CURSO DE HIBERNATE 5

OpenWebinars

HIBERNATE

(1) INTRODUCCION

Persistencia, desfase objecto-relacional, ORM. Productos y estándares

(2) HIBERNATE

Más que un ORM. Comparativa con otros productos. JPA. Maven. Módulos

(3) PRIMER PROYECTO

Hibernate.cfg.xml, EntityManager y persistence.xml

(4) ENTIDADES

Definición del modelo del dominio. Entidades y ciclo de vida. XML y anotaciones. Tipos de datos.

(5) ASOCIACIONES

ManyToOne, OneToMany, OneToOne, ManyToMany

HIBERNATE

(6) ELEMENTOS AVANZADOS

Campos calculados, herencia.

(7) COLECCIONES

Mapeo de colecciones. Tipos (list, set, map). Colecciones ordenadas (sorted vs. ordered).

(8) GENERACION DEL ESQUEMA

Customización del proceso de generación del esquema.

(9) CONTEXTO DE PERSISTENCIA

Almacenamiento, recuperación y borrado de entidades.

(10) TRANSACCIONES

Control de concurrencia. Patrones y antipatrones.

HIBERNATE

(12)
ENVERS

Introducción a la
auditoria de entidades.

(11)
CONSULTAS
HPQL VS JPQL

Consultas con
parámetros,
Anotaciones. SQL nativo

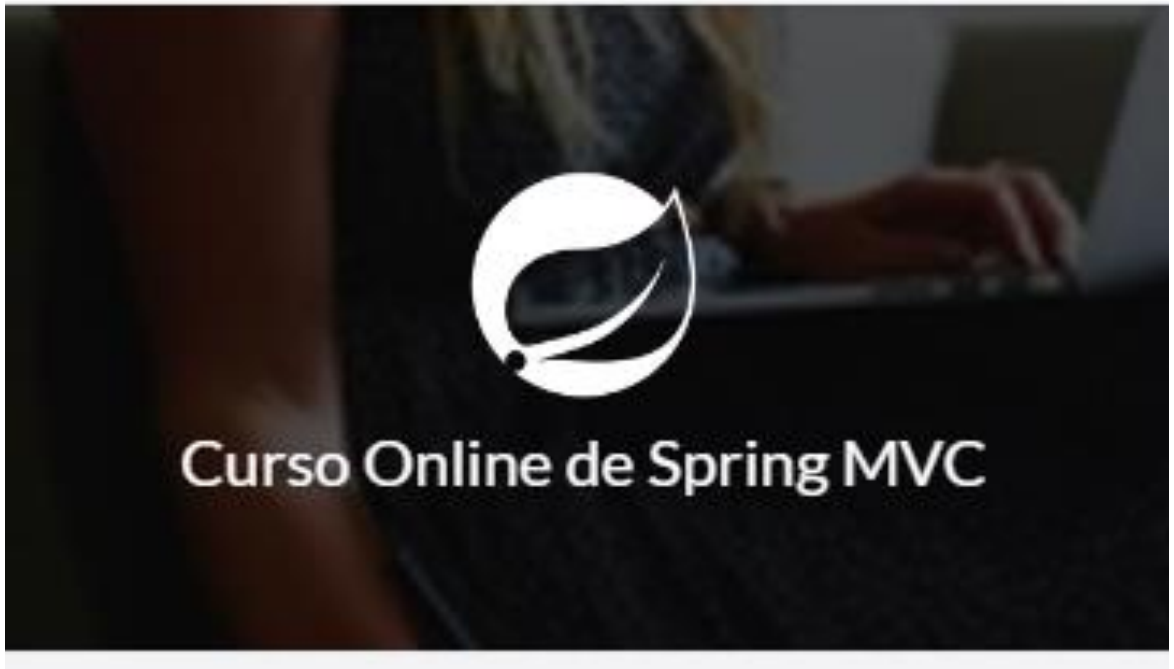
1.

PASOS PARA CREAR EL NUEVO PROYECTO

¡A por ellos!
(otra vez más)

- (0) Visitar el curso de Spring Web MVC
- (1) Creación del proyecto
- (2) Configuración de la base de datos
- (3) Clases modelo y *DAO*
- (4) Controlador
- (5) Ejecución

(0) Visitar el curso de Spring Web MVC



(0) Visitar el curso de Spring Web MVC

OpenWebinars

Introducción a Spring

- Introducción a Spring
- Configuración Spring
- Inversión de control
- Inyección de dependencias. Tipos de inyección.
- Beans (simples, referencias, anidaciones, colecciones)
- Inyección automática
- Ámbitos de un Bean.
- Ciclo de vida de los Beans
- Anotaciones Required, Autowired y Qualifier
- Estereotipos

Desarrollo de Aplicaciones con Spring MVC y acceso a datos con Spring Data

- Una palabra sobre patrones de diseño: MVC, Front Controller, DAO...
- Otra palabra sobre HTTP
- Configuración del entorno
- Controladores y Vistas
- Mapeo de URLs
- Lectura de parámetros HTTP
- Elección de la vista: ViewResolver
- Formularios
- Validación
- Introducción a Spring Data. Súbmodulos.
- Spring Data JPA. Configuración y dependencias.
- Repositorios y entidades
- Consultas básicas
- Persistiendo entidades
- Consultas avanzadas

Spring Boot: haz mi vida más fácil

- Tareas a la hora de desarrollar un proyecto Spring
- Introducción a Spring Boot
- Convención sobre Configuración
- Nuestro primer proyecto con Spring Initializr
- Anotaciones
- Uso del asistente de STS (Spring Tool Suite)
- Ejecutando nuestra aplicación
- Empaquetando nuestra aplicación en un jar independiente
- Starters POMs y el Asistente de STS
- Customización de propiedades
- Configuración de una aplicación MVC
- Spring Boot CLI (command line interface)

Servicios REST con Spring: Restify my life

- Principios básicos REST
- Cliente/Servidor
- Diferencias entre JAX-RS y Spring REST MVC
- Primer EndPoint
- REST y el patrón MVC
- Mapeo de peticiones
- Representación de los datos
- Formato de respuesta
- Gestión de errores
- Operaciones CRUD en servicios REST
- Mapeo de operaciones crud a métodos HTTP
- Creación de recursos
- Actualización de recursos
- Borrado de recursos
- Seguridad
- Autenticación: HTTP, Token
- Autorización: Mapeo de URLs, Anotaciones de recursos.
- Construcción de un cliente REST
- Manejo de la seguridad.

(1) Creación del proyecto

- Creación mediante el asistente de STS.
- Dependencias Web, JPA y MySQL

(2) Configuración de la base de datos

- Uso de JavaConfig y del fichero de *properties*.
- Definición de un *dataSource*.
- Definición de un *entityManagerFactory*.
- Definición de un *transactionManager*.
- Definición de propiedades.

(2) Configuración de la base de datos

```
@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(env.getProperty("db.driver"));
    dataSource.setUrl(env.getProperty("db.url"));
    dataSource.setUsername(env.getProperty("db.username"));
    dataSource.setPassword(env.getProperty("db.password"));
    return dataSource;
}
```

(2) Configuración de la base de datos

```java
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
    LocalContainerEntityManagerFactoryBean entityManagerFactory = new LocalContainerEntityManagerFactoryBean();

    //Le asignamos el dataSource que acabamos de definir.
    entityManagerFactory.setDataSource(dataSource);

    // Le indicamos la ruta donde tiene que buscar las clases anotadas
    entityManagerFactory.setPackagesToScan(env.getProperty("entitymanager.packagesToScan"));

    // Implementación de JPA a usar: Hibernate
    HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();
    entityManagerFactory.setJpaVendorAdapter(vendorAdapter);

    // Propiedades de Hiberante
    Properties additionalProperties = new Properties();
    additionalProperties.put("hibernate.dialect", env.getProperty("hibernate.dialect"));
    additionalProperties.put("hibernate.show_sql", env.getProperty("hibernate.show_sql"));
    additionalProperties.put("hibernate.hbm2ddl.auto", env.getProperty("hibernate.hbm2ddl.auto"));
    entityManagerFactory.setJpaProperties(additionalProperties);

    return entityManagerFactory;
}
```

(2) Configuración de la base de datos

```
@Bean
public JpaTransactionManager transactionManager() {
    JpaTransactionManager transactionManager = new JpaTransactionManager();
    transactionManager.setEntityManagerFactory(entityManagerFactory.getObject());
    return transactionManager;
}
```

```
@Bean
public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {
    return new PersistenceExceptionTranslationPostProcessor();
}
```

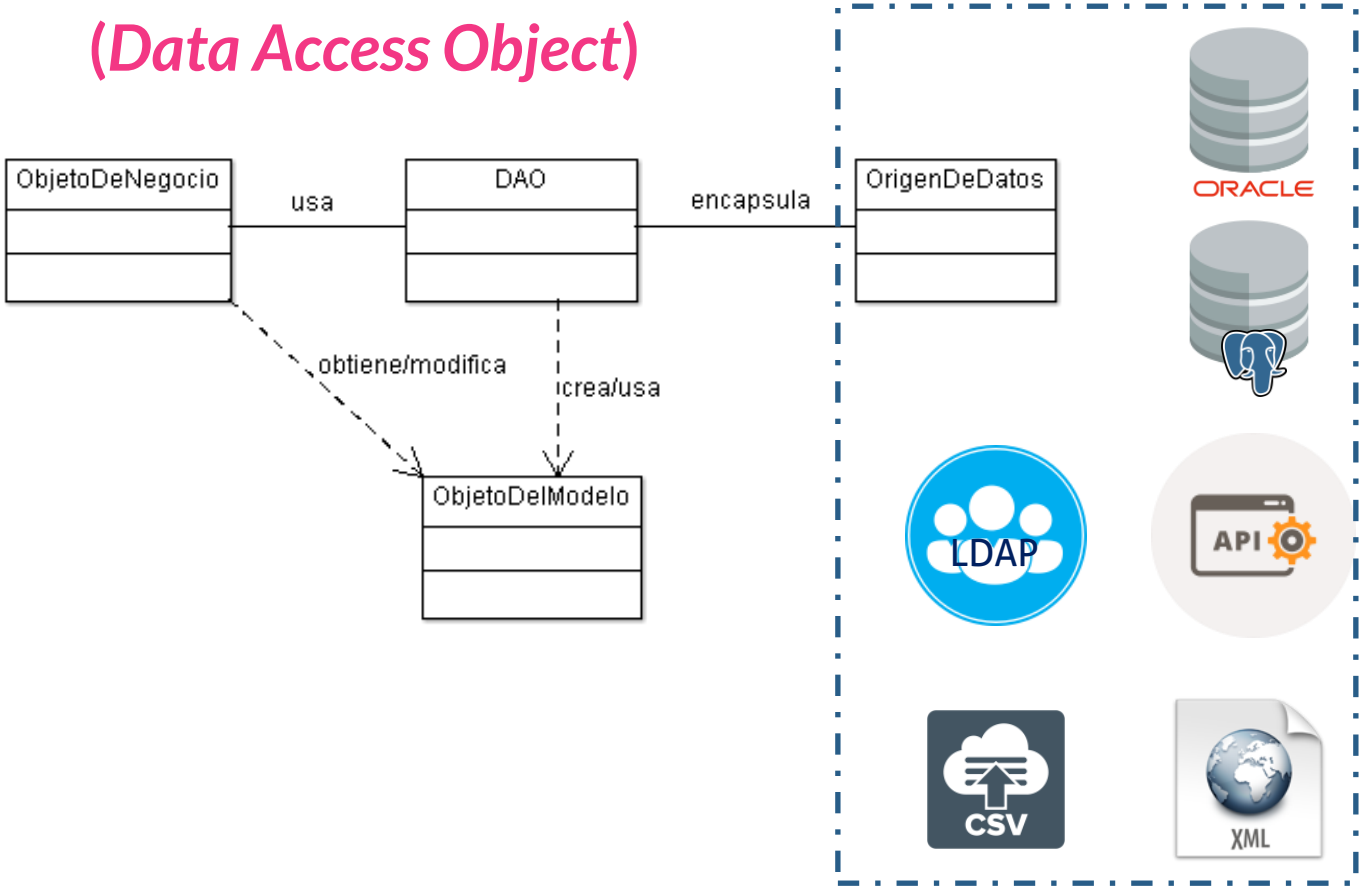
(2) Configuración de la base de datos

```
@Autowired
private Environment env;

@Autowired
private DataSource dataSource;

@Autowired
private LocalContainerEntityManagerFactoryBean entityManagerFactory;
```

(3) Clases modelo y *DAO*

- En la clase *entidad…*
  - Anotamos la clase con @Entity
  - Anotamos la propiedad que será clave primaria con @Id
  - Anotamos el resto de columnas con @Column

PATRÓN DAO
(Data Access Object)



(3) Clases modelo y *DAO*

- En la clase *DAO*…
  - Anotamos la clase con @Repository y @Transactional
  - Añadimos un @PersistenceContext
  - Implementamos los métodos necesarios

(4) Controlador

- Anotamos con @Controller
- Autocableamos (*@Autowired*) el *DAO*.
- Definimos los métodos necesarios del controlador

  /create?name=...&message=....
  /delete?id=...
  /update?id=...&name=...&message=...

.