## Cel projektu

Celem projektu jest opracowanie aplikacji, która na podstawie definicji klas zawartych w plikach YAML wygeneruje klasy w wybranym języku programowania (w tym projekcie wspierana jest Java). Klasy opisane są w plikach YAML, zgodnie ze specyfikacją OpenAPI. (Aplikacja w tej chwili potrafi generować klasy DTO. Nie potrafi generować interfejsów opisujących API).

## Pojęcia

### OpenAPI

Open API jest publicznie dostępnym interfejsem programistycznym aplikacji, który umożliwia programistom dostęp do własnościowego programowania lub serwisu. API jest zbiorem wymagań, które opisuje jak jedna aplikacja może komunikować się z inną.

Specyfikacja OpenAPI (zna także jako specyfikacja Swaggera) jest czytelnym dla komputera sposobem opisu interfejsów serwisów REST API.

Aplikacje, które implementują interfejs OpenAPI mogą autmatycznie generować dokumentację metod, parametrów oraz modeli. Ułatwia to utrzymywanie dokumentacji API w najnowszej postaci.

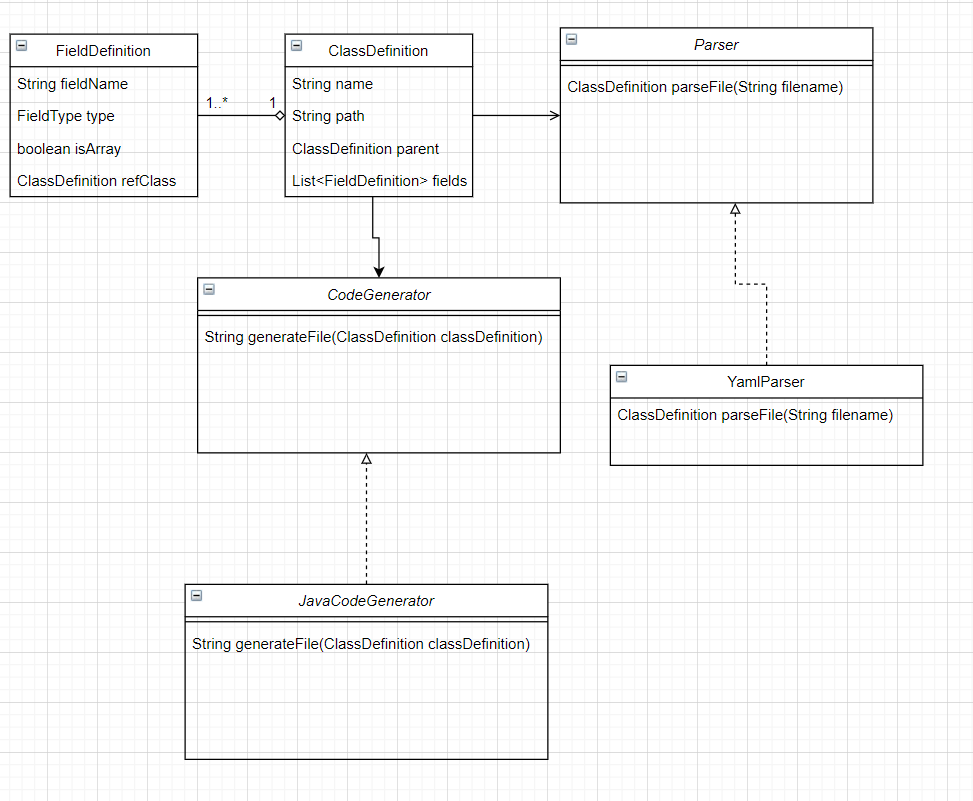
Zaprojektowana aplikacja wspiera jedynie generowanie modeli DTO wg specyfikacji OpenAPI. Nie wspiera generowania pełnego kodu aplikacji serwera/klienta.

### Yaml

YAML jest uniwersalnym językiem przeznaczonym do reprezentowania różnych danych w ustrukturalizowany sposób.

Poszczególne elementy struktury danych są oddzielane znakami nowej linii, a ich hierarchia ustalana jest na podstawie wcięcia linii. Język wprowadza trzy podstawowe struktury danych, które mogą być wkomponowane w dokumenty: listy, słowniki i skalary. Język obsługuje również referencje, które eliminują konieczność redundancji danych.

## Diagram klas



Rysunek 1: Uproszczony diagram klas programu (tylko dla generatora Java)

Rysunek 1 przedstawia diagram pokazujący klasy oraz ich relację w programie yaml2ProgConverter.

*ClassDefinition* oraz *FieldDefinition* są klasami odpowiedzialnymi za uniwersalny sposób opisu klas znajdujących się w parsowanych plikach.

Klasy *CodeGenerator* oraz *Parser* są uniwersalnymi interfejsami, które udostępniają metody odpowiednio do: generowania kodu oraz parsowania plików.

Domyślną implementacją klasy *CodeGenerator* jest *JavaCodeGenerator*. Klasa ta, na podstawie dostarczonej definicji klasy generuje kod w języku Java.

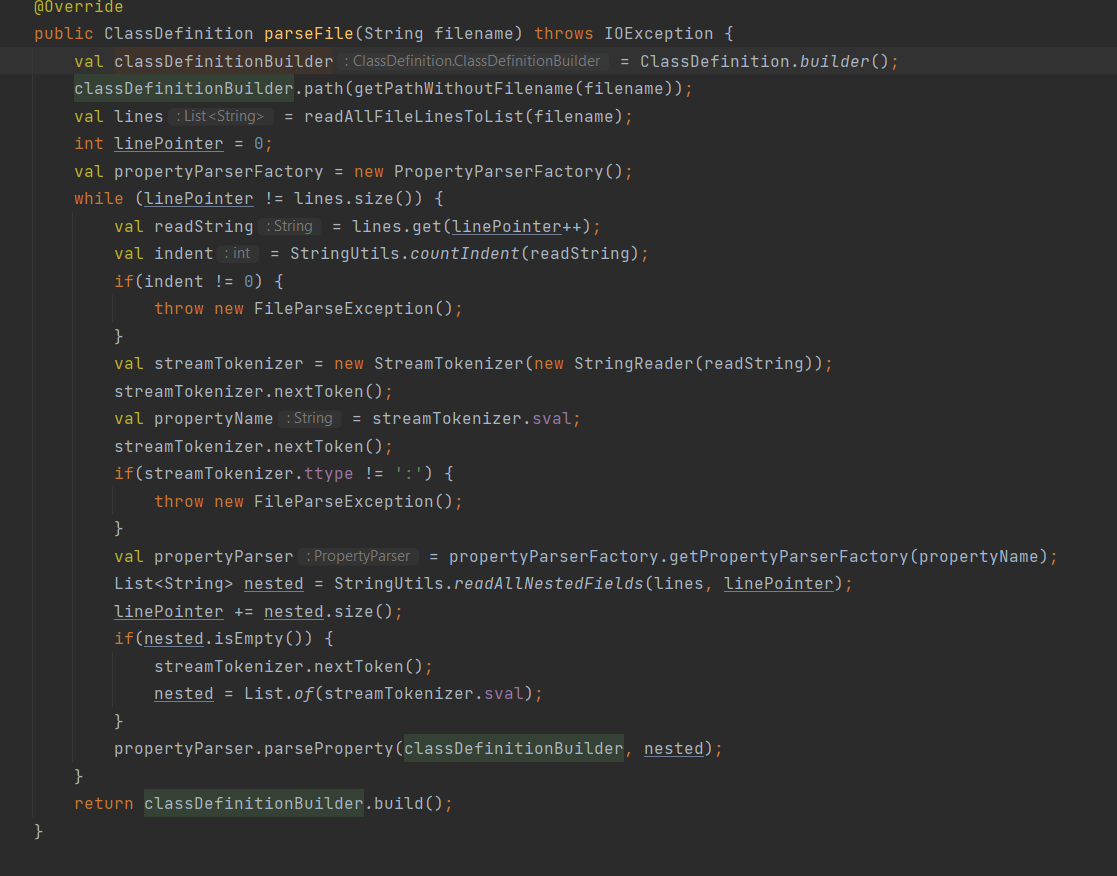
Domyślnym parserem jest *YamlParser* który na podstawie pliku (którego nazwa przekazywana jest jako argument) generuje definicję klasy oraz jej atrybutów.

Parser został zaimplementowany samodzielnie w oparciu o StreamTokenizer.

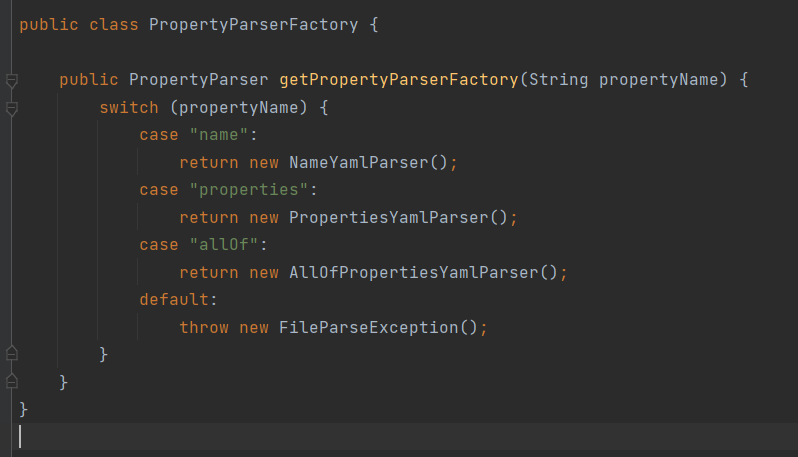
## Algorytm parsowania pliku yaml

Parsowanie pliku yaml zostało wykonane ręcznie w oparciu o StreamTokenizer i standardowe metody obsługujące wejście/wyjście w Javie.

Poniższy screen pokazuje pełną implementację klasy parsującej plik Yaml.



Pierwszym krokiem jest odczytanie wszystkich linii z pliku. Następnie plik jes czytany linijka po linijce. Za pomocą streamtokenizer pobierane są kolejne elementy pliku. W celu odpowiedniego sparsowania danej sekcji, tworzony jest odpowiedni parser za pomocą propertyParserFactory.



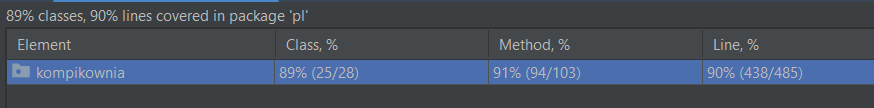
Klasa PropertyParserFactory została przedstawiona powyżej. W zależności od tego, jaką właściwość lub sekcję przetwarzamy, tworzony jest odpowiedni parser. Jeśli nie znajdziemy odpowiedniego parsera, wyrzucany jest wyjątek.

Gdy plik zostanie sparsowany, w zależności od wybranego języka docelowego używany jest odpowiedni generator kodu. Generator kodu, na podstawie ogólnej struktury klasy znajdującej się w pliku ClassDefinition, generuje kod źródłowy w docelowym języku programowania, który następnie może zostać skompilowany.

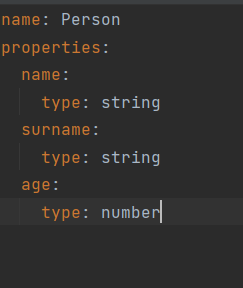
Opcjonalnie może zostać wygenerowany także projekt (obecnie wspierany jest jedynie jeden rodzaj projektu: Maven dla języka Java).

## Testy jednostkowe

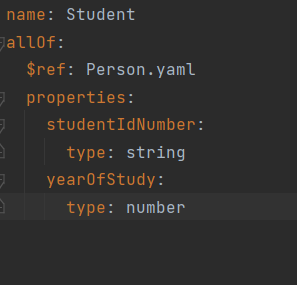
Do każdego elementu programu zostały wykonane testy jednostkowe. Pokrycie testami wynosi prawie 90%, a więc jest bardzo dobre.



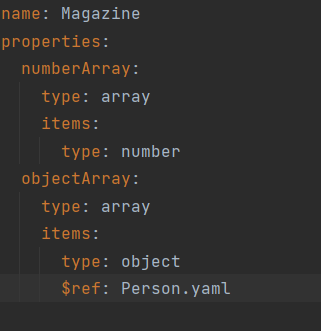
## Przykładowe plik Yaml



Na podstawie powyższego pliku yaml zostanie wygenerowana klasa Person zawierająca trzy atrybuty: name typu String, surname typu String oraz age typu Double.



Powyższy plik generuje klasę Student, która dziedziczy wszystkie atrybuty klasy Person. Atrybuty dodane przez klasę student to studentIdNumber typu String oraz yearOfStudy typu Double.



Powyższy plik generuje klasę o nazwie Magazine, która posiada dwa atrybuty. Są to listy: typu prostego (Double) oraz klasowego (Person).

## Instrukcja obsługi programu

Program należy uruchomić z listy komend systemu operacyjnego. Wymagana jest obecność środowiska wykonawczego języka Java (JRE).

Obsługiwane parametry:

**--file [nazwa\_pliku] -** nazwa pliku zawierającego opis klasy. Jeśli plik zawiera referencje do innych plików, także one zostaną przetworzone na kod.

**--packageName [pakiet]-** nazwa pakietu, w którym będą znajdowały się wygenerowane klasy

**--projectType [typ\_projektu] -** (nieobowiązkowy) typ projektu. (Jedyny wspierany: maven)

**--projectGroupId -** grupa artefaktu

**--projectArtifactId** - nazwa artefaktu

**--projectVersion -** wersja projektu