## Ejercicio 1

1. Explicar que significa el Autowired, Repository y que tiene que ver con Inyección de Dependencias.
2. Revisar el método EmployeeJDCB “find” y explicar qué es lo que hace.
3. ¿Qué pruebas realizarían para este método?

TIP: No olvidar que las pruebas son requerimientos que debe cumplir la clase.

1. Elegir una prueba e implementarla utilizando la estructura de una prueba de BD.
   1. Crear únicamente método de la prueba sin ningún contenido, no olvidar utilizar un buen nombre.
   2. Comenzaremos por el Arrange y Act. Instanciar la clase JDBC bajo prueba

TIP: Para instanciar una fecha puede utilizar “Date.valueOf("2010-12-30")”

* 1. Inserta data que cumpla los filtros utilizados en el método “find” y data que no cumple los criterios.
  2. ASSERT, verifica que los datos devueltos son la misma cantidad que datos ingresados que cumplen con los filtros.

Antes de continuar comprueba que la prueba sea similar a la siguiente:

@Test

**public** **void** find\_withLastNameFilter\_returnTheEmployeesWithTheExactLastName() **throws** Exception {

LoadData(**new** Employee("Pacheco", Date.*valueOf*("2012-12-29")));

List<Employee> employees = employeeJDBC.find("Pacheco", **null**, **null**);

Assert.*assertEquals*(1, employees.size());

}

* 1. Reestablecer la BD usando TRANSACITON ROLLBACK.- Sprint-Test nos permitirá ejecutar nuestras pruebas dentro de una transacción y realizar un rollback si así lo deseamos. Agregar las siguientes anotaciones a la clase EmployeeJDBCTest.
     + “@RunWith(SpringJUnit4ClassRunner.class)” Nos permite utilizar funcionalidad de spring dentro de JUnit, por ejemplo instanciar e inyectar las dependencias.
     + “@ContextConfiguration("jdbc-ctx.xml")” Nos permite indicar a Spring cuál es el archivo (“context file”) donde se encontrará la definición de las clases a instanciar e inyectar.
     + “@Transaction” Nos permite ejecutar cada método de una clase dentro de su propia transacción.
     + “@TransactionConfiguration(defaultRollback=true)” Nos permite realizar siempre un rollback automático al finalizar la transacción.

Nota: Para utilizar transacciones con Spring se necesita la definición de un bean “transactionManager” dentro del “context file”. Revisar que existe un bean “transactionManager” dentro de este archivo.

1. Implementa la 2da prueba para el método “find”.
2. Implementa una prueba para el método “créate”

TIP: ¿Esta prueba necesita inicializar datos previamente?

TIP: En el Assert, utilizar el método “get” de la clase “EmployeeJDBC” para consultar si el registro existe en la BD.

1. Implementa una prueba para el método “delete”.

TIP: En el Assert, utiliza nuevamente el método “get” de la clase “EmployeeJDBC” para consultar si el registro existe en la BD.

1. REFLEXION:

¿Qué pasaría si necesitamos probar el método “find” y no existe un método “create” para inicializar datos?   
¿Qué pasaría si necesitamos probar los métodos “create” o “delete no existiría un método “find” para consultar si el registro existe en la BD?

## Ejercicio 2

1. Del ejercicio anterior “EmployeeJDBCTest\_SelfContained\_Transaction”, elegir una de las pruebas implementadas para el método “find” y copiarla completamente a este ejercicio. Luego de copiar la prueba, eliminar las líneas que contienen el método LoadData.  
     
   Con esto ya tenemos el Arrange-Act-Assert de nuestra prueba pero nos falta un mecanismo para inicializar y restablecer datos en la BD.
2. INICIALIZAR LA BD (EXTERNAL DATA SOURCE)

Para inicializar la BD vamos a utilizar DBUnit, esta herramienta nos permite insertar datos que se encuentran en un archivo externo XML. Para esto realizar los siguientes pasos:

1. CREAR EL ARCHIVO XML CON LOS DATOS

Abrir el archivo src/test/resources/testautomation/databasetesting/jdbc/employeeDataSet.xml, examinar su contenido y estructura.

¿Se percataron que tiene los mismos nombres que la tabla y columnas de la BD?

1. INSERTAR LOS DATOS DEL ARCHIVO EN LA BD

Como los datos van a ser inicializados cada vez que se ejecute una prueba, vamos a colocar la inicialización dentro de un método “SetUp”, creamos este método encima de nuestro método de prueba.

@Before

**public** **void** loadData() **throws** Exception{

}

Utilizando BDUnit cargarmos los datos en la BD. Par esto colocar el siguiente fragmento de código dentro del método “loadData()”.

InputStream file = **this**.getClass().getResourceAsStream("employeeDataSet.xml");

IDataSet dataSet = **new** FlatXmlDataSetBuilder().build(file);

IDatabaseConnection connection = **new** DatabaseConnection(dataSource.getConnection());  
DatabaseOperation.*INSERT*.execute(connection, dataSet);

PREGUNTA: ¿Se comprende qué es lo que hace cada una de estas líneas de código?

Existen 3 maneras básicas de insertar datos:

* DatabaseOperation.INSERT.execute(connection, dataSet);
* DatabaseOperation.CLEAN\_INSERT.execute(connection, dataSet);
* new InsertIdentityOperation(DatabaseOperation.INSERT).execute(connection,dataSet);

PREGUNTAS:

* CLEAN\_INSERT elimina todos los datos de la tabla antes de insertar, ¿Por qué que en esta oportunidad solo utilizamos INSERT? ¿En qué circunstancias utilizaríamos CLEAN\_INSERT?
* En nuestra inicialización utilizamos InsertIdentityOperation(DatabaseOperation.INSERT), ¿Cuál es crees que sea la diferencia entre InsertIdentityOperation(DatabaseOperation.INSERT) y DatabaseOperation.INSERT? (La respuesta está relacionada con la columna ID de la tabla de BD).

1. REESTABLECER LA BD

Como los datos van a ser limpiados cada vez que se finalice una prueba, vamos a colocar la eliminación dentro de un método “TearDown”, creamos este método encima de nuestro método de prueba.

@After

**public** **void** cleanData() **throws** Exception{

}

Utilizando BDUnit cargar los datos en la BD. Par esto colocar el siguiente fragmento de código dentro del método “cleanData()”.

InputStream file = **this**.getClass().getResourceAsStream("employeeDataSet.xml");

IDataSet dataSet = **new** FlatXmlDataSetBuilder().build(file);

IDatabaseConnection connection = **new** DatabaseConnection(dataSource.getConnection());

DatabaseOperation.*DELETE\_ALL*.execute(connection, dataSet);

PREGUNTA: ¿Se comprende qué es lo que hace cada una de estas líneas de código?

Existen 2 maneras básicas de eliminar datos:

* DatabaseOperation.DELETE\_ALL.execute(connection, dataSet);
* DatabaseOperation.DELETE.execute(connection, dataSet);

PREGUNTA:

* DELETE\_ALL elimina todos los datos de las tablas que se encuentra definidos en el “dataset”, ¿Cuál crees que sea la diferencia entre DELETE\_ALL y DELETE?

RECOMENDACIONES PARA ORACLE

<http://3rdstage.blogspot.com/2008/09/dbunit-with-oracle.html>