

Предметно-ориентированные
ЯЗЫКИ
и Lisp как средство их
построения

Дмитрий Бушенко

Что это такое DSL?

Предметно-ориентированный язык (Domain Specific Language) – это язык программирования ограниченной выразительности, фокусирующийся на некоторой предметной области

Set c
Set c
Mov
Mov
Mov
Mov
Take



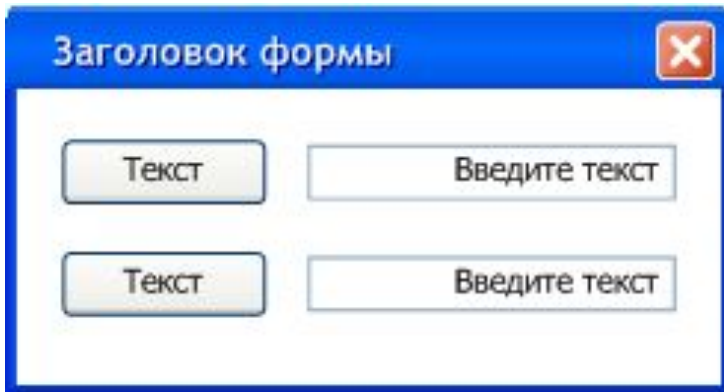
Пример программы на DSL

Computer:	c = new Computer();
processor:	P = new Processor();
cores -- 2	p.setCores(2);
type -- i386	p.setType(ProcTypes.i386);
disk:	c.setProcessor(p);
size -- 75	d = new Disk();
speed -- 7200	d.setSize(75);
interface -- SATA	d.setSpeed(7200);
	d.setInterface(DiskTypes.SATA);
	c.setDisk(d);

Представление

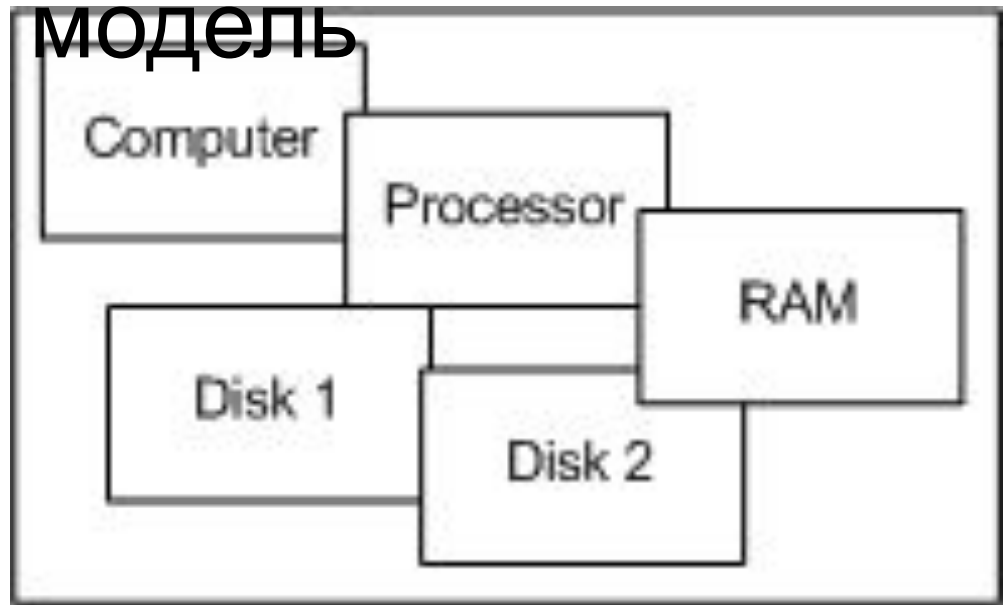
языка

```
Computer:  
  processor:  
    cores -- 2  
    type  -- i386  
  disk:  
    size  -- 75  
    speed -- 7200  
    interface -- SATA
```



Семантическая

МОДЕЛЬ



DSL и eDSL

Внешний DSL

- Реализуется средствами создания ЯП.
- Произвольный синтаксис.
- Чужой по отношению к основному языку проекта.

Встроенный DSL

- Реализуется на базе основного языка.
- Синтаксис ограничен синтаксисом хостового языка.
- Тот же язык, на базе которого реализован.

eDSL средствами java и ruby

```
computer()  
  .processor()  
    .cores(2)  
    .speed(2500)  
    .i386()  
  .disk()  
    .size(150)  
  .disk()  
    .size(75)  
    .speed(7200)  
  .sata()  
.end();
```

```
computer(  
  processor(:cores => 2,  
            :type => :i386),  
  disk(:size => 150),  
  disk(:size => 75,  
        :speed => 7200,  
        :interface => :sata))
```

Другой пример DSL на ruby

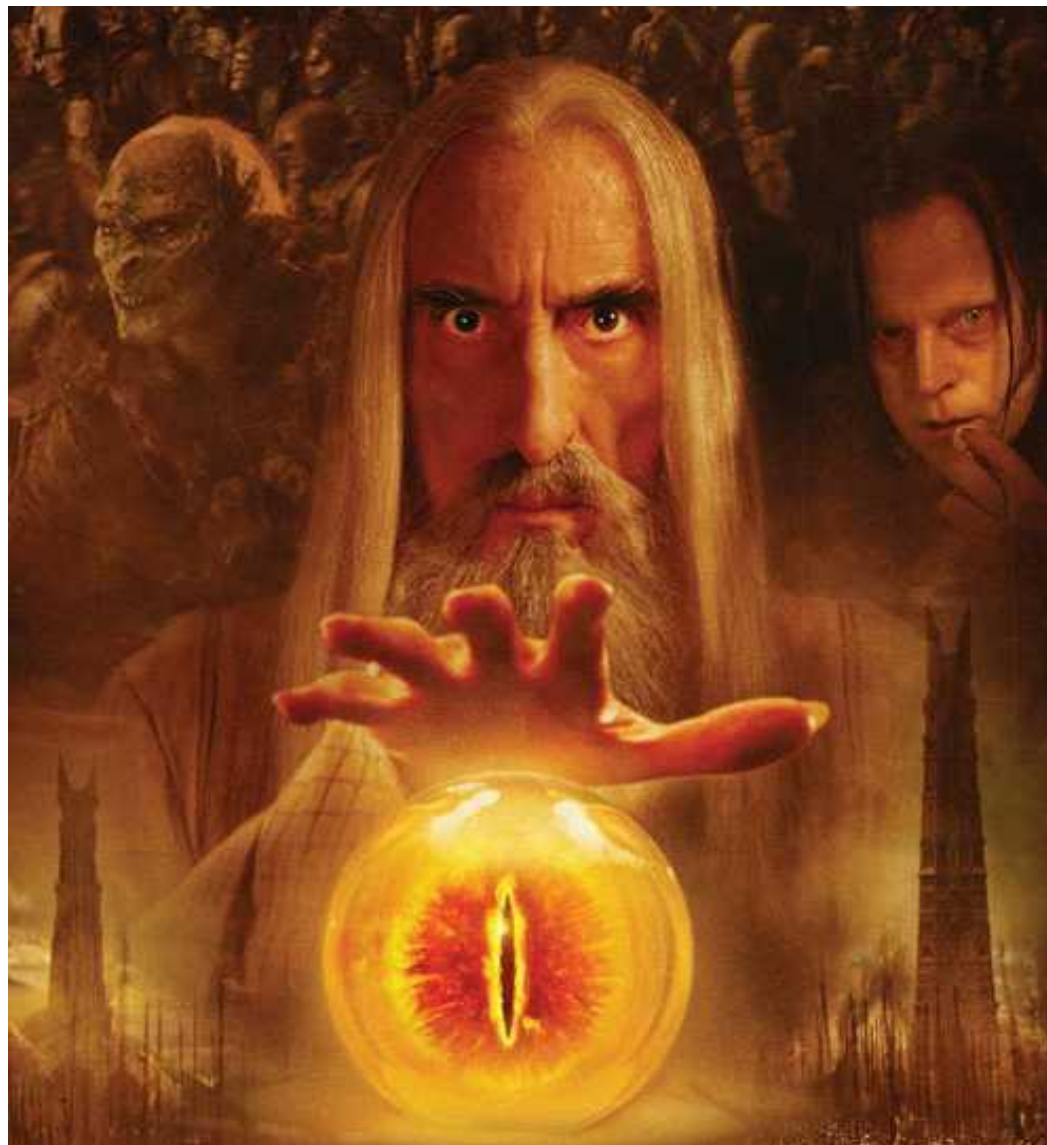
```
Computer:      [:computer,
  processor:    [:processor,
    cores -- 2  [:cores, 2],
    type -- i386 [:type, :i386]],
  disk:        [:disk,
    size -- 75  [:size, 75],
    speed -- 7200 [:speed, 7200],
    interface -- SATA [:interface, :sata]]]
```


Как добавить сюда управляющие конструкции?

```
computer()           [:computer,  
...                 ...  
.disk()             [:disk,  
.size(75)           [:size, 75],  
.speed(7200)        [:speed, 7200],  
.sata()              [:interface, :sata]]  
.end();
```

Как добавить сюда управляющие конструкции?

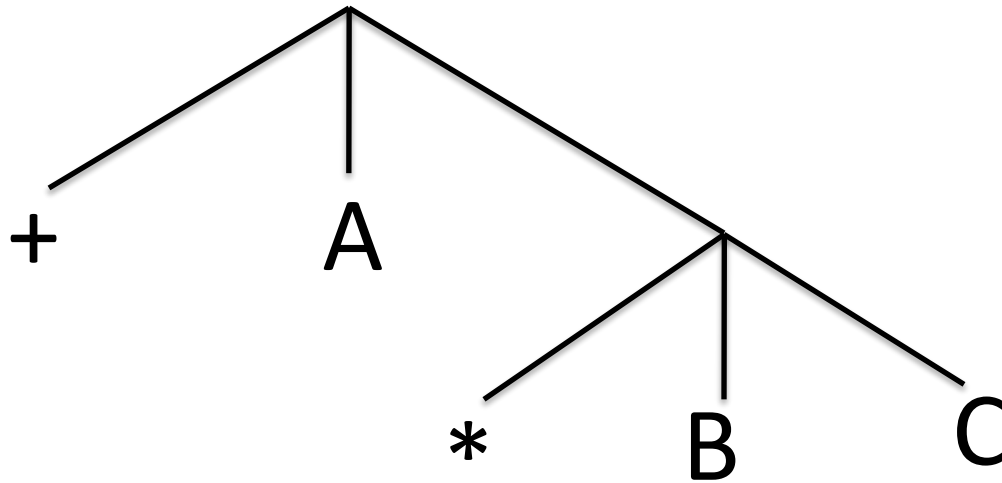
```
computer()           [:computer,
...                 ...
.times(2)           2.times do |i|
  .disk()           [:disk,
    .size(75)       [:size, 75],
    .speed(7200)    [:speed, 7200],
    .sata()         [:interface, :sata]]
  .end_times()      end]
.end();
```



Мы, программисты, представляем себя
20 волшебниками, повелителями кода

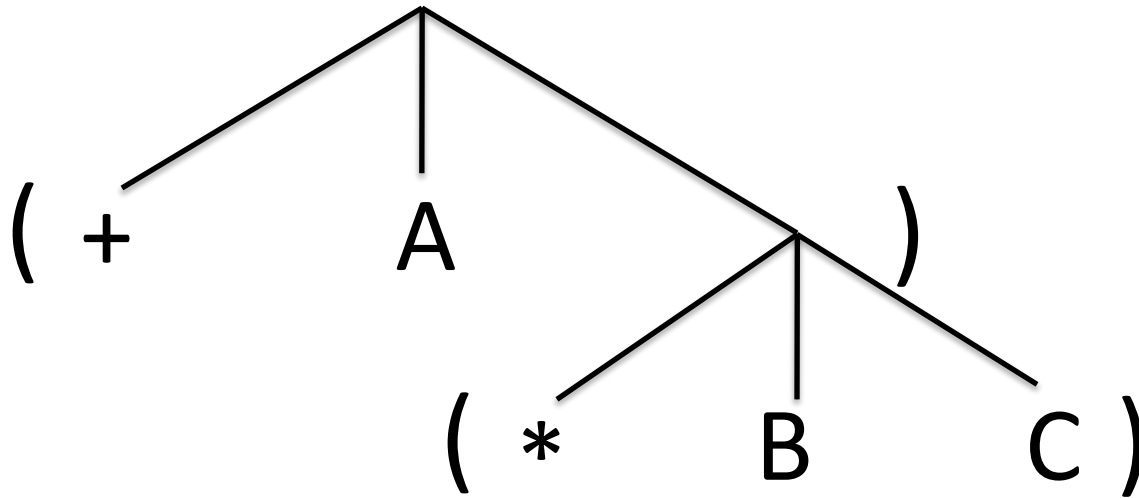
Дерево разбора выражения

$A + B * C$



Дерево разбора выражения

$A + B * C$



Дерево разбора выражения

$A + B * C$

(+ A)

(* B C)

```
c = new Computer();  
P = new Processor();  
p.setCores(2);  
p.setType(ProcTypes.i386);  
c.setProcessor(p);
```

Computer:

processor:

cores -- 2

type -- i386



(Computer

(processor

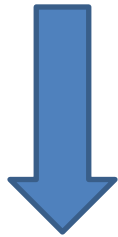
(cores 2)

(type i386)))



Unless == not if

(unless (= a b)
(do something))



(if (not (= a b))
(do something))

Шаблон: unless

Параметры: **cond**,
body

(if (not (**cond**)
body))

Unless == not if

Шаблон: unless

Параметры: **cond**,
body

```
(defmacro unless  
  [cond body]  
  `(if (not ~cond)  
      ~body))
```

```
(if (not (cond)  
    body)
```

Порядок выполнения функций и макросов

Компиляция
Макросы



Исполнение
Функции

Инфиксная нотация

```
(defmacro infix [arg1 op arg2]  
  (list op arg1 arg2))
```

```
(infix 2 + 5)
```

```
7
```

(for (i = 0, i < 3, i ++)
(println i))

```
(defmacro for [args & body]
  (let [a1 (nth args 0)  a2 (nth args 1)
        a3 (nth args 2)  a4 (nth args 3)
        a5 (nth args 4)  a6 (nth args 5)
        a7 (nth args 6)  a8 (nth args 7)]
    (cond
      (not (= a1 a4 a7)) (throw (Exception. "Use the same variable for the cycle"))
      (not (= a2 '=')) (throw (Exception. "Use the '=' for the variable assignment"))
      (not (contains? #{'> '< '>= '<= '='} a5)) (throw (Exception. "Use one of the
        operators: =, <, <=, >, >="))
      (not (contains? #{'++ '--} a8)) (throw (Exception. "Use one of the operators: ++,
        --"))
      :default (let [op (if (= a8 '++) 'inc 'dec)]
        `(loop [~a1 ~a3] (if (not (~a5 ~a1 ~a6))
          ~a1
          (do ~@body
            (recur (~op ~a1))))))))))
```



I'll be back!

09

Анафорический макрос

```
(defmacro not-nil  
  ([expr then & else]  
   (let [result (symbol "result")]  
     `(let [~result ~expr  
           (if (not (nil? ~result)) ~then (do ~@else))))))
```

```
user> (not-nil (+ 2 3) (println result))
```

```
5
```

```
user> (not-nil nil (println result) (println "The result  
is nil!"))
```

```
The result is nil!
```



В C# 4.0 мы добавили
новую
фантастическую
ВОЗМОЖНОСТЬ:
бла-бла-бла!

Программисты



C#

Барбара Стрейзанд!
Ууу-у-уу!

Пример М.Фаулера

SVCLFOWLER	10101MS0120050313
SVCLHONPE	10201DX0320050315
SVCLTWO	x10301MRP220050329
USGE103	x50214..7050329

Как все это распарсить?

Разные типы – разные поля

SVCLFOWLER

4-18: CustomerName

19-23: CustomerID

24-27 : CallTypeCode

28-35 : DateOfCallString

USGE103

4-8 : CustomerID

9-22: CustomerName

30-30: Cycle

31-36: ReadDate

Расставим скобочки...

```
(def-reader SVCLFOWLER  
  [4 18 CustomerName]  
  [19 23 CustomerID]  
  [24 27 CallTypeCode]  
  [28 35 DateOfCallString])
```

```
(def-reader USGE103  
  [4 8 CustomerID]  
  [9 22 CustomerName]  
  [30 30 Cycle]  
  [31 36 ReadDate])
```

```
(defmacro def-reader [class-name & fields]
  (let [method-names (map #(vector (symbol (nth % 2)) '[] 'String) fields)
        methods (map (fn [method-name]
                       [(symbol (first method-name))
                        (symbol (second method-name))])
                     method-names)]
    `(do (gen-class
         :name ~class-name
         :prefix ~prefix
         :init "init"
         :state "state"
         :methods ~methods
         [~(symbol (first (first method-names)))
          ~@method-names]
         (defn ~(symbol (first (first method-names)))
             (defn ~(symbol (second (first method-names)))
                 (do (println "state this#")
                     assoc :data ~state))))))
```



```
1 package fowlertest;
2
3 import fowler.core.ThirdClass;
4
5 public class FowlerTest {
6
7     public static void main(String[] args) {
8         ThirdClass t = new ThirdClass();
9         t.setData("01234567890abcdefghijklmnopqrstuvwzyz");
10        System.out.println( t.field1() );
11        System.out.println( t.field2() );
12        System.out.println( t.field3() );
13    }
14 }
15
```

(def-reader fowler.core.ThirdClass

[2 7 field1]

[9 15 field2]

[17 20 field3])

02



01 Почувствуй

Вопросы