Object Oriented Programming

Extensible markup language

Definition (1)

- The format of a document is performed according to two methods:

1. **Embed**: Formatting is absorbed internally by the tool by special commands (via keyboard or menus) and one only see the end result.
   Referred as **What You See is What You Get**
   - Examples: Microsoft Word.
   - Advantages:
     - Simplicity.
     - Avoids intermediate processing.

Definition (2)

2. **Annotation**: marks are visible in the document being processed by a tool to generate the presentation.
   Examples: LaTeX, HTML, XML.
   - Advantages:
     - Flexibility.
     - Independent from the hardware and display device.
     - Ease the development of tools to process the documents.
     - Gives more freedom to the user to establish their structure and formatting settings.

XML – introduction (1)

- The **XML** (Extensible Markup Language) expresses information:
  - **structured**, through annotations.
  - with **semantic**, through attributes.
- Documents in XML are stored in files with extension **.xml**
- Defined in 1996 to the W3C - http://www.w3.org/XML/
- Related technologies:
  - **Parse: SAX, DOM (both available in J2SE)**
  - Transforming documents in XML to XML: XSLT (Extensible Stylesheet Language)
  - Formatting definition: CSS (Cascade Style Sheets)
  - *…*

XML – introduction (2)

- Advantages:
  - Easy to develop tools to process documents.
  - Often used to transport information between different applications.
- Disadvantages:
  - Hardly readable.
  - Significantly increases the size of documents.
- Supported by popular browsers:
  - IE 5.0+
  - Netscape 6.0+

XML – introduction (3)

- An **element** is a part of a document, which is delimited by the marks `<mark>` e `</mark>`.
  - `<course>`OOP`</course>`
  - `<empty/>` for null elements (equivalent to `<empty></empty>`)
  - The XML marks are case-sensitive: `<Course>`OOP`</course>` is an incorrect element!

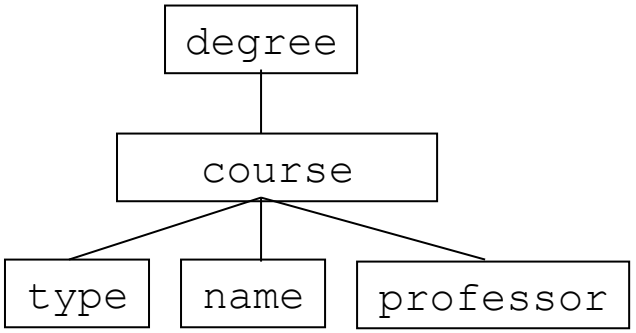
XML – introduction (4)

- An **attribute** provides additional information about the element, with the following format **name="value"**
  - An element may contain more than one element, separated by spaces:

    ```
    <degree univ="IST" msc="MEEC" >
    ```
- Element may contain **nested** elements, but **NOT intertwined**!
  - Nested (correct): <A> <B></B> </A>
  - Intertwine (incorrect): <A> <B> </A> </B>
- A XML document is a tree of elements.

XML – introduction (5)

```
<degree univ="IST" msc="MEEC">
   <course>
      <type>MTP</type>
      <name semester="winter">Algorithms and Data Structures</name>
      <professor>Prof. Mário Rui</professor>
   </course>
   <course>
      <type>MTP</type>
      <name semester= "spring">Object Oriented Programming</name>
      <professor>Prof. Sara Lopes</professor>
   </course>
   <course>
      <type>Thesis</type>
   </course>
</degree>
```



XML – introduction (6)

- **Comments** are delimited by <!-- and -->
- P**redefined entities** are XML markup that authors use to represent characters that would otherwise be interpreted as having a special meaning, such as a start-tag or an entity reference.

Character	Substitution
<	<
>	>
&	&
‘	'
“	"

XML – introduction (7)

A data is an element or an attribute?

- Elements
  - Data structures (attributes are only represented by strings).
  - Data for the user.
- Attributes
  - Data with few alternatives, rarely updated.
  - Data for processing XML.

XML – introduction (8)

- **XML is not "evolved HTML"!!!**
  - XML is developed for **describing** information.
  - HTML is developed for **presenting** information.

XML – document format (1)

- All XML documents have the following format
  `[ prolog ] element`

1. The prolog contains:
   a) `<?xml ... ?>` XML document declaration (mandatory) with attributes:
      - `version` – XML version (mandatory, current vs is 1.0)
      - `encoding` – charset used (by default, UTF-8)
      - `standalone` – independent of external definitions (by default, yes)

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

XML – document format (2)

b) Document type declaration `<!DOCTYPE`, containing:
   1. Root mark.
   2. Reference to the document that defines the syntax of the elements and their attributes, for XML document validation. (note: see DTD section)
   3. Declaration of entities `<!ENTITY...>`, delimited by `[ ]`
   4. Close `>`

2. The XML document contains exactly one **root** element which contains other nested elements (in the form of a tree).

XML – entity (1)

- An **entity** is declared as

  ```
  <!ENTITY name "text">
  ```

  (note: the text maybe an element)
- An entity is referred by `&name;`

XML – entity (2)

```
<?xml version="1.0"?>
<!DOCTYPE mensagem
[
  <!ENTITY sign "Prof. Mário Rui">
  <!ENTITY phone "<phone>2398</phone>">
]
>
<message>
  <hello>Caro colega!</hello>
  <body>Peço que me ajudes na vigilância
  do exame no dia 28 de Janeiro, às
  9:00, no Salão Nobre. Dá uma apitadela
  para o telefone &phone;.
  Saudações académicas.
  &sign;
  </boby>
</message>
```

Documento com entidades resolvidas:

```
<?xml version="1.0"?>
<!DOCTYPE message>
<message>
  <hello>Caro colega!</hello>
  <corpo>Peço que me ajudes na
  vigilância do exame no dia 28 de
  Janeiro, às 9:00, no Salão Nobre. Dá
  uma apitadela para o telefone
  <phone>2398</phone>.
  Saudações académicas.
  Prof. Mário Rui
  </body>
</message>
```

DTD – introduction (1)

- The **DTD** (Document Type Definition) defines the XML document structure with a list of legal elements and attributes.
- In XML documents, the DTD is given in the prolog, inside the DOCTYPE.
  - If the document obey to the DTD rules, is considered **valid**.
    (validator: **http://validator.w3.org/**)
  - If no DTD is given, the document can only be considered **well formed**.

DTD – introduction (2)

1. The DTD is given in the XML document in the DOCTYPE after the root element, in one of the two forms:
   - A. `PUBLIC`: general accessible object
     - **"(+|-) // responsible // type-version // language"**
       - **+** indicates normalized DTD, **-** non-normalized
       - Language is given with two chars (e.g. : **EN**-English)
     - `URL`: indicated between quotation marks (http://…, or file:///drive:/…)
   - B. `SYSTEM`: resource (file or URL)
     (note: directories separated by /, even in Windows)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!DOCTYPE root SYSTEM "file:///C:/Documents and Settings/ASC/exemple.dtd">
```

DTD – introduction (3)

2. The DTD contains a sequence of directives defining:
   1. **Elements**

      ```
      <!ELEMENT idElem category>
      <!ELEMENT idElem (content)>
      ```

   2. **Attributes**

      ```
      <!ATTLIST idElem idAttr typo [defaultVal]>
      ```

DTD – elements (1)

Structure of the elements:

- `EMPTY` – necessarily empty.
- `ANY` – any combination of elements.
- `(#PCDATA)` Any sequence of characters.
- `(ListaElem)` Sequence of elements and characters, separated by commas, with operators:
  - `?` Optional.
  - `|` Alternative.
  - `*` Zero, one or more.
  - `+` Uma, or more.

DTD – elements (2)

```
<!ELEMENT article (title, subtitle?, author+, (section, txt)+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT txt (#PCDATA)>
```

- An article has a title, optionally followed by the subtitle, followed by an author (at least) and finally followed by a sequence of (at least) one section or text.
- The title is a string.
- The txt is a string.

DTD – elements (3)

```
<article>
   <title>Distributed Resolution of Feature Interactions for
   Internet Applications</title>
   <author>Mário Rui, Luigi Logrippo</author>
   <section>Resumo</section>
      <txt>Propõe-se a criação de conselheiros em cada nó, que
   propõem a resolução de interacções não desejadas entre serviços
   com base num conjunto de fórmulas deônticas.</txt>
   <section>Introdução</section>
      <txt>...</txt>
   <section>Arquitectura</section>
      <txt></txt>
   <section>Conclusões e trabahos futuros</section>
      <txt></txt>
</article>
```

DTD – attributes (1)

Type of the attributes:

Type	Possible values
`CDATA`	Any string
`(en1|en2|…)`	An enumerated data
`ID`	Unique identifier
`IDREF`	Identifier of other element
`IDREFS`	List of other identifies
`NMTOKEN`	Valid XML name
`NMTOKENS`	List of valid XML names
`ENTITY`	External entity
`ENTITIES`	List of entities
`NOTATION`	Notation

DTD – attributes (2)

Default values: the default value might be a literal, or a #ccc class.

Type	Meaning
`value`	Default value
`#REQUIRED`	Required attribute
`#IMPLIED`	Attribute not necessarily included
`#FIXED value`	Fixed attribute value

DTD

```
<!ELEMENT phone EMPTY>
<!ATTLIST phone num CDATA "0">
```

Valid XML

```
<phone/>                <! - number is 0 -->
<phone num="2398"/>     <! - number id 2398 -->
```

DTD – attributes (3)

DTD

```
<!ELEMENT person>
<!ATTLIST person nbMec #REQUIRED>
```

Valid XML

```
<person nbMec="2531">Mário Rui</person>
```

Invalid XML

```
<pessoa>Mário Rui</pessoa>
```

DTD

```
<!ELEMENT fac>
<!ATTLIST fac univ #FIXED "UTL">
```

Valid XML

```
<fac univ="UTL”>IST</fac>
```

Invalid XML

```
<fac>Instituto Superior Técnico</fac>
```

DTD

```
<!ELEMENT pay>
<!ATTLIST pay
     with (money|card)“money”>
```

XML válido

```
<pay>620</pag>                <! -- equiv -->
<pay with="money">62</pay> <! -- equiv -->
<pay with="card">1200</pay>
```

DTD – disadvantages

- The DTD file is not formatted in XML.
- All data (elements and attributes) are strings: one cannot force integers.
- Difficulty in expressing non-sorted subelements (important in DBs).
- `ID` and `IDREF` have no types.

Alternative solution: **XML Schema**.

Parsing XML documents (1)

- The XML processing is based on **parser** tools.
- There are 2 tools with distinct strategies:
  1. **SAX** – (*Simple API for XML*)
     It goes through the document and launches an event every time it finds an element.
  2. **DOM** – (*Document Object Model*)
     It builds a tree that represents the all document.

Parsing XML documents (2)

- Available tools:
  - **SAX**: SourceForge and **J2SE**
  - **DOM**: W3C and **J2SE**
  - **MSXML** (from Microsoft): embedded in IE (vs 5.0+)
  - **Mozilla** (da Netscape): embedded in Netscape (vs 6.0+)

Parsing XML documents (3)

- Advantages of SAX/DOM:
  - SAX
    - Consumes little memory.
    - Used in long documents, when one wants to access only a few elements.
  - DOM
    - Allows access to elements already processed.
    - Used if one needs to change the XML document.

Parsing XML documents (4)

- Analysis of a XML document follows the following steps:
  - Create an instance of the parser.
  - Pass the document to the created object.

Note: DOM implementations often resort to SAX to perform the initial parsing of the XML document.

Introduction to SAX (1)

- **SAX** (*Simple API for XML*) is a common interface to various parsers that recognize **serial** elements expressed in XML documents.
- Developed in 1998 by David Megginson, available to diverse languages:
  - Java (v2, disclosed in 2000, accessible in http://www.saxproject.org/ and included in J2SE)
  - Perl, C++, Python

Introduction to SAX (2)

- The SAX parser, when recognizes specific positions of the document invoke methods (operation called by launching an event).
- The four most important positions are:
  1. Begin of the document
  2. Begin of the element
  3. End of the element
  4. End of the document
- SAX may also identify the attributes of the elements.
- SAX throws exceptions on invalid documents.

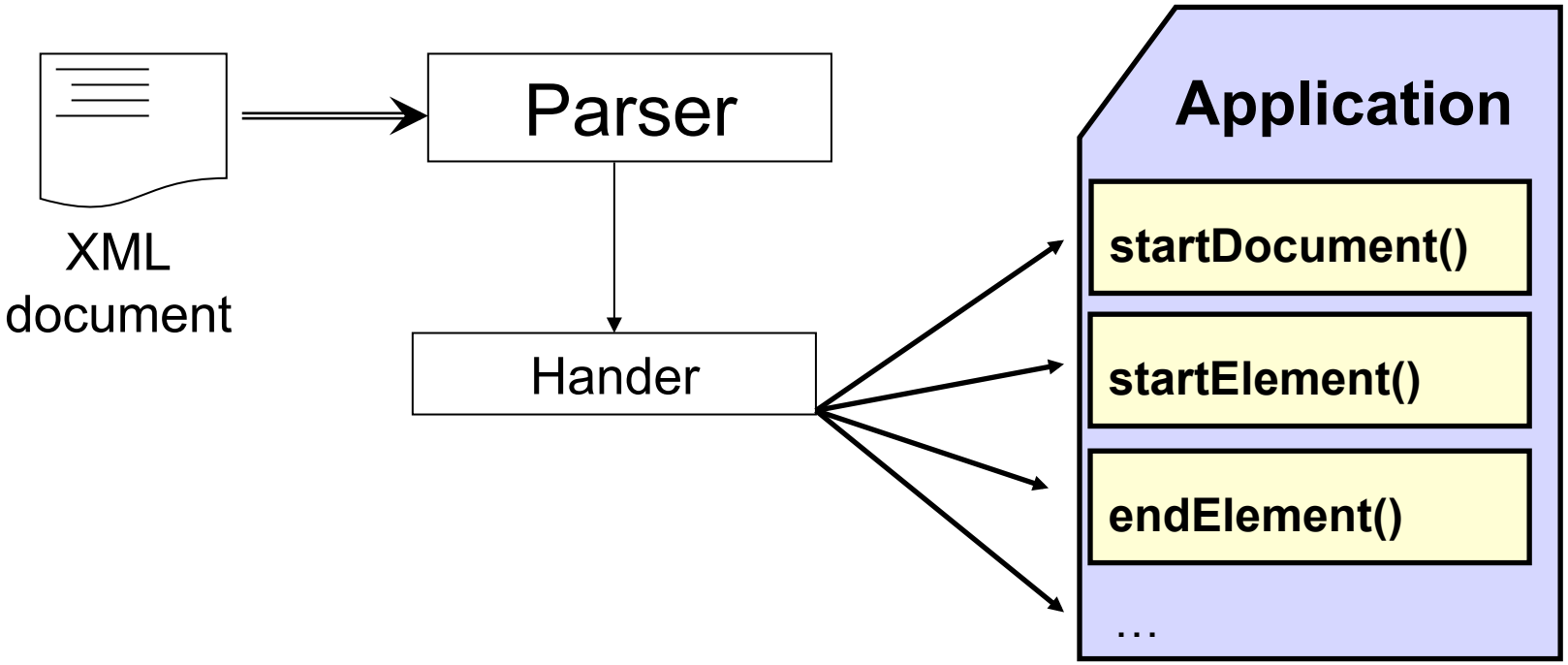
Introduction to SAX (3)

- SAX and DOM are APIs:
  - included in **JAXP** (*Java API for XML Processing*) da J2SE.
  - defined generically in the **javax.xml.parsers** package.
  - the basic APIs are defined in the **org.xml.sax** package.
- Classes to import:

```
import javax.xml.parsers.*;        // SAX and DOM parsers
import org.xml.sax.*;              // Generic API for SAX
import org.xml.sax.helpers.*;      // Handlers
```

Introduction to SAX (4)

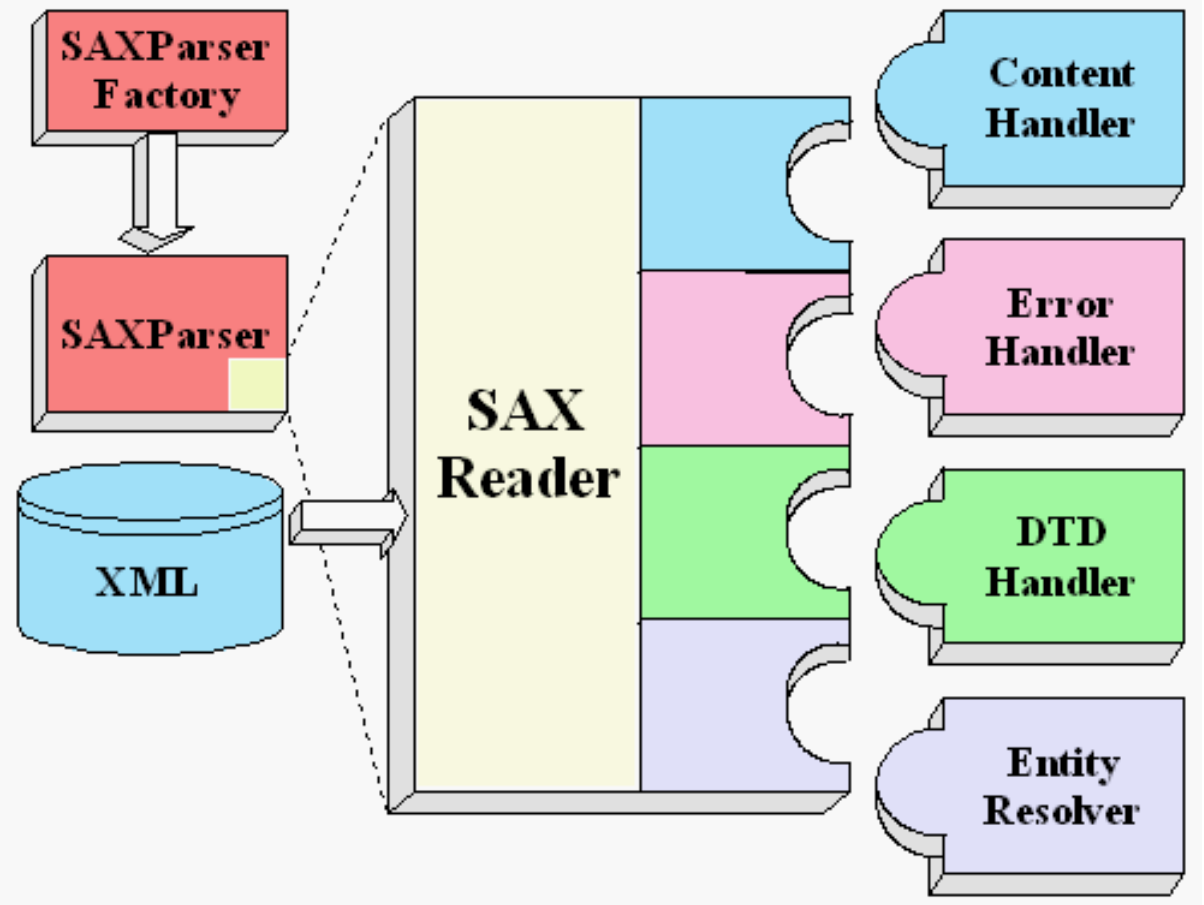
General architecture



Handlers (1)

- SAX works with 4 handlers defined by the interfaces:
  - `public interface ContentHandler`: notification of the events related to the XML document.
  - `public interface DTDHandler`: notification of the events related with the DTD.
  - `public interface EntityResolver`: for resolving external entities.
  - `public interface ErrorHandler`: for resolving errors.

Handlers (2)


SAXParser Factory
SAXParser
XML
SAX Reader
Content Handler
Error Handler
DTD Handler
Entity Resolver

Handlers (3)

- **`DefaultHandler`** is the base class of the SAX applications; it implements the `ContentHandler`.

```
public class DefaultHandler extends java.lang.Object
        implements EntityResolver, DTDHandler,
                   ContentHandler, ErrorHandler
```

- Usually the programmer extends `DefaultHandler`.

Content handler (1)

- `ContentHandler` methods that might be overridden by the user:
  - `public void startDocument();`
  - `public void endDocument();`

  // text within the element
  - `public void characters(char[] ch, int start, int length);`

  // mark given in the tag, attributes in atts
  - `public void startElement(String uri, String name, String tag, Attributes atts);`
  - `public void endElement(String uri, String name, String tag);`

Content handler (2)

```
public class Main extends DefaultHandler{
     static String fileName;
     public void startDocument(){
          System.out.println(”Beginning the parsing of"+ fileName);
     }
     public void endDocument(){
          System.out.println(“Parsing concluded");
     }
     public void startElement(String uri, String name,
                              String tag, Attributes atts){
          System.out.print("Element <" + tag + "> ");
     }
     public void characters(char[]ch,int start,int length){
          System.out.print(new String(ch,start,length));
     }
```

Content handler (3)

```
    public static void main(String[] argv) throws Exception {
        fileName = argv[0];

        // builds the SAX parser
        SAXParserFactory fact = SAXParserFactory.newInstance();
        SAXParser saxParser = fact.newSAXParser();

        // parse the XML document with this handler
        DefaultHandler handler = new Main();
        saxParser.parse(new File(fileName), handler);
    }
}
```

Content handler (4)

- Using the following input file:

```
<?xml version="1.0"?>
<Disciplina>
  <nome>Programacao por Objectos</nome>
  <aluno>
     <numero>34914</numero>
     <resultado>16</resultado>
  </aluno>
  <aluno>
     <numero>36731</numero>
     <resultado/>
  </aluno>
</Disciplina>
```

Content handler (5)

- The following output is obtained:

```
Inicio analise de C:\Users\PO\AnaliseDoc\test\Test.txt
Elemento <Disciplina>
  Elemento <nome> Programacao por Objectos
  Elemento <aluno>
     Elemento <numero> 34914
     Elemento <resultado> 16

  Elemento <aluno>
     Elemento <numero> 36731
     Elemento <resultado>

Concluida analise
```

Content handler (6)

- The `startElement` method usually contains code in the form:

```
if (tag.equals("object1")
     new Class-object1();
else if (tag.equals("objecto")
     new Class-objecto();
/*…*/
else
     System.err.println(“Erro");
```

Content handler (7)

- The elements are identified in serie and for each element the method `startElement` is invoked.
- If the programmer want to access freely the elements, they must be stored in a structure (array, ...).
- The DOM tool stores the elements in a tree, which can be navigated by the programmer!!

Content handler (8)

- In order to the parser check the validity of the document against the DTD is necessary to invoke the method `setValidating`.

```
SAXParserFactory fact = SAXParserFactory.newInstance();
fact.setValidating(true);
SAXParser saxParser = fact.newSAXParser();
```

Error Handler (1)

- During the analysis of documents expressed in XML several errors may occur:
  - Access to the document.
  - Not well-formed document.
  - Bad parser configuration.

Error Handler (2)

```
public static void main(String[] argv) {
   SAXParserFactory fact = SAXParserFactory.newInstance();
   try{
        SAXParser saxParser = fact.newSAXParser();
        DefaultHandler handler = new Main();
        saxParser.parse(new File(argv[0]), handler);
   } catch(IOException e) {
        System.err.println("IO error");
   } catch(SAXException e) {
        System.err.println("Parser error");
   } catch(ParserConfigurationException e) {
        System.err.println("Parser configuration error");
   }
}
```

Error Handler (3)

- The `ErrorHandler` interface define methods which are invoked whenever the parser detects an error in a document, following classes indicated by the W3C.
  - `public void warning(SAXParseException exception) throws SAXException`
    non-fatal error, **by default nothing is done**.
  - `public void error(SAXParseException exception) throws SAXException`
    serious error, **for instance, non-valid document; by default nothing is done**.
  - `public void fatal(SAXParseException exception) throws SAXException`
    fatal error, **for instance, bad-formatted document, which might abort the analysis**.

Error Handler (4)

- The `warning`, `error` and `fatal` errors are implemented in the `DefaultHandler`, but they can (and, typically, should) be redefined by the programmer.

Attributes access (1)

- The attributes are not reported by events.
- All attributes of an element are passed as the last parameter of the `startElement` method.
- The attributes are an unordered list (may not match the original document order) of `Attributes`.

Attributes access (2)

- Methods of the `Attributes` interface:
  - **`public int getLength()`**
    number of attributes in the list
  - **`public String getLocalName(int index)`**
    identifier of the attribute ate position `index`
  - **`public String getType(int index)`**
    type of the attribute in position `index` (`"CDATA"`, `"ID"`, `"IDREF"`, `"IDREFS"`, `"NMTOKEN"`, `"NMTOKENS"`, `"ENTITY"`, `"ENTITIES"` or `"NOTATION"`)
  - **`public String getValue(int index)`**
    value of the attribute in position `index`

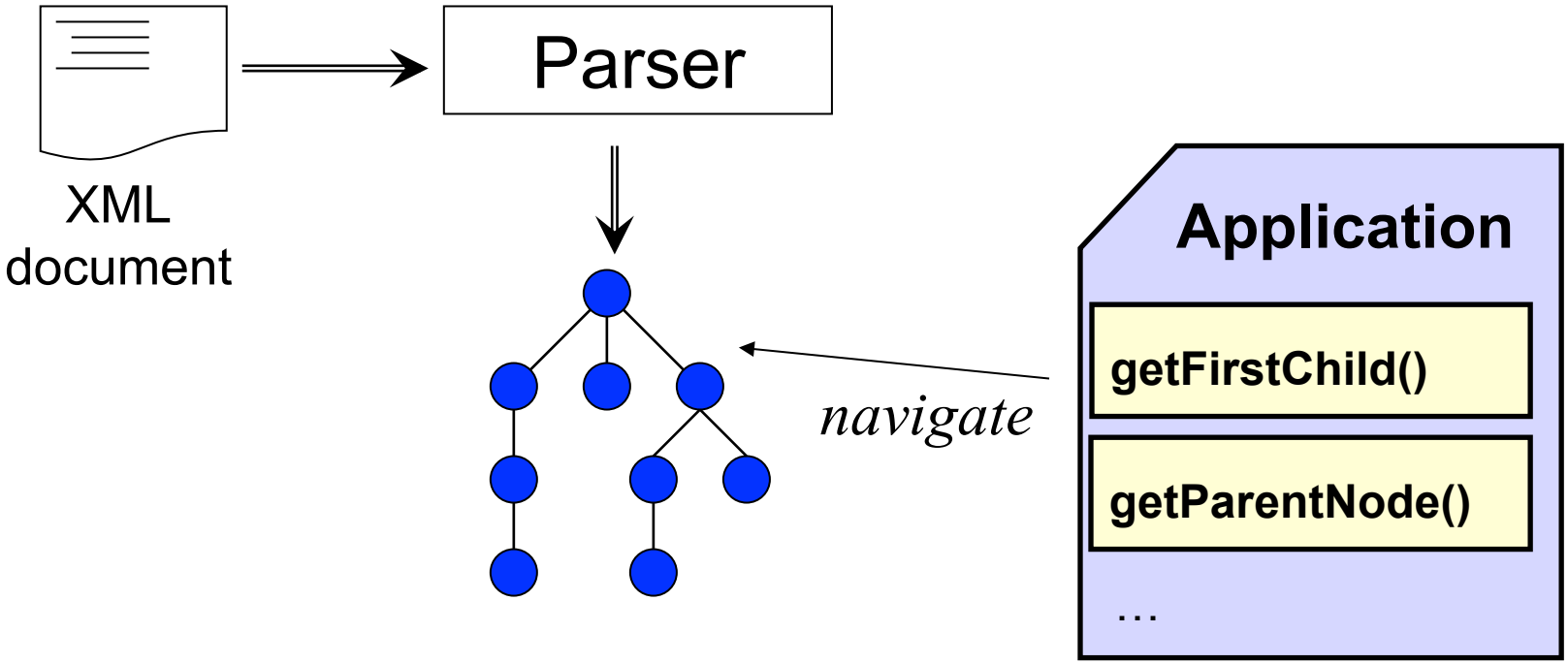
Introduction to DOM (1)

- **DOM** (*Document Object Model*) is a common interface to multiple parsers, which generates a tree formed by the elements expressed in XML documents.
- Classes to import:

```
import javax.xml.parsers.*;          // SAX and DOM parsers
import org.w3c.dom.*;                // Generic DOM API
```

Introduction to DOM (2)

General architecture



Introduction to DOM (3)

- The DOM parser returns a **Document**, object positioned at the root.

```
public static void main(String[] argv) throws Exception{
   // build a DOM parser
   DocumentBuilderFactory fact =
        DocumentBuilderFactory.newInstance();
   DocumentBuilder builder = fact.newDocumentBuilder();

   // obtains the document
   Document doc = builder.parse(new File(argv[0]));
   // obtains the root of the document
   Node node = doc.getDocumentElement();
}
```

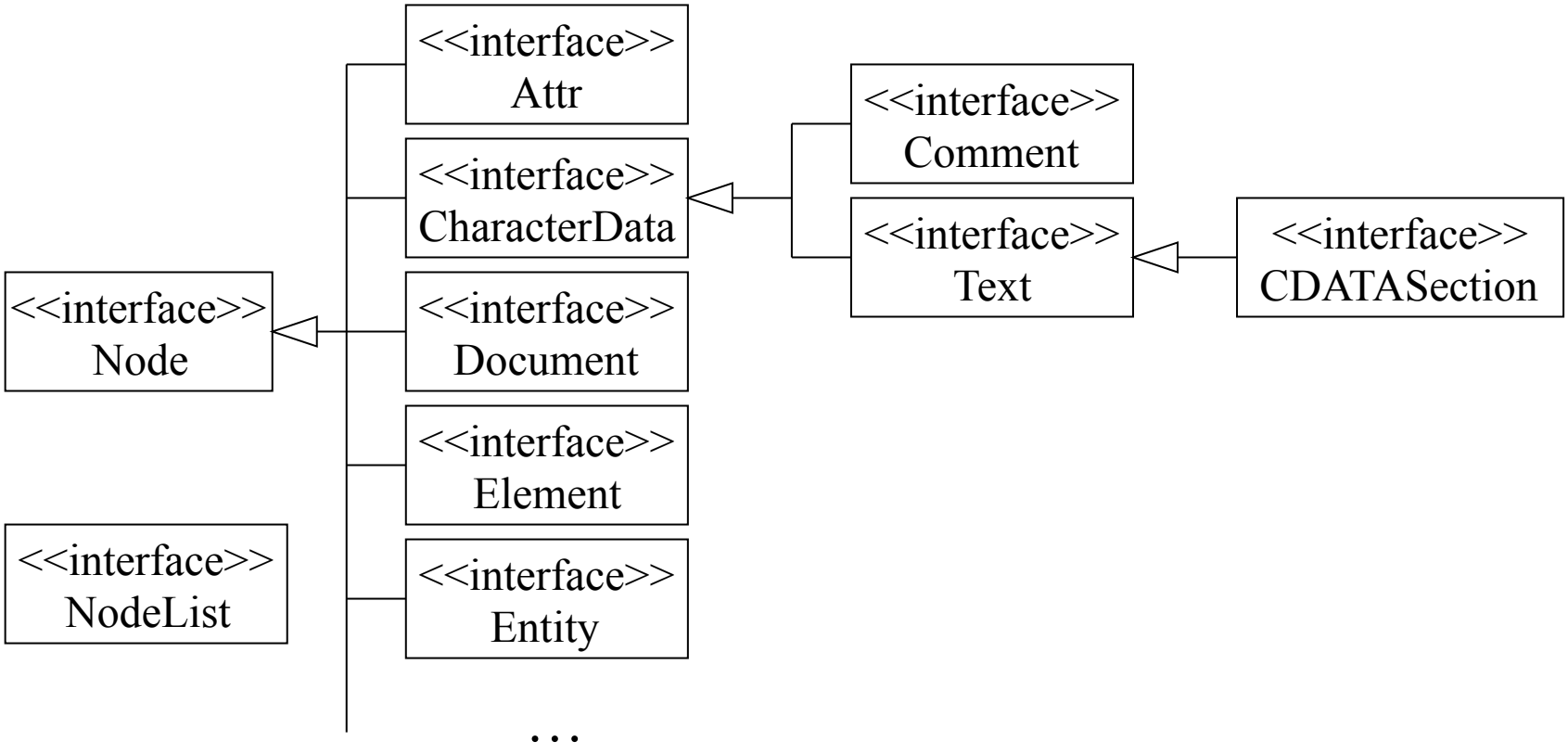
Node access (1)

- Constants: `Node` define 18 constants to determine the type of the nodes.

Constant	Type of the node
ATTRIBUTE_NODE	ATTR
CDATA_SECTION_NODE	CData
COMMENT_NODE	Comment
ELEMENT_NODE	Element
ENTITY_NODE	Entity
TEXT_NODE	Text

Node access (2)

- Hierarchy of the interfaces:



Node access (3)

- **Some methods of `Node`** (39 altogether):
  1. **Research on descendants:**
     - `boolean hasChildNodes()`: have descendants?
     - `NodeList getChildNodes()`: obtain list of descendant nodes.
  2. **Nodes' navigation:**
     - `Node getParentNode()`: obtain ascendant node.
     - `Node getFirstChild()`: obtain left-most descendant node.
     - `Node getLastChild()`: obtain right-most descendant node.
     - `Node getNextSibling()`: obtain node immediately of the right.

Node access (4)

3. **Nodes' information:**

- `short getNodeType()`: type of the node.
- `String getNodeValue()`: value of the node, which depends on its type.