Spring Data JPA

Daniel Lastra R. (dlastra@ext.uc3m.es)

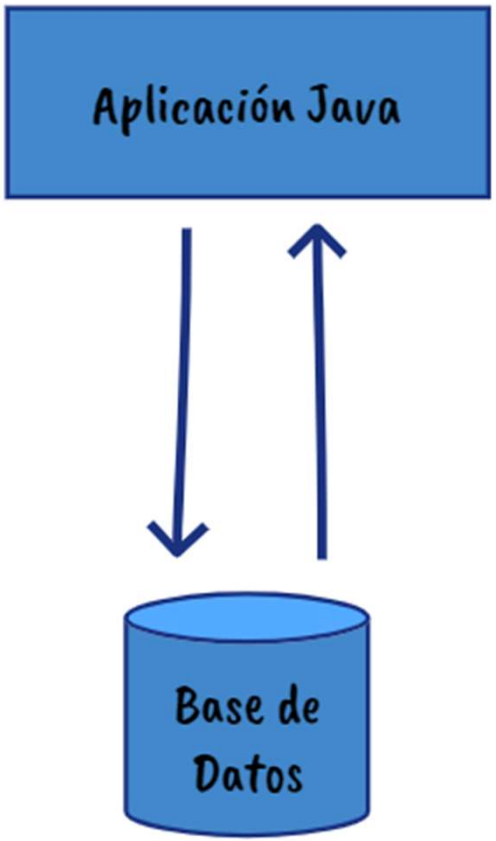
Computación Web (2024/25)

uc3m | Universidad Carlos III de Madrid
Departamento de Ingeniería Telemática

Índice

- Introducción
- Patrones de Diseño
- JDBC
- JPA
- Spring Data

Introducción

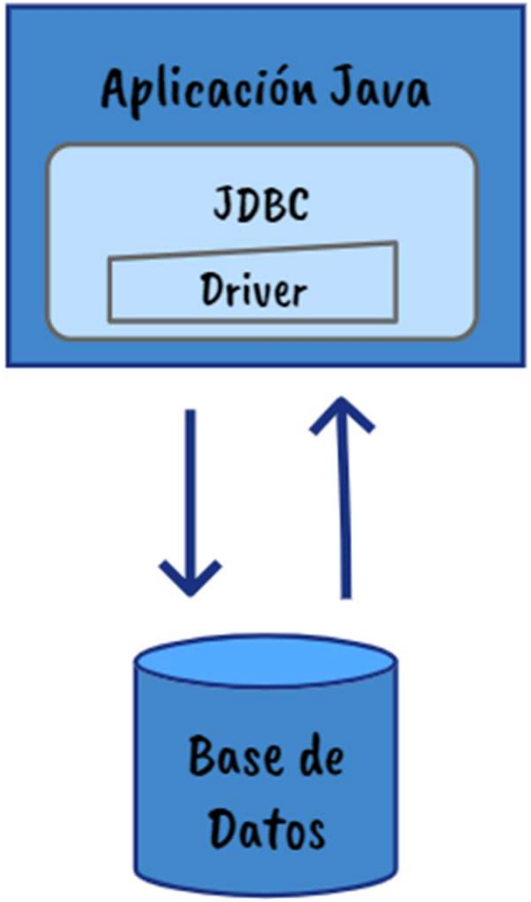


- La capa de persistencia:
  - Uno de los elementos del Patrón MVC
  - Operaciones básicas:
    - **C**reate
    - **R**ead
    - **U**pdate
    - **D**elete
  - Otros operaciones:
    - Búsquedas con paginación y ordenación
    - Queries personalizadas

Patrones de Diseño

- Active Record
  - Una clase que se encarga de definir la entidad y de implementar todas las operaciones de consulta y modificación de una tabla concreta de la base de datos.
- Data Access Object (DAO)
  - Una clase define la entidad y otra clase se encarga de implementar todas las operaciones de consulta y modificación de una tabla concreta de la base de datos.
- Repository
  - Una clase por objeto de dominio con el objetivo de desacoplar la lógica de negocio de la forma en la que administramos (persistimos, eliminamos, actualizamos o recuperamos) la información

JDBC



- La API de JDBC (Java Data Base Connection) permite acceder a bases de datos relacionales mediante SQL desde programas Java.
- JDBC es un estándar:
  - permite acceder de la misma forma a cualquier sistema gestor de bases de datos relacionales (Oracle, MySQL, etc.).

JDBC

- Las interfaces de JDBC están integradas en la API estándar de Java:
  - Paquete java.sql
  - Paquete javax.sql
- Pero se necesita adicionalmente un driver JDBC, que es una implementación de dichas interfaces:
  - Específico para cada programa gestor de base de datos.
  - Proporcionado habitualmente por el proveedor del gestor.
    - Para MySQL: MySQL Connector/J.

Principales componentes de JDBC

- **DriverManager:** gestiona el conjunto de controladores que están disponibles
  - getConnection(…)
- **Connection:** Carga lo necesario para establecer conexión: URL de la BD, usuario y contraseña.
  - prepareStatement(), prepareCall(), createStatement()
- **Statement:** Se utiliza para enviar consultas SQL simples sin parámetros.
  - executeQuery(), executeUpdate()
- **PreparedStatement:** permite crear consultas SQL con parámetros variables.
- **CallableStatement:** para ejecutar procedimientos almacenados SQL javax.sql.DataSource
- **ResultSet:** para obtener los resultados de una consulta SQL una BD.
  - next(), getString(), getInt(), getDate()
- **DataSource**: permite definir un pool de conexiones

Ejemplo

```
public static void main(String[] args) throws Exception {

    Connection connection = DriverManager
            .getConnection("jdbc:mysql://127.0.0.1:3306/database", user, pass);

    String consulta = "select * from Paises";
    PreparedStatement statement = connection.prepareStatement(consulta);
    ResultSet rs = statement.executeQuery();

    while (rs.next()) {
        System.out.println(rs.getInt(1));
        System.out.println(rs.getString(2));
    }
    rs.close();
    statement.close();

    connection.close();
}
```

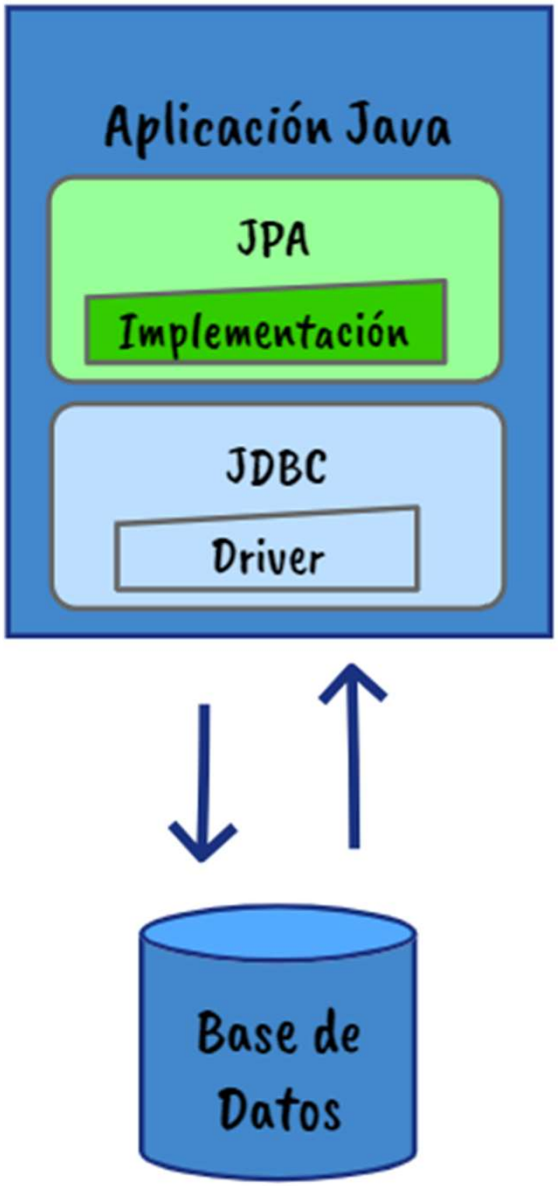
Transacciones

- Un objeto ***Connection*** define el contexto de las transacciones y proporciona métodos para su gestión:
  - *getAutoCommit*
  - *setAutoCommit*
  - *commit*
  - *rollback*

Reutilización de Conexiones

- Establecer una conexión con la base de datos supone un retardo y consumo de recursos en el cliente, base de datos y red
- En programas ejecutados en concurrencia (por ejemplo, aplicaciones web) es habitual mantener un pool de conexiones permanentemente abiertas y reutilizarlas
- La interfaz javax.sql.DataSource de JDBC Proporciona un mecanismo alternativo a DriverManager para obtener objetos Connection. Algunas implementaciones
  - ***hikari***
  - ***tomcat***
  - ***dbcp2***
  - ***c3p0***

JPA



- La API de Persistencia de Java (JPA) proporciona una interfaz estándar para entornos de persistencia de objetos.
- Ahora conocida como Jakarta Persistence API

Principales componentes de JPA

- **Persistence:** Clase que contiene los métodos estáticos que nos permiten obtener una instancia del EntityManagerFactory.
- **EntityManagerFactory:** Factoría que nos permite crear y administrar instancias del EntityManager.
- **EntityManager:** Interfaz cuya implementación controla las operaciones de persistencia sobre los objetos
- **Entity:** Permite gestionas los objetos que se almacenan como registros en una base de datos.
- **Persistence Unit:** Define el conjuto de todas las entidades que se almaceran en un único Sistema de Almacenamiento
- **EntityTransaction:** Maneja las transacciones a nivel de entidades
- **Query:** Interfaz implementado por cada implementación de JPA que permite ejecutar consultas sobre el Sistema de Almacenamiento

Entidades de dominio

```java
import javax.persistence.*;

@Entity
@Table(name = "paises")
public class Pais {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @ManyToOne
    @JoinColumn(name = "continente")
    private Continente continente;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;

    @Column(name = "gobierno")
    private String gobierno;

    @OneToOne
    @JoinColumn(name = "capital")
    private Ciudad capital;
```

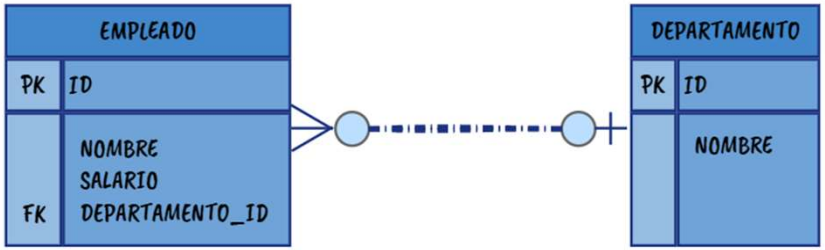
```java
@Entity
@Table(name = "ciudades")
public class Ciudad {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;
```

```java
@Entity
@Table(name = "continentes")
public class Continente {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;
```

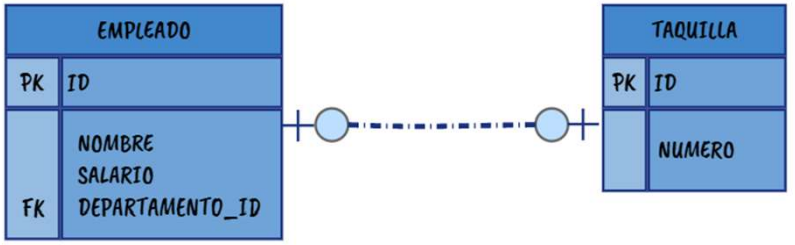
Relaciones entre Entidades

- **Relaciones Individuales:**

@ManyToOne

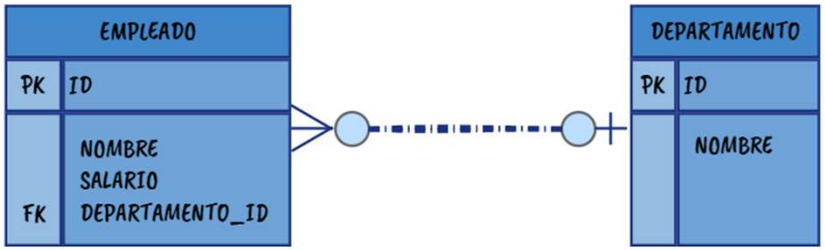


@OneToOne

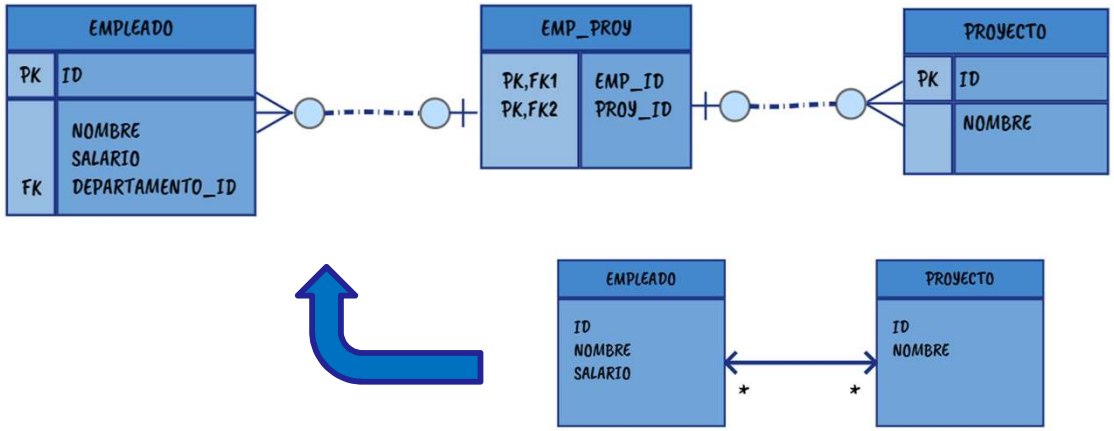


- **Relaciones de Colecciones:**

@OneToMany



@ManyToMany



Unidades de Persistencia

```xml
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
             http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
    version="2.1">

    <persistence-unit name="PERSISTENCE">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>

        <class>es.uc3m.it.model.Ciudad</class>
        <class>es.uc3m.it.model.Pais</class>

        <properties>
            <property name="javax.persistence.jdbc.driver" value="org.h2.Driver" />
            <property name="javax.persistence.jdbc.url" value="jdbc:h2:file:./testdb" />
            <property name="javax.persistence.jdbc.user" value="sa" />
            <property name="javax.persistence.jdbc.password" value="" />
            <property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect" />
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.show_sql" value="true" />
        </properties>

    </persistence-unit>

</persistence>
```

Ejemplo CRUD

```java
public static void main(String[] args) {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);

    EntityManager entityManager = factory.createEntityManager();
    entityManager.getTransaction().begin();

    Ciudad ciudad = new Ciudad();
    ciudad.setNombre("Leganes");
    ciudad.setPoblacion(180000);
    ciudad.setSuperficie(546783);

    entityManager.persist(ciudad);

    String jpql = "SELECT o FROM Ciudad o";
    Ciudad result = entityManager.createQuery(jpql, Ciudad.class).getSingleResult();
    System.out.println(result.getId());

    result = entityManager.find(Ciudad.class, result.getId());

    entityManager.remove(result);

    Number ciudades = entityManager.createQuery("SELECT COUNT(o) FROM Ciudad o", Long.class).getSingleResult();
    System.out.println(ciudades);

    entityManager.getTransaction().commit();
    entityManager.close();

    factory.close();

}
```

Ejemplo SQL

```
private static final String PERSISTENCE_UNIT_NAME = "PERSISTENCE";

public static void main(String[] args) {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);

    EntityManager entityManager = factory.createEntityManager();
    entityManager.getTransaction().begin();

    String sql = "select * from paises";
    List<Pais> result = (List<Pais>) entityManager.createNativeQuery(sql, Pais.class).getResultList();
    System.out.println(result);

    entityManager.getTransaction().commit();
    entityManager.close();

    factory.close();

}
```

Ejemplo JPQL

```
private static final String PERSISTENCE_UNIT_NAME = "PERSISTENCE";

public static void main(String[] args) {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);

    EntityManager entityManager = factory.createEntityManager();
    entityManager.getTransaction().begin();

    String jpql = "SELECT o FROM Pais o";
    List<Pais> result = (List<Pais>) entityManager.createQuery(jpql, Pais.class).getResultList();
    System.out.println(result);

    entityManager.getTransaction().commit();
    entityManager.close();

    factory.close();

}
```

Ejemplo Criterias

```
private static final String PERSISTENCE_UNIT_NAME = "PERSISTENCE";

public static void main(String[] args) {
    EntityManagerFactory factory = Persistence.createEntityManagerFactory(PERSISTENCE_UNIT_NAME);

    EntityManager entityManager = factory.createEntityManager();
    entityManager.getTransaction().begin();

    CriteriaBuilder criteriaBuilder = entityManager.getCriteriaBuilder();
    CriteriaQuery<Pais> criteriaQuery = criteriaBuilder.createQuery(Pais.class);
    Root<Pais> from = criteriaQuery.from(Pais.class);
    criteriaQuery.select(from);
    List<Pais> result = entityManager.createQuery(criteriaQuery).getResultList();
    System.out.println(result);

    entityManager.getTransaction().commit();
    entityManager.close();

    factory.close();

}
```

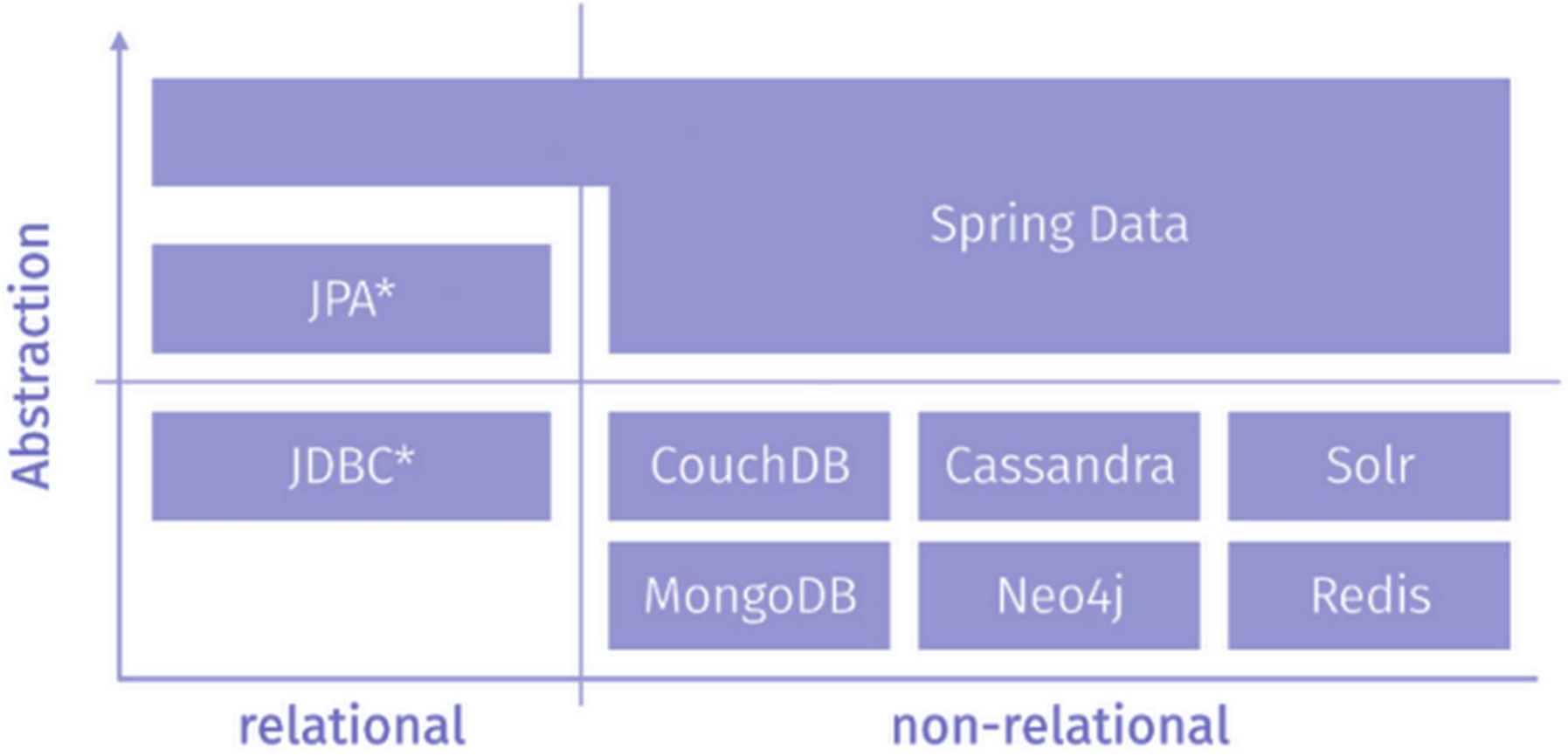
Implementaciones JPA

- ***DataNucleus***
- ***ObjectDB***
- ***Apache OpenJPA***
- ***EclipseLink***
- ***Hibernate***

Spring Data

- Framework de Spring
- Objetivo:
  - Simplificar el proceso de implementación de la **capa de persistencia** de una aplicación
- Basado en el patrón **Repository**
- Soporte para:
  - JDBC
  - JPA
  - MongoDB
  - Redis
  - Couchbase
  - Elasticsearch
  - …

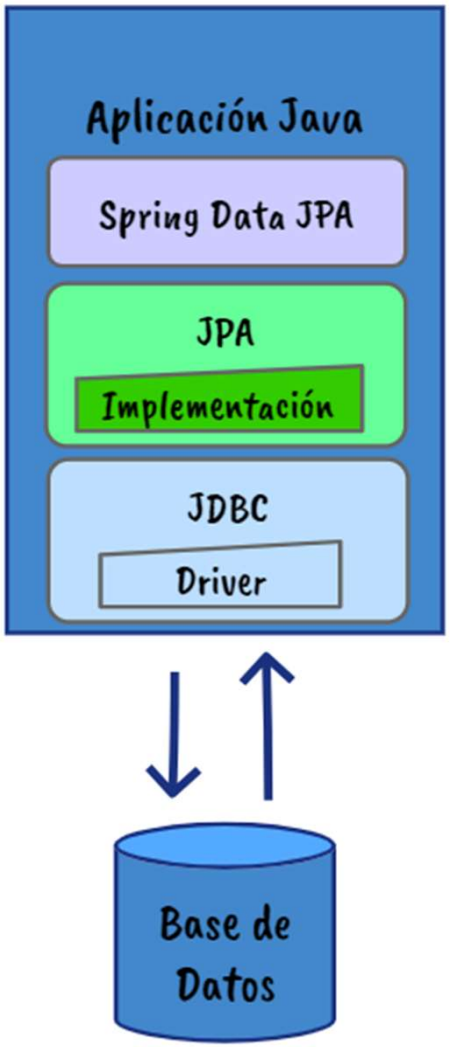
Abstracción



Características

- Alto nivel de abstracción para la implementación de repositorios y mapeo de objetos
- Construcción dinámica de queries a partir de la definición de los métodos
- Soporte transparante para auditoría:
  - creación
  - última modificación
- Posibilidad de intergrar código personalizado en los repositorios
- Fácil integración con spring mediante Java (anotaciones) y XML
- Integración avanzada con los controladores de Spring MVC

Spring Data JPA



- Spring Data JPA, es uno de los components de la familia Spring Data
- Facilita la implementación del patron repository basados en JPA

Configuración

- Driver JDBC
- Pool de Conexiones
- Implementación JPA
- Si utilizamos Spring Boot, sólo tendremos que indicar el driver JDBC, ya que, por defecto, incluye:
  - Hikari (implementación del pool de conexiones)
  - Hibernate (implementación de JPA)

```
spring.datasource.url=jdbc:h2:file:./testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

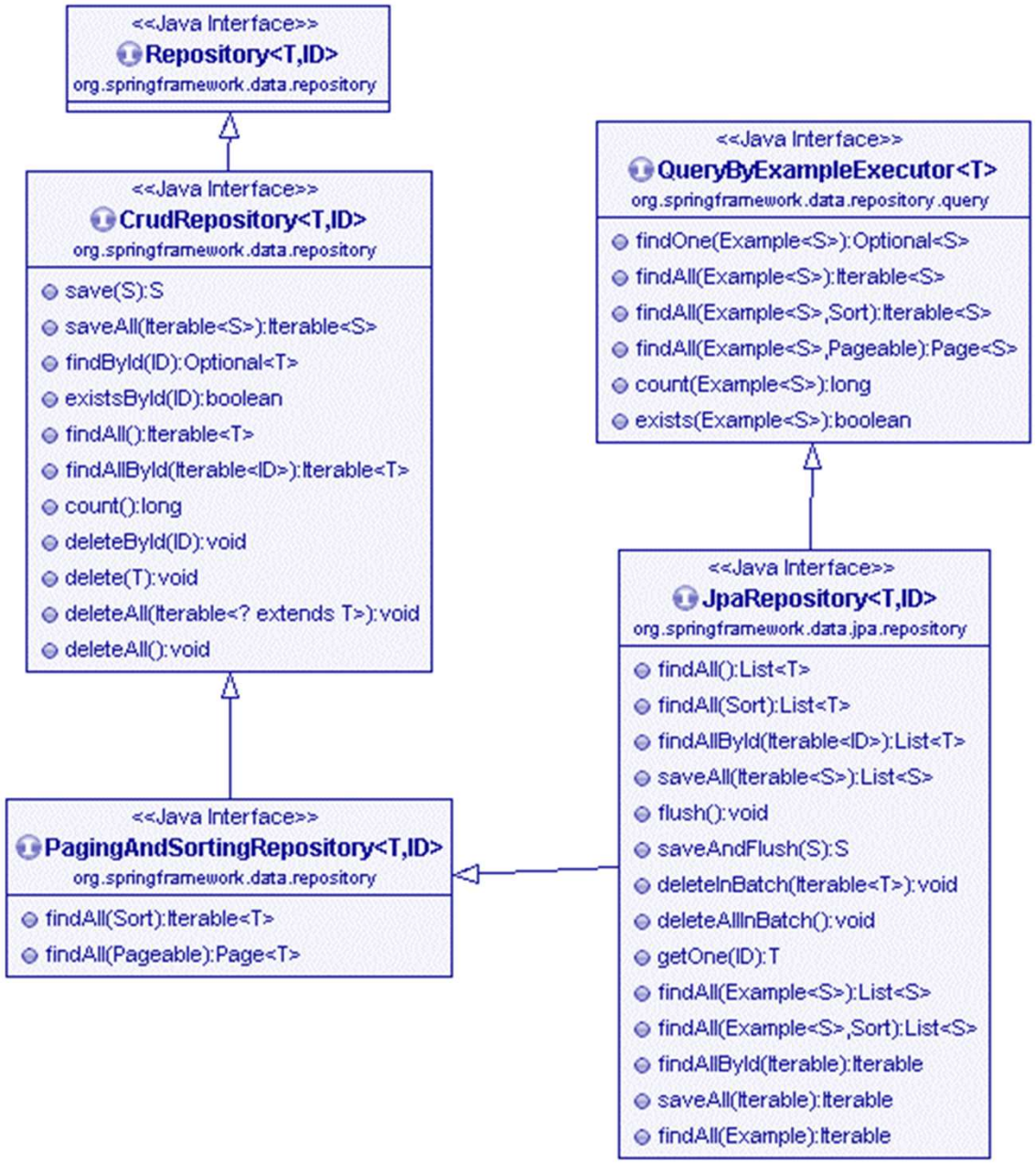
Entidades de dominio

```java
import javax.persistence.*;

@Entity
@Table(name = "paises")
public class Pais {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @ManyToOne
    @JoinColumn(name = "continente")
    private Continente continente;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;

    @Column(name = "gobierno")
    private String gobierno;

    @OneToOne
    @JoinColumn(name = "capital")
    private Ciudad capital;
```

```java
@Entity
@Table(name = "ciudades")
public class Ciudad {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;
```

```java
@Entity
@Table(name = "continentes")
public class Continente {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(name = "nombre")
    private String nombre;

    @Column(name = "superficie")
    private Number superficie;

    @Column(name = "poblacion")
    private Number poblacion;
```

Repositorios



- Los métodos de los repositorios son, por defecto, transaccionales
- En el caso de los métodos de consulta, la transacción se marca como ***readOnly***.

Consultas personalizadas

- Además de las operaciones derivadas del repositorios que extendamos, podemos implementar nuestras propias consultas
  - Derivadas a partir del nombre del método
  - Indicando la propia consulta

```java
public interface PaisRepository extends PagingAndSortingRepository<Pais, Long> {

    List<Pais> findByNombre(String nombre);

    List<Pais> findByNombreContainingIgnoreCase(String nombre);

    List<Pais> findByContinente(String continente);

    List<Pais> findByCapital(Ciudad capital);

    Pais findByCapital_Nombre(String nombre);

    Boolean existsByPoblacionGreaterThan(Number poblacion);

    @Query("select p from Pais p where p.poblacion > ?1")
    Pais findByPoblacionMayorQue(Number poblacion);

    @Query("select p from Pais p where p.superficie > :superficie")
    Pais findBySuperficieMayorQue(Number superficie);

}
```

Ejemplo

```
paisRepository.deleteAll();
ciudadRepository.deleteAll();
continenteRepository.deleteAll();

Continente europa = new Continente();
europa.setNombre("Europa");
europa.setSuperficie(10180000);
europa.setPoblacion(731000000);
continenteRepository.save(europa);

Continente america = new Continente();
america.setNombre("America");
america.setSuperficie(42330000);
america.setPoblacion(910000000);
continenteRepository.save(america);

Continente oceania = new Continente();
oceania.setNombre("Oceania");
oceania.setSuperficie(9008458);
oceania.setPoblacion(38889988);
continenteRepository.save(oceania);
```

```
Ciudad madrid = new Ciudad();
madrid.setNombre("Madrid");
madrid.setSuperficie(605.77);
madrid.setPoblacion(3141991);
ciudadRepository.save(madrid);

Ciudad barcelona = new Ciudad();
barcelona.setNombre("Barcelona");
barcelona.setSuperficie(102.15);
barcelona.setPoblacion(1604555);
ciudadRepository.save(barcelona);

Pais espana = new Pais();
espana.setNombre("España");
espana.setContinente(europa);
espana.setSuperficie(505370);
espana.setPoblacion(46438422);
espana.setGobierno("monarquia");
espana.setCapital(madrid);
paisRepository.save(espana);
```

```
System.out.println(paisRepository.findByCapital_Nombre("Madrid").getNombre()); // España
System.out.println(paisRepository.findByPoblacionMayorQue(40000000).getNombre()); // España
System.out.println(paisRepository.findBySuperficieMayorQue(500000).getNombre()); // España
System.out.println(paisRepository.existsByPoblacionGreaterThan(40000000)); // true
System.out.println(paisRepository.existsByPoblacionGreaterThan(80000000)); // false
```

Ordenación y Paginación

- Podemos incluir propiedades de ordenación y paginación a nuestras consultas gracias a Sort y Pageable

```java
public interface ContinenteRepository extends CrudRepository<Continente, Long> {

    List<Continente> findByNombreLike(String nombre, Sort sort);

    Page<Continente> findByNombreLike(String nombre, Pageable pageable);

}
```

```java
continenteRepository
    .findByNombreLike("%a%", Sort.by(Direction.ASC, "nombre"))
    .stream()
    .forEach(x -> System.out.println(x.getNombre())); // America, Europa, Oceania

continenteRepository
    .findByNombreLike("%a%", PageRequest.of(0, 2, Sort.by(Direction.ASC, "nombre")))
    .stream()
    .forEach(x -> System.out.println(x.getNombre())); // America, Europa
```

Referencias

- *Martin Fowler*. **Patterns of Enterprise Application Architecture**. Addison-Wesley Professional (2011)
  - Capítulo 10 (Active Record, Data Access Object)
  - Capítulo 13 (Repository)
- *Kishori Sharan*. **Java APIs, Extensions and Libraries: With JavaFX, JDBC, jmod, jlink, Networking, and the Process API. Apress** (2018).
  - Capítulo 5 (JDBC API)
- **Jakarta Persistence Specification Version 3.1** (2022)
  - https://jakarta.ee/specifications/persistence/3.1/
- **Spring Data JPA Reference Documentation Version 2.7.9** (2023):
  - https://docs.spring.io/spring-data/data-jpa/docs/2.7.9/reference/html