Práctica 1
Statement, PreparedStatement y Pool de conexiones

Objetivos
Los objetivos perseguidos por esta práctica son:
- Repasar conceptos ya conocidos de JDBC
- Comparativa Statement/PreparedStatement
- Conexiones a diversos SGBD: Oracle, HSQLdb.
- Uso de un Pool de conexiones. Comparativa de tiempos.

Preparación
- Descarga el material del Campus Virtual. Se proporciona
  - una base de datos HSQLDB
  - un script para crear una base de datos similar en Oracle
  - drivers JDBC para conectarse a ambos SGBD
  - un driver para manejar pools de conexión (c3p0)
- Crear un directorio workspace y arrancar el IDE Eclipse
- Arrancar el servidor HSQLdb (ejecutar data/startup).
- Conectarse al servidor de Oracle. Ejecutar el script para crear carworksop en Oracle.

Datos de conexión
- **Oracle**
  - URL = "jdbc:oracle:thin:@156.35.94.98:1521:desa19"
  - PUERTOS: 1521, 1526, 5504, 5850, 3389
  - USER = UO
  - PASS = password Oracle
- **HSQLDB**
  - URL = "jdbc:hsqldb:hsql://localhost"
  - USER = "sa"
  - PASS = ""

Ejercicios

1. Reconstruir la base de datos Oracle

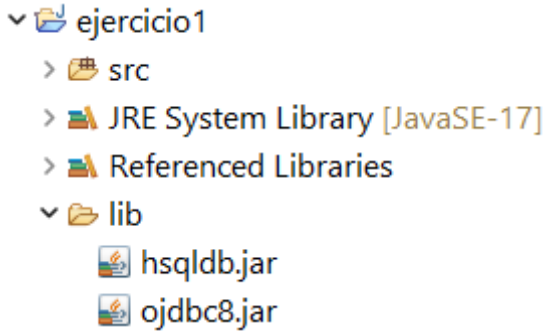
SELECT 'DROP TABLE '||table_name||' CASCADE CONSTRAINTS;' FROM user_tables
Ejecuta el script script_create_carworkshop_Oracle.sql

2. Instalar drivers

Crea un proyecto para solucionar el primer ejercicio. Repite luego con los demás ejercicios.
- Busca, entre el material descargado, los drivers que el cliente necesite (jar). Recuerda que vas a trabajar con dos bases de datos (Oracle y hsqldb). Necesitarás dos drivers.
- Hazlos accesibles a la aplicación cliente

- Añadir el fichero .jar a una carpeta lib del proyecto.

- Configura el build path de la aplicación.



3. Statement vs. PreparedStatement: comparación de tiempos

Escribir un programa en Java que nos permita comparar el rendimiento de **Statement** frente al de **PreparedStatement**.

Para ello, el programa debe conectarse a la base de datos y obtener unos datos utilizando `Statement`. A continuación, obtener los mismos datos utilizando `PreparedStatement`. Al finalizar ambas, liberar la conexión.

Para obtener los datos, en ambos casos se debe crear el objeto (Statement o PreparedStatement), escribir la sentencia y ejecutarla. No es necesario procesar el resultado de ningún modo. Al final, liberar la sentencia.

Para que la diferencia sea significativa la sentencia debe tener cierta complejidad y ejecutarse muchas veces, por ejemplo **1000**, cambiando cada vez.
Se sugiere **actualizar la tabla TINVOICES, incrementando amount en 1 cuando el id coincida con el valor de la variable de control del bucle.**
Debe medir el tiempo que se tarda en obtener los datos en cada caso y compararlo.

En el caso del Statement se construirá la sentencia SQL y se ejecutará (concatenando cadenas, por ejemplo), y en el caso del PreparedStatement se deben utilizar marcadores **"?".**

Para medir tiempos se puede utilizar `System.currentTimeMillis()` que nos devuelve la fecha/hora en número de milisegundos desde el 01/01/1970.

Primero ejecute el programa conectándose a HSQLDB. Luego, ejecute nuevamente conectándose a Oracle esta vez.

Como esta segunda ejecución será un poco decepcionante, principalmente debido al retraso de la red, implemente una tercera versión que utiliza procesamiento por lotes[1].

En cada ejecución, compare ambos tiempos. ¿Cuál lleva más tiempo? ¿Es este el resultado esperado?
Se puede ver una discusión muy interesante al respecto de lo trabajado en esta práctica aquí.

1 https://www.oreilly.com/library/view/database-programming-with/1565926161/ch04s02.html

4. Comprobación de costes de apertura y cierre de conexiones

Una conexión a una base de datos es un objeto grande y complejo que maneja todas las comunicaciones entre una aplicación y la base de datos subyacente.
Como tal, las conexiones de bases de datos requieren mucho tiempo para establecerse y consumen una cantidad significativa de memoria, tanto en la aplicación cliente como en el servidor de la base de datos.
En este ejercicio, deberá escribir un programa para tener una mejor idea del coste de conectarse a una base de datos.
Para ello procederemos de dos formas diferentes, midiendo tiempos y comparándolos:

1. Se realizará un bucle de 100 repeticiones y dentro del bucle se abrirá una conexión, se realizará una consulta y se cerrará la conexión.
   Repetir 100 veces
   Abrir conexión
   Hacer algo (ej: contar los vehículos de la tabla `TVEHICLES`)
   Cerrar conexión
2. Similar al anterior, pero la apertura y cierre de la conexión se realizará fuera del bucle.
   Abrir conexión
   Repetir 100 veces
   Hacer algo (lo mismo que en el caso anterior)
   Cerrar conexión

5. Pool de Conexiones

Para esta práctica utilizaremos el pool de conexiones c3p0 cuyos drivers están en la carpeta lib (descargados de http://www.mchange.com/projects/c3p0/).
La práctica consiste en crear un pool de conexiones con las siguientes características:

- Máximo de conexiones (*maxPoolSize*): 30
- Mínimo de conexiones (*minPoolSize*): 3
- Tamaño inicial del pool (*initialPoolSize*): 3

Para comprobar el funcionamiento correcto haremos como en el ejercicio anterior en el que medimos el coste en tiempo de abrir y cerrar conexiones dentro de un bucle. Se debe realizar el mismo bucle pero comparando entre utilizar una conexión del pool o una conexión obtenida sin pool. (Ej hazAlgo: contar el número de vehículos de la tabla tvehicles)

```
for (int i=0; i<1000; i++){
     crearConexion();
     hazAlgo();
     cerrarConexion();
}
```

```
crearPoolConexiones();
for (int i=0; i<1000; i++){
     getConexionFromPool();
     hazAlgo();
     releaseConexion();
```

```
    }
```

A modo de ejemplo de la utilización de c3p0 se copia aquí un extracto de código disponible en la web de c3p0 (http://www.mchange.com/projects/c3p0/#using_combopooleddatasource). Con el propio driver viene código de ejemplo. El código siguiente es para iniciar un Pooled DataSource. Una vez creado se podrán obtener Connection.

```
DataSource ds_unpooled = DataSources.unpooledDataSource(URL, USER, PASS);

Map overrides = new HashMap();
overrides.put("minPoolSize", 3);
overrides.put("maxPoolSize", 50);
overrides.put("initialPoolSize", 3);
ds_pooled = DataSources.pooledDataSource(ds_unpooled, overrides );
//ya está inicializado
//Para obtener conexión
con = ds_pooled.getConnection();
```

Se debe probar tanto con Oracle como con HSQLDB.
A continuación, intenta implementar hazAlgo con hilos, de forma que se reciban 100 intentos de conexión simultáneos y no secuenciales.

Tomar los tiempos después de crear el pool y al finalizar la ejecución. Compararlos con los tiempos anteriores. Tratar de explicar la diferencia.

Si la tarea fuese suficientemente costosa, sería todavía mejor dividirla en subtareas (Threads), cada uno de los cuales ejecutaría parte de las mismas.