TECHNICAL UNIVERSITY OF VARNA

Faculty Communication and Computer Engineering

SIT

OBJECTIVE-ORIENTED PROGRAMMING PROJECT

Variant 3

Author: Oleksandr Porokhnya

Speciality: SIT

Group: SIT English

Faculty Number: 22221019

2024

Contents:

CHAPTER 1. INTRODUCTION

1.1 Description and idea of project

This project is – XML Parser. It is primitive realization of such libraries like DOM or SAX, it's helps work with familiar to XML syntaxis text documents. All information is stored in .txt document and helps to store, navigate and execute all data inside.

1.2 Development objectives

1. Create controller which will save data and receive data from file
2. Create object which will store all data from file
3. Find way to output data on screen
4. Create controller for manage data from file
5. Implement CLI
6. Test and debugging

CHAPTER 2. REVIEW OF THE SUBJECT AREA

2.1 Basic Definitions, Concepts and Algorithms to be used

- **Console application**. Using console user can manipulate with data in file. Writing a specific commands user can open, save, edit, manage, retrieve and finish work with files and application.

- **Core object**. Class in which program store all data from file such as attributes of tags, and then manipulate, save and change them.

- **Array of objects**. I use List object which contains all objects, and then program can use this List for manipulating with all of them in it.

- **Command Line Interface (CLI)**. Application uses CLI to interact with users. User input specific command and receive response from application in terminal or console.

- **Regular Expressions (Regexes)**. Regexes it the easier way to parse data from file or console/terminal, it is looking for matches in request and when find it execute code. Patterns help to avoid creating millions of cases and make code clearer.

- **Opening a file**. Using scanner and patterns application read file and save data from it into array of objects for further usage.

- **Saving a file.** When the user wants to save data, instead of a finding change, the application just rewrites the whole file with current data from object and save in new or same file.

- **Editing a file.** When the user wants to change something in a file, the application rewrites data in the specified object and then when the file is saved, these changes will apply in the file.

- **Printing.** Printing just reads all data from array of objects and using toString function output information on screen.

2.2 Defining problems of given task

- **Parsing non-standard XML data**. For this task we can`t use DOM or SAX or another already complete parsers

- **Command Line Interface (CLI)**. Creating of intuitive and understandable for user interface

- **Efficient Data storage and Retrieval**. Creating of a comfortable structure for storage and receiving data

- **Algorithm Design.** Developing algorithms for operations like opening, editing, saving, printing data and others.

- **Identifier problem**. Identifiers of all objects must be unique or if objects don’t have an id we must create it.

- **XPath operations and Axes**. The need to implement various XPath operations and axes without using specialized libraries.

- **Error Handling and validation**. Error handling and data validation to ensure reliable parser operation

2.3 Approaches, Methods for solving problem

- **Parsing non-standard XML data**
  Parsing data has become the first problem, how to read data correctly and without different troubles. First of all, I want to create parser which will look for not only tags, but I also want to make sure that open and close tag are same. Let's we'll see my file structure:

  ```
  <root xmlns:h="http://www.w3.org/TR/html4/">
  <h:people>
  <h:person id="1">
  <h:name> Ivan Ivanov </h:name>
  <h:address> USA </h:address>
  </h:person>
  <h:person id="2">
  <h:name> Ivan Petrov </h:name>
  <h:address> Bulgaria </h:address>
  </h:person>
  <h:person id="1_2">
  <h:name> Fridrich Petrov </h:name>
  <h:address> France </h:address>
  </h:person>
  </h:people>
  </h:root>
  ```

  As we can see some tags have attributes, some tags have a lot of children inside and we need to explain to computer where and how we want to read. So, I decided to use patterns for parsing data, because it is the easiest way to read all documents and split it by logical units correctly.

  Patterns help check is both tags are same, also we can split our patterns into a groups and load or use only necessary parts of data, also it's help read our lines without namespaces, what helps check only type of tags, and also check only namespaces without tags type.

  Also, it helps me check the type of tag and if we need to look for something it's much easier to make using patterns, because we don't worry about incorrect writing of come tag like when we use for example equals.

  Then we store all this data into some object and this object put into some List for further usage.

- **Command Line Interface.** CLI is also done using Patterns, the main reason to use exactly them - splitting. For example, we have our open command:

open <file path>

Usnig patters we can easily split out line on command (open) and file path using just *matcher.group()* and check which command we want to use and what this command consists of. And such for all another commands, and now we just need to write correct pattern and our Pattern split it for so many groups as we want.

- **Efficient data storage and Retrieval**. For storage data I choose class, when user write open command, program read all data from file and save information into an object and then this object will be stored in List.

- **Algorithm design**.
  - **Opening a file.** When user write command for reading some file, program start function which open file and read, using scanner and patterns it read file and save it into objects, and output message that file opening was successful or not.

  - **Saving a file**. When a program saves data, it reads information from list of objects and saves new data, it rewritten all file using XML syntaxis and push new data into a file.

  - **Editing a file.** Using CLI commands like delete or set, user must enter id and element which he wants to change, then program look for this id and change values in List and then when user want to save file change them also in file.

  - **Printing**. Calling the command print in CLI, the program iterates through the List of objects and outputs all information from objects.

  - **Selecting an element**. Calling CLI command select, program read file and look for selected id, and then when find id start to look for selected attribute and when find all output result on screen.

  - **Children element**. Calling CLI command children, program start to look for all children of element with selected id, if Id was found, program output all information from list of objects about this element

- **Child element**. Calling CLI command child, the program reads file and looks for selected id and when find it, start counter for find attribute with selected index.

- **Text**. Calling CLI command text, the program reads file and looks for selecting Id, and when find it just output all text of this element in XML form till the close parent tag.

- **New Child**. Calling CLI command new child, the program adds select id into List of objects without any attributes just id and then when user want to save data save into file

- **Identifier problem.** Id mist be unique and for this we must create new or modify already existing Id. For this I check all elements in List and using Map checking if it already exists or not and modify, or if Id equals null create new identifier using size of array and then check existing of its ones more.

- **XPath operations and axes**.

  - **"/" operator**. Calling the operator "/" with a specific key, the program reads the file. If the key matches, the corresponding value is extracted and added to a list. Finally, the function returns this list of all values associated with the specified key.

  - **"[]" operator**. Calling the operator "[]" with a specific key and index, reads the file. If the key matches and the current match count equals the specified index, the corresponding value is returned. If the key matches but the count does not equal the index, the counter increments.

  - **"@" operator**. Calling the operator "@" with a specific key, reads the file. If the matcher finds a match and the line contains the key as an attribute, the corresponding attribute value is added to a list. The function returns this list of attribute values

  - **"=" operator**. Calling the operator "=" with a specific key, value, and outputKey, the program reads the file. When it encounters a line with an id attribute, it starts a nested scan to check for lines that match the outputKey. If it finds such a line, it saves its value. Then, if it encounters a line matching the provided key and value, it adds the previously saved value to the list of results. The function returns a list of matched values based on the specified criteria.

- **Ancestor axis**. The ancestor axis, the program reads the file. If a line does not match any XML tag pattern, it checks if the line contains the provided key . If the line contains the key but is not an end tag, it returns the accumulated keys. Otherwise, it adds the line to the list of keys. After scanning all lines, it returns the list of keys.

- **Child axis**. The child axis, the program reads the file. Then, it checks if the line contains the provided key. If so, it proceeds to the next line to extract the child element. Once the child element is identified, it continues scanning the document for lines containing the child element until it encounters the end of the parent element or the end of the document. During this process, it accumulates the lines representing the child elements and adds them to the list of keys. After scanning all lines, it returns the list of keys.

- **Parent axis**. The parent axis, the program reads the file. If a it checks if the line contains the specified child tag, the function returns the current tag, which represents the parent tag. If a line matches the child tag directly, the function immediately returns the current tag, indicating the parent tag for the specified child tag.

- **Descendent axis**. The descendent axis, the program reads the file. Upon finding a line containing the parent tag, it enters a nested loop. Within this loop, it reads each line and checks if it contains the parent tag. If such a line is found, it appends the line to a string builder object. The function continues scanning until it encounters a line containing the parent tag again, indicating the end of descendants for that parent. Finally, it returns a single string.

- **Error Handling.** To handle errors, try-catch constructions and exceptions were used.

2.4 User and Quality requirements

- **User requirements**
  - **Functional requirements**. Users expect to see basic functions like opening, saving, editing, XPath functions and add new elements
  - **Interface requirements**. Users want to use simple interface for easy navigation and access to commands
  - **Performance requirements.** Program must have good optimization for comfortable using
- **Quality requirements**
  - Reliability. Program must be sustainable to errors and failures, also it must save files without losses or errors.

- **Performance.** Program must read and process data fast and effectively, even with large amount of data. Search operations must process fast with large amount of data too.
- **Usability**. Interface must be simple and understandable; documentation must be in open source for understanding of functionality of program.
- **Maintainability**. Code must be good optimized and easily modified.

CHAPTER 3. DESIGN

3.1 Main Packages

Com.company - main package with main function and console handler respectively.

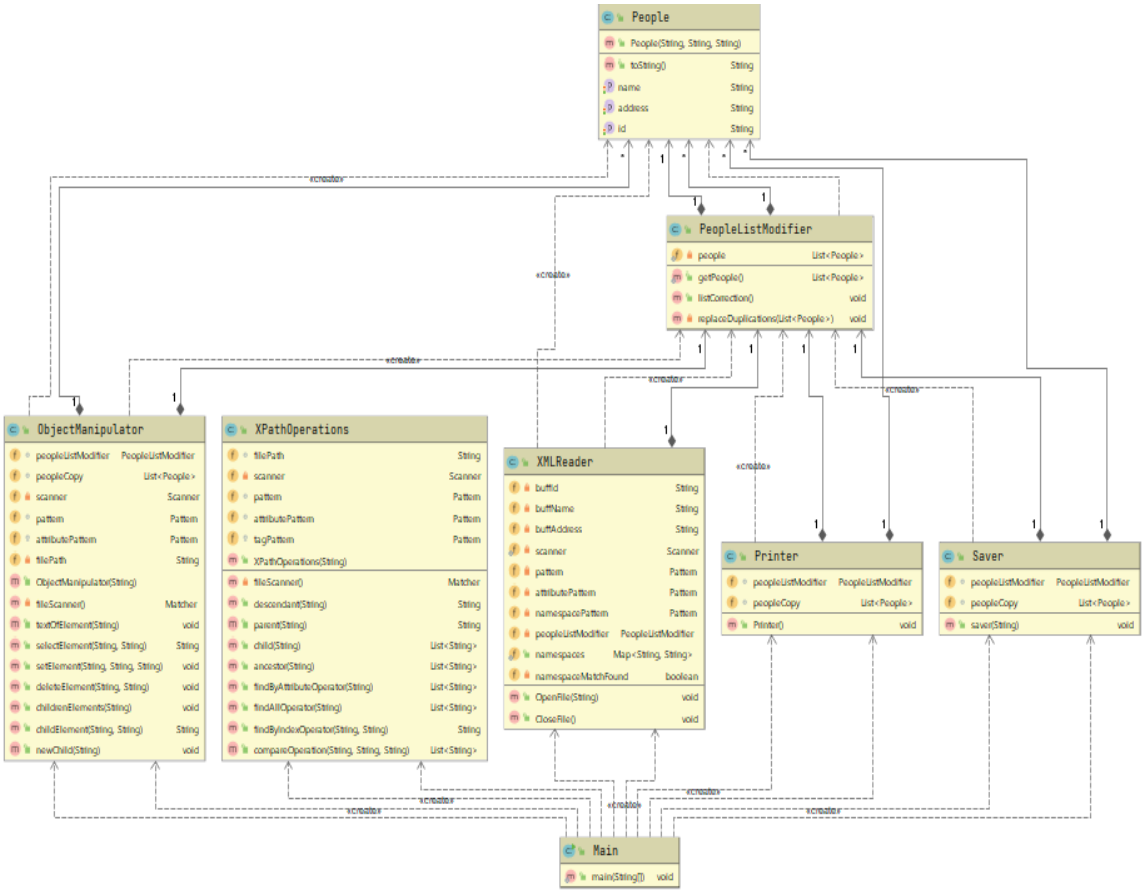
Manipulator – package where store all classes for manipulate data in file

Modifier – package where store classes for editing id and print information

Opener – package where store classes for open and using data

Saves – package where store classes for save data

3.2 Diagram



Chapter 4: Implementation, testing

4.1 Com.main package classes

- Main

In this class main function and console handler respectively

Initialization:
In main method I have some main field like:

```
boolean isFileOpen = false; //check is file open
String filePath = null; //file path
```

Also have patterns for all commands in my program, for example this:

```
Pattern xPathComparePattern = Pattern.compile("([p][e][r][s][o][n])\\s?\\(([a-zA-Z]+)\\s?\\=\\s?\\\"([a-zA-Z0-9\\s?]+)\\\"\\)\\s?\\/([a-zA-Z]+)");
```

User interaction:
Message in console for user, console reads command and look for suitable pattern

```
System.out.println("Enter your next command(you can write help" +
        " to see list of all commands)");
while (true) {
    Scanner consoleScanner = new Scanner(System.in);
    String inputData = consoleScanner.nextLine();
```

Loop:
I use while(true) loop for my program and while users want to use it, it will be work

```
while (true) {
\\Code
}
```

If statements:
I use if statements for searching the right command for example if user write:

open <path>

Function will check if it suitable with pattern:

```
Pattern openPattern  = Pattern.compile("([o][p][e][n])\\s?([a-zA-Z0-9:/\\\\\\\\\. ]+)");
```

And start to execute if statment:

```
Matcher openMatcher = openPattern.matcher(inputData);
if (openMatcher.find()) {
```

```
    try {
        if (!isFileOpen) {
            XMLReader opener = new XMLReader();
            opener.OpenFile(openMatcher.group(2));
            filePath = openMatcher.group(2);
            isFileOpen = true;
            System.out.println("Successfully opened file");
        }
    } catch (Exception e) {
        System.out.println("Cannot open this file or incorrect file name!");
    }
}
```

4.2 Opener package classes

- XMLReader

  Initialize:

```
private String buffId = null; //buffer for id
private String buffName = null; //buffer for name
private String buffAddress = null; //buffer for address



private static Scanner scanner;
private Pattern pattern  =
Pattern.compile("<(\\w+):(\\w+)>([^<]*)</\\1:\\2>");//pattern for last childs
private Pattern attributePattern = Pattern.compile("[<]([a-zA-Z]+):([a-zA-
Z]+)\\s+([a-zA-Z]+)\\s*=\\s*\"([0-9_]+)\"[>]");//pattern for attribute like id
private Pattern namespacePattern =
Pattern.compile("xmlns:(\\w+)=\"([^\"]+)\"");//namespace pattern

private PeopleListModifier peopleListModifier = new
PeopleListModifier();//list of peoples

public static Map<String, String> namespaces = new HashMap<>();//hash
map for namespaces for storing one namespace one time
private boolean namespaceMatchFound;
```

  Functions:

  Open file function open and save all from file and send message in case something went wrong.

  Use buffer for save data from current person and when complete read person store it into the array.

```
if (attributeMatchFound && attributeMatcher.group(3).equals("id")) {
   if(attributeMatcher.group(4).trim() == "") {
      buffId = null;
```

```
            }
            buffId = attributeMatcher.group(4).trim();
        }
        if (matchFound && matcher.group(2).equals("name")) {
            buffName = matcher.group(3).trim();
        }
        if (matchFound && matcher.group(2).equals("address")) {
            buffAddress = matcher.group(3).trim();
        }
        if(text.equals("</h:person>")){
            peopleListModifier.getPeople().add(new People(buffId, buffName,
        buffAddress));
            buffId = null;
            buffName= null;
            buffAddress = null;
        }
```

Error handler:

If during work with scanner an error will occur, try catch will catch error and output message.

```
try {
//Code
}
catch (IOException e){
    System.out.println("Open file exception! Cannot open or read file");
}
```

Close a file. Function which close file using Scanner.close().

- People

Initializing:

Creating necessary fields for store data for our persons and constructor for this

```
private String id;
        private String name;
        private String address;

        public People(String id, String name, String address) {
                this.id = id;
                this.name = name;
                this.address = address;
        }
```

Also use getters and setters cause our fields are private and toString function for output information about people on screen.

4.3 Manipulator

- Object manipulator

Initializing:

In constructor create scanner with file path which transmitted as a parameter

```
PeopleListModifier peopleListModifier = new PeopleListModifier();
List<People> peopleCopy = peopleListModifier.getPeople();

private Scanner scanner;
Pattern pattern  = Pattern.compile("<(\\w+):(\\w+)>([^<]*)</\\1:\\2>");
Pattern attributePattern = Pattern.compile("[<]([a-zA-Z]+):([a-zA-Z]+)\\s+([a-zA-Z]+)\\s*=\\s*\"([0-9_]+)\"[>]");
private String filePath = null;

public ObjectManipulator(String filePath){
   this.filePath = filePath;
   try {
      scanner = new Scanner(new File(filePath));
   }catch(IOException e){
      e.printStackTrace();
   }
}
```

Error handler:

Use try catch to catch some IOExeptions.

Functions:

FileScanner:

Return Matcher if find some matches from patterns:

```
String text = scanner.nextLine();
Matcher matcher = pattern.matcher(text);
Matcher attributeMatcher = attributePattern.matcher(text);
boolean attributeMatchFound = attributeMatcher.find();
boolean matchFound = matcher.find();
if (attributeMatchFound)
   return attributeMatcher;
if (matchFound)
   return matcher;
```

TextOfElement:

Takes id as a parameter and looks it for in file, when find it output all information inside this id

```
if(matcher.groupCount() > 2 && matcher.group(3).equals("id") &&
matcher.group(4).equals(id)){//only tags with attribute have more than 2 groups so I check is it an attribute and if it is so continue
   do{
      text = scanner.nextLine().trim();
```

```
        System.out.println(text);
    }while (!text.equals("</h:person>"));
    break;
}
```

SelectElement:
Takes id and key as a parameter and looks them for in file, when find output data in key of selecting by id element

```
if(matcher.groupCount() > 2 && matcher.group(3).equals("id") &&
matcher.group(4).equals(id)) {
    while (true) {
        if (matcher.group(2).equals(key)) {
            return matcher.group(3);
        } else {
            matcher = fileScanner();
        }
    }
}
```

SetElement:
Takes id, key and value as a parameter and look for id in List of objects, when find, change value of key using new value and rewrite it in object

```
for(int i = 0; i < peopleCopy.size(); i++){
    if(peopleCopy.get(i).getId().equals(id)){
        if(key.equals("name")){
            peopleCopy.get(i).setName(value);
            System.out.println("Name successfully changed");
        }else if(key.equals("address")) {
            peopleCopy.get(i).setAddress(value);
            System.out.println("Address successfully changed");
        }
    }
}
```

Delete element:
Takes id and key as a parameter and look for id in List of objects, when find, change value of key on null and rewrite it in object

```
for(int i = 0; i < peopleCopy.size(); i++){
    if(peopleCopy.get(i).getId().equals(id)) {
        if (key.equals("name")) {
            peopleCopy.get(i).setName(null);
            System.out.println("Name successfully deleted");
        } else if (key.equals("address")) {
            peopleCopy.get(i).setAddress(null);
```

```
            System.out.println("Address successfully deleted");
        }
    }
}
```

ChildElement:
Takes id and n as a parameter and look for them in file, when find, start to count in the loop n element and then output it on the console

```
if(childMather.groupCount() > 2 &&
childMather.group(3).equals("id") &&
childMather.group(4).equals(id)){
    for(int i = 0; i < Integer.parseInt(n);i++){
        childMather = fileScanner();
    }
    return childMather.group(3);
}
```

ChildrenElement
Takes id as a parameter and look for them List of objects, when find, output all information about this object on the console

```
for(int i = 0; i < peopleCopy.size(); i++){
    if(peopleCopy.get(i).getId().equals(id)) {
        System.out.println("Id: " + peopleCopy.get(i).getId() +"\n" +
                "Name: " + peopleCopy.get(i).getName() + "\n" +
                "Address: " + peopleCopy.get(i).getAddress()+"\n");
    }
}
```

NewChild
Takes id as a parameter and add new object into the List of objects without name and address

```
peopleCopy.add(new People(id, null,null));
```

- XPath Operations

Initializing:
In constructor create scanner with file path which transmitted as a parameter

```
String filePath = null;
private Scanner scanner;
Pattern pattern  = Pattern.compile("<(\\w+):(\\w+)>([^<]*)</\\1:\\2>");
Pattern attributePattern = Pattern.compile("[<]([a-zA-Z]+):([a-zA-Z]+)\\s+([a-zA-Z]+)\\s*=\\s*\"([0-9_]+)\"[>]");
Pattern tagPattern = Pattern.compile("[<]([a-zA-Z]+):([a-zA-Z\\/]+)[>]");
```

```
public XPathOperations(String filePath) {
   this.filePath = filePath;
   try {
      scanner = new Scanner(new File(filePath));
   }catch(IOException e){
      e.printStackTrace();
   }
}
```

Functions:

FileScanner

Return Matcher if find some matches from patterns:

```
private Matcher fileScanner(){
   while (scanner.hasNext()) {
      String text = scanner.nextLine();
      Matcher matcher = pattern.matcher(text);
      Matcher tagMatcher = tagPattern.matcher(text);
      Matcher attributeMatcher = attributePattern.matcher(text);
      boolean attributeMatchFound = attributeMatcher.find();
      boolean tagMatchFound = tagMatcher.find();
      boolean matchFound = matcher.find();
      if(attributeMatchFound)
         return attributeMatcher;
      if (matchFound)
         return matcher;
      if(tagMatchFound)
         return tagMatcher;
   }
   return null;
}
```

Descendant:

Takes parent as a parameter and output all information from open tag to the close tag

```
if (matcher.group(2).equals(parent)) {
   while (scanner.hasNext()) {
      line = scanner.nextLine().trim();
      if (line.contains(parent) && !line.startsWith("</")) {
         break;
      }
      stringBuilder.append(line).append("\n");
   }
}
```

Parent
Take child as a parameter and if not find it save previous tag as a parent, if we find child element and it is not name or address, we return parent tag if it is, we try find child element and return parent tag

```
if(!matcher.find()){
    if(line.contains(child) && !line.startsWith("</")){
        return currentTag;
    }
    currentTag = line;
}
if(line.contains(child) && !line.startsWith("</")){
    return currentTag;
}
```

Child
Take key as a parameter and try to find all children of key tag, we find tag with attributes or tag like name or address we continue work with them and output all lines where we meet out child element, and output all children of tag or if our parent has only one child output it

```
if (line.contains(key) && !line.startsWith("</")) {
    line = scanner.nextLine().trim();
    Matcher matcher = attributePattern.matcher(line);
    if (matcher.find())
        childLine = matcher.group(2);
    else {
        matcher = tagPattern.matcher(line);
        if (matcher.find())
            childLine = matcher.group(2);
    }
    while (scanner.hasNext()) {
        if (line.contains(childLine) && !line.startsWith("</")) {
            keys.add(line.trim());
            while (scanner.hasNext()) {
                matcher = pattern.matcher(line);
                if (matcher.find()) {
                    line = scanner.nextLine().trim();
                    keys.add(line.trim());
                    line = scanner.nextLine().trim();
                } else {
                    line = scanner.nextLine().trim();
                    break;
                }
```

```
            }

        } else
            line = scanner.nextLine().trim();
    }
```

Ancestor
Takes key as a parameter and output all information till parent tag

```
while (scanner.hasNext()){
    line = scanner.nextLine().trim();
    Matcher matcher = pattern.matcher(line);
    if(!matcher.find()){
        if(line.contains(key) && !line.startsWith("</")){
            return keys;
        }
        keys.add(line);
    }
    if(line.contains(key) && !line.startsWith("</")){
        return keys;
    }
}
```

FindByAttributeOperator
Takes key as a parameter and find all tags with same key and which contains attributes and output on the screen

```
while (scanner.hasNext()){
    Matcher matcher = fileScanner();
    if(matcher!=null && matcher.groupCount() > 2 &&
matcher.group(3).equals(key)){
        keyArray.add(matcher.group(4));
    }
}
```

FindAllOperator
Takes key as a parameter and find all tags with same keys and output all of them to the console

```
while (scanner.hasNext()){
    Matcher matcher = fileScanner();
    if(matcher != null && matcher.group(2).equals(key)){
        keyArray.add(matcher.group(matcher.groupCount()));
    }
}
```

FindByIndexOperator

Takes key and n as a parameter and find all tags with same key and find n element of them

```
while (scanner.hasNext()){
   Matcher matcher = fileScanner();
   if(matcher!=null && matcher.group(2).equals(key)){
      if (counter == Integer.parseInt(n))
         return matcher.group(matcher.groupCount());
      else
         counter++;
   }
}
```

CompareOperation

Take key, value and outputKey as a parameter and try to find element with same key and value and when find output to the console outputKey

```
while (scanner.hasNext()){
   Matcher matcher = fileScanner();
   String saveKey = null;
   if(matcher != null && matcher.groupCount() > 2 &&
matcher.group(3).equals("id")){
      while (scanner.hasNext()){
         if(matcher.group(2).equals(outputKey)){
            saveKey = matcher.group(3);
         }
         if(matcher.group(2).equals(key) &&
matcher.group(3).trim().equals(value)){
            if(saveKey!=null) {
               keyArray.add(saveKey);
               break;
            }
            matcher = fileScanner();
            if(matcher.group(2).equals(outputKey)){
               keyArray.add(matcher.group(3));
               break;
            }
         }else{
            matcher = fileScanner();
         }
      }
   }
}
```

4.4 Modifier package

- PeopleListModifier class

Initialize:

```
private static List<People> people = new ArrayList<People>();

public static List<People> getPeople() {
   return people;
}
```

Functions

ListCorrenctions

Check our List and if find object without Id create id giving size of array as Id and then call function for replace duplications in List

```
for(int i = 0; i < people.size(); i++) {
   if (people.get(i).getId() == null) {
      people.get(i).setId(String.valueOf(people.size()).trim());
   }
}
replaceDuplications(people);
```

ReplaceDuplications

Create HashMap for checking duplication and add "_" if find already existing Id, and put it into our Map

```
for(int i = 0; i < list.size(); i++){
   String element = list.get(i).getId().trim();
   if(map.containsKey(element)){
      int frequency = map.get(element);
      frequency++;
      String temp = element + "_" + frequency;
      map.put(temp.trim(), list.size());
      list.get(i).setId(temp.trim());
   }
```

- Printer Class

Initializing:

```
PeopleListModifier peopleListModifier = new PeopleListModifier();
List<People> peopleCopy = peopleListModifier.getPeople();
```

Functions:

Printer

Output all information on the screen using toString function

```
peopleListModifier.listCorrection();
for(int i = 0; i < peopleListModifier.getPeople().size(); i++){
```

```
            System.out.println(peopleListModifier.getPeople().get(i).toString());
        }
```

4.5 Saver

Saver class

Initialize:

```
PeopleListModifier peopleListModifier = new PeopleListModifier();
List<People> peopleCopy = peopleListModifier.getPeople();
```

Functions

Saver

Make simulation that we save our file just make syntaxis of XML file

```
File file = new File(path);
FileWriter fileWriter = new FileWriter(file);
fileWriter.write("<root xmlns:h=\"http://www.w3.org/TR/html4/\">\n");
fileWriter.write("\t<h:people>\n");
for(int i = 0; i < peopleCopy.size();i++){
   fileWriter.write("\t\t<h:person id=\""+peopleCopy.get(i).getId()+"\">\n");
   if(peopleCopy.get(i).getName() != null) {
      fileWriter.write("\t\t\t<h:name> " + peopleCopy.get(i).getName() + "
</h:name>\n");
   }
   if(peopleCopy.get(i).getAddress() != null) {
      fileWriter.write("\t\t\t<h:address> " + peopleCopy.get(i).getAddress() + "
</h:address>\n");
   }
   fileWriter.write("\t\t</h:person>\n");
}
fileWriter.write("\t</h:people>\n");
fileWriter.write("</h:root>\n");
fileWriter.close();
```

Error hadler:

Use try catch instruction to catch IOExeption while open file

```
catch (IOException e){
   System.out.println("Cannon create/modify file");
}
```

4.2 Planning, Description, and Creation of Test Scenarios

Open valid file:

Description: Attempting to open file

Expected outcome:

```
open D:\Java OOP\JavaSemestrialProject\src\test.txt
Successfully opened file
```

Print info:

Description: Attempting to print info

Expected outcome:

```
print
People{id='1', name='Ivan Ivanov', address='USA'}
People{id='2', name='Ivan Petrov', address='Bulgaria'}
People{id='1_2', name='Fridrich Petrov', address='France'}
People{id='3', name='null', address='null'}
```

Save file:

Description: Attempting to save file

Expected outcome:

```
save
Successfully saved file
```

SaveAs file:

Description: Attempting to save file in another file

Expected outcome:

```
saveas D:\Java OOP\JavaSemestrialProject\src\project.txt
Successfully saved file
```

Select 1 name

Description: Attempting to select name from element with id 1

Expected output:

```
select 1 name
 Ivan Ivanov
```

NChild 1 2

Description: Attempting to output 2 child from element with id 1

Expected output:

```
child 1 2
 USA
```

Delete 1 name

Description: Attempting to delete element with id 1 key name

```
delete 1 name
Name successfully deleted
```

Expected output:

Text 1

Description: Attempting to output text of element with id 1

```
text 1
<h:name> Ivan Ivanov </h:name>
<h:address> USA </h:address>
```

Expected output:

NewChild 4

Description: Attempting to add new child with id 4

```
newchild 4
You successfully add new child
```

Expected output:

Set 1 name Ivan Ivanov

Description: Attempting to set new name for 1 id

```
set 1 name Ivan Ivanov
Name successfully changed
```

Expected output:

XPath person/address:

Description: Attempting to find all addresses in file

```
person/address
 USA
 Bulgaria
 France
```

Expected output:

XPath person/address[2]:

Description: Attempting to find address of 2 person element

```
person/address[2]
 France
```

Expected output:

XPath person(@id)

Description: Attempting to find all id from file

```
person(@id)
1
2
1_2
3
```

Expected output:

XPath person(address="USA")/name

Description: Attempting to output person name with address USA

```
person(address="USA")/name
 Ivan Ivanov
```

Expected output:

XPath axis ancestor::person

Description: Attempting to output all lines till the person

```
ancestor::person
<root xmlns:h="http://www.w3.org/TR/html4/">
<h:people>
```

Expected output:

XPath axis person/child:

Description: Attempting to output all children of person

```
person/child
<h:name> Ivan Ivanov </h:name>
<h:address> USA </h:address>
<h:name> Ivan Petrov </h:name>
<h:address> Bulgaria </h:address>
<h:name> Fridrich Petrov </h:name>
<h:address> France </h:address>
```

Expexted output:

XPath axis person/parent:

Description: Attempting to output parent of tag person

```
person/parent
<h:people>
```

Expecting output:

XPath axis people/descendant

Descripting: Attempting to output all text between <people> and </people>

Expecting output:

```
people/descendant
<h:person id="1">
<h:name> Ivan Ivanov </h:name>
<h:address> USA </h:address>
</h:person>
<h:person id="2">
<h:name> Ivan Petrov </h:name>
<h:address> Bulgaria </h:address>
</h:person>
<h:person id="1_2">
<h:name> Fridrich Petrov </h:name>
<h:address> France </h:address>
</h:person>
<h:person id="3">
</h:person>
```

Chapter 5: Conclusion

5.1 Summary of the Achievement of Initial Goals

Implementation of parsing data from XML files. Flex realization of parsing data from file and handling of data. Openner and Saver handle operations allowing saving and opening XML files correctly.

XPath and another manipulation. Realization of XPath operations and other operations like select, editing and other help users easy and fast manipulate with data in file

Command Line Interface (CLI). Implementing CLI helps users easily use all functionality of program.

Test and debugging. All errors and bugs were fixed when testing and debug the application and now it is working pretty well and without big problems.

5.2 Directions for Future Development and Improvement

Implement GUI. Using GUI user can easily understand functionality of application

Full support of namespaces. Now it's just a simulation of namespace

Completely refuse of Person class. To make my project more flexible and read all types of XML I mask refuse of class for storing data