



La mera declaración del interface ha sido suficiente para poder salvar la información. Al usar Spring Data la mayor parte del código que tendríamos que implementar **es aportado por el propio framework**. Sus ventajas y versatilidad son enormes.

Otros artículos relacionados :

- ¿Qué es Spring Boot?
- Java Override y encapsulación
- Tomcat context xml y su configuración
- JPA Join Fetch y su uso.