1. XML Schema Basics

In my house I have a bookcase with the capacity of 5 shelves, but I only use two of them. One contains technical books of all kinds, like dictionaries (French, English and Portuguese), academic books (computer languages - like C, Java and Web languages - statistics, maths and physics). These are ordered by sizes and types.
The other shelf contains non-technical books and magazines. Some comic books ("Astérix and Obélix", "Tintin", "Lucky Luke" and "Where's Waldo?"), which are very used and full of cookie crumbs, history and leisure books. These last ones are usually bought when I find them interesting and fun to read.
Since I'm studying abroad, when I feel homesick I take a Portuguese book just to feel more homey.

After defining our story, we could see which components were important to include in the xml schema. Here's a flow chart that describes our implementation strategy:

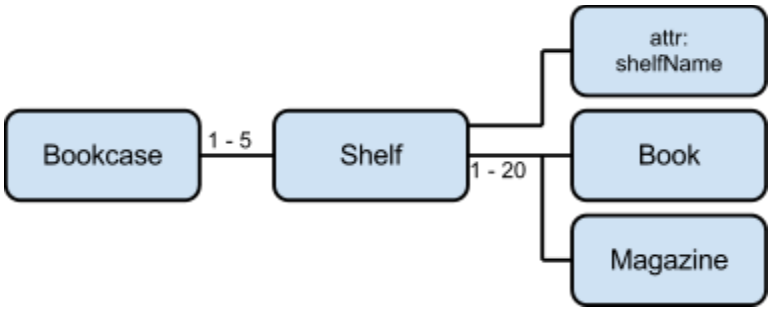

Image 1 - Representation of the composition of the bookcase and each shelf

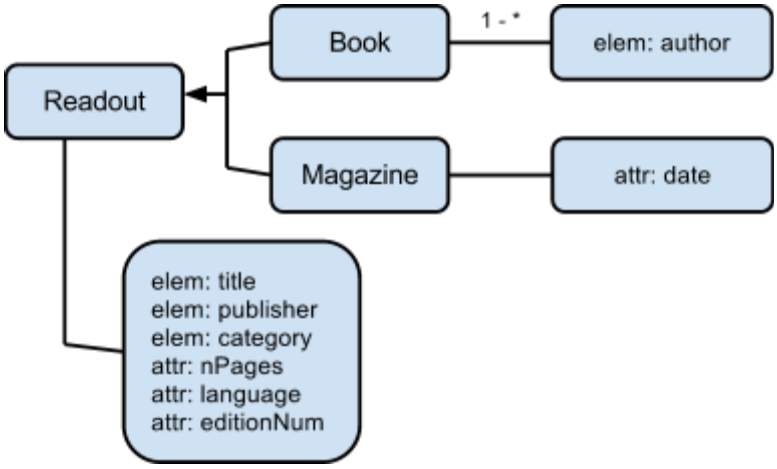

Image 2 - Representation of the composition of the book and the magazine

In order to implement the *extension* requirement, the books and magazines were grouped in readouts, since the elements and attributes that define them are almost the same. As can be seen in *Image 2* (above), either books and magazines have an additional element/attribute that is defined separately. In the case of the books, there are the authors, which are not needed in defining the magazines and in the case of the magazines there's the publish date.

Due to the existence of many different basic elements and attributes, the basic schema (the one that defines the bookcase) was defined in one xsd document (*Bookcase.xsd*) and the elements that define the types included in the basic schema were defined in another schema (*BookcaseBasicTypes.xsd*), so that the *import other schemas* requirement could be fulfilled and the program could be more organized.

2.Programmatic Approach to XML

Brief explanation of the functions:
The main program is the *BookcaseParser.java,* which defines the functions validate and list.

The validate function (named *xmlValidator*) will validate the given xml file against the schema defined previously. Briefly, it will check if the document is well-formed according to the XML rules and after it will parse the xml file against the schema (this feature is provided by the *validate* Java function - from javax.xml.validation.Validator package).

This java file also defines another function: list (named *listing*). What it does is, it lists the corresponding books and magazines existing in the bookcase. All these components, elements and attributes, are defined in the xml File (*MyBookcase.xml*).

This function uses the javax.xml.parsers.* package, parsing the xml file into a DOM, in order to be possible to process the XML document. Otherwise it wouldn't be possible to get any values from it, since the main focus of XML is to carry information and not display it.

Usage Instructions:
In order for the program to work without any errors, all the files (.xsd, .xml and .java) must be in the same directory.

The first thing to be done is compile the java file, where the functions are defined. In order to do so, write the next line in the command line:

```
> javac BookcaseParser.java
```

At this point, no errors should be found and the file must compile correctly.

To execute the program, the sample command line is:

```
> java BookcaseParser [xyz].xml [command]
```

Where `[xyz]` will correspond to the name of the xml file, which in this case will be *MyBookcase* and `[command]` will correspond to the available functions, *validate* and *list* (explained earlier).
So,
to validate:

```
> java BookcaseParser MyBookcase.xml validate
```

This will print a sentence in the window, informing the user if the given xml file is valid or not. If not, a more complete information line is printed.

to list:

```
> java BookcaseParser MyBookcase.xml list
```

The result of this line is, first saying if the xml is valid and then, if valid, print the components of the existing shelves in the bookcase.

Known bugs:
- The fact that the shelf is defined as a sequence of *Book* and *Magazine* (*Bookcase.xsd*), limits the creation of the components in the shelf. For instance, if we want to define a shelf without any magazines it is possible, but if we want to define 3 books and 1 magazine (in one shelf) it will not work, because, as said before, it is defined as a sequence.

Learning Experience:
How did you do the exercise?
We started by writing the story, sharing ideas between us, putting them on paper (to see if they were feasible). Some of the things had to be changed, so the story was reformulated.
After this part, the xsd files were written based on the story and the list of requirements. After this, we were able to write the xml file (also according to the story).
In order to test if the xml file was valid, we had to write in java a function that tested this function. To do so, we had to reserve some time studying the operation and the method of resolution of this problem, finally we came up with a solution. When this function was finally working properly, we started doing the listing function. After all of the commands were implemented, we tested our program to find bugs and correct them.

What was easy?
- Coming up with the story
- Creating the XML Schema and the XML documents

What was difficult?
- Writing the validate function, because it needed some understanding of the features of Java
- Writing the listing function (even though it was easier than the previous one, we had to reformulate the story and consequently the schema)

Working hours per person:
We worked as a group, so we worked the same amount of time, which was in average per item:
- Schema Basics:
    - Story + xml schema: 8 hours
- Programming Approach to XML:
    - XML document: 30 minutes
    - Java:
        - validate function : 7h
        - list function: 2h
    - report (documentation + README): 2h
- Tests (bug fix): 20 minutes
- Studying: 5h

TOTAL: approx. 25 hours