Autorzy:
Krzysztof Nowakowski
Krystian Ujma

Program rysujący diagram UML na podstawie kodu źródłowego javy.

Krótki opis programu:

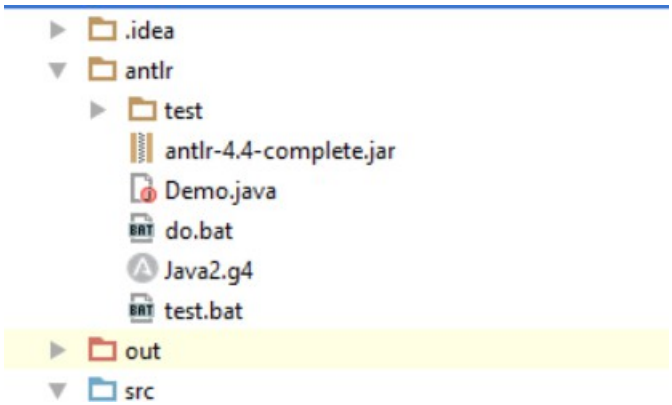
Nasz program tworzy uproszczony diagram UML z kodu źródłowego javy używając do tego generatora parserów i lekserów ANTLR 4.

Program zbiera informacje potrzebne do wygenerowania UML-a chodząc po drzewie i używając listenerów znajdujących się w klasie MyListener. Zebrane informacje są przechowywane w klasie UmlClass i następnie wykorzystane do narysowania GUI.

Główne komponenty projektu:

1. Katalog antlr:

- **Demo.java**: plik z kodem źródłowym javy, na podstawie którego zostanie wygenerowany diagram UML.



Ilustracja 1: plik Demo.java

- **do.bat**: skrypt który generuje parser, lekser i listenery i kopiuje je do odpowiedniego katalogu.

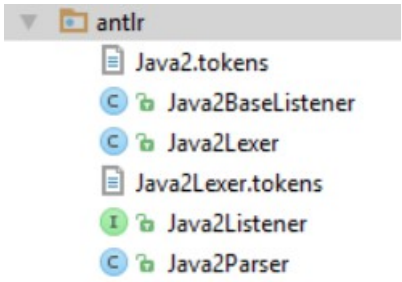
```
antlr4 -o ../src/pl/edu/agh/java/antlr -package pl.edu.agh.java.antlr Java2.g4
```

Ilustracja 2: skrypt do.bat

- **Java2.g4**: plik z pełną gramatyką Javy
- **antlr-4.4-complete.jar**: biblioteka ANTLR4

2. paczka: src.pl.edu.agh.java.antlr

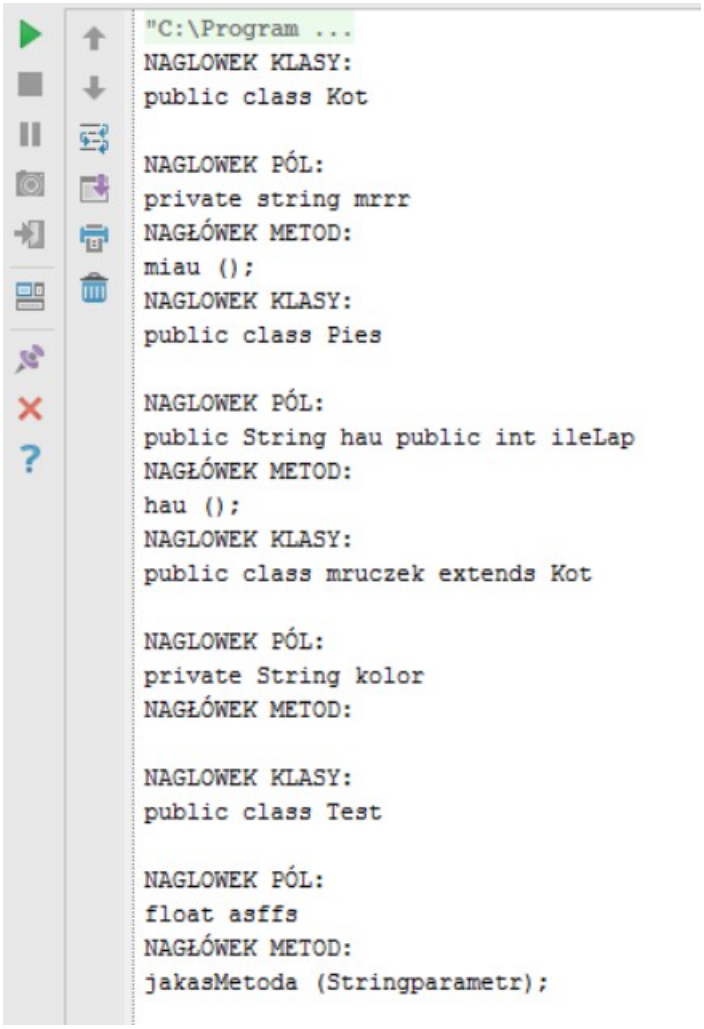
Tutaj znajdują się pliki wygenerowane za pomocą skryptu do.bat



Ilustracja 3: Pliki wygenerowane za pomocą skrytpu do.bat

3. **paczka : src.pl.edu.agh.java.application**

- **ApplicationTester**: główna klasa testowa uruchamiająca aplikację i rysująca diagram UML:
- **Starter**: pomocnicza klasa, która analizuje plik źródłowy i zwraca interesującą nas zawartość na konsolę. Przykładwy dane wyjściowe:



Klasa Starter ma jedną z naszego punktu widzenia ważne metody:

- parse2(): stworzony tam jest lekser i parser do którego wrzucony jest strumień wejściowych danych. Następnie stworzone jest drzewo (ParseTree) po którym chodzi walker (ParseTreeWalker). Podczas tego przejścia uruchamiany jest interesujący nas trigger (znajdujący się w klasie MyListener).

4. **paczka : src.pl.edu.agh.java.extended:**

- MyListener: Klasa rozszerzająca antlr-owską klasę Java2BaseListener. Za jej pomocą odapalane są zdarzenia kiedy chodzimy po drzewie. Główna metoda to enterTypeDeclaration():

```
@Override
public void enterTypeDeclaration(@NotNull
Java2Parser.TypeDeclarationContext ctx) {
  UmlClass umlClass = new UmlClass();
  String classDeclaration = "";
  String fieldDeclaration = "";
  String methodDeclaration = "";
  // CLASS DECLARATION:
  if (ctx.classDeclaration() != null) {
    // get modifiers (i.e. "public"):
    if (ctx.classOrInterfaceModifier() != null) {
      for (int i = 0; i < ctx.classOrInterfaceModifier().size(); i+
+) {
        classDeclaration +=
ctx.classOrInterfaceModifier().get(i).getText() + " ";
      }
    }
    // get "class" + class name :
    classDeclaration += "class " +
ctx.classDeclaration().Identifier().getText() + " ";
    // get extended class
    if (ctx.classDeclaration().type() != null) {
      classDeclaration += "extends " +
ctx.classDeclaration().type().getText() + " ";
    }
    // get implemented interface
    if (ctx.classDeclaration().typeList() != null) {
      classDeclaration += "implements " +
ctx.classDeclaration().typeList().type().get(0).getText() + " ";
    }
    classDeclaration += "\n";
  }
  umlClass.setClassName(classDeclaration);
  // FIELD/CONSTRUCTOR/METHOD DECLARATION:
  if (ctx.classDeclaration() != null) {
    if (ctx.classDeclaration().classBody().classBodyDeclaration() !
= null) {
      for (int i = 0; i <
ctx.classDeclaration().classBody().classBodyDeclaration().size();
i++) {
        Java2Parser.MemberContext member =
ctx.classDeclaration().classBody().classBodyDeclaration().get(i).m
ember();
        List<Java2Parser.ModifierContext> modifier =
```

```
ctx.classDeclaration().classBody().classBodyDeclaration().get(i).m
odifiers().modifier();
        // GET MODIFIER:
        String modifierString = "";
        if (modifier != null) {
          for (int j = 0; j < modifier.size(); j++) {
            modifierString += modifier.get(j).getText() + " ";
          }
        }
        // GET FIELD:
        if (member.fieldDeclaration() != null) {
          fieldDeclaration += modifierString +
member.fieldDeclaration().type().getText() + " ";
          // get field info only up to equation mark
          List<Java2Parser.VariableDeclaratorContext> varDec =
member.fieldDeclaration().variableDeclarators().variableDeclarator
();
          if (varDec != null) {
            for (int j = 0; j < varDec.size(); j++) {
              fieldDeclaration +=
varDec.get(j).variableDeclaratorId().getText() + " ";
            }
          }
          umlClass.AddField(fieldDeclaration);
        }
        // GET METHOD:
        if (member.methodDeclaration() != null) {
          methodDeclaration += modifierString;
          // get method type:
          if (member.methodDeclaration().type() != null) {
            methodDeclaration +=
member.methodDeclaration().type().getText() + " ";
          }
          // get method name + formal parameters:
          methodDeclaration +=
member.methodDeclaration().Identifier().getText() + " "
            +
member.methodDeclaration().formalParameters().getText() + ";";
          umlClass.addMethod(methodDeclaration);
        }
      }
    }
  }
  extractedClasses.add(umlClass);
  System.out.println("NAGLOWEK KLASY:\n" + classDeclaration);
  System.out.println("NAGLOWEK PÓL:\n" + fieldDeclaration);
  System.out.println("NAGŁÓWEK METOD:\n" + methodDeclaration);
}
```

-

- UmlClass: klasa służąca do przechowywania informacji ze sparsowanego kodu źródłowego.

5. **paczka : src.pl.edu.agh.java.gui:**
Klasy służące do rysowania uproszczonego UML-a.

Składnia (gramatyka)

Aby dany ciąg znaków mógł być rozpoznany jako program napisany w danym języku, musi spełniać pewne reguły, zwane składnią.

Składnia opisuje:
• rodzaje dostępnych symboli
• zasady, według których symbole mogą być łączone w większe struktury

Składnia najczęściej opisywana jest w formalnym zapisie będącym połączeniem wyrażeń regularnych oraz notacji BNF lub EBNF

Gramatyka w naszym projekcie znajduje się w pliku Java2.g

Semantyka

Semantyka języka programowania definiuje precyzyjnie znaczenie poszczególnych symboli oraz ich funkcję w programie. Semantykę najczęściej definiuje się słownie, ponieważ większość z jej zagadnień jest trudna lub wręcz niemożliwa do ujęcia w jakikolwiek formalizm.

Semantyka (jej część) znajduje się w pliku Java2.g

Dwa podejścia do gramatyki języka w naszym projekcie.

Początkowo stworzyliśmy własną uproszczoną gramatykę języka javy. Jednak wynikły problemy z odczytywaniem pól klasy. Mimo usilnych i długich starań (między innymi konsultacja z Panem na laboratoriach) problemu nie udało nam się rozwiązać i zdecydowaliśmy się użyć pełnej gramatyki javy pobranej ze strony projektu ANTLR. Poniżej zamieszczam stworzoną przez nas uproszczoną gramatykę:

```
grammar JavaUproszczona;

// plik z kodem
compilationUnit :
       packageDeclaration? importDeclaration* typeDeclaration
       EOF
;
```

```
// nazwa paczki
packageDeclaration :
       'package' qualifiedName ';'
;

// importowane rzeczy
importDeclaration :
       'import' 'static'? qualifiedName ('.*')? ';'
;

// typ deklarowanej rzeczy: klasa, interfejs, enum
typeDeclaration
       : (ClassOrInterfaceModifier)* 'class' qualifiedName ('extends' qualifiedName)?
('implements' qualifiedName)*
       (ClassOrInterfaceModifier)* classTotal
         (   classTotal
         |   interfaceDeclaration
         |   enumDeclaration
         )
    |   ';'
;

classTotal :
       classDeclaration classBody
;

classDeclaration :
        'class' qualifiedName ('extends' qualifiedName)? ('implements' qualifiedName)*
;

classBody :
       '{' (field | method | Identifier)* '}'
;

field :
       ClassOrInterfaceModifier* Identifier Identifier ('[' ']')* anychar ';'
;

method:
       ClassOrInterfaceModifier* Identifier Identifier
       '(' Identifier Identifier ('[' ']')* (Identifier Identifier ('[' ']')* ',')* ')'
        '{' (. | WS)* '}'
;

method:
       ClassOrInterfaceModifier* Identifier ('[' ']')* Identifier ('[' ']')* '('
Identifier Identifier ('[' ']')* (Identifier Identifier ('[' ']')* ',')* ')' '{'
(anychar | ';')* '}'
;

// nazwa klasy lub paczki
qualifiedName     :
       Identifier ('.' Identifier)*
;

anychar:
       Letter | Digit | Sign
  ;


ClassOrInterfaceModifier
```

```
    :   'public'
    |   'protected'
    |   'private'
    |   'abstract'
    |   'static'
    |   'final'
    ;
```

```
Identifier : Letter (Letter|Digit)* ;

Letter: [a-zA-Z];

Digit: [0-9];

Sign: ('=' | '"' | '(' | ')' | ',' | '[' | ']' | '<' | '>');

WS:   [ \r\t\u000C\n]+ -> skip ;
```

Program uruchomiony na przykładowych testowych danych: