

# Woźniak Mikołaj

## Generator Diagramu Klas

<https://github.com/mxd1232/Reverse-Engineering>

## Diagram klas

– statyczny diagram strukturalny w UML, przedstawiający strukturę systemu w modelach obiektowych przez ilustrację struktury klas i zależności między nimi.

Diagram klas pokazuje określony fragment struktury systemu. Diagramów klas używa się do modelowania statycznych aspektów perspektywy projektowej. Wiąże się z tym silnie modelowanie słownictwa systemu, kooperacji lub schematów. Diagramy klas pozwalają na sformalizowanie specyfikacji danych i metod. Mogą także pełnić rolę graficznego środka pokazującego szczegóły implementacji klas.

### Klasy

Klasa w modelu UML programu obiektowego jest reprezentowana przez prostokąt z umieszczoną wewnątrz nazwą klasy. Oddzielona część prostokąta pod nazwą klasy może zawierać atrybuty klasy, czyli metody, właściwości lub pola. Każdy atrybut pokazywany jest przynajmniej jako nazwa, opcjonalnie także z typem, wartością i innymi cechami.

Metody klasy mogą znajdować się w osobnej części prostokąta. Każda metoda jest pokazywana przynajmniej jako nazwa, a dodatkowo także ze swoimi parametrami i zwracanym typem.

Atrybuty (zmienne i właściwości) oraz metody mogą mieć też oznaczoną widoczność (zakres znaczenia ich nazw) jak następuje:

"+" dla public – publiczny, dostęp globalny

"#" dla protected – chroniony, dostęp dla pochodnych klasy (wynikających z generalizacji)

"-" dla private – prywatny, dostępny tylko w obrębie klasy (przy atrybucie statycznym) lub obiektu (przy atrybucie zwykłym)

"~" dla package – pakiet, dostępny w obrębie danego pakietu, projektu.

### Związki (powiązania)

## Oznaczenia związków klas w UML

### Zależność

Zależność (ang. dependency) – najsłabszy związek znaczeniowy między klasami, gdy jedna z klas używa innej. Na diagramie klas oznaczana przerywaną linią zakończoną strzałką wskazującą kierunek zależności.

### Asocjacja

Asocjacja (ang. association) wskazuje na silniejsze powiązanie pomiędzy obiektami danych klas (np. firma zatrudnia pracowników). Na diagramie asocjację oznacza się za pomocą linii, która może być zakończona strzałką z otwartym grotem (oznaczającą kierunek powiązania klas). Nazwy cech (np. zatrudniony, zatrudniający) wraz z krotnością umieszcza się w punkcie docelowym asocjacji. Nazwę asocjacji podaje się pośrodku (np. zatrudnia).

### Generalizacja

Generalizacja lub dziedziczenie – związek opisujący klasy i podklasy (ogólne klasy i szczegółowe lub inaczej rodziców i dzieci). Na diagramie generalizację oznacza się za pomocą niewypełnionego trójkąta symbolizującego strzałkę (skierowaną od klasy pochodnej do klasy bazowej).

### Agregacja

Agregacja reprezentuje związek typu całość-część, czyli jakaś większa całość jest rozbita na elementy. Oznacza to, że elementy częściowe mogą należeć do większej całości, jednak również mogą istnieć bez niej (np. koła i samochód). Na diagramie agregację oznacza się za pomocą linii zakończonej pustym rombem.

### Kompozycja

Kompozycja, jest silniejszą formą agregacji. W związku kompozycji, części należą tylko do jednej całości, a ich okres życia jest wspólny — razem z całością niszczone są również części. W dużej mierze jest to kwestia umowna, zależna od danego systemu. Na diagramie, kompozycję oznacza się za pomocą linii zakończonej wypełnionym rombem.

# Refleksja

- służy to uzyskaniu informacji o typie w trakcie wykonywania programu. Klasy, które mają dostęp do metadanych działającego programu są zdefiniowane w przestrzeni nazw System.Reflection.

Przestrzeń ta zawiera klasy, które pozwalają na uzyskanie informacji o aplikacji oraz pozwalają na dynamiczne dodawanie typów, wartości i obiektów do aplikacji.

## Możliwości refleksji:

podgląd atrybutów w trakcie wykonywania programu;

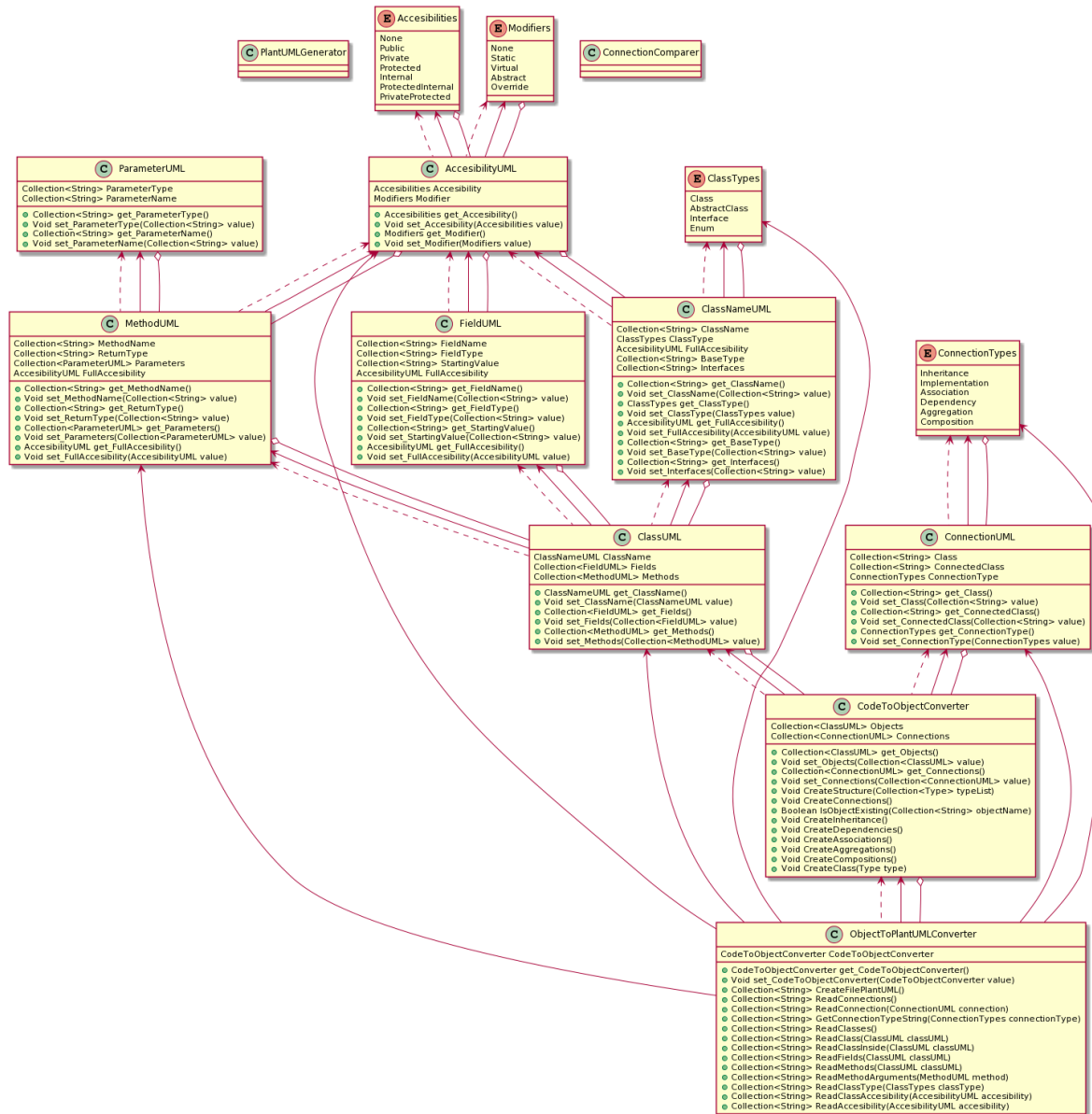
sprawdzenie różnych typów danych w danej bibliotece oraz utworzenie ich instancji;

wykonanie późnego wiązania do metod i właściwości (późne wiązanie oznacza, że np. docelowa metoda jest poszukiwana w trakcie wykonywania programu. Wiązanie takie ma zwykle wpływ na wydajność. Poszukiwanie takie wymaga dopasowania w trakcie wykonywania programu, oznacza to, że wywołania metod są wolniejsze. Przeciwnieństwem jest wczesne wiązanie, tj. docelowa metoda jest znana już w trakcie kompilacji kodu);

tworzenie nowych typów w trakcie wykonywania programu a następnie wykonywanie różnych zadań przy użyciu tych typów.

Program napisany jest w języku c#. Przy użyciu refleksji oraz pośredniej struktury classUML generowany jest kod w języku plantUML na podstawie którego można generować diagramy klas.

## STRUKTURA PROGRAMU:



Powyżej znajduje się struktura programu:

ObjectToPlantUMLConverter – Parser który konwertuje strukturę classUML do postaci ciągu znaków.

CodeToObjectConverter – Przy pomocy refleksji konwertuje Assembly do postaci obiektowej struktury class UML

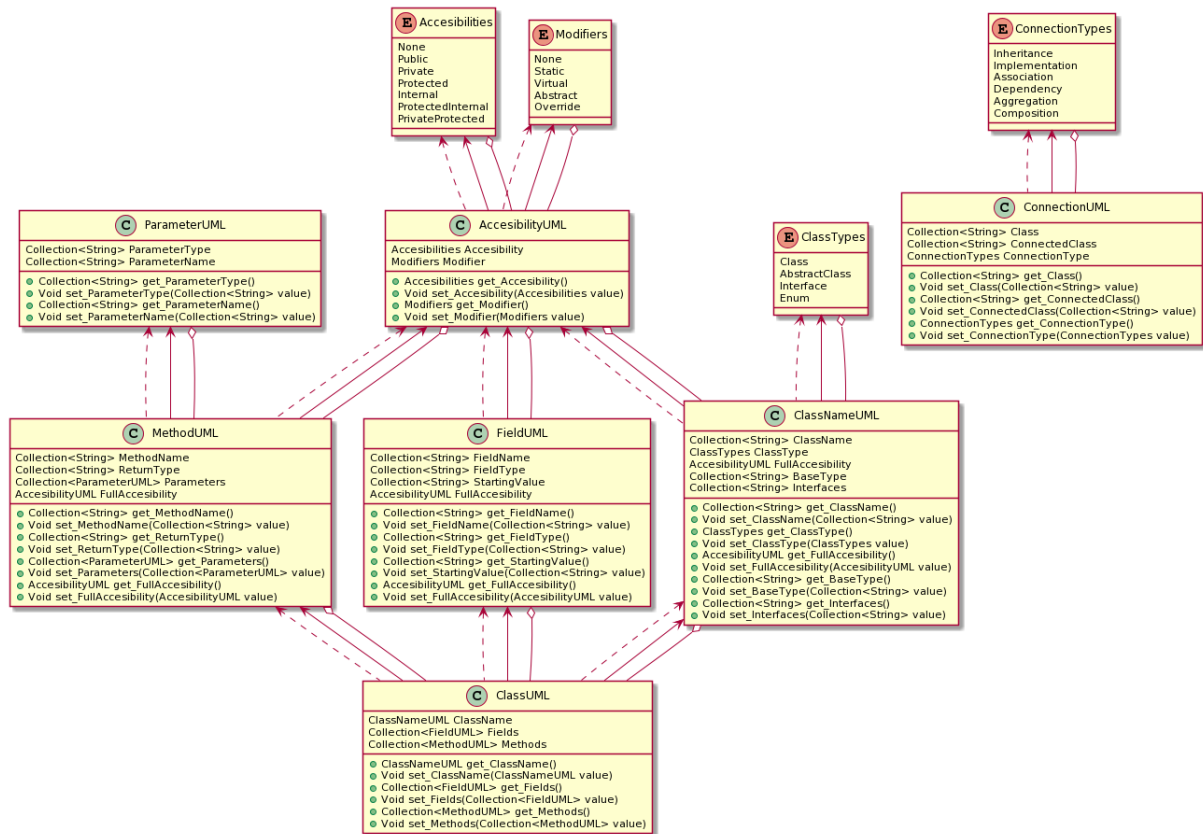
Struktura class UML -> odwzorowanie naturalne klas, pól metod, argumentów metod.

ConnectionUML – uzupełnienie struktury class UML zawierające listę połączeń (relacji) między klasami.

ConnectionComparator – klasa pomocnicza porównująca połączenia w hashmapie. (by uniknąć powtórzeń).

PlantUMLGenerator- klasa rozpoczynająca program i ładująca Assembly z pliku dll lub z folderu Classes (opcjonalnie może wygenerować strukturę programu)

Struktura Class UML:



ClassUML – zawiera nazwę klasy, pola i metody

ClassNameUML – nazwa klasy, typ, dostępność, klasa bazowa i lista interfejsów

MethodUML – reprezentacja metody

ParameterUML – reprezentacja argumentu metody

AccessabilityUML – reprezentacja modyfikatoru dostępności

ConnectionUML – Związek pomiędzy klasami

ENUMY:

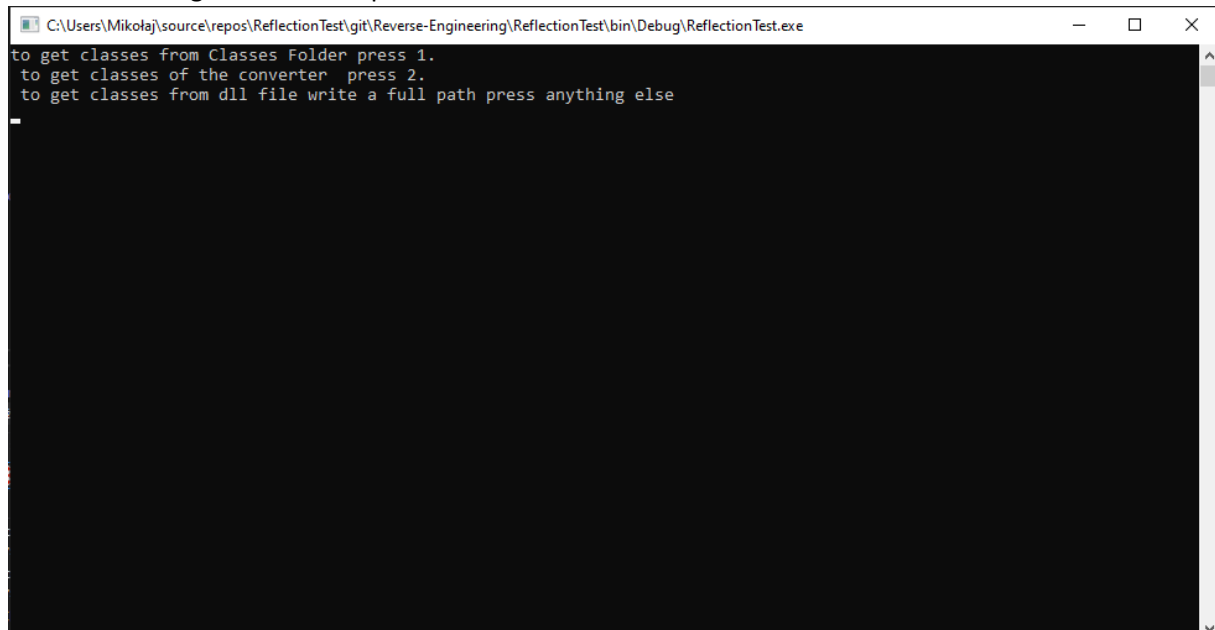
Accessibilities, Modifiers – typy wyliczeniowe zawierające dostępność i dodatkowe modyfikatory typu static, abstract itp.

ClassTypes- typ klasy

ConnectionTypes- typ związków.

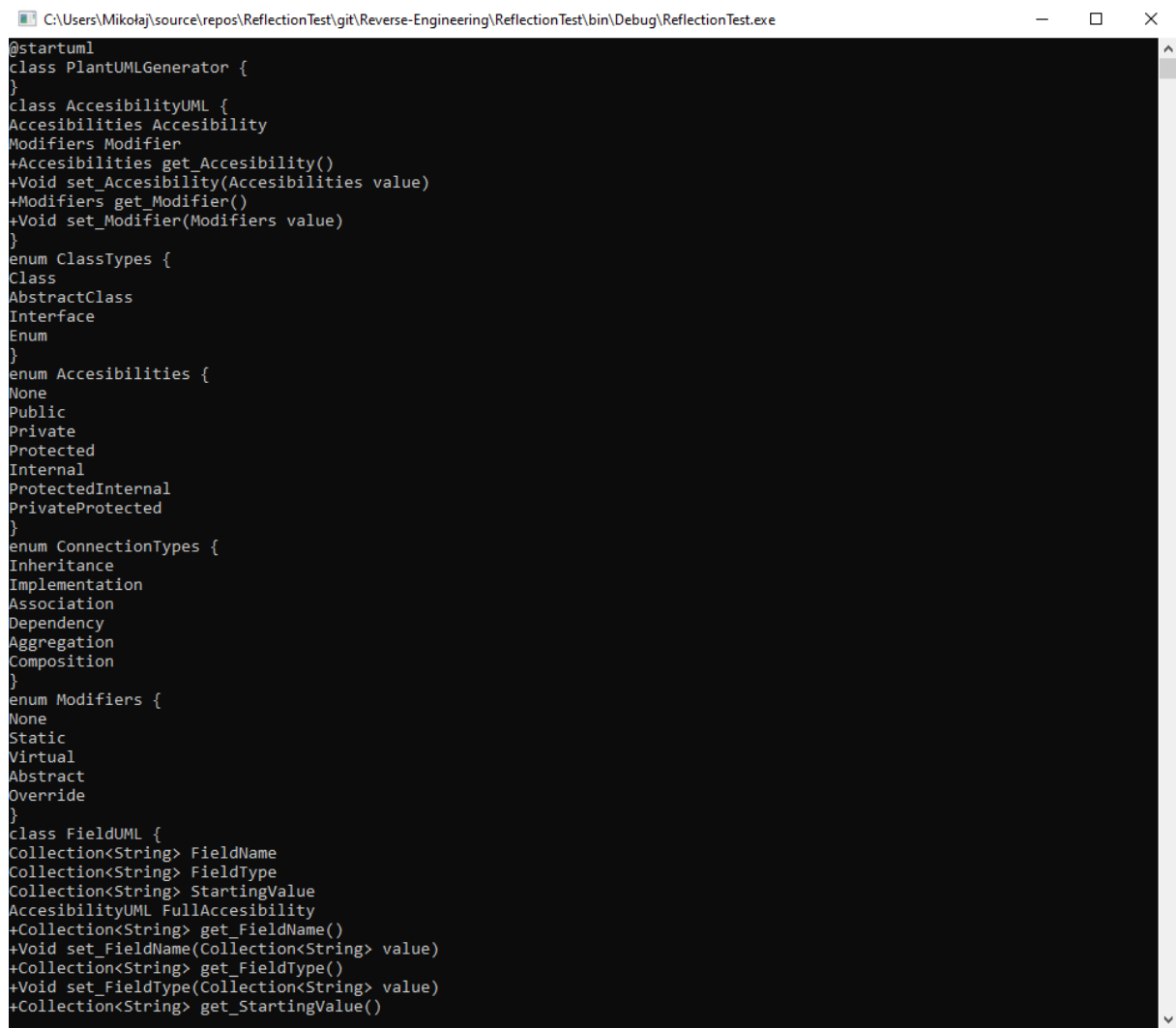
#### DZIAŁANIE PROGRAMU:

Ekran początkowy. Można wybrać generowanie klas z folderu classes, generowanie schematu convertera lub generowanie z pliku dll.



```
C:\Users\Mikołaj\source\repos\ReflectionTest\git\Reverse-Engineering\ReflectionTest\bin\Debug\ReflectionTest.exe
to get classes from Classes Folder press 1.
to get classes of the converter press 2.
to get classes from dll file write a full path press anything else
_
```

Następnie po podaniu poprawnej ścieżki lub pozostałych opcji zostanie wygenerowany plik języka plantUML.



```
@startuml
class PlantUMLGenerator {
}
class AccesibilityUML {
    Accesibilities Accesibility
    Modifiers Modifier
    +Accesibilities get_Accesibility()
    +Void set_Accesibility(Accesibilities value)
    +Modifiers get_Modifier()
    +Void set_Modifier(Modifiers value)
}
enum ClassTypes {
    Class
    AbstractClass
    Interface
    Enum
}
enum Accesibilities {
    None
    Public
    Private
    Protected
    Internal
    ProtectedInternal
    PrivateProtected
}
enum ConnectionTypes {
    Inheritance
    Implementation
    Association
    Dependency
    Aggregation
    Composition
}
enum Modifiers {
    None
    Static
    Virtual
    Abstract
    Override
}
class FieldUML {
    Collection<String> FieldName
    Collection<String> FieldType
    Collection<String> StartingValue
    AccesibilityUML FullAccesibility
    +Collection<String> get_FieldName()
    +Void set_FieldName(Collection<String> value)
    +Collection<String> get_FieldType()
    +Void set_FieldType(Collection<String> value)
    +Collection<String> get_StartingValue()
}
```

<https://plantuml.com/class-diagram>

Należy wkleić kod i nacisnąć przycisk submit.

The screenshot displays the PlantUML online editor. The code editor contains the following PlantUML diagram:

```

classDiagram
    classnameuml
    AccessibilityUML <-- ClassnameUML
    ClassUML <-- CodeToObjectConverter
    ConnectionUML <-- CodeToObjectConverter
    CodeToObjectConverter <-- ObjectToPlantUMLConverter
    ConnectionUML <-- ObjectToPlantUMLConverter
    ConnectionTypes <-- ObjectToPlantUMLConverter
    ClassUML <-- ObjectToPlantUMLConverter
    MethodUML <-- ObjectToPlantUMLConverter
    ClassTypes <-- ObjectToPlantUMLConverter
    AccessibilityUML <-- ObjectToPlantUMLConverter
    Accessibilities o-- AccessibilityUML
    Modifiers o-- AccessibilityUML
    AccessibilityUML o-- FieldUML
    ClassnameUML o-- ClassUML
    FieldUML o-- ClassUML
    MethodUML o-- ClassUML
    ConnectionTypes o-- ConnectionUML
    ParameterUML o-- MethodUML
    AccessibilityUML o-- MethodUML
    ClassTypes o-- ClassnameUML
    AccessibilityUML o-- ClassnameUML
    ClassUML o-- CodeToObjectConverter
    ConnectionUML o-- CodeToObjectConverter
    CodeToObjectConverter o-- ObjectToPlantUMLConverter
    enduml
  
```

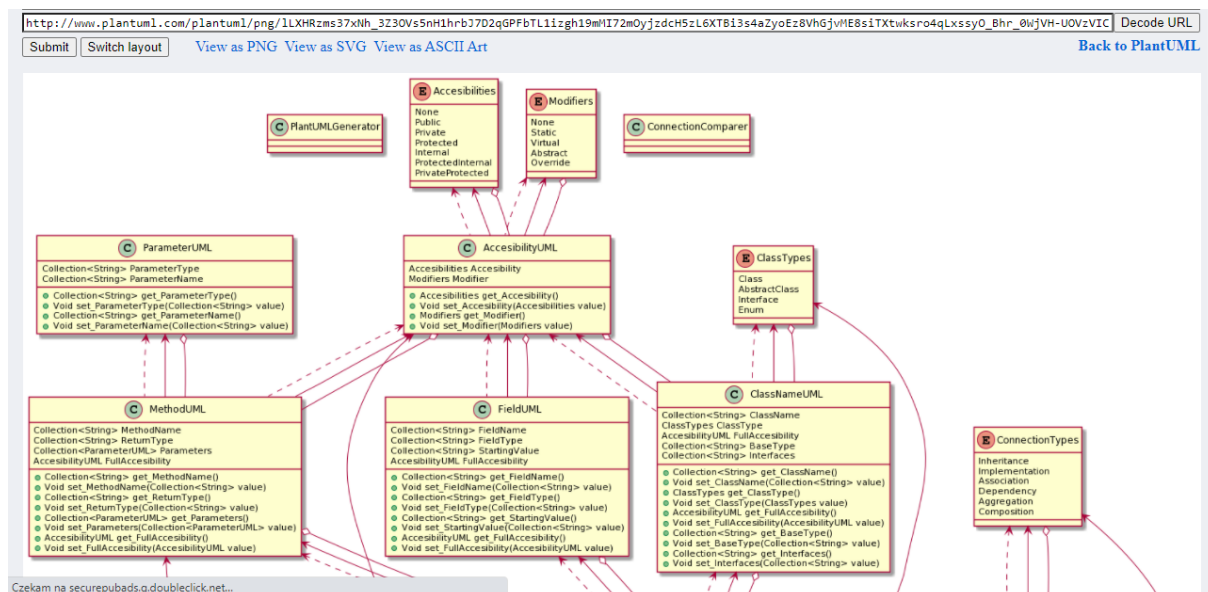
Below the code editor is a URL bar with the address: [http://www.plantuml.com/plantuml/png/LP1831C430NtFvNaghr3LUKcaGj4GG8ZGfVHsx581y8RpVv3ybe4HSUK0z56AqOpShZ02EH0ImoVnYmGA0jAaAIJd546j6Rd1RmU9efJH\\_bo2pRdZc1l2Ku](http://www.plantuml.com/plantuml/png/LP1831C430NtFvNaghr3LUKcaGj4GG8ZGfVHsx581y8RpVv3ybe4HSUK0z56AqOpShZ02EH0ImoVnYmGA0jAaAIJd546j6Rd1RmU9efJH_bo2pRdZc1l2Ku). To the right of the URL bar is a "Decode URL" button.

At the bottom, there are buttons for "Submit", "Switch layout", and "View as PNG View as SVG View as ASCII Art". To the right of these buttons is a "Back to PlantUML" link.

The preview area shows the generated diagram. It includes a class hierarchy with nodes for "abstract", "abstract class", "annotation", "circle", "circle\_short\_form", and "class". The "class" node is highlighted with a green circle.



Wygenerowany zostanie obrazek który będzie można pobrać/obejrzeć w formie PNG lub SVG.



Wnioski:

Projekt nauczył mnie przede wszystkim możliwości jakie daje refleksja w języku c#. Nauczyłem się również korzystanie języka plantUML oraz odświeżyłem swoją wiedzę na temat diagramów klas.