

JAVA



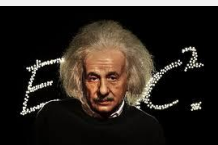
Saiba mais sobre a Linguagem Java

—
Primeiro....

Existem duas formas principais para se interagir com uma aplicação.

1ª -> acontece passando-se argumentos no momento da execução do programa.

2ª -> se dá por meio da leitura de valores em tempo real



Saiba mais sobre a Linguagem Java

6

```
public class ArgsLinhaDeComandoBasico {  
    public static void main(String[] args) {  
        System.out.printf("qtd de argumentos = %d%n",  
args.length);  
        for (int i = 0; i < args.length; i++) {  
            System.out.printf("\targs[%d] = %s%n", i, args[i]);  
        }  
    }  
}
```

Assinatura do ponto de
entrada da aplicação

↑ ↑
Especificadores de formato

String[] - Vetor de strings
que permite a passagem
de quantos argumentos
quisermos

args - parâmetro que
recebe valores no
momento da execução
do código.



Saiba mais sobre a Linguagem Java

7

Sequências de scape

Sequência de Escape	Descrição	Exemplo de utilização
<code>\n</code>	Inserir nova linha.	<code>System.out.print("Introdução\n\nProgramação\ncom\nJava");</code>
<code>\t</code>	Inserir tabulação na horizontal.	<code>System.out.print("Col A\tCol B\tCol C\tCol D");</code>
<code>\\</code>	Inserir barra invertida.	<code>System.out.print("C:\\Windows\\system32");</code>
<code>\"</code>	Inserir aspa dupla.	<code>System.out.print("Nome do livro \"Dom Quixote\" de Miguel de Cervantes");</code>
<code>\r</code>	Realiza retorno do carro.	<code>System.out.print("Texto Não Mostrado \rEsse Texto Aparece\n");</code>



Saiba mais sobre a Linguagem Java

8

Especificadores de formato

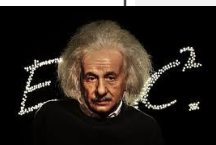
Especificador	Descrição	Exemplo de Utilização	Saída
%d	Valor inteiro em decimal com sinal (pode ser usado para <i>byte</i> , <i>short</i> , <i>int</i> e <i>long</i>).	<code>System.out.println("%%d", 127);</code>	127
%o	Valor inteiro em octal com sinal.	<code>System.out.println("%%o", 127);</code>	177



Saiba mais sobre a Linguagem Java

Especificador	Descrição	Exemplo de Utilização	Saída
%x	Valor inteiro em hexadecimal com sinal (minúsculo).	<code>System.out.println("0x", 127);</code>	7f
%X	Valor inteiro em hexadecimal com sinal (maiúsculo).	<code>System.out.println("0X", 127);</code>	7F
%f	Valor real (<i>float</i> ou <i>double</i>).	<code>System.out.println("f", 3.141592);</code>	3,141592
%e	Valor real (notação exponencial) (minúsculo).	<code>System.out.println("e", 3.14);</code>	3,14e+00
%E	Valor real (notação exponencial) (maiúsculo).	<code>System.out.println("E", 3.14);</code>	3,14E+00
%X	Valor inteiro em hexadecimal com sinal (maiúsculo).	<code>System.out.println("X", 10);</code>	7F
%b	Valor lógico (<i>boolean</i>) (minúsculo).	<code>System.out.println("b", 3 > 2);</code>	true
%B	Valor lógico (<i>boolean</i>) (maiúsculo).	<code>System.out.println("B", 2 > 3);</code>	FALSE

Especificadores de formato



Saiba mais sobre a Linguagem Java

10

Exemplo - especificador de formato

```
1  /**
2   * @Leonardo Rocha
3   */
4
5  public class DefinePeso{
6
7  public static void main(String[] args) {
8
9      String nome = "Leonardo";
10     int idade = 30;
11     double peso = 95.5;
12     System.out.printf("%s, %d anos, pesa %.2f kg!", nome, idade, peso);
13 }
14 }
```



Saiba mais sobre a Linguagem Java

11

Estrutura de pasta - visão geral

Diretório	Descrição do Conteúdo
<i>src/</i>	Código-fonte da aplicação.
<i>test/</i>	Código de teste unitário (não utilizado neste livro).
<i>lib/</i>	Dependências do projeto.
<i>dist/</i>	Arquivos de distribuição como .jar e suas dependências.
<i>build/</i>	Arquivos gerados pelo processo de compilação.



Saiba mais sobre a Linguagem Java

12

Leitura de dados em tempo real

A leitura de dados em tempo real pode ser feita via classe `Scanner`, que fornece métodos de leitura com sintaxes diferentes.

```
import java.util.Scanner;

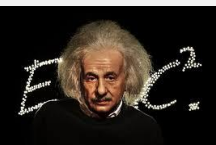
public class ExemploLeituraDados {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        System.out.print("Digite um valor inteiro (int): ");
        int entradaInt = scan.nextInt();
        System.out.print("Digite um valor real (double): ");
        double entradaDouble = scan.nextDouble();
        System.out.print("Digite um valor lógico (boolean): ");
        boolean entradaBoolean = scan.nextBoolean();
        System.out.print("Digite uma string (uma palavra): ");
        String entradaPalavra = scan.next();
    }
}
```



Saiba mais sobre a Linguagem Java

Construir um código legível e bem documentado é extremamente importante, pois ele necessitará de atualizações e poderá ser lido por outros programadores.

A forma mais simples de documentar um código se dá por meio de comentários



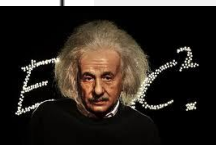
Saiba mais sobre a Linguagem Java

26

Tipos de comentários

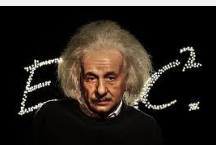
Uma única linha - utiliza-se o comando `//` para inserção do comentário;

Em blocos - utiliza-se o comando `/* conteúdo */` para inserção de comentários.



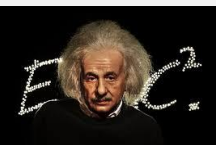
Argumentos variáveis – Varargs

—
A linguagem Java nos permite chamar um método diretamente passando n valores e os parâmetros enviados são automaticamente adicionados em um Array de mesmo tipo.



Argumentos variáveis – Varargs

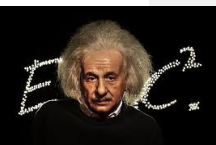
Em Java, Varargs (abreviação de "variable arguments") é uma característica que permite a um método aceitar um número variável de argumentos do mesmo tipo. Isso significa que você pode chamar um método com diferentes quantidades de argumentos sem precisar definir múltiplas versões do método com diferentes números de parâmetros.



Argumentos variáveis – Varargs

Exemplo para entender ...

```
Varargs.java
1 package explicacao.varargs;
2
3 public class Varargs {
4
5     public static void main(String[] args) {
6
7         int resultado1 = somar(1, 2); // Resultado: 3
8         int resultado2 = somar(1, 2, 3); // Resultado: 6
9         int resultado3 = somar(1, 2, 3, 4, 5); // Resultado: 15
10    }
11
12    public static int somar(int... numeros) {
13        int soma = 0;
14        for (int numero : numeros) {
15            soma += numero;
16        }
17        System.out.println(soma);
18        return soma;
19    }
20 }
21
--
Problems Javadoc Declaration Console
<terminated> Varargs [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 19:52:50 – 19:52:51)
3
6
15
```



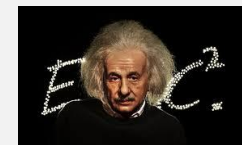
Argumentos variáveis – Varargs

```
public static int somar(int... numeros) {  
    int soma = 0;  
    for (int numero : numeros) {  
        soma += numero;  
    }  
    System.out.println(soma);  
    return soma;  
}
```

É aqui que os Varargs entram em jogo.

Em vez de definir várias versões da função com diferentes quantidades de parâmetros, você pode usar Varargs para lidar com qualquer quantidade de números.

Veja como ficaria:



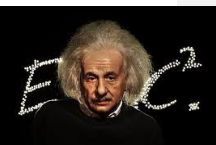
Argumentos variáveis – Varargs

```
Varargs.java
1 package explicacao.while_varargs;
2
3 public class Varargs {
4
5     public static void main(String[] args) {
6
7         int resultado1 = somar(1, 2); // Resultado: 3
8         int resultado2 = somar(1, 2, 3); // Resultado: 6
9         int resultado3 = somar(1, 2, 3, 4, 5); // Resultado: 15
10
11     }
12
13     public static int somar(int... numeros) {
14         int soma = 0;
15         for (int numero : numeros) {
16             soma += numero;
17         }
18         System.out.println(soma);
19         return soma;
20     }
21 }
```

Problems @ Javadoc Declaration Console

<terminated> Varargs [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 19:52:50 – 19:52:51)

3
6
15



Java – Orientação Objeto

—
Um paradigma de programação que aproxima a manipulação das estruturas de um programa ao manuseio das coisas no mundo real.

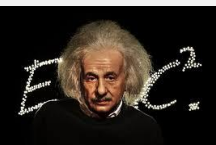
Pode ser considerado uma filosofia que guia todo o desenvolvimento de código.

Considerada a principal diferença para OO, a programação procedural é uma forma de desenvolvimento de código que se caracteriza por utilizar, principalmente, funções e procedimentos como núcleo de organização estrutural.



Paradigmas de programação

—
Paradigma de programação, a grosso modo, é a forma de escrita (estrutura) de um determinado código, desde que ele a aceite. Existem linguagens que aceitam mais de um paradigma de programação. Um deles, a programação orientada a objetos apresenta como uma das principais características: a reutilização de código, isso é possível, por exemplo, pela construção de classes, outro exemplo é o paradigma estruturado tem como principal característica seguir sequência.

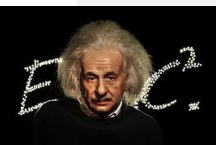


Paradigmas de programação

POO

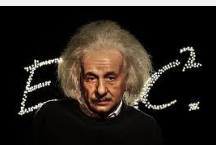
A programação orientada a objetos tem como base a:

- 1 - Abstração
- 2 - Encapsulamento
- 3 - Herança
- 4 - Polimorfismo



Java - abstração

Imagine que você está construindo um carro. Para a maioria das pessoas, um carro é uma coisa bem simples de entender: tem rodas, um motor, um volante, etc. Mas se você começar a pensar sobre como um carro é realmente construído e como ele funciona, você vai perceber que há muitos detalhes complicados e coisas acontecendo sob o capô.



Java - abstração

A abstração em Java é como uma simplificação desse processo. Em vez de lidar com todos os detalhes internos de como as coisas funcionam, você pode pensar em termos mais simples e abstratos.

Por exemplo, vamos pensar em um carro em termos de uma classe em Java. Uma classe em Java é como um modelo ou um plano para criar objetos.

Vamos criar uma classe Carro:



Java - abstração

```
Carro.java
1 package explicacao.abstracao;
2
3 public class Carro {
4     String marca;
5     String modelo;
6     int ano;
7
8     public Carro(String marca, String modelo, int ano) {
9         this.marca = marca;
10        this.modelo = modelo;
11        this.ano = ano;
12    }
13
14    public void ligar() {
15        System.out.println("O carro está ligado!");
16    }
17
18    public void acelerar() {
19        System.out.println("O carro está acelerando!");
20    }
21
22 }
23
```

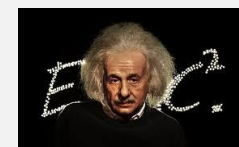


Java - abstração

Aqui, Carro é uma abstração de um carro real. Ele tem atributos como marca, modelo e ano, e métodos como ligar() e acelerar().

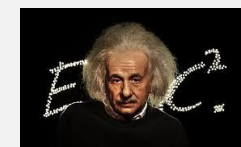
Você pode pensar nessa classe como uma versão simplificada de um carro real.

A abstração permite que você se concentre nos aspectos importantes de um objeto ou sistema sem se preocupar com todos os detalhes internos.



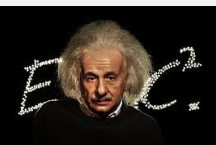
Java - abstração

—
Por exemplo, quando você usa um carro, não precisa entender como o motor funciona internamente; você só precisa saber como ligar e dirigir o carro. Da mesma forma, ao usar uma classe em Java, você só precisa saber quais métodos ela oferece e como usá-los, sem precisar entender todos os detalhes de como a classe foi implementada internamente.



Java - abstração

Em resumo, a abstração em Java é sobre simplificar as coisas, concentrando-se nos aspectos importantes e ignorando os detalhes internos complicados. Isso torna mais fácil entender e trabalhar com objetos e sistemas complexos.

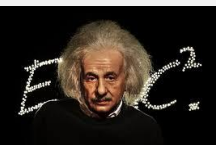


Java - encapsulamento

Encapsulamento em Java é um conceito fundamental da programação orientada a objetos que permite proteger os dados de uma classe, controlando o acesso a eles por meio de métodos públicos.

Imagine que você tem uma caixa, e dentro dessa caixa você guarda algumas coisas importantes.

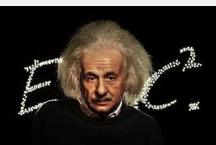
Você não quer que qualquer pessoa possa acessar diretamente o conteúdo da caixa, então você coloca uma tampa nela.



Java - encapsulamento

—
Agora, para acessar o conteúdo da caixa, as pessoas têm que usar a tampa - elas não podem simplesmente pegar as coisas de dentro da caixa.

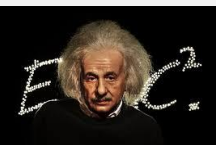
Em Java, uma classe é como essa caixa, e os dados que ela armazena são como as coisas dentro dela. O encapsulamento é como a tampa da caixa - ele protege os dados da classe, controlando o acesso a eles.



Java - encapsulamento

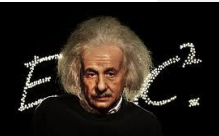
Você faz isso definindo os atributos da classe como privados e fornecendo métodos públicos para acessá-los e modificá-los, se necessário. Esses métodos públicos são conhecidos como métodos getters e setters.

Um método getter é usado para recuperar o valor de um atributo privado, enquanto um método setter é usado para definir ou modificar o valor de um atributo privado. Dessa forma, você tem controle sobre como os dados da classe são acessados e modificados.



Java - encapsulamento

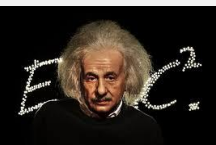
```
Pessoa.java
1 package explicacao.encapsulamento;
2
3 public class Pessoa {
4     private String nome;
5     private int idade;
6
7     // Método getter para o atributo 'nome'
8     public String getNome() {
9         return nome;
10    }
11
12    // Método setter para o atributo 'nome'
13    public void setNome(String novoNome) {
14        this.nome = novoNome;
15    }
16
17    // Método getter para o atributo 'idade'
18    public int getIdade() {
19        return idade;
20    }
21
22    // Método setter para o atributo 'idade'
23    public void setIdade(int novaIdade) {
24        if (novaIdade >= 0) { // Verifica se a idade é positiva
25            this.idade = novaIdade;
26        } else {
27            System.out.println("Idade inválida");
28        }
29    }
30 }
31
```



Java - enccapsulamento

A classe **Pessoa** tem dois atributos privados: nome e idade. Os métodos getters (getNome() e getIdade()) permitem acessar os valores desses atributos, enquanto os métodos setters (setNome() e setIdade()) permitem modificar esses valores, seguindo as regras de validação necessárias (no caso da idade, garantindo que seja um número positivo).

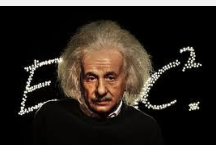
Dessa forma, o encapsulamento ajuda a proteger os dados da classe e a manter a integridade do objeto.



Java - herança

Imagine que você está construindo uma cidade de robôs, e cada robô tem suas próprias características e habilidades especiais. Por exemplo, alguns robôs podem voar, outros podem nadar, alguns podem escavar e assim por diante.

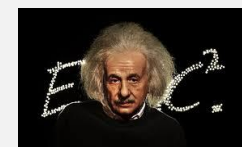
Agora, em vez de construir cada tipo de robô do zero, você pode usar a herança.



Java - herança

Em vez de criar uma nova classe para cada tipo de robô, você pode criar uma classe base, chamada por exemplo de Robo, que contém características e habilidades comuns a todos os robôs.

Em seguida, você pode criar subclasses específicas, como RoboVoador, RoboNadador, RoboEscavador, etc., que herdam todas as características e habilidades da classe Robo, mas também podem ter suas próprias características e habilidades exclusivas.



Java - herança

Em Java, a herança é implementada usando a palavra-chave **extends**

```
*Robo.java
1 package explicacao.heranca;
2
3 //Classe base
4 class Robo {
5     void mover() {
6         System.out.println("O robô está se movendo.");
7     }
8
9
10 // main apenas para executar
11 public static void main(String[] args) {
12
13     RoboVoador robzinho = new RoboVoador();
14     robzinho.mover();
15     robzinho.voar();
16
17 }
18 }
19

RoboVoador.java
1 package explicacao.heranca;
2
3 //Subclasse que herda de Robo
4 class RoboVoador extends Robo {
5     void voar() {
6         System.out.println("O robô está voando.");
7     }
8 }
```

Problems @ Javadoc Declaration Console

<terminated> Robo [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 16:35:58 – 16:35:58)

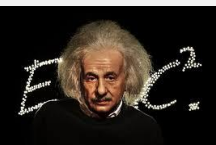
O robô está se movendo.
O robô está voando.



Java - herança

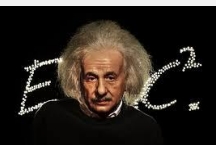
Neste exemplo, a classe RoboVoador é uma subclasse de Robo, o que significa que ela herda todos os métodos e atributos de Robo. Assim, um objeto da classe RoboVoador também pode chamar o método mover(), assim como um objeto da classe Robo, mas também tem seu próprio método exclusivo voar().

Essa abordagem torna o código mais organizado e reutilizável.



Java - herança

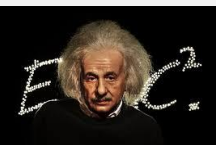
Em resumo, a herança em Java é um mecanismo que permite que uma classe herde características e comportamentos de outra classe, promovendo a reutilização de código e a organização do design do programa.



Java - polimorfismo

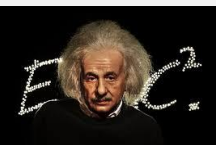
Polimorfismo é um conceito importante na programação orientada a objetos que nos permite tratar objetos de diferentes classes de maneira uniforme.

Imagine que você tem vários tipos de animais em uma fazenda: vacas, galinhas, cavalos e assim por diante. Cada animal faz um som diferente. Por exemplo, uma vaca faz "muu", uma galinha faz "cocoricó" e um cavalo faz "hihihi".



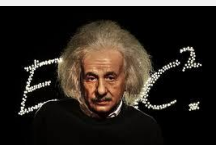
Java - polimorfismo

—
Agora, se você quisesse escrever um programa para fazer esses animais "falarm", como você faria? Você poderia criar um método falar() em cada classe de animal, mas isso pode ser trabalhoso e complicado de gerenciar. É aqui que o polimorfismo entra em cena.



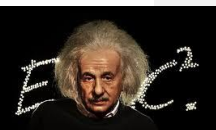
Java - polimorfismo

Em Java, o polimorfismo nos permite tratar objetos de diferentes classes da mesma maneira, usando uma referência comum. Por exemplo, podemos ter uma classe base `Animal` com um método `falar()`, e então podemos criar subclasses como `Vaca`, `Galinha`, `Cavalo`, cada uma sobrescrevendo o método `falar()` com seu próprio comportamento.



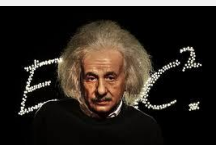
Java - polimorfismo

```
*Animal.java  ⌵
1  package explicacao.polimorfismo;
2
3  //Classe base
4  class Animal {
5      public void falar() {
6          System.out.println("Este animal emite algum som.");
7      }
8  }
9
10 //Subclasse Vaca
11 class Vaca extends Animal {
12     @Override
13     public void falar() {
14         System.out.println("Muu");
15     }
16 }
17
18 //Subclasse Galinha
19 class Galinha extends Animal {
20     @Override
21     public void falar() {
22         System.out.println("Cocoricó");
23     }
24 }
25
26 //Subclasse Cavalo
27 class Cavalo extends Animal {
28     @Override
29     public void falar() {
30         System.out.println("Hihihi");
31     }
32 }
33
```



Java - polimorfismo

Agora, podemos criar objetos de cada uma dessas subclasses e tratá-los como objetos da classe Animal. Quando chamamos o método falar() em cada objeto, o método apropriado é executado, dependendo do tipo real do objeto.



Java - polimorfismo

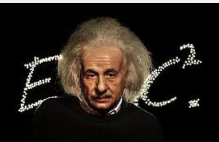
```
Animal.java
1 package explicacao.polimorfismo;
2
3 //Classe base
4 class Animal {
5     public void falar() {
6         System.out.println("Este animal emite algum som.");
7     }
8 }
9
10 //Subclasse Vaca
11 class Vaca extends Animal {
12     @Override
13     public void falar() {
14         System.out.println("Muu");
15     }
16 }
17
18 //Subclasse Galinha
19 class Galinha extends Animal {
20     @Override
21     public void falar() {
22         System.out.println("Cocoricó");
23     }
24 }
25
26 //Subclasse Cavalo
27 class Cavalo extends Animal {
28     @Override
29     public void falar() {
30         System.out.println("Hihihi");
31     }
32 }
33
```

```
TestePolimorfismo.java
1 package explicacao.polimorfismo;
2
3 public class TestePolimorfismo {
4     public static void main(String[] args) {
5         Animal animal1 = new Vaca();
6         Animal animal2 = new Galinha();
7         Animal animal3 = new Cavalo();
8
9         animal1.falar(); // Saída: Muu
10        animal2.falar(); // Saída: Cocoricó
11        animal3.falar(); // Saída: Hihihi
12    }
13 }
14
```

Problems @ Javadoc Declaration Console

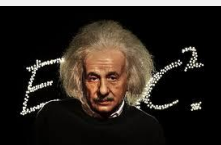
<terminated> TestePolimorfismo [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 16:40:44 - 16:40:44)

Muu
Cocoricó
Hihihi



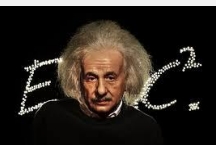
Java - polimorfismo

Essa capacidade de tratar objetos de diferentes classes de maneira uniforme é o polimorfismo em ação. Ele nos permite escrever código mais flexível, reutilizável e fácil de entender, promovendo uma melhor organização e manutenção do código.



Java – classes e objetos

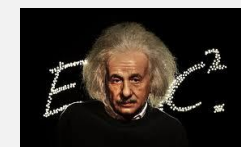
—
Imagine que você está construindo uma casa. Você pode pensar em uma casa como um modelo ou plano para construir muitas casas semelhantes. O modelo da casa descreve suas características básicas, como o número de quartos, banheiros, a cor das paredes, e assim por diante.



Java – classes e objetos

Em Java, uma classe é como esse modelo ou plano para criar objetos. Uma classe é uma estrutura de programação que define o comportamento e as características de um objeto. Ela define quais informações um objeto pode armazenar (atributos) e quais ações ele pode realizar (métodos).

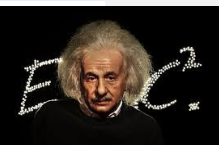
Por exemplo, vamos criar uma classe Carro em Java:



Java – classes e objetos

*Carro.java

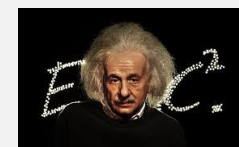
```
1 package explicacao.classes_e_objetos;
2
3 public class Carro {
4     // Atributos
5     String marca;
6     String modelo;
7     int ano;
8
9     // Método
10    void ligar() {
11        System.out.println("O carro está ligado!");
12    }
13 }
14
```



Java – classes e objetos

A classe Carro define três atributos: marca, modelo e ano, que representam as características de um carro. Além disso, a classe possui um método chamado ligar(), que representa uma ação que um carro pode realizar.

Agora, para usar essa classe e criar um objeto (ou instância) dela, você pode fazer assim:

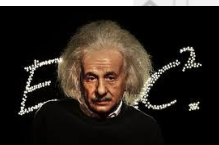


Java – classes e objetos

```
Carro.java
1 package explicacao.classes_e_objetos;
2
3 public class Carro {
4     // Atributos
5     String marca;
6     String modelo;
7     int ano;
8
9     // Método
10    void ligar() {
11        System.out.println("O carro está ligado!");
12    }
13 }
14

TesteCarro.java
1 package explicacao.classes_e_objetos;
2
3 public class TesteCarro {
4     public static void main(String[] args) {
5         // Criando um objeto da classe Carro
6         Carro meuCarro = new Carro();
7
8         // Atribuindo valores aos atributos
9         meuCarro.marca = "Toyota";
10        meuCarro.modelo = "Corolla";
11        meuCarro.ano = 2022;
12
13        // Chamando um método do objeto
14        meuCarro.ligar(); // Saída: O carro está ligado!
15    }
16 }
17

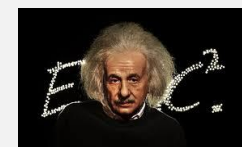
Problems @ Javadoc Declaration Console
<terminated> TesteCarro [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 16:44:45 – 16:44:47)
O carro está ligado!
```



Java – classes e objetos

Neste exemplo, criamos um objeto meuCarro da classe Carro usando a palavra-chave new. Em seguida, atribuímos valores aos atributos marca, modelo e ano do objeto. Por fim, chamamos o método ligar() do objeto.

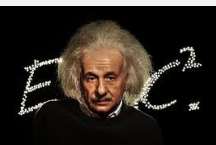
Em resumo, uma classe em Java é como um modelo ou plano para criar objetos. Os objetos são instâncias específicas de uma classe, que possuem seus próprios valores de atributos e podem realizar as ações definidas pelos métodos da classe. Classes e objetos são fundamentais na programação orientada a objetos e ajudam a organizar e estruturar o código de maneira eficiente.



Java – atributos e métodos estáticos

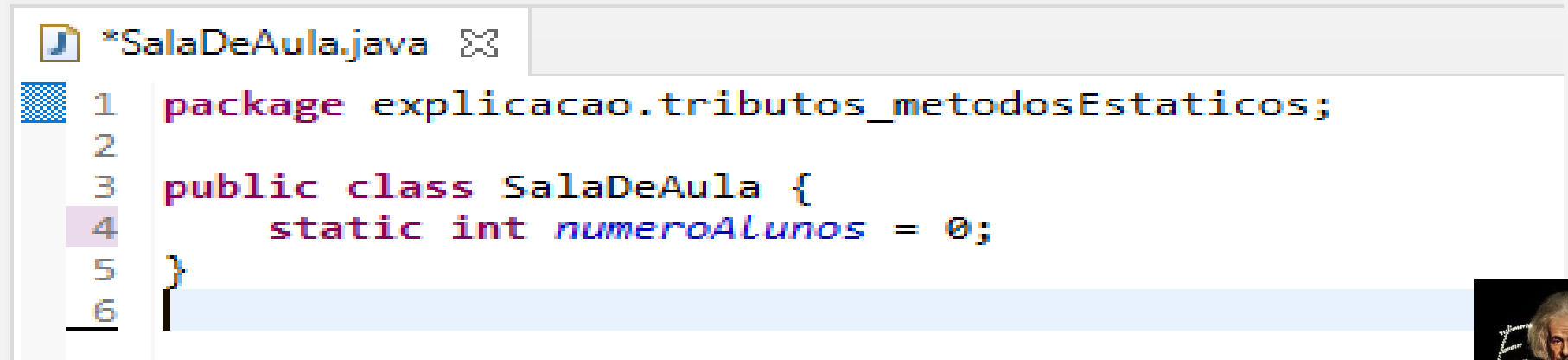
—
Você está em uma sala de aula e quer contar quantos alunos já passaram por ali.

Você pode criar uma variável para armazenar esse número, mas essa informação é comum a todos os alunos, então faz sentido que seja uma característica estática da sala de aula, e não de cada aluno individualmente.

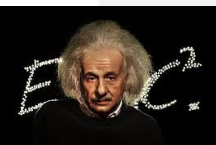


Java – atributos e métodos estáticos

—
Em Java, um atributo estático é uma variável que pertence à classe como um todo, em vez de pertencer a instâncias individuais (objetos) da classe. Isso significa que todos os objetos da classe compartilham o mesmo valor do atributo estático. Você declara um atributo estático usando a palavra-chave `static`.



```
*SalaDeAula.java ✕  
1 package explicacao.tributos_metodosEstaticos;  
2  
3 public class SalaDeAula {  
4     static int numeroAlunos = 0;  
5 }  
6
```

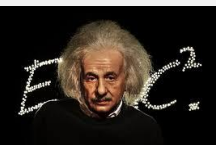


Java – atributos e métodos estáticos

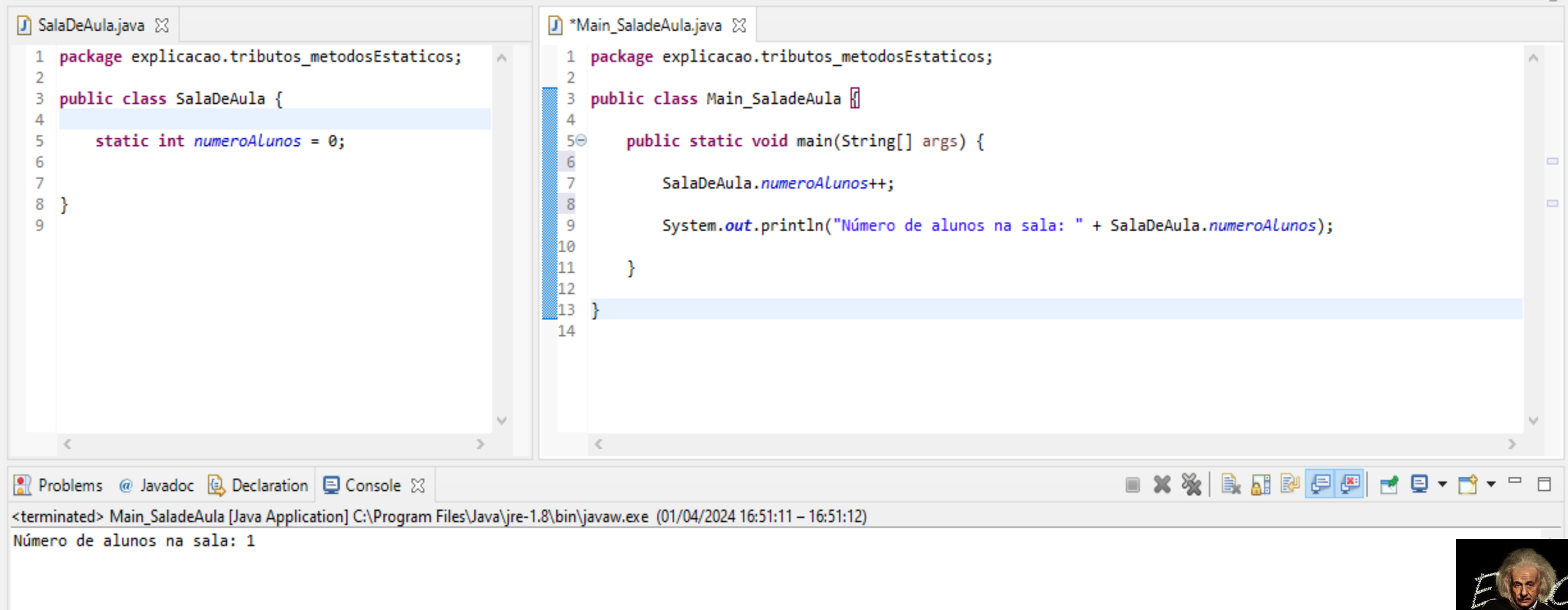
—
Neste exemplo, **numeroAlunos** é um atributo estático da classe **SalaDeAula**.

Isso significa que você pode acessá-lo sem criar um objeto da classe **SalaDeAula**.

Por exemplo:



Java – atributos e métodos estáticos



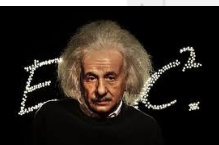
```
1 package explicacao.tributos_metodosEstaticos;
2
3 public class SalaDeAula {
4
5     static int numeroAlunos = 0;
6
7 }
8
9
```

```
1 package explicacao.tributos_metodosEstaticos;
2
3 public class Main_SaladeAula {
4
5     public static void main(String[] args) {
6
7         SalaDeAula.numeroAlunos++;
8
9         System.out.println("Número de alunos na sala: " + SalaDeAula.numeroAlunos);
10
11     }
12
13 }
14
```

Problems @ Javadoc Declaration Console

<terminated> Main_SaladeAula [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 16:51:11 – 16:51:12)

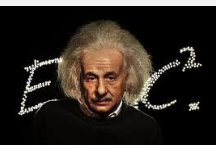
Número de alunos na sala: 1



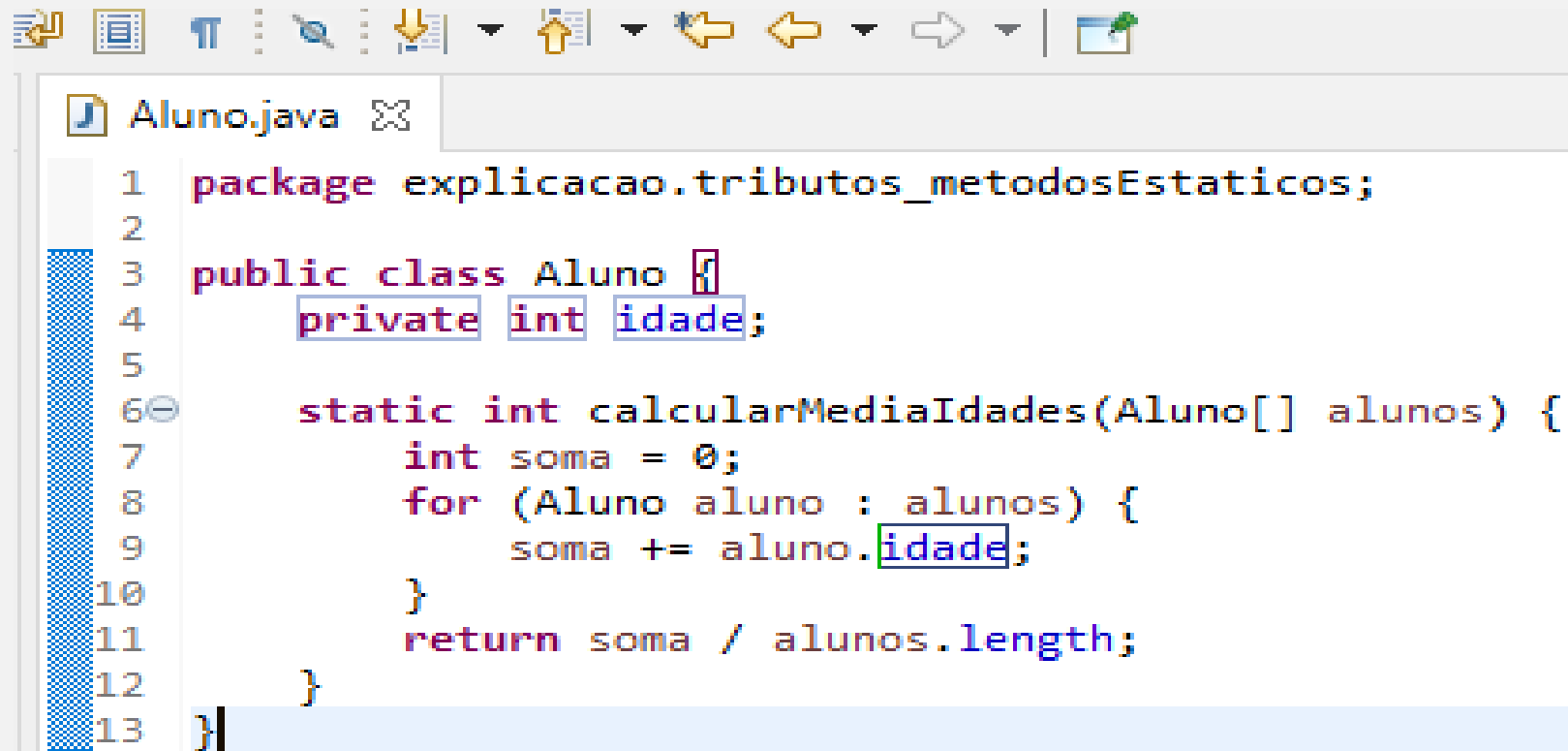
Java – atributos e métodos estáticos

Métodos Estáticos -> imagine que você quer calcular a média das idades de todos os alunos da escola. Este é um cálculo que não depende de um aluno específico, então faz sentido que seja um método estático.

Em Java, um método estático é um método que pertence à classe como um todo, em vez de pertencer a instâncias individuais (objetos) da classe. Isso significa que você pode chamar o método sem criar um objeto da classe. Você declara um método estático usando a palavra-chave `static`.

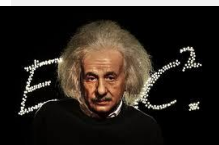


Java – atributos e métodos estáticos



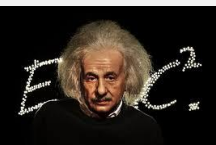
The screenshot shows an IDE window titled 'Aluno.java'. The code defines a package, a class, a private attribute, and a static method. Line numbers 1 through 13 are visible on the left. The code is as follows:

```
1 package explicacao.tributos_metodosEstaticos;
2
3 public class Aluno {
4     private int idade;
5
6     static int calcularMediaIdades(Aluno[] alunos) {
7         int soma = 0;
8         for (Aluno aluno : alunos) {
9             soma += aluno.idade;
10        }
11        return soma / alunos.length;
12    }
13 }
```



Java – atributos e métodos estáticos

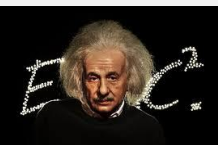
Em resumo, atributos e métodos estáticos em Java pertencem à classe como um todo, em vez de pertencerem a instâncias individuais (objetos) da classe. Isso permite compartilhar dados e funcionalidades entre todos os objetos da classe sem a necessidade de criar uma instância da classe.



Java - Métodos construtores

—
declaração de um construtor é obrigatória e deve sempre possuir o mesmo nome da classe. Não existe nenhum tipo de retorno, nem mesmo a palavra void pode ser especificada. É invocado no momento da criação do objeto através do operador new e é responsável por criá-lo em memória, ou seja, instanciar a classe que foi definida. O retorno desse operador, é uma referência para objeto recém-criado. É possível criar mais de um construtor dentro da mesma classe.

Em java, apenas as Interfaces não possuem construtores.

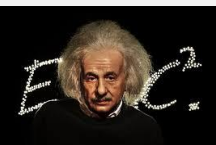


Java - Métodos construtores

—
você está construindo uma casa e, quando terminar, gostaria de decorá-la com móveis.

Você pode definir alguns móveis básicos que todas as casas devem ter, como uma cama, uma mesa e uma cadeira.

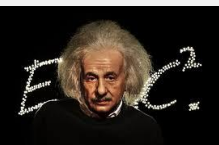
Em vez de colocar esses móveis um por um toda vez que construir uma casa, você pode criar um método que automaticamente coloca esses móveis na casa assim que ela for construída.



Java - Métodos construtores

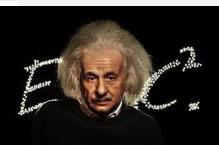
Em Java, um método construtor é um método especial que é chamado automaticamente quando um objeto de uma classe é criado.

Ele é usado para inicializar os atributos do objeto e realizar outras tarefas de inicialização necessárias.



Java - Métodos construtores

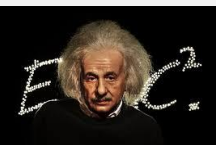
```
*Casa.java ✕  
1 package explicacao.construtores;  
2  
3 public class Casa {  
4     String cor;  
5     int numeroComodos;  
6  
7     // Método construtor  
8     public Casa(String cor, int numeroComodos) {  
9         this.cor = cor;  
10        this.numeroComodos = numeroComodos;  
11    }  
12 }  
13
```



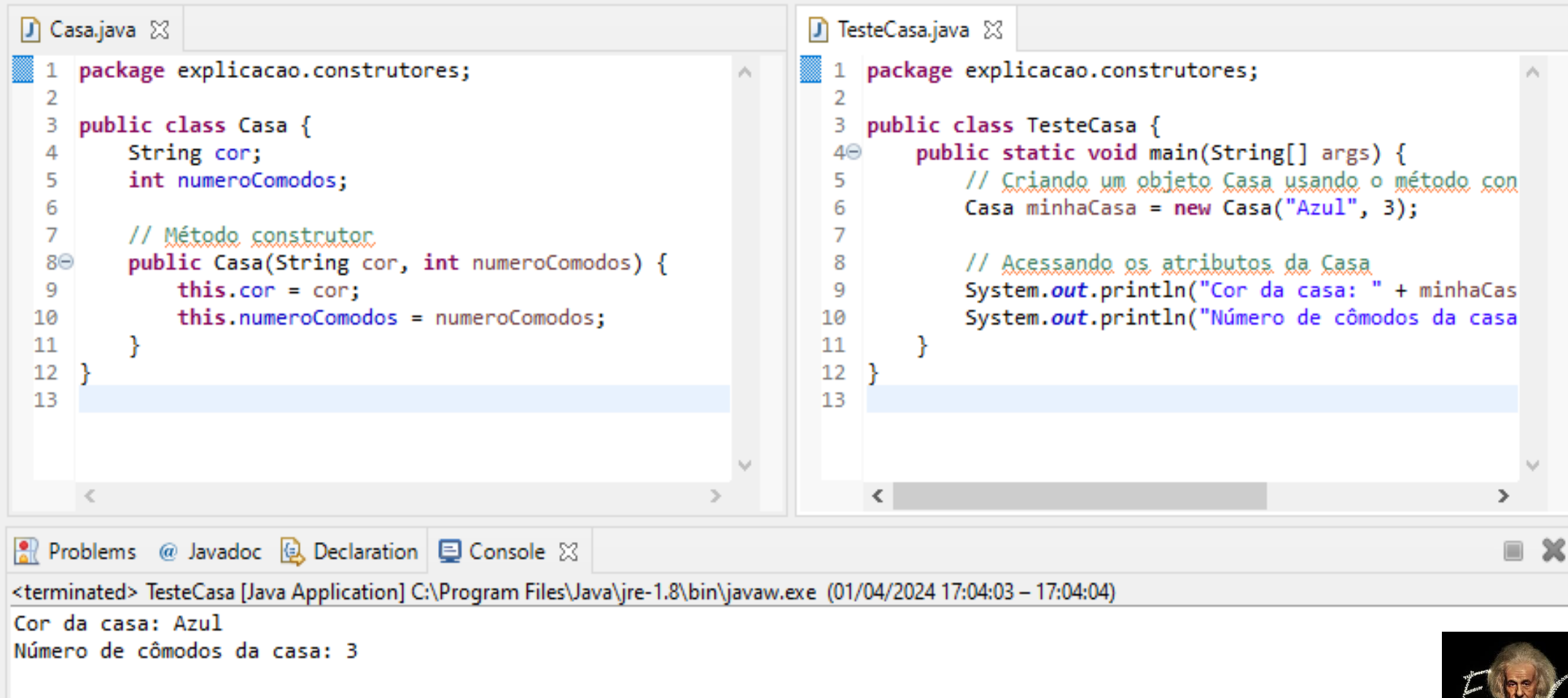
Java - Métodos construtores

—
criamos um método construtor para a classe Casa. Este método tem o mesmo nome da classe (Casa) e não possui tipo de retorno. Ele recebe parâmetros (como a cor e o número de cômodos da casa) e os utiliza para inicializar os atributos correspondentes (cor e numeroComodos).

Agora, quando criamos um objeto Casa, podemos passar os valores desejados para o método construtor:



Java - Métodos construtores



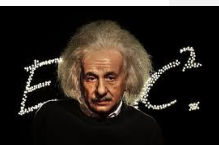
```
1 package explicacao.construtores;
2
3 public class Casa {
4     String cor;
5     int numeroComodos;
6
7     // Método construtor
8     public Casa(String cor, int numeroComodos) {
9         this.cor = cor;
10        this.numeroComodos = numeroComodos;
11    }
12 }
13
```

```
1 package explicacao.construtores;
2
3 public class TesteCasa {
4     public static void main(String[] args) {
5         // Criando um objeto Casa usando o método com
6         Casa minhaCasa = new Casa("Azul", 3);
7
8         // Acessando os atributos da Casa
9         System.out.println("Cor da casa: " + minhaCasa.cor);
10        System.out.println("Número de cômodos da casa: " + minhaCasa.numeroComodos);
11    }
12 }
13
```

Problems @ Javadoc Declaration Console

<terminated> TesteCasa [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:04:03 – 17:04:04)

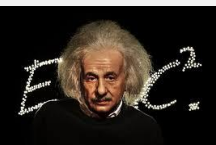
Cor da casa: Azul
Número de cômodos da casa: 3



Java - Métodos construtores

Quando você cria um novo objeto Casa usando `new Casa("Azul", 3)`, o método construtor é chamado automaticamente e os atributos `cor` e `numeroComodos` são inicializados com os valores fornecidos.

Em resumo, métodos construtores em Java são usados para inicializar os atributos de um objeto quando ele é criado. Eles fornecem uma maneira conveniente de configurar o estado inicial de um objeto e são chamados automaticamente quando o objeto é instanciado.

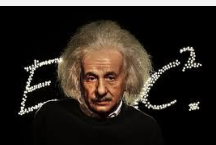


Java - Sobrecarga de métodos

—
A sobrecarga (overload) consiste num conceito de polimorfismo que é empregado na criação de variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe.

O que os difere são seus argumentos.

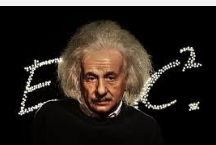
Só dessa forma é feita a separação destes.



Java - Sobrecarga de métodos

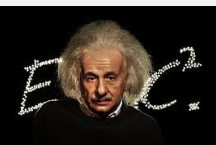
Sua principal característica é possuir mais uma classe.

Nesse sentido, consiste na primeira herdar a segunda, na sobreposição é importante fazer o uso da annotation `@Override` para indicar que o método foi sobreposto.



Java - Sobrecarga de métodos

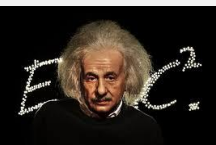
Você tem uma calculadora e quer adicionar dois números. Você pode criar um método chamado adicionar que aceita dois números como argumentos e retorna a soma deles. Mas e se você quiser adicionar três números? Ou quatro? Seria útil ter diferentes versões do método adicionar que aceitem diferentes números de argumentos, certo?



Java - Sobrecarga de métodos

Em Java, a sobrecarga de métodos permite fazer isso. Ela permite definir vários métodos com o mesmo nome em uma classe, desde que tenham diferentes tipos de parâmetros ou números de parâmetros. O compilador Java pode distinguir entre esses métodos com base nos tipos ou números de argumentos passados.

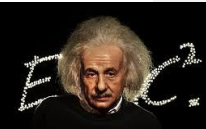
Por exemplo, vamos criar uma classe Calculadora em Java com vários métodos adicionar sobrecarregados:



Java - Sobrecarga de métodos

Calculadora.java

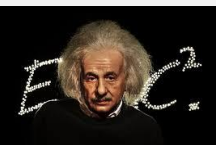
```
1 package explicacao.sobrecarga;
2
3 public class Calculadora {
4     // Método para adicionar dois números inteiros
5     public int adicionar(int a, int b) {
6         return a + b;
7     }
8
9     // Método para adicionar três números inteiros
10    public int adicionar(int a, int b, int c) {
11        return a + b + c;
12    }
13
14    // Método para adicionar dois números decimais
15    public double adicionar(double a, double b) {
16        return a + b;
17    }
18 }
19
```



Java - Sobrecarga de métodos

Neste exemplo, temos três versões do método adicionar na classe Calculadora. Uma versão aceita dois argumentos inteiros, outra aceita três argumentos inteiros e a terceira aceita dois argumentos decimais (do tipo double).

Agora podemos usar esses métodos adicionar de acordo com nossas necessidades:



Java - Sobrecarga de métodos

```
1 package explicacao.sobrecarga;  
2  
3 public class Calculadora {  
4     // Método para adicionar dois números inteiros  
5     public int adicionar(int a, int b) {  
6         return a + b;  
7     }  
8  
9     // Método para adicionar três números inteiros  
10    public int adicionar(int a, int b, int c) {  
11        return a + b + c;  
12    }  
13  
14    // Método para adicionar dois números decimais  
15    public double adicionar(double a, double b) {  
16        return a + b;  
17    }  
18 }  
19
```

```
1 package explicacao.sobrecarga;  
2  
3 public class TesteCalculadora {  
4     public static void main(String[] args) {  
5         Calculadora calc = new Calculadora();  
6  
7         // Usando o método adicionar para somar dois números inteiros  
8         System.out.println("Soma de 5 e 7: " + calc.adicionar(5, 7));  
9  
10        // Usando o método adicionar para somar três números inteiros  
11        System.out.println("Soma de 5, 7 e 10: " + calc.adicionar(5, 7, 10));  
12  
13        // Usando o método adicionar para somar dois números decimais  
14        System.out.println("Soma de 3.5 e 2.5: " + calc.adicionar(3.5, 2.5));  
15    }  
16 }  
17
```

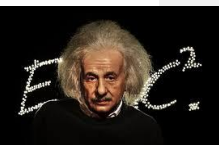
Problems @ Javadoc Declaration Console

<terminated> TesteCalculadora [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:09:34 - 17:09:34)

Soma de 5 e 7: 12

Soma de 5, 7 e 10: 22

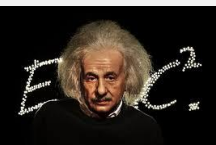
Soma de 3.5 e 2.5: 6.0



Java - Operador ternário

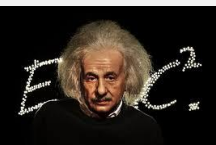
Cada chamada do método adicionar acima chama uma versão diferente do método, dependendo do número e tipo de argumentos passados.

Em resumo, a sobrecarga de métodos em Java permite definir vários métodos com o mesmo nome em uma classe, desde que tenham diferentes tipos ou números de parâmetros. Isso proporciona flexibilidade e conveniência ao trabalhar com métodos que têm comportamentos semelhantes, mas diferentes requisitos de entrada.



Java - Sobreposição de métodos

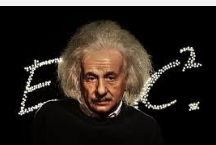
Imagine que você tem uma classe base chamada `Animal` com um método `fazerSom()`, que representa o som que um animal faz. Agora, você quer criar subclasses específicas, como `Cachorro`, `Gato` e `Vaca`, e cada uma delas faz um som diferente. Em vez de criar métodos diferentes para cada animal, você pode usar a sobreposição de métodos.



Java - Sobreposição de métodos

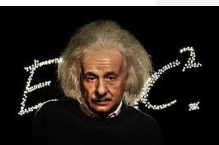
A sobreposição de métodos em Java ocorre quando uma classe filha fornece uma implementação específica de um método que já está sendo usado em sua classe pai. Isso significa que a classe filha "substitui" a implementação do método na classe pai com sua própria implementação.

Por exemplo, vamos criar uma classe Animal com um método fazerSom() e subclasses como Cachorro, Gato e Vaca que sobrepõem esse método:



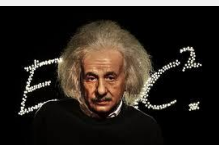
Java - Sobreposição de métodos

```
Animal.java
1 package explicacao.sobreposicao;
2
3 //Classe base
4 class Animal {
5     void fazerSom() {
6         System.out.println("Este animal emite algum som.");
7     }
8 }
9
10 //Subclasse Cachorro
11 class Cachorro extends Animal {
12     @Override
13     void fazerSom() {
14         System.out.println("Au Au");
15     }
16 }
17
18 //Subclasse Gato
19 class Gato extends Animal {
20     @Override
21     void fazerSom() {
22         System.out.println("Miau");
23     }
24 }
25
26 //Subclasse Vaca
27 class Vaca extends Animal {
28     @Override
29     void fazerSom() {
30         System.out.println("Muu");
31     }
32 }
33
```



Java - Sobreposição de métodos

Agora, se criarmos objetos dessas subclasses e chamarmos o método `fazerSom()`, obteremos o som específico de cada animal:



Java - Sobreposição de métodos

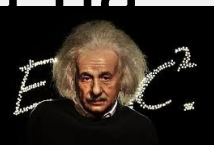
```
Animal.java
1 package explicacao.sobreposicao;
2
3 //Classe base
4 class Animal {
5     void fazerSom() {
6         System.out.println("Este animal emite algum som.");
7     }
8 }
9
10 //Subclasse Cachorro
11 class Cachorro extends Animal {
12     @Override
13     void fazerSom() {
14         System.out.println("Au Au");
15     }
16 }
17
18 //Subclasse Gato
19 class Gato extends Animal {
20     @Override
21     void fazerSom() {
22         System.out.println("Miau");
23     }
24 }
25
26 //Subclasse Vaca
27 class Vaca extends Animal {
28     @Override
29     void fazerSom() {
30         System.out.println("Muu");
31     }
32 }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657

```

Java - Sobreposição de métodos

Neste exemplo, cada objeto Cachorro, Gato e Vaca substitui a implementação do método fazerSom() da classe Animal com sua própria implementação específica.

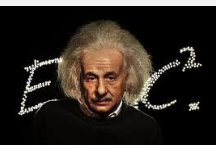
Em resumo, a sobreposição de métodos em Java permite que as subclasses forneçam uma implementação específica de um método definido em sua classe pai, substituindo a implementação original. Isso é útil quando queremos que objetos de diferentes subclasses se comportem de maneiras diferentes, mas mantenham uma interface comum definida na classe pai.



Java - IF e IF-ELSE

A principal estrutura de decisão é o comando if e if-else. Esses comandos também podem ser concatenados sucessivamente por outros ifs, como em if-elseif-else, etc.

you are taking a decision about whether you should go out today or not. You may have some conditions that influence your decision, like the weather. If it is sunny, you will go out. If it is raining, you may decide to stay at home. In Java, you can express these decisions using the if structure.



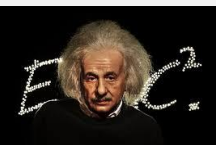
Java - IF e IF-ELSE

IF: —

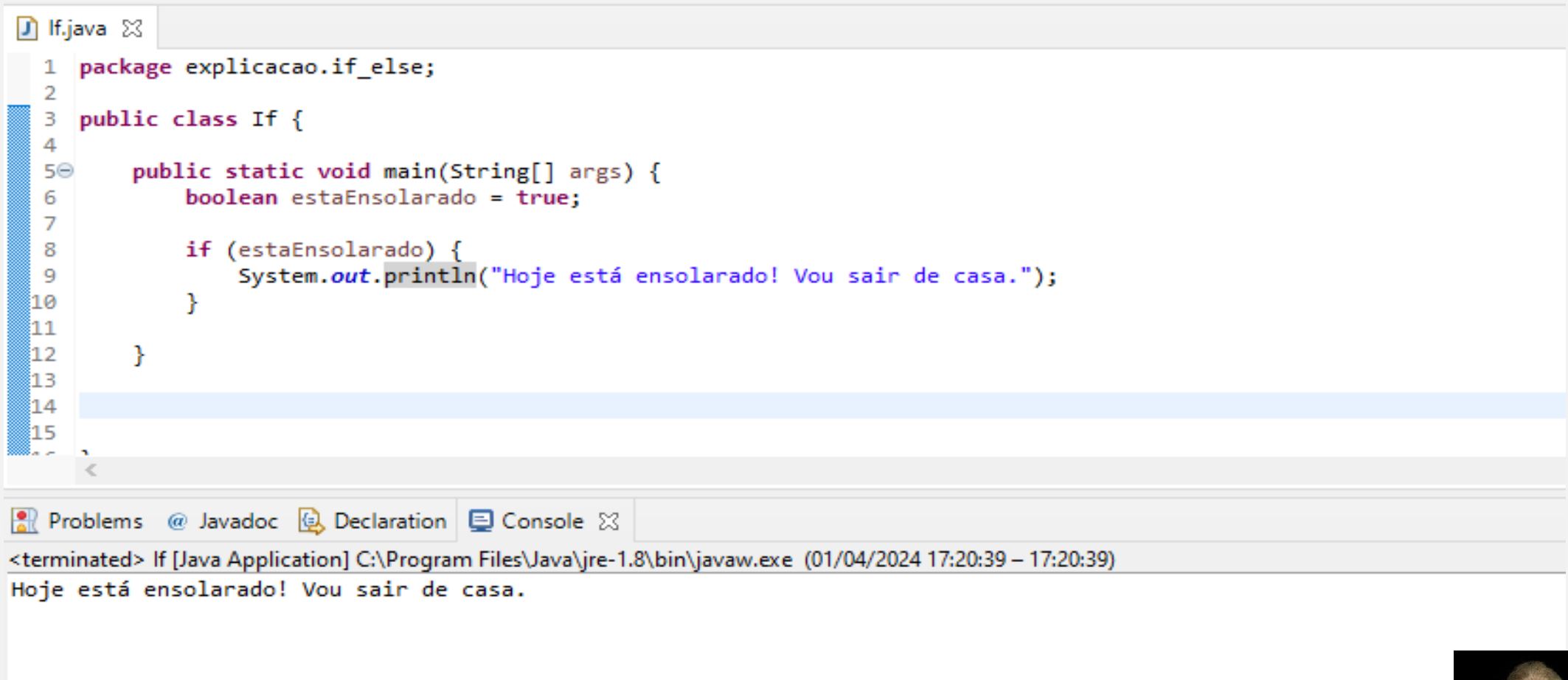
O if é uma estrutura de controle de fluxo em Java que permite executar um bloco de código se uma condição específica for verdadeira.

Se a condição não for verdadeira, o bloco de código não é executado.

Por exemplo, vamos usar o if para expressar a decisão de sair de casa com base no clima:



Java - IF e IF-ELSE

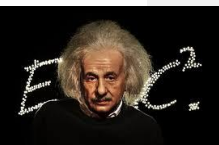


```
1 package explicacao.if_else;
2
3 public class If {
4
5     public static void main(String[] args) {
6         boolean estaEnsolarado = true;
7
8         if (estaEnsolarado) {
9             System.out.println("Hoje está ensolarado! Vou sair de casa.");
10        }
11
12    }
13
14
15
16 }
```

Problems @ Javadoc Declaration Console

<terminated> If [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:20:39 – 17:20:39)

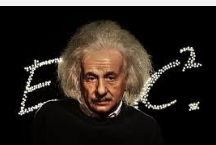
Hoje está ensolarado! Vou sair de casa.



Java - IF e IF-ELSE

IF-ELSE:

Às vezes, você pode ter mais de uma opção ou ramificação em sua decisão. Por exemplo, se estiver chovendo, você pode decidir ficar em casa, mas se estiver ensolarado, você vai sair. Nesse caso, você pode usar a estrutura if-else.



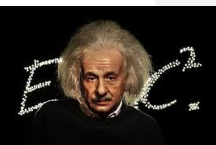
Java - IF e IF-ELSE

```
If.java ✕
1 package explicacao.if_else;
2
3 public class If {
4
5     public static void main(String[] args) {
6         boolean estaChovendo = false;
7
8         if (estaChovendo) {
9             System.out.println("Hoje está chovendo. Vou ficar em casa.");
10        } else {
11            System.out.println("Hoje não está chovendo. Vou sair de casa.");
12        }
13    }
14 }
15
16
```

Problems @ Javadoc Declaration Console ✕

<terminated> If [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:21:16 – 17:21:18)

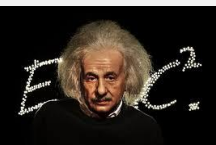
Hoje não está chovendo. Vou sair de casa.



Java - IF e IF-ELSE

Neste exemplo, o bloco de código dentro do if é executado se estaChovendo for verdadeiro, caso contrário, o bloco de código dentro do else é executado.

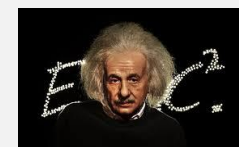
Em resumo, o if é usado para executar um bloco de código se uma condição for verdadeira, enquanto o if-else é usado para executar um bloco de código se uma condição for verdadeira e outro bloco de código se a condição for falsa. Essas estruturas de controle de fluxo permitem que seu programa tome decisões com base em certas condições.



Java - Operador ternário

Considerada uma forma simplificada de construção de comandos do tipo if-else. O nome deve-se ao fato de que o comando é quebrado em três partes separadas pelos símbolos de interrogação (?) e de dois pontos (:).

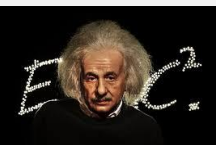
O operador ternário em Java é uma forma concisa de escrever uma estrutura condicional if-else em apenas uma linha. Ele permite que você avalie uma expressão condicional e retorne um valor com base nessa avaliação. Vou explicar de forma didática como funciona.



Java - Operador ternário

Imagine que você tem uma condição simples e deseja atribuir um valor a uma variável dependendo dessa condição. Por exemplo, vamos considerar uma situação em que queremos verificar se uma pessoa é maior de idade e, em seguida, atribuir um valor de acordo com essa verificação.

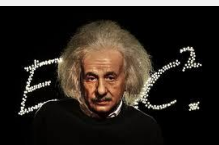
Sem o operador ternário, você teria que usar uma estrutura if-else, assim:



Java - Operador ternário

Exemplo

Comando: if-else	Comando: operador-ternário
<pre>if (ExpLógicaA) { ComandoA; } else { ComandoB; }</pre>	<pre>ExpLógicaA ? ComandoA : ComandoB;</pre>

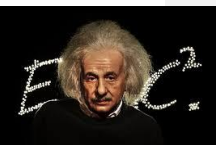


Java - Operador ternário

```
OperadorTernario.java
1 package explicacao.operadorTernario;
2
3 public class OperadorTernario {
4
5     public static void main(String[] args) {
6         int idade = 20;
7         String status;
8
9         if (idade >= 18) {
10             status = "Maior de idade";
11         } else {
12             status = "Menor de idade";
13         }
14
15         System.out.println("Status: " + status);
16
17     }
18 }
19
```

Problems @ Javadoc Declaration Console

<terminated> OperadorTernario [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:24:04 – 17:24:04)
Status: Maior de idade



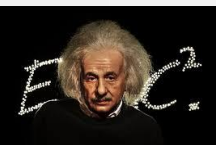
Java - Operador ternário

Com o operador ternário, você pode fazer a mesma coisa em uma única linha de código:

```
OperadorTernario.java
1 package explicacao.operadorTernario;
2
3 public class OperadorTernario {
4
5     public static void main(String[] args) {
6         int idade = 20;
7         String status = (idade >= 18) ? "Maior de idade" : "Menor de idade";
8
9         System.out.println("Status: " + status);
10
11     }
12 }
13
```

Problems Javadoc Declaration Console

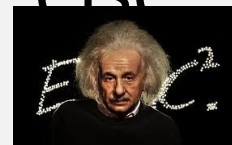
<terminated> OperadorTernario [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:24:47 – 17:24:49)
Status: Maior de idade



Java - Operador ternário

Neste exemplo, a expressão (`idade >= 18`) é avaliada primeiro. Se for verdadeira, o valor após o `?` é retornado (nesse caso, "Maior de idade"). Se for falsa, o valor após o `:` é retornado (nesse caso, "Menor de idade").

O operador ternário é especialmente útil quando você tem uma condição simples e deseja atribuir um valor com base nessa condição em uma única linha de código. No entanto, é importante usá-lo com moderação para manter o código legível. Quando a lógica fica muito complexa, é melhor usar uma estrutura `if-else` convencional.

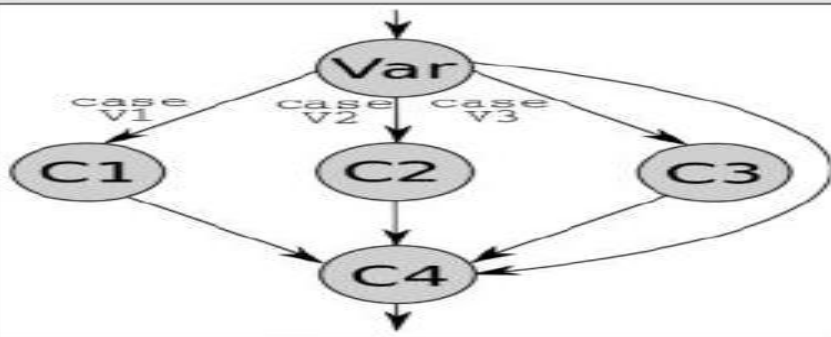


Java - Switch-case | Switch-case-default

Esse comando provê o que chamamos de estrutura de seleção múltipla baseada em alguma variável de controle.

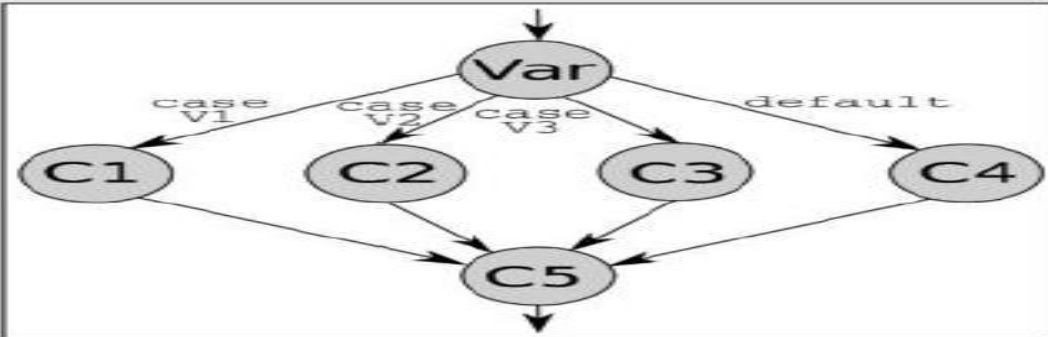
Exemplo

```
switch (Var) {  
  case V1:  
    C1; break;  
  case V2:  
    C2; break;  
  case V3:  
    C3; break;  
}  
C4
```

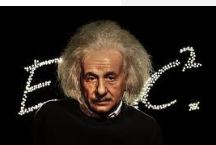


switch-case

```
switch (Var) {  
  case V1:  
    C1; break;  
  case V2:  
    C2; break;  
  case V3:  
    C3; break;  
  default:  
    C4;  
}  
C5
```

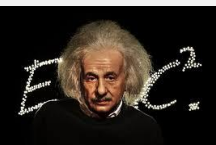


switch-case-default



Java - Switch-case | Switch-case-default

—
você está em uma sorveteria e quer escolher um sabor de sorvete. Você tem várias opções de sabores e deseja tomar uma decisão com base no sabor escolhido. O switch-case em Java é uma estrutura de controle de fluxo que permite tomar decisões com base no valor de uma variável.

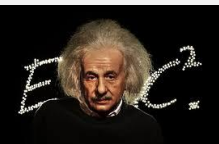


Java - Switch-case | Switch-case-default

Switch-case:

O switch-case é uma estrutura que permite avaliar uma expressão e, dependendo do resultado dessa avaliação, executar um bloco de código específico. Ele oferece uma alternativa mais clara e concisa do que encadear vários if-else quando você tem muitas condições para avaliar.

Por exemplo, imagine que você quer escolher um sabor de sorvete e seu sabor favorito é baunilha:

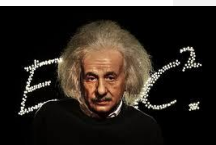


Java - Switch-case | Switch-case-default

```
SwitchCase.java
1 package explicacao.switchCase;
2
3 public class SwitchCase {
4
5     public static void main(String[] args) {
6         String sabor = "baunilha";
7
8         switch (sabor) {
9             case "baunilha":
10                 System.out.println("Sabor escolhido: Baunilha");
11                 break;
12             case "morango":
13                 System.out.println("Sabor escolhido: Morango");
14                 break;
15             case "chocolate":
16                 System.out.println("Sabor escolhido: Chocolate");
17                 break;
18             default:
19                 System.out.println("Sabor não reconhecido");
20         }
21     }
22 }
23
24
```

Problems Javadoc Declaration Console

<terminated> SwitchCase [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:29:58 – 17:29:58)
Sabor escolhido: Baunilha

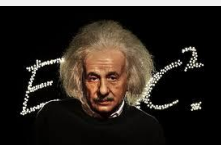


Java - Switch-case | Switch-case-default

Switch-case-default:

O bloco default é opcional e é usado para lidar com situações em que nenhum dos casos corresponde ao valor da expressão. Ele é executado quando nenhum dos outros casos foi acionado.

Por exemplo, se você tentar escolher um sabor de sorvete que não esteja na lista:



Java - Switch-case | Switch-case-default

```
SwitchCase.java
1 package explicacao.switchCase;
2
3 public class SwitchCase {
4
5     public static void main(String[] args) {
6         String sabor = "";
7
8         switch (sabor) {
9             case "baunilha":
10                 System.out.println("Sabor escolhido: Baunilha");
11                 break;
12             case "morango":
13                 System.out.println("Sabor escolhido: Morango");
14                 break;
15             case "chocolate":
16                 System.out.println("Sabor escolhido: Chocolate");
17                 break;
18             default:
19                 System.out.println("Sabor não reconhecido");
20         }
21     }
22 }
23
24
```

Problems @ Javadoc Declaration Console

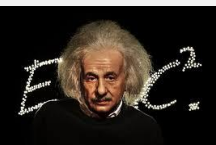
<terminated> SwitchCase [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:30:57 – 17:30:58)

Sabor não reconhecido



Java - Switch-case | Switch-case-default

Em resumo, o switch-case em Java é uma estrutura de controle de fluxo que permite avaliar uma expressão e tomar decisões com base nos diferentes valores dessa expressão. O bloco default é opcional e é executado quando nenhum dos outros casos corresponde ao valor da expressão.

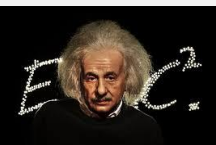


Java - Repetição com while / for / do while

1. Repetição com while:

O while é uma estrutura de repetição que permite executar um bloco de código enquanto uma condição específica for verdadeira. Ele verifica a condição antes de cada execução do bloco de código.

Vamos ver um exemplo onde queremos contar de 1 a 5 usando um while:



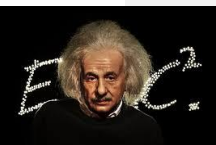
Java - Repetição com while / for / do while

```
Laco_While.java
1 package explicacao.while_for_dowhile;
2
3 public class Laco_While {
4
5     public static void main(String[] args) {
6
7         int contador = 1;
8
9         while (contador <= 5) {
10             System.out.println(contador);
11             contador++;
12         }
13
14
15     }
16 }
```

Problems @ Javadoc Declaration Console

<terminated> Laco_While [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:35:00 – 17:35:02)

```
1
2
3
4
5
```

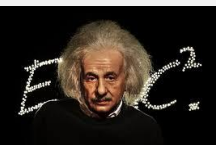


Java - Repetição com while / for / do while

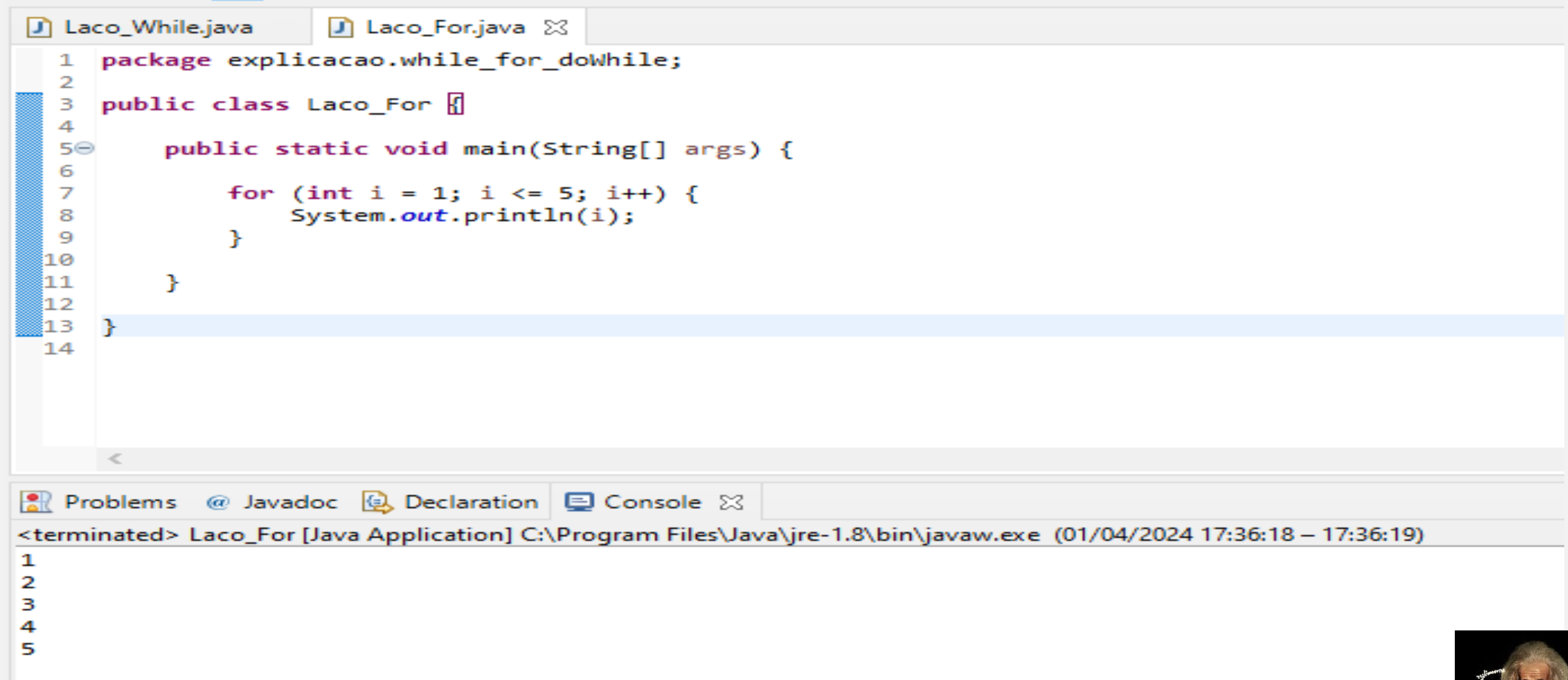
2. Repetição com for:

O for é outra estrutura de repetição em Java. Ele é frequentemente usado quando o número de iterações é conhecido.

Aqui está o mesmo exemplo anterior de contagem de 1 a 5, mas desta vez usando um for:



Java - Repetição com while / for / do while



The screenshot shows an IDE with two tabs: `Laco_While.java` and `Laco_For.java`. The `Laco_For.java` tab is active, displaying the following code:

```
1 package explicacao.while_for_dowhile;
2
3 public class Laco_For {
4     public static void main(String[] args) {
5         for (int i = 1; i <= 5; i++) {
6             System.out.println(i);
7         }
8     }
9 }
```

The code is a Java application that prints the numbers 1 through 5 using a `for` loop. The IDE interface includes a toolbar with icons for Problems, Javadoc, Declaration, and Console. The Console output shows the program's execution:

```
<terminated> Laco_For [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:36:18 – 17:36:19)
1
2
3
4
5
```

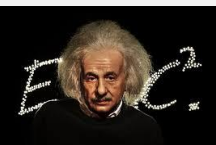
In the bottom right corner, there is a small image of Albert Einstein with the text "E.C." next to him.

Java - Repetição com while / for / do while

3. Repetição com do-while:

O do-while é semelhante ao while, mas a condição é verificada após a execução do bloco de código, garantindo que o bloco seja executado pelo menos uma vez.

Vamos considerar um exemplo onde pedimos ao usuário para fornecer um número positivo usando do-while:



Java - Repetição com while / for / do while

```
Laco_While.java  Laco_For.java  *Laco_DoWhile.java ✖
1 package explicacao.while_for_doWhile;
2
3 import java.util.Scanner;
4
5 public class Laco_DoWhile {
6
7     public static void main(String[] args) {
8
9         int numero;
10
11         do {
12             System.out.println("Digite um número positivo: ");
13             Scanner ler = new Scanner(System.in);
14             numero = ler.nextInt();
15         } while (numero <= 0);
16         System.out.println("Número positivo fornecido: " + numero);
17     }
18
19 }
20
```

Problems Javadoc Declaration Console ✖

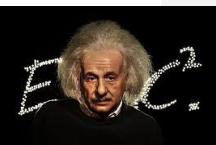
<terminated> Laco_DoWhile [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:45:23 – 17:45:28)

Digite um número positivo:

1

Número positivo fornecido: 1

|

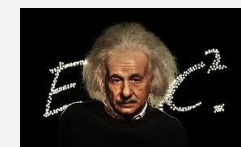


Java – Tratamento de Exceção

--- O que são exceções?

Em Java, uma exceção é um evento que ocorre durante a execução de um programa e interrompe o fluxo normal de execução.

As exceções podem ser causadas por erros de programação, erros de hardware, entrada de usuário incorreta, entre outras situações imprevistas.

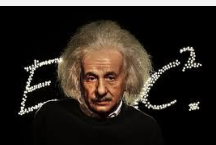


Java – Tratamento de Exceção

— Tratamento de exceções em Java:

O tratamento de exceções em Java permite lidar com essas situações imprevistas de forma controlada, evitando que o programa seja encerrado abruptamente.

Existem duas principais partes no tratamento de exceções em Java: try-catch e finally.

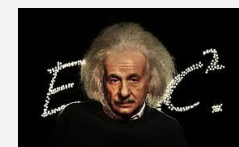


Java – Tratamento de Exceção

1. Bloco try-catch:

O bloco try-catch é usado para envolver o código que pode gerar uma exceção. Dentro do bloco try, você coloca o código que pode lançar uma exceção, e dentro do bloco catch, você trata a exceção caso ela ocorra.

Vamos modificar o exemplo anterior para incluir um bloco try-catch:

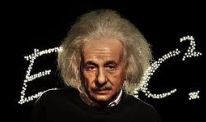


Java – Tratamento de Exceção

```
*Try_catth.java ✖
1 package explicacao.try_catth;
2
3 public class Try_catth {
4
5     public static void main(String[] args) {
6         int dividendo = 10;
7         int divisor = 0;
8
9         try {
10             int resultado = dividendo / divisor;
11             System.out.println(resultado);
12         } catch (ArithmeticException e) {
13             System.out.println("Ocorreu uma exceção: " + e.getMessage());
14         }
15     }
16 }
17
18
```

Problems @ Javadoc Declaration Console ✖

<terminated> Try_catth [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:50:48 – 17:50:48)
Ocorreu uma exceção: / by zero

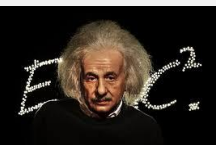


Java – Tratamento de Exceção

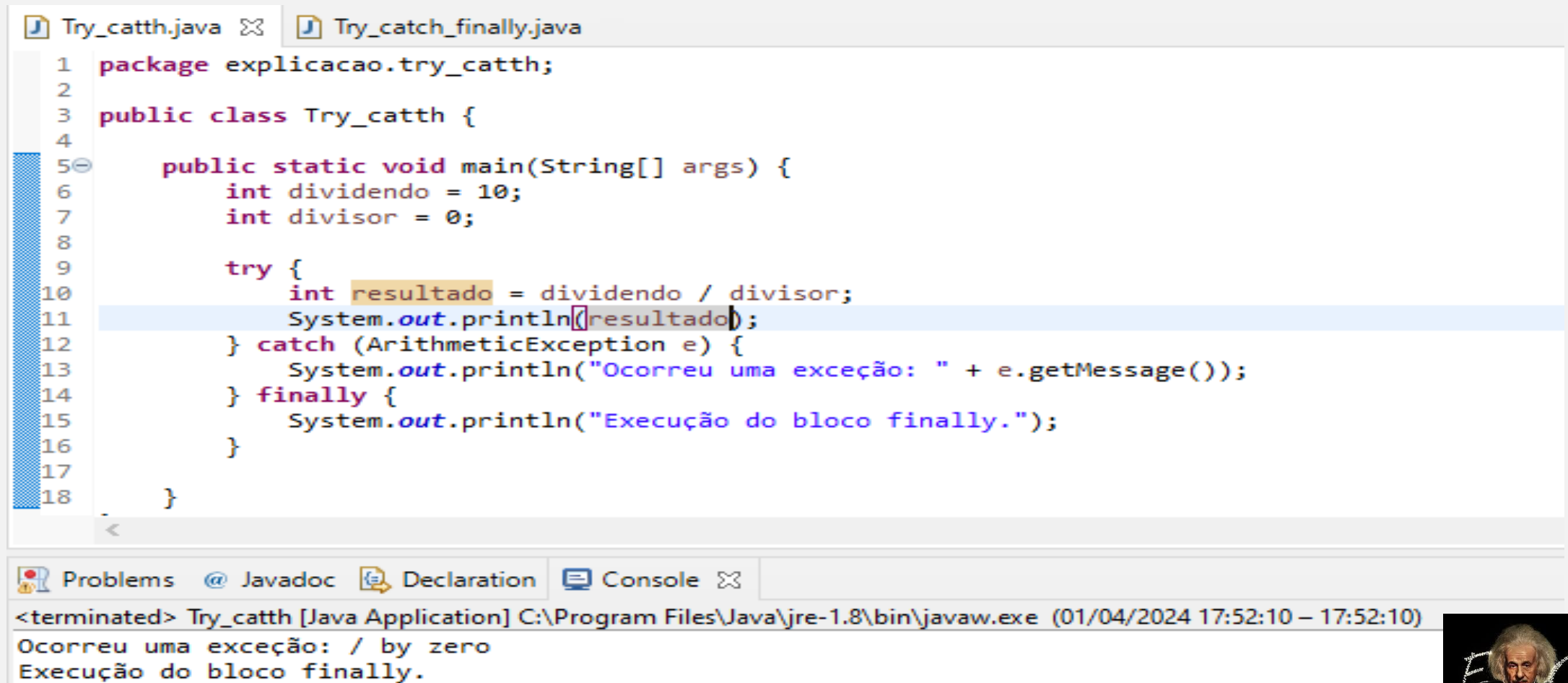
2. Bloco finally:

O bloco finally é opcional e é usado para executar código que deve ser executado independentemente de ocorrer uma exceção ou não. É útil para liberar recursos ou realizar limpeza, por exemplo.

Vamos adicionar um bloco finally ao exemplo anterior:



Java – Tratamento de Exceção



The screenshot displays an IDE with two tabs: `Try_catth.java` and `Try_catch_finally.java`. The `Try_catth.java` tab is active, showing the following code:

```
1 package explicacao.try_catth;
2
3 public class Try_catth {
4
5     public static void main(String[] args) {
6         int dividendo = 10;
7         int divisor = 0;
8
9         try {
10             int resultado = dividendo / divisor;
11             System.out.println(resultado);
12         } catch (ArithmeticException e) {
13             System.out.println("Ocorreu uma exceção: " + e.getMessage());
14         } finally {
15             System.out.println("Execução do bloco finally.");
16         }
17     }
18 }
```

The code attempts to divide 10 by 0, which triggers an `ArithmeticException`. The `catch` block captures this exception and prints the message "Ocorreu uma exceção: / by zero". The `finally` block executes and prints "Execução do bloco finally.".

The IDE's status bar at the bottom shows the execution output:

```
<terminated> Try_catth [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:52:10 – 17:52:10)
Ocorreu uma exceção: / by zero
Execução do bloco finally.
```

In the bottom right corner, there is a small portrait of Albert Einstein with the letters "E.C." next to it.

Java – Tratamento de Exceção

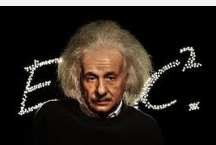
Exceções

19

ArithmeticException: lançada quando uma condição aritmética excepcional ocorre, como, por exemplo, uma divisão por zero de números inteiros.

IndexOutOfBoundsException: lançada para indicar que um índice de algum tipo, como um vetor, uma string ou uma matriz, está fora do intervalo.

ArrayIndexOutOfBoundsException: lançada para indicar que um vetor foi acessado com um índice ilegal, como valor negativo ou maior ao tamanho do vetor.



Java – Tratamento de Exceção

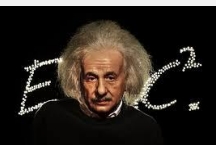
Exceções

20

IllegalArgumentException: lançada para indicar que um método recebeu um argumento ilegal ou inapropriado.

NumberFormatException: lançada para indicar que a aplicação tentou converter um valor em algum tipo numérico, mas o valor não possui o formato apropriado.

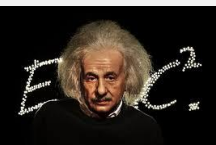
NullPointerException: lançada quando uma aplicação tenta usar um objeto null quando uma instância de objeto é necessária.



Java - For Each

O for-each em Java é uma forma simplificada de iterar sobre elementos de um array ou coleção, sem a necessidade de usar índices ou contadores. É uma estrutura de loop conveniente e legível, especialmente ao lidar com coleções de objetos.

Imagine que você tem um array de números e deseja percorrer cada elemento para realizar alguma operação com eles. Em vez de usar um for tradicional com índices, você pode usar o for-each para percorrer o array de uma maneira mais simples e limpa.



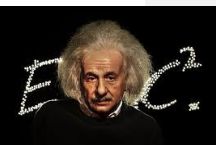
Java - For Each

```
For_each.java
1 package explicacao.for_each;
2
3 public class For_each {
4
5     public static void main(String[] args) {
6         int[] numeros = { 1, 2, 3, 4, 5 };
7
8         for (int numero : numeros) {
9             System.out.println(numero);
10        }
11    }
12 }
13
14 }
15
```

Problems @ Javadoc Declaration Console

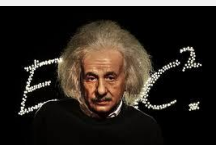
<terminated> For_each [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:57:38 – 17:57:38)

```
1
2
3
4
5
```



Java - For Each

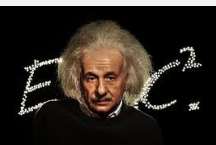
—
Neste exemplo, o for-each percorre cada elemento do array `numeros` e armazena temporariamente o valor de cada elemento na variável `numero`. Em cada iteração do loop, o valor de `numero` é atualizado para o próximo elemento do array, e o bloco de código dentro do loop é executado.



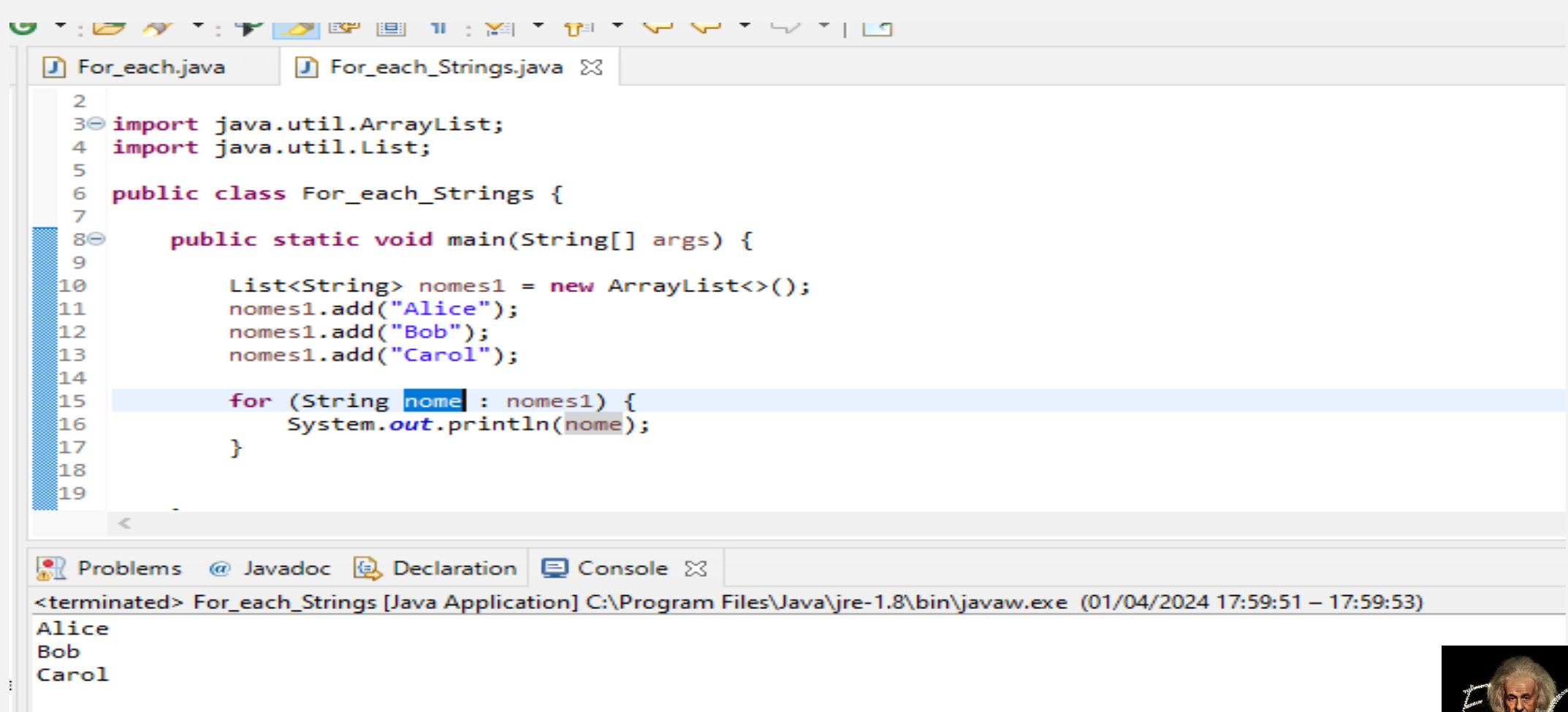
Java - For Each

O for-each também pode ser usado com coleções, como listas, conjuntos e mapas. Aqui está um exemplo de como usar o for-each com uma lista de strings:

Neste exemplo, o for-each itera sobre cada elemento da lista nomes e armazena temporariamente o valor de cada elemento na variável nome, que é uma string. O bloco de código dentro do loop é executado para cada elemento da lista.



Java - For Each



The screenshot shows an IDE with two tabs: `For_each.java` and `For_each_Strings.java`. The `For_each_Strings.java` tab is active, displaying the following code:

```
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 public class For_each_Strings {
7
8     public static void main(String[] args) {
9
10         List<String> nomes1 = new ArrayList<>();
11         nomes1.add("Alice");
12         nomes1.add("Bob");
13         nomes1.add("Carol");
14
15         for (String nome : nomes1) {
16             System.out.println(nome);
17         }
18
19     }
```

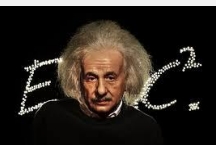
The IDE's output console at the bottom shows the execution results:

```
<terminated> For_each_Strings [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 17:59:51 - 17:59:53)
Alice
Bob
Carol
```

In the bottom right corner, there is a small, stylized image of Albert Einstein with the letters "E.C." next to him.

Java - For Each

O for-each é uma maneira conveniente e eficiente de percorrer arrays e coleções em Java, reduzindo a quantidade de código necessário e tornando o código mais legível. É uma boa prática usar o for-each sempre que possível, especialmente ao lidar com iterações simples sobre coleções de objetos



Java - Interfaces

30

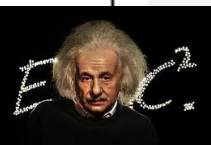
Interfaces

Algumas interfaces implementadas:

Comparable: é utilizada para impor uma ordem nos objetos de uma determinada classe que a implementa.

Runnable: é utilizada para especificar alguma tarefa a ser realizada.

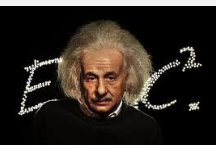
Serializable: é utilizada para identificar classes em que os objetos podem ser gravados (também chamados de serializados) ou lidos (também chamados de desserializados) de algum dispositivo de armazenamento, como HD.



Java - Interfaces

Interfaces em Java são como contratos que especificam um conjunto de métodos que uma classe deve implementar se quiser cumprir o contrato. De forma mais didática, imagine que você tem um manual de instruções para um brinquedo. Esse manual diz que o brinquedo deve ter botões específicos, fazer determinados sons e realizar certos movimentos quando acionado.

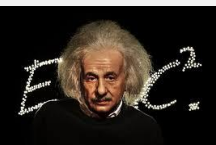
Nesse caso, o manual é como uma interface e o brinquedo é a classe que implementa essa interface.



Java - Interfaces

```
*Animal.java ✖
1 package explicacao.interfaces;
2
3 public interface Animal {
4
5     void fazerSom();
6
7     void mover();
8 }
9
```

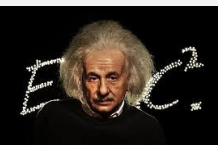
Animal é a nossa interface. Ela especifica que qualquer classe que a implementar deve ter dois métodos: `fazerSom()` e `mover()`.



Java - Interfaces

—
vamos criar uma classe Cachorro que implementa essa interface:

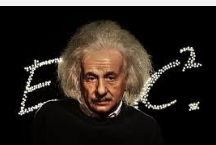
```
Animal.java  *Cachorro.java ✕  
1 package explicacao.interfaces;  
2  
3 public class Cachorro implements Animal {  
4  
5     @Override  
6     public void fazerSom() {  
7         System.out.println("Au au!");  
8     }  
9  
10    @Override  
11    public void mover() {  
12        System.out.println("Correndo...");  
13    }  
14 }  
15
```



Java - Interfaces

A classe Cachorro implementa a interface Animal e fornece implementações para os métodos fazerSom() e mover(). Isso significa que um cachorro, que é um tipo de animal, deve ser capaz de fazer som e se mover.

Agora, podemos criar uma instância de Cachorro e chamá-la:



Java - Interfaces

Animal.java

```
1 package explicacao.interfaces;
2
3 public interface Animal {
4
5     void fazerSom();
6
7     void mover();
8 }
9
```

Cachorro.java

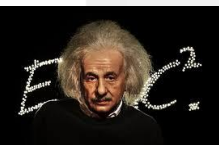
```
1 package explicacao.interfaces;
2
3 public class Cachorro implements Animal {
4
5     @Override
6     public void fazerSom() {
7         System.out.println("Au au!");
8     }
9
10    @Override
11    public void mover() {
12        System.out.println("Correndo...");
13    }
14 }
15
```

TesteInterface.java

```
1 package explicacao.interfaces;
2
3 public class TesteInterface {
4
5     public static void main(String[] args) {
6
7         Cachorro cachorro = new Cachorro();
8         cachorro.fazerSom(); // Saída: Au au!
9         cachorro.mover(); // Saída: Correndo...
10
11     }
12
13 }
14
```

Console

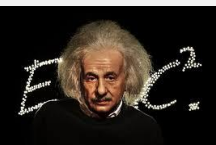
```
<terminated> TesteInterface [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bi
Au au!
Correndo...
```



Java - Interfaces

Ou seja, interfaces em Java permitem que você defina um conjunto de métodos que as classes devem implementar, o que promove um alto nível de abstração e modularidade em seu código.

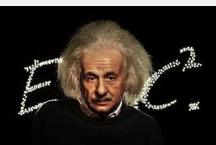
Isso é útil especialmente quando você deseja definir um contrato para diferentes classes, garantindo que elas cumpram certos requisitos.



Java - JAVAFX

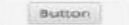




A linguagem Java possui um suporte muito bom para a criação de Interfaces Gráficas de Usuário ou, em inglês, Graphical User Interface (GUI), e existem várias formas de se fazer a criação de interfaces gráficas em Java. Ao longo da evolução dessa linguagem, diversas bibliotecas gráficas foram criadas, como:

Abstract Window Toolkit (AWT), Swing, Standard Widget Toolkit (SWT), Apache Pivot, SwingX, JGoodies, QtJambi e JavaFX.



Java - JAVAFX

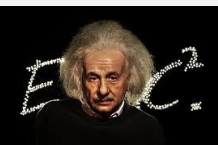
Componentes JAVAFX

Nome do Componente	Classe do Componente	Descrição do Componente	Aspecto Visual do Componente
Botão	Button	Simple botão de controle.	
Rótulo	Label	Campo que permite exibir um texto não editável.	
Campo de Texto	TextField	Campo que permite a inserção de texto em uma única linha.	
Área de Texto	TextArea	Campo que permite a inserção de texto com múltiplas linhas.	
Botão de Opção	RadioButton	Seleção de opções mutuamente excludentes.	
Caixa de Seleção	CheckBox	Seleção de opções não mutuamente excludentes.	
Caixa de Combinação	ComboBox	Seleção de opções mutuamente excludentes em forma de lista.	



Java - Vetores ou Arrays Unidimensionais

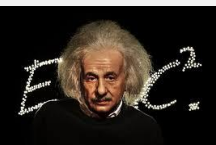
—
Vetores ou Arrays Unidimensionais -> Imagine que você tem uma prateleira (ou uma "linha") com várias caixas. Cada caixa contém um valor, como um número, uma palavra, um objeto ou qualquer outra coisa. Agora, suponha que você queira armazenar esses valores em seu programa Java. Você poderia criar uma variável para cada valor, mas isso pode ficar muito confuso e difícil de gerenciar, especialmente se você tiver muitos valores.



Java - Vetores ou Arrays Unidimensionais

É aqui que os vetores entram em cena. Um vetor é como uma prateleira virtual onde você pode armazenar vários valores do mesmo tipo. Ao invés de criar uma variável para cada valor, você pode criar um vetor para armazenar todos eles juntos.

Em Java, um vetor é uma estrutura de dados que permite armazenar uma coleção ordenada de elementos do mesmo tipo. Para criar um vetor em Java, você declara o tipo dos elementos que deseja armazenar, seguido por colchetes [] e o nome do vetor.

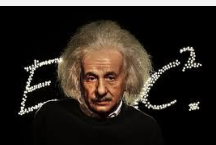


Java - Vetores ou Arrays Unidimensionais

—
Por exemplo, se você quisesse armazenar uma lista de números inteiros em um vetor, você poderia fazer assim:

```
1 package vetores;  
2  
3 public class Vetores {  
4  
5     int[] numeros;  
6  
7 }  
8
```

Isso cria um vetor chamado `numeros` que pode armazenar números inteiros.

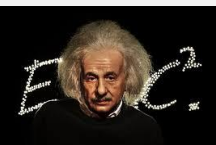


Java - Vetores ou Arrays Unidimensionais

—
Você também pode inicializar o vetor com valores iniciais ao declará-lo:

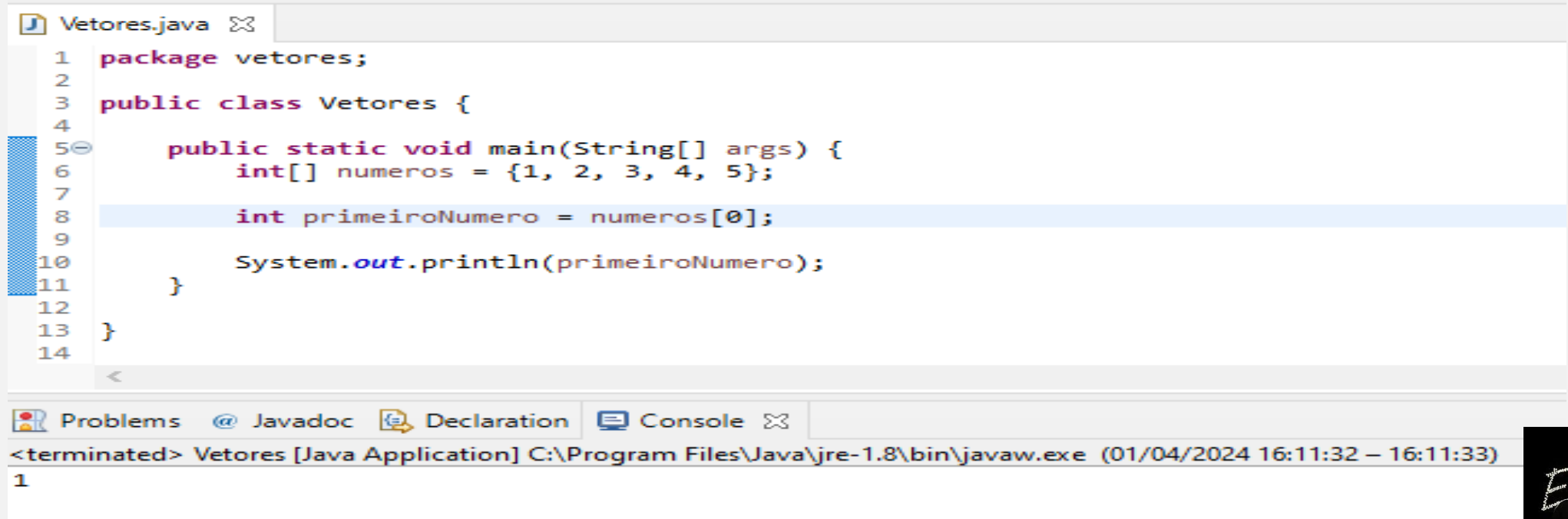
```
1 package vetores;  
2  
3 public class Vetores {  
4  
5     int[] numeros = {1, 2, 3, 4, 5};  
6  
7 }  
8
```

Agora, numeros é um vetor que contém os números de 1 a 5.



Java - Vetores ou Arrays Unidimensionais

Para acessar um elemento específico em um vetor, você usa um índice. Os índices em Java começam em 0. Por exemplo, para acessar o primeiro elemento do vetor `numeros`, você faria assim:

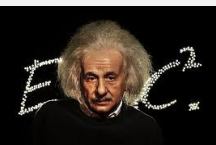


```
Vetores.java
1 package vetores;
2
3 public class Vetores {
4
5     public static void main(String[] args) {
6         int[] numeros = {1, 2, 3, 4, 5};
7
8         int primeiroNumero = numeros[0];
9
10        System.out.println(primeiroNumero);
11    }
12
13 }
14
```

Problems Javadoc Declaration Console

<terminated> Vetores [Java Application] C:\Program Files\Java\jre-1.8\bin\javaw.exe (01/04/2024 16:11:32 – 16:11:33)

1

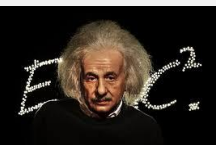


Java - Vetores ou Arrays Unidimensionais

Foi atribuído o valor 1 à variável primeiroNumero, porque 1 é o primeiro elemento do vetor números.

Em resumo, vetores em Java são como prateleiras virtuais onde você pode armazenar uma coleção ordenada de elementos do mesmo tipo.

Eles são úteis quando você precisa lidar com múltiplos valores semelhantes e quer manter seu código organizado e fácil de gerenciar.



Java - Vetores ou Arrays Unidimensionais

Inicializa um array de inteiros com tamanho 5

```
int[] numero = new int[5];  
System.out.println(numero);
```

inicializando Array

```
int[] numeros = new int[5]; // Inicializa um array de inteiros com tamanho 5
```

Atribuindo valores Array

```
numeros[0] = 10;  
numeros[1] = 20;  
numeros[2] = 30;  
numeros[3] = 40;  
numeros[4] = 50;
```

Iterando sobre um Array

```
for (int i = 0; i < numeros.length; i++) {  
    System.out.println(numeros[i]);  
}
```

inicialização de Array com Valores iniciais

```
int[] primos = { 2, 3, 5, 7, 11 }; // Inicializa um array com valores iniciais
```

Copiando Arrays

```
int[] copia = Arrays.copyOf(numeros, numeros.length); // Copia o array "numeros" para "copia"
```

ordenando um Array

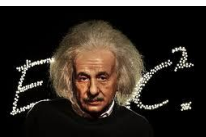
```
Arrays.sort(numeros); // Ordena o array "numeros"
```

Pesquisando e ordenando um Array

```
int indice = Arrays.binarySearch(numeros, 30); // Procura o valor 30 no array ordenado "numeros"
```

Verificando se um Valor Existe em um Array Não Ordenado

```
boolean existe = Arrays.asList(numeros).contains(30); // Verifica se o valor 30 existe no array
```



Java - Vetores ou Arrays Unidimensionais

6

Criação

Operação sobre Vetores	Exemplo de Código
Declaração de um vetor.	<code>tipo nomeVetor[];</code>
Alocação de espaço de um vetor.	<code>nomeVetor = new tipo[tamanho];</code>
Declaração e alocação de um vetor.	<code>tipo nomeVetor[] = new tipo[tamanho];</code>
Acesso de uma posição do vetor.	<code>nomeVetor[indice]</code>
Atribuição de um valor a uma posição.	<code>nomeVetor[indice] = valor;</code>
Acesso ao tamanho de um vetor.	<code>nomeVetor.length</code>



Java - Vetores ou Arrays Unidimensionais

7

Exemplo - Vetor

```
1  int vet[] = new int[5];
2  vet[0] = 6;
3  vet[1] = 3;
4  vet[2] = 7;
5  vet[3] = 4;
6  vet[4] = 2;
7  for (int i = 0; i < vet.length; i++) {
8      System.out.println("Array[" + i + "]: " + vet[i]);
9  }
```



Java - Vetores ou Arrays Unidimensionais

8

Formas de criar e inicializar

Tipo do Vetor	Exemplos usando a forma 1	Exemplos usando a forma 2
Inteiro	<code>int vet[] = {1, 5, 3, 4};</code>	<code>int[] vet = {1, 5, 3, 4};</code>
Real	<code>double vet[]={2.0, 5.3, 3.8};</code>	<code>double[] vet={2.0, 5.3, 3.8};</code>
Caractere	<code>char vet[] = {'a', 'e', 'i'};</code>	<code>char[] vet = {'a', 'e', 'i'};</code>



Java - Vetores ou Arrays Unidimensionais

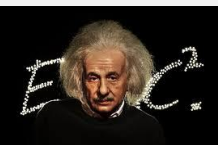
```
Arrays.java  Arrays.java x
1 package explicacao.arrays.unidimensionais;
2
3 public class Arrays {
4
5     public static void main(String[] args) {
6         // Declarando e inicializando um array unidimensional de inteiros
7         int[] numeros = new int[5];
8
9         // Preenchendo o array com alguns valores
10        for (int i = 0; i < numeros.length; i++) {
11            numeros[i] = i * 2; // Preenchendo com o dobro do índice
12        }
13
14        // Exibindo os valores do array
15        System.out.println("Array:");
16        for (int i = 0; i < numeros.length; i++) {
17            System.out.print(numeros[i] + " ");
18        }
19    }
20
...
<
Problems  @ Javadoc  Declaration  Console x
<terminated> Arrays [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 20:43:15 – 20:43:16)
Array:
0 2 4 6 8
```



Java - Arrays Multidimensionais

Arrays multidimensionais em Java são arrays que contêm outros arrays como elementos.

Isso permite representar dados em várias dimensões, como matrizes ou tabelas.

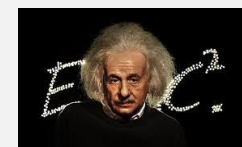


Java - Arrays Multidimensionais

Vamos começar com um exemplo simples de uma matriz bidimensional:

Neste exemplo, matriz é um array bidimensional de inteiros com dimensões 3x3. Isso significa que temos três linhas e três colunas.

Para inicializar e acessar elementos de uma matriz bidimensional, podemos fazer o seguinte:



Java - Arrays Multidimensionais

```
Arrays.java
1 package explicacao.arrays.multidimensionais;
2
3 public class Arrays {
4
5     public static void main(String[] args) {
6
7         int[][] matriz = new int[3][3];
8
9         // Inicializando a matriz
10        matriz[0][0] = 1;
11        matriz[0][1] = 2;
12        matriz[0][2] = 3;
13        matriz[1][0] = 4;
14        matriz[1][1] = 5;
15        matriz[1][2] = 6;
16        matriz[2][0] = 7;
17        matriz[2][1] = 8;
18        matriz[2][2] = 9;
19
20        // Acessando elementos da matriz
21        int primeiroElemento = matriz[0][0]; // Retorna 1
22        System.out.println(primeiroElemento);
23
24        int segundoElemento = matriz[1][2]; // Retorna 6
25        System.out.println(segundoElemento);
26    }
27 }
```

Problems Javadoc Declaration Console

<terminated> Arrays (1) [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 20:35:04 – 20:35:04)

```
1
6
```

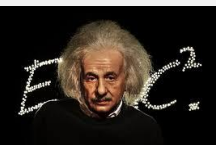


Java - Arrays Multidimensionais

Podemos pensar em uma matriz bidimensional como uma tabela com linhas e colunas, onde cada elemento pode ser acessado especificando o índice da linha e da coluna.

Além de matrizes bidimensionais, é possível ter arrays multidimensionais com mais dimensões. Por exemplo, uma matriz tridimensional seria declarada assim:

```
int[][][] cubo = new int[3][3][3];
```

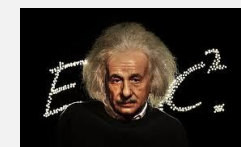


Java - Arrays Multidimensionais

—
Isso cria um cubo com 3x3x3 elementos.

A ideia é a mesma, podemos acessar cada elemento especificando os índices de cada dimensão.

Em resumo, os arrays multidimensionais em Java são úteis para representar estruturas de dados mais complexas, como matrizes, tabelas ou volumes, e podem ser acessados e manipulados de maneira semelhante aos arrays unidimensionais

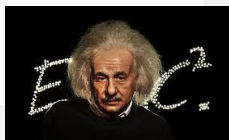


Java - Arrays Multidimensionais

```
Arrays.java  Arrays.java
1 package explicacao.arrays.multidimensionais;
2
3 public class Arrays {
4
5     /** EXEMPLO DE ARRAY MULTIDOMENSIONAL */
6     public static void main(String[] args) {
7         // Declarando e inicializando uma matriz bidimensional de inteiros
8         int[][] matriz = new int[3][3];
9
10        // Preenchendo a matriz com alguns valores
11        for (int i = 0; i < matriz.length; i++) {
12            for (int j = 0; j < matriz[i].length; j++) {
13                matriz[i][j] = i + j; // Preenchendo com a soma dos índices da linha e da coluna
14            }
15        }
16
17        // Exibindo os valores da matriz
18        System.out.println("Matriz:");
19        for (int i = 0; i < matriz.length; i++) {
20            for (int j = 0; j < matriz[i].length; j++) {
21                System.out.print(matriz[i][j] + " ");
22            }
23            System.out.println(); // Adicionando uma quebra de linha após cada linha da matriz
24        }
25    }
26 }
```

<terminated> Arrays (1) [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 20:40:56 – 20:40:56)

Matriz:
0 1 2
1 2 3
2 3 4



Java - Exemplo - Matriz

11

Exemplo - Matriz

```
1 double mat[][] = {{1.5, 5.2}, {3.6, 4.9}, {2.4, 8.1}};  
2 for (int i = 0; i < mat.length; i++) {  
3     for (int j = 0; j < mat[i].length; j++) {  
4         System.out.println("M[" + i + "][" + j + "]: " + mat[i][j]);  
5     }  
6 }
```



Java - Exemplo - Utilizando arrays

12

Exemplo - Utilizando arrays

```
1 package sp1_aula4;
2
3 public class SP1_aula4 {
4
5     public static void main(String[] args) {
6
7
8         String[] vet = new String[]{"leonardo", "João", "Maria"};
9         System.out.println("Os nomes informados foram: ");
10        for(int cont = 0; cont < vet.length; cont++)
11        {
12            System.out.println(vet[cont]);
13        }
14    }
15 }
```

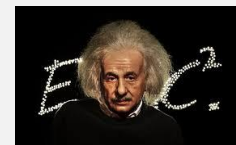


Java - Strings

A classe String possui uma importante característica, que é ser imutável.

Isso quer dizer que uma vez atribuído um valor literal para a variável, existirá uma memória, só que sem nenhum objeto apontando para si.

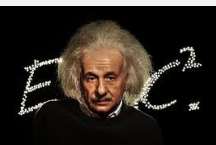
Assim, se você construir uma aplicação que modifique constantemente o valor da variável, você terá um desempenho ruim em termos de gastos de memória e de processamento.



Java – StringBuilder e StringBuffer

StringBuilder - Outra classe que existe para manipular strings e é uma boa alternativa à classe String se desejar que o conteúdo da string seja mutável.

StringBuffer - é uma boa alternativa às outras duas classes se desejar que o conteúdo seja mutável e necessitar de uma aplicação thread safe.



Java – StringBuilder e StringBuffer

16

Exemplo

```
1 String str1 = new String("Orientação a objetos");
2 String str2 = "Orientação a objetos";
3 StringBuilder str3 = new StringBuilder("Orientação a objetos");
4 StringBuilder str4 = "Orientação a objetos"; //não é aceito (dá
  erro)
5 StringBuffer str5 = new StringBuffer("Orientação a objetos");
6 StringBuffer str6 = "Orientação a objetos"; //não é aceito (dá
  erro)
```

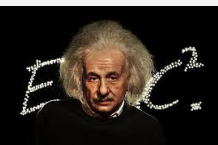


Java - Heap x String Pool

Heap -> O Heap é uma região de memória onde objetos são alocados durante a execução de um programa Java.

É uma área de memória dinâmica que é compartilhada por todas as partes do programa.

Os objetos criados dinamicamente usando a palavra-chave `new` são alocados no heap.

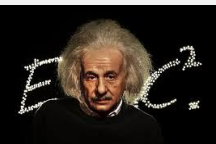


Java - Heap x String Pool

String Pool -> O String Pool é uma área específica da memória do heap que armazena literais de string.

Quando você cria uma string literal em seu código Java (por exemplo, `String str = "hello";`), ela é armazenada no String Pool.

O String Pool é uma otimização na JVM para economizar espaço de memória, reutilizando strings idênticas em vez de criar novas instâncias.



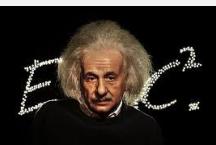
Java - Heap x String Pool

Aqui está uma explicação mais detalhada sobre a diferença entre o Heap e o String Pool:

Quando você cria uma string literal, como "hello", o Java verifica se uma string com esse valor já existe no String Pool.

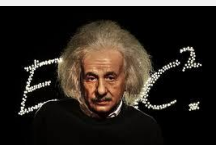
Se a string já existe no String Pool, uma nova referência para essa string é retornada.

Se a string não existe no String Pool, uma nova string é criada e adicionada ao String Pool.



Java - Heap x String Pool

—
Por outro lado, quando você cria uma string usando a palavra-chave new, como `String str = new String("hello");`, uma nova instância de string é sempre criada no heap, independentemente de já existir uma string com o mesmo valor no String Pool.

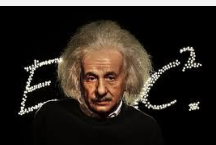


Java - Heap x String Pool

—
Portanto, mesmo que duas strings tenham o mesmo valor, elas podem estar localizadas em diferentes áreas de memória:

Se ambas as strings forem criadas como literais de string, elas serão armazenadas no String Pool.

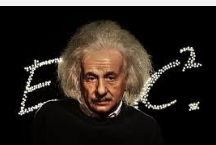
Se pelo menos uma delas for criada usando new, ela será armazenada no heap, mesmo que o valor já exista no String Pool.



Java - Heap x String Pool

—
Em suma,

- o String Pool é uma parte do heap que armazena literais de string para otimizar o uso de memória, enquanto
- o heap é a área geral de memória onde todos os objetos são alocados durante a execução do programa Java.



Java - Heap x String Pool

```
HeapStringPool.java
1 package explicacao.HeapStringPool;
2
3 public class HeapStringPool {
4     public static void main(String[] args) {
5         // Criando strings literais
6         String str1 = "hello";
7         String str2 = "hello";
8
9         // Criando strings usando a palavra-chave 'new'
10        String str3 = new String("hello");
11        String str4 = new String("hello");
12
13        // Verificando se as strings literais estão no mesmo local na memória (String
14        // Pool)
15        System.out.println("str1 e str2 referenciam o mesmo objeto? " + (str1 == str2)); // Deve imprimir true
16
17        // Verificando se as strings criadas com 'new' estão no mesmo local na memória
18        // (heap)
19        System.out.println("str3 e str4 referenciam o mesmo objeto? " + (str3 == str4)); // Deve imprimir false
20
21        /**
22         Neste exemplo:
23         str1 e str2 são strings literais e, portanto, são armazenadas no String Pool.
24         Como são iguais, elas referenciam o mesmo objeto no String Pool.
25
26         str3 e str4 são criadas usando a palavra-chave new, portanto,
27         mesmo que tenham o mesmo valor "hello", elas são instâncias separadas de objetos no heap.
28         Portanto, elas não referenciam o mesmo objeto.
29         */
30    }
31 }
32
```

Problems Javadoc Declaration Console

<terminated> HeapStringPool [Java Application] C:\Program Files\RedHat\java-1.8.0-openjdk-1.8.0.212-3\bin\javaw.exe (01/04/2024 20:56:28 – 20:56:29)

str1 e str2 referenciam o mesmo objeto? true
str3 e str4 referenciam o mesmo objeto? false

