



**11<sup>a</sup>**  
Emisión

**DIPLOMADO**  
**Desarrollo de Sistemas**  
**con Tecnología Java**

**Módulo 7**  
**Persistencia con Spring Data**

*Dr. Omar Mendoza González*

*omarmendoza564@aragon.unam.mx*



**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO**  
Dirección General de Cómputo y de Tecnologías de información y Comunicación  
Dirección de Docencia en TIC



Educación  
Continua  
1971 - 2021

# Convenios

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



# Evaluación

- Practicas 20%
- Ejercicios 30%
- Practica final 20%
- 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.

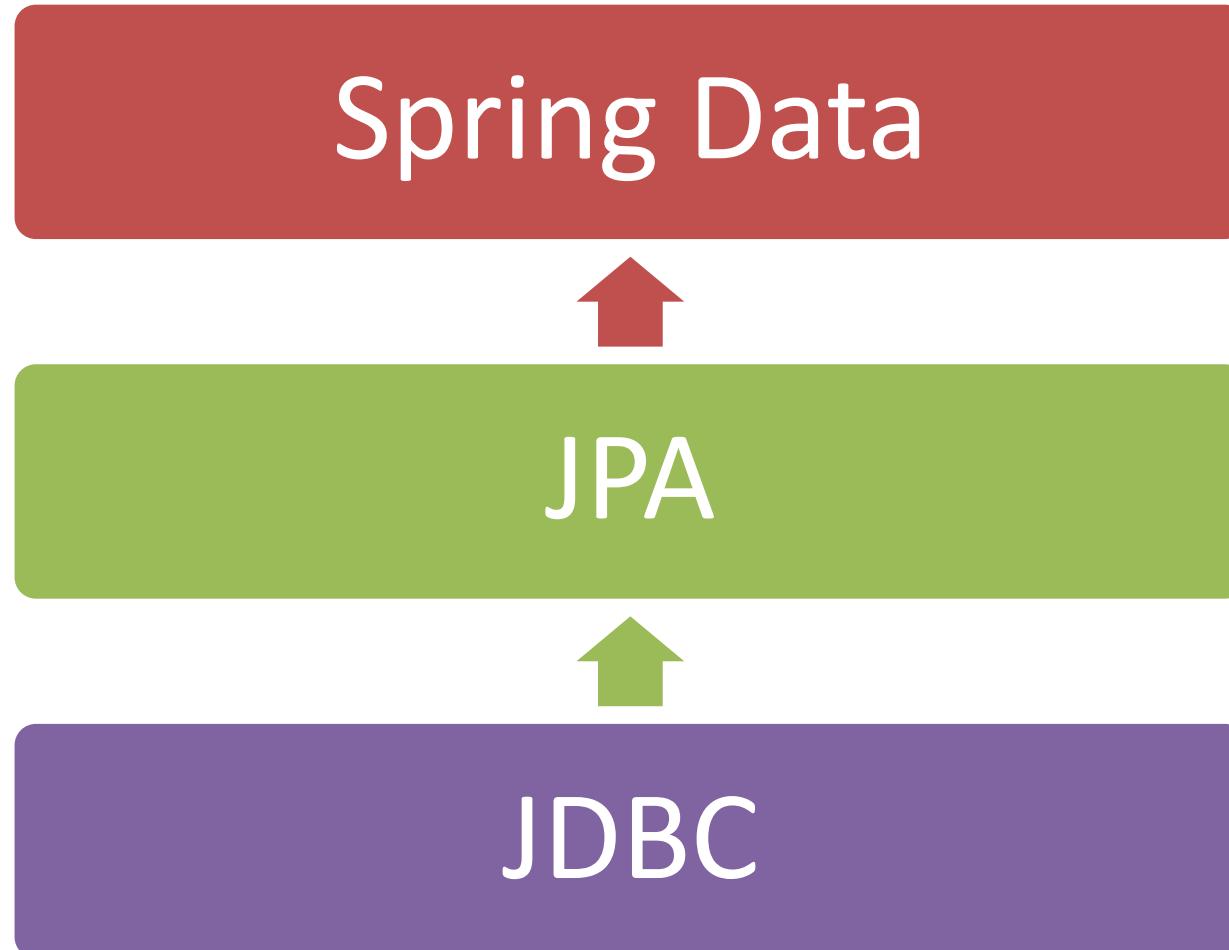


# Spring Data

- 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 almacen de datos 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 muchos subproyectos que son específicos de una base de datos determinada.

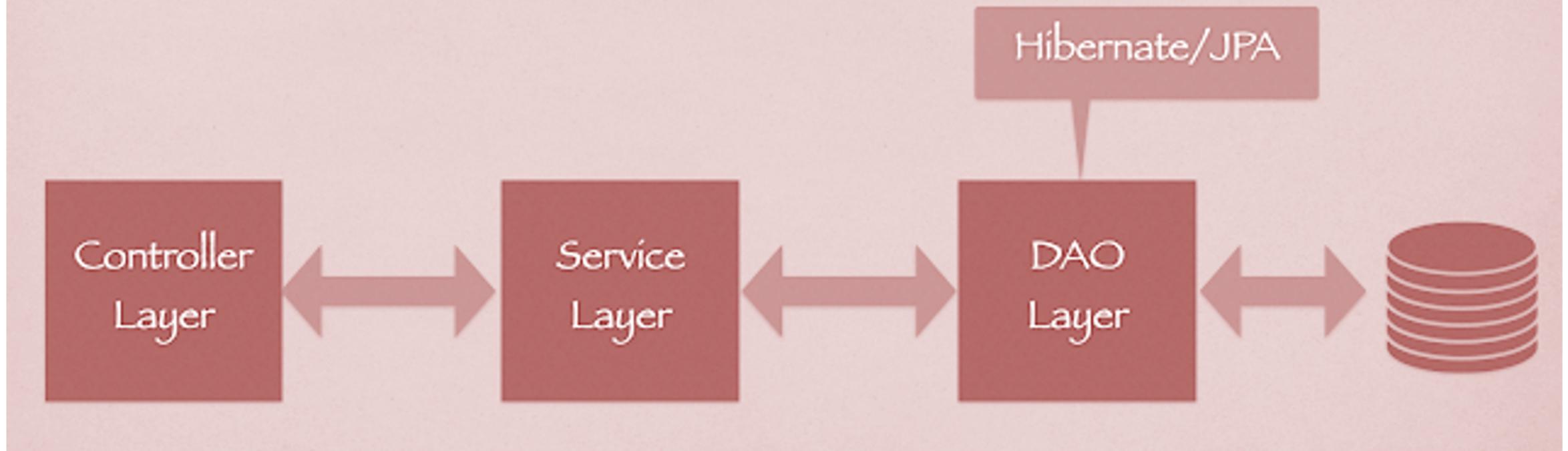


# Spring Data



## DAO/Repository layer

# Application Architecture



# Spring Data

- Características
  - 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.



# 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

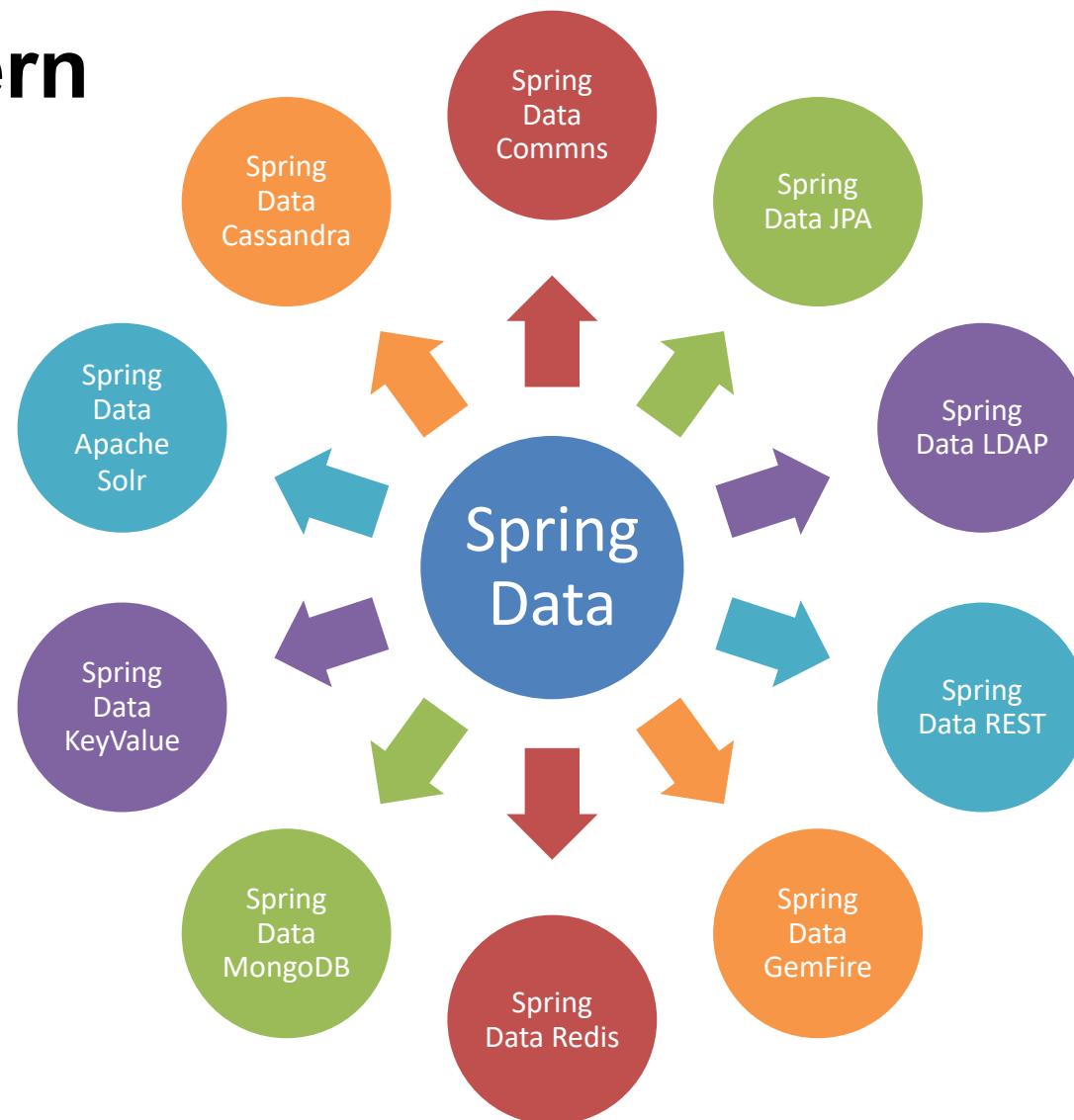
- Commons realiza una abstracción de cualquier fuente de datos en particular.
- Independientemente de la fuente de datos, el objetivo es siempre el mismo, tener una forma de convertir las entidades de objetos de Java en registros de la fuente de datos de destino y conservarlos, así como convertir los registros de nuevo en entidades.
- Java Business Entities <--> Registros permanentes de almacenamiento de datos de destino
- Búsqueda de registros
- Actualización de registros
- Eliminación de registros



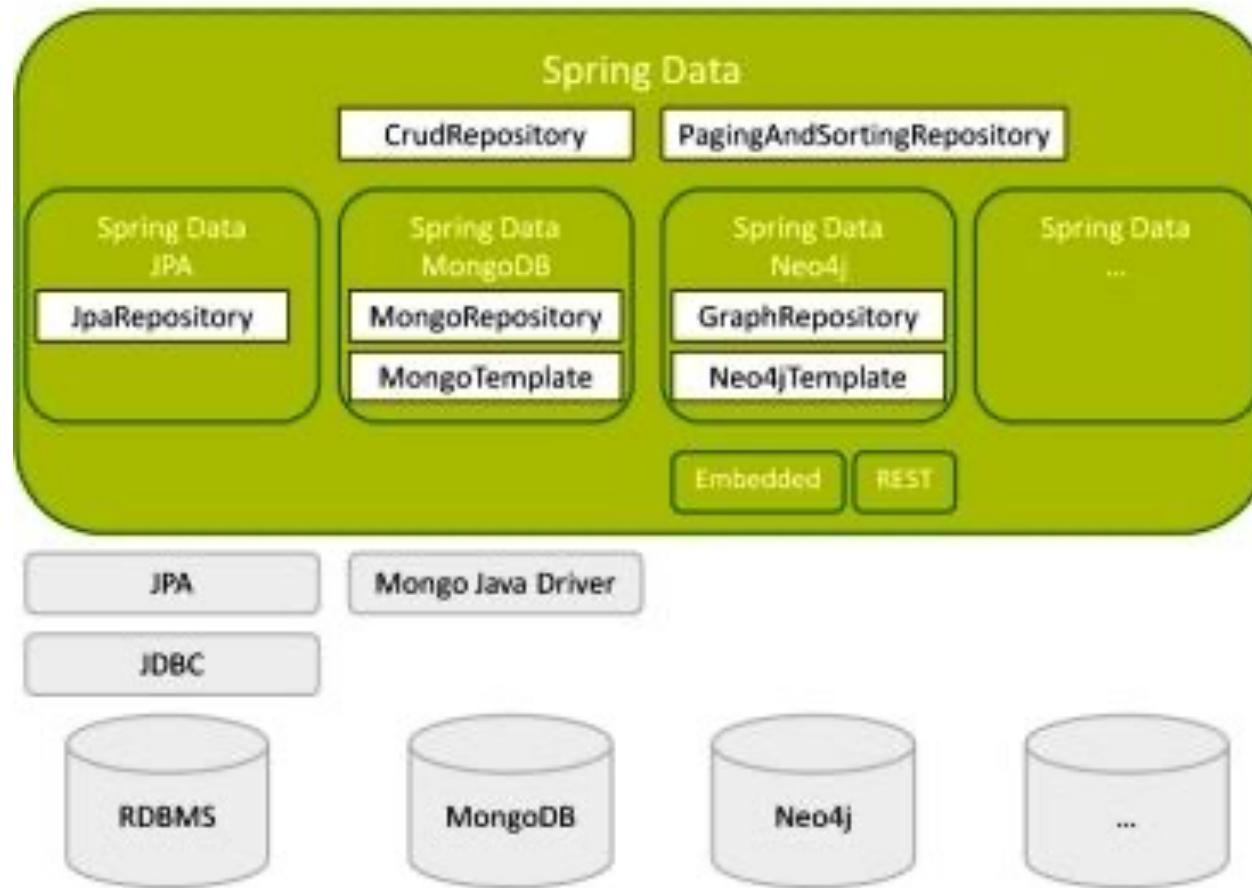
# Repository pattern

- Es la abstracción utilizada por Spring Data commons para lograr sus objetivos.
- Se utiliza a lo largo del proyecto Spring Data para crear, leer, actualizar y eliminar registros citando entidades. **CRUDRepository**
- Cualquier módulo para una fuente de datos en particular tiene un repositorio que se extiende desde el genérico.
  - JpaRepository
  - MongoRepository
  - GemfireRepository

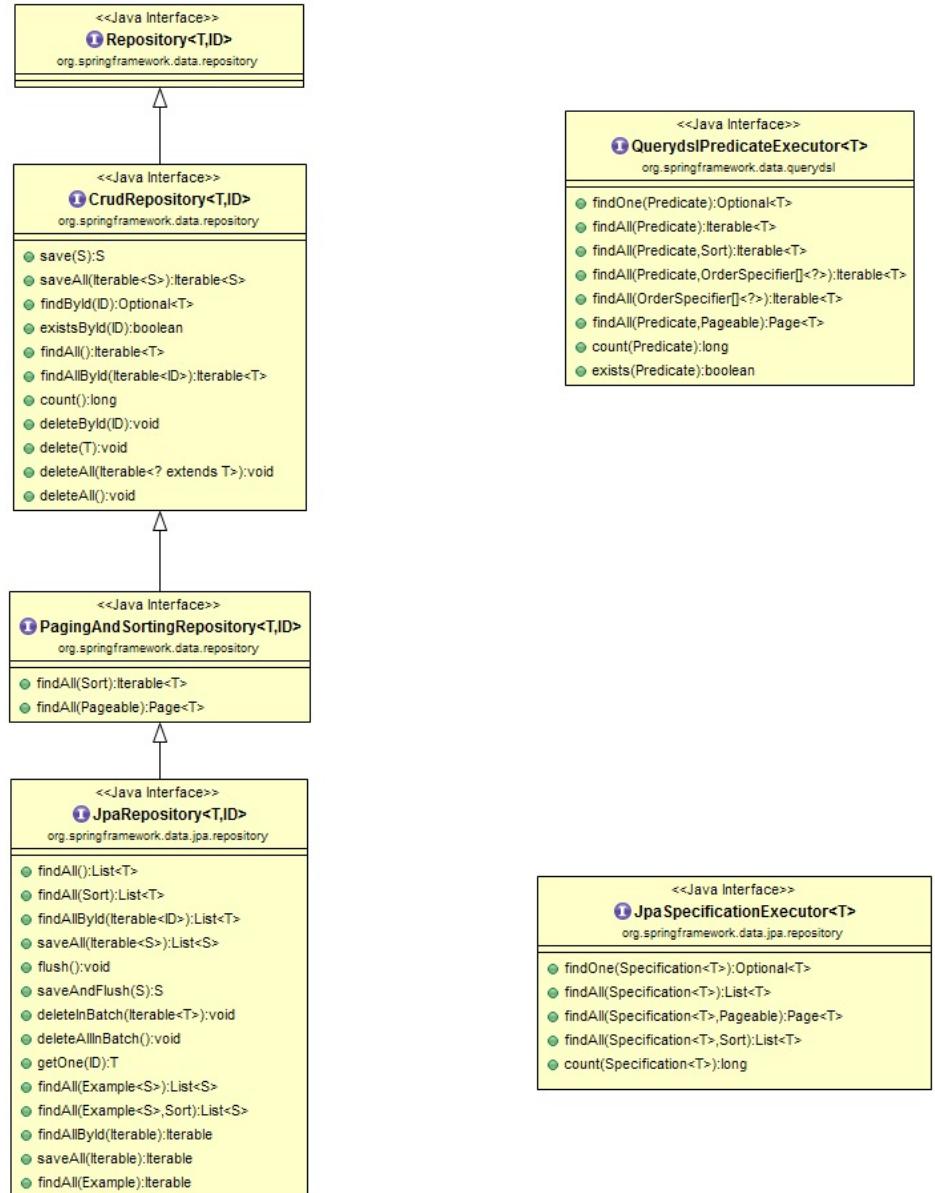
# Repository pattern



# Spring Data



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



# Spring Data Commons

- Interfaces
  - `Repository<T, ID extends Serializable>`
  - `CrudRepository<T, ID extends Serializable>`
  - `PagingAndSortingRepository<T, ID extends Serializable>`
  - `QueryDslPredicateExecutor`

# Spring Data Commons

- **Repository<T, ID extends Serializable>**
  - Captura el tipo de entidad gestionada y el tipo del ID de la entidad.
  - Ayuda al contenedor Spring a descubrir las interfaces de repositorio "concretas" durante el escaneo de classpath.

# 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)*



# Spring Data Commons

- **PagingAndSortingRepository<T, ID extends Serializable>**
  - Es una extensión de *CrudRepository* para proporcionar métodos adicionales para recuperar entidades usando abstracción de la paginación y ordenamiento.

# Spring Data Commons

- **QueryDslPredicateExecutor**
  - no es una "interfaz de repositorio".
  - Declara los métodos que se utilizan para recuperar entidades de la base de datos mediante el uso de objetos *QueryDsl*

# Spring Data JPA

- No es un proveedor de JPA.
- Es una biblioteca / framework que agrega una capa adicional de abstracción en la parte superior del proveedor JPA (como Hibernate).
- **Spring Data JPA usa Hibernate como proveedor JPA predeterminado.**



# Spring Data JPA

Spring Data JPA



Spring Data Commons



JPA provider (Hibernate)



# 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>`
  - `JpaSpecificationExecutor`



# Anotaciones

- Una **anotación en Java** es aquella característica que le permite **incrustar información suplementaria** en un archivo fuente.
- Esta información no cambia las acciones de un programa, pero puede ser utilizada por varias herramientas, tanto durante el desarrollo como durante el despliegue.
- Pueden ser procesadas por un generador de código fuente, por el compilador o por una herramienta de despliegue.

# Anotaciones

- Las **anotaciones en Java** comienzan con '@'.
- **No cambian la actividad** de un programa ordenado.
- Ayudan a **relacionar metadatos** (datos) con los componentes del programa, es decir, constructores, estrategias, clases, etc.
- No son comentarios sin adulteraciones, ya que pueden cambiar la forma en que el compilador trata el programa.



# Anotaciones

- El uso de anotaciones 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](mailto:omarmendoza564@aragon.unam.mx)

