Unidade 9: Noções de Orientação a Objetos

Prof. Daniel Caetano

Objetivo: Apresentar os conceitos inciais de Orientação a Objetos.

Bibliografia: BEZERRA, 2007; JACOBSON, 1992; COAD, 1992.

INTRODUÇÃO

Em geral, quanto maior o software, mais complexo é seu desenvolvimento, devido às muitas partes que o compõem e o inter-relacionamento entre elas. Adicionalmente, a execução de testes e a solução de problemas frequentemente se torna um pesadelo em sistemas de software mal-elaborados.

Uma razão frequente para as dificuldades de implementação, testes e manutenção é que, em geral, quando se segue uma lógica de projeto de software baseado em decomposição funcional, como vimos anteriormente, **o sistema é desenvolvido estruturado de acordo com o que ele faz**, sem uma preocupação em representar no software o processo de uma maneira similar ao que ocorre na realidade.

O problema dessa abordagem é que, de tempos em tempos, as empresas mudam seus processos e procedimentos. Isso significa que, talvez, o que o software faz mude também... e se ele foi todo estruturado de acordo com o que ele precisava fazer antes, pode ser que agora ele precise ser totalmente reestruturado. Em outras palavras, pode ser que ele precise ser refeito (isto é, se não quisermos fazer "gambiarras").

Entretanto, alguns desenvolvedores pensaram: "*Se as funções de uma empresa e de um software mudam com muita frequência, eu não posso usá-las para basear a organização de meu programa*". Essa foi a primeira grande conclusão dos fatos acima. A pergunta que levou à segunda grande conclusão foi:

"*Mas o que é que, em uma empresa e em seus processos, raramente muda*?"

Por sorte alguém conseguiu perceber coisas que raramente mudam: **as coisas**!

Como assim, as coisas? Simples: tudo aquilo que é físico. Existe uma grande constância no uso de formulários, na produção de um determinado produto, nos funcionários envolvidos em um procedimento... as **entidades** são muito constantes... ou, em outras palavras, os **objetos** envolvidos na realização dos processos são, quase sempre, relativamente constantes. E disso surgiu a segunda grande conclusão:

"*Vamos **basear a estrutura do software nos objetos** envolvidos em seus processos*".

1. ORIENTAÇÃO A OBJETOS

Orientação a objetos é um conceito de representação da realidade em que os componentes são objetos e não funções ou estruturas de dados. Em outras palavras, se nos modelos estruturados os componentes eram definidos de acordo com características intrínsecas à implementação, nos modelos orientados a objetos estes componentes se baseiam objetos (entidades) do mundo real.

- O mundo real é composto de objetos que interagem entre si.
- Um modelo orientado a objetos é composto de objetos que interagem entre si.

Da teoria de sistemas, temos que um sistema é um conjunto de entidades que interagem entre si a fim de produzir um resultado comum. Assim, é natural o uso de "objetos programa" a fim de compor um sistema computacional.

1.1. Como São os Objetos?

No mundo real, objetos podem ser animados ou inanimados, mas qualquer um deles possui carcterísticas que podem ser classificadas como atributos ou comportamentos.

Exemplos de objetos: átomos, veículos, vias, pessoas...

Isso faz com que exista uma diferença semântica muito pequena entre modelo e a realidade que ele representa, proporcionando maior clareza.

Vantagens principais:

- **Concepção do sistema mais simples**: a transição da *realidade* para o *modelo* é facilitada.
- **Compreensão do modelo é simples**: como o *modelo* é mais próximo da *realidade*, a compreensão do modelo por quem compreende o problema real é quase automática.
- **Gerenciamento do sistema mais simples**: assim como na realidade, os objetos são estáveis na solução de um problema, ou seja, os objetos mudam muito pouco; quando é necessário resolver problemas ligeiramente diferentes, modificamos a forma com que os objetos interagem e não os objetos em si.

Mas afinal, o que são objetos em programação?

Em programação (e, de certa forma também na vida real), um objeto é um ente caracterizado por um conjunto de operações e um estado, caracterizados por *métodos* e *campos*, podendo ainda ser compostos por outros objetos.

Note que a propriedade de um objeto poder ser composto de outros objetos também atende à teoria de sistemas, já que uma entidade que faz parte de um sistema pode ser ela mesma um *subsistema*. Da mesma forma, é uma característica que está em perfeito acordo com a realidade, visto que usualmente um objeto é composto de outros objetos (ex.: uma geladeira é composta de porta, prateleiras, caixa, motor, fios...). Os próprios seres vivos são compostos de elementos chamados *células*.

Note que um objeto é uma estrutura similar à uma "estrutura de dados"; porém, além de "dados", um objeto pode armazenar também "funções". Em um objeto os dados são chamados de *atributos* e as funções são chamadas de *métodos*.

Exemplos de objetos do mundo real:

Objeto	**Métodos**	**Atributos**
TV	Liga	Canal
	Desliga	Volume
	Muda canal	Estado(ligada/desligada)

Objeto	**Métodos**	**Atributos**
Carro	Liga	Cor
	Desliga	Velocidade
	Acelera	Quilometragem (odômetro)
	Breca	Portas

1.2. Conceitos da Programação Orientadas a Objetos

Além dos objetos, a oritentação a objetos também se baseia em outros conceitos, como os de classes, mensagens e associações. Vejamos cada um destes conceitos a seguir.

CLASSES

Uma classe pode ser considerada como um "molde" de um objeto, sendo uma descrição de como um objeto pode ser criado. Uma forma interessante de explicar é que uma classe está para um objeto assim como a planta de uma casa está para a casa. Uma outra maneira de explicar é que se o objeto é um bolo, então a classe seria uma combinação entre a forma e a receita do bolo.

Exemplo:
Classe: Carro
Objetos: Carro vermelho, Carro azul, Ferrari do Daniel etc.

MENSAGENS

Objetos são capazes de executar operações. Entretanto, estas operações não são ativadas de maneira aleatória. É preciso que um objeto receba um estímulo para executar uma operação, ou seja, é preciso que alguma coisa **solicite que o objeto faça algo**.

Essa solicitação, esse estímulo para que o objeto realize uma tarefa, é chamado de *mensagem*. Em outras palavras, uma mensagem é a forma como um objeto se comunica com outro (ou estimula a outro). As mensagens que um objeto entende, em geral, são os métodos definidos em sua classe. Às mensagensagens que um objeto entende é dado o nome de *interface*.

ASSOCIAÇÕES

Como já foi visto anteriormente, um objeto pode ser composto de outros objetos diferentes. Quando objetos mais simples se unem para formar um objeto mais complexo, dizems que houve uma *associação de objetos.*

Exemplo: Carro = chassi + motor + acessórios + etc.

1.3. Principais Propriedades da Orientação a Objetos

As principais caracterísitas das classes de objetos constituem também as fundações da programação orientada a objetos. Estas características são: encapsulamento, herança e polimorfismo e a herança.

ENCAPSULAMENTO

É a propriedade que permite que um objeto seja tratado como uma "caixa preta". O interior do objeto, ou seja, "como" ele realiza as tarefas é invisível para os clientes daquele objeto. Os clientes só podem se comunicar com um objeto através da *interface* deste objeto, sendo que a interface de um objeto nada mais é do que a definição de quais mensagens ele "sabe" responder.

Exemplo: o carro é um objeto que pode ser tratado como uma caixa preta; uma pessoa pode dirigir sem saber como funciona o motor do carro, basta conhecer o uso de sua interface, isto é, como girar a direção, pisar nos pedais e trocar de marcha com o câmbio.

HERANÇA

Herança é a propriedade que nos permite criar uma nova classe especificando que ela "é uma" outra classe também. Por exemplo, se temos a classe "Pessoa", podemos criar a classe "Trabalhador" dizendo que *Trabalhador é uma Pessoa.* Assim, um objeto da classe Trabalhador vai também possuir todos os atributos e métodos de um objeto da classe Pessoa (como *nome*, por exemplo).

Assim, se existe uma relação de que a ***Classe B*** "é uma" ***Classe A***, isso significa que a Classe B é capaz de fazer tudo que a Classe A faz, ou seja, a Classe B **herda** os atributos e métodos da Classe A. Neste caso, pode-se dizer:

1) A é a ***superclasse*** de B;
2) A é a ***generalização*** de B;
3) B é uma ***subclasse*** de A;
4) B é uma ***especialização*** de A.

5) B respeita a mesma interface que A.

POLIMORFISMO

Polimorfismo é uma propriedade que permite que um objeto que conheça uma determinada *interface*, pode se comunicar, isto é, trocar mensagens com qualquer outro objeto que respeite aquela *interface* (ou seja, que tenha os mesmos atributos e métodos), independentemente de qual seja o tipo do objeto com quem está se comunicado. Em outras palavras, dois objetos que conheçam uma mesma *interface* podem se comunicar, independentemente de quais sejam suas classes.

Exemplo: Se Carro Azul e Caminhonete Vermelha possuem a mesma interface, que é conhecida por João, então:

Objeto	**Ação**	**Objeto**		**Objeto**	**Ação**	**Objeto**
João	Dirige	Carro azul	=>	João	Dirige	Caminhonete Vermelha

Trocando em miúdos, se carro e caminhonete possuem a mesma interface de operação, que é conhecida por João, então João saberá usar/dirigir tanto o carro quanto a caminhonete, mesmo que o objeto caminhonete tenha sido inventado muito tempo depois da criação do objeto João.

RELAÇÃO ENTRE HERANÇA E POLIMORFISMO

De uma forma rigorosa, podemos dizer que herança é a propriedade que permite que, ao especializar uma classe, os objetos da nova classe preservem todos os comportamentos e atributos dos objetos da classe original, ou seja, os comportamentos e atributos são *herdados*. Em outras palavras, a nova classe (mais especializada) continua a respeitar a *interface* estabelecida pela classe original.

Exemplo:

classe: Pessoa ação: dirige **classe: Veículo**
objeto: joao
objeto: carla
subclasse: Carro (é um Veículo)
objeto: carroAzul
subclasse: Caminhonete (é um Veículo)
objeto: caminhoneteVermelha

Se objetos da classe Pessoa conhecem a interface que lhe permite dirigir um objeto da da classe Veículo, então eles conhecerão também como dirigir um objeto das classes Carro e Caminhonete.

Isso ocorre porque Carro e Caminhonete são classes **especializadas** da classe Veículo original. Assim, se *joao* e *carla* são objetos da classe Pessoa e *carroAzul* é um objeto da classe Carro (que é um Veículo) e *caminhoneteVermelha* é um objeto da classe Caminhonete (que é um Veículo), então tanto o objeto ***joao*** quanto o objeto ***carla*** podem interagir com carroAzul e caminhoneteVermelha.

Muitas linguagens, incluindo C++ e Java, utilizam a propriedade da Herança para implementar diversos tipos de Polimorfismo. O Java inclui também o tipo "interface" para esta finalidade.

2. EXERCÍCIOS

O objetivo deste exercício é exercitar a criação de classes e objetos, apresentando os conceitos ao aluno.

1. Usando JCreator, NetBeans, Eclipse ou qualquer outro programa de sua preferência, inicie um projeto com uma classe **Pessoa**, conforme descrita abaixo, comentando adequadamente o código:

Nome:
Pessoa
Atributos:
- nome (String)
- idade (int)

O código inicial deve ficar mais ou menos como é descrito abaixo.

```
/**
 * A classe Pessoa define características importantes de uma pessoa.
 * @author (seu nome)
 * @version 20100227_001
 */
public class Pessoa {
   // Variáveis de Instância
   public String nome;
   public int idade;
   }
```

2. Toda classe deve ter um método construtor, responsável por definir os valores iniciais dos atributos sempre que um objeto é criado. O contrutor deve ter EXATAMENTE o mesmo nome que a classe, conforme mostrado a seguir:

```
/**
 * A classe Pessoa define características importantes de uma pessoa.
 * @author (seu nome)
 * @version 20100227_001  <<== Versão é muito importante!
 */
public class Pessoa {
   // Variáveis de Instância
   public String nome;
   public int idade;

   /**
    * Construtor for objects of class Pessoa
    */
   public Pessoa() {
      // Inicializa variáveis de instância
      }

   }
```

3. Vamos modificar o construturo para que ele receba informações sobre o objeto, como nome e idade, e modifique os valores internos do objeto.

```
/**
 * A classe Pessoa define características importantes de uma pessoa.
 * @author (seu nome)
 * @version 20100227_001
 */
public class Pessoa {
   // Variáveis de Instância
   public String nome;
   public int idade;

   /**
    * Construtor for objects of class Pessoa
    */
   public Pessoa(String umNome, int umaIdade) {
     // Inicializa variáveis de instância
      nome = umNome;
      idade = umaIdade;
      }
   }
```

4. Vamos criar/modificar o método **main** para criar um objeto desta classe.

```
/**
 * A classe Pessoa define características importantes de uma pessoa.
 * @author (seu nome)
 * @version 20100227_001
 */

public class Pessoa {
   // Variáveis de Instância
   public String nome;
   public int idade;

   /**
    * Construtor for objects of class Pessoa
    */
   public Pessoa(String umNome, int umaIdade) {
      // Inicializa variáveis de instância
      nome = umNome;
      idade = umaIdade;
      }

   /**
    * Rotina de Teste da Classe
    */
   public static void main(String[] args) {
      Pessoa joao = new Pessoa ("João Alves", 18);
      Pessoa maria = new Pessoa ("Maria Alves", 17);
      System.out.println("Nome: " + joao.nome + "\nIdade: " + joao.idade + "\n");
      System.out.println("Nome: " + maria.nome + "\nIdade: " + maria.idade + "\n");
      }
   }
```

5. Compile e execute este programa e observe os resultados.

Um pouco de Boas Práticas

6. A classe (muito simples) criada não segue alguns padrões de boas práticas. O primeiro deles é a falta de comentário adequado no construtor.

```
/**
 * A classe Pessoa define características importantes de uma pessoa.
 * @author (seu nome)
 * @version 20100227_001
*/

public class Pessoa {
   // Variáveis de Instância
   public String nome;
   public int idade;

   /**
    * Construtor para objetos da classe Pessoa
    * @param      umNome       o nome da pessoa
    * @param      umaIdade     a idade da pessoa
    * @return     não há valor de retorno
    */
   public Pessoa(String umNome, int umaIdade) {
      // Inicializa variáveis de instância
      nome = umNome;
      idade = umaIdade;
      }

   /**
    * Rotina de Teste da Classe
    */
   public static void main(String[] args) {
      Pessoa joao = new Pessoa ("João Alves", 18);
      Pessoa maria = new Pessoa ("Maria Alves", 17);
      System.out.println("Nome: " + joao.nome + "\nIdade: " + joao.idade + "\n");
      System.out.println("Nome: " + maria.nome + "\nIdade: " + maria.idade + "\n");
      }
   }
```

BIBLIOGRAFIA

BEZERRA, E. **Princípios de Análise e Projeto de Sistemas com UML**. 2ed. Rio de Janeiro: Editora Campus Elsevier, 2007.

JACOBSON, I; CHRISTERSON, M; JONSSON, P; ÖVERGAARD, G. **Object-oriented software engineering: a use case driven approach.** Essex, England: Addison-Wesley Longman Ltd, 1992.

COAD, P; YOURDON, E. **Análise baseada em objetos.** Editora Campus, 1992.