



Curso de Engenharia da Computação

Paradigmas de Programação (60h/a)

POO: Criação de Classes em Java

Aula 04

Prof. Lenardo Chaves e Silva, D.Sc. 

lenardo@ufersa.edu.br

terça-feira, 28 de novembro de 2017



Universidade Federal Rural do Semi-Árido (UFERSA)

☛ Criação de Classes (Parte I):

☐ Regras Básicas de Sintaxe:

- Classes;
- Atributos;
- Métodos;
- Escopo;
- Modificadores de Acesso.

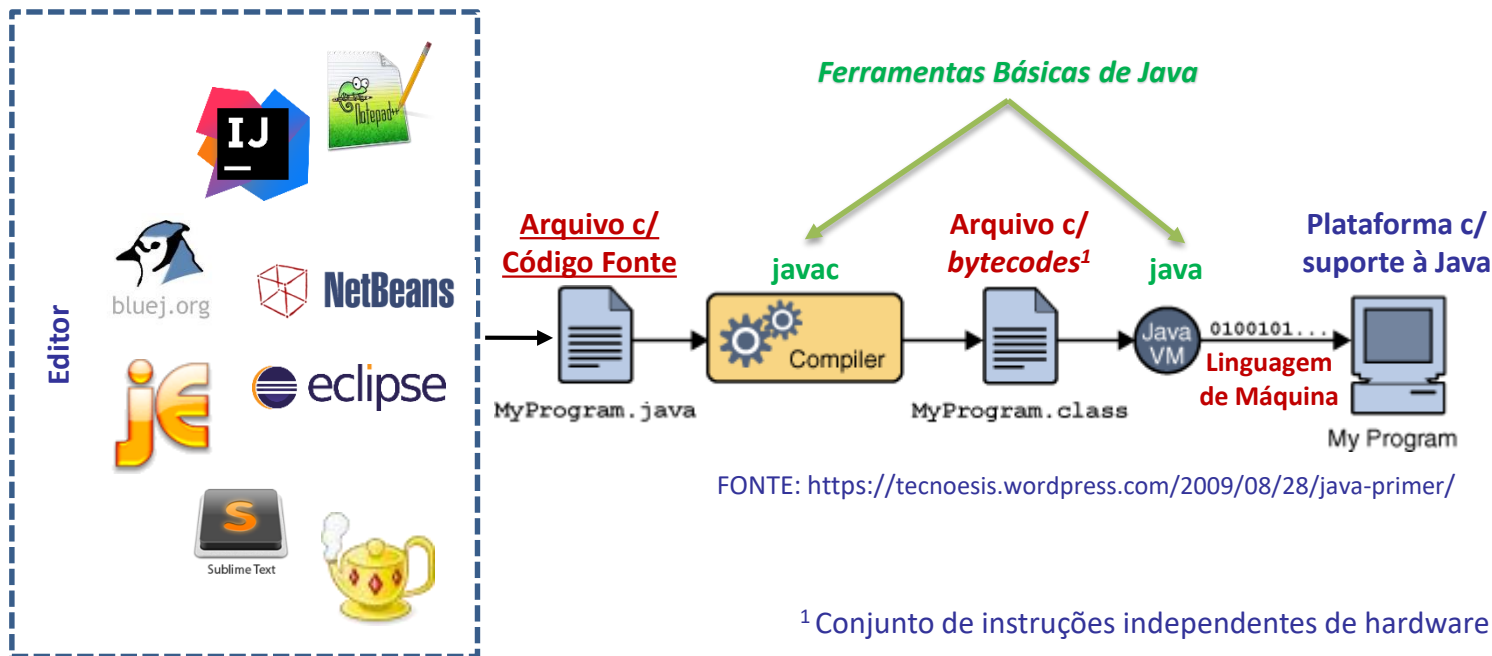
☛ Exercícios de Fixação;

☛ Referências.

O que vimos antes...

Ambiente de Desenvolvimento Java Típico:

➡ Processo de Compilação e Interpretação:



¹ Conjunto de instruções independentes de hardware particular.

☛ Alguns Exemplos de Programas em Java:

- ☐ **Primeiro Programa:** Como seria um HelloWorld em Java?
- ☐ **Segundo Programa:** Mín. e Máx. dentre 3 inteiros (Solução 1-5);
- ☐ **Terceiro Programa:** Planejamento de Aposentadora.
- ☐ **Quarto Programa:** Ler 10 inteiros e imprime-os em ordem inversa (v1 e v2);
- ☐ **Quinto Programa:** Exibe argumentos de linha de comando;
- ☐ **Sexto Programa:** Mostra uma forma de inicializar *arrays*.

☛ Tipos Primitivos:

- ☐ Limites de Representação;
- ☐ Conversão entre Tipos (Implícita/Promoção e Explícita/*Casting*).

☛ Operadores Java:

- ☐ Operadores Unários e Binários;
- ☐ Operadores Relacionais e Lógicos;
- ☐ Operador Ternário e outros operadores;
- ☐ Precedência dos operadores.

Relembrando...

Programação Orientada a Objetos...

Modelos → entidades do mundo real ← coleção de **objetos** relacionados!

ContaBancariaSimples
<ul style="list-style-type: none"> - nomeDoCorrentista : String - saldo : Double - contaEspecial : Boolean
<ul style="list-style-type: none"> - abreConta(nome : String, deposito : Double, especial : Boolean) : void - abreContaSimples(nome : String) : void - deposita(valor : Double) : void - retira(valor : Double) : void - mostraDados() : void



Mundo Bancário:

- Contas corrente
- Contas poupança
- Clientes
- Caixas
- Agências
- Cheques
- Extratos, ...

☛ **Objetos** podem ser agrupados em **Classes**;

- ☐ +1 Conta Corrente (objetos) ← “Conta Corrente” (classe).
- ☐ Classe ≈ algo abstrato **vs.** Objeto ≈ algo materializado/concreto.

Planta (Classe)



Casas (objetos)

Programação Orientada a Objetos...

Modelos → entidades do mundo real ← coleção de **objetos** relacionados!

☛ Objetos de uma **Classe** possui *atributos*:

☐ Exemplos:

- Conta → número, saldo, histórico de transações, ...
- Cliente → nome, cpf, endereço, ...
- Cheque → valor, data, ...

☛ Objetos de uma mesma **Classe** possui um mesmo *comportamento*:

☐ Exemplos:

- Clientes → consultam saldo, fazem depósitos, realizam saques, ...

Programação Orientada a Objetos...

☛ **Mundo Real** → existem objetos sem comportamento próprio:

☐ **Exemplo:** Conta (não fazem sentido sozinhas).

☛ **Objetos** podem estar relacionados:

☐ **Exemplos:**

▪ Cliente → possui +1 Conta.

☛ **Uso de objetos** no projeto de Software:

☐ **Algumas Vantagens:**



- Facilitar a comunicação com o cliente;
- Facilitar o desenvolvimento e a evolução;
- Agregar qualidade.

Regras Básicas de Sintaxe:

☛ **Classe:** `class NomeDaClasse { // corpo da classe }`

☐ **Restrições** quanto ao *NomeDaClasse*:

1. Não pode conter espaços;
2. Convenção → iniciar com letras ***MaiusculaAlternandoEntrePalavras***, incluindo '_' e '\$'.
3. Apesar de poder conter acentos, **NÃO** aconselha-se o uso!
4. Pode conter números após a primeira letra;
5. Não podem ser idênticos às palavras reservada de Java.
6. Java é *case-sensitive*: ~~Class~~, ~~CLASS~~, ~~ClasS~~ e class.



Java Keywords				
abstract	assert	boolean	break	byte
case	catch	char	class	continue
default	do	double	else	enum
extends	final	finally	float	for
if	implements	import	instanceof	int
interface	long	native	new	package
private	protected	public	return	short
static	strictfp	super	switch	synchronized
this	throw	throws	transient	try
void	volatile	while		
<i>Keywords that are not currently used</i>				
const	goto			

Regras Básicas de Sintaxe:

☛ Classe:

❑ Restrições quanto ao Corpo da Classe:

1. Delimitado por um par de chaves '{' e '}' (i.e., bloco de código);
2. Atributos e Métodos da classe → entre os delimitadores.

❑ Quanto ao espaçamento vertical e a indentação (reco horizontal):

- Sem restrições;



Boa Prática: espaçamento adequado e indentação regular!

Tipos de
Comentários

```

Vazia.java
1 package pex0246.exemplos;
2
3 /**
4  * A classe Vazia, que não possui campos nem métodos, mas mesmo assim pode ser usada
5  * para exemplificar as regras sintáticas básicas de Java, podendo até mesmo ser
6  * compilada.
7  */
8
9 public class Vazia // esta é a declaração da classe !
10 {
11     /* Se houvessem campos ou métodos para a classe Vazia,
12     eles deveriam ser declarados aqui dentro. */
13
14 } // fim da classe Vazia
15
    
```

Regras Básicas de Sintaxe:

- ☛ **Atributos da Classe:** *tipo nomeAtributo; //padrão*
 - ❑ Declarados dentro do Corpo da Classe;
 - ❑ Representados: *tipo primitivo* da LP / instância de outra classe “pronta”.
 - Ex.: Classe *String* → cadeia de caracteres (2 bytes):
 - **NÃO** é um tipo primitivo Java, mas tratados como...

RegistroAcademicoSimples.java

```

1
2 /**
3  * A classe RegistroAcademicoSimples contém somente al
4  * as declarações de atributos e tipos em Java. Esta c
5  * Data para ser compilada com sucesso.
6  */
7 class RegistroAcademicoSimples // declaração da classe
8 {
9
10 /**
11  * Declaração dos atributos da classe
12  */
13 String nomeDoAluno; // uma cadeia de caracteres para representar um nome
14 int numeroDeMatricula; // pode representar números com até 9 dígitos !
15 Data dataDeNascimento = new Data(); // uma referência a uma instância da classe Data
16 boolean eBolsista; // valor simples: sim ou não (true ou false)
17 short anoDeMatricula; // um short basta para representar anos
18
19 /* Se houvesse métodos para esta classe, eles seriam declarados aqui dentro. */
20
21 } // fim da classe RegistroAcademicoSimples
  
```

RegistroAcademico

- nomeDoAluno : String
- numeroDeMatricula : int
- dataDeNascimento : Data
- eBolsista : Boolean
- anoDeMatricula : int

- inicializaRegistro(nome : String, matricula : int, data : Data, bolsa : Boolean, ano : int) : void
- calculaMensalidade() : void
- mostraRegistro() : void

Referências à uma outra Classe

```

String nomeDoAluno; // uma cadeia de caracteres para representar um nome
int numeroDeMatricula; // pode representar números com até 9 dígitos !
Data dataDeNascimento = new Data(); // uma referência a uma instância da classe Data
boolean eBolsista; // valor simples: sim ou não (true ou false)
short anoDeMatricula; // um short basta para representar anos
  
```

Regras Básicas de Sintaxe:

➤ Atributos da Classe:

Data
- dia : int - mes : int - ano : int
- inicializaData(d : int, m : int, a : int) : void - dataValida(d : int, m : int, a : int) : Boolean - mostraData() : String

```
DataSemMetodos.java ✕
1
2 /**
3  * A classe DataSemMetodos contém somente alguns dados que exemplificam as
4  * declarações de atributos e tipos em Java.
5  */
6 class DataSemMetodos // declaração da classe
7 {
8
9     /**
10     * Declaração dos atributos da classe
11     */
12     byte dia; // tanto o dia quanto o mês podem ser representados por bytes
13     byte mês; // tanto o dia quanto o mês podem ser representados por bytes
14     short ano; // o ano completo (quatro dígitos) deve ser representado por um short
15
16     /* Se houvesse métodos para esta classe, eles seriam declarados aqui dentro. */
17
18 } // fim da classe DataSemMetodos
```

Criação de Classes em Java

Regras Básicas de Sintaxe:

☛ Atributos da Classe: Restrições quanto ao *nomeAtributo*:

- Em geral, similar às regras para o *NomeDaClasse*;
- **Convenção:** inicia-se com letra *minusculaMaiuscula*:

Ex.: *dia*, *nomeDoAluno*, *estadoDaLâmpada*.



Não Permitido: atributos duplicados → nomes iguais dentro da mesma classe, mesmo se o tipo for diferente.

☐ Outras formas de declaração:

1. *tipo-primitivo-comum nomeAtributo1, nomeAtributo2; //* 👍 *String*.
2. *NomeDaClasse nomeDaReferenciaAClasse = new NomeDaClasse();*
3. *modificador-de-acesso tipo-ou-classe nomeAtributo;*

Obs.: A inicialização dos atributos pode ser feita depois.

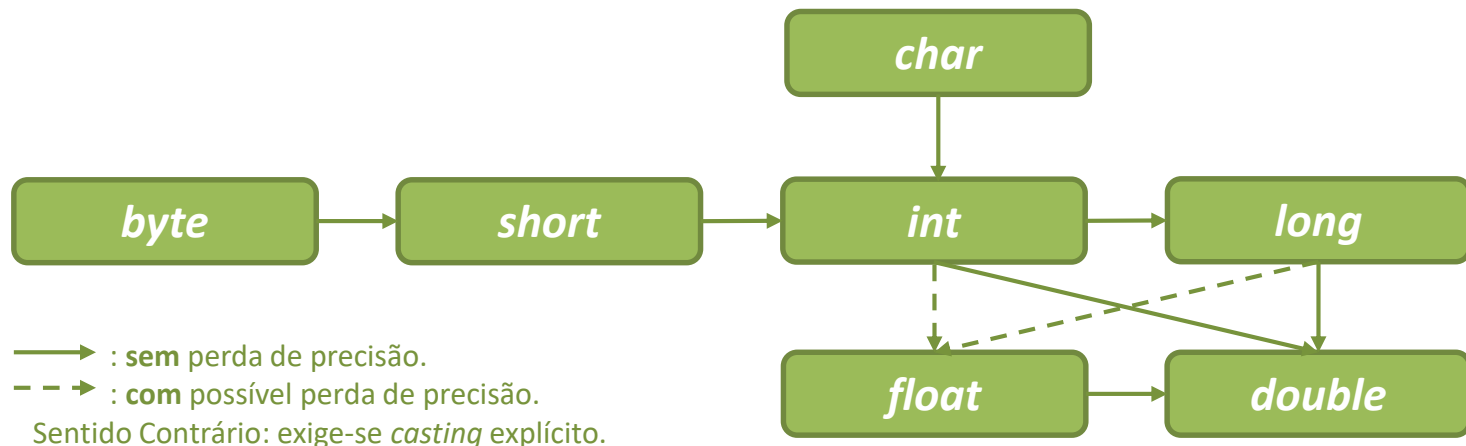


DICA: Evitar a declaração de muitos atributos em uma mesma linha !

Atributos da Classe:

☛ Escolhendo o Tipo de Dado:

- ❑ Verificar os limites de representação;
 - Ex.: *numeroDeMatricula* → **long** (9 dígitos)/ **int** / **short** (até 32767)?
- ❑ Escolha inadequada do tipo pode custar caro!
 - Complexidade desnecessária (e.g., *casting*, + memória, operações);
 - Erros de Lógica de Programação (e.g., perda de precisão do dado).



Atributos da Classe:

☛ Escolhendo o Tipo de Dado:

❑ Economizar demais também pode trazer prejuízos!

▪ Ex.: *short* ano; //...até +32.767 ← alterar

→ *byte* ano; //...até +127

→ *static final short* ANO_MINIMO = 1970; //constante global

Logo: *ano* = 31; → 1970 + *ano* = 2001.

ano = 127; //valor máximo → ANO_MINIMO + *ano* = 2097



Custo de usar essa técnica vale a pena?

R.: Não, pois economiza-se apenas um byte.

Regras Básicas de Sintaxe:

☛ Métodos da Classe:

```
tipo-retorno nomeMetodo(lista-argumentos) //padrão  
{ // corpo do método }
```

☐ Restrições quanto ao *nomeMetodo*:

- Em geral, similar às regras para o *nomeAtributo*;
- **Convenção:** inicia-se com letra *minusculaMaiuscula*:
Ex.: *acende, mostraDados, inicializaData*.



Boa Prática: refletir ações a serem efetuadas → seus atributos e/ou argumentos.



Não Permitido: declarar métodos aninhados → métodos dentro de outros métodos e nem fora de uma classe (isolados).

Regras Básicas de Sintaxe:

☛ Métodos da Classe:

☐ Outras formas de declaração:

1. *modificador-de-acesso* *tipo-ou-classe-retorno* *nomeMetodo* (*lista-argumentos*) { *corpo do método* }



Obs.: A declaração de um método **NÃO** finaliza com ‘;’.

☐ Restrições quanto à declaração:

▪ Tipo/Classe de Retorno:

- Valor a ser retornado pelo método.

Ex.: DataSimples: *dataÉVálida* e *éIgual*.

- Sem retorno → palavra reservada **void**.

Ex.: DataSimples: *inicializaDataSimples* e *mostrarDataSimples*.

- Os que retornam algum valor (e.g., *dataÉVálida*):

- Palavra reservada

return *constante/variável de tipo ou classe retornada;*

Criação de Classes em Java

Regras Básicas de Sintaxe:

☛ Métodos da Classe:

- ❑ Lista de Argumentos (0 ou +): referências/variáveis contendo valores.

Ex.: DataSimples: `void inicializaDataSimples(byte d, byte m, short a) {...}`
`boolean éIgual(DataSimples outraDataSimples) {...}`



Obs.: O recebimento de argumentos por um método é **OPCIONAL**.

```
DataSimples.java
2+  * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4  class DataSimples // declaração da classe {
5
7-   * Declaração dos atributos da classe
9   byte dia, mês; // dia e mês são representados por bytes
10  short ano; // ano é representado por um short
11
13+  * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21+  void inicializaDataSimples(byte d, byte m, short a) {
35
37+  * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47+  boolean dataÉVálida(byte d, byte m, short a) {
58
60+  * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66+  boolean éIgual(DataSimples outraDataSimples) {
76
78+  * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83+  void mostraDataSimples() {
91 } // fim da classe DataSimples
```

Regras Básicas de Sintaxe:

- Métodos da Classe: *tipo-retorno nomeMetodo() { // corpo do método }*

```
DataSimples.java
20 * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4 class DataSimples // declaração da classe {
5
7 * Declaração dos atributos da classe
9 byte dia, mês; // dia e mês são representados por bytes
10 short ano; // ano é representado por um short
11
13 * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21 void inicializaDataSimples(byte d, byte m, short a) {
22     if (dataÉVálida(d, m, a)) // se a data for válida, inicializa os campos com os
23         // valores passados como argumentos
24     {
25         dia = d;
26         mês = m;
27         ano = a;
28     } else // caso contrário, inicializa os campos com zero
29     {
30         dia = 0;
31         mês = 0;
32         ano = 0;
33     }
34 } // fim do método inicializaDataSimples
35
37 * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47 boolean dataÉVálida(byte d, byte m, short a) {
58
60 * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66 boolean éIgual(DataSimples outraDataSimples) {
76
78 * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83 void mostraDataSimples() {
91 } // fim da classe DataSimples
```

Regras Básicas de Sintaxe:

- Métodos da Classe: *tipo-retorno nomeMetodo() { // corpo do método }*

```

DataSimples.java
2+ * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4 class DataSimples // declaração da classe {
5
7- * Declaração dos atributos da classe
9 byte dia, mês; // dia e mês são representados por bytes
10 short ano; // ano é representado por um short
11
13+ * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21+ void inicializaDataSimples(byte d, byte m, short a) {
35
37+ * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47- boolean dataÉVálida(byte d, byte m, short a) {
48     if ((d >= 1) && // se o dia for maior ou igual a 1 E
49         (d <= 31) && // se o dia for menor ou igual a 31 E
50         (m >= 1) && // se o mês for maior ou igual a 1 E
51         (m <= 12)) // se o mês for menor ou igual a 12 ENTÃO
52     {
53         return true; // a data é válida, retorna true
54     } else {
55         return false; // a data não é válida, retorna false
56     }
57 } // fim do método dataÉVálida
58
60+ * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66+ boolean éIgual(DataSimples outraDataSimples) {
76
78+ * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83+ void mostraDataSimples() {
91 } // fim da classe DataSimples
    
```

Regras Básicas de Sintaxe:

☛ Métodos da Classe: *tipo-retorno nomeMetodo() { // corpo do método }*

```
DataSimples.java
2+ * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4 class DataSimples // declaração da classe {
5
7- * Declaração dos atributos da classe
9 byte dia, mês; // dia e mês são representados por bytes
10 short ano; // ano é representado por um short
11
13+ * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21+ void inicializaDataSimples(byte d, byte m, short a) {
35
37+ * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47+ boolean dataÉVálida(byte d, byte m, short a) {
58
60+ * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66- boolean éIgual(DataSimples outraDataSimples) {
67     if ((dia == outraDataSimples.dia) && // se os dois dias forem iguais E
68         (mês == outraDataSimples.mês) && // se os dois meses forem iguais E
69         (ano == outraDataSimples.ano)) // se os dois anos forem iguais então
70     {
71         return true; // a data é igual, retorna true
72     } else {
73         return false; // a data é diferente, retorna false
74     }
75 } // fim do método éIgual
76
78+ * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83+ void mostraDataSimples() {
91 } // fim da classe DataSimples
```

Criação de Classes em Java

Regras Básicas de Sintaxe:

- Métodos da Classe: *tipo-retorno nomeMetodo() { // corpo do método }*

```
DataSimples.java ✕
2+ * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4 class DataSimples // declaração da classe {
5
7- * Declaração dos atributos da classe
9 byte dia, mês; // dia e mês são representados por bytes
10 short ano; // ano é representado por um short
11
13+ * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21+ void inicializaDataSimples(byte d, byte m, short a) {
35
37+ * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47+ boolean dataÉVálida(byte d, byte m, short a) {
58
60+ * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66+ boolean éIgual(DataSimples outraDataSimples) {
76
78+ * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83- void mostraDataSimples() {
84
85     System.out.print(dia); // O método print do campo out da classe System faz com
86     System.out.print("/"); // que o argumento passado a ele seja transformado em uma
87     System.out.print(mês); // string e impresso no terminal. O método println faz a
88     System.out.print("/"); // mesma coisa, mas adiciona uma quebra de linha ('\n')
89     System.out.println(ano); // ao final da string impressa.
90 } // fim do método mostraDataSimples
91 } // fim da classe DataSimples
```

Criação de Classes em Java

Escopo → Visibilidade: atributos e métodos dentro da classe.



Acessados/Modificados por “quem”?

❑ Atributos:

- Declarados na classe → Válidos em toda a classe.

Ex.: *lado1*, *lado2* e *lado3*.

❑ Variáveis e Instâncias:

- Declaradas no método → Válidas apenas naquele método.


Ex.: *igualdade12*, *igualdade23*.

Obs.: Declaradas = nome e tipo em ≠ métodos → **diferentes**.

Ex.: *resultado*.

```

Triangulo.java
2+ * A classe Triangulo representa os três lados de um triângulo qualquer.
4 class Triangulo // declaração da classe
5 {
6
8+ * Declaração de um dos atributos da classe
10 float lado1;
11
13+ * O método éEquilátero verifica se o triângulo é equilátero ou não.
16- boolean éEquilátero() {
17     boolean igualdade12, resultado;
18     igualdade12 = (lado1 == lado2); // o lado 1 é igual ao lado 2 ?
19     boolean igualdade23;
20     igualdade23 = (lado2 == lado3); // o lado 2 é igual ao lado 3 ?
21     if (igualdade12 && igualdade23) // os três lados são iguais ?
22     {
23         resultado = true;
24     } else {
25         resultado = false;
26     }
27     return resultado;
28 } // fim do método éEquilátero
29
31+ * O método calculaPerímetro calcula o perímetro do triângulo usando seus lados.
34- float calculaPerímetro() {
35     float resultado = lado1 + lado2 + lado3;
36     return resultado;
37 } // fim do método perímetro
39- * Declaração dos outros atributos da classe
41 float lado2, lado3;
42 } // fim da classe Triangulo
    
```

IMPORTANTE! 

Ordem das Declarações!

IRRELEVANTE!

Criação de Classes em Java

Escopo → Visibilidade: atributos e métodos dentro da classe.



Acessados/Modificados por “quem”?

❑ **Variáveis e Instâncias:**

- Passadas como argumento de um método → Válidas apenas naquele método.

Ex.: *d*, *m* e *a*.

```

DataSimples.java
2+ * A classe DataSimples contém atributos e métodos que permitem a manipulação de datas.
4 class DataSimples // declaração da classe {
5
7- * Declaração dos atributos da classe
9 byte dia, mês; // dia e mês são representados por bytes
10 short ano; // ano é representado por um short
11
13+ * O método inicializaDataSimples recebe argumentos para inicializar os campos da
21+ void inicializaDataSimples(byte d, byte m, short a) {
35
37+ * O método dataÉVálida recebe três valores como argumentos e verifica de maneira
47+ boolean dataÉVálida(byte d, byte m, short a) {
58
60+ * O método éIgual recebe uma instância da própria classe DataSimples como argumento
66+ boolean éIgual(DataSimples outraDataSimples) {
76
78+ * O método mostraDataSimples não recebe argumentos nem retorna valores. Este método
83+ void mostraDataSimples() { // d,m e a não são visíveis aqui!
91 } // fim da classe DataSimples
    
```


Criação de Classes em Java

Modificadores de Acesso → Encapsulamento: atributos e métodos.

☛ **Motivação:** evitar o acesso direto aos dados.



Nos exemplos anteriores... houve encapsulamento, mas não ocultação.

```

1 package pex0246.exemplos;
2
3
4+ * A classe DemoDataSimples demonstra usos da classe DataSimples, em especial os .
8 class DemoDataSimples { // declaração da classe
9
10+ * O método main permite a execução desta classe. Este método contém declarações.
17- public static void main(String[] argumentos) {
18
19     // Criamos duas instâncias da classe DataSimples, usando a palavra-chave new. As
20     // instâncias serão associadas a duas referências, que permitirão o acesso aos
21     // campos e métodos das instâncias.
22
23     DataSimples hoje = new DataSimples();
24     DataSimples independênciaDoBrasil = new DataSimples();
25
26     // E três variáveis para receber o dia, mês e ano para as datas
27     byte umDia, umMês;
28     short umAno;
29
30     // Inicializamos "hoje" com uma data não-válida
31     umDia = 40;
32     umMês = 1;
33     umAno = 2001;
34     hoje.inicializaDataSimples(umDia, umMês, umAno); // inicializa os campos da instância
35     hoje.mostraDataSimples(); // imprime 0/0/0
36

```

“Execução de Métodos” ↔ “Envio de Mensagens” aos objetos

Criação de Classes em Java

Modificadores de Acesso → Encapsulamento: atributos e métodos.



Problema: “Impossível garantir o processamento adequado de datas”.



Solução: ocultar dados e permitir a manipulação usando métodos.

```

DataSimples.java  DemoDataSimples.java X
37      // Inicializamos "independênciaDoBrasil" com uma data válida
38      umDia = 7;
39      umMês = 9;
40      umAno = 1822;
41      independênciaDoBrasil.inicializaDataSimples(umDia, umMês, umAno);
42      independênciaDoBrasil.mostraDataSimples(); // imprime 7/9/1822
43
44      // Vamos testar o método éIgual:
45      if (hoje.éIgual(independênciaDoBrasil)) {
46          System.out.println("As datas são iguais !");
47      } else {
48          System.out.println("As datas são diferentes !");
49      }
50
51      // O problema: podemos facilmente "invalidar" datas válidas acessando os seus
52      // campos diretamente:
53      hoje.dia = 0;
54      hoje.mês = 1;
55      hoje.ano = 2001;
56      hoje.mostraDataSimples(); // imprime 0/1/2001 - é válida ou não ?
57      independênciaDoBrasil.mês = 13;
58      independênciaDoBrasil.mostraDataSimples(); // imprime 7/13/1822 - é válida ou não ?
59
60      } // fim do método main
61  } // fim da classe DemoDataSimples
    
```

Operador ponto (.) → invocar métodos de uma classe qualquer!

Modificadores de Acesso → Encapsulamento: atributos e métodos.

☐ Java:

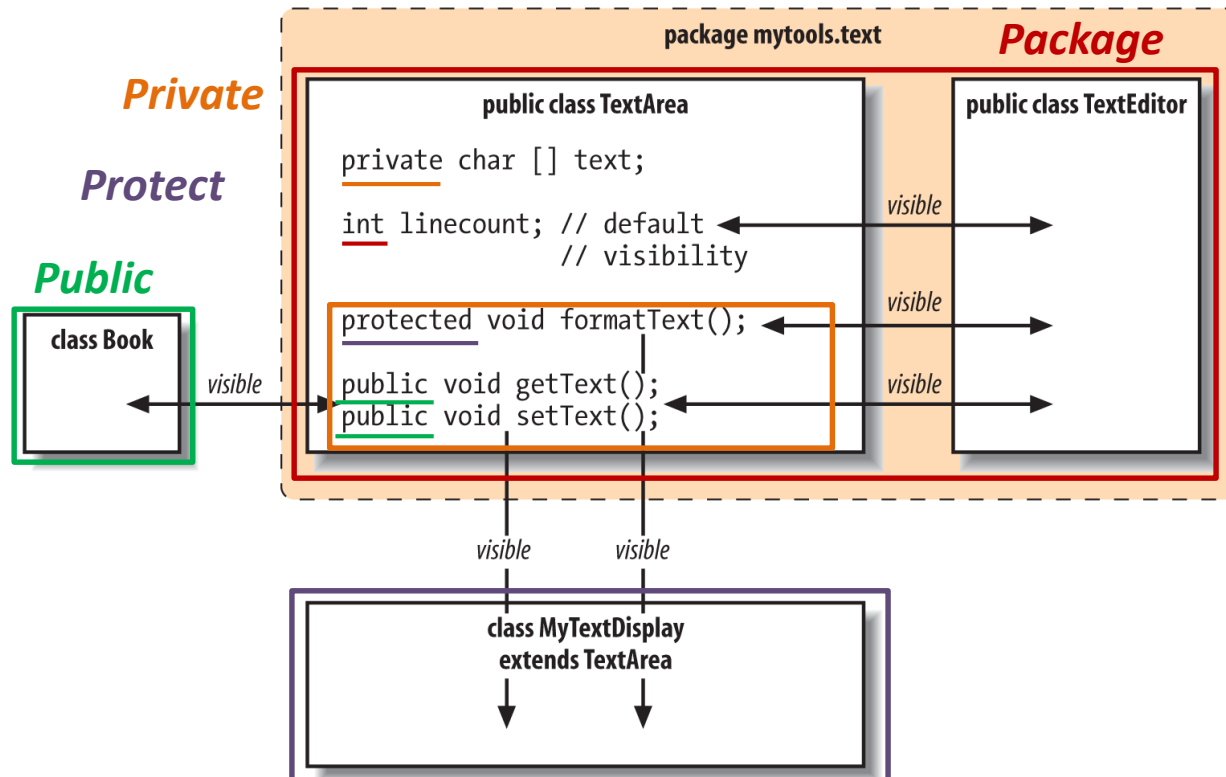
1. **public:** acessado/executado → por qualquer outra classe.
2. **private:** acessado/executado → por métodos da própria classe.
3. **protect:** acessado/executado → como *private* e por classes **herdeiras/derivadas**.
4. **package/friendly:** acessado/executado → por classes dentro do mesmo pacote.

Obs.: Quando não especificado explicitamente.

- **Declaração** → dentro das classes, antes dos métodos e atributos:
 - **Atributos:** *modificador-de-acesso tipo-ou-classe nomeAtributo;*
 - **Métodos:** *modificador-de-acesso tipo-ou-classe-retorno nomeMetodo(lista-argumentos) { \\ corpo do método }*

Modificadores de Acesso → Encapsulamento: atributos e métodos.

❑ Java: *public*, *private*, *protect* e *package/friendly*.



<http://chimera.labs.oreilly.com/books/1234000001805/ch06.html#learnjava3-CHP-6-SECT-4.1>

Modificadores de Acesso → Encapsulamento: atributos e métodos.

❑ Política de ocultação ou de acesso a dados e métodos internos:

1. Todos os atributos da classe → declarados como ***private*** ou ***protect***;
2. Métodos da classe → acessíveis usando ***public*** (i.e., explícito);
3. Para os atributos ***private*** ← manipular → criar métodos com ***public***;
4. Os métodos ***private*** da classe → executados apenas p/ outros métodos da própria classe;



Programador
de Classe



DICA: Funcionam bem para casos básicos de criação de classes!

FONTE: SANTOS, R. Introdução à programação orientada a objetos usando JAVA. 2. ed. Rio de Janeiro: Campus, 2013. 336p. Capítulo 2.

Mix dos Exercícios 2.1, 2.2 e 2.3:

Quais dos identificadores abaixo podem ser usados como nomes de classes, campos, métodos e variáveis em Java? Quais não podem, e por quê?

A. Four
B. for
C. from
D. 4
E. FOR

F. dia&noite
G. diaENoite
H. dia & noite
I. dia E noite
J. dia_e_noite

L. contador
M. 1contador
N. contador de linhas
O. Contador
O. count

Exercício 2.4:

Considerando a tabela 2.2, escolha o tipo de dado ou classe mais adequada para representar:

- O número de municípios de um estado do Brasil.
- O nome de um estado do Brasil.
- A população de um estado do Brasil.
- A área do Brasil em quilômetros quadrados.
- A população total do mundo.
- O CEP de um endereço no Brasil.
- O nome de uma rua em um endereço no Brasil.

Exercícios de Fixação - Java



Exercício 2.7: Identifique e explique o(s) erro(s) na classe abaixo.

```
ExDoisValores.java
1 package pex0246.exemplos;
2
3 class DoisValores {
4     /** Declaração dos campos desta classe */
5     int valor1, valor2;
6
7     /** Declaração dos métodos desta classe */
8     int maior() {
9         if (valor1 > valor2)
10             return true;
11         else
12             return false;
13     }
14
15     void menor() {
16         if (valor1 < valor2)
17             return valor1;
18         else
19             return valor2;
20     }
21
22 } // fim da classe
```

Exercício 2.29:

Escreva a classe Contador que encapsule um valor usado para contagem de itens ou eventos. Essa classe deve esconder o valor encapsulado de programadores-usuários, fazendo com que o acesso ao valor seja feito através de métodos que devem zerar, incrementar e imprimir o valor do contador.

Exercício 2.38:

Escreva, na classe Data, um método duplicaData que receba como argumento uma outra instância da classe Data, e duplique os valores dos campos da instância passada como argumento para os campos encapsulados.

Exercício 2.48:

Uma das operações que podemos efetuar com datas é a comparação para ver se uma data ocorre antes de outra. O algoritmo para comparação é muito simples, e seus passos estão abaixo. Nesse algoritmo, consideramos que dia1, mês1 e ano1 são os dados da primeira data, e que dia2, mês2 e ano2 são os dados da segunda data.

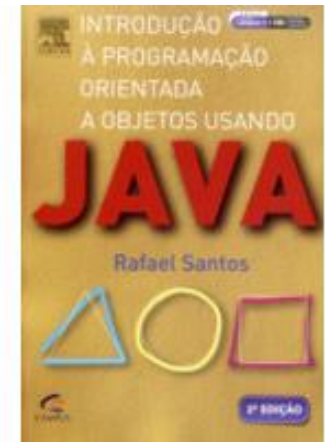
1. Se ano1 < ano2 a primeira data vem antes da segunda.
2. Se ano1 > ano2 a primeira data vem depois da segunda.
3. Se ano1 == ano2 e mês1 < mês2 a primeira data vem antes da segunda.
4. Se ano1 == ano2 e mês1 > mês2 a primeira data vem depois da segunda.
5. Se ano1 == ano2 e mês1 == mês2 e dia1 < dia2 a primeira data vem antes da segunda.
6. Se ano1 == ano2 e mês1 == mês2 e dia1 > dia2 a primeira data vem depois da segunda.
7. Se nenhum desses casos ocorrer, as datas são exatamente iguais.

Escreva um método vemAntes na classe Data (figura 2.7) que receba como argumento outra instância da classe Data e implemente o algoritmo acima, retornando true se a data encapsulada vier antes da passada como argumento e false caso contrário. Se as datas forem exatamente iguais, o método deve retornar true.

Referências



- ❏ SANTOS, R. Introdução à programação orientada a objetos usando JAVA. 2. ed. Rio de Janeiro: Campus, 2013. 336p. Capítulo 2.
- ❏ DEITEL, P.; DEITEL, H. Java: como programar. 6/8. ed. São Paulo: Pearson, 2010. 1176p.



Próxima Aula...



➤ Aula 05:

- ❑ **Laboratório:** Implementar os modelos de entidades vistos na Aula 02 e aqueles apresentados na lista de exercício dessa mesma aula.

