



DIPLOMADO
**Desarrollo de sistemas con
tecnología Java**

Módulo 8
Persistencia con Spring Data

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx

Convenios

- Tolerancia de inicio de clase 15 min
- 20 minutos de receso

Evaluación

- Practicas 30%
- Ejercicios 40%
- Avance de proyecto final 30%

Objetivo

- El participante será capaz de realizar la persistencia de datos a nivel avanzado a través del uso de Spring Data.

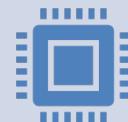
Spring Data



La misión de Spring Data es proporcionar un modelo de programación familiar y coherente basado en Spring para el acceso a los datos y al mismo tiempo, conservar las características especiales del almacenamiento subyacente.

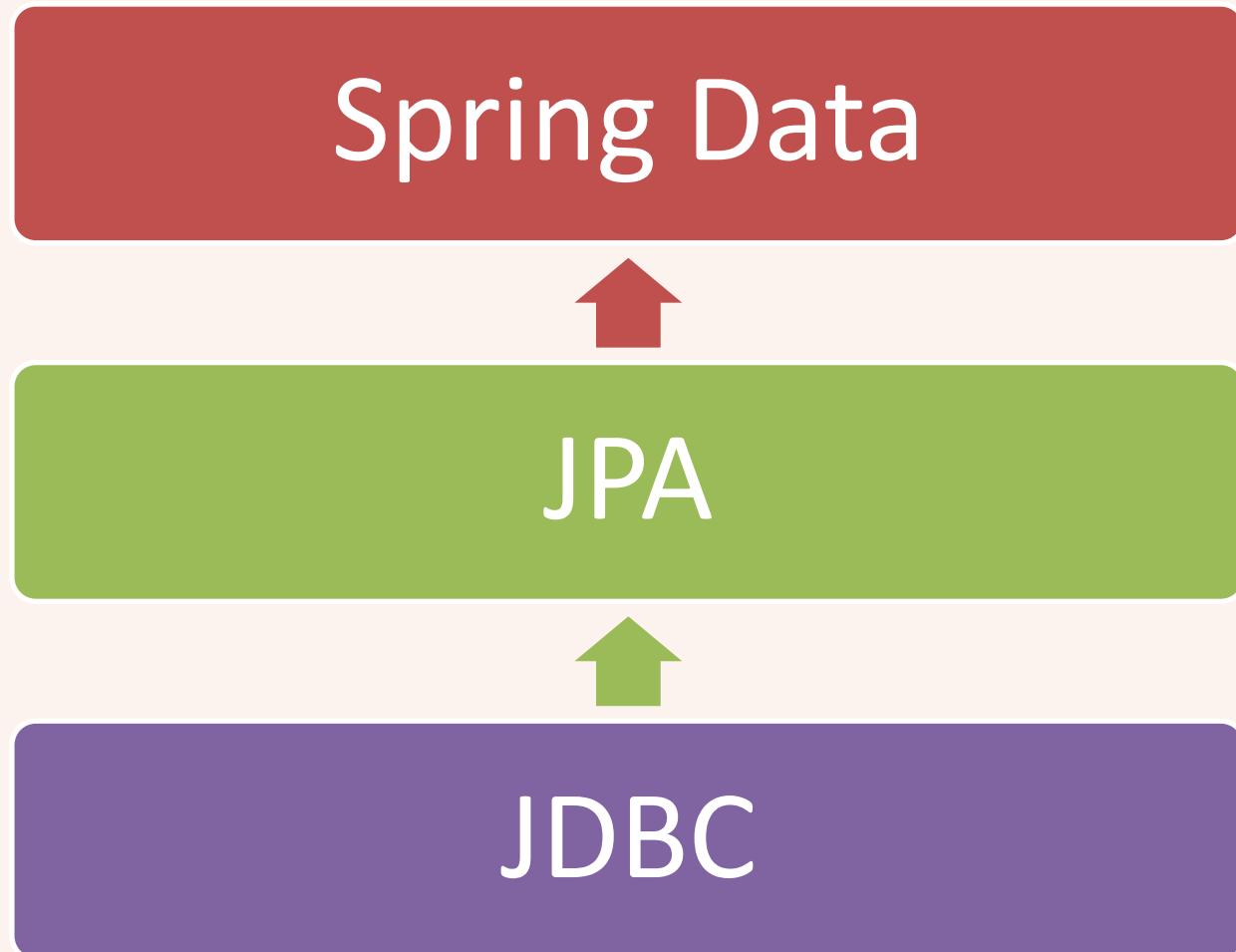


Facilita el uso de tecnologías de acceso a datos, bases de datos relacionales y no relacionales, marcos de reducción de mapas y servicios de datos basados en la nube.

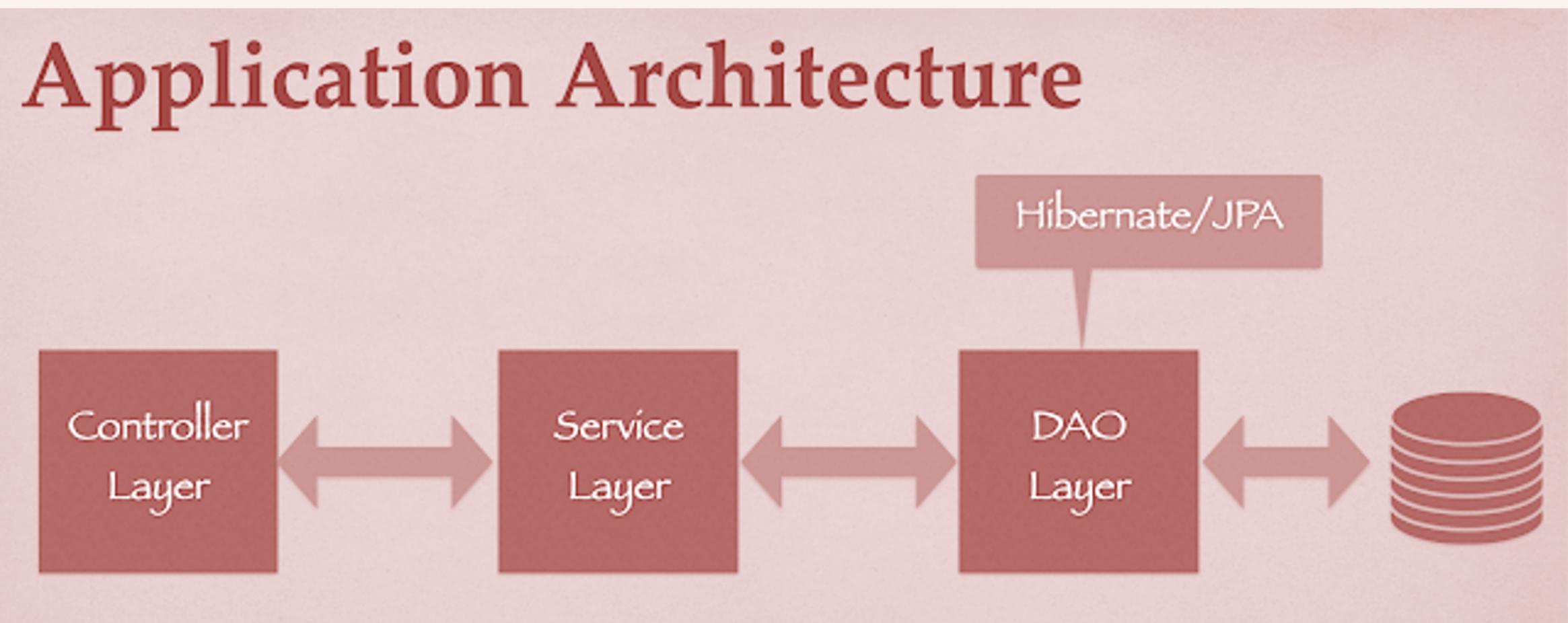


Es un proyecto general que contiene subproyectos específicos para diversas tecnologías de bases de datos, como JPA, MongoDB, Redis, entre otros.

Spring Data



DAO/Repository layer



Características Spring Data

Poderoso repositorio y abstracciones de mapeo de objetos personalizados

Derivación de consultas dinámicas a partir de nombres de métodos de repositorio

Implementación de clases base de dominio que proporcionan propiedades básicas.

Soporte para auditoría transparente (created, last changed)

Posibilidad de integrar código de repositorio personalizado

Fácil integración de Spring a través de JavaConfig y namespaces XML personalizados

Integración avanzada con controladores Spring MVC

Spring Data

Proporciona

Una solución para reducir una gran cantidad de código repetitivo.

Una implementación lista para usar para todas las operaciones **CRUD** requeridas para la entidad JPA.

Repositorios, por lo que solo es necesario extenderlos para obtener la implementación completa lista para usar para las operaciones CRUD para una entidad.

Módulos de integración Spring Data

Spring Data
Commons

Spring Data
JDBC

Spring Data
JDBC Ext

Spring Data
JPA

Spring Data
KeyValue

Spring Data
LDAP

Spring Data
MongoDB

Spring Data
Redis

Spring Data
REST

Spring Data
for Apache
Cassandra

Spring Data
for Apache

Spring Data
for Pivotal
GemFire

Spring Data Commons

- Proporciona una abstracción unificada para acceder a diversas fuentes de datos.
- Independientemente de la fuente de datos, el objetivo es proporcionar una forma de mapear las entidades de objetos Java a registros de la fuente de datos de destino y persistirlos, así como convertir esos registros nuevamente en entidades.
- Java Business Entities <--> Registros persistentes de almacenamiento de datos de destino
- Operaciones de CRUD
 - Búsqueda, Actualización y Eliminación de registros.

Spring Data Commons Interfaces

Repository<T, ID extends Serializable>

- La interfaz base que todas las demás interfaces de repositorio extienden.
- Proporciona una abstracción genérica para las entidades y sus identificadores.

CrudRepository<T, ID extends Serializable>

- Extiende Repository y agrega métodos CRUD estándar como save(), findById(), delete().

PagingAndSortingRepository<T, ID extends Serializable>

- Extiende CrudRepository y agrega métodos para paginación y ordenamiento de resultados, como findAll(Pageable pageable) y findAll(Sort sort).

QueryDslPredicateExecutor<T>

- Proporciona soporte para la construcción de consultas dinámicas mediante predicados en tiempo de ejecución usando QueryDSL.

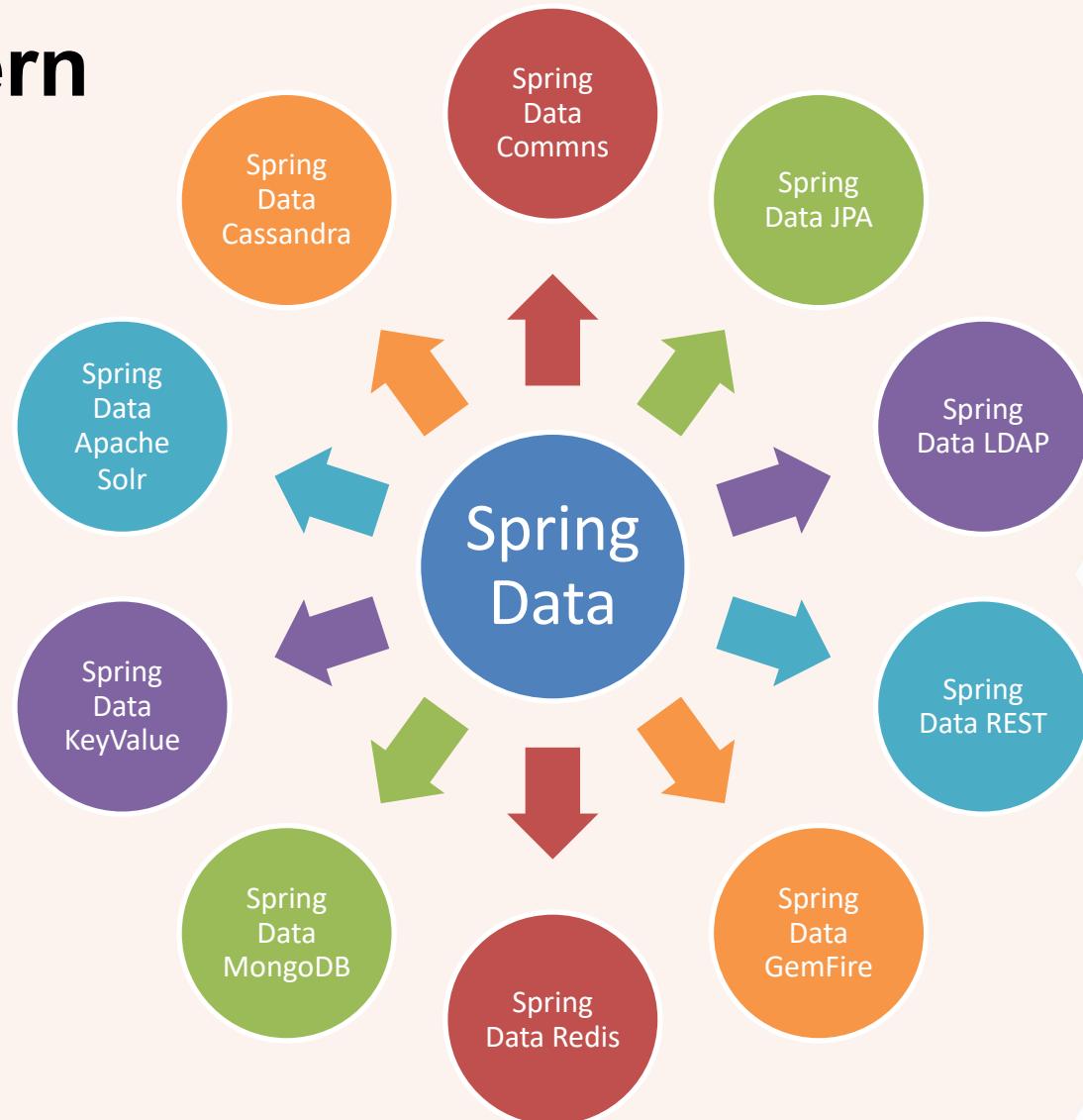
Spring Data Commons

- **CrudRepository<T, ID extends Serializable>**
 - Proporciona operaciones CRUD para la entidad gestionada.
 - Métodos
 1. *long count()*
 2. *void delete(T entity)*
 3. *void deleteAll()*
 4. *void deleteAll(Iterable<? extends T> entities)*
 5. *void deleteById(ID id)*
 6. *boolean existsById(ID id)*
 7. *Iterable findAll()*
 8. *Iterable findAllById(Iterable ids)*
 9. *Optional findById(ID id)*
 10. *save(S entity)*
 11. *Iterable saveAll(Iterable entities)*

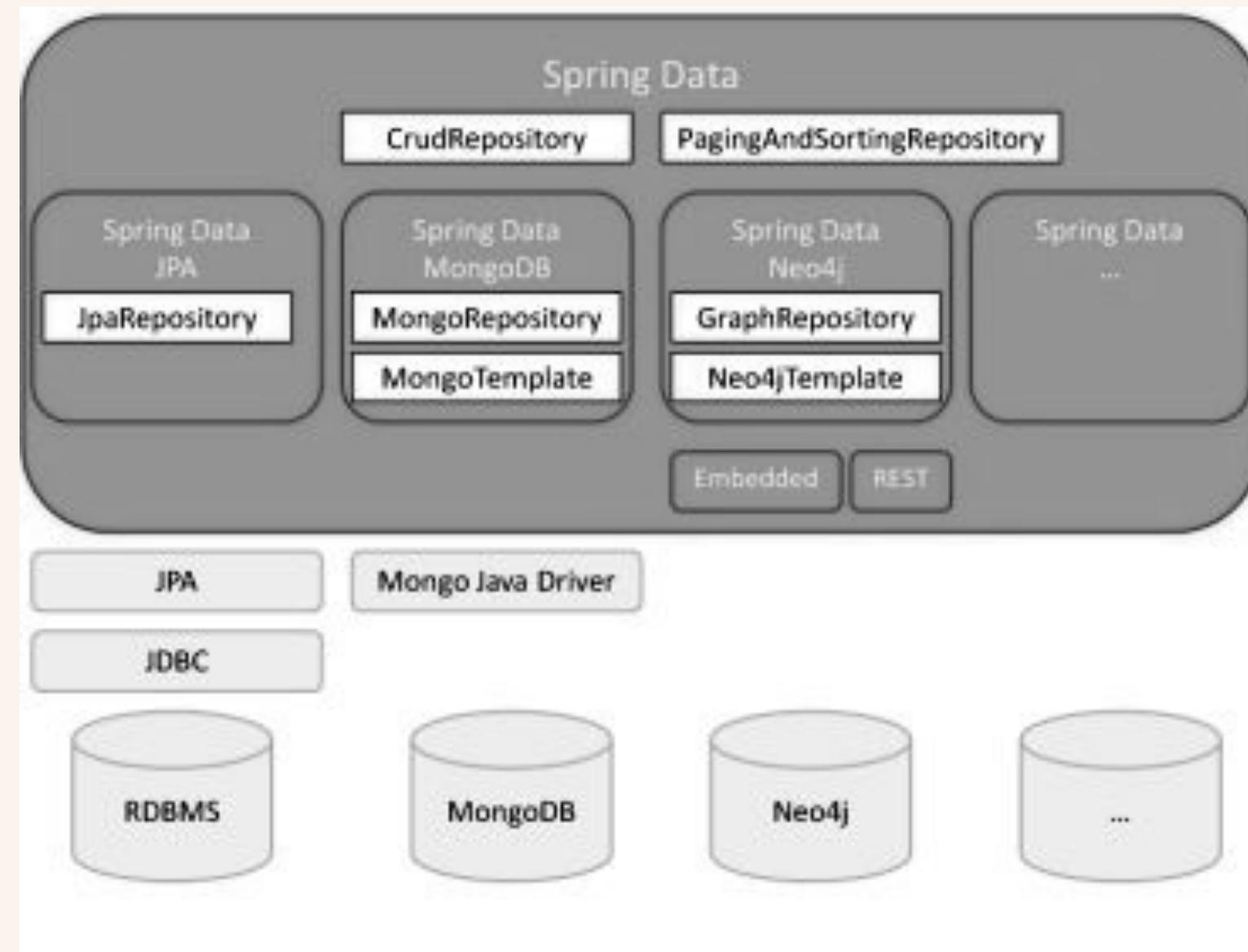
Repository pattern

- Es la abstracción utilizada por Spring Data Commons para encapsular la lógica de acceso a datos y proporcionar una interfaz coherente.
- El patrón se utiliza en todo Spring Data para las operaciones CRUD de entidades, a través de la interfaz **CRUDRepository**
- Cada módulo específico de una fuente de datos cuenta con un repositorio que extiende de la interfaz genérica, como
 - JpaRepository
 - MongoRepository
 - GemfireRepository.

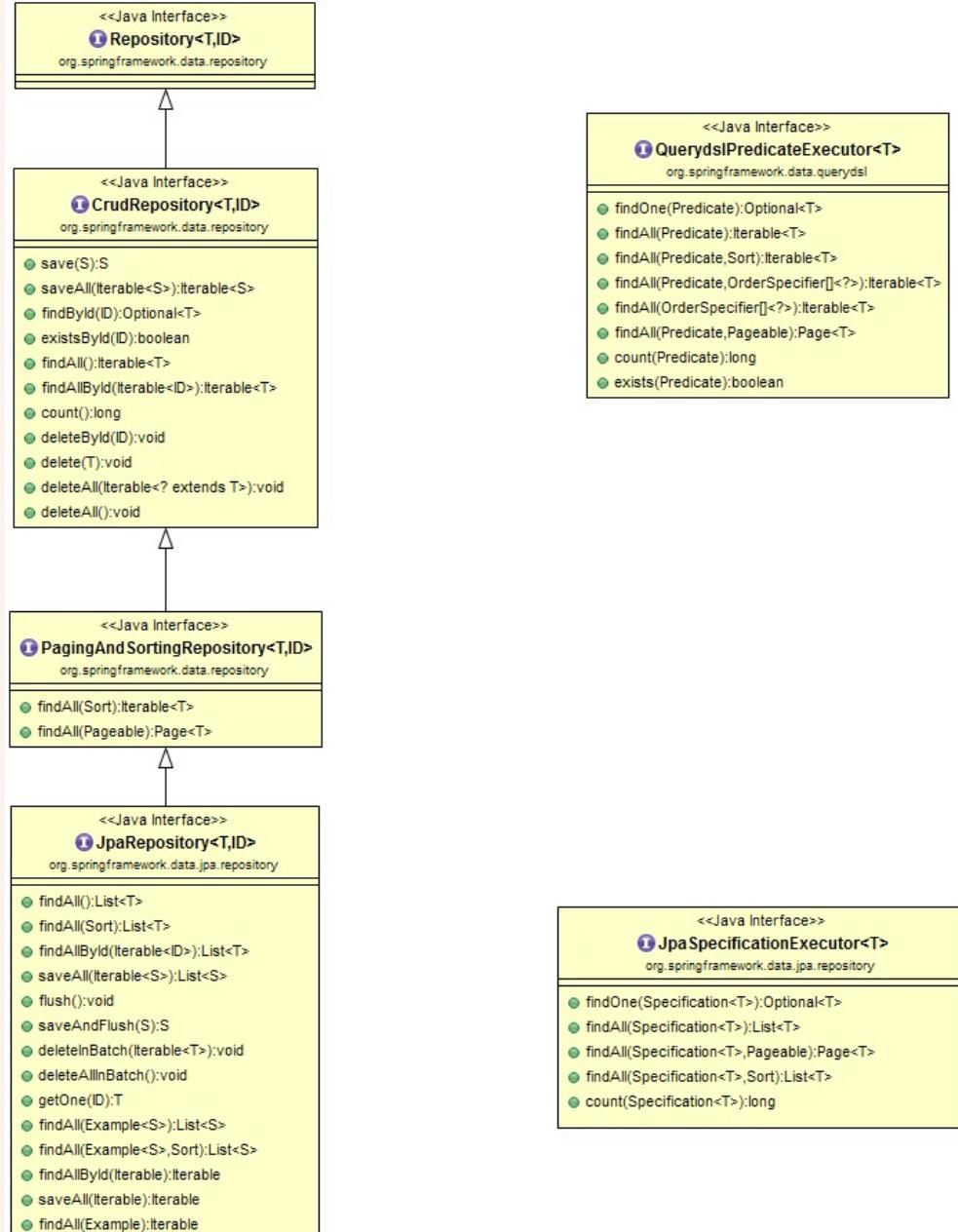
Repository pattern



Spring Data



Interfaces principales de los módulos Spring Data Commons y Spring Data JPA.



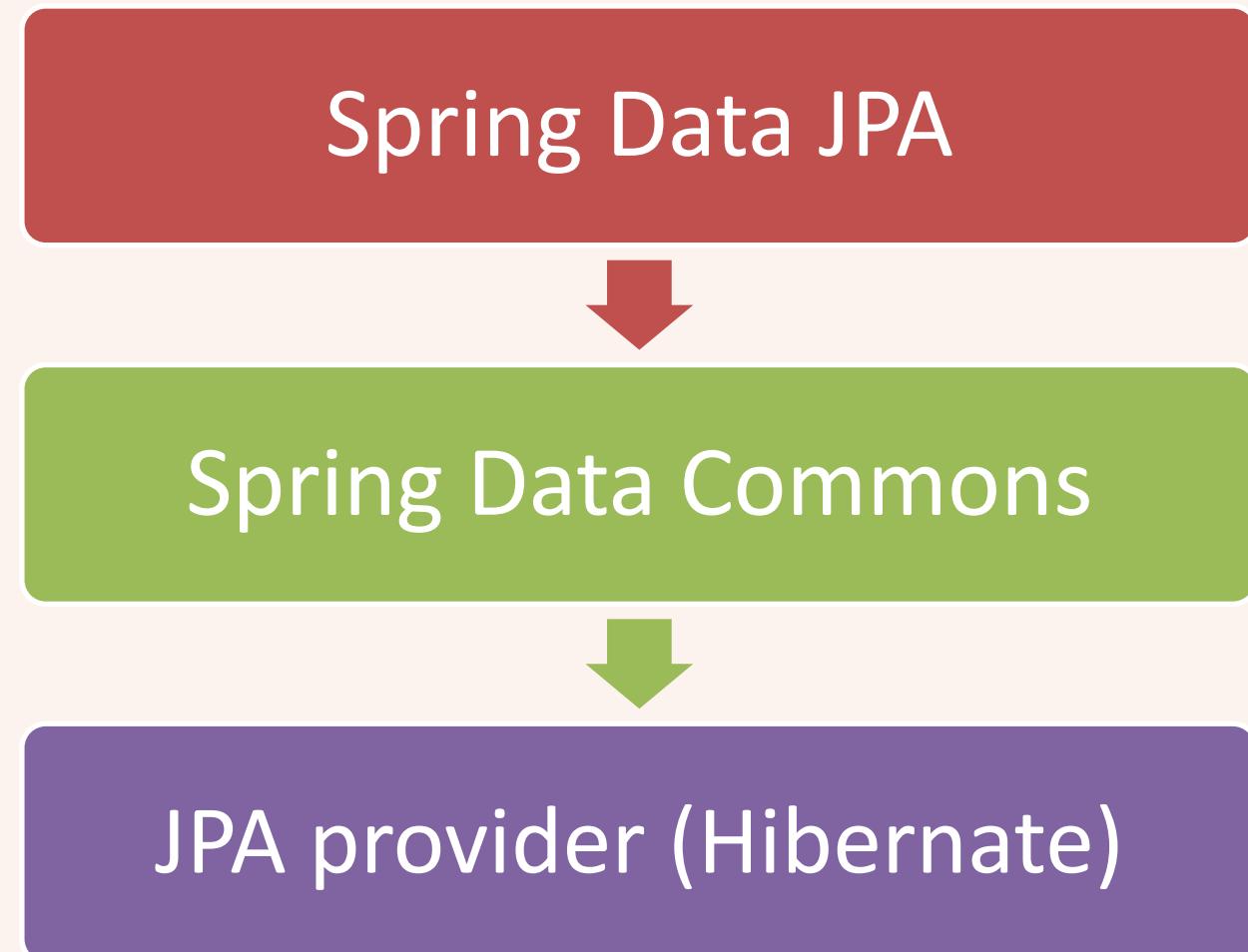
Spring Data JPA

Es un framework que agrega una capa adicional de abstracción sobre el proveedor JPA (como Hibernate), facilitando el acceso a datos mediante repositorios.

No es un proveedor de JPA.

Spring Data JPA utiliza Hibernate como su proveedor JPA predeterminado, pero puede configurarse para usar otros proveedores compatibles con JPA.

Spring Data JPA



Spring Data JPA

Se ocupa de la compatibilidad mejorada para las capas de acceso a datos basadas en JPA.

Interfaces

- **JpaRepository<T, ID extends Serializable>**
 - Extiende PagingAndSortingRepository y CrudRepository, proporcionando métodos adicionales específicos de JPA para el manejo de entidades.
- **JpaSpecificationExecutor<T>**
 - Permite la ejecución de consultas con criterios complejos basados en Specification para entidades JPA.

Anotaciones

Una anotación en Java es una característica que permite incrustar información suplementaria en un archivo fuente, comenzando con '@'.

No cambian el comportamiento de un programa, pero pueden influir en cómo el compilador o las herramientas de desarrollo tratan el código.

Ayudan a relacionar metadatos con los componentes del programa, como constructores, métodos, clases, etc.

Pueden ser procesadas por generadores de código, el compilador o herramientas de despliegue.

Aunque no son simples comentarios, pueden modificar la forma en que el compilador interpreta el programa.

Su uso proporciona capacidades amplias en la forma en que se configuran los comportamientos de Spring Framework

Anotaciones Core Spring Framework

- **@Required**
 - Se aplica a los métodos de “setters” de beans cuando se necesita hacer cumplir una propiedad requerida.
- **@Autowired**
 - Se aplica a campos, métodos de “setters” y constructores, inyecta la dependencia del objeto implícitamente.
- **@Qualifier**
 - Se usa junto con la anotación **@Autowired** cuando se necesita más control del proceso de inyección de dependencia
 - Se utiliza para evitar la confusión que ocurre cuando se crea más de un bean del mismo tipo y se desea conectar solo uno de ellos con una propiedad.
- **@Configuration**
 - Se usa en clases que definen beans.
 - Es un análogo para un archivo de configuración XML.

Anotaciones Core Spring Framework

- **@ComponentScan**
 - Se usa con la anotación **@Configuration** para permitir que Spring conozca los paquetes para buscar componentes anotados.
 - También se utiliza para especificar paquetes base usando basePackageClasses o basePackage
- **@Bean**
 - Se utiliza a nivel de método, funciona con **@Configuration** para crear beans Spring.
 - El método anotado con esta anotación funciona como la ID del bean, y crea y devuelve el bean real
- **@Lazy**
 - Se usa en clases de componentes si se desea inicializar un bean “como una carga diferida”
 - El bean se creará e inicializará solo cuando se solicite por primera vez.
- **@Value**
 - Se utiliza en los niveles de campo, parámetro de constructor y parámetro de método.
 - Indica una expresión de valor predeterminado para el campo o parámetro para inicializar la propiedad.

Spring Framework Stereotype

- **@Component**
 - Se usa en clases para indicar un componente Spring.
 - Marca la clase Java como un bean o componente para que el mecanismo de exploración de componentes de Spring pueda agregarla al contexto de la aplicación.
- **@Controller**
 - Se usa para indicar que la clase es un controlador Spring.
 - Se puede utilizar para identificar controladores para Spring MVC o Spring WebFlux.
- **@Service**
 - Esta anotación se usa en una clase que realiza algún servicio, como ejecutar lógica de negocios, realizar cálculos y llamar a API externas.
 - Esta anotación es una forma especializada de la anotación **@Component** destinada a ser utilizada en la capa de servicio.
- **@Repository**
 - Se utiliza en clases Java que acceden directamente a la base de datos.
 - Funciona como un marcador para cualquier clase que cumpla la función de repositorio u Objeto de acceso a datos.

Contacto

Dr. Omar Mendoza González

omarmendoza564@aragon.unam.mx