Enums são mais que constantes

Entenda o poder das enums no mundo Java e como você pode tirar melhor provei código orientado a objetos

Rafael Ponte - *Desenvolvedor e instrutor na TriadWorks*

September 22, 2015

Embora o Java 8 tenha sido lançado há mais de 1 ano é muito comum encontrar códig uso de **enums** ainda é muito mal aproveitado. A verdade é que enums são subestimad apenas como um simples substituto para constantes dentro do sistema. Os maiores be enums vão além disto e, na minha opinião, vale a pena um desenvolvedor ter idéia do

Para entender melhor do que estou falando nada melhor do que um exemplo…

Imagine que temos as seguintes constantes para representar os diferentes tipos de doc identificar um cliente no nosso sistema:

```
public class TiposDeDocumento {

    public static final int DOCUMENTO_RG    = 0;
    public static final int DOCUMENTO_CPF   = 1;
    public static final int DOCUMENTO_CNPJ  = 2;
}
```

A lista de constantes acima é uma excelente candidata para uma enum, pois temos un de opções. Dessa forma, para converter a lista de constantes para um enum nós poden *enum* `TipoDeDocumento` e adicionar cada constante como no código a seguir:

```
public enum TipoDeDocumento {

    RG,
    CPF,
    CNPJ;
}
```

Pronto! Eu poderia dizer sem medo, e acredito que você também, que a maioria dos ca enums são usadas param por aí. Elas são simplesmente utilizadas para **substituir con**

Não que isso seja ruim, é até algo bom e indicado, pois substituir constantes por enum benefícios evidentes de manutenção e corretude ao código, com tipagem segura (*type-* legibilidade. Graças a sua natureza *type-safe* seria impossível um desenvolvedor atribu incorreto a enum:

```
TipoDeDocumento tipo = 8; // erro de compilação!
```

Embora tenhamos o compilador para ajudar o desenvolvedor a não cometer erros bob idéia é irmos mais longe e termos ciência que a utilidade de enums vai muito, mas mu

Enums são objetos

Existem vários detalhes que podemos aprender sobre enums, mas sem dúvida a prime precisamos compreender é que elas são mais do que constantes, **enums são objetos** e O tipo `enum` em Java nada mais é do que uma "açúcar sintático" (*syntax sugar*) de cri subclasse que herda de `java.lang.Enum` com alguns membros estáticos.

Não acredita? Pois bem, se fizéssemos a engenharia reversa no *.class* da enum `TipoDe` nós teríamos algo como:

```
public final class TipoDeDocumento extends java.lang.Enum {

    public static final TipoDeDocumento RG = new TipoDeDocumento("RG", 0);
    public static final TipoDeDocumento CPF = new TipoDeDocumento("CPF", 1);
    // ...
}
```

Como você pode ver, a lista de nomes separados por vírgula dentro da enum vira variá na classe gerada. Cada constante é construída em cima de 2 valores: uma string conte constante e um inteiro com valor incrementado (*ordinal*) que é único para a instância.

Ou seja, basicamente uma enum é convertida em uma lista de constantes de objetos Ja irônico, não é?

A grande vantagem de uma enum ser uma classe Java é que ela tem por definição prat quase todos) os recursos de uma classe qualquer. Nesse momento você já deve ter ima poder que temos em mãos, certo?

Enums tem estado

Sabemos que no paradigma orientado a objetos um objeto possui dados (estado) e com logo, uma enum também possui tais características e por esse motivo a coisa toda mu

Por exemplo, para adicionar dados a enum `TipoDeDocumento` que criamos anteriorr uma **descrição** sobre o tipo do documento, bastaria adicionar um atributo de instânci

```
public enum TipoDeDocumento {

    RG("RG - Registro Geral"),
    CPF("CPF - Cadastro de Pessoas Físicas"),
    CNPJ("CNPJ - Cadastro Nacional da Pessoa Jurídica");

    private String descricao;

    /**
     * Construtor privado para montar a enum
     */
    private TipoDeDocumento(String descricao) {
```

```
        this.descricao = descricao;
    }

    public String getDescricao() {
        return descricao;
    }
}
```

O valor do atributo `descricao` é definido diretamente em cada constante através de u privado que tivemos que criar. Caso a enum tenha mais de um atributo basta alterar o Repare também que para expor o atributo para outras classes nós criamos um método muito semelhante ao que fazemos com nossas classes de negócio.

Mas qual a vantagem deste atributo? É muito comum aproveitar os atributos de uma e camada de visão, por exemplo para montar uma combobox na JSP via tag `c:forEach` seguir:

```
<select name="tipo">
    <c:forEach items="${tipos}" var="tipo">
        <option value="${tipo}" label="${tipo.descricao}" />
    </c:forEach>
</select>
```

Assim como uma classe Java, uma enum pode ter vários atributos e eles podem ser do quiser. Dessa forma podemos representar melhor um conceito de negócio dentro do n orientado a objetos.

Como você pode ver, podemos ter atributos, construtores e métodos dentro de uma en coisa não pára por aí...

Enums tem comportamentos

Mais importante do que dados numa classe são seus comportamentos, pois este é o pr de toda a orientação a objetos: **objetos gerenciam seu estado e sofrem mudanças d estímulos externos**. E estes comportamentos são representados através de métodos

Não estou falando de métodos getters ou setters, mas sim de métodos com responsabi definidas que te auxiliarão durante as regras de negócio...

Como estamos falando de documentos de identificação de clientes, então dentro da en
ter métodos pertinentes ao negócio, como um método para **validar o número** do docu
mesmo um método para **formatar o número** para que este seja exibido em um relatór
exemplo.

Nesse caso, poderíamos ter um método para formatar o número do documento de acor
do documento em questão. Dessa forma, bastaria implementar o método `formata` na

```
public enum TipoDeDocumento {

    RG("RG - Registro...", null), // sem formatador
    CPF("CPF - Cadastro...", new FormatadorDeCpf()),
    CNPJ("CNPJ - Cadastro...", new FormatadorDeCnpj());

    private String descricao;
    private Formatador formatador;

    private TipoDeDocumento(String descricao, Formatador formatador) {
        this.descricao = descricao;
        this.formatador = formatador;
    }

    /**
     * Formata número do documento
     */
    public String formata(String numero) {
        if (this.formatador == null) {
            return numero;
        }
        return this.formatador.formata(numero)
    }
}
```

Repare que aproveitamos o construtor para passar a instância do `Formatador` de acor
do documento, dessa forma conseguimos usar a instância dentro do método `formata`
código muito mais limpo e simples.

Todas as contantes da enum herdam o novo método. Portanto, para usar este método
um código como abaixo:

```
String numero = "63703867582";
String cpf = TipoDeDocumento.CPF.formata(numero); // 637.038.675-82
```

Esta mesma estratégia poderia ser utilizada para implementar o método de validação dentro da enum. No final teríamos um código muito mais coeso e orientado a objetos.

Enums e polimorfismo

Mesmo sabendo que podemos colocar atributos e métodos dentro da enum, a pergunt vezes fica na cabeça do desenvolvedor é:

> “ Onde eu posso usar este conhecimento?

Na minha opinião, a melhor maneira de um desenvolvedor visualizar as vantagens do ou de outra prática orientada a objetos é ver sua aplicação em um código pouco orient ou mal desenhado.

Então vamos imaginar a aplicabilidade do conceito **tipo de documento** dentro do sist o tipo do documento então precisamos de uma classe para abstrair e representar o doc Além disso, ela já teria um método para formatar o número do documento, com abaix

```
public class Documento {

    private String numero;
    private int tipo; // usa inteiro

    // métodos getters e setters

    public String formata() {
        switch (tipo) {
            case TiposDeDocumento.DOCUMENTO_RG:
                return this.numero;
            case TiposDeDocumento.DOCUMENTO_CPF:
                return new FormatadorDeCpf().formata(this.numero);
            case TiposDeDocumento.DOCUMENTO_CNPJ:
                return new FormatadorDeCnpj().formata(this.numero);
            default:
                return null;
        }
    }
}
```

O código acima não está ruim nem difícil de ler ou entender. Mas o uso de constantes não é legal. Um desenvolvedor mais desatento poderia passar um valor incorreto sem

```
Documento cpf = new Documento("63703867582");
cpf.setTipo(7); // tipo 7 não existe!
```

Esse é o típico problema na qual uma enum resolve facilmente, pois a enum define um de constantes que um desenvolvedor pode usar. Usar qualquer outro valor causaria er compilação!

Como a idéia é acabar com o uso das constantes, o próximo passo é refatorar a classe para usar nossa enum `TipoDeDocumento`:

```
public class Documento {

    private String numero;
    private TipoDeDocumento tipo; // usando enum

    public String formata() {
        switch (tipo) { // switch também suporta enums
            case RG:
                return this.numero;
            case CPF:
                return new FormatadorDeCpf().formata(this.numero);
            case CNPJ:
                return new FormatadorDeCnpj().formata(this.numero);
            default:
                return null;
        }
    }
}
```

A modificação é bem simples e até sutil, mas ainda temos um problema...

Nosso código pode se tornar difícil de manter caso o quantidade de tipos de document dentro do sistema. Quanto mais tipos mais condicionais no código. Embora estejamos instrução `switch`, que nada mais é do que um `if` mais bonitinho, ainda caímos de ca problemática do **excesso de condicionais** no código.

Já discutimos aqui no blog que uma boa maneira de eliminar condicionais dentro do c do uso de **polimorfismo**. Como enums são objetos com **métodos muito bem definid** acabamos com uma interface pública para usar polimorfismo. Dessa forma, o código `Documento` ficaria assim:

```
public class Documento {

    private String numero;
    private TipoDeDocumento tipo;

    public String formata() {
        return this.tipo.formata(this.numero);
    }
}
```

Graças ao uso de enums fica fácil eliminar o excesso de ifs dentro do código de maneir
Certamente esta é uma das vantagens que eu mais faço questão de aproveitar no meu

Por fim, usar a classe não seria complicado:

```
Documento cpf = new Documento("63703867582", TipoDeDocumento.CPF);
String formatado = cpf.formata(); // 637.038.675-82
```

Repare que nosso código ficou mais simples e com um design mais orientado a objetos
responsabilidades se encontram em seus devidos lugares. E aí, melhorou o entendime
como usar enums?

Outros benefícios no uso de enums

Uma enum possui diversos aspectos que a tornam uma excelente candidata para usar
orientado a objetos em vez de objetos comuns. Ao usar enums podemos simplificar n
uma melhor gerência de recursos na JVM ou garantir a imutabilidade dos objetos em
multi-thread. Para deixar claro o que estou querendo dizer, segue alguns aspectos:

- Enums são **imutáveis**, o que é ótimo para ambientes com alta concorrência;
- Enums são **singleton** por padrão: temos uma única instância de cada constante p
- Sua natureza **polimórfica** nos permite trabalhar com polimorfismo de diferentes
  sobrecarga de métodos, métodos abstratos ou interfaces;
- Facilita trabalhar com os padrões de projeto Template method, Strategy e State;
- Uma enum já possui uma boa implementação para os métodos `equals`, `hashCo`
- Comparar enums é muito simples, basta usarmos o operador `==`;
- Uma é enum é facilmente serializada para *String* ou *int*;

- Tem sua própria API de coleções que é *type safe*: EnumMap e EnumSet;

O bom uso de enums é tão importante e traz tantos benefícios que ele é um assunto ba
discutido em todos nossos cursos, em especial o curso de Java e Orientação a Objetos e
Persistência com JPA 2 e Hibernate.

Enfim, o tipo `enum` foi incorporado no Java 5.0 e ainda assim sua utilidade é bastant
Para não cair nessa armadilha, **trate uma enum como um objeto** e não como uma sir
constante, desta forma você tende a escrever um código mais orientado a objeto e com
legibilidade.

E você, já conhecia algumas destas características no uso de enums?

Rafael Ponte

Desenvolvedor e instrutor na TriadWorks

Posted in: constantes enum enums java oo orientação a objetos polimorfismo