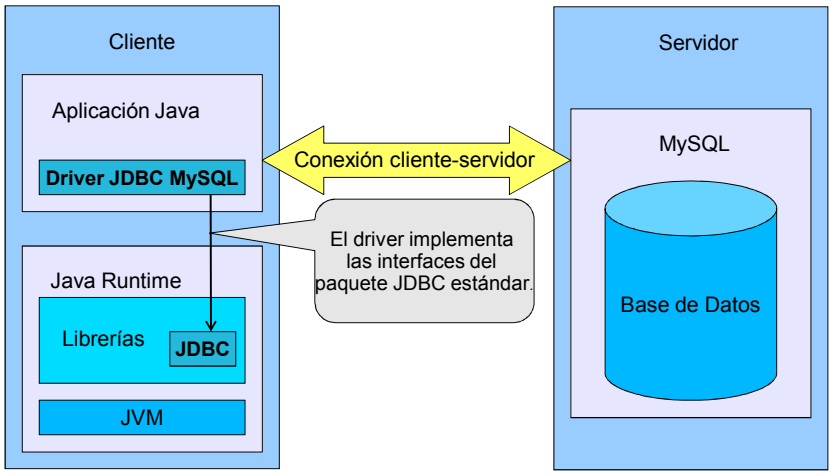
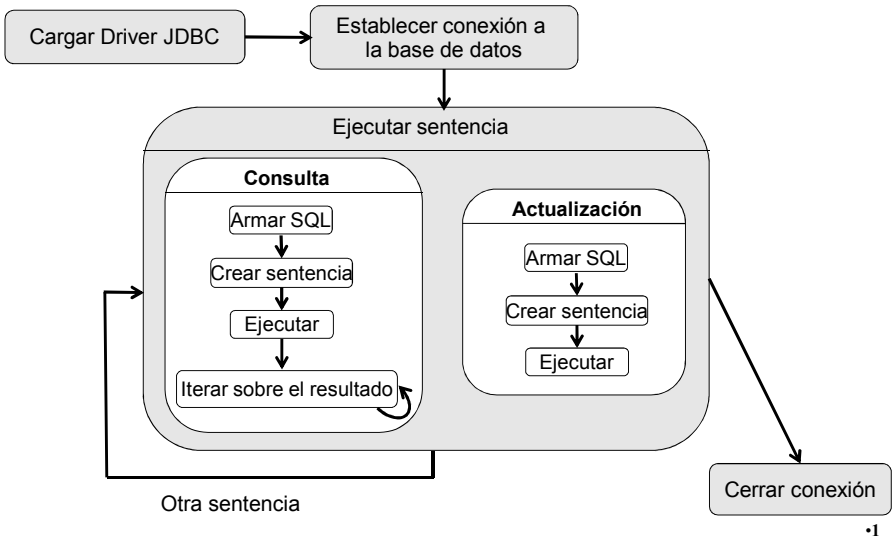
Java y MySQL

- JDBC: Java DataBase Connectivity



•0

Ciclo de ejecución



•1

Clase DriverManager (Paquete **java.sql**)

- **DriverManager**: clase estática (no requiere instanciación de objetos) que administra los drivers jdbc disponibles para iniciar conexiones.
  - **Connection** **getConnection**(**String** url): intenta iniciar una conexión a una base de datos según los parámetros especificados en el url. Generalmente el string de conexión tiene el siguiente formato:
    jdbc:<driver>:<propiedades de la conexión>
    Para MySQL:
    jdbc:mysql://<servidor>:<puerto>/<base_datos>?<parametros>
    Por ejemplo:
    jdbc:mysql://localhost:3306/batallas?user=barco&password=pwbarco
  - **Connection** **getConnection**(**String** url, **String** usuario, **String** clave): idem al anterior pero por compatibilidad y seguridad, el usuario y la clave de acceso, son parámetros individuales.
  - **setLoginTimeout**(**int** segundos): configura la cantidad de segundos de espera para intentar establecer la próxima conexión a una base de datos.

•2

Clase Connection (Paquete **java.sql**)

- **Connection**: interfaz para implementar una sesión cliente-servidor con una base de datos.
  - **Statement** **createStatement**(): crea una nueva sentencia para ejecutar código SQL en forma directa en el servidor a traves de la conexión.
  - **PreparedStatement** **prepareStatement**(**String** sql): crea una sentencia preparada con una estructura predeterminada dada por parámetros, para luego enviar los datos efectivos.
  - **boolean** **isValid**(**int** timeout): verifica que la conexión está abierta y disponible para ejecutar una operación. Es necesaria para determinar si la conexión de red aun permanece activa desde la ejecución del último SQL.
  - **close**(): cierra la conexión y libera los recursos utilizados.
  - **setAutoCommit**(**boolean** autoCommit), **commit**(), **rollback**(): utilizados para el manejo de transacciones en la conexión actual.

•3

Conexión JDBC a MySQL

```
// Se carga y registra el driver JDBC de MySQL (JDK 1.5 o anterior)
try
{
   Class.forName("com.mysql.jdbc.Driver").newInstance();
}
catch (Exception ex) {}

// Intento de conexión a una base de datos
String servidor = "localhost:3306";
String baseDatos = "batallas";
String usuario = "admin_batallas";
String clave = "pwbatallas";
String url = "jdbc:mysql://" + servidor + "/" + baseDatos;

java.sql.Connection cnx;
try
{
   cnx = java.sql.DriverManager.getConnection(url, usuario, clave);
}
catch (java.sql.SQLException ex) {}
```

•4

Clase Statement (Paquete **java.sql**)

- **Statement**: se utiliza para ejecutar una sentencia SQL en base a un string estático, ya sea un comando o una consulta.
  - **boolean execute**(**String** sql): ejecuta cualquier tipo de SQL. Si es una consulta se debe recuperar el resultado mediante el método **ResultSet getResultSet**();
  - **int executeUpdate**(**String** sql): sólo para comandos de actualización de datos (insert, delete, update) o configuración dinámica de la sesión (transacciones, concurrencia, etc).
  - **ResultSet executeQuery**(**String** sql): sólo para consultas que retornan un resultado en filas o registros.
  - **addBatch**(**String** sql), **int**[] **executeBatch**(), **clearBatch**(): permiten ejecutar una secuencia de comandos enviados en un lote.
  - **void setQueryTimeout**(**int** seconds): impone un límite de espera para la ejecución de la sentencia.
  - **close**(): cierra la sentencia liberando los recursos utilizados.

•5

Clase ResultSet (Paquete **java.sql**)

- **ResultSet**: contiene el conjunto resultado de una consulta SQL, estructurado en filas y columnas, con el comportamiento de un iterador.
  - **boolean next**(): avanza el índice interno del iterador a la proxima fila. Retorna false si no hay mas filas.
  - **String getString(int** columnIndex), **String getString(String** columnLabel): permiten recuperar los valores de las columnas como un String, según su posición en la fila (la primer columna es 1) o mediante su nombre respectivamente.
  - **int getInt**(...), **long getLong**(...), **float getFloat**(...), **double getDouble**(...), **boolean getBoolean**(...), **Date getDate**(...), **Timestamp getTimestamp**(...), **Object getObject**(...), etc...: una función para cada tipo de dato de las columnas.
  - **boolean wasNull**(): verifica si el último valor recuperado de una columna correpondía al valor NULL de SQL.
  - **boolean previous**(), **boolean first**(), **boolean last**(), **boolean absolute**(**int** row), **boolean relative**(**int** rows): funciones para navegar en el conjunto resultado.
  - **ResultSetMetaData getMetaData**(): para recuperar los meta-datos (cantidad de columnas, tipos, …) del conjunto resultado y de las columnas.

Consultas SQL

```
try
{
  // Se crea una sentencia jdbc para realizar la consulta
  java.sql.Statement stmt = cnx.createStatement();

  // Se prepara el string SQL de la consulta
  String sql = "SELECT nombre_barco, id, capitan FROM barcos";

  // Se ejecuta la sentencia y se recibe un resultado
  java.sql.ResultSet rs = stmt.executeQuery(sql);

  // Se recorre el resultado
  while (rs.next())
  {
    String nombreBarco = rs.getString("nombre_barco");
    int id = rs.getInt("id");
    String capitan = rs.getString("capitan");
  }
  rs.close();
  stmt.close();
}
catch (java.sql.SQLException ex) {}
```

Actualización de datos

- Comandos o sentencias de manipulación de datos (insert, update, delete) que no retornan un resultado.

```
try
{
  // Se crea una sentencia jdbc para realizar la consulta
  java.sql.Statement stmt = cnx.createStatement();
  // Se prepara el string SQL de la inserción
  String sql = "INSERT INTO barcos (nombre_barco, id, capitan) " +
           "VALUES ('Bismark', 22, 'Ernst Lindeman')";
  // Se ejecuta la inserción
  stmt.execute(sql);
        // Se retornan los recursos utilizados cerrando la sentencia
  stmt.close();
}
catch (java.sql.SQLException ex)
{
  System.out.println("Mensaje: " + ex.getMessage());
  System.out.println("Código: " + ex.getErrorCode());
  System.out.println("SQLState: " + ex.getSQLState());
}
```


Sentencias Preparadas

- Se utilizan cuando una misma sentencia – consulta o actualización – debe ejecutarse repetidamente con la misma estructura pero distintos valores.
- Más comunmente usado en inserciones de registros en masa.
- Acelera la ejecución al evitar la interpretación del SQL de manera individual. Una vez preparada la sentencia en el servidor, sólo se necesitan enviar los datos efectivos.
- El método puede interpretarse como una pre-compilación de las sentencias para un posterior uso repetitivo. No sólo se ahorran ciclos de cpu en el servidor, sino también ancho de banda para la trasmisión de comandos y datos.

```
String sql = "INSERT INTO barcos (nombre_barco, id, capitan)   VALUES (?, ?, ?)";
// Se crea un sentencia preparada
java.sql.PreparedStatement stmt = cnx.prepareStatement(sql);
// Se ligan los parámetros efectivos
stmt.setString(1, "Bismark");
stmt.setInt(2, 22);
stmt.setString(3, "Ernst Lindeman");
// Se ejecuta la inserción
stmt.executeUpdate();
// se cierra la sentencia
stmt.close();
```

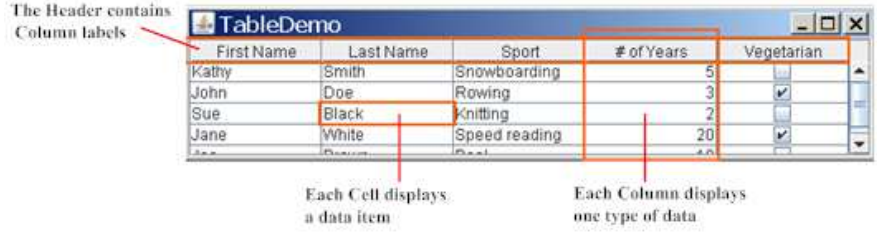
Clase ResultSetMetaData (Paquete **java.sql**)

- **ResultSetMetaData**: permite obtener los tipos y propiedades de las columnas de un conjunto resultado (ResultSet):
  - **int getColumnCount**(): cantidad de columnas en el resultado.
  - **String getColumnLabel**(**int** column), **String getColumnName**(**int** column): recuperar el nombre modificado o el nombre real de una columna respectivamente.
  - **String getColumnClassName**(**int** column): recuperar la clase de Java determinada como predefinida para el tipo de dato de una columna.
  - **int getColumnType**(**int** column), **String getColumnTypeName**(**int** column): recuperar el tipo de dato SQL estándar y SQL específico de una columna respectivamente.
  - **int isNullable**(**int** column), **boolean isAutoIncrement**(**int** column): propiedades particulares de una columna.
  - **boolean isSearchable**(**int** column): determina si es posible ejecutar un filtro en el WHERE de una consulta sobre la columna indicada.

•10

Clase JTable (paquete **javax.swing.table**)

- Permite mostrar graficamente tablas con datos, permitiendo opcionalmente al usuario editar los datos.



- Cada tabla JTable usa un objeto TableModel para manejar y almacenar los datos.
- Si no se especifica ningun modelo de tabla, JTable utiliza por defecto el modelo DefaultTableModel que almacena los datos como vector de vectores.
- Para crear un modelo de tabla se debe implementar la interface TableModel. Generalmente se implementa extendiendo la clase DefaultTableModel o AbstractTableModel.
- **Clase DBTable** (paquete **quick.dbtable**) Esta construido sobre JTable (comparte muchos métodos y propiedades) y provee funciones específicas para bases de datos.

(**Ver ejemplos de su uso en proyecto Batallas)**

•11

Manejo de Fechas

Para convertir **String** a **java.util.Date**:

```
Date fecha = (new SimpleDateFormat("dd/MM/yyyy")).parse("31/12/2009");
```

Para convertir **java.util.Date** a **String**:

```
String fechaStr = (new SimpleDateFormat("dd/MM/yyyy")).format(fecha);
```

Para convertir **java.util.Date** a **String** para SQL:

```
String fechaStrSQL  = (new SimpleDateFormat("yyyy-MM-dd")).format(fecha);
```

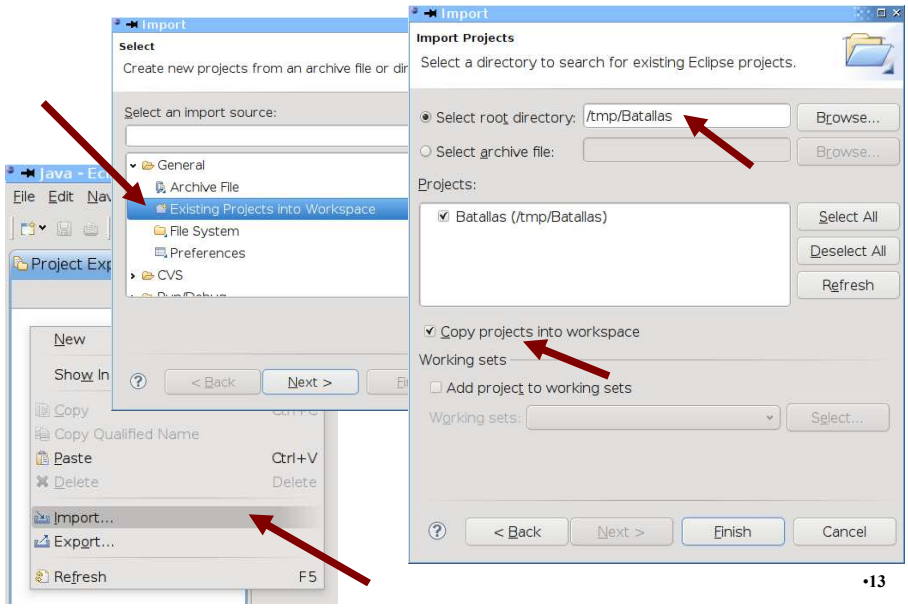
Para convertir **java.util.Date** a **java.sql.Date** de JDBC:

```
java.sql.Date retorno = java.sql.Date.valueOf(
                        (new SimpleDateFormat("yyyy-MM-dd")).format(fecha));
```

(Mas funciones ver fechas.java en proyecto Batallas)

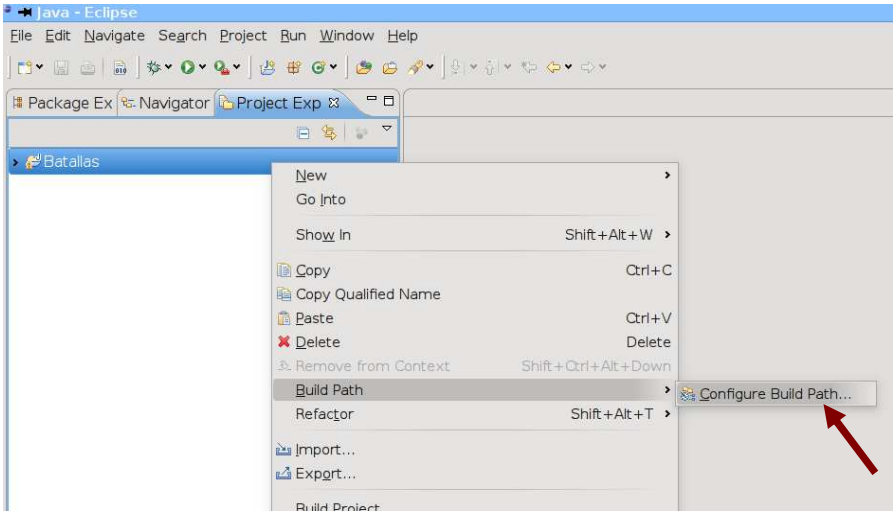
•12

Importar el proyecto ”Batallas” en Eclipse



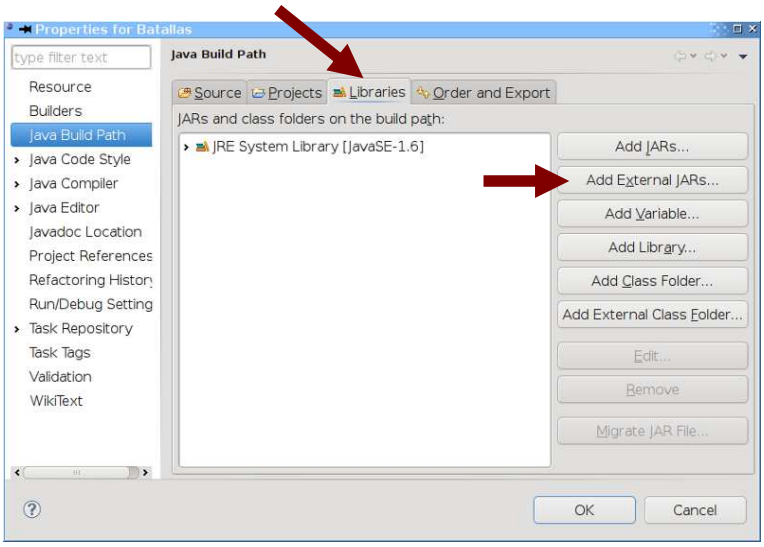
•13

Instalación del Driver JDBC de MySQL en Eclipse



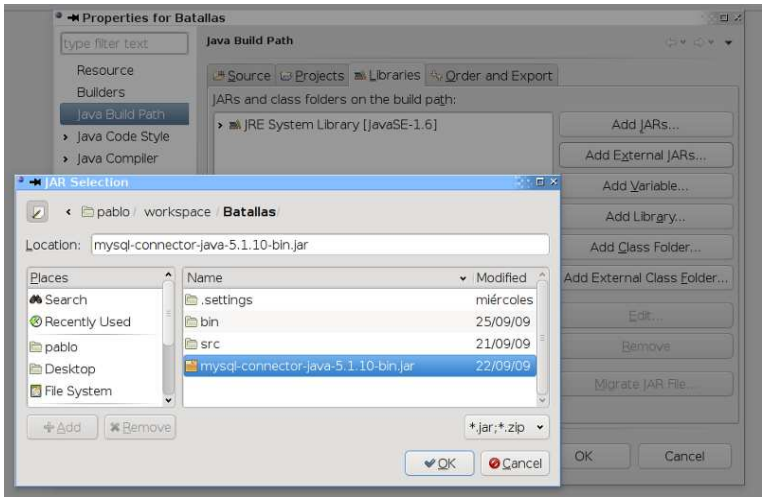
•14

Selección de la librería externa



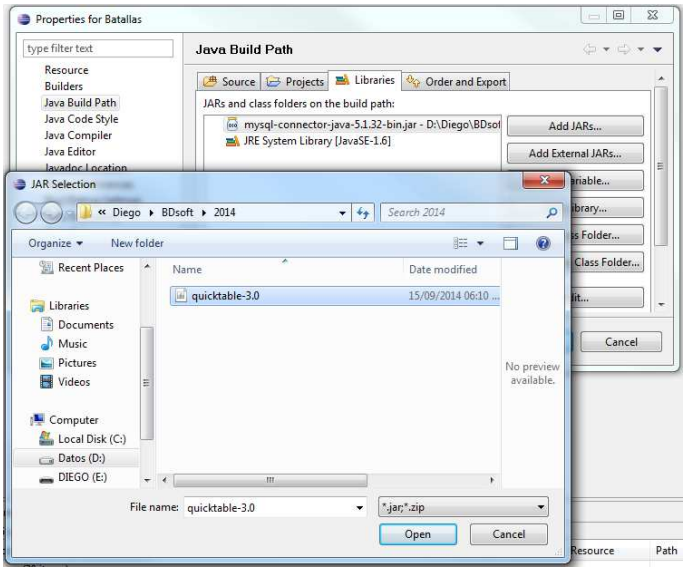
•15

Selección de la librería externa: JDBC MySQL connector



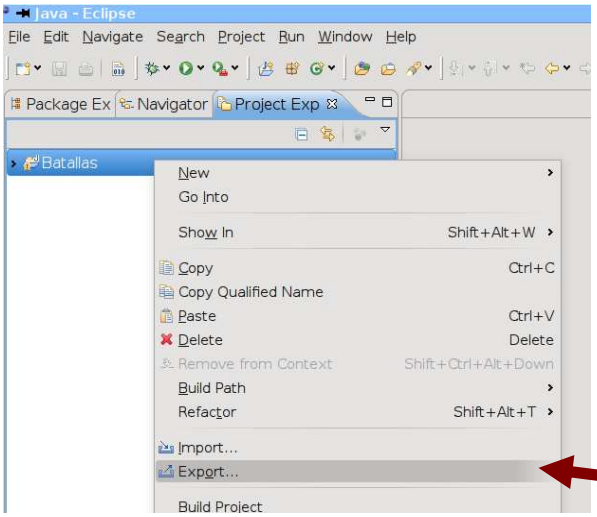
•16

Selección de la librería externa: quicktable (DBTable)



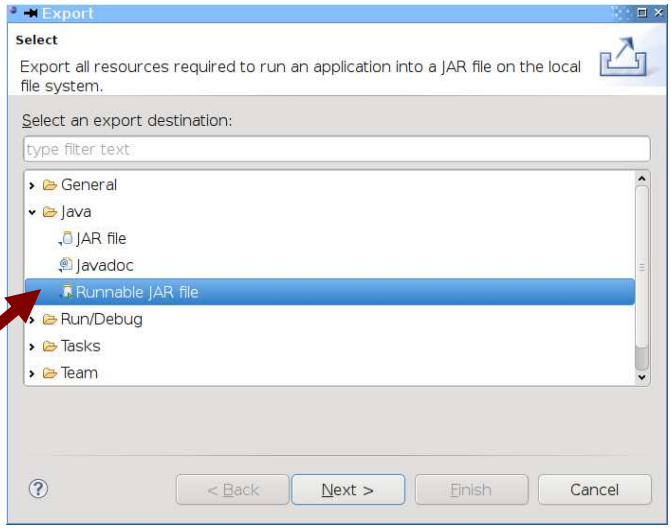
•17

Generar el archivo JAR de un proyecto en Eclipse



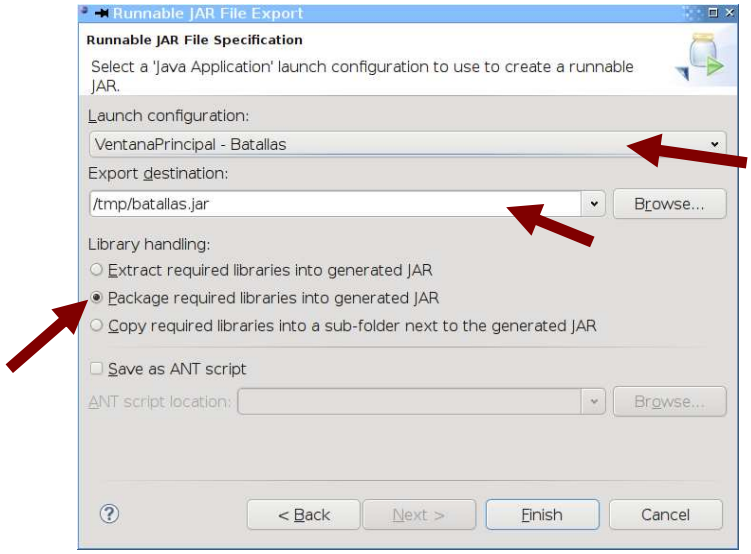
•18

Generar el archivo JAR de un proyecto en Eclipse (cont.)



•19

Generar el archivo JAR de un proyecto en Eclipse (cont.)



Referencias

Driver JDBC de MySQL: http://dev.mysql.com/downloads/connector/j/

Swing / JTable:
- http://java.sun.com/docs/books/tutorial/uiswing
- http://download.oracle.com/javase/tutorial/uiswing/components/table.html

Quick.DBTable:
- http://quicktablejava.appspot.com/home.html
- https://java.net/projects/quicktable/downloads

Eclipse: http://www.eclipse.org/downloads
Window builder: http://www.eclipse.org/windowbuilder/download.php
JDK 7 SE: http://java.sun.com/javase/downloads/index.jsp
Java API:
- http://java.sun.com/javase/7/docs
- http://java.sun.com/javase/7/docs/technotes/guides/jdbc