

IES CHAN DO MONTE

C.S. de Desarrollo de Aplicaciones Multiplataforma

Modulo Acceso a datos

ACTIVIDAD 4: Persistencia XML (DOM y XPath) y Validación DTD

Esta práctica tiene como finalidad aplicar de forma integrada el **manejo de ficheros XML mediante el modelo DOM (Document Object Model)**, las **consultas XPath**, y la **Validación** de la Estructura del documento utilizando un DTD o esquemas.

Se mantendrá la estructura de capas de la aplicación y la jerarquía de clases modelo existente.

Objetivos

Consolidar el manejo del DOM para la lectura, escritura y modificación de documentos XML.

Aplicar consultas XPath para la búsqueda eficiente de nodos dentro del árbol DOM.

Garantizar la integridad estructural del documento XML mediante la validación con DTD o XSD.

Reforzar el principio de la arquitectura por capas, delegando la persistencia XML a la capa Persistencia.

Gestión de Corredores (Persistencia XML)

Objetivo: Desarrollar una aplicación para almacenar y gestionar la información de los corredores que participan en las competiciones. Esta información debe ser almacenada en un fichero XML (**corredores.xml**) y debe incluir datos personales, la relación con el equipo, y un historial de puntuaciones por año. La estructura del XML debe ser **validada contra un DTD (corredores.dtd)**. Se va a utilizar un parser DOM

1. Formato de fichero Corredores.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<corredores>
    <velocista codigo="C01" dorsal="1" equipo="E1">
        <nombre>Lucía Fernández Álvarez</nombre>
        <fecha_nacimiento>2000-10-20</fecha_nacimiento>
        <velocidad_media>10.4</velocidad_media>
        <historial>
            <puntuacion anio="2019">78.5</puntuacion>
            <puntuacion anio="2020">85.0</puntuacion>
            <puntuacion anio="2021">69.4</puntuacion>
            <puntuacion anio="2022">61.5</puntuacion>
            <puntuacion anio="2023">83.5</puntuacion>
        </historial>
    </velocista>
    <fondista codigo="C02" dorsal="2" equipo="E2">
        <nombre>Ana Pérez</nombre>
        <fecha_nacimiento>1995-06-02</fecha_nacimiento>
        <distancia_max>10000.0</distancia_max>
    </fondista>
    <velocista codigo="C03" dorsal="3" equipo="E1">
        <nombre>Mario López</nombre>
        <fecha_nacimiento>1998-04-12</fecha_nacimiento>
        <velocidad_media>9.9</velocidad_media>
        <historial>
            <puntuacion anio="2020">78.5</puntuacion>
            <puntuacion anio="2021">82.0</puntuacion>
        </historial>
    </velocista>
....
```

2. Definición (de atributo ID)

El atributo `codigo` definidos en los elementos `<velocista>` y `<fondista>` es el identificador único del documento marcado en el DTD como:

codigo	ID	#REQUIRED
--------	----	-----------

3. Información que almacenar por corredor:

Adaptación del Modelo y Estructura XML: La persistencia migra de un fichero binario secuencial (`corredores.dat`) a un fichero XML (`corredores.xml`), el cual debe estar enlazado a un DTD (`corredores.dtd`).

Las Clases Modelo (Corredor, Puntuacion, etc.) hay que hacer los siguientes cambios:

- Serialización: Se deben eliminar las interfaces y atributos de serialización (implements Serializable, serialVersionUID) de todas las clases modelo.
- Identidad: El corredor se identifica ahora mediante el atributo `codigo` (String, ej. "C01"), que actúa como ID primario en el XML. Adaptar la clase corredor para poder trabajar con este atributo
- ✓ Cada corredor será representado por un elemento raíz (`<fondista>` o `<velocista>`) y tendrá los siguientes elementos/atributos comunes :
 - **Código (String):** Identificador único del corredor dentro del sistema
 - **dorsal** (int): clave alternativa.
 - **nombre** (String): Nombre completo del corredor.
 - **fechaNacimiento** (LocalDate): Fecha de nacimiento del corredor.
 - **equipo** (int): Identificador o código del equipo al que pertenece el corredor. En esta fase, se almacenará solo la clave al equipo. Los datos completos del equipo se gestionarán en otro fichero **equipos.xml** que se implementará más tarde.
 - **Información de puntuaciones del corredor:** Los corredores tienen un historial de puntuaciones por año. La información de cada puntuación será:
 - **año** (int): Año de la puntuación.
 - **puntos** (float): Valor numérico obtenido en ese año.

El historial de puntuaciones de un corredor se guardará en un **array** de objetos de la clase **Puntuacion**, que debe estar ordenado por el año.

- ✓ Los corredores se clasificarán en dos tipos diferentes, y deben implementarse como clases derivadas de una clase base **Corredor**:

a. Fondista:

- Un **fondista** es un corredor especializado en pruebas de larga distancia, como maratones.
- Atributo adicional: **distanciaMax** (float), que representa la distancia máxima que el corredor puede recorrer (en kilómetros). Ejemplo: 10.0 (10 km), 42.195 (maratón).

b. Velocista:

- Un **velocista** es un corredor especializado en pruebas de corta distancia.
- Atributo adicional: **velocidadMedia** (float), que representa la velocidad media del corredor (en km/h). Ejemplo: 10.34 km/h.

4.- Estructura de la aplicación:

La aplicación debe estar estructurada en clases que deleguen responsabilidades de manera clara y modular, siguiendo el principio de **Responsabilidad Única**.

Cada clase se encargará de una sola responsabilidad bien definida. Evitar clases “todo-en-uno” que mezclen dominio, persistencia, formato y validación.

Capas y componentes (visión general)

◆ Capa Modelo (Dominio)

Responsabilidad: Representar los datos y las reglas sencillas del dominio.

- Contiene las clases que representan un corredor y las estructuras de valor que este usa (ej., **Corredor.java**, **Puntuacion.java**, Fondista.java, Velocista.java).
- **No** realiza I/O, persistencia XML, ni validación de negocio.

◆ Capa Persistencia (I/O)

Responsabilidad: Acceso y manipulación directa del **Document Object Model (DOM)** del fichero **corredores.xml**.

- **CorredorXML.java:** Contiene la lógica específica para la entidad Corredor (ej., anadirOModificarPuntuacion, borrarPuntuacionPorAnio).
- **XMLDOMUtils.java:** Contiene métodos genéricos y reutilizables para el manejo del DOM y XPath (ej., evaluarXPathNodo, eliminarElemento, guardarDocumentoXML, addAtributo,).
- Estos componentes **no** hacen validación de negocio ni impresión por consola.

◆ Capa Lógica (controladora o servicio)

Responsabilidad: Orquestar las operaciones de negocio, gestionar la lógica de *feedback* al usuario, y coordinar la persistencia. Es como el director de orquesta de la aplicación. Es el único que "habla" con la clase principal (el main) y es el único que "habla" con la capa de persistencia (CorredorXML).

- **GestorCorredores.java:**

- **Orquestación:** Llama a la capa Persistencia para manipular el DOM en memoria.
- **Control Transaccional:** **Guarda el XML en disco** (**corredores.xml**) solo si la operación del DOM (modificar, eliminar) fue exitosa.
- **Feedback:** Imprime mensajes informativos al usuario (ej., ÉXITO/ERROR) basados en el resultado de la capa de Persistencia.
- **Carga:** Gestiona la carga inicial del XML, incluyendo la **activación y gestión de la Validación DTD**.

◆ Capa Aplicación (Main)

Responsabilidad: Interacción con el usuario y punto de entrada del programa.

- Actúa como el punto de entrada principal del programa.
- Llama a los métodos de la **Capa Lógica (GestorCorredores)**.
- **No** debe conocer los detalles del DOM, XML, DTD, ni cómo se accede al disco.

Operaciones a realizar:

A. Inicialización y Lectura

1. **Carga del Documento XML y Validación:**

- **Requisito:** El sistema debe cargar el fichero **corredores.xml** en memoria.
- **Este método debe capaz de manejar los tres tipos de validación : (NO_VALIDAR, DTD, o XSD).**
- **Requisito:** Al realizar la carga, debe **activarse la Validación DTD** contra corredores.dtd. Si la validación falla, el sistema debe abortar la operación con un mensaje de error.

2. **Listar todos los corredores:**

- **Requisito:** Leer todos los datos de los corredores del documento en memoria y mostrar su información completa por consola.

3. **Mostrar la información de un corredor dado su Código (ID):**

- **Requisito:** Consultar y mostrar los datos de un corredor a partir de su **codigo** (ID). Gestionar el caso en que el corredor no sea encontrado.

4. **Mostrar la información de un corredor dado su dorsal:**

- **Requisito:** Consultar y mostrar los datos de un corredor a partir de su **dorsal**. Gestionar el caso en que el corredor no sea encontrado.

B. Modificación en Memoria (Operaciones CRUD)

Las siguientes operaciones manipulan el DOM en memoria. La **persistencia en disco** se gestiona por separado al final de la Capa Lógica si la operación fue exitosa.

5. Añadir Nuevo Corredor:

- **Requisito:** Añadir un nuevo corredor (de tipo Fondista o Velocista) al final del documento en memoria.
- **Requisito:** La lógica de negocio debe asignar un nuevo **dorsal** autoincremental (uno más que el último dorsal existente).

6. Eliminar un corredor por código (ID):

- **Requisito:** Eliminar un corredor completo del documento en memoria a partir de su **codigo**.

7. Método para introducir o modificar una puntuación:

- **Requisito:** Permitir agregar o actualizar una puntuación en el historial de un corredor existente (buscado por **codigo**).
- Si el **año** de la nueva puntuación **no existe**, se debe **agregar** un nuevo registro de puntuación.
- Si el **año** ya está presente, se debe **actualizar** el valor de la puntuación para ese año.
- Tras la operación exitosa, debe darse un *feedback* claro sobre si la puntuación se añadió o se modificó.

8. Método para eliminar una puntuación específica:

- **Requisito:** Eliminar un único registro de puntuación del historial de un corredor, utilizando su **codigo** y el **año** de la puntuación a borrar como claves.

C. Persistencia Final

9. Guardar el Árbol DOM en el Disco:

- **Requisito:** Implementar un método para **escribir** el Document DOM modificado de la memoria al fichero **corredores.xml** en el disco.
- **Aplicación:** Este método debe ser llamado por la Capa Lógica (GestorCorredores.java) después de cada operación de modificación (5, 6, 7 y 8) que haya finalizado con éxito.

Estructura de Métodos y Responsabilidades por Capa

1. Capa de Utilidades (XMLDOMUtils.java) - Herramientas Genéricas

Esta clase contiene métodos **estáticos** que actúan como una librería de **bajo nivel**. Su único enfoque es la manipulación directa del **Document Object Model (DOM)** o de los archivos XML, sin importar la naturaleza de los ficheros xml (corredores, coches, libros, puntuaciones, etc.). Sirven para cualquier documento XML

Por Ejemplo, podemos incluir:

Método	Responsabilidad	Razón de la Ubicación
cargarDocumentoXML()	Manejo de I/O y Configuración del Parser.	Es la función base para leer <i>cualquier</i> archivo XML del disco y crear el árbol DOM en memoria, incluyendo la gestión de validación (DTD/XSD).
guardarDocumentoXML()	Persistencia Física de la Estructura DOM.	Es el método que toma <i>cualquier</i> árbol DOM y lo escribe al disco. Es el proceso inverso a la carga.
evaluarXPathNodo()	Ejecución de Consultas Lógicas sobre el DOM.	Proporciona la funcionalidad de búsqueda avanzada (XPath) que es reutilizable en cualquier parte del árbol DOM.
eliminarElemento()	Manipulación Pura del Árbol DOM.	Implementa la lógica para desconectar un nodo de su padre; una función genérica del DOM.
buscarElementoPorId()	Búsqueda Optimizada del DOM.	Utiliza la función nativa del DOM (<code>getElementById</code>), que es ajena a la lógica de negocio.
updateValorElemento()	Modificación de Contenido.	Cambia el texto contenido en cualquier nodo (<code><nOMBRE>TEXTO</nOMBRE></code>). Es una operación universal sobre nodos.

<code>addElement(doc, nombre, parente)</code>	Creación de Elemento (Sin Valor).	Crea una nueva etiqueta (<code><nombre></code>) y la adjunta como hija a un nodo <code>parente</code> existente en el DOM. Es una operación básica de construcción de estructura.
<code>addElement(doc, nombre, valor, parente)</code>	Creación de Elemento (Con Valor).	Crea una nueva etiqueta (<code><nombre></code>) con su contenido de texto (<code>valor</code>) y la añade como hija al nodo <code>parente</code> . Extiende la función anterior para el contenido.
<code>addAtributo(doc, nombre, valor, elemento)</code>	Adición de Atributo Estándar.	Crea un atributo (<code>nombre</code> y <code>valor</code>) y lo adjunta al <code>elemento</code> DOM especificado. Necesario para definir propiedades en las etiquetas.
<code>addAtributoId(doc, nombre, valor, elemento)</code>	Adición de Atributo ID.	Crea un atributo, lo adjunta al <code>elemento</code> , y notifica al DOM que este atributo debe ser tratado como un ID (importante para optimizar búsquedas como <code>buscadorElementoPorId()</code>).
<code>obtenerTexto()</code>	Extracción Segura de Contenido de Texto.	Recibe un elemento <code>padre</code> y busca una etiqueta específica dentro. Es la función base para leer el valor contenido en un nodo (<code><etiqueta>VALOR</etiqueta></code>) de forma segura, devolviendo una cadena vacía si el nodo no existe. Es clave para el mapeo de lectura .

2. Capa de Persistencia (CorredoresXML.java) - Mapeo de Entidad

Esta clase actúa como el **Data Access Object (DAO)** para la entidad Corredor. Su responsabilidad es el **mapeo** entre la estructura XML y los objetos Java, utilizando las utilidades de **XMLDOMUtils.java**. **No contiene lógica de negocio**

Método	Responsabilidad Fundamental	Razón de la Ubicación
<code>cargarCorredores()</code>	Mapeo XML a Objeto (Lectura).	Sabe que tiene que buscar nodos <code><fondista></code> y <code><velocista></code> y cómo construir los objetos <code>Corredor</code> a partir de los subelementos y atributos XML.
<code>insertarCorredor()</code>	Mapeo Objeto a XML (Escritura).	Sabe qué atributos (<code>codigo</code> , <code>dorsal</code>) y qué subelementos (<code><nombre></code> , <code><historial></code>) son necesarios para crear la representación XML de un objeto <code>Corredor</code> .
<code>eliminarCorredorPorCodigo()</code>	Búsqueda + Delegación de Borrado.	Busca el nodo del corredor por código y llama a la utilidad <code>XMLDOMUtils.eliminarElemento()</code> para ejecutar el borrado en el DOM.
<code>addOrUpdatePuntuacion()</code>	Manipulación Específica de Entidad.	Sabe que debe buscar el historial, usar XPath con el contexto del historial para encontrar la puntuación, y determinar si debe modificar o añadir un nodo.
<code>eliminarPuntuacionPorAnio()</code>	Manipulación de Sub-estructura.	Localiza la puntuación por año (usando XPath relativo) y llama a la utilidad <code>eliminarElemento()</code> sobre ese nodo.
<code>obtenerSiguienteDorsal()</code>	Acceso a Datos para Lógica de Negocio.	Lee el conjunto de datos para determinar el dorsal más alto. Este resultado es consumido por la Capa de Negocio.

3. Capa Lógica (GestorCorredores.java) - Orquestación y Control de Negocio

Esta clase es la **controladora central**. Contiene la **lógica de negocio** (las reglas de la aplicación) y es el único punto de contacto con la Capa de Aplicación (main) y la Capa de Persistencia. **No tiene llamadas al DOM**.

- Aislamiento del DOM: La clase no contiene código DOM (no usa Node, Element, XPathExpression, etc.). Solo maneja el Document y los objetos de negocio (Corredor, Puntuacion). Esto mantiene la Responsabilidad Única.
- Delegación a Persistencia: Todas las operaciones complejas (cargar lista, buscar por ID, obtener dorsal) son delegadas a la instancia de CorredorXML .
- Todas las operaciones de modificación (insertar, eliminar, actualizar puntuación) se realizan sobre este documento en memoria a través del gestor (CorredorXML).
- Las operaciones son temporales (no se consolidan) hasta que se llama explícitamente al método guardarDocumentoDOM(rutaDestino).
- La capa controla el flujo de la aplicación y se comunica con el usuario:
 - Manejo de Excepciones: Los métodos como cargarDocumento y insertarCorredor capturan las ExcepcionXML lanzadas desde la Capa de Persistencia, muestran el mensaje de error de forma amigable y evitan que el programa falle de forma inesperada.
 - Información al Usuario: Métodos como mostrarCorredorPorCodigo y actualizarPuntuacion reciben los datos/resultados de la persistencia y los formatean para su visualización, incluyendo mensajes de ÉXITO, ERROR o DETALLES (como "Puntuación MODIFICADA" vs. "AÑADIDA").

Ejemplos de clases

1. Control y Persistencia (Carga y Guardado)

Estos métodos son cruciales para inicializar el sistema y consolidar los cambios.

Método	Propósito	Tareas de la Capa de Negocio
cargarDocumento()	Inicialización del Sistema y Control de Carga.	Llama a CorredorXML.cargarDocumentoDOM(). Captura la ExcepcionXML para gestionar errores críticos de formato o validación DTD/XSD y comunica el resultado al usuario.
guardarDocumentoDOM()	Consolidación de la Transacción.	Llama a CorredorXML.guardarDocumentoDOM(). Esta es la única operación que escribe todos los cambios pendientes de la memoria DOM al disco duro.

2. Lectura y Consulta de Datos (Visualización)

Estos métodos obtienen los datos mapeados por la capa de persistencia y los presentan al usuario.

Método	Propósito	Tareas de la Capa de Negocio
mostrarCorredores()	Listado General	Llama a gestor.cargarCorredores() para obtener la List<Corredor> completa desde el DOM en memoria y se encarga de formatearla y mostrarla por consola.
mostrarCorredorPorCodigo()	Búsqueda por Identificador	Llama a gestor.buscarCorredorPorId() y, si el objeto Corredor existe, se encarga de formatear y mostrar todos sus detalles (incluido el historial de puntuaciones).
mostrarCorredorPorDorsalRecorrido()	Búsqueda por Dorsal (Recorrido)	Llama al método de persistencia que recorre secuencialmente todos los nodos en el DOM para encontrar el Corredor con el dorsal coincidente. Si lo encuentra, se encarga de formatear y mostrar todos sus detalles .

<code>mostrarCorredorPorDorsalXPath()</code>	Búsqueda por Dorsal (XPath)	Llama al método de persistencia que usa una expresión XPath (<code>//corredor[@dorsal='X']</code>) para localizar el nodo directamente en el DOM. Si lo encuentra, se encarga de formatear y mostrar todos sus detalles .
--	-----------------------------	---

3. Modificación y Escritura en Memoria (Operaciones CRUD)

Estos métodos modifican el DOM en memoria y aplican las reglas de negocio antes de la persistencia. Los cambios quedan **pendientes de guardado** hasta que se ejecute guardarDocumentoDOM().

Método	Propósito	Lógica de Negocio Aplicada / Control
<code>insertarCorredor()</code>	Inserción con Reglas.	Implementa la regla del dorsal autoincremental (llama a <code>obtenerSiguienteDorsal()</code> , asigna valor al objeto) y la regla de unicidad (ve <code>existeCodigo()</code>) antes de llamar a <code>gestor.insertarCorredor()</code> .
<code>eliminarCorredorPorCodigo()</code>	Eliminación por ID.	Llama a <code>gestor.eliminarCorredorPorCodigo</code> . Controla el <code>boolean</code> de retorno para informar si el corredor fue eliminado de la memoria DOM o no fue encontrado.
<code>eliminarCorredorPorDorsal()</code>	Eliminación por Dorsal.	Llama a <code>gestor.eliminarCorredorPorDorsal</code> . Controla el <code>boolean</code> de retorno.
<code>actualizarPuntuacion()</code>	Modificación/Adición Inteligente.	Llama a <code>gestor.addORupdatePuntuacion()</code> . Lógica más importante aquí es evaluar el enum de retorno (<code>MODIFICADO</code> , <code>AÑADIDO</code> , <code>NO_ENCONTRADO</code>) para dar feedback muy preciso al usuario.
<code>borrarPuntuacion()</code>	Eliminación de Sub-Estructura.	Llama a <code>gestor.borrarPuntuacionPorAnio</code> . Controla el <code>boolean</code> de retorno para informar sobre la eliminación de la puntuación específica.