Programação Orientada a Objetos - Aula 11:

More *Smalltalk Goodies* - assuntos *Smalltalkísticos* variados

Raphael Mendes de O. Cóbe
raphael@ccsa.ufrn.br

Instituto de Matemática e Estatística - IME
Universidade de São Paulo - USP

Conteúdo

Classes e Métodos Abstratos

Classes Abstratas:

- Classes que somente existe para que possam ser herdadas.
- Nunca podem ser Instanciadas;
- Normalmente são consideradas como classes não completas;

Um pouco sobre projeto de software

- Trechos de código que somente serão especializados futuramente;
- Utilização em padrões de projeto como *Strategy* e *Template Method*;
- Operações específicas nas especializações;
- Subclasses possuem a responsabilidade de implementar esses métodos;

Um pouco sobre projeto de software - Exemplos

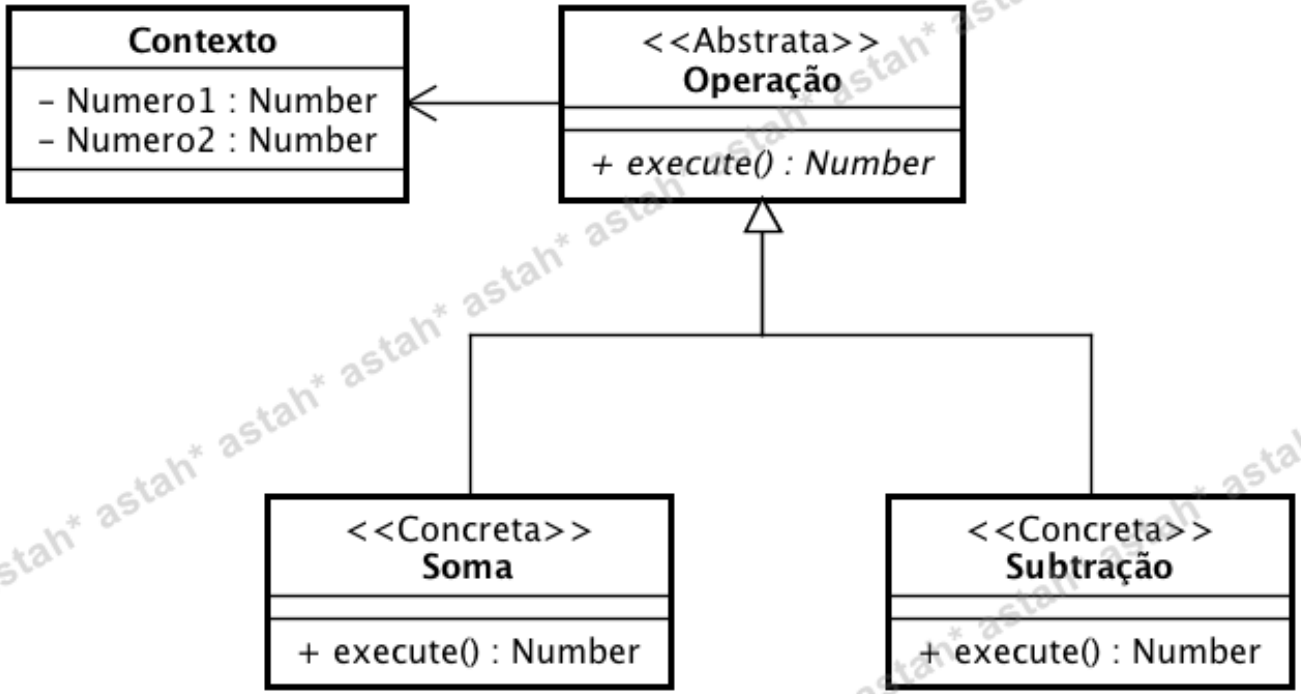


Figura: Estratégias

Implementação em *Smalltalk* I

- *Smalltalk* não possui sintaxe exclusiva para tornar uma classe ou método abstrato;
- Por convenção, o corpo do método abstrato possui uma chamada a operação `subclassResponsibility` (*Marker Methods*);
- Implementação na classe *Object*;
- Uma classe é considerada abstrata se algum de seus métodos é abstrato;

Implementação em *Smalltalk* II

- Métodos abstratos nunca deveriam ser invocados!
- Nada impede instanciação de classes abstratas;
- Problemas ocorrem ao se enviar mensagens (abstratas) para o objeto.

Mostrar implementação do `subClassResponsibility`;

Implementação em *Smalltalk* III

- Exemplo dentro da implementação do *Squeak*;
- A classe Magnitude torna responsabilidade de suas subclasses a implementação da operação $<$;
- Operações de comparação ($<=$, $>=$, $>$, `max:` e `min:`) são baseadas na operação $<$;

Mostrar implementação do $<$ em Magnitude e em Character;

Tratamento de Erros I

- Um mecanismo em OO para tratar casos **excepcionais** de forma elegante;
- Evita testar códigos de retorno ao executar operações;
- **Escrita de código assumindo a execução bem sucedida!**
- Separação de tratamento de Erros de "Regras de Negócio";
- Separação de erros por tipos;
- Tratamento específico para cada tipo de erro;

Tratamento de Erros II

- Não use exceções para identificar casos que não representam um erro:
  - Uma consulta a um banco de dados **não deveria** gerar uma exceção caso o elemento não está no banco, ela deveria simplesmente informar isso;
  - **Erro de acesso** ao banco ou um erro de leitura no disco, deveria sim gerar uma exceção;

Implementando e Tratando Exceções em *Squeak Smalltalk*

- Exceções são subclasses de *Exception*;
- *Squeak* vem com algumas exceções pré-definidas (73 ao todo!):
  - *Error*;
    - Exceções que não podem ser resumidas;
    - É aberto o depurador;
  - *Notification*;
    - Exceções que podem ser continuadas;
    - Mensagem exibida pra o usuário que pode apertar Ok!;
- Operação `isResumable`;
- Atributos *messageText*, *signalContext*, *handlerContext* e *outerContext*;

Mostrar hierarquia de Exception;
Mostrar exemplo com Warning;
Mostrar exemplo com Error;

Implementando e Tratando Exceções em *Squeak Smalltalk*

- Para capturar exceções:
  - Colocar o trecho de código dentro de um bloco e enviar uma das seguintes mensagens: `on: classeDaExceção do: tratamentoDaExceção`
  - Exceção é passada como parâmetro para o Bloco;
  - Parâmetros para o seletor `on:` podem ser passados separados por vírgula;
  - Envio da mensagem `ifError: BlocoDeTratamento`;

Implementando e Tratando Exceções em *Squeak Smalltalk*

Tratando Exceções com on:do:!

```
[[10 / 0]
  on: ZeroDivide
  do: [:exception | Transcript show: 'Divisão por zero']]
  on: Error
  do: [:exception | Transcript show: 'Qualquer outro Erro!'].
```

Tratando Exceções com ifError: !

```
[10 / 0]
  ifError: [:message |
    (message = 'ZeroDivide')
      ifTrue:[Transcript show: 'Divisão por Zero']
      ifFalse:[Transcript show: 'Qualquer outro Erro!']
  ].
```

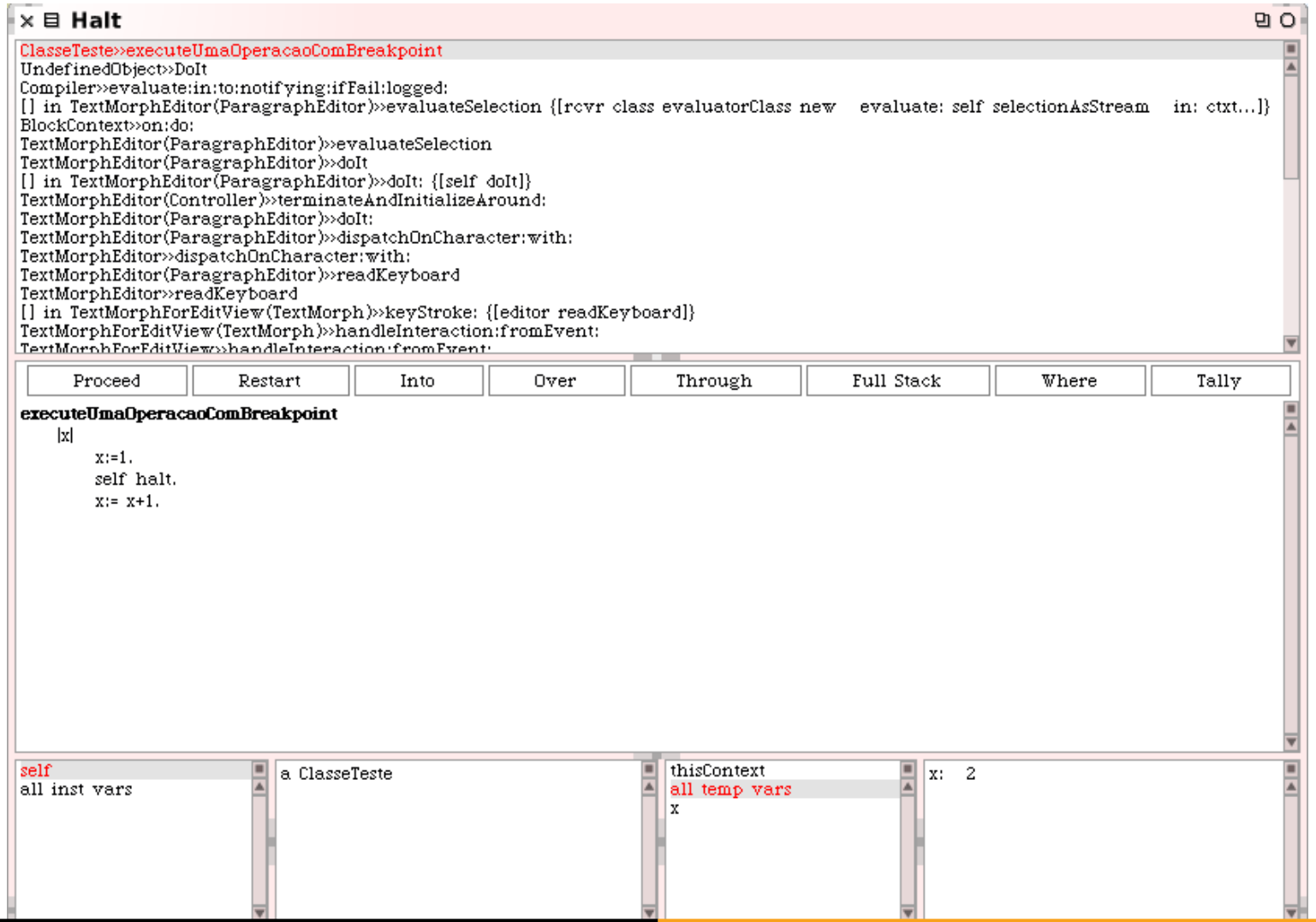
Mostrar Exemplo com ClasseTeste;

O depurador do *Squeak Smalltalk*

- O depurador de Smalltalk é muito sofisticado;
- **Ferramenta tão poderosa que alguns programadores experientes preferem desenvolver seu código dentro dele!**
- Opção Debug it para executar um trecho de código dentro do depurador;
- O método halt no ponto onde desejamos parar (*Breakpoints*);
- Normalmente é utilizado junto do inspetor de Objetos;

Mostrar Exemplo `executeUmaOperacaoComBreakpoint;`

O depurador do *Squeak Smalltalk*



O Inspetor de Objetos do *Squeak Smalltalk*

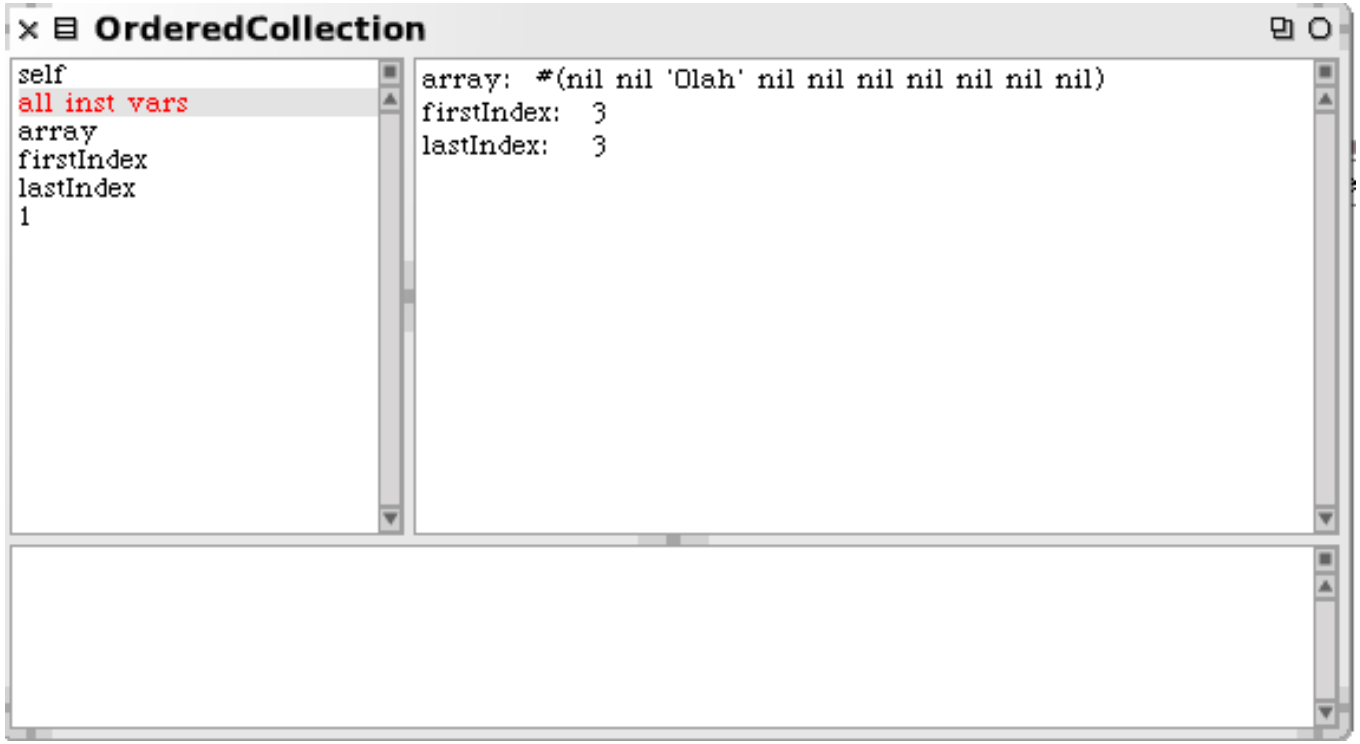


Figura: Inspetor de Objetos

Mostrar Exemplo `executeUmaOutraOperacaoComBreakpoint`.
Mostrar Exemplo Array.

Ponteiros e Referências I

- Referência: variável que guarda o endereço de memória onde um objeto reside;
- Podemos atribuir a ela um novo valor no momento da criação de um novo objeto;
- Fazer com que seja associada ao mesmo objeto de outra referência;
- Não nos preocupamos com como se encontra a memória;
- Comparar referências para saber se elas apontam para o mesmo objeto;
- Java e Smalltalk;

Ponteiros e Referências II

- Apontadores: possível fazer as mesmas operações com referências
- Também possível fazer:
  - aritmética de ponteiros;
  - atribuir valores diretamente a ponteiros (Cuidado!);
  - C/C++;

Comparações de Referências e Valores

- Dois objetos são equivalentes (=) se possuem o mesmo conteúdo (O equals() de Java);
- Duas referências são iguais (==) se apontam para o mesmo objeto(O == de Java);
- O contrário de = é $\sim=$
- O contrário de == é $\sim\sim$

Igualdade Vs. Equivalência

```
p1 := (Point x: 10 y: 20)
p2 := (Point x: 10 y: 20)
Transcript show: (p1 =  p2)
Transcript cr; show: (p1 == p2)
```

Mostrar Exemplo dos Pontos.

Utilização de Caracteres especiais para a criação de Objetos

- Criação de instâncias de *Point* pode ser feita com o operador @ da classe número:
  - 3@5;
- Criação de instâncias de *Association* pode ser feita com o operador $->$ de *Object*:
  - 'casa' $->$ 'maison'

Associações

```
Dictionary new add: 'SP' -> 'São Paulo';
   add: 'RJ' -> 'Rio de Janeiro'; add: 'PR' -> 'Paraná'.
```

Mostrar Implementação da Operação @ em Number

Impressão de Objetos

- O método `printString` é a forma padrão de converter um objeto qualquer para um *String*;
- Equivalente ao `toString()` de Java;
- Implementar o método `printOn: aStream` que é chamado pelo `printString` de *Object*.

Mostrar o método `printOn` de String Mostrar Exemplo de `printOn:`