



Open-source Education →

Sintaxe da linguagem Java e Orientação a objetos

Iniciativa Globalcode

Mini-cursos Globalcode

MC1 – Introdução à plataforma Java

MC2 – Sintaxe da linguagem e orientação a objetos com Java

MC3 – Introdução à plataforma J2EE – Java 2 Enterprise Edition

MC4 – Desenvolvimento de aplicativos Web com Java

MC5 – J2EE modelando arquiteturas para demandas de 10 a mais de 10.000 usuários

MC6 – Java e mainframe: analogias, integrações e arquiteturas

MC7 – Metodologias de desenvolvimento para Java e UML

MC8 – Desenvolvimento Web com design-patterns e Struts

MC9 – Desenvolvimento de componentes Enterprise JavaBeans

MC10 – Planejamento e execução de stress-test

MC11 ao MC13 – Preparatórios para certificações Java

Agenda

1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

Agenda

1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

A Globalcode

The Developers Company

Educação treinamentos gratuitos, vídeo-aulas, palestras em empresas e universidades, cursos individuais, carreiras e serviços de consultorias pontuais e mentoring;

Pesquisa desenvolvimento de experiências com publicações em conferências internacionais - eXPerience Group -, convênio com ITA e IPEN;

Produção de software pequena fábrica de desenvolvimento de componentes Java, em expansão para 2006;

Palestrante / Instrutor

Vinicius Senger – vinicius@globalcode.com.br

- Sócio e fundador da Globalcode, foi instrutor e consultor da Sun e Oracle no Brasil;
- Trabalhou em projetos de grande porte em bancos. Começou a programar com 8 anos e trabalha com desenvolvimento de softwares profissionalmente desde os 13 anos;
- Certificações: Sun Java Programmer / Sun Enterprise Architect P1, Microsoft Certified Professional, Microsoft Certified Trainer;

Agenda

1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

Sintaxe da linguagem

- Alguma similaridade com a linguagem C: `if()`, `int x`; demarcação por `{}`, arrays com `[]`, etc.
- Permite escrita de código mais legível se compararmos com C / C++;
- Características de programação orientada a objetos são herdadas de diversas linguagens;

Sintaxe da linguagem

- A linguagem é case sensitive, ou seja, x é diferente de X;
- O uso de letras maiúsculas e minúsculas nos apresentam características do código (convenções);
- Blocos de código são colocados entre {};
- Ao término de cada instrução utilizamos ;
- Podemos definir uma instrução em mais de uma linha (texto livre);
- Normalmente colocamos uma classe por arquivo;

Agenda

1. Sintaxe da linguagem

- ✓ **Comentários**
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

Comentários

- Três formas:

- ✓ Na mesma linha:

- ```
// comentários não são compilados
```

- ✓ Com múltiplas linhas:

- ```
/* podemos utilizar  
mais de  
uma linha */
```

- ✓ Comentários com documentação JavaDoc

- ```
/**
 * @param conta
 */
```

# Agenda

---

## 1. Sintaxe da linguagem

- ✓ Comentários
- ✓ **Palavras reservadas**
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

## Palavras reservadas

|           |            |        |            |              |
|-----------|------------|--------|------------|--------------|
| abstract  | boolean    | break  | byte       | case         |
| catch     | char       | class  | const      | continue     |
| default   | do         | double | else       | extends      |
| final     | finally    | float  | for        | goto         |
| 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  |            |              |

# Agenda

---

## 1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ **Convenções**
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

## Convenções

---

- São regras de formatação de nomes de código;
- Não são obrigatórias, mas representam uma best-practice por aumentar a legibilidade do código;
- Identificamos características através das convenções: o que é uma classe, um método / função, uma variável, uma constante, etc.

## Convenções

- Convenção 1: nomes das classes com primeira letra da(s) palavra(s) maiúscula. Exemplo:

```
public class Conta { ... }
public class ContaCorrente { ... }
public class CaixaEletronico { ... }
```



## Convenções

---

- Convenção 2: nomes de variáveis e métodos (funções / procedures) com a primeira letra minúscula:

```
public class Conta {
 private double saldoDaConta;
 public void deposito(double valor) {
 ...
 }
}
```

variável

método / função / procedure

## Convenções

---

- Convenção 3: constantes devem ser definidas com todos caracteres maiúsculos:

```
public class Produto {
 public static final int COR_BRANCO = 0;
}
```

← constante

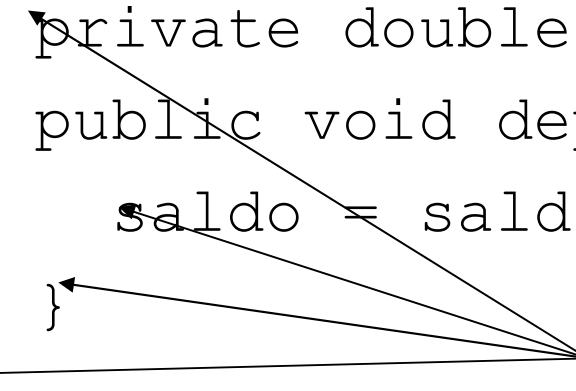
- Identação de código: tabulações (com 4 ou 2 espaços) devem ser abertas após a chave {e retroceder após o fechamento da chave}.

## Convenções

---

- Identação de código: tabulações (com 4 ou 2 espaços) devem ser abertas após a chave {e retroceder após o fechamento da chave}. Exemplo:

```
public class Conta {
 private double saldo;
 public void deposito(double valor) {
 saldo = saldo + valor;
 }
}
```



Identações com 2 espaços

# Agenda

---

## 1. Sintaxe da linguagem


- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ **Variáveis**
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

## Variáveis

---

- Representam um espaço de memória;
- Toda variável tem um tipo de dado e um nome;

```
public class Conta {
 private double saldo;
 public void deposito(double valor) {
 saldo = saldo + valor;
 }
}
```



## Variáveis

---

- As variáveis se dividem em dois grupos:
  - Primitivas: variáveis de “baixo nível”, como números inteiros, com casas decimais, caractere e booleanas (sim / não, verdadeiro / falso) ;
  - Compostas / Referência: representam uma estrutura de dado já preparada para um determinado fim como Data, Conta, Produto, Moeda, String / Cadeia de caracteres, etc.

## Variáveis

---

- Declaramos variáveis da seguinte forma:

```
public class OlaMundo {
 public static void main(String args[]) {
 //apenas declaramos
 int numeroInteiro;
 //declaramos e inicializamos
 char letra = 'a';
 double numeroComDecimal = 2.3456789;
 }
}
```

## Variáveis

- Variáveis primitivas: números inteiros

| Tipo  | Valor Mínimo              | Valor Máximo            | Bytes |
|-------|---------------------------|-------------------------|-------|
| byte  | -128                      | 127                     | 1     |
| short | - 32.768                  | 32.767                  | 2     |
| int   | - 2.147.483.648           | 2.147.483.647           | 4     |
| long  | - 922.337.203.685.475.808 | 922.337.203.685.475.807 | 8     |

```
short ano = 2005;
```

```
int contador = 10000;
```

```
long chaveCriptografia = 213455533223445432L;
```



## Variáveis

- Variáveis primitivas: números com ponto flutuante

| Tipo   | Mínimo        | Máximo       | Bytes |
|--------|---------------|--------------|-------|
| float  | $1.4e^{-45}$  | $3.4e^{38}$  | 4     |
| double | $4.9e^{-324}$ | $1.7e^{308}$ | 8     |

```
float saldo = 234.45F;
```

```
double cotacao = 2.34234556643213D;
```

## Variáveis

---

- Variáveis primitivas: caractere e booleanos
  - ✓ Caractere é representado pelo tipo **char**, utiliza dois bytes de memória pois trabalha com padrão unicode (tabela de caracteres para atender idiomas remotos);
  - ✓ Booleanos armazenam verdadeiro ou falso e utilizam apenas 1 bit

```
char letra = 'c';
boolean temSaldo = true;
```

## Variáveis

---

- Variáveis tipo String
  - ✓ É um tipo composto / classe presente na linguagem para manipular cadeias de caracteres;
  - ✓ Chama-se **String** e não **string**;
  - ✓ Inclui recursos de conversão para maiúsculo / minúsculo, recorte, pesquisa, entre outros;

```
String palavra = "Nome do Produto";
String recorte = palavra.substring(0, 4);
```

# Agenda

---

## 1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ Arrays

# Operadores

---

| Operador | Descrição                                              |
|----------|--------------------------------------------------------|
| +        | soma: <code>int resultado = x + y;</code>              |
| -        | subtração: <code>int resultado = x - y;</code>         |
| /        | divisão: <code>int resultado = x / y;</code>           |
| *        | multiplicação: <code>int resultado = x * y;</code>     |
| %        | módulo / resto da divisão: <code>int um = 10%3;</code> |
| ++       | auto-incremento: <code>x++; //x = x + 1;</code>        |
| --       | auto-decremento: <code>x--; //x = x - 1;</code>        |

# Operadores

---

| Operador                           | Descrição                                                                              |
|------------------------------------|----------------------------------------------------------------------------------------|
| <code>==</code>                    | operador lógico igual: <code>if (x==y)</code>                                          |
| <code>!=</code>                    | operador lógico diferente: <code>if (x!=y)</code>                                      |
| <code>&gt;= &lt;= &gt; &lt;</code> | operadores lógicos, maior igual, menor igual, maior e menor: <code>if (x&gt;=y)</code> |
| <code>&amp;&amp;</code>            | operador lógico and: <code>if (x&gt;y &amp;&amp; y&gt;100)</code>                      |
| <code>  </code>                    | operador lógico or: <code>if (x&gt;y    y&lt;10)</code>                                |

# Operadores

---

- Exemplos

```
int x = 10;
x++; // valor de x passa a ser 11
double resultado = (double) x / 3;
if(x>=10) {
 System.out.println("Valor maior que 10");
}
```

# Agenda

---

## 1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ **Condições e laços**
- ✓ Arrays



## Condições

---

- Podemos condicionar a execução de uma parte do código com operadores lógicos e a instrução **if**

```
if(condição) {
 }
else if(outra condicao) {
 }
else { //nenhuma das condições acima
 }
```

- Podemos utilizar apenas **if** sem **else if / else;**

## Condições

---

- Exemplo

```
System.out.println("Digite um ano:");
int ano = Teclado.lerNumeroInt();
if(ano==0) {
 System.out.println("Ano inválido!");
}
else {
 boolean resultado = (ano%4==0 &&
 ano%100!=0) || (ano%400==0);
}
```

## Laços

- Podemos repetir a execução de um determinado bloco de código;
- Exemplo: em uma nota fiscal, precisamos imprimir n produtos;
- Exemplo: em um extrato bancário, precisamos apresentar n movimentações no períodos,
- Temos dois tipos de laços:
  - ✓ Laço for: repete um bloco por n vezes;
  - ✓ Laço while: repete

## Laços

---

- Laço for: utilizado quando se conhece o número de execuções ou por constante ou por variável:

```
for(int contador=0;contador<100;contador++) {
 System.out.println("O valor do contador é:" +
 contador);
}
```

- Laço while: não conhecemos o número de vezes que queremos executar o código, apenas uma condição:

```
while(bancoDeDados.temRegistro()) {
 System.out.println(bancoDeDados.obterRegistro());
}
```

# Agenda

---

## 1. Sintaxe da linguagem

- ✓ Comentários
- ✓ Palavras reservadas
- ✓ Convenções
- ✓ Variáveis
- ✓ Operadores
- ✓ Condições e laços
- ✓ **Arrays**

## Arrays

---

- Uma forma de declarar múltiplas variáveis de uma só vez;
- Utilizado para representar coleções de dados;
- Exemplo:

**//sem array**

```
int x0 = 0, x1 = 1, x2 = 2, x3 = 3;
```

**//com array**

```
int x[] = new int[4];
```

```
x[0]=0;
```

```
x[1]=1;
```

```
x[2]=2;
```

```
x[3]=3;
```

## Arrays

---

- Tem tamanho fixo e não podemos redimensionar;

```
int x[] = new int[4];
```

- Podemos declarar arrays de qualquer tipo, incluindo String:

```
String palavras[] = new String[3];
palavras[0] = "palavra1";
palavras[1] = "palavra2";
```

## Arrays

---

- Exemplo de laços com arrays

```
String palavras[] = new String[3];
palavras[0] = "palavra1";
palavras[1] = "palavra2";
for(int pos=0;pos<palavras.length;pos++) {
 System.out.println(palavras[pos]);
}
```



# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Orientação a objetos

- Uma forma diferente de organizar código e bibliotecas;
- Parte do princípio que tudo é um objeto: uma data, um número, uma conta, um produto, um preço, um imóvel;
- Vantagens:
  - ✓ Aumenta a legibilidade do código;
  - ✓ Facilita a extensão / adição de novas funcionalidades;
  - ✓ Facilita a substituição de funcionalidades;
  - ✓ Aumenta o reuso de código-fonte;

# Agenda

---

## 1. Orientação a objetos

- ✓ **Classes e Objetos**
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Classes e objetos

- Um objeto deve representar no sistema o objeto do mundo real;
- Um objeto tem características, exemplo objeto Conta Corrente:
  - ✓ Número da conta;
  - ✓ Número da agência;
  - ✓ Saldo;
- Do ponto de vista de dados, um objeto é uma coleção de variáveis (estrutura / level-01)

## Classes e objetos

- Um objeto tem funcionalidades / operações, exemplo objeto Conta Corrente:
  - ✓ Depositar;
  - ✓ Sacar;
  - ✓ Transferir;
  - ✓ Alterar número da agência;
- Do ponto de vista de funcionalidades, um objeto é uma coleção de funções (módulo);

## **Classes e objetos**

---

- Um objeto, portanto, pode conter dados e funcionalidades que operam seus dados, exemplo Conta Corrente:
- Características / Atributos / Propriedades:
  - Número agência;
  - Número conta;
  - Data abertura;
  - Saldo;
- Funcionalidades / procedures / métodos:
  - Sacar
  - Depositar
  - Transferir

## Classes e objetos

---

- Devemos definir o formato (características e funcionalidades) de um objeto em um classe;
- Uma classe é um molde para n objetos;
- Uma classe é um template de objetos;
- Uma classe é a estrutura de um objeto;
- Um objeto é uma instancia de uma classe;
- Exemplos:
  - ✓ Classe Moeda, objetos real, dolar, pesos, etc.
  - ✓ Classe Agencia, objetos agência Paulista, agência Centro, agência Aclimação;

## **Classes e objetos**

---

- Analogia com Banco de dados:
  - classe = tabela
  - objeto = linha da tabela
- Analogia com Word:
  - classe = template de documento (.dot)
  - objeto = um documento criado a partir de template (.doc)
- UML (unified modeling language) é uma forma padrão de modelarmos graficamente uma classe;



# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ **Abstração**
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Abstração

---

- Abstrair significa transmitir as características relevantes do objeto no mundo real para o sistema;
- Exemplo: um produto pode conter características como matéria prima, validade, garantia, preço, fornecedores, quantidade em estoque;
- Na abstração de uma assistência técnica, matéria prima **não é relevante**;
- Na abstração de uma vendas on-line, preço, garantia e quantidade em estoque **são relevantes**;

## Abstração

---

- Podemos ter diversos níveis de abstração;
- Devemos trabalhar com abstração do simples para o complexo, do pequeno para o grande, do abstrato para o concreto;
- **Exemplo:** antes de criar uma classe ContaCorrente, devemos criar um classe Conta com as características comuns para todas as contas;
- **Exemplo:** antes de criar a classe ClientePessoaFisica, devemos criar uma classe Cliente com as características comuns para todas os clientes;

# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ **Atributos**
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Atributos

---

- Atributos definem as características de um objeto:

```
public class Data {
 int dia, mes, ano;
}

public class UsaData {
 public static void main(String args[]) {
 Data data1 = new Data();
 data1.dia=10;
 data1.mes=12;
 data1.ano=2005;
 }
}
```

## Atributos

---

```
public class Conta {
 int numero;
 double saldo;
}

public class UsaConta {
 public static void main(String args[]) {
 //cada conta tem seu número e saldo
 Conta conta1 = new Conta()
 conta1.numero=38379;
 Conta conta2 = new Conta()
 conta2.numero=12234;
 }
}
```

# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ **Métodos**
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Métodos

---

- Definem as funcionalidades do objeto:

```
public class Data {
 int dia, mes, ano;
 public boolean anoBissexto() {
 return (ano%4==0 && ano%100!=0) ||
 (ano%400==0);
 }
}
```

- Um método pode receber n parâmetros e retornar um único valor;



## Métodos

---

- Operam os dados do objeto em questão:

```
Data d1 = new Data();
Data d2 = new Data();
d1.ano=2005;
d2.ano=2004;
System.out.println(
 "é bissexto?" + d1.anoBissexto());
System.out.println(
 "é bissexto?" + d2.anoBissexto());
```

## Métodos

---

- Um método é declarado com a seguinte estrutura:

```
public class Conta {
 double saldo;
 public void deposito(double valor) {
 saldo = saldo + valor;
 }
}
```

Tipo de retorno:  
void = nenhum  
retorno

Argumento(s) de entrada:  
podemos receber mais que um  
argumento, separados por vírgula

| Memória         |   |
|-----------------|---|
| conta1<br>saldo | 0 |
| conta2<br>saldo | 0 |

```
public class Conta {
 double saldo;
 public void deposito(double valor) {
 saldo = saldo + valor;
 }
}
```

| Memória         |     |
|-----------------|-----|
| conta1<br>saldo | 100 |
| conta2<br>saldo | 200 |

```
Conta conta1 = new Conta();
Conta conta2 = new Conta();
conta1.saldo = 100;
conta2.saldo = 200;
conta1.deposito(10);
conta2.deposito(20);
```

| Memória         |     |
|-----------------|-----|
| conta1<br>saldo | 110 |
| conta2<br>saldo | 220 |

# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ **Encapsulamento**
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Encapsulamento

---

- Um objeto pode encapsular seus dados de forma que o usuário não terá acesso direto ao dado;
- Isso nos dá a oportunidade de efetuarmos validações durante as interações entre usuários X objetos;

```
/* Sem encapsulamento, podemos colocar inclusive um
* ano negativo
*/
```

```
Data d1 = new Data()
d1.ano = -2000;
```

## Encapsulamento

---

- Data com encapsulamento:

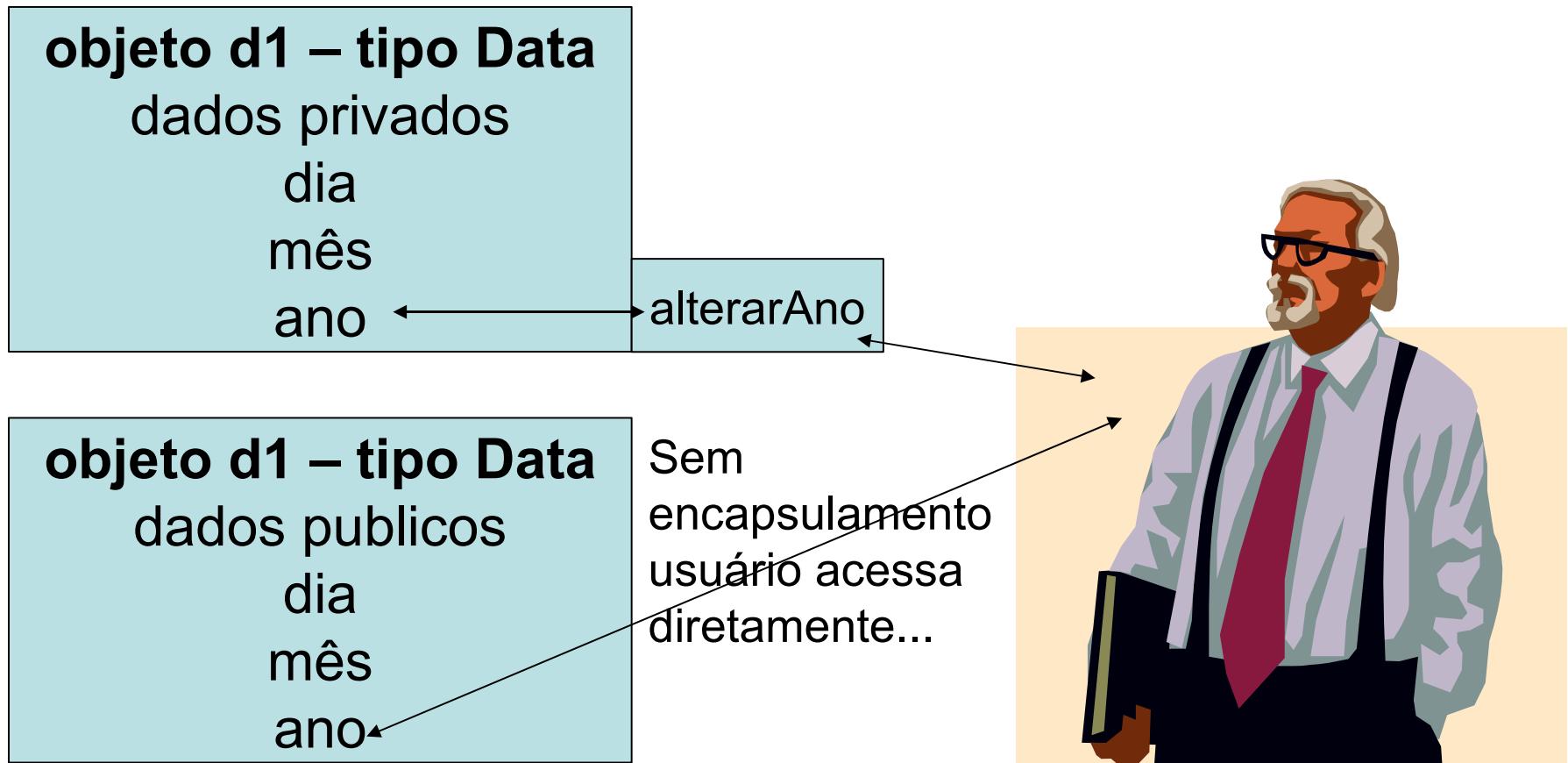
```
public class Data {
 private int dia, mes, ano;
 public void alterarAno(int novoAno) {
 if(novoAno<0)
 System.out.println("Ano não suportado...");
 else
 ano = novoAno;
 }
}
```

~~Data d1 = new Data()  
d1.ano = 2005;~~

Data d1 = new Data()  
d1.alterarAno(2005);

- Para encapsularmos um dado, devemos torná-lo privado, ou seja, o usuário não poderá ter acesso ao ano, dia e mês diretamente;

# Encapsulamento



# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança



## Sobrecarga

---

- Podemos escrever métodos em classes com mesmo nome e diferentes argumentos:
- Também chamado de polimorfismo estático;

```
/* Sem encapsulamento, podemos colocar inclusive um
 * ano negativo
 */
public class Calculadora {
 public int soma(int x, int y) {
 return x + y;
 }
 public float soma(float x, float y) {
 return x + y;
 }
}
```

# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Construtores

---

- Da mesma forma que podemos criar e inicializar uma variável, podemos definir formas de criar e inicializar objetos:

```
public class Data {
 private int dia, mes, ano;
 public Data(int dia1, int mes1, int ano1) {
 dia = dia1; mes = mes1; ano = ano1;
 }
}
Data d1 = new Data(); // antiga forma, com construtor
Data d1 = new Data(1,1,2005); //com construtor
```

- São métodos especiais, **sem tipo de retorno**, com o mesmo nome que a classe;

# Agenda

---

## 1. Orientação a objetos

- ✓ Classes e Objetos
- ✓ Abstração
- ✓ Atributos
- ✓ Métodos
- ✓ Encapsulamento
- ✓ Sobrecarga de métodos
- ✓ Construtores
- ✓ Herança

## Herança

---

- Podemos criar uma classe a partir de outra classe, ou seja, estendendo um classe antiga:

```
public class Conta {
 public double saldo;
 public boolean saque(double valor) {
 if(saldo<valor) return false;
 else {
 saldo = saldo - valor;
 return true;
 }
 }
}
```

## Herança

---

- Podemos criar uma classe a partir de outra classe, ou seja, estendendo um classe antiga:

```
public class ContaEspecial extends Conta {
 public double limite;
}
```

```
Conta c1 = new Conta();
c1.saldo = 200;
ContaEspecial c2 = new ContaEspecial();
c2.saldo = 100; //atributo herdado de conta
c2.limite = 1000; //exclusivo da ContaEspecial
```