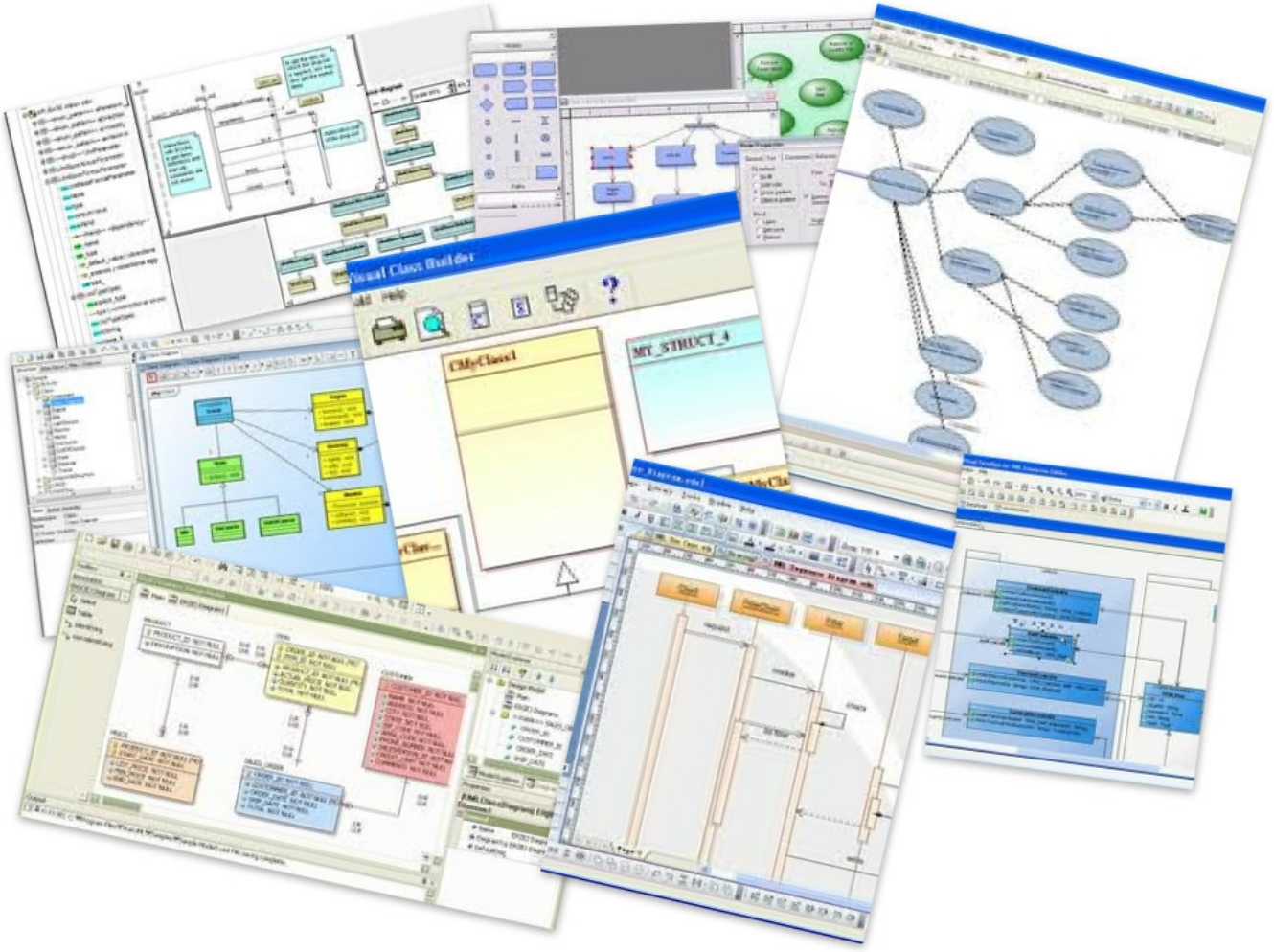
Omelette - UML dla programistów

Sławomir Blatkiewicz Jakub Górniak Piotr Piechal
Bartosz Pieńkowski Barnaba Turek Michał Zochniak

14 czerwca 2011

Omelette
UML dla programistów

Dostępne oprogramowanie



Alternatywa w postaci opisu tekstowego

```
class Student
  – imie
  – nazwisko

class Uczelnia
  – nazwa

association
        source-object: Student
        target-object: Uczelnia
```

Wynikowy diagram

Główne zalety

- Szybkość modelowania
- Minimalizm składni
- Rozszerzalność języka
- Modularna architektura klas rysujących

Wykorzystane technologie – Python

Wykorzystane technologie – Qt

Wykorzystane technologie – cd.

- Python
- Qt
- pyQt
- pyparsing
- graphviz

Omelette
UML dla programistów

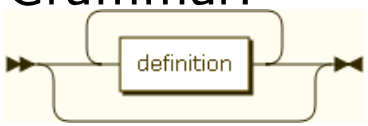
Opis formalny

```
grammar ::= (definition)*

definition ::= ('\n')* header (element | '\n')*

header ::= ( | prototype) parent_name ( | object_name) '\n'

element ::= (operation | attribute | property | constraint) '\n'

attribute ::= ( | static) visibility attribute_name ( |(':' attribute_type))
    ( |('=' attribute_default))

operation ::= ( | static) visibility method_name '(' ( | parameters) ')'
    ( |(':' return_type))

property ::= property_name ':' property_value

constraint ::= constraint_type constraint_key constraint_value
```

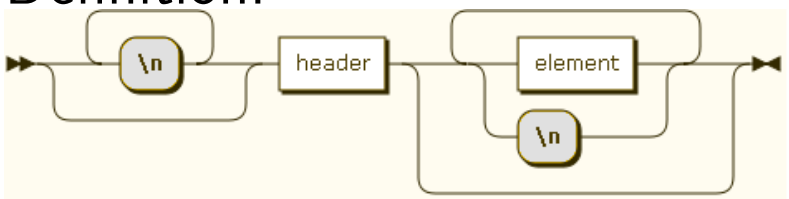
Przykładowy rozbior

```
dziewczynka ala
    + kot : String = 'buras'
    + idz(gdzie : String) : void
```

Grammar:



Definition:



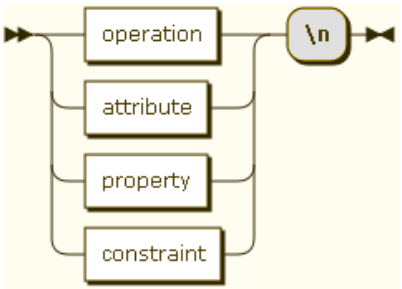
Header:

grammar $\longrightarrow$ definition $\longrightarrow$ parent_name name $\longrightarrow$ 'dziewczynka' 'ala'

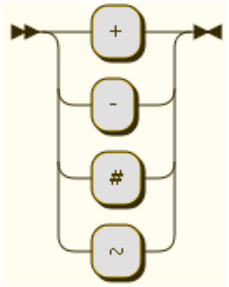
Przykładowy rozbior 2

```
dziewczynka ala
    + kot : String = 'buras'
    + idz(gdzie : String) : void
```

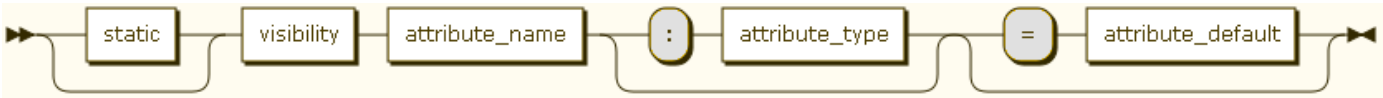
Element:



Visibility:



Attribute:



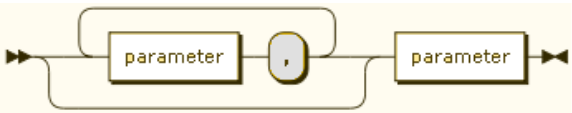
element → attribute → visibility ':' type '=' attribute_value → '+' 'kot' ':' 'String' '=' 'buras'

Przykładowy rozbior 3

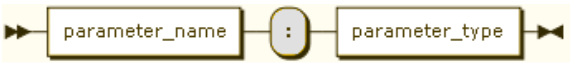
```
dziewczynka ala
    + kot : String = 'buras'
    + idz(gdzie : String) : void
```

Operation:

Parameters:



Parameter:



element $\longrightarrow$ operation $\longrightarrow$ visibility method_name '(' parameters ')' ':' return_type

$\longrightarrow$ 'idz' '(' parameter ')' ':' void' $\longrightarrow$ 'idz' '(' parameter_name ':' parameter_type ')' ':' 'void'

$\longrightarrow$ 'idz' '(' 'gdzie' ':' 'String' ')' ':' 'void'

Omelette
UML dla programistów

Przesłanki

- Biblioteka standardowa
- Powszechność cech wspólnych pomiędzy konstrukcjami
- Podobne obiekty na diagramie

Biblioteka standardowa

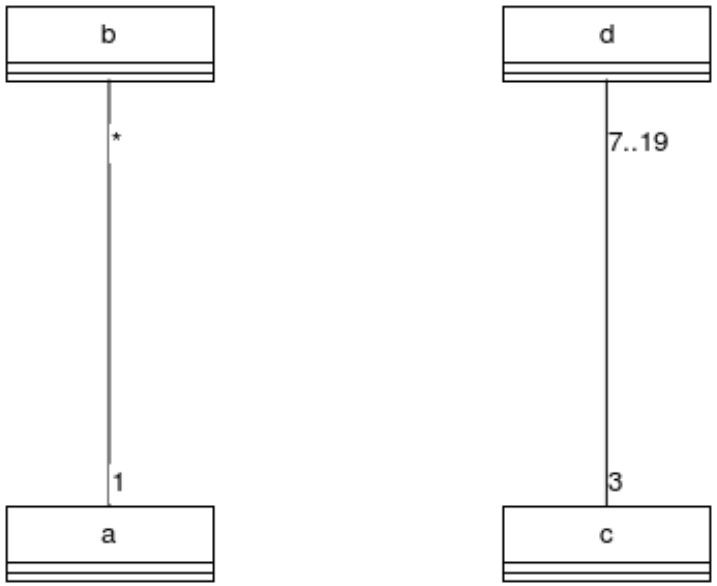
Biblioteka w kodzie

- Łatwe wczytywanie

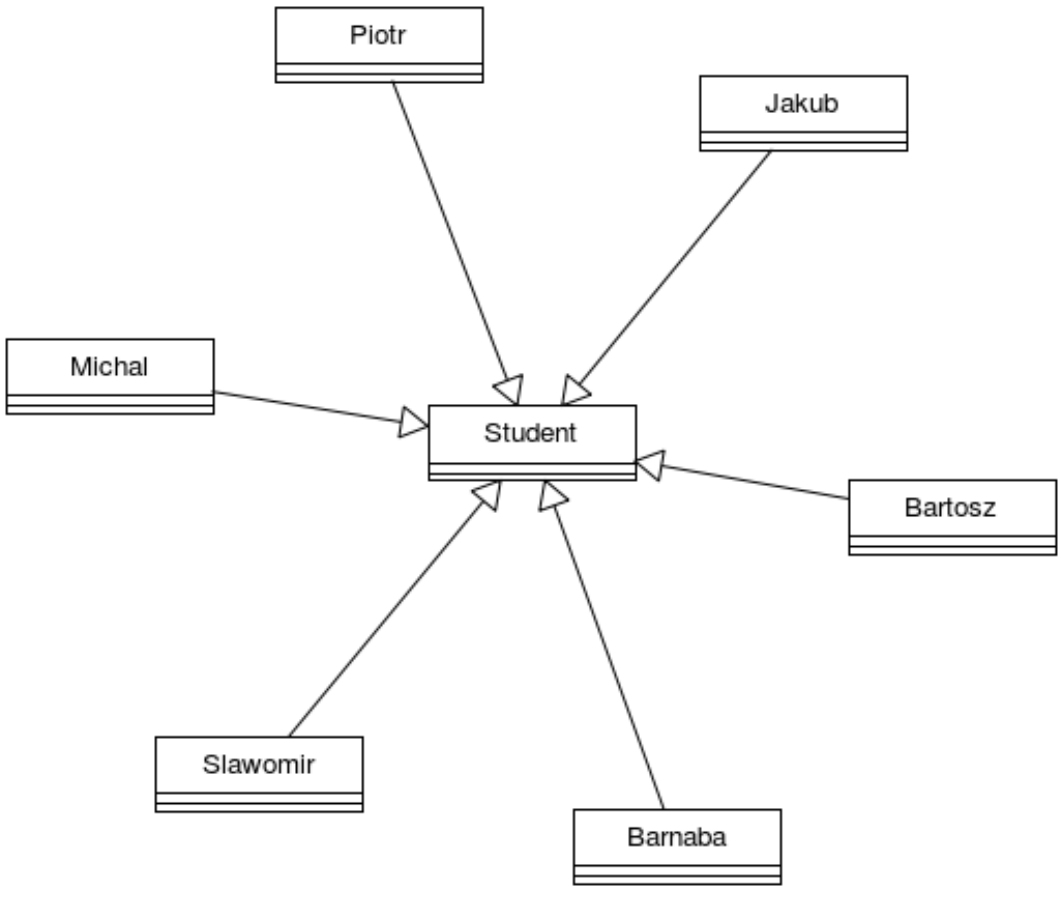
Oddzielna biblioteka

- Łatwa modyfikacja
- Łatwe rozszerzanie
- Łatwe wykorzystanie

Powszechność cech wspólnych pomiędzy konstrukcjami



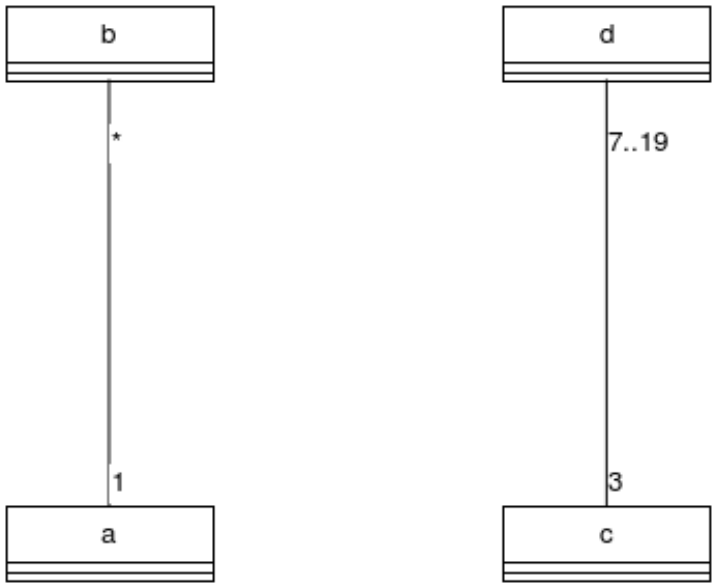
Podobne obiekty na diagramie



Podobne obiekty na diagramie - implementacja

```
generalization student
    source-object : Barnaba
    target-object : Student

student
    source-object : Michal

student
    source-object : Bartosz
...
```

Cechy wspólne konstrukcji - przypomnienie



Cechy wspólne konstrukcji - implementacja

```
prototype relation jeden-do-wielu
  source-count  : 1
  target-count  : *

jeden-do-wielu
  source-object : a
  target-object : b
```

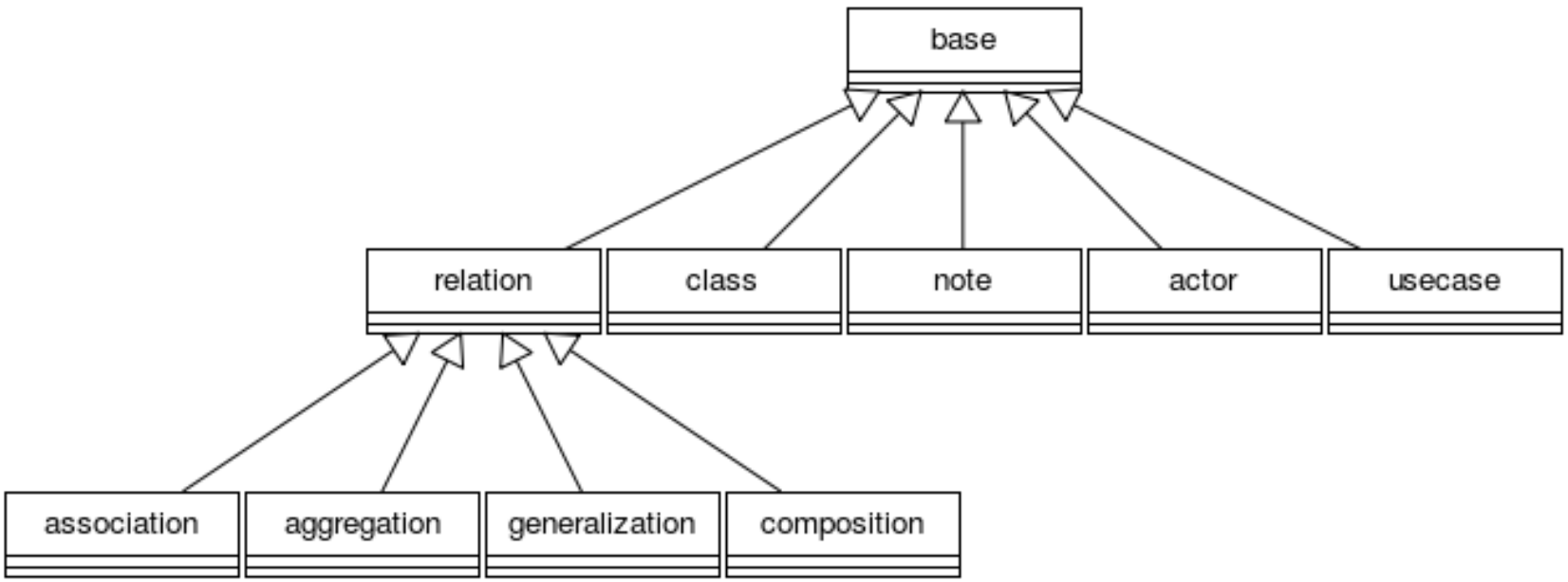
Dane walidacji

Możliwości

Możemy **wymusić** ustawienie jakiegoś klucza, podać listę **dozwolonych** dla tego klucza wartości, lub **zabronić** używania tego klucza.

```
prototype base relation
    allow name STRING

    allow arrow STRING
    allow direction [none, source, target, both]

    require source-object OBJECT
    allow source-count MULTIPLICITY
    ...
```

Struktura biblioteki



Błędy związane z prototypami

- Cykliczne zależności
- Nieistniejące zależności
- Błędy walidacji
  - Brak wymaganego klucza
  - Zły typ wartości klucza
  - Niedozwolona wartość klucza
  - Referencja do nieistniejącego obiektu

Omelette
UML dla programistów

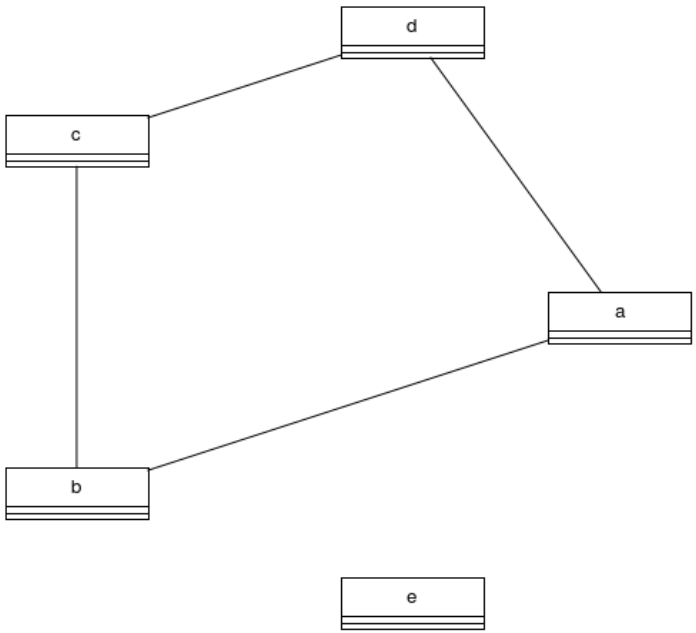
Układ diagramów

- Co to jest układ diagramów?

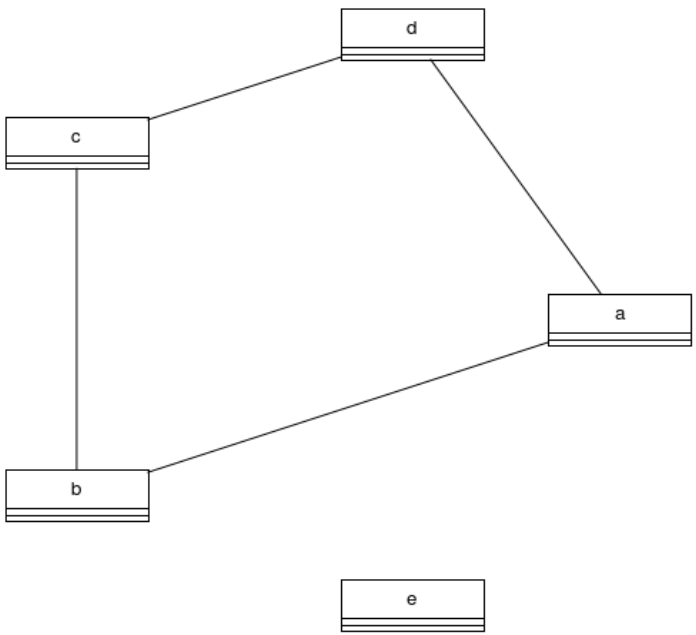
Układ diagramów

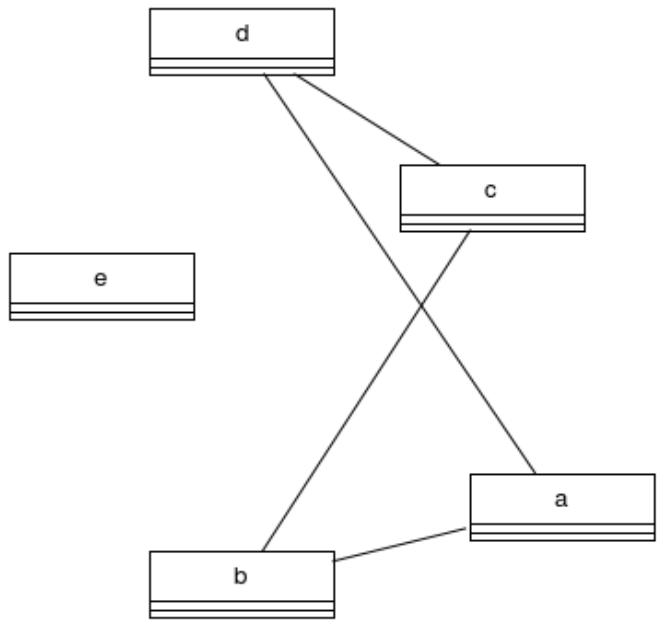
- Co to jest układ diagramów?
- Dlaczego jest taki ważny?

Układ diagramów



Układ diagramów





Układ diagramów

- Co to jest układ diagramów?
- Dlaczego jest taki ważny?
- Czym tak naprawdę jest diagram?

Wybrane algorytmy grafowe

- Planaryzacja
- Rozkład kołowy
- Metody energetyczne

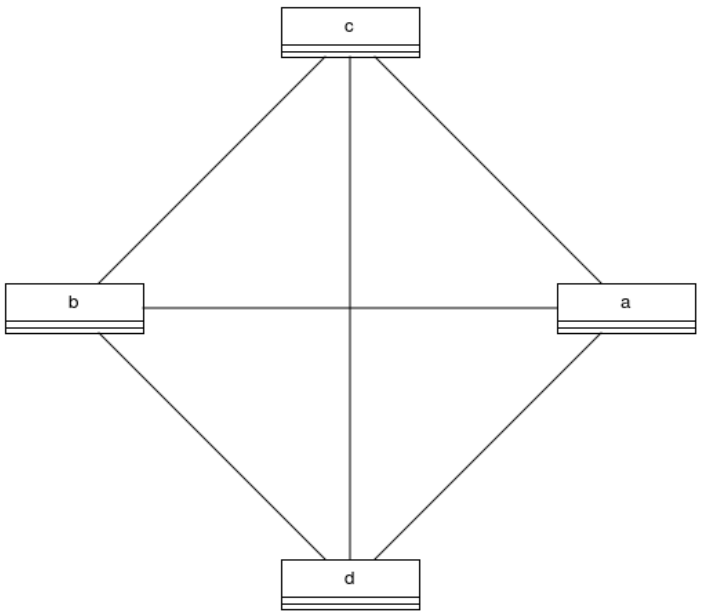
Planaryzacja

- Co znaczy planarny?

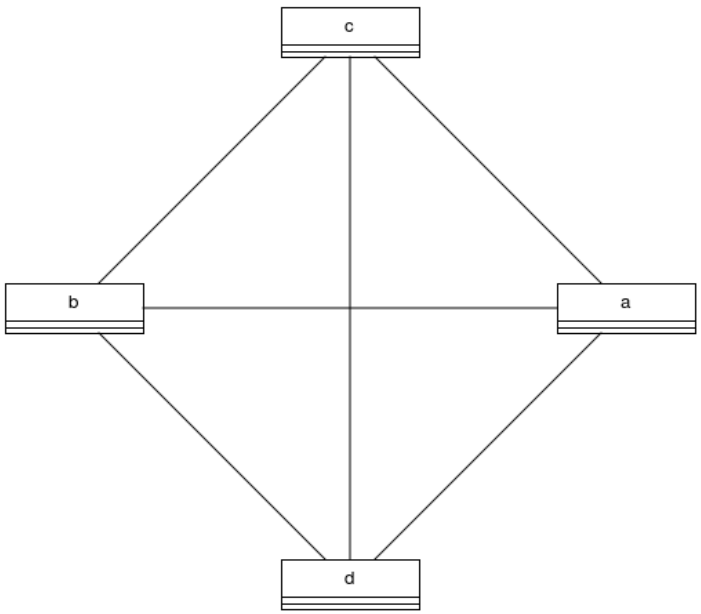
Planaryzacja

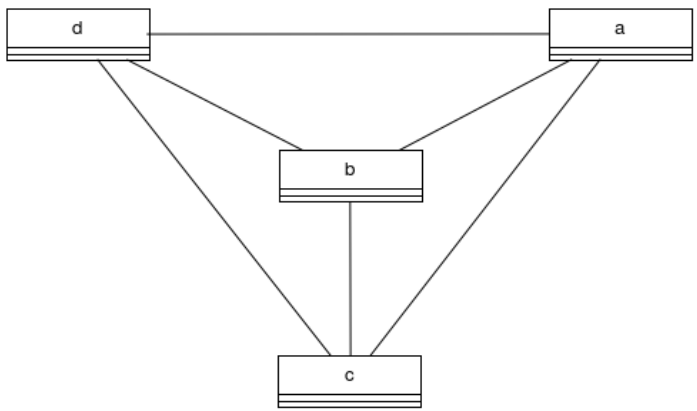
- Co znaczy planarny?
- Czym więc jest planaryzacja?

Planaryzacja



Planaryzacja

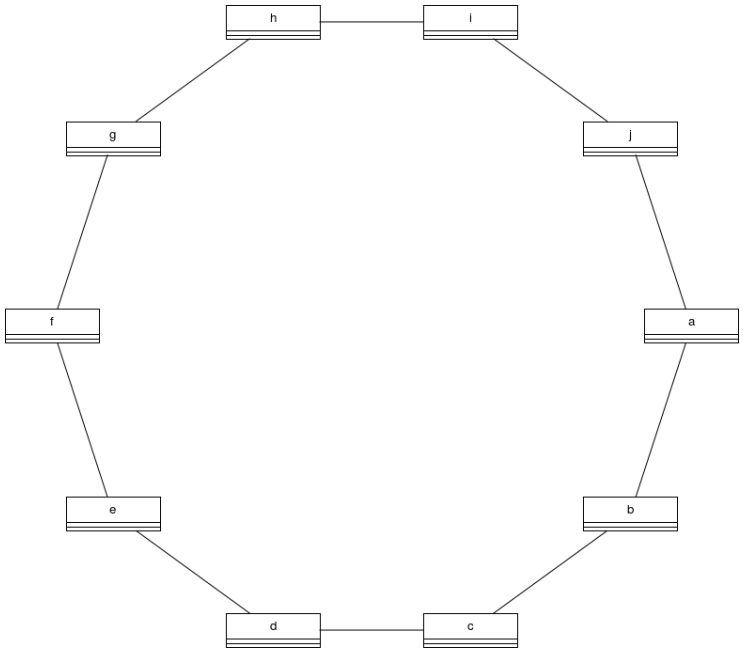




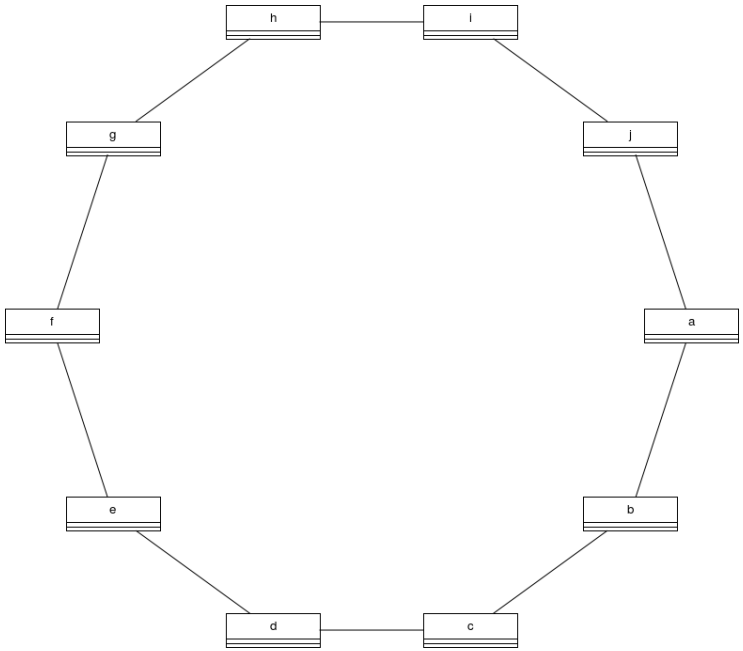
Algorytmy kołowe

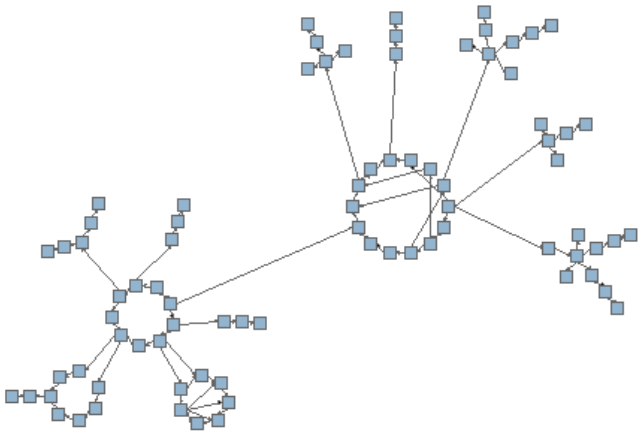
- Jak działają?

Algorytmy kołowe



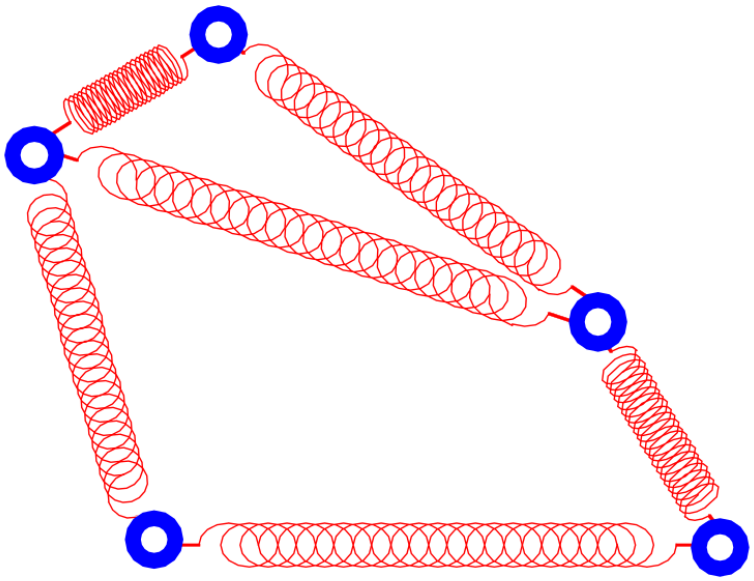
Algorytmy kołowe





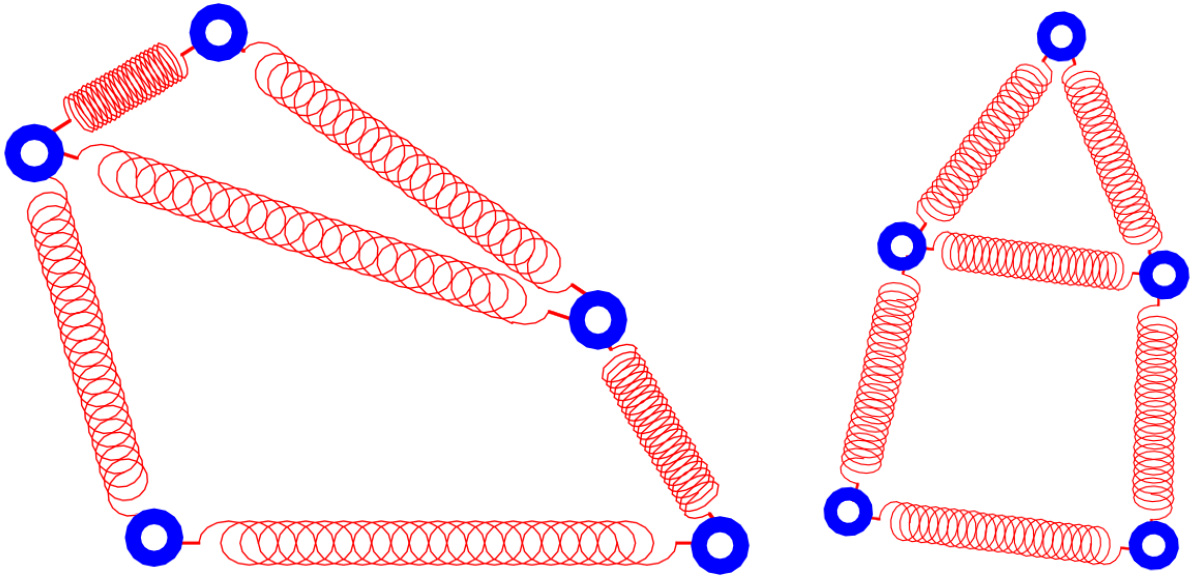
Algorytmy energetyczne

- Model mechaniczny



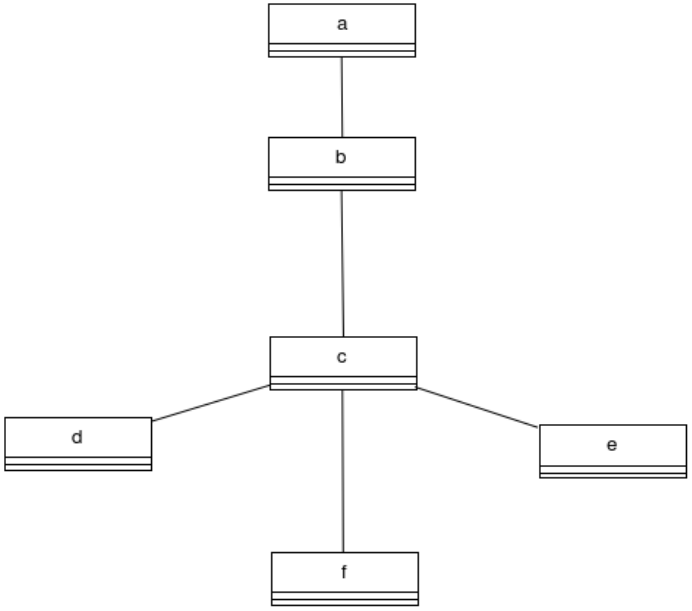
Algorytmy energetyczne

- Model mechaniczny



Algorytmy energetyczne

- Model mechaniczny



Algorytmy energetyczne

- Modyfikacje

Algorytmy energetyczne

- Modyfikacje
  - Kamada-Kawai

Algorytmy energetyczne

- Modyfikacje
  - Kamada-Kawai
  - Fruchterman-Reingold

Algorytmy energetyczne

- Modyfikacje
  - Kamada-Kawai
  - Fruchterman-Reingold
  - Hadany-Harel

Algorytmy energetyczne

- Modyfikacje
  - Kamada-Kawai
  - Fruchterman-Reingold
  - Hadany-Harel
  - Harel-Koren

Algorytmy energetyczne

- Modyfikacje
  - Kamada-Kawai
  - Fruchterman-Reingold
  - Hadany-Harel
  - Harel-Koren
  - Model grawitacyjny

Graphviz

- Co to jest?

Graphviz

- Co to jest?
- Dlaczego o nim mówię?

Omelette
UML dla programistów

Demonstracja

Omelette
UML dla programistów

Podsumowanie

github.com/pienkowb/omelette