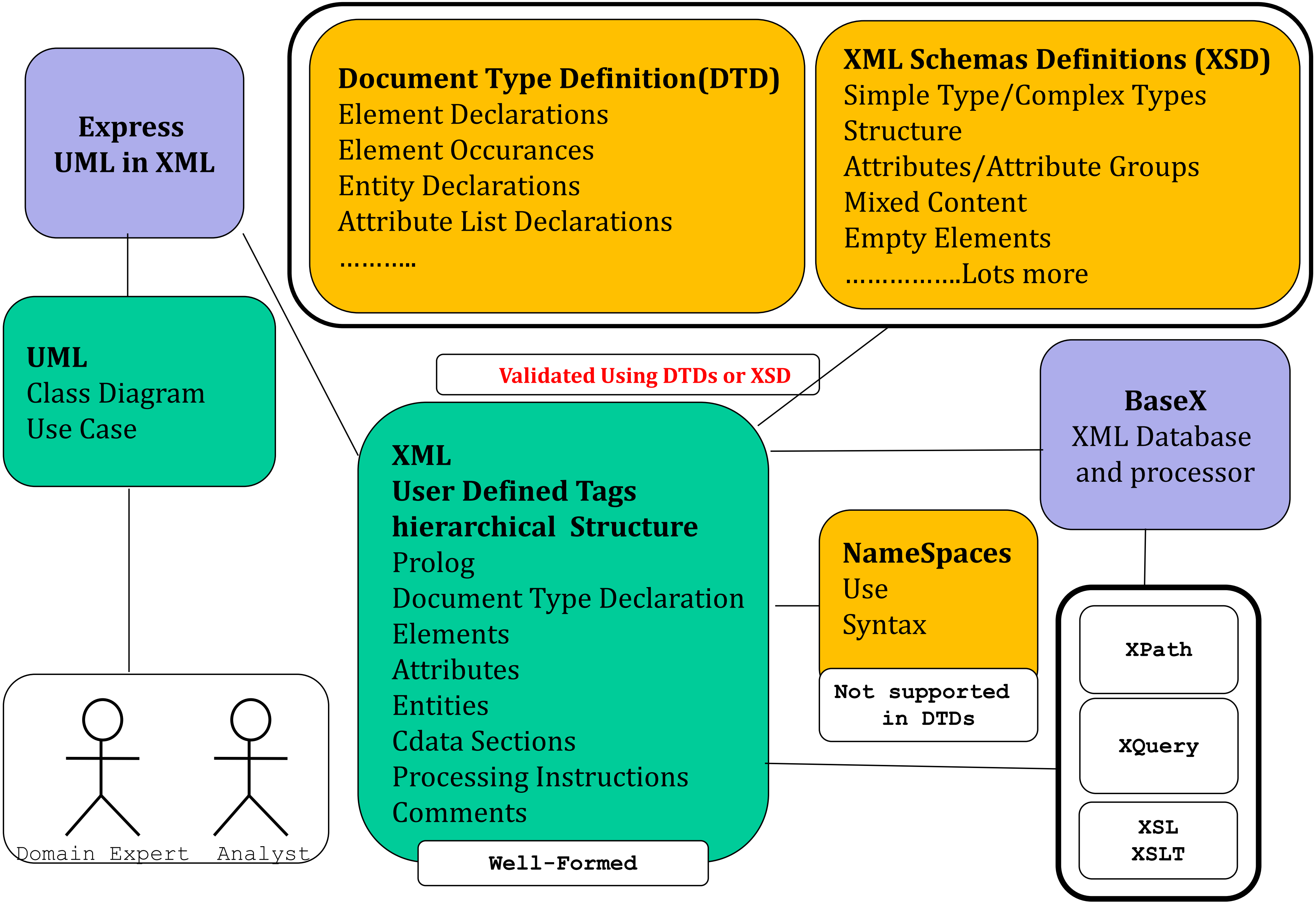


Express UML in XML

UML
Class Diagram
Use Case

Domain Expert
Analyst

Document Type Definition(DTD)
Element Declarations
Element Occurances
Entity Declarations
Attribute List Declarations
………..

XML Schemas Definitions (XSD)
Simple Type/Complex Types
Structure
Attributes/Attribute Groups
Mixed Content
Empty Elements
…………….Lots more

Validated Using DTDs or XSD

XML
User Defined Tags
hierarchical Structure
Prolog
Document Type Declaration
Elements
Attributes
Entities
Cdata Sections
Processing Instructions
Comments

Well-Formed

NameSpaces
Use
Syntax

Not supported in DTDs

BaseX
XML Database
and processor

XPath

XQuery

XSL
XSLT

Exercise 1- XML example to Hand Up

- Write down a “well formed” XML snippet, using elements and/or attributes, describing:
  - Your name (distinguishing first, middle, surname)
  - Student ID
  - Favourite music groups
  - County
  - Expected date of graduation

- **Well formed-** XML Declaration required, Exactly one root element, Empty elements are written in one of two ways:Closing tag or Special start tag, For non-empty elements, closing tags are required, Attribute values must always be quoted, Start tag must match closing tag (name & case),Correct nesting of elements

Sample XML to show Syntax

```
<?xml version="1.0‟?>
<!DOCTYPE catalog SYSTEM "books.dtd">
<catalog>
   <book id=’bk101’ type=’softback’>
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
<price>44.95</price>
      <publish_date>2000-10-
01</publish_date>
      <description>An in-depth look at
creating applications with XML.
</description>
</book>
<book id=’bk102’ type=’hardback’>
      <author nationality=‘irish’>Jenkins,
Fred</author>
      <title>XML Technology Guide</title>
          <price>50.00</price>
      <publish_date>2000-10-
01</publish_date>
      <description>An in-depth look at
using XML technologies.</description>
          <stocked_by>Easons</stocked_by>
          <stocked_by>Amazon</stocked_by>
   </book >
</catalog>
```

Solution Exercise 1

```
<?xml version="1.0" ?>

<student_info>
    <student_name student-id=“1234 >
        <firstname> Declan </firstname>
        <surname>O’Sullivan</surname>
    </student_name>
    <fav_music>
            <band> U2 </band>
            <band> beatles </band>
    </fav_music>
</student_info>
```

Exercise 2- Suggest a DTD

```
<?xml version="1.0“?>
<!DOCTYPE catalog SYSTEM "books.dtd">
<catalog>
   <book id=’bk101’ type=’softback’>
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
<price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating
applications with XML.
</description>
</book>
<book id=’bk102’ type=’hardback’>
      <author nationality=‘irish’>Jenkins,
Fred</author>
      <title>XML Technology Guide</title>
         <price>50.00</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at using XML
technologies.</description>
         <stocked_by>Easons</stocked_by>
         <stocked_by>Amazon</stocked_by>
   </book >
</catalog>
```

EXAMPLE DTD to show SYNTAX

```
<!DOCTYPE NEWSPAPER [

<!ELEMENT NEWSPAPER (ARTICLE+)>
<!ELEMENT ARTICLE
      (HEADLINE,BYLINE+,LEAD?,BODY,NOTES*)>
<!ELEMENT HEADLINE (#PCDATA)>
<!ELEMENT BYLINE (#PCDATA)>
<!ELEMENT LEAD (#PCDATA)>
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>

<!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>
<!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>
<!ATTLIST ARTICLE DATE CDATA #IMPLIED>
<!ATTLIST ARTICLE EDITION CDATA #IMPLIED>

<!ENTITY NEWSPAPER ”Trinity Times">
<!ENTITY PUBLISHER ”Trinity Press">
<!ENTITY COPYRIGHT "Copyright 1998 TCD Press">

]>
```

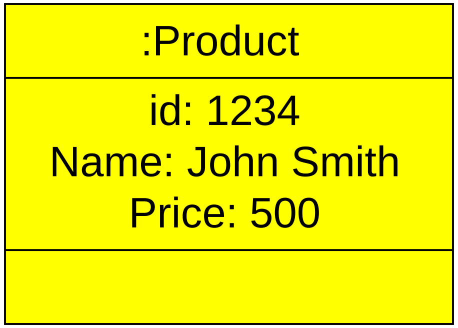
Solution Exercise 2

DTD

```
<!DOCTYPE catalog [
<!ELEMENT catalog    (book+) >
<!ELEMENT book       (author, title, genre?,
price, publish_date, description,stocked_by*) >
<!ATTLIST book   id ID #REQUIRED >
<!ATTLIST book   type (hardback|softback)
#REQUIRED >

<!ELEMENT author           (#PCDATA)   >
<!ATTLIST author nationality CDATA #IMPLIED >
    <!ELEMENT title            (#PCDATA)   >
    <!ELEMENT genre            (#PCDATA)   >
    <!ELEMENT price            (#PCDATA)   >
    <!ELEMENT publish_date     (#PCDATA)   >
    <!ELEMENT description      (#PCDATA)   >
<!ELEMENT stocked_by     (#PCDATA)   >
]>
```

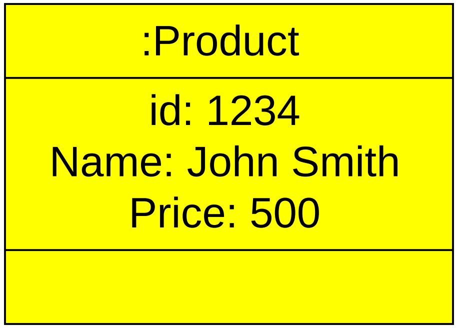
Exercise 3- Convert UML 2 XML



:Product

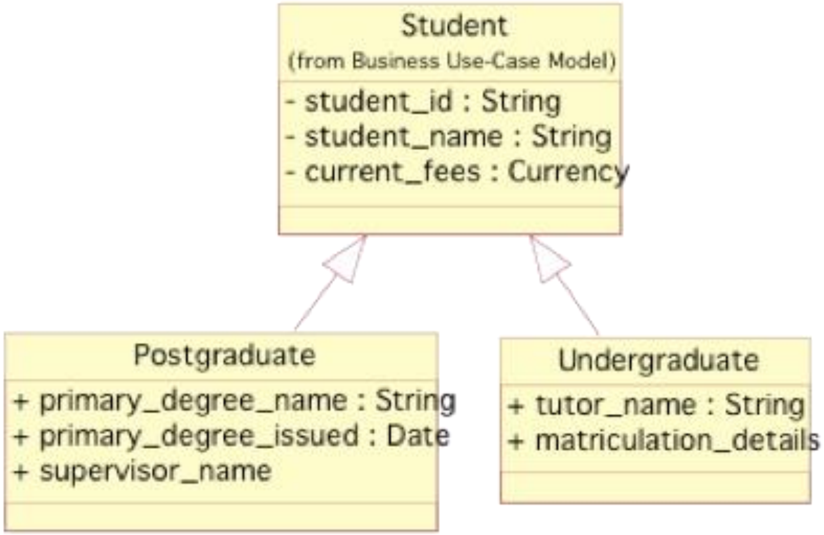
id: 1234
Name: John Smith
Price: 500

Solution Exercise 3



```
<Product>
   <Product. id> 1234 </Product.id>
   <Product.name> Lens </Product.name>
   <product.price> 500 </Product.price>
</Product>
```

Exercise 4 - Convert UML 2 XML



Student
(from Business Use-Case Model)
- student_id : String
- student_name : String
- current_fees : Currency

Postgraduate
+ primary_degree_name : String
+ primary_degree_issued : Date
+ supervisor_name

Undergraduate
+ tutor_name : String
+ matriculation_details

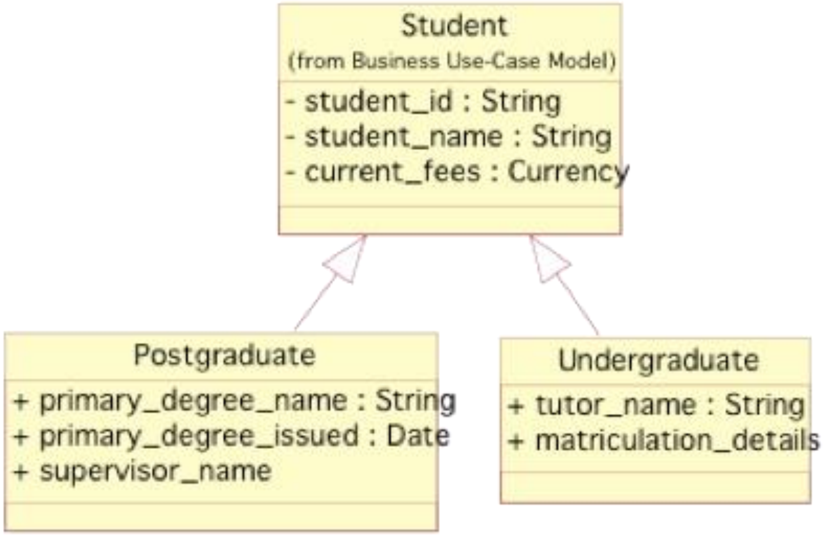
:Student

Student_id: 99124
Student_name: Frank Clarke
Current_fees: 6500

:Postgraduate

Primary_degree_name:BA
Primary_degree_issued:12 Nov 2003
Supervisor_name: John Smith

Solution Exercise 4



:Student

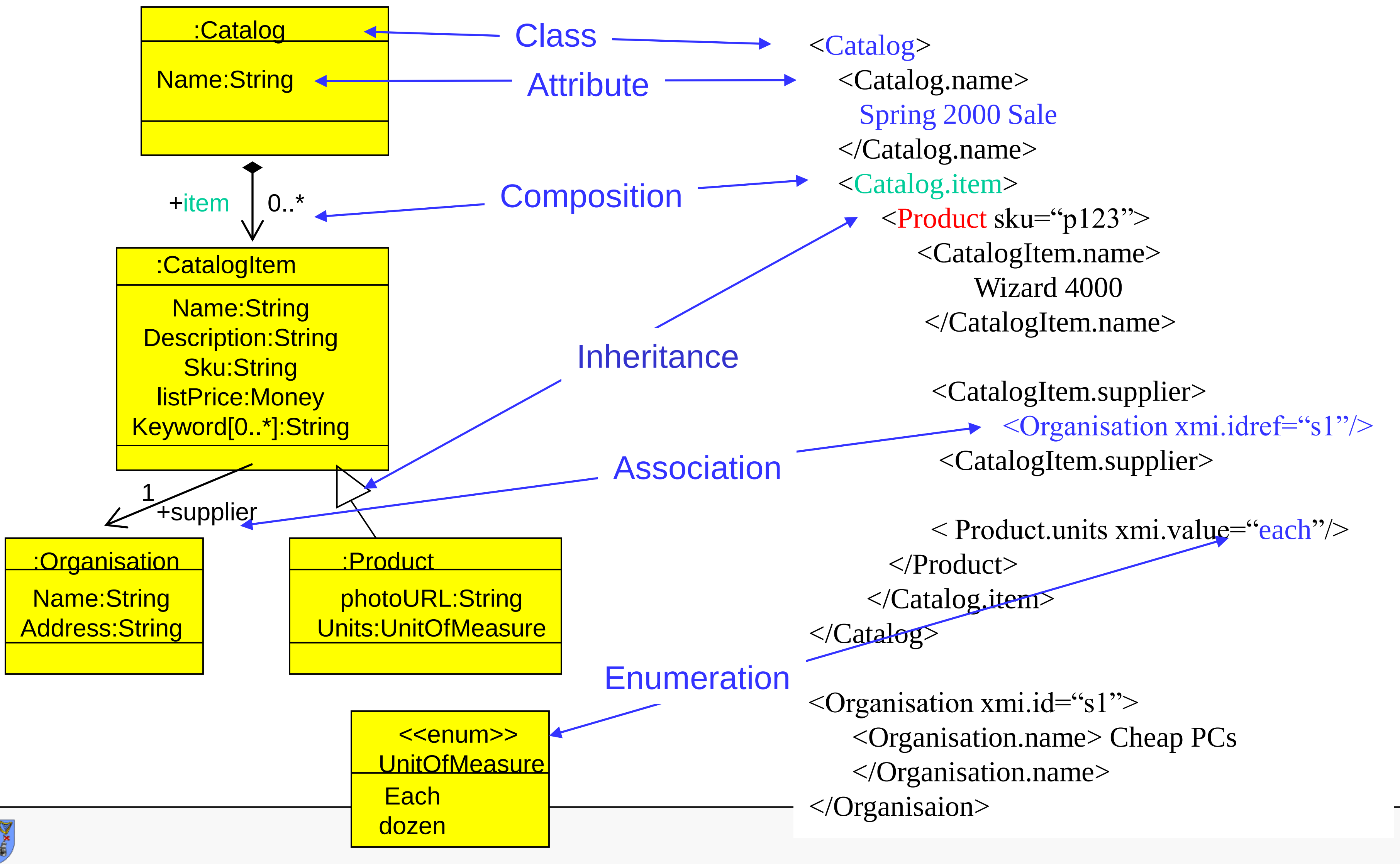
Student_id: 99124
Student_name: Frank Clarke
Current_fees: 6500

:Postgraduate

Primary_degree_name:BA
Primary_degree_issued:12 Nov 2003
Supervisor_name: John Smith

```
<Postgraduate>
   <Student.student_id> 99124 </Student.student_id>
   <Student.student_name> Frank Clarke </Student.student_name>
   <Student.current_fees> 6500 </Student.current_fees>
   <Postgraduate.primary_degree_name> BA </Postgraduate.primary_degree_name>
   <Postgraduate.primary_degree_issued > 12 November 2003 </Postgraduate.primary_degree_issued>
   <Postgraduate.supervisor_name > John Smith </Postgraduate.supervisor_name>
</Postgraduate>
```

Summary Example



```
<Catalog>
   <Catalog.name>
     Spring 2000 Sale
   </Catalog.name>
   <Catalog.item>
      <Product sku="p123">
         <CatalogItem.name>
              Wizard 4000
         </CatalogItem.name>

          <CatalogItem.supplier>
             <Organisation xmi.idref="s1"/>
          <CatalogItem.supplier>

          < Product.units xmi.value="each"/>
      </Product>
    </Catalog.item>
</Catalog>

<Organisation xmi.id="s1">
     <Organisation.name> Cheap PCs
     </Organisation.name>
</Organisaion>
```

Part 2: Group Project- XML Task

STEP 1: XML DESIGN

From your group's UML Class diagram, pick **at least** 7 classes and for each create a different XML document (that is they each have a different DTD for each XML document), with the following characteristics for each XML document:

a) **At least 6** different XML elements/tags are used.

b) **At least one third** of the XML elements should have 1 XML attribute

c) There is **interlinks between** some of the documents (reflecting the assocations/relationships between the classes within the UML design), with enough information information to allow for interesting cross document XML Queries to be designed

1. For each DTD
Use comments to clearly state what is the purpose of the document, and comments describing purpose of each element and for each attribute, and why certain cardinality (*,+ etc.) is used.

You should end up **7 XML** documents with **7 commented DTDs**.

Part 2: Group Project - XML Task

STEP 2: XML QUERY DESIGN

Design and Document **at minimum 8** interesting **XQuery** queries that support some of your UML use cases, with the following characteristics:

- At least 3 of the queries should retrieve information from two or more interlinked XML documents, using the WHERE clause
- At least 2 of the queries should use the FOR clause
- At least 1 of the queries should use the LET clause
- At least 2 of the queries should use a Built-in XQuery function
- At least 2 of the queries should use User Defined Functions

In the report, for each query, you need to document:

(a) identification of the UML use case that it supports
(b) description of the purpose of the query
(c) provide example of output that you expect when query is executed.

Part 2: Group Project XML Task Deliverables

1. **ALL** GROUPS Sign in Group Report(See below) on **Monday 19th November 2018 at 10am**

2. Demonstrate your XQueries at allocated lab on either Monday **19th November** or Thursday **22nd November 2018**

XML REPORT

- What (if anything) did you need to change in going from UML design to XML implementation?- Include revised diagrams/ethics canvas, if appropriate.
- List who did what in the group for XML implementation
- Strengths and Weaknesses of the XML design and XQueries design
- Include XML documents and commented XML DTDs (see earlier slides)
- Include the documented XML Queries (see earlier slides)

Querying XML documents

XML as a Tree structure

X Path for navigating the tree

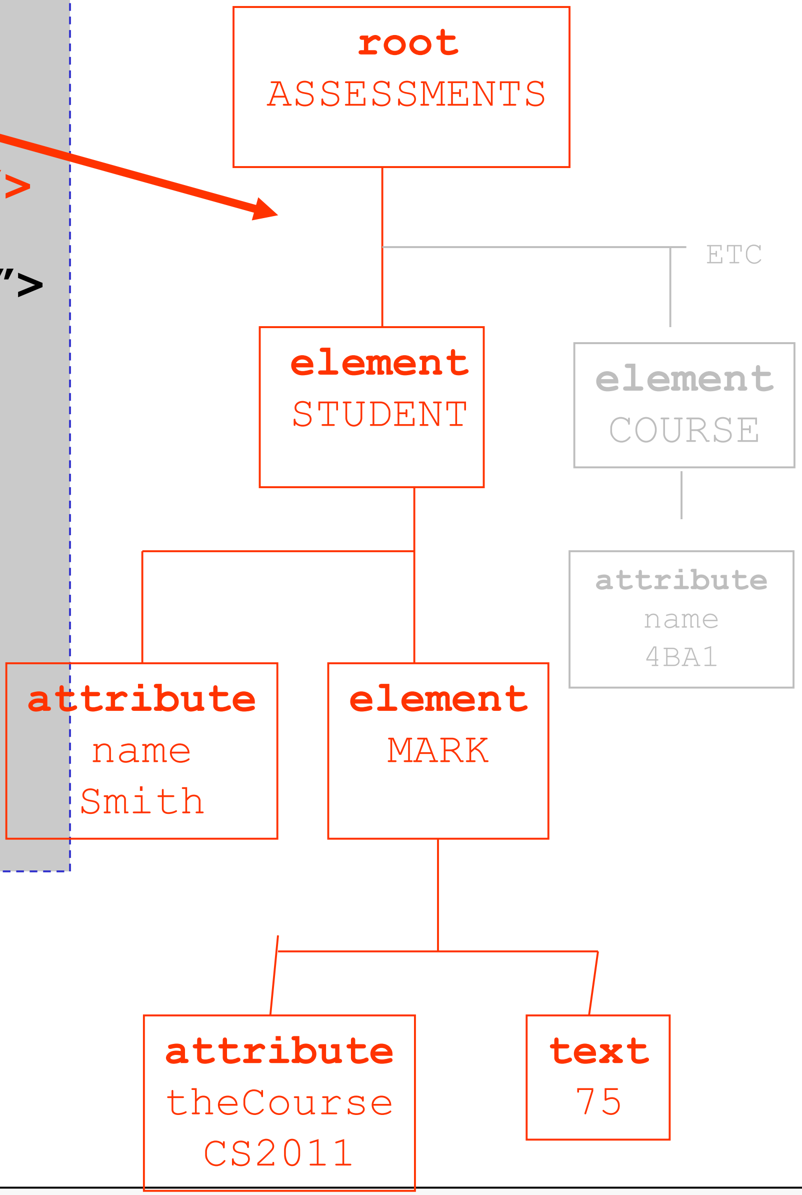
Xpath is used in XQueries

XML as a tree structure

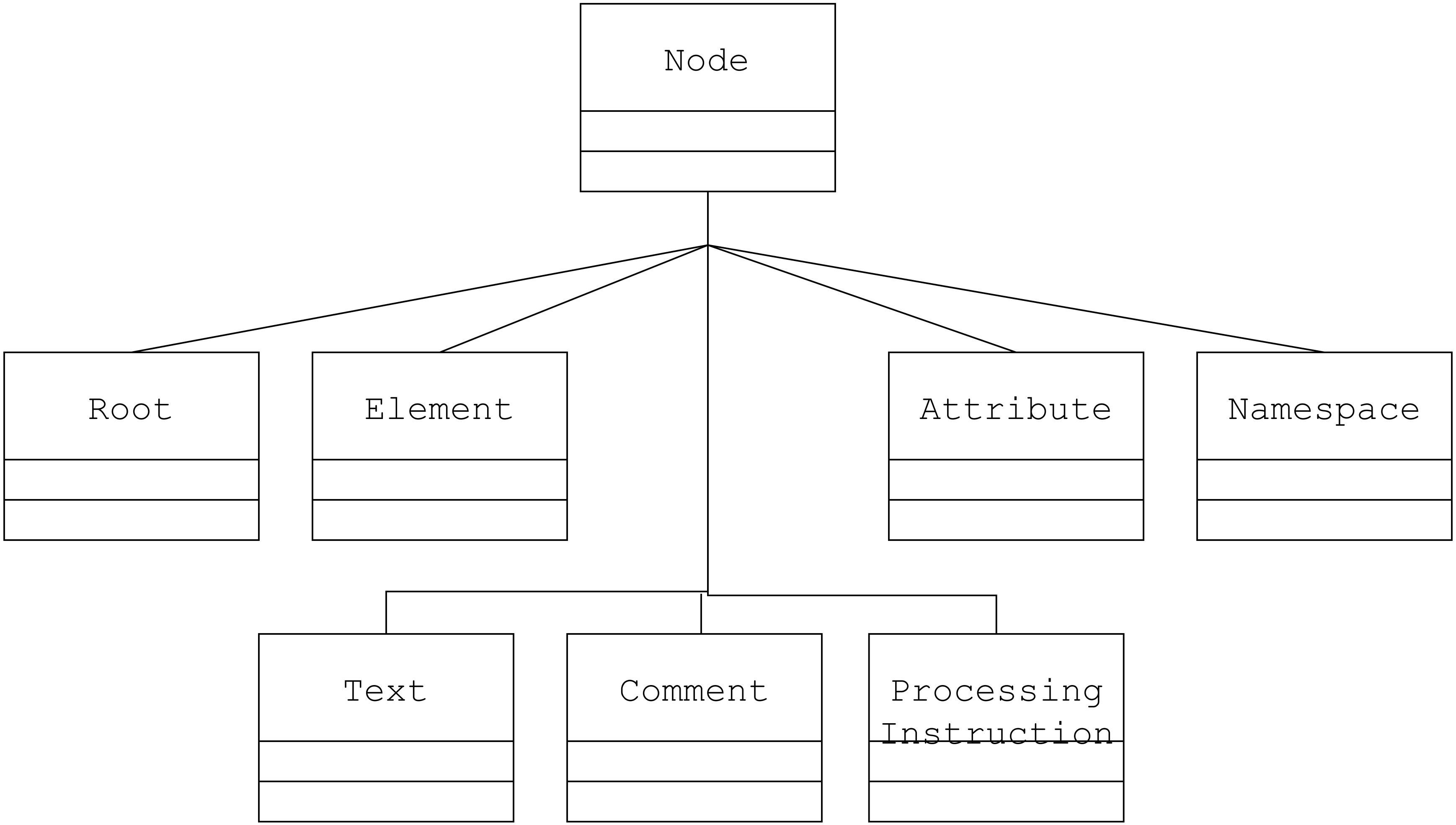
```
<ASSESSMENTS>
            <STUDENT name = “Smith”>
                <MARK theCourse = “CS2011”>
75 </MARK>
                 <MARK theCourse = “CS2012”>
99 </MARK>
            </STUDENT> …



            <COURSE name = “CS2011”,
takenBy = “Smith, Jones, … ”>
             <MARK> 60 </MARK>
            </COURSE> …
</ASSESSMENTS>
```



Nodes in a Tree Model



Exercise 5

- Create a XML Tree representation for the snippet of XML

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
</book>
<!-- Next Book --!>
  <book year="2000">
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
<publisher>Morgan Publishers</publisher>
    <price>39.95</price>
  </book>

</bib>
```

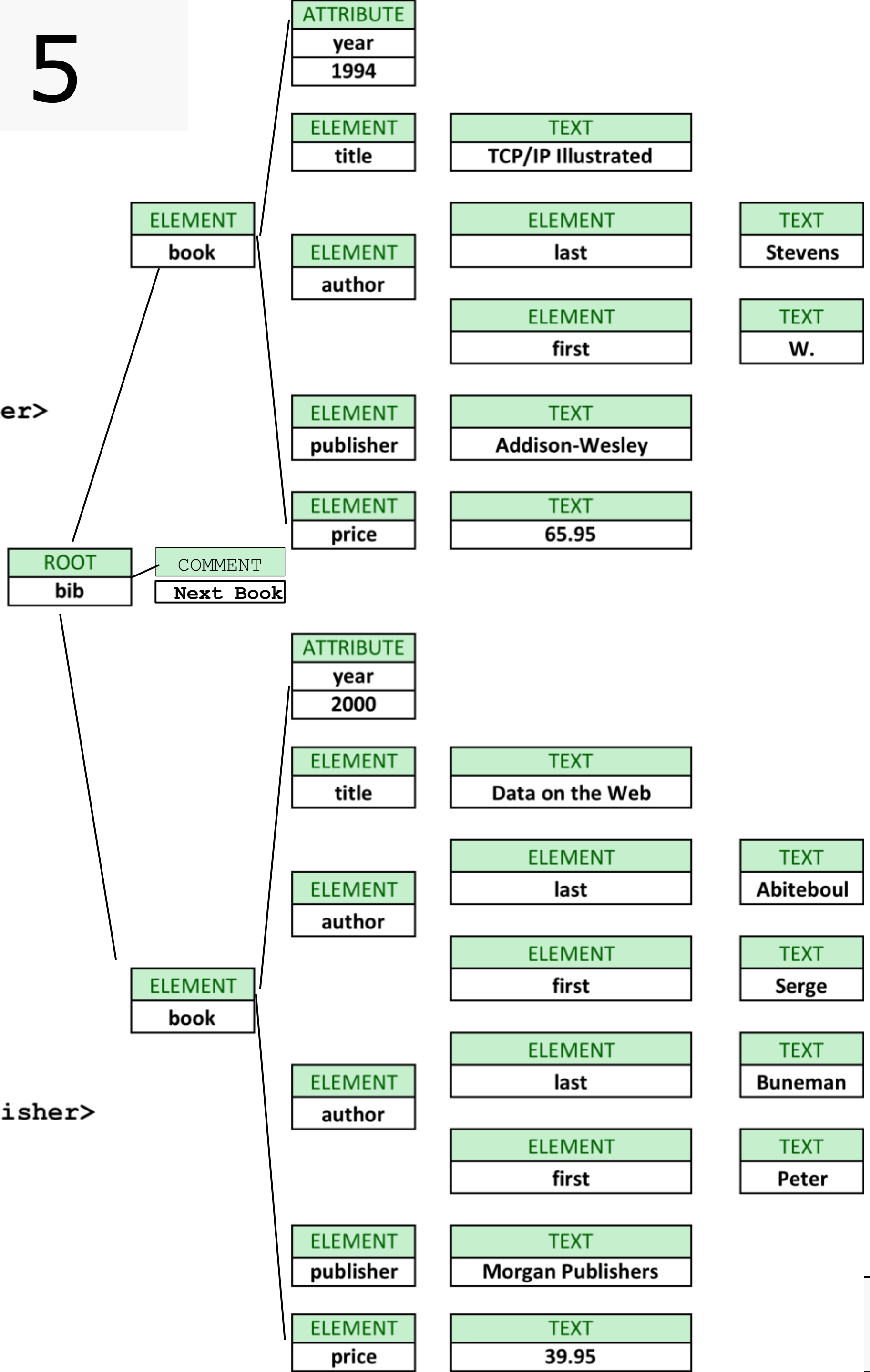
Solution Exercise 5

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>

   <!-- Next Book --!>

  <book year="2000">
    <title>Data on the Web</title>
    <author>
      <last>Abiteboul</last>
      <first>Serge</first>
    </author>
    <author>
      <last>Buneman</last>
      <first>Peter</first>
    </author>
    <publisher>Morgan Publishers</publisher>
    <price>39.95</price>
  </book>

</bib>
```



ROOT bib

ELEMENT book

ATTRIBUTE year 1994

ELEMENT title — TEXT TCP/IP Illustrated

ELEMENT author — ELEMENT last — TEXT Stevens; ELEMENT first — TEXT W.

ELEMENT publisher — TEXT Addison-Wesley

ELEMENT price — TEXT 65.95

COMMENT Next Book

ELEMENT book

ATTRIBUTE year 2000

ELEMENT title — TEXT Data on the Web

ELEMENT author — ELEMENT last — TEXT Abiteboul; ELEMENT first — TEXT Serge

ELEMENT author — ELEMENT last — TEXT Buneman; ELEMENT first — TEXT Peter

ELEMENT publisher — TEXT Morgan Publishers

ELEMENT price — TEXT 39.95

What is XPath?

- Addresses parts of an XML document
- W3C Recommendation
- Expression language
- Wildcards allowed
- Provides basic facilities for manipulation of strings, numbers and booleans
- Compact, non XML syntax for use within URIs
- Operates on the abstract, logical structure of the XML document

XPath Expression

- “Xpath, essentially specification of path for walking the XML tree”

- Simple path expression is a sequence of *steps* to walk the tree. The sequence of steps are separated by slashes (/)

  – More formally, "/" is a binary operator that applies the expression on its right-hand side to the set of nodes selected by the expression on the left hand side

  – Informally, try to find a match for what is right of the slash, in the tree(set of nodes) returned by sequence of operations to the left of the slash

Example Document

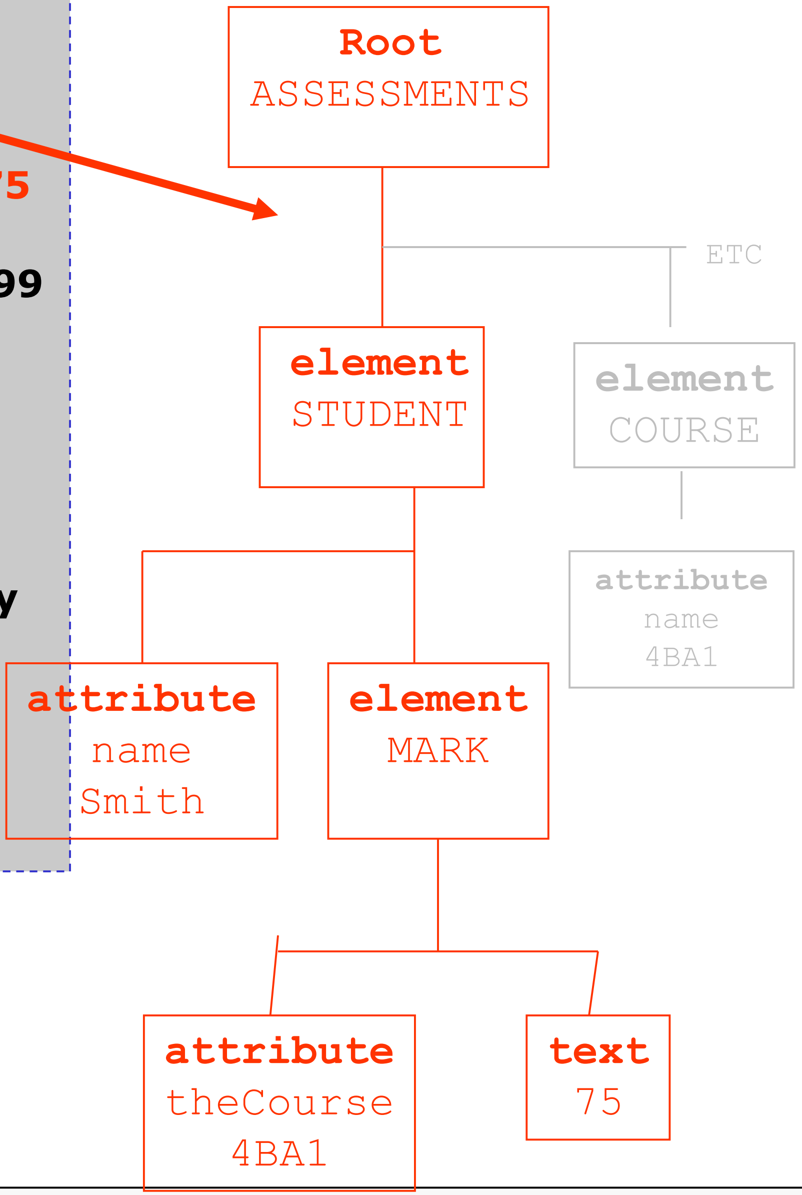
```
<ASSESSMENTS>
            <STUDENT name = 'Smith'>
                <MARK theCourse = '4BA1'> 75
</MARK>
                 <MARK theCourse = '4BA5'> 99
</MARK>
            </STUDENT> ...



            <COURSE name = '4BA1', takenBy
= 'Smith, Jones, ... '>
             <MARK> 60 </MARK>
            </COURSE> ...
</ASSESSMENTS>
```

Describes mark for individual student

Describes average mark for course



Example X Path expression: /ASSESSMENTS

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones, ... ">
        </COURSE> ...
</ASSESSMENTS>
```

Example: /ASSESSMENTS/STUDENT

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones, … ">
        </COURSE> …
</ASSESSMENTS>
```

Example: /ASSESSMENTS/STUDENT/MARK

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones, … ">
        </COURSE> …
</ASSESSMENTS>
```

Describes the set with these two MARK element nodes as well as any other MARK elements nodes for any other STUDENT

Some Defaults

- By default trying to apply expression against any immediate child nodes in the left hand side set of nodes

- If Xpath expression begins with //
  - Selects nodes in the document from the current node that match the selection no matter where they are i.e. trying to match any descendent nodes in the set of nodes

Example: //MARK

```
<ASSESSMENTS>
  <STUDENT name = "Smith">
     <MARK theCourse = "4BA1"> 75 </MARK>
     <MARK theCourse = "4BA5"> 99 </MARK>
  </STUDENT> …

  <COURSE name = "4BA1", takenBy = "Smith, Jones, … ">
     <MARK> 60 </MARK>
  </COURSE> …
</ASSESSMENTS>
```

Still returns set of nodes from the document
with an element node named "MARK"
but this time not just those noted in
student assessment statements
e.g. a mark allocated to a course
by an external examiner

Example: //MARK/string()

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

Getting just the text from any "mark" elements
Using the string() function

Attribute @

- Attributes are referred to by putting an “at” symbol (@) before the name
- Appear in the path as if nested within the tag

Example:
/ASSESSMENTS/STUDENT/string(@name)

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> ...

        <COURSE name = "4BA1", takenBy = "Smith, Jones, ... ">
                <MARK> 60 </MARK>
        </COURSE> ...
</ASSESSMENTS>
```

Getting at an attribute value, string() function

Predicate Filters []

- A part of the path that allows for expression of a condition.
- [..] will ensure that only nodes that satisfy the condition are included in the resultant set

Example:

```
/ASSESSMENTS/STUDENT[MARK > 80]
```

```
<ASSESSMENTS>
        <STUDENT name = "Smith">
                <MARK theCourse = "4BA1"> 75 </MARK>
                <MARK theCourse = "4BA5"> 99 </MARK>
        </STUDENT> …

        <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
                 <MARK> 60 </MARK>
        </COURSE> …
</ASSESSMENTS>
```

Example:

/ASSESSMENTS/STUDENT[MARK > 80]

```
<ASSESSMENTS>
      <STUDENT name = "Smith">
            <MARK theCourse = "4BA1"> 75 </MARK>
            <MARK theCourse = "4BA5"> 99 </MARK>
      </STUDENT> …

      <COURSE name = "4BA1", takenBy = "Smith, Jones,
… ">
             <MARK> 60 </MARK>
      </COURSE> …
</ASSESSMENTS>
```

This set of nodes is returned
as it satisfies the condition

Example Attribute in the filter:

/ASSESSMENTS/STUDENT/MARK[@theCourse = "4BA1"]

```
<ASSESSMENTS>
      <STUDENT name = "Smith">
            <MARK theCourse = "4BA1"> 75 </MARK>
            <MARK theCourse = "4BA5"> 99 </MARK>
      </STUDENT> ...

      <COURSE name = "4BA1", takenBy = "Smith, Jones,
... ">
            <MARK> 60 </MARK>
      </COURSE> ...
</ASSESSMENTS>
```

This set of nodes is returned as well as any other student MARK subtree nodes for 4BA1 elsewhere

Wildcard *

- An asterix (*) Can be used as a wildcard
- Example /*/*/MARK will return any MARK Element appearing at the third level of nesting in the document

Consider what part of the tree (set of nodes) the following Xpath expressions will return

```
<database>
<person age='34'>
    <name>
          <title> Mr </title>
          <firstname> John </firstname>
          <firstname> Paul </firstname>
          <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
          <firstname> Mary </firstname>
          <surname> Donnelly </surname>
    </name>
</person>
</database>
```

1. /database
2. //surname
3. /*/person/@age
4. /*/person/string(@age)

```
<database>
<person age='34'>
    <name>
          <title> Mr </title>
          <firstname> John </firstname>
          <firstname> Paul </firstname>
          <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
          <firstname> Mary </firstname>
          <surname> Donnelly </surname>
    </name>
</person>
</database>
```

1. /database
2. //surname
3. /*/person[@age]
4. /*/person/string(@age)

```
<database>
<person age='34'>
    <name>
            <title> Mr </title>
            <firstname> John </firstname>
            <firstname> Paul </firstname>
            <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
            <firstname> Mary </firstname>
            <surname> Donnelly </surname>
    </name>
</person>
</database>
```

1. /database
2. //surname
3. /*/person[@age]
4. /*/person/string(@age)

```
<database>
<person age='34'>
    <name>
          <title> Mr </title>
          <firstname> John </firstname>
          <firstname> Paul </firstname>
          <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
          <firstname> Mary </firstname>
          <surname> Donnelly </surname>
    </name>
</person>
</database>
```

1. /database
2. //surname
3. /*/person[@age]
4. /*/person/string(@age)

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

1. /database
2. //surname
3. /*/person[@age]
4. /*/person/string(@age)

Selecting Several Paths

- By using the | operator in an XPath expression you can select several paths.
- //book/title | //book/price
  - Selects all the title together with price elements of all book elements
- //title | //price
  - Selects all the title together with price elements in the document
- //book/title | //price
  - Selects all the title elements of the book element together with all the price elements in the document

Summary XPath example

```
<doc type="book" isbn="1-56592-796-9">
  <title>A Guide to XML</title>
  <author>Norman Walsh</author>
  <chapter>[...]</chapter>
  <chapter>
    <title>What Do XML Documents Look
      Like?</title>
    <paragraph>If you are [...]</paragraph>
    <paragraph>A few things [...]</paragraph>
    <ol>
      <item><paragraph>The document begins
         [...]</paragraph></item>
      <item><paragraph type="warning">There's
         no document [...]</paragraph></item>
      <item><paragraph>Empty elements have
         [...]</paragraph>
         <paragraph>In a very
           [...]</paragraph></item>
    </ol>
    <paragraph>XML documents are
      [...]</paragraph>
    <section>[...]</section>
    [...]
  </chapter>
</doc>
```

//paragraph

```
<paragraph>If you are [...]</paragraph>
<paragraph>A few things[...]</paragraph>
<paragraph>The document begins
  [...]</paragraph>
<paragraph type="warning">There's
   no document [...]</paragraph>
<paragraph>Empty elements have
   [...]</paragraph>
<paragraph>In a very [...]</paragraph>
<paragraph>XML documents are
   [...]</paragraph>
```

//ol//paragraph[@type=‘warning’]

```
<paragraph type="warning">
  There's no document [...]
</paragraph>
```

/doc/chapter[2]/ol/item[position()=last()]

```
<item><paragraph>Empty elements have
  [...]</paragraph>
  <paragraph>In a very [...]</paragraph>
</item>
```

Exercise 6:Design XPath queries

2. Get all the titles of books in the file (without using //)

3. Get just the text from the first name elements of author

4. Return only the book elements that has an editor

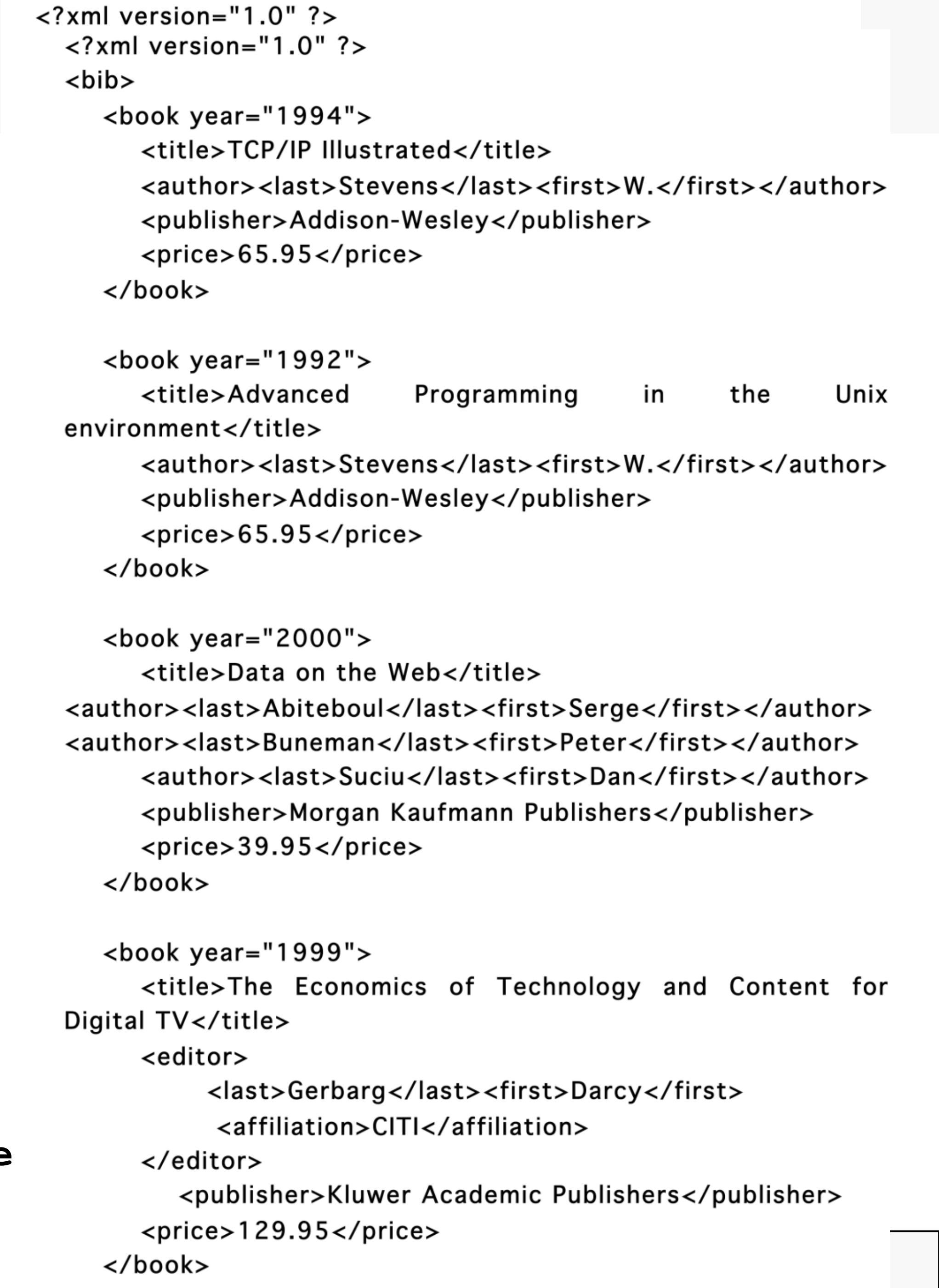
5. Return only the books that are published after 1998

6. Return the entire book element whose title is "Data on the Web"

7. Alter the last query to just return the second author

8. Return those books which are priced between 50 and 100 only

9. Return all those books that are NOT published by Addison-Wesley



```
<?xml version="1.0" ?>
  <?xml version="1.0" ?>
  <bib>
     <book year="1994">
        <title>TCP/IP Illustrated</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
     </book>

     <book year="1992">
        <title>Advanced   Programming   in   the   Unix
  environment</title>
        <author><last>Stevens</last><first>W.</first></author>
        <publisher>Addison-Wesley</publisher>
        <price>65.95</price>
     </book>

     <book year="2000">
        <title>Data on the Web</title>
  <author><last>Abiteboul</last><first>Serge</first></author>
  <author><last>Buneman</last><first>Peter</first></author>
        <author><last>Suciu</last><first>Dan</first></author>
        <publisher>Morgan Kaufmann Publishers</publisher>
        <price>39.95</price>
     </book>

     <book year="1999">
        <title>The  Economics  of  Technology  and  Content  for
  Digital TV</title>
        <editor>
             <last>Gerbarg</last><first>Darcy</first>
              <affiliation>CITI</affiliation>
        </editor>
           <publisher>Kluwer Academic Publishers</publisher>
        <price>129.95</price>
     </book>
```

Exercise 6:Sample Solution

2. Get all the titles of books in the file (without using //)

```
/bib/book/title
```

3. Get just the text from the first name elements of author

```
//first/string()
```

4. Return only the book elements that has an editor

```
//book[editor]
```

5. Return only the books that are published after 1998

```
//book[@year>=1998]
```

6. Return the entire book element whose title is "Data on the Web”

```
 //book[title/string()="Data on the Web"]
```

7. Alter the last query to just return the second author

```
//book[title/string()="Data on the Web"]/author[2]
```

8. Return those books which are priced between 50 and 100 only

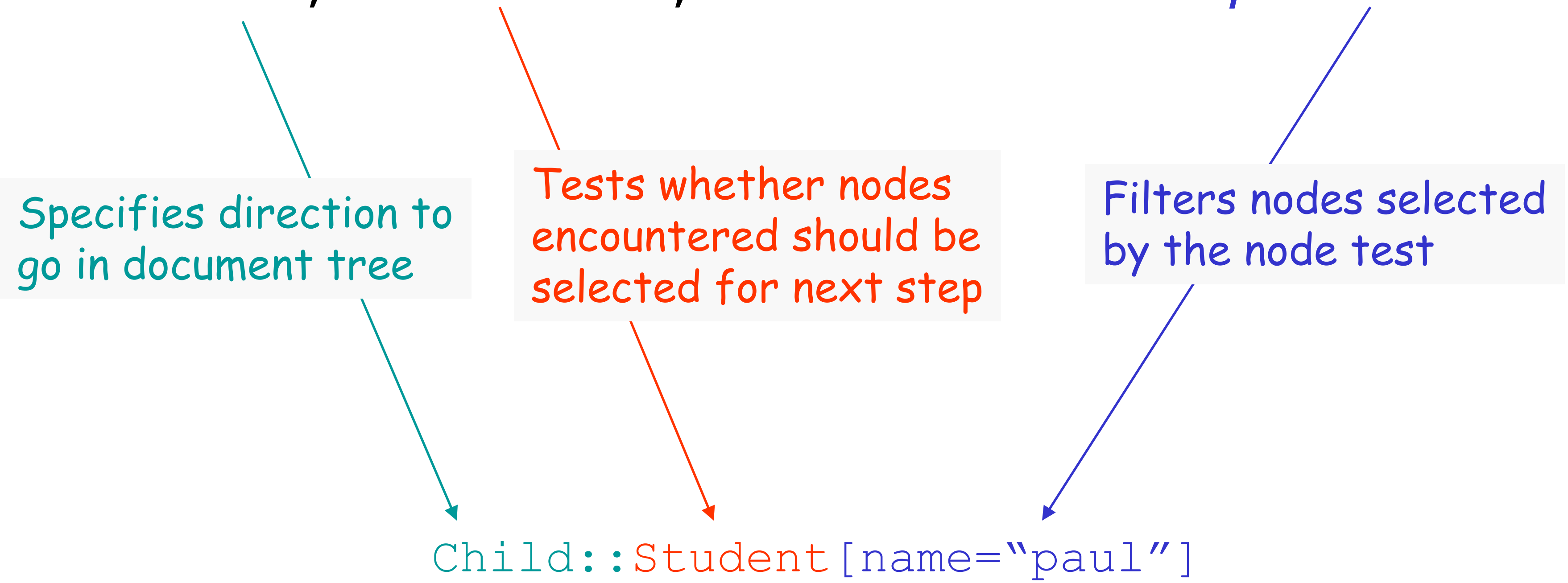
```
 //book[price>50][price<100]
```

9. Return all those books that are NOT published by Addison-Wesley

```
 //book[publisher!="Addison-Wesley"]
```

Location Steps

- A step in an XPath expression consists of three parts: an *axis*, a *node* test, and zero or more *predicate* tests



Specifies direction to go in document tree

Tests whether nodes encountered should be selected for next step

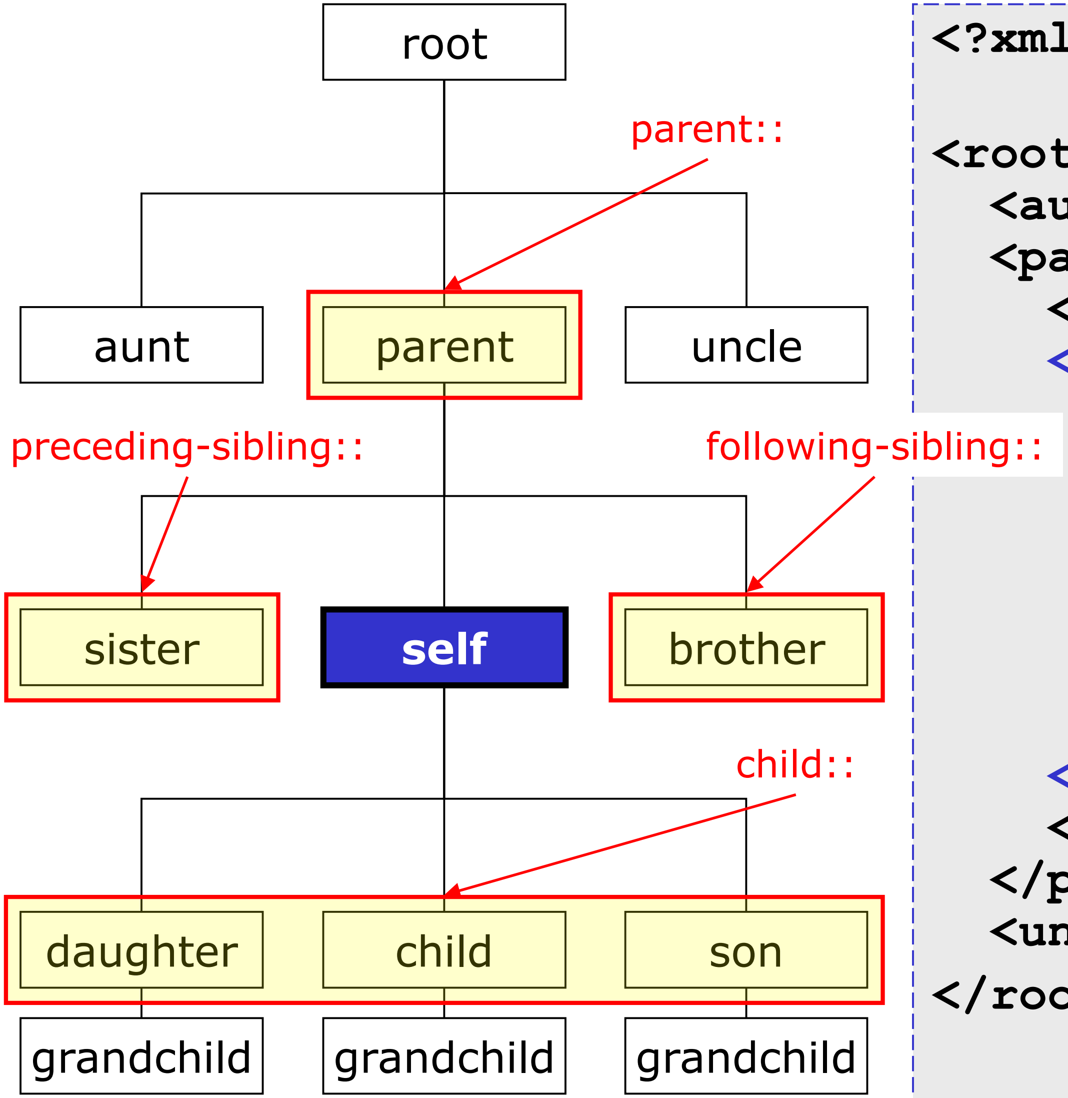
Filters nodes selected by the node test

```
Child::Student[name=“paul”]
```

https://www.w3schools.com/xml/xpath_axes.asp

Axes spec (1)

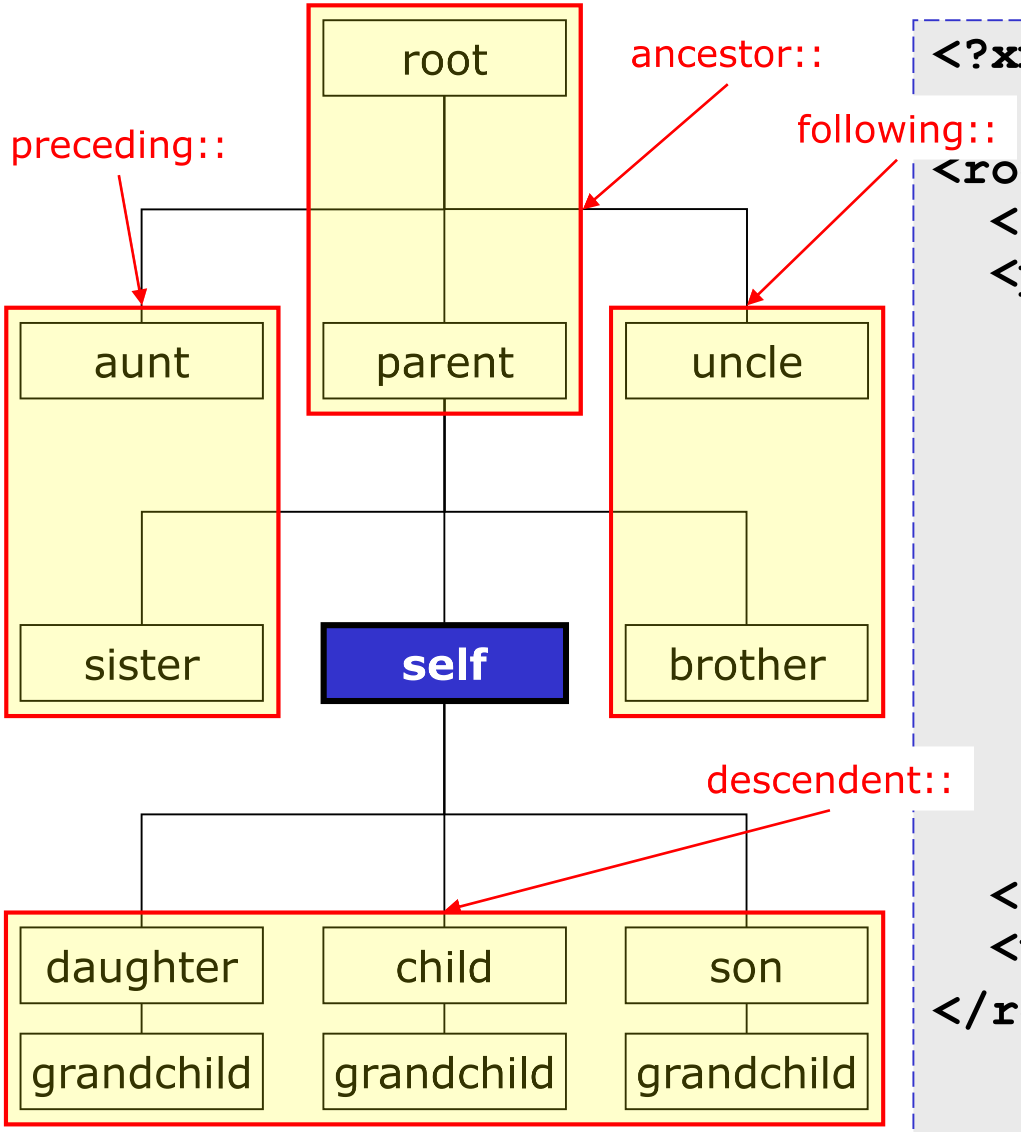
There are several directions/axes we can traverse from a node



```
<?xml version='1.0' ?>

<root>
  <aunt />
  <parent>
    <sister />
    <self>
      <son>
        <grandchild />
      </son>
      <child />
      <daughter>
        <grandchild />
      </daughter>
    </self>
    <brother />
  </parent>
  <uncle></uncle>
</root>
```

Axes spec (2)


root
ancestor::
following::
preceding::
aunt
parent
uncle
sister
self
brother
descendent::
daughter
child
son
grandchild
grandchild
grandchild

```
<?xml version='1.0' ?>

<root>
  <aunt />
  <parent>
    <sister />
    <self>
      <son>
        <grandchild />
      </son>
      <child />
      <daughter>
        <grandchild />
      </daughter>
    </self>
    <brother />
  </parent>
  <uncle></uncle>
</root>
```

Node tests

- The default is to test the node to see if it has an element name the same as that specified
  - E.g. child::Student would test if the child node has an element named “Student”
- Tests for checking element, attribute, and namespace name
- Tests for checking if the node is a text, comment, or processing instruction node
  - E.g. text()

Predicate Filters

- [] are used to hold predicates (conditions)
  - consecutive predicates are indicted using [][]

  - The words filter or function can also be used instead of condition

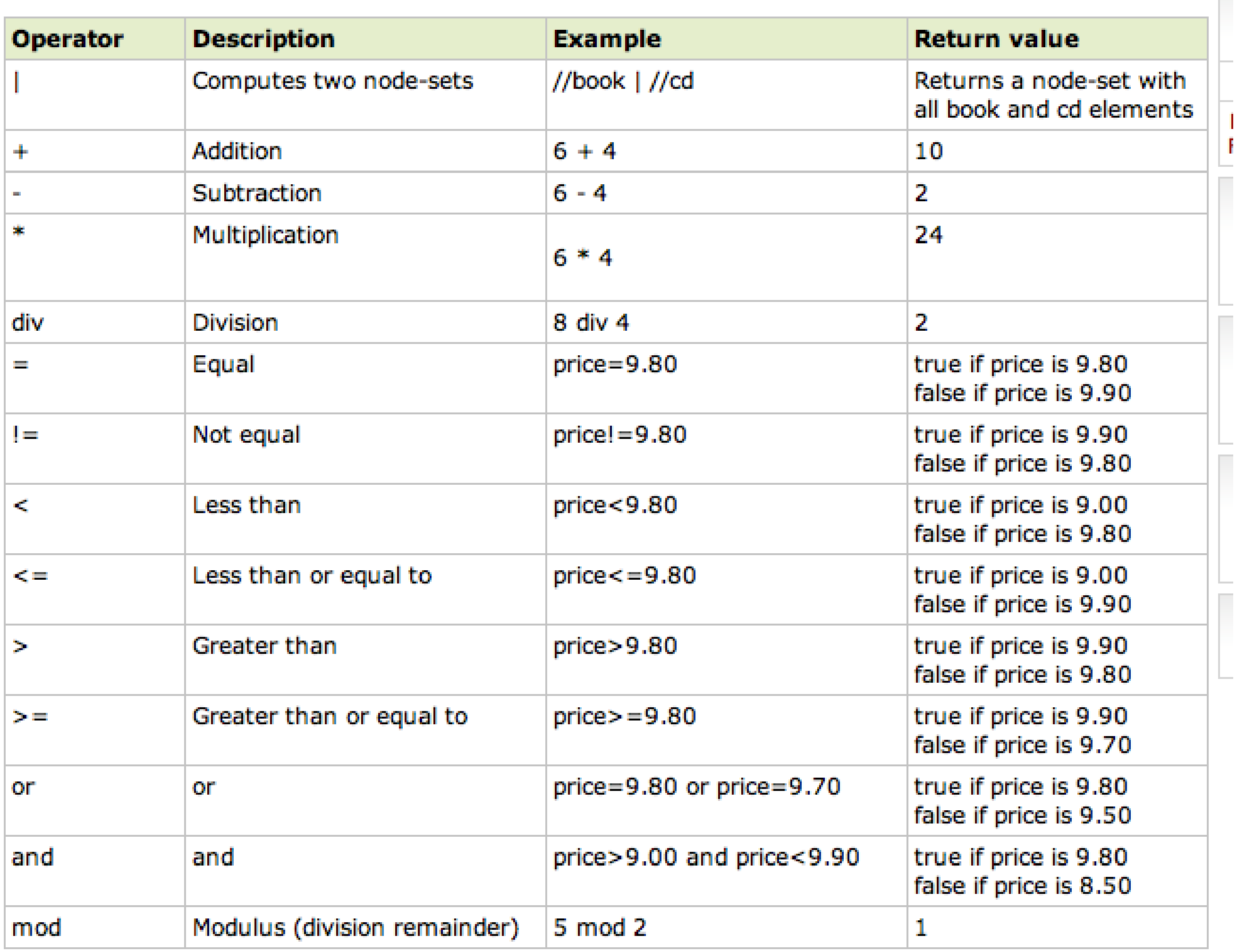
Built-In Functions

- Accessor Functions
  - e.g. fn:node-name(*node) Returns the node-name of the argument node*
- Functions on Numeric Values
- Functions on Strings
- Functions on Durations, Dates and Times
- Functions on Nodes
- Functions on Sequences
- Aggregate Functions
- Context Functions

https://www.w3schools.com/xml/xsl_functions.asp

XPath Operators

An XPath expression returns either a node-set, a string, a Boolean, or a number.



Operator	Description	Example	Return value
|	Computes two node-sets	//book | //cd	Returns a node-set with all book and cd elements
+	Addition	6 + 4	10
-	Subtraction	6 - 4	2
*	Multiplication	6 * 4	24
div	Division	8 div 4	2
=	Equal	price=9.80	true if price is 9.80 false if price is 9.90
!=	Not equal	price!=9.80	true if price is 9.90 false if price is 9.80
<	Less than	price<9.80	true if price is 9.00 false if price is 9.80
<=	Less than or equal to	price<=9.80	true if price is 9.00 false if price is 9.90
>	Greater than	price>9.80	true if price is 9.90 false if price is 9.80
>=	Greater than or equal to	price>=9.80	true if price is 9.90 false if price is 9.70
or	or	price=9.80 or price=9.70	true if price is 9.80 false if price is 9.50
and	and	price>9.00 and price<9.90	true if price is 9.80 false if price is 8.50
mod	Modulus (division remainder)	5 mod 2	1

Summary

- Selects (a set of) nodes within an XML document based on
  - Conditions
  - Hierarchy
- Usage
  - Retrieving info from a single XML document
  - Making Xquerys
  - Applying XSL style sheet rules

Tutorial available at:
http://www.w3schools.com/xml/xpath_intro.asp

XPath and XQuery Labs Dates

- Monday 12th November 2018 10 to 11am
  - Groups 16 to 22 inclusive
- Monday 12th November 2018 11am to 12 noon
  - Groups 8 to 15 inclusive
- Thursday 15th November 2018 11am to 12 noon
  - Groups 1 to 7 inclusive
- Venue: To Be Confirmed

Demos Dates

- Monday 19th November 2018 10 to 11am
  - Groups 16 to 22 inclusive
- Monday 19th November 2018 11am to 12 noon
  - Groups 8 to 15 inclusive
- Thursday 22nd November 2018 11am to 12 noon
  - Groups 1 to 7 inclusive
- Venue: To Be Confirmed

Querying XML Documents

XQuery

What is XQuery?

- Originally focused on **retrieval** of information from XML documents
  - **Update** features added in 2011
    https://www.w3.org/TR/xquery-update-10/
- XQuery is a language for finding and extracting elements and attributes from XML documents.
  - Here is an example of a question that XQuery could solve:
  - "Select all CD records with a price less than 10 euro from the CD collection stored in the XML document called cd_catalog.xml”
- Used in conjunction with XPath
- Latest version W3C recommendation
  “XQuery 3.0” – April 2014
  https://www.w3.org/TR/xquery-30/

For-Let-Where-OrderBy-Return:“FLWOR” expressions

(pronounced “FLOWER”)

1. One or more FOR and/or LET expressions
   - For gathering nodes into sets from a series of XPath queries to operate upon in other clauses
2. Optional WHERE clause
   - For filtering nodes in the sets to be operated upon in other clauses
3. Optional ORDER BY clause
   - For returning nodes in the sets in particular order in other clauses
4. RETURN clause
   - How to return the identified nodes in the sets

LET Clause

- LET <variable> := <xpath expression>, <xpath expression>, …
  - Variable (starting with $) “binds to” **the set** returned by xpath expression
  - Does not iterate over set like the FOR clause does
  - More than one variable/path expression binding can be specified by separating with comma (,)

Example LET Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
let $c:=
doc("data/tcd.xml")/assessments/co
urse/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>
```

Result

```
<list_of_avg_course_marks>
    <mark>60</mark>
    <mark>70</mark>
</list_of_avg_course_marks>
```

Curly brackets {} are used for enclosed expressions and indicate that the expression enclosed in the return clause needs to be evaluated by the Xquery processor

FOR Clause

FOR <variable> IN <xpath expression>, <xpath expression>, <xpath expression>,…

- Variable (starting with $) “binds to” **in turn each member in the set** returned by Xpath expression(s)
- For each variable binding the rest of FLOWR expression is executed
- More than one variable/path expression binding can be specified by separating with comma (,)

Example FOR Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/co
urse
return
("Course Node:”,$j)
```

Result

```
Course Node: <course name="4BA1"
takenby="Smith,Jones">
    <mark>60</mark>
</course>
Course Node: <course name="4BA5"
takenby="Smith,Bond">
    <mark>70</mark>
</course>
```

Round Brackets useful for grouping sequence of Operations.

RETURN Clause

- One limitation of Xpath is that it can only operate on existing elements/attributes within the document

- XQuery allows the generation of new elements/attributes nodes
  - The element's content (if any) is either literally given between start- and end-tag, or provided as an “enclosed expression”, or as a mixture of both.
  - Curly brackets {} are used for enclosed expressions in the return clause and indicate that the expression enclosed needs to be evaluated by the Xquery processor

Example RETURN Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
       doc("data/tcd.xml")/assess
ments/course/@name
return
       <one_of_courses_is>
       {$j}
       </one_of_courses_is>
```

Example of Xquery
node generation

Result

```
<one_of_courses_is name="4BA1"/>
<one_of_courses_is name="4BA5"/>
```

WHERE Clause

- Filters the binding tuples produced by the FOR and LET clauses
- If the filter expression evaluates to true then the RETURN clause is executed

Example WHERE Clause

XML Source

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
        </mark>
      <mark thecourse="4BA1"> 75
        </mark>
  </student>
  <course name="4BA1"
          takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
          takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XQuery

```
for $j in
doc("data/tcd.xml")/assessments/co
urse
where contains($j/@takenby,"Bond")
return
       <Bond_courses_is>
       {string($j/@name)}
       </Bond_courses_is>
```

Result

```
<Bond_courses_is>4BA5</Bond_courses_is>
```

Querying over several interlinked documents

XML Source
Tcd.xml

```
<?xml version="1.0"?>
<assessments>
  <student name="Smith">
      <mark thecourse="4BA5"> 99
         </mark>
      <mark thecourse="4BA1"> 75
         </mark>
  </student>
  <course name="4BA1"
takenby="Smith,Jones">
     <mark>60</mark>
  </course>
  <course name="4BA5"
             takenby="Smith,Bond">
     <mark>70</mark>
  </course>
</assessments>
```

XML Source
details.xml

```
<?xml version="1.0"?>
<studentdetails>
 <student name=“Smith”>
    <address> 101 Pine </address>
    <enrolled> 2001 </enrolled>
  </student>
<student name=“Bond”>
    <address> 007 Fleming </address>
    <enrolled> 2002 </enrolled>
  </student>
```

XQuery

```
for $w in
doc("data/details.xml")/studentdet
ails/student,
$x in
doc("data/tcd.xml")/assessments/st
udent
where $x/@name = $w/@name
return
<studentpercourse>
   {$w/@name}
   {$w/address}
   {$x/mark/@thecourse}
</studentpercourse>
```

Result

```
<studentpercourse name="Smith"
thecourse="4BA5" thecourse="4BA1">
    <address> 101 Pine </address>
</studentpercourse>
```

Exercise 7

Source

```
<database>
<person age='34'>
    <name>
        <title> Mr </title>
        <firstname> John </firstname>
        <firstname> Paul </firstname>
        <surname> Murphy </surname>
    </name>
    <hobby> Football </hobby>
    <hobby> Racing </hobby>
</person>

<person >
    <name>
        <firstname> Mary </firstname>
        <surname> Donnelly </surname>
    </name>
</person>
</database>
```

- Example syntax

```
let $c:=
   doc("data/tcd.xml")/assessments
   /course/mark
return
   <list_of_avg_course_marks>
   {$c}
   </list_of_avg_course_marks>

for $j in
         doc("data/tcd.xml")/assess
   ments/course/@name
return
   <one_of_courses_is>
   {$j}
   </one_of_courses_is>
```

Define a query which will return an element called "paul_hobbys" which contains the hobby elements for each of person elements who have "Paul" as a firstname

Solution Exercise 7

Source

```
<database>
<person age='34'>
     <name>
             <title> Mr </title>
             <firstname> John </firstname>
             <firstname> Paul </firstname>
             <surname> Murphy </surname>
     </name>
     <hobby> Football </hobby>
     <hobby> Racing </hobby>
</person>

<person >
     <name>
             <firstname> Mary </firstname>
             <surname> Donnelly </surname>
     </name>
</person>
</database>
```

XQuery

```
for $p in
    doc("persondb.xml")/database/person
where $p/name/firstname=" Paul "
return
<paul_hobbys>
{$p/hobby}
</paul_hobbys>
```

Result

```
<paul_hobbys>
   <hobby> Football </hobby>
   <hobby> Racing </hobby>
</paul_hobbys>
```