Programação Orientada à Objetos

Estruturas e Exceções

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Estruturas de seleção

- Smalltalk permite a escrita de blocos de código que só serão executados caso uma condição seja verdadeira (ou falsa)
- Isto é feito de acordo paradigma da orientação aos objetos: a condição é representado por um objeto do tipo booleano, o qual recebe uma mensagem cujo argumento é um bloco, que também é um objeto
- De fato, um bloco é um objeto composto por *statements* executáveis
- Assim, a estrutura de seleção `if-else`, comum em outras linguagens, de fato não é uma forma da linguagem Smalltalk, e sim um comportamento dos objetos booleanos

Métodos `ifTrue` e `ifFalse`

- O método `ifTrue` recebe como argumento um bloco de comandos e o executa, caso o objeto seja `true`, ou retorna sem executá-lo, caso o objeto seja `false`

```
booleana ifTrue: [ bloco_de_comandos ]
```

- O método `ifFalse` tem comportamento análogo, executando o bloco caso o objeto seja `false` e retornando sem executar se o objeto é `true`

```
booleana ifFalse: [ bloco_de_comandos ]
```

- Eles podem ser usados para simular o comportamento do `if-else` padrão das linguagens imperativas:

```
condicao ifTrue: [ bloco_true ]; ifFalse: [ bloco_false ]
```

Exemplo de uso de condicionais em SmallTalk

```
1  Object subclass: QuadraticPolynomial [
2      | a b c |
3
4      QuadraticPolynomial class >> coefA: av coefB: bv coefC: cv [
5          | r |
6          r := super new . r setA: av . r setB: bv . r setC: cv . ^r
7      ]
8
9      setA: av [
10         (av == 0) ifTrue: [ ^self error: 'a must be non-zero' ] .
11         a := av
12     ]
13
14     setB: bv [ b := bv ]
15     setC: cv [ c := cv ]
16
17     printOn: stream [
18         (a < 0) ifTrue: [ stream nextPutAll: '- ' ] .
19         (a abs ~= 1) ifTrue: [ (a abs) printOn: stream ] .
20         stream nextPutAll: 'x^2 ' .
```

Exemplo de uso de condicionais em SmallTalk

```
21          (b ~= 0) ifTrue: [
22              (b < 0) ifTrue: [ stream nextPutAll: '- ' ];
23                      ifFalse: [ stream nextPutAll: '+ ' ] .
24              (b abs ~= 1) ifTrue: [ (b abs) printOn: stream ] .
25              stream nextPutAll: 'x '
26          ] .
27          (c ~= 0) ifTrue: [
28              (c < 0) ifTrue: [ stream nextPutAll: '- ' ];
29                      ifFalse: [ stream nextPutAll: '+ ' ] .
30              (c abs) printOn: stream
31          ]
32      ]
33 ]
34
35 p := QuadraticPolynomial coefA: 1 coefB: -5 coefC: 6 .
36 p printNl .
37 q := QuadraticPolynomial coefA: -2 coefB: 0 coefC: -1 .
38 q printNl
```

Laços

- Smalltalk possui métodos que permitem executar um bloco repetidas vezes
- Os blocos possuem o método `whileTrue`, que recebe um bloco de comandos que será executado enquanto o bloco que o invocou seja avaliado como verdadeiro

```
[ bloco_condicao ] whileTrue: [ bloco_comandos ]
```

- O bloco de comandos deve, em algum momento, modificar as variáveis que compõem a condição, caso contrário o laço executará indefinidamente
- O método `whileFalse` tem comportamento semelhante, executando o bloco de comandos enquanto a condição for falsa

```
[ bloco_condicao ] whileFalse: [ bloco_comandos ]
```

Exemplo de uso de laços em SmallTalk

```
Object subclass: Math [
    Math class >> numDigits: x [
        | count n |
        count := 1 .
        n := x quo: 10 .
        [ n > 0 ] whileTrue: [ count := count + 1 . n := n quo: 10 ] .
        ^count
    ]
]

(Math numDigits: 0) printNl .
(Math numDigits: 7) printNl .
(Math numDigits: 123) printNl .
(Math numDigits: 123456789) printNl
```

Repetição

- Se um bloco de comandos deve ser executado exatamente n vezes, uma forma mais concisa e apropriada do que `whileTrue` é o método `timesRepeat` dos inteiros

```
n timesRepeat: [ bloco_de_comandos ]
```

- De fato, são equivalentes os códigos

```
x := n .
[ x > 0 ] whileTrue: [ c1 . c2 . "..." . cN . x := x - 1]
```

e

```
n timesRepeat: [ c1 . c2 . "..." . cN ]
```

- Os blocos também podem, opcionalmente, ter variáveis locais ou, no caso de coleções, capturar argumentos
- A sintaxe de um bloco é

```
[ :arg1 :arg2 "..." :argN | var1 var2 "..." varM | comandos ]
```

Exemplo de uso de blocos repetidos em SmallTalk

```
1  Object subclass: Math [
2
3      Math class >> fib: n [
4          | a b |
5          a := 0 .
6          b := 1 .
7          n timesRepeat: [ | c | c := a + b . a := b . b := c ] .
8          ^a
9      ]
10 ]
11
12 (Math fib: 1) printNl .
13 (Math fib: 2) printNl .
14 (Math fib: 3) printNl .
15 (Math fib: 4) printNl .
16 (Math fib: 5) printNl .
17 (Math fib: 100) printNl
```

Travessia em contêineres

- Em SmallTalk os contêineres oferecem o método `do`, o qual recebe um bloco de comandos como argumento e o executa para cada um dos elementos do contêiner
- Os valores destes elementos serão passados como argumentos dos blocos
- O contêiner `Interval`, que abstrai uma sequência de inteiros, pode ser utilizado para simular o laço `for` das linguagens imperativas:

```
a to: b by: step do: [:i | bloco_de_comandos ]
```

- Os valores a e b representam os valores iniciais e finais da variável i
- A partir de a, os valores serão incrementados em $step$ a cada iteração
- Se o método `by` não for utilizado, $step$ terá valor igual a 1

Exemplo de uso de travessia de contêineres

```
1 Object subclass: Primes [
2     Primes class >> upTo: n [
3         | sieve ps |
4         sieve := Array new: n .
5         1 to: n do: [ :i | sieve at: i put: 1 ] .
6         ps := OrderedCollection new .
7         ps addLast: 2 .
8         3 to: n by: 2 do: [ :i |
9             ((sieve at: i) == 1) ifTrue: [
10                ps addLast: i .
11                (i*i) to: n by: (2*i) do: [ :j | sieve at: j put: 0 ]
12            ]
13        ] .
14        ^ps
15    ]
16 ]
17
18 (Primes upTo: 30) printNl .
19 (Primes upTo: 50) do: [ :p | p print . Transcript space ] .
20 Transcript cr
```

Tratamento padrão de erros

- O mecanismo padrão de SmallTalk-80 para sinalizar erros é o método `error`
- Ele recebe como argumento a mensagem que será exibida caso o erro ocorra
- Se invocado, ele finalizará a execução do programa imediatamente
- Uma série de mensagens de erros, iniciada pela mensagem passada como parâmetros, indicarão os possíveis erros
- Algumas informações nestas mensagens podem ser utilizadas por ferramentas de diagnóstico
- Contudo, às vezes o erro a ser sinalizado por não ser fatal, permitindo que a execução do programa continue
- Nestas situações, SmallTalk permite o uso de exceções

Exceções

- Em SmallTalk, todas as exceções são subclasses da classe `Exception`
- O método `printHierarchy` da classe `Exception` lista todas as exceções já definidas pela linguagem
- Para tratar uma exceção, as mensagens `on` e `do` devem ser passadas para o bloco que gera a exceção
- O primeiro indica a exceção que será tratada
- O segundo recebe como parâmetro o bloco de comandos que tratará a exceção
- Há seis ações básicas para o tratamento de exceções, todas elas passadas como mensagens para a instância da exceção

Exemplo de tratamento de exceções em SmallTalk

```
1 Object subclass: Math [
2
3     Math class >> harmonicMean: xs [
4         | h |
5         h := 0 .
6         xs do: [ :x | h := h + (1 / x) ] .
7         ^(1 / h)
8     ]
9 ]
10
11 (Math harmonicMean: #(1 2 3)) printNl .
12 [ Math harmonicMean: #(0 1 2 3) ] on: ZeroDivide do:
13     [ Transcript show: 'Média não definida' . Transcript cr ]
```

Ações básicas de tratamento de exceções

- `return`: encerra o bloco passado para o método `do`, retornando o valor indicado. Se o valor for omitido, o retorno será igual a `nil`
- `retry`: reinicia a execução do bloco que gerou a exceção. Deve ser usado com cuidado, pois pode gerar um laço infinito
- `retryUsing`: reinicia a execução usando o bloco indicado, ao invés do bloco original
- `pass`: repassa a exceção para o objeto que está acima do objeto original na hierarquia. Equivale ao `rethrow` de outras linguagens
- As outras duas ações básicas são `resume` e `outer`
- Se nenhuma das seis forem utilizadas, SmallTalk assumirá que o tratamento será `sig return`
- Contudo, o indicado é que o tratamento seja dado explicitamente
- Para tratar mais de uma exceção com o mesmo bloco, separe com vírgulas os argumentos de `on`

Exemplo de tratamento de execeções

```
1 Object subclass: Math [
2
3     Math class >> inverses: xs [
4         | i is |
5         i := 1 .
6         is := OrderedCollection new .
7         [
8             [i <= (1 + xs size)] whileTrue:
9                 [ is addLast: 1 / (xs at: i) . i := i + 1 ]
10        ] on: ZeroDivide, SystemExceptions.IndexOutOfRange do:
11            [ :sig | is addLast: nil . i := i + 1 . sig retry ] .
12        ^is
13    ]
14
15 ]
16
17 (Math inverses: #(-2 0 -1 0 1 0 2 0 3)) printNl
```

Referências

1. GNU Operation System. GNU Smalltalk Tutorial, acesso em 08/09/2020.
2. **RATHMAN**, Chris. Smalltalk notes, acesso em 08/09/2020.
3. Savannah. GNU Smalltalk – Resumo, acesso no dia 08/09/2020.
4. **SHALOM**, Elad. *A Review of Programming Paradigms Throughout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.