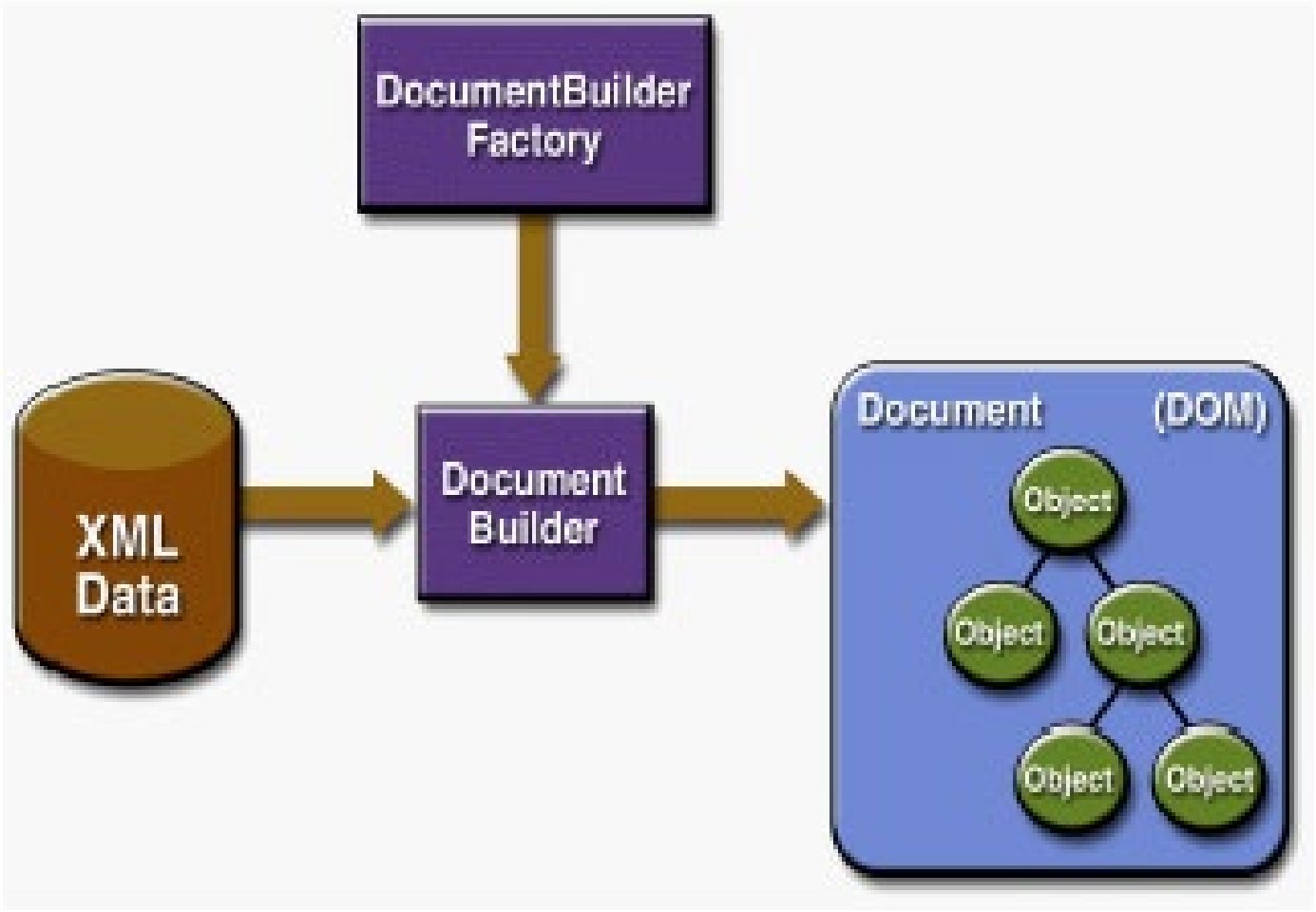
DOM
(Document Object Model)

DOM. Introducción

- Es una API orientada a objetos, para trabajar con documentos HTML válidos y XML bien formados.
- En DOM un documento es un conjunto de objetos jerárquicamente organizados en forma de árbol.
- Cada objeto será un nodo.
- Un nodo puede ser un elemento, un atributo, una entidad, ...

Arquitectura de la API DOM



DOM. Paquetes

`org.w3c.dom`	Define los interfaces DOM para documentos XML
`java.xml.parsers`	La clase factoría `java.xml.parsers.DocumentBuilderFactory` proporciona una instancia de la clase `DocumentBuilder` que se usa para instanciar un objeto `Document` que es conforme a la especificación DOM

DOM. Paquetes. Java Docs

`org.w3c.dom`	https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/package-summary.html
`java.xml.parsers`	https://docs.oracle.com/javase/8/docs/api/javax/xml/parsers/package-summary.html

DOM. Introducción

- DOM permite:
  - Construir nuevos documentos.
  - Modificar documentos ya existentes.
  - Navegar por la jerarquía del documento.
  - Modificar la jerarquía del documento.
  - Gestionar los objetos del documento.

DOM. ¿Dónde está?

- DOM Level 3 Core Specification. Emitido por el W3C en abril de 2004. Disponible en http://www.w3.org/TR/DOM-Level-3-Core

- DOM Level 3 Core Load and Save Specification. Emitido por el W3C en abril de 2004. Disponible en http://www.w3.org/TR/DOM-Level-3-LS

- DOM Level 3 Validation Specification. Emitido por el W3C en enero de 2004. Disponible en http://www.w3.org/TR/DOM-Level-3-Val

- DOM Level 4 (Review Draft). Emitido por el W3C en junio de 2019. Disponible en https://dom.spec.whatwg.org/review-drafts/2019-06/

DOM. Generalidades (I)

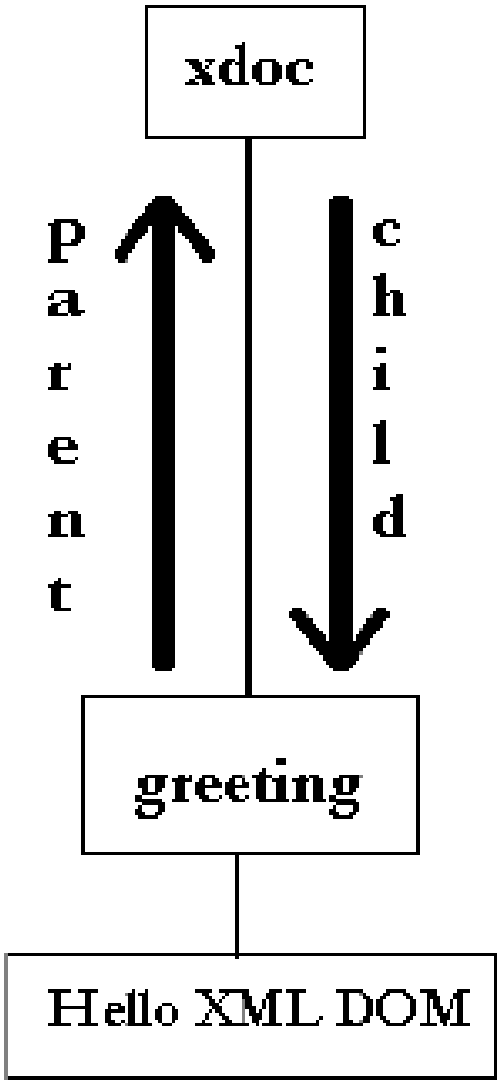
- DOM organiza un documento como un árbol de nodos, en forma de árbol y con una única raíz.
- El NODO es la unidad básica de procesamiento del documento.
- Todo nodo, excepto el raíz, tiene padre (*parent*), puede tener hijos (*child*) y puede tener hermanos (*sibling*).

DOM. Generalidades (II)

- DOM posee 12 tipos de nodos:
  - Document
  - Element
  - Attribute
  - Processing Instruction
  - Comment
  - Text
  - Cdata section
  - Document fragment
  - Entity
  - Entity reference
  - Document type
  - Notation

DOM. Ejemplo 1 (I)

```
<xdoc>
    <greeting>
        Hello XML DOM
    </greeting>
</xdoc>
```



Nodos:

- **Un nodo Document.**
- **Dos nodos Element: xdoc y greeting.**
- **Un nodo Text: "*Hello XML DOM*".**

DOM. Ejemplo 1 (II)

The **document** node:

Type: -document
NodeList: - [0]- document, [1]- <xdoc>, [2]- <greeting>, [3]- "Hello XML DOM"
Parent: -Null
ChildList -[0]-<xdoc>
Cdata: -

The **xdoc** node:

Type: -element
NodeList: - [0]- <xdoc>, [1]- <greeting>, [2]- "Hello XML DOM"
Parent: -document
ChildList -[0]-<greeting>, [1] "Hello XML DOM"
Cdata: -<greeting>Hello XML DOM</greeting>

DOM. Ejemplo 1 (III)

The **greeting** node:

Type: -element
NodeList: - [0]- <greeting>, [1]- "Hello XML DOM"
Parent: -xdoc
ChildList - [0] "Hello XML DOM"
Cdata: Hello XML DOM

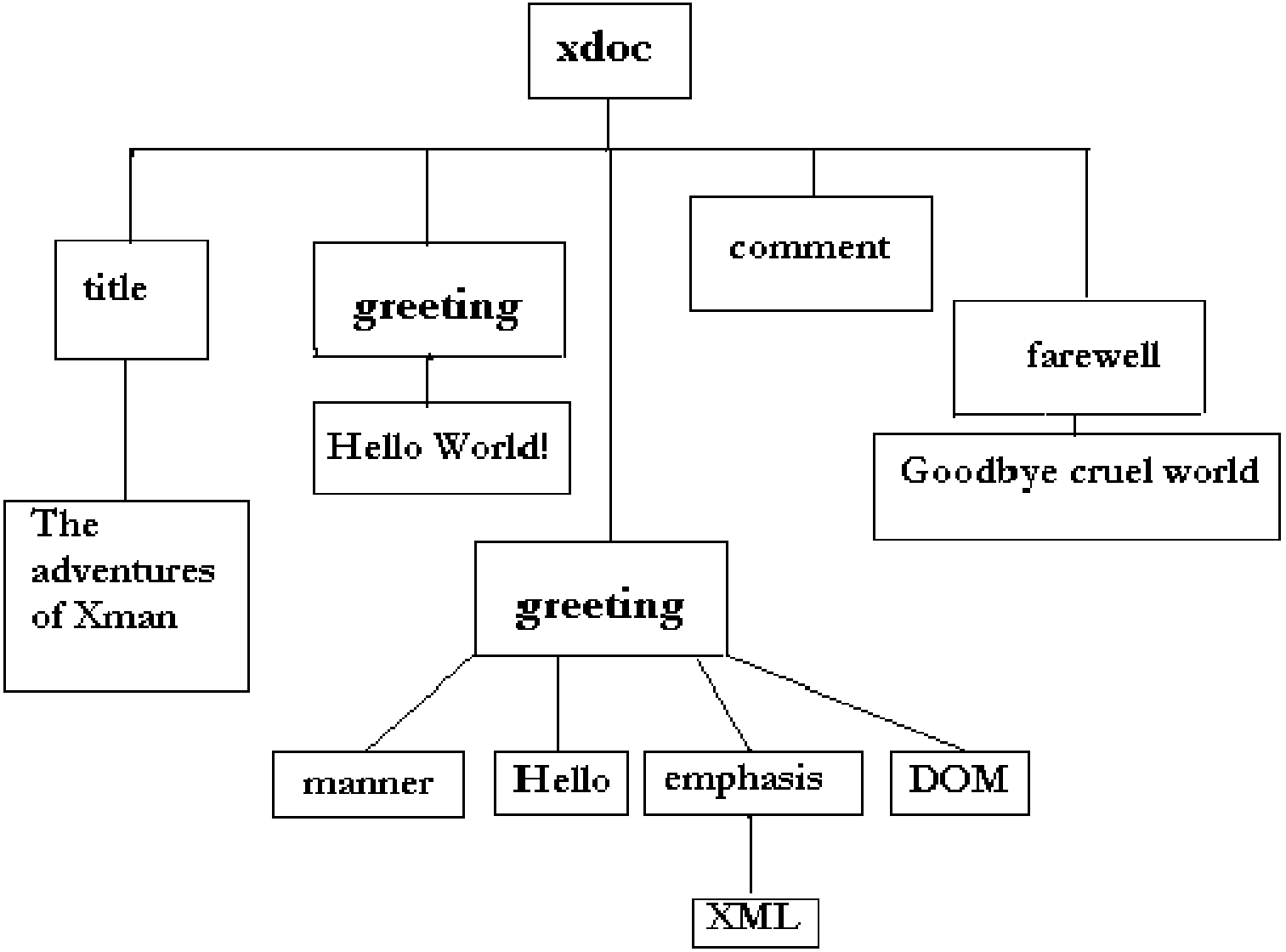
The "**Hello XML DOM**" node:

Type: -text
NodeList: null
Parent: -greeting
ChildList - null
TextContent - "Hello XML DOM"
Cdata: -null

DOM. Ejemplo 2 (I)

```
<xdoc>

      <title> The Adventures of Xman </title>

      <greeting> Hello world! </greeting>

      <greeting manner="cordial">
            Hello <emphasis> XML </emphasis> DOM
      </greeting>

      <!- Serie de aventuras-->

      <farewell>
            Goodbye cruel world
      </farewell>

</xdoc>
```

DOM. Ejemplo 2 (II)



Creación de un documento DOM vacío

```
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

  public class CreateBlankDocument {

   public static void main(String[] args){
     //Se instancia de DocumentBuilderFactory ->factoría
     DocumentBuilderFactory factory =
                    DocumentBuilderFactory.newInstance();
     try{
         // Se instancia la factoría -> DocumentBuilder
         DocumentBuilder builder =
                        factory.newDocumentBuilder();
         //Se crea un DOM vacío
          Document doc = builder.newDocument();
      }catch(Exception e){
        System.out.println(e.getMessage());
      }
   }
  }
```

Configurando la factoría y validando el XML

```
import org.w3c.dom.*;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

public class CreateBlankDocument {
 public static void main(String[] args){
        if (args.length !=1) {……}
        //Se instancia de DocumentBuilderFactory ->factoría
        DocumentBuilderFactory factory =
                        DocumentBuilderFactory.newInstance();
        factory.setValidating (true);
        factory.setNamespaceAware (true);
        try{
            // Se instancia la factoría -> DocumentBuilder
            DocumentBuilder builder =
                                factory.newDocumentBuilder();
            //Crear un DOM a partir de un fichero
            Document doc = builder.parse(new File (args[0]));
        }catch(ParserConfigurationException|SAXException|IOException e){
          System.out.println(e.getMessage());
        }
    }
  }
```

Creación de la raíz de un documento

- El método `createElement` sirve para crear al elemento `raiz` que se añade con el método `appendChild:`

```
//crear el elemento raíz
Element raiz = doc.createElement("raiz");
//añadirlo al árbol
doc.appendChild(raiz);
```

Añadir un nodo hijo

- Con el método `createElement` se crea el elemento `hijo`, al que se le añaden atributos con `setAttribute`, se cuelga del árbol con el método `appendChild` y se añade el nodo `texto`:

```
//crea el elemento hijo
Element childElement =doc.createElement("hijo");
//Añade atributos al elemento
childElement.setAttribute("atributo1",
                          "valor del atributo1");
//Se cuelga de la posición adecuada en el árbol
raiz.appendChild(childElement);
//Se crea un nodo texto
Node valuetext = doc.createTextNode("texto");
//Se cuelga del elemento creado
childElement.appendChild(valuetext);
```

Resultado del ejemplo anterior

- Después de tratar adecuadamente la salida se obtiene:

```
<?xml version="1.0" encoding="UTF-8" ?>
<raiz>
  <hijo atributo1="valor del atributo1">
     texto
  </hijo>
</raiz>
```

The most common DOM methods at a glance

Reaching Elements in a Document

`document.getElementById('id')`: Retrieves the element with the given `id` as an object

`document.getElementsByTagName('tagname')`: Retrieves all elements with the tag name `tagname` and stores them in an array-like list

Reading Element Attributes, Node Values and Other Data

`node.getAttribute('attribute')`: Retrieves the value of the attribute with the name `attribute`

`node.setAttribute('attribute', 'value')`: Sets the value of the attribute with the name `attribute` to `value`

`node.nodeType`: Reads the type of the `node` (1 = element, 3 = text node)

`node.nodeName`: Reads the name of the `node` (either element name or #textNode)

`node.nodeValue`: Reads or sets the value of the `node` (the text content in the case of text nodes)

Navigating Between Nodes

`node.previousSibling`: Retrieves the previous sibling node and stores it as an object.

`node.nextSibling`: Retrieves the next sibling node and stores it as an object.

`node.childNodes`: Retrieves all child nodes of the object and stores them in an list. here are shortcuts for the first and last child node, named `node.firstChild` and `node.lastChild`.

`node.parentNode`: Retrieves the node containing `node`.

Creating New Nodes

`document.createElement(element)`: Creates a new element node with the name `element`. You provide the name as a string.

`document.createTextNode(string)`: Creates a new text node with the node value of `string`.

`newNode = node.cloneNode(bool)`: Creates `newNode` as a copy (clone) of `node`. If `bool` is `true`, the clone includes clones of all the child nodes of the original.

`node.appendChild(newNode)`: Adds `newNode` as a new (last) child node to node.

`node.insertBefore(newNode,oldNode)`: Inserts `newNode` as a new child node of node before `oldNode`.

`node.removeChild(oldNode)`: Removes the child `oldNode` from node.

`node.replaceChild(newNode, oldNode)`: Replaces the child node `oldNode` of node with `newNode`.

`element.innerHTML`: Reads or writes the HTML content of the given element as a string—including all child nodes with their attributes and text content.

Known browser quirks:

`getAttribute` and `setAttribute` are not reliable. Instead, assign the property of the element object directly: `obj.property = value`. Furthermore, some attributes are actually reserved words, so instead of `class` use `className` and instead of `for` use `HTMLfor`.

Real DOM compliant browsers will return linebreaks as text nodes in the `childNodes` collection, make sure to either remove them or test for the `nodeType`.

Recorrido de un árbol DOM

- Los pasos a seguir para recorrer un árbol son:
  - Localizar la raíz.
  - Obtener los nodos hijo directos

o bien

  - Obtener todos los nodos descendientes
  - Tratamiento de los atributos.

Localizar la raíz de un árbol

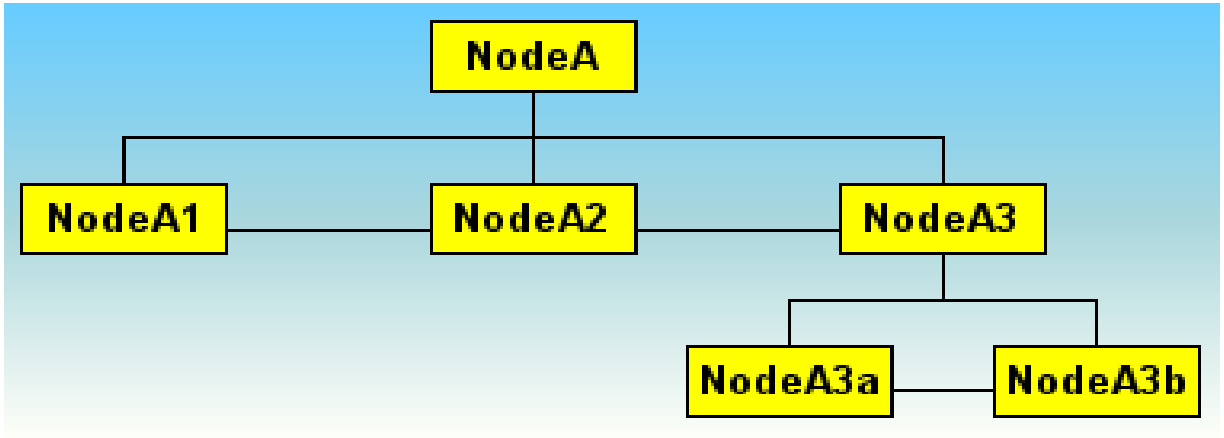
```
import org.w3c.dom.Element;

public class obtenerRaiz {

.............

    Element raiz= doc.getDocumentElement();
    System.out.println( "La raíz es: " +
                        raiz.getNodeName());
.............

}
```

Obtener los hijos directos (I)

- Usando `NodeList` (que entre otras, posee una propiedad que indica el número de hijos):

```
import org.w3c.dom.NodeList;

...........

    NodeList hijos = raiz.getChildNodes();
    System.out.println( "Número de hijos:" +
                        hijos.getLength());
...........

}
```

Node



- NodeA.firstChild = NodeA1
- NodeA.lastChild = NodeA3
- NodeA.childNodes.length = 3
- NodeA.childNodes[0] = NodeA1
- NodeA.childNodes[1] = NodeA2
- NodeA.childNodes[2] = NodeA3

- NodeA1.parentNode = NodeA
- NodeA1.nextSibling = NodeA2
- NodeA3.prevSibling = NodeA2
- NodeA3.nextSibling = null
- NodeA.lastChild.firstChild= NodeA3a
- NodeA3b.parentNode.parentNode= NodeA

Obtener los hijos directos (II)

- O bien, iterando por `Node`:

```
import org.w3c.dom.Node;
 ...........
 for (Node hijo = raiz.getFirstChild();
      hijo != null;
      hijo = hijo.getNextSibling()) {
  System.out.println(hijo.getNodeName ());
 }
 ...........

}
```

Obtener todos los nodos descendientes

- Haciendo un recorrido recursivo a partir del elemento raíz.

```
public class recorrerArbol{
 private static void pasarAOtro (Node inicio){
   System.out.println(inicio.getNodeName());
   for (Node hijo = inicio.getFirstChild();
        hijo != null;
        hijo = hijo.getNextSibling()) {
            pasarAOtro (hijo);
   }
 }
}
```

DOM
(Document Object Model)