# Język SELF

**Michał Kozłowski**
**22.03.2004**

# 1.Metody

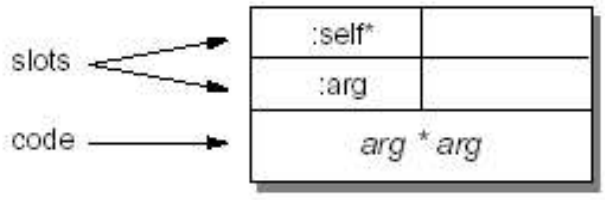
W niektórych slotach obiektów mogą znajdować się **obiekty z kodem** czyli po prostu **metody**. Kiedy odwołujemy się do slotu przechowującego metodę. Wykonywane są następujące kroki:

- Obiekt metody jest klonowany. Stworzony obiekt nazywamy **obiektem aktywacji metody (OAM)**.

- W OAM tworzony jest slot **self*** oraz ustawiany odbiorcę komunikatu

- Sloty odpowiadające argumentom (jeśli są) – inicjalizowane są na wartości parametrów aktualnych

- Kod wykonywany jest w kontekście właśnie stworzonego OAM



## 2.Bloki

Napis reprezentujący blok (czyli wyrażenie ograniczone nawiasami kwadratowymi) powoduje stworzenie dwóch obiektów: (**block data object** oraz **block method object**).

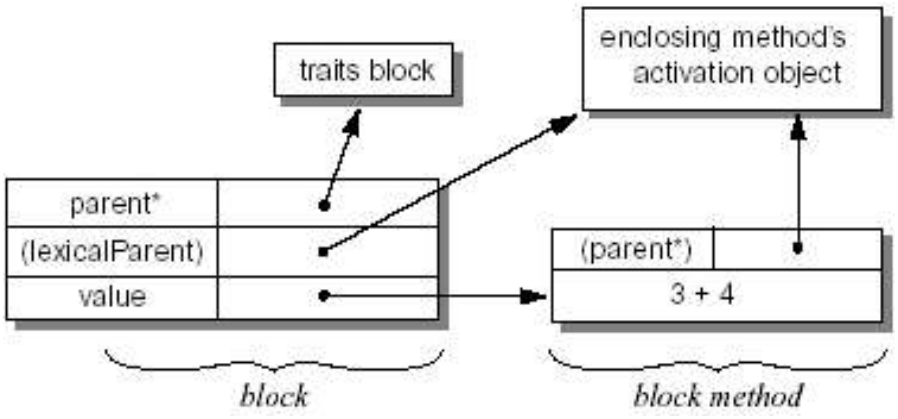
**Block Data Object** ma ustawiony następujące sloty:

- parent* - wskazuje pewien obiekt, który definiuje właściwości bloków (na przykład metody **loop, whileTrue**).
- value - wskazuje na **block method object**

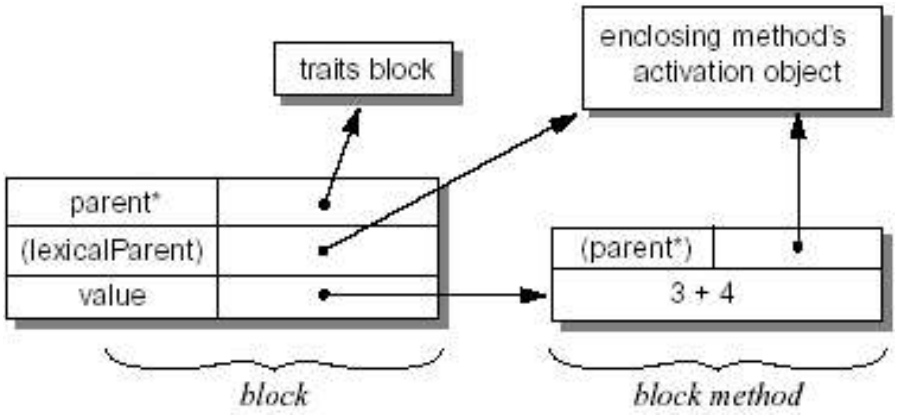
**Block Method Object**

- Zawiera slot parent* niedostępny z poziomu języka. Podczas wykonywania bloku slot ten ustawiany jest na obiekt, w którym tekstowo zawiera się blok.
- Zawiera kod bloku

Na przykład blok [3+4] wygląda następująco:



## 2.1.Wykonywanie Bloków




Wykonanie bloku przebiega dwuetapowo. Przy wyliczaniu obiektu **block** jego ukryty slot (lexicalParent) ustawiany jest na aktualny obiekt aktywacji.

Po wykonaniu value na tak wyliczonym obiekcie, obiekt **block method** jest klonowany, ukrty slot (parent*) ustawiany jest na (lexicalParent), sloty argumentów są wypełniane, po czym wykonywany jest kod.
Dzięki temu blok wykonuje się w kontekście metody, w której był zawarty tekstowo.

Próba wykonania bloku zawartego w metodzie, która już zakończyła działanie jest zgłaszane jako błąd (nawet jeśli blok nie odwołuje się do slotów zewnętrznych).

## 3.Semantyka wysyłania komunikatów

Definicja funkcji **send**(rec, sel, args)

rec – odbiorca komunikatu
sel – nazwa komunikatu
args – argumenty

```
if begins_with_underscore(sel) then
    invoke_primitive(rec,sel,args)
else
begin
    M = lookup(rec,sel,empty)
    if (size(M)==0) then error(“Message not understood”);
    if (size(M)==1) then return eval(rec,M,args);
    if (size(M)>1) then error(“Message is ambiguous”);
end
```

## 3.1.Funkcja lookup

Funkcja lookup próbuje znaleźć obiekt, który ma wskazane sloty. Przeszukiwanie polega na przejściu w głąb grafu obiektów, gdzie krawędzie wyznaczane są przez sloty typu **parent** (czyli z gwiazdką na końcu). Zwracany jest zbiór pasujących slotów.

```
function lookup(obj,sel,V)
Wej?cie:
obj – obiekt, od którego zaczynamy przeszukiwanie
sel – szukany slot
V   – zbiór ju? odwiedzonych obiektów
Wyj?cie:
M   – zbiór pasuj?cych slotów

if (obj in V) then
    M:=empty;
else
    M:={s in obj|s.name=sel}
if (M==empty) then M:=parent_lookup(obj,sel,V);

parent_lookup(obj,sel,v)
```

$$P \leftarrow \{s \in obj \mid s.isParent\}$$
$$M \leftarrow \bigcup_{s \in P} lookup(s.contents, sel, V \cup \{obj\})$$

**return** $M$

## 4.1.Wielodziedziczenie z priorytetem

Istnieje rozszerzenie języka SELF, które pozwala zdefiniować, kolejność w jakiej będą poszukiwane obiekty w metodzie **lookup_parent**.

Odpowiednia definicja obiektu wygląda następująco:

```
(|parent1* =(...).
  parent2** =(...).
  parent3*** = (...)|)
```

Liczba gwiazdek oznacza priorytet przy wyszukiwaniu (najpierw wybieramy slot z mniejszą liczbą gwiazdek). Kiedy zostanie znaleziony pasujący obiekt – nie szukamy na ścieżkach o mniejszym priorytecie. W przypadku dwóch slotów o tym samym priorytecie kolejność przeszukiwania nie jest ustalony (ale jeśli znajdziemy odpowiedni slot w jednej gałęzi – to **nie** szukamy w pozostałych).

## 5.Literatura


### 1.dSelf

http://www.cs.tu-berlin.de/~tolk/dself/

### 2.Strona Self'a na SUN

http://self.sun.com

(w dystrybucji SELF'a znajduje się dokumentacja)