Module 3

XML Processing

(XPath, XQuery, XUpdate)

Part 1: Overview

Position in Lecture “Storyline”

- We know how to write *data*
  - XML
  - Namespaces
- We know how to describe data = *metadata*
  - DTD
  - XML Schema
- Next step: how to process/manage data

Managing XML data

- Huge amounts of XML information, and growing
- We need to "*manage*" it
  - *Store it efficiently*
  - Verify the correctness
  - *Filter, search, select, join, aggregate*
  - *Create new data*
  - *Update it*
  - Take actions based on the existing data
- No conceptual organization like for relational databases (applications are too heterogeneous)

Frequent solutions to XML data management

1. Map it to *generic* programming APIs
2. *Manually* map it to *non-generic* APIs
3. *Automatically* map it to *non-generic* structures
4. Use *XML extensions* of existing languages
5. Shredding for relational stores (later)
6. *Native* XML processing through XSLT and XQuery

1. Mapping to generic structures

- Represent the data:
  - Original UNICODE form or
  - Some binary representation
- Store it:
  - Directly on a file system or
  - On a "transacted" file system (e.g. SleepyCat, or a relational database)
- Map the XML data to generic XML programmatic APIs
  - E.g. Dom, Sax, Stax (JSR 173), XMLReader
- Use the native programming languages (e.g. Java, C#) to manipulate the data
- Re-serialize it at the end

1. Manual mapping to generic structures (example)

```
<book>
        <author> .....
        </author>
        <title> .....
        </title>
        .....
</book>
```

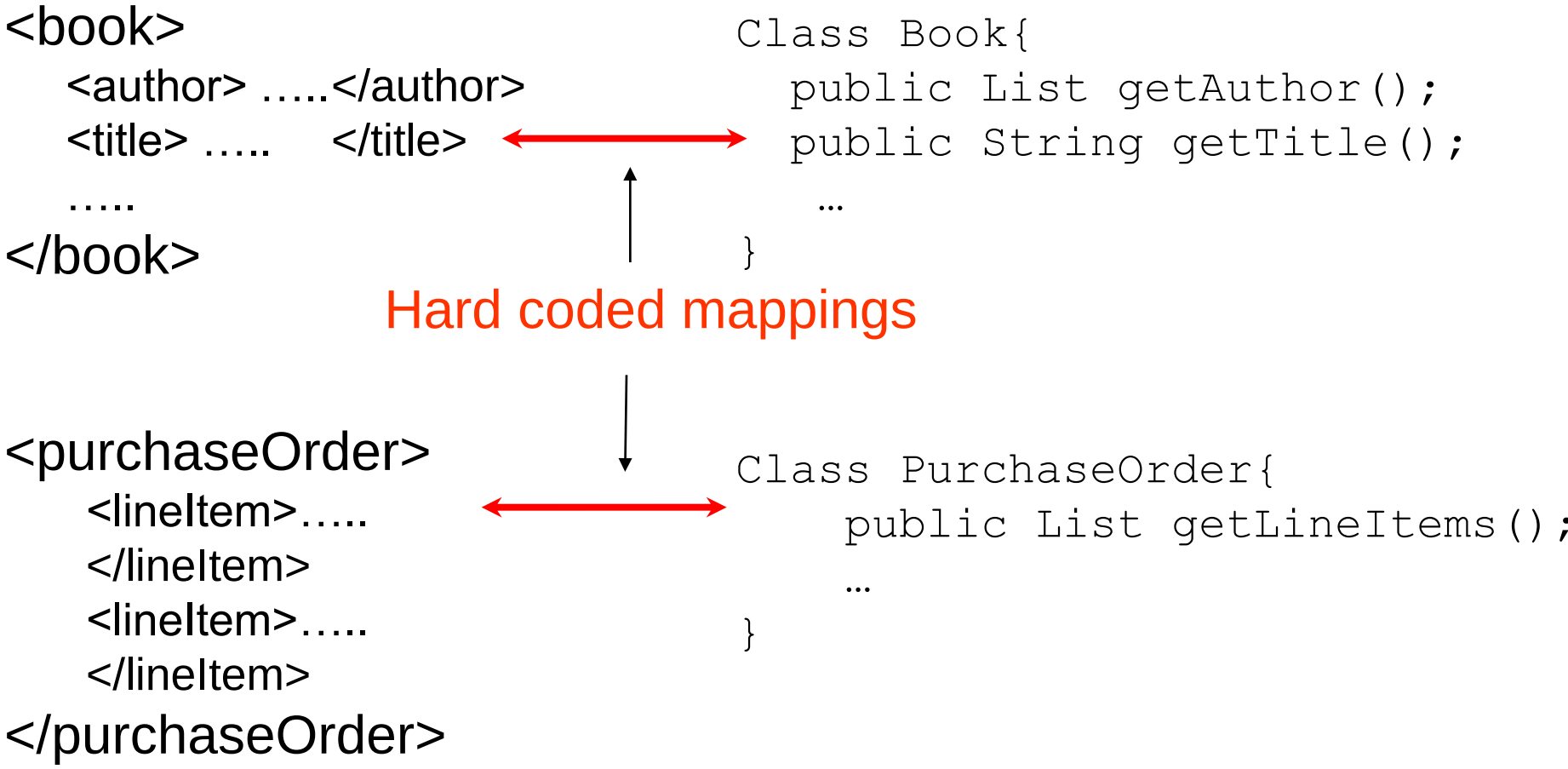
```
Class DomNode{

    public String getNodeName();
    public String getNodeValue();
    public void
       setNodeValue(nodeValue);
    public short getNodeType();

}
```

Hard coded mappings

2. Manual mapping to non-generic structures

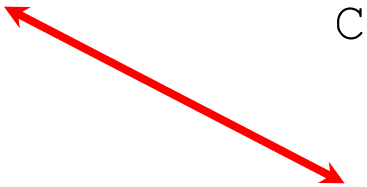


3. Automatic mapping to non-generic structures

```
<type name="book-type">
    <sequence>
       <attribute name="year" type="xs:integer">
       <element name="title" type="xs:string">
       <sequence minoccurs="0">
         <element name="author" type="xs:string>
       </sequence>
   </sequence>
</type>
<element name="book" type="book-type">
```

Automatic mapping

e.g.XMLBeans



```
Class Book-type{
    public integer getYear();
    public string getTitle();
    public List getAuthors();
    ……..}
```

4. XML extensions of existing procedural languages

- Examples:
  - C-omega/LINQ/.NET 3.0, ECMAscript/Javascript, PHP extensions, Python extensions, etc.
- Most of them define:
  - A way of importing XML data into their native type system
  - A rich API for XML data manipulation
  - A way of navigating/searching/querying the XML data via their extensions (XPath based or XPath inspired)

6. Native XML processing: XSLT and XQuery

- Most promising alternative for the future (***)
- The *only* alternative such that:
  - the data is modeled only once
  - is well integrated with XML Schema type system
  - it preserves the logical/physical data independence
  - the code deals with non-generic structures
- Data is stored:
  - in plain file systems *or* in sophisticated data stores
- Currently an incomplete solution: No procedural logic!
- Other disadvantages (hopefully temporary):
  - Perceived complexity
  - Perceived loss of performance