Bases de Datos y Sistemas de Información. Curso 2012/2013
Tratamiento de datos semiestructurados a través de una aplicación Java - Lab 3

El objetivo de esta práctica es el diseño de una base de datos XML y el uso de la API XMLDB (http://xmldb-org.sourceforge.net/) para acceder a **eXist** desde una aplicación Java que permita recuperar datos de la base de datos diseñada previamente.

- **Escenario de la práctica**.
  Considerar una base de datos que almacena datos sobre agencias inmobiliarias y anuncios inmobiliarios con las siguientes características:

  - Agencias inmobiliarias (especificación de datos de las diferentes agencias inmobiliarias).
    - Cada agencia inmobiliaria se identifica con un código único (obligatorio) y ha de registrarse su nombre (obligatorio), dirección, provincia (obligatorio), localidad y código postal.
    - Opcionalmente se podrán registrar uno o varios teléfonos de contacto, que podrán ser de varios tipos: móvil, fijo o fax.
    - La dirección de correo electrónico es obligatoria. También ha de registrarse la persona de contacto a la que va dirigido el correo electrónico.
    - También se desea registrar la página web de cada una de las agencias.
  - Inmuebles (características de cada uno de los anuncios a publicar).
    - Cada anuncio contiene información de la agencia a la que pertenece el anuncio (código de agencia) y un código o referencia que lo identifica (ambos obligatorios).
    - Los anuncios se pueden referir a distintos tipos de inmuebles: Pisos, Casas y chalets, Solares, terrenos y fincas, Locales, oficinas, naves y almacenes y Parkings y trasteros.
    - Cada anuncio se refiere a un tipo de operación, que puede ser venta o alquiler.
    - Los anuncios se pueden clasificar por categorías dependiendo del estado del inmueble (obligatorio). Para ello ha de registrarse el estado del inmueble, que puede ser: Nuevo, Segunda mano buen estado, Segunda mano reformado y Segunda mano necesita reformar.
    - La información básica y necesaria asociada a cada uno de los inmuebles es la siguiente: superficie útil del inmueble en m^2, precio, provincia donde se encuentra localizado el inmueble y localidad.
    - Los anuncios incluyen cierta información opcional relacionada con las características de inmueble. Esta información es proporcionada por la agencia. Las características pueden ser infinitas, así que se debe registrar el nombre de la característica y su valor. Por ejemplo, (número de dormitorios, n), número de baños, (piscina grande, si/no), (piscina de sal,si/no), (soleado,si/no), etc.
    - Finalmente, cabe la posibilidad de adjuntar un número variable de fotos al anuncio.

  Cada uno de los elementos de la base de datos (agencia y anuncio) requiere un identificador único y un conjunto de atributos de acuerdo a la descripción dada (por ejemplo, para cada anuncio, el tipo del inmueble, tipo de operación ...).

- **Se pide:**

  - Realizar el diseño de dos DTD's para representar los datos expuestos en el apartado anterior. Se pueden utilizar editores XML, como por ejemplo NetBeans, Essential XML Editor, ...
  - Generar dos fichero XML de prueba válidos con respecto a las DTD's diseñadas en el paso anterior.

- A partir de los documentos XML anteriores, se pide realizar una aplicación Java en Eclipse (ver anexo 1) que realice lo siguiente:
  - Debe tomar como argumentos de entrada el nombre de una colección, la ruta donde se encuentran almacenados los documentos XML anteriores, y un documento en el que aparecen descritas ciertas consultas xQuery (ver anexo 2).
  - Con los tres primeros argumentos, se deben cargar los documento XML de agencias y anuncios en la base de datos **eXist**.
  - Con la lectura del cuarto argumento, se debe ejecutar cada una de las consultas. Los resultados se deben almacenar en un fichero de texto denominado `resultado.txt` con el siguiente formato:

```
Consulta 1: Listar el nombre de las agencias.
--------
Input: -----
for $b in ...
return ....
Output:
       <<Resultado de la consulta>>
Consulta 2: Listar todos los anuncios de la primera agencia.
--------
Input: -----
for $b in ...
return ....
Output:
       <<Resultado de la consulta>>
...
...
```

- **Entrega (día 23 de abril de 2013 (23:00h))**.
  Generar un fichero comprimido con nombre `lab3.zip`. El fichero anterior debe contener la siguiente información:

  - Los dos documentos DTD, una breve descripción de las DTD´s y los dos documentos XML utilizados para llevar a cabo las consultas.
  - Carpeta con todos los archivos del código fuente (archivos.java).

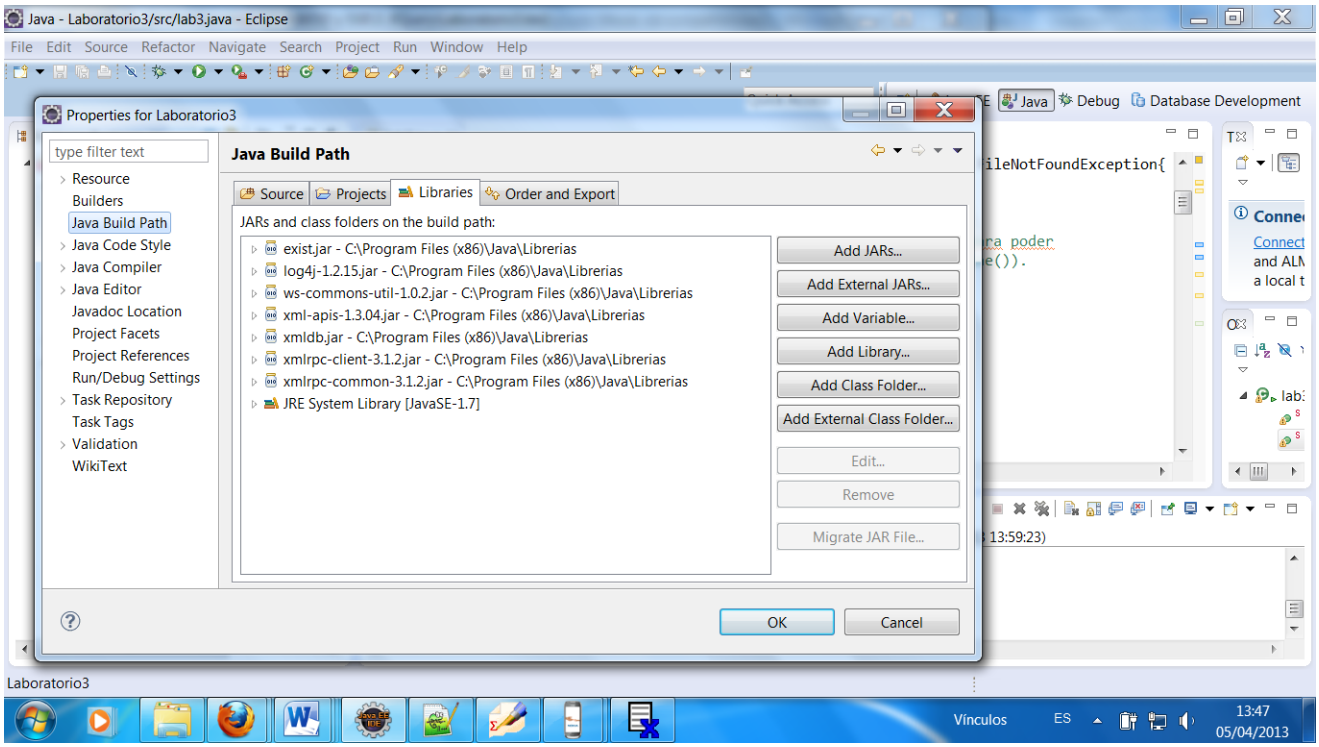

Figura 1: Incluir librerías

Anexo 1 : Guía de Implementación

1. Incluir las siguientes librerías en el proyecto (ver Figura 1):
   - exist.jar
   - xmldb.jar
   - xml-apis-1.3.04.jar
   - xmlrpc-common-3.1.2.jar
   - xmlrpc-client-3.1.2.jar
   - log4j-1.2.15.jar
   - ws-commons-util-1.0.2.jar
2. Los componentes básicos del API XML:DB son: *drivers*, *colecciones*, *recursos* y *servicios*. XMLResource es un recurso definido por el API y representa un documento XML o un fragmento de un documento XML. Los servicios se emplean para el tratamiento de las colecciones.
   A continuación se muestran algunos ejemplos que permiten implementar aplicaciones Java con el API XML:DB.

   Ejemplo 1:
   El siguiente código permite crear desde Java una colección en eXist y almacenar un documento XML en la base de datos. Observar que en el código dado habrá que particularizar el usuario, password y url de la base de datos, de acuerdo a la instalación que se tenga. En los laboratorios url (`xmldb:exist://localhost:8080/exist/xmlrpc/db` con el usuario y la password (admin/XMLFdILABs).

   ```
   import org.xmldb.api.*;
   ```

```
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import java.io.*;
import org.exist.util.*;
import org.exist.xmldb.*;

public class JavaApplication1  {
  public static void main(String args[]) throws Exception {

    // Se inicializa el driver
    String driver = "org.exist.xmldb.DatabaseImpl";
    //Cargar el driver
    Class cl = Class.forName(driver);//Cargar el driver
    //Instancia de la BD
    Database database = (Database) cl.newInstance();
    database.setProperty("create-database", "true");
    //Registrar la BD
    DatabaseManager.registerDatabase(database);
    //-------------------------------------------------------------------
    // Se intenta acceder a la colección especificada como primer argumento
    //-------------------------------------------------------------------
    String collection = args[0];
    String file = args[1];
    //Eliminar "/db" si es especificado en la entrada
    if (collection.startsWith("/db")) {
      collection = collection.substring(3);
    }
    String urlBD1 = "xmldb:exist://localhost:8080/exist/xmlrpc/db" + collection;
    Collection col = DatabaseManager.getCollection( urlBD1, "user", "password");
    XPathQueryService service =
       (XPathQueryService) col.getService("XPathQueryService", "1.0");
    // Si la colección especificada como primer argumento no existe,
    // entonces se crea
    if (col == null) {
    String urlBD2 = "xmldb:exist://localhost:8080/exist/xmlrpc/db";
    Collection root =  DatabaseManager.getCollection( urlBD2  ,
                               "user", "password");
    CollectionManagementService mgtService =
    (CollectionManagementService) root.getService("CollectionManagementService",
                               "1.0");
    col = mgtService.createCollection(collection);
    }
    //-------------------------------------------------
    // Se crea un recurso XML para almacenar el documento
    // -------------------------------------------------
    XMLResource document = (XMLResource) col.createResource(null, "XMLResource");
    File f = new File(file);
    if (!f.canRead()) {
    System.err.println("No es posible leer el archivo " + file);
    }
```

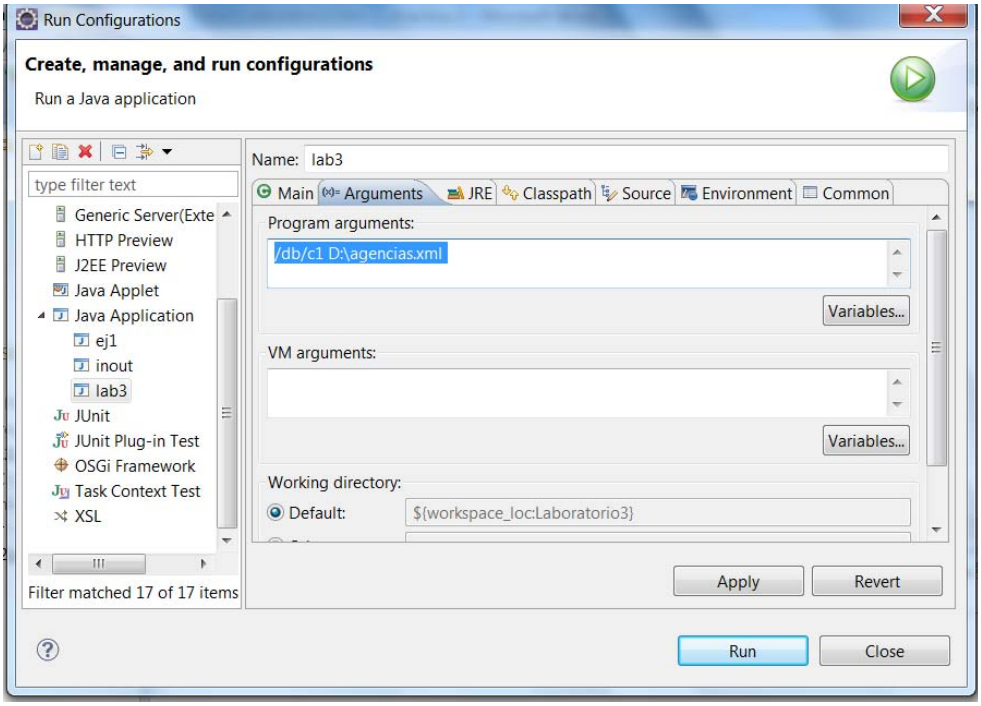

Figura 2: Parámetros de ejecución

```
        System.out.println(f.toString());
        document.setContent(f);
        System.out.println("Almacenando el archivo " + document.getId() + "...");
        col.storeResource(document);
        System.out.println("Almacenado correctamente");
        }
    }
```

Además hay que configurar la aplicación para que tome los dos argumentos de entrada. Opción `run configuration` (ver Figura 2).

Ejemplo 2:
El siguiente código permite realizar consultas XQuery sobre documentos XML almacenados en la base de datos **eXist**. Observar que en el código dado habrá que particularizar el usuario, password y url de la base de datos, de acuerdo a la instalación que se tenga. En los laboratorios url (`xmldb:exist://localhost:8080/exist/xmlrpc/db` con el usuario y la password (admin/XMLFdILABs).

```
import org.xmldb.api.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import java.io.*;
import org.exist.util.*;
import org.exist.xmldb.*;

public class JavaApplication1  {
  public static void main(String args[]) throws Exception {
```

```
    // Se inicializa el driver
    String driver = "org.exist.xmldb.DatabaseImpl";
    //Cargar el driver
    Class cl = Class.forName(driver);//Cargar el driver
    //Instancia de la BD
    Database database = (Database) cl.newInstance();
    database.setProperty("create-database", "true");
    //Registrar la BD
    DatabaseManager.registerDatabase(database);
    //---------------------------------------------------------------------
    // Accedemos a la colección y realizamos una consulta
    //---------------------------------------------------------------------
    String collection = args[0];
    String file = args[1];
    //Eliminar "/db" si es especificado en la entrada
    if (collection.startsWith("/db")) {
      collection = collection.substring(3);
    }
    String urlBD1 = "xmldb:exist://localhost:8080/exist/xmlrpc/db" + collection;
    Collection col = DatabaseManager.getCollection( urlBD1, "user", "password");
    XPathQueryService service =  (XPathQueryService)
                                 col.getService("XPathQueryService", "1.0");
    service.setProperty("pretty", "true");
        service.setProperty("encoding", "ISO-8859-1");

    //-------------------------------------------------
    // Consulta a lanzar (el documento debe existir ya en la base de datos)
    // -------------------------------------------------
        ResourceSet result =
            service.query("for $b in doc('agenda.xml')//persona return $b");
        ResourceIterator i = result.getIterator();
        //Se procesa el resultado.
        while (i.hasMoreResources()) {
           Resource r = i.nextResource();
           System.out.println((String) r.getContent());
        }
    }
}
```

Ejemplo 3: Lectura en Java

```
import java.io.*;

class LeeFichero {
   public static void main(String [] arg) {
      File archivo = null;
      FileReader fr = null;
```

```
      BufferedReader br = null;

      try {
      // Apertura del fichero y creacion de BufferedReader para poder
         // hacer una lectura comoda (disponer del metodo readLine()).
         archivo = new File ("C:\\archivo.txt");
         fr = new FileReader (archivo);
         br = new BufferedReader(fr);

         // Lectura del fichero
         String linea;
         while((linea=br.readLine())!=null)
            System.out.println(linea);
      }
      catch(Exception e){
         e.printStackTrace();
      }finally{
         // En el finally cerramos el fichero, para asegurarnos
         // que se cierra tanto si todo va bien como si salta
         // una excepcion.
         try{
            if( null != fr ){
               fr.close();
            }
         }catch (Exception e2){
            e2.printStackTrace();
         }
      }
   }
}
```

Ejemplo 4: Escritura en Java

```
import java.io.*;

public class EscribeFichero
{
    public static void main(String[] args)
    {
        FileWriter fichero = null;
        PrintWriter pw = null;
        try
        {
            fichero = new FileWriter("c:/prueba.txt");
            pw = new PrintWriter(fichero);

            for (int i = 0; i < 10; i++)
                pw.println("Linea " + i);

        } catch (Exception e) {
```

```
            e.printStackTrace();
         } finally {
            try {
            // Nuevamente aprovechamos el finally para
            // asegurarnos que se cierra el fichero.
            if (null != fichero)
               fichero.close();
            } catch (Exception e2) {
               e2.printStackTrace();
            }
         }
      }
   }
```

Anexo 2: Consultas

1) Listado de agencias en la provincia de Madrid. Por cada agencia, se pide el nombre y el número de anuncios gestionados por ella.
2) Listado de pisos en alquiler con mas de 2 dormitorios. Se pide la siguiente información:
   - Localización del piso, superficie, precio
   - Estado del piso
   - Si tiene calefacción o no
   - Nombre de la agencia que lo anuncia
3) Listado de chalets nuevos anunciados por agencias con código postal 28210 (Valdemorillo) que estén valorados por debajo de 500.000 euros. El listado ha de estar ordenado por identificador de agencia. Por cada agencia, ha de estar ordenado por el precio del inmueble en orden ascendente.