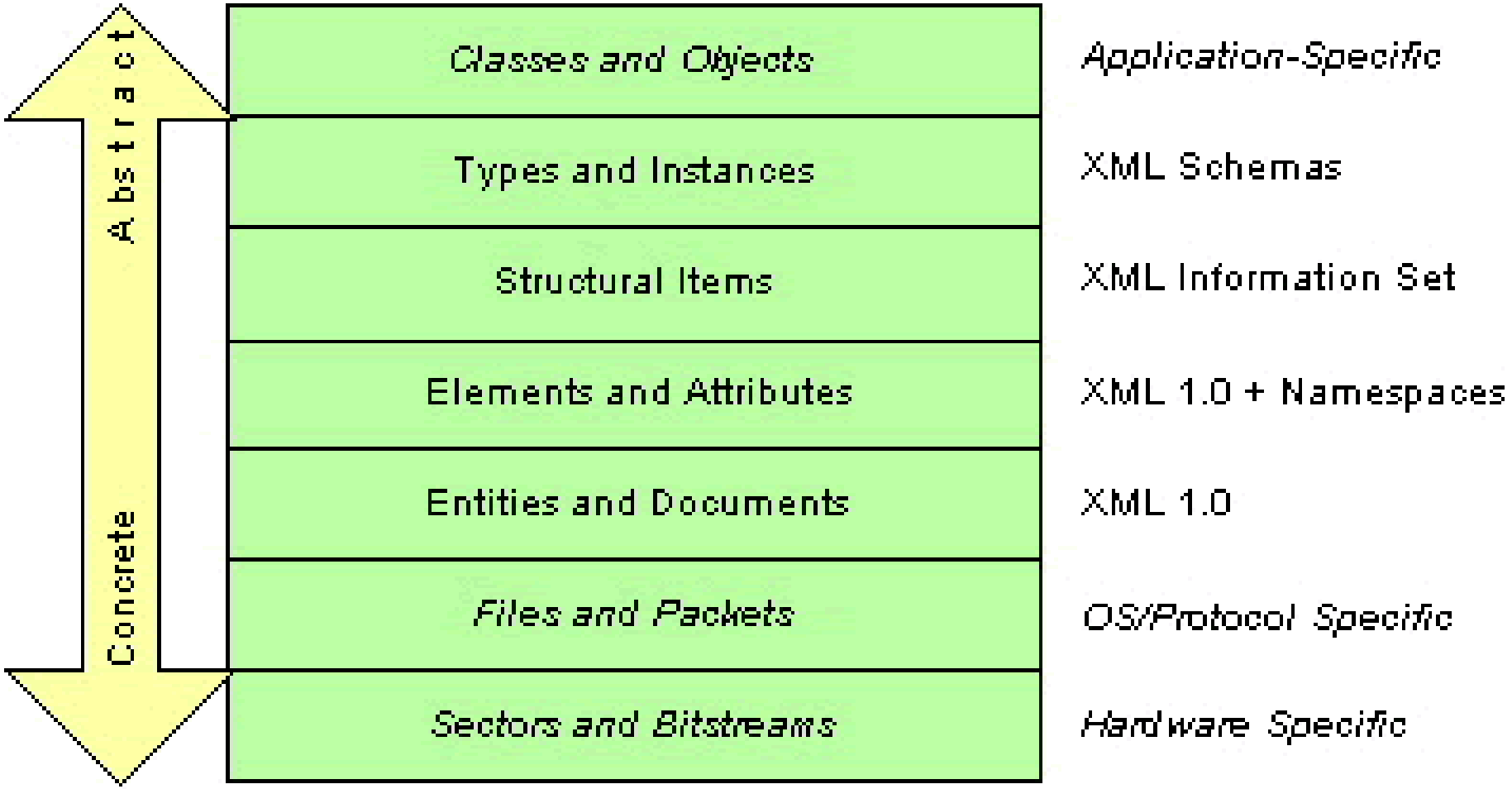
Parsing XML documents

DOM, SAX, StAX

XML-parsers

- XML-parsers are such *programs*, that are able to read XML documents, and provide access to the contents and structure of the document
  - XML-parsers are controlled by another program – e.g. a Java application
- There exist validating and non-validating XML-parsers
- Both types of XML-parsers have to notify about the errors regarding the errors that violate the constraints of well-formedness defined in the specification, and which occur in the document instance or in the instances read in

The XML protocol stack

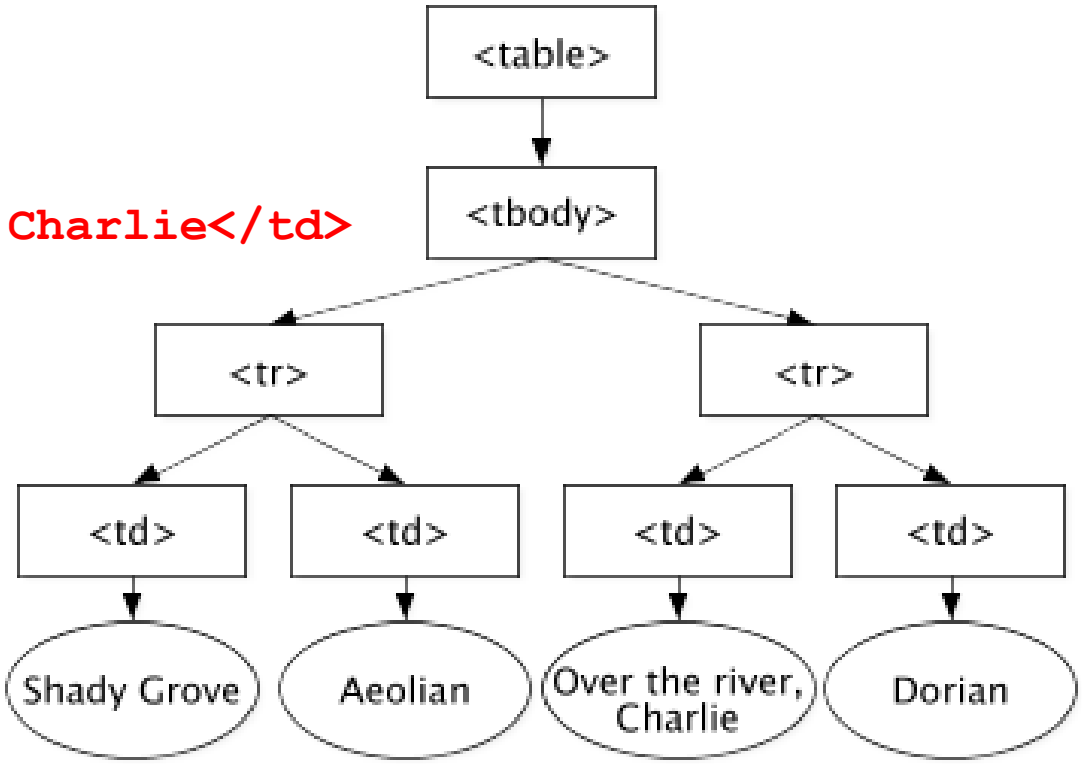


The document object model

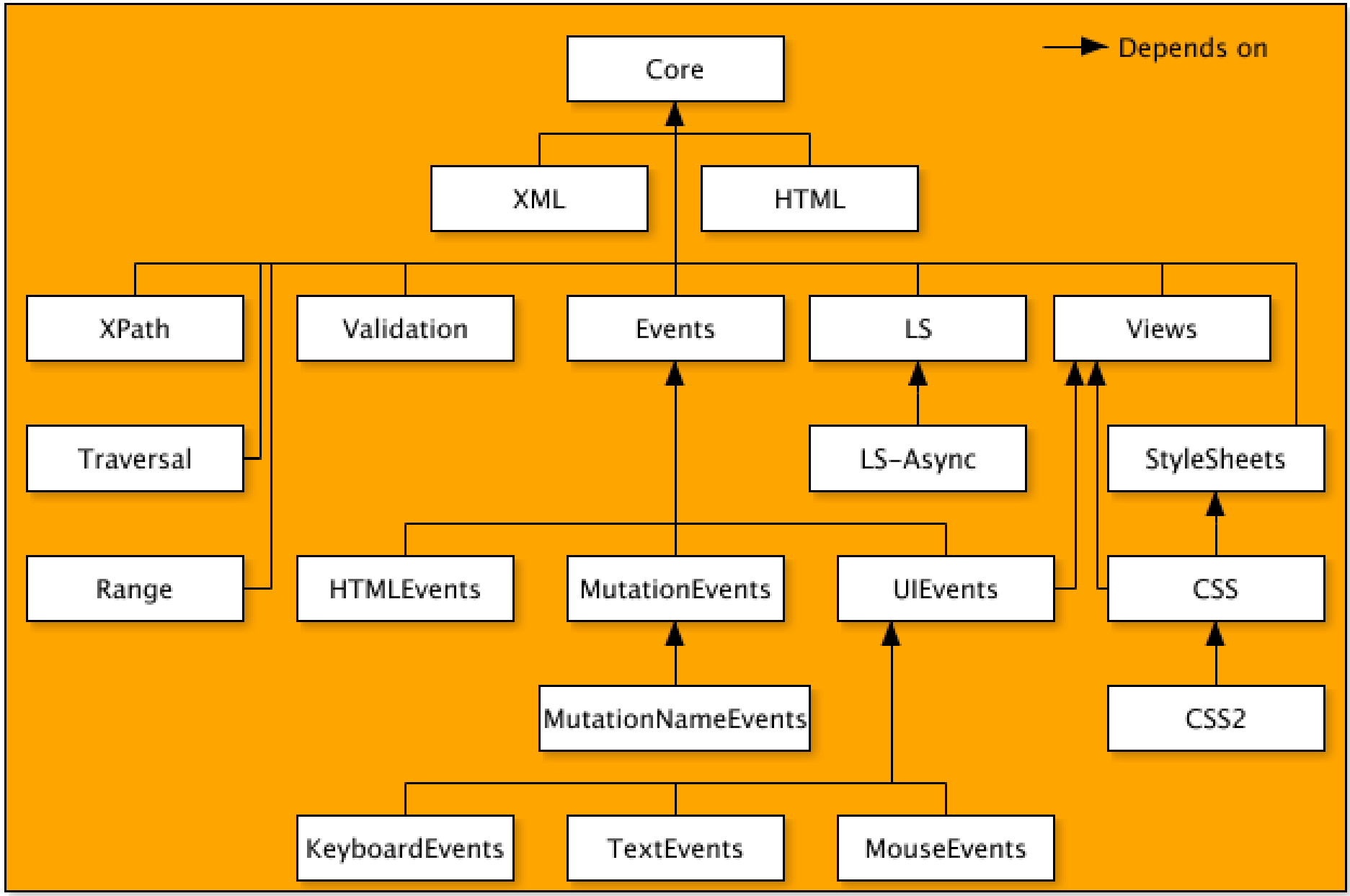
- What is DOM?
  - „The Document Object Model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents. The document can be further processed and the results of that processing can be incorporated back into the presented page.”
  - DOM is an API for manipulating HTML and well-formed XML documents

DOM example

```
<table>
  <tbody>
    <tr>
      <td>Shady Grove</td>
      <td>Aeolian</td>
    </tr>
    <tr>
      <td>Over the River, Charlie</td>
      <td>Dorian</td>
    </tr>
  </tbody>
</table>
```



The architecture of DOM (modules)

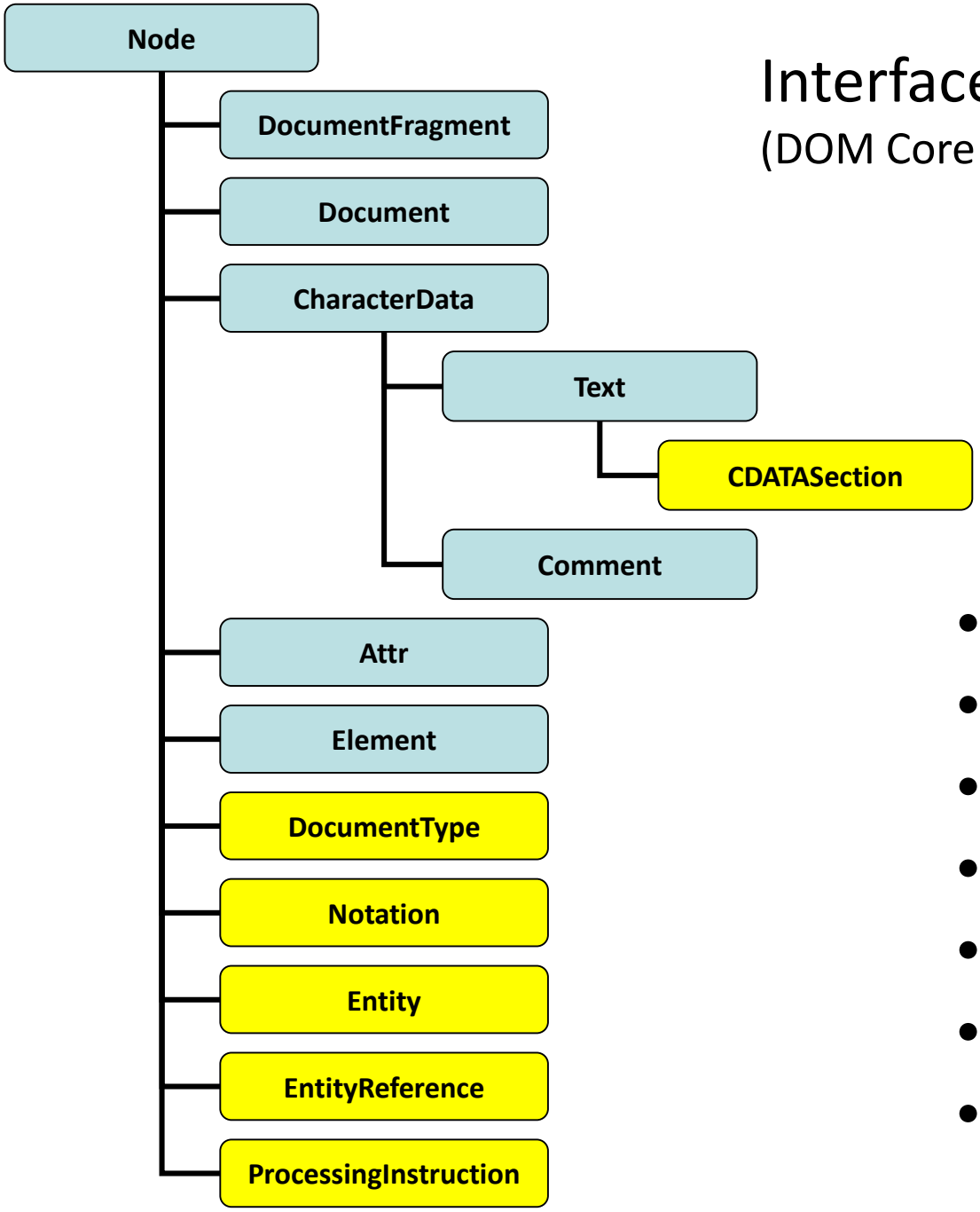


The characteristics of DOM (1)

- DOM treats the document logically as a tree (a hierarchy of Node objects) (*structural model*)
- DOM is an object model in the classical OO sense
  - Documents (and their individual parts) are objects wth identity, structure, behaviour and relationships
- The DOM API provides two possibilities:
  - An OO approach through an inheritence hierarchy
  - A simple (flattened) view („everything is a Node”)

The characteristics of DOM (2)

- Suitable for
  - Creating and building objects
  - Going through their structure
  - Adding, modifying and deleting elements and content
- DOM consists of modules (see architecture)



Interface hierarchy

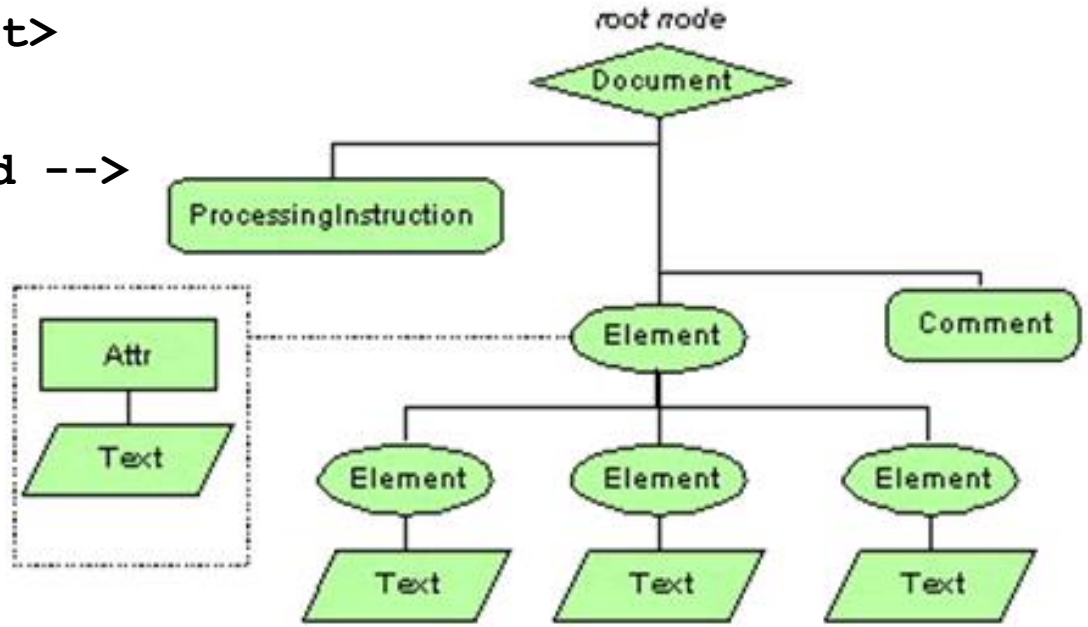
(DOM Core in blue, XML DOM in yellow)

- NodeList
- NamedNodeMap
- DOMException
- DOMImplementation
- DOMString
- DOMTimeStamp
- …

DOM-tree of a document

```
<?xml version="1.0"?>
<?order alpha ascending?>

<period name="Renaissance">
  <artist>Leonardo da inci</artist>
   <artist>Michelangelo</artist>
   <artist>Donatello</artist>
</period>
<!-- renaissance art period -->
```



DOM

- Text node: only in elements!
- Data only in text nodes!
- Document oriented vs. Data oriented
  - DOM: document model
  - Mixed content-modell (elements, texts can be mixed)

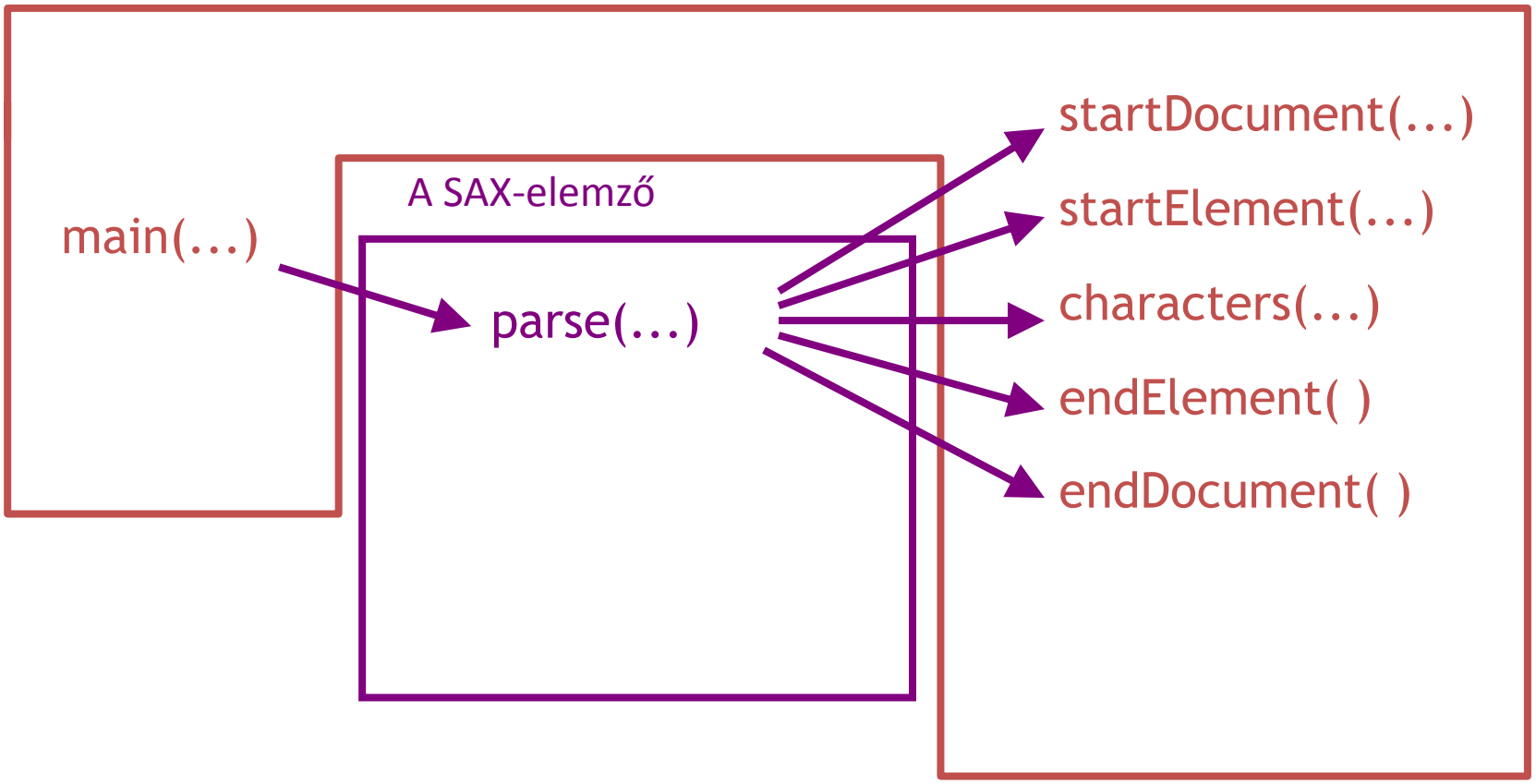
Simple API for XML (SAX)

- Event driven parsing of XML documents using callbacks
  - Does not build inner tree, rather handling methods are called by the individual handling events (start tag, end tag, …) e.g.:

```
public void startElement (String uri, String name,
                          String qName, Attributes atts) {
  if ("".equals (uri))
    System.out.println("Start element: " + qName);
  else
    System.out.println("Start element: {" + uri + "}" + name);
}
```

- Smaller memory requirements
- Not everything can be parsed this way (e.g. handling inner references)

SAX style parsing



Streaming API for XML (StAX)

- Pull parser
  - Application driven, not document driven
  - cursor API
    - Going through the XML documents from the beginning to the end
  - Event iterator API
    - The XML streamet is treated as a series of event objects
- The data of the XML document come as a stream, their parsing is done in order
  - A small fraction of the document can be accessed at a given time
  - The application accesses the data only if it requests it

Streaming API for XML (StAX)

- Bridge solution between DOM and SAX
- Drawback: harder to handle errors

```
FileInputStream fis = new FileInputStream(file);
XMLInputFactory factory =
   (XMLInputFactory)XMLInputFactory.newInstance();
XMLStreamReader staxXmlReader = (XMLStreamReader)
   factory.createXMLStreamReader(fis);
```

Streaming API for XML (StAX)

```
for (  int event = staxXmlReader.next();
       event != XMLStreamConstants.END_DOCUMENT;
       event = staxXmlReader.next())
{
   switch (event) {
       case XMLStreamConstants.START_DOCUMENT:
                    System.out.println("Start document " +
                    staxXmlReader.getLocalName()); break;
       case XMLStreamConstants.START_ELEMENT:
                    System.out.println("Start element " +
                    staxXmlReader.getLocalName());
                    System.out.println("Element text " +
                    staxXmlReader.getElementText()); break;
       case XMLStreamConstants.END_ELEMENT:
                    System.out.println("End element " +
                    staxXmlReader.getLocalName()); break;
       default: break;
   }
}
```

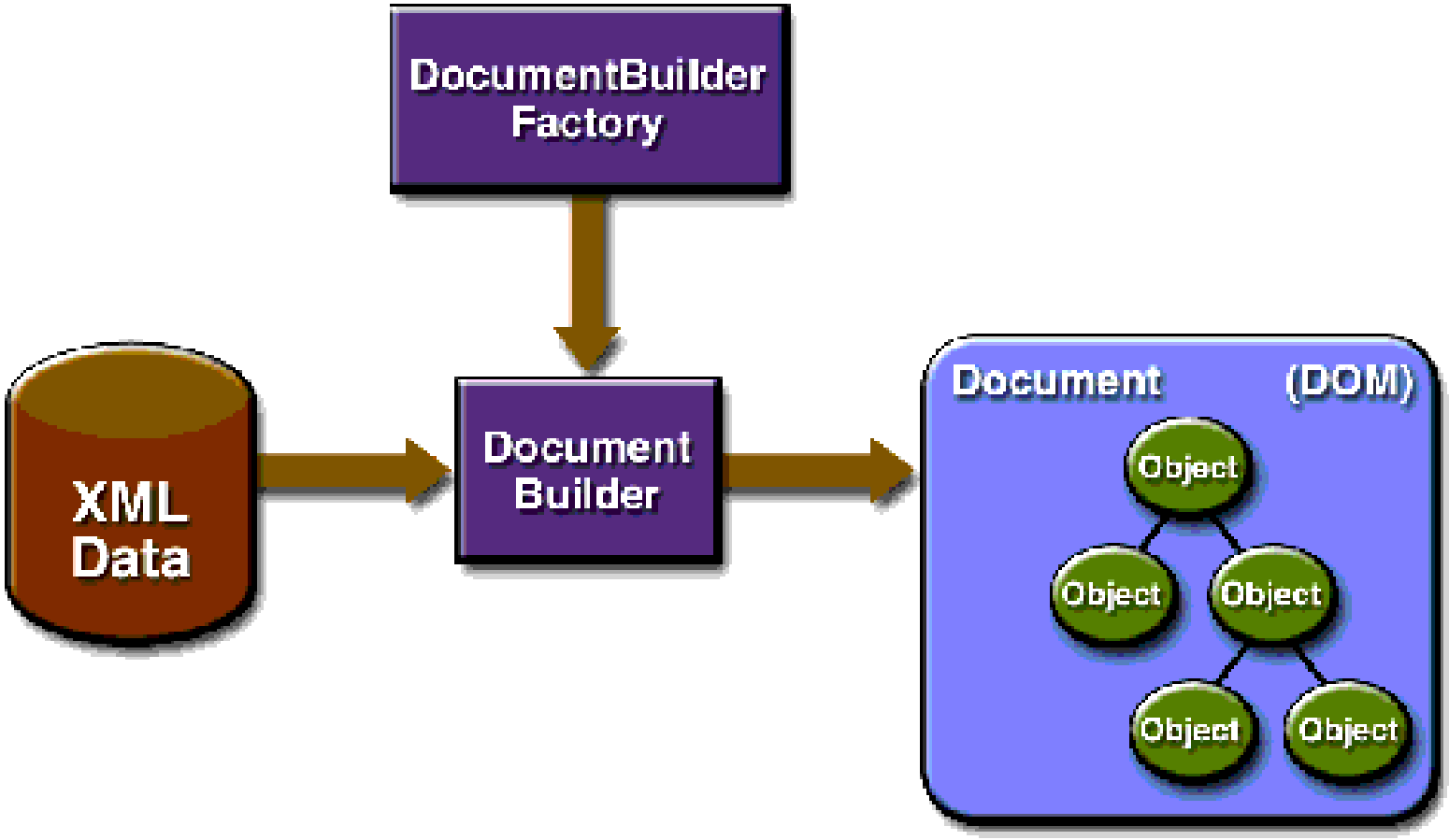
Java API for XML Processing (JAXP)

- https://jaxp.dev.java.net/
- JAXP 1.3
  - J2SE 5.0
  - XPath, validation, data types, Xinclude, …
- JAXP 1.4
  - J2SE 6
  - StAX (Streaming API for XML), classloading, factory methods
- Implementation independently in running time

JAXP 1.4

- Basic packages:
  - org.xml.sax: SAX 2.0
  - org.w3c.dom: DOM Level 3
  - javax.xml.parsers: initialization and handling of analyzers
  - javax.xml.transform: initialization and handling of transformers (XSLT parsers)
  - javax.xml.namespace: handling namespaces
  - javax.xml.stream: StAX
  - javax.xml.xpath: evaluation of XPath-expressions
  - javax.xml.validation: validation of XML documents

DOM API - basics



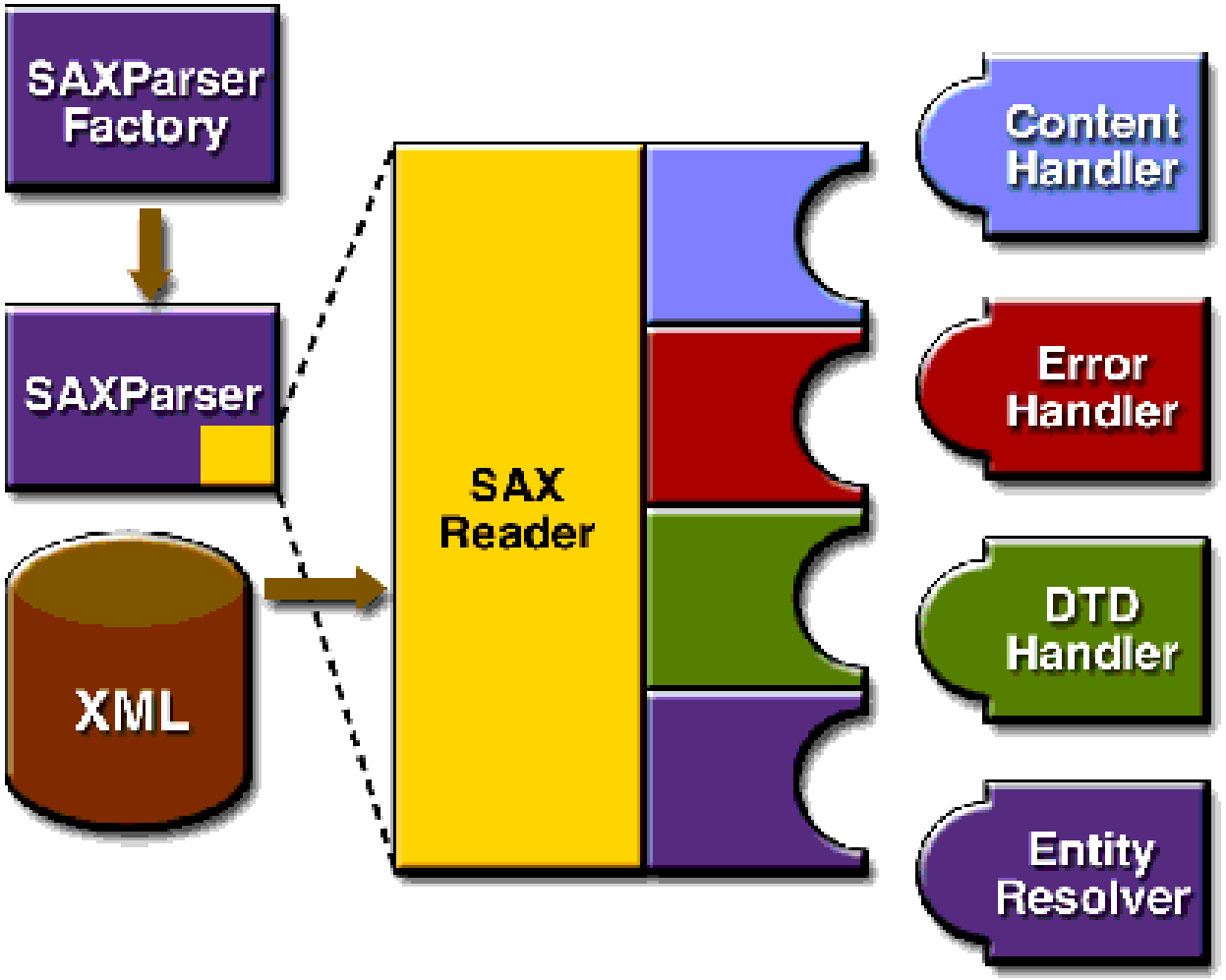
DOM API - basics

- javax.xml.parsers.DocumentBuilderFactory
- DocumentBuilder
  - newDocument()
- Document
- Element, Node, TextNode, stb.
- Packages:
  - org.w3c.dom
  - javax.xml.parsers

DOM – a simple Java applications

- Importing text classes
- Querying the DOM implementation
- Handling errors
- Creating Factory instances
- Analysis (parsing)
- Validation: factory.setValidating(true);
- Namespace handling: factory.setNamespaceAware(true);

SAX API – basics



SAX API - basics

- SAXParserFactory
  - Creates a SAXParser object
- SAXParser
  - Processes an XML data source and calls the methods of a DefaultHandler
- SAXReader
  - Hidden. Reads an XML data source.
- **DefaultHandler**
  - Defines events. Default handler
- **ContentHandler**
  - startDocument, endDocument, startElement, and endElement

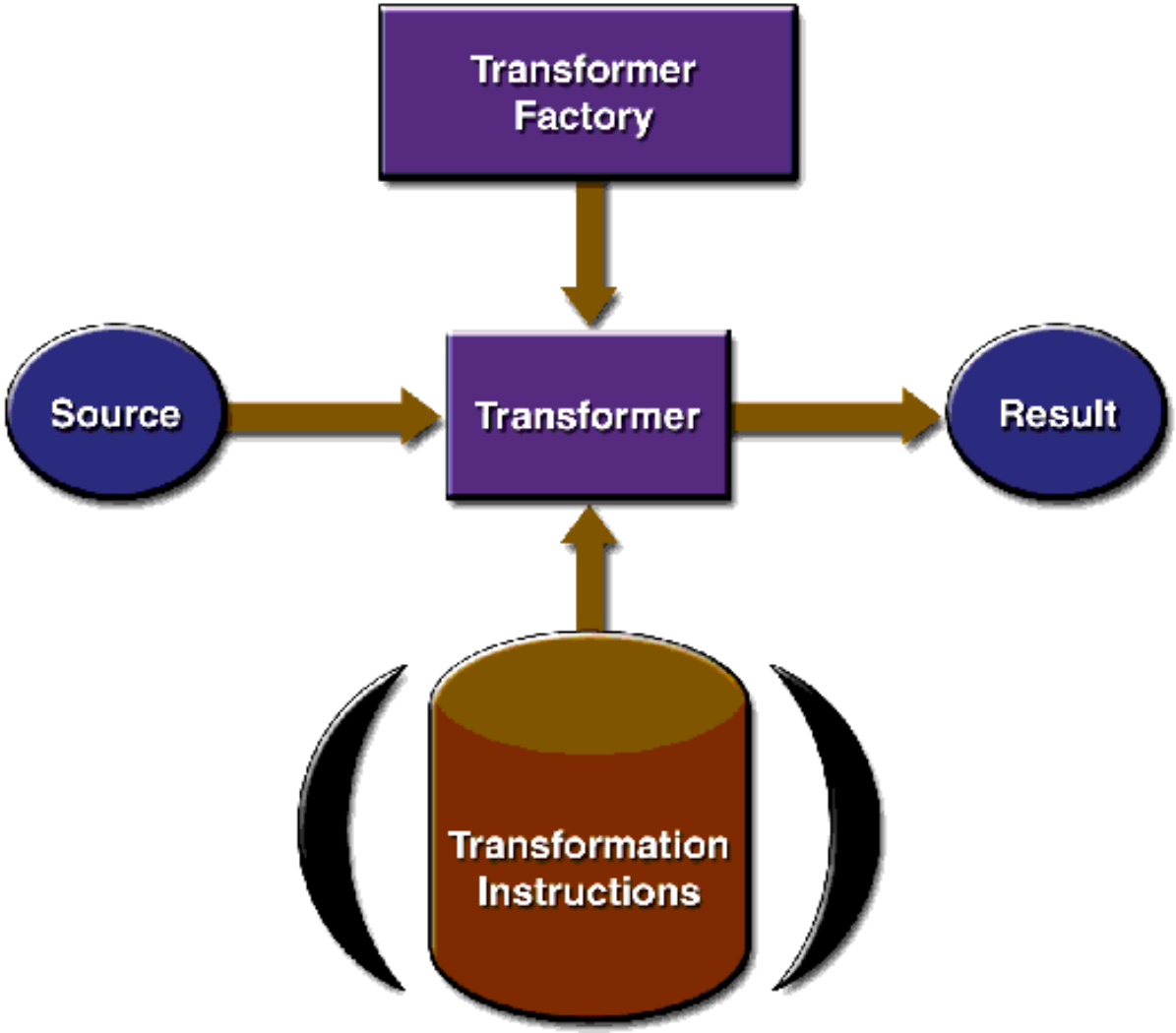
SAX API - basics

- ErrorHandler
  - error, fatalError methods
  - Validational exceptions
- DTDHandler
  - For DTD parsing
- EntityResolver
  - resolveEntity method
  - Document search URN based on - the *public identifier*
- Packages in the SAX API:
  - org.xml.sax
  - org.xml.sax.ext
  - org.xml.sax.helpers
  - javax.xml.parsers

SAX – a simple Java applicatipon

- Importing classes
- Doing I/O
- Implementing the ContentHandler interface (startDocument , endDocument, startElement, endElement, and characters)
- Setting up the Parser
- Handling I/O errors
- Formatting the output
- Handling content events

XSLT API - basics



XSLT API - basics

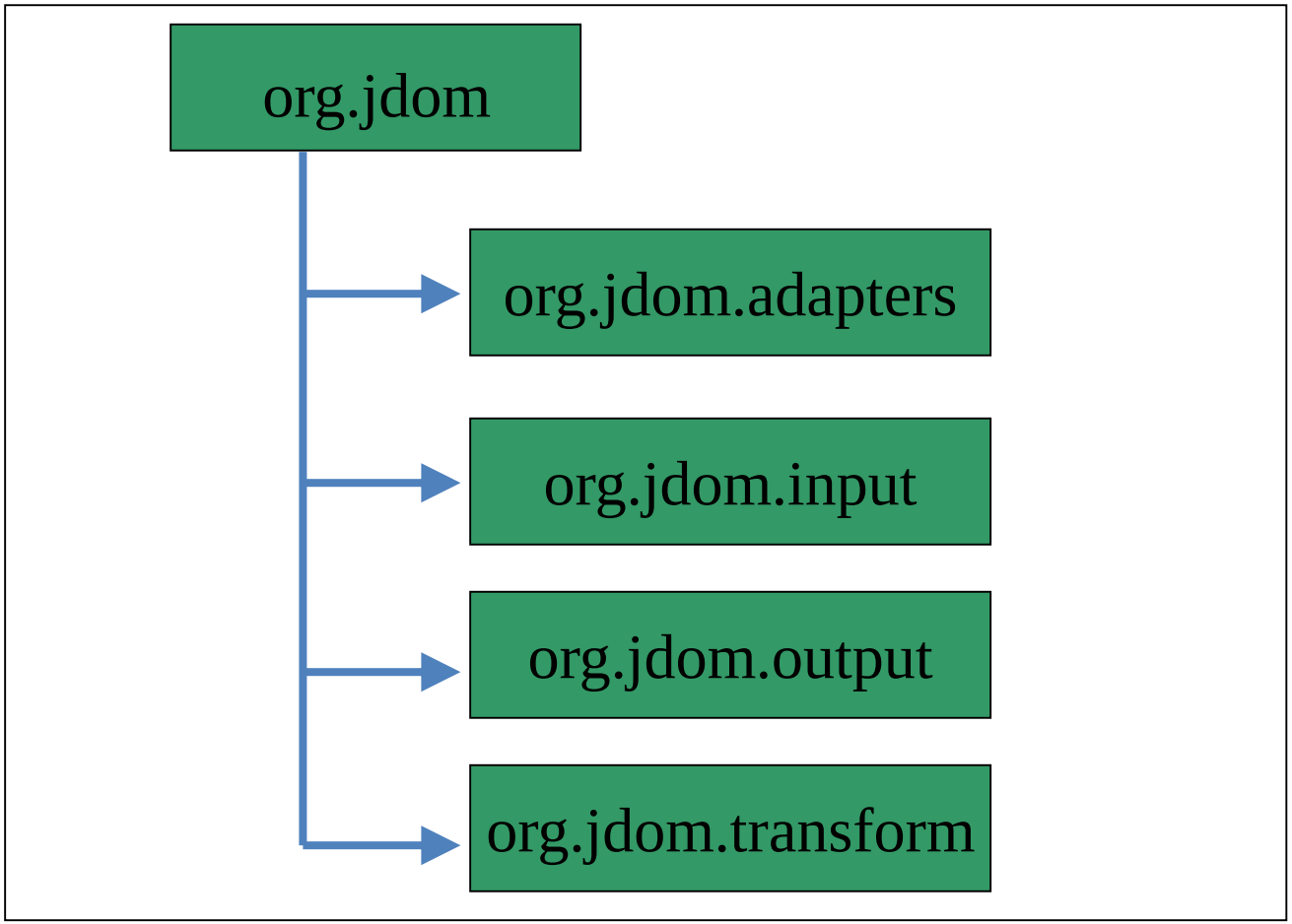
- TransformerFactory
- Transformer
- Packages:
  - javax.xml.transform
  - javax.xml.transform.dom
  - javax.xml.transform.dom
  - javax.xml.transform.stream

JDOM

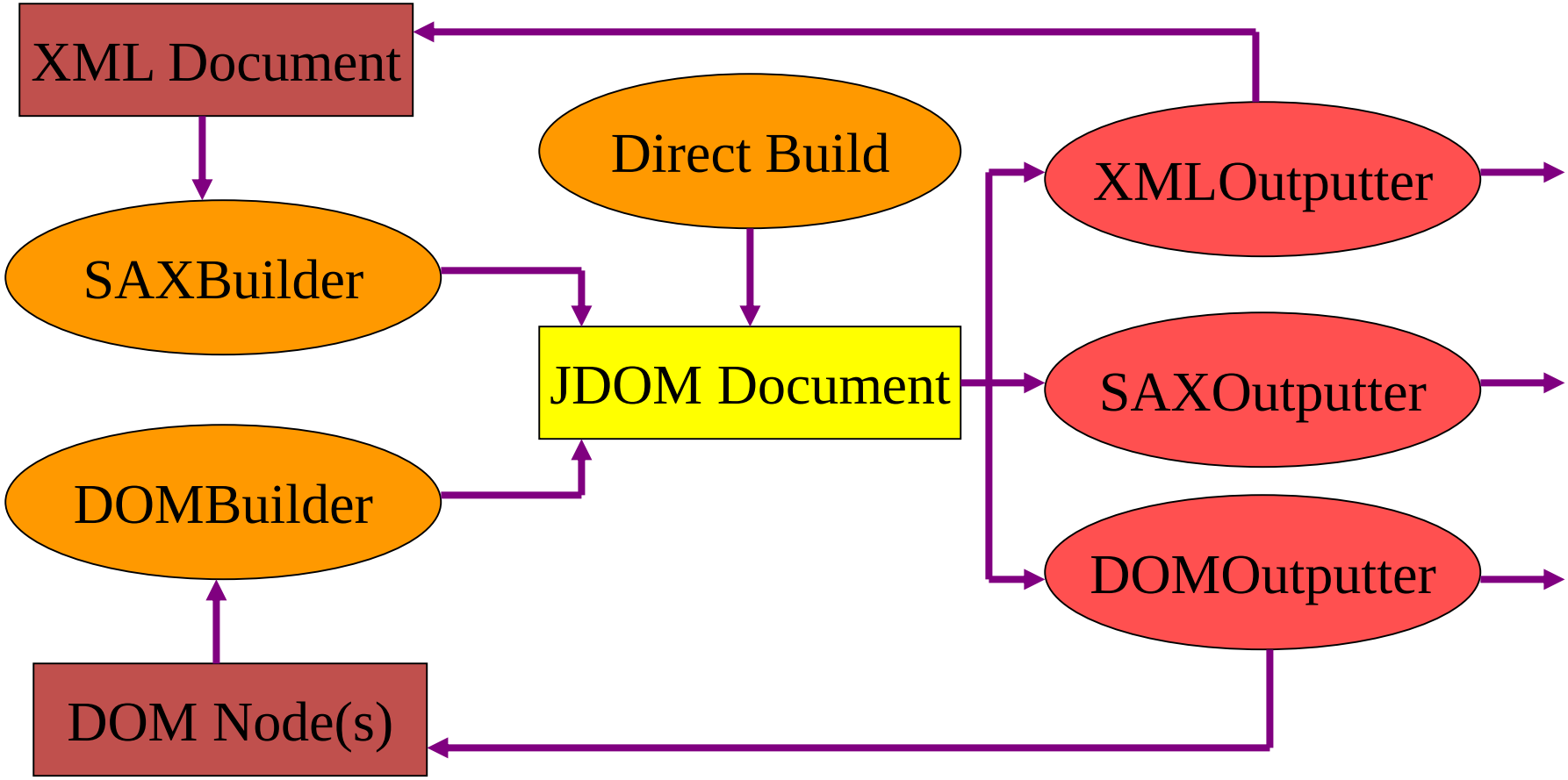
- In spite of its name, does not rely on DOM
  - But can easily be integrated with DOM and SAX
- Optimized for Java
  - Exploiting the overriding of methods, the Collections API, and reflection
- Oppositely to DOM, elements have contents, and not their text children

JDOM

- Consists of five packages:



JDOM – the process of parsing



JDOM vs. DOM

```
Document doc = new Document(
  new Element("rootElement")
  .setText("This is a root element"));
```

```
Document myDocument =
  new org.apache.xerces.dom.DocumentImpl();
Element root =
  myDocument.createElement("myRootElement");
Text text =
  myDocument.createText(
  "This is a root element");

root.appendChild(text);
myDocument.appendChild(root);
```

Reading mixed content with JDOM

```
List mixedContent = table.getMixedContent();
Iterator i = mixedContent.iterator();
while (i.hasNext()) {
  Object o = i.next();
  if (o instanceof Comment) {
    // Comment has a toString()
    out.println("Comment: " + o);
  }
  else if (o instanceof String) {
    out.println("String: " + o);
  }
  else if (o instanceof Element) {
    out.println("Element: " +
                ((Element)o).getName());
  }
  // etc
}
```