P07B- Pruebas de integración

Integración con una base de datos

El objetivo de esta práctica es automatizar pruebas de integración con una base de datos, implementando los drivers correspondientes. Para controlar el estado de la base de datos durante los tests utilizaremos la librería DbUnit.

Recordamos que es importante, no solamente la implementación de los drivers, sino también todas las acciones conducentes a posibilitar la automatización de los tests (organización del código de pruebas, configuración de la herramienta de construcción del sistema y comando para ejecutar los tests de integración).

En este caso, utilizamos drivers con verificación basada en el estado, y ya no estamos interesados en controlar las dependencia externas (una de las cuales será la base de datos), sino que precisamente queremos utilizar el código real. Hay que tener claras las diferencias entre pruebas unitarias y de integración.

Bitbucket

El trabajo de esta sesión también debes subirlo a *Bitbucket*. Todo el trabajo de esta práctica deberá estar en el directorio **P07B-Integracion** dentro de tu espacio de trabajo, es decir, dentro de tu carpeta: ppss-Gx-apellido1-apellido2-2019.

Recuerda que si trabajas desde los ordenadores del laboratorio primero deberás configurar git y clonar tu repositorio de Bitbucket.

Base de datos MySQL

En la máquina virtual tenéis instalado un servidor de bases de datos MySQL, al que podéis acceder con el usuario "***root***" y password "***ppss***" a través del siguiente comando, desde el terminal:

```
> mysql –u root –p
```

Dicho servidor se pone en marcha automáticamente cuando arrancamos la máquina virtual.

Vamos a usar un ***driver jdbc*** para acceder a nuestra base de datos. Por lo tanto, desde el código, usaremos la siguiente cadena de conexión:

```
jdbc:mysql://localhost:3306/DBUNIT?useSSL=false
```

En donde *localhost* es la máquina donde se está ejecutando el servidor mysql. Dicho proceso se encuentra "escuchando" en el puerto *3306*, y queremos acceder al esquema *DBUNIT* (el cual crearemos automáticamente durante el proceso de construcción usando el plugin sql).

Ejecución de scripts sql con Maven: plugin sql-maven-plugin

Hemos visto en clase que se puede utilizar un plugin de maven (*sql-maven-plugin*) para ejecutar scripts sql sobre una base de datos mysql.

El plugin sql-maven-plugin puede tener configuradas diferentes *ejecuciones* (varias secciones <execution>, cada una con un valor de <id> diferente). Para ejecutarlas podemos hacerlo de diferentes formas, teniendo en cuenta que:

- Si hay varias <execution> asociadas a la misma fase y ejecutamos dicha fase maven, se ejecutarán todas ellas

- Si queremos ejecutar solamente una de las <executions>, podemos hacerlo con mvn sql:execute@execution-id, siendo execution-id el identificador de la etiqueta <id> de la <execution> correspondiente
- Si ejecutamos simplemente mvn sql:execute, sin indicar ninguna ejecución en concreto, por defecto se ejecutará aquella que esté identificada como <id>default-cli</id>. Si no hay ninguna <execution> con ese nombre, no se ejecutará nada.

Ejemplo: Dado el siguiente fragmento de código con las <executions> del plugin,

```
...
<executions>
  <execution>
     <id>create-customer-table</id>
     <phase>pre-integration-test</phase>
     <goals>
        <goal>execute</goal>
     </goals>
     <configuration>
        <srcFiles>
            <srcFile>src/test/resources/sql/script1.sql</srcFile>
        </srcFiles>
    </configuration>
  </execution>
  <execution>
    <id>create-customer-table2</id>
    <phase>pre-integration-test</phase>
    <goals>
      <goal>execute</goal>
    </goals>
    <configuration>
      <srcFiles>
        <srcFile>src/test/resources/sql/script2.sql</srcFile>
      </srcFiles>
    </configuration>
  </execution>
</executions>
...
```

- La ejecución ***mvn pre-integration-test*** ejecutará los scripts script1.sql y script1.sql
- La ejecución ***mvn sql:execute@create-customer-table2*** solamente ejecutará el script2.sql
- La ejecución ***mvn sql:execute*** no ejecutará nada

Ejercicios

En esta sesión trabajaremos con un proyecto IntelliJ **vacío**, e iremos añadiendo los módulos (proyectos Maven), en cada uno de los ejercicios.

Para **crear el proyecto IntelliJ**, simplemente tendremos que realizar lo siguiente:

- ***File→New Project***. A continuación elegimos “Empty Project” y pulsamos sobre Next,
- ***Project name*** : "*P07B-dbunit*". ***Project Location:*** "$HOME/ppss-Gx-.../*P07B-Integracion/P07B-dbunit*". Es decir, que tenemos que crear el proyecto dentro de nuestro directorio de trabajo, y del directorio *P07B-Integracion*.

De forma automática, IntelliJ nos da la posibilidad de añadir un nuevo módulo a nuestro proyecto (desde la ventana ***Project Structure***), antes de crear el proyecto. Cada ejercicio lo haremos en un módulo diferente. Recuerda que CADA módulo es un PROYECTO MAVEN.

➪ ➪ Ejercicio 1: proyecto dbunitexample

Añade un módulo a nuestro proyecto IntelliJ *P07B-dbunit* (***File→New Module)***:

- Seleccionamos ***Maven***, y nos aseguramos de elegir el ***JDK 1.8***
- ***GroupId***: "*ppss*"; ***ArtifactId:*** *"dbunitexample*".
- ***ModuleName***: "*dbunitexample*". ***Content Root:*** *"$HOME/ppss-Gx-.../P07B-Integracion/P07B-dbunit/dbunitexample*". ***Module file location***: *"$HOME/ppss-Gx-.../P07B-Integracion/P07B-dbunit/dbunitexample*".

Como siempre, necesitamos incluir las *<properties>* en el pom, que copiaremos de proyectos anteriores.

En el directorio Plantillas-P07B/ejercicio1 encontraréis los ficheros que vamos a usar en este ejercicio:

- la implementación de las clases sobre las que realizaremos las pruebas: *ppss.Customer* y *ppss.CustomerFactory*,
- la implementación de dos tests de integración (*CustomerFactoryIT.java*),
- ficheros xml con los datos iniciales de la BD (*customer-init.xml*), y resultado esperado de uno de los tests (*customer-expected.xml*),
- fichero con el script sql para restaurar el esquema y las tablas de la base de datos (*create-table-customer.sql*)

Se pide:

A) Organiza el código proporcionado en el proyecto Maven que has creado según hemos visto en clase, y modifica el pom.xml convenientemente para poder utilizar DbUnit y el plugin *sql-maven-plugin.* Añade, además, las siguientes dependencias, se trata de librerías de *logging.* Solamente las incluimos para que no nos aparezcan advertencias (*warnings)* al construir el proyecto con Maven.

```
<!-- Logging -->
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
  <version>1.2.17</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

Nota: Asegúrate de que la cadena de conexión del ***plugin sql*** NO contenga el esquema de nuestra base de datos DBUNIT, ya que precisamente el plugin sql es el que se encargará de crearlo cada vez que construyamos el proyecto. Por lo tanto, la cadena de conexión que usa el plugin sql debe ser: *jdbc:mysql://localhost:3306/?useSSL=false*. Cuando uses la cadena de conexión en el código del proyecto ésta SÍ deberá incluir el esquema DBUNIT.

B) En el código de pruebas, explica qué representan las variables *_customerFactory* y *databaseTester* indicando para qué y dónde se utilizan.

C) En el código de pruebas, explica la diferencia entre un *dataset* y una *table*. Identifica dónde y para qué se utilizan en el código proporcionado.

D) Utiliza Maven para ejecutar los tests, para ello debes asegurarte previamente de que tanto la cadena de conexión, como el login y el password sean correctos, tanto en el *pom.xml*, como en el código de pruebas y en el código fuente. Observa la consola que muestra la salida de la construcción del proyecto y comprueba que se ejecutan todas las goals que hemos indicado en clase. Fíjate bien en los artefactos (ficheros) generados (**Nota**: las librerías descargadas por maven en nuestro repositorio local son necesarios para construir el proyecto, pero NO son artefactos generados por Maven durante la construcción del proyecto).

E) Una vez ejecutados los tests, cambia los datos de prueba del método test_insert() y realiza las modificaciones necesarias adicionales para que los tests sigan en "verde". Puedes probar a realizar cambios en los datos de entrada del otro test para familiarizarte con el código.

F) Implementa dos tests adicionales. Un test para actualizar los datos de un cliente (*testUpdate()*) y otro para recuperar los datos de un cliente (*testRetrieve()*). Utiliza los siguientes casos de prueba:

Tabla cliente inicial	Datos del cliente a modificar	Resultado esperado
customer._id =1 customer._firstname="John" customer._lastName= "Smith" customer._street = "1 Main Street" customer._city = "Anycity"	customer._id =1 customer._firstname= "John" customer._lastName= "Smith" customer._street = "Other Street" customer._city = "NewCity"	customer._id =1 customer._firstname= "John" customer._lastName= "Smith" customer._street = "Other Street" customer._city = "NewCity"

Tabla cliente inicial	id del cliente a recuperar	Resultado esperado
customer._id =1 customer._firstname="John" customer._lastName= "Smith" customer._street = "1 Main Street" customer._city = "Anycity"	customer._id =1	customer._id =1 customer._firstname="John" customer._lastName= "Smith" customer._street = "1 Main Street" customer._city = "Anycity"

➪ ➪ Ejercicio 2: proyecto multimódulo `matriculacion`

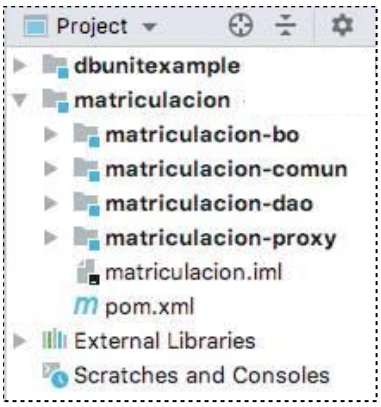
Vamos a añadir un segundo módulo a nuestro proyecto IntelliJ *P07B-dbunit*. Usaremos el proyecto multimódulo "*matriculacion*" de la práctica anterior.

Copia, desde un terminal, el proyecto maven multimódulo "matriculacion" de la práctica anterior (carpeta matriculación y todo su contenido) en el directorio ".../*P07B-dbunit*".

Ahora añadiremos el proyecto como un nuevo módulo al nuestro proyecto IntelliJ ***(File→New →Module from Existing Sources...)***:

- Seleccionamos la carpeta matriculación que acabamos de copiar.
- En la siguiente pantalla seleccionamos: ***Import module from external model →Maven***

Finalmente debemos ver nuestro proyecto matriculacion como un módulo en la vista Project de IntelliJ, tal y como mostramos en imagen de la derecha:



Contenido de nuestro proyecto IntelliJ P07B-dbunit

En el directorio Plantillas-P07B/ejercicio2 encontraréis los ficheros que vamos a usar en este ejercicio:

- fichero con el script sql para restaurar el esquema y las tablas de la base de datos (*matriculacion.sql*)
- fichero xml con un dataset ejemplo (*dataset.xml*) y fichero dtd con la gramática de los ficheros XML para nuestro esquema de base de datos (*matriculacion.dtd*)

Se pide:

A) Dado que hay una relación de herencia entre *matriculación* y sus módulos agregados, vamos a modificar el **pom de *matriculacion***, para añadir aquellos elementos comunes a los submódulos.

Concretamente, estos deben heredar lo siguiente: la librería para usar junit5 (*junit-jupiter-engine*), y los *plugins surefire* y *failsafe*. Indica para qué sirven exactamente cada una de estas dependencias.

B) Incluye en el **pom del módulo *matriculacion-dao*** las librerías dbunit, log4j, slf4j-simple, y mysql-connector-java (igual que en el ejercicio anterior). Debes añadir también el plugin sql-maven-plugin. El *script* sql será el fichero *matriculacion.sql* que encontrarás en el directorio de plantillas (deberás copiarlo en el directorio *src/test/resources/sql*).

C) En la clase ***FuenteDatosJDBC*** (en src/main/java del módulo matriculacion-dao) se configura la conexión con la BD. Debes cambiar el driver hsqldb por el driver jdbc (*com.mysql.jdbc.Driver*), así como los parámetros del método getConnection, es decir, la cadena de conexión, login y password (debes usar los valores que ya hemos usado para el ejercicio anterior).

D) Copia el fichero ***matriculacion.dtd*** del directorio de plantillas en src/test/resources. Como ya hemos indicado, contiene la gramática de los datasets de nuestro esquema de base de datos. El fichero ***dataset.xml*** tiene un ejemplo de cómo definir nuestros datasets en formato xml. Puedes copiar también dicho fichero en src/test/resources.

E) Queremos implementar tests de integración para los métodos AlumnoDAO.addAlumno() y AlumnoDAO.delAlumno(). Para ello vamos a necesitar crear los ficheros ***tabla2.xml***, ***tabla3.xml***, y ***tabla4.xml***, que contienen los ***datasets* iniciales** y ***datasets esperados*** necesarios para implementar los casos de prueba de la tabla 1 (dichos datasets se definen en las **tablas 2, 3 y 4**). Crea los ficheros xml correspondientes en la carpeta src/test/resources, con la opción New→File (desde el directorio src/test/resources), indicando el nombre del fichero: por ejemplo *tabla2.xml*. Puedes copiar el contenido de dataset.xml y editar los datos convenientemente.

Tabla 2. Base de datos de entrada. Contenido tabla ALUMNOS

NIF	Nombre	Dirección	E-mail	
11111111A	Alfonso Ramirez Ruiz	Rambla, 22	alfonso@ppss.ua.es	1982-02-22 00:00:00.0
22222222B	Laura Martinez Perez	Maisonnave, 5	laura@ppss.ua.es	1980-02-22 00:00:00.0

Tabla 3. Base de datos esperada como salida en testA1

NIF	Nombre	Dirección	E-mail	
11111111A	Alfonso Ramirez Ruiz	Rambla, 22	alfonso@ppss.ua.es	1982-02-22 00:00:00.0
22222222B	Laura Martinez Perez	Maisonnave, 5	laura@ppss.ua.es	1980-02-22 00:00:00.0
33333333C	Elena Aguirre Juarez			1985-02-22 00:00:00.0

Tabla 4. Base de datos esperada como salida en testB1

NIF	Nombre	Dirección	E-mail	
22222222B	Laura Martinez Perez	Maisonnave, 5	laura@ppss.ua.es	1980-02-22 00:00:00.0

F) **Implementa**, usando *DbUnit,* **los casos de prueba de la tabla 1** en una clase `AlumnoDAOIT`. Cada caso de prueba debe llevar el nombre indicado en la columna ID de la tabla 1. Ejecuta los *tests*. Hazlo desde el proyecto matriculacion-dao (comando mvn verify).

Para hacer las pruebas, tendremos que utilizar un objeto que implemente la interfaz IAlumnoDAO. Así, por ejemplo para cada test Ax de la tabla 1, utilizaremos una sentencia del tipo:

```
new FactoriaDAO().getAlumnoDAO().addAlumno(alumno);
```

Para especificar la fecha, debes utilizar la clase Calendar. Por ejemplo, para indicar la fecha de nacimiento "1985-02-22 00:00:00.0" de un objeto alumno, puede hacerse de la siguiente forma:

```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 0);
cal.set(Calendar.MINUTE, 0);
cal.set(Calendar.SECOND, 0);
cal.set(Calendar.MILLISECOND, 0);
cal.set(Calendar.YEAR, 1985);
cal.set(Calendar.MONTH, 1); //Nota: en la clase Calendar, el primer mes es 0
cal.set(Calendar.DATE, 22);
alumno.setFechaNacimiento(cal.getTime());
```

Tabla 1. Casos de prueba para AlumnoDAO

ID	Método a probar	Entrada	Salida Esperada
testA1	void addAlumno (AlumnoTO p)	p.nif = "33333333C" p.nombre = "Elena Aguirre Juarez" p.fechaNac = 1985-02-22 00:00:00.0	Tabla 3
testA2	void addAlumno (AlumnoTO p)	p.nif = "11111111A" p.nombre = "Alfonso Ramirez Ruiz" p.fechaNac = 1982-02-22 00:00:00.0	DAOException
testA3	void addAlumno (AlumnoTO p)	p.nif = "44444444D" p.nombre = null p.fechaNac = 1982-02-22 00:00:00.0	DAOException
testA4	void addAlumno (AlumnoTO p)	p = null	DAOException
testA5	void addAlumno (AlumnoTO p)	p.nif = null p.nombre = "Pedro Garcia Lopez" p.fechaNac = 1982-02-22 00:00:00.0	DAOException
testB1	void delAlumno (String nif)	nif = "11111111A"	Tabla 4
testB2	void delAlumno (String nif)	nif = "33333333C"	DAOException

G) Ejecuta los tests anteriores teniendo en cuenta que si los algún test falla debemos interrumpir la construcción del proyecto. El test "testA4" falla. Repara el error en el fuente, de forma que devuelva el resultado correcto.

H) Añade 3 **tests unitarios** en una clase AlumnoDAOTest. NO es necesario que implementes los dobles ni los tests, estos últimos pueden estar "vacíos". Ejecuta sólo los tests unitarios usando el comando mvn test y comprueba que efectivamente sólo se lanzan los tests unitarios

I) Añade también tests unitarios y de integración en los proyectos *matriculacion-bo* y *matriculacion-proxy*. Luego ejecuta mvn verify desde el proyecto padre. Anota la secuencia en la que se ejecutan los diferentes tipos de tests de todos los módulos. ¿Qué estrategia de integración estás siguiendo?

J) Finalmente añade un nuevo **elemento de configuración**, dentro del proyecto *matriculacion* con el comando maven "*mvn -Dgroups="Integracion-fase1"*. Dado que los plugins *failsafe* y *surefire* comparten la variable *groups*, etiqueta tanto los tests unitarios como los de integración del proyecto *matriculacion-dao* con la etiqueta "*Integracion-fase1"*. Ejecútala y comprueba que ahora sólo se ejecutan los tests unitarios y de integración del proyecto *matriculacion-dao*.