

Aula 5

Campos e Métodos Estáticos, Tipos Primitivos e Tipos por Referência

Rafael Geraldelli Rossi

Métodos e Campos Static

- Até agora vimos que para **acessar um campo** ou chamar um método do objeto era necessário instanciar um objeto

```
new BlaBlaBla(argumentos do construtor);
```

- Lembrando que a instanciação do objeto reserva um espaço de memória e insere nesse espaço os campos e métodos do objeto
- Porém, as vezes é interessante ou necessário **criar/utilizar métodos** ou até mesmo **acessar de valores de campos** de um objeto **sem precisar criar o objeto**

Métodos e Campos Static

Isso te parece estranho?



Métodos e Campos Static

Isso te parece estranho?



Se sim, por que vocês vêm fazendo uso desse conceito desde o começo?

Métodos e Campos Static

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         System.out.println("Hello World!");  
6  
7     }  
8  
9 }
```

Alguém aí precisou criar um objeto da classe `System` para usar o método `Println(...)`?

Métodos e Campos Static

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         System.out.println("Hello World!");  
6  
7     }  
8  
9 }
```

Já perceberam também que não precisamos criar um objeto da classe que contém o método `main(...)`

Métodos e Campos Static

- Lembrando que quando instanciamos objetos, estamos criando espaço para seus campos e métodos **dinamicamente**, ou seja, **os membros da classe são criados a medida que os objetos são criados**
- Quando criamos métodos ou campos os quais não precisam da instanciação de um objeto para serem utilizados, vamos nos referir a estes como **estáticos**

Métodos e Campos Static

- Um membro de classe estático é criado automaticamente mesmo que não seja instanciado uma objeto da classe que os contém
- A criação de um membro estático faz com que **métodos e campos pertençam à uma classe e não à um objeto**, ou seja, irão existir mesmo sem a criação do objeto

Métodos e Campos Static

- Se o método `println(...)` não fosse estático, teríamos que criar um objeto que o contém para depois invocá-lo

```
1 import java.io.PrintStream;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         PrintStream teste = new PrintStream(System.out);
8         teste.println("Hello World!");
9     }
10 }
11
12
13 }
```

Salda - Teste (run) x Resultados da Pesquisa x

run:
Hello World!

- OBSERVAÇÃO:** vamos esquecer esse exemplo acima já que o método `println(...)` é estático

Métodos Static

- Muitos métodos `static` são utilizados para realizar **tarefas corriqueiras** em que **não se que perder tempo com a criação de objetos**
- Para declarar um método como `static`, coloque a palavra-chave `static` antes do tipo de retorno na declaração do método
- **OBSERVAÇÃO:** um método estático só pode chamar outro método se esse também for estático

Métodos Static

```
1 public class Matematica {  
2  
3     public static double eleva(double base, double expoente){  
4         double resultado = 1;  
5  
6         for(int i=0;i<expoente;i++){  
7             resultado *= base;  
8         }  
9  
10        return resultado;  
11    }  
12  
13 }
```

- Um método `static` é chamado especificando o nome da classe em que o método foi declarado, seguido por um "." e pelo nome do método (com seus respectivos argumentos)

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         System.out.println("O resultado de 2 elevado a 3 é: " + Matematica.eleva(2, 3));  
6  
7     }  
8  
9 }
```

Métodos Static

- **OBSERVAÇÃO:** a classe `Math`, do pacote `java.lang`, apresenta vários métodos `static` interessantes para se fazer operações matemáticas

Método	Descrição	Exemplo
<code>abs(x)</code>	Valor absoluto de x	<code>abs(-23.7)</code> é 23.7
<code>ceil(x)</code>	Arredonda x para o menor inteiro não menor que x (arredonda para cima)	<code>ceil(9.3)</code> é 10.0
<code>cos(x)</code>	Co-seno trigonométrico de x (x em radianos)	<code>cos(0.0)</code> é 1.0
<code>exp(x)</code>	Método exponencial e^x	<code>exp(1.0)</code> é 2.71828
<code>floor(x)</code>	Arredonda x para o maior inteiro não maior que x (arredonda para baixo)	<code>floor(9.3)</code> é 9.0
<code>log(x)</code>	Logaritmo natural de x (base e)	<code>log(1.0)</code> é 0.0
<code>log10(x)</code>	Logaritmo de x na base 10	<code>log10(10.0)</code> é 1.0
<code>max(x, y)</code>	Maior valor entre x e y	<code>max(1.0, 5.0)</code> é 5.0
<code>min(x, y)</code>	Menor valor entre x e y	<code>min(1.0, 5.0)</code> é 1.0
<code>pow(x, y)</code>	x elevado a potência de y (isto é, x^y)	<code>pow(2.0, 3.0)</code> é 8.0
<code>sin(x)</code>	Seno trigonométrico de x (x em radianos)	<code>sin(0.0)</code> é 0.0
<code>sqrt(x)</code>	Raiz quadrada de x	<code>sqrt(4.0)</code> é 2.0
<code>tan(x)</code>	Tangente trigonométrica de x (x em radianos)	<code>tan(0.0)</code> é 0

Métodos Static

Exemplo de uso da classe Math

```
1 public class Teste {
2
3     public static void main(String args[]){
4
5         double num1 = 3.4;
6         double num2 = 8.1;
7
8         num1 = Math.ceil(num1); // Arredondando o num1 para cima
9         num2 = Math.floor(num2); // Arredondando o num2 para baixo
10
11         System.out.println("Num1: " + num1);
12         System.out.println("Num2: " + num2);
13         System.out.println("O maior dos dois é: " + Math.max(num1, num2));
14         System.out.println("A raiz quadrada de " + num2 + " é: " + Math.sqrt(num2));
15     }
16 }
17
18 }
19
```

run:
Num1: 4.0
Num2: 8.0
O maior dos dois é: 8.0
A raiz quadrada de 8.0 é: 2.8284271247461903

Campos Static

- Um campo estático é declarado colocando a palavra-chave **static** antes do tipo do campo

```
1 public class Pessoa {  
2  
3     private static int qtd_pessoas;  
4     private int id;  
5     private String nome;  
6     private String cpf;  
7 }
```

Criando um
campo estático

Campos Static

- Manipulação de campos estáticos
 - Acesso restrito (ex: privado)
 - Por meio de métodos get e set se o campo for privado
 - Publico não restrito (ex: público)
 - Por meio da utilização do "." em frente ao nome da classe seguido pelo identificador do campo estático

Campos Static

- Uma diferença importante de campos estáticos para campos não estáticos é que **não é criada uma cópia do campo estático para cada novo objeto que é instanciado**
- Portanto, **os campos estáticos são compartilhados entre todos os objetos da classe**
- Essas variáveis podem ser usadas, por exemplo, para monitorar o número de objetos que foram criados em uma classe, ou ainda definir um valor ou informação que deve ser compartilhada entre todos os objetos da mesma classe

Campos Static

Exemplo: monitorando o número de objetos criados para uma classe

```
1 public class Pessoa {  
2  
3     private static int qtd_pessoas;  
4     private int id;  
5     private String nome;  
6     private String cpf;  
7  
8     public Pessoa(String nome, String cpf) {  
9         qtd_pessoas++;  
10        this.id = qtd_pessoas;  
11        this.nome = nome;  
12        this.cpf = cpf;  
13    }  
14  
15    public void imprimeStatus(){  
16        System.out.println("=====");  
17        System.out.println("Total de pessoas: " + this.qtd_pessoas);  
18        System.out.println("ID: " + this.id);  
19        System.out.println("Nome: " + this.nome);  
20        System.out.println("CPF: " + this.cpf);  
21        System.out.println("=====");  
22    }  
}
```

Campos Static

Exemplo: monitorando o número de objetos criados para uma classe

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         Pessoa pessoa1 = new Pessoa("Rafael", "000.000.000-78");
6         Pessoa pessoa2 = new Pessoa("Ricardo", "111.111.111-45");
7         Pessoa pessoa3 = new Pessoa("Vitor", "222.222.222-12");
8
9         pessoa1.imprimeStatus();
10        pessoa2.imprimeStatus();
11        pessoa3.imprimeStatus();
12    }
13 }
14 }
```

Saída - Teste (run) x Resultados da Pesquisa x

```
run:
=====
Total de pessoas: 3
ID: 1
Nome: Rafael
CPF: 000.000.000-78
=====
Total de pessoas: 3
ID: 2
Nome: Ricardo
CPF: 111.111.111-45
=====
Total de pessoas: 3
ID: 3
Nome: Vitor
CPF: 222.222.222-12
=====
```

Campos Static

Exemplo: classe Conta na qual é possível desabilitar operações de saque e depósito de todas as contas com campo estático

```
11 public class Conta {  
12  
13     public static boolean permitirMovimentacao = true;  
14  
15     private String nome;  
16     private double saldo;  
17  
18     public Conta(String nome, double saldo){  
19         this.nome = nome;  
20         this.saldo = saldo;  
21     }  
22  
23     public void sacar(double valor){  
24         if(permitirMovimentacao == true){  
25             this.saldo -= valor;  
26         }  
27     }  
28  
29     public void depositar(double valor){  
30         if(permitirMovimentacao == true){  
31             this.saldo += valor;  
32         }  
33     }  
}
```

Campos Static

- A classe `Math` declara dois campos que representam constantes matemáticas comumente utilizadas
 - `Math.PI`: 3,141592653589... (relação da circunferência de um círculo com seu diâmetro)
 - `Math.E`: 2,7182818284590... (valor da base para logaritmos naturais)
- Esses campos são declarados na classe `Math` com os modificadores
 - `public`: permite que você use os campos nas suas próprias classes
 - `final`: significa que o valor é constante e não pode ser alterado depois que o campo for inicializado
 - `static`: permite que sejam acessados pelo nome da classe sem a necessidade da criação de um objeto

Campos Static

Exemplo de uso de campos estáticos da classe Math

```
1  import java.util.Scanner;
2
3  public class Programa {
4
5      public static void main(String[] args){
6
7          Scanner teclado = new Scanner(System.in);
8
9          System.out.print("Digite o raio do círculo: ");
10         double raio = teclado.nextDouble();
11
12         double area = Math.PI * Math.pow(raio, 2);
13         double perimetro = 2 * Math.PI * raio;
14
15         System.out.println("A área do círculo é: " + area);
16         System.out.println("O perímetro do círculo é: " + perimetro);
17     }
18 }
19
20
21
```

run:
Digite o raio do círculo: 5,8
A área do círculo é: 105.68317686676065
O perímetro do círculo é: 36.4424747816416

Observações

- Para chamar um método dentro de outro método estático, o **método sendo chamado tem que ser estático também** → garantia que o método sendo chamado já está criado também

```
public class TesteReferen
{
    public static void main(String[] args)
    {
        imprimeBlaBlaBla();
    }

    public void imprimeBlaBlaBla(){
        for(int i=0;i<3;i++){
            System.out.print("Bla");
        }
    }
}
```

non-static method imprimeBlaBlaBla() cannot be referenced from a static context

(Alt-Enter mostra dicas)



Observações

- Para chamar um método dentro de outro método estático, o **método sendo chamado tem que ser estático também** → garantia que o método sendo chamado já está criado também

```
14 public class TesteReferencia {  
15  
16     public static void main(String[] args) {  
17         imprimeBlaBlaBla();  
18     }  
19  
20     public static void imprimeBlaBlaBla(){  
21         for(int i=0;i<3;i++){  
22             System.out.print("Bla");  
23         }  
24     }  
25  
26 }
```



Observações

- Um método estático não pode fazer uso de um campo não estático → não a garantias que o campo foi criado na memória

```
10 public class ContaBancaria {  
11  
12     private static String descricaoClasse;  
13  
14     private Pessoa proprietario;  
15     private double saldo;  
16     private int numeroConta;  
17     private int agenciaCo  
18  
19     public static Pessoa  
20         return proprietario;  
21 }  
22  
23 }
```

non-static variable proprietario cannot be referenced from a static context

(Alt-Enter mostra dicas)



Tipos Primitivos e Tipos por Referência

- Os tipos dos atributos (variáveis) em Java são divididos em **tipos primitivos** e **tipos por referência**
- Ambos os tipos podem ser utilizado como **tipos de retorno** ou como **parâmetros** de **métodos**

Tipos Primitivos

- As variáveis de tipos primitivos são aquelas que **armazenam informações mais usuais e básicas**
- Podemos entender também as variáveis de tipos primitivos como aquelas que **armazenam um único valor** (sem métodos ou propriedades pertinentes ao objetos)
- Os tipos primitivos em Java são:
 - **boolean** (1 bit): não é um valor numérico e só admite os valores true ou false
 - **char** (2 bytes): usa o código UNICODE 16 bits para representar um caractere
(<https://www.fileformat.info/info/charset/UTF-16/list.htm>)

Tipos Primitivos

- Existem **diferentes tipos primitivos para armazenar números inteiros**
- Eles se **diferem na quantidade de bytes** que usam para armazenar os números → quanto maior o número de bytes, maior o intervalo de valores que podem ser armazenados
 - Os tipos primitivos para armazenar número inteiros em Java são:
 - **byte** (1 byte): aceita até 256 valores e compreende os valores no intervalo [-128, 127]
 - **short** (2 bytes): aceita valores entre [-32.768, 32.767]
 - **int** (4 bytes): compreende valores entre [-2.147.483.648, 2,147,483,647]
 - **long** (8 bytes): compreende valores entre [-9.223.372.036.854.775.808, 9.223.372.036.854.775.807]

Tipos Primitivos

- Também existem **diferentes tipos primitivos para armazenar números reais** (ponto flutuante) diferindo no número de bits utilizados para armazenar os números
 - Os tipos primitivos para armazenar números reais em Java são:
 - **float** (4 bytes): não é aconselhável para representar valores com casas decimais (até 6 dígitos significativos)
 - **double** (8 bytes): precisão maior que o float (até 15 dígitos significativos)

Tipos Primitivos

- **OBSERVAÇÃO:** caso as variáveis de tipo primitivo sejam campos de objetos e caso não sejam atribuídos valores no momento da criação das variáveis, o Java atribui valores padrão:
 - 0 para os tipos byte, char, short, int, long, float e double
 - false para o tipo boolean

Tipos por Referência

- Os programas utilizam as variáveis de tipos por referência (ou somente **referência**) para armazenar as localizações de objetos na memória do computador
- Diz-se que tal variável **referencia um objeto**
- Os objetos que são referenciados podem todos conter muitas variáveis de instância e métodos

Tipos por Referência

- Um tipo por referência é criada utilizando a palavra-chave **new**
- **OBSERVAÇÃO:** quando criado um objeto, por padrão, os campos do tipo referência são inicializados como `null`

Passagem de Parâmetros por Valor e por Referência

Lembram das outras disciplinas a questão de passagem por valor e passagem por referência na chamada de métodos/funções?

- **Passagem por valor:** argumento é copiado e passado para o método → alterações da variável dentro do método não terão efeito no valor da variável na classe que originou a chamada
- **Passagem por referência:** a referência da variável é passada para o método → qualquer alteração na região de memória apontada pela referência será refletida no valor da variável na classe que originou a chamada

Passagem de Parâmetros por Valor

- Tipos primitivos passados como parâmetros não têm seus valores alterados (**passagem por valor**)

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         int valor = 10;  
6         modificaValor(valor);  
7         System.out.println("Valor dentro da função principal após a chamada da função modificadora: " + valor);  
8     }  
9  
10    public static void modificaValor(int valor){  
11        valor = valor * 1000000;  
12        System.out.println("Valor dentro da função modificadora: " + valor);  
13    }  
14  
15 }  
16
```

Saída - Teste (run) x Resultados da Pesquisa x

```
run:  
Valor dentro da função modificadora: 10000000  
Valor dentro da função principal após a chamada da função modificadora: 10
```

Passagem de parâmetros por Valor e por Referência

- Pode-se alterar os valores de campos de objetos dentro dos métodos (**passagem por referência**)

```
1 public class Programa {  
2  
3     public static void main(String[] args){  
4  
5         Pessoa pessoa = new Pessoa("Rafael", "000.000.000-98");  
6         modificaValor(pessoa);  
7         System.out.println("Valor dentro da função principal após a chamada da função modificadora: " + pessoa.getNome());  
8     }  
9  
10    public static void modificaValor(Pessoa pessoa){  
11        pessoa.setNome(pessoa.getNome() + " Rossi");  
12        System.out.println("Valor dentro da função modificadora: " + pessoa.getNome());  
13    }  
14 }  
15  
16
```

Salda - Teste (run) x Resultados da Pesquisa x

run:
Valor dentro da função modificadora: Rafael Rossi
Valor dentro da função principal após a chamada da função modificadora: Rafael Rossi

Classes Wrappers

- As classes Wrappers são **classes especiais** pelas seguintes razões
 - Por serem criadas sem a utilização da palavra-chave `new`
 - Podem encapsular tipos primitivos para serem trabalhados como objetos
 - Possuem métodos para fazer conversões para variáveis primitivas (principalmente *strings*).
- Existe uma classe Wrapper para cada tipo primitivo em Java: Boolean, Character, Byte, Short, Integer, Long, Float e Double

Classes Wrappers

- A instanciação de objetos dessas classe pode ser feita de maneira direta ou da forma tradicional de instanciação de objetos

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         Integer num1 = 5;
6         Integer num2 = new Integer(7);
7
8         System.out.println("O valor de num1 é " + num1);
9         System.out.println("O valor de num2 é " + num2.intValue());
10
11     }
12
13 }
14 }
```

Salda x Resultados da Pesquisa x

Teste (run) x Console do Depurador x

run:
O valor de num1 é 5
O valor de num2 é 7

Classes Wrappers

- No caso da instanciação, os construtores dos Wrappers podem tanto receber o valor correspondente no tipo primitivo ou uma *string* (com exceção da classe Character)

```
1 public class Programa {
2
3     public static void main(String[] args){
4
5         Integer num1 = new Integer(5);
6         Integer num2 = new Integer("7");
7
8         System.out.println("O valor de num1 é " + num1);
9         System.out.println("O valor de num2 é " + num2.intValue());
10
11     }
12
13 }
14
15
```

Programa

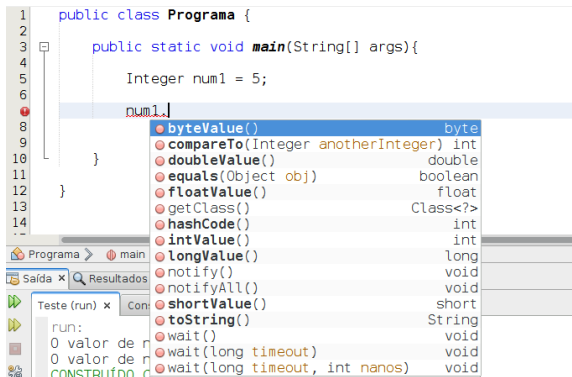
Saída x Resultados da Pesquisa x

Teste (run) x Console do Depurador x

run:
O valor de num1 é 5
O valor de num2 é 7

Classes Wrappers

- Os Wrappers possuem uma serie de métodos utilitários, principalmente para a conversão de tipos



Classes Wrappers

- Os Wrappers possuem métodos estáticos para a conversão de *strings* em tipos primitivos (métodos parse)
- Exemplos:
 - Conversão para booleano:** `Boolean.parseBoolean("true")`
 - Conversão para inteiro:** `Integer.parseInt("10")`
 - Conversão para float:** `Float.parseFloat("1.156")`
 - Conversão para double:** `Double.parseDouble("15.668897")`
- OBSERVAÇÃO:** a classe `String`, por definição, não é um *wrapper*, mas terá o mesmo comportamento de um *wrapper* (pois encapsula um vetor de tipos primitivos)

Classes Wrappers

- Apesar de serem classes, os *Wrappers* quando passados para função, realizam a passagem por valor

```
13 public class TesteReferencia {
14
15     public static void main(String[] args) {
16
17         Integer teste = 5;
18         alteraInteger(teste);
19
20         System.out.println(teste);
21
22     }
23
24     public static void alteraInteger(Integer num){
25
26         num = num + 5;
27     }
28
29 }
```

Localizar: compute Anterior Próximo

Resultados da Pesquisa Saída x

Console do Depurador x Teste (run) x

run:
Teste de String.
CONSTRUÍDO COM SUCESSO (tempo total: 1 segundo)

Classes Wrappers

- Apesar de serem classes, os *Wrappers* quando passados para função, realizam a passagem por valor

```
13 public class TesteReferencia {
14
15     public static void main(String[] args) {
16
17         String teste = "Teste de String.";
18         alteraString(teste);
19
20         System.out.println(teste);
21     }
22
23     public static void alteraString(String str){
24
25         str = str + "Alterada";
26     }
27
28 }
29
```

Localizar: compute Anterior Próximo

Resultados da Pesquisa Saída x

Console do Depurador x Teste (run) x

run:
Teste de String.
CONSTRUIDO COM SUCESSO (tempo total: 1 segundo)

Extra: Trabalhando Com Números Muito Grandes

- Apesar dos tipos primitivos para armazenar números inteiros e reais do Java suportarem um grande intervalo de valores, em algumas situações é necessário armazenar valores maiores que os suportados por um `long` ou por um `double`
- **Exemplos:**
 - Experimentos científicos ou cálculos matemáticos onde a precisão é realmente importante
 - Cálculos monetários
 - Armazenar um fatorial de um número nem tão grande
 - Cálculo exponencial de números primos grandes (muito utilizado em criptografia)
 - Somar o número de estrelas de galáxias
 - ...

BigInteger

- A classe `BigInteger` é utilizada para armazenar valores inteiros que não podem ser armazenados pelos tipos primitivos (como `int` e `long`)
- Como classe, é necessário criar os objetos do tipo `BigInteger` e os valores inteiros são armazenados nesse objetos
- Construtor mais comum para a classe `BigInteger`:
`BigInteger(String val)`
- Como não há operações matemáticas sobre objetos, não é possível, por exemplo, somar dois objetos

BigInteger

- Para fazer operações matemáticas sobre valores armazenados em um objeto do tipo `BigInteger`, é necessário invocar métodos para realizar tais operações
- **Exemplos de métodos da classe `BigInteger`**
 - `add(BigInteger value)`
 - `divide(BigInteger divisor)`
 - `subtract(BigInteger subtrahend)`
 - `multiply(BigInteger multiplicand)`
- **OBSERVAÇÃO:** vale ressaltar que todos os métodos acima retornam um objeto `BigInteger` como resultado da operação

BigInteger

- Vale ressaltar que a classe `BigInteger` possui outros métodos parecidos com os da classe `Math` (`java.lang`) além de outros métodos interessantes
- **Exemplos**
 - `abs()`: retorna o valor absoluto do número
 - `max(BigInteger val)`: retorna o máximo
 - `negate()`: Returns a `BigInteger` whose value is (-this)
 - `valueOf(double valor)` ou `valueOf(int valor)`: retorna um objeto `BigInteger` com valor correspondente ao `long` ou `double` que foi passado como parâmetro
 - `toString()`: retorna a `String` correspondente ao valor armazenado no `BigInteger`

OBSERVAÇÃO: os três primeiros métodos retornam um objeto `BigInteger` como resultado da operação

BigInteger

Exemplo da soma das estrelas de duas galáxias com o uso de BigInteger

```
1 import java.math.BigInteger;
2
3 public class Teste {
4
5     public static void main(String args[]){
6
7         BigInteger numStarsMilkyWay = new BigInteger("8731409320171337804361260816606476");
8         BigInteger numStarsAndromeda = new BigInteger("5379309320171337804361260816606476");
9         BigInteger totalStars = numStarsMilkyWay.add(numStarsAndromeda);
10
11         System.out.println("Numero total de estrelas nas galáxias Via Lactea e Andromeda: " + totalStars.toString());
12
13     }
14
15 }
```

Saida x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run:
Numero total de estrelas nas galáxias Via Lactea e Andromeda: 14110718640342675608722521633212952

BigInteger

Exemplo de um cálculo fatorial sem o uso de BigInteger

```
1 public class Teste {  
2  
3     public static void main(String args[]){  
4  
5         // Calculando o fatorial de x  
6         int x = 40;  
7  
8         int fact = 1;  
9         for (int i = 1; i <= x; i++) {  
10             fact *= i;  
11         }  
12  
13         System.out.println("0 fatorial de " + x + " é " + fact);  
14  
15     }  
16  
17 }
```



BigInteger

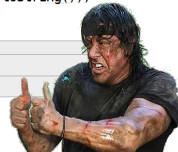
Exemplo de um cálculo fatorial com o uso de BigInteger

```
1 import java.math.BigInteger;
2
3 public class Teste {
4
5     public static void main(String args[]){
6
7         // Calculando o fatorial de x
8         int x = 40;
9
10        BigInteger fact = new BigInteger("1");
11        for (int i = 1; i <= x; i++) {
12            fact = fact.multiply(new BigInteger(i + ""));
13            //fact = fact.multiply(BigInteger.valueOf(i));
14        }
15
16        System.out.println("O fatorial de " + x + " é " + fact.toString());
17    }
18 }
```

Salda x Resultados da Pesquisa x

Console do Depurador x Teste (run) x

run: O fatorial de 40 é 815915283247897734345611269596115894272000000000



BigDecimal

- Para armazenar grandes números com casas decimais (e mantendo a precisão nas casas), pode-se utilizar a classe `BigDecimal`
- Porém, a classe `BigDecimal` oferece mais construtores que a classe `BigInteger`
- Exemplos de métodos construtores comuns da classe `BigDecimal`
 - `BigDecimal(BigInteger val)`
 - `BigDecimal(double val)`
 - `BigDecimal(long val)`
 - `BigDecimal(String val)`

Extra 1: brincando um pouco com GUI

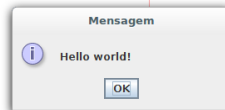
- Nessa aula vamos introduzir o `JOptionPane`
- Essa classe pertence ao pacote `javax.swing`
- Esse pacote contém muitas classes que ajudam a **criar interfaces gráficas com o usuário** (ou *Graphical User Interface – GUIs*)

Exibindo Caixas de Diálogo

- Inicialmente vamos utilizar três métodos da classe `JOptionPane`: **`showMessageDialog(...)`**, **`showInputDialog(...)`** e **`showConfirmDialog(...)`**
- O método `showMessageDialog`, em sua versão mais simples, requer dois parâmetros:
 - O primeiro refere-se onde posicionar a caixa de diálogo (pode-se passar o componente no qual a caixa de diálogo irá aparecer no centro do mesmo, ou `null`, que fará com que o diálogo seja exibido no centro da tela)
 - O segundo é a *string* a ser exibida na caixa de diálogo

Exibindo Caixas de Diálogo

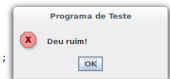
```
1  import javax.swing.JOptionPane;
2
3  public class Programa {
4
5      public static void main(String[] args){
6
7          JOptionPane.showMessageDialog(null, "Hello world!");
8      }
9
10 }
```



Exibindo Caixas de Diálogo

- Em uma versão um pouco mais “sofisticada”, pode-se informar um **título** para a caixa de diálogo e um tipo para a caixa de diálogo (plana, alerta, informação e erro)
- Para facilitar a especificação do tipo da caixa de diálogo, a classe `JOptionPane` contém **variáveis estáticas que armazenam os valores dos tipos**

```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         JOptionPane.showMessageDialog(null, "Deu ruim!", "Programa de Teste", JOptionPane.ERROR_MESSAGE);
8
9     }
10
11 }
```

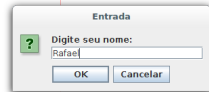


Retornando Textos Inseridos em uma Caixa de Diálogo

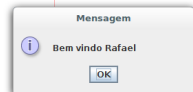
- O método *showInputDialog* da classe `JOptionPane` pode ser utilizado para
 - 1 Exibir uma caixa de diálogo
 - 2 Permitir o usuário inserir um texto na caixa de diálogo
 - 3 Retornar o texto digitado pelo usuário, possibilitando assim armazená-lo em uma variável
- O tipo de retorno do método `showInputDialog` é `String`
- Em sua forma mais simples, o método *showInputDialog* necessita de um único parâmetro → texto a ser exibido na caixa de diálogo

Retornando Textos Inseridos em uma Caixa de Diálogo

```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         String nome = JOptionPane.showInputDialog("Digite seu nome: ");
8
9         JOptionPane.showMessageDialog(null, "Bem vindo " + nome);
10
11     }
12
13 }
```



```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         String nome = JOptionPane.showInputDialog("Digite seu nome: ");
8
9         JOptionPane.showMessageDialog(null, "Bem vindo " + nome);
10
11     }
12
13 }
```

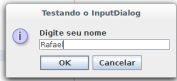


- **OBSERVAÇÃO:** vale ressaltar que se o usuário clicar na opção Cancelar o valor retornado é `null`

Retornando Textos Inseridos em uma Caixa de Diálogo

- Em uma versão um pouco mais sofisticada, pode-se informar o título da caixa de diálogo e o tipo da caixa de diálogo

```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         String nome = JOptionPane.showInputDialog(null, "Digite seu nome", "Testando o InputDialog", JOptionPane.INFORMATION_MESSAGE);
8
9         JOptionPane.showMessageDialog(null, "Bem vindo " + nome);
10
11     }
12 }
13
14
15
```

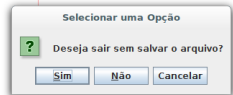


Exibindo caixas de Confirmação

- O método `showConfirmDialog` da classe `JOptionPane` pode ser utilizado para pedir que o usuário confirme, negue ou ainda cancele a execução de uma determinada ação
- O método `showConfirmDialog` irá retornar um valor inteiro correspondente à ação que o usuário tomou (clicou)
- O significado dos valores inteiros retornados pelo método `showConfirmDialog` podem ser obtidos na documentação da classe ou ainda utilizando campos estáticos da referida classe

Exibindo caixas de Confirmação

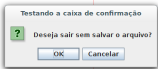
```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7
8         int opcao = JOptionPane.showConfirmDialog(null, "Deseja sair sem salvar o arquivo?");
9
10        switch(opcao){
11            case JOptionPane.CANCEL_OPTION:
12                System.out.println("O usuário cancelou");
13                break;
14            case JOptionPane.YES_OPTION:
15                System.out.println("O usuário quer sair sem salvar");
16                break;
17            case JOptionPane.NO_OPTION:
18                System.out.println("O usuário não quer sair sem salvar");
19                break;
20        }
21    }
22 }
23
24
25 }
```



Exibindo caixas de Confirmação

- Pode-se utilizar versões mais sofisticadas do método `showConfirmDialog` para, por exemplo, definir o título da caixa de diálogo de confirmação e o tipo da caixa de confirmação

```
1 import javax.swing.JOptionPane;
2
3 public class Programa {
4
5     public static void main(String[] args){
6
7         int opcao = JOptionPane.showConfirmDialog(null, "Deseja sair sem salvar o arquivo?", "Testando a caixa de confirmação", JOptionPane.OK_CANCEL_OPTION);
8
9         switch(opcao){
10             case JOptionPane.CANCEL_OPTION:
11                 System.out.println("O usuário cancelou");
12                 break;
13             case JOptionPane.OK_OPTION:
14                 System.out.println("O usuário deu OK");
15                 break;
16         }
17     }
18 }
19
20 }
```



1º Quiz

Enzo criou uma classe Player. O objetivo de enzo é toda vez que uma instância da classe Player seja criado, seja armazenada a quantidade total de instancias criadas. Para isso, enzo criou a seguinte classe:

```
public class Playe{  
  
    //campos de classe  
    private int total = 0;  
  
    public Jogador(...){  
        total++;  
    }  
  
    //Restante da classe  
}
```

1º Quiz

O programa de Enzo não está funcionando corretamente já que:

- a) O campo total deveria ser público.
- b) Não tem como fazer o que Enzo quer.
- c) O campo total deveria ser estático.
- d) Nenhuma das anteriores.

1º Quiz

O programa de Enzo não está funcionando corretamente já que:

- a) O campo total deveria ser público.
- b) Não tem como fazer o que Enzo quer.
- c) O campo total deveria ser estático.**
- d) Nenhuma das anteriores.

Projeto Banco

- Vamos complementar o exercício realizado na última aula (Aula 4)
- O complemento se dará da seguinte forma:
 - Entradas e saídas por meio dos métodos estáticos da classe `JOptionPane`
 - Adicionar um campo estático que será responsável por habilitar/desabilitar operações bancárias
 - Modificar os métodos da classe `Conta` de forma que qualquer método que realizasse alguma impressão na tela agora irá retornar a mesma informação, porém, na forma de `String`

Projeto Banco

- O complemento se dará da seguinte forma:
 - Adicionar um item no menu (Área do Gerente) para permitir habilitar/desabilitar as operações bancárias
 - Só realizar operações bancárias se houver permissão para tal
 - Após cada interação com o sistema (ex: depósito ou saque em uma conta), perguntar se o usuário deseja realizar outra operação (apenas opções SIM ou NÃO)
 - Se SIM, exibir o menu novamente
 - Se NÃO, encerrar o sistema

Material Complementar

- Classe BigInteger

<https://docs.oracle.com/javase/7/docs/api/java/math/BigInteger.html>

- Classe BigDecimal

<https://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html>

- Classe Math

<https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html>

- Classes Wrappers em Java

<http://www.linhadecodigo.com.br/artigo/3667/classes-wrappers-em-java.aspx>

Material Complementar

- Curso de Java #06 - Tipos Primitivos e Manipulação de Dados

https:

[//www.youtube.com/watch?v=JEAQeT7YGs4&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=11](https://www.youtube.com/watch?v=JEAQeT7YGs4&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=11)

- Curso de Java #07 - Operadores Aritméticos e Classe Math

- https:

[//www.youtube.com/watch?v=W9V5wt00ZHs&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=13](https://www.youtube.com/watch?v=W9V5wt00ZHs&list=PLHz_AreHm4dkI2ZdjTwZA4mPMxWTfNSpR&index=13)

- Unicode

<https://pt.wikipedia.org/wiki/Unicode>

- Fundamentos da Notação de Ponto Flutuante

<http://producao.virtual.ufpb.br/books/camyle/introducao-a-computacao-livro/livro/livro.chunked/ch03s07.html>

Imagem do Dia

PROGRAMANDO NA UNIVERSIDADE



PROGRAMANDO NO TRABALHO



Programação Orientada a Objetos

<http://lives.ufms.br/moodle/>

Rafael Geraldeli Rossi
rafael.g.rossi@ufms.br

Slides baseados em [Deitel and Deitel, 2010]

Referências Bibliográficas I



Deitel, P. and Deitel, H. (2010).

Java: How to Program.

How to program series. Pearson Prentice Hall, 8th edition.