**JSON парсер**



**Технически Университет Варна**

Обектно ориентирано-програмиране

1 част-**Проект**

**Изготвил:** Александър Димитров

**Факултетен номер:** 21621618

**Група:** 1б

Глава 1. Увод 3

1.1. Описание и идея на проекта

1.2. Цел и задачи на разработката

1.3. Структура на документацията

Глава 2. Запознаване с проекта 4

2.1. Основни дефиниции, концепции и алгоритми

2.2. Дефиниране на проблеми и сложност на поставената задача

2.3. Подходи, методи за решаване на поставените проблемите

Глава 3. Проектиране 6

3.1. Обща структура на проекта пакети който ще се реализират

3.2. Диаграми и структура

Глава 4. Реализация 8

4.1.-4.8. Реализация на класове, алгоритми и оптимизации.

Глава 5. Command Line Interface 23

5.1. Успешна работа с файл

5.2. Невалидност

5.3. Работа със запазване на файлове

Глава 6. Заключение 28

5.1. Потребител

5.2. Подобрения по програмата

Глава 1. Увод

* 1. Този проект представлява приложение на команден ред. Предназначено да осигури функционалност за редактиране и манипулиране на JSON структури. Предоставя на потребителя методи за отваряне, модификация, валидация и запис на JSON файлове чрез текстовия интерфейс. Потребителя има простата задача да въведе команда заедно с определени аргументи, а програмата има функцията да разчете дадената команда, да я предаде на съответния метод, преработвайки дадените низове в параметри. Където тя най-накрая да се обработи и върне на потребителя.
  2. Програмата бива инструмент за работа с JSON файлове. Опростявайки процеса

на редактиране на данните то го прави удобен за работа от потребителя като му предоставя различни команди за работа с тях. Задачите приложени в моята реализация са свързани с работа с JSON файлове като включват:  
 -отваряне или създаване на файл

-запазване на модификации в същия файл или нов.

-валидиране на структурата им.

-отпечатване на данните или информация за валидните операции.

-намиране, създаване и изтриване на стойности с помощта на даден ключ.

-заместване на стойност или елемент в отворения файл.

Проекта се стреми да покрива основните принципи за създаване на добър софтуер, отделните функционалности са отделени в различни класове за по удобно четене, разработване и планиране. Чрез постигането на тези цели и задачи проектът позволява на потребителя да взаимодейства ефективно с JSON файлове, което улеснява редактирането и управлението на JSON данни в различни сценарии.

* 1. В документацията са разгледани отделните класове на програмата, обяснена е работата която трябва да извършва, как са създадени, как е разпределена работата между отделните класове за цялостната работа на програмата, и как потребителя трябва да взаимодейства с нея..

Дадена е структурата на проекта. Как е организиран всеки клас, техните променливи и методи и как са обвързани. Показана е функцията и логиката на всяка операция. Разглеждаме пътя на командата въведена от потребителя, до изходната реализация.

Глава 2. Преглед на предметната област

**2.1** Структурата на реализацията на този JSON парсер разглежда няколко дефиниции, концепции и алгоритми.

Програмата се състои от няколко класа: Main class, JsonFile, JsonEditor, JsonFinder, JsonPrinter, JsonSetter и JsonWriter,.

**2.2** В главният клас е реализирано изпълнението на парсера. Той създава връзката между потребителя и програмата, чрез въведената от него функция и параметри. Метода чете въведенета функция и я предава за обработка от съответния метод в JsonEditor

От своя страна, JsonEditor обработва операциите извършвани върху файла, като извиква метода от съответния клас. JsonEditor е определен да съдържа всички функции и от него, да се извикват нужните метода. Този клас пък имплементира класа JsonFile, използвайки го за всички методи свързани с обработка на файл, като се проверява дали има създаден файл, както и се използван неговите методи, за повечето функции.

Класът свързан с обработка на JSON файловете е JsonFile. Той има методи за зареждане на файлови данни и запазване на данните, както и прости методи нужни за много важни функции като за манипулация на пътя на файла и на данните.

Класът JsonFinder съдържа метод за намиране на стойност, от зададен ключ, чрез намирането и на символи срещани във всеки JSON файл като ‘,’ , ‘:’ , ‘}’ .

JsonSetter предоставя методи за модификация, създаване, изтриване на JSON данни от зададения път до стойността.

Клас JsonValidator с метод за просто валидиране на JSON данни, преминавайки всяко ниво на JSON структурата, отбелязва всеки символ за начало и край на низ/обект определящ един JSON файл. То преглежда дали за всеки символ ‘{‘ или ‘[‘ съществува съответстващ символ ‘}’ или ’]’. Ако не е открит такъв се подава съобщение с първото открито несъответствие, и неговата символна позиция.

JsonPrinter предоставя метод за принтиране на данните. По подобен начин на валидирането, то създава форматирането на файла чрез следене на символната архитектура на файла. При всяка отваряща скоба, пренасяме на нов ред, и отместваме следващата група данни с толкова места, според нивото на йерархията.

JsonWriter отстъпва метод за запис на JSON данни, използвайки библиотека за извършването на тази работа.

Програмата изпълнява основни функционалности. Сложността на повечето алгоритмите е относително ниска, достатъчно, имайки предвид повечето зависимости които трябва да покрие. Сложността е определена и от обработвания файл, зависейки от размера му.

**2.3** Манипулацията на JSON данните се извърша с помощта на превръщането им в StringBuilder обект.

Подхода на обработка е базиран на въведената от потребителя функция. Като в главния метод на класа Main я върти в цикъл и извиква съответните методи.Потребителя може да отваря, запазва, валидира, отпечатва, намира, задава, създава, изтрива и премества JSON данните. Предоставя и функции за изход и показване информация за гореспоменатите методи. Кода може да бъде лесно разширяван и модифициран.

Глава 3. Проектиране

**3.1** Основният пакет на проекта е bg.tu\_varna.sit.b1.f21621618, от който произлизат пакетът .controllers-съдържащ класове отговорни за контролирането на потока на приложението и обработка на въвеждане от потребителя.

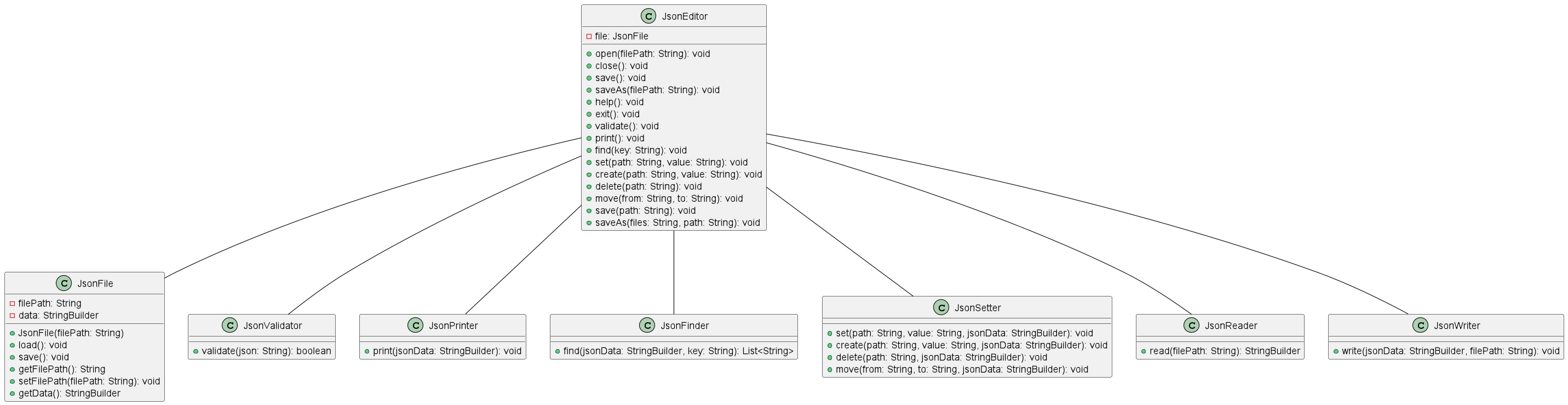
.models-пакет включващ класовете представляващи моделите на данни, използвани в приложението като JsonFile.

.utils-за помощни класове и методи.

.validators-класове за валидиране на JSON данни като JsonValidator

.io-за входно/изходните операции, за четене и запис на файлове.

**3.2**



[3.1]

Проекта има проста зависимост като класа JsonEditor зависи от:  
JsonFile-използван за манпулиране на JSON файловете за отваряне, запазване, управление.  
JsonValidator-използван за валидиране на JSON синтаксиса.

JsonPrinter-използван за отпечатване на JSON данните.

JsonFinder-използван за намиране конкретна стойност на JSON елемент.

JsonSetter-използван за създаване, изтриване, заместване на JSON елементите

JsonWriter-използван за запис на JSON данни във файл.

[3.1]

Използва се техниката Композиция, за създаване на променлива от тип на класа в JsonEditor от където се извиква метод на съответния клас.

private JsonFile file;

public void print(){  
 if (file==null)  
 System.out.println("No file is opened");  
 else {  
 JsonPrinter printer = new JsonPrinter();  
 printer.print(file.getData());  
 }  
}

[3.2]

В клас JsonEditor например създаваме променлива от типа на друг клас JsonFile. С нея верифицираме дали даден файл съществува и продължаваме с алгоритъма. Тук ние създаваме нова променлива от типа на класа принтер. С която извикваме съответния негов метод с аргумент съдържанието на файла.[3.2]

Глава 4. Реализация

**4.1** Main

JsonEditor editor = new JsonEditor(); [4.1]  
Console console = System.*console*();  
String input;  
  
if (console == null) {  
 System.*out*.println("No console available. Exiting...");  
 System.*exit*(1);  
}

Програмата започва с инициализация на нужните класове които ще ползваме сега или по нататък в програмата [4.1]:

JsonEditor за редактирането на JSON данните, следователно ще се ползва за извикване на следващите команда зададени от потребителя.

Console за извикване на конзолата, в която ще се пише входа на потребителя, който въвежда в нея

while (true) {  
 input = console.readLine("> ");  
  
 String[] commandParts = input.split(" ", 2);  
 String command = commandParts[0];

[4.2]

След като сме инициализирали нужните обекти, в Main класа [4.2], можем да видим безкраен цикъл, обработващ заявката на клиента, докато той не я приключи със съответната команда.

Програмата чете въведения текст първо го разделя на няколко фрагмента, според празните символи във входа. След това се взима първия фрагмент и се сравнява аргумент със различните методи във вътрешния цикъл. Там превръща фрагментите в параметри на функция. Накрая се извиква функцията от класа на JsonEditor със съответните аргументи[4.2].

if (command.equalsIgnoreCase("saveas")) {  
 String[] parameters = commandParts[1].split(" ", 2);  
 String filePath = parameters[1];  
 if (parameters.length == 1)  
 editor.saveAs(filePath);  
 else {  
 String file = parameters[0];  
 editor.saveAs(file, filePath);  
 }  
}

else if (command.equalsIgnoreCase("find")) {  
 String key = commandParts[1];  
 editor.find(key);  
} else if (command.equalsIgnoreCase("set")) {  
 String[] parameters = commandParts[1].split(" ", 2);  
 String path = parameters[0];  
 String value = parameters[1];  
 editor.set(path, value);  
}

else if (command.equalsIgnoreCase("exit")) { [4.3]  
 editor.exit();  
 break;  
}

else { [4.4]  
 console.printf("Invalid command. Type 'help' for a list of supported commands.");  
}

За край на програмата също се смята команда. При въвеждане на “exit” от потребителя, автоматично ще излезе от програмата, промени направени във файловете няма да се запазят, ако в момента е отворен файл, и той ще се затвори [4.3].

Ако командата на потребителя мине през всички сравнения без да даде позитивен резултат, ще се изведе съобщение за грешна команда, и ще поттикне потребителя, да види възможните команди като напише “help” [4.4].

**4.2** JsonEditor

Класът JsonEditor представлява частта от програмата, отговорна за обработката и изпълнението на командите на потребителя върху JSON данни. Този клас има следните методи и функционалности:

private JsonFile file; [4.5]  
  
public void open(String filePath){  
 file =new JsonFile(filePath);  
 if (!Files.*exists*(Paths.*get*(filePath))) { [4.6]  
 try {  
 Files.*createFile*(Paths.*get*(filePath));  
 } catch (IOException e) {  
 System.*out*.println("Can't create file: " + filePath);  
 e.printStackTrace();  
 return;  
 }  
 }  
 file.load(); [4.7]  
 System.*out*.println("Successfully opened "+filePath);  
}

частно поле от тип JsonFile представляващо текущо отворения JSON файл, с който програмата ще работи

С open е представен първия метод, който се използва за отваряне на JSON файл. Той създава нова инстанция на JsonFile и зарежда данните от файла [4.5], ако такъв съществува. Ако файла не съществува, се опитва да го създаде [4.6]. Накрая се извиква метод load на file за зареждане съдържанието на JSON файла [4.7]. A close метода просто затваря отворения файл. Без да се променя нищо по него. Включително и направените промени с други функции.

save и saveAs методите работят като извикват метода save() от file. Като saveAs извиква и метода за манипулация на пътя setFilePath(filePath) [4.8].

public void close(){  
 System.*out*.println("Successfully closed "+file.getFilePath());  
 file=null;  
}

public void save(){  
 if (file==null)  
 System.*out*.println("No file is opened");  
 else {  
 file.save();  
 System.*out*.println("Successfully saved "+file.getFilePath());  
 }  
}  
public void saveAs(String filePath){  
 if (file==null)  
 System.*out*.println("No file is opened");  
 else {  
 file.setFilePath(filePath);  
 file.save();  
 System.*out*.println("Successfully saved another "+file.getFilePath());  
 }  
}

[4.8]

public void help(){  
 System.*out*.println("The following commands are supported: \n" +  
 "open <file>\t\topens <file> \n" +  
 "close\t\t\tcloses currently opened file \n" +  
 "save\t\t\tsaves the currently open file \n" +  
 "saveas <file>\t\tsaves the currently open file in <file> \n" +  
 "help\t\t\tprints this information \n" +  
 "exit\t\t\texits the program \n"+  
 "validate\t\tchecks if the file is of JSON type \n" +  
 "print\t\t\tprints file's data \n" +  
 "find <key>\t\treturns the value of the given key \n" +  
 "set <path> <value>\tsets a new value at the given path \n" +  
 "create <path> <value>\tcreates a new element at the given path with the specified value \n" +  
 "delete <path>\t\tdeletes the specified element \n" +  
 "move <from> <to>\tmoves the element to a new path \n" +  
 "save <path>\t\tsaves the currently open file to the file path \n" +  
 "saveas <file> <path>\tsaves the currently open file to a specific file path\n"  
  
  
 );  
}

[4.9]

Имаме и функция help, с която първо се занимава потребителя. Определяща метода който да въведен, както и какви параметри се изискват [4.9].

public void exit(){  
 System.*out*.println("Exiting the program...");  
 System.*exit*(0); //https://linuxhint.com/end-java-program/  
}

[4.10]

Тук се обработва функцията за изход, като тя взаимодейства със самата система, за да посочи края на програмата. Прекратява се действието на нея, и CMD се връща в началното състояние [4.10].

Следващите функции са съставени от инстанция на класа който ще обработва функцията, и следователно се извика съответния метод, но не преди да определи дали има отворен файл. [4.11]

public void validate(){ [4.11]  
 if (file==null)  
 System.*out*.println("No file is opened");  
 else {  
 JsonValidator validator = new JsonValidator();  
 if (validator.validate(String.*valueOf*(file.getData())))  
 System.*out*.println("The file is valid");  
 else System.*out*.println("The file is not valid");  
 }  
}  
public void print(){  
 if (file==null)  
 System.*out*.println("No file is opened");  
 else {  
 JsonPrinter printer = new JsonPrinter();  
 printer.print(file.getData());  
 }  
}

public void find(String key) { [4.12]  
 if (file == null)  
 System.*out*.println("No file is opened");  
 else {  
 JsonFinder finder = new JsonFinder();  
 List<String> results = finder.find(file.getData(), key);  
 if (results.isEmpty()) {  
 System.*out*.println("No matches found");  
 } else  
 for (String result : results) {  
 System.*out*.println(result);  
 }  
 }  
}

Във find обаче работим с повече от една изходна данна. Следователно трябва да се създаде лист за нея още там [4.12]. Следва цикъл за извеждане на получените резултати.

**4.3** JsonFile

JsonFile е прост клас нужен за работа с файлове. Съдържа следните библиотеки нужни за програмата:

import java.io.FileWriter;  
import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.List;

Започва с създаване на частни полета за пътят и StringBuilder за извличане да данните на файла, заедно с конструктор за да се зададе желаният път

public class JsonFile {  
 private String filePath;  
 private StringBuilder data;  
  
 public JsonFile(String filePath) {  
 this.filePath = filePath;  
 this.data = new StringBuilder();  
 }

Следват функции за load и save

public void load(){//https://www.digitalocean.com/community/tutorials/java-read-file-line-by-line  
 try{  
 Path path= Paths.*get*(filePath);//https://www.tabnine.com/code/java/methods/java.nio.file.Files/readAllLines  
 List<String> lines= Files.*readAllLines*(path);  
 for (String line:lines){  
 data.append(line);  
 }  
} catch (IOException e){  
 e.printStackTrace();  
 }  
}  
public void save(){ [4.13]  
 try {  
 FileWriter writer=new FileWriter(filePath);  
 writer.write(data.toString());  
 writer.close();  
 } catch (IOException e){  
 e.printStackTrace();  
 }  
}

Load метода работи като се:

Създава обект от тип Path, който представлява пътя към файла с JSON данни на основата на filePath.

С използването на Files.readAllLines(path) чете всички редове от файла и ги зарежда в списък от низове lines.

След това методът обхожда всички редове в списъка и прилепя всякакви символи на JSON данни към общия data низ, за да събере цялостната JSON структура.

Save работи като:

Създава обект от тип FileWriter, който се използва за запис на данните във файл с път filePath.

Използвайки writer.write(data.toString()), методът записва цялостната JSON структура, представена като низ, във файла.

Накрая, методът затваря FileWriter обекта. [4.13]

public String getFilePath() {   
 return filePath;  
 }  
  
 public void setFilePath(String filePath) {  
 this.filePath = filePath;  
 }  
  
 public StringBuilder getData() {  
 return data;  
 }  
}

следват и базовите getters и setters нужни за работата на отварянето и запаметяването на файлове

**4.4** JsonWriter

import java.io.FileWriter;  
import java.io.IOException;  
  
public class JsonWriter {  
 public void write(StringBuilder jsonData, String filePath){  
 try{  
 FileWriter writer=new FileWriter(filePath);  
 writer.write(jsonData.toString());  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
}

Класа представлява нова важна част, отговорна за записването на промени в отворения, създадения или прехвърления файл.

Неговата структура е:

jsonData -низът, който съдържа JSON данните, които трябва да бъдат записани във файла.

filePath е пътят към файла, в който трябва да се запишат JSON данните.

В try блока на метода се създава FileWriter обект, който ще се използва за записване на данните във файла, чиито път е предоставен чрез filePath.

След това методът използва writer.write(jsonData.toString()), за да запише JSON данните във файла. jsonData се превръща в низ чрез toString(), преди да бъде записан.

След успешния запис методът затваря файла, използвайки writer.close(), за да освободи ресурсите и осигури коректно записване на данните.

Ако при записа се хвърли изключение от тип IOException, този код ще го обработи и изпечата стековата следа на грешката, за да се помогне за диагностицирането на проблемите.

**4.5** JsonValidator

private boolean validate(String json, int i, int depth) {  
 if (i >= json.length()) {  
 return true;  
 }  
 char curr = json.charAt(i);  
 switch (curr) {  
 case '{':  
 depth++;  
 return isObjectValid(json, i, depth); [4.11]  
 case '[':  
 depth++;  
 return isArrayValid(json, i, depth);  
 case '}':  
 if (depth == 0) { [4.12]  
 System.*out*.println("Error: Unexpected '}' at position " + i);  
 return false;  
 }  
 depth--;  
 break;  
 case ']':  
 if (depth == 0) {  
 System.*out*.println("Error: Unexpected ']' at position " + i);  
 return false;  
 }  
 depth--;  
 break;  
 default:  
 break;  
 }  
 return validate(json, i + 1, depth);  
}

private boolean isObjectValid(String json, int i, int depth) {  
 int balance = 1;  
 i++;  
 while (i < json.length() && balance > 0) {  
 char curr = json.charAt(i);  
 if (curr == '{') {  
 balance++;  
 } else if (curr == '}') {  
 balance--;  
 }  
 i++;  
 }  
 if (balance != 0) { [4.13]  
 System.*out*.println("Error: Missing '}' at position " + (i - 1));  
 return false;  
 }  
 return validate(json, i, depth - 1);  
}

Метода за валидиране в класа JsonValidator, се извършва чрез разглеждане на всеки символ в подадения JSON низ. Следейки баланса на броя отварящи и затварящи скоби. Използва се рекурсия и суитч в която за всяка отваряща скоба ‘{‘ или ’[‘ се увеличава променливата за „дълбочина“, както и се извиква друг метод за валидиране isObjectValid или isArrayValid съответно [4.11]. В тези методи, се проверява баланса на скобите. При неравен брой, става ясно, че липсва затваряща скоба. Така връщаме съобщение за липсваща такава, и позицията в която се е генерирал проблема [4.12]. Докато ако изгубим дълбочината на йерархията и той стане 0, се разбира, че има допълнителна затваряща скоба, означаващ неочакван край на структурата, следователно, извеждаме съответното съобщение [4.13].

**4.6** JsonFinder

public class JsonFinder {  
 public List<String> find(StringBuilder jsonData, String key){  
 List<String> results=new ArrayList<>();  
 String jsonSt= jsonData.toString();  
 int i=jsonSt.indexOf(key);  
 while (i!=-1){  
 int iStart=jsonSt.indexOf(":", i+key.length()); [4.14]  
 if(iStart!=-1){  
 int iEnd=jsonSt.indexOf(",", iStart);   
 if (iEnd==-1){ [4.15]  
 iEnd=jsonSt.indexOf("}", iStart);  
 }  
 if (iEnd!=-1){ [4.16]  
 String value=jsonSt.substring(iStart+1, iEnd).trim();  
 results.add(value);  
 }}  
 i=jsonSt.indexOf(key,i+key.length());  
 }  
 return results;  
 }  
 }

Метода за намиране на стойност в класа JsonFinder, се търсят инстанции на подадения като аргумент ключ, в низа jsonData, запазвани в списъка “results”, с индекс променлива за първото срещане на ключа [4.14] се прави цикъл извършван докато не се намерят всички инстанции на ключа. Извличайки началния индекс на символа „:“, успяваме да отделим ключа от стойността, съответно можем да намерим края на стойността чрез намиране на индекса на символ “,” или ”}” [4.15]. Стойността се извлича и с метода на substring, премахвайки го от външния текст [4.16].

import java.io.FileWriter; [4.17]  
import java.io.IOException;  
  
public class JsonWriter {  
 public void write(StringBuilder jsonData, String filePath){  
 try{ [4.18]  
 FileWriter writer=new FileWriter(filePath);  
 writer.write(jsonData.toString());  
 }  
 catch (IOException e){  
 e.printStackTrace();  
 }  
 }  
}

В класа JsonWriter импортираме 2 класа, java.io.FileWriter и java.io.IOException за записване на данни в текстови файлове, и за обработване на изключения [4.17]. С метода write записваме данни от StringBuilder в текство файл чрез предаване на пътят до него, както и показва информация за настъпала грешка чрез блока try [4.18].

**4.7** JsonPrinter

public class JsonPrinter {  
 public void print(StringBuilder jsonData){  
 int level = 0;  
 boolean commas = false;  
 boolean newLine = true;  
 for (int i = 0; i < jsonData.length(); i++){ [4.19]  
 char curr = jsonData.charAt(i);  
 if (curr == '"')  
 commas = !commas;  
 if (!commas){  
 if (curr == '{' || curr == '['){ [4.20]  
 level++;  
 newLine = true;  
 }  
 else if (curr == '}' || curr == ']'){  
 level--;  
 newLine = true;  
 }  
 else if (curr == ','){  
 newLine = true;  
 }}  
 if (newLine){  
 System.*out*.println();  
 for (int j = 0; j < level; j++){  
 System.*out*.print(" ");  
 }  
 newLine = false;  
 }  
 if (!Character.*isWhitespace*(curr) || (curr == ' ' && commas)) {  
 System.*out*.print(curr);  
 }  
 }  
 System.*out*.println();  
 }  
}

В класа JsonPrinter се предоставя метод за принтиране на данните, във форматиран вид. По подобен начин на метода за валидиране, представя променлива за следене дълбочината на структурата- „level“. Започва с цикъл минаващ през всеки символ в JSON низа [4.19]. Определя символите дали са с кавички и принадлежат на низа игнорирайки другите специални символи в него. След него определя следващите символи, като при съвпадение със запетая или отваряща запетая ще маркира отпечатване на нов ред, както и се разглежда и променливата за ниво, според която прави определен отстъп от началото. При отваряща скоба променливата се увеличава, при затваряща се намалява. Така се получава, лесно четим и разбираем начин за отпечатване на отворения файл [4.20].

**4.8** В класа JsonSetter разглеждаме 4 метода за модификация на данни:

public void set(String path, String value, StringBuilder jsonData) {  
 String[] divider = path.split("\\."); [4.21]  
 String json = jsonData.toString();  
  
 for (int i = 0; i < divider.length - 1; i++) {  
 String div = divider[i].trim();  
 int iStart = json.indexOf(div);  
 int iEnd = json.indexOf(":", iStart);  
  
 if (iStart != -1 && iEnd != -1) {  
 json = json.substring(0, iEnd) + ":" + json.substring(iEnd + 1);  
 }  
 else {  
 System.*out*.println("Path not found");  
 }  
 }  
  
 String segment = divider[divider.length - 1].trim();  
 int iStart = json.indexOf(segment); [4.22]  
 int iEnd = json.indexOf(":", iStart) + 1;  
  
 if (iStart != -1 && iEnd != -1) {  
 int iStartValue = json.indexOf(":", iStart) + 1;  
 int iEndValue = json.indexOf(",", iStartValue)+1;  
  
 if (iEndValue == -1||iEndValue>json.indexOf("}", iStartValue)) {  
 iEndValue = json.indexOf("}", iStartValue);  
 }  
  
 if (iEndValue != -1) {  
 json = json.substring(0, iStartValue) + "\"" + value + "\"" + json.substring(iEndValue-1);  
 }  
 }  
  
 jsonData.setLength(0);  
 jsonData.append(json);  
}

Със метода set редактираме стойността на определен JSON обект. Метода обработва подадения път като го разделя на части според точките в него [4.21]. Така програмата обхожда всяка част от пътя стигайки до исканият елемент. Когато локализира стойността нa елемент, го заменя с подадената към метода стойност [4.22].

public void create(String path, String value, StringBuilder jsonData) {  
 String[] divider = path.split("\\.");  
 String json = jsonData.toString();  
  
 String segment = divider[divider.length - 1].trim();  
  
 if (json.contains(segment)) {  
 System.*out*.println("Path already exists"); [4.23]  
 }  
 else  
 System.*out*.println("Value created in " + path);  
  
 int iStart = json.lastIndexOf("}"); [4.24]  
 if (iStart != -1) {  
 json = json.substring(0, iStart) + ",\"" + segment + "\":\"" + value + "\"" + json.substring(iStart);  
 }  
  
 jsonData.setLength(0);  
 jsonData.append(json);  
}

Със метода create се разделя даденият път на части, проверява дали съществува (подавайки и съобщение за това [4.23]) и създава новата променлива в обекта [4.24].

public void delete(String path, StringBuilder jsonData) {  
 String[] divider = path.split("\\.");  
 String json = jsonData.toString();  
 String segment = divider[divider.length - 1].trim();  
  
 if (!json.contains(segment)) { [4.25]  
 System.*out*.println("Path not found");  
 }  
 else  
 System.*out*.println("Value deleted in " + path);  
 int iTest=json.indexOf(segment)-10; [4.26]  
 int iStart = json.indexOf(",", iTest);  
 if (iStart>json.indexOf("{", iTest)&&json.indexOf("{", iTest)!=-1){  
 iStart=json.indexOf("{", iTest)+1;  
 }  
 int iEnd = json.indexOf(":", iStart);  
 int iEndValue = json.indexOf(",", iEnd);  
  
 if (iEndValue == -1||iEndValue>(json.indexOf("}", iEnd)-1)) {  
 iEndValue = json.indexOf("}", iEnd)-1;  
 }  
 json = json.substring(0, iStart) + json.substring(iEndValue + 1);  
 jsonData.setLength(0);  
 jsonData.append(json);  
}

По подобен начин в метода delete се локализира подадения елемент (извежда съответното съобщение ако не е намерен [4.25]) и изтрива ключа, стойността и съответните специални символи присъщи на обекта, без да нарушава структурата му [4.26].

public void move(String from, String to, StringBuilder jsonData) {  
 String json = jsonData.toString();  
  
 if (!json.contains(from)) {  
 System.*out*.println("Element not found");  
 }  
 else  
 System.*out*.println("Value moved from " + from + " to " + to);  
  
 json = json.replace(from, to);  
  
 jsonData.setLength(0);  
 jsonData.append(json);  
}

[4.27]

В метода за заместване move, се търси подадения като аргумент елемент, при неналичие на такъв се извежда съответното съобщение, в противен случай, елемента се замества с този подаден като аргумент [4.27].

Глава 5. Command Line Interface

**5.1** Успешна работа с файл

В този пример представяме успешна работа с файл с име developer.json. Той преминава през проверката за JSON валидност, търсим първото име на студента, създаваме нов елемент „възраст“, променяме стойността за година с команда set student.college.year 3, изтриваме елемента за оценка, и заместваме елемента “subject” с “course”. Показано е новото извеждане на текста, с успешно направени промени, минаващ проверката за валидност отново. След това бива запазен, като можем да видим след повторно отваряне, промените за запазени.

> open developer.json

Successfully opened developer.json

> validate

The file is valid

> print

{"student":

{"subject":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"2"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

,"grade":"NA"

}

}

}

> find firstName

"Alexander"

> create student.age 20

Value created in student.age

> set student.college.year 3

Value set in student.college.year

> delete student.project.grade

Value deleted in student.project.grade

> move subject course

Value moved from subject to course

> print

{"student":

{"course":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"3"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

}

}

,"age":"20"

}

> validate

The file is valid

> save

Successfully saved developer.json

> open developer.json

Successfully opened developer.json

> print

{"student":

{"course":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"3"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

}

}

,"age":"20"

}

**5.2** Невалидност

> open develop.json

Successfully opened develop.json

> print

{"student":

{"course":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"3"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

}

,"age":"20"

}

> validate

Error: Missing '}' at position 367

The file is not valid

> find professor

No matches found

> set student.college.status closed

Path not found

> delete student.driverLicense

Path not found

> move something nothing

Element not found

> close

Successfully closed develop.json

В този пример работим с файл който липсва затворена скоба. След като бъде отворен той не преминава през валидацията. Получавайки съобщение за липсваща такава и символната позиция на която се очаква да има. В последствие се опитваме да търсим, задаваме, изтриваме и преместваме несъществуващ елемент, за което получаваме и логичните съобщения.

**5.3** Работа със Запазване на файлове

> open developer.json

Successfully opened developer.json

> print

{"student":

{"subject":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"2"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

,"grade":"NA"

}

}

}

> saveas an otherfile.json

Successfully saved another otherfile.json

> delete student.project.grade

Value deleted in student.project.grade

> print

{"student":

{"subject":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"2"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

}

}

}

> save

Successfully saved otherfile.json

> close

Successfully closed otherfile.json

> open developer.json

Successfully opened developer.json

> print

{"student":

{"subject":"OOP1project"

,"firstName":"Alexander"

,"lastName":"Dimitrov"

,"college":

{"collegeName":"TU-Varna"

,"faculty":"FITA"

,"facultyNumber":"21621618"

,"group":"1b"

,"specialty":"SIT"

,"year":"2"

}

,"project":

{"name":"JsonParser"

,"developer":"Alexander"

,"grade":"NA"

}

}

}

> close

Successfully closed developer.json

В този пример показваме работата на метода за “запазване като” (saveas). Отваряма файла developer.json, след това извикваме метода saveas който създава нов файл с даденото име прехвърля JSON данните от отворения файл и прехвърля контрола към него. След това правим промени по него и ги запаметяваме, правейки 2 различни файла.

Глава 6. Заключение

**6.1** Потребител

Началната точка на проекта е main метода от съответния клас. Инициализира се JsonEditor инстанция и такава за Console.

В цикъл се иска от потребителя да въведе команда.

Въведеното от клиента се разделя на няколко части, и според функция програмата извиква съответния метод от JsonEditor инстанцията.

В него се съдържат всички методи използвайки други класове за реализацията им.

За потребителя е нужно да зададе командата за помощ за получаване на всички поддържани функции от програмата.

Крайната команда за програмата е exit за изход от вечния цикъл.

**6.2** Подобрения по програмата:

Метода за верификация използва единствено баланса на специалните символи. Като файлове с ненужни или липсващи запетаи, двуеточия и т.н. ще минат безпроблемно през верификацията.

Метода за създаване, не може да създаде нова група или низ от елементи.

След изтриване на цяла група обекти, метода за изтриване, ще остави ненужните скоби във файла.

За по-добър редактор, трябват повече инструменти и функции, за което е възможно да разширим нашата програма с допълнителни методи и класове.

Методите извършват доста работа и функции, поради което кодът би могъл да бъде по-добре оптимизиран.