

Programação Orientada à Objetos

POO

Objetos

Os objetos são a chave para a compreensão da tecnologia orientada a objeto. Olhe em volta agora e você vai encontrar muitos exemplos de objetos do mundo real: seu cachorro, sua mesa, sua televisão.

Os objetos do mundo real partilham duas características: eles possuem **estado e comportamento**.

POO

Objetos

Os cães têm estado (nome, cor, raça) e comportamento (latidos, cheirando, abanando o rabo).

Bicicletas também têm estado (marcha atual, velocidade atual) e comportamento (mudança de velocidade, alterando a marcha, freando).

Identificar o estado e o comportamento de objetos do mundo real é uma ótima maneira de começar a pensar em termos de programação orientada a objeto.

POO

Classes

No mundo real, muitas vezes você vai encontrar objetos que são parecidos, ou que pertencem à mesma espécie. Por exemplo, um cão da raça **labrador** e um cão da raça **bulldog** são diferentes na aparência, mas ambos são cães.

POO

Classes

Podemos ter uma bicicleta de corrida, e outra infantil, mas ambas são bicicletas e partilham características comuns.

Na programação orientada a objetos, uma classe é um modelo que contém a especificação de um objeto, ou seja, toda a lista de características e ações possíveis desse objeto.

POO

Classes

A **Classe** determina os atributos e métodos dos objetos que serão instanciados (criados).

POO

Classes

Os **atributos** são as características que os objetos terão. Ex.: modelo, marca, ano, cor, etc.

Os **métodos** são as ações que os objetos irão executar ou sofrer. Ex.: A pessoa tem a ação de andar, falar, comer, o carro sofre a ação de ser ligado, desligado, acelerado. Portanto, a pessoa teria os métodos: andar(), falar(), comer(), ligar(), desligar().

POO

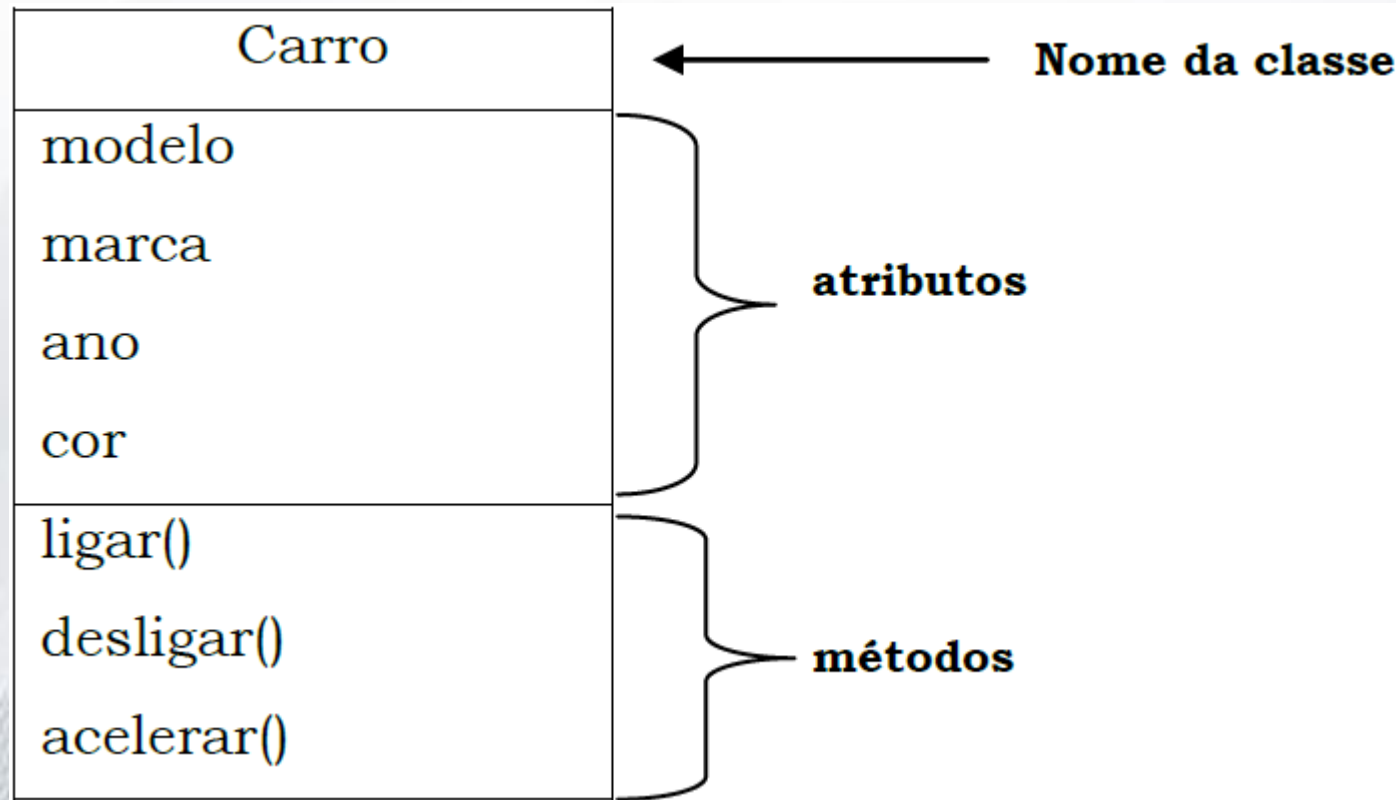
Diagramas UML (Unified Modeling Language)

Modelo UML é um modelo universal da linguagem, serve como um projeto, escopo do nosso programa, através dele se representa como serão os objetos do sistema.

Exemplo: uma classe Carro, sendo que todo carro possui um modelo, marca, ano e cor, o mesmo pode ser ligado, desligado, acelerado e freado.

POO

Diagramas UML (Unified Modeling Language)



Atributos

Ao modelar uma classe, listamos todas as características que os objetos criados a partir dela irão possuir e que são relevantes para nosso problema. Essas características do objeto que estamos modelando recebem o nome de “atributos”.

Atributos

Na Programação Orientada a Objetos, um atributo é uma variável, ou seja, um espaço reservado na memória do computador para guardar um valor temporariamente. É chamada de variável porque seu valor pode mudar durante sua existência.

Atributos

Cada atributo representa um dado que compõe o estado de um objeto. Como estamos lidando com dados digitais, o computador precisa saber como armazenar cada dado. Por isso existem as definições de tipos de dados, que variam de linguagem para linguagem.

Ao definir um atributo de uma classe, é necessário especificar o tipo de dado que ele armazenará.

Atributos

Aluno
nome:String endereco:String ra:String idade:int nota:double
acessarPortal() visualizarNotas()

Métodos

- Tipos de métodos
 - Métodos de ação (sem retorno) – void
 - Métodos de retorno – double, int, byte, String, boolean
- Argumentos
 - Métodos com ou sem argumento
 - Passagem por Parâmetro

Métodos

- Métodos são blocos de código que pertencem a uma classe e tem por finalidade realizar uma tarefa. Ou seja, são as ações, comportamentos que nossos objetos poderão ter. Métodos podem ou não alterar o estado (dados) de um objeto.

Métodos

Tipos de Métodos

- Métodos de ação (sem retorno) - void

Apenas realizam a ação sem dar nenhum resultado. A palavra “**void**” significa ausência de retorno.

Exemplo: sentar(), levantar(), etc.

Métodos

Tipos de Métodos

- Métodos de retorno – double, int, byte, String, boolean

Realizam a ação e ao final retornam um valor de resposta.

Exemplo: verificarStatus(), calcularMedia(), etc.

Os métodos podem possuir ou não argumentos.

Métodos

Argumentos com ou sem argumento

São os dados adicionais que o método requer para realizar a sua tarefa. Por exemplo, para uma pessoa realizar a ação “andar”, é necessário informar a quantidade de passos e a direção. Para uma pessoa girar, é necessário indicar a direção e os graus.

Em alguns casos o método pode não ter argumento. Nesse caso, apenas utilizamos um conjunto vazio de parênteses.

Métodos

Argumentos com ou sem argumento

Para cada argumento devemos especificar o tipo de dado, assim como nos atributos.

Ex: andar(passos:int, direcao:String).

Métodos

Argumentos com ou sem argumento

Exemplo: Imagine o seguinte: Se eu falar para uma pessoa sentar, não preciso falar mais nada, isso basta, ela sabe que é para sentar-se. Agora se eu falar para andar, a pessoa provavelmente perguntará para onde. A direção seria o argumento, para que a pessoa execute a ação, ela precisa de mais informações além da própria ação imposta.

Métodos

Passagem por Parâmetro

A passagem por parâmetro, nada mais é, do que o argumento do método, ou seja, o que deverá ser informado ao método para que o mesmo seja executado.

Exemplo: `andar(direção:String,passos:int):void`

O exemplo acima é um método sem retorno e com argumento, temos dois parâmetros: direção e passos.

Métodos

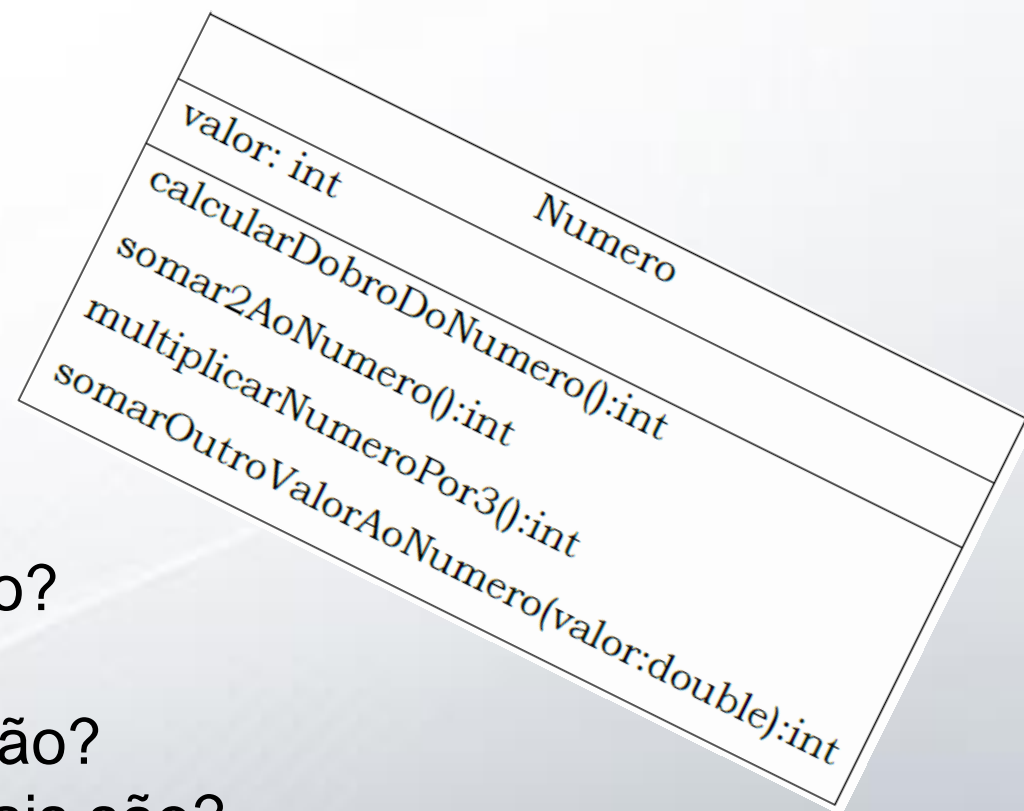
Exemplo de Métodos no Diagrama

ContaBancaria
numero:int agencia:int conta:String titular:String saldo:double
sacar(valor:double):void depositar(valor:double):void obterSaldo(): double

Métodos

Exemplo de Métodos no Diagrama

1. Qual o nome da classe?
2. Quantos atributos a classe possui? E quais são?
3. Quantos métodos a classe possui?
4. Quantos métodos possuem retorno? E quais são?
5. Quantos métodos não possuem retorno? E quais são?
6. Cite um método que possua argumento.
7. Qual o parâmetro passado pelo método que possui argumento?
8. Temos algum método sem retorno e com argumento? Qual?



Métodos

Como os métodos afetam os objetos

ContaBancaria
numeroDaConta:String saldo:double
sacar(valor:double):void depositar(valor:double):void obterSaldo(): double

Métodos

Como os métodos afetam os objetos

conta1(numeroDaConta=123-0,
saldo=100)

conta2(numeroDaConta=321-1,
saldo=500)

Métodos

Como os métodos afetam os objetos

conta1(numeroDaConta=123-0,
saldo=100)

conta2(numeroDaConta=321-1,
saldo=500)

`conta1.sacar(50)`

`conta2.depositar(200)`

`conta1.depositar(400)`

`conta2.sacar(150)`

`conta1.sacar(20)`

`conta2.depositar(120)`

Métodos

Como os métodos afetam os objetos

$$100 - 50 + 400 - 20 = 430$$

Métodos

Como os métodos afetam os objetos

$$500 + 200 - 150 + 120 = 670$$

Métodos

Como os métodos afetam os objetos

conta1(numeroDaConta=123-0,
saldo=430)

conta2(numeroDaConta=321-1,
saldo=670)

Diagrama de UML

Pessoa
+nome:String +idade:int +salario:double
+falar(texto:String):void +andar(passos:int):void

Diagrama de UML

Sintaxe JAVA

```
public class NomeClasse {  
  
}
```

Diagrama de UML

Sintaxe JAVA

```
public class NomeClasse {  
  
}
```

As chaves funcionam como delimitadores de bloco. Elas determinam o início e o fim da classe.

Diagrama de UML

Atributos

Em Java, o sinal de + na frente do atributo ou método se transforma na palavra “*public*”. O tipo de dado fica na frente do nome do atributo separados por espaço, e o final da linha o “;” que determina o fim do comando.

Diagrama de UML

Atributos

Em Java, o sinal de + na frente do atributo ou método se transforma na palavra “*public*”. O tipo de dado fica na frente do nome do atributo separados por espaço, e o final da linha o “;” que determina o fim do comando.

No diagrama	Na linguagem Java
+nome:String	public String nome;
+idade:int	public int idade;
+salario:double	public double salario;

Diagrama de UML

Métodos

No diagrama	Na linguagem Java
+falar(texto:String):void	<pre>public void falar(String texto){ //comandos }</pre>
+andar(passos:int):void	<pre>public void andar(int passos){ //comandos }</pre>

Operações aritméticas

Básicos

Operador	Representação algorítmica	Notação para Java	Descrição para Java
Adição	+	+	Adiciona dois números ou variáveis: $a + b$.
Subtração	-	-	Subtrai dois números ou variáveis: $a - b$.
Multiplicação	*	*	Multiplica dois números ou variáveis: $a * b$.
Divisão	/	/	Divide dois números ou variáveis: a / b .
Módulo	mod	%	Retorna o resto da divisão de dois números ou variáveis: $a \% b$.

Operações aritméticas

Básicas

Ordem de precedência:

1º ()

2º * (multiplicação) / (divisão) % (módulo – resto da divisão)

3º + (adição) - (subtração)

Obs.: O módulo é utilizado somente com números inteiros.

Operações aritméticas

Juros e Descontos

Para calcularmos o valor de juros, basta pegarmos o valor e multiplicá-lo pelo percentual de juros, porém no programa não poderemos utilizar o símbolo “%” para representar percentuais, pois “%” significa módulo.

Exemplo:

10% em Java se converte para 0.10

5% em Java se converte para 0.05

Operações aritméticas

Juros e Descontos

Exemplo: Um produto custa 100 reais. Teremos 10% de juros. Qual o valor do juro? Qual o valor final do produto?

No papel:

juros=100x10% -----

valorFinal=100+(100x10%) -----

Em Java:

double juros=100*0.10;

double valorFinal=100+(100*0.10);

Operações aritméticas

Juros e Descontos

No papel:

juros=100x10% -----
valorFinal=100+(100x10%) -----

Em Java:

double juros=100*0.10;
double valorFinal=100*1.10;

Operações aritméticas

Desconto / valor com desconto (valor à vista)

Para calcularmos o valor já com desconto, basta pegarmos o valor e subtraí-lo do valor multiplicado pelo percentual de desconto. Sempre cuidando para converter este percentual em um número decimal.

Operações aritméticas

Desconto / valor com desconto (valor à vista)

Exemplo: Um produto custa 100 reais. Teremos 10% de desconto. Qual o valor do desconto? Qual o valor final do produto?

No papel:

$\text{desconto} = 100 \times 10\%$ -----

$\text{valorFinal} = 100 - (100 \times 10\%)$ -----

Em Java:

`double desconto = 100 * 0.10;`

`double valorFinal = 100 - (100 * 0.10);`

Operações aritméticas

Desconto / valor com desconto (valor à vista)

No papel:

desconto=100x10% -----
valorFinal=100x90%) -----

Em Java:

double desconto=100*0.10;
double valorFinal=100*0.90);

Operações aritméticas

Desconto / valor com desconto (valor à vista)

Exemplos:

a) Produto R\$ 200,00 com juros de 20%

$$\text{juros} = 200 * 0.20;$$

$$\text{valorFinalComJuros} = 200 + 200 * 0.20;$$

ou

$$\text{valorFinalComJuros} = 200 * 1.20;$$

Operações aritméticas

Médias

Média aritmética

A média aritmética é o resultado da soma de todos os valores, dividido pela quantidade de valores.

a) Média aritmética com 4 notas: 17, 8, 4 e 6

`double mediaAritmetica = (17 + 8 + 4 + 6)/4;`

Operações aritméticas

Médias

Médias Ponderadas

A média ponderada é o resultado da soma do primeiro valor multiplicada pelo seu peso, com o segundo valor multiplicado pelo seu peso e terceiro valor multiplicado pelo seu peso. Tudo isso dividido pela soma de todos os pesos.

Operações aritméticas

Médias

Médias Ponderadas

No papel:

$$\text{Média ponderada} = \frac{8*1 + 2*2}{(1 + 2)}$$

Em Java:

```
double mediapond = (8*1 + 2*2)/(1+2);
```

Operações aritméticas

Médias

Médias Ponderadas

a) Média aritmética com 4 notas e 4 pesos:

NOTAS: 17, 8, 4 e 6

PESOS: 2, 2, 3, 3

`double mediaPonderada = (17*2 + 8*2 + 4*3 + 6*3) / (2+2+3+3);`

Funções e constantes matemáticas

No Java existe uma classe chamada ***Math*** que contém várias funções matemáticas que podemos utilizar em nossos programas. Há três funções que utilizamos com mais frequência: Potência, Raiz Quadrada e Raiz Cúbica.

A utilização de Raiz ou Potência retorna um tipo de dado double.

Funções e constantes matemáticas

Raiz Quadrada

Para calcularmos a raiz quadrada basta informar o número como argumento para o método sqrt que está na classe Math.

`Math.sqrt(numero)`

No papel:

$\sqrt{49}$

Em Java:

`double raiz = Math.sqrt(49);`

Funções e constantes matemáticas

Raiz Cúbica

Para calcularmos a raiz cúbica basta informar o número como argumento para o método `cbirt` que está na classe `Math`.

`Math.cbirt(numero)`

No papel:

$$\sqrt[3]{8}$$

Em Java:

```
double raiz = Math.cbirt(8);
```

Funções e constantes matemáticas

Potência

Para calcularmos a potência de um número basta informar a base (o número propriamente dito) e o expoente como argumento para o método pow que está na classe Math. `Math.pow(base,expoente)`

No papel:

$$2^3$$

Em Java:

```
double potencia = Math.pow(2,3);
```

Funções e constantes matemáticas

Pi

Sempre que precisarmos utilizar o valor do PI em alguma operação matemática em Java, utilizamos a constante PI da classe Math.

No papel:

π

Em Java:

`Math.PI;`

Inserindo operações nos métodos

A fórmula pode utilizar apenas os atributos da classe, os argumentos do método e/ou valores fixos para composição do cálculo.

Inserindo operações nos métodos

A fórmula pode utilizar apenas os atributos da classe, os argumentos do método e/ou valores fixos para composição do cálculo.

Funcionario
+valorHora:double
+cargaHoraria:double
+dobrarValorHora():void
+calcularSalario():double
+aumentarCargaHoraria(horas:double):void

Inserindo operações nos métodos

Como fica o código Java baseado no diagrama

```
1 package testefuncionario;
2
3 public class Funcionario {
4
5     public double valorHora;
6     public double cargaHoraria;
7
8
9     public void dobrarValorHora() {
10         this.valorHora = this.valorHora * 2;
11     } //fim método dobrarValorHora
12
13
14     public double calcularSalario() {
15         return this.valorHora * this.cargaHoraria;
16     } //fim método calcularSalario
17
18
19     public void aumentarCargaHoraria(double horas) {
20         this.cargaHoraria = this.cargaHoraria + horas;
21     } //fim método aumentarCargaHoraria
22 } //fim classe Funcionario
```


Inserindo operações nos métodos

Como fica o código Java baseado no diagrama

Observe que ao compor uma fórmula, evitamos usar números, pois não sabemos ainda qual o valor que cada atributo vai armazenar. Assim, a fórmula deve funcionar para qualquer valor que esteja armazenado nos atributos ou argumentos.

Uso do *this*

‘*This*’ significa ‘este’: utilizamos o *this* para representar o uso de um atributo ou método da classe. Em Java isso é uma convenção dentro do código da classe, sempre que utilizar um atributo dentro de um método, ou um método dentro de outro método, usar o *this* na frente do nome.

Uso do this

Entendendo os métodos da classe criada

O sinal de “=” equivale a “recebe” ou “passa a valer”. É o sinal de **atribuição**. O comando acima é lido da seguinte forma:

```
public void dobrarValorHora() {  
    this.valorHora = this.valorHora * 2;  
}
```

Atributo que será atualizado

cálculo

Uso do this

Entendendo os métodos da classe criada

*O atributo valorHora **recebe** o resultado do cálculo do valor atual do atributo **valorHora multiplicado por 2.***

Uso do this

Entendendo os métodos da classe criada

```
public double calcularSalario() {  
    return this.valorHora * this.cargaHoraria;  
}
```

Cálculo que o método deve retornar

O comando **return** indica que o que vem a seguir deve ser a resposta do método quando este for executado. Aqui não se usa o sinal de “=”. O comando acima é lido da seguinte forma:

Uso do this

Entendendo os métodos da classe criada

***Retornar** o resultado do cálculo do atributo **valorHora** multiplicado pelo valor do atributo **cargaHoraria**.*

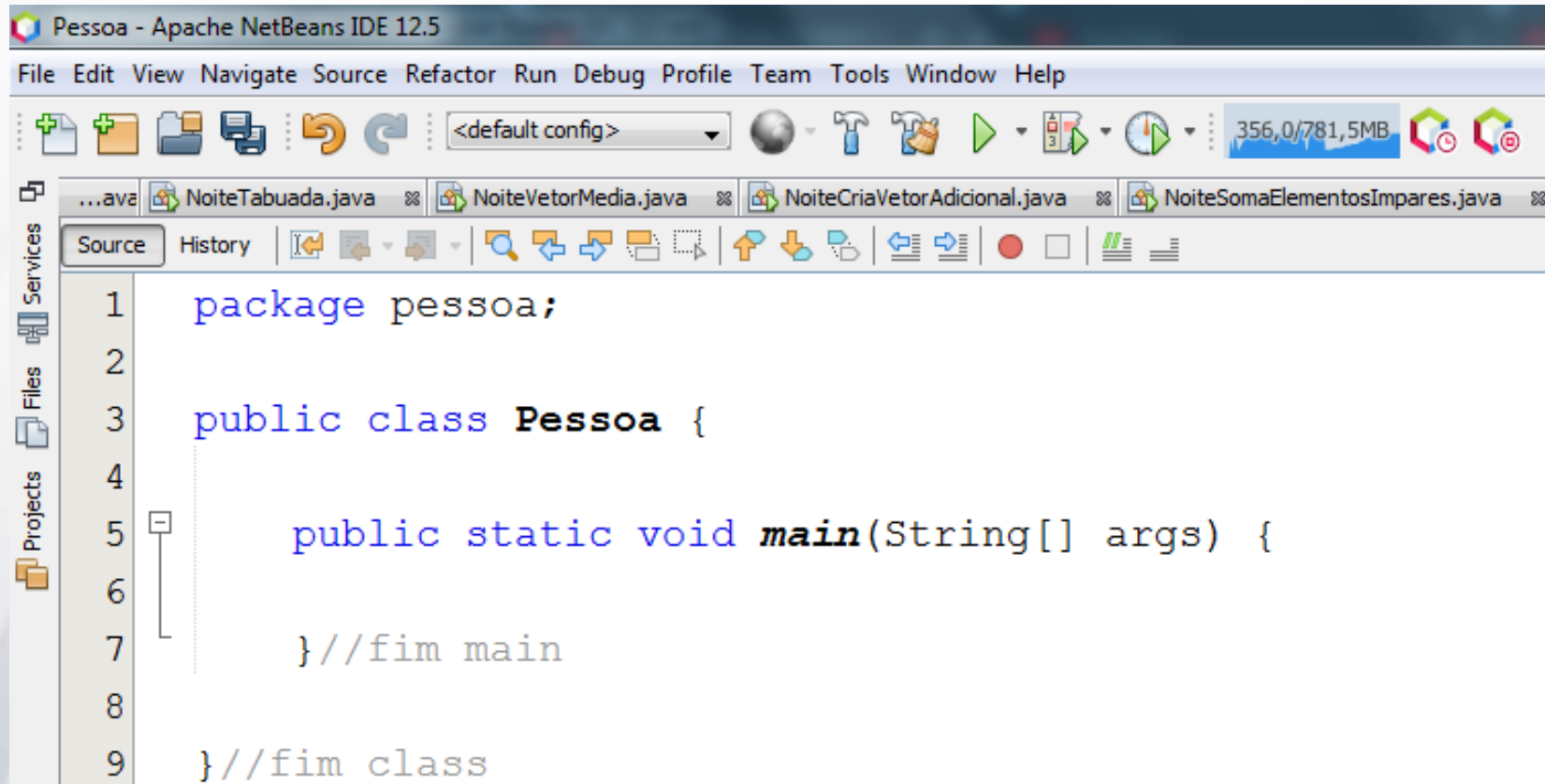
```
public void aumentarCargaHoraria(double horas) {  
    this.cargaHoraria = this.cargaHoraria + horas;  
}
```

Uso do this

Entendendo os métodos da classe criada

*O atributo **cargaHoraria** recebe o resultado do cálculo do valor atual do atributo **cargaHoraria** somado ao valor do argumento **horas**.*

Classe Main



```
1 package pessoa;
2
3 public class Pessoa {
4
5     public static void main(String[] args) {
6
7     } //fim main
8
9 } //fim class
```


Classe Main

Entendendo os métodos da classe criada

Toda a classe Principal possuirá uma linha com o seguinte comando:

```
public static void main(String args[]){  
}
```

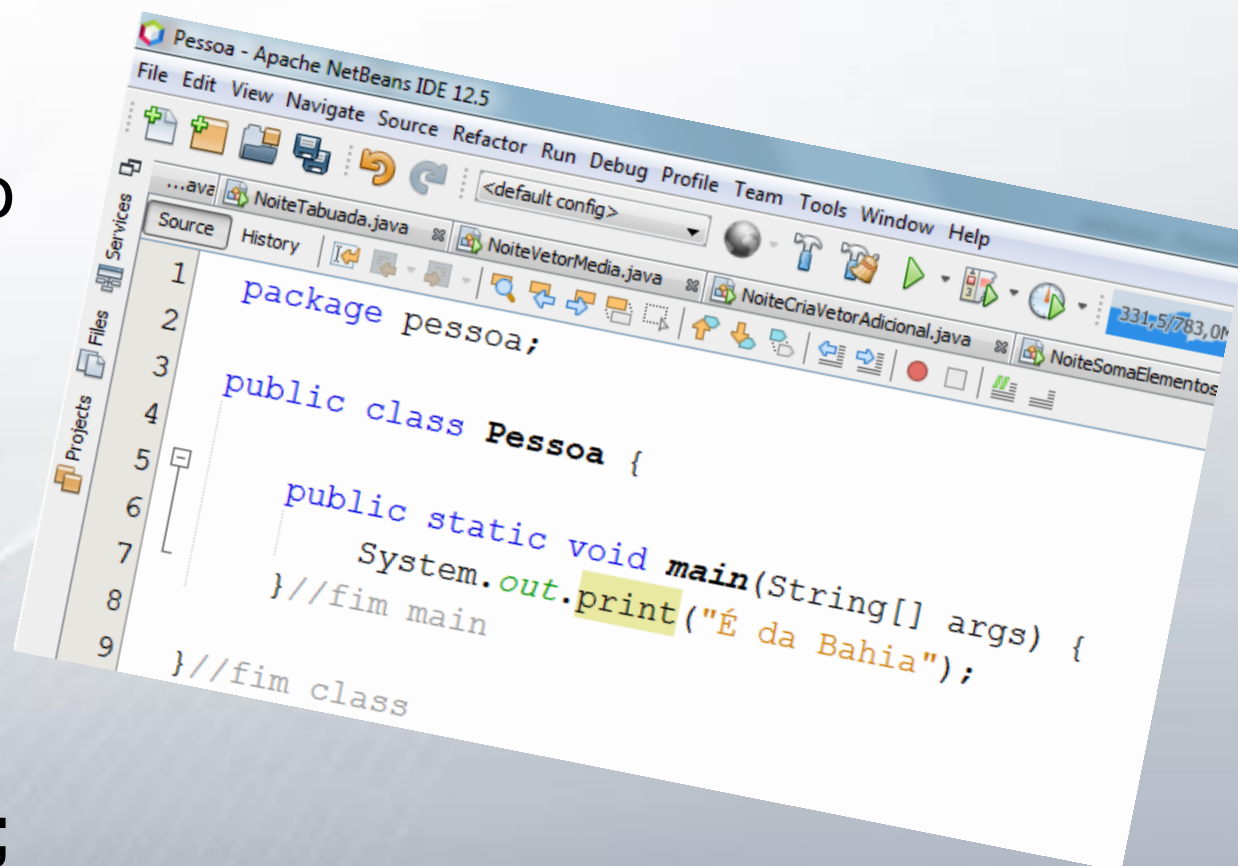
Este comando especifica que a classe é estática, sem retorno e principal.

Classe Main

Mostrando na tela

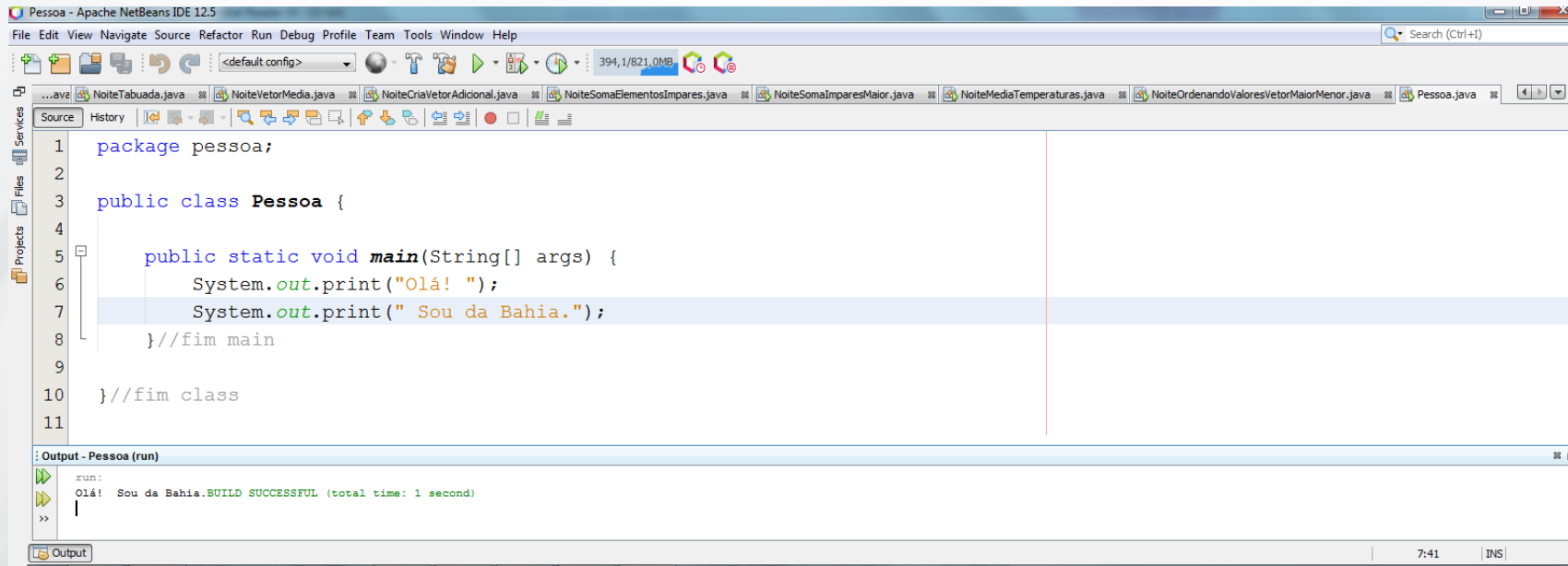
É possível mostrar mensagens ao usuário através de um método de saída de dados.

O método **out**, é da classe do sistema, **class System**, portanto o comando fica:
System.out.print("mensagem");



Classe Main

Mostrando na tela



The screenshot shows the Apache NetBeans IDE 12.5 interface. The main editor window displays the source code for the 'Pessoa' class. The code is as follows:

```
1 package pessoa;
2
3 public class Pessoa {
4
5     public static void main(String[] args) {
6         System.out.print("Olá! ");
7         System.out.print(" Sou da Bahia.");
8     } //fim main
9
10 } //fim class
11
```

Below the editor, the 'Output - Pessoa (run)' window shows the execution results:

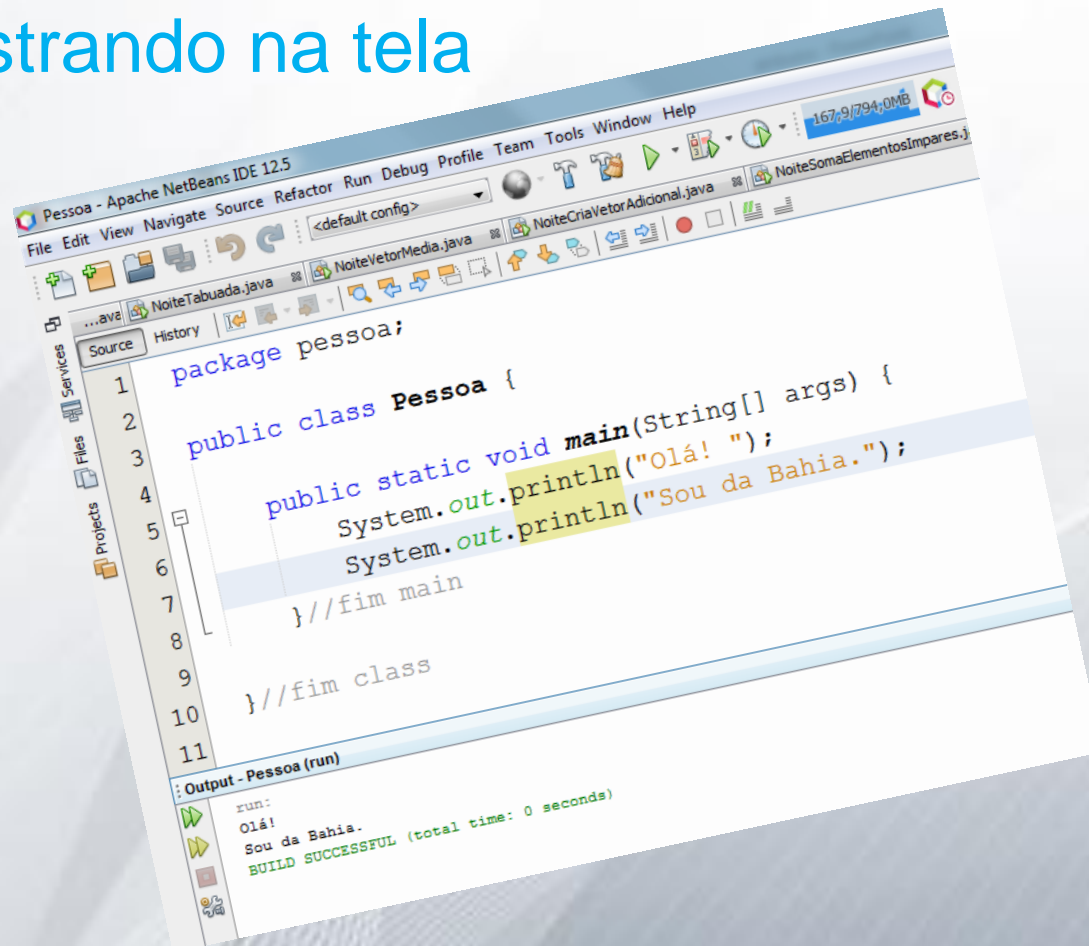
```
run:
Olá! Sou da Bahia.BUILD SUCCESSFUL (total time: 1 second)
```

The output demonstrates that the two print statements are concatenated without a space, resulting in 'Olá! Sou da Bahia.'.

Ao utilizar o método de saída com “**print**”, o final da mensagem ficará junto com a próxima mensagem.

Classe Main

Mostrando na tela



Já com o método de saída com “***println***”, a próxima mensagem ficará abaixo da anterior.

Classe Main

Mostrando na tela

```
1 package pessoa;
2
3 public class Pessoa {
4
5     public static void main(String[] args) {
6         System.out.println("Olá! ");
7         System.out.println("Sou da Bahia.");
8     } //fim main
9
10 } //fim class
11
```

Output - Pessoa (run)

```
run:
Olá!
Sou da Bahia.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Classe Main

Instanciando objetos

A **instância** de um objeto, nada mais do que a criação de um objeto a partir de uma classe, ou seja, **o objeto é a instância de uma classe**.

O objeto é quem armazena os dados e executa as ações, conforme elas estiverem definidas na classe.

Classe Main

Sintaxe para criar uma nova instância

NomeDaClasse nomeDoObjeto = new NomeDaClasse();

Exemplo:

Pessoa p1 = new Pessoa();

Classe Main

Acessando atributos dos objetos

Quando os atributos são públicos, temos acesso direto a eles. Logo, se quisermos definir os atributos do objeto p1 (nome, idade, salario), utilizamos a seguinte sintaxe:

nomeDoObjeto.nomeDoAtributo = valor;

Classe Main

Acessando atributos dos objetos

Exemplos:

```
p1.nome = "Carla";  
p1.idade = 47;  
p1.salario=15845.90;
```

Classe Main

Invocando métodos através do objeto

Invocar um método é chamá-lo, fazê-lo executar.

Funcionario
+nome:String
+salarioBase:double
+dobrarSalario():void
+calcularFerias():double
+descontarAdiantamento(valor:double):void
+calcularHorasExtras(totalDeHoras:double):double

Classe Main

Invocando métodos sem retorno e sem argumento

O nosso primeiro método, dobrarSalario(), não tem nem argumento, nem retorno.

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
f1.nome = "Paulo";  
f1.dobrarSalario();
```

Classe Main

Invocando métodos com retorno, mas sem argumento

O nosso segundo método, calcularFerias(), não tem argumento, mas tem retorno.

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
double ferias = f1.calcularFerias();
```

Classe Main

Invocando métodos sem argumento, mas com retorno

Segunda forma:

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
double ferias; //declara a variável  
ferias = f1.calcularFerias();
```

Classe Main

Invocando métodos sem retorno, mas com argumento

O terceiro método, descontarAdiantamento(valor:double) não tem retorno, mas precisa saber o valor do adiantamento para poder fazer o desconto no salário.

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
double valor = 600; //salvando o valor na variável  
f1.descontarAdiantamento(valor);
```

Classe Main

Invocando métodos sem retorno, mas com argumento

Segunda forma:

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
f1.descontarAdiantamento(600); //passando o valor diretamente
```

Classe Main

Invocando métodos com retorno e com argumento

O método, `calcularHorasExtras(totalDeHoras:double):double` tem retorno e também precisa do argumento que corresponde ao total de horas para calcular o valor das horas extras.

```
Funcionario f1 = new Funcionario();  
f1.salarioBase = 2500;  
double totalDeHoras = 15; //salvando o total de horas na variável  
double horasExtras = f1.calcularHorasExtras(totalDeHoras);
```

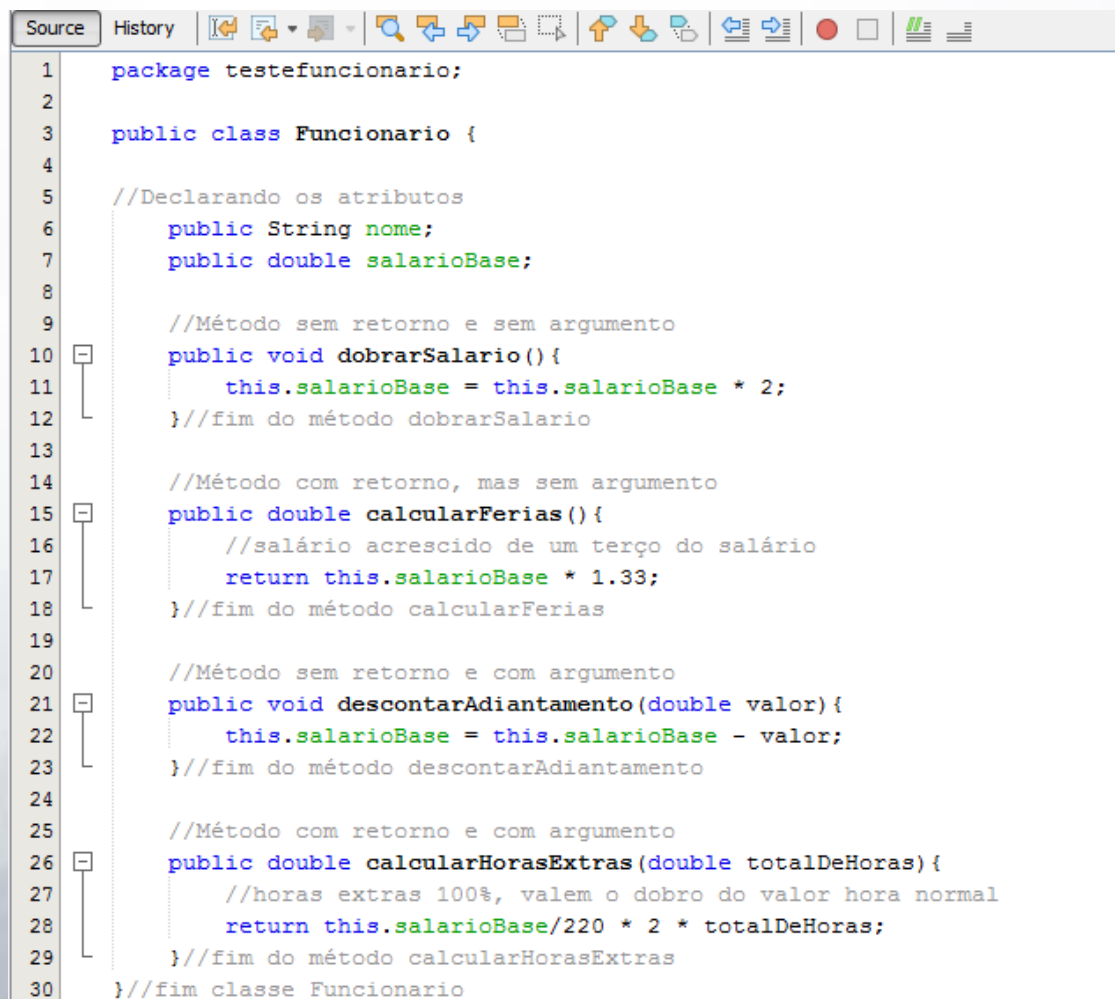

Classe Main

Exemplo completo

Funcionario
+nome:String
+salarioBase:double
+dobrarSalario():void
+calcularFerias():double
+descontarAdiantamento(valor:double):void
+calcularHorasExtras(totalDeHoras:double):double

Classe Main

Exemplo completo



```
Source History
1 package testefuncionario;
2
3 public class Funcionario {
4
5     //Declarando os atributos
6     public String nome;
7     public double salarioBase;
8
9     //Método sem retorno e sem argumento
10    public void dobrarSalario(){
11        this.salarioBase = this.salarioBase * 2;
12    } //fim do método dobrarSalario
13
14    //Método com retorno, mas sem argumento
15    public double calcularFerias(){
16        //salário acrescido de um terço do salário
17        return this.salarioBase * 1.33;
18    } //fim do método calcularFerias
19
20    //Método sem retorno e com argumento
21    public void descontarAdiantamento(double valor){
22        this.salarioBase = this.salarioBase - valor;
23    } //fim do método descontarAdiantamento
24
25    //Método com retorno e com argumento
26    public double calcularHorasExtras(double totalDeHoras){
27        //horas extras 100%, valem o dobro do valor hora normal
28        return this.salarioBase/220 * 2 * totalDeHoras;
29    } //fim do método calcularHorasExtras
30 } //fim classe Funcionario
```

Exemplo completo

Classe Main

```
1 package testefuncionario;
2
3 public class TesteFuncionario {
4
5     public static void main(String[] args) {
6
7         //Instanciando o objeto
8         Funcionario f1 = new Funcionario();
9
10        //Definindo os atributos
11        f1.nome = "Carla";
12        f1.salarioBase = 15845.90;
13
14        //Invocando seus métodos sem retorno e sem argumento
15        f1.dobrarSalario();
16
17        //Invocando seus métodos sem retorno e com argumento
18        f1.descontarAdiantamento(600);
19
20        //Invocando seus métodos com retorno e sem argumento
21        double ferias = f1.calcularFerias();
22
23        //Invocando seus métodos com retorno e com argumento
24        double horasExtras = f1.calcularHorasExtras(15);
25
26        //Exibindo informações na tela
27        System.out.println("Olá, " + f1.nome + ", seu salário é de R$ " + f1.salarioBase);
28        System.out.println("O valor de suas férias é de R$ " + ferias );
29        System.out.println("O valor de suas horas extras é R$ " + horasExtras );
30
31    } //fim método main
32
33 } //fim classe TesteFuncionario
```

Classe Main

Exemplo completo

```
10 // Declaração de variáveis
11 f1.nome = "Carla";
12 f1.salarioBase = 15845.90;
13
14 //Invocando seus métodos sem retorno e sem argumento
15 f1.dobrarSalario();
16
17 //Invocando seus métodos sem retorno e com argumento
18 f1.descontarAdiantamento(600);
19
20 //Invocando seus métodos com retorno e sem argumento
21 double ferias = f1.calcularFerias();
22
23 //Invocando seus métodos com retorno e com argumento
24 double horasExtras = f1.calcularHorasExtras(15);
25
26 //Exibindo informações na tela
27 System.out.println("Olá, " + f1.nome + ", seu salário é de R$ " + f1.salarioBase);
28 System.out.println("O valor de suas férias é de R$ " + ferias );
29 System.out.println("O valor de suas horas extras é R$ " + horasExtras );
30
31 } //fim método main
```

Output - TesteFuncionario (run)

```
run:
Olá, Carla, seu salário é de R$ 31091.8
O valor de suas férias é de R$ 41352.094000000005
O valor de suas horas extras é R$ 4239.790909090909
BUILD SUCCESSFUL (total time: 0 seconds)
```

Classe Scanner

Classe Scanner

Existem várias formas de fazermos uma leitura de dados, podemos utilizar a classe `JOptionPane`, na qual apresenta caixas de diálogos para usuário.

Porém, para trabalharmos no modo texto do terminal o ideal é a **classe `Scanner`**.

Classe Scanner

Importando a classe Scanner

Para lermos algo que o usuário irá digitar primeiro devemos importar a classe Scanner.

```
import java.util.Scanner;
```

Classe Scanner

Instanciando a classe Scanner

Para utilizarmos a classe Scanner precisamos instanciá-la:

```
Scanner ler = new Scanner(System.in);
```

ler é o nome que damos para o objeto.

System.in se refere a “entrada de sistema”.

Classe Scanner

Utilizando a classe Scanner para ler dados

Lendo String

next() → lê uma palavra;

nextLine() → lê uma linha de texto

```
p1.apelido = ler.next();  
p1.nome = ler.nextLine();
```

Classe Scanner

Utilizando a classe Scanner para ler dados

Lendo Números inteiros byte

nextByte() → lê um número inteiro até 127

```
p1.numeroDeFilhos = ler.nextByte();
```

Classe Scanner

Utilizando a classe Scanner para ler dados

Lendo Números inteiros int

nextInt() → lê um número inteiro

```
p1.idade = ler.nextInt();
```

Classe Scanner

Utilizando a classe Scanner para ler dados

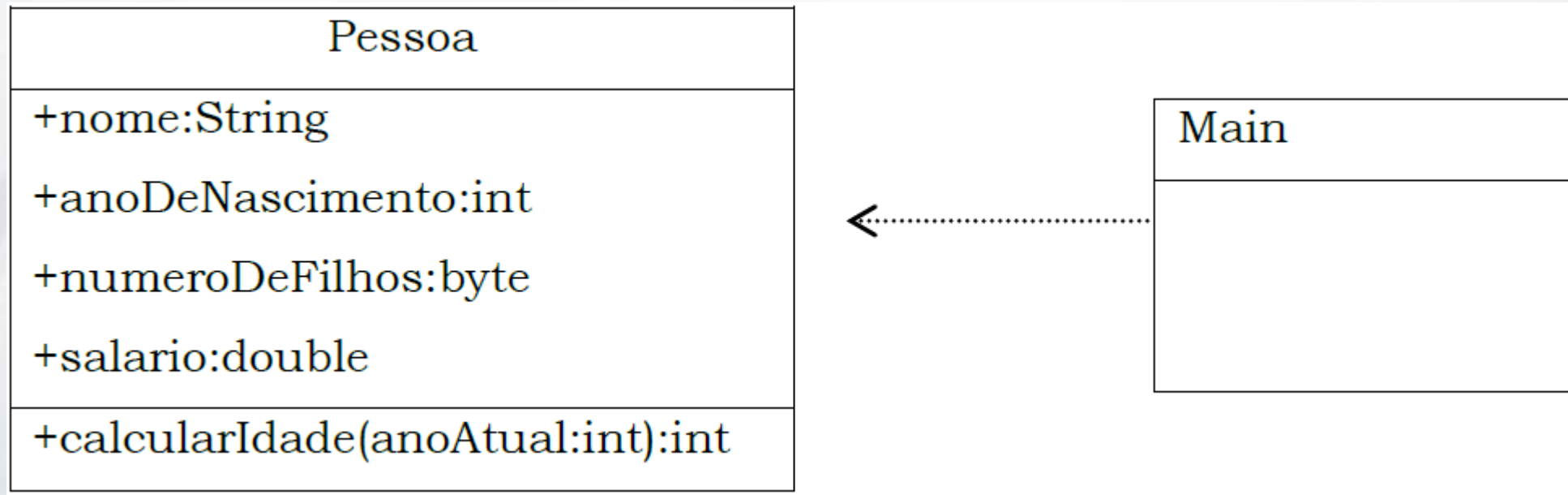
Lendo Número real double

nextDouble() → lê um número com vírgula

```
p1.salario = ler.nextDouble();
```

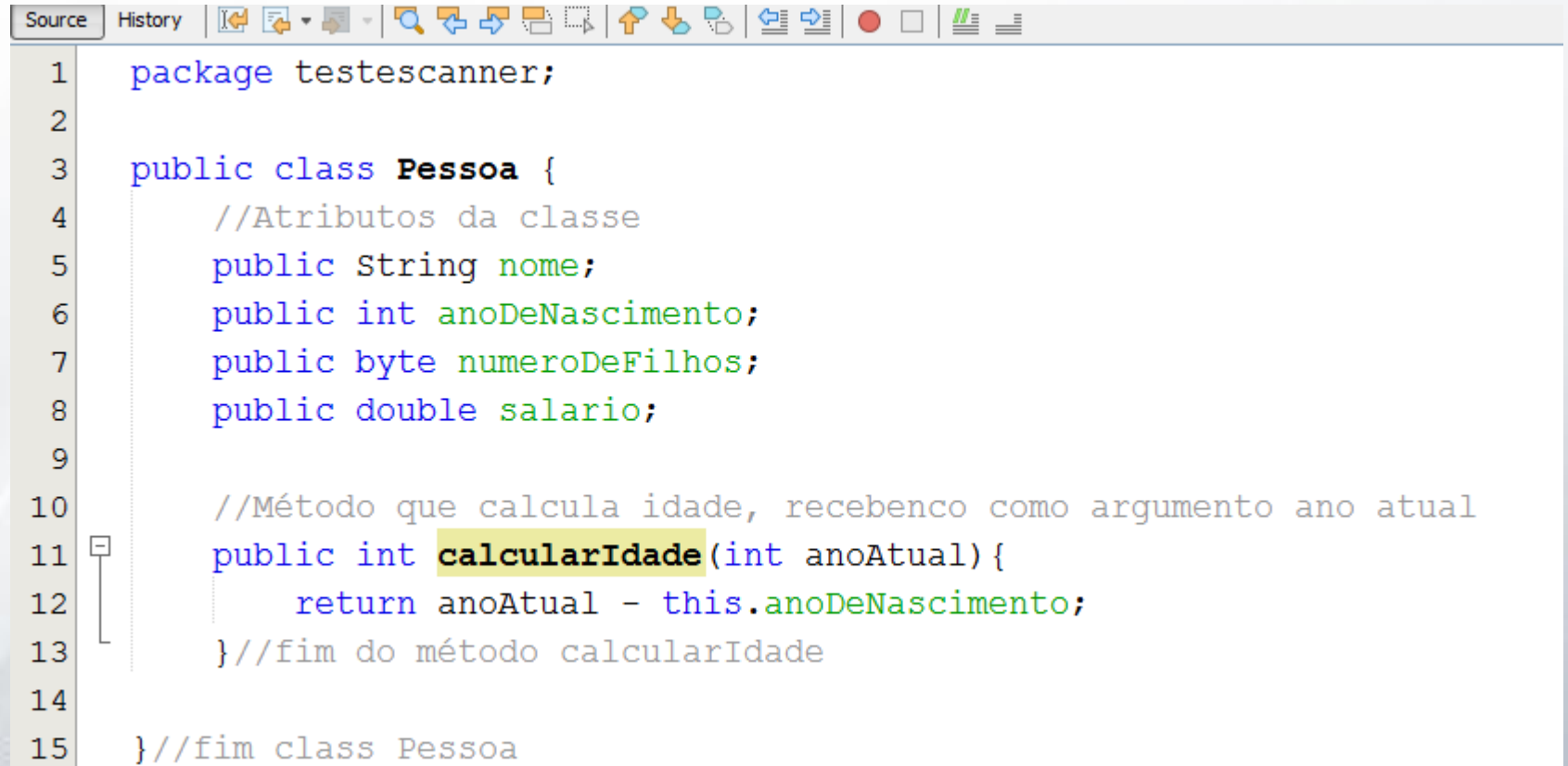
Classe Scanner

Exemplo completo



Classe Scanner

Exemplo completo



The image shows a screenshot of a code editor window. The title bar includes 'Source' and 'History' tabs, along with various icons for file operations and editing. The code is written in Java and defines a class named 'Pessoa' within the 'testescanner' package. The class has four public attributes: 'nome' (String), 'anoDeNascimento' (int), 'numeroDeFilhos' (byte), and 'salario' (double). It also has a public method 'calcularIdade' that takes an 'anoAtual' parameter and returns the difference between 'anoAtual' and 'anoDeNascimento'. The method is highlighted with a yellow background. The code is numbered from 1 to 15 on the left side of the editor.

```
1 package testescanner;
2
3 public class Pessoa {
4     //Atributos da classe
5     public String nome;
6     public int anoDeNascimento;
7     public byte numeroDeFilhos;
8     public double salario;
9
10    //Método que calcula idade, recebendo como argumento ano atual
11    public int calcularIdade(int anoAtual){
12        return anoAtual - this.anoDeNascimento;
13    }//fim do método calcularIdade
14
15 }//fim class Pessoa
```

Classe Scanner

Exemplo completo

```
1 package testescanner;
2 import java.util.Scanner; //Importando a classe Scanner
3
4 public class TesteScanner { //Início da classe
5
6     public static void main(String[] args) { //Definição da classe principal
7
8         //Instanciando a classe Scanner
9         Scanner ler = new Scanner(System.in);
10
11         //Instanciando a classe Pessoa
12         Pessoa p1 = new Pessoa();
13
14         //Lendo os atributos do objeto
15         System.out.print("Digite seu nome: "); //Solicitando o atributo nome
16         p1.nome = ler.nextLine(); //Lendo o nome que o usuário digitou
17
18         System.out.print("Digite seu ano de nascimento: "); //Solicitando o atributo ano de nascimento
19         p1.anoDeNascimento = ler.nextInt(); //Lendo o ano que o usuário digitou
20
21         System.out.print("Digite a quantidade de filhos que tem: "); //Solicitando o atributo idade
22         p1.numeroDeFilhos = ler.nextByte(); //Lendo o número de filhos que o usuário digitou
23
24         System.out.print("Digite o seu salário: "); //Solicitando o atributo salário
25         p1.salario = ler.nextDouble(); //Lendo o salário que o usuário digitou
26     }
```

Classe Scanner

Exemplo completo

```
26
27 //Lendo argumentos dos métodos
28 System.out.print("Informe o ano atual: ");
29 int anoAtual = ler.nextInt();
30
31 //Criando uma variável para armazenar o resultado do método
32 int idade = p1.calcularIdade(anoAtual);
33
34 //Exibindo os dados do objeto
35 System.out.print("\n\n\n");
36 System.out.println("Seu nome é: " + p1.nome);
37 System.out.println("Tu nascestes no ano: " + p1.anoDeNascimento);
38 System.out.println("Tu tens " + p1.numeroDeFilhos + " filho(s).");
39 System.out.println("Teu salário é de R$ " + p1.salario);
40
41 //Exibindo variáveis que contém resultados de métodos
42 System.out.println("Tua idade é " + idade + " anos.");
43
44
45 } //fim método main
46
47 } //fim classe TesteScanner
```


Classe Scanner

Exemplo completo

```
Output - TesteScanner (run)

run:
Digite seu nome: Roberto
Digite seu ano de nascimento: 1987
Digite a quantidade de filhos que tem: 2
Digite o seu salário: 2500
Informe o ano atual: 2021

Seu nome é: Roberto
Tu nascestes no ano: 1987
Tu tens 2 filho(s).
Teu salário é de R$ 2500.0
Tua idade é 34 anos.
BUILD SUCCESSFUL (total time: 26 seconds)
```

Encapsulamento

Encapsulamento

No desenvolvimento de *software* orientado a objeto, temos um recurso que auxilia a padronização e controle da criação dos códigos das classes; esse recurso tem o nome de **encapsulamento**.

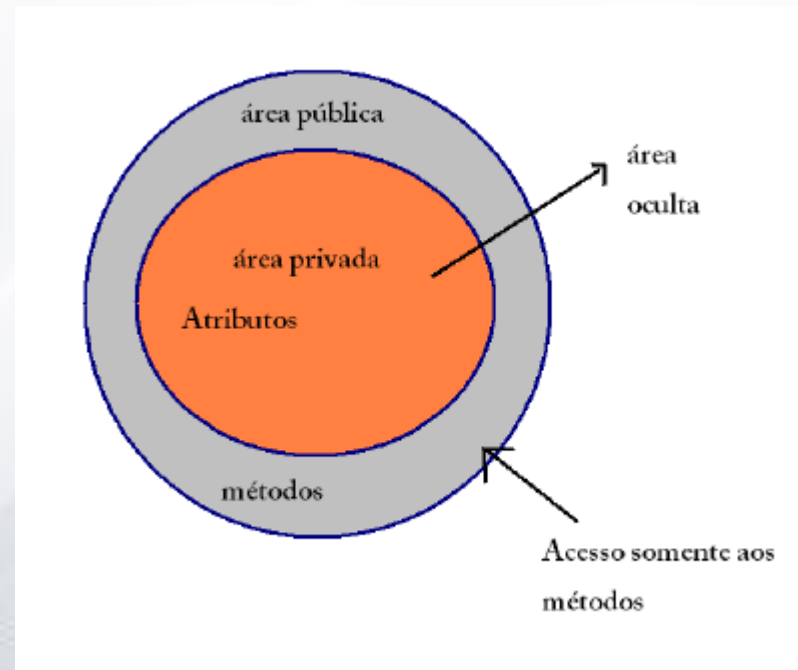
Encapsulamento

De acordo com Scott(2006), mecanismos de encapsulamento permitem que o programador agrupe dados e sub-rotinas(atributos e métodos) em um só lugar, e oculte detalhes sobre a implementação (código) de uma classe.

Encapsulamento

Encapsular significa separar em partes utilizando a abstração (definição do que é realmente relevante). A ideia é deixar o software flexível e facilitar as alterações e o reuso de código (DALE & WEEMS, 2007).

Encapsulamento



Visibilidade

Visibilidade

Para proteger os dados de uma classe encapsulada, precisamos alterar a sua **visibilidade**.

A visibilidade nada mais é do que a maneira que acessamos e enxergamos os dados da nossa classe.

* Para representarmos a visibilidade de atributos/métodos no diagrama UML, usamos os símbolos + para public e – para private.

Visibilidade

Métodos assessores e modificadores

Todo o atributo que conter visibilidade “*private*”, terá que possuir dois métodos especiais, um método de acesso **set** para o caso de poder ser alterado, e um método de consulta **get** para o caso de poder ser consultado.

Visibilidade

Métodos assessores e modificadores

Método set

O set é utilizado para que se consiga enviar uma informação para um atributo. Exemplo: informar um nome que será guardado na variável-atributo nome.

Visibilidade

Métodos assessores e modificadores

Método set

O set se caracteriza por ser um método **sem retorno**, já que seu objetivo é simplesmente armazenar um dado num atributo, e obrigatoriamente **deve conter argumento**, pois precisa receber um valor externo para poder armazená-lo no atributo.

Visibilidade

Métodos assessores e modificadores

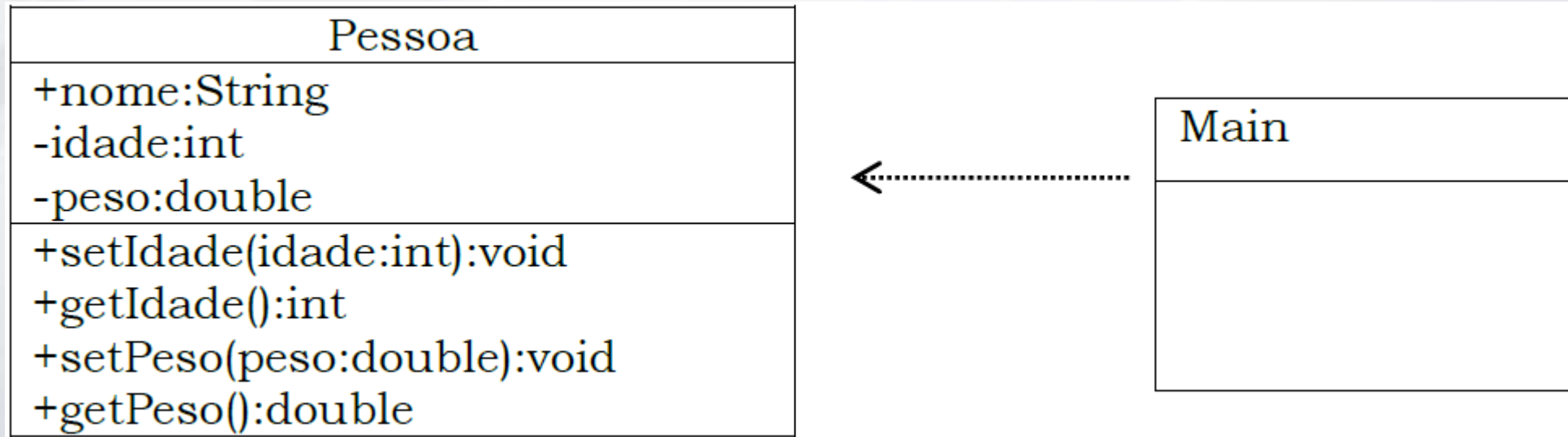
Método *get*

O *get* é utilizado para consultar/obter o valor de um atributo. Sua função é retornar o valor de um atributo específico. Portanto, **sempre tem retorno, e não precisa ter argumentos**.

Visibilidade

Exemplo de uso de encapsulamento

Representação de visibilidade no diagrama de UML



Visibilidade

Exemplo de uso de encapsulamento

```
1  package testeencapsulamento;
2
3  public class PessoaTesteEncapsulamento {
4      public String nome;
5      private int idade;
6      private double peso;
7
8      public int getIdade() {
9          return this.idade;
10     } //fim getIdade
11
12     public void setIdade(int idade) {
13         this.idade = idade;
14     } //fim setIdade
15
16     public double getPeso() {
17         return this.peso;
18     } //fim getPeso
19
20     public void setPeso(double peso) {
21         this.peso = peso;
22     } //fim setPeso
23
24 }
```

Visibilidade

Exemplo de uso de encapsulamento

```
1 package testeencapsulamento;
2 import java.util.Scanner;
3
4 public class TesteEncapsulamento {
5
6     public static void main(String[] args) {
7         //Instâncias
8         Scanner ler = new Scanner(System.in);
9         PessoaTesteEncapsulamento p1 = new PessoaTesteEncapsulamento();
10
11         /***** Lendo atributos do objeto *****/
12         System.out.print("Digite seu nome: ");
13         p1.nome = ler.nextLine();
14
15         System.out.print("Digite sua idade: ");
16         p1.setIdade(ler.nextInt());
17
18         System.out.print("Digite o seu peso: ");
19         p1.setPeso(ler.nextDouble());
20
21         //Mostrando os dados
22         System.out.println("VISUALIZANDO OS DADOS:");
23         System.out.println("Nome: " + p1.nome);
24         System.out.println("Idade: " + p1.getIdade());
25         System.out.println("Peso: " + p1.getPeso());
26
27     } //fim método main
28
29 } //fim classe TesteEncapsulamento
```

Método toString

Método toString

Implementamos o método **toString** para retornar o objeto em formato de **texto**. Ele simplifica a exibição dos atributos do objeto, convertendo o objeto para texto. Neste método, determinamos como os atributos devem ser exibidos.

Método toString

Sintaxe do método toString

Este método não pode ser criado de qualquer maneira. Ele possui uma sintaxe padrão, onde alteramos apenas o que vai no “*return*”. O nome deve ser **toString**, sempre deve retornar uma **String** e não possui argumentos.

Método toString

Sintaxe do método toString

Exemplo de implementação do toString

```
public String toString(){  
    return "mensagem";  
}
```

```
public String toString(){  
    return "Nome: " + this.nome + "\nIdade: " + this.idade + "\nPeso: " + this.peso;  
}  
  
} //fim class PessoaTesteEncapsulamento
```

Método toString

Sintaxe do método toString

Exemplo UML de classe com toString

Data
-dia: byte -mes: byte -ano: int
+getDia():byte +getMes():byte +getAno():int +setDia(dia:byte):void +setMes(mes:byte):void +setAno(ano:byte):void +toString():String

Método toString

Exemplo com toString

```
1 package testetostRING;
2
3 public class Data {
4     private int dia;
5     private int mes;
6     private int ano;
7
8     public int getDia() {
9         return this.dia;
10    } //fim getDia
11
12    public void setDia(int dia) {
13        this.dia = dia;
14    } //fim setDia
15
16    public int getMes() {
17        return this.mes;
18    } //fim getMes
19
20    public void setMes(int mes) {
21        this.mes = mes;
22    } //fim setMes
23
24    public int getAno() {
25        return this.ano;
26    } //fim getAno
27
28    public void setAno(int ano) {
29        this.ano = ano;
30    } //fim setAno
31
32    public String toString() {
33        return this.dia + "/" + this.mes + "/" + this.ano;
34    } //fim toString
35
36 } //fim classe Data
```

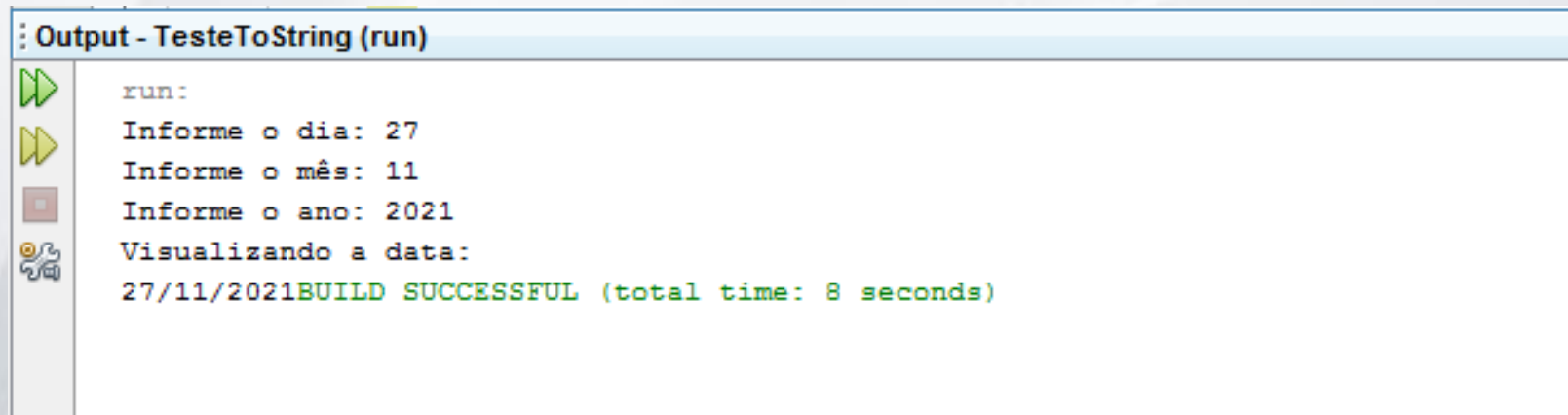
Método toString

Exemplo com toString

```
1  package testetostRING;
2  import java.util.Scanner;
3
4  public class TesteToString {
5
6      public static void main(String[] args) {
7          //Instâncias
8          Scanner ler = new Scanner(System.in);
9          Data d1 = new Data();
10
11          //Usuário informando os dados
12          System.out.print("Informe o dia: ");
13          d1.setDia(ler.nextInt());
14
15          System.out.print("Informe o mês: ");
16          d1.setMes(ler.nextInt());
17
18          System.out.print("Informe o ano: ");
19          d1.setAno(ler.nextInt());
20
21          //Mostrando as informações
22          System.out.println("Visualizando a data:");
23          System.out.print(d1);
24
25      } //fim método main
26
27  } //fim classe TesteEncapsulamento
```

Método toString

Exemplo com toString



```
Output - TesteToString (run)
run:
Informe o dia: 27
Informe o mês: 11
Informe o ano: 2021
Visualizando a data:
27/11/2021BUILD SUCCESSFUL (total time: 8 seconds)
```