Consolidated

M2 →

Task 1: writing parser for reading the xml to Json

not reading whole xml instead read till a pointer.

Task 2 : replace a certain value based on key.

we have 4 Testcases 2 for each task.

T1: our method takes this

first we get the proper Uri from the json pointer → if we have contatct/address/1/ this is invalid path so we return null.

if path is proper we form URI and split it put it in a array and return an array.

```
JSONObject jobj = XML.toJSONObject(new StringReader(xmlString), new JSONPointer("/contact/address/1"));
```

This method checks the arrays length if len == 0 return null. else call our parser

```
parse(x, jo, null, config, uri, new JSONObject(), 0, new HashMap<String, Integer>(), true);
```

we read the token one by one and get the content once we reach inside the content.

then it creates a new `JSONObject` and starts parsing the attributes of the tag.

We make a recursive call if we encounter a nested call If it encounters the end of the current tag `</...>` , it returns `true` this means the recursive call is ended.

Also we keep on accumulating until we reach the target once target path is reached we break.

we `ParseTerminationException` here we put the values into object and throw this saying we reached the target.

Also to handle the json array we are using a map to keep a key value pair mainly cause we encounter that end tag so we had difficulty in finding the solution for this.

For task 2 : we use the same parser but only diff is path/pointer is null we call the original parser and return whole xml as json.

M3 →

We need to parse a function to append a string to all the keys.

5 test cases null key, valid key, same key, reverse key, invalid/empty key.

So we used the same parser we implemented in M2 but we add prefix wherever we accumulate to main object.

How it is used :

```
Function<String, String> func = x -> new StringBuilder(x).reverse().toString();JSONObject jobj = XML.toJSONObject(new StringReader(xmlString),func);
```

Define a function first and then call the function it takes reader and function as parameters.

The `toJSONObject` method first checks whether the `add_prefix` function is null or an empty string, or if the result of applying the function(prefix) to two different string values is the same. If any of these conditions are true, the method returns `null`.

Else we create a new Json object and parse them to parser, parser takes function as prefix,

If a `ParseTerminationException` is thrown during parsing, the method creates a new `JSONObject` object with the exception message as the value, and returns it. If a `JSONException` is thrown, the method rethrows it as a new `JSONException` with the message "parse function error".

Finally, the method returns the populated `JSONObject` object. This to say we reached the target same as parser in M2.

M4 →

We have to create a stream function to handle the parsing of the xml to json object.

Our approach we wrote our own stream function.

Junits 3 tests to test stream over them, filter and mapping.

Method implemented :- first we parse the whole xml to json using regular parse.

now we use the constructed json to create a stream of objects.

`Stream<JSONObject> stream = jsonObject.toStream();` this is our method.

We have a nodeSpliterator class implementing Spliterator

so we have 2 tree one final tree called as root, another one is to hold the value to traverse.

we break once the current tree is same as the root that means we have created a tree with the each key val as it nodes.

As we are implementing Spliterator interface we need to implement certain non default methods.

So one of them is `public boolean tryAdvance(Consumer<? super JSONObject> action)`{}

here is where we handle main logic where we stream over the tree we got from the jsonObject.

So basic DFS principle go to the depth and and keep on adding it to stream.

we create a entry set of `Set<Entry<String, Object>> children = tree.entrySet();` so all the children is added now we read the value of entryset check if it is another jsonobject we make a recursive call before that we set current tree as the key val pair so that we keep track of nested cases.

if it is a regular key - value pair then create a new json object and add it to the stream.

once all the childrens are visited we exit the function.

M5 →

Using of asynchronous methods to make it concurrent.

We are implementing M3’s add prefix as our callable function.

We have 2 junits to check correct replacement and one where it throws exception.

```
Future<JSONObject> future = XML.toJSONObject(reader, func, consumer);
```

Our function takes in a reader, function key (the value to be replaced, consumer this is to display a exception). And returns a Future.

We are using `ExecutorService` interface here instead of a runnable just to maintain the memory and not to worry about the thread count.

So I have a class named Thread that implements a Callable interface we have a constructor to initialize these threads. And the main runnable function called call → this calls M3 code.

So in the toJsonObject we will submit the reader, function and consumer to executor service so this will call thread and we don’t need to wait for each thread to complete so we have Future so this future holds on to the values as a promise and finally we do a get call after all the iteration.