Chapter 4

Java™ Education and Technology Services

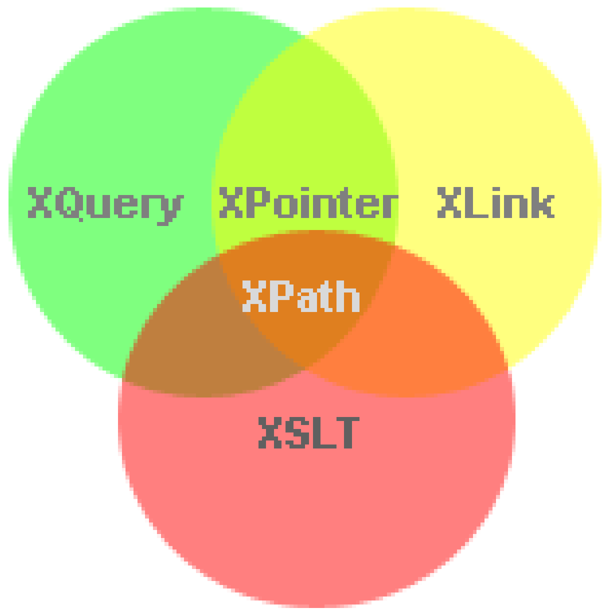
XPath

Introduction

- XPath stands for **XML Path Language**
- XPath uses "path like" syntax to identify and navigate nodes in an XML document

- XQuery and XPointer are both built on XPath expressions
- XPath is a major element in the XSLT standard

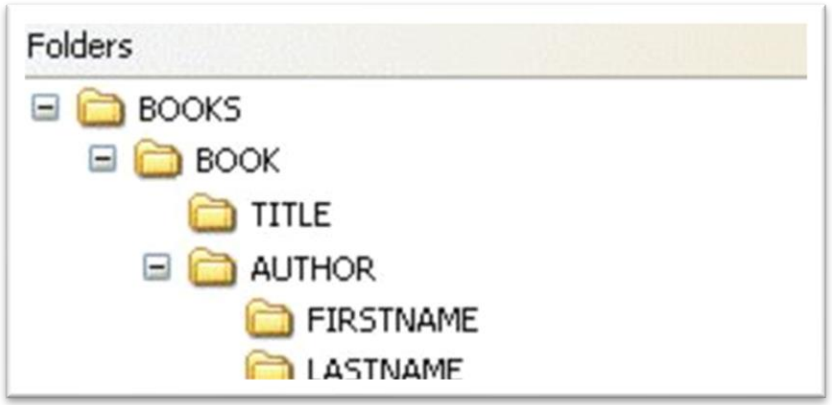
- XPath contains over 200 built-in functions
- XPath is a W3C recommendation



XPath Path Expressions

- **XPath uses path expressions to select nodes or node-sets in an XML document.**

- These path expressions look very much like the path expressions you use with traditional computer file systems:



XPath Terminology

- In XPath, there are **seven** kinds of nodes:
  - Element
  - Attribute
  - Text
  - Namespace
  - Processing-Instruction
  - Comment
  - Document

- Remember, XML documents are treated as trees of nodes.
- The topmost element of the tree is called the root element.

The XML Example Document

```
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>

        <book>
                <title lang="en">Harry Potter</title>
                <price>29.99</price>
        </book>

        <book>
                <title lang="en">Learning XML</title>
                <price>39.95</price>
        </book>

</bookstore>
```

Selecting Nodes

- XPath uses path expressions to select nodes in an XML document.
- The node is selected by following a path or steps.
- The most useful path expressions are listed below:

Expression	Description
nodename	Selects all nodes with the name "nodename"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

DEMO

- In the table below we have listed some path expressions and the result of the expressions:
- **Note:** Be careful where the cursor is in the document, when using an IDE

Path Expression	Result
bookstore	Selects all nodes with the name "bookstore"
/bookstore	Selects the root element bookstore Note: If the path starts with a slash (/) it always represents an absolute path to an element!
bookstore/book	Selects all book elements that are children of bookstore
//book	Selects all book elements no matter where they are in the document
bookstore//book	Selects all book elements that are descendant of the bookstore element, no matter where they are under the bookstore element
//@lang	Selects all attributes that are named lang

Predicates

- Predicates are used to find a specific node or a node that contains a specific value.
- Predicates are always embedded in square brackets [ ].

- Here are some path expressions with predicates and their results:

Path Expression	Result
/bookstore/book[1]	Selects the first book element that is the child of the bookstore element.
/bookstore/book[last()]	Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	Selects the first two book elements that are children of the bookstore element

DEMO

- Here are some path expressions with predicates and their results:

Path Expression	Result
//title[@lang]	Selects all the title elements that have an attribute named lang
//title[@lang='en']	Selects all the title elements that have a "lang" attribute with a value of "en"
/bookstore/book[price>35.00]	Selects all the book elements of the bookstore element that have a price element with a value greater than 35.00
/bookstore/book[price>35.00]/title	Selects all the title elements of the book elements of the bookstore element that have a price element with a value greater than 35.00

Selecting Unknown Nodes

- XPath wildcards can be used to select unknown XML nodes.

Wildcard	Description
*	Matches any element node
@*	Matches any attribute node
node()	Matches any node of any kind

Path Expression	Result
/bookstore/*	Selects all the child element nodes of the bookstore element
//*	Selects all elements in the document
//title[@*]	Selects all title elements which have at least one attribute of any kind

Selecting Several Paths

- By using the | operator in an XPath expression you can select several paths.
- In the table below we have listed some path expressions and the result of the expressions:

Path Expression	Result
//book/title | //book/price	Selects all the title AND price elements of all book elements
//title | //price	Selects all the title AND price elements in the document
/bookstore/book/title | //price	Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

Location Path Expression

- A location path can be absolute or relative.

- An absolute location path starts with a slash (/) and a relative location path does not.
- In both cases the location path consists of one or more steps, each separated by a slash:

- An absolute location path: **/step/step/...**
- A relative location path: **step/step/...**

Step

- Each step is evaluated against the nodes in the current node-set.

- A step consists of:
  - an axis (defines the tree-relationship between the selected nodes and the current node)
  - a node-test (identifies a node within an axis)
  - zero or more predicates (to further refine the selected node-set)

- The syntax for a location step is: axisname::nodetest[predicate]

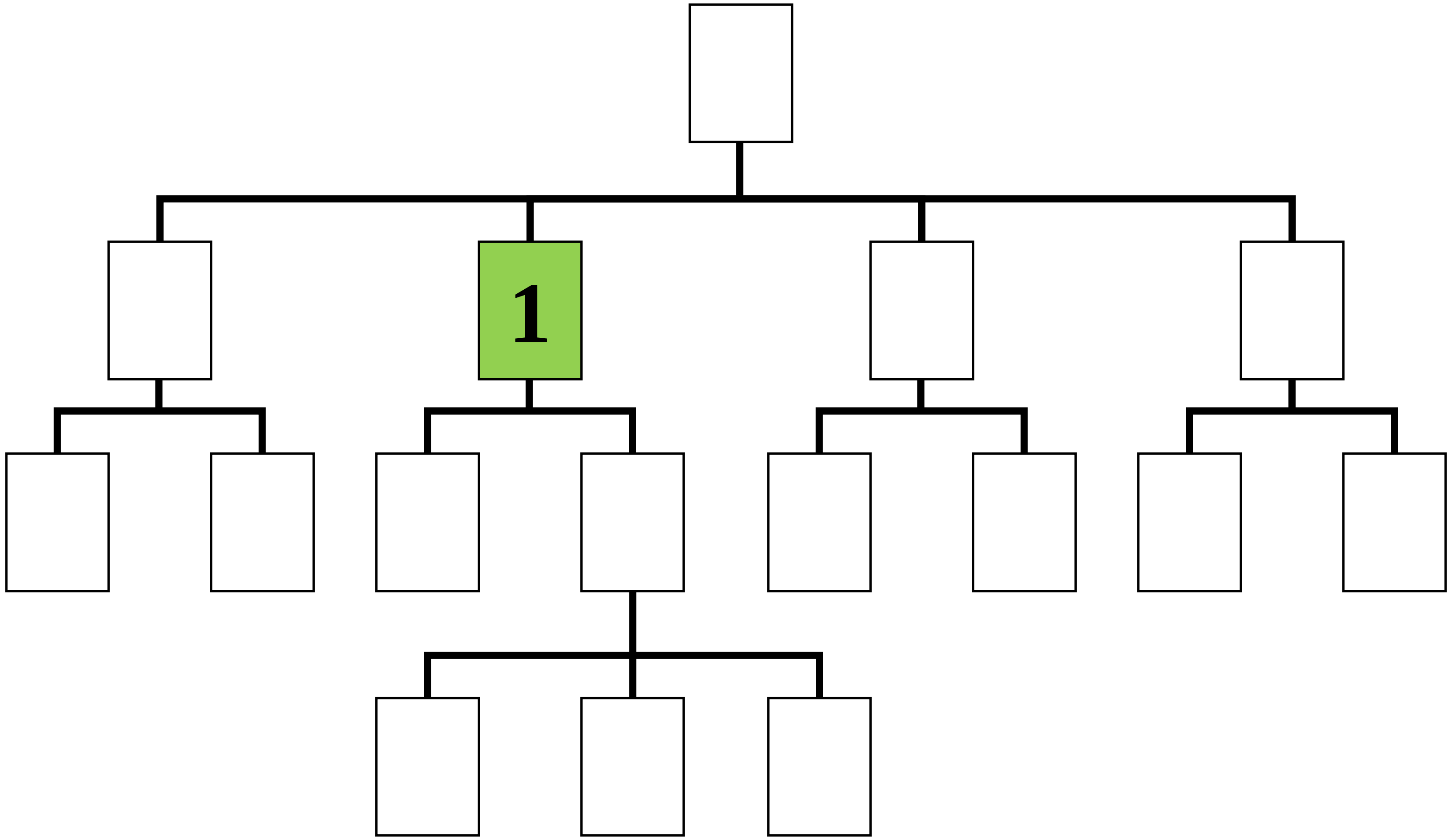
XPath Axes

- **An axis represents a relationship to the context (current) node, and is used to locate nodes relative to that node on the tree.**

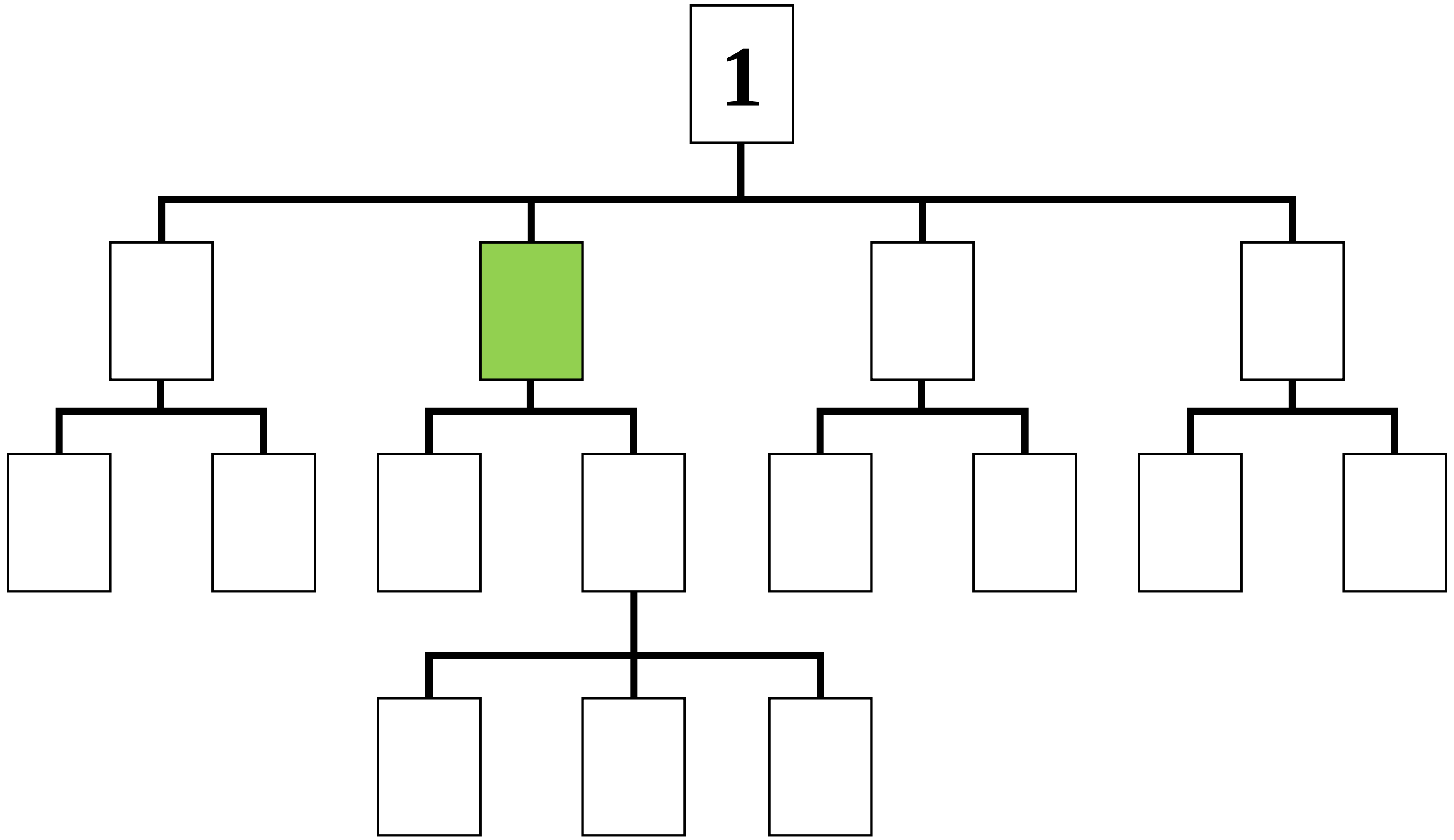
- Relationship of Nodes
  - Self “current or context ”
  - Parent
  - Child
  - Ancestor
  - Descendant
  - Following
  - Following-Sibling
  - Preceding
  - Preceding-Sibling

AxisName	Result
ancestor	Selects all ancestors of the current node
ancestor-or-self	Selects all ancestors of the current node and the current node itself
attribute	Selects all attributes of the current node
child	Selects all children of the current node
descendant	Selects all descendants of the current node
descendant-or-self	Selects all descendants of the current node and the current node itself
following	Selects everything in the document after the closing tag of the current node
following-sibling	Selects all siblings after the current node
namespace	Selects all namespace nodes of the current node
parent	Selects the parent of the current node
preceding	Selects all nodes that appear before the current node in the document
preceding-sibling	Selects all siblings before the current node
self	Selects the current node

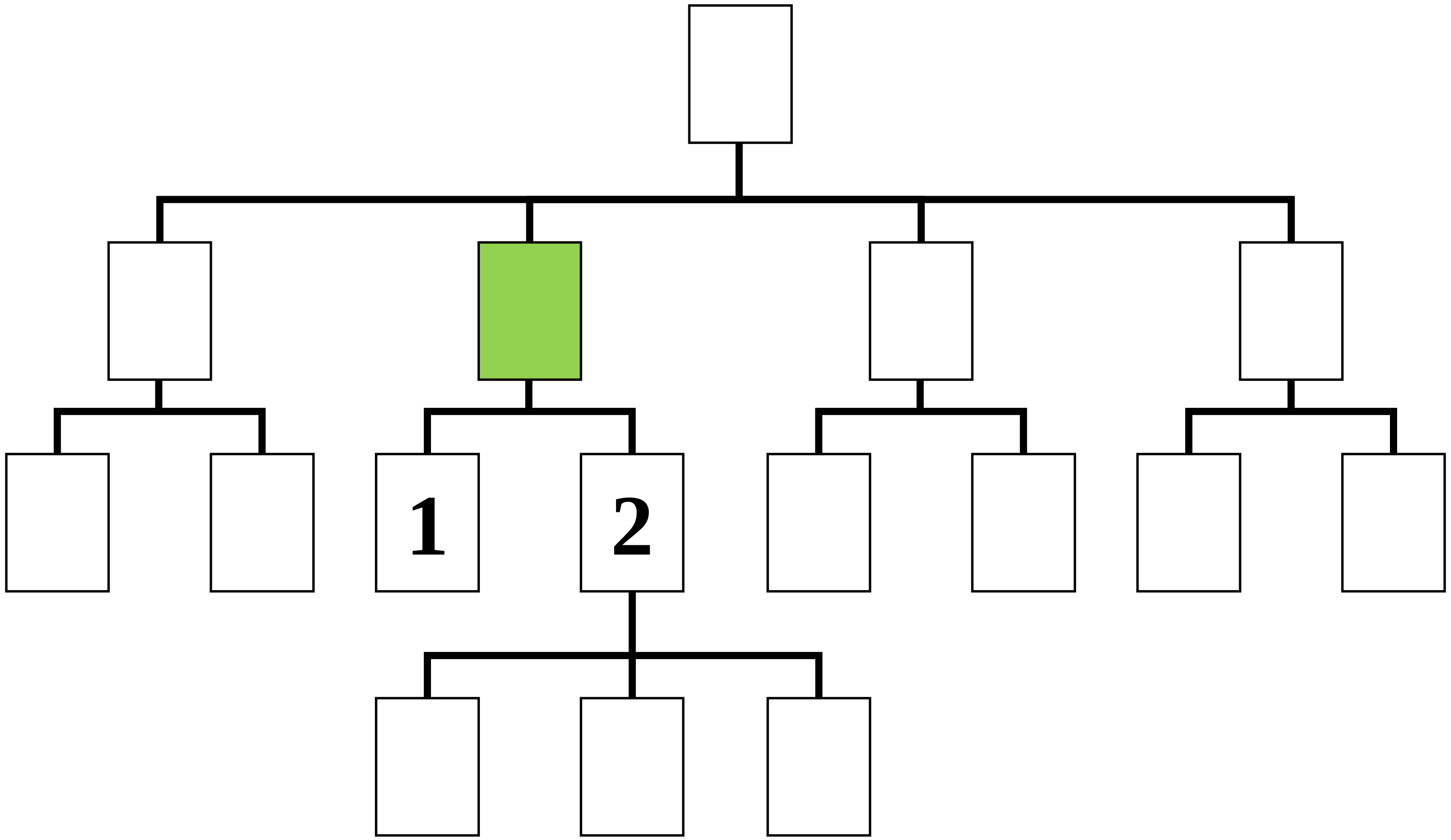
Self “*current or context* ”



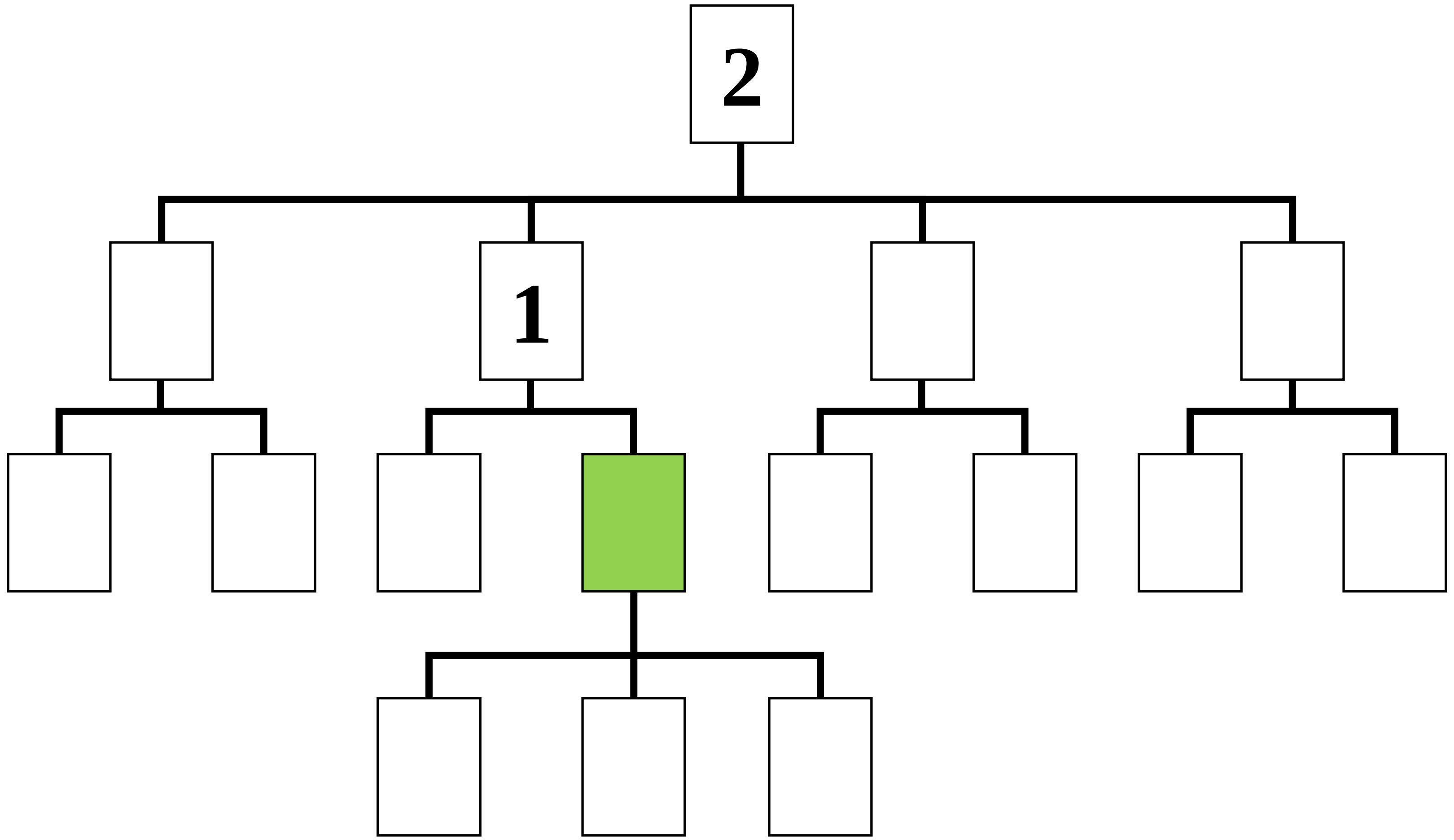
Parent



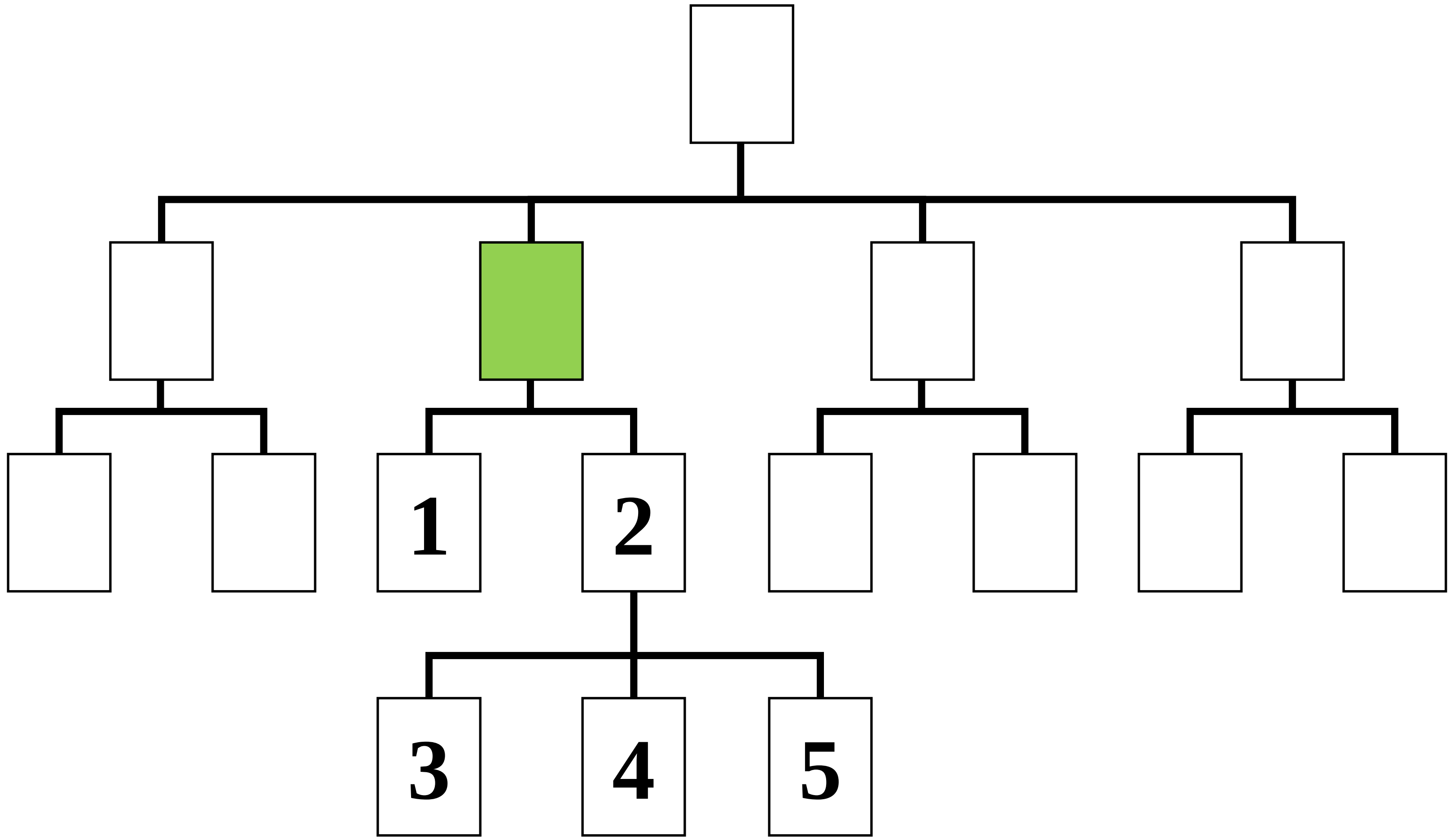
Child



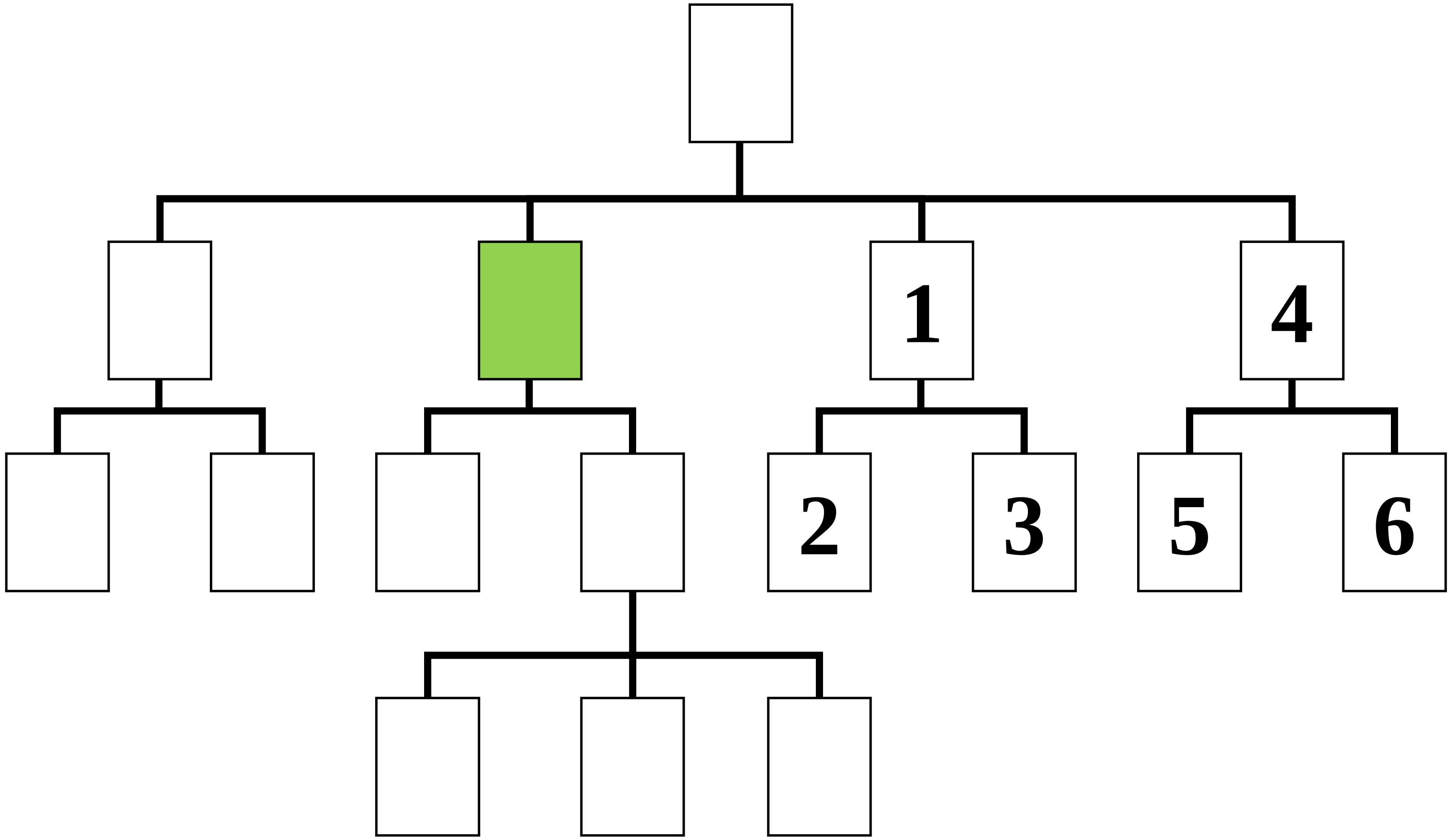
Ancestor



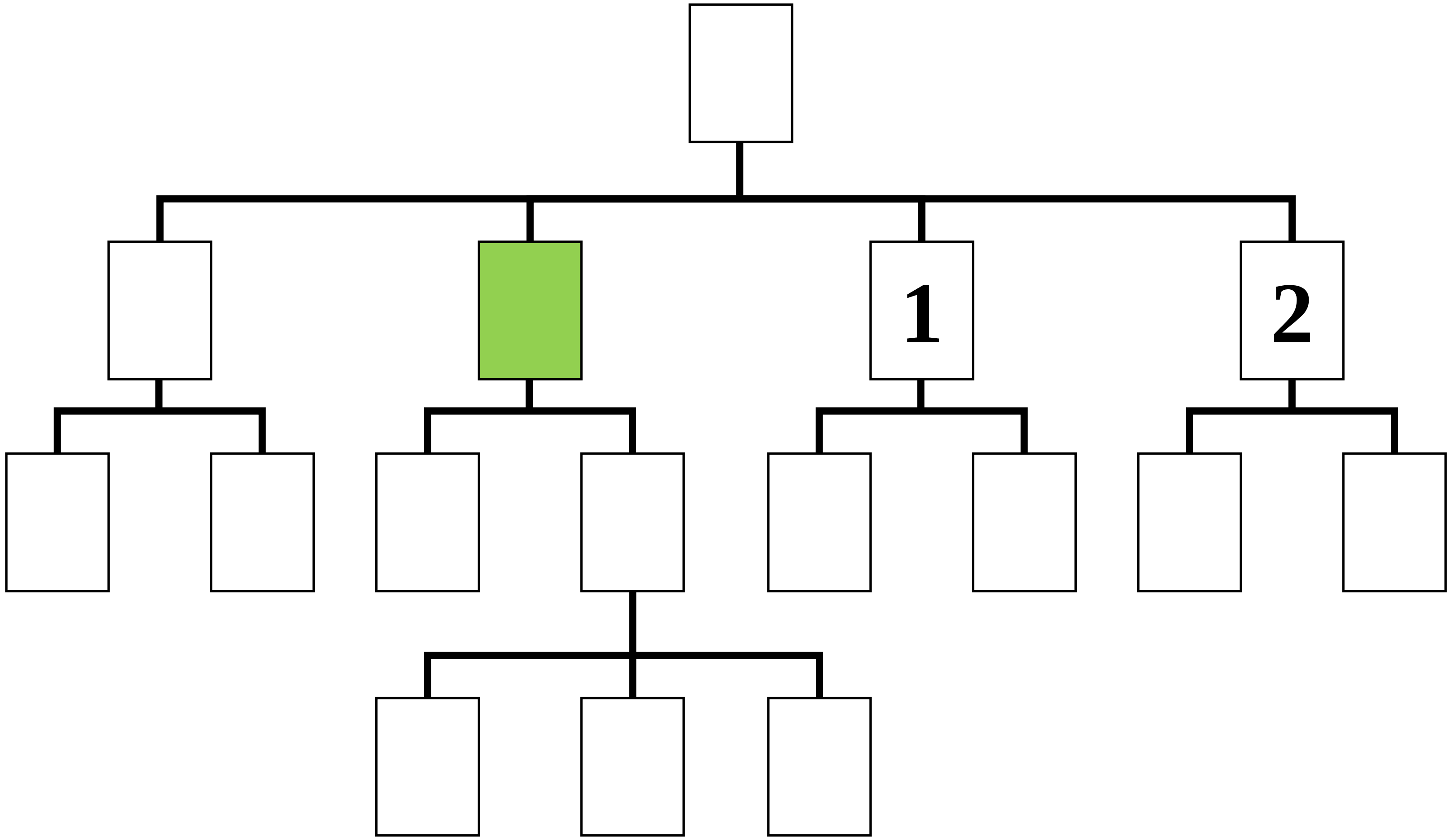
Descendant



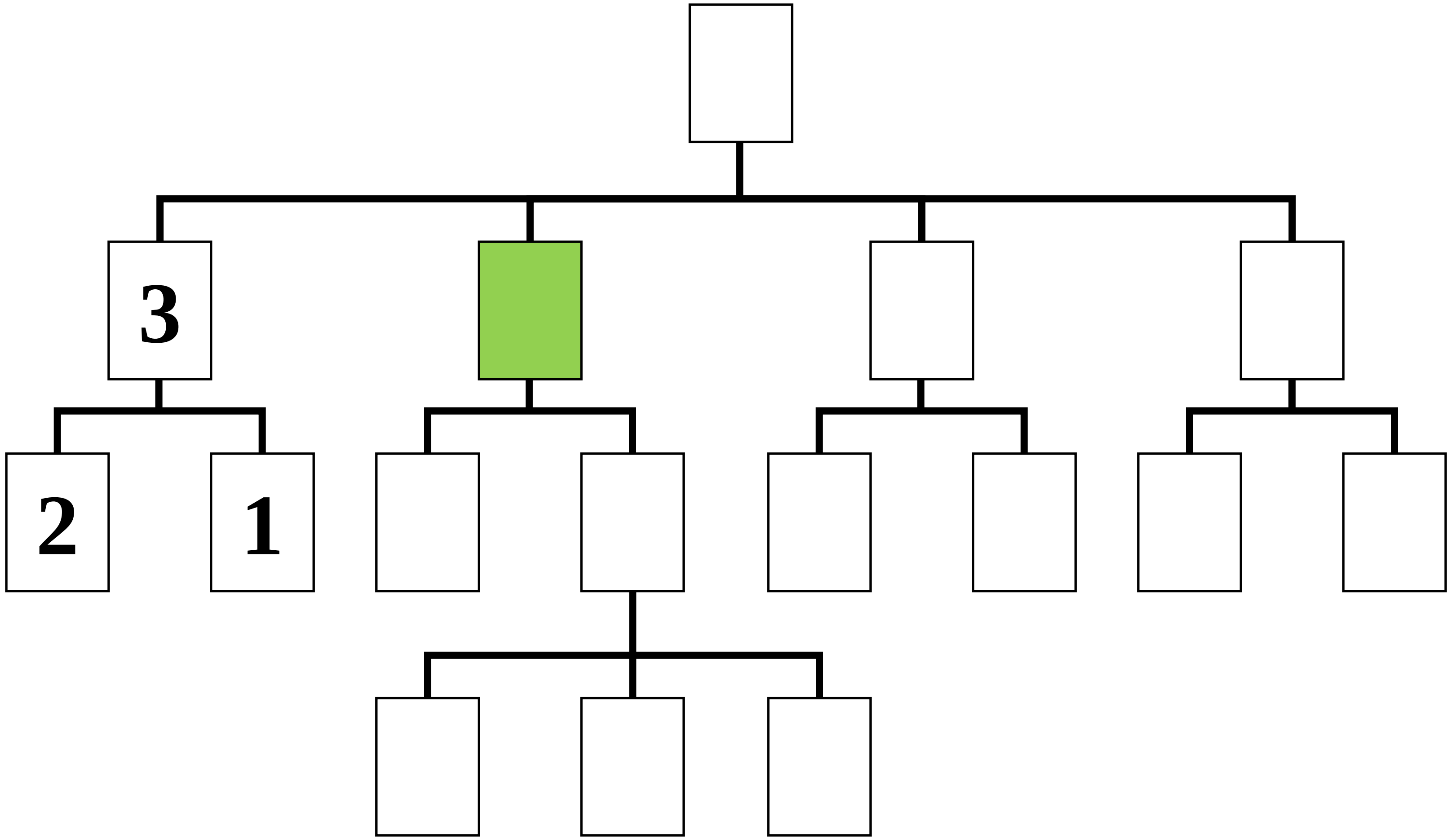
Following



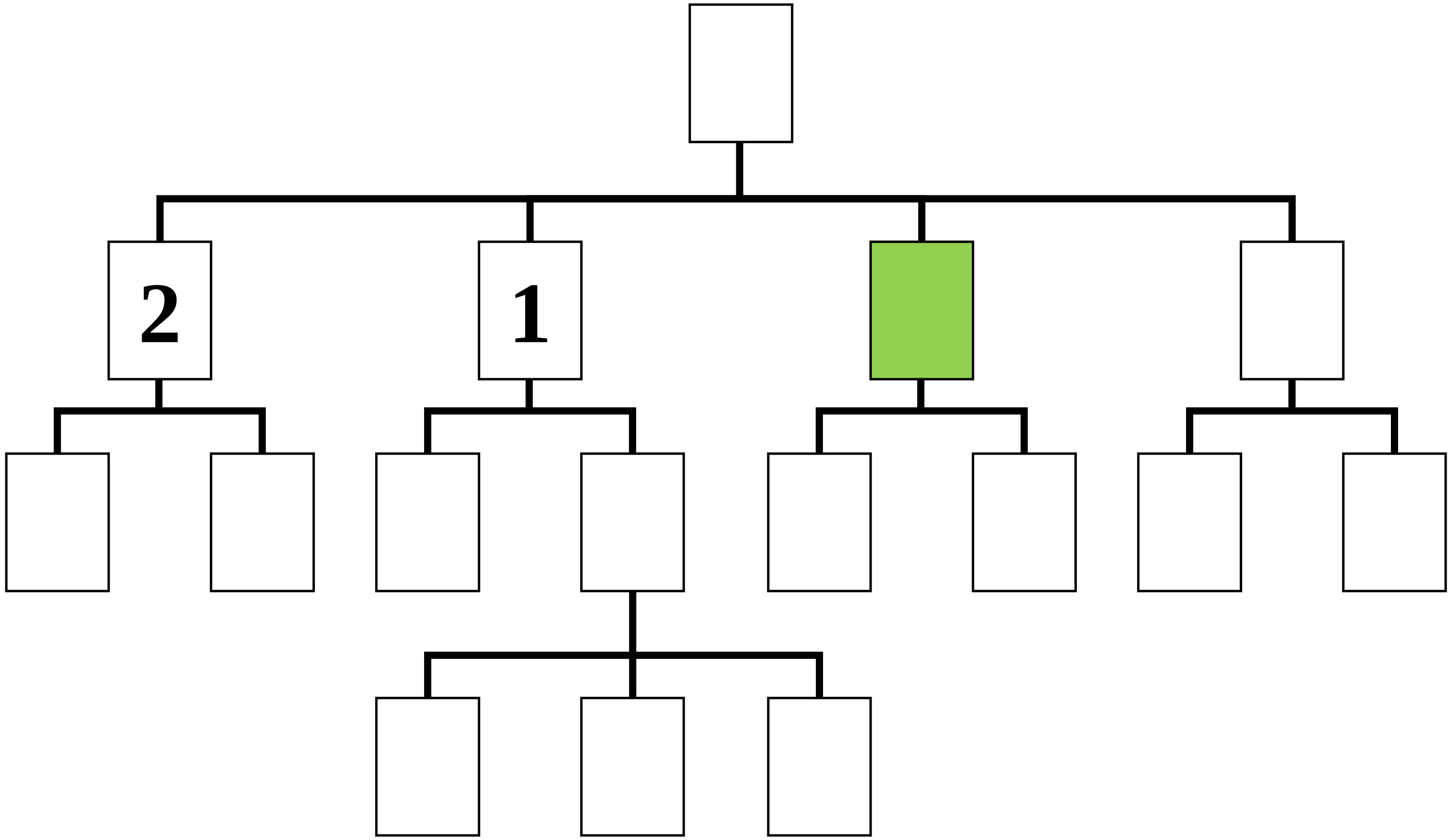
Following-Sibling



Preceding



Preceding-Sibling



DEMO

Example	Result
child::book	Selects all book nodes that are children of the current node
attribute::lang	Selects the lang attribute of the current node
child::*	Selects all element children of the current node
attribute::*	Selects all attributes of the current node
child::text()	Selects all text node children of the current node
child::node()	Selects all children of the current node
descendant::book	Selects all book descendants of the current node
ancestor::book	Selects all book ancestors of the current node
ancestor-or-self::book	Selects all book ancestors of the current node – and the current as well if it is a book node
child::*/child::price	Selects all price grandchildren of the current node

XPath Operators

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

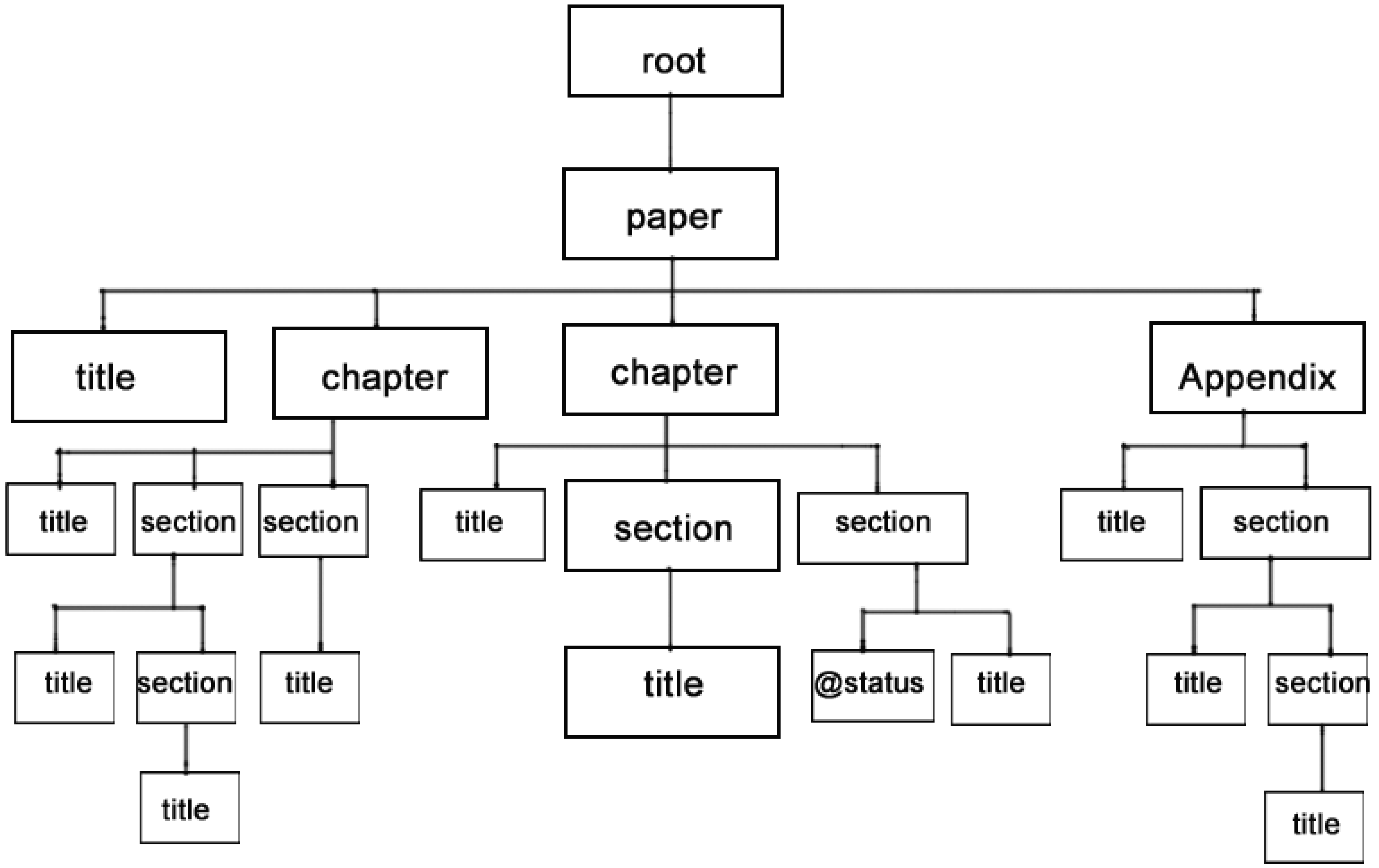
Operator	Description	Example
|	Computes two node-sets	//book | //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
or	Or	price=9.80 or price=9.70
and	And	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

XPath Operators

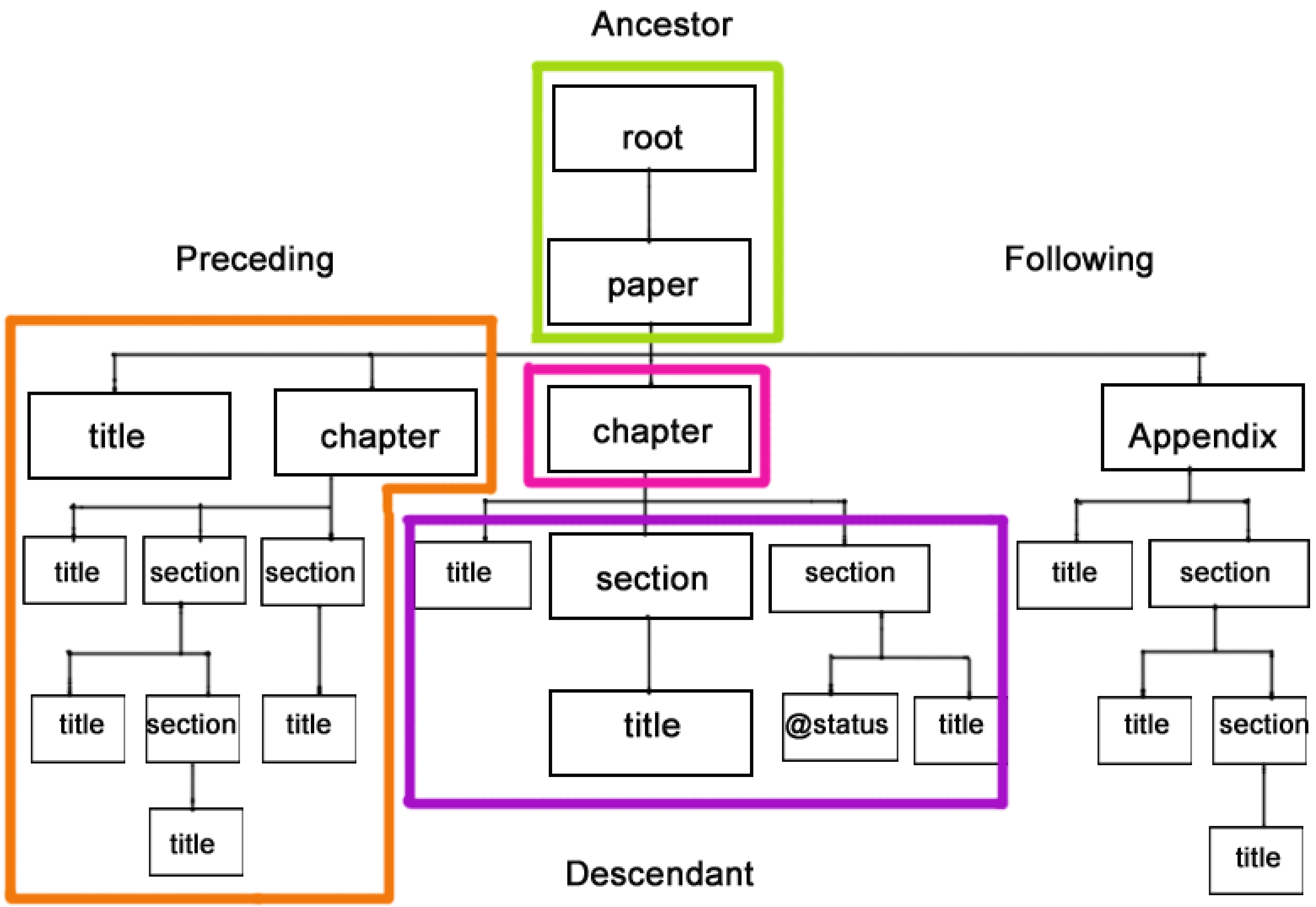
- An XPath expression returns either a node-set, a string, a boolean, or a number.

Operator	Description	Example
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80

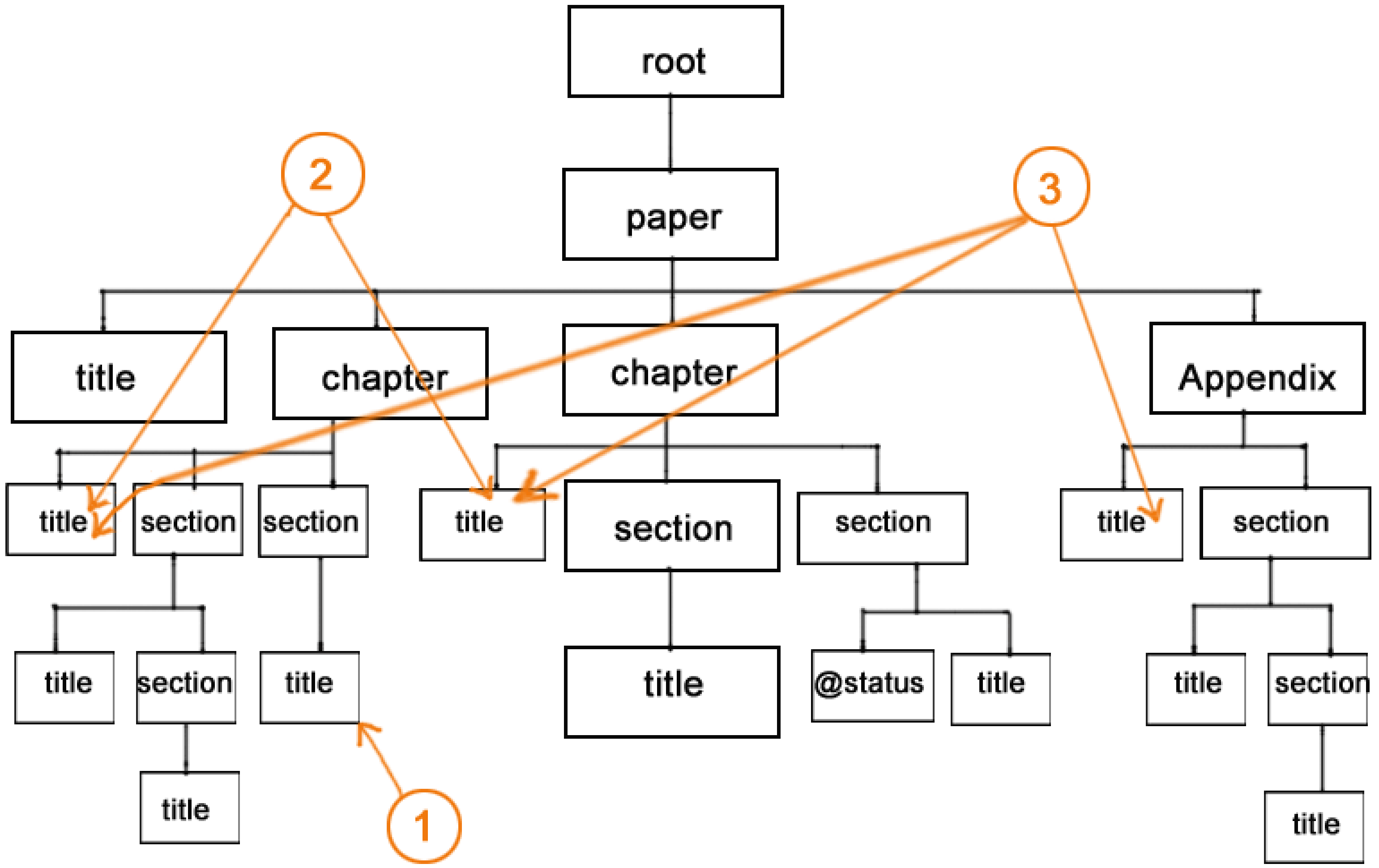
Example


root
paper
title
chapter
chapter
Appendix
title
section
section
title
section
section
title
section
title
section
title
title
@status
title
title
section
title
title

Example


Ancestor
root
paper
Preceding
Following
title
chapter
chapter
Appendix
title
section
section
title
section
section
title
section
title
section
title
title
@status
title
title
section
title
title
Descendant

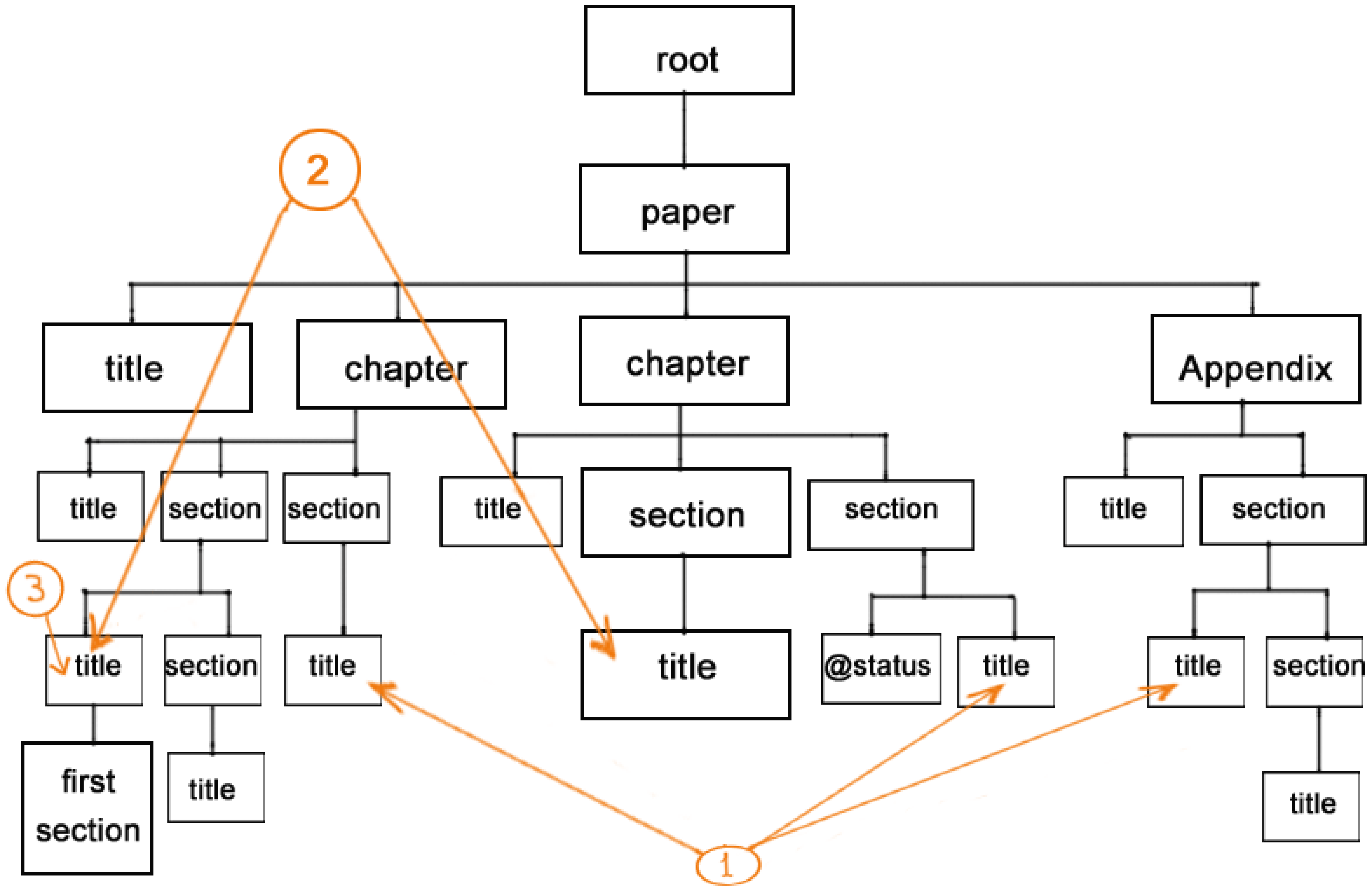
Question 1


root
paper
title
chapter
chapter
Appendix
title
section
section
title
section
section
title
section
title
section
title
title
@status
title
title
section
title
title
1
2
3

Solution 1

1. /paper/chapter[1]/section[2]/title
2. /paper/chapter/title
3. /paper/*/title

Question 2

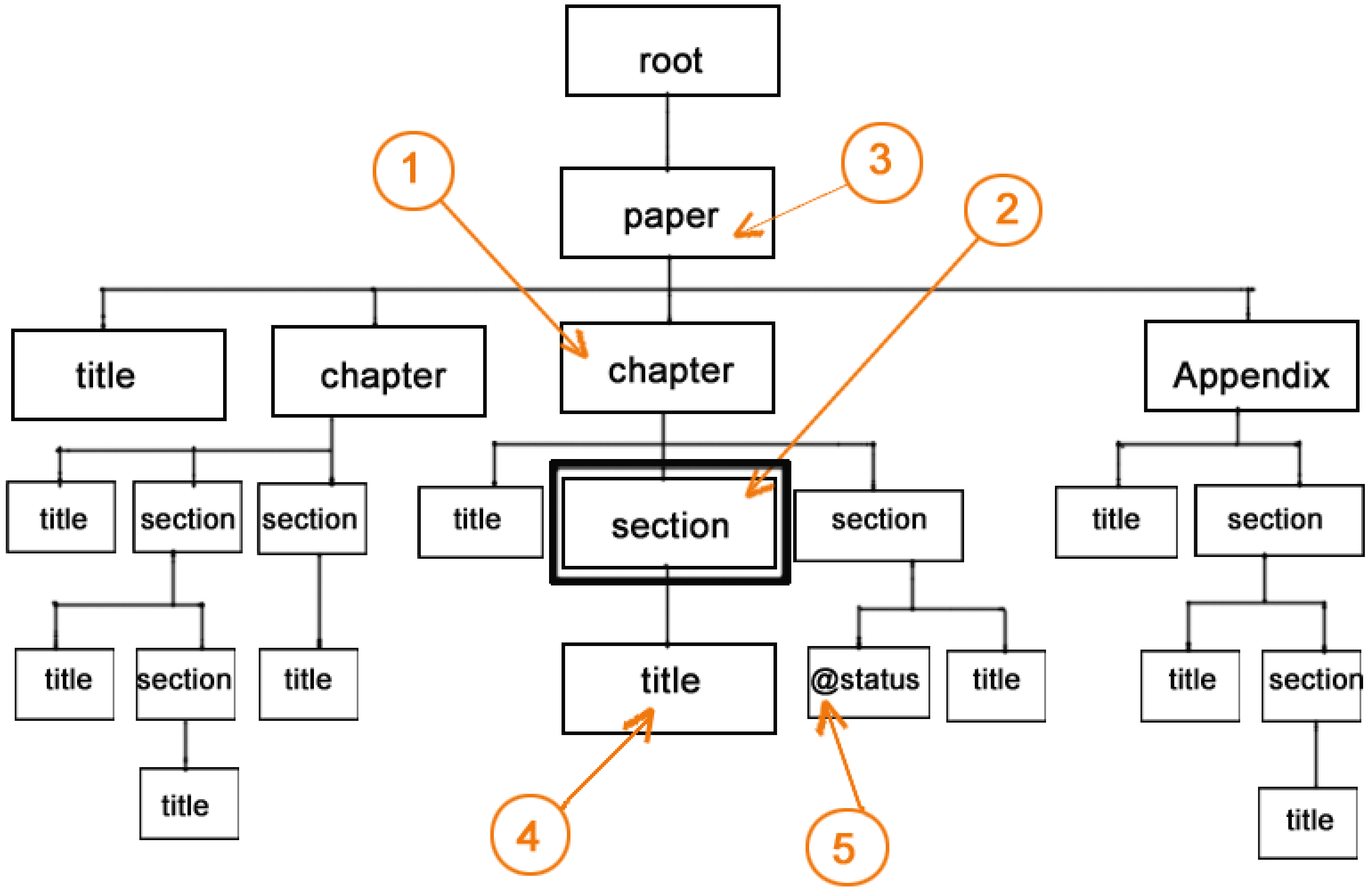


root
paper
title
chapter
chapter
Appendix
title
section
section
title
section
section
title
section
title
section
title
title
@status
title
title
section
first section
title
title
2
3
1

Solution 2

1. /paper/*/section [last()]/title
2. /paper/*/section [last()-1]/title
3. /paper/chapter[1]/section[1]/title

Question 3


root
paper
title
chapter
chapter
Appendix
title
section
section
title
section
section
title
section
title
section
title
title
section
title
@status
title
title
section
title
title
1
2
3
4
5

Solution 3

1. parent::node() or ..
2. self:: or .
3. ../..
4. child::node()
5. ./following-sibling::node()/@status

XPath Demo

- Using: ***book-store.xml***

- Write a suitable XPath Expression for following use cases:
  - Select all the titles
    - /bookstore/book/title
  - Select the title of the first book
    - /bookstore/book[1]/title
  - Select all the prices
    - /bookstore/book/price[text()]
  - Select price nodes with price>35
    - /bookstore/book[price>35]/price
  - Select title nodes with price>35
    - /bookstore/book[price>35]/title

Assignment

- Using: ***cd-catalog.xml***

- Write a suitable XPath Expression for following use cases:
    - Select all the CD's titles with price more than 10$
    - Select all the CDs that came before 1990
    - Select the titles and prices of all the CDs from "UK"
    - Select the artists names in the CDs that came before "Dolly Parton" 's CD
    - Select the titles of all the CDs after the "Private Dancer" CD

Chapter 5

Java™ Education and Technology Services

XSLT

XML Technologies

- **XML** is a language for **marking** documents.
- **DTD** is a way to **validate** XML documents.
- **XSD** is a more powerful way to **validate** XML documents.
- **XPath** is an expression format for **navigating** XML documents.
- **XQuery** is a way to **query** XML documents for information.
- **XSL** is a language for **styling** XML documents.
- **XSLT** is a language for **transforming** XML documents.

What is XSL ?

- **XSL** stands for e**X**tensible **S**tylesheet **L**anguage.

- XSL is an XML-based Stylesheet Language.
- XSL describes how the XML elements should be displayed.

- XSL consists of four parts:
  - XSLT - a language for transforming XML documents
  - XPath - a language for navigating in XML documents
  - XSL-FO - a language for formatting XML documents (discontinued in 2013)
  - XQuery - a language for querying XML documents

What is **XSLT** ?

- **XSLT** stands for **XSL T**ransformations
- XSLT is the most important part of XSL
- XSLT transforms an XML document into another document
- XSLT uses XPath to navigate in XML documents
- XSLT is a W3C Recommendation

XSLT = XSL Transformations

- XSLT is used to transform an XML document into another document.
- With XSLT you can add/remove elements and attributes to or from the output file.
- You can also rearrange and sort elements, perform tests and make decisions about which elements to hide and display, and a lot more.
- ***A common way to describe the transformation process is to say that XSLT transforms an XML source-tree into an XML result-tree.***

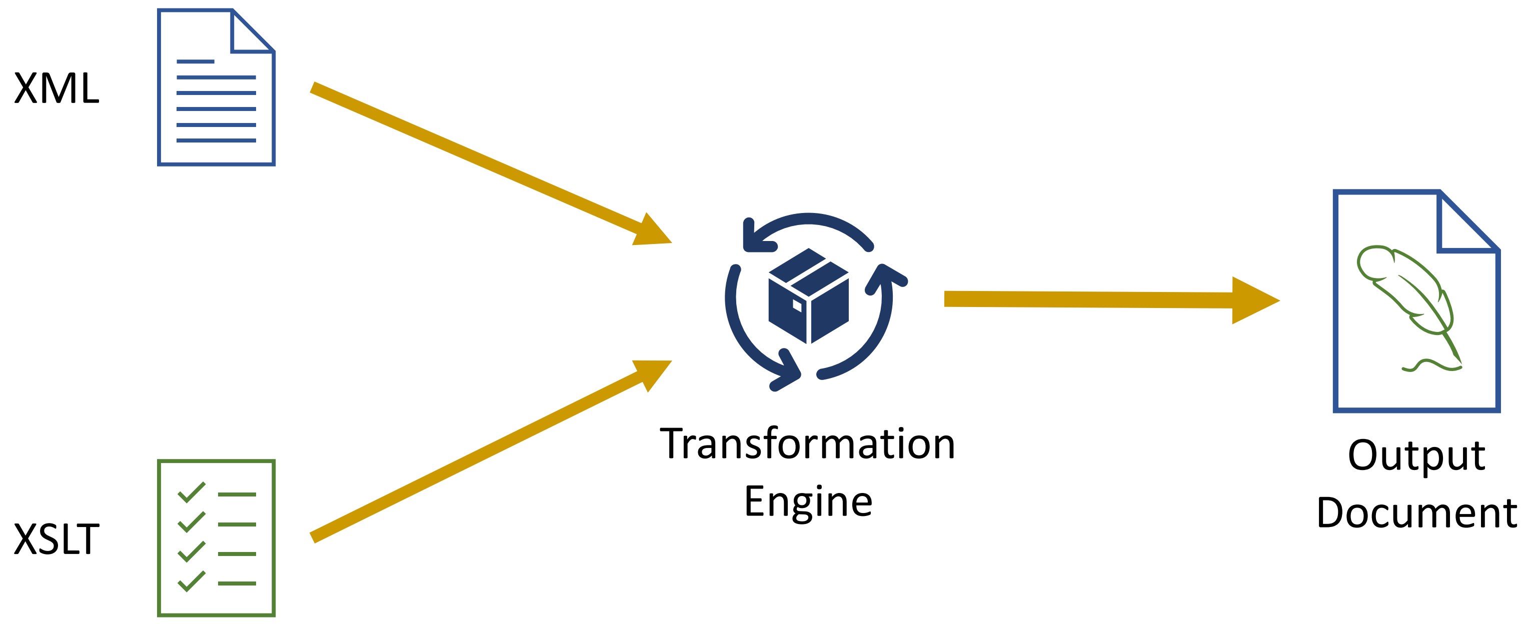
XSLT Uses XPath

- XSLT uses XPath to find information in an XML document.
- XPath is used to navigate through elements and attributes in XML documents.

- How Does it Work?
  - In the transformation process, XSLT uses XPath to define parts of the source document that should match one or more predefined templates.
  - When a match is found, XSLT will transform the matching part of the source document into the result document.

XSLT Transformation Engines

- If you have an XSLT compliant browser
  it will nicely transform your XML into XHTML.
- Most Browsers nowadays will not handle XSLT transformation.



How To use XSLT ?

1) Correct Style Sheet Declaration
2) Start with a Raw XML Document
3) Create an XSL Style Sheet
4) Link the XSL Style Sheet to the XML Document

Correct Style Sheet Declaration

- The root element that declares the document to be an XSL style sheet is ***<xsl:stylesheet>*** or ***<xsl:transform>***. either can be used!
- The correct way to declare an XSL style sheet, the W3C XSLT Recommendation is:

```
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- or:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

- To get access to the XSLT elements, attributes and features we must declare the XSLT namespace at the top of the document.
- The xmlns:xsl="http://www.w3.org/1999/XSL/Transform" points to the official W3C XSLT namespace.
- If you use this namespace, you must also include the attribute version="1.0".

Start with a Raw XML Document

- We want to transform the following XML document into XHTML:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
 <cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
 </cd>
  …
  …
</catalog>
```

- You’ll be provided the full ***cd-catalog.xml*** file in the course materials.

Create an XSL Style Sheet

- Create an XSL Style Sheet ("***cd-catalog.xslt***") with a transformation template:

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="/">
    <html>
      <body>
        <h2>My CD Collection</h2>
         ..
         ..
      </body>
    </html>
    </xsl:template>

</xsl:stylesheet>
```

- You'll be provided the full ***cd-catalog.xslt*** file in the course materials.

Link the XSL Style Sheet to the XML Document

- Add the XSL style sheet reference to your XML document:

```
<?xml version="1.0" encoding="UTF-8"?>

<?xml-stylesheet type="text/xsl" href="cd-catalog.xslt"?>

<catalog>
  <cd>
    <title>Empire Burlesque</title>
    <artist>Bob Dylan</artist>
    <country>USA</country>
    <company>Columbia</company>
    <price>10.90</price>
    <year>1985</year>
  </cd>
      ...
      ...
</catalog>
```

XSLT Elements

- XSLT **<xsl:template>** Element
- XSLT **<xsl:value-of>** Element
- XSLT **<xsl:for-each>** Element
- XSLT **<xsl:sort>** Element
- XSLT **<xsl:if>** Element
- XSLT **<xsl:choose>** Element
- XSLT **<xsl:apply-templates>** Element

XSLT <xsl:template> Element

- An XSL style sheet consists of one or more set of rules that are called templates.
- A template contains rules to apply when a specified node is matched.
- The **<xsl:template>** element is used to build templates.
- The match attribute is used to associate a template with an XML element.
- The match attribute can also be used to define a template for the entire XML document.
- The value of the match attribute is an XPath expression (i.e., **match="/"** defines the whole document).

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <xsl:template match="/">
                <html>
                        <body>
                                <h2>My CD Collection</h2>
                                <table border="1">
                                        <tr bgcolor="#9acd32">
                                                <th>Title</th>
                                                <th>Artist</th>
                                        </tr>
                                        <tr>
                                                <td>.</td>
                                                <td>.</td>
                                        </tr>
                                </table>
                        </body>
                </html>
        </xsl:template>

</xsl:stylesheet>
```

Example Explained

- The element, **<xsl:stylesheet>**, defines that this document is an XSLT style sheet document (along with the version number and XSLT namespace attributes).

- The **<xsl:template>** element defines a template. The match="/" attribute associates the template with the root of the XML source document.

- The content inside the **<xsl:template>** element defines some HTML to write to the output.

- The last two lines define the end of the template and the end of the style sheet.

- The result from this example was a little disappointing, because no data was copied from the XML document to the output.

XSLT <xsl:value-of> Element

- The **<xsl:value-of>** element can be used to extract the value of an XML element and add it to the output stream of the transformation:

```
...
          <table border="1">
                    <tr bgcolor="#9acd32">
                    <th>Title</th>
                    <th>Artist</th>
                    </tr>
                    <tr>
                    <td>
                              <xsl:value-of select="catalog/cd/title" />
                    </td>
                    <td>
                              <xsl:value-of select="catalog/cd/artist" />
                    </td>
                    </tr>
          </table>
...
```

XSLT **<xsl:for-each>** Element

- **<xsl:for-each>** is used to select every XML element of a specified node-set

```
…
          <table border="1">
          <tr bgcolor="#9acd32">
                    <th>Title</th>
                    <th>Artist</th>
          </tr>
          <xsl:for-each select="catalog/cd[artist='Bob Dylan']">
                    <tr>
                    <td>
                                        <xsl:value-of select="title" />
                    </td>
                    <td>
                                        <xsl:value-of select="artist" />
                    </td>
                    </tr>
          </xsl:for-each>
          </table>
…
```

XSLT **<xsl:sort>** Element

- To sort the output, add an **<xsl:sort>** element inside the **<xsl:for-each>** element in the XSL file, *The **select** attribute indicates what XML element to sort on.*

```
...
            <table border="1">
            <tr bgcolor="#9acd32">
                        <th>Title</th>
                        <th>Artist</th>
            </tr>
            <xsl:for-each select="catalog/cd">
                        <xsl:sort select="artist" />
                        <tr>
                        <td> <xsl:value-of select="title" /> </td>
                        <td> <xsl:value-of select="artist" /> </td>
                        </tr>
            </xsl:for-each>
            </table>
...
```

XSLT **<xsl:if>** Element

```
...
            <table border="1">
            <tr bgcolor="#9acd32">
                        <th>Title</th>
                        <th>Artist</th>
                        <th>Price</th>
            </tr>
            <xsl:for-each select="catalog/cd">
                        <xsl:if test="price &gt; 10">
                                    <tr>
                                                <td><xsl:value-of select="title"/></td>
                                                <td><xsl:value-of select="artist"/></td>
                                                <td><xsl:value-of select="price"/></td>
                                    </tr>
                        </xsl:if>
            </xsl:for-each>
            </table>
...
```

XSLT <xsl:choose> Element

- The **<xsl:choose>** element is used in conjunction with **<xsl:when>** and **<xsl:otherwise>** to express multiple conditional tests.

```
<xsl:choose>

        <xsl:when test="expression1">
                ... some output ...
        </xsl:when>

        <xsl:when test="expression2">
                ... some output ...
        </xsl:when>

        <xsl:otherwise>
                ... some output ....
        </xsl:otherwise>

</xsl:choose>
```

```
...
   <table border="1">
      <tr bgcolor="#9acd32">
              <th>Title</th> <th>Artist</th>
      </tr>
      <xsl:for-each select="catalog/cd">
              <tr>
                          <td> <xsl:value-of select="title"/> </td>
                          <xsl:choose>
                          <xsl:when test="price &gt; 10">
                                    <td bgcolor="#ff00ff">
                                              <xsl:value-of select="artist"/>
                                                        </td>
                          </xsl:when>
                          <xsl:otherwise>
                                    <td>
                                                                  <xsl:value-of select="artist"/>
                                                        </td>
                          </xsl:otherwise>
                          </xsl:choose>
              </tr>
      </xsl:for-each>
   </table>
...
```

XSLT **<xsl:apply-templates>** Element

- The **<xsl:apply-templates>** element applies a template to the current element or to the current element's child nodes.
- If we add a "**select**" attribute to the **<xsl:apply-templates>** element, it will process only the child elements that matches the value of the attribute.
- We can use the "**select**" attribute to specify in which order the child nodes are to be processed.

Example

```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <xsl:template match="/">
                <html>
                        <body>
                                <h2>My CD Collection</h2>
                                <xsl:apply-templates/>
                        </body>
                </html>
        </xsl:template>

        <xsl:template match="cd">
                <p>
                        <xsl:apply-templates select="title"/>
                        <xsl:apply-templates select="artist"/>
                </p>
        </xsl:template>
```

Example

```
        <xsl:template match="title">
                Title:
                <span style="color:#ff0000">
                        <xsl:value-of select="."/>
                </span>
                <br />
        </xsl:template>

        <xsl:template match="artist">
                Artist:
                <span style="color:#00ff00">
                        <xsl:value-of select="."/>
                </span>
                <br />
        </xsl:template>

</xsl:stylesheet>
```

XSLT Demo

- Demo 1:
  - Display Title, Artist of all the CDs in the catalog.
- Demo 2:
  - Display Title, Price of CDs, colored according to their price.

Chapter 6

JSON

Introduction to JSON

- Stands for Java Script Object Notation

- What is JSON?
  - JSON is a syntax for storing and exchanging text information.
  - JSON is lightweight text-data interchange format
  - JSON is language independent.

- JSON is smaller than XML, and faster and easier to parse.

Introduction to JSON

- JSON Files:
  - The filename extension is **".json"**
  - JSON Internet Media type is **"application/json"**

- All modern programming languages support JSON.

JSON vs XML

- Much Like XML:
  - JSON is plain text
  - JSON is "self-describing" (human readable)
  - JSON is hierarchical (values within values)
  - JSON can be parsed by JavaScript
  - JSON data can be transported using AJAX

- Much Unlike XML:
  - No end tag
  - Shorter
  - Quicker to read and write
  - Can be parsed using built-in JavaScript eval()
  - Uses arrays
  - No reserved words

Why JSON?

- For AJAX applications, JSON is faster & easier than XML:

- Using XML
  - Fetch an XML document
  - Use the XML DOM to loop through the document
  - Extract values and store in variables

- Using JSON
  - Fetch a JSON string
  - eval() the JSON string

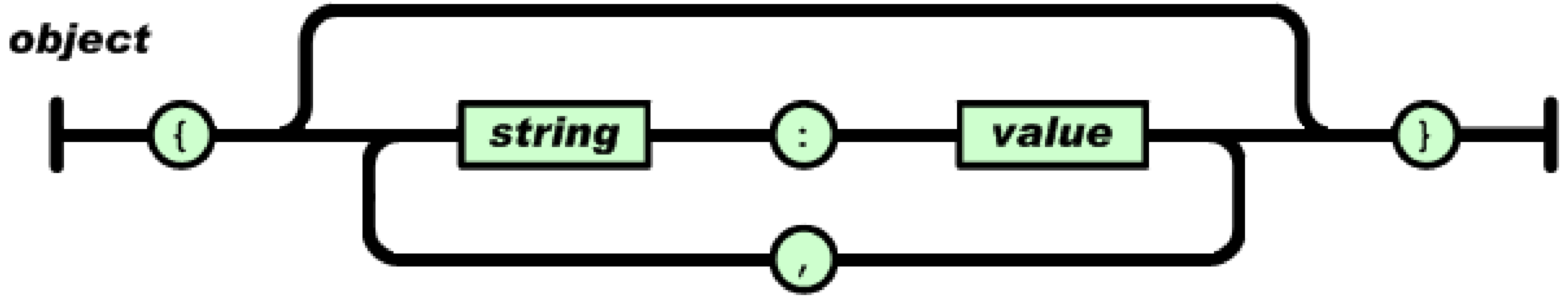
Example

```
{
   "book": [
   {
      "id":100,
      "language":"Java",
      "edition":"third",
      "author":"Herbert Schildt"
   },
   {
      "id":200,
      "language":"C++",
      "edition":"second‶,
      "author":"E.Balagurusamy"
   }]
}
```

JSON Structure

- **JSON is built on two structures:**
  - ***A collection of name/value pairs.***
    - Like an *object*, record, struct, dictionary, hash table.
  - ***An ordered list of values.***
    - Like an *array*, vector, list, or sequence.

Object

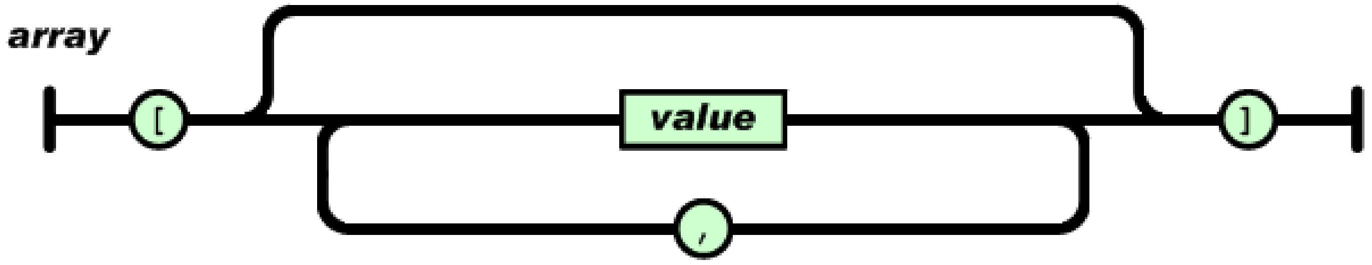


- An ***object***
  - is an unordered set of name/value pairs.
  - An object begins with **{** and ends with **}**
  - Each name is followed by **:** and the name/value pairs are separated by **,**
- Example:

```
{ "firstName":“Ahmed" , "lastName":“Omar" }
```

Array

- **An *array***
  - is an ordered collection of values.
  - An array begins with [and ends with]
  - Values are separated by ,



- **Example:**

```
{
  "employees": [
    {"firstName":"John","lastName":"Doe"},
    {"firstName":"Anna","lastName":"Smith"},
    {"firstName":"Peter","lastName":"Jones"}
  ]
}
```

Value

- A ***value*** can be
  - a *string* in double quotes,
  - a *number*,
  - true or false
  - null,
  - an *object*
  - an *array*

```
name:"Amin"
```

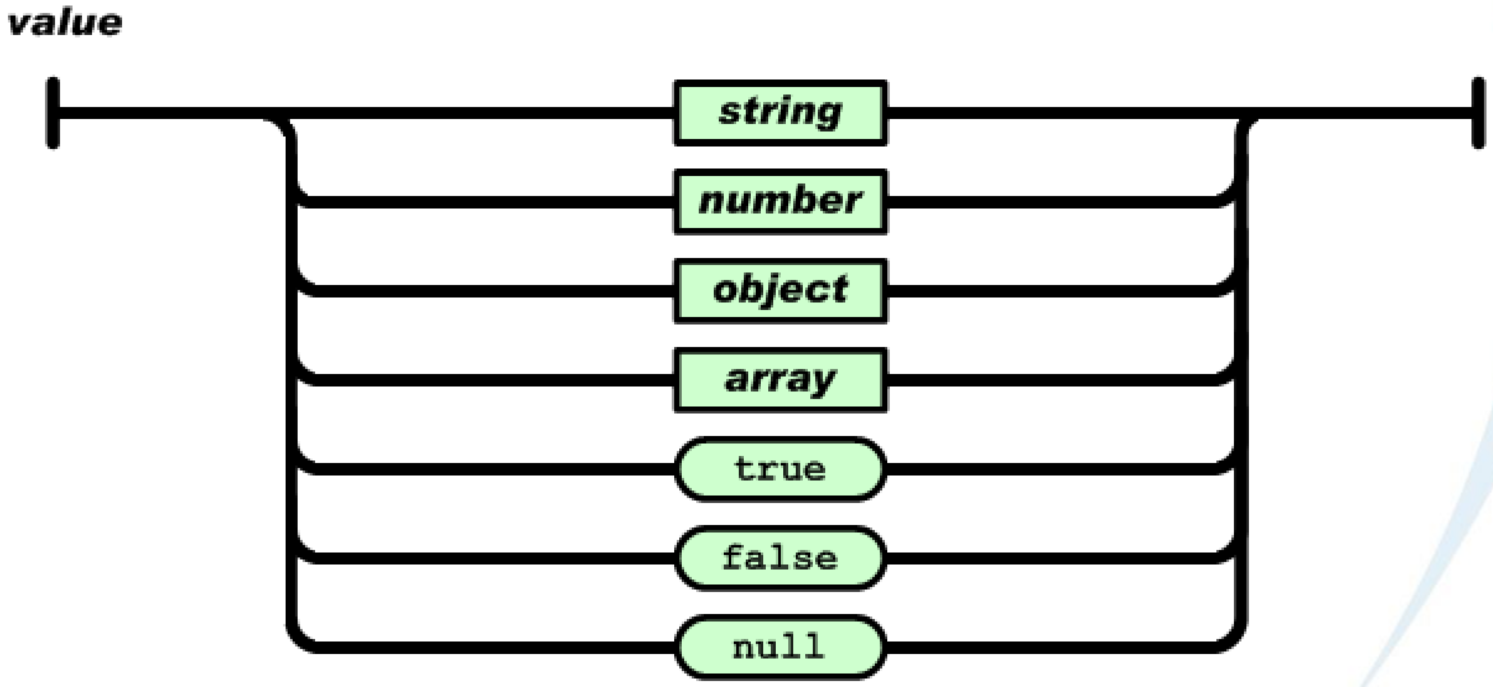
```
id:100
```

```
flag:true
```

```
MyVar:null
```

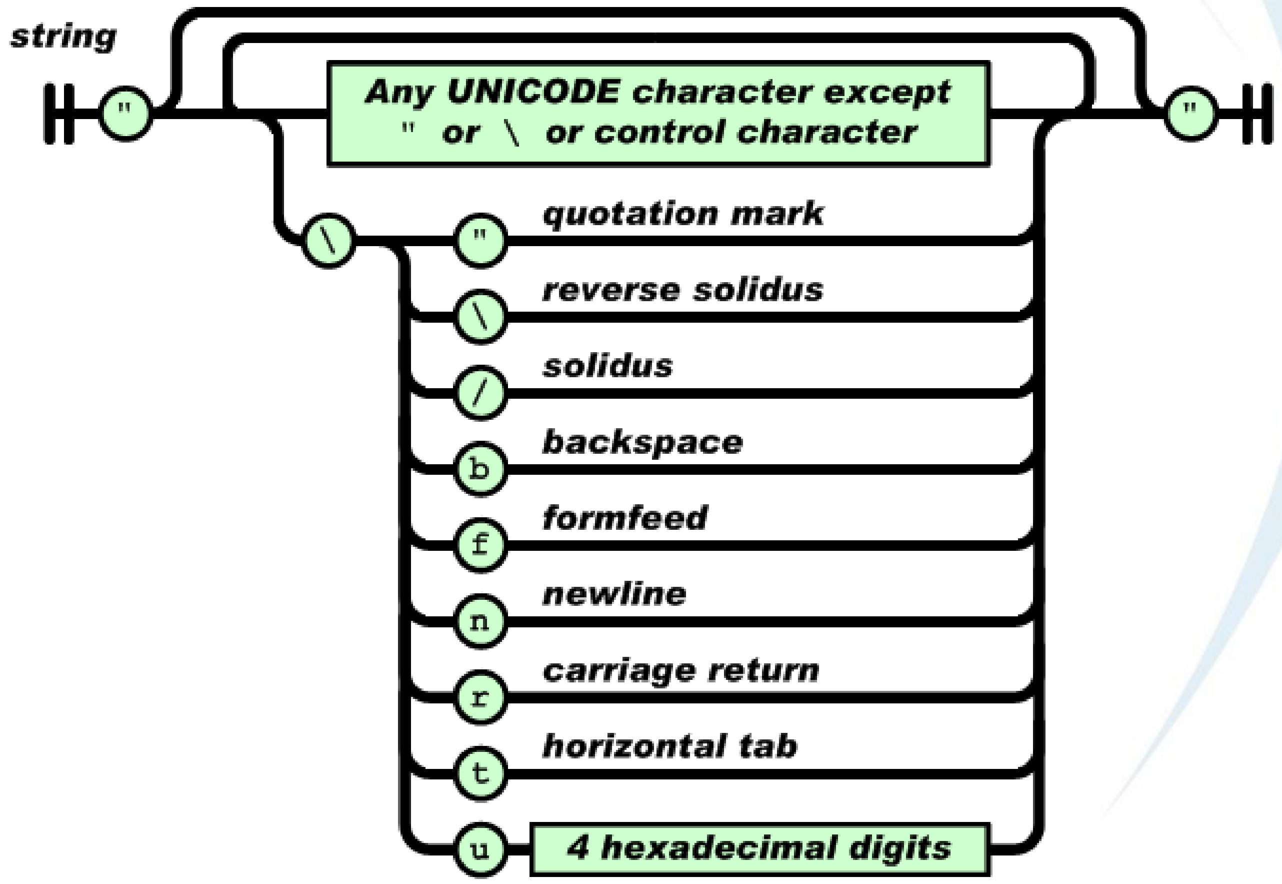
```
{"id":1,"x":"y"}
```

```
"Students":[...,...]
```



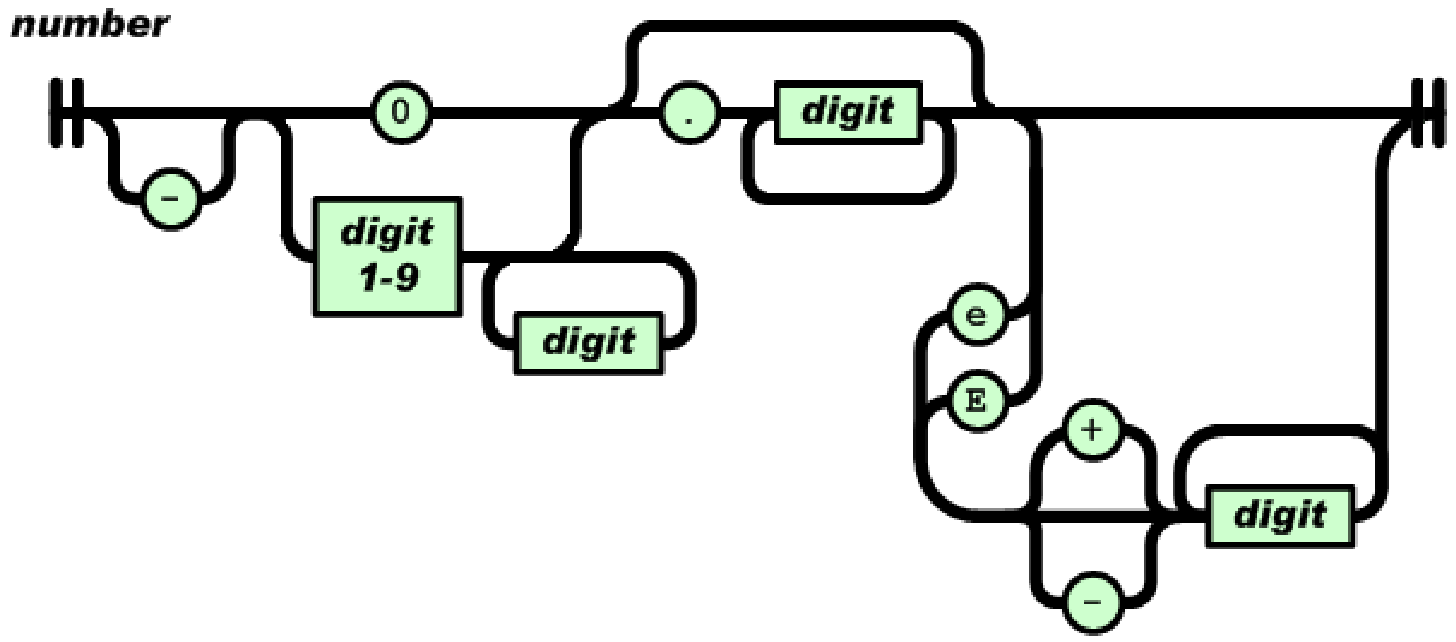
String

- A ***string*** is
  - Very much like a C or Java string.
  - A sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes.
  - A character is represented as a single character string.



Number

- A ***number*** is
  - Very much like a C or Java number.
  - Except that the octal and hexadecimal formats are not used.



JSON Technologies

- JSON started unlike XML without any Validation Rules

- After the widespread usage of JSON,
  new Technologies emerged related to JSON similar to XML Technologies.

- **JSON Schema**: to validate JSON Documents
- **JSON Pointer**: to navigate and fetch data from JSON Documents
- And new JSON Technologies are still emerging

- Both XML and JSON are widely Adopted Data Exchange Formats

Assignment: A Configuration File

- Design a configuration file for a library.
  - Info. of library consists of a location, a description of the library, a librarian and a lot of books.
  - Each book has title, ISBN, and Author.
  - The book contains also a preface and many of parts.
  - Each part has title and contains many of chapters.
  - Each chapter has title and contains a summary and many of sections.
  - Sections contain the content of the book as paragraphs.