Unidade 3: Classes em Java Para Programadores C

Classes, Objetos e Tratamento de Erros

Prof. Daniel Caetano

Objetivo: Apresentar resumidamente os conceitos de classes, objetos e tratamento de erros com exceções em Java.

Bibliografia: DEITEL, 2005; CAELUM, 2010; HOFF, 1996.

INTRODUÇÃO

Na unidade anterior foi apresentado que o Java possui alguns tipos de dados "não nativos", isto é, programados. Estes tipos de dados são programados através de classes, que serão apresentadas nesta unidade.

Adicionalmente, o Java possui um sistema de tratamento de erros baseado em exceções, que também será apresentado ao final da unidade.

Esta aula está dividida nas seguintes seções:

1) Classes e Objetos
2) Tratamento de Erros

NOTA: Este conteúdo é **complementar** ao da aula / apresentação PowerPoint

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 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:

TV	- Liga	- Canal
	- Desliga	- Volume
	- Muda canal	- Estado(ligada/desligada)
Carro	- Liga	- Cor
	- Desliga	- Velocidade
	- Acelera	- Quilometragem (odômetro)
	- Breca	- Portas

9.2. Classes x Objetos

É importante, neste momento, diferenciar uma classe de um objeto. Uma classe é um conceito genérico: pessoa, carro, cliente. Um objeto, por outro lado, é um conceito específico: Alberto, Mustang Azul, O Comprador do Mustang Azul do Alberto. Uma classe pode ser considerada como um "molde" de um objeto.

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

Quando programamos, nós programamos **classes**. Para realizar uma tarefa, podemos precisar de objetos específicos e, então, solicitamos ao Java que construa um objeto com base em uma classe específica. Um objeto é criado da seguinte forma:

```
ClasseDoObjeto nomeDoObjeto = new ClasseDoObjeto(parâmetrosDeCriação);
```

Por exemplo, para criar a Pessoa chamada Alberto, uma possibilidade seria essa:

```
Pessoa alberto = new Pessoa("Alberto");
```

Observe que o nome do objeto não precisa ter relação nenhuma com os atributos do objeto. O código a seguir, por exemplo, realiza a mesma tarefa que o código anterior:

```
Pessoa umaPessoa = new Pessoa("Alberto");
```

9.3. Chamando Métodos

Uma vez que um objeto tem atributos e métodos, pode ser que desejemos solicitar a algum objeto que ele nos realize uma tarefa específica. Por exemplo, podemos querer solicitar que um personagem de um jogo se desenhe na tela. Isso poderia ser feito conforme indicado abaixo:

```
personagem.desenhar(tela);
```

Observe que primeiramente foi indicado o nome do objeto (personagem), seguido do método (desenhar) e, como parâmetro, foi passado o nome de outro objeto (tela).

De maneira geral, uma chamada a um método segue o seguinte formato:

```
[dono_do_metodo.]<nome_do_metodo>([parametros_do_metodo]);
```

9.4. Principais Propriedades da Orientação a Objetos

As principais caracterísitas das classes de objetos constituem também as fundações do modelo orientado a objetos. Estas características são: encapsulamento, 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.

Note que essa propriedade permite que pensemos em termos de "classe de análise" antes de pensarmos em "classe de projeto". Nas "classes de análise" são definidas, basicamente, as interfaces dos objetos. Posteriormente, nas "classes de projeto", é que existirá a preocupação em como fazer tais objetos funcionarem a partir da interface estabelecida.

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*, 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 operar tanto o carro quanto a caminhonete, mesmo que o objeto caminhonete tenha sido inventado muito tempo depois da criação do objeto João.

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).

De forma mais 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 da classe Veículo, conhecem também a interface das classes Carro e Caminhonete, que são classes **especializadas** da classe Carro original. Assim, se *joao* e *carla* são objetos da classe Pessoa e *carroAzul* é um objeto da classe Carro e *caminhoneteVermelha* é um objeto da classe Caminhonete, então tanto 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 o Polimorfismo. O Java inclui também o tipo "interface" para esta finalidade.

10. TRATAMENTO DE ERROS

Em muitas situações da programação, uma sequência de tarefas pode ser interrompida pela ocorrência de algum tipo de erro. Por exemplo: se solicitarmos ao usuário que digite um número, para que possamos fazer uma conta, e o usuário digitar um texto, haverá um problema para realizar a conta. Para este tipo de situação, existe uma estrutura de tratamento de erros denominada *try ~ catch ~ finally*.

A idéia é colocar dentro de um bloco "try" toda a sequência de instruções propensas a erro e, para cada tipo de erro, acrescentar um bloco "catch". O bloco finally existe caso desejemos que algumas operações sejam executadas sempre, ocorra um erro ou não; caso típico é desfazer a conexão com o banco de dados.

O código abaixo mostra um exemplo de uso de try-catch-finally:

Main.java

```
// Programa não tão mínimo em Java
package meuprimeiroprograma;

// Classe Principal
public class Main {

// Método Principal
public static void main( String args[] ) {

   try {
      String numeroDigitado = "5";
      double numero = Double.parseDouble(numeroDigitado);
      double resultado = numero / 2.0;
      System.out.println(resultado);
      }
   catch (NumberFormatException ex) {
      System.out.println("Número inválido!");
      }
   finally {
      System.out.println("Programa encerrado!");
      }
   }

}
```

Execute o programa e veja o resultado. Depois disso, modifique o valor inicial de "numeroDigitado" para "5A", ao invés de "5". Execute novamente o programa e veja o que ocorre.

11. BIBLIOGRAFIA

CAELUM, FJ-11: Java e Orientação a Objetos. Acessado em: 10/01/2010. Disponível em: < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos/ >

DEITEL, H.M; DEITEL, P.J. **Java: como programar** - Sexta edição. São Paulo: Pearson-Prentice Hall, 2005.

HOFF, A; SHAIO, S; STARBUCK, O. **Ligado em Java**. São Paulo: Makron Books, 1996.