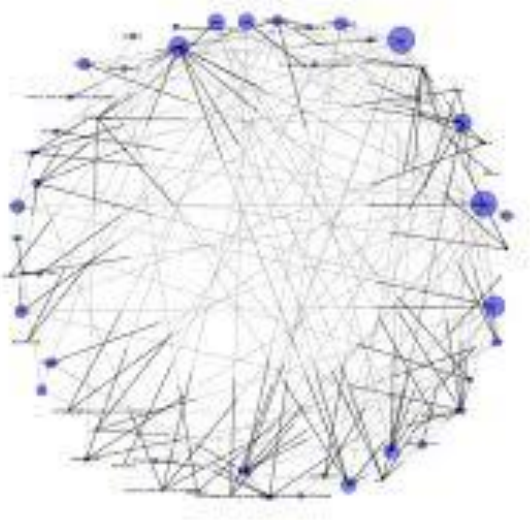
DAM
ACCESO A DATOS



UD1
MANEJO DE FICHEROS EN JAVA
6. XPATH SOBRE DOCUMENTO XML

Xpath (*XML Path Language*)

- **XPath** es lenguaje que permite buscar rápidamente datos y nodos dentro de un fichero XML.
- **XPath** permite seleccionar partes del documento XML mediante el uso de cadenas de consultas.
- El **XPath** ha sido estandarizado por el W3C, siendo uno de los elementos principales del estándar XSLT.
- Java posee, en su paquete javax.xml.xpath clases que implementan **XPath**. HTTP://WWW.W3.ORG/TR/XPATH
- Cuando trabajamos con **XPath** utilizaremos una cadena que contendrá la estructura de nodos que vamos a buscar separados por"/". La forma de trabajar es similar a trabajar con SQL.

Aplicaciones de XPath

```
<libreria>
 <libro>
  <autor>Gerald Durrel</autor>
  <titulo>Un zoo en la isla</titulo>
  <precio>50.0</precio>
 </libro>
 <libro>
  <autor>Miguel de Unamuno</autor>
  <titulo>Niebla</titulo>
  <precio>40.0</precio>
 </libro>
 <libro>
  <autor>Castelao</autor>
  <titulo>Cousas</titulo>
  <precio>23.0</precio>
 </libro>
 <libro>
  <autor>George Orwell</autor>
  <titulo>1984</titulo>
  <precio>20.0</precio>
 </libro>
</libreria>
```

- Con **XPath** podemos ejecutar consultas del tipo:
- ¿Qué libros escribió Castelao?
- ¿Qué libros tienen un precio menos a 30€?
- ¿Qué libros tiene la palabra “zoo” en su título?

Consultar el documento del aula virtual donde se indican los métodos de hacer consultas.

Sintaxis de expresiones XPath

- Es decir, **XPath** es una expresión abreviada que **representa a un nodo** en el árbol **DOM** teniendo en cuenta los nodos que hay que recorrer en el árbol **DOM** para llegar a él, a partir del raíz. Es una representación similar a las rutas de archivos en **Windows** o **GNU/Linux**.
- Evaluar una expresión **XPath** es buscar si hay nodos en el documento que se ajustan a la expresión.
- El resultado de la evaluación serán todos los nodos que se ajustan a la expresión.
- Las expresiones **XPath** se pueden dividir en pasos de búsqueda. Cada paso de búsqueda se puede a su vez dividir en tres partes:
  - **eje:** Selección de los nodos elemento o atributo basándose en sus nombre.
  - **predicado**: Restricción de la selección, filtra los nodos para que cumplan ciertos predicados.
  - **selección de nodos**: De los nodos seleccionados y filtrados selecciona los elementos, el texto que contiene o ambos.

Eje (*Axis*)

- El **eje** permite seleccionar un subconjunto de nodos del documento.
- Los nodos elemento se indican mediante el **nombre** del elemento. Los nodos atributos se indican mediante **@ y el nombre del atributo**.
- **/** → si está al principio de la expresión, indica el nodo raíz, si no, indica "hijo". Debe ir seguida del nombre de un elemento.
- **//** → Si aparece en el medio de una expresión, indica **descendientes** (hijos, hijos de hijos, etc.), si aparece al principio, busca los campos en todo el documento.
- **/..** → indica el elemento padre.
- **|** →permite indicar varios recorridos.

Predicado

- El predicado se escribe entre corchetes, a continuación del eje.
- Si el eje ha seleccionado unos nodos, el predicado permite restringir esa selección a los que cumplan determinadas condiciones.
- **[@atributo]**: selecciona los elementos que tienen el atributo.
- **[número]**: si hay varios resultados selecciona uno de ellos por número de orden; **last()** selecciona el último de ellos.

Condición

- Se puede seleccionar los nodos que cumplen con una condición.
- Se pueden definir condiciones sobre los valores de los elementos y atributos. En las condiciones se pueden utilizar los operadores siguientes:
  - **operadores lógicos: and, or, not()**
  - **operadores aritméticos: +, -, *, div, mod**
  - **operadores de comparación: =, !=, <, >, <=, >=**
- Las comparaciones se pueden hacer entre valores de nodos y atributos o con cadenas de texto o numéricas. Las cadenas de texto deben escribirse entre comillas simples o dobles. En el caso de las cadenas numéricas, las comillas son optativas.
- La condición puede utilizar el valor de un atributo (utilizando @) o el texto que contiene el elemento.
- Para saber más1
- Para saber más2

XPath

- **XPath** es lenguaje que permite buscar rápidamente datos y nodos dentro de un fichero XML.
- El **XPath** ha sido estandarizado por el W3C, siendo uno de los elementos principales del estándar XSLT
- Java posee, en su paquete javax.xml.xpath clases que implementan **XPath**. HTTP://WWW.W3.ORG/TR/XPATH
- Las clases necesarias para ejecutar consultas **XPath** son:
  - **XPathFactory**, esta clase contiene un método **compile()** que comprueba si la sintaxis de una consulta **XPath** es correcta y crear una expresión **XPath** (**XPathExpression**).
  - **XPathExpression**, esta clase contiene el método **evaluate()** para ejecutar la expresión **XPath**.
  - **DocumentBuilderFactory**, ya la hemos estudiado, permite crear el objeto **Document** que representa al fichero **XML** como un árbol **DOM**.

Java. Procesador XPath

Crear una instancia de una factory de constructores de documentos

Anteriormente hemos visto como crear un árbol DOM a partir de un fichero serializado de objetos, ahora crearemos el DOM directamente desde el fichero XML.

```
DocumentBuilderFactory factoryDocument = DocumentBuilderFactory.newInstance();
```

Crear un *creador de documento* XML

```
DocumentBuilder builderDocument = factoryDocument.newDocumentBuilder();
```

Construir un DOM a partir de un fichero XML bien formado (*parseamos el ficheroXML*) a partir de un objeto File, String o ImputStream (*fichero*)

```
Document documento = builderDocument.parse (fichero);
```

A partir de ese momento, todas las **acciones sobre el documento XML** se deberán realizar sobre la **variable *documento*** de la clase Document de Java, que permite la gestión de documentos XML basándose en el modelo DOM.

Java. Procesador XPath

Crear una instancia XPath para poder ejecutar consultas XPath

```
XPath elXPath = XPathFactory.newInstance().newXPath();
```

Crear un objeto que representa la expresión de la consulta de tipo XPathExpression

```
XPathExpression laXPathExpression = elXPath.compile("/consultaXPath");
```

Construir un DOM a partir de un fichero XML bien formado (*parseamos el ficheroXML*) a partir de un objeto File, String o ImputStream (*fichero*)

```
NodeList resultadoConsulta = (NodeList)
                    laXPathExpression.evaluate(documento, XPathConstants.NODESET);
```

A partir de ese momento, debemos recorrer el **NodeList** y, para cada nodo, hacer un tratamiento recursivo para mostrar por pantalla su contenido

Implementación del proyecto "Procesador de consultas Xpath"

- **Objetivos**:
  1. Que **muestre los ficheros XML** que hay en la carpeta raíz del proyecto.
  2. Que **permita seleccionar un fichero XML** de entre los mostrados.
  3. Que **muestre el contenido del fichero XML** seleccionado.
  4. Que **ejecute consultas XPath** sobre el fichero seleccionado y construir el DOM resultado de la consulta.
  5. Que **muestre el resultado** de la consulta **XPath** en formato **XML**.

1. Mostrar **los ficheros XML** de la carpeta raíz del proyecto

- **public static File[] listarFicherosByExtension(String filePath, String... extensiones)**
  1. El método tiene como parámetros
     a) un **String** con la ruta en la que buscar los ficheros XML (que será "." →*significa la carpeta actual, que en este caso será la que se corresponde con el proyecto).*
     b) un parámetro opciones **String… extensiones**, que permitirá realizar llamadas con un número variable de cadenas con distintas extensiones, por ejemplo,
        a) listarFicherosByExtension ("." , ".xml", ".dat")
        b) listarFicherosByExtension ("." , ".xml")

        El parámetro opcional se podrá tratar dentro del método como un array de **String** que tendrá el mismo número de elementos número de parámetros opcionales con que se realice la llamada.

  2. Crear un File[] con los ficheros con una determinada extensión de la carpeta especificada. Es el parámetro que se devuelve en el método listarFicherosByExtension
     I. Usaremos la interfaz **FilenameFilter** para crear un filtro de ficheros por extensión en una carpeta determinada:
        a) La interfaz **FilenameFilter** contiene un único método **public boolean accept (File dir, String name)** que se puede sobrescribir *para crear nuestro propio filtro.*
        b) La forma de crear una objeto de una Interfaz es crear una clase anónima que sobrescriba los métodos que necesitamos para adaptarlo a nuestros requisitos: *carpeta actual y extensión XML* .
     II. A partir de un objeto **File** ejecutaremos su método **listFiles (FileFilter filtro)** al que pasaremos como parámetro el objeto creado a partir de la interfaz **FilenameFilter** de la que hemos creado una clase anónima para sobrescribir el método **accept(File dir, String name)**.
        I. La ejecución de **File**[] listaOfFiles = new **File***(filePath).**listFiles**(elFilenameFilter)* siendo:
           I. *filePath, un String que contiene el path en el que queremos buscar los fichero (".")*
           II. *elFilenameFilter, un objeto de la interfaz FilenameFilter creado a partir de una clase anónima.*

```
public static File[] listarFicherosByExtension(String filePath, String... extensiones) {
    File[] listaOfFiles = new File[0];
    /* FilenameFilter es una interfaz que contiene únicamente el método  public boolean accept (File dir, String name) y
     devuelve true:entonces el archivo se añade a la lista false: entonces no se añade  el archivo a la lista El método accept
    se llama automáticamente al realizar file.listFiles(filtro)
     */
    FilenameFilter elFilenameFilter = new FilenameFilter() {
        /*Creamos una clase anónima y sobreescribimos el método accept cuya  signatura/firma está declarada en la interfaz
        FilenameFilter y que  debe devolver true para aquellos ficheros que cumplen la condición de  tener una extensión
        igual a las que se pasan por parámetro (String... extensiones)*/
        @Override
        public boolean accept(File dir, String nombre) {
            boolean filtrado = false;
            int indice = 0;
            while (indice < extensiones.length && !filtrado) {
                if (nombre.toLowerCase().endsWith(extensiones[indice])) {
                    filtrado = true;
                } else {
                    indice++;
                }
            }
            return filtrado;
        }
    };
    //Se crea un objeto File con la ruta que se ha pasado por parámetro
    File elFile = new File(filePath);
    if (elFile.exists()) {//Si existe esa ruta/pathfilePath);
        /*listFiles aplica el filtro que se pasa por parámetro y crea un File[] con todos los ficheros de la ruta indicada que
        pasan el filtro*/
        listaOfFiles = elFile.listFiles(elFilenameFilter);
    }
    return listaOfFiles;
}
```

Devuelve en array de **File** con aquellos ficheros que coincidan con las extensiones que se pasan por parámetro. Por ejemplo: listarByExtension ("C:\\ficheros\\", ".xml", ".java"); devolverá un File[] con los ficheros de C:\Ficheros que tengan extensión xml o java.

2. **Seleccionar un fichero XML** de entre los mostrados.

- **public static String seleccionarFicheroXML()**
  1. El método que filtra los ficheros con extensión **.xml** devuelve un array de **File** que tendrá un tamaño diferente según los ficheros que filtre, por eso nos interesa recoger ese **Array** y convertirlo en un **ArrayList** que permite manejar bien grupos variables de objetos.
     El método estático **Array.asList*(array de objetos)*** permite convertir un array de objetos (**File[]**) en un **List** de esos mismos objetos (**List<File>**)
  2. Cuando tenemos almacenados los ficheros filtrados en un **List<File>** podremos recorrer los objetos **File** y crear otro array con los nombres de los ficheros correspondientes.
  3. Utilizaremos la clase **Menu** de **LibreriaDam** para mostrar un menú con los nombres de los ficheros para que el usuario seleccione el fichero con el que quiere trabajar para realizar consultas **XPath**.

```
public static String seleccionarFicheroXML() {
    byte numFichero;
    boolean numFicheroCorrecto;
    /* List es una clase abstracta de la cual hereda ArrayList. Se tiene que crear una instancia de List pues
    queremos convertir un array de Files a ArrayList  y el metodo que nos permite hacer esta operación,
    Arrays.asList, devuelve una instancia List */
    List<File> losFicherosObjetos;

    ArrayList<String> losFicheros = new ArrayList<String>();

    /* LLamo al método listarFicherosByExtension para que me muestre los fichero con extensión xml que hay en la
    carpeta del proyecto actual. Este método devuelve un arraya de File y con Arrays.asList lo convertimos en una
    lista de File (List<File>)  */
    losFicherosObjetos = Arrays.asList(listarFicherosByExtension(".", "xml"));
    for (int i = 0; i < losFicherosObjetos.size(); i++) {
        losFicheros.add(losFicherosObjetos.get(i).getName());
    }
    /*La clase Menu permite imprimir el menú a partir de los datos de un ArrayList<String> y utilizar métodos para
     control de rango*/
    Menu miMenu = new Menu(losFicheros);
    System.out.println("Los fichero losFicheros con extensión xml que hay en la carpeta del proyecto son:");
    miMenu.printMenu();
    System.out.println("Introduzca el nº del fichero para realizar consultas XPath");

    do {
        /*La clase ControlData permite hacer un control de tipo leído*/
        numFichero = ControlData.lerByte(sc);
        numFicheroCorrecto = miMenu.rango(numFichero);
    } while (!numFicheroCorrecto);

    System.out.println("Ha seleccionado el fichero " + losFicherosObjetos.get(numFichero - 1).getName());
    return losFicherosObjetos.get(numFichero - 1).getName();
}
```

3. Mostrar el contenido del fichero XML seleccionado

- **static void leeTodosLosCaracteres(File ficheroSecuencialTexto)**
  1. Usar un objeto **FileReader** para muestrar por consola el contenido del fichero seleccionado.

4. Ejecutar consultas XPath sobre el fichero seleccionado y construir el DOM resultado de la consulta

1. Pedir la consulta por teclado.
2. Crear un objeto que representa la expresión de la consulta de tipo **XPathExpression** y controlar que la expresión sea correcta mediante el control de la excepción **XPathExpressionException**. (*diapositiva 10*)
3. Construir un DOM a partir de un fichero XML bien formado (parseamos el ficheroXML) a partir de un objeto File, String o ImputStream (fichero). (*diapositiva 10*)

5. **Mostrar el resultado** de la consulta **XPath** en formato **XML**

1. Recorrer el DOM, resultado de la consulta, con el método recursivo que ya tenemos implementado para mostrar por pantalla el contenido del DOM