Project: XML Parser

Documentation

Made by Emmanouela Zapanti

Using IntelliJ

1

--------Information about the developer------

Name: Emmanouela Zapanti

Faculty Number: 23221030

Email: [emmaz102005@gmail.com](mailto:emmaz102005@gmail.com)

Institution: Technical University of Varna

2

--------The Project------

Contents

General Information about the project…………….……….5

Working with the command line……………………….…..6

Chapter 1. The lead

* 1. Description of the idea of the project…………..……..7
  2. Purpose and objectives of the development……….….7
  3. Structure of the documentation ………………………7

Chapter 2. Overview of the subject area

2.1 Basic definitions, concepts and algorithms to be used......8

2.2 Defining problems and complexity of the task…………..8

2.3 Approaches, methods for solving the problems posed…...9

Chapter 3. Design

3.1 General structure of the project…………………………..11

3.2 Diagrams and Flowcharts………………………………...11

3.3 Description of the classes………………………………...13

3

Chapter 4. Implementation, testing

4.1 Class implementation……………………………………17

Chapter 5. Conclusion

5.1 Summary of the implementation of the initial objectives..22

5.2 Directions for future development and improvement……22

Bibliography and References…………………….…………..24

4

Introduction

General Information about the project

The purpose of XML Parser is to operate with structured text files, efficiently execute various commands on them and implement the command line arguments.

(The task description is included in the .zip folder.)

Analysis of the task and approach to solving

The nature of the task is simple enough and can be divided into three parts:

* One class for the Record. The file will consist of a list of many Records.

Their structure is <person id=’1’>

<name> James </name>

<address> Chicago </address>

</person>

The information is for demonstration.

* Every functionality has its own class and is executable. It extends the XMLParserCLI class and inherits the attributes of

protected static List<Record> *records* = new ArrayList<>();  
protected static String *currentFile* = null;

5

* The XMLParserCLI class puts everything together. Selects the file to be opened and if it doesn’t exist, creates one. It consists of a switch clause and calls every chosen functionality.

Working with the command line

Here are simple instructions on how to operate the application using the command line.

After opening the Command Line Prompt, navigate to the application folder:

->cd XML\_Parser

->cd src

Execute the command “java XMLParserCLI.java”. It should start running.

“>help” lists the available commands. It is important that before any other functions such as close, write or read, some file must be opened first.

6

Chapter 1: The Lead

* 1. Description of the idea

The main idea of the XML Parser is to create a program that handles structured text files, using various functions such as open, close, save etc.…The most intriguing functionality is the implementation of the command-line arguments which enable the commands to be run through runtime and outside of the editor’s environment.

* 1. Purpose and objectives of the development

During the development of the project the main goal was to keep things as simple as possible, always having the SOLID principles in mind. The main objective was a simple, efficient program that is easy to use, also:

* Implements command line arguments
* Handles structured text files (resembling XML)
  1. Structure of the documentation

It consists of the following:

* Introduction(General Information and Command Line instructions)
* Main Body (About the program, its development, its functions and capabilities. It contains a description of different aspects of the development process, objectives and goals.
* Conclusion (Sum up of purpose, objectives and future opportunities for improvement) 7

Chapter 2. Overview of the subject area

2.1 Basic definitions, concepts and algorithms to be used

Logic of the program:

We treat “<person id=’ ’> … </person>” as an object that has attributes. The class’s name is Record. The file we are working on consists of many Records, that makes up a list. Then we have separate classes for the various functions like save , saveas, read, write, etc.… that perform changes and parse the Records and the current file we are working on. All of this comes together in the XMLParserCLI class. It consists of a big switch that handles all the options a user can make.

2.2 Defining problems and complexity of the task

Problems encountered:

* How to implement command line arguments and make the program run outside of the editor’s environment.
* How to organize the classes and the hierarchy of them. Which classes should be instantiated as objects. How to work with the file and perform many operations on the same one.
* How to read the structured text file and extract the desired information.
* How to alter the structure of the file. Its nature is strict consisting of person with id, name and address. The idea is to make the program more free and flexible with the ability to add more elements.
* How to implement Xpath. 8
* How to perform operations with id since it is a String but should be treated as an integer.
* Regarding the xpath command, how to split the command into parts and perform operation based on their value. For example xpath person/address=USA. We need “address”, “=”, “USA”.

2.3 Approaches, methods for solving the problems posed

* To solve the most important issue of making the program run on the command line we implement the Command Line Arguments. Thorough research was made and included in the bibliography.

In the XMLParserCLI class we use a while loop to keep the program running unless ‘exit’is chosen. In the functionalities classes we make them executable with ‘public static void writeRecord (String id ,String name, String address)’ for example.

* The organization of the classes and the hierarchy was simple and consists of the main XMLParserCLI and the other functionalities that extend the main. Only the Record class works as an actual object and is instantiated. To perform many operation on the same file we save it each time with the ‘currentFile’ attribute. This is cleared when we close the file and stop working with it.
* To read the structured text file we and extract the important information we use “line = line.trim()”. Then we save id for example like this:

9

if(line.startsWith("<person id='")&&line.endsWith("'>")){  
 id = line.substring(12,line.length() - 2);

12 is the index after which we want to read the characters, and line.length()-2 the index we want to stop reading. Now we have successfully removed the xml tags and kept the value inside.

* Another challenging issue was about the implementation of Xpath. The xpath class consists of an ‘if’ clause that calls a different function depending on the symbol included in the query command. Especially for the indexed query we use regex not a symbol.
* The last problem concerns the behavior of id. It is declared as a String in order to be passed as an input[] parameter but should be treated as an integer. For example if the id of a file is the same with an already saved one we append an extra counter. We solved this using:
* if(id.equals(r.id)){  
    
   String strCounter = Integer.*toString*(counter);  
   id = id.concat("\_"+strCounter);  
   counter++;  
  }

2.3.1 Code from the ‘write’ class.

10

Chapter 3. Design

3.1 General structure of the project, packages to be implemented

XML Parser doesn’t follow any specific structure. All the classes are free and together in the “src” directory. This serves the purpose of simplicity, since there are not many different classes. This also solves the problem of visibility and compiling the program in order to work in the command line. We have XMLParserCLI as the main class, save, read, write etc.…as the functions and test.txt as the file containing XML. A new one could be created anytime and will be situated in the “src” directory as well.

3.2 Diagrams and Flowcharts

For the Record class.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, σχεδίαση

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

3.2.1 Generated UML diagram for the Record class

11

For the XMLParserClI class.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

3.2.2 Generated UML diagram for the XMLParserCLI class

12

3.3 Description of the classes

-XMLParserCLI

This class is the main. Contains all the functionalities together and calls the main loop for running in the command line.

-open

The usage of the ‘open’ class is to open a file, load its content into memory. If it doesn’t exist, it creates a new file.

-close

Closes the file.Sets the currentFile to null and clears the loaded memory so that we avoid mistakes and overwriting when we open another one.

-delete

Parses the Records and when we have a matching id, it deleted the corresponding Record from the list.

-help

Prints available commands in the console.

-children

Parses the Records and when we have a matching id, we print the children this id has, like the name and the address.

13

-child

We select an id and a child specific. The function parses the Records and when we have a matching id and value (‘1’ for name and ‘2’ for address) we print the corresponding value.

-Newchild

We add a new Record with only id, the rest of the attributes are automatically set to null. If the id matched with anything else we append a counter to it, or if it is null we automatically add id.

-print

This function prints the structured file exactly as it is.

-read

We parse the Records and print them one by one, according to the toString() method of the Record class.

-Record

Has attributes about id, name and address. Constructor, getters, setters and toString() method.

14

-save

We parse the Record list and write the information passed as parameters in a specific way. So that we keep the file structure.

<person id=’ ’>

<name></name>

<address></address>

</person>

-select

We parse the Record list and print the output depending on the given id and value (name or address).

-set

We parse the Record list and if we match the given id, we can change either the name or the address with a new value.

-text

We parse the Record list and depending on the id we print one specific Record in the format of its toString() method.

-write

This function adds a Record in the list. If the id is matching, we append a counter and if it is null we put a counter as its id.

15

-xpath

This function is responsible for executing queries. It supports the operators ‘@’, ‘/’, ‘’[index], ‘=’. It consists of an ‘if’ clause and depending on the operator included in the query calls the corresponding function.

‘alphaOperator’ prints the specified value of all the Records, ex. person@id.

‘indexedOperator’ prints the value of a specified Record, ex. person/address[2].

‘equalOperator’ prints the Record where some value must be matching, either the name or the address or the id with some given key. Ex. person/name=Sandra.

‘slashOperator’ prints a list of the selected value that there is in the file. Ex. person/name – this gives a list of all the names.

-saveas

Here we save all the changes in a different destination. If the specified file does not exist, we create a new one. Again we parse the Record list and write the information in a structured order.

16

Chapter 4. Implementation, testing

4.1 Class implementation and test scenarios

Run the file XMLParserCLI.java with

“ >java XMLParserCLI.java” in the cmd.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.1 class implementation. Start of the application.

Open a file, already existing “test.txt”.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.2 class implementation of ‘open’. Snippet of code from cmd.

Open a file that doesn’t exit. We close the first, then open the second.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.3 class implementation of ‘close’. Snippet of code from cmd.

17

If we read that file, the result is empty content. Then we write a record missing id, we get an error. Then re-open the file, write and read.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.4 class implementation of ‘read’ and ‘write’. Snippet of code from cmd.

Then we close this one. Open “test.txt”. Print its content.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.5 class implementation of ‘print’. Snippet of code from cmd.

Change some name with the “set” function and save to “test2.txt”.



4.1.6 class implementation of ‘set’. Snippet of code from cmd.

18

We read the file. The value ‘name’ with id 8 is changed.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.7 ‘read’ the changed file, ‘saveas’ in another file. Snippet of code from cmd.

We perform the ‘child’, ‘children’ and ‘xpath’ commands.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.8 class implementation of ‘child’. Code form cmd.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.9 class implementation of ‘children’. Code from cmd.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, μαύρο, γραμματοσειρά

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.10 class implementation of ‘xpath’. Code from cmd.

19

Then ‘delete’,’ help’ and ‘Newchild’.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.11 class implementation of ‘delete’. Code from cmd.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.12 class implementation of ‘help’. Code from cmd.

In the newchild command, we intentionally set the id =2. There is already a record with the same id, and as we will see the new record obtains the id of 2\_1.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.13 class implementation of ‘newchild’. Code from cmd.

20

Finally ‘select’ and ‘text’.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.14 class implementation of ‘select’. Code from cmd.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης, μαύρο

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.15 class implementation of ‘text’. Code from cmd.

We save and exit the program. Without having closed the file, we get a warning. After the file is closed, we freely exit the app.

Εικόνα που περιέχει κείμενο, γραμματοσειρά, στιγμιότυπο οθόνης

Το περιεχόμενο που δημιουργείται από AI ενδέχεται να είναι εσφαλμένο.

4.1.16 class implementation of ‘save’ and ‘exit’. Code from cmd.

21

Chapter 5. Conclusion

5.1 Summary of the implementation of the initial objectives

The result of the XML Parser is a stable program, functional and efficient. Simple in its nature and the way it works. We can easily handle files resembling XML, add and remove elements, read and write on the file, perform queries on the data. Though this is a very simple application various changes could be made to improve and elevate the logic and functionality of the program.

This is described in the next sub-chapter.

5.2 Directions for the future development and improvement

The application as a whole is satisfactory and efficient. Changes and improvements are possible and would make the program even better.

* Regarding its file handling structure, the idea is that the program should not only handle the format of id, name, address but more free and flexible XML. We could also add the tags of <people> … </people> at the start and end of the file. This was not possible and posed problems with reading.
* Adding elements and children for example for:

person id=1 add child nickname=SuperDeveloper

or other attributes.

* With adding another layer of abstraction we could have the chance to make the file from scratch. It should not necessary be about people, but anything. The main opening and closing tags, also the Records. 22
* Another opportunity for development regards the xpath commands, we could add a great variety of commands and supported operators, to bring the program closer to real xpath and xml behavior.

23

Link for GitHub repository

<https://github.com/LilGhost44/XML-Parser>

Bibliography and References

For xpath:

<https://www.w3schools.com/xml/xpath_examples.asp>

<https://www.lambdatest.com/blog/complete-guide-for-using-xpath-in-selenium-with-examples/>

For java command line arguments:

<https://www.geeksforgeeks.org/command-line-arguments-in-java/>

<https://www.baeldung.com/java-command-line-arguments>

<https://docs.oracle.com/javase/tutorial/essential/environment/cmdLineArgs.html>

<https://www.programiz.com/java-programming/command-line-arguments>

<https://www.digitalocean.com/community/tutorials/command-line-arguments-in-java>

Strictly for debugging and diagram generation:

<https://chatgpt.com/>

24