

Enumeradores e Exceptions

Apresentação



Fonte: <https://goo.gl/BT5RDt>

Esta é a última aula da disciplina. Chegou a hora de tratarmos de dois assuntos interessantes: os Enumeradores e as Exceptions. Em muitos livros, esses assuntos são tratados de forma espalhada e em momentos distintos. Foi uma opção na organização do conteúdo desta disciplina deixar por último esses assuntos, por considerar que você estaria mais maduro e gastaria energia focando no conteúdo central do conteúdo, que é a orientação a objeto e seus pilares, e não se perderia em conteúdos novos. Sem contar que eles são dependentes em alguns pontos do que você já estudou. Por exemplo, como falar da hierarquia de heranças das Exceptions se você não tivesse estudado Herança?

As exceptions dão robustez ao Java e permitem que os programas se recuperem de situações excepcionais. Possui uma árvore de herança que você precisa compreender para saber o que, quando e como tratar.

Já, a enumeração é um conteúdo mais moderno, que surgiu com o Java 5.

Esse tema trata da representação de constantes e agrupamento de informações.

Desafio

A era da internet ilimitada acabou, afirmou João Rezende, presidente da Anatel (Agência Nacional de Telecomunicações). Segundo a agência, não há mais possibilidade para que as operadoras de banda larga fixa ofereçam serviços sem uma limitação, o que obrigará o segmento a migrar para o modelo de franquias, semelhante aos serviços de internet móvel. A Anatel está criando um sistema de controle de franquias, que terá você como desenvolvedor. Uma franquia tem quantidade de GB, dia de expiração da franquia, preço da franquia, código da franquia e nome da empresa que fornece o serviço (VIVO, CLARO, OI ou TIM). Após receber do usuário todas as franquias que ele desejar informar. Sabe-se que o usuário irá informar a quantidade de franquias a ser informada, faça o seguinte:

- a. Apresente todas as franquias.
- b. A franquia mais barata.
- c. As franquias oferecidas pela empresa VIVO.
- d. As franquias que vão fornecer mais de 2GB.
- e. Preço médio das franquias.
- f. As franquias que vão vencer entre os dias 10 e 20.

Observações:

- Considera-se mês contábil, de 1 a 30.
- Não são aceitas franquias repetidas.

Conteúdo

Enum no Java

Os enumeradores são estruturas de dados organizados, que podem agrupar valores que tenham o mesmo sentido em um determinado contexto e que, obrigatoriamente, sejam constantes (que não muda). Constantes no Java são representadas pela palavra reservada final.

Para representar o sexo de uma pessoa de forma computacional, muitos programadores usam um caractere representativo (M ou F) para fins de armazenamento (para economizar espaço em disco). Da mesma forma, para armazenar a titulação de um professor, Mestre, como valor '1' ou Doutor, sendo '2'.

Mas cabe considerar que, na apresentação em tela, os códigos dificultam a vida do usuário. Imagine aparecendo que a titulação seja '2' para um determinado professor; certamente, um funcionário novato não saberia qual a titulação do indivíduo. Perceba que o mesmo dado é representado de formas diferentes, dependendo da utilização, em tela ou no banco de dados.

Para fazer uma representação deste tipo de dado e agrupar esses valores, o Java, a partir da versão 5, lançou um recurso chamado enum.

Como o enum representa uma constante ou mais dentro de um mesmo contexto, na sua declaração deve ser utilizado caixa alta.

Observe o exemplo na Tabela 1:

Tabela 1 – Exemplo de enumerador.

Programa em Java: Programa.java

Programa em Java: Programa.java

```
public class Programa {  
    public static void main(String args[]){  
        Sexo sx = Sexo.MASCULINO;  
        System.out.println(sx.name());  
        System.out.println(sx.ordinal());  
    }  
}  
enum Sexo {  
    MASCULINO, FEMININO //enum sexo tem duas instâncias constantes.  
}
```

No exemplo anterior, `MASCULINO` e `FEMININO`, cada um deles é uma instância do tipo enum Sexo e são constantes (não aceitam modificações).

Todos os enums têm dois métodos importantes de serem comentados: o `name` e o `ordinal`. Este retorna um inteiro, de acordo com a posição da declaração. No caso do enum Sexo como `MASCULINO` é o primeiro, então é retornado '0'. Aquele, o `name`, retorna uma String, com mesmo nome da declaração, no caso, "MASCULINO".

Leia com atenção as características dos enums:

- As instâncias dos tipos enum são criadas e nomeadas no momento da declaração, sendo fixas e imutáveis.
- Não é permitido criar novas instâncias com o operador `new`, pois o método construtor é declarado como `private` implicitamente, logo, não fica disponível de fora do enum.
- Os nomes declarados recebem todas as letras em MAIÚSCULAS.
- As instâncias dos tipos enum devem obrigatoriamente ter apenas um nome.
- **Opcionalmente, a declaração da classe pode incluir variáveis de instância, construtor, métodos de instância, de classe, etc.**

O método `ordinal` retorna um número, de acordo com a posição. Imagine se existisse um código já existente que você tivesse que atribuir a um enumerador para representar uma constante? Suponha que, em uma base dados já existente, o Sexo masculino seja representado com '1' e o feminino como '2'. Perceba que o método `ordinal` não

atende este padrão, não é verdade? Nesses casos, você terá que definir um atributo para o enumerador. Observe o exemplo apresentado na Tabela 2, que inicializa o enumerador com outro valor para utilização.

Tabela 2 – Inicialização de enumerador.

Programa em Java: Programa.java

```
public class Programa{  
    public static void main(String args[]){  
        Sexo sx = Sexo.MASCULINO;  
        System.out.println(sx.name());  
        System.out.println(sx.ordinal());  
        System.out.println(sx.valor);  
    }  
}  
enum Sexo{  
    MASCULINO(1), //o argumento passado é atribuido a valor  
    FEMININO(2); // o ponto e vírgula finaliza a declaração.  
    public int valor; //atributo do enum sexo  
    private Sexo (int valor){//construtor private do enum Sexo  
        this.valor = valor;//atribuição do valor  
    }  
}
```

Perceba que duas novidades apareceram. A primeira é o atributo valor para o enum Sexo. Logo, tanto `MASCULINO` quanto `FEMININO` terão o atributo valor. O enum `MASCULINO` vale 1 e `FEMININO` vale 2, de acordo com a inicialização. A segunda foi o método construtor definido para o enumerador, que na sua declaração exige a passagem do argumento valor. Lembre-se: "A inicialização acontece no momento da declaração (criação) do enumerador quando for definido um construtor".

Para Refletir ⚡

Qual é o impacto de um método construtor ser private?

Os enums possuem obrigatoriamente métodos construtores private, pois não podem ser instanciados. Logo, se uma classe for criada e seu construtor for definido como private, o operador new não poderá ser utilizado. Isso garante que exista apenas uma instância em memória e que o valor não mude.

Outro método importante a ser falado é que independe das instâncias, como o ordinal e o name, é o values.

Pense no seguinte, e se você tivesse que descobrir todos os valores que existem dentro de enumerador. Como faria? Pois bem, o método values retorna todas as instâncias declaradas (criadas) dentro do enum. Ele retorna um `[]` dos tipos enum existentes. A assinatura do método é static, ou seja, diretamente pelo nome do enum. No exemplo do enum Sexo, a chamada deste método seria o seguinte: `Sexo.values();` que retornará um array do tipo Sexo. Observe o exemplo de utilização do método values apresentado na Tabela 3.

Tabela 3 – Utilização do método values.

Programa em Java: Programa.java

```
public class Programa{
    public static void main(String args[]){
        Turno [] turnos = Turno.values();
        for (Turno trn : turnos) {
            System.out.println("Descrição: "+trn.name());
            System.out.println("Descrição: "+trn.ordinal());
        }
    }
    enum Turno{
        MATUTINO, VESPERTINO, NOTURNO
    }
}
```

Exceptions

Desde o início da sua jornada, alguns eventos inesperados podem ter acontecido (`NullPointerException` e `ArrayIndexOutOfBoundsException`), enquanto você fazia seus programas.

As exceções citadas anteriormente são *exceptions*, que acontecem normalmente com programadores iniciantes. A primeira é quando você tenta acessar algum atributo de um objeto, sem tê-lo instanciado. Exemplo: `Pessoa p = null; p.getNome();`

A chamada do método `getNome()` irá gerar um evento inesperado lançado pela máquina virtual, o `NullPointerException`. Já, a outra acontece quando, ao percorrer um *array*, o programa acessa uma posição do *array* que não existe. Por exemplo:

`int []idades = new int[5]; idades[7] = 10;` veja que o trecho de atribuição tenta colocar o valor 10 na posição 7, que não existe para o array de idades criado, logo é gerado um evento inesperado chamado `ArrayIndexOutOfBoundsException`.

Baseado no exposto, uma exceção é um **evento inesperado**, representada por um **objeto**, que ocorre no sistema quando está em **execução** e o **fluxo da aplicação é interrompido**.

O benefício das exceptions é que os programadores não precisam prever todas as possibilidades de erros por meio de if antes de executar alguma operação. Basta executar e, caso algo excepcional aconteça, será gerada uma exceção com possibilidade de tratamento. Aliás, esse é o motivo do Java ser considerado robusto, porque dá a chance ao programador de se recuperar deste tipo de evento por meio do tratamento de exceções.

Em sua jornada, você também pode ter enfrentado na entrada de dados a `InputMismatchException`. Este evento inesperado acontece quando a entrada de dados não corresponde ao tipo de método da leitura. Caso o programa espere uma nota com um `nextDouble` e o usuário digite um texto, é lançada uma exceção que interrompe o fluxo do programa.

Neste caso, para conseguir não interromper o fluxo do programa, você deve **tratar a exceção**. Esta ação envolve a captura e a ação que deseja realizar quando algo acontecer.

Para conseguir tratar uma exceção, você precisa conhecer duas estruturas básicas: o `try` e o `catch`. O bloco `try` delimita a zona de perigo, ou seja, onde pode acontecer uma exceção, e o `catch`, que é o bloco que terá o tratamento que você dará ao evento inesperado, que pode ser uma mensagem, mudança de valores ou até mesmo fechamento do programa. Vamos ao exemplo? Observe a Tabela 4.

Tabela 4 - Exemplo de tratamento de exceções.

Programa em Java: Programa.java

Programa em Java: Programa.java

```
import java.util.Scanner;
import java.util.InputMismatchException;
public class Programa{
    public static void main(String args[]){
        double salario;
        try{//delimita a zona de perigo
        //na leitura pode ter exception
        System.out.println("Informe o salário: ");
        salario = new Scanner(System.in).nextDouble();
        //o que acontece qdo acontecer a exception será definido no catch.
        }catch(InputMismatchException ex){
            System.out.println("Ops! O valor informado é inválido");
        }
        System.out.println("Finalizando o programa normalmente.");
    }
}
```

EXEMPLO DA CONSOLE

```
Informe o salário: A
Ops! O valor informado é inválido
```

No programa anterior, quando acontece uma exceção na leitura do `double`, ao ser informado um texto, o fluxo interrompe e o bloco `catch` entra em execução imprime a mensagem. E o programa continua sua execução normalmente.

Para Refletir ⚡

Parou para se perguntar o que significa o `ex` que aparece no bloco `catch`?

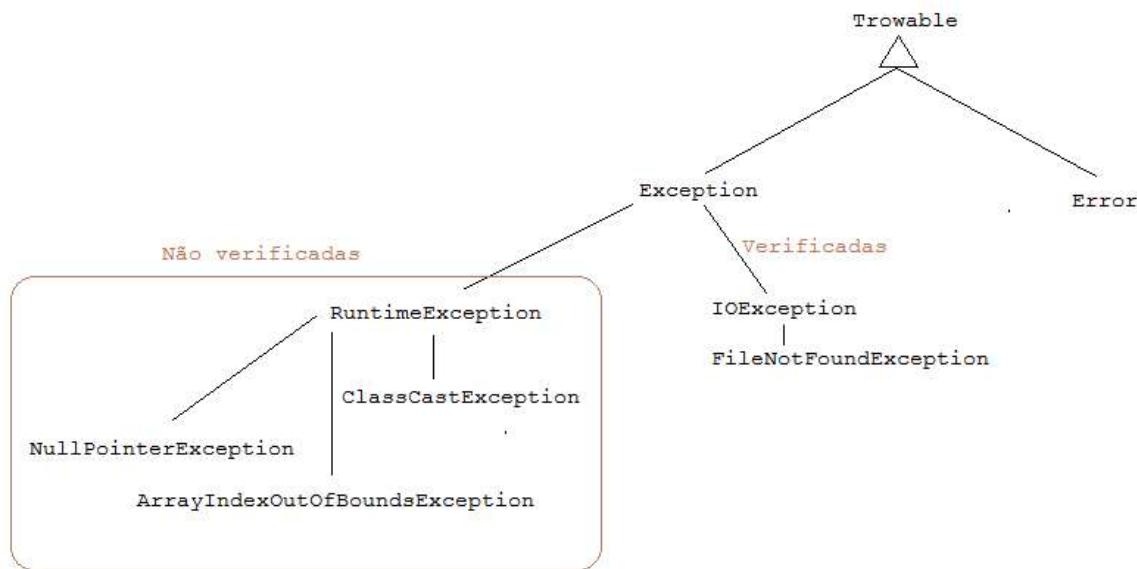
Observe o exemplo da Tabela 4 no comentário L1. Neste ponto, é possível de acontecer uma exceção, caso o usuário informe uma letra, ao invés de um número, não é mesmo? Caso ele informe uma letra, a máquina virtual não vai conseguir atribuir uma letra ao número (salário). Neste ponto, a JVM irá instanciar um objeto do tipo exceção

`InputMismatchException` e atribuir à referência ex e colocar o bloco `catch` em execução. Conforme mencionado anteriormente, a partir de ex será possível acessar todos os métodos do objeto correspondente da exceção, a exemplo, o `printStackTrace()`, que imprime toda a pilha de erros da máquina virtual. Experimente colocar ex. `printStackTrace()` abaixo da mensagem de erro no código da Tabela 4 e veja o que acontece.

A Herança das Exceções

Existem vários tipos de excepcionalidades no Java e elas são representadas por classes de exceções. No Java, existe uma árvore de herança que representa as possíveis exceções e cada uma com suas características. A Figura 1 é uma representação de uma pequena parte desta árvore. Observe:

Figura 1 - Pequena parte da árvore de herança das exceptions no Java.



O topo da árvore de herança é `Trowable`. É comum aparecer questões nas provas de certificações, afirmando que o topo da árvore de exceções é `Exception`, o que é errado, pois, como é mostrado pela figura, é `Trowable`.

Dois aspectos importantes da pequena árvore apresentada são bem importantes e merecem especial atenção.

Primeiramente, a classe Error. Erros não podem e não devem ser tratados. Quando um erro acontece é porque algo irrecuperável aconteceu. Imagine que um disco está corrompido e a JVM lança um Error. Não há o que fazer, o programador não conseguirá se recuperar desse tipo de erro por intermédio de programação. É sentar e chorar.

Em segundo lugar, é a subdivisão que existe abaixo da classe `Exception`. Uma exceção pode ser de dois tipos basicamente: as verificadas e as não verificadas.

As exceções verificadas são obrigadas a serem tratadas, ou seja, serem cercadas por um bloco `try`. As não verificadas podem ser tratadas de forma opcional.

Um exemplo de exceção verificada é a classe `IOException`, que estende (herda) `Exception`. Esta exceção é passível de ser lançada quando um programador está lendo, escrevendo ou copiando (manipulação) arquivos. Obrigatoriamente, nas chamadas de métodos de manipulação que lançam esse tipo de exceção, você deve cercar a codificação com bloco `try`.

Por outro lado, um exemplo de exceções não verificadas é a classe `ArrayIndexOutOfBoundsException`, que estende `RuntimeException`, conforme a Figura 1. Essa, apesar de poder ser lançada na manipulação de arrays, o código não é obrigado a ser cercado com bloco `try`.

Para Refletir ⚡

Qual é o propósito de tratar uma exceção não verificada do tipo `ArrayIndexOutOfBoundsException` ou `NullPointerException`? Devemos tratar esse tipo de exceção?

Na maioria das vezes, tratar exceções do tipo não verificadas é atestar a incompetência na manipulação de algum objeto. Por exemplo, tratar uma `ArrayIndexOutOfBoundsException`. O que você diria ao usuário? "Atenção! Não sabemos percorrer um array, contrate um programador mais capacitado...". Portanto, cuidado ao tratar exceções não verificadas.

Finalizando... C

Chegamos ao final... Nesta aula, você aprendeu como criar os enumeradores, que são constantes e agrupadores de informações, e teve uma introdução sobre as Exceptions no Java, que garantem a robustez da linguagem. Não foi objetivo deste conteúdo esgotar todo o conteúdo de Orientação a Objetos, e nem tão pouco preparar você para a certificação Java. Entretanto, o conteúdo da disciplina lhe deu uma base sólida e respeitou em todas as aulas o que você já conhecia de outras linguagens. Espera-se que você tenha feito todos os exercícios, pois somente a prática pode melhorar sua capacidade técnica em programação. Para fechar sua jornada, assista ao vídeo final que em um único projeto será abordado todos os conceitos vistos neste material das duas unidades, será uma despedida de honra, com uma visão prática. Afinal, esta disciplina apesar de ter vários conceitos, é puramente prática. Considere-se preparado para iniciar seus estudos. Dê uma passada no item “Na prática” e faça todas as questões. Ah! Que tal estudar para uma certificação Java com o desafio desta aula?

Sucesso!

Na Prática

"Prezado(a) estudante,

Esta seção é composta por atividades que objetivam consolidar a sua aprendizagem quanto aos conteúdos estudados e discutidos. **Caso alguma dessas atividades seja avaliativa, seu (sua) professor (a) indicará no Plano de Ensino e lhe orientará quanto aos critérios e formas de apresentação e de envio.**"

Bom Trabalho!

Atividade 01

^

Uma loja realiza o cadastro de seus clientes com as seguintes informações: nome, endereço e a forma de pagamento (Visa, Master ou Boleto). Crie uma classe Cliente que represente a abstração da loja e, para a forma de pagamento, crie um Enumerador chamado

EnumFPagamento. Faça um programa de teste que instancie uma entidade cliente popule com valores a sua escolha e faça a apresentação dos dados.

Atividade 02

^

Usando o mesmo contexto do exercício anterior, a representação da forma de pagamento já possui um código conhecido de cada um: para Visa é 20, Master é 50 e Boleto é 30. Use o recurso de criar atributos no Enumerador e acrescente esta peculiaridade. Para teste, crie um programa que receba os dados de vários clientes e o usuário informará os dados dos clientes e, em cada forma de pagamento, será informado o código conhecido. Seu programa deve fazer a atribuição ao cliente do enumerador, de acordo com o código informado.

Atividade 03



Crie uma entrada de dados validada e protegida contra exceções para todos os tipos primitivos apresentados na Aula 2. Crie uma classe Leitor que ofereça esse serviço para outros programadores utilizarem.

Referências

- BOOCH, G. **Object-Oriented Design with Applications**, Benjamin-Cummings, 1991.
- BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML – guia do usuário**. 2. ed. Rio de Janeiro: Campus. 2006.
- CARDELLI, L.; WEGNER, P. **On Understanding Types, Data Abstraction, and Polymorphism**. ACM Computing Surveys (CSUR). vol. 17, pp. 471-523. 1985.
- DEITEL H. M.; DEITEL, P. J. **Java, como programar**. 6. ed. Porto Alegre: Bookman. 2006.
- DICIONÁRIO AURELIO. 2017. Disponível em:
<https://contas.tcu.gov.br/dicionario/home.asp> Acesso em: 19 mar. 2017.
- ERICH, G. et al. **Padrões de projeto**: soluções reutilizáveis de software rientado a objetos. Porto Alegre: Bookman. 2000.
- HORSTMANN, C. S.; CORNELL, G. **Corejava 2 – Volume I – Fundamentals**. São Paulo: Makron Books. 2010.
- NEWRELIC. **The Most Popular Programming Languages of 2016**. 2016. Disponível em:
<https://blog.newrelic.com/2016/08/18/popular-programming-languages-2016-go>. Acesso em: 9 Mar 2017.
- NIEMEYER, P.; KNUDSEN, J. **Aprendendo Java**. Rio de Janeiro: Campus. 2000.
- ORACLE. **Java Licensing Logo**. 2017. Disponível em:
<http://www.oracle.com/us/technologies/java/java-licensing-logo-guidelines-1908204.pdf>. Acesso em: 13 mar. 2017.
- SIERRA, K.; BATES, B. **Certificação Sun para Programador JAVA 5 Guia de Estudo**. Rio de Janeiro: Alta Books, 2006.
- SILBERSCHATZ, A; GALVIN, P. B.; GAGNE, G. **Sistemas operacionais com Java**. 6. ed. Rio de Janeiro. Editora Campus, 2004.

Saiba Mais

Para ampliar seu conhecimento a respeito desse assunto, veja abaixo a(s) sugestão(ões) do professor:

- Imagine que, a partir de um código, você deseja fazer uma atribuição de um enum a um objeto. Por exemplo, o usuário informa um código 0 e você atribui o enumerador MATUTINO a um aluno, por exemplo. Assista ao **vídeo 1** para visualizar um exemplo deste tipo de implementação.

Vídeo 1

Prática Profissional - Programação orientada à objeto - Unida...



- Acesse o [blog de Sérgio Taborda](#) e conheça os métodos que existem nas exceptions.
- As exceções têm ampla aplicação e são muito utilizadas em arquiteturas e frameworks. Leia o [material sobre catch e throws em Exception](#).

- STUCKEY, P. J.; SULZMANN, M. **A theory of overloading.** International Conference on Functional Programming. Proceedings of the seventh ACM SIGPLAN international conference on Functional programming. Pittsburgh, PA, USA, 2002. pp. 167-178.
- WIKIPÉDIA. Desenvolvido pela Wikimedia Foundation. Conteúdo sobre o **Ambiente de Desenvolvimento Integrado.** 2017. Disponível em: <https://pt.wikipedia.org/wiki/Ambiente_de_desenvolvimento_integrado>. Acesso em: 5 mar. 2017.