Bartłomiej Przewoźniak, grupa 8ISI

**Techniki kompilacji, projekt własny**

Wstępna dokumentacja projektu

**I. Funkcjonalność:**

Wymagania funkcjonalne:

Program będzie generował pliki klas Javowych dla istniejącego opisu struktur danych zapisanego w schemacie AVRO w notacji JSON realizując następujące funkcje:

* czytanie pliku wejściowego znak po znaku,
* tworzenie tokenów zgodnych z notacją Json,
* sprawdzenie poprawności semantycznej pliku

(z uwzględnieniem reguł Json i Avro),

* budowanie drzewa obiektów typu JsonNode,
* generowanie pliku wyjściowego – klasy Java.

Wymagania niefunkcjonalne:

Tekst pliku wejściowego powinien być kodowany za pomocą UTF-8.

**II. Sposób uruchomienia, wejścia/wyjścia:**

Rozwiązaniem zadania będzie program z interfejsem wiersza poleceń napisany w języku Java. Po uruchomieniu mamy wybór scenariusza: główna funkcjonalność programu, testy modułowe lub testy całościowe. Podstawowa funkcjonalność programu przyjmuje na wejściu jeden plik tekstowy w notacji JSON, zaś na wyjściu zwraca wygenerowany plik zawierający opis klasy w języku Java.

**III. Gramatyka notacji JSON:**

Notacja JSON jest prostym formatem wymiany danych.

* **Typy proste:**

|  |  |
| --- | --- |
| **Nazwa** | **Wartość** |
| LEFT\_BRACE | { |
| RIGHT\_BRACE | } |
| LEFT\_BRACKET | [ |
| RIGHT\_BRACKET | ] |
| COLON | : |
| COMMA | , |
| STRING | ”chars” |
| NUMBER | 0 | [1-9][0-9]\*(.[0-9]+) |
| LITERAL | true | false | null |

* **Typy złożone:**

JSON powstał w oparciu o dwie struktury: obiekty oraz tablice.

Obiekt jest nieuporządkowanym zbiorem par nazwa/wartość. Opis obiektu rozpoczyna **{** (lewa klamra) a kończy **}** (prawa klamra). Po każdej nazwie następuje **:** (dwukropek) oraz pary nazwa/wartość, oddzielone **,** (przecinkiem).

Tablica jest uporządkowanym zbiorem wartości. Opis tabeli rozpoczyna znak **[** (lewy nawias kwadratowy) a kończy znak **]** (prawy nawias kwadratowy). Poszczególne wartości rozdzielane są znakiem **,** (przecinek).

* **Składnia:**

**object:**

{ }

{ members }

**members:**

pair

pair, members

**pair:**

string : value

**array:**

[ ]

[ elements ]

**elements:**

value

value, elements

Wartość to łańcuch znakowy, którego początek i koniec oznacza podwójny cudzysłów, lub liczba, lub wartość **true** (prawda) lub **false** (fałsz) lub **null**. Wartością może być również obiekt lub tabela. Struktury te można zagnieżdżać.

**value:**

string

number

object

array

„true”

„false”

„null”

Łańcuch znakowy jest zbiorem zera lub większej ilości znaków Unicode, opakowanym w podwójne cudzysłowy. Pojedynczy znak jest reprezentowany jako łańcuch jednoznakowy. Łańcuch znakowy JSON jest podobny do łańcucha znakowego Java.

Liczby zapisywane w formacie JSON są bardzo podobne do liczb w języku Java, poza tym wyjątkiem, że nie używa się formatów ósemkowych i szesnastkowych.

**string:**

„ ”

„ chars ”

**chars:**

char

char chars

**char:**

any Unicode character except **"** or **\** or control character

Wolne miejsce (spacje, znaki tabulatora, itp.) można wstawić między dowolną parę składowych. Poza kilkoma detalami dotyczącymi kodowania, na tym kończy się opis języka JSON.

**IV. Gramatyka schematu AVRO:**

Schemat AVRO powstał w oparciu o notację JSON.

Dla typów prostych zachodzi następujące mapowanie wartości:

|  |  |  |
| --- | --- | --- |
| **AVRO type** | **JSON type** | **Przykład** |
| null | null | null |
| boolean | boolean | true |
| int,long | integer | 1 |
| float,double | number | 1.1 |
| bytes | string | "\u00FF" |
| string | string | "foo" |
| record | object | {"a": 1} |
| enum | string | "FOO" |
| array | array | [1] |
| map | object | {"a": 1} |
| fixed | string | "\u00ff" |

* **Typy proste:**

|  |  |
| --- | --- |
| **Nazwa** | **Wartość** |
| null | no value |
| boolean | a binary value |
| int | 32-bit signed integer |
| long | 64-bit signed integer |
| float | single precision (32-bit) IEEE 754 floating-point number |
| double | double precision (64-bit) IEEE 754 floating-point number |
| bytes | sequence of 8-bit unsigned bytes |
| string | unicode character sequence |

* **Typy złożone:**

Schemat AVRO wspiera sześć typów złożonych: record, enum, array, map, union oraz fixed.

* **Składnia:**

Definicje typów złożonych prezentują się w następujący sposób:

**enum:**

{

"type": "enum",

"name": "Suit",

"symbols" : ["SPADES", "HEARTS", "DIAMONDS", "CLUBS"]

}

**array:**

{

"type": "array",

"items": "string"

}

**map:**

{

"type": "map",

"values": "long"

}

**union:**

{

"type": "array",

"items": ["null", "string"]

}

**fixed:**

{

"type": "fixed",

"size": 16,

"name": "md5"

}

**V. Mapowanie obiektów:**

* Typy proste:

|  |  |
| --- | --- |
| **Schemat AVRO:** | **Klasa JAVA:** |
| {      **"name"**:"name",    **"type"**:"string" } | private String name;  public String getName() {  return name; }  public void setName(String name) {  this.name = name; } |

* Typy złożone:

|  |  |
| --- | --- |
| **Schemat AVRO:** | **Klasa JAVA:** |
| {      **"name"**:"array",    **"type"**:"string",    **"items"**:"string" } | private String[]ingredients;  public String[] getIngredients() {  return ingredients; }  public void setIngredients(String[] ingredients) {  this.ingredients = ingredients;  } |
| {      **"type"**:"record",    **"name"**:"Ingredient",    **"fields"**:[         {            **"name"**:"name",          **"type"**:"string"       },       {            **"name"**:"sugar",          **"type"**:"double"       },       {            **"name"**:"fat",          **"type"**:"double"       }    ] } | class Ingredient {  private String name;  private double sugar;  private double fat;  public Ingredient() {}  public Ingredient(String name, double sugar, double fat) {  this.name = name;  this.sugar = sugar;  this.fat = fat;  }  public String getName() {  return name;  }  public void setName(String name) {  this.name = name;  }  public double getSugar() {  return sugar;  }  public void setSugar(double sugar) {  this.sugar = sugar;  }  public double getFat() {  return fat;  }  public void setFat(double fat) {  this.fat = fat;  }  } |

**VI. Wykorzystywane struktury:**

enum TokenType {

LEFT\_BRACE,

RIGHT\_BRACE,

LEFT\_BRACKET,

RIGHT\_BRACKET,

COMMA,

COLON,

STRING,

NUMBER,

LITERAL,

ERROR,

EOF

}

class Token

{

private int row;

private int column;

private int hashValue;

private String tokenValue;

private TokenType tokenType;

}

enum NodeType

{

ROOT,

OBJECT,

ARRAY,

PAIR,

NAME,

VALUE,

ERROR,

EOF

}

class Node

{

string name;

string value;

Node parent;

Node firstChild;

Node nextSibling;

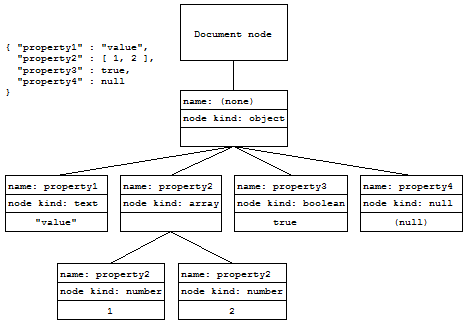
NodeType nodeType;

}

**VII. Moduły:**

1. Analizator leksykalny:
   * pobiera po kolei znaki z pliku tekstowego tworząc z nich token zgodny z gramatyką i zwraca go jako wynik przekazany do parsera.
2. Analizator składniowy wraz z akcjami semantycznymi:
   * porozumiewa się z analizatorem leksykalnym, który przesyła kolejne tokeny. Na ich podstawie tworzy drzewa składające się z obiektów JsonNode.
   * sprawdza poprawność występujących pól z warunkami schematu AVRO: słowa kluczowe, typy prymitywne i złożone.
3. Generator pliku wyjściowego:
   * przekształca powstałą strukturę na plik tekstowy zawierający opis klasy w języku Java.

**VIII. Drzewo dokumentu:**



**IX. Obsługa sytuacji wyjątkowych:**

Błędy w pliku wejściowym powodują wyświetlenie komunikatu o błędzie i przerwanie działania programu. W komunikacie o błędzie wypisuje jaki niepoprawny token otrzymaliśmy oraz miejsce w pliku (numer wiersza i numer kolumny). Przewidywane przypadki:

* InvalidFileException – błąd obsługi pliku,
* InvalidTokenException – konstrukcje niepoprawne leksykalnie,
* UnexpectedTokenException – konstrukcje niepoprawne semantycznie,
* InvalidAvroException – konstrukcje niespełniające warunków schematu AVRO,

**X. Przykładowy plik:**

* **Schemat AVRO w notacji JSON:**

{

"type": "record",

"name": "Movie",

"namespace": "org.kitesdk.examples.data",

"fields": [

{

"name": "id",

"type": "int"

},

{

"name": "title",

"type": "string"

},

{

"name": "releaseDate",

"type": "string"

},

{

"name": "imdbUrl",

"type": "string"

}

]

}

* **Klasa w języku Java:**

package org.kitesdk.examples.data;

public class Movie {

private int id;

private String title;

private String releaseDate;

private String imdbUrl;

public Movie() {}

public Movie(int id, String title, String releaseDate, String imdbUrl) {

this.id = id;

this.title = title;

this.releaseDate = releaseDate;

this.imdbUrl = imdbUrl;

}

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public String getReleaseDate() {

return releaseDate;

}

public void setReleaseDate(String releaseDate) {

this.releaseDate = releaseDate;

}

public String getImdbUrl() {

return imdbUrl;

}

public void setImdbUrl(String imdbUrl) {

this.imdbUrl = imdbUrl;

}

}

**XI. Literatura:**

<http://www.ietf.org/rfc/rfc4627.txt> - opis notacji JSON.

<https://avro.apache.org/docs/current/spec.html> - opis schematu AVRO.