JPL dwukierunkowy interfejs Prolog/Java

Agnieszka Jamróg, Michał Szady

PWSZ Tarnów, IV rok IS

7 lutego 2009

JPL jest biblioteką umożliwiającą dwukierunkową komunikację pomiędzy Javą a Prologiem, co oznacza możliwość wbudowania Javy w Prolog jak również Prolog w Javę.

James Gosling czyli autor Javy wychodził z założenia, że jego język służył będzie do obsługi lodówki czy ekspresu do kawy. Jednak jego zamysł nie znalazł ujścia w rzeczywistości, ponieważ szybko okazało się, że postęp w dziedzinie artykułów gospodarstwa domowego nie jest na tyle szybki na ile liczył Gosling.
Projekt Goslina nazwany miał być początkowy Oak(ang. Dąb), ale nazwa była już zajęta, co nie było dziwne, jeśli wziąć pod uwagę, że istnieje kilkadziesiąt tysięcy języków programowania.

Ostatecznie język został nazwany Java na część jednego z gatunków kawy, której Gosling był miłośnikiem. Java w rezultacie znalazła swoje zastosowanie przy tworzeniu aplikacji użytkowych na jjak największą ilość platform. Język ten ma być prosty dla programisty. Autor swierdził kiedyś, że jeśli program w Javie da się uruchomić to z reguły działa .

1. JPL jest to zestaw klas oraz C-funkcji dostarczających interfejsu pomiędzy Javą, a Prologiem.
2. Do połączenia Javy z silnikiem Prologa JPL używa Java Native Interface (JNI) oraz Prologowego Foreign Language Interface (FLI).
3. JPL nie jest klasyczną implementacją Javy w Prologu, pracuje tylko z SWI - Prologiem.
4. JPL został zaprojektowany w dwóch warstwach:
   - warstwa interfejsu niskiego poziomu(Low-Level-Interface)służąca do FLI,
   - warstwa interfejsu wysokiego poziomu(High-Level-Interface) dla programistów.

- jest w pełni dynamiczy,
- jpl_call automatycznie przeładowuje wywoływane metody Javy,
- współdziała z API Javy biblioteki JPL,
- wymagane jest środowisko uruchomieniowe Javy w wersji nie starszej niż 1.4,
- API Javy podobne jest do XPCE, czyli GUI Prologa,
- referencje do obiektów Javy powinny być traktowane jako uchwyty do obiektów

- kiedy wywoływane są w Prologu metody Javy, które nic nie zwracają, w prologu zwracana jest pusta wartość,
- zaimplementowany w jednym module - jpl.pl (zawiera on sporych rozmiarów kod Prologa),
- metody klas Javy powinny zaczynać się od 'JPL',
- część Java-calls-Prolog(wywołanie Prologa z poziomy Javy) oraz Prolog-calls-Java(wywołanie Javy z poziomu Prologa) są od siebie niezależne.
- JPL obsługuje wszystkie podstawowe typy Javy w formie Termów,

Prolog zawiera następujące typy Javy podzielone według 3 notacji.

Notacja 'structured'

void jest reprezentowany przez **void**
null jest reprezentowany przez **null**

typy elementarne reprezentowane są przez:
boolean, char, byte, short, int, long, float, double

klasa jest reprezentowana przez **class**
tablica jest reprezentowana przez **array**

Notacja 'desriptor'

Przykłady:

Z oznacza **boolean**
B oznacza **byte**
C oznacza **char**
S oznacza **short**
I oznacza **int**
J oznacza **long**
F oznacza **float**
D oznacza **double**
Ljava/util/Date ; oznacza **java.util.Date**
[typ oznacza **tablicę**

Notacja 'classname'

Przykłady:

java.util.Vector oznacza klasę Javy **java.util.Vector**

[B oznacza **tablicę** typu **boolean**

[Ljava.lang.String; oznacza **tablicę** typu **string**

Abyśmy mogli korzystać w Prologu z Javy do tego celu wykorzystujemy 4 podstawowe predykaty:

jpl_new(+Class, +Params, -Result) - tworzy nowy obiekt klasy Javy

Class może być:

- odpowiednim typem: klasą lub tablicą,
- pełną ścieżką do klasy (np. 'java.lang.String'),
- obiektem klasy ('class([java,lang],['Class'])'),

`Odpowiednik w Javie:`
Class Result Class(Params)

1. Jeśli **Class** jest tablicą i **Params** jest wartością dodatnią to **Result** jest nową tablicą o rozmiarze **Params**.

2. Jeśli **Class** jest tablicą i **Params** jest listą danych o tym samym typie elementów jak w tablicy **Class**, to **Result** jest nową tablicą posiadającą tyle samo elementów ile jest danych w liście i jest zainicjalizowana tymi samymi wartościami.

*jpl_call(+Object, +Method, +Params, -Result) - **wywołuje metodę Javy***

Object może być:

- typem,
- egzemplarzem klasy lub tablicą,

Method jest:

- metodą obiektu Object,

Params to:

- parametry metody oddzielone przecinkami.

`Odpowiednik w Javie:`

Result = Object.Method(Params)

jpl_set(+Object, +Field, +Value) - ustawia wartości obiektów lub klas Javy

Object może być:

- obiektem klasy,
- egzemplarzem klasy,
- tablicą,
- nie może być stringiem.

Field może być:

- zmienną,
- indeksem tablicy, musi być listą wartości: **Object[I-J]** będzie przypisana **Value**).

jpl_get(+Object, +Field, -Value) - pobiera wartości obiektów lub klas Javy

Object może być:

- obiektem klasy,
- egzemplarzem klasy,
- tablicą,
- nie może być stringiem.

Field może być:

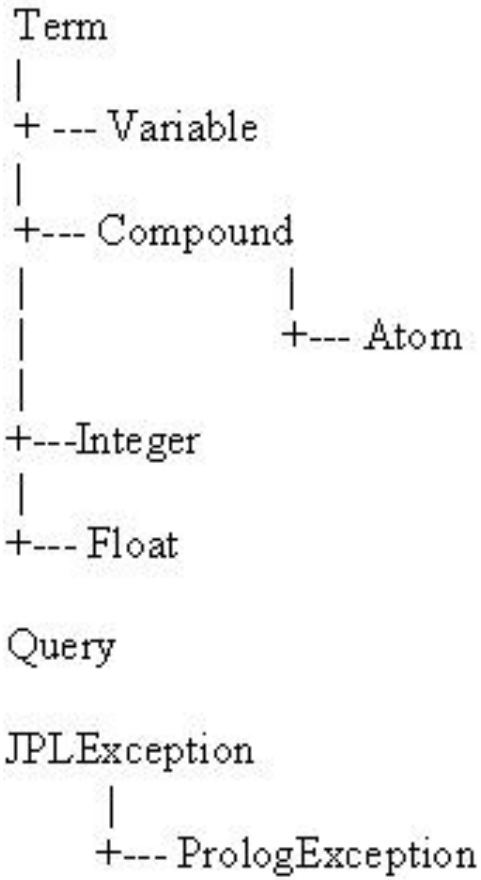
- zmienną,
- indeksem tablicy,

Więcej predykatów i ich opisów w języku angielskim można znaleźć na stronie:

`http://www.swi-prolog.org/packages/jpl/prolog_api/api.html`

Chcąc wykorzystywać Prologa w Javie musimy bliżej przyjrzeć się interfejsowi wysokiego poziomu (ang. High- Level- Interface). JPL Java- calls- Prolog API zawiera prawie cały interfejs niskiego poziomu, a mimo to jest bardziej przejrzysty i elastyczny.

Hierarchia klas:



Termy są klasą abstrakcyjną.
Każde **zapytanie** zawiera Term (określa cel, który ma zostać sprawdzony).
Na każdy **Compound** składa się nazwa (Java.lang.String) oraz tablica argumentów (Termów) musi zawierać przynajmniej jeden.
Atom jest specyficznym Compound z zerową listą argumentów.

INICJALIZACJA PROLOGA

Klasa jpl.JPL inicjalizuje Prolog VM (np. libpl.dll w Win32). Poniżej są omówione niektóre metody.

getDefaultInitArgs

public static java.lang.String[] getDefaultInitArgs()

Zwraca tablicę stringów w kolejności argumentów linii poleceń w jakiej były używane. Zwraca null, gdy Prolog VM jest właśnie inicjalizowany (w tym przypadku *domyślne* init args nie są stosowane, więc w obszarze zainteresowania są *aktualne* init args). Podsumowywując zwracane są domyślne zainicjalizowane argumenty, lub null jeśli inicjalizacja trwa.

setDefaultInitArgs

public static void setDefaultInitArgs(java.lang.String[] args)

Parametrem funkcji jest tablica Stringów. Argumenty jej są ułożone w takiej kolejności w jakiej później zostaną zainicjalizowane.
Parametr: args nowe domyślnie zainicjalizowane argumenty

getActualInitArgs

public static java.lang.String[] getActualInitArgs()

Zwraca tablicę Stringów, elementy są ułożone w kolejności aktualizowania ich. Zwraca null, jeśli silnik Prologa nie został zainicjalizowany.
Funkcja jest przydatna do testów, zwraca faktycznie zainicjalizowane argumenty.

Metody te pozwalają pracować JPL bez dodatkowego obciążenia.
JPL nie inicjalizuje, a jedynie sprawdza dzięki powyższym metodom czy Prolog VM został zainicjalizowany.
Wyraźnie jest to widoczne w JPL 1.0.1.

Init

public static boolean init()

Metoda inicjalizuje domyślne parametry i zwraca true lub false (gdy inicjalizacja nie została ukończona).

public static boolean init(java.lang.String[] args)

Do inicjalizacji używane są przekazane argumenty typu String.
Metodę wywołuje się aby:

- zainicjować silnik Prologa z parametrami innymi niż domyślne,
- wykonać automatycznie pierwsze zapytanie.

Metoda ta musi być wykonana przed wywołaniem zapytań!
Parametr args inicjalizowany parametr listy.

Atoms

Atom jest to stała napisowa. Aby stworzyć atom należy podać jego nazwę (String) do konstruktora.

Atom aristotle = New Atom(aristotle);
Atom alexander = new Atom(alexander);
Klasa Atom dziedziczy z Compound name().

Variables

posiadają identyfikator, który musi być zgodny z syntaktyką Prologa.

Variable X = new Variable("X"); // konkretna zmienna X
Variable X = new Variable("_"); // anonimowa zmienna
Variable X = new Variable("_Y"); // nieznana zmienna; której nie chcemy znać

Integers

specyficzny rodzaj Termu, który przechowuje długie zmienne Javy. Odpowiednikiem w Prologu tej klasy jest typ integer.

jpl.Integer i = new jpl.Integer(5); // tworzenie nowej zmiennej typu Integer

Floats

- przechowuje zmienne Javy typu Double. Pod taką samą nazwą istnieje odpowiednik w Prologu.

jpl.Float f = new jpl.Float(3.14159265); // tworzenie nowej zmiennej typu Float

Compounds

Term składający się z nazwy i sekwencji Termów (realizowanych za pomocą listy).

```
Compound teacher_of = new Compound(
"teacher_of",
new Term[] {
new Atom(äristotle"),
new Atom(älexander") } );
```

Queries / Zapytania

zapytania zawierają Termy, reprezentujące cel Prologa:

Term goal = new Compound("teacher_of", new Term[]new Atom(ąristotle"),new Atom(ąlexander"));
Query q = new Query(goal);

W Prologu wyglądałoby to tak:

?- teacher_of(aristotle,alexander).

Util Class

dostarcza różne statyczne metody służące do zarządzania Termami JPL, np.:

Term termArrayToList(Term t[])

Term[] listToTermArray(Term t)

Term[] bindingsToTermArray(Hashtable bs)

W programach, które są umieszczone jako przykłady, wykorzystana jest klasa Query oraz jej metody:

public Hashtable oneSolution();

Gdy nie ma rozwiązań, metoda zwraca null, w przeciwnym razie zwraca rozwiązanie. Jeśli nie ma zmiennych, zwracany jest pusty obiekt Hashtable.

public Hashtable[] allSolutions();

Metoda allSolutions() zwraca wszystkie rozwiązania w postaci tablicy. Jeśli nie ma rozwiązań, metoda zwraca pustą tablicę.

Czasami potrzebujemy jedynie wiedzieć czy istnieje rozwiązanie. Wówczas przydatna jest metoda:

```
public boolean hasSolution();
```

Podobnymi metodami są:

```
public boolean hasMoreElements();
public Object nextElement();
```

Więcej metod i ich opisów w języku angielskim można znaleźć na stronie:

http://www.swi-prolog.org/packages/jpl/java_api/javadoc/jpl/JPL.html

Informacje w tym dokumencie był pisany na podstawie:

textithttp://www.swi-prolog.org/packages/jpl/java_api/high-level_interface.html

- SWI-Prolog w wersji 3.1.0 lub wyższej, dostępny na stronie `http://www.swi-prolog.org/`
- Środowisko Javy: `http://java.sun.com/`
- JPL: `http://sourceforge.net/projects/jpl/`

KONFIGURACJA ŚRODOWISKA

Należy ściągnąć plik jpl.zip i umieścić go np. w folderze /pl/jpl. Po rozpakowaniu powinniśmy mieć taką strukturę:

```
jpl
 +--- examples
 |     +--- Exceptions
 |     +--- Exceptions2
 |     +--- Family
 |     +--- Test
 |     +--- Test2
 |     +--- Time
 |     +--- Zahed
 |     +--- (and maybe more...)
 |
 +--- docs  (HTML files in here are accessible via links from the home page)
 |
 +--- src
 |     +--- c
 |     |    +--- build.bat  (Windows script to recompile jpl.c to jpl.dll)
 |     |    +--- build.sh  (Linux script to recompile jpl.c to libjpl.so)
 |     +--- java
 |
 +--- jpl.dll  (a native library - for Windows in this case)
 |
 +--- jpl.jar  (a Java library)
 |
 +--- jpl.pl  (a Prolog library)
 |
 +--- README.html  (JPL's documentation "home page")
```

Ważne są trzy pliki z bibliotekami: jpl.dll, jpl.jar i jpl.pl na które trzeba ustawić zmienne środowiskowe.
W Windows Vista:
Start -> Mój Komputer, właściwości -> Zaawansowane ustawienia systemu -> Zmienne środowiskowe
lub Kontrol Panel -> System -> Zmienne środowiskowe

- jp.dll, w zmiennych systemowych szukamy PATH i po średniku dopisujemy ścieżkę do folderu bin, np. C:/Program Files/pl/bin
- jpl.pl, w tym samym miejscu co PATH, zamiast Edit musimy nacisnąć New.. i stworzyć zmienną SWI_HOME_DIR, która będzie wskazywała na folder w którym został zainstalowany Prolog, np. C:/ProgramFiles/pl
- jpl.jar ustawiamy CLASSPATH porzez uruchomienie konsoli i wpisanie w niej: set CLASSPATH=%SWI_HOME_DIR%/lib/jpl.jar

WAŻNE

Kiedy wyskoczy Ci taki błąd w programie:

Exception in thread "main"java.lang.UnsatisfiedLinkError: C:/paul/bin/jpl.dll: Can't find dependent libraries

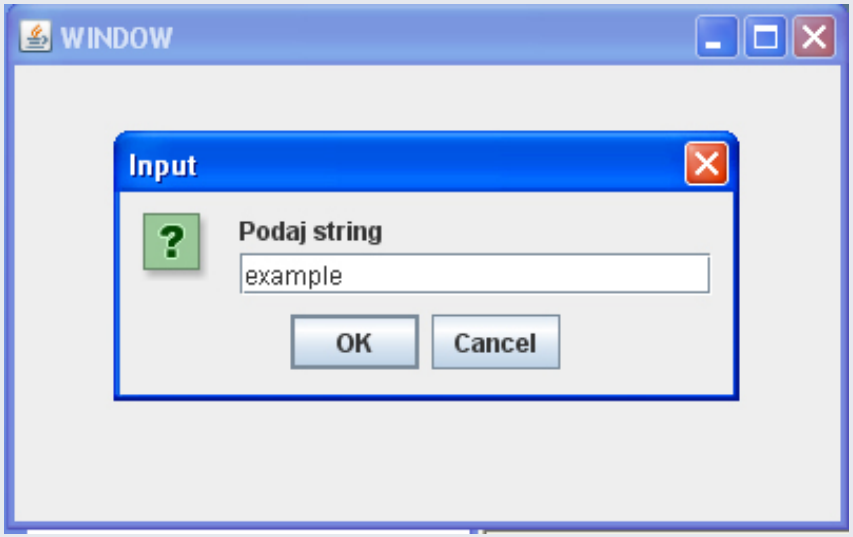
Sprawdź czy prawidłowo masz ustawioną zmienną PATH

C:/Program Files/pl/bin

KILKA PRZYKŁADÓW

Java - Prolog

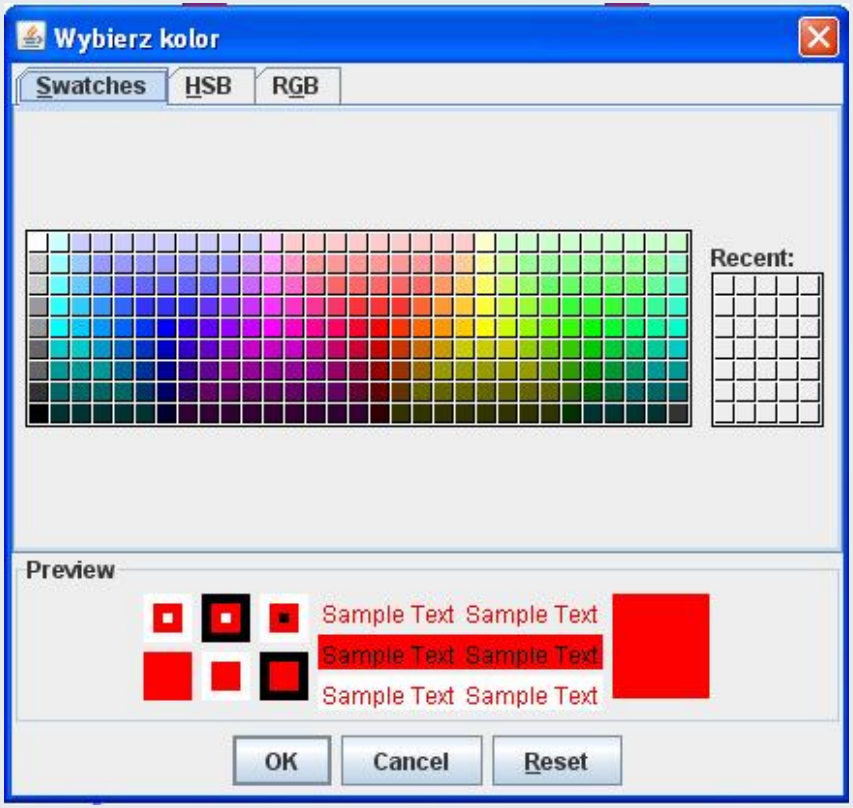
Okienko Javy w Prologu



```
:- use_module(library(jpl)).
//Załączenie biblioteki 'JPL';

jpl_string_entry :-

jpl_new( 'javax.swing.JFrame', ['WINDOW'], F),
//Pusta ramka 'WINDOW'

jpl_call( F, setLocation, [200,200], _),
jpl_call( F, setSize, [400,250], _),
jpl_call( F, setVisible, [@(true)], _),
jpl_call( F, toFront, [], _),
//Ustawianie właściwości obiektu 'F'


jpl_call( 'javax.swing.JOptionPane', showInputDialog, [F,'Podaj string'], N),
jpl_call( F, dispose, [], _),
//Tworzenie okna dialogowego


(jpl_is_null(N)
-> write( 'Anulowałeś')
; (N == '' 
-> write( 'Pole jest puste'),jpl_name_entry
; jpl_call( 'javax.swing.JOptionPane', showMessageDialog, [F,N], _)
) ),nl.
```

Końcowy efekt:

Okienko do wyboru koloru:



```
:- use_module(library(jpl)).
//Załączenie biblioteki 'JPL';

jpl_colour_choose_demo :-

jpl_new( 'javax.swing.JFrame', ['WINDOW'], F),
//Pusta ramka WINDOW

jpl_call( F, setLocation, [400,300], _),
jpl_call( F, setSize, [400,300], _),
jpl_call( F, setVisible, [@(true)], _),
jpl_call( F, toFront, [], _),
//Ustawianie właściwości obiektu F

jpl_get( 'java.awt.Color', red, Red),
//Pobranie koloru

jpl_call( 'javax.swing.JColorChooser', showDialog, [CP,'Wybierz kolor',Red], C),
//Uruchomienie okna dialogowego

jpl_call( F, dispose, [], _),
( C == @(null)
-> write( 'Anulowałeś')
; write( 'Wybrałeś '), write( C)
),
nl.
```

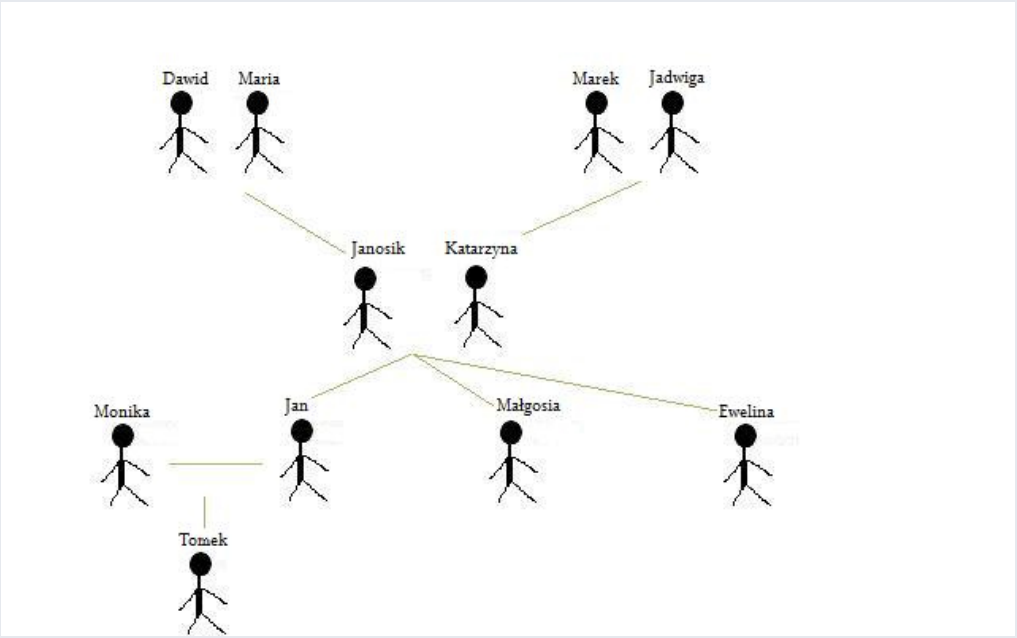
Prolog - Java

Klasyczny przykład rodziny. Plik napisany w prologu (rodzina.pl):

```
rodzic(dawid,janosik).
rodzic(maria,janosik).
rodzic(marek,katarzyna).
rodzic(jadwiga,katarzyna).
rodzic(janosik,jan).
rodzic(janosik,malgosia).
rodzic(janosik,ewelina).
rodzic(katarzyna,jan).
rodzic(katarzyna,malgosia).
rodzic(katarzyna,ewelina).
rodzic(monika,tomek).
rodzic(jan,tomek).


kobieta(maria).
kobieta(jadwiga).
kobieta(katarzyna).
kobieta(monika).
kobieta(malgosia).
kobieta(ewelina).
```

```
mezczyzna(dawid).
mezczyzna(marek).
mezczyzna(janosik).
mezczyzna(jan).
mezczyzna(tomek).


matka(X,Y) :-
rodzic(X,Y),
kobieta(X).
ojciec(X,Y) :-
rodzic(X,Y),
mezczyzna(X).


dziadkowie(X,Y) :-
rodzic(X,Z),
rodzic(Z,Y).
```

oraz drzewo genealogiczne wykorzystanej rodziny:



kod napisany w Javie (Rodzina.java)

```
import java.util.Hashtable;
import jpl.Query;

public class Rodzina {
public static void main(String[] args) {
// zapytanie podajemy w formie tekstowej / String
String tekst = ćonsult('rodzina.pl')";
// sprawdz plik family.pl / polacz sie z nim i czerp z niego informacje


// przy pomocy klasy Query wykonujemy nasze zapytanie
Query zapytanie = new Query(tekst);


// wypisanie wyniku naszego zapytania
System.out.println(tekst ++ (zapytanie.hasSolution()?ók":ńie udalo sie"));


// — przyklad nr 1 ——————————-
System.out.println(Śprawdz czy mama i tata Janosika sa podanymi osobami.");
String tekst2 = "matka(maria,janosik)";
Query zapyt2 = new Query(tekst2);
System.out.println(tekst2 ++ (zapyt2.hasSolution()?"prawda": "falsz"));
String tekst3 = ójciec(maria,janosik)";
Query zapyt3 = new Query(tekst3);
System.out.println(tekst3 ++ (zapyt3.hasSolution() ? "prawda": "falsz") );
```

```
// — przyklad nr 2 ——————————-
System.out.println("Pokaz dziadkow Malgosi.");
tekst = "dziadkowie(X,malgosia)";
Query zapyt = new Query(tekst);
System.out.println( tekst +  + ( zapyt.hasSolution() ) );
while(zapyt.hasMoreSolutions()){
Hashtable rozw = zapyt.nextSolution();
System.out.println(rozw);
}


}
```

```
% rodzina.pl compiled 0.00 sec, 11,224 bytes
consult('rodzina.pl') ok

 Sprawdz czy mama i tata Janosika sa podanymi osobami.
matka(maria,janosik) prawda
ojciec(maria,janosik) falsz

 Pokaz dziadkow Malgosi.
dziadkowie(X,malgosia) true
{X=dawid}
{X=maria}
{X=marek}
{X=jadwiga}
BUILD SUCCESSFUL (total time: 0 seconds)
```

Przykład rozwiązujący równanie kwadratowe, plik Prologa (rownKwadr.pl):

```
start :- write('Obliczam rownanie kwadratowe: a*x ^ 2 + b*x + c'), nl,
write('Wprowadz wartosc zmiennej a i zakoncz kropka \'.\' '),nl,
read(A),
assert(a(A)),
write('Wprowadz wartosc zmiennej b i zakoncz kropka \'.\' '),nl,
read(B),
assert(b(B)),
write('Wprowadz wartosc zmiennej c i zakoncz kropka \'.\' '),nl,
read(C),
assert(c(C)),
Delta is B * B - 4 * A * C,
rozw(Delta,A,B); rozw(Delta).


rozw(Delta) :- Delta < 0, write('Rownanie nie ma pierwiastkow'), nl.


rozw(Delta,A,B) :- Delta = 0, X is (-B) / (2 * A),
write('X = '), write(X), nl.


rozw(Delta,A,B) :- Delta > 0,
X1 is (-B + sqrt(Delta)) / (2 * A),
X2 is (-B - sqrt(Delta)) / (2 * A),
write('X1 = '), write(X1), nl,
write('X2 = '), write(X2) , nl.
```

W konsoli Prologa, przykład uruchamiamy po wpisaniu: start. Oczywiście, wcześniejszym bezbłędnym skompilowaniu pliku.

W Javie, uruchomimy to w ten sposób (RownKwadr.java):

```
import jpl.Query;


/*
* Program wymaga podania 3 wartosci (a,b,c) i na ich podstawie oblicza
z rownania kwadratowego wartosci x-ow
*/
public class RownKwadr {


public static void main(String[] args) {
// zapytanie podajemy w formie tekstowej / String
String tekst = ćonsult('rownKwadr.pl')";


// przy pomocy klasy Query wykonujemy nasze zapytanie
Query zapytanie = new Query(tekst);

// wypisanie wyniku naszego zapytania
System.out.println(tekst +
+ (zapytanie.hasSolution() ? ók": ńie udalo sie"));


String tekst1 = śtart";
Query zapyt1 = new Query(tekst1);
System.out.println(zapyt1.hasSolution());
} }
```

```
% rownKwadr.pl compiled 0.00 sec, 4,120 bytes
consult('rownKwadr.pl') ok
Obliczam rownanie kwadratowe: a*x^2 + b*x + c
Wprowadz wartosc zmiennej a i zakoncz kropka '.'
|:
3.
Wprowadz wartosc zmiennej b i zakoncz kropka '.'
|:
4.
Wprowadz wartosc zmiennej c i zakoncz kropka '.'
|:
1.
X1 = -0.333333
X2 = -1.0
true
BUILD SUCCESSFUL (total time: 8 seconds)
```

Zachęcamy do zgłębiania tego tematu, gdyż jest interesujący i wciągający. Dzięki tym dwóm językom można stworzyć m.in. ciekawe zagadki logiczne z przyjemnym interfejsem.

DZIĘKUJEMY ZA UWAGĘ