

# XML

---

<subtitle>DOM Parsers</subtitle>

<author>Prof. Dr. Christian Pape</author>

<term>Summer 2005</term>

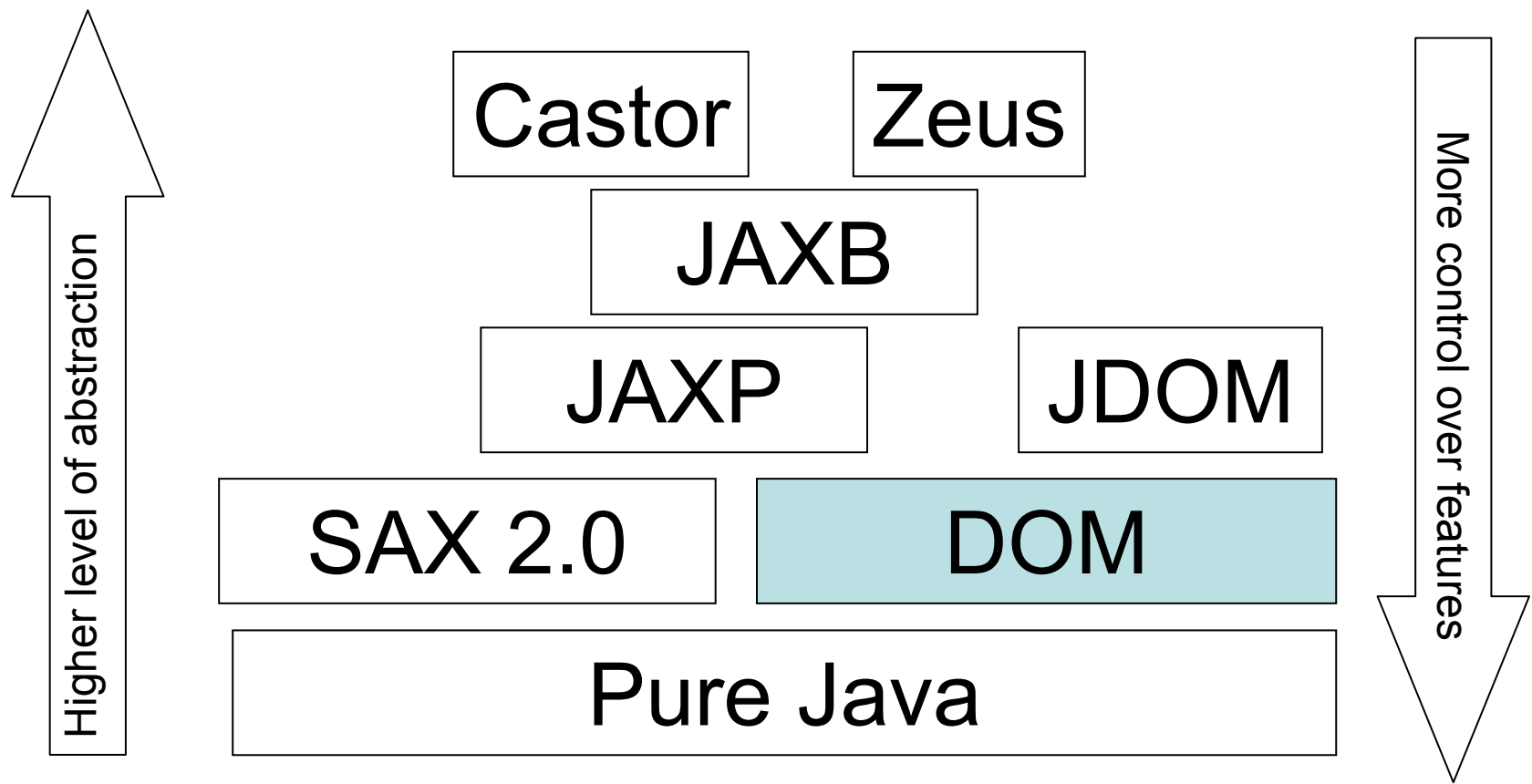
# Content

---

- Overview
- Java DOM API
  - API parts for read access
  - Example (reading documents)
  - Write access

# Overview

---



# Overview

---

- Document Objekt Model (DOM)
  - Cross platform standard of the W3C to represent content and model of documents
  - Independent of programming languages
  - DOM specifies content with a tree format and certain node types (document, element, ...)
  - DOM used for HTML, JavaScript, too
  - Different levels of DOM exists
    - Level 1 basic types and modules
    - Level 2 adds modules for HTML , XML, and CSS
    - Level 3 (upcoming) adds - for instance - XML validation handlers
- DOM Parser
  - Language specific implementation of DOM
  - Read XML documents: Navigate through the document from node to node with getter Methods
  - Create XML documents: Create new nodes and insert them after / before existing nodes
  - No write support at level 1 and 2 (level 3 only)

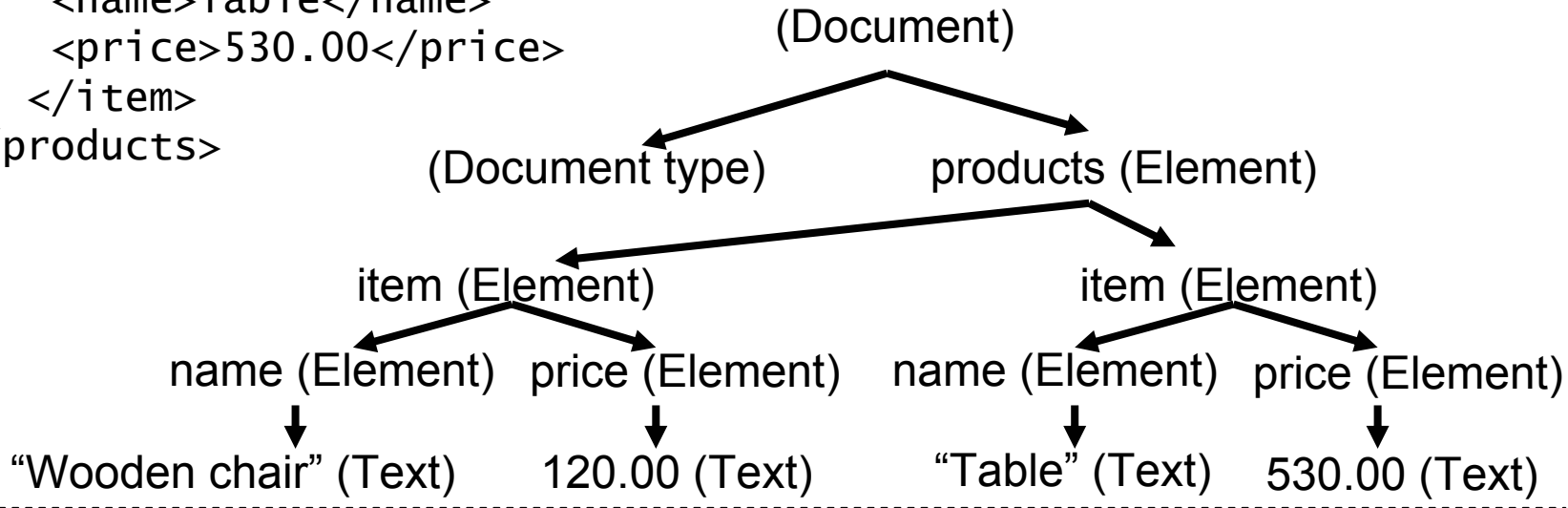
# Overview

```

<products>
  <item>
    <name>Wooden chair</name>
    <price>120.00</price>
  </item>
  <item>
    <name>Table</name>
    <price>530.00</price>
  </item>
</products>

```

- DOM Tree example
  - node types in parenthesis
- DOM Implementation
  - Let your navigate the tree



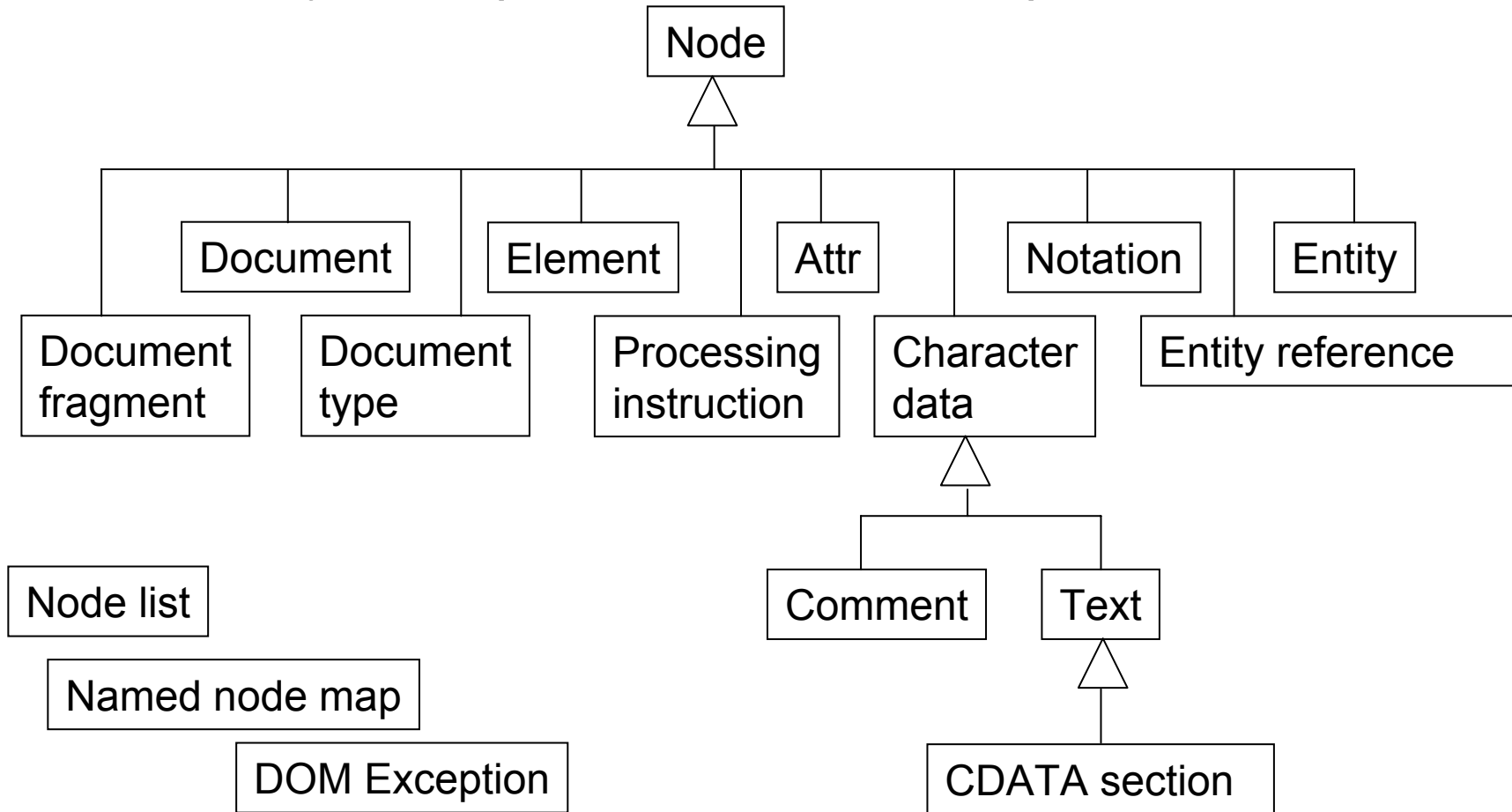
Navigation with JavaScript:

```
document.elements[0].elements[1].elements[1].text
```



# Overview

- DOM Types (inheritance tree)



# Content

---

- Overview
- **Java DOM API**
  - API parts for read access
  - Example (reading documents)

# Overview

---

- **Java DOM API**
  - Provided by W3C (not Sun)
  - Java Implementation of the DOM
  - Like SAX DOM is specified by a collection of Java interfaces (e.g. Node, Document, Element, Text)
  - Package name of API: org.w3c.dom
  - Like SAX different implementation of the Java DOM API exists (e.g. Xerces)
- **Examples with Xerces**
  - DOMParser main class
  - Read in document from InputSource (A SAX API)
  - Gives access to main object of the DOM API ( Document )

## **DOMParser**

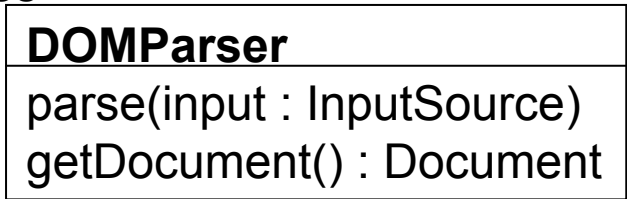
```
parse()  
getDocument() : Document
```

## **InputSource**

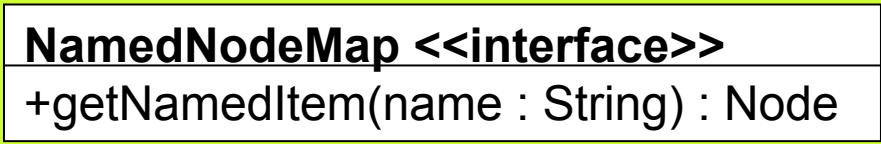
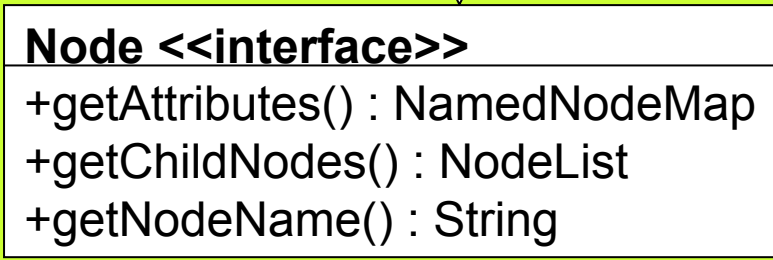
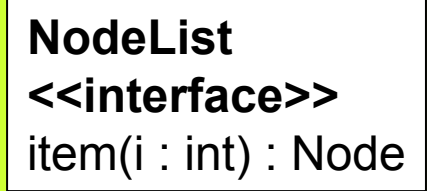
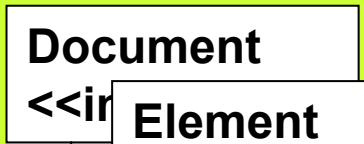
Some non DOM API interfaces  
Used by Xerces

# Java DOM API

Xerces



## DOM API



# Java DOM API

---

```
Reader reader = new FileReader("products.xml");
InputStream inputStream = new InputStream(reader);
inputSource.setSystemId("product.dtd");
```

```
// unlike SAX there is no vendor independent parser interface
DOMParser domParser = new DOMParser();
domParser.parse(inputStream);
```

```
Document document = domParser.getDocument();
// Return reference to DTD (if any exists)
DocumentType doctype = document.getDoctype();
// Returns the root node of the document
Element element = document.getDocumentElement();
// Returns the text of the element (or null if no text exists)
System.out.println("root tag name is " + element.getTagName());
// gets the 2n item element
Node node = element.getElementsByTagName("item").item(1);
System.out.println("node name is " + node.getNodeName());
```

↓ vendor  
independent  
code

# Java DOM API

---

Interface	getNodeName()	getNodeValue()	Example
Attr	Attribute name	Attribute value	
CDATASection	"#cdata-section"	the content of the CDATA Section	„<hi></hi>“
Comment	"#comment"	the content of the comment	„ a comment“
Document	"#document"	null	
DocumentType	Document Type name	null	„products“
Element	Tag name	null	„item“
Entity	Entity name	null	„auml“
EntityReference	Name of entity referenced	null	„ä“
Text	"#text"	the content of the text node	„523.00“

# Content

---

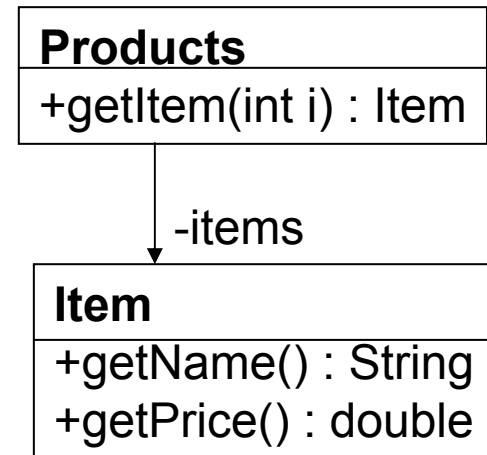
- Overview
- Java DOM API
  - API parts for read access
  - Example (reading documents)

# Example

---

- Mapping Object to Elements
  - For each element provide a Java class that wraps the content of the element
  - Getter methods to return content
  - Constructor that initializes object from a given DOM Element
- Advantage
  - Use Java Objects to access XML content
  - Similar to Castor classes

```
<products>
  <item>
    <name>wooden chair</name>
    <price>120.00</price>
  </item>
  <item>
    <name>Table</name>
    <price>530.00</price>
  </item>
</products>
```



# Example

---

```
public Products(Element products) {
    this.items = products.
        getElementsByTagName("item");
}
```

```
public Item getItem(int i) {
    return new Item( (Element)
        items.item(i) );
}
```

## Products

-items : NodeList

+Products(Element products)

+getItem(int i) : Item

## Item

-name : String

-price : double

+Item(Element item)

+getName() : String

+getPrice() : double

- Mapping Object to Elements
  - Convenience method `getElementsByTagName(String)`
  - Relation items implemented with `NodeList` of parsed items
  - Getter creates new `Item`
  - Item creation and initialization in `Item(Element)` Constructor

# Example

- All initialization in Item(Element)
  - Getter methods return attribute value

## Products

-items : NodeList

+Products(Element products)

+getItem(int i) : Item

```
public Item(Element item) {
    Element ne; // name element
    Element pe; // price element
    ne = (Element)item.
        getElementsByTagName("name").item(0);
    pe = (Element)item.
        getElementsByTagName("price").item(0);
    name = ne.getFirstChild().getNodeValue();
    price = Double.parseDouble(
        pe.getFirstChild().getNodeValue() );
}
```

## Item

-name : String

-price : double

+Item(Element item)

+getName() : String

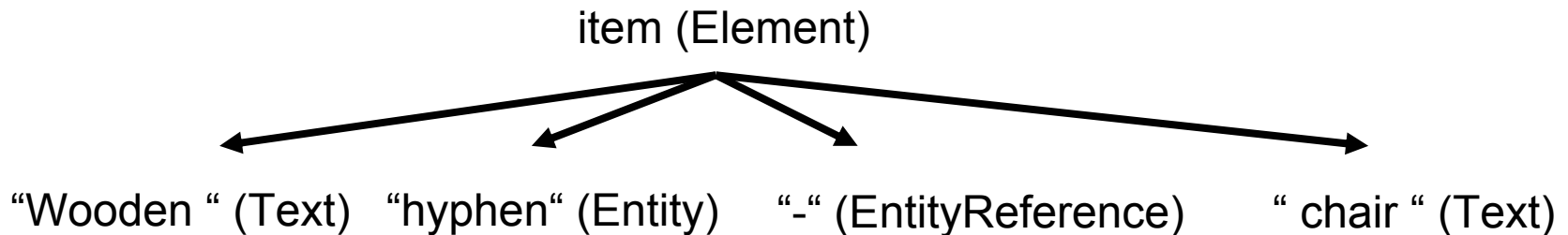
+getPrice() : double

# Example

---

- Solution not correct, yet
  - Consider text with entity references or CDATA section
  - Comments may occur in Text
  - Text content of element scattered over multiple text, CDATA, comment, entity, or entity references nodes
- No? convenience method exists to retrieve parsed character data

```
<products>
  <item>
    <name>wooden &hyphen; chair</name>
    <price>120.00</price>
  </item>
</products>
```



# Example

---

- Write own method to retrieve parsed character data of an element with text content
  - Utility class for DOM (only static methods)

```
public class DOMUtility {
    public static String getText(Element element) {
        StringBuffer pcdData = new StringBuffer();
        NodeList nodeList = element.getChildNodes();

        for (int i=0; i < nodeList.getLength(); i++) {
            Node node = nodeList.item(i);
            if (node instanceof CharacterData
                && ! ( node instanceof Comment
                    || node instanceof EntityReference )) {
                pcdData.append(node.getNodeValue());
            }
        }
        return pcdData.toString();
    }
}
```

# Example

---

- All initialization in Item(Element)
  - Using utility method

## Products

-items : NodeList

+Products(Element products)

+getItem(int i) : Item

```
public Item(Element item) {
    Element ne; // name element
    Element pe; // price element
    ne = (Element)item.
        getElementsByTagName("name").item(0);
    pe = (Element)item.
        getElementsByTagName("price").item(0);
    name = DOMUtility.getText(ne);
    price = Double.parseDouble(
        pe.getFirstChild().getNodeValue() );
}
```

## Item

-name : String

-price : double

+Item(Element item)

+getName() : String

+getPrice() : double

# Content

---

- Overview
- Java DOM API
  - API parts for read access
  - Example (reading documents)

# DOM and SAX Comparison

---

	DOM	SAX
Developers Productivity	Medium	Low (depending on task) (low level API)
Main memory use	At least $O(n)$ main memory consumption if $n$ is length of XML document	Depends on implementation, but need not larger than deepest nested level (practically constant)
Performance	Could be slow when parsing large documents	Can be fast (depends on developers code)
When to use	Creation of XML documents Access medium large pieces of XML data	Fast XML processing Simple task