"Persistencia y Acceso a Base de Datos"

Módulo 6 / 1

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Objetivo

En este laboratorio veremos y aprenderemos todo lo relacionado a la persistencia y base de datos con Spring, con una completa aplicación CRUD (Crear, actualizar, Borrar y Listar), un mantenedor de Productos utilizando Hibernate y JPA2.

Spring framework soporta varias tecnologías de acceso a datos - JDBC, Hibernate, JPA, etc – implementada de forma altamente abstracta permitiendo ocultar los detalles específicos de nuestra aplicación y códigos, incluyendo el manejo de excepciones.

Las tecnologías ORM son utilizadas hoy en día por la gran mayoría (por no decir todas) de las aplicaciones de tipo empresarial, pensando en un contexto de servidor de aplicaciones con objetos distribuidos EJBs o Spring.

Spring incluye e integra dos de las tecnologías de persistencia más robustas y utilizadas en la plataforma Java EE: Hibernate, y Java Persistence API 2.1 (JPA2). Permite la integración ORM más sofisticada, simple de configurar e implementada de forma altamente abstracta ocultando los detalles específicos de nuestra aplicación y códigos, incluyendo el manejo de excepciones.

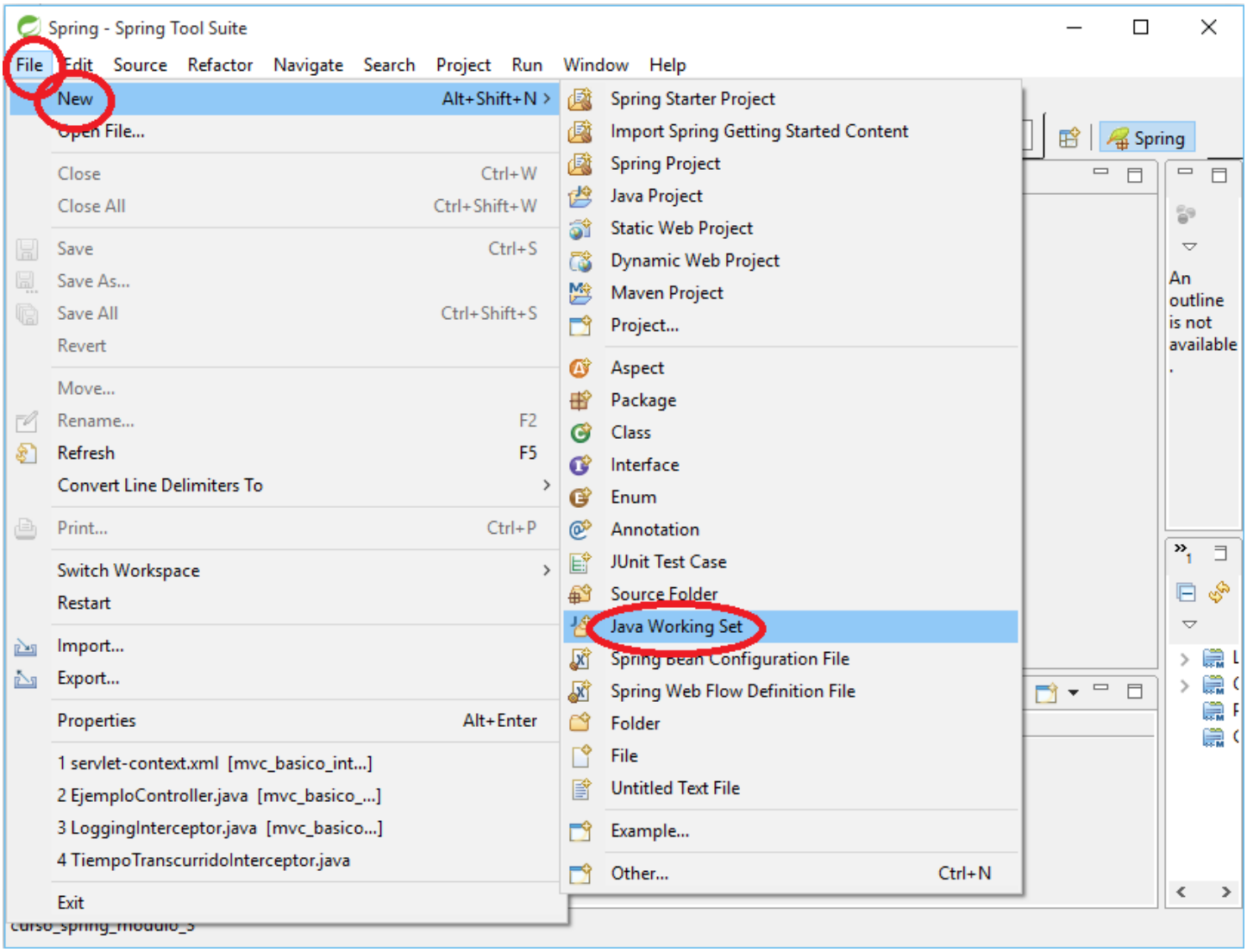
"Quemar etapas"

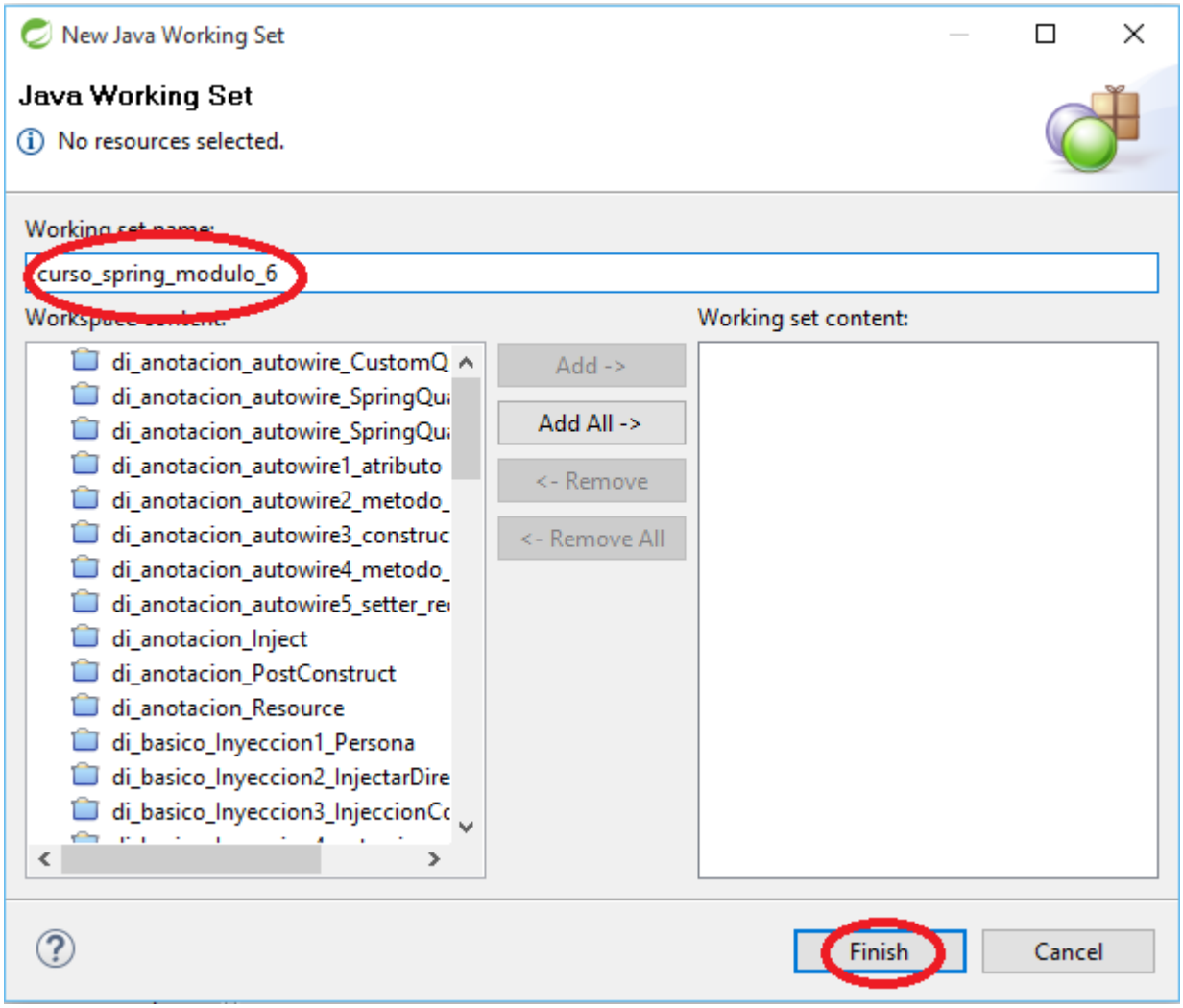
Es importante que saques provecho de cada módulo y consultes todos los temas que se van tratando, sin adelantar etapas.

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

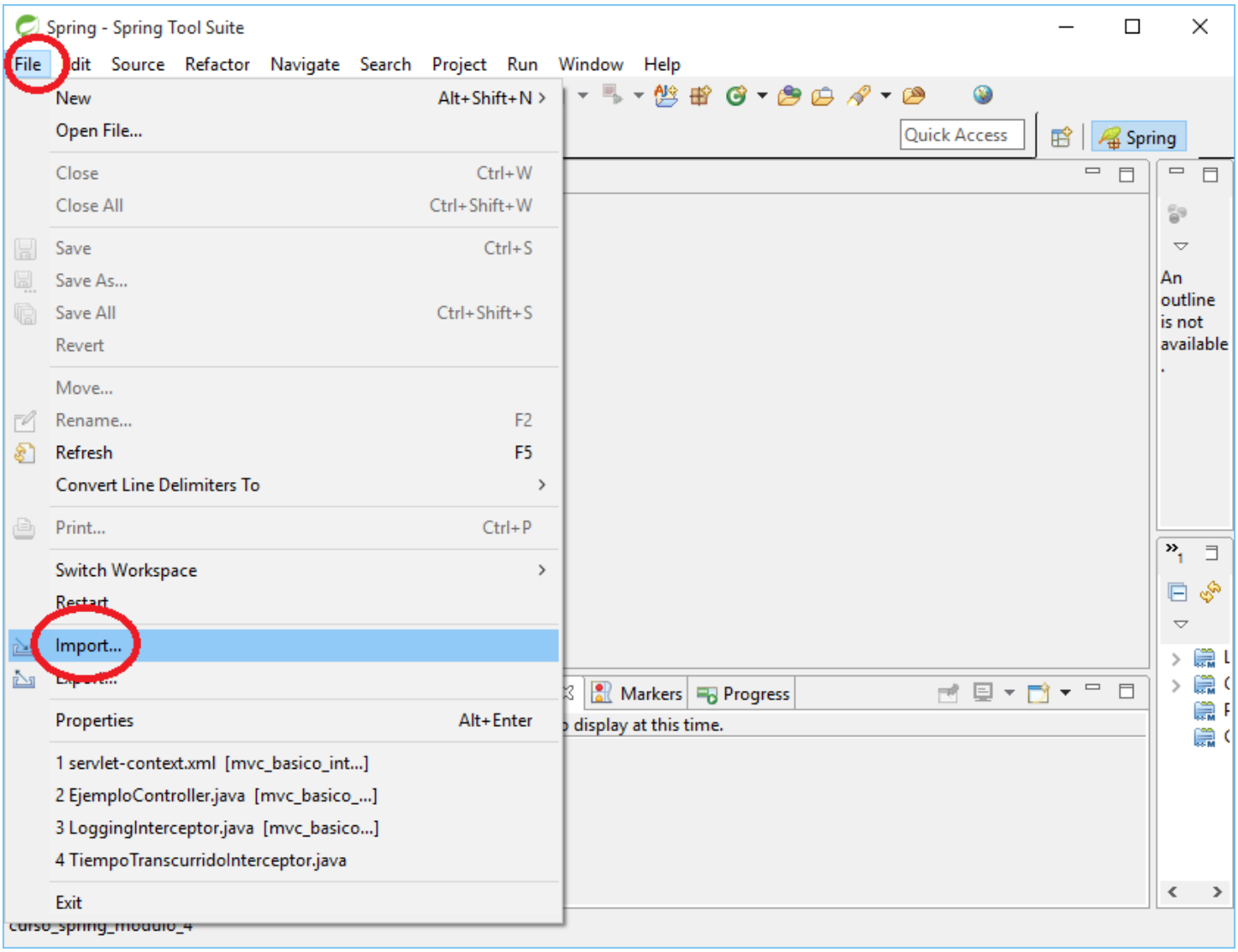
Ejercicio 0: Importar todos los proyectos de ejemplo

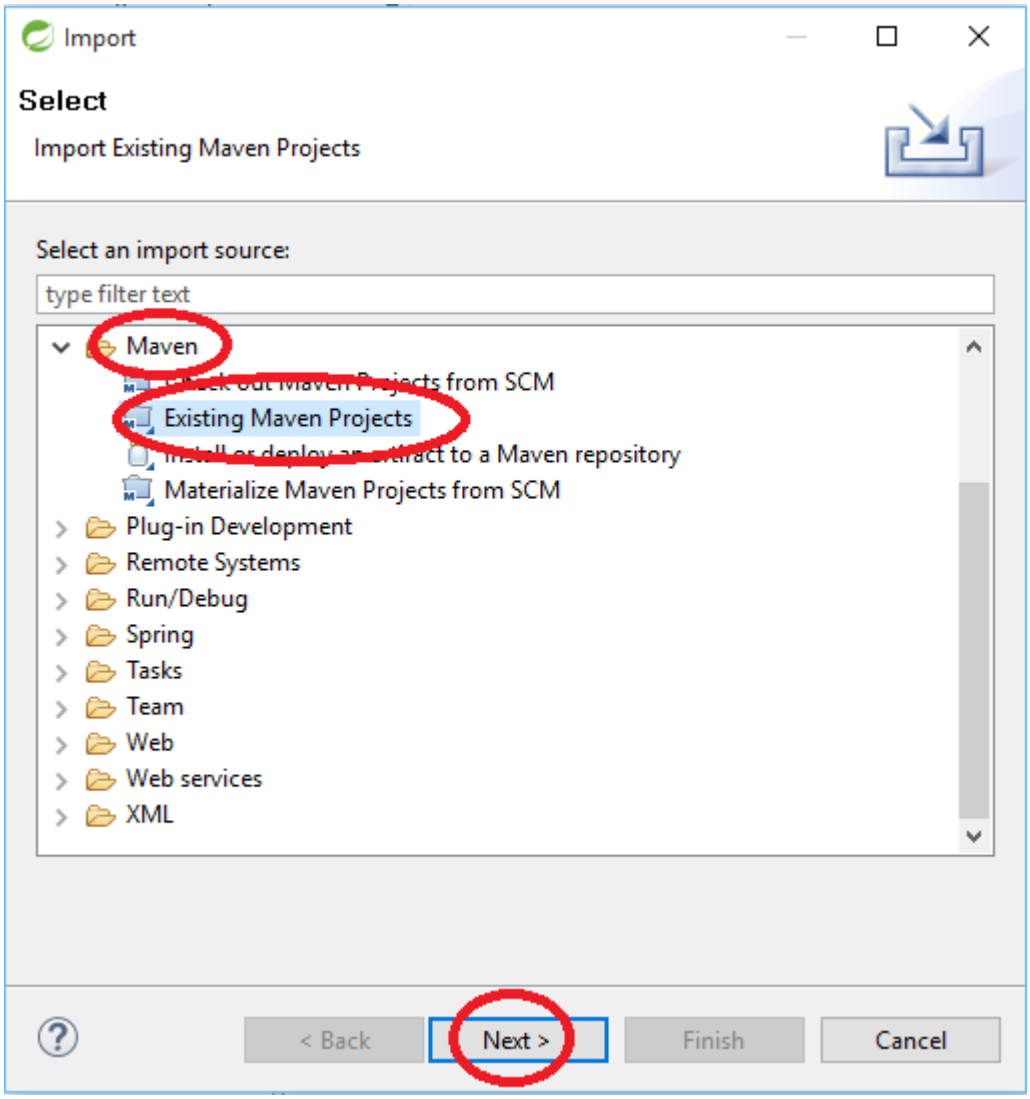
- Crear un nuevo "**Java Working Set**" llamado "**curso_spring_modulo_6**". Esto es para organizar los proyectos bajo un esquema llamado **Working Set**, similar a como organizamos archivos en directorios.
    - Seleccionar **File->New->Java Working Set**.



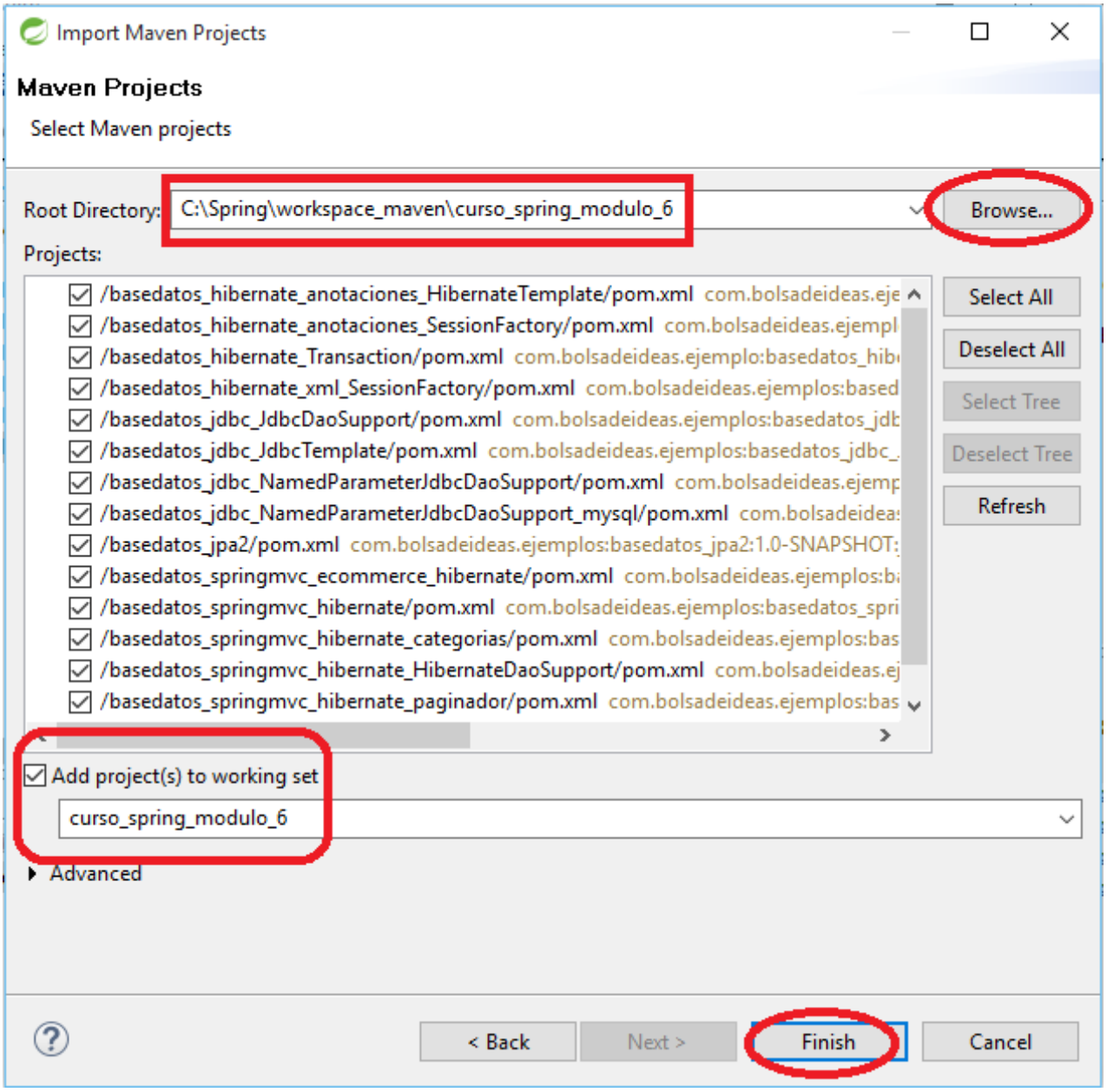

New Java Working Set
Java Working Set
No resources selected.
curso_spring_modulo_6
Working set content:
Add ->
Add All ->
<- Remove
<- Remove All
di_anotacion_autowire_CustomQ
di_anotacion_autowire_SpringQu
di_anotacion_autowire_SpringQu
di_anotacion_autowire1_atributo
di_anotacion_autowire2_metodo_
di_anotacion_autowire3_construc
di_anotacion_autowire4_metodo_
di_anotacion_autowire5_setter_re
di_anotacion_Inject
di_anotacion_PostConstruct
di_anotacion_Resource
di_basico_Inyeccion1_Persona
di_basico_Inyeccion2_InjectarDire
di_basico_Inyeccion3_InjeccionCc
Finish
Cancel

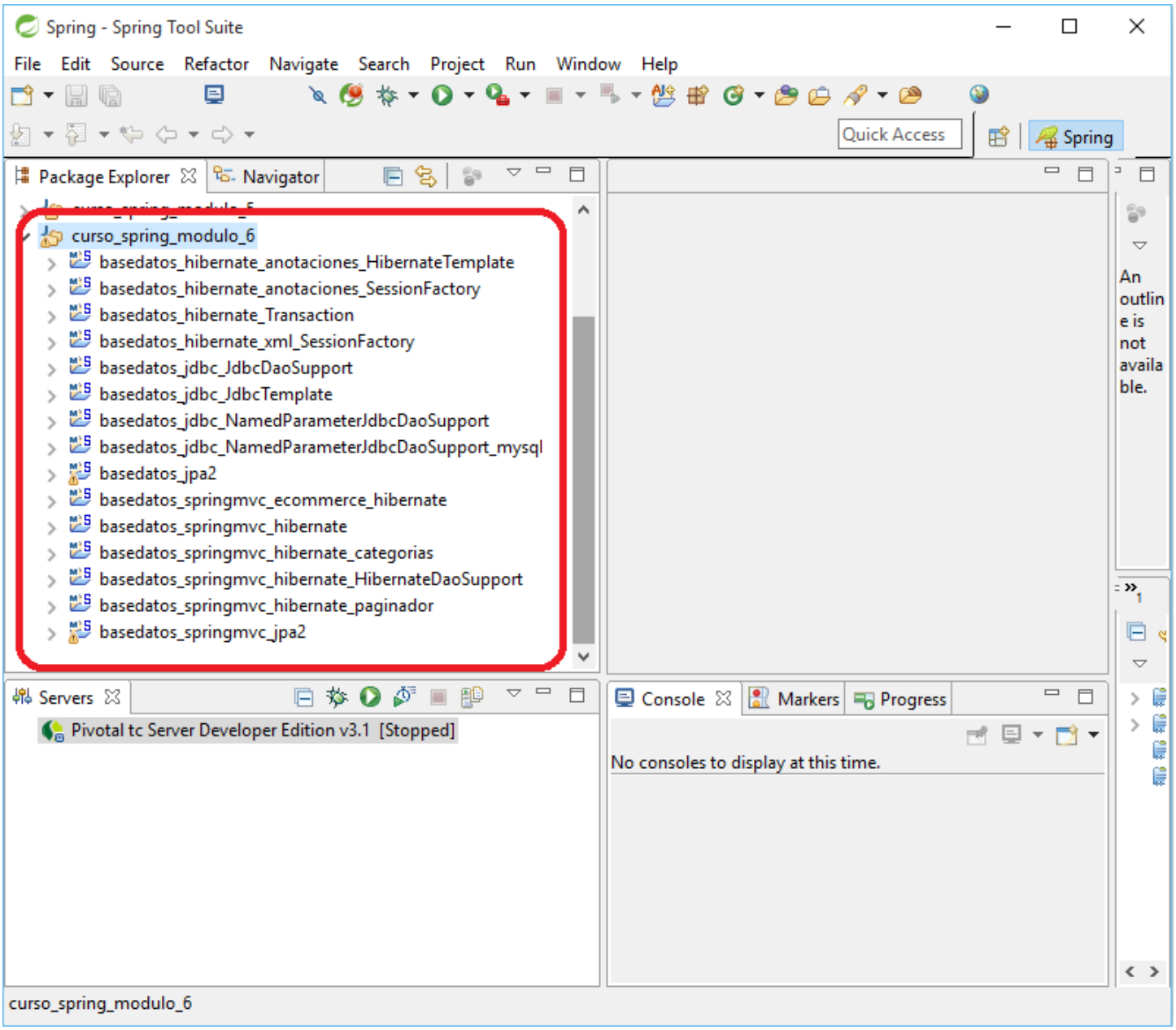
- Importar los proyectos de ejemplos en maven.
    - Seleccionar **File->Import**.




Import
Select
Import Existing Maven Projects
Select an import source:
type filter text
Maven
Existing Maven Projects
Materialize Maven Projects from SCM
Plug-in Development
Remote Systems
Run/Debug
Spring
Tasks
Team
Web
Web services
XML
< Back
Next >
Finish
Cancel

- Clic **Browse**
- Seleccionamos el directorio donde vienen los proyecto de ejemplo del laboratorio
- Agregamos los proyectos al **Working Set curso_spring_modulo_6**
- **Finish**




Spring - Spring Tool Suite
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Spring
Package Explorer
Navigator
curso_spring_modulo_6
basedatos_hibernate_anotaciones_HibernateTemplate
basedatos_hibernate_anotaciones_SessionFactory
basedatos_hibernate_Transaction
basedatos_hibernate_xml_SessionFactory
basedatos_jdbc_JdbcDaoSupport
basedatos_jdbc_JdbcTemplate
basedatos_jdbc_NamedParameterJdbcDaoSupport
basedatos_jdbc_NamedParameterJdbcDaoSupport_mysql
basedatos_jpa2
basedatos_springmvc_ecommerce_hibernate
basedatos_springmvc_hibernate
basedatos_springmvc_hibernate_categorias
basedatos_springmvc_hibernate_HibernateDaoSupport
basedatos_springmvc_hibernate_paginador
basedatos_springmvc_jpa2
An outline is not available.
Servers
Pivotal tc Server Developer Edition v3.1 [Stopped]
Console
Markers
Progress
No consoles to display at this time.
curso_spring_modulo_6

Ejercicio 1: Generar y ejecutar el ejemplo "basedatos_jdbc_JdbcDaoSupport"

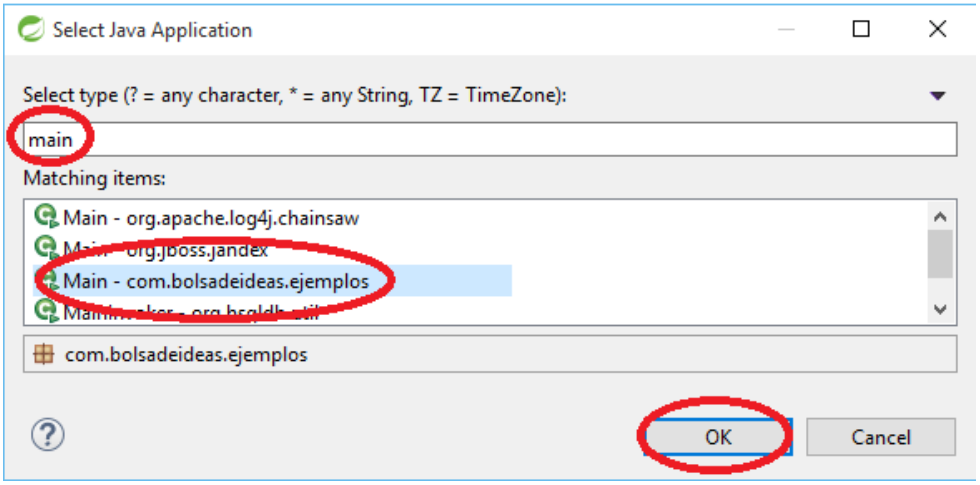
El Spring Framework se encarga de todos los detalles de bajo nivel para que el código esté aislado de los detalles y se centre en lo realmente importante que es la lógica de negocio y los objetos del dominio.

Spring proporciona la clase JdbcTemplate, lo que hace varias tareas cotidianas detrás de escena por nosotros:

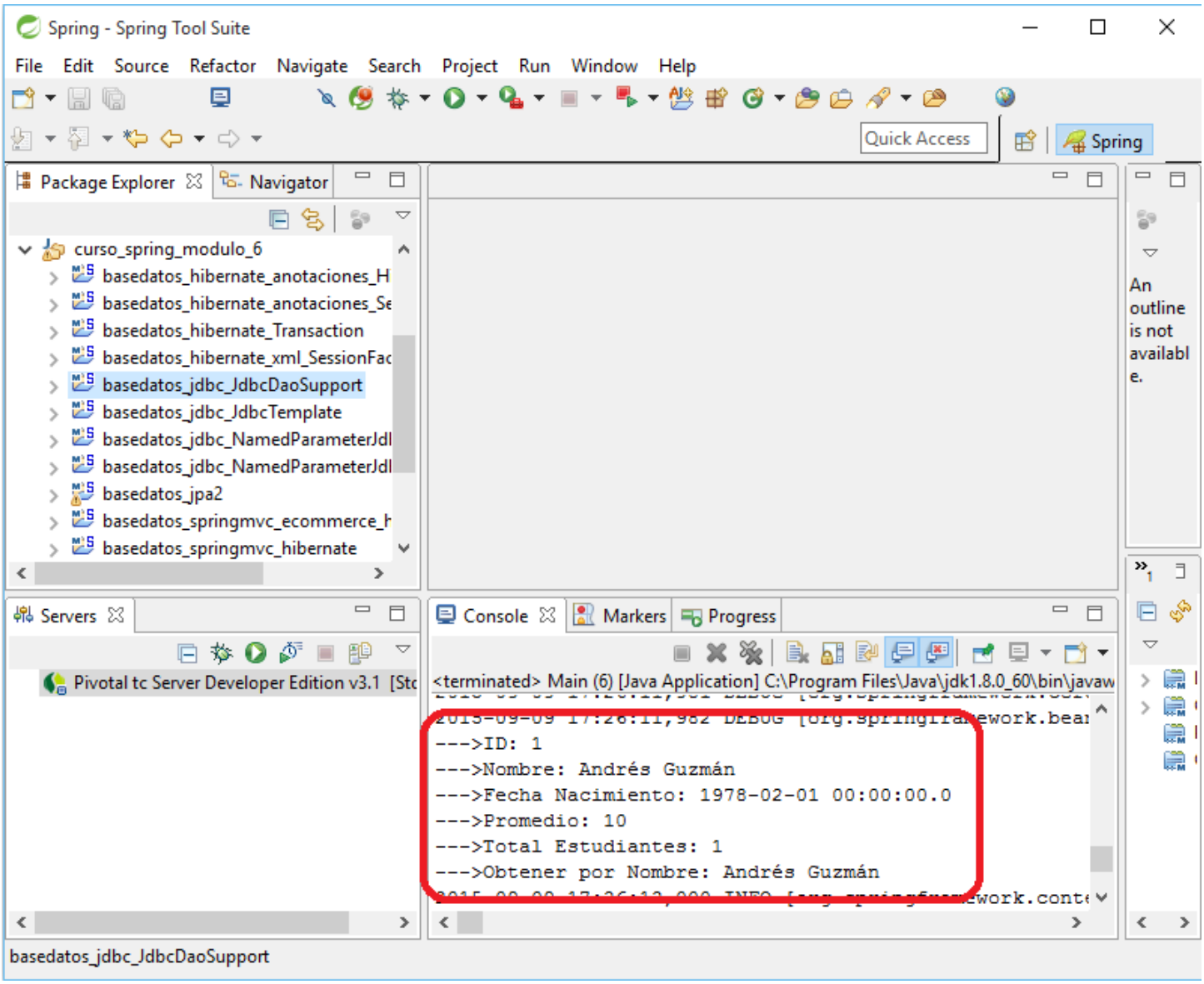
- Se encarga de la creación y liberación de recursos, que ayuda a evitar errores comunes como olvidar el cierre de la conexión.
- Realiza y ejecuta las operaciones básicas de JDBC: crear el objeto sentencia, prepara las consultas, ejecución, etc., dejando para el código sólo la declaración de la consulta SQL y extraer los resultados en forma de objetos relacionales del dominio (POJO o Entity).
- Ejecuta las consultas SQL, las sentencias update y llamadas a procedimientos almacenados, realiza iteración sobre los resultados y extrae los valores retornando objetos del dominio o lógica de negocio.
- Capturas excepciones JDBC y las traduce a excepciones más claras y específicas,

En este ejemplo, vamos a utilizar la clase JdbcDaoSupport, que implementa internamente las funciones de la clase JdbcTemplate para trabajar de forma más simple, directa y con menos código con JDBC.

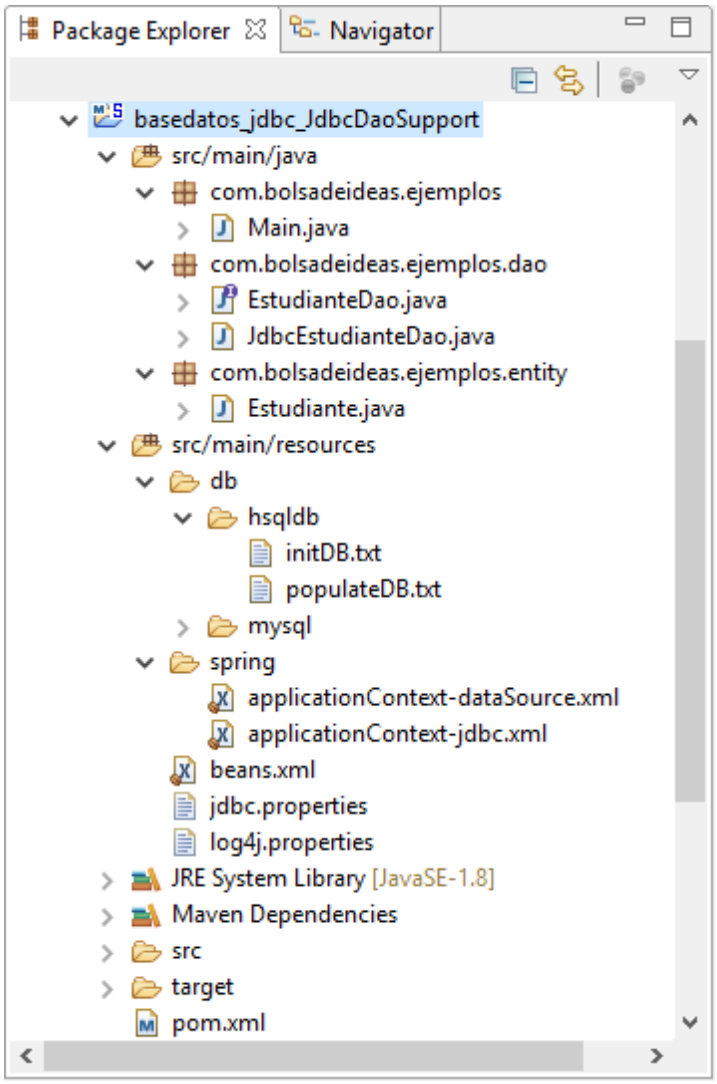
1. Clic derecho sobre el proyecto y "**Run As->Maven Clean**".
2. Clic derecho sobre el proyecto y "**Run As->Maven Install**".
3. Clic derecho sobre el proyecto y **Maven->Update Project…**
4. Clic derecho sobre el proyecto **basedatos_jdbc_JdbcDaoSupport**
    - **Run As-> Java Application**



5. Observe el resultado en la consola



6. Estudiemos la estructura del proyecto:



7. Abrir y estudiar la clase de Dominio o Entity **/src/main/java/com.bolsadeideas.ejemplos.entity/Estudiante.java** (Típica clase Java con getter y setter)

```
package com.bolsadeideas.ejemplos.entity;
import java.util.Date;

public class Estudiante {
    private int id;
    private String nombre;
    private  Date fechaNacimiento;
    private int promedio;

    public Estudiante() {}

    public Estudiante(int id, String nombre, Date fechaNacimiento, int promedio) {
        this.id = id;
        this.nombre = nombre;
        this.fechaNacimiento = fechaNacimiento;
        this.promedio = promedio;
    }

  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public String getNombre() {
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  public Date getFechaNacimiento() {return fechaNacimiento;}

  public void setFechaNacimiento(Date fechaNacimiento) {
    this.fechaNacimiento = fechaNacimiento;
  }

  public int getPromedio() {return promedio;}

  public void setPromedio(int promedio) {
    this.promedio = promedio;
  }
}
```

8. Abrir y estudiar la interfaz **EstudianteDao**, que contiene el contrato para el CRUD, varios métodos típicos de persistencia:
   /src/main/java/com.bolsadeideas.ejemplos.dao/EstudianteDao.java

```
package com.bolsadeideas.ejemplos.dao;

import java.util.List;

import com.bolsadeideas.ejemplos.entity.Estudiante;

public interface EstudianteDao {
    public void save(Estudiante estudiante);
    public void update(Estudiante estudiante);
    public void delete(Estudiante estudiante);
    public Estudiante findById(int id);
    public List<Estudiante> findAll();
    public String getNombre(int id);
    public int count();
}
```

9. Abrir y estudiar la clase de implementación del EstudianteDao – JdbcEstudianteDao- , esta clase implementa la interfaz Dao usando JDBC con Spring:
   /src/main/java/ com.bolsadeideas.ejemplos.dao/JdbcEstudianteDao.java

```
package com.bolsadeideas.ejemplos.dao;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.support.JdbcDaoSupport;

import com.bolsadeideas.ejemplos.entity.Estudiante;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

// Heredamos de JdbcDaoSupport que agrega una implementación de JdbcTemplate.
public class JdbcEstudianteDao extends JdbcDaoSupport implements EstudianteDao {

    public void save(Estudiante estudiante) {
        String sql = "INSERT INTO ESTUDIANTE (ID, NOMBRE, FECHA_NACIMIENTO, PROMEDIO) "+"VALUES (?,?,?,?)";
        getJdbcTemplate().update(sql,
                new Object[] {estudiante.getId(), estudiante.getNombre(),
                        estudiante.getFechaNacimiento(),
                        estudiante.getPromedio() });
    }

    public Estudiante findById(int id) {
        return getJdbcTemplate().queryForObject("SELECT * FROM ESTUDIANTE WHERE ID = ?",
                BeanPropertyRowMapper.newInstance(Estudiante.class), id);
    }

    public void update(Estudiante estudiante) {
        String sql = "UPDATE ESTUDIANTE SET NOMBRE = ?, PROMEDIO = ?, FECHA_NACIMIENTO = ? WHERE ID = ?";
        getJdbcTemplate().update(sql,
                new Object[] { estudiante.getNombre(),
                        estudiante.getPromedio(),
                        estudiante.getFechaNacimiento(),
                        estudiante.getId() });
    }

    public void delete(Estudiante estudiante) {
        Map<String, Object> args = new HashMap<String, Object>();
        args.put("id", estudiante.getId());
        getJdbcTemplate().update("DELETE FROM ESTUDIANTE WHERE ID = ?", args);
    }

    public List<Estudiante> findAll() {
        return getJdbcTemplate().query("SELECT * FROM ESTUDIANTE",
                                       BeanPropertyRowMapper.newInstance(Estudiante.class));
    }

    public String getNombre(int id) {
      return getJdbcTemplate().queryForObject("SELECT NOMBRE FROM ESTUDIANTE WHERE ID=?", String.class, id);
    }
```

```
    public int count() {
        return getJdbcTemplate().queryForObject("SELECT COUNT(*) FROM ESTUDIANTE", Integer.class);
    }
}
```

10. Abrir y estudiar la clase Main **/src/main/java/com.bolsadeideas.ejemplos/Main.java**

```
package com.bolsadeideas.ejemplos;

import java.util.GregorianCalendar;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.bolsadeideas.ejemplos.dao.EstudianteDao;
import com.bolsadeideas.ejemplos.entity.Estudiante;

public class Main {

    public static void main(String[] args) {
        try (ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml")) {

            EstudianteDao estudianteDao = (EstudianteDao) context.getBean("estudianteDao");
            Estudiante estudiante = new Estudiante(0001, "Andrés Guzmán",
                                        new GregorianCalendar(1978, 1, 1).getTime(), 10);

            estudianteDao.save(estudiante);

            estudiante = estudianteDao.findById(0001);
            System.out.println("--->ID: " + estudiante.getId());
            System.out.println("--->Nombre: " + estudiante.getNombre());
            System.out.println("--->Fecha Nacimiento: " + estudiante.getFechaNacimiento());
            System.out.println("--->Promedio: " + estudiante.getPromedio());
            System.out.println("--->Total Estudiantes: " + estudianteDao.count());
            System.out.println("--->Obtener por Nombre: " + estudianteDao.getNombre(0001));
        }
    }
}
```

11. Abrir y estudiar beans.xml. El archivo de configuración del contexto de Spring, usado en la clase Main.java.
/src/main/resources/bean.xml.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
         http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

    <!-- importamos las configuraciones jdbc -->
     <import resource="spring/applicationContext-jdbc.xml"/>
</beans>
```

12. Abrir y estudiar applicationContext-jdbc.xml. Archivo de configuración de contexto con las definiciones JDBC - Es referenciado e incluido en el archivo beans.xml.
/src/main/resources/spring/applicationContext-jdbc.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:aop="http://www.springframework.org/schema/aop"
  xmlns:p="http://www.springframework.org/schema/p" xmlns:context="http://www.springframework.org/schema/context"
  xmlns:tx="http://www.springframework.org/schema/tx"
  xsi:schemaLocation="
      http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop.xsd
      http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
      http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context.xsd
      http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">

  <!-- importamos las definicion del dataSource -->
  <import resource="applicationContext-dataSource.xml" />

  <bean id="estudianteDao" class="com.bolsadeideas.ejemplos.dao.JdbcEstudianteDao">
    <property name="dataSource" ref="dataSource" />
  </bean>

</beans>
```

13. Abrir y estudiar applicationContext-dataSource.xml. Archivo de configuración de contexto con las definiciones de conexión DataSource - Es referenciado e incluido en el archivo applicationContext-jdbc.xml.
/src/main/resources/spring/applicationContext-dataSource.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:p="http://www.springframework.org/schema/p"
           xmlns:context="http://www.springframework.org/schema/context"
           xmlns:jdbc="http://www.springframework.org/schema/jdbc"
           xsi:schemaLocation="
                  http://www.springframework.org/schema/beans
                  http://www.springframework.org/schema/beans/spring-beans.xsd
                  http://www.springframework.org/schema/context
                  http://www.springframework.org/schema/context/spring-context.xsd
                  http://www.springframework.org/schema/jdbc
                  http://www.springframework.org/schema/jdbc/spring-jdbc.xsd">

   <!-- ========================= DEFINICION DATASOURCE ========================= -->

   <!-- Configuracion que reemplaza los placeholders ${...} placeholders con los valores del
archivo properties -->
   <!-- (En este caso, los datos relacionados a la conexion JDBC para el DataSource) -->
   <context:property-placeholder location="classpath:jdbc.properties"/>

   <!-- Configuracion DataSource usamos Apache Commons DBCP. -->
   <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-
method="close"
                p:driverClassName="${jdbc.driverClassName}"
                p:url="${jdbc.url}"
                p:username="${jdbc.username}"
                p:password="${jdbc.password}"/>

   <!-- Inicializamos la base de datos y sus tablas. Si falla los scripts,
          se detiene la Inicializacion . -->
   <jdbc:initialize-database data-source="dataSource">
          <jdbc:script location="${jdbc.initLocation}"/>
          <jdbc:script location="${jdbc.dataLocation}"/>
   </jdbc:initialize-database>

</beans>
```

14. Abrir y estudiar **jdbc.properties**. Archivo de configuración con los datos de conexión - Es referenciado e incluido en el archivo applicationContext-dataSource.xml.
/src/main/resources/jdbc.properties

```
# Properties file with JDBC and JPA settings.
#
# Applied by <context:property-placeholder location="jdbc.properties"/> from
# various application context XML files (e.g., "applicationContext-*.xml").
# Targeted at system administrators, to avoid touching the context XML files.

#-------------------------------------------------------------------------------
# HSQL Settings

jdbc.driverClassName=org.hsqldb.jdbcDriver
jdbc.url=jdbc:hsqldb:mem:estudiante
jdbc.username=sa
jdbc.password=

# Properties that control the population of schema and data for a new data source
jdbc.initLocation=classpath:db/hsqldb/initDB.txt
jdbc.dataLocation=classpath:db/hsqldb/populateDB.txt

#-------------------------------------------------------------------------------
# MySQL Settings

#jdbc.driverClassName=com.mysql.jdbc.Driver
#jdbc.url=jdbc:mysql://localhost:3306/test
#jdbc.username=root
#jdbc.password=

# Properties that control the population of schema and data for a new data source
#jdbc.initLocation=classpath:db/mysql/initDB.txt
#jdbc.dataLocation=classpath:db/mysql/populateDB.txt
```

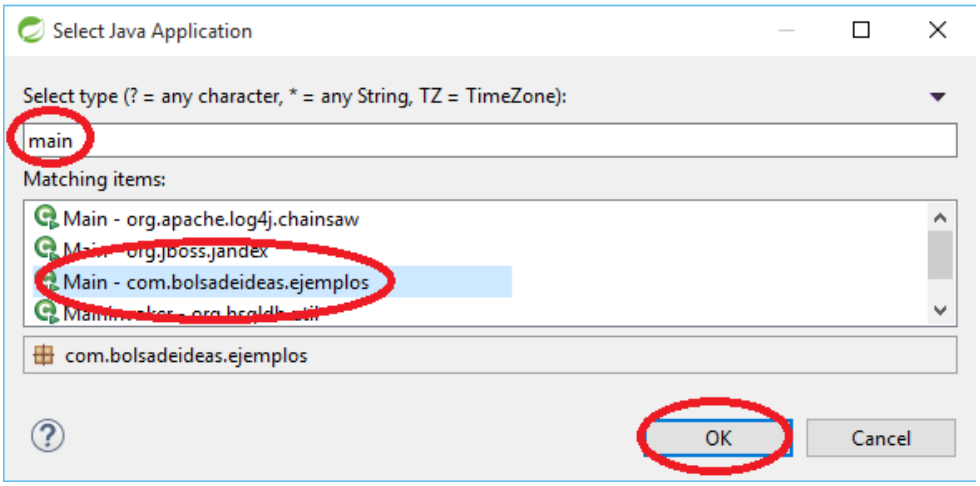
15. Abrir y estudiar el script SQL initDB.txt. Es referenciado e incluido en el archivo jdbc.properties.
/src/main/resources/db/hsqldb/initDB.txt

```
/* Create tables */
CREATE TABLE ESTUDIANTE (
    ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR(20) NOT NULL,
    FECHA_NACIMIENTO DATE NOT NULL,
    PROMEDIO INTEGER NOT NULL
);
```

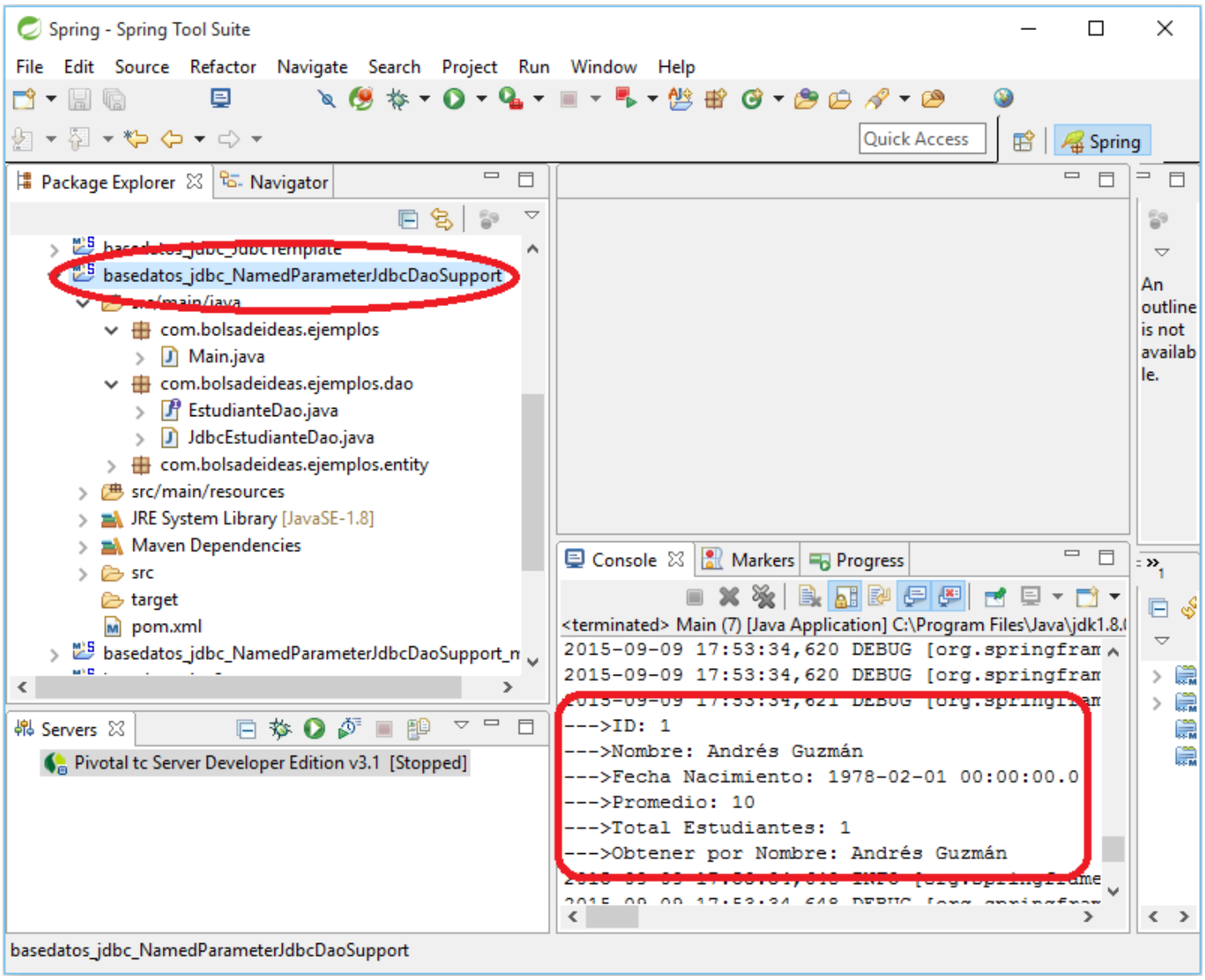
Ejercicio 2: Generar y ejecutar el ejemplo "basedatos_jdbc_NamedParameterJdbcDaoSupport"

En este ejemplo, a diferencia del anterior vamos a utilizar la clase NamedParameterJdbcDaoSupport que da soporte a parámetros de la consulta en base a **nombres de parámetros**, para implementar acceso a datos o DAOs, también proveen una plantilla que se encargan transparentemente de las operaciones típicas como consultas update, insert, delete. En resumen la única diferencia es que soporta y se basa de parámetros nombrado de las consultas, al estilo: **ID = :id**

1. Clic derecho sobre el proyecto y "**Run As->Maven Clean**".
2. Clic derecho sobre el proyecto y "**Run As->Maven Install**".
3. Clic derecho sobre el proyecto y **Maven->Update Project…**
4. Clic derecho sobre el proyecto **basedatos_jdbc_NamedParameterJdbcDaoSupport**
   - **Run As-> Java Application**



5. Observe el resultado en la consola



6. Abrir y estudiar la clase de implementación del EstudianteDao – NamedParameterJdbcDaoSupport - , esta clase implementa la interfaz Dao usando JDBC con Spring:

   /src/main/java/ com.bolsadeideas.ejemplos.dao/JdbcEstudianteDao.java

```
package com.bolsadeideas.ejemplos.dao;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.jdbc.core.namedparam.BeanPropertySqlParameterSource;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcDaoSupport;
import org.springframework.jdbc.core.namedparam.SqlParameterSource;

import com.bolsadeideas.ejemplos.entity.Estudiante;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

// NamedParameterJdbcDaoSupport hereda de JdbcDaoSupport y agrega una implementación de
NamedParameterJdbcTemplat.
public class JdbcEstudianteDao extends NamedParameterJdbcDaoSupport implements EstudianteDao {

    public void save(Estudiante estudiante) {

        String sql = "INSERT INTO ESTUDIANTE (ID, NOMBRE, FECHA_NACIMIENTO, PROMEDIO) "
                + "VALUES (:id, :nombre, :fechaNacimiento, :promedio)";
        // BeanPropertySqlParameterSource() recibe el objeto entity (Objeto Java
        // con getter y setter) como parámetro
        // con todos los datos, automaticamente getSimpleJdbcTemplate ejecuta la
        // consulta.
        SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(estudiante);
        getNamedParameterJdbcTemplate().update(sql, parameterSource);
    }

    public Estudiante findById(int id) {
        String sql = "SELECT * FROM ESTUDIANTE WHERE ID = ?";
        BeanPropertyRowMapper<Estudiante> estudianteRowMapper =
                                            BeanPropertyRowMapper.newInstance(Estudiante.class);

        Estudiante estudiante = getJdbcTemplate().queryForObject(sql, estudianteRowMapper, id);

        return estudiante;
    }

    public void update(Estudiante estudiante) {
        String sql = "UPDATE ESTUDIANTE SET NOMBRE = :nombre, PROMEDIO = :promedio,
                                            FECHA_NACIMIENTO = :fechaNacimiento WHERE ID = :id";

        SqlParameterSource parameterSource = new BeanPropertySqlParameterSource(estudiante);
        getNamedParameterJdbcTemplate().update(sql, parameterSource);
    }
```

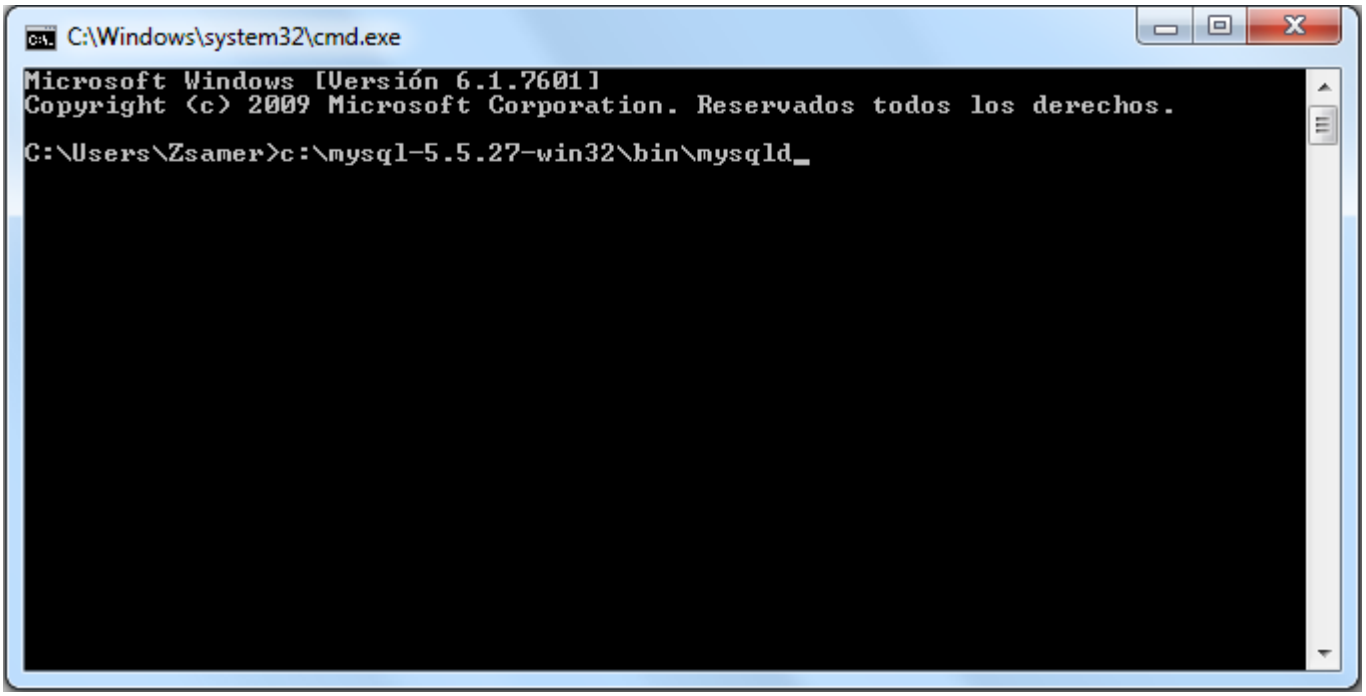
```
    public void delete(Estudiante estudiante) {
        Map<String, Object> args = new HashMap<String, Object>();
        args.put("id", estudiante.getId());
        String sql = "DELETE FROM ESTUDIANTE WHERE ID = :id";
        getNamedParameterJdbcTemplate().update(sql, args);
    }

    public List<Estudiante> findAll() {
        String sql = "SELECT * FROM ESTUDIANTE";
        RowMapper<Estudiante> rm = BeanPropertyRowMapper.newInstance(Estudiante.class);
        List<Estudiante> estudiantes = getJdbcTemplate().query(sql, rm);
        return estudiantes;
    }

    public String getNombre(int id) {
        String sql = "SELECT NOMBRE FROM ESTUDIANTE WHERE ID = :id";
        SqlParameterSource namedParameters =
                new MapSqlParameterSource("id", id);
        String nombre = getNamedParameterJdbcTemplate().queryForObject(sql, namedParameters, String.class);
        return nombre;
    }

    public int count() {
        String sql = "SELECT COUNT(*) FROM ESTUDIANTE";
        int count = getJdbcTemplate().queryForObject(sql, Integer.class);
        return count;
    }
}
```

Ejercicio 3: Generar y ejecutar el ejemplo " basedatos_jdbc_NamedParameterJdbcDaoSupport con mysql"

Es muy simple cambiar a cualquier otro motor de base de datos, sin siquiera cambiar el código.

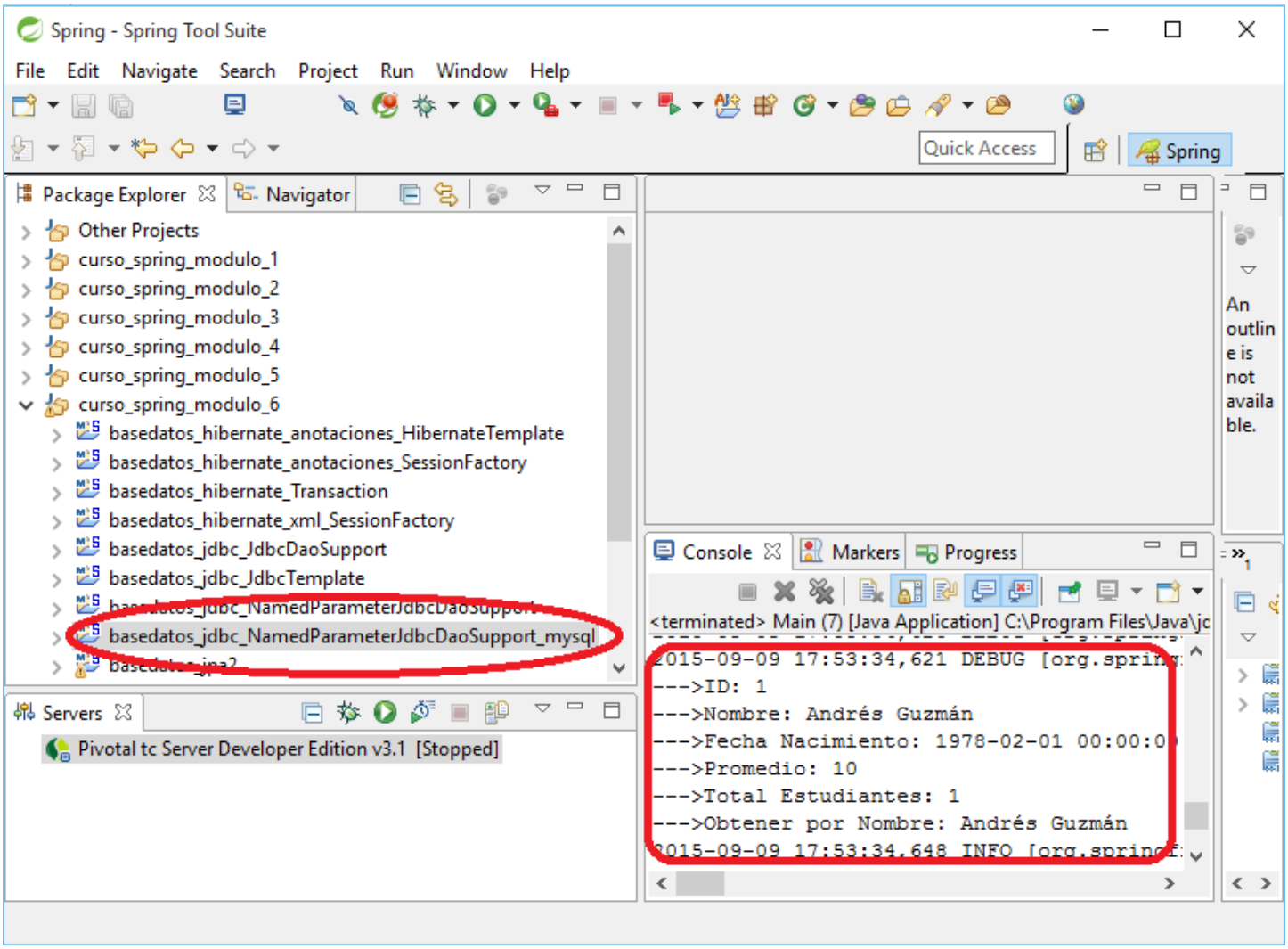
En este ejemplo, vamos a utilizar la base de datos MySQL.

1. Iniciar la base de datos Mysql:



2. Clic derecho sobre el proyecto y "**Run As->Maven Clean**".
3. Clic derecho sobre el proyecto y "**Run As->Maven Install**".
4. Clic derecho sobre el proyecto y **Maven->Update Project…**
5. Clic derecho sobre el proyecto **basedatos_jdbc_NamedParameterJdbcDaoSupport_mysql**
    - **Run As-> Java Application**

6. Observe el resultado en la consola



7. El único cambio que hay que hacer para usar MySQL en vez de HSQLDB es modificar el archivo jdbc.properties.

```
# Properties file with JDBC and JPA settings.
#
# Applied by <context:property-placeholder location="jdbc.properties"/> from
# various application context XML files (e.g., "applicationContext-*.xml").
# Targeted at system administrators, to avoid touching the context XML files.
#-------------------------------------------------------------------------------
# HSQL Settings

#jdbc.driverClassName=org.hsqldb.jdbcDriver
#jdbc.url=jdbc:hsqldb:mem:estudiante
#jdbc.username=sa
#jdbc.password=

# Properties that control the population of schema and data for a new data source
#jdbc.initLocation=classpath:db/hsqldb/initDB.txt
#jdbc.dataLocation=classpath:db/hsqldb/populateDB.txt

#-------------------------------------------------------------------------------
# MySQL Settings
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/curso_spring
jdbc.username=root
jdbc.password=

# Properties that control the population of schema and data for a new data source
jdbc.initLocation=classpath:db/mysql/initDB.txt
jdbc.dataLocation=classpath:db/mysql/populateDB.txt
```

8. initDB.txt para MySQL.

```
/* Create tables */
DROP TABLE IF EXISTS ESTUDIANTE;
CREATE TABLE IF NOT EXISTS ESTUDIANTE (
    ID INTEGER NOT NULL PRIMARY KEY,
    NOMBRE VARCHAR(20),
    FECHA_NACIMIENTO DATE NOT NULL,
    PROMEDIO INTEGER NOT NULL
    ) engine=InnoDB;
```

9. Y finalmente agregar la dependencia del conector de MySQL al pom.xml de maven:

```
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <version>5.1.36</version>
    </dependency>
```

Ejercicio 4: Generar y ejecutar el ejemplo "Hibernate mapping con XML"

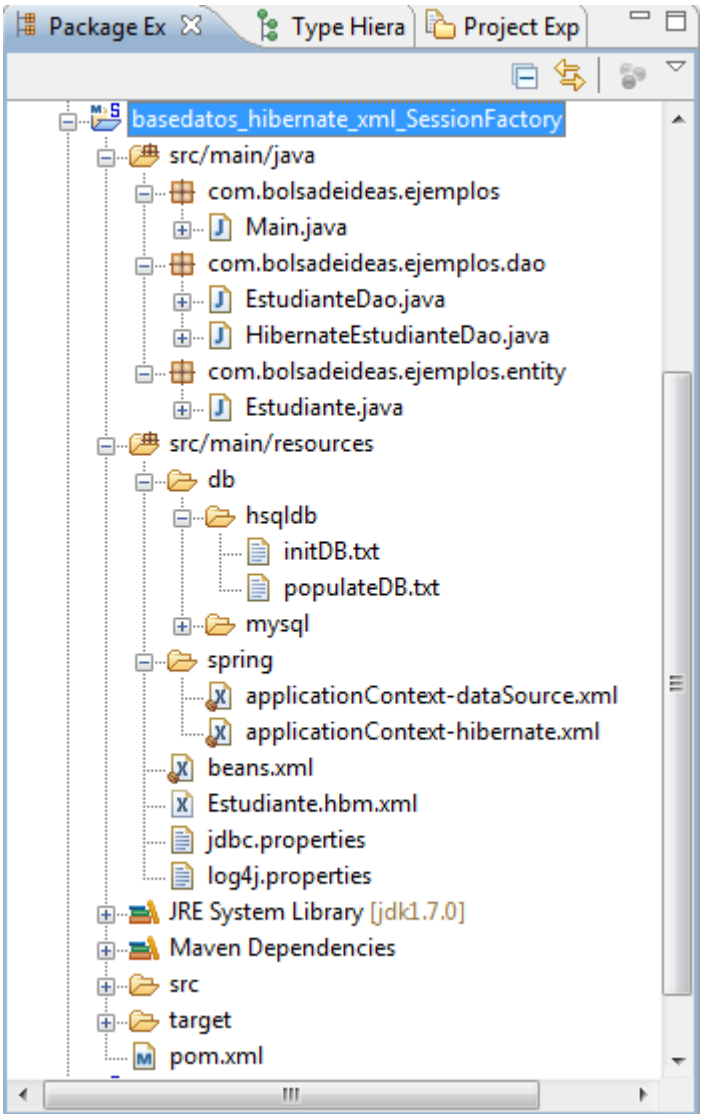
Para describir y configurar el mapeo de las clases de dominio o entity hacia las tablas de la base de datos, existen dos opciones – la primera es usar un archivo XML de mapeo la segunda es usar el API JPA con anotaciones, en este ejemplo veremos la primera forma.

Para la programación e implementación de clases de acceso a datos como los DAO (Objeto de acceso a datos), también tenemos dos opciones – la primera es usar el API Hibernate SessionFactory y la segunda es usar el API proveído por Spring HibernateTemplate, en este ejemplo también usaremos la primera forma con SessionFactory, luego veremos la segunda.

1. Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho sobre el proyecto y **Maven->Update Project…**
3. Clic derecho sobre el proyecto **basedatos_hibernate_xml_SessionFactory**
    - **Run As-> Java Application**
4. Observe el resultado en la consola

```
Hibernate: insert into ESTUDIANTE (ID, NOMBRE, FECHA_NACIMIENTO, PROMEDIO) values (default, ?, ?, ?)
Hibernate: select estudiante0_.ID as ID1_0_, estudiante0_.NOMBRE as NOMBRE2_0_,
estudiante0_.FECHA_NACIMIENTO as FECHA_NA3_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_ from ESTUDIANTE
estudiante0_
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.NOMBRE as NOMBRE2_0_0_,
estudiante0_.FECHA_NACIMIENTO as FECHA_NA3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
--->ID: 0
--->Nombre: Andrés Guzmán
--->Fecha Nacimiento: 1978-02-01
--->Promedio: 8
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.NOMBRE as NOMBRE2_0_0_,
estudiante0_.FECHA_NACIMIENTO as FECHA_NA3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
Hibernate: delete from ESTUDIANTE where ID=?
```

5. Estudiemos la estructura del proyecto:



6. Abrir y estudiar el archivo applicationContext-hibernate.xml. Archivo de configuración de contexto con las definiciones Hibernate - Es referenciado e incluido en el archivo beans.xml. **/src/main/resources/spring/applicationContext-hibernate.xml.**
    - En el ejemplo, usamos el archivo de mapeo XML - Estudiante.hbm.xml - para mapear la clase de entity Estudiante con la tabla ESTUDIANTE de la base de datos.

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd">
    <!-- ======================== DEFINICION HIBERNATE ======================== -->
    <!-- importamos las definicion del dataSource -->
    <import resource="applicationContext-dataSource.xml" />

    <!-- Configuracion que reemplaza los placeholders ${...} placeholders con los valores del archivo properties -->
    <!-- (En este caso, los datos relacionados a la conexion JDBC para el DataSource) -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- Hibernate SessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="mappingResources">
            <list>
                <value>com/bolsadeideas/ejemplos/entity/Estudiante.hbm.xml</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
                <prop key="hibernate.generate_statistics">${hibernate.generate_statistics}</prop>
            </props>
        </property>
    </bean>

    <!-- Transaction manager para un solo SessionFactory (es una alternativa a JTA) -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager"
        p:sessionFactory-ref="sessionFactory" />

    <!-- ============ DEFINICION OBJETOS DE NEGOCIO O ACCESO A DATOS ============ -->
    <!-- Scans el classpath de la aplicación, buscar y detecta los estereotipos
        @Components, @Repository, @Service, y @Controller y registra los beans en el contenedor de Spring -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.dao" />

    <!-- Activa las anotaciones para ser detectadas en las clases bean de Spring:@Required y @Autowired,@Resource -->
    <context:annotation-config />

    <!-- Instrucción de Spring para activar y manejar el transaction management de forma automática
        sobre las clases anotadas con @Repository. -->
    <tx:annotation-driven />
</beans>
```

7. Abrir y estudiar el archivo de mapeo XML Estudiante.hbm.xml. Mapea la clase entity con la tabla de la base de datos.
   /src/main/java/com/bolsadeideas/ejemplos/entity/Estudiante.hbm.xml.

```
<!DOCTYPE hibernate-mapping
        PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="com.bolsadeideas.ejemplos.entity">
   <class name="Estudiante" table="ESTUDIANTE">
          <id name="id" type="int" column="ID">
                <generator class="identity" />
          </id>
          <property name="nombre" type="string">
                <column name="NOMBRE" length="20" not-null="true" />
          </property>
          <property name="fechaNacimiento" type="date" column="FECHA_NACIMIENTO" />
          <property name="promedio" type="int" column="PROMEDIO" />
   </class>
</hibernate-mapping>
```

8. Abrir y estudiar la clase HibernateEstudianteDao , esta clase implementa la interfaz Dao usando Hibernate con Spring:

   /src/main/java/ com.bolsadeideas.ejemplos.dao/JdbcEstudianteDao.java

```
package com.bolsadeideas.ejemplos.dao;

import com.bolsadeideas.ejemplos.entity.Estudiante;

import org.hibernate.Query;
import org.hibernate.SessionFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Repository("estudianteDao")
public class HibernateEstudianteDao implements EstudianteDao {

    private SessionFactory sessionFactory;

    @Autowired
    public void setSessionFactory(SessionFactory sessionFactory) {
        this.sessionFactory = sessionFactory;
    }

    @Transactional
    public void save(Estudiante estudiante) {
        sessionFactory.getCurrentSession().saveOrUpdate(estudiante);
    }

    @Transactional
    public void delete(Estudiante estudiante) {
        Estudiante est = (Estudiante) sessionFactory.getCurrentSession().get(Estudiante.class,
estudiante.getId());
        sessionFactory.getCurrentSession().delete(est);
    }

    @Transactional(readOnly = true)
    public Estudiante findById(Integer id) {
        return (Estudiante) sessionFactory.getCurrentSession().get(Estudiante.class, id);
    }

    @SuppressWarnings("unchecked")
       @Transactional(readOnly = true)
    public List<Estudiante> findAll() {
        Query query = sessionFactory.getCurrentSession().createQuery("from Estudiante");
        return query.list();
    }
}
```

9. Abrir y estudiar la clase Entity **Estudiante.java**

```
package com.bolsadeideas.ejemplos.entity;
import java.util.Date;
public class Estudiante {
    private int id;
    private String nombre;
    private     Date fechaNacimiento;
    private int promedio;

    public Estudiante() {}

    public Estudiante(int id, String nombre, Date fechaNacimiento, int promedio) {
        this.id = id;
        this.nombre = nombre;
        this.fechaNacimiento = fechaNacimiento;
        this.promedio = promedio;
    }

   public int getId() {
          return id;
   }

   public void setId(int id) {
          this.id = id;
   }

   public String getNombre() {
          return nombre;
   }

   public void setNombre(String nombre) {
          this.nombre = nombre;
   }

   public Date getFechaNacimiento() {
          return fechaNacimiento;
   }

   public void setFechaNacimiento(Date fechaNacimiento) {
          this.fechaNacimiento = fechaNacimiento;
   }

   public int getPromedio() {
          return promedio;
   }

   public void setPromedio(int promedio) {
          this.promedio = promedio;
   }
}
```

Librerías y dependencias en el pom.xml de Hibernate

Las librerías que necesitamos agregar en el pom.xml para un proyecto Spring con hibernate son las relacionadas con Hibernate (ORM bases de datos), y serían las siguientes:

```
Etc…
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>4.3.11.Final</version>
        </dependency>
        <!-- JDBC drivers -->
        <dependency>
            <groupId>org.hsqldb</groupId>
            <artifactId>hsqldb</artifactId>
            <version>2.3.3</version>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>5.1.36</version>
        </dependency>
        <!-- DBCP2 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-dbcp2</artifactId>
            <version>2.1</version>
        </dependency>
```

Todas las que están marcadas en rojo son propias de Hibernate, el resto son dependencias de Spring para la integración con Hibernate y librerías externas o de terceros requeridas por Hibernate, por ejemplo hsqldb que es el motor de base de datos, si usáramos mysql se cambia por la librería de mysql, javassist es otra dependencia requerida por hibernate para manipular el bytecodes de Java que viene incluida dentro de hibernate-core. Otra dependencia importante es el commons-dbcp para manejar un data source con pool de conexiones en hibernate.

Finalmente también son necesarias las librerías de Simple Logging Facade for Java o slf4j, para el logger de Hibernate. Para el detalle completo de la estructura del archivo de maven pom.xml pueden revisarlo en el ejemplo.

Ejercicio 5: Generar y ejecutar el ejemplo "Hibernate mapping con anotaciones"

En el ejemplo, usaremos la anotación **@Entity** para describir el mapping de una clase entity o dominio. También usaremos el API **SessionFactory**.

1. Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho sobre el proyecto y **Maven->Update Project…**
3. Clic derecho sobre el proyecto **basedatos_hibernate_anotaciones_SessionFactory**
   - **Run As-> Java Application**
4. Observe el resultado en la consola

```
Hibernate: insert into ESTUDIANTE (ID, FECHA_NACIMIENTO, NOMBRE, PROMEDIO) values (default, ?, ?, ?)
Hibernate: select estudiante0_.ID as ID1_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_,
estudiante0_.NOMBRE as NOMBRE3_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_ from ESTUDIANTE estudiante0_
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_0_,
estudiante0_.NOMBRE as NOMBRE3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
--->ID: 0
--->Nombre: Andrés Guzmán
--->Fecha Nacimiento: 1978-02-01 00:00:00.0
--->Promedio: 8
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_0_,
estudiante0_.NOMBRE as NOMBRE3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
Hibernate: delete from ESTUDIANTE where ID=?
```

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

5. Abrir y estudiar el archivo applicationContext-hibernate.xml. Ahora modificando para que los archivos de mapeas sean las mismas clases Entity anotadas.
   /src/main/resources/spring/applicationContext-hibernate.xml.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="
            http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd
            http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-
context.xsd
            http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">

    <!-- ======================== DEFINICION HIBERNATE ======================== -->
    <!-- importamos las definicion del dataSource -->
    <import resource="applicationContext-dataSource.xml" />

    <!-- Configuracion que reemplaza los placeholders ${...} placeholders con los valores del archivo properties -->
    <!-- (En este caso, los datos relacionados a la conexion JDBC para el DataSource) -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- Hibernate SessionFactory, para anotaciones en las clases entity usamos atributo annotatedClasses -->
    <bean id="sessionFactory"
         class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
         <property name="dataSource" ref="dataSource" />
          <property name="annotatedClasses">
              <list>
                  <value>com.bolsadeideas.ejemplos.entity.Estudiante</value>
              </list>
          </property>
         <property name="hibernateProperties">
             <props>
                 <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                 <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
                 <prop key="hibernate.generate_statistics">${hibernate.generate_statistics}</prop>
             </props>
         </property>
     </bean>

    <!-- Transaction manager para un solo SessionFactory (es una alternativa a JTA) -->
    <bean id="transactionManager"
        class="org.springframework.orm.hibernate4.HibernateTransactionManager"
        p:sessionFactory-ref="sessionFactory" />

    <!-- ======================== BUSINESS OBJECT DEFINITIONS ======================== -->
    <!-- ============ DEFINICION OBJETOS DE NEGOCIO O ACCESO A DATOS ============ -->
    <!-- Scans el classpath de la aplicación, buscar y detecta los estereotipos
        @Components, @Repository, @Service, y @Controller y registra los beans en
        el contenedor de Spring -->
    <context:component-scan base-package="com.bolsadeideas.ejemplos.dao" />

    <!--Activa las anotaciones para ser detectadas en las clases bean de Spring: @Required y @Autowired,@Resource -->
    <context:annotation-config />

    <!-- Instrucción de Spring para activar y manejar el transaction management de forma automática
        sobre las clases anotadas con @Repository. -->
    <tx:annotation-driven />

</beans>
```

6. Abrir y estudiar la clase Entity **Estudiante.java**

```
package com.bolsadeideas.ejemplos.entity;
import java.util.Date;

import javax.persistence.*;

@Entity
@Table(name = "ESTUDIANTE")
public class Estudiante {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "ID")
   private Integer id;
   @Column(name = "NOMBRE", length = 20, nullable = false)
   private String nombre;
   @Column(name = "FECHA_NACIMIENTO")
   private Date fechaNacimiento;
   @Column(name = "PROMEDIO")
   private int promedio;

   public Estudiante() {}

   public Estudiante(int id, String nombre, Date fechaNacimiento, int promedio) {
          this.id = id;
          this.nombre = nombre;
          this.fechaNacimiento = fechaNacimiento;
          this.promedio = promedio;
   }

   public int getId() {return id;}

   public void setId(int id) {this.id = id;}

   public String getNombre() {return nombre;}

   public void setNombre(String nombre) {this.nombre = nombre;}

   public Date getFechaNacimiento() {return fechaNacimiento;}

   public void setFechaNacimiento(Date fechaNacimiento) {
        this.fechaNacimiento = fechaNacimiento;
   }

   public int getPromedio() {return promedio;}

   public void setPromedio(int promedio) {this.promedio = promedio;}
}
```

- **@Entity**: Indica que es una clase POJO de Entidad (representa una tabla en la base de datos)
- **@Table**: Específica la tabla principal relacionada con la entidad.
    - name – nombre de la tabla, por defecto el de la entidad si no se especifica.
    - catalog – nombre del catálogo.
    - schema – nombre del esquema.
    - uniqueConstraints – constrains entre tablas relacionadas con la anotación @Column y @JoinColumn
- **@Column**: Especifica una columna de la tabla a mapear con un campo de la entidad.
    - name - nombre de la columna.
    - unique - si el campo tiene un único valor.
    - nullable - si permite nulos.
    - insertable - si la columna se incluirá; en la sentencia INSERT generada.
    - updatable - si la columna se incluirá; en la sentencia UPDATE generada.
    - table - nombre de la tabla que contiene la columna.
    - length - longitud de la columna.
    - precision - número de dígitos decimales.
    - scale - escala decimal.
- **@Id**: Indica la clave primaria de la tabla.
- **@GeneratedValue**: Asociado con la clave primaria, indica que ésta se debe generar por ejemplo con una secuencia de la base de datos.
    - strategy – estrategia a seguir para la generación de la clave: AUTO (valor por defecto, el contenedor decide la estrategia en función de la base de datos), IDENTITY (utiliza un contador, ej: MySQL), SEQUENCE (utiliza una secuencia, ej: Oracle, PostgreSQL) y TABLE (utiliza una tabla de identificadores).
    - generator – forma en la que genera la clave.

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Ejercicio 6: Generar y ejecutar el ejemplo "Hibernate mapping con anotaciones y usando HibernateTemplate"

En el ejemplo, también usaremos anotaciones para el mapping. Pero además usaremos el API HibernateTemplate en vez de SessionFactory.

1. Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho sobre el proyecto y **Maven->Update Project…**
3. Clic derecho sobre el proyecto **basedatos_hibernate_anotaciones_HibernateTemplate**
   - **Run As-> Java Application**
4. Observe el resultado en la consola

```
Hibernate: insert into ESTUDIANTE (ID, FECHA_NACIMIENTO, NOMBRE, PROMEDIO) values (default, ?, ?, ?)
Hibernate: select estudiante0_.ID as ID1_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_,
estudiante0_.NOMBRE as NOMBRE3_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_ from ESTUDIANTE estudiante0_
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_0_,
estudiante0_.NOMBRE as NOMBRE3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
--->ID: 0
--->Nombre: Andrés Guzmán
--->Fecha Nacimiento: 1978-02-01 00:00:00.0
--->Promedio: 8
Hibernate: select estudiante0_.ID as ID1_0_0_, estudiante0_.FECHA_NACIMIENTO as FECHA_NA2_0_0_,
estudiante0_.NOMBRE as NOMBRE3_0_0_, estudiante0_.PROMEDIO as PROMEDIO4_0_0_ from ESTUDIANTE
estudiante0_ where estudiante0_.ID=?
Hibernate: delete from ESTUDIANTE where ID=?
```

1. Abrir y estudiar la clase HibernateStudentDao

```
package com.bolsadeideas.ejemplos.dao;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.orm.hibernate3.HibernateTemplate;
import org.springframework.stereotype.Repository;
import org.springframework.transaction.annotation.Transactional;

import com.bolsadeideas.ejemplos.entity.Estudiante;

@Repository("estudianteDao")
public class HibernateStudentDao implements EstudianteDao {

    @Autowired
    private HibernateTemplate hibernateTemplate;

    @Transactional
    public void save(Estudiante estudiante) {
        hibernateTemplate.saveOrUpdate(estudiante);
    }

    @Transactional
    public void delete(Estudiante estudiante) {
    Estudiante est = (Estudiante) hibernateTemplate.get(Estudiante.class, estudiante.getId());
        hibernateTemplate.delete(est);
    }

    @Transactional(readOnly = true)
    public Estudiante findById(Integer id) {
        return (Estudiante) hibernateTemplate.get(Estudiante.class, id);
    }

    @Transactional(readOnly = true)
    public List<Estudiante> findAll() {
        return hibernateTemplate.find("from Estudiante");
    }
}
```

- Spring nos provee la clase **HibernateTemplate** para brindarle a nuestros DAO soporte para Hibernate de forma más sencilla y directa.
- **HibernateTemplate**, en particular, contiene varios métodos útiles, tales como guardar, borrar, listar y obtener los objetos de la base de datos que simplifican el uso de Hibernate.
- Estos métodos suelen encapsular varias excepciones propias de acceso a datos de Hibernate (y SQL) dentro de una DataAccessException (que hereda de RuntimeException). Internamente manipula el SessionFactory la session de Hibernate.

Ejercicio 7: Generar y ejecutar el ejemplo "Hibernate manejo de Transacciones"

Sin el uso de transacciones, nuestra base de datos podría tener inconsistencia en sus datos. En el siguiente ejemplo llevaremos a cabo dos transferencias bancarias entre cuentas y contabilizaremos "las transferencias totales realizadas".

En el campo SALDO de la tabla CUENTA tendremos una restricción que valide que el valor del SALDO no sea negativo **CHECK (SALDO > = 0)**, cualquier operación de transferencia que viole esta regla, generará una excepción en la base de datos.

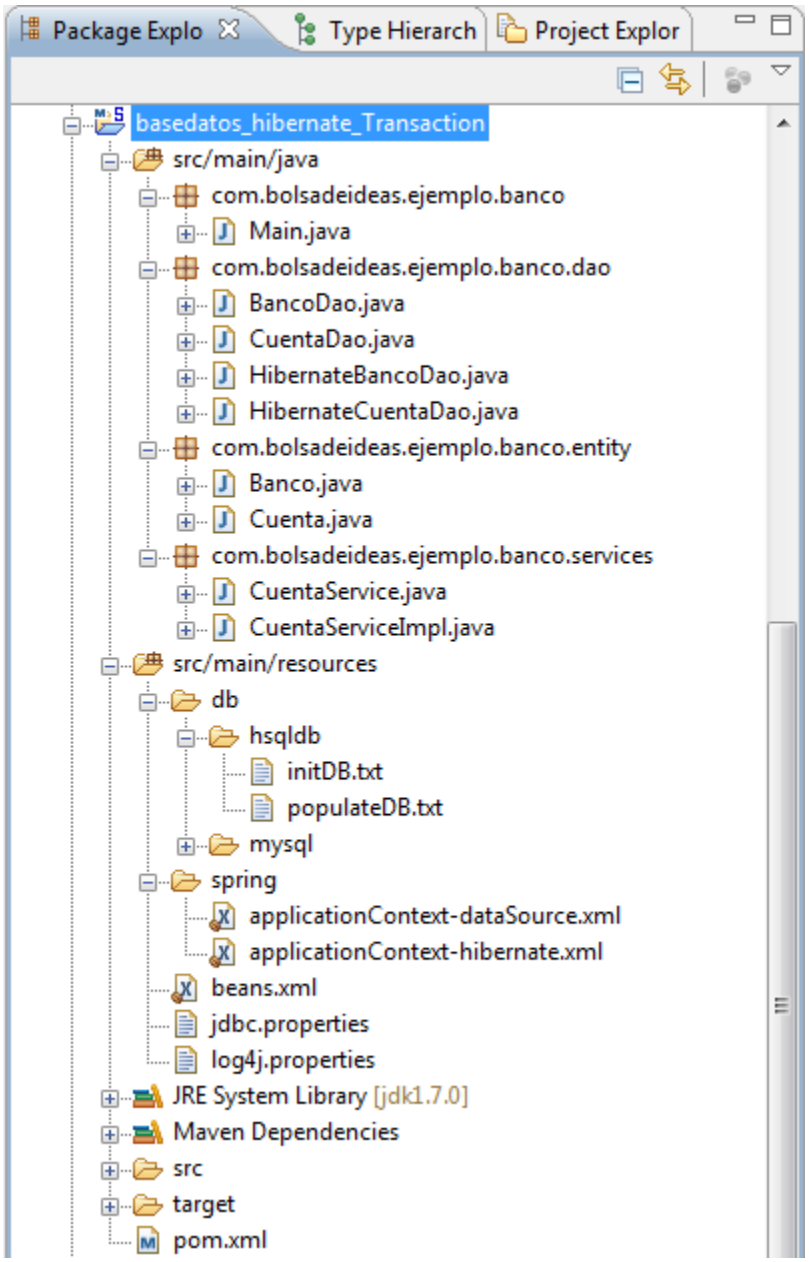
En cuanto a la configuración de transacción, se agrega el bean del manejador de transacción de Spring en el archivo XML, **Transaction Manager**, y anotar los métodos de las clases Service o Repository (Los DAO) con **@Transaction**, mediante la definición de **<tx:annotation-driven />** en el archivo de contexto. En otras palabras, no hay cambio alguno en el código.

1. Clic derecho sobre el proyecto **Run As->Maven Clean** y **Run As->Maven Install**.
2. Clic derecho sobre el proyecto y **Maven->Update Project…**
5. Clic derecho sobre **basedatos_hibernate_Transaction -> Run As-> Java Application**
3. Observe el resultado en la consola
    - Por favor no se alarme con la siguiente excepción, es justo lo que estábamos esperando esperado. Es de esperar que en la segunda transferencia ocurra el error y no se lleve a cabo, ya que estamos tratando transferir 1200 dólares de la cuenta origen, cuando tenemos sólo 900 dólares, lo cual viola la regla en el que el saldo tiene que ser no-negativo.
    - Observamos que a pesar del fallo de la segunda transferencia, el número total de transferencias no cambia, se mantiene igual en 1.
    - Es justamente el comportamiento esperado, ya que al fallar la segunda transacción y pese el haber incrementado el número de transacciones antes de fallar, al ocurrir la excepción hace un rollback (cuenta atrás) y deja sin efecto las transacciones anteriores.

```
Antes de la 1ra transferencia: Saldo de la cuenta origen = 1000 Saldo de la cuenta destino = 2000
Antes de la 1ra transferencia: Total de transferencias = 0
Después de la 1ra transferencia: Saldo origen = 900 Saldo cuenta destino = 2100
Después de la 1ra transferencia: Total transferencias = 1
2012-08-04 15:16:56,246 WARN [org.hibernate.util.JDBCExceptionReporter] - <SQL Error: 0, SQLState: null>
2012-08-04 15:16:56,246 ERROR [org.hibernate.util.JDBCExceptionReporter] - <failed batch>
2012-08-04 15:16:56,261 ERROR [org.hibernate.event.def.AbstractFlushingEventListener] - <Could not
synchronize database state with session>
org.hibernate.exception.GenericJDBCException: Could not execute JDBC batch update
    at org.hibernate.exception.SQLStateConverter.handledNonSpecificException(SQLStateConverter.java:126)
    at org.hibernate.exception.SQLStateConverter.convert(SQLStateConverter.java:114)
    at org.hibernate.exception.JDBCExceptionHelper.convert(JDBCExceptionHelper.java:66)
    at org.hibernate.jdbc.AbstractBatcher.executeBatch(AbstractBatcher.java:275)
    at org.hibernate.engine.ActionQueue.executeActions(ActionQueue.java:266)
```

```
    at org.hibernate.engine.ActionQueue.executeActions(ActionQueue.java:168)
    at
org.hibernate.event.def.AbstractFlushingEventListener.performExecutions(AbstractFlushingEventListener.jav
a:321)
    at org.hibernate.event.def.DefaultFlushEventListener.onFlush(DefaultFlushEventListener.java:50)
    at org.hibernate.impl.SessionImpl.flush(SessionImpl.java:1028)
    at org.hibernate.impl.SessionImpl.managedFlush(SessionImpl.java:366)
    at org.hibernate.transaction.JDBCTransaction.commit(JDBCTransaction.java:137)
    at
org.springframework.orm.hibernate3.HibernateTransactionManager.doCommit(HibernateTransactionManager.java:
656)
    at
org.springframework.transaction.support.AbstractPlatformTransactionManager.processCommit(AbstractPlatform
TransactionManager.java:754)
    at
org.springframework.transaction.support.AbstractPlatformTransactionManager.commit(AbstractPlatformTransac
tionManager.java:723)
    at
org.springframework.transaction.interceptor.TransactionAspectSupport.commitTransactionAfterReturning(Tran
sactionAspectSupport.java:393)
    at
org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.java:120
)
    at
org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:172)
    at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:202)
    at $Proxy17.transferir(Unknown Source)
    at com.bolsadeideas.ejemplo.banco.Main.main(Main.java:46)
Caused by: java.sql.BatchUpdateException: failed batch
    at org.hsqldb.jdbc.jdbcStatement.executeBatch(Unknown Source)
    at org.hsqldb.jdbc.jdbcPreparedStatement.executeBatch(Unknown Source)
    at org.apache.commons.dbcp.DelegatingStatement.executeBatch(DelegatingStatement.java:297)
    at org.hibernate.jdbc.BatchingBatcher.doExecuteBatch(BatchingBatcher.java:70)
    at org.hibernate.jdbc.AbstractBatcher.executeBatch(AbstractBatcher.java:268)
    ... 16 more
--->Código de Error: java.sql.BatchUpdateException: failed batch
--->Mensaje de Error: Hibernate flushing: Could not execute JDBC batch update; uncategorized SQLException
for SQL [update CUENTA set PERSONA=?, SALDO=? where ID=?]; SQL state [null]; error code [0]; failed
batch; nested exception is java.sql.BatchUpdateException: failed batch
Después de la 2da transferencia: Saldo cuenta origen = 900 Saldo cuenta destino  = 2100
Después de la 2da transferencia: Total transferencias = 1
```

4. Estudiar la estructura del proyecto:



5. Abrir y estudiar el script SQL initDB.txt. Tenga en cuenta que la columna de SALDO tiene que ser 0 (cero) o mayor a cero, de ser negativo ocurrirá una excepción.
   /src/main/resources/db/hsqldb/initDB.txt

```
/* Create tables */
CREATE TABLE CUENTA (
    ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    PERSONA VARCHAR(50)  NOT NULL,
    SALDO INT NOT NULL,
    CHECK (SALDO >= 0)
);

CREATE TABLE BANCO (
    ID INTEGER GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    NOMBRE VARCHAR(50) NOT NULL,
    TOTAL_TRANSFERENCIAS INT NOT NULL
   );
```

6. Abrir y estudiar el script SQL populateDB.txt.
   /src/main/resources/db/hsqldb/populateDB.txt

```
/* Populate tables */
INSERT INTO CUENTA(ID, PERSONA, SALDO) VALUES(0001, 'Andrés Guzmán', 1000);
INSERT INTO CUENTA(ID, PERSONA, SALDO) VALUES(0002, 'John Doe', 2000);

INSERT INTO BANCO(ID, NOMBRE, TOTAL_TRANSFERENCIAS) VALUES(0001, 'El Banco Financiero', 0);
```

7. Abrir y estudiar la clase Main.java.
    - En el código, se realizan dos operaciones de transferencia - la primera transferencia es de $100 y la segunda es de $1200. El saldo inicial del saldo origen es $ 1000, por lo que la primera transferencia debería funcionar sin problemas, mientras que la segunda transferencia resulta el lanzamiento de una excepción de la base de datos, ya que se queda con saldo negativo.

/src/main/java/com.bolsadeideas.ejemplo.banco/Main.java

```
package com.bolsadeideas.ejemplo.banco;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import org.springframework.dao.DataAccessException;

import com.bolsadeideas.ejemplo.banco.services.CuentaService;

public class Main {
  public static void main(String[] args) throws Throwable {
    ApplicationContext context = new ClassPathXmlApplicationContext(
        "beans.xml");

    int numCuentaOrigen = 0001;
    int numCuentaDestino = 0002;
    int bancoId = 0001;

    CuentaService cuentaService = (CuentaService) context.getBean("cuentaService");

    int saldoOrigen = cuentaService.revisarSaldo(numCuentaOrigen);
    int saldoDestino = cuentaService.revisarSaldo(numCuentaDestino);

    System.out.println("Antes de la 1ra transferencia: Saldo de la cuenta origen = " +
saldoOrigen + " Saldo de la cuenta destino = " + saldoDestino);
    System.out.println("Antes de la 1ra transferencia: Total de transferencias = " +
cuentaService.revisarTotalTransferencias(bancoId));

    try {
      // Transferir 100 USD desde la cuenta de origen hacia la cuenta de destino.
      // La cuenta de origen tiene suficientes fondos, por lo tanto aceptará la transferencia.
      int monto = 100;
      cuentaService.transferir(numCuentaOrigen, numCuentaDestino, monto);

    } catch (DataAccessException dataAccessException) {
      System.out.println("--->Código de Error: " + dataAccessException.getCause());
      System.out.println("--->Mensaje de Error: " + dataAccessException.getMessage());
    }

    saldoOrigen = cuentaService.revisarSaldo(numCuentaOrigen);
    saldoDestino = cuentaService.revisarSaldo(numCuentaDestino);
    System.out.println("Después de la 1ra transferencia: Saldo origen = " + saldoOrigen + "
Saldo cuenta destino = " + saldoDestino);
```

```
    System.out.println("Después de la 1ra transferencia: Total transferencias = " +
cuentaService.revisarTotalTransferencias(bancoId));

    try {
      // Transferir 1200 USD desde la cuenta de origen hacia la cuenta de destino.
      // La cuenta de origen NO tiene fondos suficientes, por lo tanto NO aceptará la
transferencia
      // y generará una excepción
      int monto = 1200;
      cuentaService.transferir(numCuentaOrigen, numCuentaDestino, monto);
    } catch (DataAccessException dataAccessException) {
      System.out.println("--->Código de Error: " + dataAccessException.getCause());
      System.out.println("--->Mensaje de Error: " + dataAccessException.getMessage());
    }

    saldoOrigen = cuentaService.revisarSaldo(numCuentaOrigen);
    saldoDestino = cuentaService.revisarSaldo(numCuentaDestino);
    System.out.println("Después de la 2da transferencia: Saldo cuenta origen = " + saldoOrigen
+ " Saldo cuenta destino  = " + saldoDestino);
    System.out.println("Después de la 2da transferencia: Total transferencias = " +
cuentaService.revisarTotalTransferencias(bancoId));

  }
}
```

8. Abrir y estudiar la interfaz Service **CuentaService**
   /src/main/java/com.bolsadeideas.ejemplo.banco.services/CuentaService.java

```
package com.bolsadeideas.ejemplo.banco.services;

public interface CuentaService {
    public int revisarTotalTransferencias(int bancoId);
    public int revisarSaldo(int numCuenta);
    public void transferir(int numCuentaOrigen, int numCuentaDestino, int monto);
}
```

9. Abrir y estudiar la clase **CuentaServiceImpl** implementación de la interfaz CuentaService.
   - Nótese que en la clase, la operación de transferencia se realiza dentro del marco de una transacción, a través de la anotación en el método **@Transaction**.
   - Además mantenemos un registro del "número de totales de transferencias", el incremento no debería hacerse cuando ocurre una excepción, ya que está dentro de una misma transacción y en el momento de ocurrir un problema y lanzamiento de una excepción debería hacer un rollback y anular el incremente hecho previamente al error.
   - De esta forma contamos con la protección de una transacción, manteniendo por lo tanto, una consistencia e integridad en los datos de nuestra base de datos.

```
package com.bolsadeideas.ejemplo.banco.services;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.bolsadeideas.ejemplo.banco.dao.CuentaDao;
import com.bolsadeideas.ejemplo.banco.dao.BancoDao;
import com.bolsadeideas.ejemplo.banco.entity.Cuenta;
import com.bolsadeideas.ejemplo.banco.entity.Banco;

@Service("cuentaService")
public class CuentaServiceImpl implements CuentaService {

  @Autowired
  private CuentaDao cuentaDao;

  @Autowired
  private BancoDao bancoDao;

  public int revisarTotalTransferencias(int bancoId) {
    Banco banco = bancoDao.findById(bancoId);
    return banco.getTotalTransferencias();
  }

  public int revisarSaldo(int numCuenta) {
    Cuenta cuenta = cuentaDao.findById(numCuenta);
    return cuenta.getSaldo();
  }

  @Transactional
  public void transferir(int numCuentaOrigen, int numCuentaDestino, int monto) {

    // Incrementamos el numero total de transferencias del banco
    Banco banco = bancoDao.findById(0001);
    int totalTransferencias = banco.getTotalTransferencias();
    banco.setTotalTransferencias(++totalTransferencias);
    bancoDao.update(banco);

    // Retirar la cantidad a ser transferida de la cuenta de origen
    Cuenta cuentaOrigen = cuentaDao.findById(numCuentaOrigen);
    cuentaOrigen.debito(monto);
    cuentaDao.update(cuentaOrigen);

    // Depositar la cantidad a la cuenta de destino
    Cuenta cuentaDestino = cuentaDao.findById(numCuentaDestino);
    cuentaDestino.credito(monto);
    cuentaDao.update(cuentaDestino);
  }
}
```

10. Abrir y estudiar el archivo de configuración del contexto Spring

/src/main/resources/spring/applicationContext-hibernate.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"
    xmlns:context="http://www.springframework.org/schema/context" xmlns:tx="http://www.springframework.org/schema/tx"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/context
                        http://www.springframework.org/schema/context/spring-context.xsd
                        http://www.springframework.org/schema/tx
                        http://www.springframework.org/schema/tx/spring-tx.xsd">
    <!-- ========================= DEFINICION HIBERNATE ========================= -->
    <!-- importamos las definicion del dataSource -->
    <import resource="applicationContext-dataSource.xml" />

    <!-- Configuracion que reemplaza los placeholders ${...} placeholders con
        los valores del archivo properties -->
    <!-- (En este caso, los datos relacionados a la conexion JDBC para el DataSource) -->
    <context:property-placeholder location="classpath:jdbc.properties" />

    <!-- Hibernate SessionFactory -->
    <bean id="sessionFactory"
        class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
        <property name="dataSource" ref="dataSource" />
        <property name="annotatedClasses">
            <list>
                <value>com.bolsadeideas.ejemplo.banco.entity.Cuenta</value>
                <value>com.bolsadeideas.ejemplo.banco.entity.Banco</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">${hibernate.dialect}</prop>
                <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
                <prop key="hibernate.generate_statistics">${hibernate.generate_statistics}</prop>
            </props>
        </property>
    </bean>

    <!-- Transaction manager para un solo SessionFactory (es una alternativa a JTA) -->
     <bean id="transactionManager"
         class="org.springframework.orm.hibernate4.HibernateTransactionManager"
         p:sessionFactory-ref="sessionFactory" />

    <!-- ========================= BUSINESS OBJECT DEFINITIONS ========================= -->
    <!-- ============ DEFINICION OBJETOS DE NEGOCIO O ACCESO A DATOS ============ -->
    <!-- Scans el classpath de la aplicación, buscar y detecta los estereotipos
        @Components, @Repository, @Service, y @Controller y registra los beans en
        el contenedor de Spring -->
    <context:component-scan base-package="com.bolsadeideas.ejemplo.banco" />

    <!--Activa las anotaciones para ser detectadas en las clases bean de Spring: @Required y @Autowired, @Resource-->
    <context:annotation-config />

    <!-- Instrucción de Spring para activar y manejar el transaction management de forma automática
        sobre las clases anotadas con @Repository. -->
     <tx:annotation-driven />

</beans>
```

Resumen

En este capítulo vimos todo lo relacionado a la persistencia y base de datos con Spring.

Estudiamos diferentes ejemplos de tecnologías de acceso a datos, principalmente con JDBC e Hibernate.

FIN.

Envía tus consultas a los foros!

Aquí es cuando debes sacarte todas las dudas haciendo consultas en los foros correspondientes

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com

Lectura Recomendada y Bibliografía

- JDBC: recomendable lectura del manual oficial para complementar con este workshop.
- Dao Support: Recomendable lectura de esta sección del manual oficial para complementar con este workshop.
- ORM: manual oficial ORM Spring para complementar con este workshop.
- Hibernate: manual oficial de Hibernate para complementar con este workshop.

AUTOR: Andrés Guzmán Fontecilla.
Email: andresguzf@gmail.com
WEB: http://www.bolsadeideas.com