**POLITECHNIKA ŚWIĘTOKRZYSKA**
**Wydział Elektrotechniki, Automatyki i Informatyki**

# Technologie Obiektowe

**Projekt:** Generacja schematu dla dokumentów JSON

| Autor: **Bartosz Kelner** | Grupa: **1ID21A** |
| --- | --- |

# 1. Cel projektu i wstęp teoretyczny

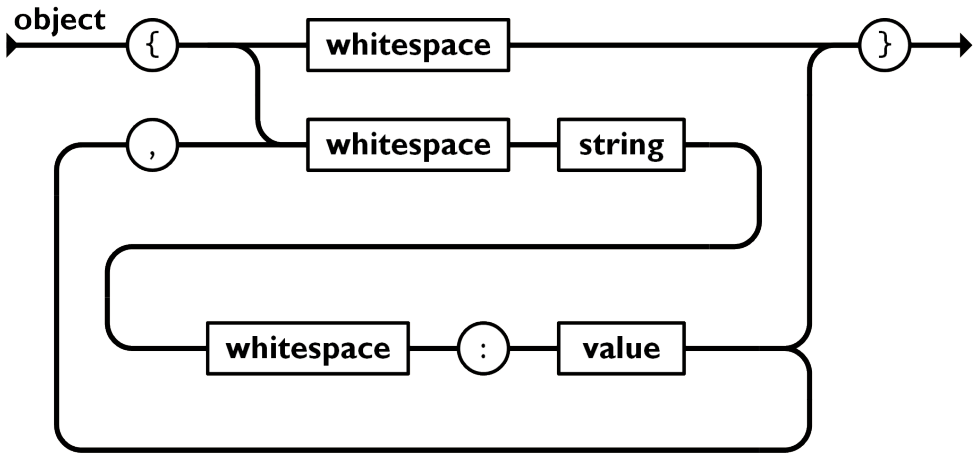
Celem projektu było opracowanie narzędzia, które generuje schematy dla podawanych plików JSON. Dokumenty JSON powinny być zgodne ze specyfikacją formatu JSON, natomiast schematy ze specyfikacją JSON Schema.
Narzędzie opracowano jako bibliotekę języka Java oraz jako program z graficznym interfejsem użytkownika (z użyciem biblioteki JavaFX).
Poza podstawową funkcjonalnością generowania schematów, narzędzie posiada funkcję walidacji dokumentu JSON na podstawie podanego schematów.

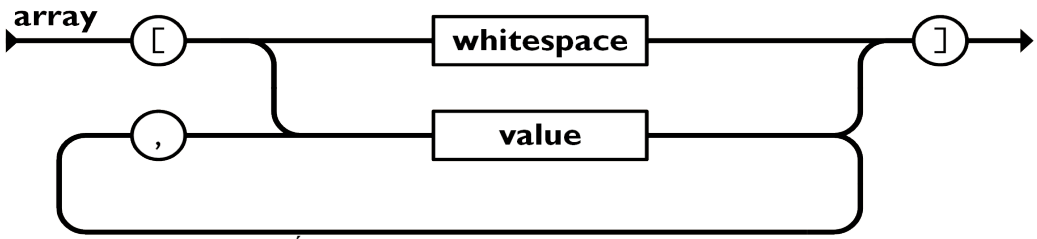
## 1.1 Format JSON (JavaScript Object Notation)

JSON to format danych zaprojektowany z myślą o jego czytelności dla człowieka oraz łatwym przetwarzaniu przez komputery. Bazuje na sposobie zapisu obiektów w języku JavaScript, ale może być przetwarzany przez programy napisane w dowolnym języku.

W tym formacie występują dwie główne struktury:

- Kolekcja par klucz/wartość, która w językach programowania może być realizowana jako np. obiekt, słownik czy tablica asocjacyjna,
- Uporządkowana lista wartości – w języka programowania występuje jako tablica lub lista.

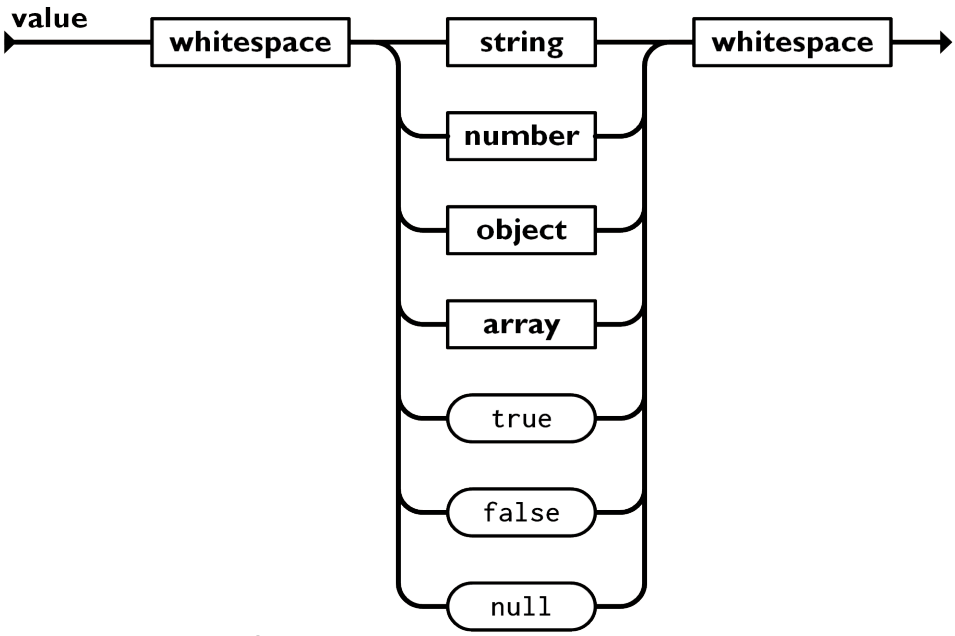



Źródło: https://www.json.org/json-en.html

Ponadto specyfikacja określa 4 typy prymitywne:

- string - ciąg znaków zaczynający i kończący się znakiem podwójnego cudzysłowu,
- number – liczba rzeczywista,
- boolean – wartość logiczna przyjmująca wartości true/false,
- null.



Źródło: https://www.json.org/json-en.html

## 1.2 JSON Schema

Dokumenty JSON Schema to dokumenty JSON służące do opisu struktury innych dokumentów. Schematy umożliwiają sprawdzenie czy struktura danego dokumentu jest zgodna z ze strukturą określoną przez schemat. Opisują także sposób interakcji z JSONem – jak pobrać z niego konkretne dane, jakie typy mają dane.

Dla następującego dokumentu JSON:

```
{
 "first_name": "George",
 "last_name": "Washington",
 "birthday": "1732-02-22",
 "address": {
  "street_address": "3200 Mount Vernon Memorial Highway",
  "city": "Mount Vernon",
  "state": "Virginia",
  "country": "United States"
 }
}
```

Można zbudować następujący schemat:

```
{
 "type": "object",
 "properties": {
  "first_name": { "type": "string" },
  "last_name": { "type": "string" },
  "birthday": { "type": "string", "format": "date" },
  "address": {
    "type": "object",
    "properties": {
     "street_address": { "type": "string" },
     "city": { "type": "string" },
     "state": { "type": "string" },
     "country": { "type" : "string" }
    }
  }
 }
```

```
}
```

Walidacja dokumentu na podstawie schematu polega na porównaniu wartości asercji zawartych w schemacie z wartością odpowiedniego elementu dokumentu. W powyższym przykładzie wartość pola *first_name* musi mieć typ string, jeżeli typ byłby inny to dokument nie byłby zgodny ze schematem. Istnieje wiele różnych asercji, odpowiednich dla różnych typów danych m.in. minimum dla liczb czy maxLength dla ciągów znaków.

# 2.Implementacja

## 2.1 Parsowanie JSONów

Zdecydowano, że przed generacją schematu, dokument powinien zostać zapisany w pamięci jako drzewo obiektów, które będzie przechodzone podczas generacji schematu. Utworzono klasy odpowiadające typom danych występujących w JSONie:

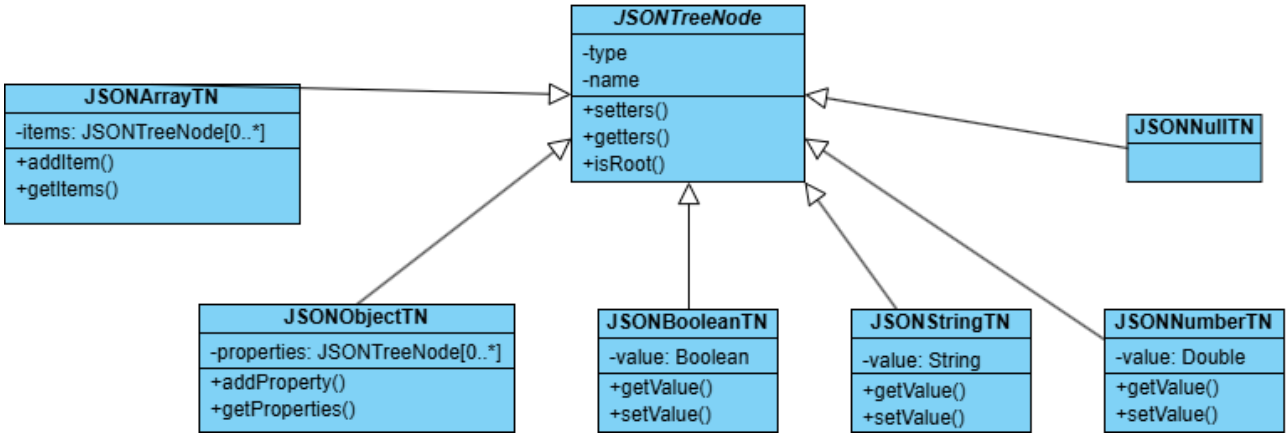



Diagram klas 1

Algorytm generujący drzewo przedstawiono na diagramach przepływu:

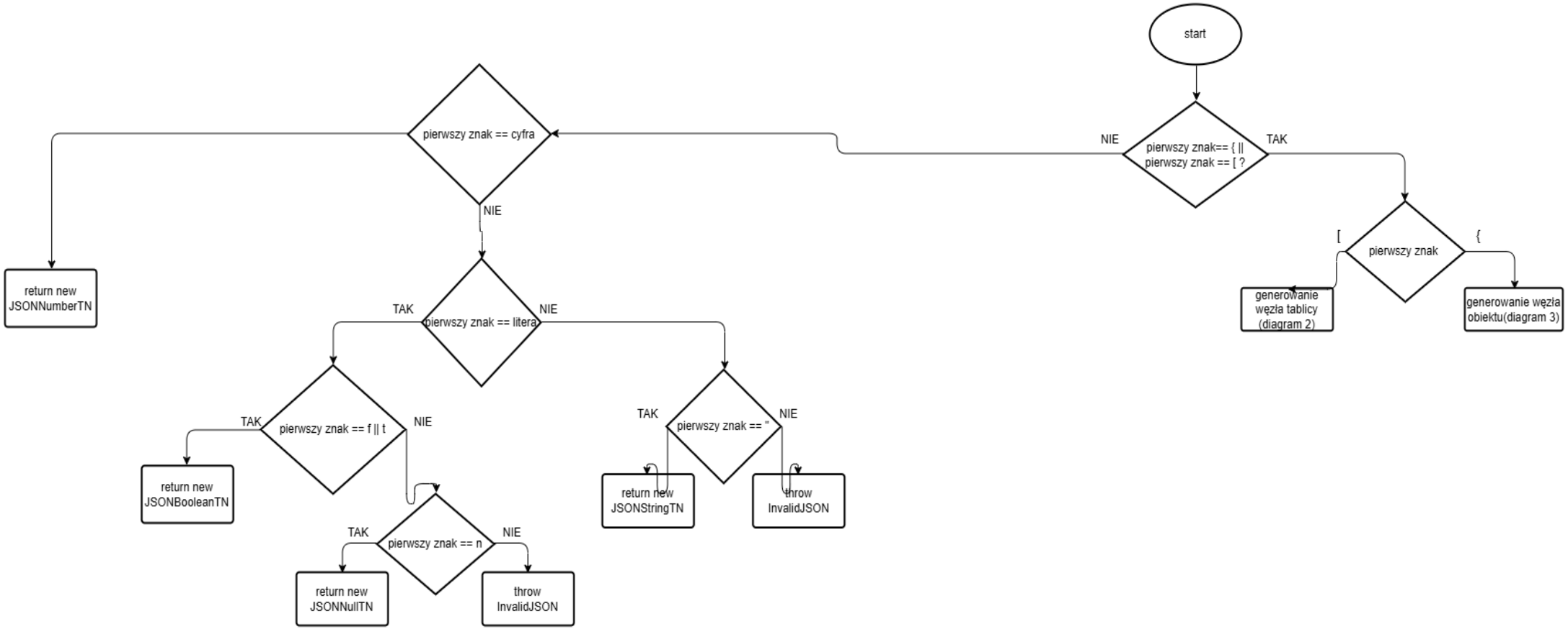



Diagram przepływu 1 – metoda generateJsonTree()

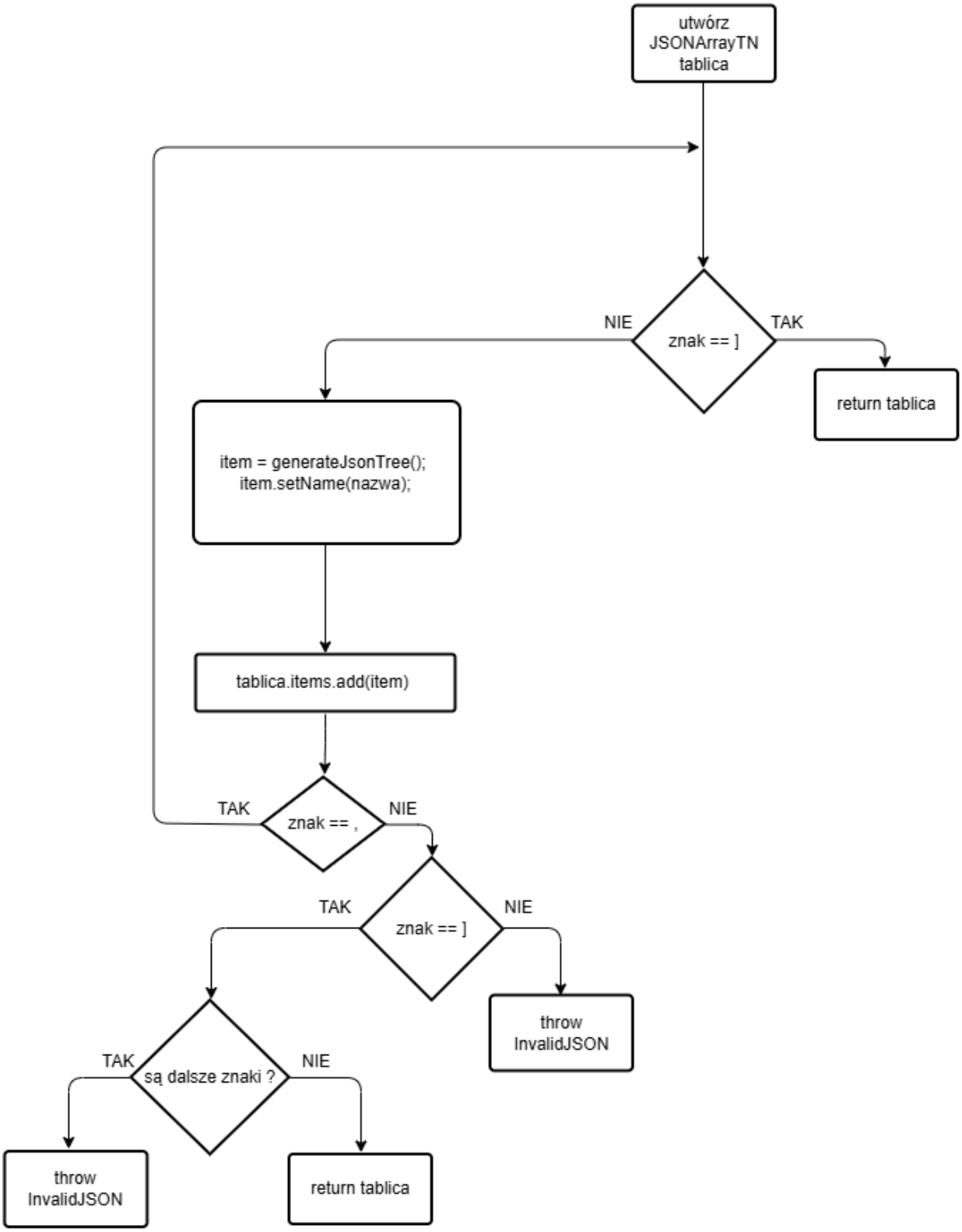



Diagram przepływu 2 – generowanie poddrzewa tablicy

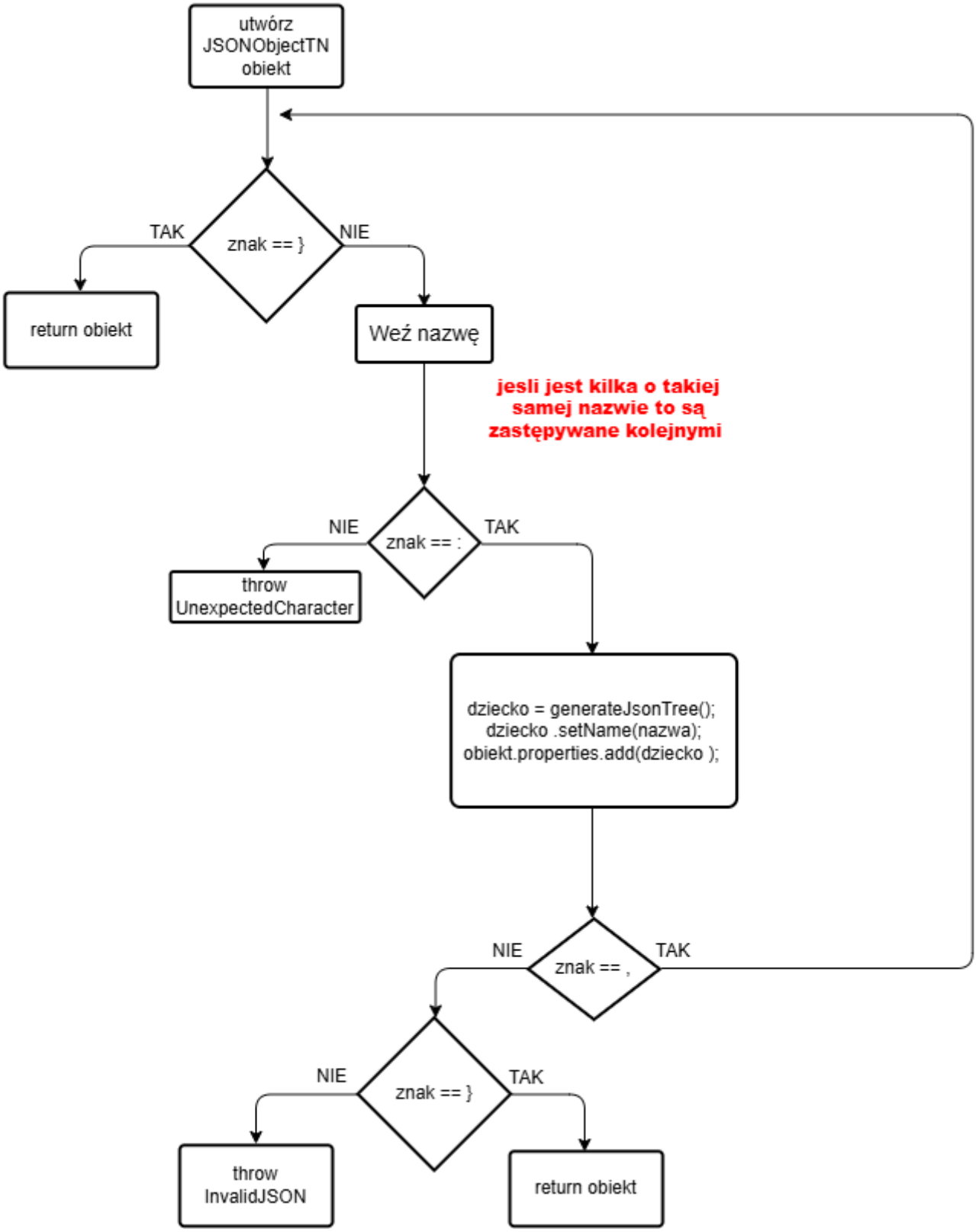



Diagram przepływu 3 – generowanie poddrzewa obiektu

## 2.2 Generacja schematu

Poniższe diagramy pokazują algorytm generowania ciągu znaków zawierającego schemat dla wybranego dokumentu JSON.

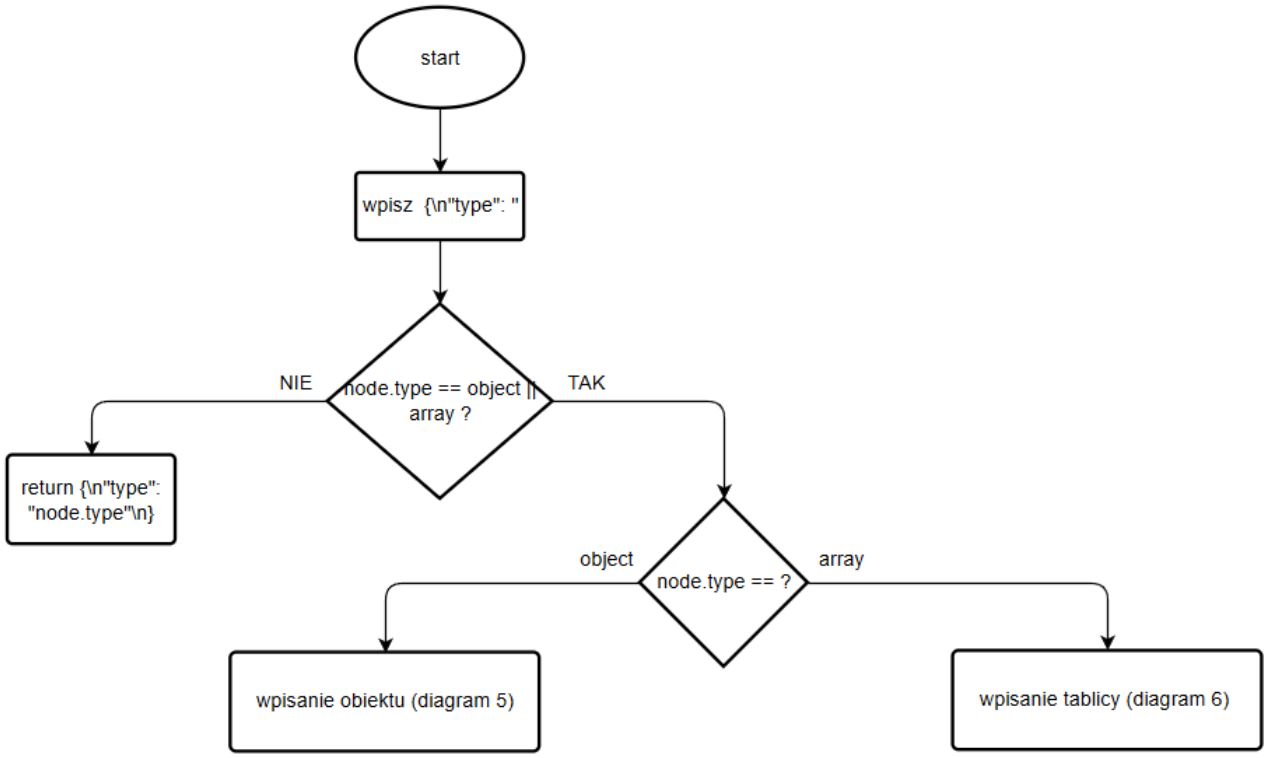



Diagram przepływu 4 – generowanie schematu

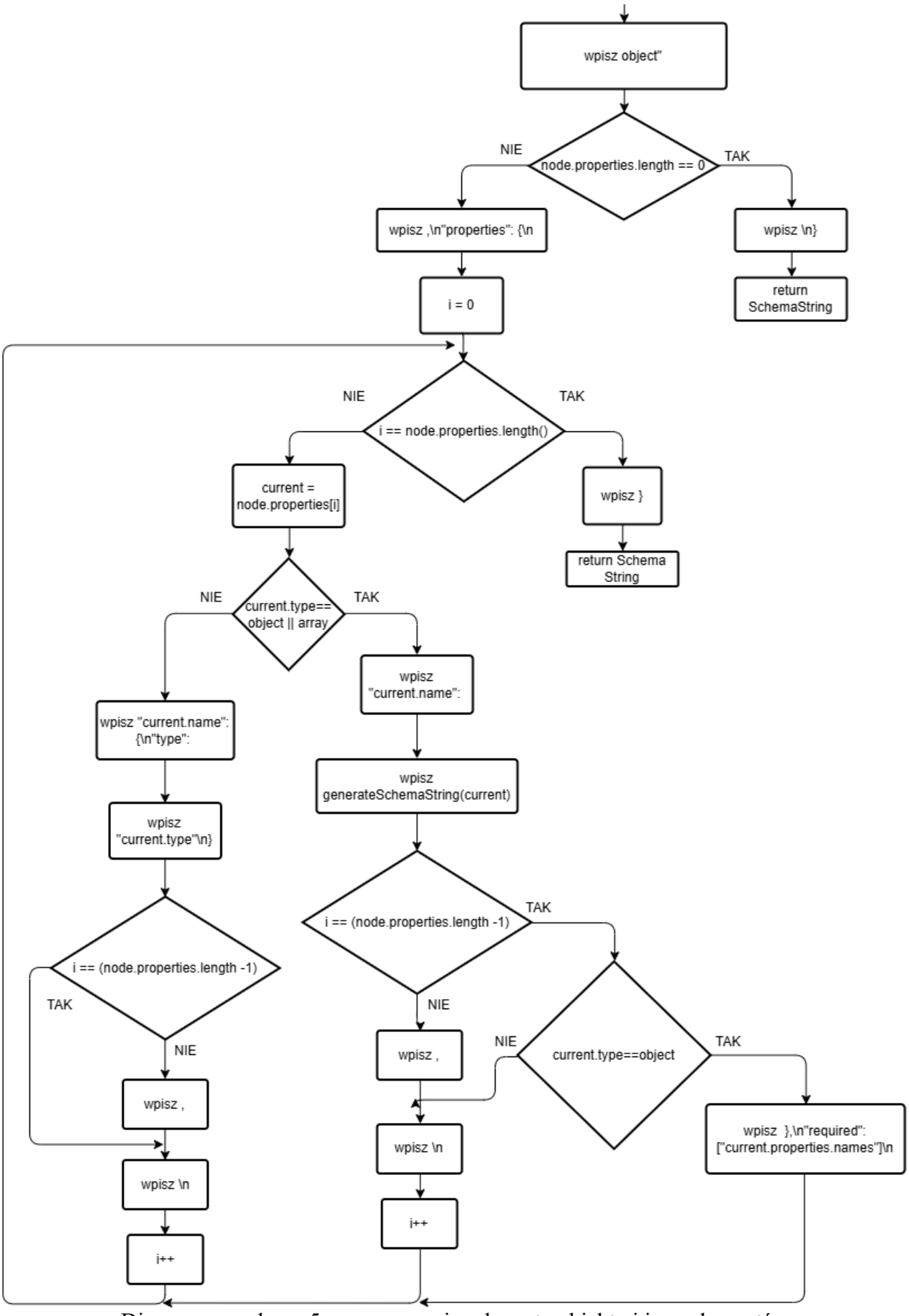



Diagram przepływu 5 – generowanie schematu obiektu i jego elementów

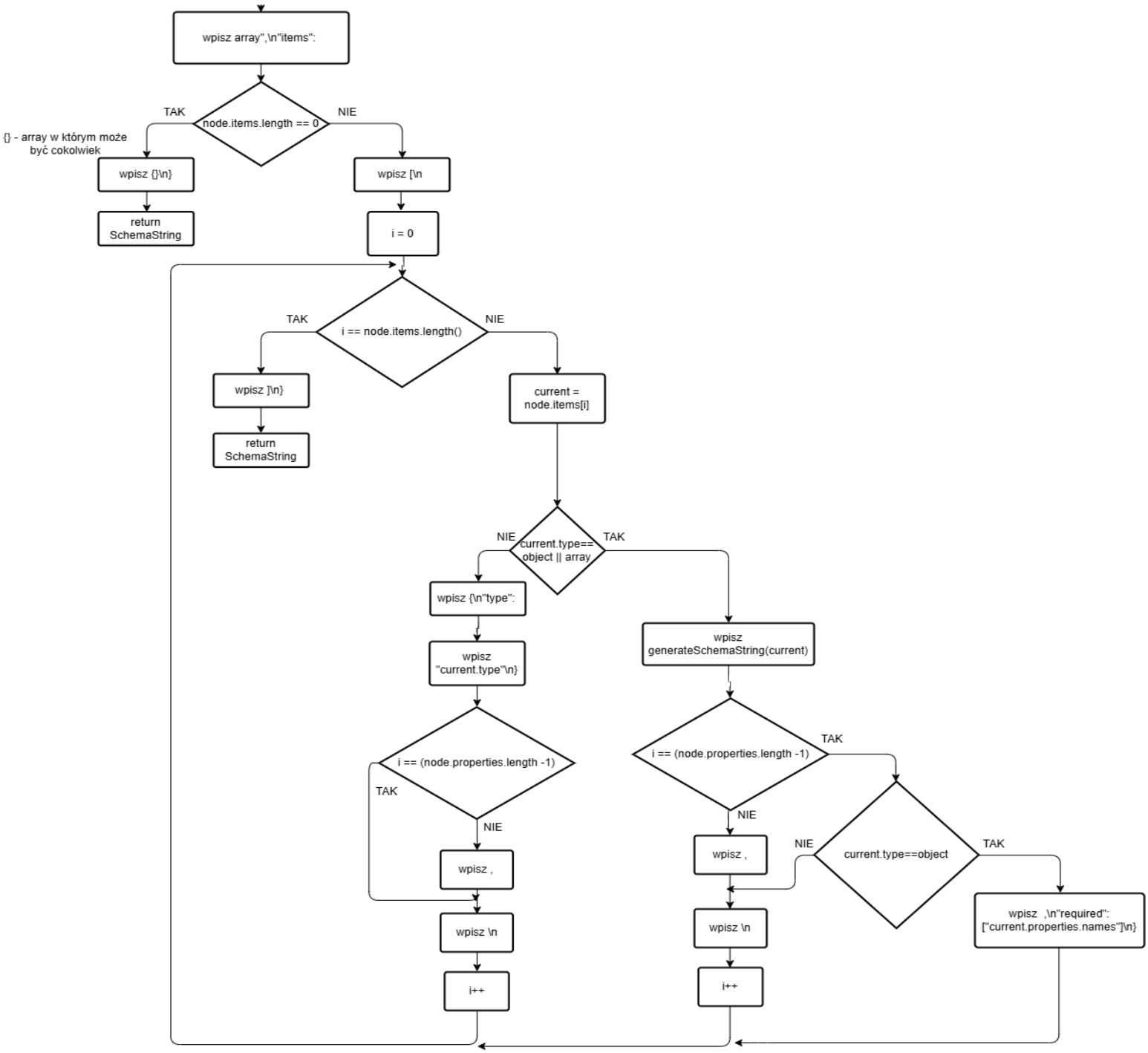



Diagram przepływu 6 – generowanie schematu tablicy i jej elementów

## 2.3 Walidacja dokumentów

Walidacja JSONów opiera się na następującym algorytmie:

1. Tworzone jest drzewo wg. algorytmu przedstawionego wcześniej zarówno dla jsona jak i schematu – korzeń drzewa dla schematu jest typu JSONObjectTN (reprez. typ obiekt), a jego dzieci reprezentują asercje, wg. których dokument ma być walidowany
2. Dla każdej asercji pobierana jest metoda określająca sposób walidacji
3. Metoda ta zostaje wykonana i zwraca wynik walidacji – true lub false (w przypadku false również komunikat dlaczego walidacja się nie powiodła)
4. W przypadku nieudanej walidacji dla asercji, wyrzucany jest wyjątek, walidowanie zostaje zakończone

Za walidację odpowiada klasa JSONValidator, która udostępnia interfejs, poprzez który użytkownik biblioteki może skonfigurować własną metodą walidującą, dla wybranej asercji.
Przykład użycia: ustawienie metody dla asercji "items", dla typu tablicy:

```
VerifyBoolAndVerifierMethod<JSONTreeNode, JSONTreeNode> verifyItems =
VerifyBoolAndVerifierMethod.withAssertionValueAsJSONTreeNode();
verifyItems.setVerifierMethod((node, assertion, validatorInstance) ->
{
    LinkedHashSet<JSONTreeNode> itemsCopy = new LinkedHashSet<>(((JSONArrayTN)
assertion).getItems());
    int itemsCopyCount = itemsCopy.size();
    LinkedHashSet<JSONTreeNode> nodeItems = ((JSONArrayTN) node).getItems();
    int i = 1;
    for (JSONTreeNode nodeItem : nodeItems) {
        if (i > itemsCopyCount)
            break;
        JSONTreeNode item = itemsCopy.removeFirst();
        try {
            validatorInstance.validateAgainstSchema(nodeItem, item);
        } catch (JSONValidationException | UnknownValidationKeywordException e)
{
            return new ValidationResultAndErrorMessage(false, e.getMessage());
        }
    }
    return ValidationResultAndErrorMessage.newInstanceValid();
});
JSONValidator validator = new JSONValidator();
validator.setVerifyingMethodForKeyword("array", "items", verifyItems);
```

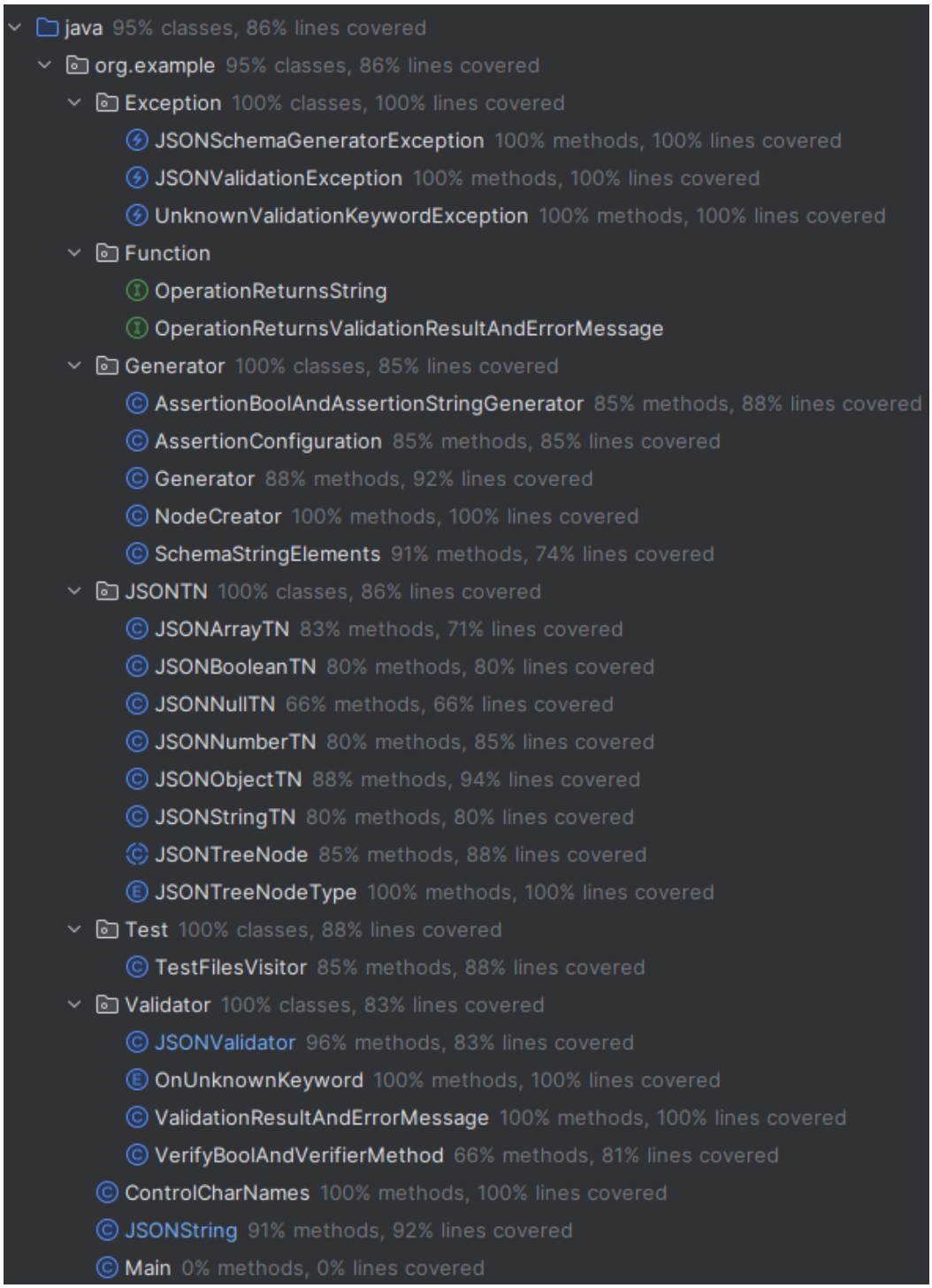
Klasa VerifyBoolAndVerifierMethod to klasa sparametryzowana. Aktualnie poprzez podanie drugiego parametru można określić typ wartości asercji, dzięki czemu w kodzie metody nie ma potrzeby rzutowania typów występujących w drzewie JSONa (diagram klas 1) i pobierania z nich wartości asercji – jest to wykonywane automatycznie. Dzięki temu użytkownik nie musi znać szczegółów klas biblioteki. W momencie pisania sprawozdania automatyczne konwertowanie walidowanego elementu JSONa (na takiej samej zasadzie) nie jest jeszcze dostępne, aczkolwiek jest w planach.
Dodatkowo klasa ta udostępnia metodę forEachElementInArray(), przystosowaną dla asercji, których wartości to tablice elementów tego samego typu. Walidowany element zostanie sprawdzony względem każdego elementu tablicy, wg. algorytmu zapisanego w metodzie.
Poniżej zamieszczono porównanie metod dla asercji "properties" dla obiektów, bez użycia sparametryzowanej klasy ani metody forEachElementInArray() oraz z użyciem tych udogodnień.

```
VerifyBoolAndVerifierMethod verifyRequired = new
VerifyBoolAndVerifierMethod((node,assertion, validatorInstance) ->
{
    ArrayList<String> nodeProperties = ((JSONObjectTN)
node).getPropertyNames();
    LinkedHashSet<JSONTreeNode> items = ((JSONArrayTN) assertion).getItems();
    ArrayList<String> requiredProperties = new ArrayList<>();
    items.forEach(item -> {
        String propertyName = ((JSONStringTN) item).getValue();
        requiredProperties.add(propertyName);
    });

    for(var rp : requiredProperties){
        System.out.println(rp);
    }
    boolean allPresent = true;
    String missingProperty="";
    for(var requiredProperty : requiredProperties){
        allPresent = nodeProperties.stream().anyMatch(property ->
property.equals(requiredProperty));
        if(!allPresent){
            missingProperty = requiredProperty;
            break;
        }
    }
    ValidationResultAndErrorMessage result = new
ValidationResultAndErrorMessage();
    if(allPresent)
        result.setValid(true);
    else{
        result.setValid(false);
        result.setMessage("Object " +node.getName()+" should contain property:
"+ missingProperty); //TODO else gdy node to root
    }
    return result;
});
```

```
VerifyBoolAndVerifierMethod<JSONTreeNode, String> verifyRequired2 =
VerifyBoolAndVerifierMethod.withAssertionValueAsArrayOfString();
verifyRequired2.forEachElementInArray(((node, assertionValue,
validatorInstance) -> {
    ArrayList<String> nodeProperties = ((JSONObjectTN)
node).getPropertyNames();
    boolean propertyIsPresentInNode = nodeProperties.stream().anyMatch(property
-> property.equals(assertionValue));
    ValidationResultAndErrorMessage result = new
ValidationResultAndErrorMessage();
    if(propertyIsPresentInNode)
        result.setValid(true);
    else{
        result.setValid(false);
        result.setMessage("Object " +node.getName()+" should contain property:
"+ assertionValue); //TODO else gdy node to root
    }
    return result;
}));
```

# 3. Testy

Od początku tworzenia kodu starano się na bieżąco testować wszystkie nowe metody, klasy, aczkolwiek nie stosowano podejścia TDD. Niestety wraz ze zbliżaniem się terminu oddania projektu tworzonych było coraz mniej testów, skupiano się bardziej na dodawaniu nowych funkcjonalności. Łącznie stworzono 104 testy, o pokryciu kodu wynoszącym 86% (wg. środowiska IntelliJ):



# 4. Instrukcja użytkownika

## 4.1 Biblioteka

### 4.1.1 Generowanie schematów

Aby wygenerować schemat, należy utworzyć instancję klasy Generator, a następnie wywołać metodę generateSchema(String json), podając jako parametr obiekt String zawierający dokument JSON. Metoda ta zwraca obiekt String z zapisanym schematem, w przypadku napotkania nieprawidłowości w dokumencie wyrzuca wyjątek:

```
try{
  String json = "{\"prop1\": [[\"string\"], null]}";
  String schema = generator.generateSchema(json);

}catch(JSONSchemaGeneratorException e){

System.out.println(e.getMessage());

}
```

Jest również możliwe otrzymanie drzewa obiektów reprezentujących dokument:

```
JSONString json = new JSONString("{\"prop1\": [[\"string\"], null]}");
JSONTreeNode result = generator.generateJsonTree(json);
```

Generowanie asercji można skonfigurować poprzez użycie klasy AssertionConfiguration:

```
ssertionConfiguration config = new AssertionConfiguration();
config.setAssertion("number", "minimum", node -> {
    if(node.getTypeAsString().equals("number"))
        return "Double.toString(((JSONNumberTN) node).getValue()";
    else
        return "Double.toString(((JSONNumberTN)
node).getValue()).split(\"\\.\")[0]";
}); //ustawienie metody generującej wartość asercji "minimum" dla typu "number"

config.setAllUnused(); //wyłączenie generowania asercji

config.setAllUsed(); //włączenie generowania asercji (domyślne)

Generator generator = new Generator(config); //ustawienie konfiguracji poprzez
konstruktor

generator.setAssertionConfiguration(config); //ustawienie konfiguracji na
istniejącym obiekcie typu Generator
```

### 4.1.2 Walidacja dokumentu

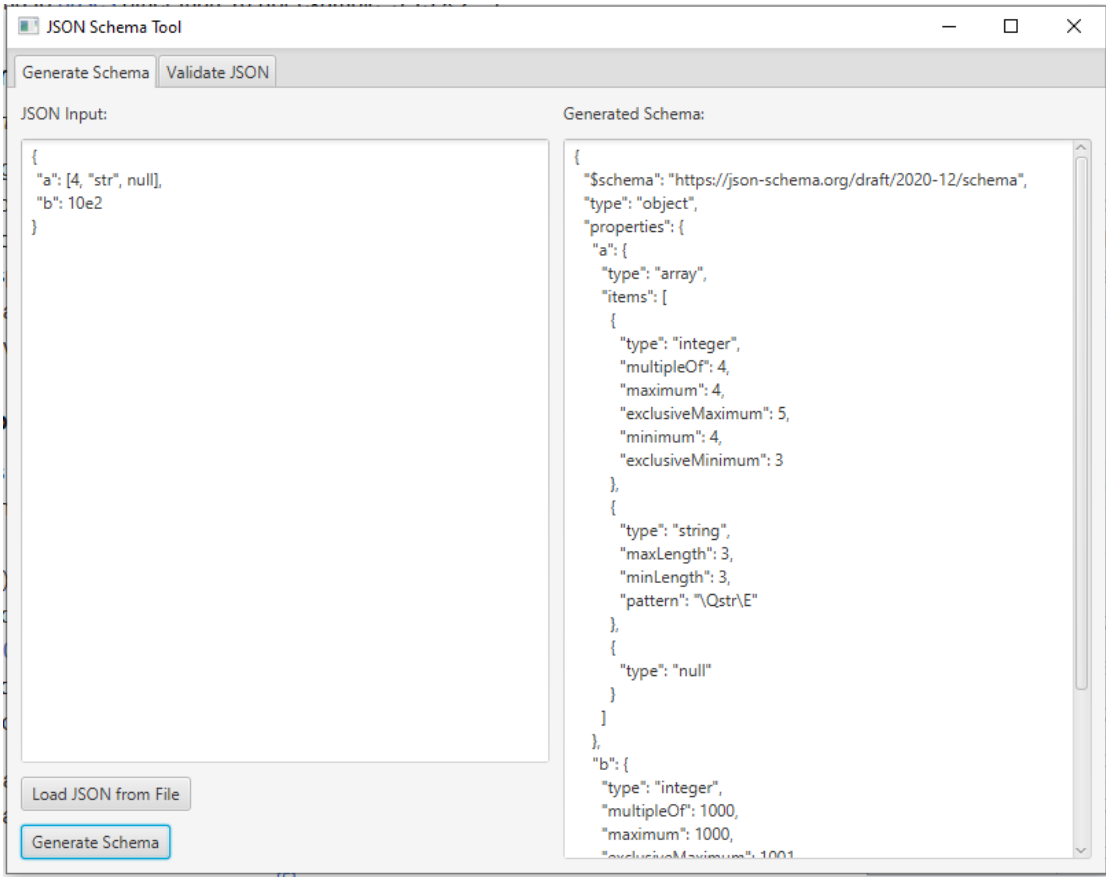
Walidacją zajmuje się klasa JSONValidator. Należy utworzyć jej instancję, a następnie wywołać jedną z trzech metod:

```
JSONValidator validator = new JSONValidator();
boolean validateAgainstSchema(String json, String schemaString) throws
JSONValidationException, JSONSchemaGeneratorException,
UnknownValidationKeywordException

boolean validateAgainstSchema(JSONTreeNode node, String schemaString) throws
JSONValidationException, JSONSchemaGeneratorException,
UnknownValidationKeywordException

boolean validateAgainstSchema(JSONTreeNode node, JSONTreeNode schema) throws
JSONValidationException, UnknownValidationKeywordException
```

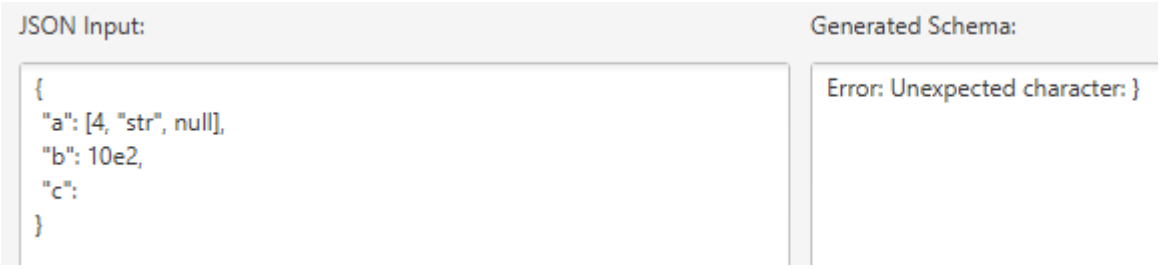
Konfiguracja własnej metody walidacji dla wybranej asercji została przedstawiona na początku rozdziału 2.3.

## 4.2 Graficzny interfejs

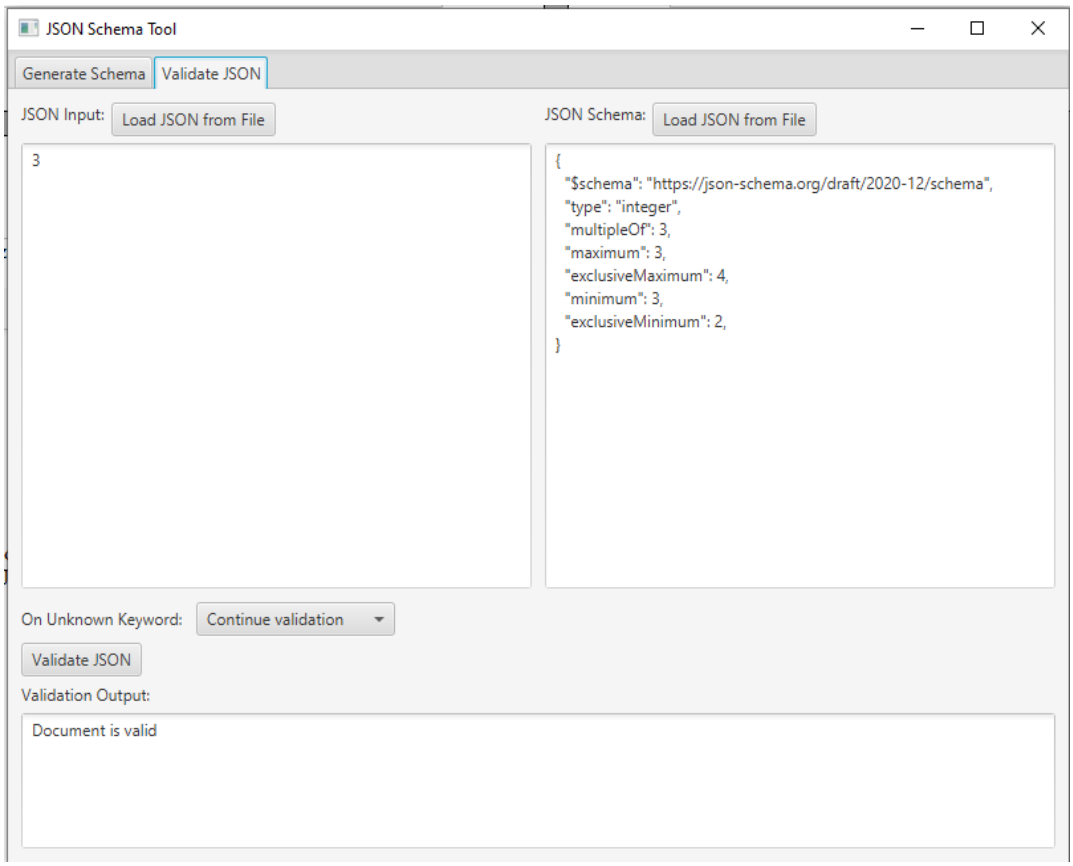
Aby wygenerować schemat należy wpisać dokument JSON do pola tekstowego po lewej stronie (lub wczytać z pliku), a następnie wcisnąć przycisk "Generate Schema".



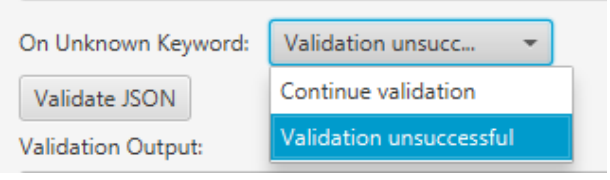
W przypadku wadliwego JSONa, w oknie po prawej stronie pojawi się stosowny komunikat:




Aby przeprowadzić walidację, w zakładce "ValidateJSON" należy wprowadzić plik JSON oraz schemat, a następnie kliknąć przycisk "Validate JSON".



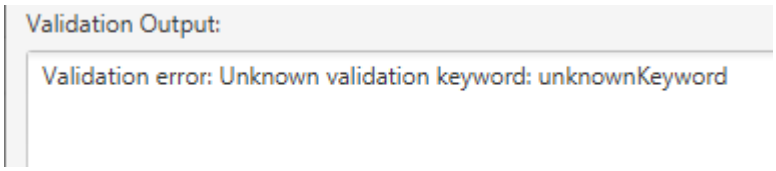

Za pomocą pola wyboru "On Unknown Keyword" można ustawić zachowanie walidatora dla nierozpoznanych asercji – kontynuowanie walidacji lub zwrócenie błędu.



Schemat z nieznaną asercją:

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "type": "integer",
  "multipleOf": 3,
  "maximum": 3,
  "exclusiveMaximum": 4,
  "minimum": 3,
  "exclusiveMinimum": 2,
  "unknownKeyword": 123
}
```

Wynik walidacji:



# 5. Wnioski

Wykonanie projektu potwierdziło łatwość tworzenia oprogramowania przetwarzającego format JSON. Kod wykonujący główne algorytmy, czyli generowanie drzewa z dokumentu oraz generowanie schematu z drzewa były dość łatwe do zaprojektowania i implementacji.

Pewne trudności sprawiła implementacja opisanych wcześniej udogodnień dla użytkownika biblioteki – praca z klasami sparametryzowanymi.

Przed pisaniem kodu nowych funkcjonalności starano się dogłębnie przemyśleć strukturę kodu, podział na klasy, metody, użycie dobrych zasad programowania obiektowego i wzorców, w taki sposób aby być przygotowanym na zmiany. Przez to projekt był tworzony powolnie, jednak rzeczywiście, kiedy pojawiła się konieczność wprowadzania zmian, to ich zasięg, w większości przypadków ograniczał się do jednej metody/klasy oraz były one łatwe do zaimplementowania.