Enumerações e Classe BigDecimal

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI herculano.debiasi@unoesc.edu.br

TÓPICOS

- Bibliografia
- Enumerações
- Classe BigDecimal























Apoiadores:



BIBLIOGRAFIA

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



- 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











- 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





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

New Enum Type	_		×				
Enum Type Create a new enum	type.	E					
Source fol <u>d</u> er:	TiposDeDados/src	Br <u>o</u> wse					
Pac <u>k</u> age:	enumeracoes	Bro <u>w</u> se					
Enclosing type:		Bro <u>w</u> se					
Na <u>m</u> e: Modifiers: <u>I</u> nterfaces:	EstadoPedido public package private protected	<u>A</u> dd <u>R</u> emove	<u>.</u>				
Do you want to add comments? (Configure templates and default value here) Generate comments							
?	<u>E</u> inish	Cancel					



















Exemplo básico com enumeração



```
package enumeracoes;
 3 import java.util.EnumSet;
   import java.util.Random;
    public class TestaEnumDias {
        // Criando uma enumeração como uma subclasse
80
        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
            // Métodos mais comuns
23
            System.out.println(DiasSemana.SEXTA.ordinal());
24
25
            System.out.println(DiasSemana.SEXTA.name());
26
            System.out.println(DiasSemana.valueOf("SEGUNDA"));
27
28
            // Transformando enumeração em array
            DiasSemana diasSemana[] = DiasSemana.values();
29
            System.out.println("\nExemplo com arrays: " + diasSemana[4]);
30
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];
41
            switch (diaSemana) {
                case SEGUNDA:
43
                case TERCA:
                case QUARTA:
45
                case QUINTA:
                case SEXTA:
47
                    System.out.println("Dia útil 22:");
48
                    break;
49
                case SABADO:
                case DOMINGO:
                    System.out.println("Finde @");
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 22!";
58
                case SABADO, DOMINGO -> "Finde @";
59
            };
60
61
            System.out.println(resultado);
62
```

Exemplo com enumeração avançada

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



30 }

```
package enumeracoes;
 2
   public enum Meses {
       JANEIRO(1, "Janeiro", 31),
       FEVEREIRO(2, "Fevereiro", 28),
       MARCO(3, "Março", 31),
       ABRIL(4, "Abril", 30),
       MAIO(5, "Maio", 31),
       JUNHO(6, "Junho", 30),
 9
       JULHO(7, "Julho", 31),
10
11
       AGOSTO(8, "Agosto", 31),
       SETEMBRO(9, "Setembro", 30),
12
       OUTUBRO(10, "Outubro", 31),
13
       NOVEMBRO(11, "Novembro", 30),
14
       DEZEMBRO(12, "Dezembro", 31);
15
16
       private final int numero, dias;
17
18
       private final String nome;
19
20⊝
       Meses(int numero, String nome, int dias) {
21
           this.numero = numero;
22
           this.nome = nome;
           this.dias = dias;
23
24
25
       public int getNumero() {
26⊜
27
           return this.numero;
28
29
       public String getNome() {
300
           return this.nome;
31
32
33
       public int getDias() {
34⊖
           return this.dias;
35
36
```

4

5⊚

6

9

10

11

12

13

14 15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30 31

32

33 34

35

36

- 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 <u>IEEE 754</u>
 - Para mais detalhes veja esta <u>playlist</u>

```
UNOESC
```

```
1 package bigdecimal;
  public class ProblemasFloatDouble {
       public static void main(String[] args) {
           // Problema com números em ponto flutuante
           double a = 0.1;
           double b = 0.2;
           double c = a + b;
           if (c != 0.3) {
               System.out.println("Deu ruim: 0,1 + 0,2 != " + c);
           }
           // Problema com números em ponto flutuante
           float num1 = 0.2f;
           double num2 = 0.2;
           System.out.println("float num1.: " + num1);
           System.out.println("double num2: " + num2);
           if (num1 == num2) {
               System.out.println("São iguais");
           } else {
               System.out.println("São diferentes");
           }
           double valor = 0.2;
           double total = 0.0;
           for (int i = 0; i < 10; i++) {
               total += valor;
           }
           System.out.println(total);
38 }
```

- 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
 - lacksquare -1 se o primeiro valor for menor do que o segundo





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



















39 40 }

Exemplos

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



Resumo do funcionamento dos modos de arredondamento

	Res	ult of	rounding	input t	o one dig	it with the g	jiven roundi	ng 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















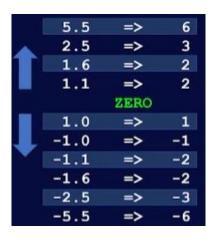


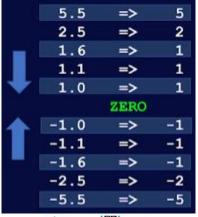


Arredondamento RoundingMode

■ UP: O arredondamento se distancia do 0

■ DOWN: O arredondamento aproxima o número do 0



















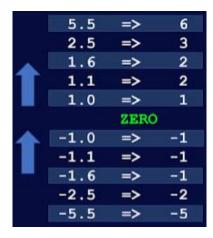


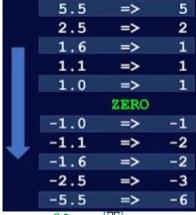


Arredondamento RoundingMode

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

FLOOR: Funcionamento oposto ao do CEILING



















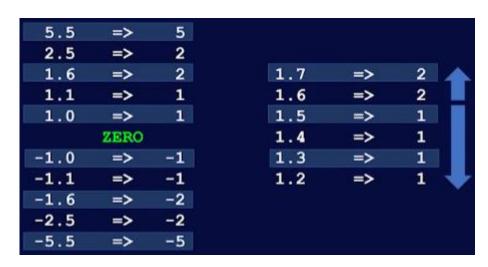


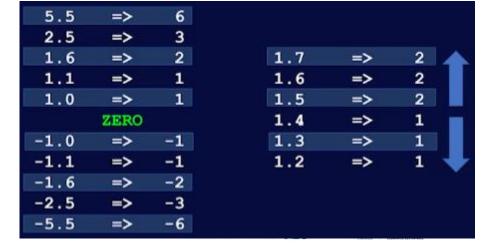


Arredondamento RoundingMode

■ HALF_DOWN: Se a próxima casa decimal for ≤ 0,5 então arredonda para baixo caso contrário arredonda para cima

■ HALF_UP: Se a próxima casa decimal for ≥ 0,5 então arredonda para cima caso contrário arredonda para baixo







- Arredondamento RoundingMode
 - HALF_EVEN: Se a casa decimal for ≤ 0,4 então arredonda para baixo, se for ≥ 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 <u>ABNT NBR 5891</u>

5.5	=>	6	1.7	=>	2
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
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 <u>resolução</u> 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





Exemplos

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