

**UFS - UNIVERSIDADE FEDERAL DE SERGIPE**  
**CCET - CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA**  
**DCOMP - DEPARTAMENTO DE COMPUTAÇÃO**  
**CIÊNCIA DA COMPUTAÇÃO**

**COMP0201 – TURMA – PARADIGMAS DE PROGRAMAÇÃO**

**Relatório sobre a linguagem de**  
**Programação: Smalltalk**

**PROF. ALBERTO COSTA NETO**

**Alkyly Samyr Santos - 201120001016**  
**Dimitri Carvalho Menezes – 201120000786**  
**Isaias Santana dos Santos - 201210007951**

**São Cristóvão - SE**

**Dia, 28 de fevereiro de 2014**

## **Introdução:**

Smalltalk é uma linguagem de programação orientada a objetos que fornece um ambiente de programação, uma biblioteca de classes e um modo de armazenamento permanente dos dados. Este relatório mostrará os conceitos da linguagem em comparação com a linguagem Java, bem como os critérios de avaliações da linguagem. As imagens feitas foram de códigos implementados na IDE Pharo.

A primeira parte contém a história e a evolução da linguagem, sua importância no mercado de software e características gerais.

Enquanto que, a segunda parte analisará a linguagem, de forma a mostrar os conceitos do paradigma orientado a objeto e de como o interpretador avalia e entende o que está sendo transmitido no código.

## SUMÁRIO

1. Apresentação da Linguagem SmallTalk	
1.1 História.....	4
1.2 Aplicações.....	6
1.3 Características Marcantes.....	7
2. Critérios de avaliação da Linguagem SmallTalk	
2.1. Avaliação sobre Tipos e Valores.....	8
2.1.1. Objetos de Classes.....	8
2.1.2. Operações em Coleções.....	8
2.1.3. Tipos Recursivos.....	8
2.1.4. Sistemas de Tipos.....	8
2.1.5. Outras Características.....	12
2.2. Avaliação de Expressões.....	12
2.2.1. Mensagens Binárias, Unárias, N árias.....	13
2.2.2. Precedências de mensagens.....	14
2.2.3. Atribuições.....	15
2.2.4. Expressões Condicionais.....	18
2.3. Vínculos e Escopos.....	21
2.4. Variáveis.....	23
2.5. Parâmetros.....	24
2.6. Abstração de Funções.....	25
2.7. Polimorfismo.....	26
2.8 Herança.....	28
3. Conclusão.....	29
4. Referências Bibliográficas.....	30

## 1. Apresentação da Linguagem Smalltalk

### 1.1. História:

Os passos iniciais para a criação da linguagem Smalltalk foram descritos por Alan Kay no início de 1970, influenciados pela ideia de classes do Simula-67 como uma forma de organizar o código, de gráficos de tartaruga idealizados no projeto LOGO no MIT. Segundo o site da LOGO<sup>1</sup>, *"a tartaruga gráfica é capaz de desenhar linhas na tela. Mais do que isso, a tartaruga gráfica permite fazer o que as linguagens tradicionais não conseguem: ela dá retorno imediato (feedback), e esse retorno imediato é que torna a linguagem LOGO divertida e mais fácil de aprender."* A linguagem Smalltalk também foi influenciada diretamente em algo que é chamado de "manipulação direta de interfaces", inspirados no sistema de desenho de bloco de notas, desenvolvido por Ivan Sutherland no MIT Lincoln Laboratories no início dos anos 1960.

Entre 1971 e 1975, o grupo de Alan Kay junto ao Xerox PARC<sup>2</sup> implementou a primeira versão Smalltalk, com seus ambientes gráficos e aplicações. Este sistema inclui:

- A linguagem foi inteiramente baseada nos conceitos de classe e mensagens do Simula.
- A linguagem não tinha sintaxe fixa. Cada classe era responsável não só por seu próprio comportamento e definição do estado, mas até mesmo para analisar o fluxo de sinal que se seguiu uma menção de uma instância.

Durante os anos 1975-1976, tornou-se claro que a falta de atenção às questões de desempenho e isto tornou dificultando as pesquisas. O grupo de Kay continuou com uma grande reformulação de todos os aspectos do sistema Smalltalk:

---

<sup>1</sup> LOGO é uma linguagem com ênfase no ensino criada com base nas teorias de aprendizagem natural de Papert e que influenciou várias outras linguagens, inclusive Smalltalk. Mais em: <<http://projetologo.webs.com/texto2.html>>

<sup>2</sup> Xerox PARC, projetos de pesquisa que se tornaram pioneiras em invenções de interfaces gráficas GUI. Mais em < <http://www-sul.stanford.edu/mac/parc.html> >

- A ideia de hierarquia de herança e subclasse foi incorporada Smalltalk.
- A sintaxe da linguagem foi corrigida. Isto permitiu compilação eficiente, interpretador compacto (bytecode), e um variado conjunto de instruções.
- Introdução da tela de acesso Browser por Larry Tesler. O Browser aumentou muito a produtividade de um programador Smalltalk.

Em 1979-1980, a atenção da equipe de Smalltalk pensou em uma possibilidade de Smalltalk ser comercializada além do projeto junto ao Xerox PARC. A equipe projetou e implementou mais uma geração de sistemas de Smalltalk, desta vez com algumas mudanças especificamente destinadas a portabilidade e exploração de hardware padrão. Estas incluíram:

- A adoção do conjunto de caracteres ASCII.
- Smalltalk-80 removeu a capacidade dos métodos primitivos para acessar diretamente qualquer posição de memória e introduziu métodos primitivos que previa a funcionalidade necessária.
- A linguagem Smalltalk -80 introduziu o conceito de metaclasses, para fornecer uma maneira de falar sobre comportamento (mensagens) que eram específicas para uma classe individual.
- O sistema de modelo visão-controlador (MVC) foi introduzido para aplicações interativas.
- Os programas criados em Smalltalk rodavam sobre uma máquina virtual, sendo esta linguagem uma das primeiras a abordar o conceito de máquina virtual.

Por fim, em 1981 um número significativo de a equipe Smalltalk sentiu que era importante tomar a ação direta para propagar Smalltalk além da Xerox PARC. Adele Goldberg, substituído Alan Kay como chefe do grupo, e Dave Robson, um membro do grupo há muito tempo, decidiram escrever uma série de livros sobre Smalltalk.

## **1.2. Aplicações:**

Durante os anos 90 o mercado de ferramentas de Smalltalk cresceu, por exemplo , em 1994, nos EUA o valor de mercado da linguagem valia cerca de 17% no mercado de desenvolvimento de sistemas cliente-servidor.

Smalltalk tem sido usada em aplicações variadas onde a ênfase está na simulação de modelos de sistemas, como automação de escritórios, animação gráfica, informática educativa, instrumentos virtuais, editores de texto, bancos de dados orientados a objeto (Magma, GemStone), relacionais (MySQL, PostgreSQL, mapeamentos objeto-relacional (Glorp) e desenvolvimento web: Seaside, Aida, Komanche, Swazoo.

A maior implementação do Smalltalk talvez seja o VisualWorks<sup>3</sup>. que inclusive é uma IDE da própria linguagem. Ela fornece não apenas o conjunto básico de aulas, mas também tela instalações de pintura, conectividade de banco de dados, gráficos de negócios, bem como interfaces para C e outras linguagens compiladas. É possível desenvolver um sistema em um Mac, executá-lo em um DOS ou em um UNIX.

Existem também aplicações da IBM, com o seu IBM Smalltalk (a Smalltalk interface baseada em standard) e VisualAge (uma versão de Smalltalk com uma interface estilo VisualWorks ) . Hewlett Packard também está no mercado como uma versão distribuída do Smalltalk chamado Distributed Smalltalk.

### **1.3. Características marcantes:**

Os próprios desenvolvedores da Smalltalk falam que, a linguagem não é apenas uma linguagem de programação. É possível classificar a linguagem como:

- Uma linguagem de programação orientada a objeto, que oferece sintaxe e semântica da linguagem.

---

<sup>3</sup> Eles próprios se apelidam como "Do Anything, Faster, Better", ou seja, faça algo mais rápido e melhor utilizando a portabilidade em amplas variedades de plataformas. Mais em < <http://www.cincomsmalltalk.com/main/products/visualworks/> >

- Um ambiente de programação. Associado a ele um grande conjunto de classes. O desenvolvedor passa a maior parte do seu tempo estendendo classes ao invés de programando a partir do zero.
- Um ambiente de desenvolvimento de aplicações (ADE). Como os navegadores, inspetores e depuradores são todos derivados da mesma fonte, também há coerência entre (algumas das) as várias implementações . Além disso, o código-fonte para essas ferramentas também está disponível com o ADE.

Smalltalk é uma linguagem interpretada em termos de execução, ou seja, um *bytecode*, é interpretado por uma máquina virtual. Em vez de um arquivo executável, o que é construído em Smalltalk é chamado de imagem virtual. A imagem contém todos os objetos do sistema, incluindo as bibliotecas de classes, várias ferramentas, e o nosso próprio código. Quando se abre a imagem já temos um programa Smalltalk a correr. Conforme se vai adicionando o nosso próprio código, ele torna-se parte da imagem. Guardar a imagem irá preservar todo o estado do ambiente, códigos e objetos.

Em Smalltalk não há a noção de tipo. Os objetos são aplicados para representar tudo, desde os tipos abstratos de dados (inteiros, reais, *arrays*, etc.) até menus, janelas, *kernel* e até o próprio compilador. Também não há herança múltipla, sendo o relacionamento apenas de um-para-muitos. Possui abstração de dados (encapsulamento) e tratamento de exceções. O envio de mensagens é uma operação polimórfica, como por exemplo um operador de soma que pode somar números ou concatenar Strings. A mesma mensagem, enviada a objetos diferentes, pode resultar em respostas diferentes. Todos os conceitos serão explicados de uma forma mais ampla na seção a seguir.

## 2. Critérios de avaliação da linguagem:

Brevemente, algumas regras de sintaxes devem ser informadas:

- Palavras reservadas em que não podem existir variáveis com este nome (**true**, **false**, **nil**, **self** (semelhante ao this do Java) e **super**).
- Declaração de variável é feita entre os operadores | |. Exemplo: | x y|
- Uma String é identificada com aspas simples ' '. Exemplo: 'Hello World'.
- Um char é declarado com o caractere \$ antes dele. Exemplo \$d.
- Operador de atribuição é := .Exemplo a:= 5.
- Operador de retorno de uma função é ^: Exemplo: ^nil.
- É necessário um ponto final ao final de uma instrução.
- Código indentado não faz diferença para o interpretador.
- O nome da variável pode conter letras e dígitos, começando por uma letra.
- Os comentários são colocados entre aspas duplas.

### 2.1. Avaliação de valores e tipos:

Em Smalltalk, não existem noções de objetos primitivos da mesma forma que qualquer outra linguagem de programação orientada a objetos tem. Os valores são objetos das classes:

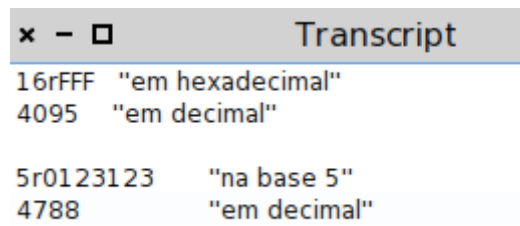
#### 2.1.1. Objetos de Classes:

- Classe Number: representa valores inteiros positivos e negativos. É possível definir a base da representação dos números pela notação *r*.

(base) *notação r* (numero)

Os códigos a seguir, mostram números nas bases hexadecimal e na base 5.



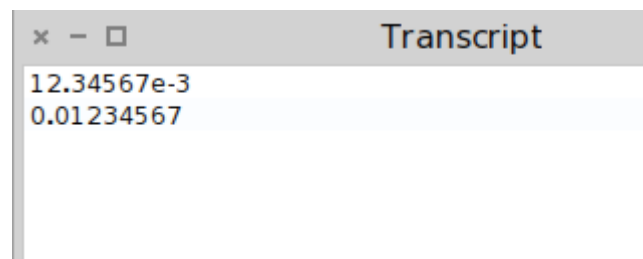


```
x - □ Transcript
16rFFF "em hexadecimal"
4095 "em decimal"

5r0123123 "na base 5"
4788 "em decimal"
```

**Figura 2.1.1 Representação de bases numéricas**

- Classe Float: representa números decimais em pontos flutuantes. Podemos representar pontos flutuantes em termos de notação científica, utilizando a notação e:



```
x - □ Transcript
12.34567e-3
0.01234567
```

**Figura 2.1.2 Representação de notação científica**

- String: Fazem parte da hierarquia da classe Collection. Representa cadeia de caracteres.
- Symbols: São praticamente iguais as Strings porém não contem métodos mais complexos. Representado por #.Exemplo: #Good.
- Char: representa um único caractere.
- Classes True e False herdam da classe Boolean e contem as instancias true e false respectivamente.
- Arrays: Fazem parte da hierarquia da classe Collection. É possível usar o símbolo # para indicar que é uma declaração de array, e seus dados são incluídos dentro de chaves, colchetes, ou parênteses. Os arrays são dos tipos dinâmicos e declarados sem limites de espaços. É possível fazer concatenação de arrays:

```
x - Transcript
| z x y |
z:= #('abc' 'def').
x:= #[16rff 0 2r111 2e2]. "elementos em diversas notações"
y:= {28}, (z,x). "Concatenacao de coleções"
#(28 'abc' 'def' 255 0 7 200)
```

Figura 2.1.3 Diversas formas de representar um Array

### 2.1.2. Operações em coleções:

Não possível fazer um tipo composto sem ser por classe. Existem diversas classes de coleções em Smalltalk. A maioria ou boa parte delas é possível fazer operações de união, intersecção e diferença. As imagens a seguir são operações feitas em *arrays* (que também faz parte da hierarquia de Collections) e a coleção Bag:

```
x - Transcript
| z x result |
x:= #(1 2 3 4 5).
z:= #(5 4 2 9 18 21).

result := x union: z.
Transcript show: result.
#(18 1 2 3 4 5 21 9)

result := x difference: z.
Transcript show: result.
#(1 3)

result := x intersection: z.
Transcript show: result
#(5 2 4)
```

Figura 2.1.4 Operações com Arrays

```
x - □ Transcript
| x y z|
x:= Bag new.
y:= Bag new.

x add: 0.
x add: 1.
x add: 3.
x add: 5.
x add: 7.

y add: 0.
y add: 2.
y add: 4.
y add: 6.

z:= x union: y.
Transcript show: z.    a Bag(0 1 2 3 4 5 6 7)

z:= x intersection: y.
Transcript show: z.    a Bag(0)

z:= x difference: y.
Transcript show: z.    a Bag(1 3 5 7)
```

**Figura 2.1.4 Operações com Coleção Bag**

### 2.1.3. Tipos Recursivos:

É possível definir um tipo de dado com uma variável dela mesmo. O código a seguir define um nó de lista encadeada: *instanceVariableNames* se refere aos nomes das variáveis, enquanto que *classVariableNames* define os tipos dessas variáveis:

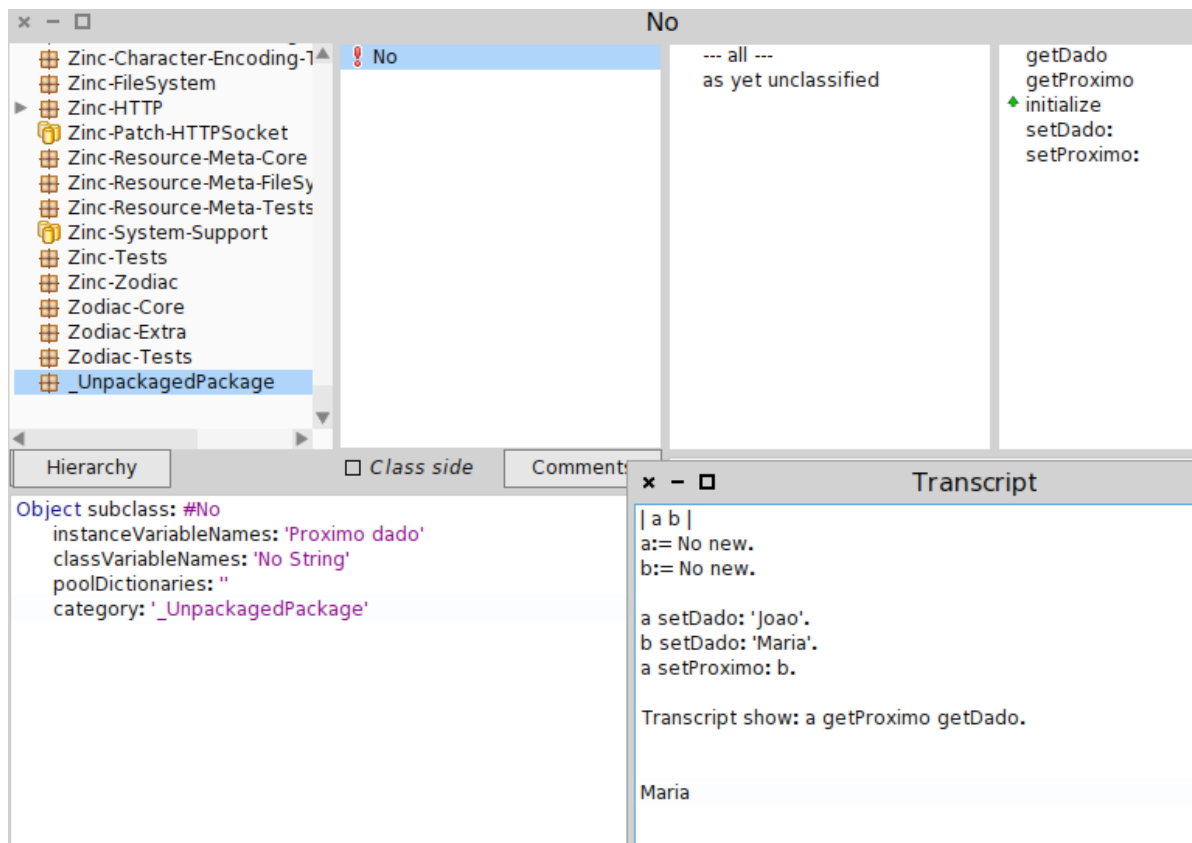
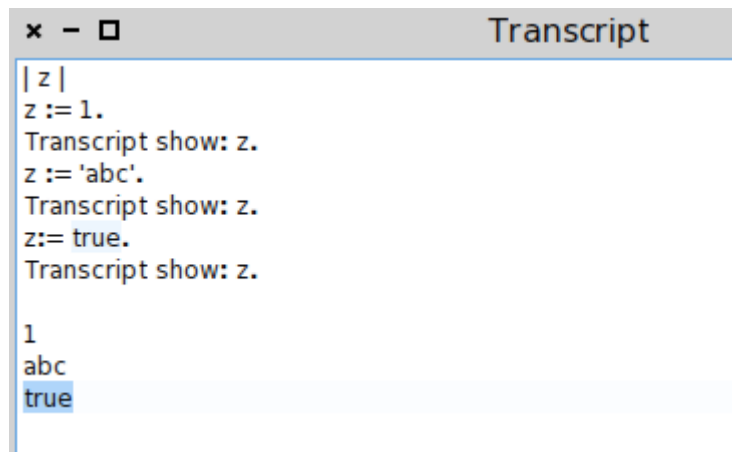


Figura 2.1.3 Exemplo de um tipo de dado recursivo.

### 2.1.4. Sistemas de Tipos

Smalltalk é uma linguagem dinamicamente tipada, ou seja, variáveis podem ter tipos diferentes em tempos diferentes e, portanto só se sabe o tipo de uma expressão no momento da execução:



**Figura 2.1.3 Uma variável assume diversos valores**

#### 2.1.5. Outras Características:

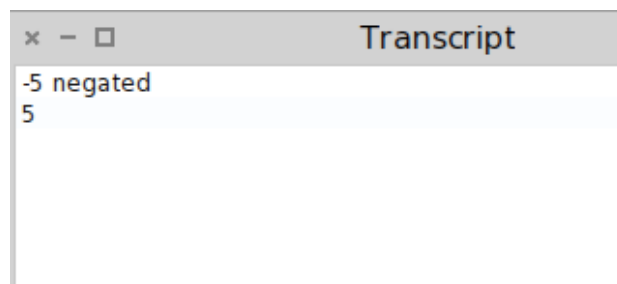
- Smalltalk não suporta o conceito de tipos enumerados igual a Java 5. A única forma é criando uma classe especificando os atributos.
- Não há *casting* como, por exemplo, em Java, que dá para converter um *float* em *int*. Isso só é possível utilizando métodos como por exemplo o *asString*, *asNumber*, *asArray*.

## 2.2. Avaliação de Expressões:

Em Smalltalk as expressões são chamadas de mensagens. Um método é iniciado com o envio de uma mensagem a um objeto. O objeto responde avaliando o método do mesmo nome, se ele tiver um. Se não, a mensagem é enviada ao objeto da sua superclasse. Esse processo continua até que seja encontrado o método, caso contrário, é gerado um erro.

#### 2.2.1. Mensagens Binárias, Unárias, N árias.

Uma mensagem unária apenas é possível uma mensagem ser enviada a um objeto:



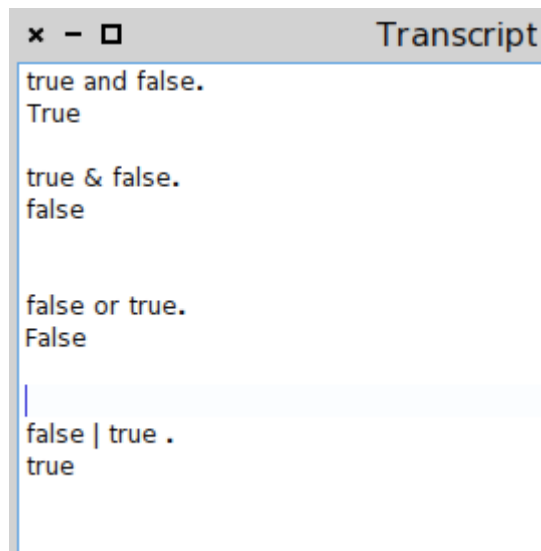
**Figura 2.2.1.1 Exemplo de mensagem Unária**

Existe também mensagens binárias. O interpretador entende que a mensagem que sai de um primeiro objeto, tem que chegar ao argumento passado no parâmetro da mensagem. A tabela a seguir mostrará alguns operadores binários:

Operador	Função	Exemplo	Resultado
+	Soma	3 + 5	8
-	Subtração	3 - 5	-2
*	Multiplicação	3 * 5	15
/	Divisão	1 / 3	(1/3)
**	Expoente	5**5	3125
//	Divisão com arredondamento floor	3 // 2	1
\%	Função resto de divisão	20 \% 4	0
<	Menor que	4 < 5	True
<=	Menor ou igual a que	12 <= 12	True
>	Maior que	32 > 18	True
>=	Maior ou igual a que	81 >= 80	True
=	Igual	0 = 1	False
~=	Diferente	0 ~=1	True
~~	Objetos distintos	3 ~~ 9	True
==	Objetos idênticos	5 == 5	True
&	And lógico (os dois parâmetros são avaliados)*	true & false	False
	Or lógico (os dois parâmetros são avaliados)*	false   true	True
,	Concatenar coleções	'abc', 'def'	'abcdef'
:=	Atribuição	A:=5	-

**Tabela 2.2.1.1 Exemplos de mensagens binárias**

\*Existem também os operadores *or* e *and*, que utilizam a avaliação somente do primeiro parâmetro. Isso pode ter um retorno errado, como mostra a imagem:



```
x - □ Transcript
true and false.
True

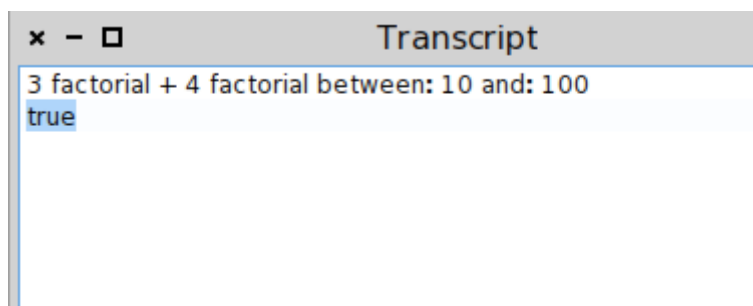
true & false.
false

false or true.
False

false | true .
true
```

**Figura 2.2.1.2 Diferenças entre os operadores *and* e *&*, *or* e *|*.**

Mensagens key words: É forma em que se faz a operação ternária, ou n-árias. Exemplo:



```
x - □ Transcript
3 factorial + 4 factorial between: 10 and: 100
true
```

**Figura 2.2.1.3 Exemplo de mensagem ternária**

A seguinte avaliação é feita assim:

- 3 recebe a mensagem "factorial" e retorna 6
- 4 recebe a mensagem "factorial" e retorna 24
- 6 recebe a mensagem "+" e 24 como argumento e por fim retorna 30
- 30 recebe a mensagem "between:and:" com 10 e 100 como argumentos e retorna true, pois o número 30 está dentro do intervalo 10 e 100.

## 2.2.2. Precedências de mensagens:

É priorizada da seguinte forma:

- Avaliação é feita da esquerda para direita.
- Mensagens unárias têm a maior precedência.
- Mensagens binárias têm a segunda maior precedência.

- Mensagens keywords têm a terceira maior precedência.
- Por fim, Atribuições tem a menor precedência.

Uma coisa que é bem diferente da maioria das linguagens é que os operadores como + e \* não são símbolos algébricos, eles são simplesmente mensagens. Usando as regras de precedência acima, a expressão  $3 + 4 * 5$  é avaliada como  $(3 + 4) * 5$ , retornado 35 e não 23 como de costume. A precedência pode ser resolvida utilizando um parêntese, para determinar melhor o que o sistema deve ser feito:

```

x - □ Transcript
3 factorial ** 9 log
5.527706834508981

(3 factorial ** 9) log
7.003361253452793

(3 factorial ** 9 ) log between:10 and:100
false

3 factorial ** (9 log between:10 and:100 )
false

```

**Figura 2.2.2.2 Exemplos de precedências entre mensagens**

No primeiro código, as mensagens unárias são feitas primeiros. 3 factorial resulta em 6. Depois, 9 log resulta em 0,954. Após isso,  $6 ** 0,954$  resulta finalmente em 5,5277.

No segundo código, há precedência. Objeto 3 envia mensagem a factorial, e retorna 6. Depois 6 envia mensagem \*\* (função *pow*) e retorna 10077696, faz o log na base 10 e retorna o valor 7,003.

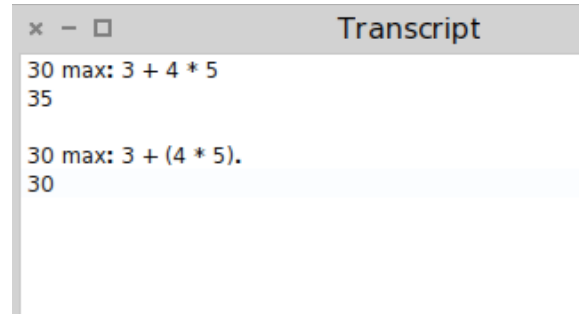
No terceiro código, a mensagem binária precede a mensagem *keyword*. Portanto, 3 factorial resulta em 6,  $6 ** 9$  resulta em 10077696, e este aplicado a log na base 10, resulta em 7,003. Por fim, o numero não está entre 10 e 100, portanto a expressão é falsa.

E por fim, no ultimo código existe uma precedência de que a mensagem key-word tem mais prioridade que a mensagem unária. Na medida em que ele



dá o resultado como falso, a operação `**` não aceita um booleano como parâmetro. Portanto, aconteceu um erro e a mensagem retornou falso.

Outro exemplo é verificado qual o numero máximo.



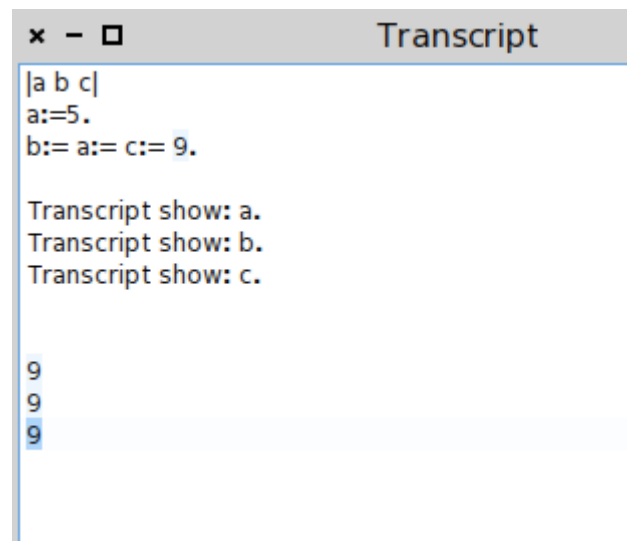
```
30 max: 3 + 4 * 5
35

30 max: 3 + (4 * 5).
30
```

**Figura 2.2.2.3 Outro Exemplo de precedência entre mensagens**

### 2.2.3. Atribuições:

Sobre atribuições, o interpretador permite fazer uma múltipla atribuição, sendo avaliada da mesma forma que Java: da direita para esquerda (É um pouco diferente da precedência em mensagens).



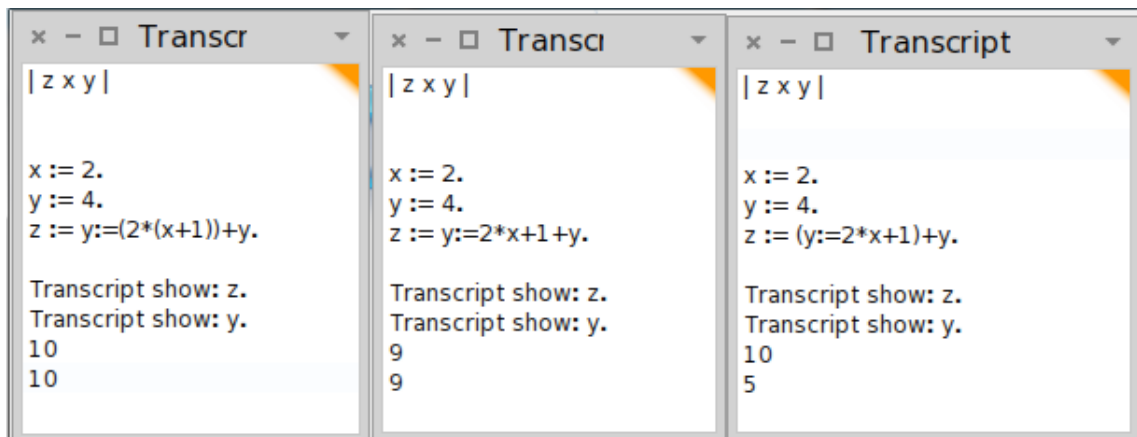
```
|a b c|
a:=5.
b:= a:= c:= 9.

Transcript show: a.
Transcript show: b.
Transcript show: c.

9
9
9
```

**Figura 2.2.3.1 Atribuições Múltiplas**

É possível fazer uma atribuição múltipla quando uma atribuição depende de outros valores que ainda serão modificados:



**Figura 2.2.3.1 Exemplos de atribuições com resultados diferentes**

O primeiro código é avaliado da seguinte forma:

O resultado da expressão  $x+1$  é multiplicado por 2, o resultante é somado com o valor  $y$ . Tudo isso é inserido em  $y$  e depois copiado a  $z$ .

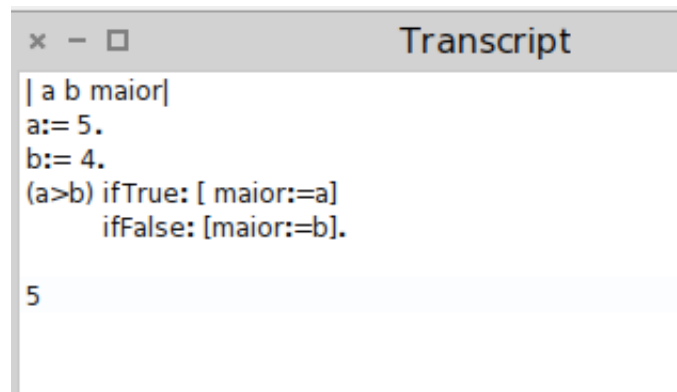
Na segunda expressão, 2 é multiplicado com  $x$ , 1 é somado com o resultado de  $2*x$ , depois somado junto a  $y$ . Os resultados são somados e atribuídos a  $y$  e  $z$ .

Enquanto que na terceira expressão, é atribuído a  $y$  a expressão  $2*x$  somados com 1. Após isso, soma-se com a outra variável  $y$ , e o resultado final é copiado tanto para  $y$  quanto para  $z$ .

Smalltalk influenciou Java em muitos requisitos, um deles foi à questão de referências. Não existe noção de alocação de ponteiro, nem manipulação direta de endereços. As atribuições são feitas da forma que o lado esquerdo expressão retorna o endereço de memória, e o lado direito o conteúdo a ser armazenado.

#### 2.2.4. Expressões Condicionais:

Em Java, é possível fazer um operador ternário expressando uma condição. Em Smalltalk, até onde foi pesquisado e testado, não há. As estruturas condicionais em Java são tratadas como objetos em Smalltalk. O conteúdo dentro de uma dessas estruturas é chamado de bloco. Um bloco é representado por colchetes e contém código não avaliado.



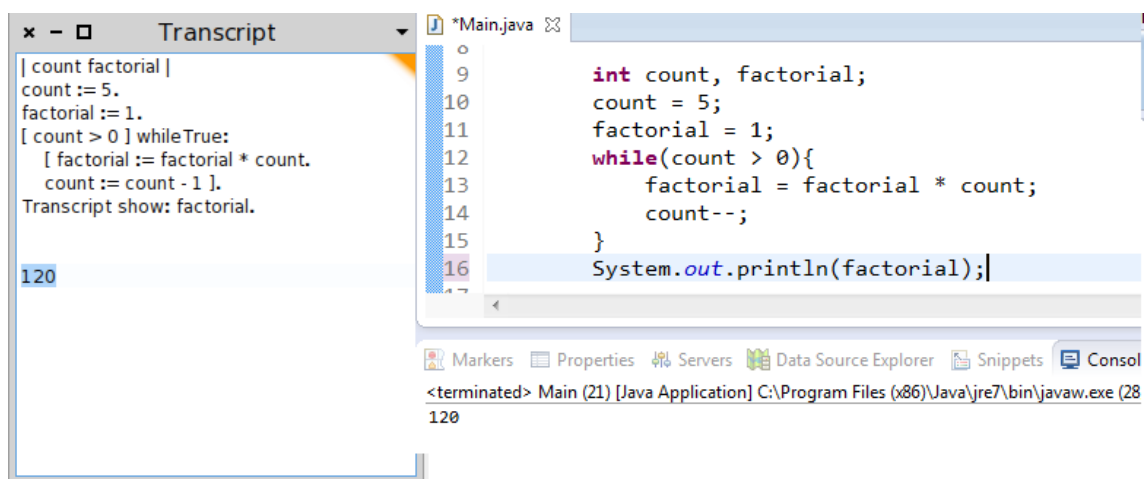
```
Transcript
| a b maior|
a:= 5.
b:= 4.
(a>b) ifTrue: [ maior:=a]
      ifFalse: [maior:=b].

5
```

Figura 2.2.4.1: Exemplo de estrutura de decisão

Esse trecho de código é entendido da seguinte forma: Primeiro, é avaliada a expressão  $a > b$ , resultando em um objeto booleano `true` ou `false`. Esse objeto é enviado à mensagem *ifTrue:* e *ifFalse:* com os dois blocos não avaliados como argumentos. Logo, o bloco da mensagem *ifTrue* foi acionado, então objeto força a execução do primeiro bloco.

Smalltalk não tem loops condicionais como parte da linguagem. São mensagens enviadas para o Objeto *BlockClosures*. Assim como o comando *while* do Java, o *whileTrue* continuará sempre que a mensagem for avaliada como `true`. É possível também fazer um *whileFalse* com a mensagem avaliando sempre como `false`.



```
Transcript
| count factorial |
count := 5.
factorial := 1.
[ count > 0 ] whileTrue:
  [ factorial := factorial * count.
    count := count - 1 ].
Transcript show: factorial.

120

Main.java
9      int count, factorial;
10     count = 5;
11     factorial = 1;
12     while(count > 0){
13         factorial = factorial * count;
14         count--;
15     }
16     System.out.println(factorial);
```

Figura 2.2.4.2 Exemplo de Fatorial usando while em Smalltalk e em Java

Semelhante ao `do.. while` do Java, quando um bloco de código vem primeiro do que a condição, o bloco é executado e a condição é avaliada.

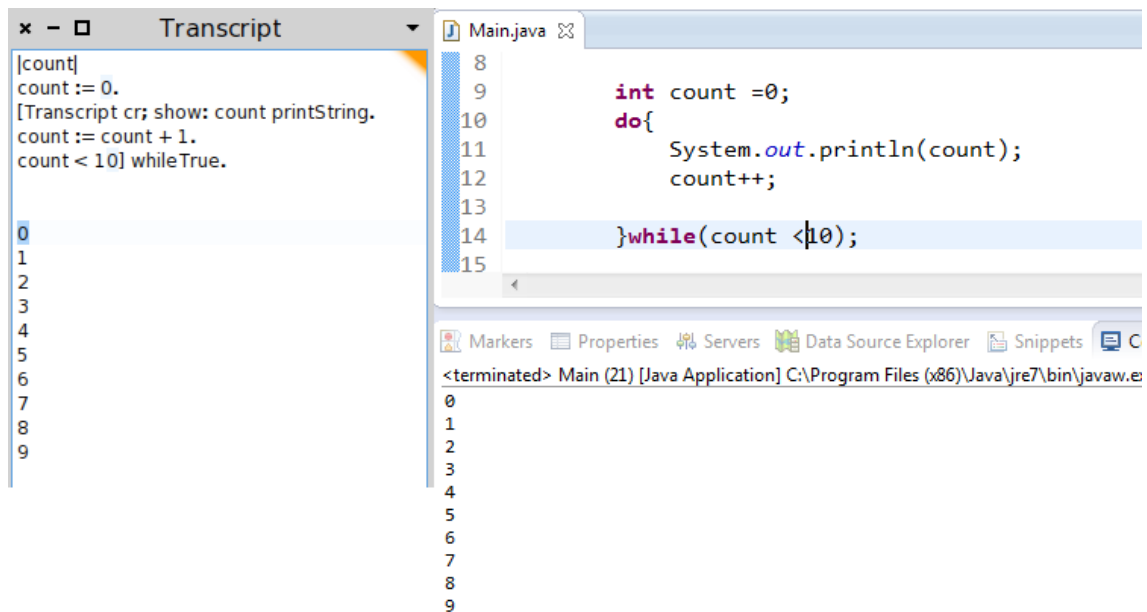


Figura 2.2.4.3 Exemplo simples de um do...while

É possível também fazer o iterador *For* semelhante ao Java. A condição de parada é feita pela mensagem *to:*

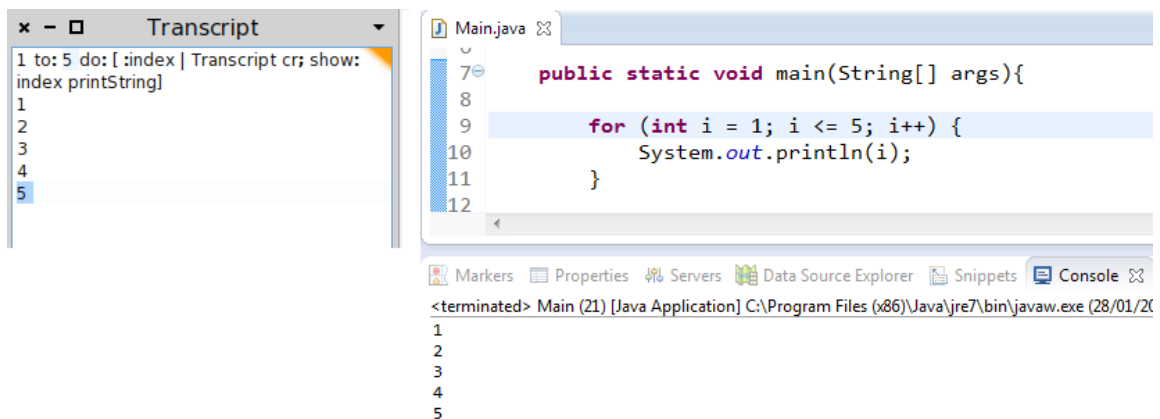


Figura 2.2.4.4 Exemplo de um iterador FOR em Smalltalk e em Java

Existe também, um repetidor sem uma condição explícita que em Java não contém:

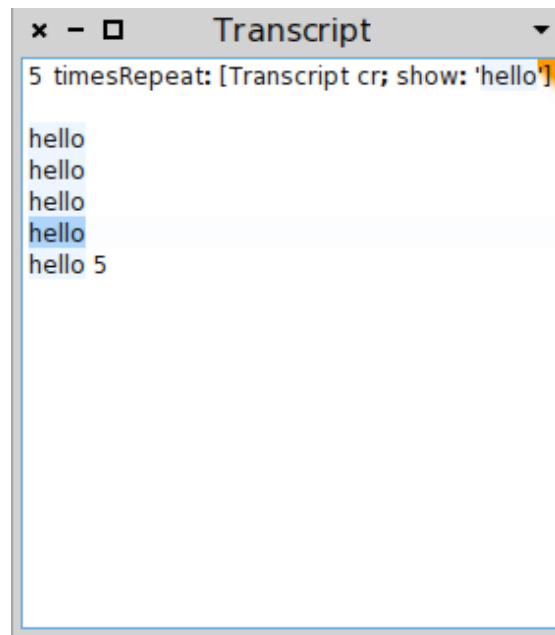


Figura 2.2.4.5 Exemplo de um repetidor simples em Smalltalk

### 2.3. Vínculos:

Vínculo é uma associação da variável com o seu conteúdo que irá ser representado. O vínculo deve estar associado a um escopo, ou seja, um trecho de código em que esse vínculo é utilizado. Em Smalltalk, uma declaração de um vínculo no programa principal, não modifica o vínculo que tem em um bloco de instruções dentro de uma função:

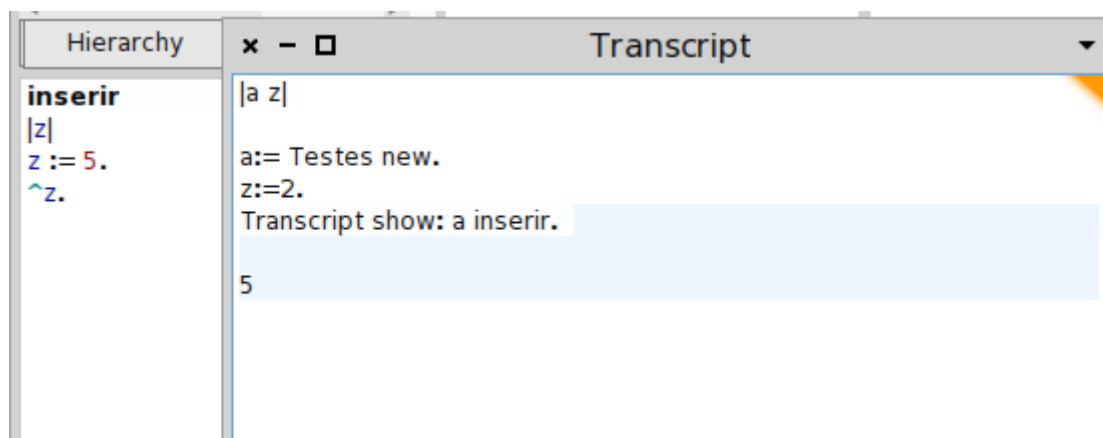


Figura 2.3.1. Variáveis iguais, vínculos Diferentes

Isso significa que uma variável global não tem visibilidade dentro de um procedimento, e também uma variável privada não é acessada no programa principal. Podemos dizer que quando a variável foi vinculada, aconteceu uma Ocorrência de vínculo, e quando ela é utilizada, houve uma Ocorrência aplicada deste vínculo. É possível fazer com que variáveis tenham identificações semelhantes em blocos diferentes:

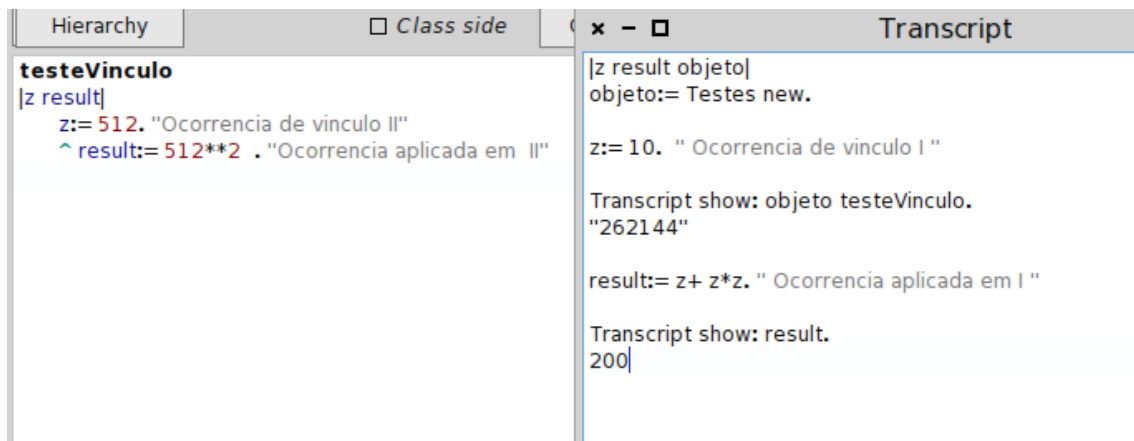


Figura 2.3.2. Ocorrências de vínculos

Porém, existem definições de blocos aninhados em que o identificador perde a visibilidade, e o compilador fica sem saber o que fazer: Tanto em Java quanto em Smalltalk, não é permitido identificar variáveis iguais mesmo estando em escopos aninhados e distintos:

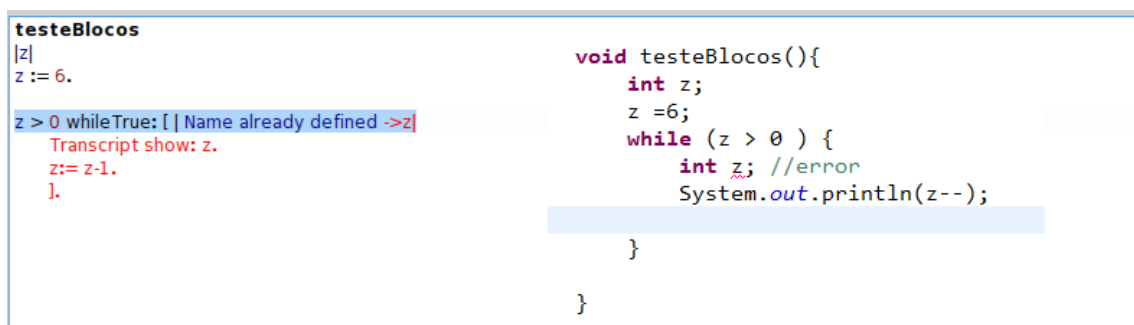


Figura 2.3.3. Erro ao definir uma variável igual em um escopo aninhado.

É possível fazer e usar vínculos que o próprio escopo faz. É o caso das recursões:

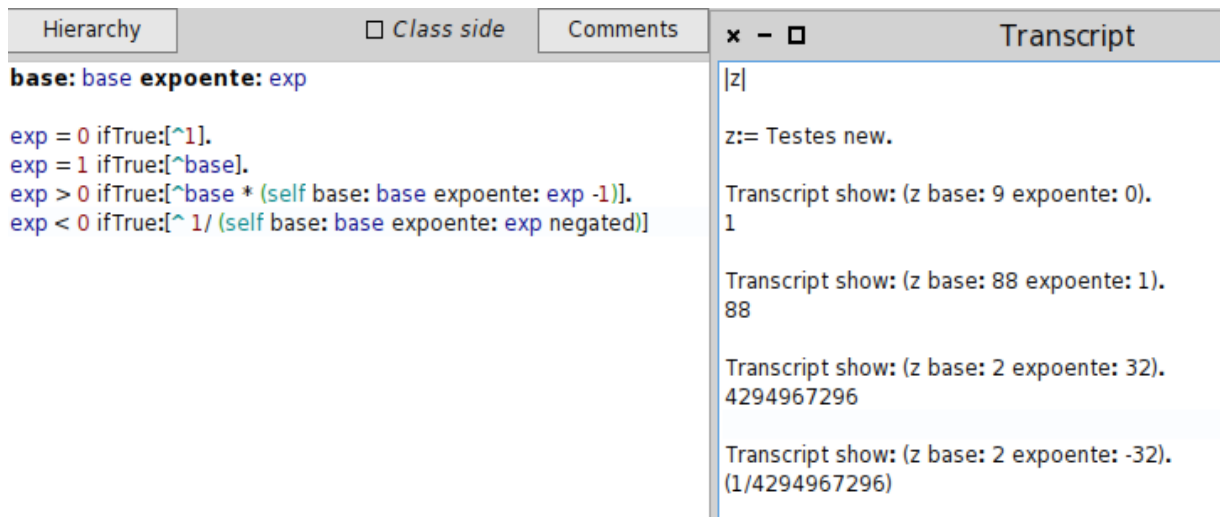


Figura 2.3.4. Escopo recursivo.

Outras Características:

- Não é possível declarar e inicializar uma variável ao mesmo tempo, como é feito em Java.
- Como em Smalltalk não se declara tipos nem constantes, não há um vínculo estatico ou dinâmico de constantes.

## 2.4. Variáveis

Variáveis em Smalltalk não são tipadas, qualquer nome pode ser vinculado a qualquer objeto. Qualquer operação em uma variável é determinada pela classe do objeto para o qual a variável está vinculada, ou seja, isso significa que nenhum código está dependente a um só único tipo em particular. As variáveis em Smalltalk são do tipo *stack*- dinâmica, ou seja, variáveis são criadas durante a elaboração de declarações, e seu tempo de vida está determinado dentro de seu escopo. As variáveis locais são declaradas dentro de um bloco para ser usada somente dentro dele, enquanto que variáveis globais são declaradas no bloco mais externo de um programa.

A seguir, um diagrama representando o tempo de vida das variáveis:

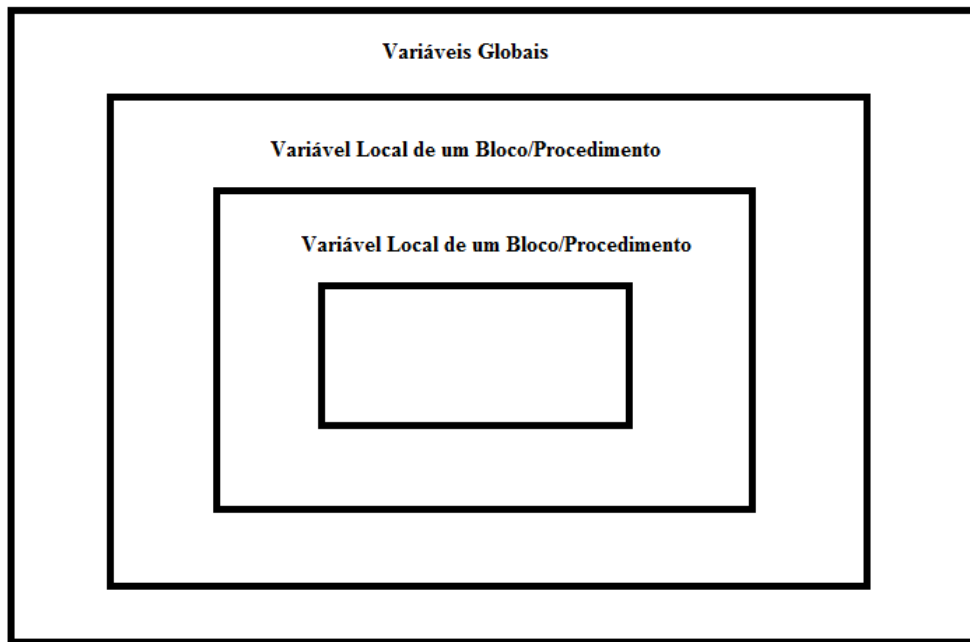


Figura 2.4.1 Diagrama do tempo de vida das variáveis

## 2.5. Parâmetros

O mecanismo de passagem de parâmetros em Smalltalk é igual à passagem de parâmetros em Java, é feita por valor, sendo que é possível atualizar o valor de variáveis internas. O exemplo a seguir mostra a classe Nome que contém os métodos *setFirst* e *setLast* que modificam o valor das variáveis.

A captura de tela mostra o ambiente de desenvolvimento Smalltalk com duas janelas principais:

- Workspace:** Contém o código Smalltalk para criar e manipular objetos da classe Nome.
 

```

|x y z|

x := Nome new.
x setFirst: 'Jose'.
x setLast: 'Roberto'.

y := Nome new.
y setFirst: 'Lucas'.
y setLast: 'Queiroz'.

z := Nome new.
z := x.
z setFirst: 'Andre'.
z := y.

Transcript show: x getFirst.
Transcript show: x getLast.
Transcript show: y getFirst.
Transcript show: y getLast.
Transcript show: z getFirst.
Transcript show: z getLast.
      
```
- Transcript:** Exibe o resultado das mensagens enviadas ao Transcript, mostrando a sequência de nomes:
 

```

Andre Roberto
Lucas Queiroz
Lucas Queiroz
      
```

Figura 2.5.1. Exemplo de uma passagem por valor.



Verificando que x foi atribuído a z, logo z passou a ter as mesmas referências na memória da variável x. Quando o *firstName* da variável Z é configurado para 'Andre', também o *firstName* da variável X foi configurado para 'André'. Por fim, A atribuição de y a z faz os dois compartilharem o mesmo conteúdo.

O modelo semântico de passagem de parâmetro usado em Smalltalk é o *inout*, ou seja, um parâmetro de entrada pode ser útil no processamento, e por fim na saída. A imagem a diante mostra uma definição de uma mensagem unida com as ideias do *Get* (retorno) e com a do *Set* (configurar). O retorno da mensagem é exibido no *Transcript*.

```
getSetFirstName: nome  
first:= nome.  
^nome.
```

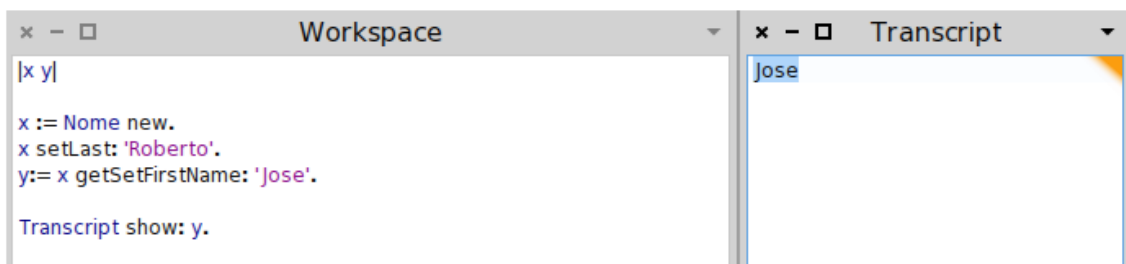


Figura 2.5.2 Exemplo de INOUT.

## 2.6. Abstração de Função

Na parametrização de argumentos, existem dois tipos de parâmetros, o parâmetro formal é o identificador usado na abstração para denotar um argumento enquanto que um parâmetro real é uma expressão passada como argumento e é fornecida no momento da chamada da abstração.

Smalltalk adota a avaliação preguiçosa como estratégia de avaliação de parâmetros. Diferentemente de Java, no qual o parâmetro real é avaliado uma única vez e o valor resultante substituído para cada ocorrência do parâmetro formal na abstração, em Smalltalk só é realizado a avaliação se realmente o parâmetro real é usado, e essa avaliação pode ser útil em processamentos futuros. A imagem a seguir mostra uma inicialização *lazy evaluated*, pois o

bloco ifNil: [] só é avaliado quando necessário. Como o atributo não foi inicializado, logo ele é nil, e o bloco precisou ser avaliado.

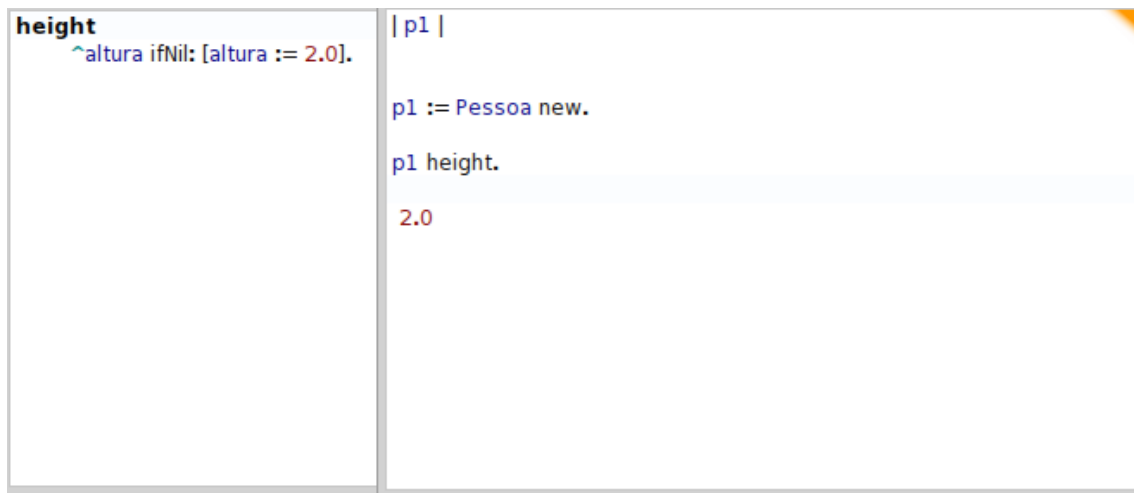


Figura 2.6.1 Exemplo de Lazy Evaluation.

## 2.7. Polimorfismo

Existem dois tipos de polimorfismo, o *ad-hoc* e o universal, Smalltalk não suporta o *ad-hoc* por coerção e nem o universal paramétrico, sendo apenas implementado o *ad-hoc* por sobrecarga: métodos com o mesmo nome mas com assinaturas diferentes e o universal por substituição: uma classe filha substitui um método da classe pai. Em Smalltalk a sobrecarga é independente de contexto, isso significa que uma lista de parâmetros de um identificador devem ser distintas na quantidade, como os tipos são checados dinamicamente não é possível distinguir um identificador de outro pelos tipos do parâmetro mas sim por sua quantidade, possibilitando assim a sobrecarga.

Na figura 1 foi declarada uma classe chamada Números, na linha 20 foi criado o método `imprime` que mostra o valor da variável de classe `val`, na linha 25 o método `imprime` foi sobrecarregado recebendo um parâmetro retornando-o como resultado. Na figura 2 foi declarada a classe `Primos` que herda de `Numeros`, onde na linha 13 o método `imprime` da superclasse é substituído (sobescrito) pela subclasse.

```

1  Object subclass: #Numeros
2      instanceVariableNames: 'val'
3      classVariableNames: ''
4      poolDictionaries: ''
5      category: 'Numeros'
6
7  Initialize
8      super initialize.
9      val := 0.
10
11  getVal
12      ^ val.
13
14  setVal: aValor
15      val := aValor.
16
17  verificaSeEMembro: aInteger
18      ^Transcript show: self class == aInteger class
19
20  imprime
21      Transcript show: val.
22
23  "temos aqui uma sobrecarga do metodo imprimir na mesma classe,
24  o que mostra que smalltalk suporta polimorfismo Ad-Hoc
25  por sobrecarga"
26  imprime: aValor
27      Transcript show: aValor.

```

Figura 2.6.1: exemplo de sobrecarga.

```

1  Numeros subclass: #Primos
2      instanceVariableNames: ''
3      classVariableNames: ''
4      poolDictionaries: ''
5      category: 'Numeros'
6
7  Initialize
8      super initialize.
9
10  verificaSeEPrimo: temp "a função isPrime já existe na classe Integer"
11      Transcript show: temp isPrime.
12
13  imprime
14      Transcript show: '
15          sobrescrevi o metodo imprime da classe pai(Numeros).
16          Isso mostra que Smalltalk suporta
17          polimorfismo universal por inclusao'.

```

Figura 2.6.2: Exemplo sobrecarga por substituição

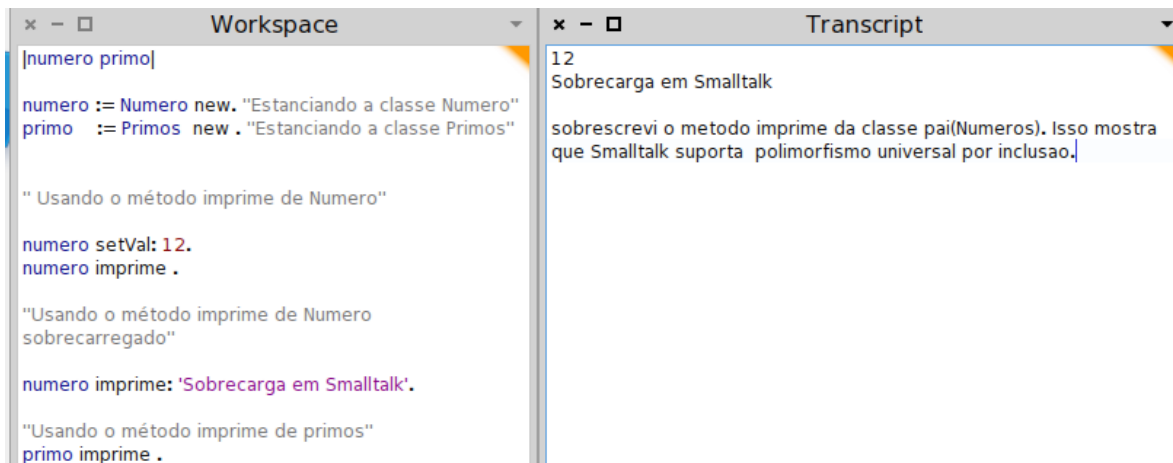


Figura 2.6.3: Exemplo de execução.

## 2.8. Herança

Herança é um mecanismo na qual permite a reutilização da estrutura e do comportamento de uma classe ao se definir novas classes. Existe dois tipos de herança:

- Simples: na qual uma classe apenas herda de uma única superclasse.
- Múltiplas: uma classe pode herdar de mais de uma classe.

Em Smalltalk só é possível utilizar a herança simples. Como exemplo tomemos novamente as figuras 1 e 2, onde a classe Primos herda da classe Numero.

### 3. Conclusão:

A programação orientada a objetos é reconhecida pela flexibilidade e fácil manutenção. Focando especificamente na linguagem Smalltalk, uma linguagem puramente orientada a objetos e muito vantajosa por possuir uma estrutura para melhor simular o mundo real, um conceito de tipo de dado abstrato, em conjunto com o encapsulamento e o princípio de esconder dados, aumenta a confiabilidade e operabilidade separando a especificação do objeto de sua implementação, juntos somados ao polimorfismo e a herança com uma maior flexibilidade e reutilização de código. Para um programador que já conhece o paradigma de OO, Smalltalk é totalmente legível visto que a mesma filosofia de código é implementada, só mudando a ideia de checagem de tipos de dados. Em termos de legibilidade, a sintaxe atrapalha um pouco em termos de chamadas de mensagens binárias, mensagens *key-words*, pois o programador deve conhecer muito bem as regras de precedência. Durante o relatório, foi comum o acontecimento de erros gerados pela falta de precedência de mensagens, alguns como, por exemplo, expressões matemáticas que na verdade o interpretador Smalltalk avalia como mensagens comuns, outro exemplo é a expressão (vetor at: i getNome), no qual o índice de array tenta enviar a mensagem getNome, gerando um erro de sintaxe, e dentre outros exemplos.

Em termos de confiabilidade, o programador deve ficar bem atento ao critério de variáveis dinamicamente tipadas, pois não há checagem de tipos. Este critério pode até dar uma maior flexibilidade para o programador, mas deixa os programas menos eficientes e os erros são descobertos somente na execução (facilmente um vetor de dados virava uma instância qualquer). Também durante o desenvolver deste relatório, aconteceram alguns erros de retorno de valor nulo (nil) no qual deveria ser tratado facilmente por uma estrutura de tratamento de exceções, outro detalhe fundamental para um código confiável, e que a sintaxe em Smalltalk é um pouco diferente das demais. Outra questão importante é sobre o gerenciamento de memória semelhante ao Java, pois o programador não precisa se preocupar com coleta de lixo. Finalizando, o relatório foi bem produtivo para o conhecimento do que acontece em um interpretador de código de linguagem.

#### 4. Referências

- [1] TUCKER, Allen B.; NODNAN, Robert E. Linguagens de Programação - Principios e Paradigmas. Segunda Edição.
- [2] HUNT, John. Smalltalk and Object Orientation: An Introduction.
- [3] VisualWorks® . Disponível em: <http://www.cincomsmalltalk.com/main/products/visualworks/>> Acesso em 12 de janeiro de 2104.
- [4] The Xerox PARC Visit Disponível em: <<http://www.sul.stanford.edu/mac/parc.html>> Acesso em 12 de janeiro de 2104.
- [5] Smalltalk, The Developer's Guide. Disponível em: <<http://stephane.ducasse.free.fr/FreeBooks/ByExample/>> Acesso em 12 de janeiro de 2104.
- [6] LEON, Ramon .Objects, Classes, and Constructors, Smalltalk Style Disponível em: <http://onsmalltalk.com/objects-classes-and-constructors-smalltalk-style> Acesso em 19 de janeiro de 2104.
- [7] **Lazy Evaluation (Caching of Values)** Disponível em: <<http://computer-programming-forum.com/3-smalltalk/7706dc6e5062a8e0.htm>> Acesso em: 21 de fevereiro de 2014.
- [8] Assignment and Parameter Passing in Smalltalk Disponível em: <http://people.cs.clemson.edu/~turner/courses/cs428/summer98/section1/webct/content/st/stasg.html> Acesso em: 8 de fevereiro de 2014.
- [9] MOSER, Heinrich. Smalltalk: Overview and Implementation Issues. Disponível em: <http://www.heinzi.at/texte/smalltalk.pdf> Acesso em: 7 de fevereiro de 2014.
- [10] BREVE INTRODUÇÃO AO SMALLTALK Disponível em: <http://w3.ualg.pt/~hdaniel/poo/Smalltalk.pdf> Acesso em: 12 de janeiro de 2014.