



ENUMERAÇÕES E CLASSE BIGDECIMAL

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

herculano.debiasi@unoesc.edu.br



TÓPICOS

- Bibliografia
- Enumerações
- Classe `BigDecimal`

BIBLIOGRAFIA

■ Playlist 'COMO USAR BigDecimal COM JAVA AO INVÉS DE Double e Float'

YouTube video player interface showing a tutorial by Rodrigo Tavares from the channel Expertos Tech. The video is titled "COMO USAR BigDecimal COM JAVA AO INVÉS DE Double e Float | PARTE 1". The player shows the video progress at 0:03 / 12:24. The channel has 6,16 mil inscritos. The video has 141 likes and 1 dislike. The playlist on the right includes 6 videos: "COMO USAR BigDecimal COM JAVA AO INVÉS DE Double e...", "PARTE 2 | COMO USAR BigDecimal COM JAVA AO...", "PARTE 3 | COMO USAR BigDecimal COM JAVA AO...", "JAVA DESKTOP SE E SWING, COMO CRIAR UMA TELA...", "PARTE 2 | JAVA DESKTOP SE E SWING, COMO CRIAR UMA...", and "PARTE 2 | COMO USAR BigDecimal COM JAVA AO...". The video player also shows a guitar in the background and a "MATRIX" t-shirt on the presenter.

ENUMERAÇÕES

- Enumeração (*enum*) é um tipo de dados que serve para especificar, de forma **literal**, através de símbolos, um conjunto de constantes relacionadas
 - Enumerações deixam o código mais legível, aumentando sua semântica (significado)
 - Atualmente enumerações em Java são **classes** (herdam da classe `java.lang.Enum`) ou seja, podem possuir métodos e atributos, sendo muito mais poderosas do que as enumerações da maioria das outras linguagens de programação
 - Para iniciar os valores declarados dentro dos atributos da enumeração, é preciso declarar um construtor, que é sempre privado
 - Devem ser utilizadas letras maiúsculas para declarar as constantes da enumeração
 - Podem ser declaradas em arquivo próprio ou junto a outra classe (neste caso elas não podem ser públicas)
 - Enumerações podem ser utilizadas dentro de uma instrução `switch`
 - Os valores de uma enumeração podem ser iterados em um laço `for`
 - As constantes são implicitamente `static` e `final`
 - Utilize o operador `==` para comparar enumerações

ENUMERAÇÕES

- Principais métodos herdados da classe `java.lang.Enum`
 - `values()`: Retorna uma lista com os valores em um *array*
 - `ordinal()`: Retorna o índice da constante
 - `name()`: Retorna uma *string* correspondente ao nome de uma constante
 - `valueOf("NOME")`: Retorna a constante da enumeração (objeto da classe de enumeração) cujo nome foi especificado no argumento `NOME`
 - Este método é muito útil para converter um valor entrado pelo usuário com o teclado ou um formulário e convertê-lo para uma instância do tipo enumerado

ENUMERAÇÕES

- Para criar uma enumeração no STS pode-se clicar com o botão direito sobre a pasta *source* e escolher *new* → *enum*

New Enum Type

Create a new enum type.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected

Interfaces:

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

ENUMERAÇÕES

■ Exemplo básico com enumeração

```
1 package enumeracoes;
2
3 import java.util.EnumSet;
4 import java.util.Random;
5
6 public class TestaEnumDias {
7     // Criando uma enumeração como uma subclasse
8     private enum DiasSemana {
9         SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO
10    }
11
12    public static void main(String[] args) {
13        // Tipo da constante da enumeração
14        System.out.println(DiasSemana.SEGUNDA.getClass());
15        System.out.println();
16
17        // Atribuição e teste simples
18        DiasSemana diaDaSemana = DiasSemana.DOMINGO;
19        if (diaDaSemana == DiasSemana.DOMINGO) {
20            System.out.println("Acertou!");
21        }
22
23        // Métodos mais comuns
24        System.out.println(DiasSemana.SEXTA.ordinal());
25        System.out.println(DiasSemana.SEXTA.name());
26        System.out.println(DiasSemana.valueOf("SEGUNDA"));
27
28        // Transformando enumeração em array
29        DiasSemana diasSemana[] = DiasSemana.values();
30        System.out.println("\nExemplo com arrays: " + diasSemana[4]);
31
32        // Criando um subconjunto a partir da enumeração
33        EnumSet<DiasSemana> diasUteis = EnumSet.range(DiasSemana.SEGUNDA, DiasSemana.SEXTA);
34        System.out.println("\nDias úteis: " + diasUteis);
35
36        // Escolhendo uma constante da enumeração de forma aleatória
37        Random aleatorio = new Random();
38        int dia = aleatorio.nextInt(diasSemana.length);
39        DiasSemana diaSemana = diasSemana[dia];
40
41        switch (diaSemana) {
42            case SEGUNDA:
43            case TERCA:
44            case QUARTA:
45            case QUINTA:
46            case SEXTA:
47                System.out.println("Dia útil 🌞!");
48                break;
49            case SABADO:
50            case DOMINGO:
51                System.out.println("Fim de 🌙");
52                break;
53        }
54
55        // Utilizando 'switch expressions', disponível a partir do Java 14
56        String resultado = switch (diaSemana) {
57            case SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA -> "Dia útil 🌞!";
58            case SABADO, DOMINGO -> "Fim de 🌙";
59        };
60
61        System.out.println(resultado);
62    }
63 }
```

ENUMERAÇÕES

■ Exemplo com enumeração avançada

```
1 package enumeracoes;
2
3 import java.time.LocalDate;
4
5 public class TestaEnumMeses {
6
7     public static void main(String[] args) {
8         String mensagem;
9
10        for (Meses mes : Meses.values()) {
11            if (mes == Meses.FEVEREIRO) {
12                LocalDate dataAtual = LocalDate.now();
13
14                mensagem = String.format("%s é o %dº mês do ano e em %d tem %d dias",
15                    mes.getNome(),
16                    mes.getNumero(),
17                    dataAtual.getYear(),
18                    dataAtual.isLeapYear() ? 29 : 28);
19            } else {
20                mensagem = String.format("%s é o %dº mês do ano tem %d dias",
21                    mes.getNome(),
22                    mes.getNumero(),
23                    mes.getDias());
24            }
25
26            System.out.println(mensagem);
27        }
28    }
29 }
30 }
```

```
1 package enumeracoes;
2
3 public enum Meses {
4     JANEIRO(1, "Janeiro", 31),
5     FEVEREIRO(2, "Fevereiro", 28),
6     MARCO(3, "Março", 31),
7     ABRIL(4, "Abril", 30),
8     MAIO(5, "Maio", 31),
9     JUNHO(6, "Junho", 30),
10    JULHO(7, "Julho", 31),
11    AGOSTO(8, "Agosto", 31),
12    SETEMBRO(9, "Setembro", 30),
13    OUTUBRO(10, "Outubro", 31),
14    NOVEMBRO(11, "Novembro", 30),
15    DEZEMBRO(12, "Dezembro", 31);
16
17    private final int numero, dias;
18    private final String nome;
19
20    Meses(int numero, String nome, int dias) {
21        this.numero = numero;
22        this.nome = nome;
23        this.dias = dias;
24    }
25
26    public int getNumero() {
27        return this.numero;
28    }
29
30    public String getNome() {
31        return this.nome;
32    }
33
34    public int getDias() {
35        return this.dias;
36    }
37 }
```


CLASSE BIGDECIMAL

- O Java dispõe das classes `float` e `double` para realizar cálculos que exijam casas decimais, o problema é que elas exigem problemas de precisão por causa da forma como elas são representadas na memória do computador
- Essa forma de representação é definida pelo padrão [IEEE 754](#)
- Para mais detalhes veja esta [playlist](#)

```
1 package bigdecimal;
2
3 public class ProblemasFloatDouble {
4
5     public static void main(String[] args) {
6         // Problema com números em ponto flutuante
7         double a = 0.1;
8         double b = 0.2;
9         double c = a + b;
10
11         if (c != 0.3) {
12             System.out.println("Deu ruim: 0,1 + 0,2 != " + c);
13         }
14
15         // Problema com números em ponto flutuante
16         float num1 = 0.2f;
17         double num2 = 0.2;
18
19         System.out.println("float num1.: " + num1);
20         System.out.println("double num2: " + num2);
21
22         if (num1 == num2) {
23             System.out.println("São iguais");
24         } else {
25             System.out.println("São diferentes");
26         }
27
28         double valor = 0.2;
29         double total = 0.0;
30
31         for (int i = 0; i < 10; i++) {
32             total += valor;
33         }
34
35         System.out.println(total);
36     }
37
38 }
```

CLASSE `BigDecimal`

- Para cálculos que exijam precisão, como cálculos financeiros pode-se utilizar a classe `BigDecimal`
 - `BigDecimal` armazena os números em formato decimal
 - No caso de divisões é necessário especificar a precisão, caso contrário uma exceção poderá ser gerada em caso de resultados com dízimas ou infinitas casas decimais
 - Não se pode usar os operadores convencionais para fazer os cálculos, devendo-se usar os métodos apropriados da classe
 - Nunca se deve usar o método `equals()` para comparar objetos `BigDecimal` pois ele irá comparar objetos e não seus valores
 - Para compara valores deve-se usar o método `compareTo()` que devolve
 - 0 se os números forem iguais
 - 1 se o primeiro valor for maior do que o segundo
 - -1 se o primeiro valor for menor do que o segundo

CLASSE `BigDecimal`

■ Métodos

- Construtor: Não se deve usar o construtor que usa os parâmetros `float` ou `double` pois isso iria inserir um erro de precisão logo na declaração do objeto – use sempre o construtor passando como argumento um objeto `String`
- O método `valueOf()` pode ser usado com *floats* ou *doubles* sem problema pois internamente converte esses valores para *string* antes de criar o objeto
- Conversões para tipos primitivos podem ser feitos com métodos como `intValue()`, `longValue()`, `doubleValue()` e `floatValue()` – no caso dos tipos inteiros, a parte decimal do `BigDecimal` será truncada e não arredondada
- Para converter para `String` há o método `toPlainString()`
- Para definir a precisão, ou seja, o número de casas decimais, pode-se usar o método `setScale(casas decimais, forma de arredondamento)`
- A classe `BigDecimal` fornece um controle completo sobre o comportamento do arredondamento através da enumeração [RoundingMode](#)
- `add()`, `subtract()`, `multiply()` e `divide()` são os principais métodos para fazer realizar cálculos aritméticos, sendo que aceita `divide()` o tipo de arredondamento

CLASSE BigDecimal

Exemplos

```
1 package bigdecimal;
2
3 import java.math.BigDecimal;
4 import java.text.DecimalFormat;
5 import java.text.NumberFormat;
6 import java.util.Locale;
7
8 public class TipoBigDecimal {
9
10     public static void main(String[] args) {
11         Locale.setDefault(new Locale("pt", "BR"));
12         DecimalFormat df = new DecimalFormat("#,##0.00");
13
14         BigDecimal big1 = new BigDecimal("0.1");
15         BigDecimal big2 = new BigDecimal("0.2");
16         BigDecimal bigResult = big1.add(big2);
17
18         if (bigResult.compareTo(BigDecimal.valueOf(0.3)) == 0) {
19             System.out.println("Deu boa: 0,1 + 0,2 == " + df.format(bigResult.doubleValue()));
20         }
21
22         NumberFormat nf = NumberFormat.getCurrencyInstance();
23         System.out.println(nf.format(new BigDecimal("1234.56")));
24
25         System.out.println("\nSoma.....: " + new BigDecimal("2.00").add(new BigDecimal("1.2")));
26         System.out.println("Subtração....: " + new BigDecimal("2.00").subtract(new BigDecimal("1.1")));
27         System.out.println("Multiplicação: " + new BigDecimal("2.00").multiply(new BigDecimal("1.8")));
28
29         try {
30             System.out.println("Divisão.....: " + new BigDecimal("10.00").divide(new BigDecimal("3")));
31         } catch (ArithmeticException e) {
32             System.out.println("\n" + e.getMessage());
33         }
34
35         System.out.println("\nComparação 2 = 2: " + new BigDecimal("2.0").compareTo(new BigDecimal("2.0")));
36         System.out.println("Comparação 3 = 2: " + new BigDecimal("3.0").compareTo(new BigDecimal("2.0")));
37         System.out.println("Comparação 2 = 3: " + new BigDecimal("2.0").compareTo(new BigDecimal("3.0")));
38     }
39
40 }
```

CLASSE BigDecimal


■ Resumo do funcionamento dos modos de arredondamento

	Result of rounding input to one digit with the given rounding mode							
Input Number	UP	DOWN	CEILING	FLOOR	HALF_UP	HALF_DOWN	HALF_EVEN	UNNECESSARY
5.5	6	5	6	5	6	5	6	throw ArithmeticException
2.5	3	2	3	2	3	2	2	throw ArithmeticException
1.6	2	1	2	1	2	2	2	throw ArithmeticException
1.1	2	1	2	1	1	1	1	throw ArithmeticException
1.0	1	1	1	1	1	1	1	1
-1.0	-1	-1	-1	-1	-1	-1	-1	-1
-1.1	-2	-1	-1	-2	-1	-1	-1	throw ArithmeticException
-1.6	-2	-1	-1	-2	-2	-2	-2	throw ArithmeticException
-2.5	-3	-2	-2	-3	-3	-2	-2	throw ArithmeticException
-5.5	-6	-5	-5	-6	-6	-5	-6	throw ArithmeticException

CLASSE BIGDECIMAL


■ Arredondamento RoundingMode

■ UP: O arredondamento se distancia do 0



5.5	=>	6
2.5	=>	3
1.6	=>	2
1.1	=>	2
ZERO		
1.0	=>	1
-1.0	=>	-1
-1.1	=>	-2
-1.6	=>	-2
-2.5	=>	-3
-5.5	=>	-6

■ DOWN: O arredondamento aproxima o número do 0




5.5	=>	5
2.5	=>	2
1.6	=>	1
1.1	=>	1
1.0	=>	1
ZERO		
-1.0	=>	-1
-1.1	=>	-1
-1.6	=>	-1
-2.5	=>	-2
-5.5	=>	-5

CLASSE BIGDECIMAL


■ Arredondamento RoundingMode

- CEILING: Se o número for positivo funciona como o UP e se for negativo funciona como o DOWN



5.5	=>	6
2.5	=>	3
1.6	=>	2
1.1	=>	2
1.0	=>	1
ZERO		
-1.0	=>	-1
-1.1	=>	-1
-1.6	=>	-1
-2.5	=>	-2
-5.5	=>	-5

- FLOOR: Funcionamento oposto ao do CEILING



5.5	=>	5
2.5	=>	2
1.6	=>	1
1.1	=>	1
1.0	=>	1
ZERO		
-1.0	=>	-1
-1.1	=>	-2
-1.6	=>	-2
-2.5	=>	-3
-5.5	=>	-6

CLASSE BigDecimal

■ Arredondamento RoundingMode

- HALF_DOWN: Se a próxima casa decimal for $\leq 0,5$ então arredonda para baixo caso contrário arredonda para cima

5.5	=>	5		
2.5	=>	2		
1.6	=>	2		
1.1	=>	1		
1.0	=>	1		
ZERO				
-1.0	=>	-1		
-1.1	=>	-1		
-1.6	=>	-2		
-2.5	=>	-2		
-5.5	=>	-5		
			1.7	=> 2
			1.6	=> 2
			1.5	=> 1
			1.4	=> 1
			1.3	=> 1
			1.2	=> 1

- HALF_UP: Se a próxima casa decimal for $\geq 0,5$ então arredonda para cima caso contrário arredonda para baixo

5.5	=>	6		
2.5	=>	3		
1.6	=>	2		
1.1	=>	1		
1.0	=>	1		
ZERO				
-1.0	=>	-1		
-1.1	=>	-1		
-1.6	=>	-2		
-2.5	=>	-3		
-5.5	=>	-6		
			1.7	=> 2
			1.6	=> 2
			1.5	=> 2
			1.4	=> 1
			1.3	=> 1
			1.2	=> 1

CLASSE BIGDECIMAL

■ Arredondamento RoundingMode

- HALF_EVEN: Se a casa decimal for $\leq 0,4$ então arredonda para baixo, se for $\geq 0,6$ arredonda para cima e se for exatamente 0,5 então arredonda para o número par mais próximo

- Esta é a forma de arredondamento usada pelo Banco Central do Brasil e é definido na [ABNT NBR 5891](#)

5.5	=>	6	1.7	=>	2	PAR
2.5	=>	2	1.6	=>	2	
1.6	=>	2	1.5	=>	2	
1.1	=>	1	1.4	=>	1	
1.0	=>	1	1.3	=>	1	
ZERO						
-1.0	=>	-1	4.7	=>	5	PAR
-1.1	=>	-1	4.6	=>	5	
-1.6	=>	-2	4.5	=>	4	
-2.5	=>	-2	4.4	=>	4	
-5.5	=>	-6	4.3	=>	4	

- Nesta [resolução](#) do Banco Central pode-se ver a exigência de precisão de 4 casas decimais para cálculo de taxas
- Essa forma de arredondamento é também chamada de '[arredondamento bancário](#)' e é o comportamento padrão para arredondamento em operações binárias em ponto flutuante do padrão IEEE 754

CLASSE BigDecimal

■ Exemplos

```
1 package bigdecimal;
2
3 import java.math.BigDecimal;
4 import java.math.MathContext;
5 import java.math.RoundingMode;
6
7 public class ArredondamentosBigDecimal {
8
9     public static void main(String[] args) {
10         BigDecimal numerador = new BigDecimal("1.00");
11         BigDecimal denominador = new BigDecimal("1.30");
12
13         System.out.println("Divisão (UP).....: " + numerador.divide(denominador, 2, RoundingMode.UP));
14         System.out.println("Divisão (DOWN).....: " + numerador.divide(denominador, 2, RoundingMode.DOWN));
15         System.out.println("Divisão (CEILING)...: " + numerador.divide(denominador, 2, RoundingMode.CEILING));
16         System.out.println("Divisão (FLOOR)....: " + numerador.divide(denominador, 2, RoundingMode.FLOOR));
17         System.out.println("Divisão (HALF-UP)...: " + numerador.divide(denominador, 2, RoundingMode.HALF_UP));
18         System.out.println("Divisão (HALF-DOWN): " + numerador.divide(denominador, 2, RoundingMode.HALF_DOWN));
19         System.out.println("Divisão (HALF-EVEN): " + numerador.divide(denominador, 2, RoundingMode.HALF_EVEN));
20
21         BigDecimal resultado = new BigDecimal("2.00").pow(10, new MathContext(5, RoundingMode.HALF_EVEN));
22         System.out.println("\nPotência: " + resultado);
23
24         MathContext mc = new MathContext(5, RoundingMode.HALF_EVEN);
25         BigDecimal potencia = new BigDecimal("2.00").pow(10);
26         BigDecimal arredondado = potencia.round(mc);
27         System.out.println("Potência: " + arredondado);
28     }
29
30 }
```