Programação Orientada à Objetos

Fundamentos

Prof. Edson Alves

Faculdade UnB Gama

Sumário

Introdução

- A **programação orientada a objetos** é um paradigma de programação baseada em objetos
- **Objetos** são estruturas de dados que armazenam informações (**atributos**) e procedimentos (**métodos**)
- Os programas consistem em interações entre os diferentes objetos
- Estas interações são conduzidas por meio de trocas de **mensagens**
- Os objetos são instâncias de **classes**, as quais determinam os tipos, atributos e métodos dos objetos
- Muitas linguagens multiparadigmas suportam a orientação a objetos
- Algumas outras, como Smalltalk, suporta apenas o paradigma da programação orientada a objetos

Histórico

- O termo objeto, no contexto de programação orientada a objetos, surgiu no MIT no final dos anos 1950 e no início dos anos 1960
- Ele se referia a um item identificado com atributos
- Também no MIT, Ivan Sutherland desenvolveu um software denominado Sketchpad, ancestral dos programas CAD, e definiu os termos objeto e instância
- A linguagem Simula67 introduziu o conceito formal de objetos em programação e também a noção de classes
- A expressão "programação orientada a objetos" foi introduzida pela linguagem Smalltalk, nos anos 1970
- Esta linguagem faz uso de objetos e mensagens como base de suas computações, e as classes podem ser modificadas dinamicamente
- Nos anos 1990 este paradigma se tornou a metodologia dominante de desenvolvimento de software, obtendo suporte na maior parte das linguagens então existentes

Visão geral

- Além de unir dados e funções, os objetos também são **tipos de dados abstratos**, isto é, são definidos por suas interfaces, e não por suas implementações
- Os objetos também oferecem suporte para **polimorfismo** e para **herança**
- Os objetos são organizados em hierarquias de **classes**
- As classes e as **subclasses** são análogos aos conjuntos e subconjuntos da matemática
- Os objetos são **modulares**, no sentido que são responsáveis por seus dados e comportamentos
- Esta característica é denominada **encapsulamento**

Visão geral

- Na programação orientada a objetos os dados devem, sempre que possível, ser protegidos de acessos diretos
- O acesso aos dados do objetos deve ser feito por meio dos métodos de sua **interface**
- Outro aspecto importante deste paradigma é o **reuso** de código
- São duas as principais formas de reuso: construir novos objetos através da **composição** de objetos já existentes ou estender um objeto por meio de herança
- Um objeto (ou módulo) está **aberto** se ele permite sua extensão por herança
- Ele está **fechado** se sua interface é estável e bem definida, de modo que não pode (ou deve) ser estendido

GNU Smalltalk

- GNU Smalltalk é uma implementação livre da linguagem Smalltalk-80
- Esta está disponível na maioria das distribuições Linux e outras plataformas que suportam o padrão POSIX
- Ela utiliza a licença GNU GPL v2 em algumas partes e a GNU Lesser GPL v2 em outras
- Ela pode ser instalada através do comando

```
$ sudo apt-get install gnu-smalltalk
```

- Para validar a instalação, pode-se utilizar o comando

```
$ gst --version
```

- O ambiente interativo pode ser iniciado com o comando

```
$ gst
```

Hello World!

- Para imprimir a mensagem '*Hello World!*' no ambiente interativo basta invocar o método `printNl`

```
st> 'Hello World!' printNl
```

- A mensagem é impressa duas vezes: a primeira por conta do método `printNl`, a segunda por conta do ambiente interativo, que imprime o valor resultante da expressão inserida
- Observe que o texto antecede o método
- Isto por que, em Smalltalk, a sintaxe para passar uma mensagem para um objeto é

```
object message: par1 par2 ... parN
```

- Note a ausência de parêntesis e da notação que utiliza o ponto final, comum em muitas outras linguagens que suportam a programação orientada à objetos

Operadores lógicos e aritméticos

- Em Smalltalk, todos os tipos são objetos
- O código abaixo ilustra o polimorfismo do método `printNl`

```
st> 7 printNl
```

- Por conta do comportamento do ambiente interativo, o método `printNl` será omitido dos demais código
- A linguagem tem suporte para os operadores lógicos

```
st> true & false
st> true | false
st> true not
```

- E também para os operadores aritméticos

```
st> 2 + 2
st> 2 - 5
st> 2 * 8
st> 8 / 2
```

Vetores

- Em Smalltalk a sintaxe para a declaração de vetores é

```
Array new: N
```

  onde N é a quantidade de elementos
- Inicialmente, todos os elementos estão vazios (`nil`)
- O tamanho do vetor é fixo, e não pode ser alterado
- A mensagem `at: i` permite o acesso ao elemento que ocupa a i-ésima posição

```
st> x := Array new: 5
st> x
(nil nil nil nil nil )
st> x at: 1
nil
```

- Para modificar o valor do elemento que ocupa a i-ésima posição, é preciso combinar as mensagens `at: i` e `put: value`

```
st> x at: 1 put: 8
```

Vetores

- Ao contrário de C e outras linguagens, Smalltalk segue o padrão da matemática e atribui a posição 1 ao primeiro elemento do vetor
- Os elementos do vetor pode ser de quaisquer tipos

```
st> x at: 2 put: true
st> x at: 3 put: 'Teste'
st> x at: 4 put: 1/2
st> x at: 5 put: 1/2 asFloat
```

- Em Smalltalk, os comentários são delimitados por aspas duplas (")
- A tentativa de acessso a um índice fora do intervalo $[1, N]$ resulta em um erro do tipo '*index out of range*'
- Um vetor também pode ser construído com a sintaxe

```
#(elem1 elem2 "..." elemN)
```

- A mensagem `size` retorna o número de elementos do vetor

```
st> x := #(1 true 'String')
st> x size
```

Conjuntos

- Em Smalltalk o conjunto (`set`) representa uma coleção de elementos únicos
- A sintaxe para a declaração de um conjunto é

```
Set new
```

- A mensagem `add: value` adiciona o valor indicado ao conjunto
- Várias expressões podem ser concatenadas para serem executadas simultaneamente por meio do operador ponto final (`.`)

```
st> x := Set new
Set ()
st> x add: 1 . x add: 2 . x add: 3 . x add: 1
st> x
Set (1 2 3 )
```

- Se as várias mensagem tem o mesmo objeto como destinatário, a notação de ponto pode ser simplificada por meio do uso do operador ponto-e-vírgula (`;`)

```
st> x add: 1; add: 2; add: 3; add: 1
```

Conjuntos

- Para testar se o elemento y pertence ao conjunto, basta usar a mensagem `includes: y`

```
st> x includes: 1
true
st> x includes: 4
false
```

- A mensagem `remove: y` permite remover um elemento do conjunto

```
st> x remove: 2
st> x
Set (1 3 )
```

- Se y não pertence ao conjunto, será emitido um erro do tipo '*object not found*'

Dicionários

- Os dicionários permitem armazenar elementos (valores) indexados por chaves
- A sintaxe para a declaração de um dicionário é

```
Dictionary new
```

- Tanto as chaves quanto os valores podem ser de tipos arbitrários

```
st> x at: 1 put: 'Teste'
st> x at: true put: false
Dictionary (
    1->'Teste'
    true->false
)
```

- A mensagem `includes: y` verifica se y está entre os valores do dicionário, enquanto que a mensagem `includesKey: z` procura z entre suas chaves

```
st> x includes: false
true
st> x includesKey: false
false
```

Objetos e referências

- Em Smalltalk, todos os tipos de dados são **objetos**
- Objetos podem ser **instanciados** por meio de constantes ou da mensagem `new`

```
st> x := 7
st> y := Set new
```

- A atribuição faz com que a variável **referencie** o objeto construído à direita
- Assim, uma mesma variável pode mudar sua referência por meio de nova atribuição

```
st> x := false
st> x := 'Teste'
```

- Quando um objeto fica sem referência, sua memória será **desalocada** pelo *garbage collector* em algum momento
- O operador **exclamação** (`!`) libera a referência

```
st> x := 9
st> !x
nil
```

Mensagens e métodos

- Em Smalltalk, as **mensagens** são o principal meio de comunicação entre os diferentes objetos
- Uma mensagem é composta por um **método** da classe, seguido de dois pontos (`:`) e seus parâmetros, separados por um espaço em branco, caso existam
- Uma série de mensagens cujo destinatário é o mesmo objeto podem ser concatenadas por meio do operador **ponto-e-vírgula** (`;`)
- O valor da expressão será o retorno da última mensagem

```
st> x := Array new: 10
st> x at: 1 put: 10; at: 2 put: 5; includes: 7
false
```

- Mensagens para objetos distintos podem ser concatenadas por meio do operador **ponto final** (`.`)

```
st> x := Set new . y := Dictionary new
```

Classes

- Em Smalltalk, as **classes** são organizadas em um árvore hierárquica, cuja raiz é a classe `Object`
- A sintaxe para a declaração de uma classe é

```
Object subclass: #NomeDaClasse
```

- Ou seja, uma nova classe é criada enviando-se a mensagem `subclass` para a classe **Object**
- **Variáveis de instância** são criadas por meio da mensagem `instanceVariableNames`

```
st> Object subclass: #Polygon
st> Polygon instanceVariableNames: 'sides'
```

- A mensagem `comment` pode ser utilizada para documentar a classe

```
st> Polygon comment: 'Exemplo de classe em Smaltalk'
st> Polygon comment
"'Exemplo de classe em Smaltalk'"
```

Classes

- Os métodos podem ser redefinidos, se necessário

```
st> Polygon comment: 'Exemplo de classe em Smalltalk'
st> Polygon comment
"'Exemplo de classe em Smalltalk'"
```

- Há outra sintaxe, mais próxima da utilizada em outras linguagens, para a definição de uma classe:

```
Object subclass: Polygon [
    | sides |
    <comment: 'Exemplo de classe em SmallTalk'>
]
```

- Uma classe já definida pode ser estendida para adicionar novos membros ou métodos

```
Polygon extend [
    | name |
]
```

Classes

- Um novo método de instância pode ser definido usando a sintaxe

```
NomeDaClasse extend [
    nomeDoMétodo [
        expressão1
        expressão2
        "..."
        expressãoN
    ]
]
```

- Para definir um método de classe, a palavra-reservada `extend` deve ser precedida da palavra reservada `class`

```
NomeDaClasse class extend [
    nomeDoMétodo [
        expressão1
        expressão2
        "..."
        expressãoN
    ]
]
```

Classes

- Para criar novas instâncias da classe, é preciso definir o método **construtor**
- O construtor é um método de **classe**, não de instância
- Em sua implementação, o construtor deve
  1. criar uma variável que conterá o espaço de memória da instância
  2. inicializar este espaço invocando o método `new` de sua superclasse
  3. inicializar as variáveis de instância da classe
  4. retornar a variável que contém a instância
- Para **retornar** um valor ao término da execução de um método, deve-se preceder o valor do retorno do operador '^'
- Se o método não tiver um retorno explícito, será retornada a própria instância por padrão
- Para definir como uma instância de uma classe será impressa, deve-se sobreescrever o método `printOn`

Exemplo de implementação de métodos de classe e instância

```
1  Polygon class extend [
2      new: n [
3          | r |
4          r := super new . r init: n . ^r
5      ]
6  ]
7
8  Polygon extend [
9      init: n [
10         sides := n
11     ]
12
13     sides [
14         ^sides
15     ]
16 ]
```

Definição completa da classe Polygon

```
Object subclass: Polygon [
    | sides |
    <comment: 'Exemplo de classe em SmallTalk'>

    Polygon class >> new: n [
        | r |
        r := super new . r init: n . ^r
    ]

    init: n [ sides := n ]

    getSides [ ^sides ]

    printOn: stream [
        super printOn: stream .
        stream nextPutAll: ' with ' .
        sides printOn: stream .
        stream nextPutAll: ' sides'
    ]
]
```

Subclasses

- A sintaxe para declarar uma **subclasse** em Smalltalk é

```
ClasseBase subclass: SubClasse [
    " Implementação da subclasse "
]
```

- As subclasses herdam os métodos e membros da classe base
- Elas também podem ter membros e métodos específicos
- Se necessário, um ou mais métodos da classe base podem ser **sobrescritos**, dando a eles uma nova implementação
- Esta nova implementação **prevalecerá** em relação à implementação herdada do pai, no momento da invocação
- Os métodos da classe base podem ser invocados na subclasse por meio da palavra reservada `super`
- Métodos da própria classe podem ser invocados usando a palavra reservada `self`

Exemplo de subclasse em Smalltalk

```
Polygon subclass: Square [
    | l |
    <comment: 'Exemplo de subclasse em SmallTalk'>

    Square class >> side: length [
        | r |
        r := super new: 4 . r setLength: length . ^r
    ]

    setLength: length [ l := length ]

    area [ ^(l * l) ]
    perimeter [ ^(self getSides * l) ]

    printOn: stream [
        stream nextPutAll: 'a Square with side ' .
        l printOn: stream
    ]
]
```

Outro exemplo de subclasse em Smalltalk

```
Polygon subclass: Rect [
    | b h |

    Rect class >> base: B height: H  [
        | r |
        r := super new: 4 . r setBase: B setHeight: H . ^r
    ]

    setBase: base setHeight: height [
        b := base . h := height
    ]

    area [ ^(b * h) ]

    printOn: stream [
        stream nextPutAll: 'a Rect with base ' .
        b printOn: stream .
        stream nextPutAll: ' and height ' .
        h printOn: stream
    ]
]
```

Polimorfismo

- Na orientação a objetos, o **polimorfismo** permite utilizar uma referência a uma instância de uma subclasse onde se espera uma referência de uma classe base
- Na maior parte das linguagens, uma **referência** (ponteiro) para a classe base pode apontar para uma instância de qualquer subclasse desta classe base
- Em Smalltalk as variáveis e referências não tem tipo explícito, de modo que esta característica não fica evidente como nas demais linguagens
- Ainda assim, é possível observar o comportamento polimórfico das instâncias de subclasses que **sobreescreveram** métodos da classe base
- Importante ressaltar que, no processo de sobrescrita de um método da classe base, a **assinatura** do mesmo (nome e lista de parâmetros) deve permanecer a mesma

Exemplo de polimorfismo em Smalltalk

```
" Este código pode ser executado via terminal:
"
"   $ gst polimorfismo.st
"
Object subclass: Printer [

    Printer class >> print: x [
        Transcript show: '++ ', x printString; cr
    ]

]

FileStream fileIn: 'polygonclass.st' .
FileStream fileIn: 'square.st' .
FileStream fileIn: 'rectangle.st' .
p := Polygon new: 5 .
s := Square side: 3 .
r := Rect base: 1 height: 2 .
Printer print: p .
Printer print: s .
Printer print: r
```

Sobrecarga

- Outra importante característica da orientação a objetos é a **sobrecarga** de métodos
- A sobrecarga permite escrever vários métodos com o **mesmo nome**, porém com **parâmetros distintos**
- A sintaxe para definição de métodos em Smalltalk permite a sobrecarga, mas assim como no caso do polimorfismo, o conceito não fica explícito como em outras linguagens
- Os métodos construtores, que inicializam uma instância de uma classe, fornecem bons exemplos deste conceito na prática
- Cada construtor pode ter um número **distinto** de parâmetros
- De fato, como Smalltalk associa um **nome** a cada parâmetro, o qual deve ser usado na invocação do construtor, uma escolha adequada para estes nomes pode aumentar a **legibilidade** do código

Exemplo de sobrecarga em Smalltalk

```
Object subclass: Point [
    | x y |

    Point class >> x: xcoord y: ycoord [
        | r |
        r := super new . r setX: xcoord setY: ycoord . ^r
    ]

    Point class >> new [
        | r |
        r := super new . r setX: 0 setY: 0 . ^r
    ]

    setX: xcoord setY: ycoord [ x := xcoord .  y := ycoord ]

    printOn: stream [
        super printOn: stream .  stream nextPutAll: ' at (' .
        x printOn: stream .  stream nextPutAll: ', ' .
        y printOn: stream .  stream nextPutAll: ')'
    ]
]
```

gst, arquivos e imagens

- Um arquivo pode ser carregado no gst por meio do método `fileIn` da classe `FileStream`:

```
gst> FileStream fileIn: 'source.st'
```

- Todas as definições já inseridas em uma sessão do gst podem ser salvas em um arquivo imagem, através do método `snapshot` da classe `ObjectMemory`

```
ObjectMemory snapshot: 'nomedaimagem.im'
```

- Esta imagem pode ser carregada na inicialização do gst:

```
$ gst -I nomedaimagem.im
```

- Uma vez carregada a imagem, a sessão anterior fica reestabelecida, no que diz respeito as **definições** das classes feitas anteriormente
- Atribuições à variáveis **não são salvas**

Referências

1. GNU Operation System. GNU Smalltalk, acesso em 25/03/2020.
2. Pharo by Example. Chapter 9 – Collections, acesso em 30/03/2020.
3. **RATHMAN**, Chris. Smalltalk notes, acesso em 30/03/2020.
4. Savannah. GNU Smalltalk – Resumo, acesso no dia 25/03/2020.
5. **SHALOM**, Elad. *A Review of Programming Paradigms Througout the History – With a Suggestion Toward a Future Approach*, Amazon, 2019.