Json Parser

Изготвил: Йово Неделчев

От: Катедра СИТ

2Курс 1А Група

Студентски номер:23621691

**1.Задача на проекта :**

JSON е популярен текстов формат за описване на данни. Да се напише програма, която работи с файлове в такъв формат.

След като приложението отвори даден файл, то трябва да може да извършва посочените по-долу операции, в допълнение на общите операции (open, close, save, save as, help и exit):

Validate : Да се направи проверка дали отворения файл е валиден спрямо синтаксиса на JSON. Ако има някакъв проблем, програмата трябва да съобщи максимално ясно какво и къде не е наред, така че потребителят да може да го поправи.

Print : Да се изведе съдържанието на обекта в максимално четим вид.

Search <key> : Да се провери дали в обекта се съдържат данни, записани под този ключ и ако да – да изведе списък от всички такива данни.

Set <path> <string> : При подаване на пълен път <path> към даден елемент, да се замени стойността на посочения елемент с нов обект, получен от низа <string> според синтаксиса на JSON стойност, ако такъв елемент съществува и е единствен и ако символният низ е коректен. В противен случай трябва да обяви каква е грешката.

Create <path> <string> : При подаване на пълен път <path> към даден елемент да се добави такъв елемент, получен от низа <string> според синтаксиса на JSON стойност. Ако такъв елемент съществува, това трябва да се обяви като грешка. В случай, че последните елементи от пътя не съществуват, да се създадат. Ако символният низ или пътят не са коректни, да се съобщи с подходящо описание на грешката.

Delete <path> : При подаване на пълен път <path> до елемент, да се изтрие, ако такъв съществува или да съобщи на потребителя при некоректен път.

Move <from> <to> : Всички елементи, намиращи се на път <from> да бъдат преместени на пътя <to>.

Save [<path>] / Saveas <file> [<path>] : Командите save и saves да работят с произволен път, като записват в текущия или в нов файл обекта на дадения път, ако съществува. Ако <path> не е подаден, да се записва целият обект, който в момента е зареден в паметта.

**2.Съдържание в програмата**

2 Папки за Json файлове : JSON\_Files и FilesFolder2

1 Абстрактен клас - BaseWriter и 5 класа , които го наследяват :

Basic Writer , ContextExtendedWriter , FullWriter , TwoFileWriter и LocationExtendedWriter

1 клас за запазване на временна информация за файловете , докато програмата работи – PathReference

1 клас със Main метод и главната логика на програмата – JsonReader

**3.Код**

**3.1. BaseWriter клас и неговите наследници**

BaseWriter абстрактен клас :

public abstract class BaseWriter {  
 public void Write(String word){};  
  
 public void FullWrite(String sentence){};  
  
 public BaseWriter() {  
 }  
}

BaseWriter е абстрактен клас с 2 функции : void Write и void FullWrite , които взимат за аргумент 1 елемент String , като във void Write елемента трябва да представлява 1 дума , а във FullWrite – 1 изречение . Класът има и празен конструктор , за да могат наследниците му да се инициализират.

BasicWriter клас :

public class BasicWriter extends BaseWriter{  
 @Override  
 public void Write(String word) {  
 System.*out*.println("Напишете името на файла след '"+word+"'");  
 }  
  
 public BasicWriter() {  
 }  
}

BasicWriter наследява BaseWriter и пренаписва void Write функцията да изписва съобщение на конзолата „Напишете името на файла след‘“ + стойността на елемента word + “’”. Класът има и празен конструктор за инициализиране.

ContentExtendedWriter клас:

public class ContentExtendedWriter extends BaseWriter{  
 @Override  
 public void Write(String word) {  
 System.*out*.println("Напишете името на файла и новото съдържание след '"+word+"'");  
 }  
  
 public ContentExtendedWriter() {  
 }  
}

ContentExtendedWriter наследява BaseWriter и пренаписва void Write функцията да изписва съобщение на конзолата „Напишете името на файла и неговото съдържание след‘“ + стойността на елемента word + “’”. Класът има и празен конструктор за инициализиране.

LocationExtendedWriter клас:

public class LocationExtendedWriter extends BaseWriter{  
 @Override  
 public void Write(String word) {  
 System.*out*.println("Напишете новата локация и името на файла след '"+word+"'");  
 }  
  
 public LocationExtendedWriter() {  
 }  
}

LocationExtendedWriter наследява BaseWriter и пренаписва void Write функцията да изписва съобщение на конзолата „Напишете новата локация и името на файла след‘“ + стойността на елемента word + “’”. Класът има и празен конструктор за инициализиране.

TwoFileWriter клас :

public class TwoFileWriter extends BaseWriter{  
 @Override  
 public void Write(String word) {  
 System.*out*.println("Напишете името на първия и втория файл след '"+word+"'");  
 }  
  
 public TwoFileWriter() {  
 }  
}

TwoFileWriter наследява BaseWriter и пренаписва void Write функцията да изписва съобщение на конзолата „Напишете името на първия и втория файл след‘“ + стойността на елемента word + “’”. Класът има и празен конструктор за инициализиране.

FullWriter клас :

public class FullWriter extends BaseWriter{  
 @Override  
 public void FullWrite(String sentence) {  
 System.*out*.println(sentence);  
 }  
  
 public FullWriter() {  
 }  
}

FullWriter наследява BaseWriter и пренаписва void FullWrite функцията , така че да изписва на конзолата стойността на елемента sentence. Класът има и празен конструктор за инициализиране.

**3.2. PathReference клас**

import java.nio.file.Path;  
  
public class PathReference {  
 private Path location;  
 private String content;  
  
 public PathReference(Path path){  
 this.location = path;  
 content = null;  
 }  
  
 public Path getLocation() {  
 return location;  
 }  
  
 public void setLocation(Path location) {  
 this.location = location;  
 }  
  
 public String getContent() {  
 return content;  
 }  
  
 public void setContent(String content) {  
 this.content = content;  
 }  
}

Импортира се java.nio.file.Path пакета за да се позволи работата с локации на файлове. Класът има 2 private променливи Path location , която съдържа информация за локацията на файл и String content , която съдържа информация за съдържанието на файл. Класът има конструктор , който взима за аргумент 1 елемент Path path и при инициализация location приема стойността на path , а content приема стойност null. Класът има още 4 функций , които са Getters и Setters за 2 му променливи.

**3.3. JsonReader клас**

Импортнати пакети :

import java.io.IOException;  
import java.nio.file.Files;  
import java.nio.file.Path;  
import java.nio.file.Paths;  
import java.util.\*;

java.io.IOException позволява работата с изключения , java.util.\* позволява работата с колекциите като List<> ,ArrayList<> и Map<>,

а java.nio.file.Files/Path/Paths позволяват работата с файлове .

open функция:

public static void open(PathReference path , String fileName,List<PathReference>list) throws IOException {  
  
 Boolean ExistsInList = false;  
  
 if(path.getLocation() == null) {  
 for(PathReference pr : list){  
 if(pr.getLocation().equals(Paths.*get*("src\\" + fileName))){  
 path.setLocation(pr.getLocation());  
 path.setContent(pr.getContent());  
 ExistsInList = true;  
 System.*out*.println("Файлът е успешно отворен");  
 }  
 }  
 if(!ExistsInList) {  
 if (Files.*exists*(Paths.*get*("src/" + fileName)) && Files.*isReadable*(Paths.*get*("src/" + fileName))) {  
 try {  
 path.setLocation(Paths.*get*("src/" + fileName));  
 path.setContent(Files.*readString*(path.getLocation()));  
 //String content = Files.readString(path.location); // Opens and reads the file  
 System.*out*.println("Файлът е успешно отворен");  
 //System.out.println("Съдържание:\n" + content);  
 } catch (IOException e) {  
 System.*out*.println("Файлът съществува , но не беше успешно отворен");  
 }  
 } else {  
 System.*out*.println("Файлът не съществува или не може да бъде прочетен");  
 return;  
 }  
 }  
 }  
 else{  
 System.*out*.println("Дръг файл е отворен . Затворете отвореният файл , за да може да отворите друг файл.");  
 return;  
 }  
   
}

Функцията взима като аргументи елементи : PathReference path , където ще се съхраняват данните за отворения файл в отделен клас , String fileName , който е името на локацията на файла и List<PathReference> list , в който се съдържат всичките PathReference елементи направени , докато програмата работи.

За начало се създава локална променлива Boolean ExistsInList , със стойност false . След това се проверява дали локацията във path елемента е празна , и ако е , се влиза във for цикъл на всички елементи от list елемента и се проверява дали там има елемент с еднаква локация , и ако има , path елемента взима стойностите си за локация и съдържание от този съществуващ елемент , след което ExistsInList става на true и се изписва съобщение в конзолата „Файлът е успешно отворен“. При случай , че няма елемент с такава локация в list , се проверява дали съществува такъв четим файл в проекта . Ако съществува , то тогава path елемента взима за стойност дадената локация fileName чрез Paths.Get(“src/”+fileName) и взима съдържанието на файла чрез Files.readString(path.getLocation()) , след което се изписва съобщение в конзолата „Файлът е успешно отворен“. Ако файлът не може да бъде успешно отворен се изписва на конзолата съобщенитео „Файлът съществува , но не беше успешно отворен“ , а ако файлът не съществува или не може да бъде прочетен се изписва на конзолата съобщенитео „Файлът не съществува или не може да бъде прочетен“. В случай , че друг файл е вече отворен (което ще стане ясно как работи в main метода) се изписва на конзолата съобщение „Друг файл е вече отворен . Затворете отвореният файл , за да можете да отворите друг файл.“.

close функция :

public static void close(PathReference path , String fileName){  
 if(path.getLocation()!=null) {  
 if (path.getLocation().equals(Paths.*get*("src\\" + fileName))) {  
 path.setLocation(null);  
 path.setContent(null);  
 System.*out*.println("Файлът е успешно затворен");  
 } else {  
 //System.out.println("Given path: " + path.location);  
 //System.out.println("Expected path: "+"src\\JSON\_Files\\" + fileName);  
 System.*out*.println("Няма отворен файл с такова наименование");  
 }  
 }  
 else{  
 System.*out*.println("Няма отворен файл с такова наименование");  
 }  
  
}

Функцията взима като аргументи елементи : PathReference path, който представлява отвореният файл в програмат и String fileName , който представлява дадена име на локация на файл.

Първо се проверява дали локацията във path елемента е празна , ако не е , се проверява дали локацията е същата като тази на даденото име на локация fileName . В слчай , че са еднакви локации , локацията и съдържанието на path елемента се премахват като им се даде стойност null и след това се изписва съобщение на конзолата „Файлът е успешно затворен“. В случай , че не се едакви локациите , тогава се изписва съобщение на конзолата „Няма отворен файл с такова наименование“. В случай , че локацията на дедения елемент path е празна се изписва съобщение на конзолата „Няма отворен файл с такова наименование“.

print функция :

public static void print(PathReference path){  
 if(path.getContent()==null || path.getLocation()==null){  
 System.*out*.println("Няма отворен файл все още");  
 }  
 else {  
 try {  
 //System.out.println(path.content);  
 System.*out*.println(Files.*readString*(path.getLocation()).trim());  
 } catch (IOException e) {  
 System.*out*.println("Грешка при изписването на съдържанието на отвореният файл");  
 }  
 }  
}

Взима като аргумент елемент от тип PathReference . Ако съдържаниете или локацията на елемента са празни (те се попълват , когато се извика open функцията) , тогава се изписва в конзолата съобщението „Няма отворен файл все още“. Ако те съществуват се изписва съдържанието на файла (чете се директно от локацията на файла вместо съдържанието на елемента , защото в комбинация с другите методи в програмата може да има разминавания при променяне и изтриване на файловете). При изключение се изпизва в конзолата съобщението „Грешка при изписването на съдържанието на отвореният файл“.

parseJson функция :

public static Map<String, String> parseJson(String json) {  
 Map<String, String> map = new LinkedHashMap<>();  
  
 json = json.trim().replaceAll("[{}\"]", "");  
 String[] pairs = json.split(",");  
  
  
 for (String pair : pairs) {  
 String[] keyValue = pair.split(":");  
 if (keyValue.length == 2) {  
 map.put(keyValue[0].trim(), keyValue[1].trim());  
 }  
 }  
 return map;  
}

Функцията връща Map<String,String> и приема String като аргумент. Първо се създава локална променлива от тип LinkedHashMap<String,String> map . След това чрез json.trim() се махат всички скоби от JSON файла , който беше подаден като аргумент и се създава масив от тип String[] , в който се пазят данните от JSON файла. След това чрез for цикъл двойките от данни се поставят в map-a след като бъдат преместени във още един масив от тип String[] , който ги разделя на 2те части ключ и стойност. На края се връща map-a с новите стойности.

search функция :

public static void search(PathReference path , String keyName) throws IOException {  
 if(path==null || path.getLocation()==null){  
 System.*out*.println("Няма Отворен файл");  
 return;  
 }  
 Boolean printed = false;  
 String content = new String(Files.*readAllBytes*(path.getLocation()));  
 Map<String, String> jsonMap = *parseJson*(content);  
 for (String key : jsonMap.keySet()) {  
 if(keyName.equals(key))  
 {  
 String value = jsonMap.get(keyName);  
 System.*out*.println(key+ ": "+value);  
 printed = true;  
 }  
 }  
 if(!printed){  
 System.*out*.println("Не бешен намерен ключ с такова име");  
 }  
}

Функцията взима за аргументи 2 елемента от тип PathReference и String и може да хвърли IOException изключение. Тя служи за намиране на двойка данни от JSON файл по името на ключа.Първо се проверява дали path елемента е валиден , и ако не е , се изписва на конзолата съобщение „Няма отворен файл“ и се излиза от функцията. Създават се локални променливи : Boolean printed , String conent и Map<String,String> jsonMap . printed е за определяне дали успешно е намерен ключ с даденото име , при което получава стойност true. content взима съдържанието на файла с локация равна на path елемента. JsonMap извиква функцията parseJson и дава като аргумент content. След това с for цикъл се взимат всички ключове от jsonMap , и ако те за равни на keyName , то тогава се създава локална променлива String value , която взима стойноста от двойката ,като извиква функцията jsonMap.get(keyName) и printed получава стойност true. В случай , че не е намерен ключ с такова наименование ,printed ще си остане със стойност false и ще се изпише на конзолата „Не беше намерен ключ с такова име“.

set функция :

public static void set(Path filePath , String newContent,List<PathReference> list)  
{  
 /\*if(jsonMap.containsKey(keyName)){  
 jsonMap.put(keyName,newValue);  
 }  
 else {  
 System.out.println("Ключ с име : '" + keyName + "' не беше намерен.");  
 }\*/  
 PathReference pathRef = new PathReference(filePath);  
 if (Files.*exists*(filePath)) {  
 /\*try {\*/  
 for(PathReference pr : list) {  
 if (pr.getLocation().equals(filePath)) {  
 pr.setContent(newContent);  
 System.*out*.println("JSON файлът беше пренаписан успешно.");  
 return ;  
 }  
 }  
 //Files.write(filePath, newContent.getBytes());  
 pathRef.setContent(newContent);  
 list.add(pathRef);  
 System.*out*.println("JSON файлът беше пренаписан успешно.");  
 } /\*catch (IOException e) {  
 System.out.println("Грешка при презаписването на JSON файлът.");  
 }\*/  
 //}  
 else{  
 System.*out*.println("JSON файлът с такъв път не е намерен.");  
 }  
}

Функцията приема като аргументи елементи Path filePath , който представлява локацията на файл , String newContent , който представлява новото съдържание , което ще бъде написано във файла и List<PathReference> list , в който се съдържат всичките PathReference елементи направени , докато програмата работи.

Първо се създава нова локална променлива PathReference pathRef , коята взима като стойност за локацията си елемента filePath. След това се проверява дали съществува файл с такава локация в проекта , и ако съществува се влиза във for цикъл , който минава през всички елементи на list , и ако намери елемент с локация равна на filePath , променя съдържанието на този елемент на newContent и се изписва съобщение на конзолата „JSON файлът беше пренаписан успешно.“ и се излиза от функцията. Ако няма елемент с еднаква локация , тогава съдържанието на pathRef локалната променлива се променя на newContent и в list се добавя pathRef елемента и се изписва съобщение на конзолата „JSON файлът беше пренаписан успешно.“. Ако няма файл с дадената локация в проекта изписва съобщение на конзолата „JSON файлът с такъв път не е намерен.“.

save функция :

public static void save(String fileName,List<PathReference> list){  
 Path filePath = null;  
 String fileContent = null;  
 for(PathReference pr : list){  
 if(pr.getLocation().equals(Paths.*get*("src\\" + fileName))){  
 filePath = pr.getLocation();  
 fileContent = pr.getContent();  
 }  
 }  
 if (filePath != null && fileContent != null) {  
 try {  
 Files.*write*(filePath, fileContent.getBytes());  
 System.*out*.println("Промените бяха запазени във файла.");  
 } catch (IOException e) {  
 System.*out*.println("Грешка при запазването на файла.");  
 }  
 } else {  
 System.*out*.println("Няма файл за запазване.");  
 }  
}

Функцията взима като аргументи елементи : String fileName , който е името на локацията на файл и List<PathReference> list , в който се съдържат всичките PathReference елементи направени , докато програмата работи.

Създават се 2 локални променливи Path filePath и String fileContent , и двете със стойност null . Влиза се във for цикъл , който преминава през всички елементи от list , и ако има елемент с локация равна на локацията с име fileName , тогава filePath и fileContent взимат стойностите си от локацията и съдържанието на елемента. След това се проверява дали filePath и fileContent не са null , и ако не са , тогава във файл с локация равна на filePath се записва съдържанието от fileContent чрез метода Files.write(filePath, fileContent.getBytes()) . След това се изписва съобщение в конзолата „Промените бяха запазени във файла.“. При върнато изключение IOException се изписва съобщение в конзолата „Грешка при запазването на файла.“. В случай , че filePath или fileContent са равни на null , се изписва съобщение в конзолата „Няма файл за запазване.“.

saveAs функция :

public static void saveAs(String fileName,String newPathName,List<PathReference> list) throws IOException {  
 Path filePath = Paths.*get*("src\\"+newPathName);  
 Path originalLocation = Paths.*get*("src\\" + fileName);  
 String fileContent = Files.*readString*(originalLocation);  
 for(PathReference pr : list){  
 if(pr.getLocation().equals(originalLocation)){  
 //filePath = pr.location;  
 fileContent = pr.getContent();  
 }  
 }  
 if (filePath != null && fileContent != null) {  
 try {  
 *delete*(originalLocation,list,false);  
 Files.*write*(filePath, fileContent.getBytes());  
 System.*out*.println("Промените бяха запазени във файла.");  
 } catch (IOException e) {  
 System.*out*.println("Грешка при запазването на файла.");  
 }  
 } else {  
 System.*out*.println("Няма файл за запазване.");  
 }  
}

Функцията взима като аргументи елементи : String fileName , който е името на локацията на файл , String newPathName който служи като името на новата локация на файл и List<PathReference> list , в който се съдържат всичките PathReference елементи направени , докато програмата работи.

Създават се 2 локални променливи Path filePath и Path originalLocation , които представляват оригиналната локацията на дадения файл и новата локация на дадения файл , като използват fileName и newPathName . След това се създава още 1 локална променлива String fileContent , която взема за стойност съдържанието на файла от съществуващата локация. След това се влиза във for цикъл , който минава през всички елементи от list и ако има елемент с еднаква локация като оригиналната локация на файла , то тогава fileContent взима за стойност съдържанието на елемента. Излиза се от for цикъла и се проверява дали filePath и fileContent не са равни на null , и ако не са , се изтрива файла с оригиналната локация (чрез функция delete ,за която се обяснява по-напред в документацията) и се записва новото съдържание от fileContent в нов файл с локация filePath и на края се изписва на конзолата съобщението „Промените бяха запазени във файла.“.

Ако се върне изключение се изписва на конзолата съобщенитео „Грешка при запазването на файла.“ , а при това filePath или fileContent да са равни на null , се изписва на конзолата съобщението „Няма файл за запазване.“.

create функция :

public static void create(Path filePath , String newContent){  
 if (Files.*exists*(filePath)) {  
 System.*out*.println("JSON файлът с тавък път вече съществува.");  
 }  
 else {  
 try {  
 Files.*write*(filePath, newContent.getBytes());  
 System.*out*.println("JSON файлът беше създаден успешно.");  
 } catch (IOException e) {  
 System.*out*.println("Грешка при създаването на JSON файлът.");  
 }  
 }  
}

Функцията взима като аргументи елементите Path filePath , който представлява локацията на файл , String newContent , който представлява новото съдържание , което ще бъде написано във файла . Първо се проверява дали същестува файл с локация като filePath , и ако съществува , се изписва съобщение на конзолата „JSON файлът с такъв път вече съществува.“. Ако не съществува такъв файл , се създава нов файл в локацията на filePath със съдържание newContent , чрез функцията Files.write(filePath,newContent.getBytes()) и се изписва съобщение на конзолата „JSON файлът беше създаден успешно.“. При върнато изключение се изписва съобщение на конзолата „Грешка при създаването на JSON файлът.“.

delete функция :

public static void delete(Path filePath , List<PathReference> list ,Boolean sendMessage){  
 if (Files.*exists*(filePath)) {  
 try {  
 Files.*delete*(filePath);  
 if(sendMessage) {  
 System.*out*.println("JSON файлът беше изтрит успешно.");  
 }  
 list.removeIf(PathRef -> PathRef.getLocation().equals(filePath));  
 } catch (IOException e) {  
 System.*out*.println("Грешка при изтриването на JSON файлът.");  
 }  
 } else {  
 System.*out*.println("JSON файлът с такъв път не е намерен.");  
 }  
}

Функцията приема като аргументи елементи Path filePath , който представлява локацията на файл , List<PathReference> list , в който се съдържат всичките PathReference елементи направени , докато програмата работи и Boolean sendMessage , с който се определя дали да се изпрати съобщение при успешно изтрит файл.

Първо се проверява дали същестува файл с локация като filePath , и ако съществува , той се изтрива чрез функция Files.delete(filePath) и ако sendMessage е равно на true , тогава се изписва съобщение на конзолата „JSON файлът беше изтрит успешно.“. Опцията да има избор между това да се изписва съобщение и да не се е с цел, когато delete функцията се извиква в друг метод , който не е main метода , да не се изписва съобщение (напримерно в saveAs функцията). След това от list се премахват елементи , които имат локация равна на filePath.

При връщане на изключение се изписва на конзолата съобщение „Грешка при изтриването на JSON файлът.“ , а ако не съществува файл с локация равна на filePath , тогава се изписва съобщение на конзолата „JSON файлът с такъв път не е намерен.“.

move функция :

public static void move(Path pathFrom , Path pathTo){  
 try {  
 if (!Files.*exists*(pathFrom) || !Files.*exists*(pathTo)) {  
 System.*out*.println("JSON файлът с такъв път не е намерен.");  
 return;  
 }  
  
 String contentFrom = Files.*readString*(pathFrom).trim();  
 String contentTo = Files.*readString*(pathTo).trim();  
  
  
 boolean fromIsEmpty = contentFrom.isEmpty() || contentFrom.equals("{}");  
 boolean toIsEmpty = contentTo.isEmpty() || contentTo.equals("{}");  
  
 String mergedContent = "";  
  
 if (fromIsEmpty && toIsEmpty) {  
 mergedContent = "{}";  
 } else if (fromIsEmpty) {  
 mergedContent = contentTo;  
 } else if (toIsEmpty) {  
 mergedContent = contentFrom;  
 } else {  
  
 if (contentFrom.endsWith("}")) {  
 contentFrom = contentFrom.substring(0, contentFrom.length() - 1);  
 }  
 if (contentTo.startsWith("{")) {  
 contentTo = "," + contentTo.substring(1);  
 }  
 mergedContent = contentFrom + contentTo;  
 }  
  
  
 byte[] mergedBytes = mergedContent.getBytes();  
 Files.*write*(pathTo, mergedBytes);  
 Files.*write*(pathFrom, new byte[0]);  
  
 System.*out*.println("Данните бяха успешно преместени.");  
 } catch (IOException e) {  
 System.*out*.println("Грешка при преместването на информация.");  
 }  
}

Функцията приема като аргументи елементи Path pathFrom , който представлява локацията на първия файл , и Path pathTo , който представлява локацията на втория файл.

Първо се проверява дали и двата файла с дадени локации съществуват , и ако и двата или 1 от тях не съществува , се изписва на конзолата съобщението „JSON файлът с такъв път не е намерен.“ и се излиза от функцията. Ако съществуват , се създават 2 локални променливи String contentFrom , който взима са стойност съдържанието на първия файл и String contentТо , който взима са стойност съдържанието на втория файл. След това се създават още 2 локални променливи Boolean fromIsEmpty и Boolean toIsEmpty , които проверяват дали съдържанието на двата файла е празно.Създава се и още 1 локална променлива String mergedContent , коята взима стойност „“. Ако и двете съдържания са празни тогава mergedContent получава стойност „{}”, ако сасмо 1 е празен , тогава mergedContent получава като стойност съдържанието на файла , който не е празен. Ако и двете съдържания не са празни , тогава от contentFrom се маха знака ‘}’ в края на съдържаниете , а в contentTo се маха знака ‘{‘ в началото на съдържанието и след това mergedContent взима като стойност комбинация от двете съдържание contentFrom + contentTo. Създава се още 1 локална променлива byte[] mergedBytes , която представлява mergedContent във формат масив от байтове , за да може съдържанието да се впише във файл. След това във файла с локация pathTo се записва новото съдържание , а във файла с локация pathFrom се изтрива съдържанието на файла. По този начин успяхме да преместим съдържанието от един файл във друг. На края се изписва съобщение на конзолата „Данните бяха успешно преместени.“. При върнато изклчение се изписва съобщението на конзолата „Грешка при преместването на информацията.“.

validate функция :

public static void validate(Path filePath){  
 if (!Files.*exists*(filePath)) {  
 System.*out*.println("Файлът не съществува.");  
 return;  
 }  
  
 try {  
 String content = Files.*readString*(filePath).trim();  
 content = content.replaceAll("\\s+","");  
  
 if (content.isEmpty()) {  
 System.*out*.println("Файлът няма съдържание.");  
 return;  
 }  
  
  
 boolean isObject = content.startsWith("{") && content.endsWith("}");  
 boolean isArray = content.startsWith("[") && content.endsWith("]");  
  
 if (!(isObject || isArray)) {  
 System.*out*.println("Файлът трябва да започва и да завършва с еднакви знаци ('{' и '}' или '[' и ']')");  
 return;  
 }  
  
  
 int curly = 0;  
 int square = 0;  
 boolean inString = false;  
 //boolean expectingKey = false;  
 //boolean expectingColon = false;  
 //boolean expectingValue = false;  
 //boolean insideObject = false;  
  
 for (int i = 0; i < content.length(); i++) {  
 char c = content.charAt(i);  
  
 if (c == '"') {  
 inString = !inString;  
 }  
  
 if (!inString) {  
 if (c == '{') {  
 curly++;  
 //insideObject = true;  
 //expectingKey = true;  
 } else if (c == '}') {  
 curly--;  
 //insideObject = false;  
 } else if (c == '[') {  
 square++;  
 } else if (c == ']') {  
 square--;  
 }  
 }  
 }/\*else if (c == ':') {  
 if (!expectingColon) {  
 System.out.println("Невалидна употреба на ':' на знак номер " + i);  
 return ;  
 }  
 expectingColon = false;  
 expectingValue = true;  
 } else if (c == ',') {  
 if (expectingColon) {  
 System.out.println("Очаквана стойност преди ',' на знак номер " + i);  
 return ;  
 }  
 if (insideObject) {  
 expectingKey = true;  
 }  
 }  
 } else {  
 if (expectingKey) {  
 expectingKey = false;  
 expectingColon = true;  
 } else if (expectingValue) {  
 expectingValue = false;  
 }  
 }  
 }\*/  
 if (curly != 0 || square != 0 || inString) {  
 System.*out*.println("Несъответстващи скоби или незатворен стринг.");  
 return;  
 }  
 //content = content.replaceAll("\\s+","");  
 //System.out.println(content.charAt(1));  
 //System.out.println(content.charAt(content.length()-2));  
 if (content.charAt(1) == ',' || content.charAt(content.length() - 2) == ',') {  
 System.*out*.println("Файла не може да има ',' като втори или предпоследен знак.");  
 return;  
 }  
  
  
 } catch (IOException e) {  
 System.*out*.println("Грешка при четенето на файла.");  
 return ;  
 }  
  
 System.*out*.println("Файлът е във валиден JSON формат");  
}

Функцията приема като аргументи елементa Path filePath , който представлява локацията на файл.

Първо се проверява дали същестува файл с локация като filePath , и ако не съществува се се изписва съобщение на конзолата „Файлът не съществува.“ и се излиза от функцията. Създава се локална променлива String content , която взима като стойност съдържанието на файла с локация равна на filePath и премахва всички празни места от съдържанието чрез фунцкии trim и replaceAll . Ако съдържанието на content е празно се изписва съобщение на конзолата „Файлът няма съдържание.“ и се излиза от функцията. Създават се още 2 локални променливи Boolean isObject и Boolean isArray , който служат да определят дали съдържанието на JSON файлът е 1 елемент или масив от елементи спряма това дали започва и свъшва с ‘{‘ и ‘}’ или с ‘[‘ и ‘]’. Ако и двете променливи имат стойност false , тогава се изписва съобщение в конзолата „Файлът трябва да започва и да завършва с еднакви знаци (‘{‘ и ‘}’ или ‘[‘ и ‘]’)“ и се излиза от функцията. Създават се още 3 локални променливи : int curly и int square , които служат за следене на броя на къдрави и прави скоби в съдържанието и boolean inString , който е с начална стойност false и служи за това да се определя , кога част от съдържанието е вътре в String , за да се знае дали да се добавя към броя скоби или не. След това се преминава през for цикъл , който минава през цялото съдържание на JSON файлът по 1 знак след знак и се създава локална променлива в него char c, която взима стойност на знака на индекс i от съдържанието. Ако се стигне до знак със стойност ‚ „‘ , тогава inString променя стойността си на обратната на сегашната си стойност. Ако знакът не се намира в String : ако знакът е равен на ‘{‘ се довабя 1 към curly , а ако е равен на ‘}’ се маха 1 от curly , ако знакът е равен на ‘[‘ се добавя 1 към square , а ако знакът е равен на ‘]’ се маха 1 от square. Излиза се от for цикъла и се проверява дали curly или square са различни от 0 и дали for цикалът е приключил със inString равно на true , и поне 1 от тези е вярно се изписва на конзолата съобщението „Несъответсващи скоби или незатворен стринг.“ и се излиза от функцията. След това се проверява дали втория или предпоследният знак на съдържанието са равни на ‘,’ , и ако поне един от тях е , се изписва съобщението на конзолата „Файла не може да има ‘,’ като втори или предпоследен знак.” и се излиза от фунцкията . При върнато изключение се изписва съобщението на конзолата „Грешка при четенето на файла.“ и се излиза от функцията. Ако се стигне до края на функцията се изписва съобщението на конзолата „Файлът е във валиден JSON формат“.

help функция:

public static void help(){  
 System.*out*.println("Когато пишете името на файла се пише целия път към файла");  
 System.*out*.println("Пример : 'JSON\_Files/file.json' или 'FilesFolder2/file.json' .");  
 System.*out*.println("Open FileName -> Отваря файл с дадено име , не може да се отвори файл ако друг файл е вече отворен");  
 System.*out*.println("Close FileName -> Затваря отворен файл по име");  
 System.*out*.println("Print -> Принтира съдържанието на отвореният файл");  
 System.*out*.println("Search KeyName -> Търси и връща ключ и неговата стойност по дадено име от отвореният файл");  
 System.*out*.println("Set FileName NewContent -> Задава се ново съдържание на файл по дадено име на файл и по дадено съдържание");  
 System.*out*.println("Save FileName -> Запазват се промените направени във файл със Set функцията по дадено име на файл");  
 System.*out*.println("SaveAs NewPathName FileName -> Запазват се промените направени във файл със Set функцията в нова локация по пълно дадено име на нова локация и дадено име на файл");  
 System.*out*.println("(функцията също работи и за преместване на файлове в други папки , не е задължително да са били променени чрез Set функцията");  
 System.*out*.println("Create FileName NewContent -> Създава се нов файл с ново съдържание по дадено име на файл и по дадено съдържание");  
 System.*out*.println("Delete FileName -> Изтрива се файл по дадено име на файл");  
 System.*out*.println("Move FromFileName ToFileName -> Данните от първия файл даден по име се преместват във втория файл по дадено име");  
 System.*out*.println("Validate FileName -> Валидира се съдържанието на файл са дадно име , като се проверява дали има правилен JSON синтаксис");  
 System.*out*.println("Help -> Отваря менюто с обяснение на командите");  
 System.*out*.println("Exit -> Излиза от програмата");  
}

Функцията изписва на конзолата съобщения на различни редове , като всяко съобщение обяснява как да се работи с командите , които трябва да се извикат от конзолата в детайли. Тази функция не взима аргументи при извикването си.

main метод :

public static void main(String[] args) {  
 {  
 try {  
  
 List<PathReference> listPathRef = new ArrayList<>();  
 PathReference pathRef = new PathReference(null);  
  
  
 Scanner scanner = new Scanner(System.*in*);  
 BasicWriter basicWriter = new BasicWriter();  
 ContentExtendedWriter contentExtendedWriter = new ContentExtendedWriter();  
 LocationExtendedWriter locationExtendedWriter = new LocationExtendedWriter();  
 TwoFileWriter twoFileWriter = new TwoFileWriter();  
 FullWriter fullWriter = new FullWriter();  
  
 while (true) {  
 //System.out.print("Напишете команда --> : ");  
 fullWriter.FullWrite("Напишете команда --> : ");  
 String input = scanner.nextLine();  
 String [] commands = input.split(" ");  
 switch (commands[0]) {  
 case "Validate":  
 if(commands.length>1) {  
 *validate*(Paths.*get*("src\\" + commands[1]));  
 }  
 else{  
 //System.out.println("Напишете името на файла след 'Validate'");  
 basicWriter.Write("Validate");  
 }  
 break;  
 case "Open":  
 if(commands.length>1) {  
 *open*(pathRef, commands[1], listPathRef);  
 }  
 else{  
 //System.out.println("Напишете името на файла след 'Open'");  
 basicWriter.Write("Open");  
 }  
 break;  
 case "Close":  
 if(commands.length>1) {  
 *close*(pathRef, commands[1]);  
 }  
 else{  
 //System.out.println("Напишете името на файла след 'Close'");  
 basicWriter.Write("Close");  
 }  
 break;  
 case "Print":  
 *print*(pathRef);  
 break;  
 case "Search":  
 if(commands.length>1) {  
 *search*(pathRef,commands[1]);  
 }  
 else{  
 fullWriter.FullWrite("Напишете името на ключа след 'Search'");  
 }

break;  
 case "Set":  
 if(commands.length>2) {  
 String[] content = Arrays.*copyOfRange*(commands,2,commands.length);  
 String result = String.*join*(" ",content);  
 *set*(Paths.*get*("src\\" + commands[1]), result, listPathRef);  
 }  
 //Set invalid.json {"name" : "Pesho","age" : "47"}  
 else{  
 //System.out.println("Напишете името на файла и новото съдържание след 'Set'");  
 contentExtendedWriter.Write("Set");  
 }  
 break;  
 case "Save":  
 if(commands.length>1) {  
 *save*(commands[1],listPathRef);  
 }  
 else{  
 //System.out.println("Напишете името на файла след 'Save'");  
 basicWriter.Write("Save");  
 }  
 break;  
 case "SaveAs":  
 if(commands.length>2) {  
 *saveAs*(commands[2],commands[1],listPathRef);  
 }  
 else{  
 //System.out.println("Напишете новата локация и името на файла след 'SaveAs'");  
 locationExtendedWriter.Write("SaveAs");  
 }  
 break;  
 case "Create":  
 if(commands.length>2) {  
 String[] content = Arrays.*copyOfRange*(commands,2,commands.length);  
 String result = String.*join*(" ",content);  
 *create*(Paths.*get*("src\\" + commands[1]), result);  
 }  
 //Create new.json {"name" : "Pesho","age" : "47"}  
 else{  
 //System.out.println("Напишете името на файла и новото съдържание след 'Create'");  
 contentExtendedWriter.Write("Create");  
 }  
 break;  
 case "Delete":  
 if(commands.length>1) {  
 *delete*(Paths.*get*("src\\" + commands[1]),listPathRef,true);  
 }  
 else{  
 //System.out.println("Напишете името на файла след 'Delete'");  
 basicWriter.Write("Delete");  
 }  
 break;  
 case "Move":  
 if(commands.length>2) {  
 *move*(Paths.*get*("src\\" + commands[1]),Paths.*get*("src\\" + commands[2]));  
 }  
 else{  
 //System.out.println("Напишете името на първия и втория файл след 'Move'");  
 twoFileWriter.Write("Move");  
 }  
 //Move JSON\_FILES\blah.json JSON\_FILES\new.json  
 break;  
 case "Help":  
 *help*();  
 break;  
 case "Exit":  
 //System.out.println("Програмата беше затворена");  
 fullWriter.FullWrite("Програмата беше затворена");  
 scanner.close();  
 return; // Exit the program  
 default:  
 //System.out.println("Невалидна опция. Опитайте отново.");  
 fullWriter.FullWrite("Невалидна опция. Опитайте отново.");  
 }  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

main метода започва като се създават 2 променливи List<PathReference> listPathRef = new ArrayList<>() , в който ще се съхраняват елементите от тип PathReference , докато програмата работи и PathReference pathRef = new PathReference(null) , която се инициализира с празна локация и ще служи за съхраняване на данните на отвореният файл в програмата (може да има само 1 отворен файл , докато върви програмата).

След това се създава елемент Scanner scanner , който служи за четене на команди от конзолата. Инициализират се и 5 елемента , които са 5те наследниак на BaseWriter класа : BasicWriter basicWriter , ContentExtendedWriter contentExtendedWriter , LocationExtendedWriter locationExtendedWriter , TwoFileWriter twoFileWriter и FullWriter fullWriter.

Влиза се в while(true) цикъл , от който може да се излезе само ако се напише командата “Exit” в конзолата. Изписва се съобщение на конзолата чрез fullWriter „Напишете команда 🡪: „.

Това изречение се изписва всеки път , когато while цикъла се повтаря. Създават се локални за цикъла променливи String input , който взема написаното на конзолата чрез scanner елемента. Другата променлива е String[] commands , която е масив от String и служи за разделяне на написания ред в конзолата на думи , като думите се разделят по „ „.

Влиза се в switch цикъл , който взима като аргумент първия елемент на commands или още казано първата дума в изречението. В случай , че commands[0] е равен на „Validate” се влиза в нов case и се проверява дали съществува втори елемент в commands и ако съществува се извика validate функцията с аргумент Paths.get(“src\\” + commands[1]) , който представлява локацията на файл с наименование на локацията равен на вторатия елемент в commands. В случай , че няма втори елемент в масива , се извиква Write функцията на basicWriter с аргумент „Validate”. По този начин се валидира файл , дали е в правилен JSON формат , по дадена локация на файл.

В случай , че commands[0] е равен на „Open” се влиза в нов case и се проверява дали съществува втори елемент в commands , който представлява наименованието на локацията на файл , и ако съществува се извика open функцията с аргументи : pathref , commands[1] и listPathRef. В случай , че няма втори елемент в масива , се извиква Write функцията на basicWriter с аргумент „Open”. Така се „отваря“ файл (данните му се записват в програмата) по дадена локация на файл. Може да има само 1 отворен файл , докато работи програмата. Трябва първо да се затвори отворения файл , за да се отвори нов.

В случай , че commands[0] е равен на „Close” се влиза в нов case и се проверява дали съществува втори елемент в commands , който представлява наименованието на локацията на файл , и ако съществува се извика close функцията с аргументи : pathref и commands[1]. В случай , че няма втори елемент в масива , се извиква Write функцията на basicWriter с аргумент „Close”. Така се затваря вече отворен файл по дадена локация на файл.

В случай , че commands[0] е равен на „Print” се влиза в нов case и се извиква функцията print с аргуметн pathRef . По този начин се изписва на конзолата съдържанието на отвореният в програмата файл.

В случай , че commands[0] е равен на „Search” се влиза в нов case и се проверява дали съществува втори елемент в commands , който представлява наименованието на ключа в отворения файл, и ако съществува се извика search функцията с аргументи : pathref и commands[1] . В случай , че няма втори елемент в масива , се извиква FullWrite функцията на fullWriter с аргумент „Напишете името на ключа селд ‚Search’”. По такъв начин се извежда на конзолата двойка ключ и стойност от отворения файл по дадено наименование на ключа.

В случай , че commands[0] е равен на „Set” се влиза в нов case и се проверява дали съществува втори елемент в commands , които представлява наименованието на локацията на файл и дали съществуват останалите елементи (поне 3 елемента в commands) , които представляват новото съдържание на файла , и ако съществуват се създават нови локални променливи String[] content , която взима стойностите на всички елементи след 2рия елемент и String result , която комбинира всички елементи от content в 1 String като ги разделя с „ “. След това се изкива set функцията с аргументи : Paths.get(“src\\” + commands[1]) , result и listPathRef. В случай , че в commands има по-малко от 3 елемента , се извиква Write функцията на contextExtendedWriter с аргумент „Set”. По този начин се задава ново съдържание на файл по дадено име на локация на файл и по дадено ново съдържание.

В случай , че commands[0] е равен на „Save” се влиза в нов case и се проверява дали съществува втори елемент в commands , който представлява наименованието на локацията на файл , и ако съществува се извика save функцията с аргументи : commands[1] и listPathRef. В случай , че няма втори елемент в масива , се извиква Write функцията на basicWriter с аргумент „Save”. Така се записват в самия файл обновените данни от програмата по дадена локация на файл.

В случай , че commands[0] е равен на „SaveAs” се влиза в нов case и се проверява дали съществува втория елемент и третия елемент в commands , които представляват наименованието на оригиналната локацията на файл и наименованието на новата локация на файл. Ако съществуват се извика saveAs функцията с аргументи : commands[2], commands[1] и listPathRef. В случай , че има по-малко от 3 елемента в масива , се извиква Write функцията на locationExtendedWriter с аргумент „SaveAs”. Така се записват обновените данните от програмата (или старите данни) на файл във нова локация , като се премества целия файл.

В случай , че commands[0] е равен на „Create” се влиза в нов case и се проверява дали съществува втори елемент в commands , които представлява наименованието на локацията на нов файл и дали съществуват останалите елементи (поне 3 елемента в commands) , които представляват новото съдържание на файла , и ако съществуват се създават нови локални променливи String[] content , която взима стойностите на всички елементи след 2рия елемент и String result , която комбинира всички елементи от content в 1 String като ги разделя с „ “. След това се изкива create функцията с аргументи : Paths.get(“src\\” + commands[1]) и result. В случай , че в commands има по-малко от 3 елемента , се извиква Write функцията на contextExtendedWriter с аргумент „Create”. По този начин се създава нов файл в проекта по дадено наименование на нова локация и дадено съдържание.

В случай , че commands[0] е равен на „Delete” се влиза в нов case и се проверява дали съществува втори елемент в commands , който представлява наименованието на локацията на файл , и ако съществува се извика delete функцията с аргументи : Paths.get(“src\\” + commands[1]), listPathRef и true(което означава , че ще се изпише съобщение). В случай , че няма втори елемент в масива , се извиква Write функцията на basicWriter с аргумент „Delete”. По този начин се изтрива съществуващ файл в проекта по дадено наименование на локацията на файла.

В случай , че commands[0] е равен на „Move” се влиза в нов case и се проверява дали съществува втория елемент и третия елемент в commands , които представляват наименованието на локацията на първи файл и наименованието на локацията на втори файл. Ако съществуват се извика move функцията с аргументи : Paths.get(“src\\” + commands[1]) и Paths.get(“src\\” + commands[2]). В случай , че има по-малко от 3 елемента в масива , се извиква Write функцията на twoFileWriter с аргумент „Move”. Така се премества съдържанието от първи файл във втори файл по дадени наименования на локации.

В случай , че commands[0] е равен на „Help” се влиза в нов case и се извиква help функцията , която изписва всичките подробности за командите , които могат да се пишат в конзолата.

В случай , че commands[0] е равен на „Exit” се влиза в нов case и се извиква FullWrite функцията на fullWriter с аргумент „Програмата беше затворена“. След това се затваря scanner елемента с функция scanner.close() и на края се излиза от while цикъла и съответствено от прогамата с return.

Default case случая за commands[0] извиква FullWrite функцията на fullWriter с аргумент „Невалидна опция. Опитайте отново.“.