Capa de Persistencia

Java Persistence API (JPA) + Spring Data

Pablo Sánchez

Dpto. Ingeniería Informática y Electrónica
Universidad de Cantabria
Santander (Cantabria, España)
p.sanchez@unican.es

Advertencia

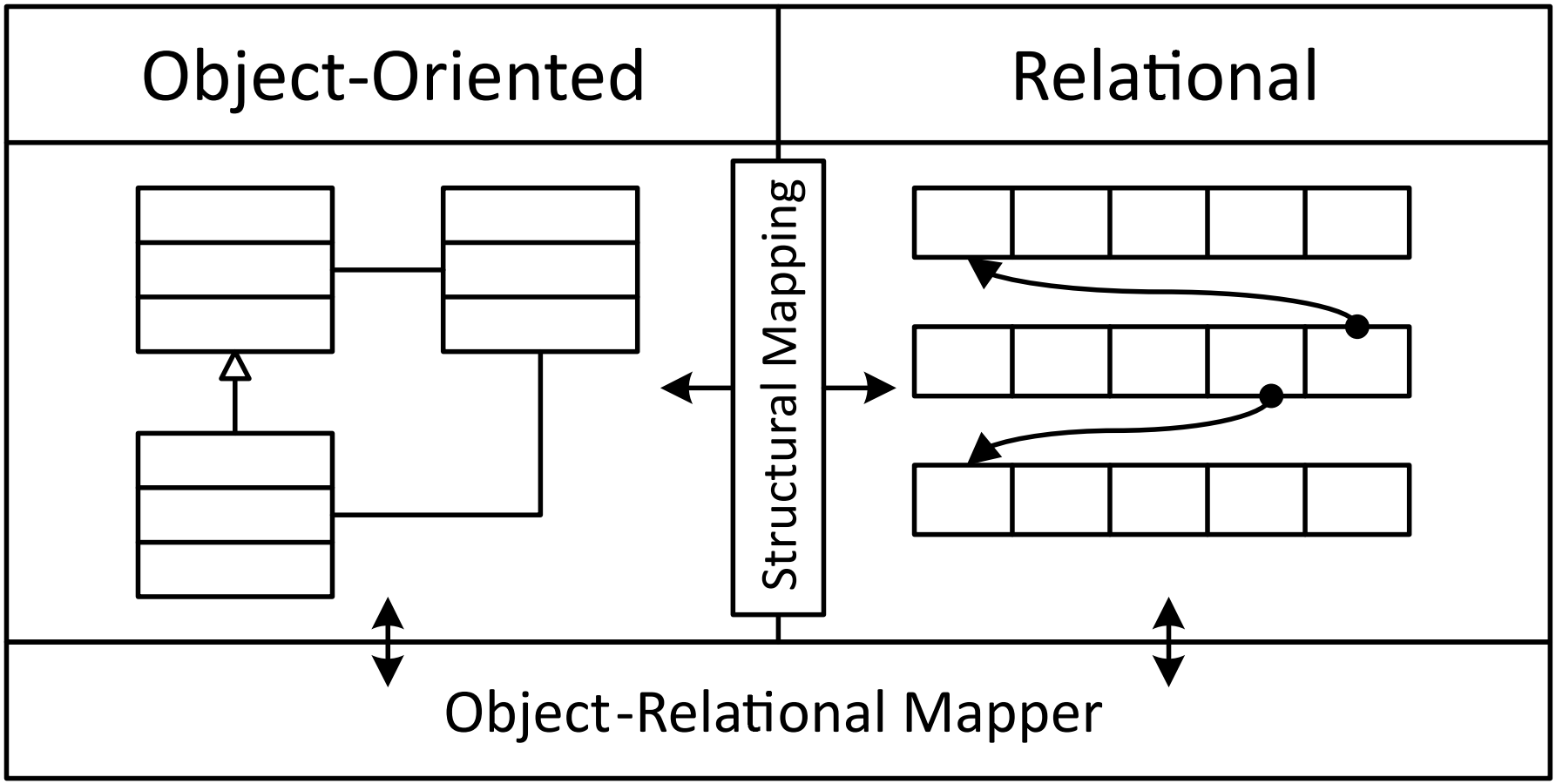
Todo el material contenido en este documento no constituye en modo alguno una obra de referencia o apuntes oficiales mediante el cual se puedan preparar las pruebas evaluables necesarias para superar la asignatura.

Este documento contiene exclusivamente una serie de diapositivas cuyo objetivo es servir de complemento visual a las actividades realizadas en el aula para la transmisión del contenido sobre el cual versarán las mencionadas pruebas evaluables.

Dicho de forma más clara, estas transparencias no son apuntes y su objetivo no es servir para que el alumno pueda preparar la asignatura.

Índice

Puentes de Persistencia de Objetos



JPA + Spring Data

JPA

Java Persistence API (JPA) es una especificación de referencia de un puente de persistencia estándar en Java.

1. Proporciona mecanismos para especificar el *metadata mapping*.
2. Especifica la interfaz y funcionamiento que deben tener una serie de elementos para realizar el acceso a datos (e.g. *EntityManager*).
3. Precisa de un ORM que la implemente (e.g. Hibernate).

Spring Data

Framework que proporciona facilidades para la definición de repositorios compatible con diversas tecnologías de persistencia, entre ellas JPA.

JPA + Spring Data

JPA

Java Persistence API (JPA) es una especificación de referencia de un puente de persistencia estándar en Java.

1. Proporciona mecanismos para especificar el *metadata mapping*.
2. Especifica la interfaz y funcionamiento que deben tener una serie de elementos para realizar el acceso a datos (e.g. *EntityManager*).
3. Precisa de un ORM que la implemente (e.g. Hibernate).

Spring Data

Framework que proporciona facilidades para la definición de repositorios compatible con diversas tecnologías de persistencia, entre ellas JPA.

JPA + Spring Data

JPA

Java Persistence API (JPA) es una especificación de referencia de un puente de persistencia estándar en Java.

1. Proporciona mecanismos para especificar el *metadata mapping*.
2. Especifica la interfaz y funcionamiento que deben tener una serie de elementos para realizar el acceso a datos (e.g. *EntityManager*).
3. Precisa de un ORM que la implemente (e.g. Hibernate).

Spring Data

Framework que proporciona facilidades para la definición de repositorios compatible con diversas tecnologías de persistencia, entre ellas JPA.

JPA + Spring Data

JPA

Java Persistence API (JPA) es una especificación de referencia de un puente de persistencia estándar en Java.

1. Proporciona mecanismos para especificar el *metadata mapping*.
2. Especifica la interfaz y funcionamiento que deben tener una serie de elementos para realizar el acceso a datos (e.g. *EntityManager*).
3. Precisa de un ORM que la implemente (e.g. Hibernate).

Spring Data

Framework que proporciona facilidades para la definición de repositorios compatible con diversas tecnologías de persistencia, entre ellas JPA.

JPA + Spring Data

JPA

Java Persistence API (JPA) es una especificación de referencia de un puente de persistencia estándar en Java.

1. Proporciona mecanismos para especificar el *metadata mapping*.
2. Especifica la interfaz y funcionamiento que deben tener una serie de elementos para realizar el acceso a datos (e.g. *EntityManager*).
3. Precisa de un ORM que la implemente (e.g. Hibernate).

Spring Data

Framework que proporciona facilidades para la definición de repositorios compatible con diversas tecnologías de persistencia, entre ellas JPA.

Objetivos del Tema

1. Ser capaz de transformar un conjunto de POJOs en un esquema relacional usando anotaciones JPA.
2. Ser capaz crear repositorios JPA y *Spring*.
3. Ser capaz de utilizar repositorios Spring para interactuar con la capa de persistencia.

Objetivos del Tema

1. Ser capaz de transformar un conjunto de POJOs en un esquema relacional usando anotaciones JPA.
2. Ser capaz crear repositorios JPA y *Spring*.
3. Ser capaz de utilizar repositorios Spring para interactuar con la capa de persistencia.

Objetivos del Tema

1. Ser capaz de transformar un conjunto de POJOs en un esquema relacional usando anotaciones JPA.
2. Ser capaz crear repositorios JPA y *Spring*.
3. Ser capaz de utilizar repositorios Spring para interactuar con la capa de persistencia.

Bibliografía


Bauer, C., King. G. y Gregory G. (2015).
Java Persistence with Hibernate. 2ª Ed.
Manning


Gierke, O., Darimont, T., Strobl, C., Paluch, M. y Bryant, J. (2019).
Spring Data JPA - Reference Documentation.
`https://goo.gl/Fhjdlu`

Índice

Índice

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML
   1. Permiten tener POJOs más limpios y reutilizables.
   2. Requieren ser actualizados cuando el modelo de dominio evoluciona.

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML
   1. Permiten tener POJOs más limpios y reutilizables.
   2. Requieren ser actualizados cuando el modelo evoluciona.

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML
   1. Permiten tener POJOs más limpios y reutilizables.
   2. Requieren ser actualizados cuando el modelo de dominio cambia,

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML
   1. Permiten tener POJOs más limpios y reutilizables.
   2. Requieren ser actualizados cuando el modelo de dominio cambia,

Metadata mapping

1. Se realiza mediante dos alternativas: anotaciones Java o basada en fichero XML.
2. Anotaciones Java
   1. Contaminan los POJOs, haciéndolos menos reutilizables y legibles.
   2. No requieren la actualización de ficheros externos cuando el modelo de dominio evoluciona.
3. Ficheros XML
   1. Permiten tener POJOs más limpios y reutilizables.
   2. Requieren ser actualizados cuando el modelo de dominio cambia,

Índice

Entidades y Value Objects

`@Entity`	La clase es una entidad y se transformará a una tabla.
`@Embeddable`	La clase es un *value object* y será incrustada en otra.

Índice

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basado en getters y setters.
   - Requiere de getters y setters publicos para todos los atributos.

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basado en getters y setters.
   - Requiere de getters y setters públicos para todos los atributos.

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basada en *getters* y *setters*.
   - Require de *getters* y *setters* públicos para todos los atributos.

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basada en *getters* y *setters*.
   - Require de *getters* y *setters* públicos para todos los atributos.

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basada en *getters* y *setters*.
   - Require de *getters* y *setters* públicos para todos los atributos.

Transformación de Propiedades

1. Basada en campos.
   - Se anotan los atributos a transformar.
   - La carga y el almacenamiento se hace basado en reflexión.
2. Basada en *getters*.
   - Se anotan los *getters* de los atributos a transformar.
   - La carga y almacenamiento se hace basada en *getters* y *setters*.
   - Require de *getters* y *setters* públicos para todos los atributos.

Transformación de Asociaciones

`@OneToOne`	Una referencia simple pertenece a una asociación 1-1.
`@ManyToOne`	Una referencia simple pertenece a una asociación 1-n.
`@Embedded`	Una referencia simple será incrustada.
`@OneToMany`	Una colección pertenece a una asociación 1-n.
`@ManyToMany`	Una colección pertenece a una asociación 1-1.
`@ElementCollection`	Un colección es de *value objects*.
`@Lob`	Una referencia se tratará como un LOB.
`@MapKey`	Permite controlar mejor la transformación de un mapa.

Transformación de Asociaciones - Detalles

`cascade`	Define qué operaciones deben propagarse a la clase referenciada.
`mappedBy`	En una asociación bidireccional, especifica el extremo que hará de clave externo.
`fetch`	Indica si la referencia se carga bajo demanda (`LAZY`) o no (`EAGER`).

Índice

Identity Field

`@Id`	El atributo marcado será la clave primaria.
`@GeneratedValue`	El atributo será una clave autogenerada.
`(strategy=...)`	Especifica la estrategia de generación.
`AUTO`	Se deja a decisión del ORM la elección.
`IDENTITY`	Generada por el gestor de pesistencia.
`SEQUENCE`	Usa una *secuencia* del gestor de persistencia.
`TABLE`	Generada por el propio ORM.
`@IdClass`	
`@EmbeededId`	Permiten crear claves primarias compuestas.

Índice

Transformación de Herencias

`@Inheritance`	Una clase es raíz de un árbol de herencia.
`(strategy=...)`	Especifica cómo se transforma un árbol de herencia.
`SINGLE_TABLE`	Aplica *Single Table Inheritance*
`JOINED`	Aplica *Class Table Inheritance*
`TABLE_PER_CLASS`	Aplica *Concrete Table Inheritance*

Índice

Opciones de Personalización y Optimización

`@Table`	Personaliza las tablas ligadas a una `@Entity`.
`@Column`	Personaliza las columnas ligadas a un atributo.
`@JoinColumn`	Personaliza las claves fóraneas en *Foreign Key*
`@JoinTable`	Personaliza las tabla intermedia en *Association Table*.
`@UniqueConstraint`	Especifica restricciones de unicidad adicionales.
`@DiscriminatorColumn`	Personaliza la columna *tipo* en *Single Table*.
`@DiscriminatorValue`	Personaliza los valores de *tipo* en *Single Table*.

Índice

Índice

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

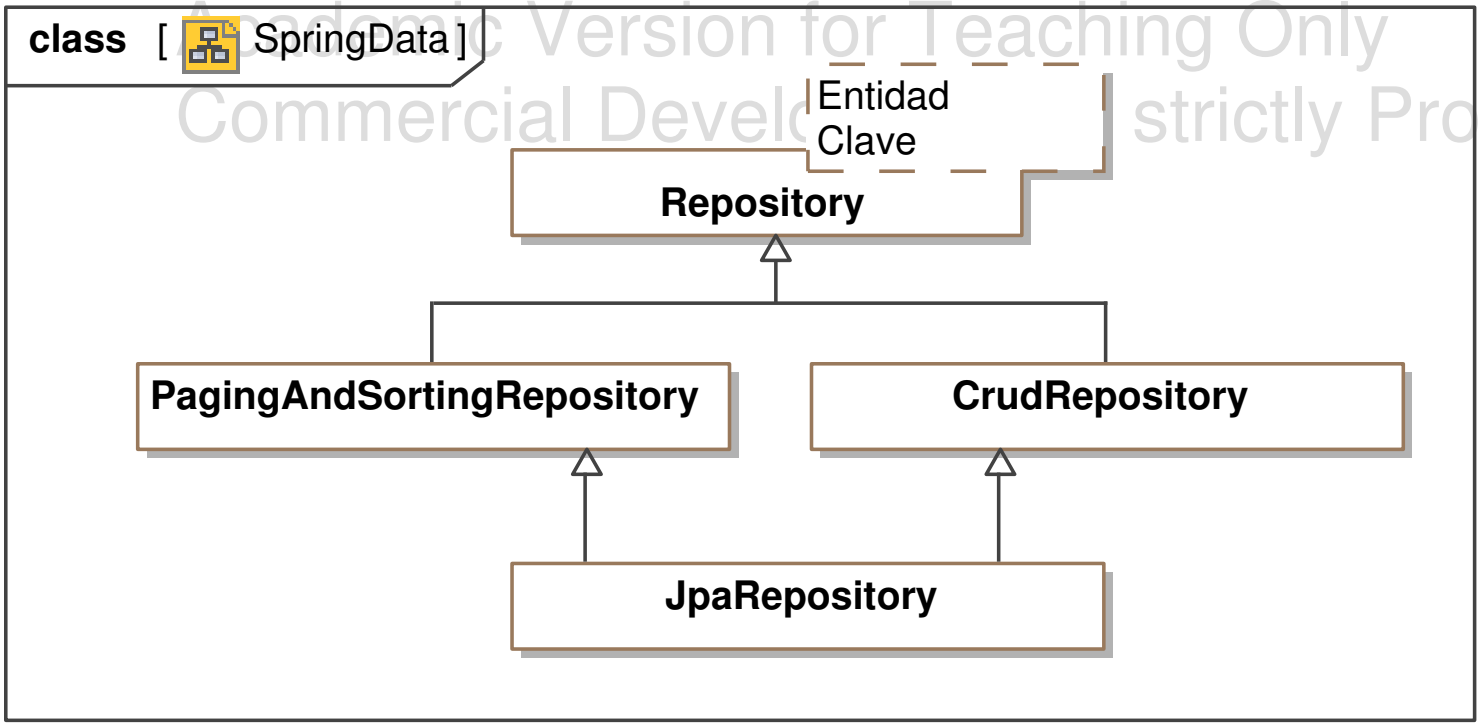
Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Spring Repositories

1. Facilidad que permite la generación automática de repositorios a partir de la extensión de una interfaz de alto nivel.
2. Soporte implícito para las operaciones básicas comunes.
3. Soporte avanzado para operaciones más específicas.
4. Facilidades de personalización avanzada.
5. Soporte para paginación, ordenado, *streams* y métodos asíncronos.
6. Soporte para *query by example*.
7. Soporte para *data transfer objects*.

Jerarquía de los Spring Repositories



Índice

Repositorios CRUD

```
public interface CrudRepository<T, ID extends Serializable>
  extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);}
```

Repositorios Page and Sorting

```
public interface PagingAndSortingRepository<T, ID extends
    Serializable>
  extends CrudRepository<T, ID> {

    Iterable<T> findAll(Sort sort);

    Page<T> findAll(Pageable pageable);
}
```

Repositorios JPA

```
public interface JpaRepository<T, ID extends Serializable>
    extends
  PagingAndSortingRepository<T, ID> {

    List<T> findAll();

    List<T> findAll(Sort sort);

    List<T> save(Iterable<? extends T> entities);

    void flush();

    T saveAndFlush(T entity);

    void deleteInBatch(Iterable<T> entities);
}
```

Crear un Spring Repository

```
public interface ViajeRepository extends
    JpaRepository<Viaje,Long>{}
```

Inyectar un Spring Repository JPA

```
public class ViajeService {

  @Autowired
  protected ViajeRepository vr;

  public boolean cerrarViaje(Long id) throws ViajeNotFound {

    Viaje v =
        vr.findById(id).orElseThrow(ViajeNotFound::new);

  }
}
```

Índice

Creación de Operaciones Personalizadas

1. Definidas a través de los nombres de los métodos.
2. Definidas a través de expresiones JPQL (Java Persistence Query Language).
3. Definidas a través de *especificaciones*.
4. Definidas a través de *Query By Example*.

Creación de Operaciones Personalizadas

1. Definidas a través de los nombres de los métodos.
2. Definidas a través de expresiones JPQL (Java Persistence Query Language).
3. Definidas a través de *especificaciones*.
4. Definidas a través de *Query By Example*.

Creación de Operaciones Personalizadas

1. Definidas a través de los nombres de los métodos.
2. Definidas a través de expresiones JPQL (Java Persistence Query Language).
3. Definidas a través de *especificaciones*.
4. Definidas a través de *Query By Example*.

Creación de Operaciones Personalizadas

1. Definidas a través de los nombres de los métodos.
2. Definidas a través de expresiones JPQL (Java Persistence Query Language).
3. Definidas a través de *especificaciones*.
4. Definidas a través de *Query By Example*.

Personalización mediante Nombres de Métodos

```
public interface UsuarioRepository extends
    JpaRepository<Usuario, String> {

        Usuario findByEmail(String email);

        Set<Usuario> findByFechaAltaAfter(Date fecha);

}
```

Personalización mediante Nombres de Métodos

Keyword	Ejemplo
`findBy,countBy`	`findByEmail`
`And, Or`	`findByOriginCiudadAndDestino`
`Like`	`findByOriginCiudadLike`
`IgnoreCase`	`findByOriginCiudadLikeIgnoreCase`
`Between`	`findByFechaBetween`
`LessThan`	`findByPrecioLessThan`
`After, Before`	`findByFechaBefore`
`Not`	`findByFechaBetweenAndNot`
`OrderBy(Desc)`	`findByPriceOrderByAsc`

Personalización mediante Nombres de Métodos

```
public interface ViajeRepository
                    extends JpaRepository<Viaje,Long>
{
   public Set<Viaje>
      findByOrigenCiudadAndDestinoCiudad(
            String ciudadOrigen,
            String ciudadDestino);

   public Set<Viaje>
      findByOrigen_CiudadAndFechaBeforeOrderByPrecio(
            String ciudad,
            Date fecha);
}
```

Personalización mediante JPQL

```
public interface ViajeRepository
                    extends JpaRepository<Viaje,Long> {

        @Query("SELECT v FROM Viaje v
                    WHERE v.origen.ciudad = ?1
                      AND v.destino.ciudad = ?2")
        public Set<Viaje> findByOrigenAndDestino(
                                            String origen,
                                            String destino);
}
```

Índice

¿Qué tengo que saber de todo ésto?

1. Ser capaz de utilizar anotaciones JPA para especificar una correspondencia objeto-relacional.
2. Ser capaz de definir repositorios *JPA* en *Spring*.
3. Ser capaz de utilizar repositorios *JPA* en *Spring*.

¿Qué tengo que saber de todo ésto?

1. Ser capaz de utilizar anotaciones JPA para especificar una correspondencia objeto-relacional.
2. Ser capaz de definir repositorios *JPA* en *Spring*.
3. Ser capaz de utilizar repositorios *JPA* en *Spring*.

¿Qué tengo que saber de todo ésto?

1. Ser capaz de utilizar anotaciones JPA para especificar una correspondencia objeto-relacional.
2. Ser capaz de definir repositorios *JPA* en *Spring*.
3. Ser capaz de utilizar repositorios *JPA* en *Spring*.