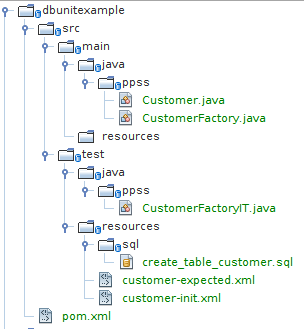
*sql-maven-plugin:* plugin de maven → ejecutar scripts sql sobre una base de datos mysql

* puede tener configuradas diferentes ejecuciones:
  + definirlas:
    - varias secciones <execution>, con un <id> diferente
  + ejecutarlas:
    - si hay varias <execution> asociadas a la misma fase (<phase>)
      * ejecutar fase de maven (se ejecutarán todas)
      * p.e.: mvn pre-integration-test
    - ejecutar solamente una de las <executions>
      * mvn sql:execute@execution-id
    - si hay una <execution> identificada como <id>default-cli</id>
      * mvn sql:execute (si no hay ninguna no se ejecuta nada)

1. dbunitexample
   1. init
      1. organizar código
         1. clases a probar → ppss.Customer, ppss.CustomerFactory
            1. src/main/java
         2. tests de integración → CustomerFactoryIT.class
            1. src/test/java/ppss
         3. ficheros.xml → customer-init.xml (datos iniciales de la DB), customer-expected.xml (resultado esperado de uno de los tests)
            1. src/test/resources
         4. fichero con el script sql para restaurar el esquema de datos → create-table-customer.sql
            1. src/test/resources/sql



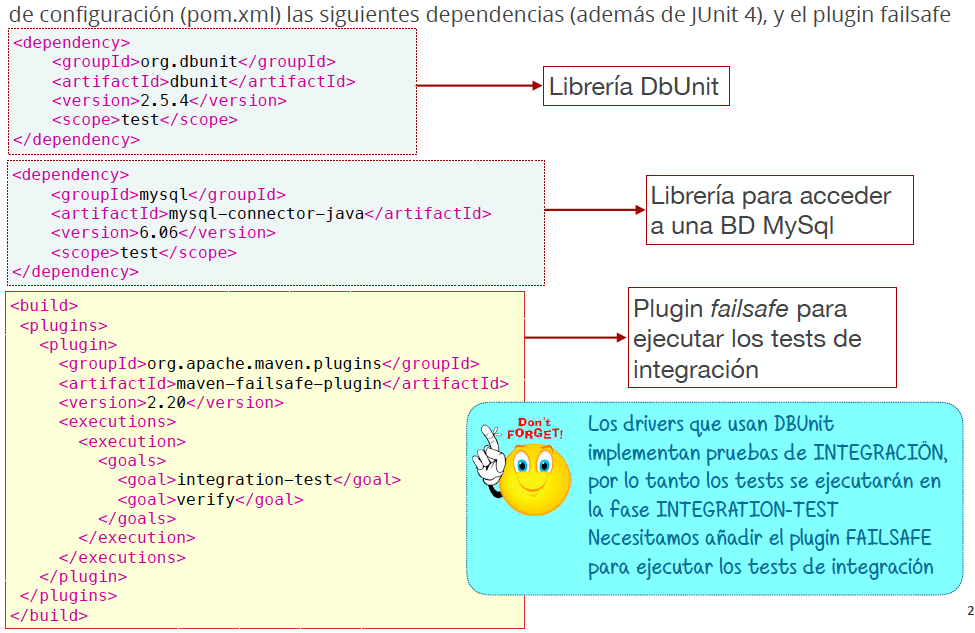
* + 1. modificar pom.xml ← usar DBUnit y plugin sql-maven-plugin
       1. DBUnit
          1. dependencies

dbunit

mysql-connector-java

junit 4

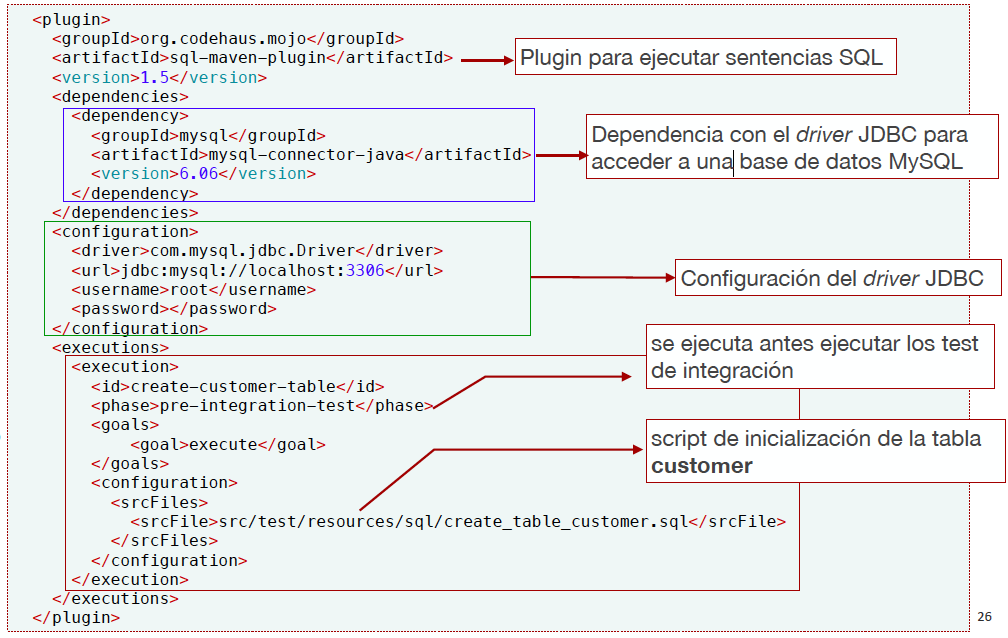
failsafe



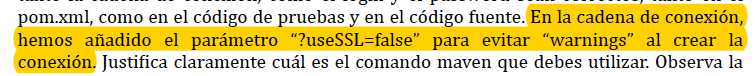
* + - 1. sql-maven-plugin
         1. plugin

dependencies

mysql-connector-java



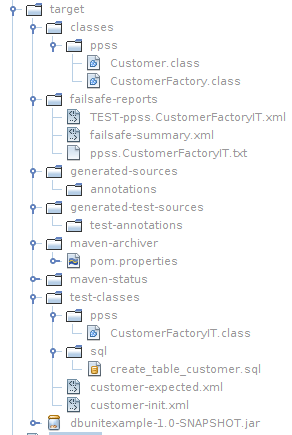
* + 1. añadir dependencias ← evitar warnings al construir el proyecto
       1. librerías de logging
          1. log4j
          2. slf4j-simple
  1. código de pruebas: variables
     1. \_customerFactory
        1. instancia que contiene nuestro SUT
     2. databaseTester
        1. instancia de IDatabaseTester para acceder a la BD
  2. código de pruebas: diferencia
     1. dataset
        1. colección de tablas que contienen los datos con los que vamos a interactuar con la BD
     2. table
        1. colección de datos tabulares
        2. se utiliza en aserciones, para comparar tablas de bd reales con esperadas
  3. ejecutar tests con maven



-- extra --

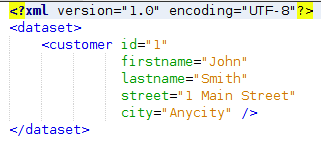
*[ Para la versión 6.0.6 de sql-maven-plugin, es necesario editar el archivo /etc/mysql/my.cnf añadiendo: timezone='UTC'. Sino, obtendremos un error al ejecutar el build del proyecto, pues en las versiones 6.\* está presente ese bug (en la versión del año pasado de 5.1.38, no está presente) ]*

* + 1. comando usado → mvn integration-test
       1. goal failsafe:integration-test →ejecutar fase integration-test (goal asociada por defecto)
    2. goals ejecutadas + orden
       1. resources:resources
       2. compiler:compile
       3. resources:testResources
       4. compiler:testCompile
       5. surefire:test
       6. jar:jar
       7. sql:execute
       8. failsafe:integration-test
    3. artefactos generados
       1. target/classes
          1. ppss → class de fuentes
       2. target/failsafe-reports
       3. target/test-classes
          1. ppss → class de test
          2. ← default → ficheros.xml
          3. sql → scripts sql



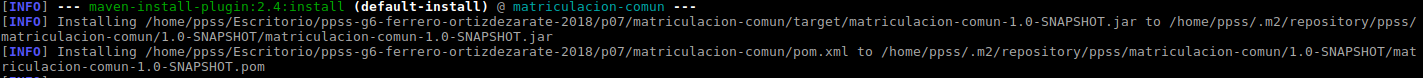
* 1. hacer cambios para que los tests sigan en "verde"
     1. cambia los datos de prueba del método test\_insert()



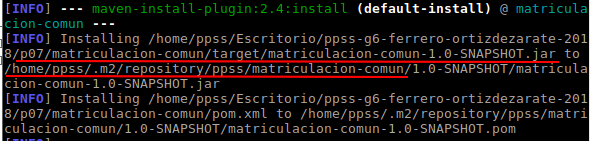


* 1. implementar 2 tests adicionales
     1. testUpdate(): actualizar los datos de un cliente
     2. testRetrieve(): recuperar los datos de un cliente

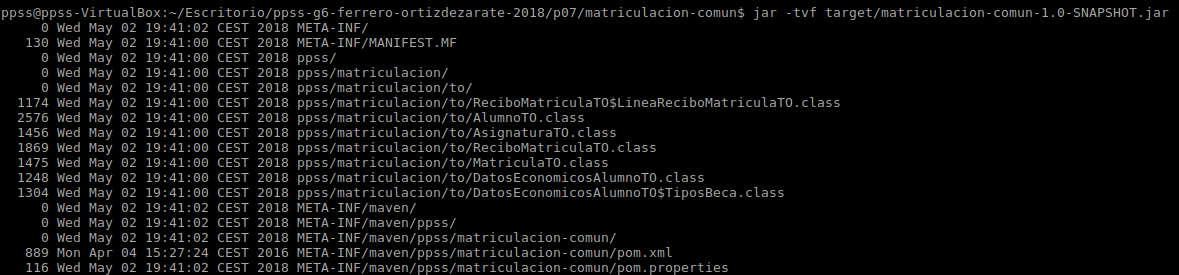
1. matriculacion-dao + matriculacion-comun
   1. clases definidas en matriculacion-comun *de las que depende matriculacion-dao*
      1. ¿Puedes indicar qué tienen en común todas las clases definidas en dicho proyecto?
         1. Todas las clases definidas acaban en TO, es decir, son TransferObjects. Un TransferObject se utiliza para el "transporte" de datos: es el objeto que representa los valores obtenidos en la BD.
      2. ¿Es necesario hacer pruebas unitarias sobre dicho proyecto?
         1. No, puesto que no tienen ningún tipo de lógica de negocio. Sólo son clases que se encargan de encapsular los datos del modelo, es decir, solo tiene getters y setters.
   2. matriculacion-comun
      1. mvn package → genera .jar
      2. mvn install → se copia el artefacto anterior en el repositorio local



*#zoom*



* + 1. jar -tvf archivo.jar *(o desde la ventana Files de Netbeans)* → ver contenido del archivo empaquetado desde terminal



* + 1. ¿por qué la fase package se realiza después de ejecutar los tests unitarios y antes de ejecutar los tests de integración?
       1. xq...
  1. matriculación-dao - dependencias
     1. junit:junit
        1. usada para realizar tests unitarios
     2. ${project.groupId}:matriculacion-comun
        1. proyecto matriculacion-comun usado para obtener los TransferObjects (AlumnoTO, etc.)
     3. mysql:mysql-connector-java
        1. librería para acceder a una BD MySQL
  2. pom.xml → plugin sql-maven-plugin
     1. ejecuciones
        1. create-db
        2. create-data
     2. actions
        1. Restaurar BD
           1. comando: mvn sql:execute@create-db
           2. goal: sql:execute@create-db
        2. Inicializar BD
           1. comando: mvn sql:execute@create-data
           2. goal: sql:execute@create-data
  3. configurar cadena de conexión en matriculacion-dao/src/main/java/FuenteDatosDB.java
  4. pom.xml → configurar para usar DBUnit
     1. dependencias
        1. dbunit
        2. mysql-connector-java
  5. exportar estado actual de la tabla alumnos
  6. crear ficheros.xml
  7. implementar casos de prueba tabla1
  8. añadir 3 tests unitarios → AlumnoDAOTest + new action → Pruebas unitarias
     1. mvn test
  9. new action → Pruebas de integración Fase 1
     1. mvn verify