Assignment 1

Report

A. Pre-documentation

i. XML schema description

[Write your answer here, try to be concise]

The task presents three example XML documents with data from 1994, 2003 and 2020. Despite these three documents being from different years, they are all consistent with their formats as they don't seem to vary at all. The documents consist of a collection of sales data organised within several elements. The root element or root node is called 'sales' and has a sub-element 'receipt'. 'Receipt' is then the parent node of 'when', 'seller', 'where' and 'items'. 'Where' and 'items' are the only nodes that have children which include 'name', 'address', 'phone' (all of these from 'Where' node). On the other hand 'items' has a single child: 'item' which in itself has 4 more children. These are 'name', 'price currency', 'quantity' and 'tax percentage'. This results in the documents having a height of 5 at its maximum depth when we look at the XML in a tree-like structure. I believe all the data in the XML files are important and don't think we could get rid of any of it when representing it into a JSON file. However one solution if data had to be condensed would be having a separate table to hold the data in all of the 'Where' nodes since the data within it does not change. For example every time we see Where id="MIT", the children from it will always be the same.

The values do change if the id is different but for every different id, the values will remain the same.

As mentioned, having a different table with all this information would be appropriate in order to shorten the documents. Unlike "where id", other ids do uniquely identify an entry in the XML file and thus we cannot get rid of them.

An appropriate XML schema should be clear so that it is easy to understand, flexible in order to update/change data and descriptive when it comes to its elements so that there are no misunderstandings. An XML schema says that the file should be consistent and reusable. Despite an XML not needing a schema to be valid, I would say that the documents provided do follow the schema since they are very easy to understand (through the use of tags and the ids mentioned before) and are fully consistent throughout, making these XML files valid.

ii. JSON structure recommendation

[Write your answer here, try to be concise]

The structure of a JSON object would typically be composed from name-value pairs, where there is a name identifying an item followed by a colon which then includes the value of that item. These pairs are separated by commas, and are all within braces such as "{" and "}". Some JSON objects may have an array as 'value' where there are square brackets "[" and "]" , enclosing multiple values for a given item. However, I don't expect any of these when

converting the XML file into JSON since I could only see one value for every item in the sample files. However, I am expecting a nested JSON object since there are elements in the XML file that have children while them already being children. A nested JSON object happens when there is another JSON object (name-value pair) as the value of another pair. However, nesting JSON objects is not mandatory and is therefore not too important when it comes to producing the JSON file. Also, since the file is intended to be read by a machine, it really does not matter whether the final JSON is human readable or not.

B. Post-documentation

[Write your answer here, max 1000 words]
The company asked for a program that would parse and convert an XML file into a JSON file in order to port the data to the new system. It is mentioned that the program should be fast and efficient as well as space saving. The two main ways of parsing an XML file in java, DOM and SAX, both have their strengths and weaknesses. DOM would be great choice if the amount of data was small, or if some document handling was needed (such as sorting parts of the file and moving them around). However, I believe SAX is the better approach for this task. SAX is not only faster, but also much more memory efficient which will be ideal since the company is asking for a space-saving solution. A SAX parser works by reading events off from the XML file as it goes through it, which are then needed to be captured in order to not lose any information from the original file. SAX does not save any of this information in memory. Differently, a DOM parser would create an internal structure in memory containing all the details from the parsed document, taking up a lot more space and making it less efficient than SAX. Another reason why SAX was the best alternative for this task was future proofing. As I looked through the documents, these grew in size exponentially as time passed by (meaning that the 1994 document is smaller than the 2003 document which is then smaller than the 2020 document). Therefore, one can only assume that future documents are going to be larger in size and if they are then required to be converted into JSON, SAX would do a faster and better job.

When it comes to my implementation of SAX parser, an extra java file was needed since it uses a handler (separate file extending a default handler) to go through all the XML data. In this handler file (called xmlHandler.java) some variables have been instantiated such as an ArrayList and a HashMap. It is interesting to see that most of the work is being done by that second file (xmlHandler.java) since the original file is only being used to call other methods.

Initially I used a conventional hash map since a hash map can offer great help when creating the SAX parser. A hash map is composed of nodes which can have keys and values (for every node), making it quite similar to the structure of a JSON object (name-value pair). Therefore, the conversion from an XML file into JSON becomes much easier. However, I ended up using a linked hash map. It provides the same advantages as a normal hash map as well as preserving the order in which elements are added into the map. This results in the same order of elements being shown in the final JSON object than the one seen in the XML. It also allows anyone to traverse through the map if necessary. Although it does use a bit more memory, the advantages outweigh the negatives and is therefore worth using since the alternatives would make the task a lot more difficult by not providing an identical order of elements in the final JSON file. The array list mentioned before is necessary in order to

properly add the values into the map. This works by retrieving these values from a new method. The final JSON file has a valid format where each children in the XML file is a new object. This means that my initial assumption of nested JSON objects was wrong as instead of nesting them, a new object was created for every set of elements in the XML. This may be slightly more complicated to understand if a human reads it (data is more spaced out) but the computer will have no problem processing it.

As mentioned, there are two different java files used in this task. The SimpleParser.java file has the main class from which the parser method is called and saved as a string variable, allowing us to print it out onto the final JSON document. Once the parser method is called, some necessary objects are instantiated such as 'SAXParserFactory' or the 'XMLReader'. The method then gets the given file path and then reads the actual file. After the file goes through the xmlHandler.java, the product is saved in a string which is then returned to the main method. Some lines of code were added in order to make sure that the XML was being processed, where it prints out the parsed file in the terminal. This allows the user to check that the XML has been read correctly and that no data is missing (this is not stored as a file and therefore barely takes any space in memory). However, if a more space-saving/smaller program is required, these output lines of code could be deleted in order to make it more efficient. In order to actually use the program, two arguments must be added on the run configurations. This can either be done through the users' IDE of choice, or simply through the terminal. This means that the program can run in any machine regardless of the IDE being used and ensures that all machines in the company will be able to take advantage of it.