


Open · Opened 2 weeks ago by  **Rubén Montero**

Guardando preferencias en XML

Resumen

- Echaremos un vistazo al formato XML
- Añadiremos un método `saveExampleXML` a nuestra clase `HomeCinemaPreferences.java` que creará y guardará un XML de ejemplo (`assets\example.xml`), con una etiqueta raíz y dos nodos
- Añadiremos `saveAsXML`, que guardará un XML (`cinemaPrefs.xml`) con los valores de las preferencias

Descripción

En las tareas anteriores hemos escrito y leído un fichero de disco muy sencillo:

```
username=John Doe
prefsDarkMode=true
```

Pero esta solución *a medida* se quedará corta en cuanto la cosa empiece a complicarse un poco:

- ¿Qué pasa si queremos que alguno de los campos tenga varios valores?
- ¿Qué pasa con los caracteres especiales? (e.g.: ¿Y si el `username` contiene el carácter igual (=)?)

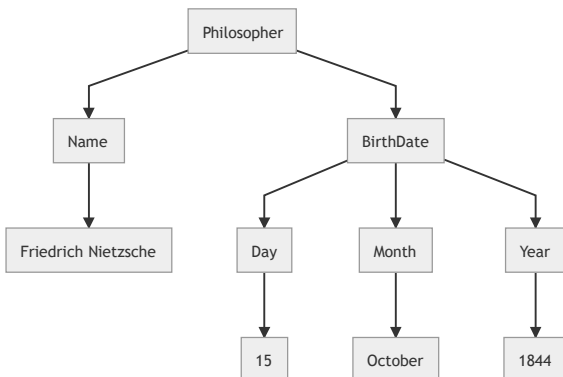
El formato XML

[eXtensible Markup Language \(XML\)](#) es un **formato estándar** que describe una serie de reglas sobre cómo almacenar información en un fichero.

Se basa en el concepto de **etiqueta**, expresado como una palabra *englobada* en los caracteres *menor que* y *mayor que* (`<` y `>`), y **anidamiento** (jerarquía) de tal forma que las etiquetas pueden tener contenido, sea éste texto u otras etiquetas.

¿Recuerdas el *árbol* de directorios de tu disco duro?

La idea es *muy similar*. Por ejemplo, esta información:



...se guardaría en XML como:

```
<Philosopher>
  <Name>Friedrich Nietzsche</Name>
  <BirthDate>
    <Day>15</Day>
    <Month>October</Month>
    <Year>1844</Year>
  </BirthDate>
</Philosopher>
```

Fíjate que las etiquetas se **abren** (`<Philosopher>`) y luego se **cierran** escribiendo un *slash* (`/`) que precede al nombre (`</Philosopher>`).

Esta estructura de información, sea la que sea, se conoce como un *modelo*. Concretamente, el [DOM](#) (*Document Object Model*) viene a significar dicha *estructura lógica*.

Veámoslo así:

- El archivo en disco duro está escrito con etiquetas (como en el segundo ejemplo)
- El DOM es una representación de ese archivo (como en el primer ejemplo), que se carga en la memoria RAM y contra la que trabajaremos empleando variables de clases específicas

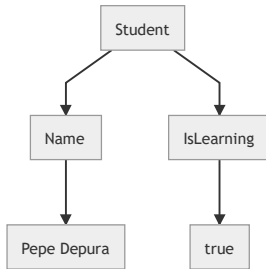
¿Y esto qué tiene que ver con Java?

XML es un formato para almacenar información, y existen clases específicas de la librería Java destinadas a usar XML.

Comencemos

Escribamos un nuevo método `saveExampleXML` en nuestro `HomeCinemaPreferences.java`.

El **objetivo final** de este **ejemplo** será crear *desde 0* un DOM como el siguiente:



...y escribirlo a disco:

```
<Student>
  <Name>Pepe Depura</Name>
  <IsLearning>true</IsLearning>
</Student>
```

Crear un `Document`

La clase que *representa* un documento se llama `Document`.

Por desgracia, no se puede *instanciar* (con `new`) directamente. A través de un [patrón factoría](#), la *instanciaremos* con estas **3** líneas:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

/* ... */

public void saveExampleXML() throws ParserConfigurationException {
    DocumentBuilderFactory factory1 = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory1.newDocumentBuilder();
    Document xmlDocument = builder.newDocument();
}
```

Crear nodo raíz

Cada *nodo* debe ser **creado** y **anexado** al documento o a un nodo *padre*.

Nosotros lo haremos así con el *nodo raíz*:

```
Student

Element rootNode = xmlDocument.createElement("Student");
xmlDocument.appendChild(rootNode);
```

Nombre

A continuación hay que: **1) Crear** un nodo `Name`:

```
Element node1 = xmlDocument.createElement("Name");
```

Name

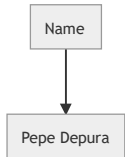
2) **Crear** un nodo *hoja* (será tipo texto, **Text**) que albergue el valor del nombre:

```
Text node1Content = xmlDocument.createTextNode("Pepe Depura");
```

Pepe Depura

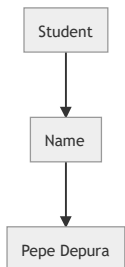
3) **Anexarlo**

```
node1.appendChild(node1Content);
```



Y 4) **anexar** dicho nodo a la *raíz*

```
rootNode.appendChild(node1);
```



¿Está aprendiendo el estudiante?

Escribiremos otras 4 líneas similares para el otro atributo de nuestro ejemplo:

```
Element node2 = xmlDocument.createElement("IsLearning");
Text node2Content = xmlDocument.createTextNode("true");
node2.appendChild(node2Content);
rootNode.appendChild(node2);
```

¡Ya está el XML!

...en la **memoria** de nuestro ordenador, en la forma de una **instancia** de **Document**

Ahora sólo falta **guardarlo** al disco duro.

Ojalá fuera tan sencillo como:

```
// Esto no existe
xmlDocument.saveToFile("assets\\example.xml")
```

Pero los pasos a seguir consisten en:

1. Instanciar, a través de una factoría, un **Transformer**
2. Convertir el **documento** a **DOMSource**
3. Instanciar un **StreamResult** asociado a un fichero
4. Invocar **.transform** para convertir el **DOMSource** a **StreamResult**

No hace falta memorizar estos pasos. Son relativamente mecánicos y acoplados a la librería **javax.xml**.

Basta con poner en práctica el código relevante:

```
TransformerFactory factory2 = TransformerFactory.newInstance();
Transformer transformer = factory2.newTransformer();
transformer.setOutputProperty(OutputKeys.INDENT, "yes");
DOMSource dom = new DOMSource(xmlDocument);
StreamResult outputStream = new StreamResult(new File("assets\\example.xml"));

transformer.transform(dom, outputStream);
```

(Tendrás que añadir también `TransformerException` a las excepciones lanzadas, señaladas en la firma del método `saveExampleXML`)

Probando, probando...

Invoquemos `saveExampleXML` desde `Main.java` .

Habrás que controlar las excepciones. Cualquiera de las soluciones propuestas por IntelliJ IDEA nos vale de momento.

Una vez verifiques el archivo guardado... ¡Enhorabuena! Has generado tu primer XML programáticamente desde Java.

La tarea

Se pide:

- Que `HomeCinemaPreferences.java` cuente con el código `saveExampleXML` visto arriba
- Que añadas un nuevo método `saveAsXML` :
 - Guardará el valor de los atributos `username` y `darkModePreferred` a disco duro en un archivo `assets\\cinemaPrefs.xml`
 - Tendrá un contenido XML *como* el siguiente:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Preferences>
  <Username>John Doe</Username>
  <PrefersDarkMode>true</PrefersDarkMode>
</Preferences>
```

Por último

Verifica que el *test* funciona correctamente.

Haz `commit` y `push` para subir los cambios al repositorio.



[Rubén Montero @ruben.montero](#) changed milestone to [%Sprint 1](#) 2 weeks ago



[Ania Blanco @ania.blanco](#) mentioned in commit [83dc0e76](#) 2 weeks ago