Disciplina de Programação Orientada a Objetos - POOS3

Curso Superior de ADS - 3º Semestre

(Professor Dênis Leonardo Zaniro)

Introdução às enumerações

Roteiro

- Criação de enumerações básicas
- Definição de propriedades associadas às constantes
- Definição de corpos de constantes

Objetivos

1- Utilizar enumerações para resolver diversos tipos de problemas em um programa Java.

Criação de enumerações básicas

Nós já aprendemos sobre como criar constantes (usando-se a palavra-reservada *final*) em uma classe em Java. Também é possível criar um conjunto de constantes que definem os únicos valores possíveis de uma variável qualquer. Para exemplificar essa situação, vamos supor que fosse necessário elaborar uma classe Veiculo composta pelos atributos *nome* e *cor*.

```
public class Veiculo {
  public static final String PRETO = "preto";
  public static final String BRANCO = "branco";
  public static final String CINZA = "cinza";
  private String nome;
  private String cor;
  public Veiculo(String nome, String cor) {
    if(cor.equals(Veiculo.PRETO) || cor.equals(Veiculo.BRANCO)
|| cor.equals(Veiculo.CINZA))
      this.cor = cor;
    else
      this.cor = Veiculo.PRETO; //cor default
    this.nome = nome;
  }
  ...
}
```

De acordo com o código, foram definidas três constantes - PRETO, BRANCO e CINZA -, cujos valores representam as cores possíveis de um veículo qualquer (apenas essas

cores devem ser aceitas). Ao se criar um objeto do tipo Veiculo, será necessário verificar se a cor informada para o veículo em questão pertence ao conjunto de cores possíveis. Em outras palavras, deveremos testar se a cor informada como argumento é igual a uma das três cores definidas pelas constantes.

Essa solução, embora seja simples, não é a mais adequada (antes da versão 5.0 do Java, não havia uma estratégia melhor para resolver esse problema). O primeiro grande problema é que, sempre que uma nova cor for acrescentada ao código, a condição deverá ser modificada para tratar a possibilidade de essa cor ter sido a escolhida para um veículo. Além disso, caso seja informada uma cor não considerada pelo programa ou qualquer outro valor, o veículo será criado e a sua cor deverá ser inicializada a partir de um valor *default*.

Outro grande problema é que, como as constantes são definidas na própria classe Veiculo, qualquer modificação na declaração de constantes exigirá que a classe Veiculo seja recompilada. Como outro problema, ao declarar uma constante, o único dado que haverá de fato será o valor dessa constante. Em outras palavras, ao usar constantes, seria impossível definir alguma propriedade que estivesse atrelada ao valor da constante (por exemplo, um número associado a uma cor).

A solução mais ideal para o problema apresentado é usar o recurso de ***enumerações*** (*enumerations*) que a linguagem Java fornece. Devemos encarar uma enumeração como uma classe especial, composta basicamente de constantes. Nessa classe, pode haver ainda construtores (com uma função diferente de um construtor comum, como será visto adiante), atributos e métodos. A sintaxe básica para a criação de uma enumeração é apresentada abaixo:

```
public enum NomeDaEnumeracao {
  CONSTANTE_1,
  CONSTANTE_2,
   ...
  CONSTANTE_N;
  ...
}
```

Conforme pode ser observado, uma enumeração é declarada usando-se a palavra reservada *enum*. No corpo de uma enumeração podemos declarar uma lista de constantes, separadas por vírgula. Após definirmos a lista de constantes, poderemos definir métodos e atributos para a enumeração. É importante observar que não há definição de tipos de constantes, mas apenas definição de valores.

Considerando-se o exemplo anterior da classe Veiculo, a seguinte enumeração poderia ser criada:

```
public enum Cor {
  PRETO,
  BRANCO,
```

```
  CINZA;
}
```

Após criar a enumeração Cor, a classe Veiculo poderia ser modificada, conforme mostra o código dado a seguir:

```
public class Veiculo {
  private String nome;
  private Cor cor; //atributo do tipo da enumeração criada
  public Veiculo(String nome, Cor cor) {
    this.cor = cor;
    ...
  }
  ...
}
```

De acordo com o novo código da classe Veiculo, o atributo *cor* foi declarado como uma variável do tipo Cor, isto é, do tipo da enumeração criada. Como as constantes que representam as cores possíveis de um veículo já constam na enumeração Cor, as constantes inicialmente declaradas na classe Veiculo foram retiradas do código.

Além disso, a verificação de cores também foi retirada, o que indica que, para que o compilador aceite um argumento do tipo Cor na criação de um veículo qualquer, esse argumento deverá estar de acordo com alguma constante declarada na enumeração. Portanto, aquela verificação anterior se tornou desnecessária. O código abaixo mostra a classe Principal, responsável pela criação de objetos do tipo Veiculo:

```
public class Principal {
  public static void main(String args[]) {
    Veiculo v;
    Cor cor; //referência para a enumeração Cor
    cor = Cor.PRETO; //inicialização da referência cor
    v = new Veiculo(“Corsa”, cor);
  }
}
```

De acordo com o código anterior, ao se criar um veiculo, deveremos informar uma determinada constante que tenha sido declarada na enumeração Cor. Em outras palavras, para que o compilador permita a instanciação de um objeto do tipo Veiculo, será obrigatório informar uma constante que pertença à lista de constantes declaradas na enumeração. Além disso, mesmo que novas cores sejam adicionadas à lista de constantes definida na enumeração, a classe Veiculo não precisará ser modificada.

Uma observação importante a respeito de enumerações é que elas não podem ser instanciadas usando-se o operador *new*, mesmo que haja construtores definidos em seu

corpo. É possível pensar no processo de instanciação de uma enumeração, mas de uma forma diferente em relação às classes comuns, conforme será visto adiante.

Definição de propriedades associadas às constantes

Como já foi citado, uma das desvantagens relacionadas ao uso de constantes em uma classe é a impossibilidade de se definir alguma propriedade relacionada ao valor da constante. Ao definir uma enumeração, cada constante declarada no seu corpo poderá ter um ou vários valores associados a ela. Esses valores podem ser de quaisquer tipos e devem ser especificados entre parênteses após a constante. O exemplo abaixo mostra a sintaxe básica para a definição de valores associados a uma constante:

```
public enum NomeDaEnumeracao {
  CONSTANTE_1(valor1, valor2, ..., valorN),
  CONTANTE_2(valor1, valor2, ..., valorN),
  ...
  CONSTANTE_N(valor1, valor2, ..., valorN);
}
```

Pelo código, percebe-se que podemos especificar diversos valores, separados por vírgula, para cada constante. Para o exemplo de veículos, vamos supor que cada cor estivesse associada a um determinado número (valor inteiro) e precisássemos recuperar esse número de alguma forma. Para resolver esse problema, bastaria, primeiramente, associar um número a cada constante definida na enumeração, conforme é mostrado pelo exemplo abaixo:

```
public enum Cor {
  PRETO(1), //valor 1 associado à constante PRETO
  BRANCO(2), //valor 2 associado à constante BRANCO
  CINZA(3); //valor 3 associado à constante CINZA
} //solução parcial para o problema apresentado
```

O trecho de código dado acima representa a solução parcial para esse problema, e, na verdade, sequer seria compilado. O motivo disso é que, ao se definir um ou vários valores associados a uma constante, será necessário declarar um atributo para armazenar cada valor associado à constante. Além disso, esse atributo deverá ser inicializado em um construtor da enumeração. O código da enumeração dado abaixo mostra a solução completa para esse problema:

```
public enum Cor {
  PRETO(1),
  BRANCO(2),
  CINZA(3);
  private int numero;
  Cor(int numero) {
```

```
    this.numero = numero;
  }
}
```

No trecho de código dado acima foi declarado um atributo chamado *numero*, do mesmo tipo dos valores associados às constantes. Além disso, foi criado um construtor para a enumeração permitindo que esse atributo seja inicializado. Nesse ponto, é importante observar que, conforme já citado, uma enumeração jamais poderá ser instanciada, mesmo havendo algum construtor criado. O construtor de uma enumeração pode apenas ser privado ou *default* (sem modificador), e terá a responsabilidade de criar a associação entre uma constante e seu valor correspondente. Para exemplificarmos, considere a instrução dada abaixo:

```
Cor cor = Cor.PRETO;
```

Nessa instrução, declaramos uma referência para a enumeração Cor e atribuímos a essa referência a constante PRETO. Nesse instante, o construtor da enumeração será executado e receberá, como argumento, o número associado à constante PRETO (número definido entre parênteses). Esse número será atribuído ao atributo *numero* apontado pela referência *cor*. Na verdade, quando a instrução anterior for executada, o atributo número para cada constante será inicializado com base no valor definido entre parênteses. Dessa forma, pode-se pensar em cada constante como um objeto (contendo dados e operações).

Considerando-se ainda o exemplo anterior, para que uma referência do tipo Cor tenha acesso ao seu número, poderemos criar um método *getNumero( )* no corpo da enumeração, conforme é mostrado pelo código dado abaixo:

```
public enum Cor {
  ... //trecho de código igual ao código anterior da enumeração
  public int getNumero() {
    return numero;
  }
}
```

Vamos supor que, além de um número inteiro, precisássemos associar a uma constante uma *string* representando, de alguma forma, a constante em questão. Nesse caso, haveria dois valores associados a uma mesma constante, o que levaria o compilador a exigir a declaração de dois atributos e um construtor composto por dois parâmetros, conforme mostra o exemplo de código a seguir:

```
public enum Cor {
  PRETO(1,”preto”),
  BRANCO(2,”branco”),
  CINZA(3,”cinza”);
  private int numero;
  private String nome;
```

```
  Cor(int numero, String nome) {
    this.numero = numero;
    this.nome = nome;
  }
}
```

Observamos que o construtor aceita, como primeiro parâmetro, o tipo inteiro e, como segundo parâmetro, o tipo String. A ordem de declaração desses parâmetros deve ser a mesma da ordem de definição dos valores entre parênteses.

Toda enumeração possui um método estático chamado *values( )* que retorna a lista de constantes declaradas em uma determinada enumeração. O tipo de retorno desse método é um *array* do mesmo tipo da enumeração, conforme é mostrado pelo exemplo a seguir:

```
public class Principal {
  public static void main(String args[]) {
    Cor[] cor = Cor.values();
    for(int i = 0; i < cor.length; i++)
      System.out.println(cor[i]);
  }
}
```

Outro método estático que pode ser usado é o método *valueOf()*, que aceita como parâmetro uma *string* que represente uma determinada constante e retorna a constante correspondente. A constante retornada é na forma do tipo da enumeração, como pode ser observado no exemplo a seguir:

```
public class Principal {
  public static void main(String args[]) {
    Cor cor = Cor.valueOf(“CINZA”);
    System.out.println(cor.getNumero()); //3
  }
}
```

Definição de corpos de constantes

Em uma enumeração, é possível ainda definir um corpo para cada constante declarada, isto é, cada constante pode ter seus próprios atributos e métodos. Para exemplificar essa situação, vamos supor que cada constante representando uma cor no exemplo anterior tivesse a capacidade de calcular e retornar seu valor correspondente a ser acrescentado ao preço de um veículo. O código para resolver esse problema é dado abaixo:

```
public enum Cor {
  PRETO(1,”preto”) {
    public double calculaValor() {
```

```
      return 100;
    }
  },
  BRANCO(2,”branco”) {
    public double calculaValor() {
      return 50;
    }
  },
  CINZA(3,”cinza”) {
    public double calculaValor() {
      return 200;
    }
  };
  private int numero;
  private String nome;
  Cor(int numero, String nome) {
    this.numero = numero;
    this.nome = nome;
  }
  public double calculaValor() {
    return 0;
  }
}
```

Pelo o que pode ser observado no código anterior, foi criado um método no corpo da enumeração chamado *calculaValor()* (Acessado globalmente), e cada constante sobrepõe esse método definindo uma nova versão dele. Isso é necessário uma vez que, se não houvesse esse método, os métodos declarados no corpo de cada constante não seriam enxergados por outras classes. Dessa forma, a decisão sobre qual método será invocado dependerá da constante atribuída à variável de referência para a enumeração Cor. A classe Principal, dada a seguir, demonstra essa situação:

```
public class Principal {
  public static void main(String args[]) {
    Cor cor;
    cor = Cor.PRETO;
    System.out.println(cor.calculaValor()); //imprimirá 100
    cor = Cor.BRANCO;
    System.out.println(cor.calculaValor()); //imprimirá 50
    cor = Cor.CINZA;
```

```
    System.out.println(cor.calculaValor()); //imprimirá 200
  }
}
```

É importante observar, pelo código anterior, que o método *calculaValor( )* somente pôde ser invocado por uma referência para a enumeração Cor por que havia uma versão dele declarada para a enumeração como um todo, conforme já citado. Além disso, o método *calculaValor( )*, nesse caso, poderia ser abstrato, o que é permitido em uma enumeração. Havendo um método abstrato no corpo de uma enumeração, todas as constantes declaradas se obrigarão a fornecer uma implementação para esse método. O exemplo abaixo mostra a declaração de um método abstrato na enumeração.

```
public enum Cor {
  PRETO(1,”preto”) {
    public double calculaValor() {
      return 100;
    }
  },
  BRANCO(2,”branco”) {
    public double calculaValor() {
      return 50;
    }
  },
  CINZA(3,”cinza”) {
    public double calculaValor() {
      return 200;
    }
  };
  ...
  ...
  public abstract double calculaValor();
}
```

Referências bibliográficas

Deitel, H. M.; Deitel, P. J. Java - Como Programar. 4. ed., Bookman, 2002.

Oracle. *Learning the Java Language*. Disponível em: http://download.oracle.com/javase/tutorial/java/. Data de acesso: 02/01/2016.

Santos, R. Introdução à Programação Orientada a Objetos usando JAVA. Elsevier, 2003.

Sierra, K; Bates, B. Use a Cabeça! Java. 2. ed., O Reilly, 2005.