Avaliação da metodologia de Programação Orientada a Objetos em Java de Isaias Camilo Boratti

Resumo

O presente documento apresenta um resumo da metodologia para ensino de programação orientada a objetos adotada por Isaias Camilo Boratti. Primeiro é apresentada uma tabela apresnetado os exemplos e conceitos trabalhos na metodologia e a seguir uma análise dos pontos fortes e fracos a partir da

Introdução

Esta metodologia foi vivenciada pelo autor deste trabalho no primeiro semestre de 2007, no curso de Sistemas de Informação da Universidade Federal de Santa Catarina, na disciplina de Programação Orientada a Objetos I.

A disciplina foi ministrada pelo próprio professor Isaias Camilo Boratti que, no mesmo ano, em agosto, publicou o livro que documenta a metodologia intitulado “Programação Orientada a Objetos em Java”.

O livro está organizado em oito capítulos conforme tabela abaixo:

Capítulo	Exemplos de Projetos Desenvolvidos	Principais Conceitos abordados
Capítulo 1	*Nenhum projeto desenvolvido*	• Relação POO com o mundo real; • Processo de Abstração; • Operações de Abstração.
Capítulo 2	*Nenhum projeto desenvolvido*	• Objetos e Classes; • Representação visual de objetos e classes; • Linguagem Java;
Capítulo 3	• Imprimir na tela uma linha de texto; • Calcular a soma de dois números; • Escrever um programa que determine a área de um círculo;	• Método main; • Identificadores; • Tipagem de valores; • Variáveis; • Criação de objetos; • String / sequência de caracteres; • Declaração de atributos; • Declaração de métodos; • Construtores; • Parâmetros e Argumentos; • Compilação Java; • Processo de modelagem de um software a partir de um problema;
Capítulo 4	• Escrever uma aplicação que determine a distância entre dois pontos;	• Implementação de métodos; • Métodos de acesso e métodos modificadores; • Implementação de construtores;

		• Escopo; • Comentários; • Parâmetros; • Implementação de objetos / referência para um espaço de memória; • Tipagem de valores; • Atribuição; • Conversão explícita de tipo; • Expressões aritméticas; • Expressões Lógicas;
Capítulo 5	• Escrever um programa que determine a idade média de um grupo de pessoas; • Escrever um programa que determine a média de idade das pessoas do sexo masculino e também a média de idade das pessoas do sexo feminino de um grupo de pessoas; • Classe Aluno; • Classe Triângulo; • Verificar se um número é primo;	• Processos de repetição / *loops*; • Processos de tomada de decisão. Ex: *if*, *switch;* • Encapsulamento; • Modificadores de acesso; • Palavra chave *this*;
Capítulo 6	• Desenvolver uma aplicação que faça uma análise do desempenho de alunos em determinada disciplina. A aplicação deve determinar, para cada um dos alunos, a sua média final, juntamente com uma informação dizendo se o aluno foi aprovado ou não. Considerar que o aluno tenha realizado três avaliações, todas com o mesmo peso; • Em um determinado jogo de dados participam dois jogadores. Cada jogador deve lançar o dado e computar para si os pontos obtidos. Será considerado vencedor o jogador que atingir um total de 100 pontos ou mais. Sempre que no lançamento do dado, o jogador obter o valor 1 ou 6, este terá direito a um novo lançamento; • Modelagem das classes *Funcionario*, *Chefe* e *Apoio*, sendo as duas últimas subclasses de *Funcionario*; • Modelagem da classe abstrata Contribuinte e das subclasses *Fisico* e *Juridico*.	• Herança; • A palavra-chave *super*; • Reutilização de código; • Sobreposição de métodos; • Polimorfismo; • Polimorfismo de método; • Sobrecarga de métodos; • Classes abstratas;
Capítulo 7	• “Desenvolver uma aplicação que avalie o desempenho de um grupo de estudantes em determinada disciplina...que prevê a realização de vários testes e uma prova final...a média dos testes terá peso de 60% e a prova final peso de 40%.... ”	• Arranjos / *Arrays;* • *Arrays* Multidimensionais; • Operações com *arrays*.
Capítulo 8	• Objetos e Classes; • Representação visual de objetos.	• Atributos de classe; • Métodos de classe;

		• Métodos da classe *Math*; • Manipulação de Strings; • Classe *Character*;

Análise Geral do Método

A metodologia inicia demonstrando que o mundo real "é constituído por entidades que interagem entre si", e que tais entidades possuem características e funções, podendo também serem chamadas de objetos.

Dessa forma, a construção de um software orientado a objetos passa pelo processo de identificação dos objetos envolvidos e de como estes objetos irão se relacionar entre si para a resolução de um determinado problema.

A partir disto, Boratti conduz o estudante pelo conceito de abstração, que corresponde:

> ao processo utilizado na análise de determinada situação, através da qual observa-se uma realidade, tendo-se por objetivo a determinação dos aspectos e fenômenos considerados essenciais, excluindo-se todos os aspectos considerados irrelevantes.

E a partir de problemas simples como *"determinar o raio de um círculo"* demonstra como o problema pode ser modelado em termos de objetos com características e funções interagindo entre si, modela as classes que representam estes objetos e apresenta sua respectiva implementação em Java.

Os conceitos chave são demonstrados na medida em que são úteis para resolver problemas mais complexos e sub seções complementares se encarregam de determinados aspectos da linguagem Java como operadores matemáticos e detalhes de sintaxe, sem necessariamente, serem parte da solução dos problemas apresentados.

Ao final de cada seção, é apresentado um resumo dos conceitos e conhecimentos trabalhados, uma lista de exercícios, geralmente extensa e abrangente, nos casos que apresentem conteúdo prático, diversos exercícios resolvidos também são fornecidos.

Por fim, ressalta-se que todos os códigos construídos por Boratti são em língua portuguesa e sempre que apresentam recursos novos, são comentados com descrições detalhadas do seu comportamento.

Pontos Fortes

Operações de Abstração

No capítulo introdutório da metodologia, Boratti traz o conceito de operações de abstração que *"mostram como o ser humano mentaliza, organiza e modela o mundo ao seu redor"*. Estas operações são úteis na medida em que definem uma taxionomia para as interações entre as entidades no mundo real e podem ser representadas diretamente através de código orientado a objetos. Logo, o processo de analisar uma interação entre entidades no mundo real e o código que deve ser construído para resolver problemas envolvendo esta interação pode ser mais adequadamente identificado.

As Operações de abstração são demonstradas utilizando representações parecidas com o utilizado pelo diagrama de classes da UML para as classes identificadas.

Segue abaixo as quatro operações de abstração:

- **Classificação / Instanciação**: Classificação corresponde a determinar um conjunto de características de um determinado grupo de objetos de modo a classificá-los em uma categoria ou classe. Instanciação é o processo inverso, no qual exemplos (instâncias) de

determinadas categorias são ressaltados. Por exemplo a Classe *Carro* e Instância *carro vermelho do josé*;

- **Generalização e Especialização**: Generalização ocorre quando determinamos características comuns a conjunto de classes e assim, criamos uma classe mais genérica. Especialização é o inverso, quando a partir de uma classe genérica, determinamos classes especialistas. Por exemplo, a classe genérica *Médico* poderia ser especializada em *Obstetra*, *Pediatra* e *Ornitolaringologista*;
- **Agregação / Decomposição**: A Agregação é caracterizada pela relação de "é composto por" entre duas classes. A decomposição é quando dizemos que um objeto "é parte de" outro objeto. Por exemplo, a classe *Carro* é composta por objetos da classe *Motor*. Já um objeto da classe Motor é parte de um objeto da classe *Carro*;
- **Associação**: Ocorre quando dois objetos possuem algum tipo de ligação entre si mas podem existir independentemente do outro, Por exemplo, na relação que ocorre entre objetos das classes Professor e Aluno.

Processo de Modelagem

Boratti coloca que o modelo de resolução de um problema, ou seja, o software, é o resultado de um processo de abstração que em orientação a objetos corresponde à identificação dos objetos com suas características e funções e a respectiva modelagem das suas classes.

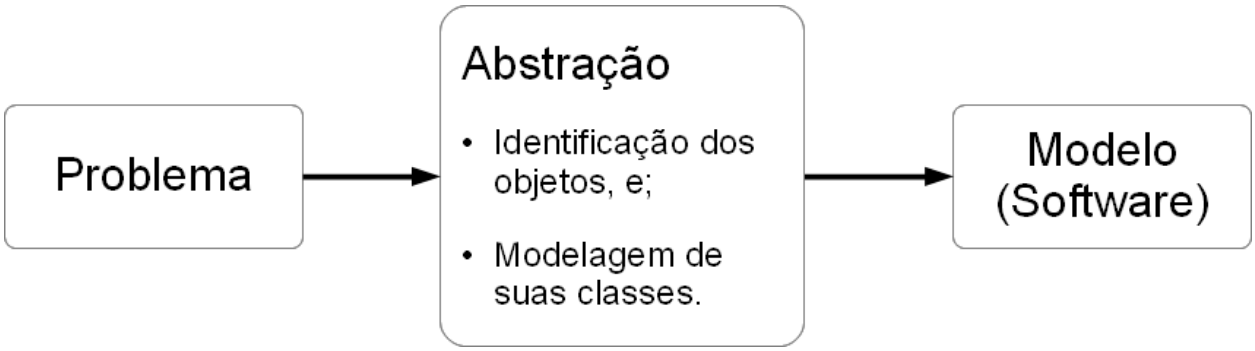

Ilustração 1: A Construção do modelo. Fonte: Boratti

Representação Visual de Objetos

Após a identificação dos objetos é necessário representá-los visualmente pelo projetista de software. A visualização proposta por Boratti utiliza um círculo para representar um objeto com um texto dentro representando o identificador deste objeto. Quando um objeto é composto por outros objetos estes são representados como círculos dentro do círculo maior. No caso de atributos simples de um objeto como números e textos, estes objetos são representados com um retângulo.

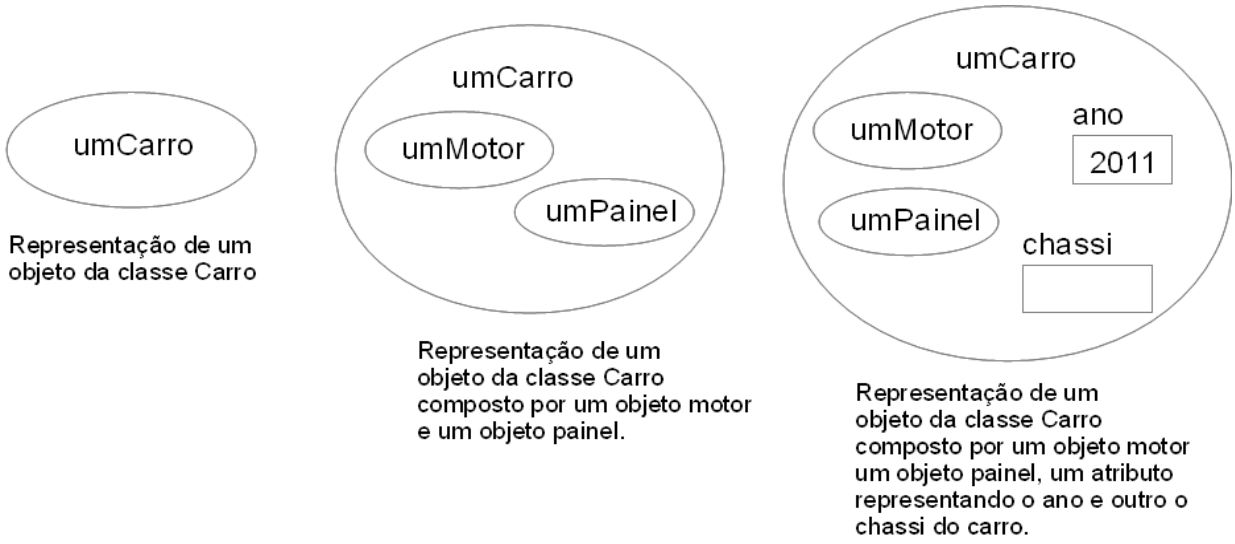

Ilustração 2: Representação Visual de Objetos. Fonte: Boratti. Adaptado por Kaléu Caminha

Implementação de Objetos

Recordo, como estuante do professor Isaías Boratti, que fortaleceu muito o meu aprendizado no momento que compreendi que cada objeto ocupava um espaço na memória e que uma variável, na verdade, era um apontador para este espaço na memória.

Este aspecto da implementação de objetos é tratada na seção 4.8 onde Boratti coloca que *"a identificação de um objeto constitui-se em uma posição na memória (portanto, uma variável) que armazena o endereço do espaço de memória ocupado por aquele objeto"*.

Para exemplificar este conceito o seguinte código é apresentado:

```
Ponto umPonto, outroPonto; // Um ponto contém um atributo inteiro x e um y
umPonto = new Ponto();
outroPonto = new Ponto();
outroPonto.redefina_se(2.0, 3.0); // este método altera os valores de x e y
```

E a partir do código acima, é utilizada uma adaptação da visualização de objetos demonstrando que as variáveis são apenas apontadores para os objetos:

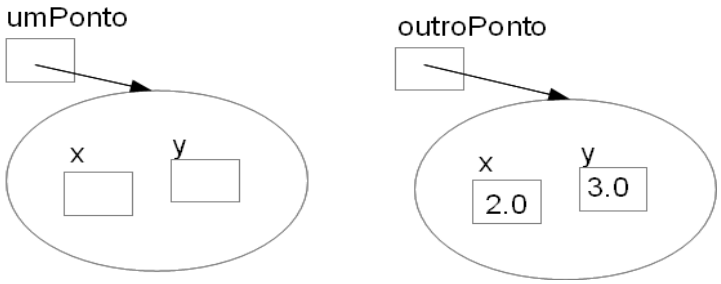

Ilustração 3: Implementação de Objetos. Fonte: Boratti. Adaptado por Kaléu Caminha

Representação Visual de Array

Arranjos, como são chamados os tradicionais Arrays por Boratti são "elementos que armazenam vários valores". Para facilitar o ensino foi criado um modelo visual para o uso de arrays. Para exemplificar o modelo usaremos o seguinte código:

```
int[] numeros = new int[8];
int cont = 2;
numeros[1] = 34;
numeros[cont] = 15;
numeros[cont+1] = numeros[cont] - 10;
```

O código acima apresentaria a seguinte representação visual:

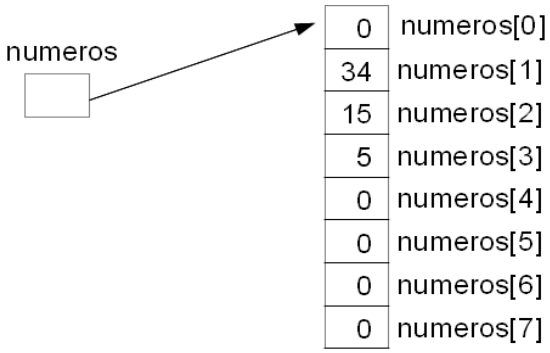

Ilustração 4: Representação Visual de um Array. Fonte: Boratti. adaptado por Kaléu Caminha

Pontos Fracos

Início pelo método main com Hello World

Após os dois primeiros capítulos de introdução, com uma explicação detalhada e abrangente sobre orientação a objetos e o processo de modelagem, o primeiro programa mostrado tem como problema "Imprimir uma linha na tela" e o seguinte código-fonte:

```
public class Programa1
{
	//Declaração do método main
	public static void main(String[] parametro)
	{
		System.out.println("Primeiro Programa!");
	}
}
```

O programa que resolve este problema não envolve a criação de nenhum objeto, e o único método chamado é um método acessado através de um atributo estático da classe System, usando notação de ponto.

E isto é feito dentro do método main, um método estático que exige como parâmetro um array de strings, conteúdos que somente serão trabalhado no capítulo sete.

É evidente que esta abordagem se torna necessária na medida em que Boratti, por não adotar nenhuma ferramenta que suporte a execução de código sem a criação do método main, não tem saída a não ser iniciar por ele.

Esta abordagem porém, não é interessante pois a primeira porção de código que o estudante enxerga é algo absolutamente incompreensível para ele, devido a grande quantidade de conceitos envolvidos (Kolling e Rosemberg).

Primeiro o código, depois o conceito

Em diversos exemplos, certas regras de sintaxe são apresentadas, cujos conceitos envolvidos somente são trabalhados em seções ou capítulos posteriores.

Um exemplo é a primeira classe com a qual os alunos tem contato. A classe *Círculo*, apresentada como o terceiro trecho de código com o qual os estudantes tem contato.

```
public class Circulo
{
	//Declaração do atributo
	protected double raio;

	//Declaração dos métodos
	public Circulo()
	{
		raio = 0.0;
	}

	public double forneceArea()
	{
		double area;
		area = 3.1415*raio*raio;
		return area;
	}

	public void recebaValorRaio(double vRaio)
	{
		raio = vRaio;
	}
}
```

Esta classe apresenta mais de 10 conceitos: comentários, declaração de atributos, tipagem de variáveis, declaração de método com retorno, declaração de método sem retorno, declaração de um construtor, assinatura de classe, parâmetros, atribuições, operações matemáticas e uso de modificadores de acesso.

A maioria destes conceitos só serão trabalhados posteriormente, por exemplo, atribuições serão detalhadas no capítulo 4 enquanto modificadores de acesso somente serão vistos no capítulo 5.

Percebe-se que esta abordagem tem como intenção demonstrar a modelagem completa de uma classe adequadamente escrita para resolver um problema. Isto fica claro pelo uso de modificadores de acesso desde o início, com a intenção de evitar que estudantes declarem atributos como públicos.

Isto causa, porém, uma sobrecarga de conceitos apresentados sem a devida possibilidade do estudante experimentar cada um deles e suas consequências para o programa, sem saber qual a função de cada palavra-chave.

Métodos estáticos desde o início

Para complementar o exemplo do capítulo 3 no qual é preciso determinar a área de um círculo a partir do seu raio, é apresentada uma classe chamada Interface (que não tem nenhuma relação com o conceito de interfaces em Java) responsável por solicitar ao usuário um número que irá representar o raio do círculo.

O principal método desta classe chamado “pegaValorRaio” contém 4 linhas, conforme apresentado abaixo:

```
 …
String valorLido;
valorLido = JoptionPane.showImputDialog(“Digite o raio: ”);
double valor = Double.parseDouble(valorLido);
return valor;
…
```

O problema deste método é que as principais instruções são métodos estáticos de classes específicos da biblioteca do Java. Isso nada ensina sobre orientação a objetos, sendo necessários apenas para que o programa seja minimamente interessante.

Construções como esta são utilizadas em outros pontos como no capítulo 4 com o uso do método sqrt da classe Math, responsável por recuperar a raiz quadrada de um número.

A grande crítica a esta abordagem é que métodos e atributos estáticos são ensinados apenas no capítulo 7, de modo que o estudante precisa simplesmente ignorar o fato da sintaxe ser diferente de outros objetos com os quais ele trabalha.

Já presenciei, como professor, estudantes que foram ensinados sob metodologias parecidas chamando métodos diretamente das classes, sem diferenciar se um método era ou não estático.

Conclusão

A metodologia se mostra muito interessante ao tornar claro para o estudante o processo de construção de um software, a sua respectiva modelagem de objetos e classes e as diversas representações visuais do programa.

Também se mostra interessante a forma de abordar os conceitos sempre que possível a partir de problemas do mundo real. Observa-se porém que a complexidade dos problemas tratados é muito pequena como por exemplo, calcular a média de um grupo de alunos ou determinar a área de um círculo.

Um dos motivos para o uso de exemplos triviais é a dificuldade em utilizar recursos avançados de programação em função da complexidade destes recursos conforme apresentado pela linguagem Java. De forma mais clara isto é visto na modelagem da classe Interface que para recuperar um valor digitado pelo usuário precisa de métodos estáticos logo no início da metodologia.

Este ponto ressalta que alguns dos pontos fracos observados na metodologia poderiam ser melhor trabalhados com o apoio de um ferramental adequado como um editor que não exigisse a criação de um método *main* para executar código ou uma ferramenta que permitisse executar métodos diretamente dos objetos.

BORATTI, Isaias Camilo. **Programação Orientada a Objetos com Java.** Florianópolis: Visual Books, 2007. 308 p.

Michael Kölling and John Rosenberg. 2001. **Guidelines for teaching object orientation with Java**. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education* (ITiCSE '01). ACM, New York, NY, USA, 33-36. DOI=10.1145/377435.377461 http://doi.acm.org/10.1145/377435.377461