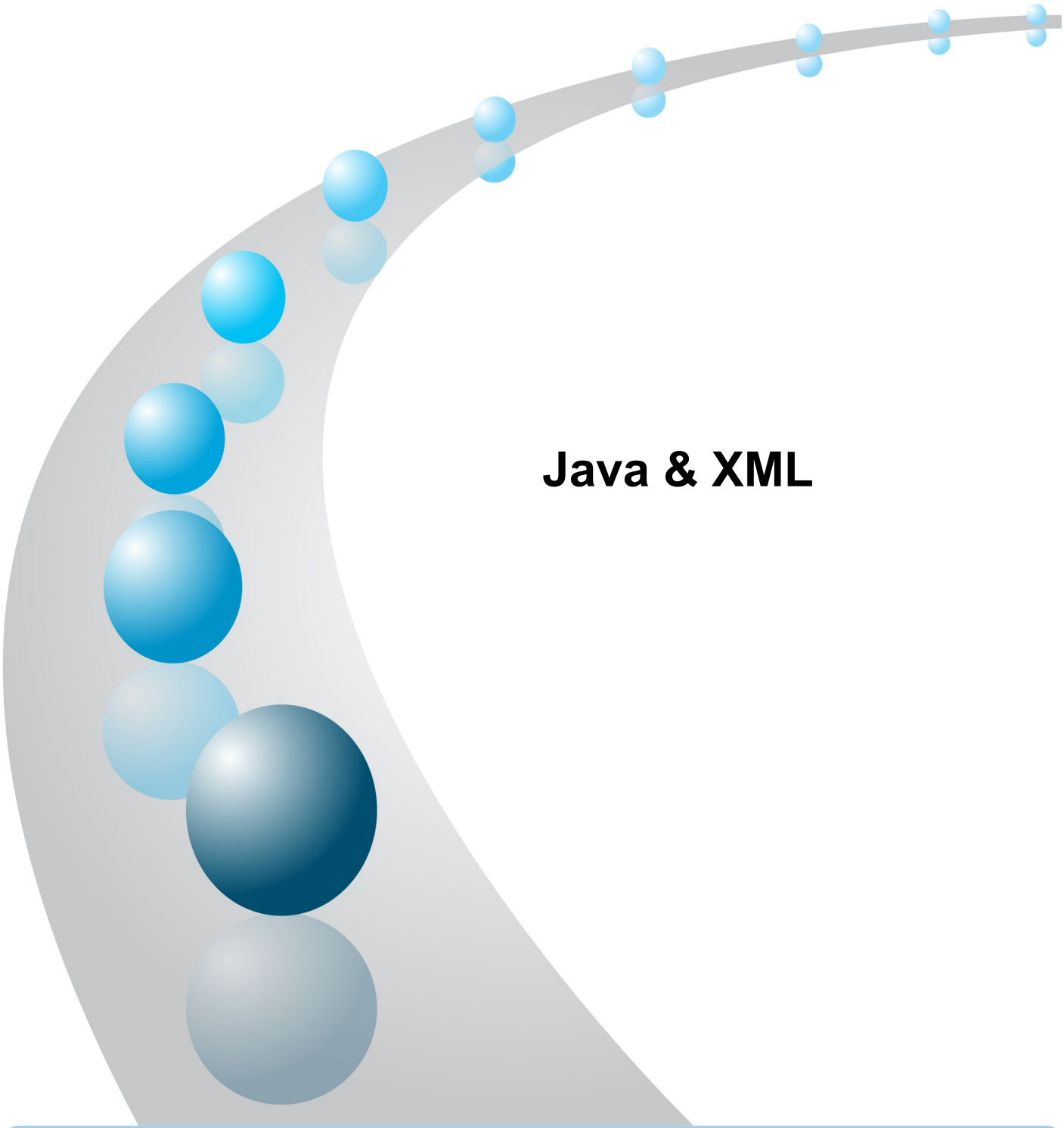
Noël Vaes
Java Trainer & Consultant



Java & XML

Roode Roosstraat 5
3500 Hasselt
België

+32 474 38 23 94
noel@noelvaes.eu
www.noelvaes.eu

Correspondentie inzake overnemen of reproductie kunt u richten aan:

Noël Vaes
Roode Roosstraat 5
3500 Hasselt
België

Tel: +32 474 38 23 94

noel@noelvaes.eu
www.noelvaes.eu

21/02/2018

Inhoudsopgave

Hoofdstuk 1. Java & XML

1.1. Inleiding

1.1.1. XML in de Java Standard Edition

Voor het omgaan met XML documenten zijn er in de *Java Standard Edition* twee technologieën:

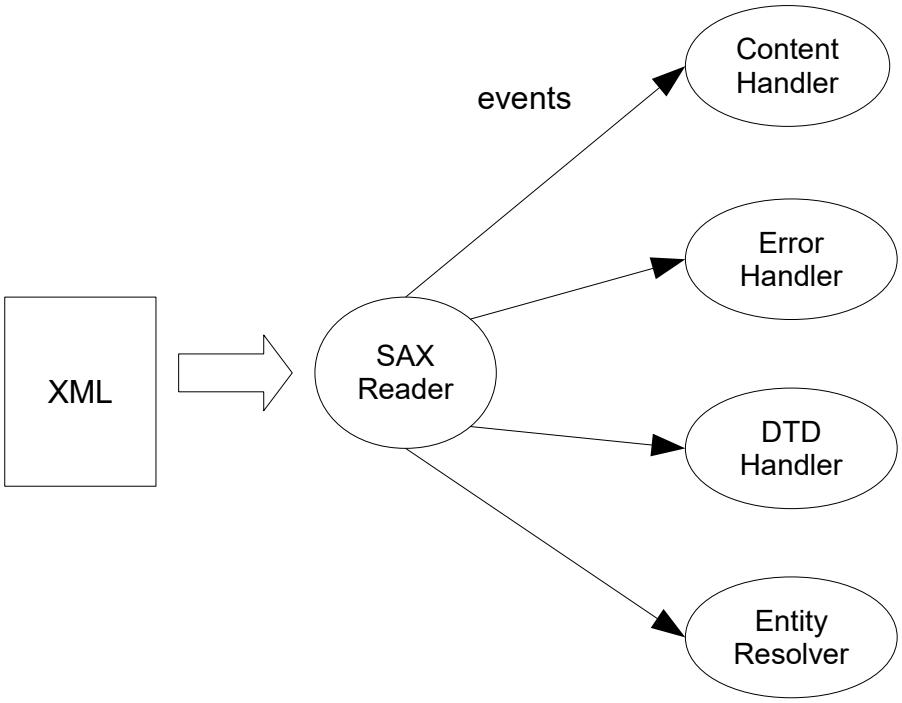
1. ***JAXP: Java API for XML Processing***
2. ***JAXB: Java Architecture for XML Binding***

JAXB zullen we in een afzonderlijk hoofdstuk behandelen.

JAXP wordt gebruikt om bestaande XML-documenten te *parsen,* aan te maken en te transformeren met behulp van XSLT. *JAXP* is enkel een specificatie en kan gebruikmaken van verschillende implementaties van de *parsers* die ingeplugd kunnen worden.

JAXP valt op zijn beurt uiteen in drie technieken om XML-documenten te behandelen:

1. ***SAX***: ***Simple API for XML*** Hierbij wordt een XML-document van het begin tot het einde overlopen door een *SAX Reader* en bij ieder element, attribuut of *processing instruction* wordt een *event* gegenereerd dat door een *event handler* wordt afgehandeld. Op die manier wordt het gehele document verwerkt door verschillende *event handlers.* Deze techniek is snel en vraagt weinig geheugen. Het is echter niet mogelijk hiermee documenten te wijzigen. Bovendien is het een verwerking die op laag niveau gebeurt en dat is niet steeds wat men nodig heeft.



2. ***DOM: Document Object Model*** Hierbij wordt van het XML-document een representatie in het geheugen gemaakt. Ieder element, attribuut enzovoort. wordt vertegenwoordigd door een Java-object (*node*). Deze objecten zijn aan elkaar gekoppeld en vormen zo een boomstructuur die overeenkomt met de structuur van het XML-document. Deze techniek maakt het mogelijk aanpassingen te doen en bovendien kan men makkelijk navigeren in de boomstructuur om het gewenste element te manipuleren. Het opbouwen van dit objectmodel vraagt wel de nodige processortijd en geheugen. Dit is vooral een probleem bij heel grote XML-documenten.

   DOM is in vele gevallen de meest aangewezen techniek als men XML-documenten wil manipuleren of creëren. De DOM API is bovendien gestandaardiseerd door W3C en is beschikbaar in verschillende programmeertalen. De DOM API is echter complex en onhandig in gebruik. Om die reden zijn er in de Java wereld een alternatieven ontwikkeld die makkelijker hanteerbaar zijn: JDOM, DOM4J

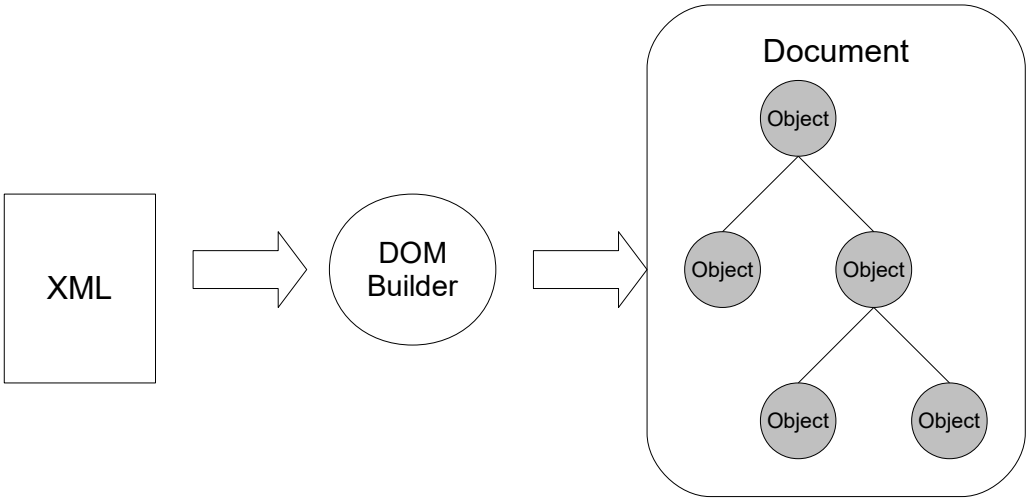
3. **XSLT:** *Extensible Stylesheet Language Transformation* Deze technologie dient om XML documenten te transformeren aan de hand van *XSL-stylesheet* documenten.

In deze cursus gaan we een alternatief voor DOM bekijken, namelijk JDOM. Daarnaast behandelen we JAXB en XSLT.

1.2. JDOM

1.2.1. Inleiding

JDOM is een *open source* project waarbij het de bedoeling is XML documenten op een eenvoudige wijze te behandelen en manipuleren en dit op een manier die voor Java programmeurs vertrouwd is. In tegenstelling tot DOM vertrekt JDOM helemaal vanuit de Java programmeertaal. JDOM kan bovendien samenwerken met *SAX* en *DOM.*



JDOM bouwt net als *DOM* een hiërarchie van Java-objecten die de structuur van een XML-document vertegenwoordigen.

JDOM is beschikbaar op de volgende website: ***www.jdom.org***

Opdracht 1: JDOM installeren

In deze opdracht gaan we het pakket JDOM integreren in een nieuw project.

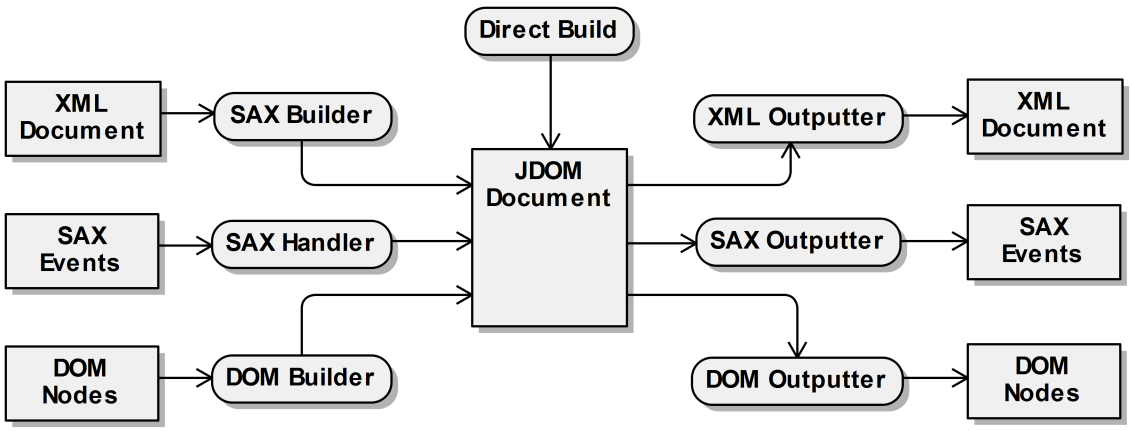
- Indien je gebruikmaakt van *Maven* voeg je de volgende *dependency* in de POM toe:

```
<dependency>
   <groupId>org.jdom</groupId>
   <artifactId>jdom2</artifactId>
   <version>2.0.6</version>
</dependency>
```

- Indien je geen gebruikmaakt van *Maven,* volg je de volgende stappen:
- Haal het pakket JDOM ***2.x.y*** van de website en pak het bestand uit in een lokale map. Lokaliseer het bestand JDOM***-2.x.y.jar*** en de documentatie in JDOM***-2.x.y-javadoc.jar***.
- Maak een nieuw project met de naam JDOM in je favoriete IDE.
- Plaats het bestand JDOM***-2.x.y.jar*** in de map ***lib*** van je project en zorg dat dit JAR-bestand opgenomen is in het *classpath* van het project.
- Koppel ook de API-documentatie zodat je makkelijk vanuit de IDE deze kan openen. Genereer eventueel de documentatie eerst met `ant javadoc`

1.2.2. Algemeen overzicht

JDOM maakt gebruik van een objectmatige voorstelling van een XML-document. We noemen dit een JDOM-document. In het onderstaande overzicht wordt aangegeven hoe een dergelijk JDOM-document gecreëerd wordt op basis van andere representaties van XML-documenten. Tevens wordt weergegeven hoe men van dit JDOM-*document* weer andere representaties kan genereren.



Voor het aanmaken van een JDOM-document hebben we de volgende mogelijkheden:

1. **Direct Build**: We kunnen het document zelf in Java-code aanmaken. Dit is nodig als we een nieuw XML-document moeten genereren. Hierbij wordt de documentstructuur stap voor stap opgebouwd in het geheugen.
2. **SAX Builder**: Via de *SAX Builder* kunnen we een bestaand XML-document lezen en hiervan een JDOM-document maken. Het *parsen* van het XML-document gebeurt op basis van een SAX-*parser*. Het XML-document kan aangeleverd worden via onder andere een bestand, *stream* of *reader*.
3. **SAX Handler**: Indien het XML-document aangeleverd wordt in de vorm van een aantal SAX-*events* kunnen we gebruikmaken van de *SAX Handler*. Deze zal op basis van de *events* het JDOM-document opbouwen.
4. **DOM Builder**: Indien we reeds beschikken over een DOM-structuur, kunnen we deze

omzetten naar een JDOM-document. De *DOM Builder* zorgt voor deze conversie.

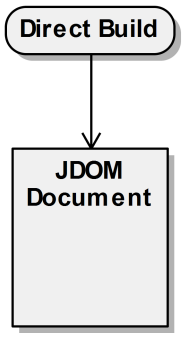
Eens we een JDOM-document in het geheugen hebben, kunnen we de verschillende elementen manipuleren. Zo kunnen we de inhoud en attributen van *tags* wijzigen, *tags* toevoegen of verwijderen enzovoort.

Na deze manipulaties kunnen we het JDOM-document weer omzetten naar een ander formaat:

1. **XML Outputter**: Hiermee kunnen we weer een XML-document in tekstvorm genereren en wegschrijven via een *outputstream* of *writer*.
2. **SAX Outputter**: Hiermee kunnen we *SAX events* genereren die dan op hun beurt afgehandeld kunnen worden door een of andere *SAX Handler*.
3. **DOM Outputter**: Hiermee kunnen we het JDOM-document omzetten naar een DOM-structuur.

In de volgende paragrafen zullen we de meest gebruikte mogelijkheden verder onder de loep nemen.

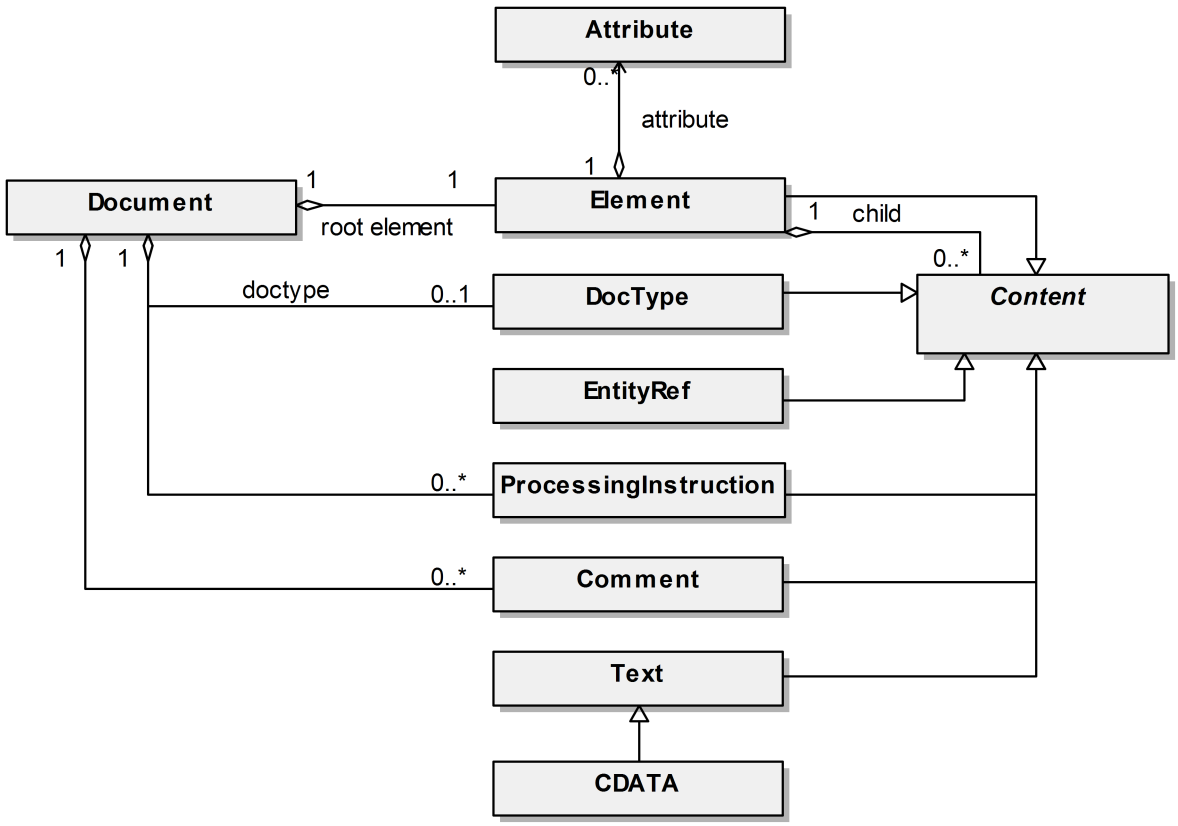
1.2.3. Een document rechtstreeks aanmaken



In sommige omstandigheden is het nodig een nieuw XML-document aan te maken. In dat geval gaan we rechtstreeks in het geheugen een JDOM-document aanmaken.
Zo'n JDOM-document wordt voorgesteld door een object van de klasse `Document`.

In het onderstaande klassendiagram worden de verschillende klassen en hun onderlinge relaties weergegeven:

Een JDOM-document bestaat dus uit een object van de klasse `Document`. Dit is de top van de boomstructuur. Dit `Document`-object heeft één *root-element* van de klasse `Element`. Daarnaast zijn eventueel nog een of meerdere objecten van de klassen `Comment` en `ProcessingInstruction` mogelijk. Tevens kan er één object van de klasse `DocType` zijn.



We richten ons nu op de klasse `Element`. Een *element* komt overeen met een *tag* in het XML-document. Zo'n *tag* kan naast attributen (`Attribute`) ook nog een inhoud hebben. Deze inhoud wordt heel algemeen voorgesteld door de klasse `Content`. Dit is een abstracte klasse met de volgende concrete subklassen:

1. `Element`: Dit stelt een *(sub)-tag* voor.
2. `EntityRef`: Dit stelt een *Entity Reference* voor. Bijvoorbeeld `&euml;`
3. `Comment`: Dit is een commentaarblok.
4. `Text`: Dit is gewone tekst als inhoud van een *tag*. Tekst die vervat is in een CDATA-sectie wordt voorgesteld door een object van de subklasse `CDATA`.

De klasse `Element` heeft methoden om `Content`-elementen en `Attribute`-elementen toe te voegen.

We illustreren dit alles met een concreet voorbeeld. Stel dat we het volgende XML-document willen opbouwen aan de hand van JDOM:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE message SYSTEM "Message.dtd" >
<message language="en">
  <greeting>Hello</greeting>
  <!--This is comment-->
  <audience>World</audience>
  <footer>&cr;</footer>
</message>
```

Hiermee komt de volgende DTD overeen: