Документация
Тема 6: JSON парсер

Глава 1. Увод

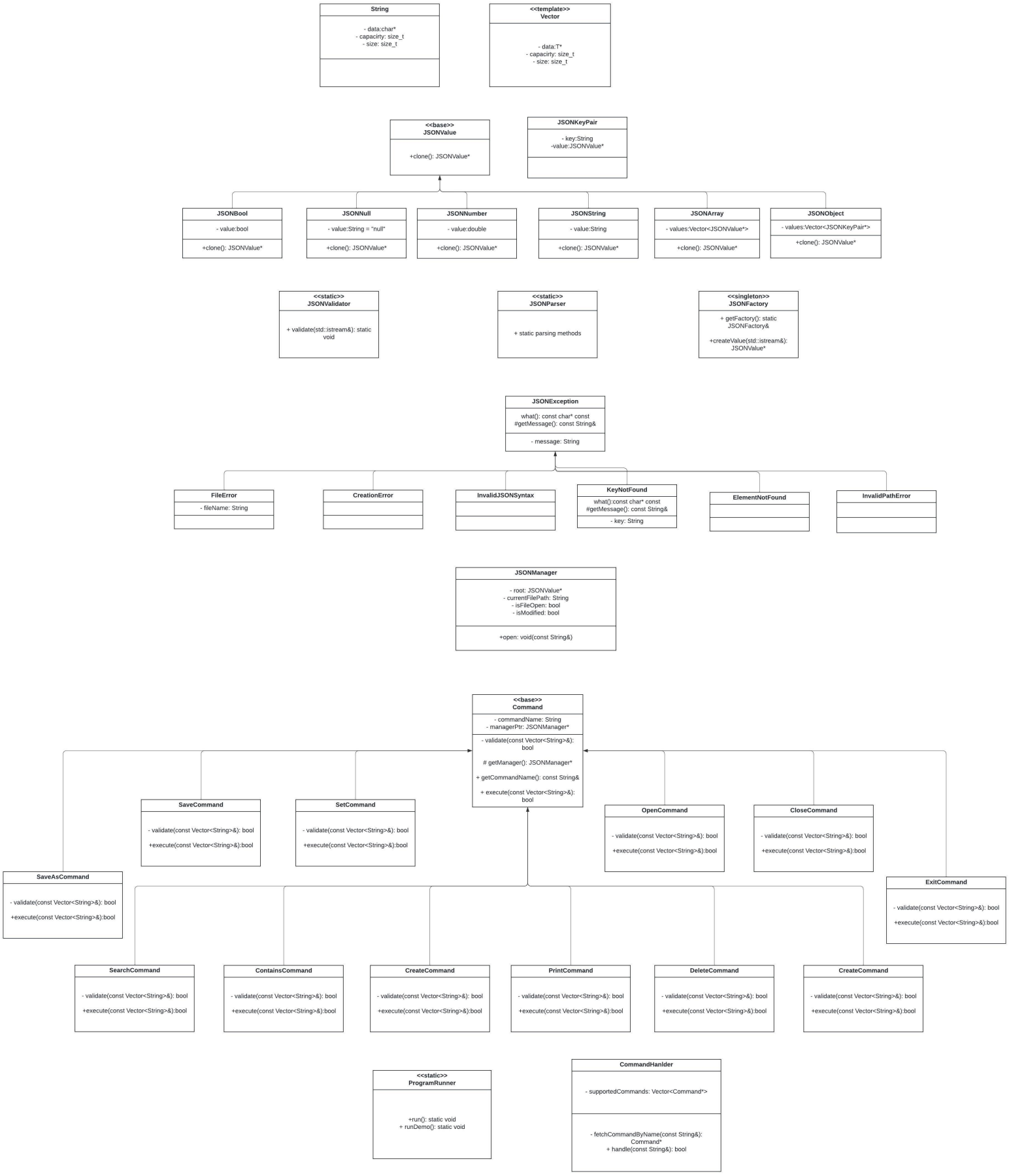
Този проект цели да създаде конзолно приложение, което да работи с файлове в JSON(*Java Script Object Notation*) формат. При подаване на валиден JSON файл, програмата трябва да прочете и съхрани съдържанието му, както и да изпълни набор от операции върху извлечените данни. Задачите за разработка, които възникват включват валидиране на JSON файла, съхранение на извлечените данни в подходяща структура,
прилагане на необходимите команди за тяхната манипулация и разработване на команден интерфейс за работа с тях.

Глава 2. Преглед на предметната област

Проектът се стреми да се придържа към принципите на обектно-ориентираното програмиране, а именно абстракция, енкапсулация, наследяване и полиморфизъм.
С оглед на поставената задача и изисквания възникват следните предизвикателства:

- Ефективно валидиране на JSON файл
- Правилно парсиране и създаване на различни JSON типове
- Оптимално съхранение и управление на парсираните данни
- Реализация на различни операции за манипулация и работа с данните

Глава 3. Проектиране



Фигура 1. Диаграма с класовете на проекта

Използвани са два класа, писани по време на практикум и семинар - Vector и String. Те са съответно мой версии на std::vector и std::string. Клас InputStringStream, наследяващ std::istringstream за третиране на низ от клас String като поток за четене. Архитектурата се базира на йерархията на формата JSON. Започва с базов клас JSONValue, който представлява типовете JSON стойности - булева стойност, *null*, число, низ, масив и обект.
Той бива наследен от класовете, представляващи типовете описани на горния ред.
Имплементиран е клас JSONKeyPair, представляващ ключ-стойност двойките в JSON обект.
За валидирането на файлове отговаря статичния клас JSONValidator, който при среща на невалиден формат хвърля изключение. Други помощни класове са статичния клас JSONParser отговарящ за парсирането и фабриката JSONFactory, чрез която се създават различните стойности. Имплементиран е клас JSONException, наследяващ std::exception, и неговите наследници за по-специфични изключения, които да сигнализират за възможните грешки, които могат да възникнат при работа с JSON файл в рамките на проекта.
Зад главната функционалност и манипулацията на JSON файлове стои класът JSONManager. Чрез него се отваря, запазва, затваря JSON файл и се работи с данните прочетени от него.
Имплементирана е йерархия от команди наследници на базовия клас Command и клас отговорен за тяхното извикване - CommandHandler. Апликацията стартира, чрез статичния метод от клас ProgramRunner, който съдържа главния цикъл за командния интерфейс.

Най-важните извадки от кода са следните:
- Главната логика за валидиране на различните типове JSON стойности

```
void JSONValidator::validateValue(std::istream& inputStream) {
    char firstChar;
    inputStream >> firstChar;
    inputStream.unget();

    if (firstChar == '-' || helpers::isDigit(character: firstChar)) {
        validateNumber(inputStream);

    } else if (firstChar == 'n') {
        validateNull(inputStream);

    } else if (firstChar == 't' || firstChar == 'f') {
        validateBool(inputStream);

    } else if (firstChar == constants::STRING_OPENING_QUOTE) {
        validateString(inputStream);

    } else if (firstChar == constants::ARRAY_OPENING_BRACKET) {
        validateArray(inputStream);

    } else if (firstChar == constants::OBJECT_OPENING_BRACKET) {
        validateObject(iinputStreams: inputStream);

    } else {
        throw InvalidJSONSyntax(msg: "Invalid value in validator");
    }
}
```

Фигура 2. Метод за валидация на стойност

Методът определя типът по първия прочетен символ и извиква съответния метод за валидиране на този тип. Ако бъде извикан методът за валидиране на обект или масив, validateValue се извиква отново в него за да валидира стойностите в обекта или масива, ако не са празни.

- Създаването на различните JSON стойности от фабриката

```
JSONValue* JSONFactory::createValue(std::istream& inputStream) {
    char firstChar;
    inputStream >> firstChar;

    inputStream.unget();

    if (firstChar == '-' || helpers::isDigit(character: firstChar)) {
        return createNumber(inputStream);

    } else if (firstChar == 'n') {
        return createNull(inputStream);

    } else if (firstChar == 't' || firstChar == 'f') {
        return createBool(inputStream);

    } else if (firstChar == constants::STRING_OPENING_QUOTE) {
        return createString(inputStream);

    } else if (firstChar == constants::ARRAY_OPENING_BRACKET) {
        return createArray(inputStream);

    } else if (firstChar == constants::OBJECT_OPENING_BRACKET) {
        return createObject(inputStream);

    } else {
        throw CreationError(msg: "Invalid character in factory");
    }
}
```

Фигура 3. Метод за създаване на стойност

Сходен с методът за проверка на тип при валидирането, този метод следи за първия извлечен символ и го праща в съответния метод за създаване на стойност.

- Отваряне и парсиране на валиден JSON файл

```
void JSONManager::open(const String& filePath) {
    if(isFileOpen) {
        throw FileError(msg: "JSON file already opened", fileName: currentFilePath);
    }

    std::ifstream file(s: filePath.C_str());
    if (!file.is_open()) {
        throw FileError(msg: "Couldn't open. Check the file path", fileName: filePath);

    } else if (!file) {
        throw FileError(msg: "File error", fileName: filePath);
    }

    JSONValidator::validate(inputStream: file);

    file.clear();
    file.seekg(pos: 0);

    char character;
    bool isEmpty = true;
    while (file.get(c: character)) {
        if (!std::isspace(c: character)) {
            isEmpty = false;
            break;
        }
    }

    if (isEmpty) {
        root = nullptr;
        std::cout << "Empty file detected." << '\n';

    } else {
        file.clear();
        file.seekg(pos: 0);
        parseJSON(is: file);
    }

    currentFilePath = filePath;
    isFileOpen = true;
    isModified = false;
}
```

Фигура 4. Метод за отваряне на JSON файл

Логиката зад отварянето на файла е следната - първо проверяваме за вече отворен файл и ако няма такъв го отваряме по подадения път. Извършва се валидация, след което се връщаме в началото на файла и правим проверка дали е празен. Ако файлът е празен, съобщаваме за това и излизаме от метода. Ако не е празен се връщаме отново в началото на файла и го парсираме.

Глава 4. Реализация, тестване

Поддържаните команди и тяхната функционалност:

- *open <path/to/file>* - отваряне на JSON файл по подаден път. Ако има отворен такъв или е подаден грешен път се съобщава за това.

- *validate <path/to/file>* - валидиране на JSON файл по подаден път. Ако е подаден грешен път или файлът е невалиден се извежда подходящо съобщение.

- *close* - затваряне на текущия отворен файл, ако има такъв. Ако има незапазени промени се съобщава за това и се дава възможност да се запазят или не.

- *save* - запазване на промените по текущия файл, ако има такива.

- *saveas <path/to/file>* - запазва се цялото съдържание на текущия файл, в друг по указан път.

- *exit* - изход от апликацията, като ако има незапазени промени по текущия файл, се съобщава и се дава възможност за запазване или не. Ако файлът не е затворен при извикване на командата, то той се затваря автоматично и се извежда съобщение за неговото затваряне.

- *create <"path"/"for"/"creation"> <value>* - създаване на стойност чрез указан път и валидна стойност. Ако стойността е невалидна се извежда съобщение.

- *set <"path"/"to"/"value"> <value>* - промяна на вече съществуваща стойност по указан път към нея и съответна стойност с която да я сменим. Ако пътят към стойността е грешен или не съществува се извежда подходящо съобщение, както и ако въведената стойност е невалидна.

- *search <"key">* - търсене на всички стойности, които съответстват на въведения ключ, и ако има такива се извеждат на екрана, а ако няма се съобщава за това. Ако ключът не съществува се извежда подходящо съобщение.

- *print* - отпечатва съдържанието на текущия файл на екрана. Ако файлът е празен или няма отворен такъв се извежда подходящо съобщение.

- *delete <"path"/"to"/"value">* - премахва стойността съответстваща на указания път. Ако въведения път е грешен или не съществува се извежда подходящо съобщение.

- *contains <value>* - проверява дали дадената стойност се съдържа някъде в съдържанието на файла и се извежда съобщени за истина или лъжа.

Тук са следните улеснения, които са използвани при търсене и създаване на стойности и спецификите свързани с работата с тях чрез терминала.

- При парсиране на файл JSONParser и JSONFactory не и извършват подробна валидация, защото се очаква че подадения файл е минал вече през JSONValidator и е валиден.

- Празен JSON файл го считам за валиден, тъй като като го отворим можем да създаваме стойности в него чрез командата *create.* Ако е празен и сме извикали *create* се създава нов JSON обект.

- При оказване на път към стойност или при търсене на ключ в JSON обект, пътят се подава по ключове, разделени с наклонена черта, като всеки ключ е ограден с кавички. Ако търсим само ключ въвеждаме го с кавички. Примерно:
  > search "key"
  > set "key1"/"key2"/"key3" value

- При създаване или променяне на JSON стойност чрез терминала след като бъде указан пътя, по споменатия горе начин, следва интервал и текстът, съответстващ на стойността, която искаме да създадем. Улеснението при създаване чрез текст в терминала е, че не се подържат в JSON текста интервали и нови редове, а трябва да се въведе всичко слято.
  Примерно:
  > За низ: <path> *"MihailDechev"*, а не *"Mihail Dechev"*
  > За масив: *<path> [1,true,"string",null]*
  > За обект: *<path> {"key1":value1,"key2":value2}*

Разработени са няколко файла за тестване на главните функционалности на проекта с различни тестове сценарии:

- testValidation.cpp - включва тестови сценарии за валидиране на всеки тип JSON стойност, както и главна функция, която да тества валидация на вложен JSON текст. Валидирането в тези сценарии се извършва чрез подаване на JSON текст, който чрез InputStringStream се третира като поток за четене и се подава като параметър в статичния метод за валидиране. Следи се за хвърлено изключение при невалиден JSON, което да се хване и изведе на екрана. Има и тестов сценарии, който вместо текст се подава име на файл, който да бъде валидиран.

- testParser.cpp - включва сценарии за парсиране на всеки тип JSON стойност. Подава се JSON текст като параметър, третиран чрез InputStringStream като поток за четене и се подава в съответния метод за парсиране на желаната стойност. Тук се очаква да се подават само валидни стойности, защото JSONParser не изпълнява ролята на валидатор.

- testFactory.cpp - включва сценарии за създаване на всеки тип JSON стойност чрез подаден JSON текст и сценарии за създаване на стойност от файл чрез подаден път към него. Отново тук се очакват валидни стойности, но тъй като главния метод за създаване хвърля изключение при неразпознат отварящ символ ще следим за хвърлено такова и за съобщаване на грешката.

- testManager.cpp - включва тестови сценарии за всеки един от методите на JSONManager, като се следи за хвърлени изключения и извеждане на грешката на екрана.

Пример за работа с апликацията:
> *open ../JSONfiles/organisation.json*

> *print*

> *search “name”*

> *set “management”/“directorId” 2*

> *search “management”*

> *save*

> *close*

> *exit*

Глава 5. Заключение

Проектът успешно реализира основните изисквания за работа с файлове в JSON формат, включително, валидация, парсиране, съхранение на данните и манипулация. Разработеният команден интерфейс предоставя удобен начин за работа с програмата и обработка на JSON данните. Вероятно може да се постигне подобрение в архитектурата. В бъдеще може да се имплементира графичен интерфейс за по-удобна работа.

Използвана литература

- "Introducing JSON." JSON, www.json.org/json-en.html. Accessed 31 Aug. 2024. Информация за формата JSON.

- "Design Patterns." Refactoring.Guru, refactoring.guru/design-patterns. Accessed 31 Aug. 2024. Страница с информация за различните видове design patterns.

- "C and C++ Reference." Cppreference.Com, en.cppreference.com/w/. Accessed 31 Aug. 2024. Документация на C++.

- "Claude AI." Anthropic, claude.ai/new. Accessed 31 Aug. 2024. Изкуствен интелект, помогнал с имплементацията на някои методи.

- "Lucidchart." Lucid Software Inc., www.lucidchart.com/. Accessed 31 Aug. 2024. Софтуер за изготвянето на диаграмата с класовете на проекта.

- "JSON Formatter & Validator." Curious Concept, jsonformatter.curiousconcept.com/. Accessed 31 Aug. 2024. Валидатор, използван за проверка на JSON текст по време на разработка и тестване.

- "Citation Machine® - Write Smarter." Citation Machine, www.citationmachine.net/. Accessed 31 Aug. 2024. Софтуер за цитиране на литература съгласно MLA style.

- Материали от лекции, практикуми и семинари.

Изготвил: Михаил Дечков Дечев