

PARADIGMAS DE PROGRAMAÇÃO

Prof. Me. Kleber Padovani

padovani@ucdb.br

Breve história da linguagem

- 1991: um grupo de engenheiros da Sun acreditavam que a união de dispositivos digitais e computadores seria tendência na computação e criaram a linguagem Java.
- 1995: linguagem adaptada para o paradigma web
- Atualmente: Java está amplamente presente em diversas aplicações (web, desktop, mobile, etc.), o que pode ser interpretado como sinal de prosperidade da tecnologia.

Características da linguagem

- Programação orientada a objetos
- Portabilidade de código
- Desocupação automática de memória
- Suporte a diversas funcionalidades:
 - Recursos de rede
 - Segurança
 - Programação distribuída

Características da linguagem

- Utilização de máquina virtual (JVM – *Java Virtual Machine*)
 - Cada sistema operacional (SO) possui uma JVM específica.
 - Após a compilação, é gerado um arquivo intermediário, chamado *bytecode*, sem características específicas de SO.
 - Posteriormente, o bytecode é interpretado pela JVM, que o traduz para o seu respectivo SO e torna possível sua execução.

Características da linguagem

- Distribuída em 3 edições:
 - JSE (*Java Standard Edition*): indicada para desktop
 - JEE (*Java Enterprise Edition*): muito utilizada na web
 - JME (*Java Micro Edition*): aplicativos móveis
- Há dois ambientes para cada edição:
 - JRE (*Java Runtime Environment*): execução de programas
 - JDK (*Java Development Kit*): criação de programas
 - Comumente distribuída com JRE

Codificação

- Estrutura básica de um código-fonte em Java

```
public class < nome do 'programa' > {  
    public static void main(String[] args){  
        <implementação>  
    }  
}
```

- Exemplo: programa chamado *Main* para a impressão do texto “TNT Technology” na tela do usuário

```
public class Main {  
    public static void main(String[] args){  
        System.out.println("TNT Technology");  
    }  
}
```

Codificação

- O código-fonte pode ser escrito em qualquer editor de texto puro, como o *bloco de notas*, *notepad++*, *kwrite*, *vi* e outros.
- O arquivo texto onde o código foi escrito deve ter o mesmo nome escolhido para o programa e extensão *java**.
 - Exemplo: O código anterior deveria ser salvo como *Main.java*.

* Essa restrição não é rígida, mas, por ora, siga-a.

Compilação

- Para compilar o programa, utilizamos o aplicativo *javac* (fornecido no JDK) e informamos o nome do arquivo texto onde nosso código foi escrito.
 - Exemplo: *javac Main.java*
- A compilação é comumente disparada por linha de comando (prompt de comando no *Windows* ou o terminal no *Linux*, por exemplo).
 - Para evitar a digitação repetitiva do caminho completo onde o *javac* foi armazenado em seu computador, utilize a variável de ambiente *PATH*.

Execução

- De modo similar, para executar o programa, utilize o aplicativo *java* (contido no JRE), informando o nome dado ao programa.
 - Exemplo: `java Main`
 - Nesse instante, o texto “TNT Technology” será exibido ao usuário.

Declaração de variáveis

- Java é uma linguagem de variáveis tipadas, portanto, toda variável deve ser associada a um respectivo tipo e esse tipo não pode ser alterado.
- O tipo de uma variável pode ser primitivo ou uma referência. Por enquanto, veremos apenas os tipos primitivos.
- No total, temos 8 tipos primitivos, sendo 4 para números inteiros, 2 para números reais, 1 para caracter e outro para valores lógicos.

Declaração de variáveis

- Tipos primitivos e seus respectivos tamanhos

Inteiros	
Tipo	Bytes
byte	1
short	2
int	4
long	8

Reais	
Tipo	Bytes
float	4
double	8

Caracter	
Tipo	Bytes
char	2

Lógico	
Tipo	Bytes
boolean	JVM-defined

Declaração de variáveis

- Restrições/considerações na nomeação de variáveis:
 - Existe diferença entre maiúsculas e minúsculas
 - Não se usam espaços
 - Não há limite máximo no tamanho da variável
 - Podem começar com uma letra, *underscore* ou cifrão
 - Os caracteres seguintes podem ser letras, números, *underscore* e cifrão.
 - Por convenção, utiliza-se *camelCase*.
 - Não podem ser utilizadas palavras reservadas

Declaração de variáveis

- Palavras reservadas (apenas curiosidade):
 - abstract, continue, for, new, switch, assert, default, package, synchronized, boolean, do, if, private, this, break, double, implements, protected, throw, byte, else, import, public, throws, case, enum, instanceof, return, transient, , catch, extends, int, short, try, char, final, interface, static, void, class, finally, long, strictfp, volatile, float, native, super e while

Declaração de variáveis

- Sintaxe para declaração de variáveis:

`<tipo da variável> <nomes das variáveis separadas por vírgulas>;`

– Exemplos:

```
int variavel1, variavel2;  
float variavel3;  
double variavel4, variavel5, variavel6;  
int variavelEspecial;
```

Declaração de variáveis

- Sintaxe para atribuição de valores a variáveis:

```
<nome da variável> = <valor>;
```

- O valor a ser atribuído pode ser um literal ou uma variável (atente-se à compatibilidade de tipos).

– Exemplos:

```
variavel1 = 5;  
variavel3 = 0.5f;  
variavel4 = 3.14;  
variavelChar = 'a';  
variavelBoolean1 = true;  
variavelBoolean2 = false;
```

Declaração de variáveis

- Declaração e atribuição simultâneas:

`<tipo da variável> <nome da variável> = <valor>;`

- Exemplos:

```
int variavel1, variavel2 = 3;
float variavel3;
double variavel4, variavel5, variavel6;
char variavelChar = 'c';
```


Operações aritméticas

- Temos 5 operações aritméticas binárias, que podem ser aplicadas a qualquer combinação entre variável e literal (atente-se aos tipos dos operandos), sendo elas representadas pelos seguintes símbolos:
 - Adição: +
 - Subtração: -
 - Multiplicação: *
 - Divisão: /
 - Resto da divisão: %

Operações aritméticas

- Especificamente a operação de divisão possui dois comportamentos distintos:
 - Divisão inteira
 - Apresenta como resultado apenas a porção inteira
 - Ocorre quando ambos os operandos são de natureza inteira
 - Divisão fracionada
 - Apresenta o resultado real (incluindo a parte fracionada)
 - Ocorre quando um dos operandos de natureza real (ou ambos são)

Operações aritméticas

- Exemplos de divisão:

- Divisões inteiras

```
float fr1 = 7 / 2;  
int ir = 7 / 2;  
double dr1 = 7 / 2;
```

- Note que, mesmo que a variável receptora seja de natureza real, são os operandos que definem o comportamento da divisão.

- Divisões fracionadas

```
float fr2 = 7 / 2.0f;  
float fr3 = 7.0f / 2;  
float fr4 = 7.0f / 2.0f;  
double dr2 = 7.0 / 2;
```

Operações aritméticas

- Atente-se à precedência dos operadores
 - Primeiro as multiplicações e divisões ($*$, $/$ e $\%$) e depois as adições e subtrações ($+$ e $-$)
 - Quando lidamos com a mesma precedência, as operações são realizadas da esquerda para a direita

Operações aritméticas

- Atente-se à precedência dos operadores
 - Exemplo 1: em vez de 4, teremos 5.33... como resultado.

```
double nota1 = 1, nota2 = 1, nota3 = 10;  
double media = nota1 + nota2 + nota3 / 3;  
System.out.println(media);
```

- Exemplo 2: em vez de 24,22, resultado será 70.

```
double peso = 70, altura = 1.7;  
double imc = peso / altura * altura;  
System.out.println(imc);
```

Operações aritméticas

- Atente-se à precedência dos operadores
 - Exemplo 1 corrigido:

```
double nota1 = 1, nota2 = 1, nota3 = 10;  
double media = (nota1 + nota2 + nota3) / 3;  
System.out.println(media);
```

- Exemplo 2 corrigido:

```
double peso = 70, altura = 1.7;  
double imc = peso / (altura * altura);  
System.out.println(imc);
```

Operações unárias

- Os seguintes operadores são aplicados a um único elemento e, por isso, são chamados de operadores unários:
 - Incremento: ++ (incrementa em 1 o elemento)
 - Decremento: -- (decrementa em 1 o elemento)
 - Complemento: ! (inverte o valor lógico do elemento)
 - Sinal negativo: - (indica que o elemento é negativo)
 - Sinal positivo: + (indica que o elemento é positivo)

Operações unárias

- Observações

- Incremento e decremento

- Aplicável apenas a variáveis

- Podem anteceder ou suceder a variável

- Antes: incrementa antes da leitura

```
int var = 5;  
System.out.println(++var);
```



Resultado: 6

- Depois: incrementa após a leitura

```
int var = 5;  
System.out.println(var++);
```



Resultado: 5

Operações unárias

- Observações
 - Complemento
 - Aplicável apenas a valores lógicos, sendo variáveis ou literais
 - Sinal negativo e positivo
 - Aplicáveis a números inteiros e reais, sendo variáveis ou literais
 - O sinal positivo torna-se redundante, pois o número sem sinal é considerado positivo.

Atribuições compostas

- Podemos combinar operações aritméticas com o operador de atribuição, formando atribuições compostas, que reutilizam o valor anterior da variável que será atribuída:
 - Exemplo: ao final, conteúdo da variável será 11.75

```
double variavel = 1.0;  
variavel += 2.5; //variavel=3.5  
variavel *= 7.0; //variavel=24.5  
variavel -= 1.0; //variavel=23.5  
variavel /= 2.0; //variavel=11.75  
System.out.println(variavel);
```

Operações relacionais

- Operadores utilizados:
 - Maior que: $>$
 - Maior que ou igual a: $>=$
 - Menor que: $<$
 - Menor que ou igual a: $<=$
 - Igual: $==$
 - Diferente: $!=$

Operações condicionais

- Operadores utilizados:
 - E lógico: `&&`
 - OU lógico: `||`
 - Ternário: `?:` (exemplo: `a = b > 3 ? 1 : 2`)
- Atente-se à precedência dos operadores:
 - Primeiro o E, depois o OU.

Estruturas condicionais

- Estrutura 1: if-else
 - Executa apenas um dentre vários trechos de código, de acordo com condições definidas.
 - Sintaxe: inicia com *if*, segue com 0 ou mais *else if* e, por fim, 0 ou 1 *else*.

```
if ( <condição> ) {  
    <Primeiro trecho de código>  
} else if ( <condição2> ){  
    <Segundo trecho de código>  
} else{  
    <Terceiro trecho de código>  
}
```

Estruturas condicionais

- Estrutura 1: if-else
 - Exemplo 1: calcular valor atualizado de uma conta com base nos dias de atraso, valor da conta e percentual de multa.

```
int diasDeAtraso = 5;
double valorConta = 1000, percentualDeMulta = 2;
double valorAtualizado = valorConta;
if (diasDeAtraso > 0){
    valorAtualizado += valorConta * percentualDeMulta / 100;
}
System.out.println(valorAtualizado);
```

Estruturas condicionais

- Estrutura 1: if-else
 - Exemplo 2: identificar se o conteúdo de uma variável inteira é negativo.

```
int variavel = 5;
if ( variavel < 0 ){
    System.out.println("Inteiro negativo!");
}else{
    System.out.println("Inteiro positivo ou neutro!");
}
```

Estruturas condicionais

- Estrutura 1: if-else
 - Exemplo 3: identificar se um aluno está aprovado, reprovado ou de exame, com base em sua nota final

```
double notaFinal = 6.5;
if ( notaFinal < 4.0 ){
    System.out.println("Reprovado!");
} else if ( notaFinal < 7.0 ){
    System.out.println("Exame!");
} else{
    System.out.println("Aprovado!");
}
```


Estruturas condicionais

- Estrutura 2: switch-case
 - Similar ao if-else, porém, utiliza uma “variável de controle”, que (considerando apenas primitivos) pode ser do tipo byte, short, char ou int.

Estruturas condicionais

- Estrutura 2: switch-case
 - Sintaxe:

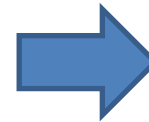
```
switch( <variável de controle> ){  
    case <valor da variável>:  
        <trecho de código>  
        break;  
    case <valor da variável>:  
        <trecho de código>  
        break;  
    ...  
    default:  
        <trecho de código>  
}
```

A utilização do caso padrão (o *default*) é facultativa.

Estruturas condicionais

- Estrutura 2: switch-case
 - Exemplo 1: escrever um número (de 1 a 4) por extenso

```
int numero = 3;
switch( numero ){
    case 1:
        System.out.println("Um");
        break;
    case 2:
        System.out.println("Dois");
        break;
    case 3:
        System.out.println("Três");
        break;
    case 4:
        System.out.println("Quatro");
        break;
    default:
        System.out.println("Desconhecido");
}
```

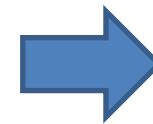


Resultado: Três

Estruturas condicionais

- Estrutura 2: switch-case
 - Exemplo 1: escrever um número (de 1 a 4) por extenso

```
int numero = 3;
switch( numero ){
    case 1:
        System.out.println("Um");
    case 2:
        System.out.println("Dois");
    case 3:
        System.out.println("Três");
    case 4:
        System.out.println("Quatro");
    default:
        System.out.println("Desconhecido");
}
```



Resultado:
Três
Quatro
Desconhecido

Estruturas de repetição

- Estrutura 1: while
 - Executa determinado trecho de código enquanto uma condição for verdadeira.
 - Se a condição for verdadeira, executa o trecho de código e avalia novamente a condição até seu resultado ser falso.
 - Sintaxe:

```
while ( <condição> ){  
    <trecho de código>  
}
```

Estruturas de repetição

- Estrutura 1: while
 - Exemplo 1: imprimir todos os inteiros positivos menores que 10

```
int i = 1;  
while ( i <= 9 ) {  
    System.out.println(i++);  
}
```



Resultado:

1
2
3
4
5
6
7
8
9

Estruturas de repetição

- Estrutura 1: while
 - Exemplo 2: imprimir a tabuada do número 7

```
int i = 0;
while ( i <= 10 ){
    int resultado = 7 * i++;
    System.out.println(resultado);
}
```



Resultado:

0
7
14
21
...
56
63
70

Estruturas de repetição

- Estrutura 1: while
 - Exemplo 3: calcular a exponenciação de um número

```
int base = 7, expoente = 0, i = 0;
int resultado = 1;
while ( i < expoente ){
    resultado *= base;
    i++;
}
System.out.println(resultado);
```



Resultado: 1

Estruturas de repetição

- Estrutura 2: do-while
 - Similar ao comando while, porém, inverte a ordem de execução (primeiro executamos o trecho de código e, depois, avaliamos a condição).
 - Executa determinado trecho de código enquanto uma condição for verdadeira.
 - Executa o trecho de código e, em seguida, avalia a condição. Se a condição for verdadeira, executa novamente o trecho e volta a avaliar a condição até que ela seja falsa.

Estruturas de repetição

- Estrutura 2: do-while
 - Sintaxe:

```
do{  
    <trecho de código>  
} while ( <condição> );
```

Estruturas de repetição

- Estrutura 2: do-while
 - Exemplo 1: imprimir todos os inteiros positivos menores que 10

```
int i = 1;  
do{  
    System.out.println(i++);  
} while ( i <= 9 );
```



Resultado:

1
2
3
4
5
6
7
8
9

Estruturas de repetição

- Estrutura 2: do-while
 - Exemplo 2: imprimir a tabuada do número 7

```
int i = 0;
do{
    int resultado = 7 * i++;
    System.out.println(resultado);
} while ( i <= 10 );
```



Resultado:

0
7
14
21
...
56
63
70

Estruturas de repetição

- Estrutura 2: do-while
 - Exemplo 3: calcular a exponenciação de um número

```
int base = 7, expoente = 0, i = 0;
int resultado = 1;
do{
    resultado *= base;
    i++;
} while ( i < expoente );
System.out.println(resultado);
```



Resultado: 7
(incoerente)

Estruturas de repetição

- Estrutura 3: for
 - Definida por 3 trechos de código:
 - Iniciação de variáveis
 - Condição de parada
 - Incremento
 - Sintaxe:

```
for ( <iniciação> ; <condição de parada> ; <incremento> ){  
    <trecho de código>  
}
```

Estruturas de repetição

- Estrutura 3: for
 - Tradução do *for* para *while*

```
for ( <iniciação> ; <condição de parada> ; <incremento> ){  
    <trecho de código>  
}
```



```
<iniciação> ;  
while( <condição de parada> ){  
    <trecho de código>  
    <incremento> ;  
}
```

Estruturas de repetição

- Estrutura 3: for
 - Exemplo 1: imprimir todos os inteiros positivos menores que 10

```
for ( int i = 1 ; i <= 9 ; i++ ) {  
    System.out.println(i);  
}
```



Resultado:

1
2
3
4
5
6
7
8
9

Estruturas de repetição

- Estrutura 3: for
 - Exemplo 2: imprimir a tabuada do número 7

```
for ( int i = 0 ; i <= 10; i++ ){  
    int resultado = 7 * i;  
    System.out.println(resultado);  
}
```



Resultado:

0
7
14
21
...
56
63
70

Estruturas de repetição

- Estrutura 3: for
 - Exemplo 3: calcular a exponenciação de um número

```
int base = 7, expoente = 0;  
int resultado = 1;  
for ( int i = 0 ; i < expoente ; i++) {  
    resultado *= base;  
}  
System.out.println(resultado);
```



Resultado: 1

Estruturas de repetição

- Em todas as estruturas de repetição, podemos utilizar os comandos break e continue com os seguintes objetivos:
 - break: encerra a repetição (“simula” um resultado falso para a condição no momento em que é executado)
 - continue: encerra a iteração atual e força a avaliação da condição de parada

Estruturas de repetição

- Uso do break/continue
 - Exemplo 1: código (rude) para imprimir ímpares menores que 10

```
for (int i = 0; i <= 10; i++) {  
    if (i % 2 == 0) {  
        continue;  
    }  
    System.out.println(i);  
}
```

Estruturas de repetição

- Uso do break/continue
 - Exemplo 2: código para encontrar o maior divisor de um número (exceto ele mesmo)

```
int numero = 175;
for (int i = numero - 1; i > 0; i--){
    if (numero % i == 0){
        System.out.println(i);
        break;
    }
}
```



Resultado: 35

Modulação

- Java permite a modulação de código (ou seja, a criação de módulos com trechos específicos de código).
- Além de organização, a modulação permite a reutilização de código e facilidades na sua manutenção.
- Existem dois tipos de módulos que podem ser criados: função ou procedimento.

Modulação

- De forma geral, função e procedimento são equivalentes (ambos executam determinado trecho de código com base em informações de entrada) e, geralmente, são referidos somente por função/método.
- Porém, além de executar um trecho de código, a função permite o retorno de uma informação, o que não é possível no procedimento.

Modulação

- Existe uma sintaxe geral que atende à criação de funções e procedimentos, que é a seguinte:

```
public static <tipo de retorno> <nome da função>( <parâmetros> ){  
    <trecho de código>  
}
```

- No caso de procedimento, o tipo de retorno deve ser *void* (como ocorre na *main*). Em função, deve ser o tipo do retorno (como *int*, *double*, etc.).

* Os termos *public* e *static* não são obrigatórios na construção de módulos. Mas, por ora, vamos supor que são. Na próxima aula, veremos suas particularidades em detalhes.

Modulação

```
public static <tipo de retorno> <nome da função>( <parâmetros> ){  
    <trecho de código>  
}
```

- O nome da função deve respeitar as mesmas regras de nomeação de variáveis e, por convenção, iniciam-se em minúsculas com um verbo.
- Os parâmetros da função são separados por vírgulas e associados a seu respectivo tipo (mesmo que o tipo dos parâmetros são os mesmos).

Modulação

- Exemplo 1: procedimento para exibir um número inteiro na tela

```
public static void imprimirResultado( int resultado ){  
    System.out.println(resultado) ;  
}
```

Modulação

- Para executar um método criado, basta escrever seu nome e informar os valores de seus parâmetros.
 - Exemplo:

```
public static void main(String args[]) {  
    imprimirResultado(7);  
}
```

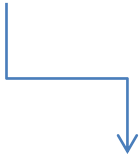
```
public static void imprimirResultado( int resultado ) {  
    System.out.println(resultado);  
}
```

Modulação

- Exemplo 2: função para somar 2 números (com sua respectiva chamada)

```
public static void main(String args[]){  
    double r = somarNumeros(8.5, 3.7);  
    System.out.println(r);  
}
```

```
public static double somarNumeros( double valor1, double valor2 ){  
    return valor1 + valor2;  
}
```



Comando para retornar o valor da função.

Modulação

- Ao encontrar a palavra reservada *return*, o método é automaticamente encerrado (mesmo que existam mais instruções seguintes a ela).
 - Exemplo 1: função para somar ou subtrair 2 números

```
public static double somarOuSubtrairNumeros( double v1, double v2, boolean somar ){  
    if (somar){ // equivalente a (somar == true)  
        return v1 + v2;  
    }  
    return v1 - v2;  
}
```

Modulação

- Ao encontrar a palavra reservada *return*, o método é automaticamente encerrado (mesmo que existam mais instruções seguintes a ela).
 - Exemplo 2: procedimento que mostra um número ou seu quadrado

```
public static void imprimirNumeroOuQuadrado( int num, boolean quadrado ){  
    if (quadrado){  
        System.out.println(num * num);  
        return;  
    }  
    System.out.println(num);  
    return;  
}
```

Modulação

- Exemplo:
 - Criar um programa que realiza soma, subtração e multiplicação de dois números utilizando somente o operador de soma e em apenas um local do código.

Laboratório

1. Escreva uma função que recebe o peso e a altura de uma pessoa e retorna o valor 0 se a pessoa estiver com o peso ideal, -1 se estiver abaixo do peso ideal e 1 se estiver acima do peso ideal. O peso ideal de uma pessoa deve ser maior que, ou igual a, $(18.5 * altura * altura)$ e menor que, ou igual a, $(25 * altura * altura)$. Por exemplo, uma pessoa que mede 1,6m, deve ter o peso entre 47,36 e 64 quilos.

Laboratório

2. Escreva uma função que retorna a média entre o maior e o menor valor de um vetor de inteiros recebido por parâmetro sem utilizar estruturas de repetição. Por exemplo, se esta função receber o vetor com os elementos $[7, 3, 9, 4, 2]$, o retorno será 5.5, ou seja, $(9+2)/2=5.5$.

Laboratório

3. Criar uma função que retorna o maior número contido em um vetor de inteiros recebido por parâmetro sem utilizar estruturas de repetição.
4. Criar um procedimento para mostrar em ordem crescente todos os números primos menores que um número inteiro positivo informado sem utilizar estruturas de repetição.

Laboratório

5. Crie um procedimento para mostrar em ordem crescente todos os números pares menores que (ou iguais a) o parâmetro recebido sem utilizar estrutura de repetição, sem exceder um parâmetro no procedimento e sem métodos auxiliares.

Não serão aceitas soluções que utilizam conceitos não vistos em sala

Estruturas homogêneas

- Até o momento, trabalhamos apenas com variáveis simples, que suportam o armazenamento de um único valor.
- No entanto, existem estruturas que permitem o armazenamento de mais de uma informação, desde que elas sejam do mesmo tipo.
- Essa estrutura é comumente chamada de vetor ou matriz.

Estruturas homogêneas

- A estrutura pode ter várias dimensões (uma, duas, três, etc.). Os vetores possuem dimensão 1 e as matrizes possuem dimensão 2.
- Ao criar uma estrutura homogênea, é necessário definir, além de sua dimensão, a quantidade de elementos por dimensão, o que vai definir o número total de elementos que essa estrutura armazenará.

Estruturas homogêneas

- Exemplo 1: vetor (dimensão 1) com 5 elementos

1º elem.	2º elem.	3º elem.	4º elem.	5º elem.
----------	----------	----------	----------	----------

- Exemplo 2: matriz (dimensão 2) com 3 elementos em uma dimensão e 4 na outra

	3		
4			

Estruturas homogêneas

- Para acessar cada elemento da estrutura, precisamos utilizar seu índice na respectiva dimensão.
- Em cada dimensão, o índice inicia em zero e termina na quantidade de elementos da dimensão menos 1.
 - Exemplo 1: os índices de um vetor com 5 elementos variam de 0 a 4

0	1	2	3	4
---	---	---	---	---

Estruturas homogêneas

- Em cada dimensão, o índice inicia em zero e termina na quantidade de elementos da dimensão menos 1.
 - Exemplo 2: os índices de uma matriz com 3 elementos na primeira dimensão e 4 na outra variam de 00 a 23

	0	1	2
0	00	10	20
1	01	11	21
2	02	12	22
3	03	13	23

Estruturas homogêneas

- Sintaxes de declaração de vetores:

```
<tipo dos elementos> <nome da estrutura>[];
```

```
<tipo dos elementos>[] <nome da estrutura>;
```

- Sintaxes de declaração de matrizes:

```
<tipo dos elementos> <nome da estrutura>[][];
```

```
<tipo dos elementos>[][] <nome da estrutura>;
```

* Note que a quantidade de colchetes define a dimensão da estrutura.

Estruturas homogêneas

- Exemplo de declaração de dois vetores de inteiros

```
int vetor1[];  
int[] vetor2;
```

- Exemplo de declaração de duas matrizes de inteiros

```
int[][] matriz1;  
int matriz2[][];
```

Estruturas homogêneas

- Apenas declarando a estrutura, não podemos ainda utilizá-la. Antes, é necessário criar seus elementos (ou seja, definir o tamanho de cada dimensão).
- Sintaxe de criação de vetores:

```
<nome da estrutura> = new <tipo dos elementos>[tamanho];
```

- Exemplo de criação de vetor de inteiros de 5 elementos

```
vetor1 = new int[5];
```

Estruturas homogêneas

- Sintaxe de criação de matrizes:

```
<nome da estrutura> = new <tipo dos elementos>[tamanho1][tamanho2];
```

- Exemplo de criação de matriz de inteiros de 3 elementos em uma dimensão e 4 na outra

```
matriz1 = new int[3][4];
```

Estruturas homogêneas

- Após a criação dos elementos de uma estrutura homogênea, os valores de seus elementos são automaticamente configurados para o valor padrão do respectivo tipo da estrutura.
- Valor padrão para cada tipo primitivo:
 - byte, short, int e long → 0
 - float e double → 0.0
 - boolean → false
 - char → caracter 0 da UTF-16

Estruturas homogêneas

- Desse modo, em Java, a criação da estrutura abaixo teria o seguinte resultado:

```
vetor1 = new int[5];
```



Estruturas homogêneas

- Para escrever ou ler um valor em um elemento da estrutura homogênea, utilizamos seu respectivo índice entre colchetes após o nome da variável.
 - Exemplo:

```
vetor1[3] = 9;  
System.out.println(vetor1[3]);
```

0	0	0	9	0
---	---	---	---	---

Estruturas homogêneas

- Após a criação de um vetor, é possível identificar seu tamanho utilizando a palavra reservada `length` após o nome da estrutura.
 - Exemplo:

```
int tamanho = vetor1.length;  
System.out.println(tamanho);
```


Estruturas homogêneas

- Para simplificação do código, podemos atribuir todos os elementos de uma estrutura homogênea simultaneamente. Para isso, não especificamos seu tamanho na sua construção e inserimos os elementos entre chaves separados por vírgulas.

– Exemplo:

```
int[] vetor;  
vetor = new int[] {5,7,3,2};
```



```
int[] vetor;  
vetor = new int[4];  
vetor[0] = 5;  
vetor[1] = 7;  
vetor[2] = 3;  
vetor[3] = 2;
```

Passagem de parâmetros

- Os parâmetros informados aos métodos são passados de duas formas: por valor ou por referência.
- Ao passar um parâmetro por valor, é criada uma cópia do valor passado, preservando o elemento original utilizado (uma variável inteira, por exemplo).
- Ao passar por referência, o método utiliza a mesma referência que foi informada, sem alocar outro espaço próprio e copiar o valor, como feito anteriormente.

Passagem de parâmetros

- Exemplo de passagem de parâmetro considerando que o parâmetro foi enviado por valor:

```
public static void main(String args[]){  
    int var = 5;  
    mostrarQuadrado(var);  
    System.out.println(var);  
}
```



Resultado:
25 (em *mostrarQuadrado*)
5 (em *main*)

```
public static void mostrarQuadrado(int numero){  
    numero *= numero;  
    System.out.println(numero);  
}
```

Passagem de parâmetros

- Exemplo (hipotético) de passagem de parâmetro por referência:

```
public static void main(String args[]){  
    int var = 5;  
    mostrarQuadrado(var);  
    System.out.println(var);  
}
```



Resultado:
25 (em *mostrarQuadrado*)
25 (em *main*)

```
public static void mostrarQuadrado(int numero){  
    numero *= numero;  
    System.out.println(numero);  
}
```

Passagem de parâmetros

- Em Java, todo parâmetro é enviado por valor.
 - Exemplo:

```
public static void main(String args[]){  
    int var = 5;  
    mostrarQuadrado(var);  
    System.out.println(var);  
}  
public static void mostrarQuadrado(int numero){  
    numero *= numero;  
    System.out.println(numero);  
}
```



Resultado:
25 (em *mostrarQuadrado*)
5 (em *main*)

Passagem de parâmetros

- Porém, nas estruturas homogêneas, copia-se o endereço da estrutura, e não seus elementos.
 - Exemplo:

```
public static void main(String args[]){  
    int vetor[] = new int[5];  
    vetor[3] = 9;  
    mostrarQuadrado(vetor, 3);  
    System.out.println(vetor[3]);  
}
```



Resultado:

81 (em *mostrarQuadrado*)
81 (em *main*)

```
public static void mostrarQuadrado(int[] estrutura, int indice){  
    estrutura[indice] *= estrutura[indice];  
    System.out.println(estrutura[indice]);  
}
```

Passagem de parâmetros

- Porém, nas estruturas homogêneas, copia-se o endereço da estrutura, e não seus elementos.
 - Exemplo:

```
public static void main(String args[]){  
    int vetor[] = new int[5];  
    vetor[3] = 9;  
    mostrarQuadrado(vetor, 3);  
    System.out.println(vetor[3]);  
}
```



Resultado:

81 (em *mostrarQuadrado*)
81 (em *main*)

```
public static void mostrarQuadrado(int[] estrutura, int indice){  
    estrutura[indice] *= estrutura[indice];  
    System.out.println(estrutura[indice]);  
    estrutura = new int[5];  
    estrutura[3] = 9;  
}
```

Passagem de parâmetros

- Exemplo: Criar um programa para calcular e imprimir o valor do salário a ser recebido pelos funcionários de uma empresa após o cálculo do bônus sabendo-se o valor do salário de cada um e supondo que:
 - Funcionários que recebem até 1000 reais (inclusive) ganham bônus de 8% de seu salário
 - Funcionários que recebem até 2000 reais (inclusive) , mas mais que 1000 reais, ganham bônus de 5% de seu salário
 - Os demais ganham 3% de bônus

Passagem de parâmetros

- Exemplo: implementação da *main*

```
public class ExemploFinalParteI{  
    public static void main(String args[]){  
        double[] salarios = obterSalarios();  
        calcularBonus(salarios);  
        imprimirSalarios(salarios);  
    }  
    ...  
}
```

Passagem de parâmetros

- Exemplo: implementação de *obterSalarios*

```
        calcularBonus(salarios),  
        imprimirSalarios(salarios);  
    }  
    public static double[] obterSalarios(){  
        double[] ordenado = new double[5];  
        ordenado[0] = 800.0;  
        ordenado[1] = 900.0;  
        ordenado[2] = 1350.0;  
        ordenado[3] = 2000.0;  
        ordenado[4] = 2300.0;  
        return ordenado;  
    }  
}
```

Passagem de parâmetros

- Exemplo: relembando a *main*

```
public class ExemploFinalParteI{  
    public static void main(String args[]){  
        double[] salarios = obterSalarios();  
        calcularBonus(salarios);  
        imprimirSalarios(salarios);  
    }  
    ...  
}
```

Passagem de parâmetros

- Exemplo: implementação de *calcularBonus*

```
ordenado[3] = 2000.0;  
ordenado[4] = 2300.0;  
return ordenado;  
}  
  
public static void calcularBonus(double[] valores) {  
    for (int i = 0; i < valores.length; i++) {  
        if (valores[i] <= 1000.0) {  
            valores[i] += valores[i] * 0.08;  
        } else if (valores[i] <= 2000.0) {  
            valores[i] += valores[i] * 0.05;  
        } else {  
            valores[i] += valores[i] * 0.03;  
        }  
    }  
}
```

Passagem de parâmetros

- Exemplo: relembrando a *main*

```
public class ExemploFinalParteI{  
    public static void main(String args[]){  
        double[] salarios = obterSalarios();  
        calcularBonus(salarios);  
        imprimirSalarios(salarios);  
    }  
    ...  
}
```

Passagem de parâmetros

- Exemplo: implementação de *imprimirSalarios*

```
        } else {  
            valores[i] += valores[i] * 0.03;  
        }  
    }  
}  
  
public static void imprimirSalarios(double[] salarios) {  
    int i = 0;  
    while (i < salarios.length) {  
        System.out.println(salarios[i++]);  
    }  
}  
}
```

Laboratório

1. Crie uma função em um programa para calcular e retornar o número de vogais contidas em um vetor de caracteres recebido por parâmetro.
2. Crie um programa em que o usuário informa uma quantidade de números inicialmente determinada por ele. Esses números devem ser enviados a um procedimento que inverte a ordem digitada pelo usuário e, em seguida, enviados para outro que os exibe nessa nova ordem.

Laboratório

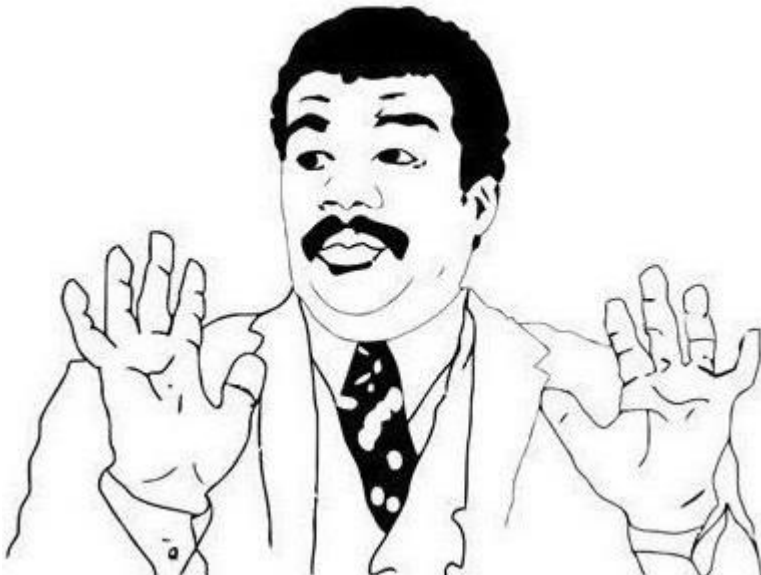
3. Crie um programa para classificar o resultado de um conjunto de alunos em uma disciplina com base nas notas obtidas. Como saída, deve ser informado ao usuário, para cada aluno, o número do seu RA e seu resultado (aprovado, reprovado ou de exame). O método main do seu programa deve ser EXATAMENTE como segue:

Laboratório

```
public static void main(String args[]) {  
    int[] ras = obterNumerosDeRA();  
    double[][] notas = obterNotasNasProvas();  
    for (int i = 0; i < ras.length; i++) {  
        double media = calcularMedia(notas, i);  
        System.out.println(ras[i]);  
        if (media >= 7) {  
            System.out.println("Aprovado");  
        } else if (media >= 4) {  
            System.out.println("Exame");  
        } else {  
            System.out.println("Reprovado");  
        }  
    }  
}
```

Programação orientada a objetos

Uuuuii... Programação orientada a objetos



Programação orientada a objetos com Java

- Java oferece **suporte à programação orientada a objetos** (POO), que, basicamente, trata-se de um mecanismo moderno que ajuda a **definir a estrutura dos programas** baseando-se nos conceitos do mundo real (concretos ou abstratos) **por meio de objetos**.
- A POO, se bem utilizada, pode oferecer **benefícios** aos desenvolvedores, como o **reuso** de código, facilidade de **manutenção**, maior **organização**, etc.

Programação orientada a objetos com Java

- Na POO, como o nome sugere, os **objetos são parte fundamental dos programas** construídos.
- A grosso modo, os programas se resumem a diversos **objetos se comunicando** e, conseqüentemente, **realizando as tarefas desejadas**.
- Para a manipulação efetiva dos objetos, a compreensão de **conceitos básicos** da OO – como classe, encapsulamento e herança – torna-se relevante.

Classes

- As **classes** definem como um conjunto de objetos se comportará e quais serão suas características. Logo, elas são compostas por **comportamentos** e **características** referentes aos seus objetos.
 - Exemplo: classe que define resumidamente seres humanos

Comportamentos:

- Andar
- Pensar
- Respirar

Características:

- Nome
- Cor dos olhos
- Cor da pele
- Dt. nascimento

Classes

- As classes são comumente representadas por um **retângulo** dividido em **3 compartimentos**: um para o **nome** da classe; outro para as **características**; e outro para os **comportamentos**.

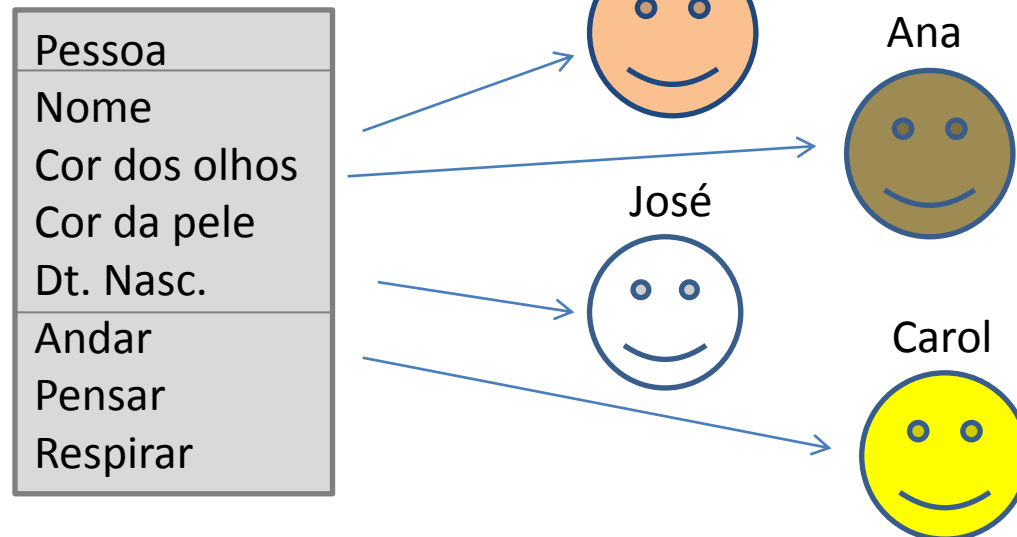
– Exemplo:

Pessoa
Nome Cor dos olhos Cor da pele Dt. Nasc.
Andar Pensar Respirar

Objetos

- Os objetos são **concretizações** de uma classe. Por meio dos objetos, são possíveis as execuções dos comportamentos e as configurações das características definidas nas respectivas classes.

– Exemplo 1:



Objetos

- Os objetos são **concretizações** de uma classe. Por meio dos objetos, são possíveis as execuções dos comportamentos e as configurações das características definidas nas respectivas classes.

– Exemplo 2:

Brasilia
Placa
Cor
Ano
Motor
Acelerar
Freiar
Buzinar



Objetos

- Os objetos são **concretizações** de uma classe. Por meio dos objetos, são possíveis as execuções dos comportamentos e as configurações das características definidas nas respectivas classes.
 - Exemplo 3:

Feio
Nome
Etnia
Nacionalidade
Endereço
Assustar
Incomodar
Se achar

Objetos

- Os objetos são **concretizações** de uma classe. Por meio dos objetos, são possíveis as execuções dos comportamentos e as configurações das características definidas nas respectivas classes.

– Exemplo 3:

Feio
Nome
Etnia
Nacionalidade
Endereço
Assustar
Incomodar
Se achar



Objetos

- Os objetos são **concretizações** de uma classe. Por meio dos objetos, são possíveis as execuções dos comportamentos e as configurações das características definidas nas respectivas classes.

– Exemplo 3:

Feio
Nome
Etnia
Nacionalidade
Endereço
Assustar
Incomodar
Se achar



As mina pira!

Classes e objetos em Java

- Em programação, os comportamentos da classe são comumente chamados de **métodos** e as características de **atributos**.
- Em Java, as classes são definidas com a seguinte sintaxe:

```
<modificador de acesso> class <nome da classe> {  
    <atributos>  
    <métodos>  
}
```

Classes e objetos em Java

- Em breve, veremos os modificadores de acesso, mas, por ora, utilizaremos apenas o modificador **public**. Portanto, usaremos a seguinte sintaxe:

```
public class <nome da classe> {  
    <atributos>  
    <métodos>  
}
```

Classes e objetos em Java

- Em breve, veremos os modificadores de acesso, mas, por ora, utilizaremos apenas o modificador **public**. Portanto, usaremos a seguinte sintaxe:
 - Exemplo: classe sem métodos nem atributos

```
public class Feio {}
```

Classes e objetos em Java

- Podemos inserir quantos métodos e atributos forem necessários dentro das classes.
- A sintaxe para definição de atributos (chamados de **atributos de instância**) é similar à sintaxe para declaração de variáveis, porém, pode ser antecedido por um modificador de acesso.

`<modificador de acesso> <tipo> <nome>;`

- Usaremos o modificador `public` temporariamente
`public <tipo> <nome>;`

Classes e objetos em Java

- Exemplo de classe com alguns atributos:

```
public class Pessoa{  
    public char[] nome;  
    public int corDosOlhos; //1- castanho, 2-verde, ...  
    public int corDaPele; //1- branca; 2- corada; 3- torrachim,...  
    public int dia, mes, ano;  
}
```

Classes e objetos em Java

- A sintaxe para definição de métodos *para os objetos* (conhecidos como **métodos de instância**) é similar à sintaxe para declaração de métodos que vimos anteriormente, porém, sem o termo *static*.

```
<modificador de acesso> <tipo de retorno> <nome>(<parâmetros>){  
    ...  
}
```

- Também usaremos o `public`:

```
public <tipo de retorno> <nome> (<parâmetros>){ ... }
```



Classes e objetos em Java

- Exemplo de classe com alguns atributos e métodos:

```
public class Pessoa{  
    public char[] nome;  
    public int corDosOlhos; //1- castanho, 2-verde, ...  
    public int corDaPele; //1- branca; 2- corada; 3- torrachim,...  
    public int dia, mes, ano;  
  
    public void respirar(){  
        System.out.println("Respirando...");  
    }  
}
```

Classes e objetos em Java

- Para criar objetos de uma classe, utiliza-se o comando **new**.
 - Sintaxe: `new <nome da classe>()`
 - Exemplo:

`new Pessoa()`  

** Essa sintaxe é aplicável apenas a alguns casos.*

Classes e objetos em Java

- Para criar objetos de uma classe, utiliza-se o comando **new**.
 - Sintaxe: `new <nome da classe>()`
 - Exemplo:

`new Feio()`

Classes e objetos em Java

- Para criar objetos de uma classe, utiliza-se o comando **new**.
 - Sintaxe: `new <nome da classe>()`
 - Exemplo:

`new Feio()`



Classes e objetos em Java

- Cada **objeto** criado **mantém** suas próprias **características** e tem condições de executar todos os **comportamentos** definidos na classe à qual pertence.
- Para manipularmos um determinado atributo ou executarmos algum método de um objeto, temos de acessá-lo e, para isso, podemos utilizar **variáveis de referência**, que simplesmente “apontam” para determinado objeto, para uma possível utilização futura.

Classes e objetos em Java

- As **variáveis de referência** são variáveis comuns, como as variáveis utilizadas anteriormente. No entanto, elas **armazenam uma referência** a um objeto, em vez de armazenar um valor, como fazem as variáveis de tipos primitivos.
- A sintaxe também é a mesma utilizada, mas, em vez de um tipo primitivo, utilizamos o nome da classe.

`<nome da classe> <nome da variável>;`

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();  
jose = ana;
```

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

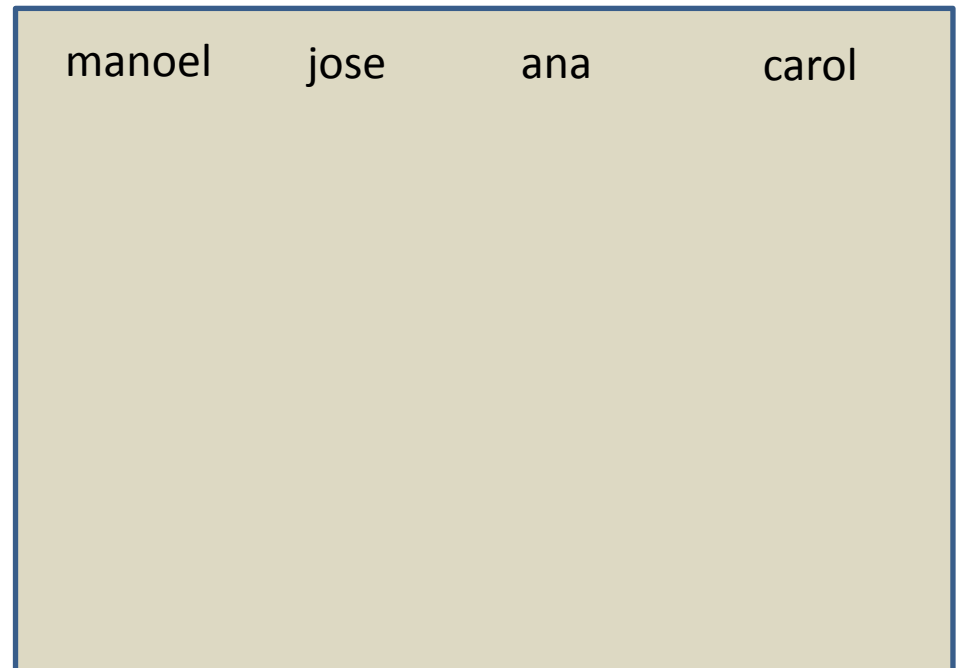
➔ `Pessoa manoel, jose;`
`Pessoa ana, carol;`
`manoel = new Pessoa();`
`ana = new Pessoa();`
`carol = new Pessoa();`
`new Pessoa();`
`jose = ana;`

manoel jose

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

➔ `Pessoa manoel, jose;
Pessoa ana, carol;
manoel = new Pessoa();
ana = new Pessoa();
carol = new Pessoa();
new Pessoa();
jose = ana;`



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

Pessoa manoel, jose;

Pessoa ana, carol;

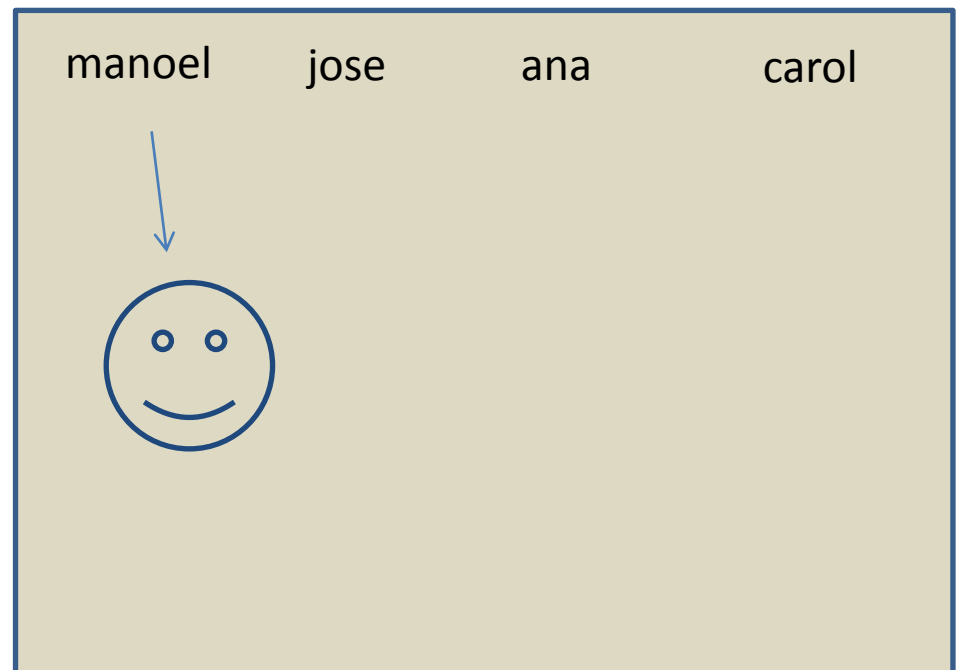
➔ `manoel = new Pessoa();`

`ana = new Pessoa();`

`carol = new Pessoa();`

`new Pessoa();`

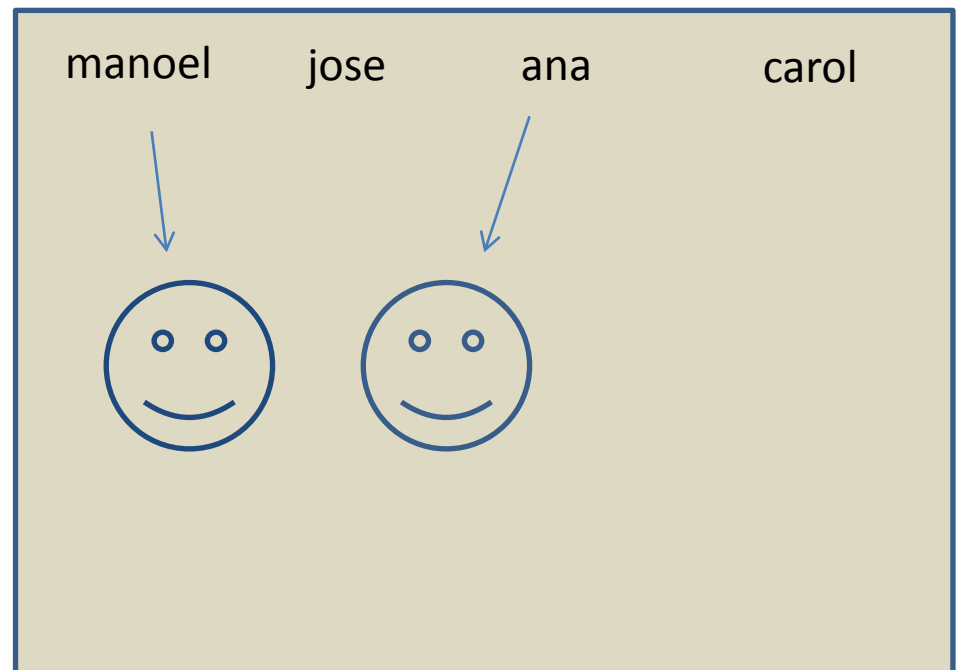
`jose = ana;`



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

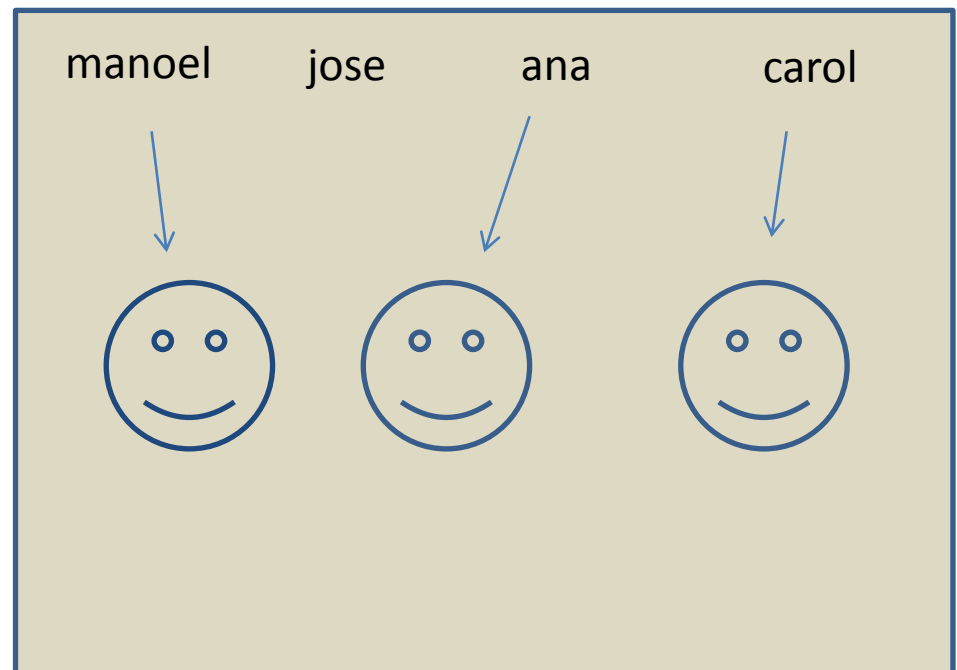
```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
→ ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();  
jose = ana;
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

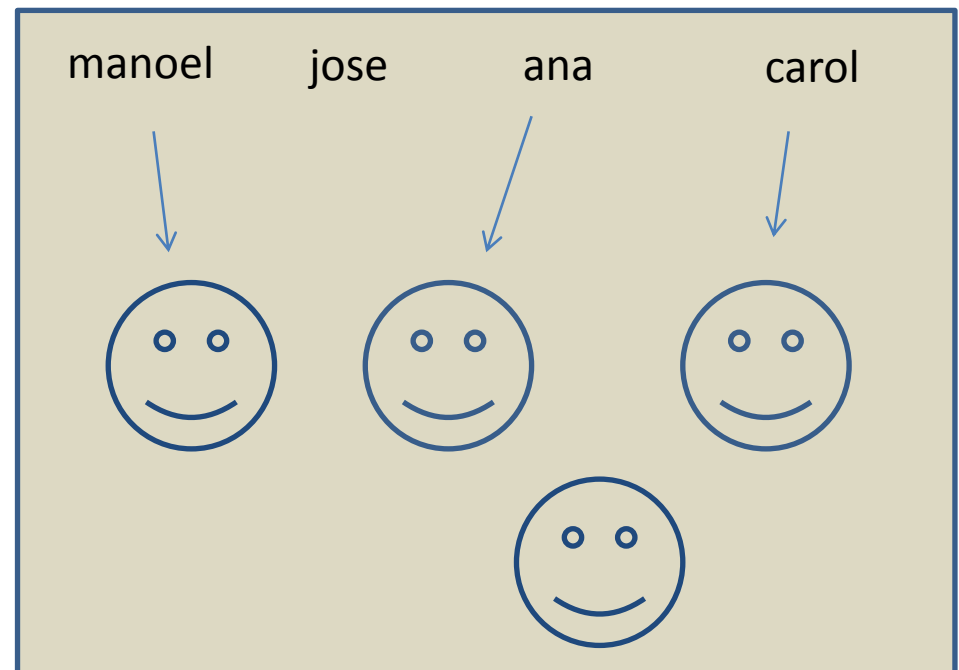
```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
➔ carol = new Pessoa();  
  new Pessoa();  
jose = ana;
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
Pessoa manoel, kleber;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
➔ new Pessoa();  
jose = ana;
```

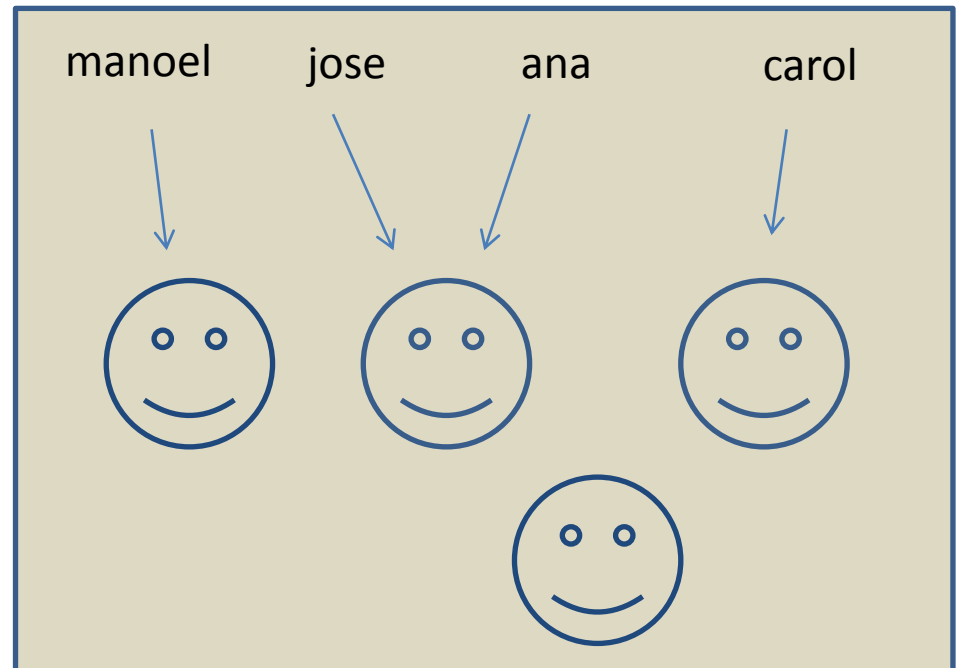


Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();
```

➡ `jose = ana;`



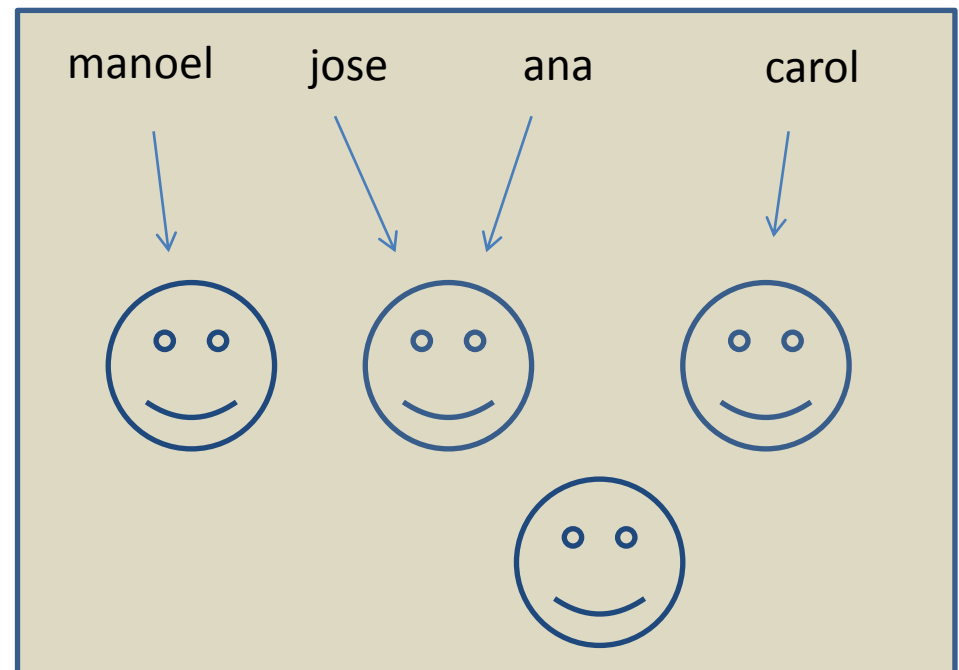
Classes e objetos em Java

- Para acessar um atributo ou um método (ou seja, um membro) de um objeto, usamos sua referência seguida de ponto final com o nome do respectivo membro.
 - Sintaxe: <referência>.<membro>
 - Exemplos:
`var1.atributo = 3;`
`var1.metodo();`

Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

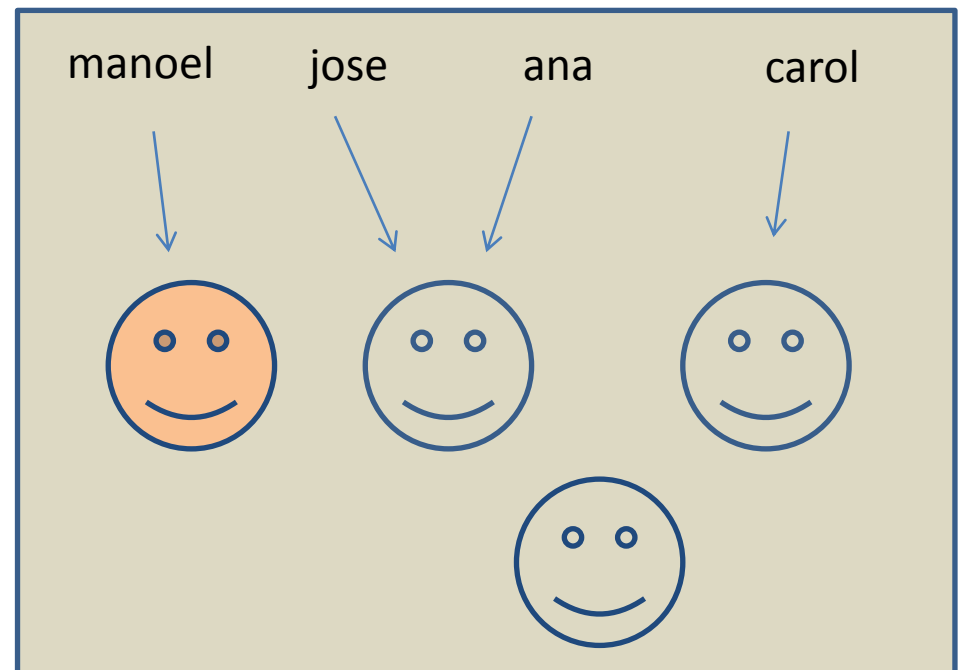
```
manoel.corDaPele = 3;  
carol.corDaPele = 1;  
jose.corDaPele = 4;  
ana.corDaPele = 2;  
jose.respirar();  
manoel.respirar();  
ana.respirar();
```



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

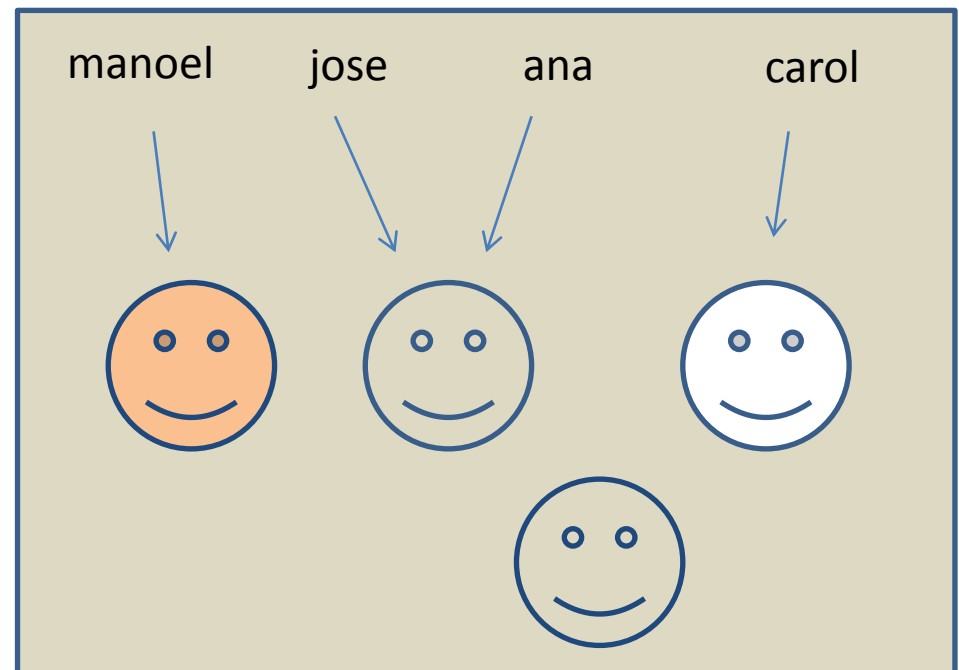
➔ `manoel.corDaPele = 3;`
`carol.corDaPele = 1;`
`jose.corDaPele = 4;`
`ana.corDaPele = 2;`
`jose.respirar();`
`manoel.respirar();`
`ana.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

➔
`manoel.corDaPele = 3;`
`carol.corDaPele = 1;`
`jose.corDaPele = 4;`
`ana.corDaPele = 2;`
`jose.respirar();`
`manoel.respirar();`
`ana.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

`manoel.corDaPele = 3;`

`carol.corDaPele = 1;`

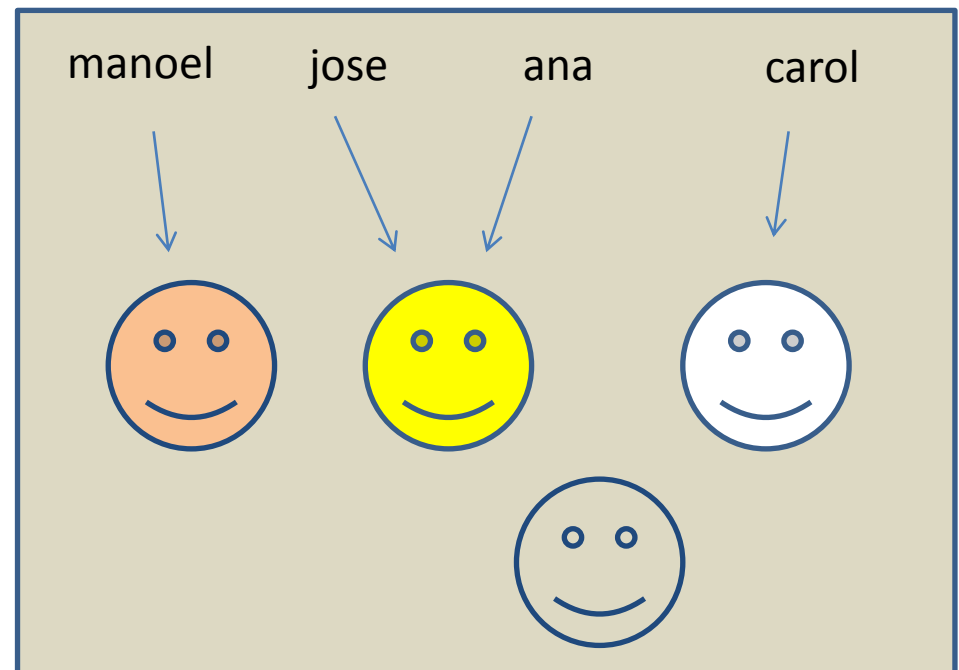
➔ `jose.corDaPele = 4;`

`ana.corDaPele = 2;`

`jose.respirar();`

`manoel.respirar();`

`ana.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

`manoel.corDaPele = 3;`

`carol.corDaPele = 1;`

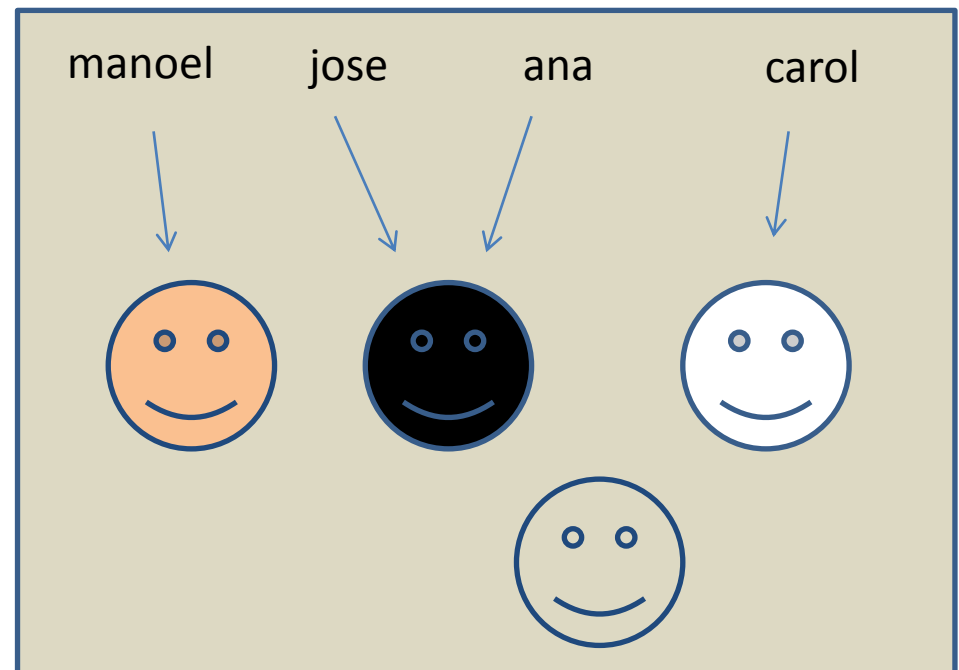
`jose.corDaPele = 4;`

➔ `ana.corDaPele = 2;`

`jose.respirar();`

`manoel.respirar();`

`ana.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

```
manoel.corDaPele = 3;
```

```
carol.corDaPele = 1;
```

```
jose.corDaPele = 4;
```

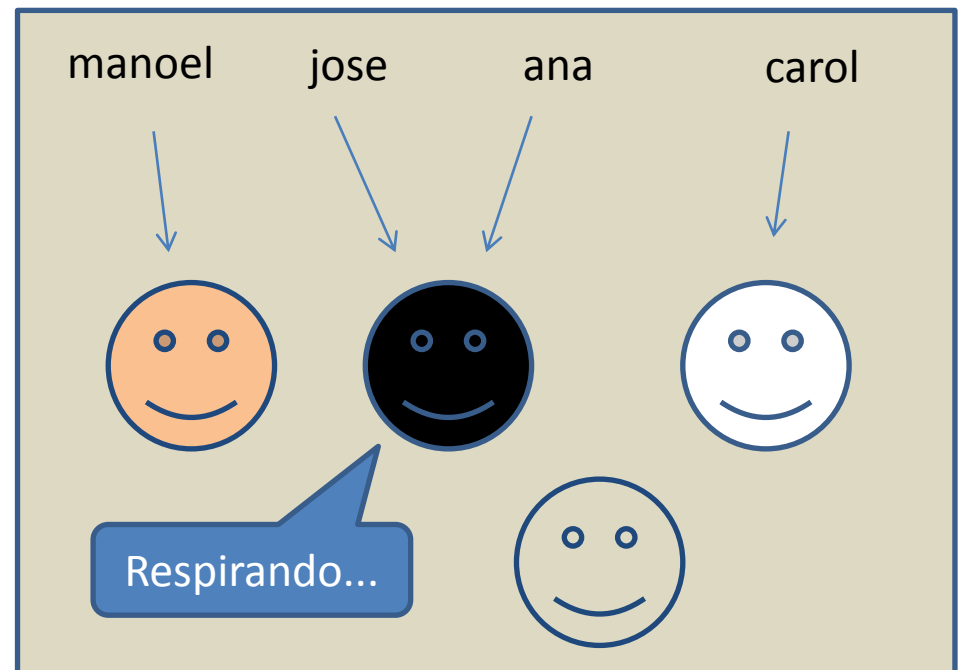
```
ana.corDaPele = 2;
```

➔

```
jose.respirar();
```

```
manoel.respirar();
```

```
ana.respirar();
```



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

```
manoel.corDaPele = 3;
```

```
carol.corDaPele = 1;
```

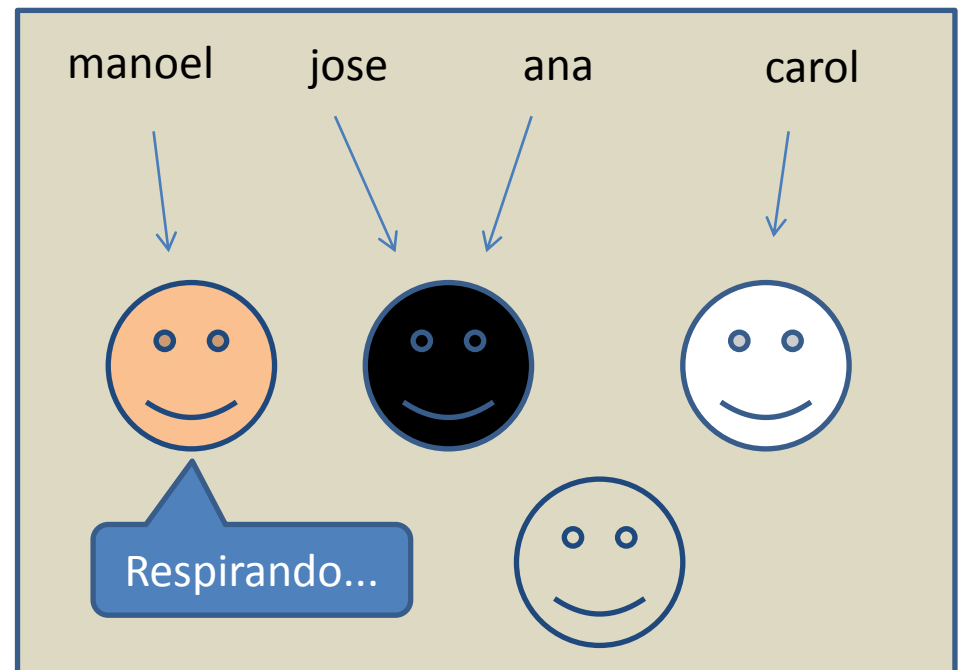
```
jose.corDaPele = 4;
```

```
ana.corDaPele = 2;
```

```
jose.respirar();
```

```
➔ manoel.respirar();
```

```
ana.respirar();
```



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

```
manoel.corDaPele = 3;
```

```
carol.corDaPele = 1;
```

```
jose.corDaPele = 4;
```

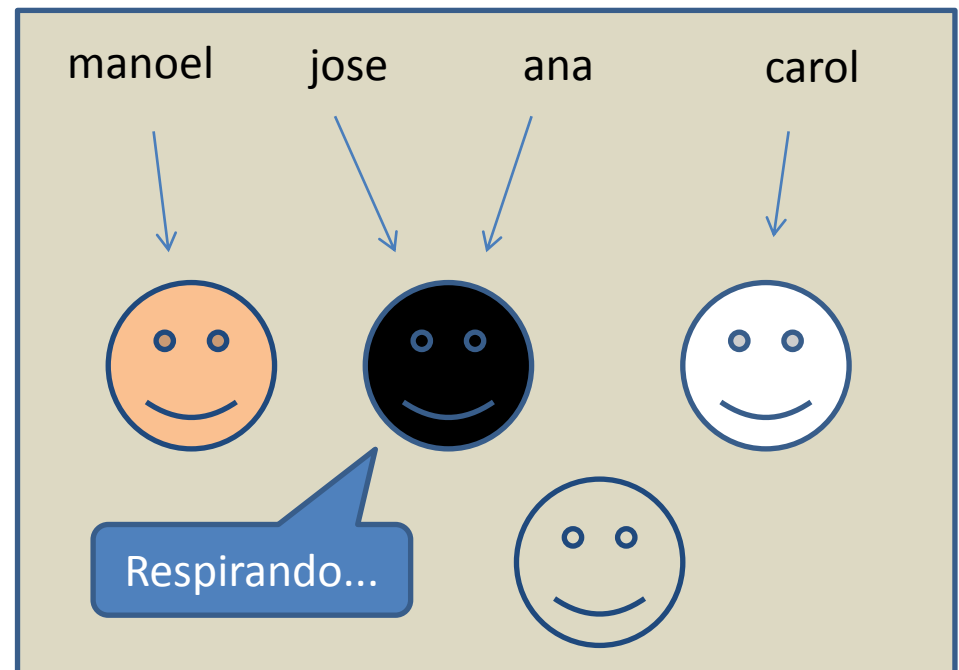
```
ana.corDaPele = 2;
```

```
jose.respirar();
```

```
manoel.respirar();
```

➡

```
ana.respirar();
```



Classes e objetos em Java

- Os métodos de instância podem utilizar as variáveis de instância como for preciso por meio dos nomes utilizados em suas declarações.
- A variável que será lida no momento da execução do método é a variável de instância do objeto que executou esse método.

Classes e objetos em Java

- Exemplo de utilização de atributo de instância dentro de método de instância:

```
public class Pessoa{  
    public char[] nome;    ...  
    public void respirar(){  
        for (char letra : nome){  
            System.out.print(letra);  
        }  
        System.out.println(" respirando...");  
    }  
}
```

Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados e utilização de seus métodos:

```
manoel.nome = new char[]{'M','a','n','o','e','l'};
```

```
ana.nome = new char[]{'A','n','a'};
```

```
carol.nome = new char[]{'C','a','r','o','l'};
```

```
ana.respirar();
```

```
carol.respirar();
```

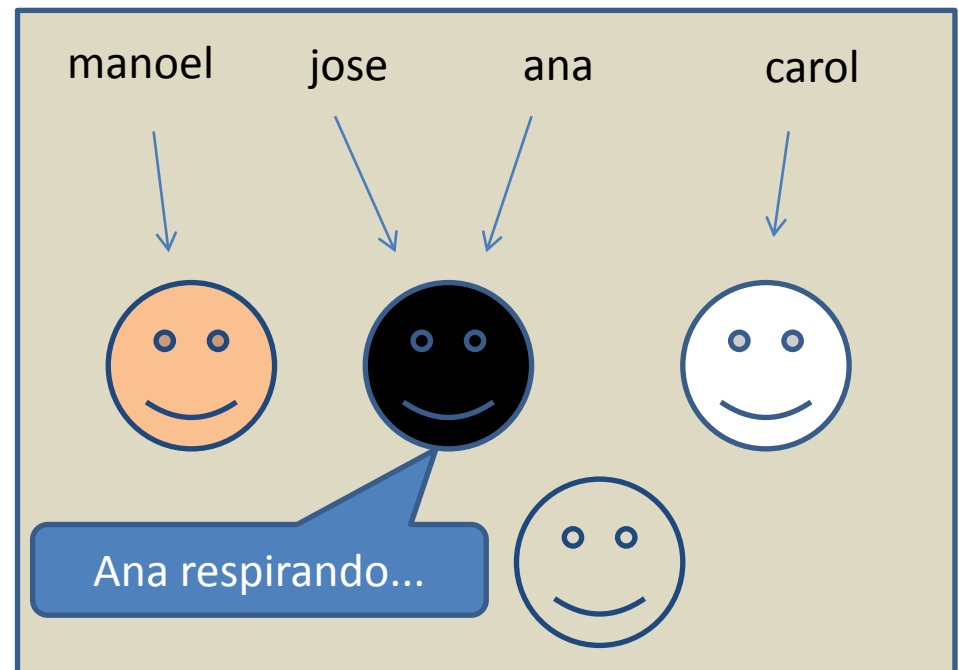
```
manoel.respirar();
```

```
jose.respirar();
```

Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

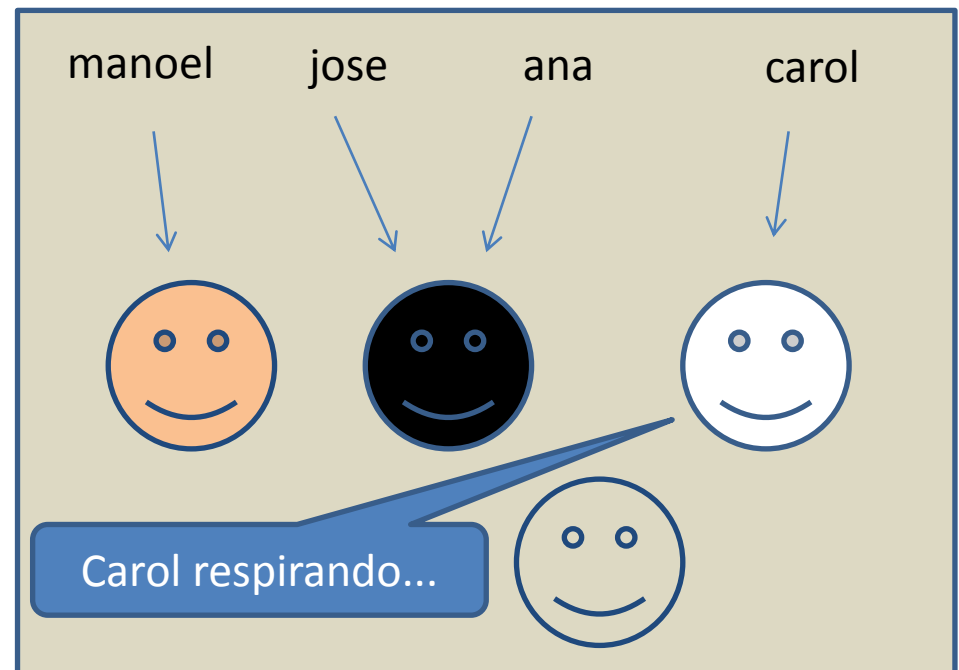
➔ `ana.respirar();`
`carol.respirar();`
`manoel.respirar();`
`jose.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

➔
`ana.respirar();`
`carol.respirar();`
`manoel.respirar();`
`jose.respirar();`



Classes e objetos em Java

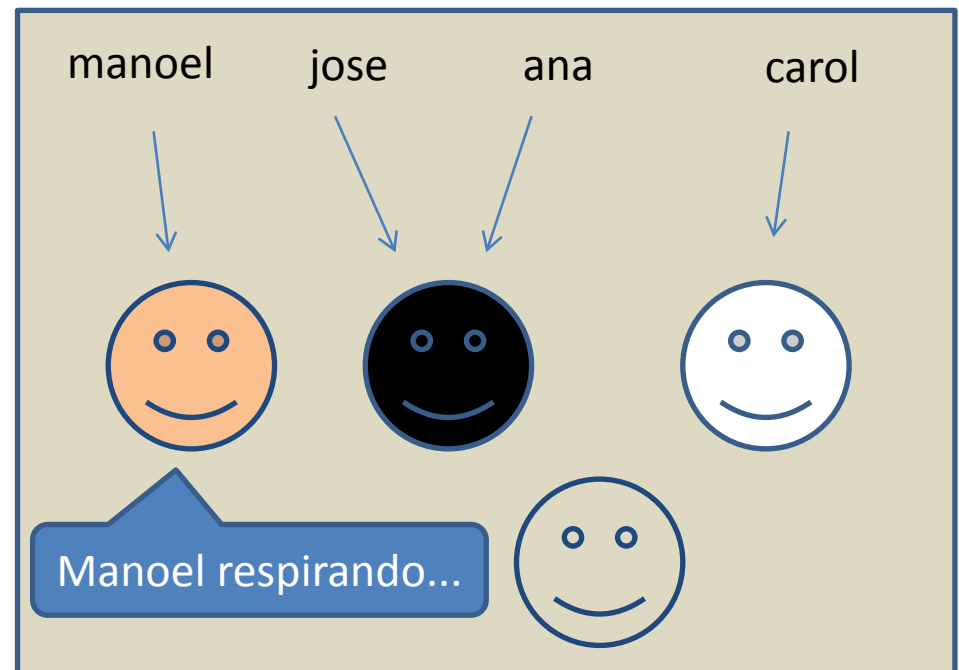
- Exemplos de alteração de atributos dos objetos criados:

`ana.respirar();`

`carol.respirar();`

➔ `manoel.respirar();`

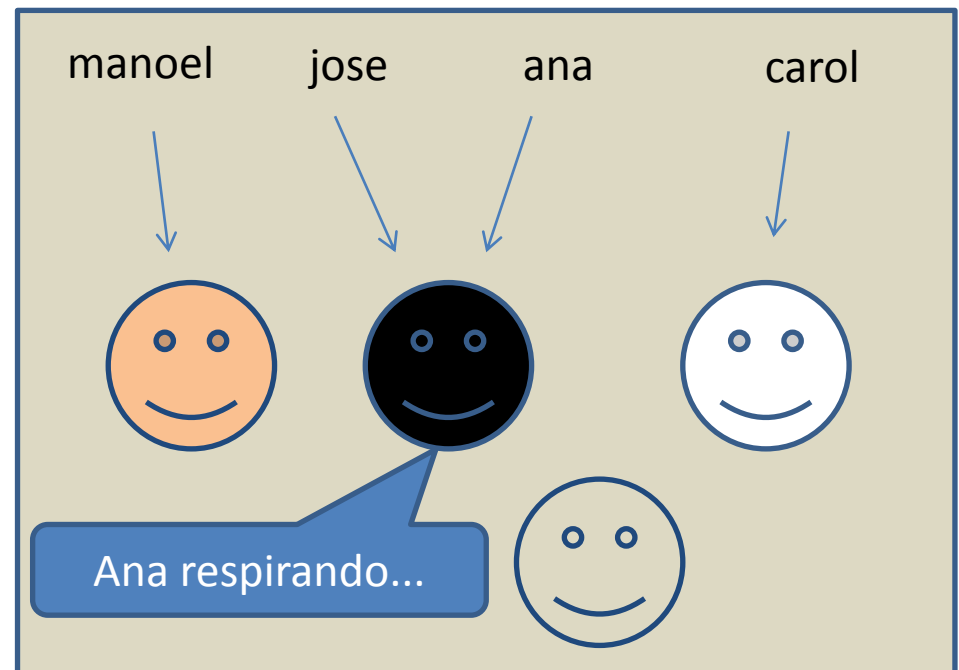
`jose.respirar();`



Classes e objetos em Java

- Exemplos de alteração de atributos dos objetos criados:

```
ana.respirar();  
carol.respirar();  
manoel.respirar();  
→ jose.respirar();
```



Classes e objetos em Java

- Durante a construções de objetos, pode ser necessária a execução de determinados procedimentos iniciais.
- Para isso, podemos utilizar os construtores, que são automaticamente executados ao construirmos novos objetos.
- Os construtores são similares aos métodos, porém, têm exatamente o mesmo nome da classe e não aceitam retornos.

Classes e objetos em Java

- Sintaxe para construtores:

<modificador de acesso> <nome da classe>(<parâmetros>){ <código> }

- Também utilizaremos o public por enquanto.
 - Exemplo de construtor:

```
public class Pessoa{  
    public Pessoa(){  
        System.out.println("Fui criado!");  
    }  
}
```

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();  
jose = ana;
```

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

➔ `Pessoa manoel, jose;`
`Pessoa ana, carol;`
`manoel = new Pessoa();`
`ana = new Pessoa();`
`carol = new Pessoa();`
`new Pessoa();`
`jose = ana;`

manoel jose

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

➡ Pessoa manoel, jose;
Pessoa ana, carol;
manoel = new Pessoa();
ana = new Pessoa();
carol = new Pessoa();
new Pessoa();
jose = ana;

manoel jose ana carol

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

Pessoa manoel, jose;

Pessoa ana, carol;

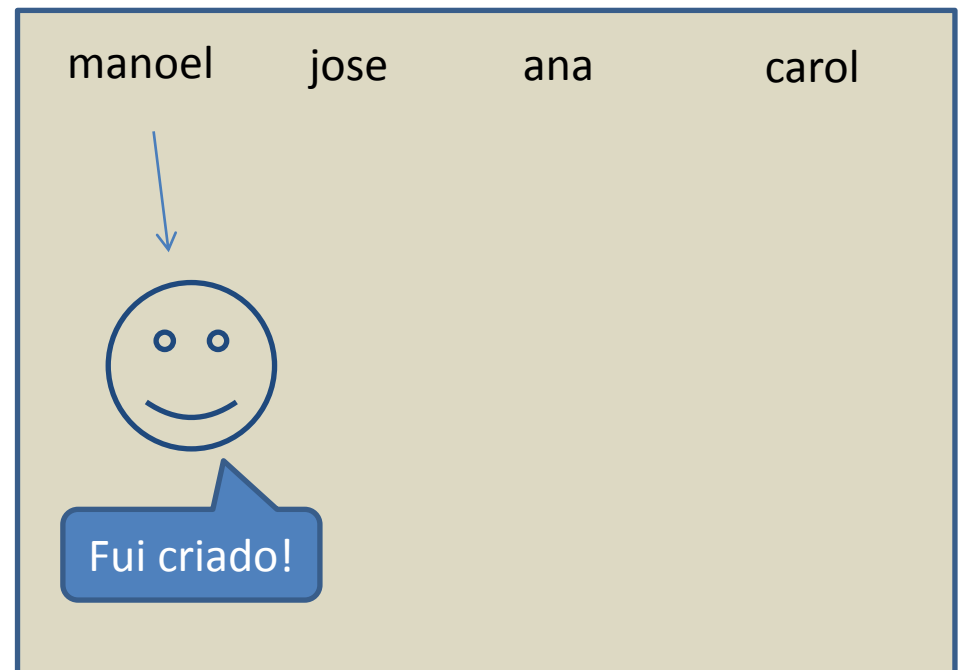
➔ `manoel = new Pessoa();`

`ana = new Pessoa();`

`carol = new Pessoa();`

`new Pessoa();`

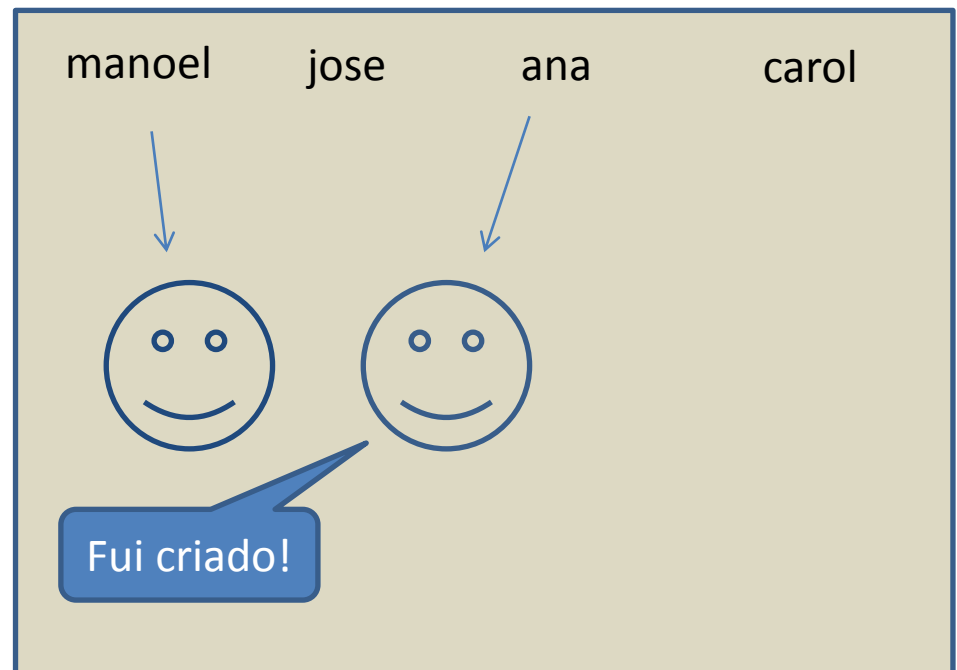
`jose = ana;`



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

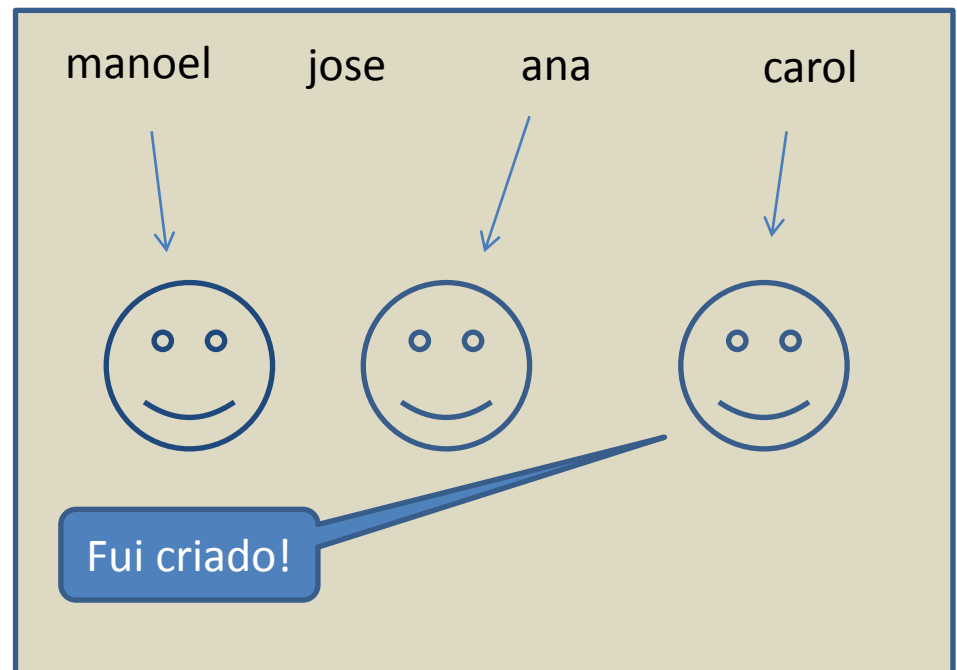
```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
→ ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();  
jose = ana;
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

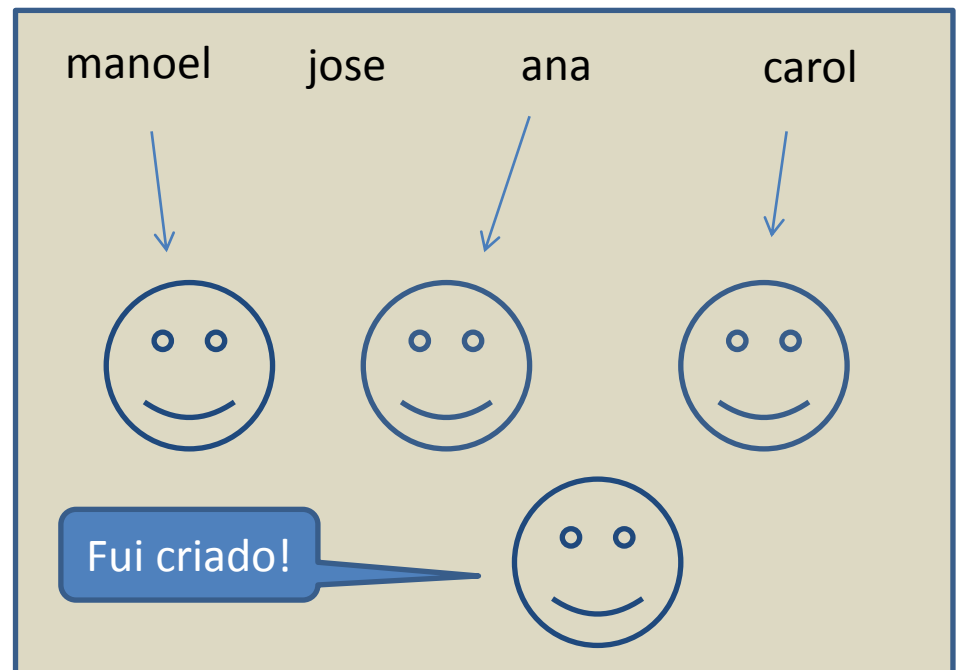
```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
➔ carol = new Pessoa();  
  new Pessoa();  
jose = ana;
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

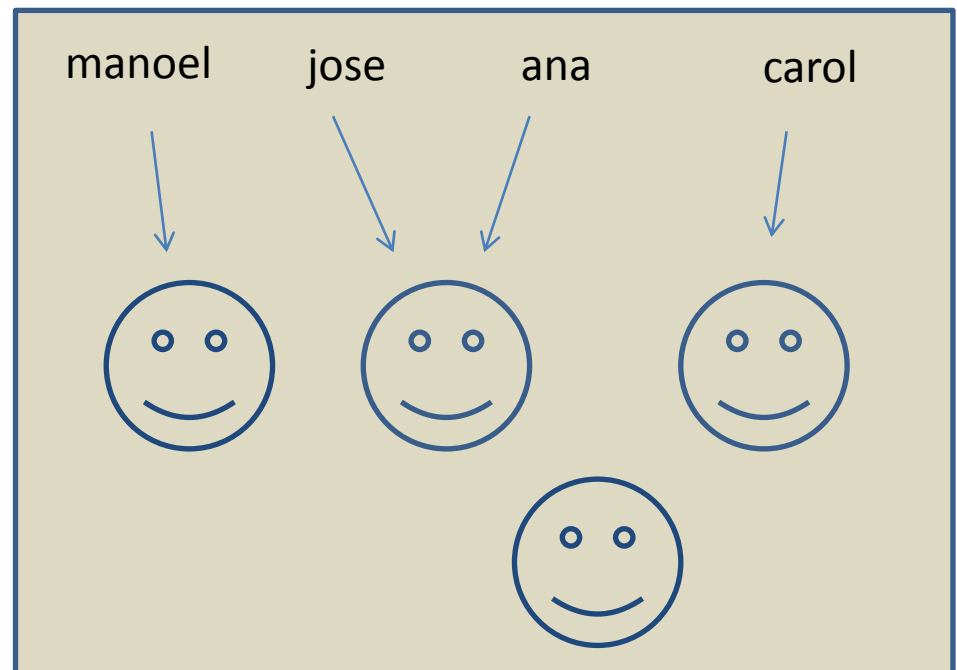
```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
➔ new Pessoa();  
jose = ana;
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
Pessoa manoel, jose;  
Pessoa ana, carol;  
manoel = new Pessoa();  
ana = new Pessoa();  
carol = new Pessoa();  
new Pessoa();  
➡ jose = ana;
```



Classes e objetos em Java

- Caso sua classe não tenha nenhum construtor definido, um construtor sem parâmetros é automaticamente inserido na classe em Java.
- Como visto na sintaxe, o construtor admite parâmetros. Esses parâmetros são variáveis quaisquer necessárias para a construção dos objetos da classe e são definidos do mesmo modo como são definidos os parâmetros dos métodos.

Classes e objetos em Java

- Exemplo de configuração do nome do indivíduo ao construir objetos de Pessoa

```
public class Pessoa{ ...  
    public Pessoa(char[] name){  
        for (char letra : name){  
            System.out.print(letra);  
        }  
        System.out.println(" sendo criado!");  
    } ...  
}
```

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

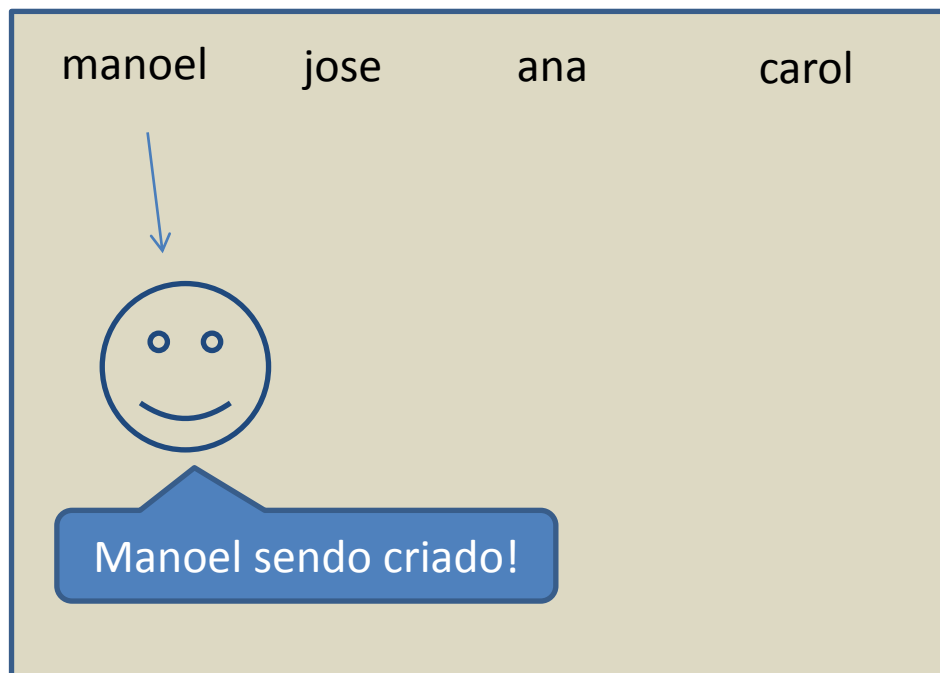
```
Pessoa manoel, jose, ana, carol;  
char[] nome1, nome2, nome3, nome4;  
nome1 = new char[]{'M','a','n','o','e','l'};  
nome2 = new char[]{'A','n','a'};  
nome3 = new char[]{'C','a','r','o','l'};  
manoel = new Pessoa(nome1);  
ana = new Pessoa(nome2);  
carol = new Pessoa(nome3);  
new Pessoa(nome3);
```

Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:



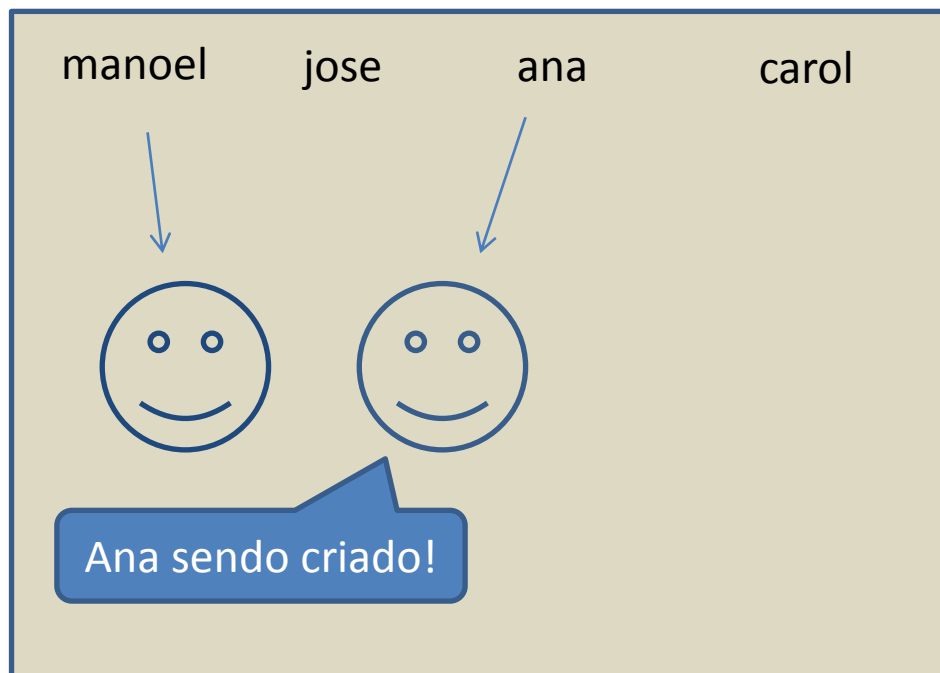
```
manoel =  
    new Pessoa(nome1);  
ana =  
    new Pessoa(nome2);  
carol =  
    new Pessoa(nome3);  
new Pessoa(nome3);
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

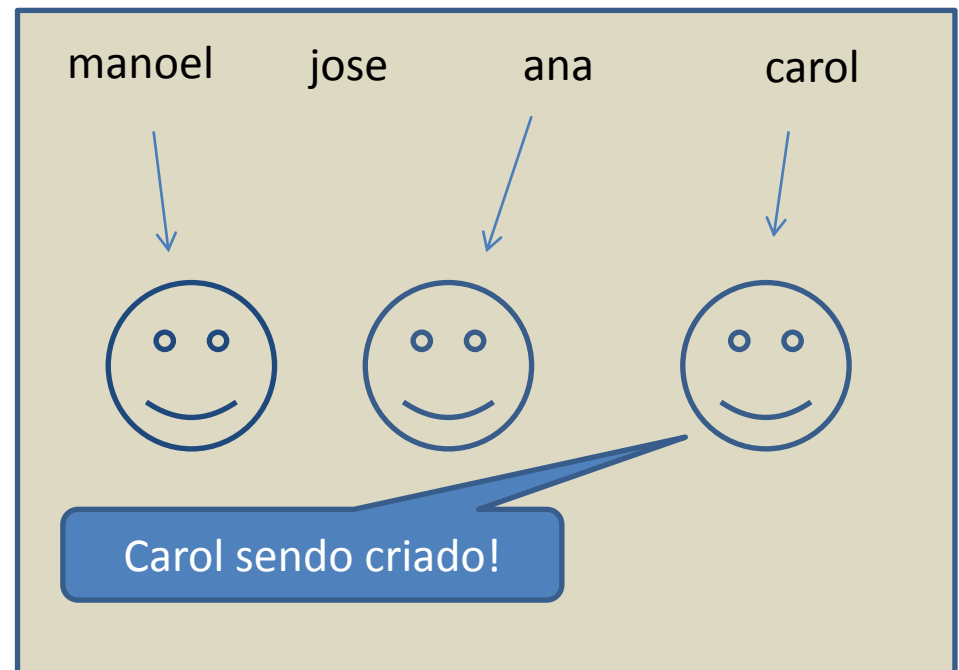
manoel =
 new Pessoa(nome1);
→ ana =
 new Pessoa(nome2);
carol =
 new Pessoa(nome3);
new Pessoa(nome3);



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

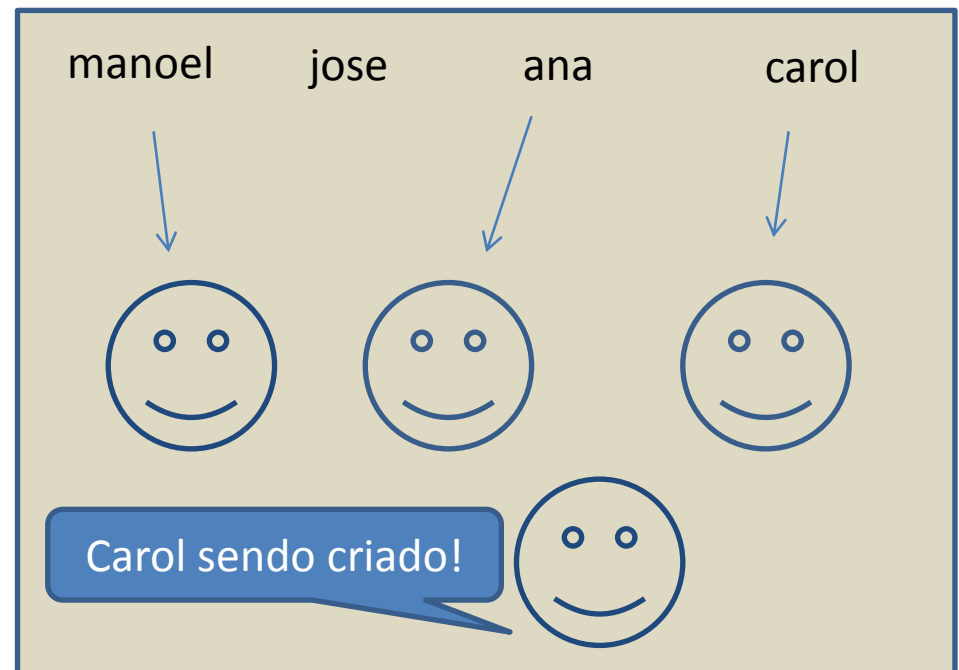
```
manoel =  
    new Pessoa(nome1);  
ana =  
    new Pessoa(nome2);  
→ carol =  
    new Pessoa(nome3);  
    new Pessoa(nome3);
```



Classes e objetos em Java

- Exemplos de declaração de 4 variáveis de referência que apontam para 3 objetos de Pessoa:

```
manoel =  
    new Pessoa(nome1);  
ana =  
    new Pessoa(nome2);  
carol =  
    new Pessoa(nome3);  
➔ new Pessoa(nome3);
```



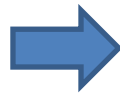
Classes e objetos em Java

- Após a construção de um objeto, os valores de todas as suas variáveis de instância são definidos com o valor padrão de acordo com seu tipo.
 - Primitivos:
 - Números: 0 ou 0.0
 - Character: character 0 da tabela UTF-16
 - Lógico: *false*
 - Referência: *null* (isso inclui variáveis compostas)

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:

Implementação de Pessoa



```
public class Pessoa{  
    public char[] nome; ...  
    public Pessoa(char[] name){  
        mostrarTexto(name);  
        System.out.println(" sendo criado!");  
    }  
    public void respirar(){  
        mostrarTexto(nome);  
        System.out.println(" respirando...");  
    }  
}
```

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:

```
char[] nome;  
nome = new char[]{'a','n','a'};  
Pessoa ana = new Pessoa(nome);  
ana.respirar();
```

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:

```
char[] nome;  
nome = new char[]{'a','n','a'};  
Pessoa ana = new Pessoa(nome);  
ana.respirar();
```



Cria o objeto e exibe o texto:
Ana sendo criado!

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:

```
char[] nome;  
nome = new char[]{'a','n','a'};  
Pessoa ana = new Pessoa(nome);  
ana.respirar();
```



Ocasiona um erro durante a execução:
O vetor deve ser percorrido (de 0 a *length-1*). No entanto, não se pode obter *length* de nulo (*null*).

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:
 - Para resolver, “roubamos” o vetor enviado para o objeto criado.

```
public class Pessoa{  
    ...  
    public Pessoa(char[] name){  
        mostrarTexto(name);  
        System.out.println(" sendo criado!");  
        → nome = name;  
    }  
    ....  
}
```

Classes e objetos em Java

- Exemplo de visualização de valores de variáveis de instância após criação:
 - Para resolver, “roubamos” o vetor enviado para o objeto criado.

Cuidado ao passar
seu vetor a um
método! Ele pode
ser modificado!

```
public class Pessoa{  
    ...  
    public Pessoa(char[] name){  
        mostrarTexto(name);  
        System.out.println(" sendo criado!");  
        nome = name;  
    }  
    ....  
}
```


Classes e objetos em Java

- Nota-se que, até o momento, a utilização de cadeias de caracteres tem sido trabalhosa. Para evitar esse e alguns outros inconvenientes, a linguagem Java possui diversas classes de suporte.
- Dentre elas, temos a classe String, que manipula conjuntos de caracteres (eliminando a necessidade de usarmos vetores de caracteres).

Classes e objetos em Java

- Uma das sintaxes para uso de String:

String <nome da variável>;

<nome da variável> = new String(<vetor de caracteres>);

– Exemplo:

```
String nome;
```

```
char[] vetor;
```

```
vetor = new char[]{'a','n','a'};
```

```
nome = new String(vetor);
```

Classes e objetos em Java

- A classe String (como esperado) possui diversos métodos, dentre eles:
 - `char charAt(int i)`: retorna o caracter da i-ésima posição da String
 - `boolean equals(String outra)`: verifica se a String é igual à outra considerando maiúsculas e minúsculas.
 - `boolean equalsIgnoreCase(String outra)`: verifica se a String é igual à outra desconsiderando maiúsculas e minúsculas.

Classes e objetos em Java

- Por especificação da linguagem, todo literal apresentado entre aspas duplas é transformado em uma String.
- O operador de adição (+) aplicado a String atua como concatenador de Strings.

Classes e objetos em Java

- Exemplos de utilização de String

```
String texto1, texto2;
```

```
char[] vetor;
```

```
vetor = new char[]{'a','n','a'}; } Atribuições equivalentes  
texto1 = new String(vetor);
```

```
texto2 = "ana";
```

```
texto3 = "Ana";
```

```
if (texto1 == texto2){  
    System.out.println("Teste 1");  
}
```


Falso!

Comparação de referências só será verdadeira se as duas variáveis se referirem ao mesmo objeto.

Classes e objetos em Java

- Exemplos de utilização de String

```
String texto1, texto2;  
char[] vetor;  
vetor = new char[]{'a','n','a'};  
texto1 = new String(vetor);  
texto2 = "ana";  
texto3 = "Ana";  
if (texto1.equals(texto2)) {  
    System.out.println("Teste 2");  
}
```

 Verdadeiro!

Classes e objetos em Java

- Exemplos de utilização de String

```
String texto1, texto2;  
char[] vetor;  
vetor = new char[]{'a','n','a'};  
texto1 = new String(vetor);  
texto2 = "ana";  
texto3 = "Ana";  
if (texto2.equals(texto3)){  
    System.out.println("Teste 3");  
}
```

Falso!

Seria verdadeiro se o método `equalsIgnoreCase` fosse utilizado no lugar de `equals`.

Classes e objetos em Java

- As variáveis de instância e de métodos podem ser de tipo primitivo, mas também pode ser de referência.
- As variáveis de instância permanecem “vivas” enquanto o objeto estiver “vivo”. Já as variáveis de método, perduram apenas durante sua execução.
- As variáveis de método, diferentemente das variáveis de instância, não possuem valor inicial e, se houver leitura antes de escrita, ocorre erro de compilação.

Classes e objetos em Java

- Exemplo:

Temos de armazenar a referência do objeto de String, pois, após a execução do construtor, a variável de referência name não existirá mais.

Se refere à variável de instância, configurada no construtor.

```
public class Pessoa{  
    public String nome;  
    public int corDaPele;  
    public int corDoOlho;  
    public Pessoa(String name){  
        System.out.println(name + " sendo criado!");  
        nome = name;  
    }  
    public void respirar(){  
        System.out.println(nome + " respirando...");  
    }  
}
```

Classes e objetos em Java

- Note que, no exemplo anterior, o nome da variável de instância e o nome do parâmetro são propositalmente diferentes. Porém, os nomes poderiam ser iguais.
- Nesse caso, para atribuir um valor à variável de instância, teríamos de usar a palavra reservada **this**, que indica o objeto atual que invocou o método ou que está sendo construído.

Classes e objetos em Java

- Exemplo:

“this.nome” indica a variável de instância e “nome” indica o parâmetro.

```
public class Pessoa{  
    public String nome;  
    public int corDaPele;  
    public int corDoOlho;  
    public Pessoa(String nome){  
        System.out.println(name + “ sendo criado!”);  
        this.nome = nome;  
    }  
    public void respirar(){  
        System.out.println(nome + “ respirando...”);  
    }  
}
```

Laboratório

1. Crie uma classe chamada SerHumano que contenha três atributos (nome, pai e mae) e seis métodos (cujos comportamentos são descritos a seguir) de modo que o método main (apresentado no próximo slide) apresente as seguintes saídas:

```
Desconhecido  
Desconhecido  
Marilda  
Joao  
Maria  
Jose  
Ana  
Basilio
```

Laboratório

- Método main:
 - Não altere nada! Só copie!

```
public static void main(String[] args){
    SerHumano pedro, joao, marilda, maria;
    SerHumano jose, basilio, ana, sebastiana;

    sebastiana = new SerHumano("Sebastiana");
    basilio = new SerHumano("Basilio");
    ana = new SerHumano("Ana");
    jose = new SerHumano("Jose");
    joao = new SerHumano("Joao");
    maria = new SerHumano("Maria");
    marilda = new SerHumano("Marilda");
    pedro = new SerHumano("Pedro");

    maria.mae = sebastiana;
    marilda.pai = jose;
    marilda.mae = maria;
    joao.pai = basilio;
    joao.mae = ana;
    pedro.pai = joao;
    pedro.mae = marilda;

    System.out.println(sebastiana.nomeDaAvoMaterna());
    System.out.println(sebastiana.nomeDaMae());
    System.out.println(pedro.nomeDaMae());
    System.out.println(pedro.nomeDoPai());
    System.out.println(pedro.nomeDaAvoMaterna());
    System.out.println(pedro.nomeDoAvoMaterno());
    System.out.println(pedro.nomeDaAvoPaterna());
    System.out.println(pedro.nomeDoAvoPaterno());
}
```

Laboratório

- Métodos da classe SerHumano:
 - nomeDoPai: retorna o nome do pai do objeto (ou “Desconhecido” caso não exista)
 - nomeDaMae: retorna o nome da mãe do objeto (ou “Desconhecido” caso não exista)
 - nomeDoAvoPaterno: retorna o nome do avô paterno do objeto (ou “Desconhecido” caso não exista)

Laboratório

- Métodos da classe SerHumano:
 - nomeDaAvoPaterna: retorna o nome da avó paterna do objeto (ou “Desconhecido” caso não exista)
 - nomeDoAvoMaterno: retorna o nome do avô materno do objeto (ou “Desconhecido” caso não exista)
 - nomeDaAvoMaterna: retorna o nome da avó materna do objeto (ou “Desconhecido” caso não exista)

Laboratório

2. Crie uma classe chamada ArmaDeFogo com, no mínimo, os seguintes atributos e métodos:
 - Atributos:
 - *número de série*: número identificador da arma;
 - *capacidade*: indica o número máximo de projéteis que a arma suporta; e
 - *portador*: o responsável pela arma.

Laboratório

2. Crie uma classe chamada *ArmaDeFogo* com, no mínimo, os seguintes atributos e métodos:

– Métodos:

- *carregar*: que recebe o número de projéteis que serão inseridos na arma;
- *disparar*: que informa que UM projétil foi disparado se houver munição, porém, trava a arma para sempre se for chamado sem munição; e
- *contar*: que conta o número de projéteis disponíveis na arma.

Laboratório

Em seguida, complemente a classe Principal, seguindo as instruções do comentário:

```
public class Principal{  
    public static void main(String args[]){  
        ArmaDeFogo a1, a2, a3;  
        //criando armas  
        a1 = new ArmaDeFogo();  
        a2 = new ArmaDeFogo();  
        a3 = new ArmaDeFogo();  
        //carregando armas  
        a1.carregar(in.nextInt());  
        a2.carregar(in.nextInt());  
        a3.carregar(in.nextInt());  
        //dispare continuamente todas as armas até o fim dos projéteis,  
        //chamando somente o método disparar, sem travar nenhuma arma  
        //(não altere o código acima)  
    }  
}
```

Pacotes em Java

- Para **organização** das classes e prevenção de criação de classes de **mesmo nome**, é possível a utilização de pacotes e subpacotes em Java, que se incorporam automaticamente ao nome da classe.
- Para acrescentar uma classe a um (sub)pacote, deve ser utilizada a palavra reservada **package** seguida do nome do (sub)pacote e ponto-e-vírgula na **primeira linha** do código-fonte da classe.

Pacotes em Java

- Após a inserção da classe no respectivo pacote, seu nome torna-se a concatenação entre o nome do pacote e o nome da classe.

Pacotes em Java

- Exemplo 1: adicionando classe ClasseA no pacote pacoteA.

```
package pacoteA;
```

```
public class ClasseA{  
}
```

- Exemplo 2: adicionando classe ClasseB ao subpacote subpacote1 do pacote A

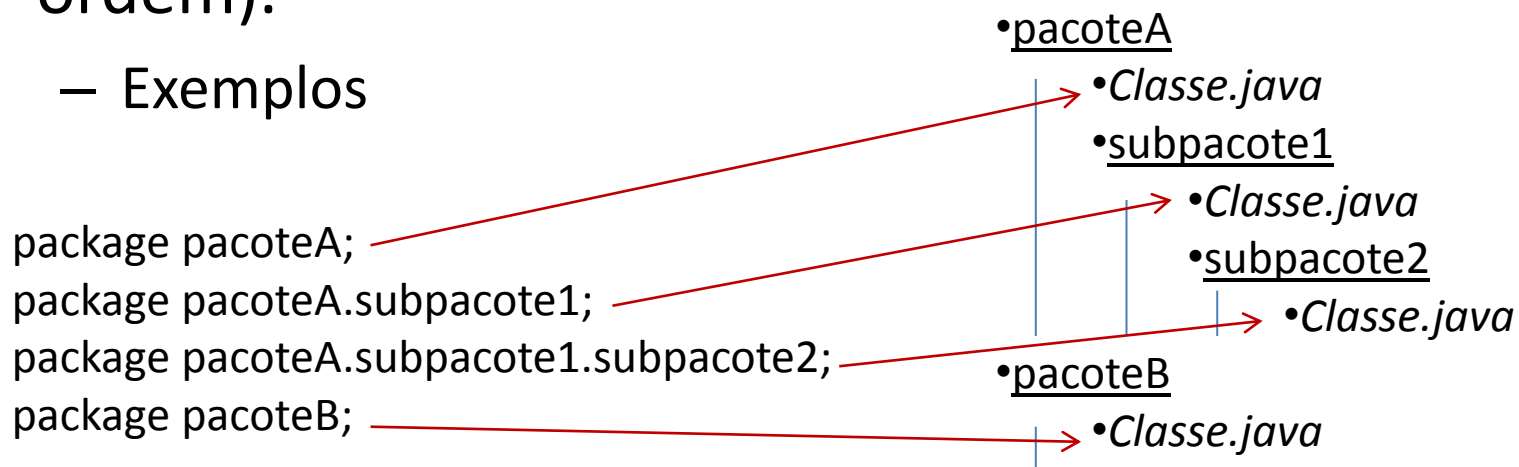
```
package pacoteA.subpacote1;
```

```
public class ClasseB{  
}
```

Pacotes em Java

- Além disso, o arquivo onde foi escrito o código-fonte deve ser salvo em **pastas** cujos nomes correspondam aos nomes dos pacotes e subpacotes (obedecendo a ordem).

– Exemplos



```

package pacoteA;
package pacoteA.subpacote1;
package pacoteA.subpacote1.subpacote2;
package pacoteB;
  
```

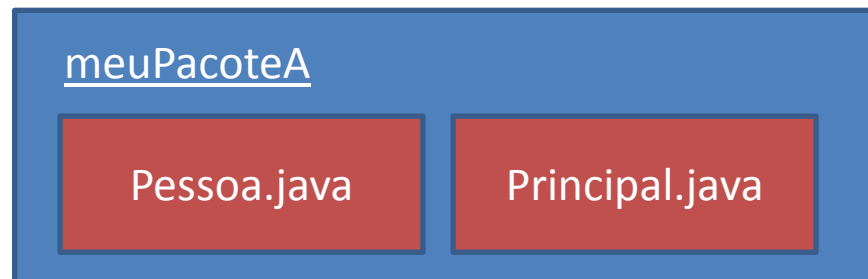
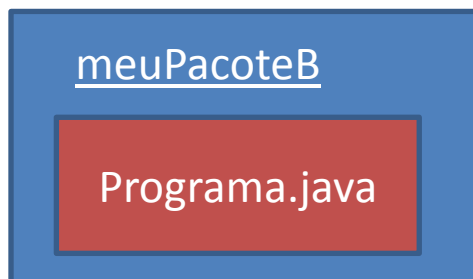
•pacoteA
 •Classe.java
 •subpacote1
 •Classe.java
 •subpacote2
 •Classe.java
 •pacoteB
 •Classe.java

Pacotes em Java

- Quando lidamos com mais de um código-fonte, pode ser necessário o **compartilhamento** de informações entre as classes codificadas em tais arquivos.
- Se as classes envolvidas estiverem no **mesmo pacote**, a utilização ocorre normalmente, como se utiliza a classe String, por exemplo.
- Caso contrário, a classe dependente deve **importar** a outra classe utilizando a palavra reservada import.

Pacotes em Java

- Exemplo de utilização de uma classe codificada externamente:
 - Suponha as seguintes classes contidas no seu respectivo pacote:



Pacotes em Java

- Exemplo de utilização de uma classe codificada externamente à classe Pessoa:

Principal.java

```
package meuPacoteA;

public class Principal{
    public static void main(String[] args){
        Pessoa p;
        p = new Pessoa();
    }
}
```

Pacotes em Java

- Exemplo de utilização de uma classe codificada externamente à classe Pessoa:

Programa.java

```
package meuPacoteB;  
  
public class Programa{  
    public static void main(String[] args){  
        meuPacoteA.Pessoa p;  
        p = new meuPacoteA.Pessoa();  
    }  
}
```

Pacotes em Java

- Exemplo de utilização de uma classe codificada externamente à classe Pessoa:

Programa.java

OU...

```
package meuPacoteB;  
import meuPacoteA.Pessoa;
```

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p;  
        p = new Pessoa();  
    }  
}
```

Laboratório

Dada a tabela de conversões de três unidades de medida de tempo (segundos, minutos e segundos) abaixo – em que T corresponde ao tempo que se deseja converter de uma unidade para a outra –, crie três novas classes de modo que, supondo que o diretório atual seja D:\, a classe prova.Conversor (dada a seguir) compile e execute corretamente ao utilizar (isoladamente) qualquer uma das três importações seguintes:

1. `import segundos.Unidade;`
2. `import tempo.minutos.Unidade;`
3. `import hours.Unidade;`

Laboratório

Para resolução deste exercício, devem ser criados
EXATAMENTE quatro arquivos, cujos nomes e pastas
DEVEM ser os seguintes:

D:\prova\Conversor.java

D:\segundos\segundos\Unidade.java

D:\auxiliar\tempo\minutos\Unidade.java

D:\auxiliar\utils\hours\Unidade.java

Laboratório

Tabela de conversões:

Conversões		Para		
		Segundo	Minuto	Hora
De	Segundo	T	$T/60$	$T/3600$
	Minuto	$T*60$	T	$T/60$
	Hora	$T*3600$	$T*60$	T

Laboratório

Altere APENAS a linha 3 deste código:

```
1 package prova;
2
3 //insira aqui uma das importações dadas
4
5 public class Conversor{
6     public static void main(String args[]){
7         double t = 60;
8         double s, m, h;
9         Unidade unidade = new Unidade();
10        s = unidade.converterParaSegundos(t);
11        m = unidade.converterParaMinutos(t);
12        h = unidade.converterParaHoras(t);
13        System.out.println("Segundo(s): " + s);
14        System.out.println("Minuto(s): " + m);
15        System.out.println("Hora(s): " + h);
16    }
17 }
```

Membros estáticos

- Note que, depois que iniciamos a POO, abandonamos a palavra reservada **static**.
- O static define que um membro da classe é **estático**.
- Um membro estático, em vez de pertencer a um objeto, **pertence à classe** e pode ser compartilhado entre os objetos (análogo a uma **variável global**).
- Em Java, um membro estático é acessado pelo **nome da classe**, mas também pode ser acessado por um objeto.

Membros estáticos

- Portanto, o acesso a membros estáticos não dependem da existência de um objeto, como ocorre com os membros de instância.
- Um método de instância (ou um construtor) pode acessar uma variável estática. No entanto, métodos **estáticos não podem acessar variáveis de instância.**
- As variáveis estáticas recebem o valor padrão de seu tipo quando a classe é carregada pela JVM.

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

Classe base:

```
public class Pessoa{  
    public int id;  
    public static int contador;  
    public Pessoa(){  
        id = ++contador;  
    }  
}
```

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(Pessoa.contador);  
    }  
}
```

3 ←

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(p1.contador);  
    }  
}
```

3 ←

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(p2.contador);  
    }  
}
```

3 ←

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(p1.id);  
    }  
}
```

1 ←

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(p2.id);  
    }  
}
```

2 ←

Membros estáticos

- Exemplo de utilização de variável estática para contar a quantidade de objetos criados de uma classe:

```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p1, p2;  
        p1 = new Pessoa();  
        p2 = new Pessoa();  
        new Pessoa();  
        System.out.println(Pessoa.id);  
    }  
}
```

**Erro de
compilação!**



Membros estáticos

- Exemplo de acesso inválido a um membro de instância por um membro estático:

**Erro de
compilação!**

```
public class Pessoa{  
    public int id;  
    public static int contador;  
    public Pessoa(){  
        id = ++contador;  
    }  
    public static void main(String[] args){  
        id++;  
    }  
}
```

Membros estáticos

- Embora as variáveis estáticas recebam o valor padrão do tipo da variável, podemos iniciar as variáveis com outros valores dentro do **bloco de iniciação/inicialização estático**.
- Para criar um bloco de iniciação estático, escrevemos `static` seguido de abertura e fechamento de chaves.
 - Exemplo:

```
static{  
    ...  
}
```

Membros estáticos

- Os blocos de iniciação estáticos são executados na ordem em que aparecem no código quando a classe é carregada pela JVM. Portanto, é um bom local para iniciarmos os valores de nossos atributos estáticos.

Membros estáticos

- Exemplo de uso de bloco de iniciação estático:

```
public class Circunferencia{
    public double raio;
    public static double pi;
    static{
        pi = 3.14;
    }
    public double obterArea(){
        return pi * raio * raio;
    }
}
```

```
public class Programa{
    public static void main(String[] args){
        Circunferencia c1;
        c1 = new Circunferencia();
        c1.raio = 2.0;
        System.out.println(c1.obterArea());
    }
}
```

Laboratório

1. Altere a classe ArmaDeFogo (feita em laboratório anterior) de modo que o número de série da arma criada seja determinado automaticamente de forma sequencial (a primeira arma criada terá código 1, a segunda 2 e assim sucessivamente).

Laboratório

2. Dada a tabela de conversões de três unidades monetárias (bufunfa, cascai e dindim) abaixo – em que V corresponde ao valor empo que se deseja converter de uma unidade para a outra –, crie três novas classes de modo que, supondo que o diretório atual seja D:\, a classe prova.Conversor (dada a seguir) compile e execute corretamente ao utilizar (isoladamente) qualquer uma das três importações seguintes:

1. `import converter.dindim.Unidade;`
2. `import prova.bufunfa.Unidade;`
3. `import prova.conversor.cascai.Unidade;`

Laboratório

Para resolução deste exercício, devem ser criados
EXATAMENTE quatro arquivos, cujos nomes e pastas
DEVEM ser os seguintes:

D:\prova\Conversor.java

D:\prova\converter\dindim\Unidade.java

D:\auxiliar\prova\bufunfa\Unidade.java

D:\prova\conversor\cascai\Unidade.java

Laboratório

Tabela de conversões:

Conversões		Destino		
		Dindim	Bufunfa	Cascai
Origem	Dindim	V	$1,25 * V$	$0,4 * V$
	Bufunfa	$0,8 * V$	V	$0,32 * V$
	Cascai	$2,5 * V$	$3,125 * V$	V

Laboratório

Altere APENAS a linha 3 deste código:

```
package prova;

//insira aqui uma das importações dadas

public class Conversor{
    public static void main(String args[]){
        double v = 60;
        double b, c, d;
        b = Unidade.converterParaBufunfa(t);
        c = Unidade.converterParaCascai(t);
        d = Unidade.converterParaDindim(t);
        System.out.println("Bufunfa: " + b);
        System.out.println("Cascai: " + c);
        System.out.println("Dindim: " + d);
    }
}
```

Herança

- Uma das grandes vantagens da POO é o uso do conceito de **herança**, que se assemelha à herança entre as espécies.
- De forma geral, na POO, uma classe (**subclasse**) pode herdar os **membros** de outra (**superclasse**) e, conseqüentemente, passa a detê-los também.
- Em Java, uma classe pode ser subclasse de apenas uma superclasse (não é permitida a **herança múltipla**).

Herança

- Para transformar uma classe em subclasse de uma superclasse, usamos a palavra reservada **extends** seguida do nome da superclasse após o nome da subclasse.
 - Exemplo 1: classe Pessoa herda os membros de Animal

```
public class Pessoa extends Animal{  
}
```

Herança

- Exemplo 2: superclasse

```
public class Animal{  
    public String nome;  
    public void mudarNome(String nome){  
        this.nome = nome;  
    }  
}
```

Herança

- Exemplo 2: subclasses adicionando membros

```
public class Pessoa extends Animal{  
    public void mostrarNome(){  
        System.out.println(nome);  
    }  
}  
  
public class Cachorro extends Animal{  
    public String raca;  
}
```

Herança

- Exemplo:

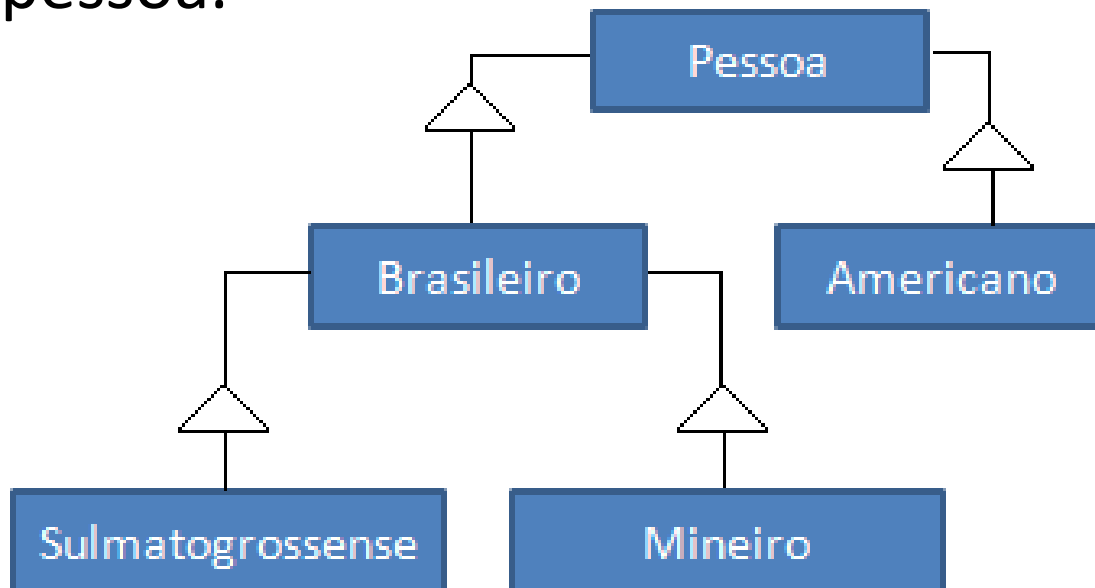
```
public class Programa{  
    public static void main(String[] args){  
        Pessoa p = new Pessoa();  
        Cachorro c = new Cachorro();  
        p.mudarNome("Maria");  
        p.mostrarNome();  
        c.mudarNome("Totó");  
    }  
}
```

Herança

- Em Java, todo construtor implicitamente faz uma chamada ao **construtor de sua superclasse** antes de executar seu conteúdo (até mesmo o construtor inserido automaticamente pela linguagem).
- Curiosidade: em Java, toda classe construída é implicitamente descendente da classe **java.lang.Object** (se a Oracle quisesse contar quantos objetos foram criados pela JVM, poderia fazer isso no construtor de Object).

Herança

- Exemplo: implementar esta hierarquia de classes de modo que sempre se saiba a quantidade de pessoas no mundo, independentemente da naturalidade da pessoa.

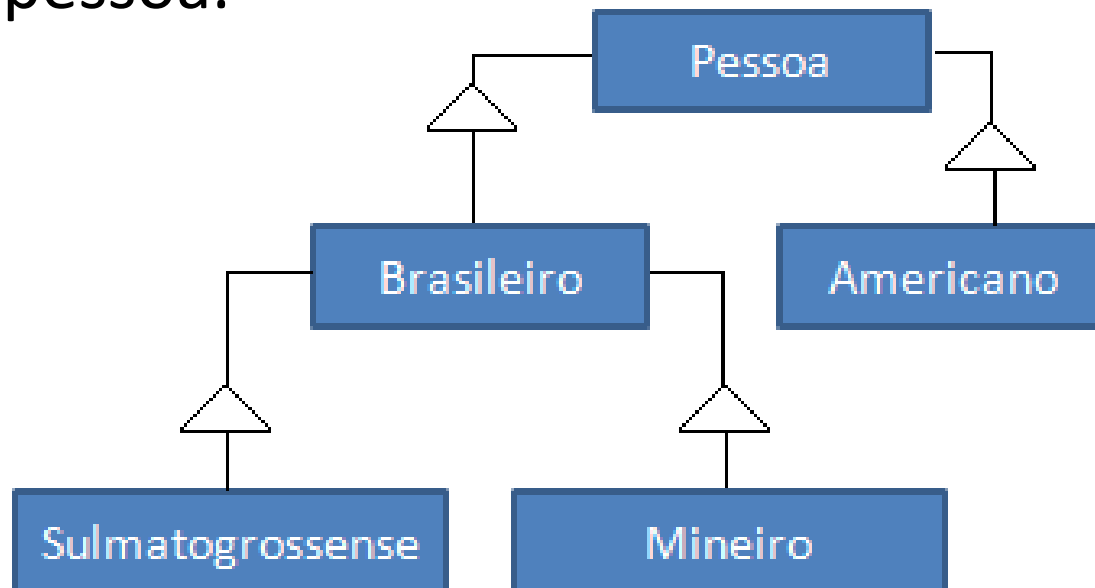


Herança

- Exemplo: implementar esta hierarquia de classes de modo que sempre se saiba a quantidade de pessoas no mundo, independentemente da naturalidade da pessoa.
 - Pessoa (nome, dataDeNascimento, sexo e imprimir())
 - Brasileiro (cpf, rg)
 - Americano (ssn)
 - Sulmatogrossense (ervaDeTererePreferida)
 - Mineiro (tipoPreferidoDePaoDeQueijo)

Herança

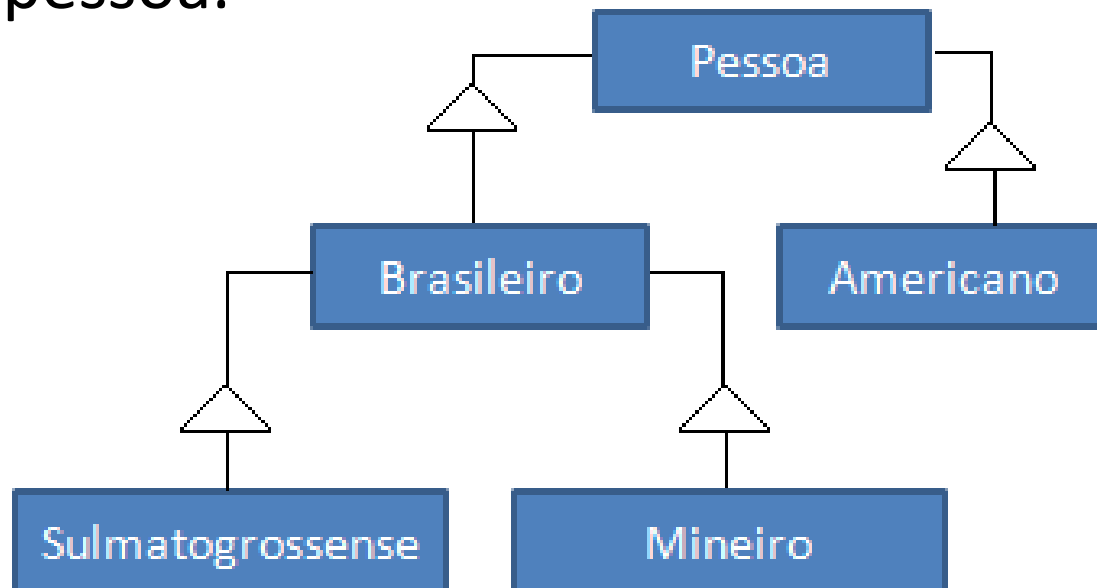
- Exemplo: implementar esta hierarquia de classes de modo que sempre se saiba a quantidade de pessoas no mundo, independentemente da naturalidade da pessoa.



O que acontece se Pessoa tiver um construtor que recebe o nome da pessoa?

Herança

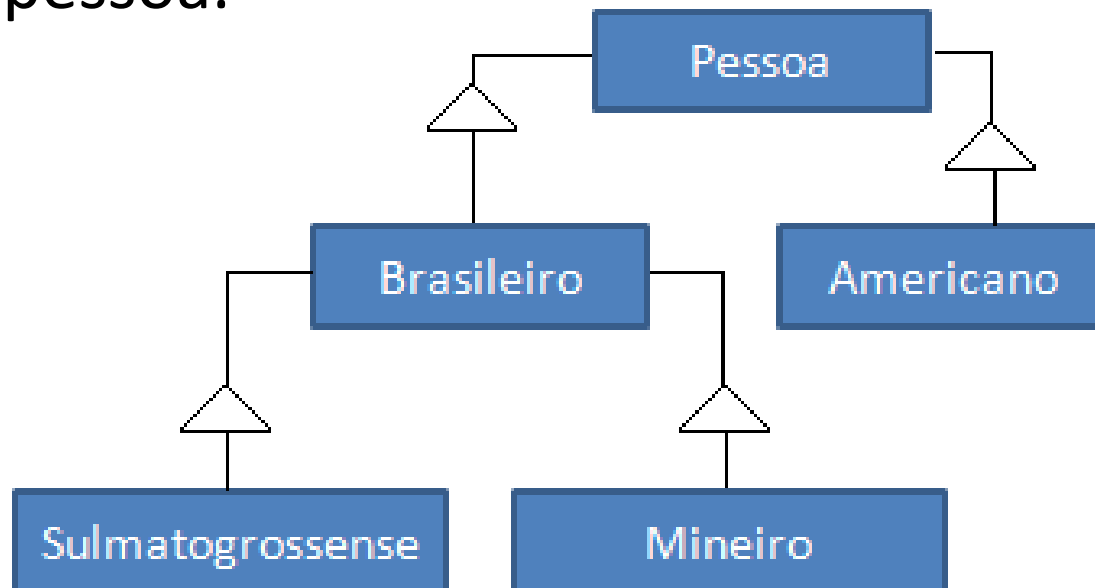
- Exemplo: implementar esta hierarquia de classes de modo que sempre se saiba a quantidade de pessoas no mundo, independentemente da naturalidade da pessoa.



O que acontece se Brasileiro também tiver uma implementação pública de imprimir()?

Herança

- Exemplo: implementar esta hierarquia de classes de modo que sempre se saiba a quantidade de pessoas no mundo, independentemente da naturalidade da pessoa.



O que acontece se Brasileiro também tiver uma implementação pública de imprimir()?

SOBREPOSIÇÃO!!!

Polimorfismo

- Com o conceito da herança, conseguimos explorar bem uma importante característica da POO chamada **polimorfismo**.
- Qualquer objeto em Java que passe por mais de um teste **É-UM** é dito um objeto polimórfico.
- Um objeto passa no teste **É-UM** quando podemos dizer que ele é uma instância de uma classe.

Polimorfismo

- Exemplos: considerando as classes Animal, Pessoa e Cachorro implementados anteriormente e supondo uma instância de Pessoa e outra de Cachorro (chamadas de dog e person), temos que:
 - dog É-UM Cachorro
 - person É-UM Pessoa
 - dog É-UM Animal
 - person É-UM Animal
 - dog É-UM Object
 - person É-UM Object
 - dog NÃO É-UM Pessoa
 - person NÃO É-UM Cachorro

Polimorfismo

- Para identificar se um objeto referenciado por uma variável de referência passa em um teste É-UM, podemos utilizar o **instanceof**.

```
Animal dog = new Dog();  
if (dog instanceof Dog){  
    System.out.println("É-UM Dog");  
}  
if (dog instanceof Pessoa){  
    System.out.println("É-UM Pessoa");  
}
```

Polimorfismo

- Podemos lidar com o polimorfismo quando usamos variáveis de referência.
- Embora toda variável de referência tenha de ser associada a um único tipo, ela pode fazer referência a qualquer objeto que passe no teste É-UM do seu tipo.

```
Animal dog;  
dog = new Cachorro();  
dog = new Pessoa();
```

```
Cachorro animal;  
animal = new Cachorro();  
animal = new Animal();  
animal = new Pessoa();
```


Polimorfismo

- Ao atribuírmos a uma referência um objeto de classe inferior (na hierarquia) à classe da referência, “perdemos” alguns membros do objeto referenciado.

```
Animal dog;  
dog = new Cachorro();  
dog.nome = “Totó”;  
dog.latir();
```

```
Object animal;  
animal = new Pessoa();  
animal.nome = “Ana”;
```

Polimorfismo

- Exemplo:

```
Sulmatogrossense s = new Sulmatogrossense();
```

```
Mineiro m = new Mineiro();
```

```
Americano a = new Americano();
```

```
Pessoa p1, p2, p3;
```

```
Brasileiro b1 , b2;
```

```
p1 = b1 = s;
```

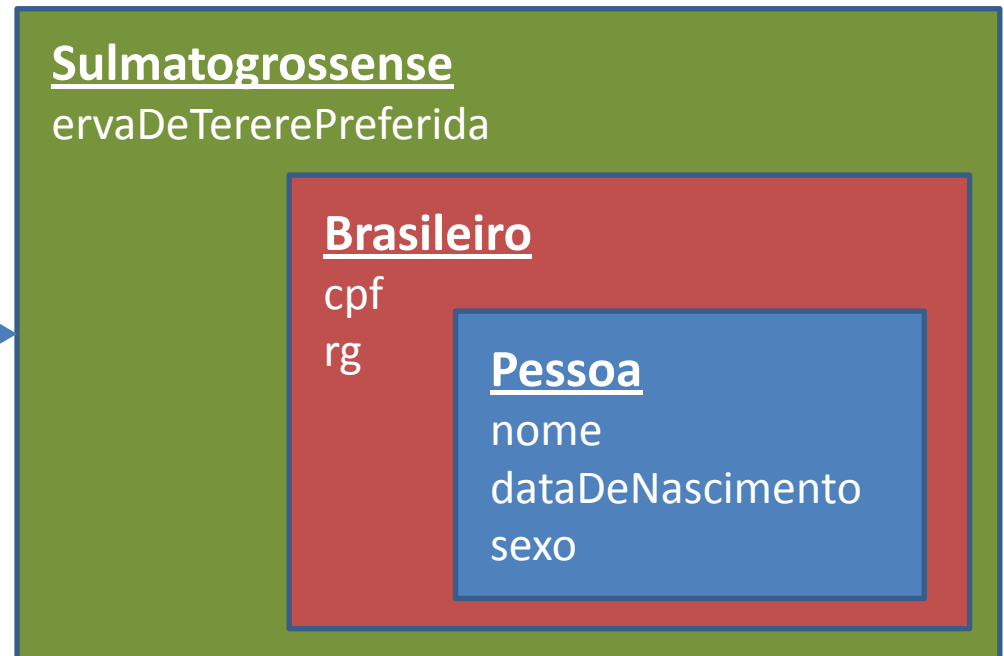
```
p2 = b2 = m;
```

```
p3 = a;
```

Polimorfismo

- Exemplo:

`new Sulmatogrossense()`



Polimorfismo

- Exemplo:

`new Mineiro()`



Mineiro

tipoPreferidoDePaoDeQueijo

Brasileiro

cpf
rg

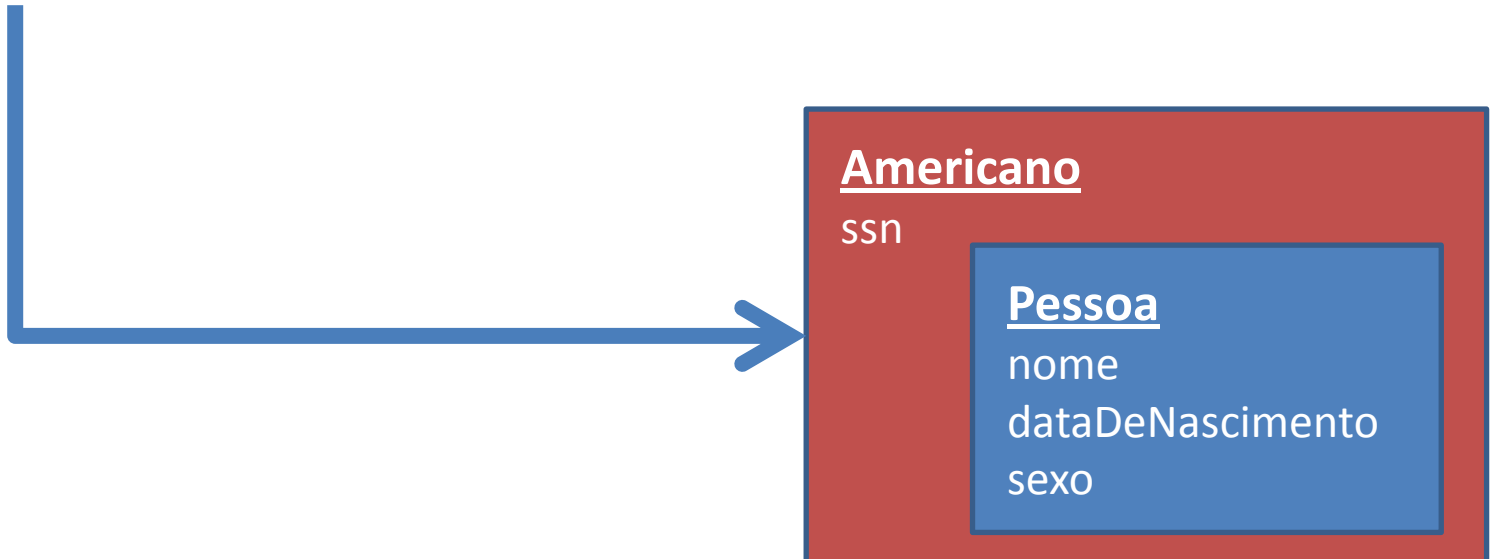
Pessoa

nome
dataDeNascimento
sexo

Polimorfismo

- Exemplo:

`new Americano()`



Polimorfismo

- Exemplo:

Sulmatogrossense s



Sulmatogrossense

ervaDeTererePreferida

Brasileiro

cpf
rg

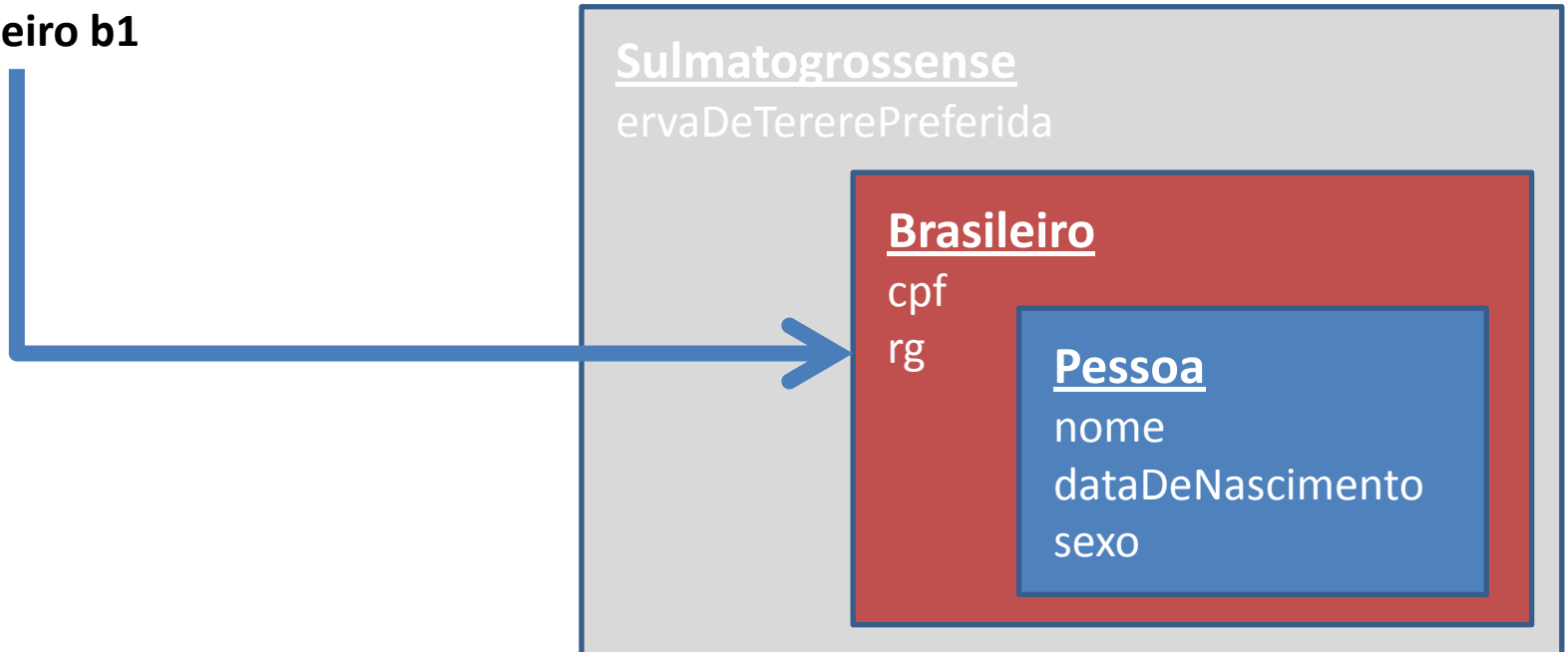
Pessoa

nome
dataDeNascimento
sexo

Polimorfismo

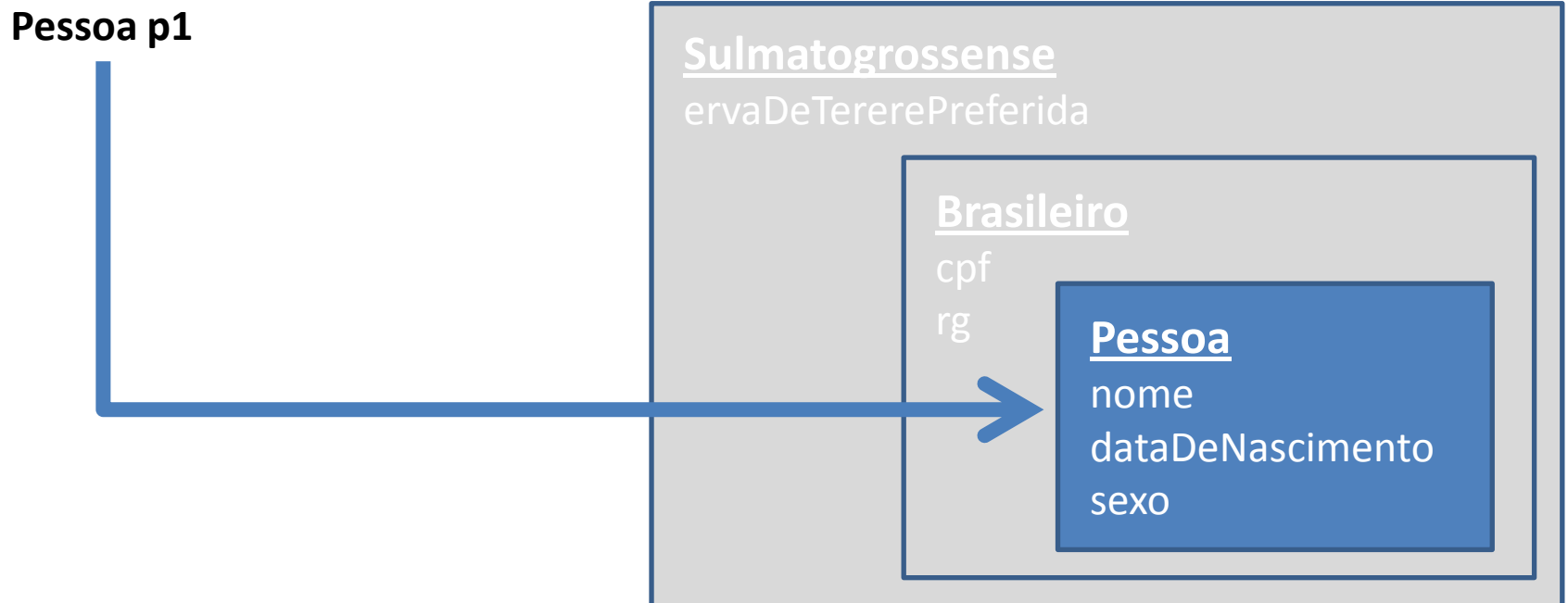
- Exemplo:

Brasileiro b1



Polimorfismo

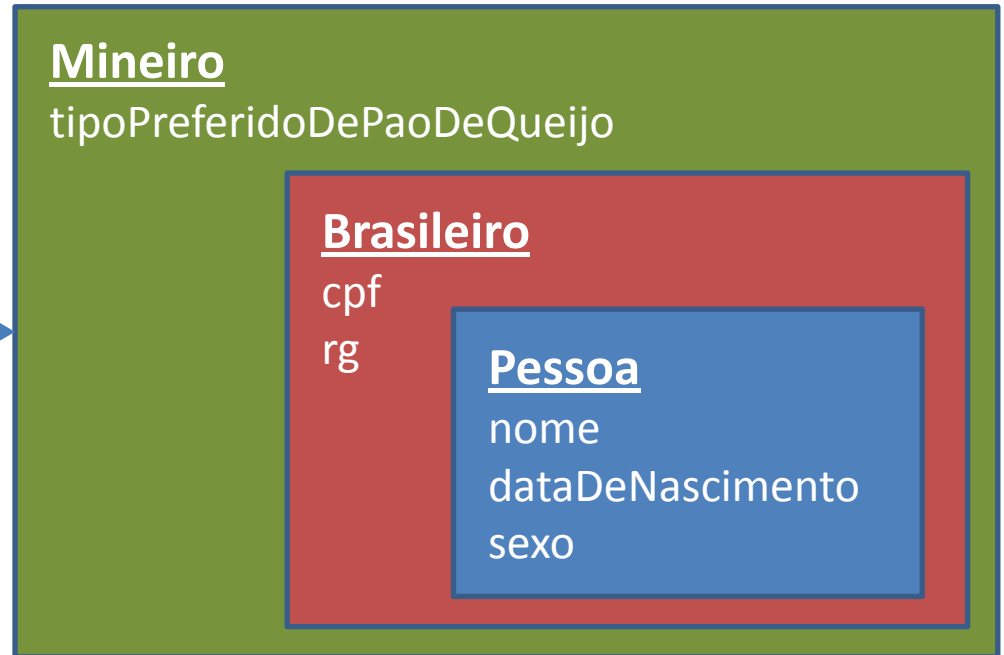
- Exemplo:



Polimorfismo

- Exemplo:

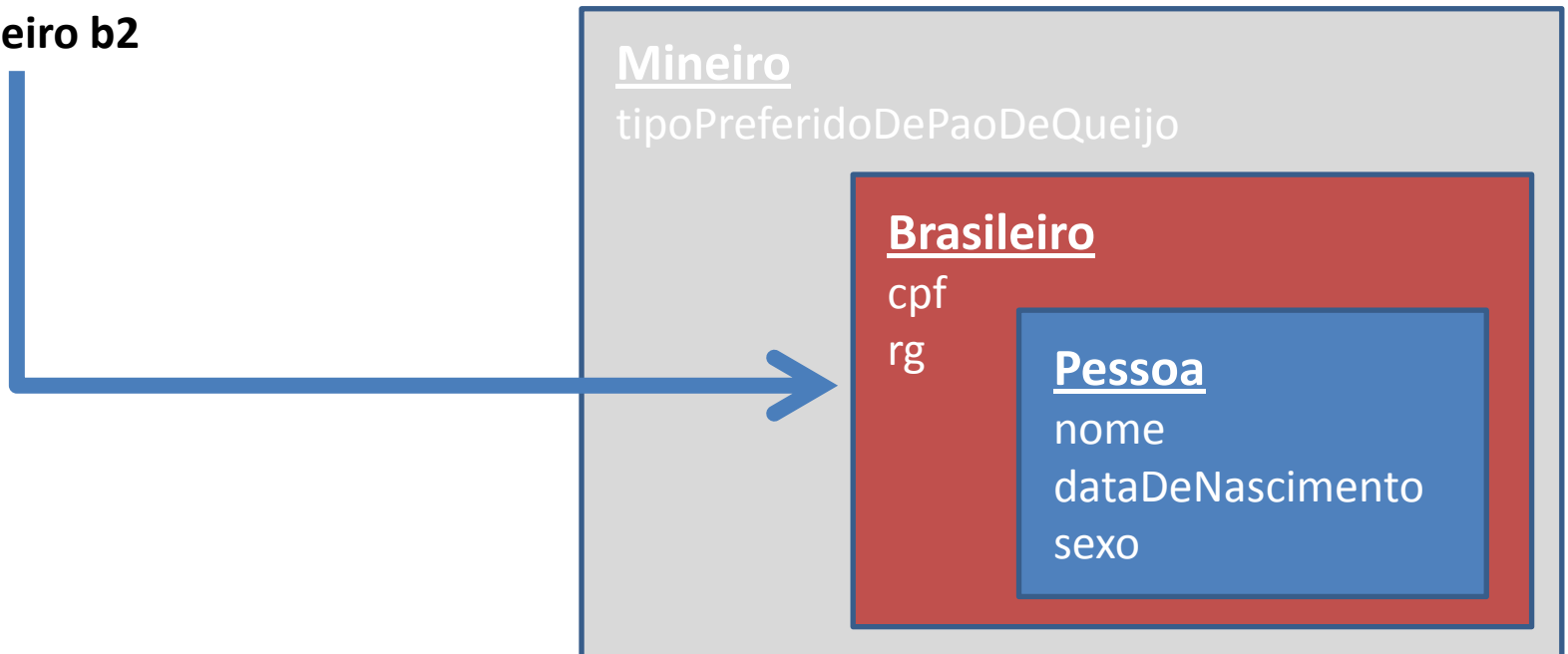
Mineiro m



Polimorfismo

- Exemplo:

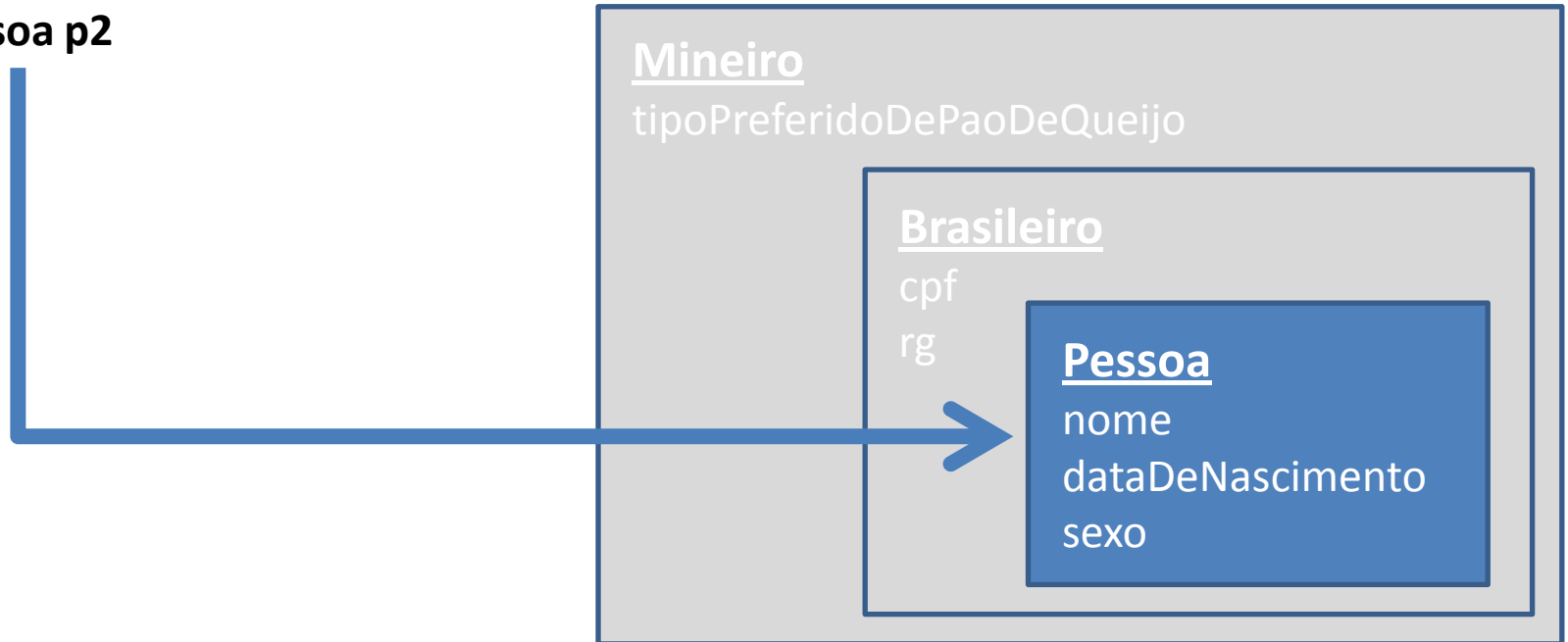
Brasileiro b2



Polimorfismo

- Exemplo:

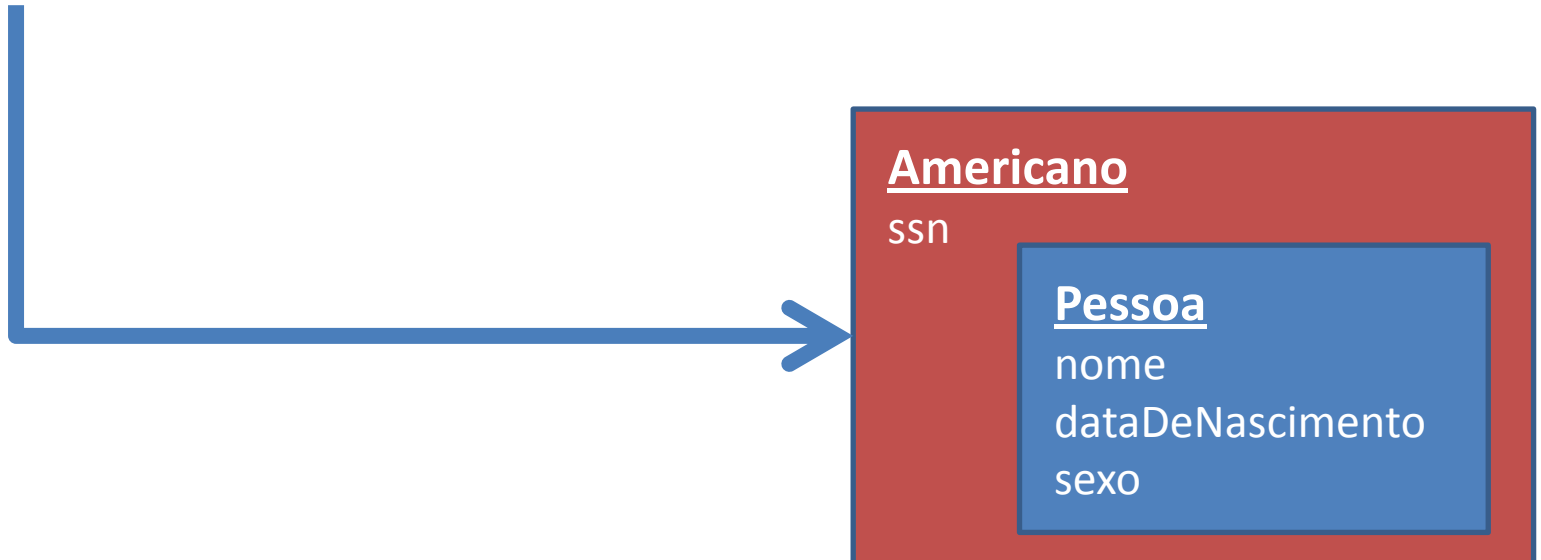
Pessoa p2



Polimorfismo

- Exemplo:

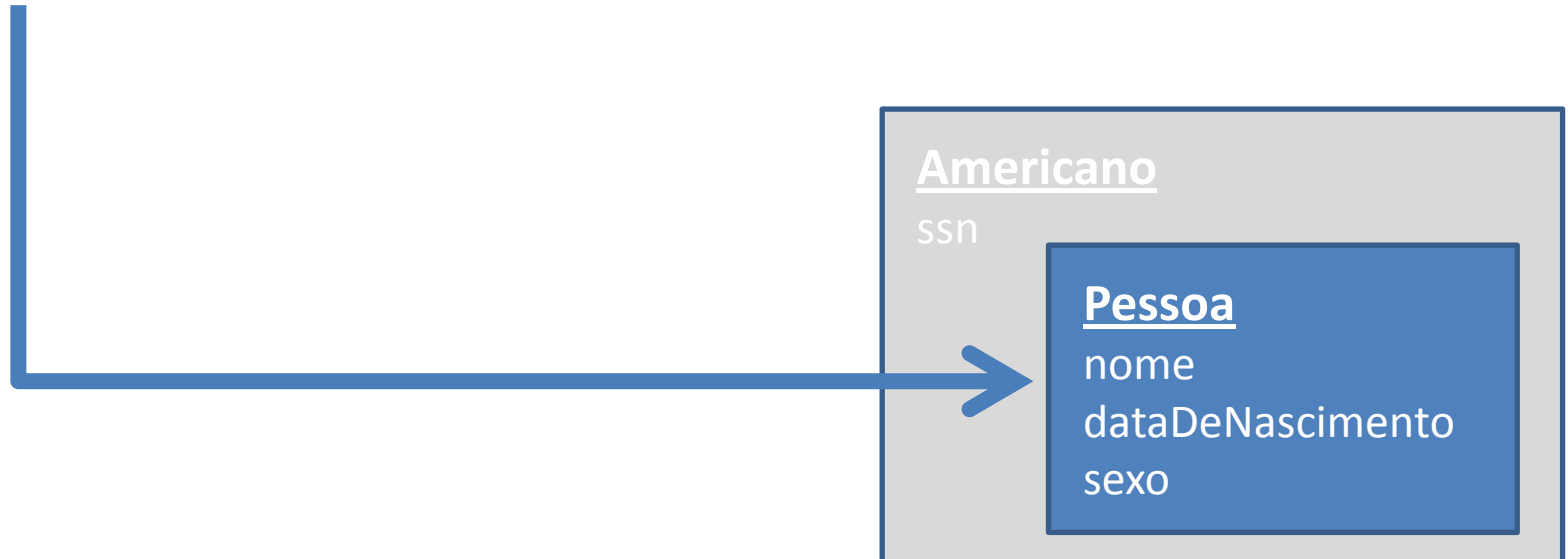
Americano a



Polimorfismo

- Exemplo:

Pessoa p3



Polimorfismo

- Para voltarmos a ver um objeto na ótica de sua classe por meio de uma variável de referência de classe superior na hierarquia, temos de realizar uma “conversão”, chamada *cast*.
- Para realizar cast, utilizamos a seguinte sintaxe:
(<classe destino>) <referência a ser convertida>
 - Exemplo:

```
Pessoa p2= new Mineiro();  
Brasileiro b2= (Brasileiro) p2;  
b2.rg = “1234 SSP/MS”;
```

Polimorfismo

- Para voltarmos a ver um objeto na ótica de sua classe por meio de uma variável de referência de classe superior na hierarquia, temos de realizar uma “conversão”, chamada *cast*.
- Para realizar cast, utilizamos a seguinte sintaxe:
(<classe destino>) <referência a ser convertida>

– Exemplo:

Relembrando:

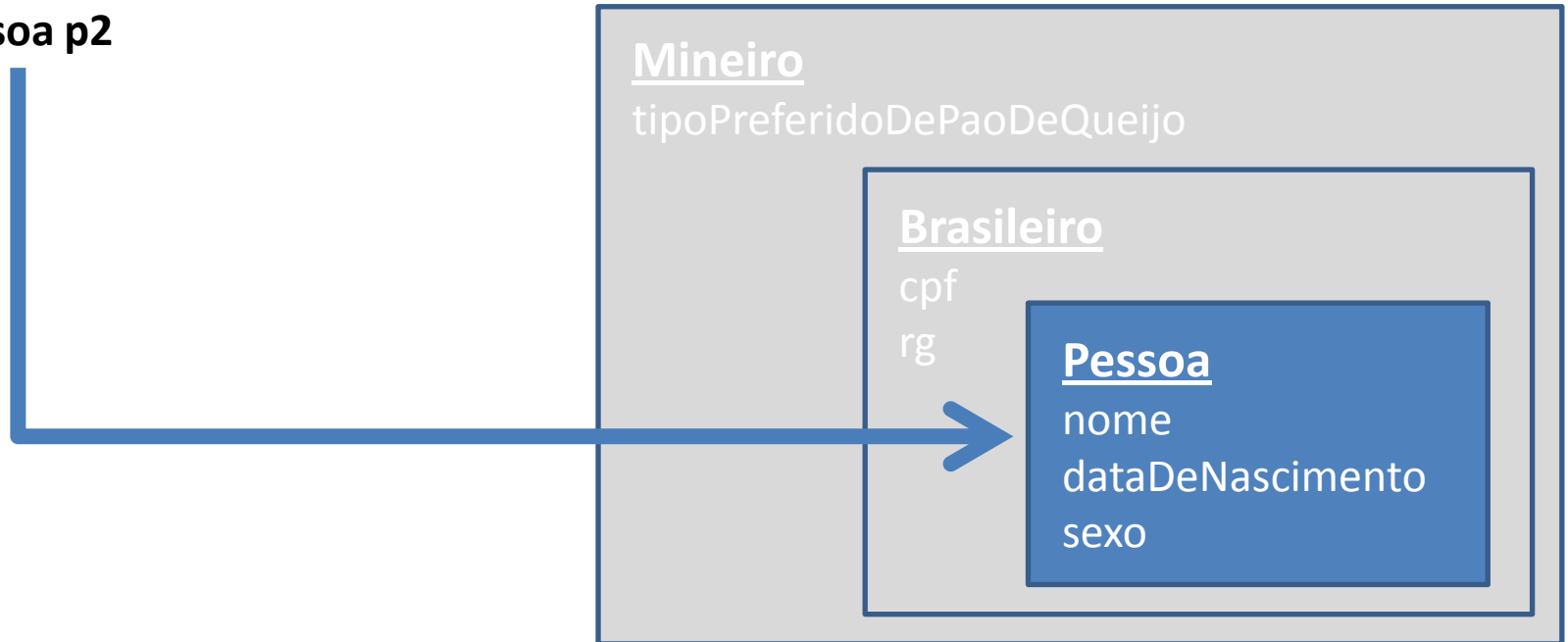
Embora o objeto tenha o membro acessado, isto NÃO compila!!

```
Pessoa p2= new Mineiro();  
Brasileiro b2= (Brasileiro) p2;  
b2.rg = "1234 SSP/MS";  
p2.rg = "1234 SSP/MS";
```

Polimorfismo

- Exemplo:

Pessoa p2

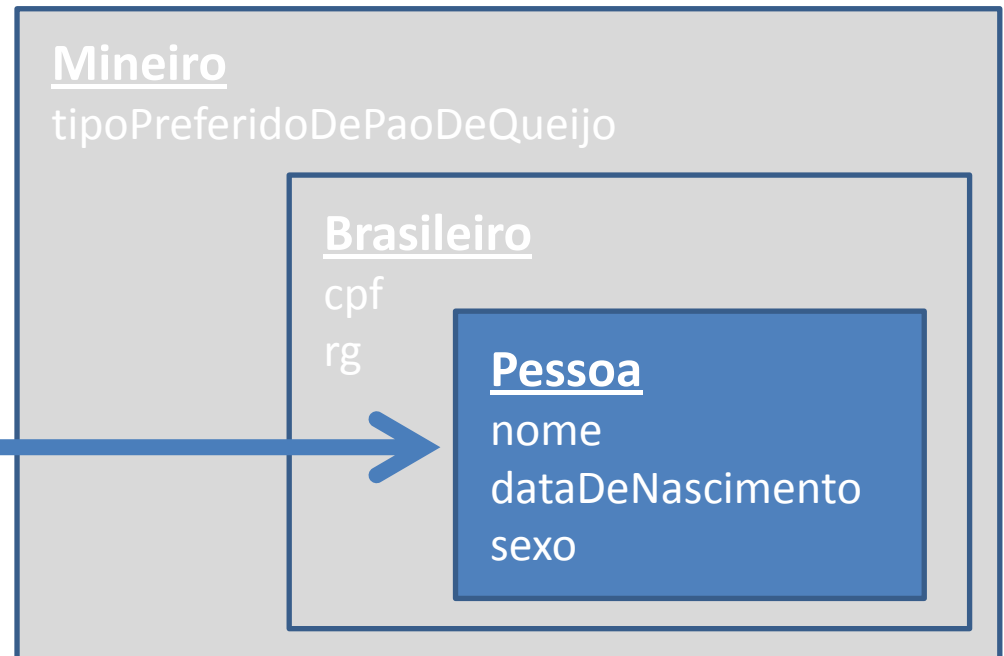


Polimorfismo

- Exemplo:

Pessoa p2

Note que o atributo **rg** existe,
mas não está visível para **p2**.
Portanto, não pode ser acessado
por p2

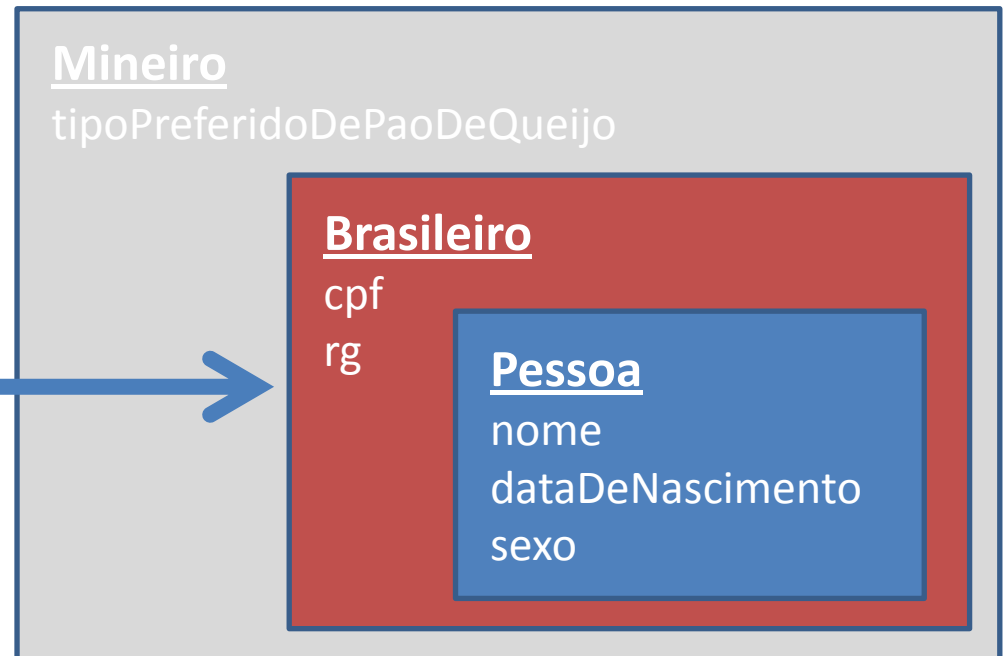


Polimorfismo

- Exemplo:

Brasileiro b2 = (Brasileiro) p2;

Porém, após o cast acima, conseguimos acessar **rg** por **b2**. Note que ainda estamos lidando com o **MESMO** objeto.



Laboratório

- Dadas as classes Conta e Programa, dadas nas páginas seguintes, construa as classes restantes e implemente os métodos necessários na classe Programa (sem alterar o método *main*) para que ela possa ser compilada e executada, produzindo a seguinte saída:

Laboratório

- *Tentativa de deposito de -5.0 na conta 1111 negada!*
- *Tentativa de deposito de 100.0 na conta 1111 autorizada!*
- *Tentativa de deposito de 100.0 na conta 2222 autorizada!*
- *Tentativa de deposito de 100.0 na conta 3333 autorizada!*
- *Tentativa de saque de 50.0 na conta 1111 autorizada!*
- *Tentativa de saque de 50.0 na conta 2222 autorizada!*
- *Tentativa de saque de 50.0 na conta 3333 negada!*
- *Tentativa de saque de 49.8 na conta 1111 autorizada!*
- *Tentativa de saque de 49.8 na conta 2222 negada!*
- *Tentativa de saque de 49.9 na conta 3333 negada!*
- *Saldo da conta 1111: 0.20*
- *Saldo da conta 2222: 49.75*
- *Saldo da conta 3333: 100.0*

Laboratório

- Suponha que seja cobrado 0,5% sobre todo saque efetuado na classe que modela a conta corrente seja (ao sacar R\$ 50,00 é cobrado R\$ 50,25).

Laboratório

```
package br.com.app1;

public class Conta{
    private double saldo;
    public String numero;
    public boolean efetuarDeposito(double valor){
        if (valor <= 0){
            return false;
        }
        ajustarSaldo(valor);
        return true;
    }

    protected void ajustarSaldo(double valor){
        saldo += valor;
    }

    public double getSaldo(){
        return saldo;
    }
}
```

Laboratório

```
package br.com.app2;

import br.com.app1.Conta;
import br.com.app1.especial.ContaCorrente;
import br.com.app1.normal.ContaPoupanca;

public class Programa{
    public static void main(String[] args){
        Conta contaPoupanca, contaCorrente, conta;
        contaPoupanca = new ContaPoupanca();
        contaPoupanca.numero = "1111";
        contaCorrente = new ContaCorrente();
        contaCorrente.numero = "2222";
        conta = new Conta();
        conta.numero = "3333";
        efetuarDeposito(contaPoupanca, -5.0);
        efetuarDeposito(contaPoupanca, 100.0);
        efetuarDeposito(contaCorrente, 100.0);
        efetuarDeposito(conta, 100.0);
        efetuarSaque(contaPoupanca, 50);
        efetuarSaque(contaCorrente, 50);
        efetuarSaque(conta, 50.0);
        efetuarSaque(contaPoupanca, 49.80);
        efetuarSaque(contaCorrente, 49.80);
        efetuarSaque(conta, 49.90);
        mostrarSaldo(contaPoupanca);
        mostrarSaldo(contaCorrente);
        mostrarSaldo(conta);
    }
}
```

Encapsulamento

- O encapsulamento permite a **ocultação** de membros das classes (e das próprias classes) sob determinadas óticas.
- A cada **membro** é atribuído um **nível de acesso**, que define quais classes podem visualizá-lo.
- Os níveis de acesso são definidos pela inserção (ou ocultação) de um modificador de acesso.
- Em Java, temos **quatro níveis de acesso**, porém, **três modificadores de acesso**.

Encapsulamento

Nível	Modificador	Descrição
Privado	private	Nível mais restrito. Permite que o membro seja acessado apenas pela classe que o definiu.
Padrão	-	É acessível na classe onde foi definido e nas classes contidas no mesmo pacote.
Protegido	protected	Permite o acesso padrão e às subclasses da classe onde o membro foi definido (mesmo que a classe pertença a outro pacote).
Público	public	Permite acesso irrestrito.

Encapsulamento

Nível	Modificador	Acesso				Aplicabilidade	
		Classe	Pacote	Subclasses	Mundo	Classes	Membros
Privado	private	SIM	NÃO	NÃO	NÃO	NÃO	SIM
Padrão	-	SIM	SIM	NÃO	NÃO	SIM	SIM
Protegido	protected	SIM	SIM	SIM	NÃO	NÃO	SIM
Público	public	SIM	SIM	SIM	SIM	SIM	SIM

Encapsulamento

- Exemplo: observe a classe abaixo:

```
package pacote1;
public class Animal{
    private String nome;
    protected String especie;
    public String obterNome(){
        return nome;
    }
    void mudarNome(String nome){
        this.nome = nome;
    }
}
```

Encapsulamento

- Exemplo: acesso de subclasse em mesmo pacote

```
package pacote1;
```

```
public class Pessoa extends Animal{  
    public void mostrarNome(){  
        System.out.println(obterNome());  
    }  
}
```

OK!

Encapsulamento

- Exemplo: acesso de subclasse de outro pacote

```
package pacote2;
```

```
public class Cachorro extends Animal{  
    public void mostrarNome(){  
        System.out.println(obterNome());  
    }  
}
```

OK!

Encapsulamento

- Exemplo: acesso a membro privado por subclasse em mesmo pacote.

```
package pacote1;
```

```
public class Pessoa extends Animal{  
    public void nomeDaPessoa(String nome){  
        this.nome = nome; Erro!  
    }  
}
```

Encapsulamento

- Exemplo: acesso a membro privado por subclasse em pacote diferente

```
package pacote2;
```

```
public class Cachorro extends Animal{  
    public void nomeDoCachorro(String nome){  
        this.nome = nome; Erro!  
    }  
}
```

Encapsulamento

- Exemplo: acesso a membro de nível padrão por subclasse de mesmo pacote

```
package pacote1;
```

```
public class Pessoa extends Animal{  
    public void nomeDaPessoa(String nome){  
        mudarNome(nome);  
    }  
}
```

OK!

Encapsulamento

- Exemplo: acesso a membro de nível padrão por subclasse de pacotes diferentes

```
package pacote2;
```

```
public class Cachorro extends Animal{  
    public void nomeDoCachorro(String nome){  
        mudarNome(nome); Erro!  
    }  
}
```

Encapsulamento

- Exemplo: acesso a membro protegido por subclasse de mesmo pacote

```
package pacote1;
```

```
public class Pessoa extends Animal{  
    public String obterEspecie(){  
        return especie;  
    }  
}
```

OK!

Encapsulamento

- Exemplo: acesso a membro protegido por subclasse de pacotes diferentes

```
package pacote2;
```

```
public class Cachorro extends Animal{  
    public String obterEspecie(){  
        return especie;  
    }  
}
```

OK!

Encapsulamento

- Note que, de nada vale o nível de acesso do membro da classe, se a própria classe não for “visível”.
 - Exemplo:

```
package pacote1;
```

```
class Classe {  
    public int atributo;  
}
```

```
package pacote1.pacoteA;
```

```
class Programa {  
    public static void main(String[] args){  
        Classe x;  
        x = new Classe();  
        x.atributo = 3;  
    }  
}
```

Encapsulamento

- Habitualmente, utilizam-se atributos com nível de **acesso privado** combinados com métodos públicos de leitura e escrita.
- Os métodos de leitura são comumente (e convencionalmente) iniciados com *get* e os métodos de escrita com *set*.

```
private int atributo;  
public void setAtributo(int atributo){  
    this.atributo = atributo;  
}  
public int getAtributo(){  
    return atributo;  
}
```

Exemplo abrangente

- Criar uma agenda de contatos (sem restrição de tamanho máximo) em que cada contato possui um código, um nome e telefones. Pessoa jurídica pode ter 5 telefones e pessoas físicas apenas 3. Deve ser armazenado o CNPJ e a IE se for pessoa jurídica. Além disso, devem ser atendidas as seguintes restrições:

Exemplo abrangente

1. O nome do contato pode ser alterado uma única vez.
2. O código do contato deve ser atribuído automaticamente e é inalterável.
3. A classe onde é manipulada a agenda deve estar em pacote diferente da classe que representa o contato.

Exemplo abrangente

4. Um telefone pode ser residencial ou comercial.
5. Somente telefones válidos podem ser associados a contatos (com 10 dígitos e tipo igual a 1 ou 2).
6. O programa deve conter as seguintes opções:
 - Cadastrar contato
 - Cadastrar telefone
 - Mostrar contato

Laboratório

1. Crie uma classe chamada `Animal` (dentro de qualquer pacote) com três atributos privados – nome, espécie e comida favorita – e, no mínimo, um método – o procedimento `comer`, que informa o nome do animal e o alimento preferido que ele está comendo.

Laboratório

Em seguida, crie uma classe chamada Principal (também em qualquer pacote), que, ao ser executada, cria dois objetos da classe Animal, atribui valores aos atributos desses objetos, invoca o método comer de ambos e exibe os atributos de cada um. A exibição dos atributos de cada objeto deve ser realizada dentro de um procedimento estático da classe Principal chamado exibir, que recebe a referência do animal a ser exibido como parâmetro.

Laboratório

2. Crie uma classe chamada Cliente com os atributos nome, telefone e CPF. Utilize os conceitos aprendidos para garantir que qualquer objeto de Cliente criado em qualquer outra classe (produzida por qualquer programador) sempre terá valores válidos nos atributos telefone e CPF.

Laboratório

Considere que um telefone válido é composto por 10 dígitos e que o CPF é composto por 11 dígitos ou 14 caracteres, sendo 11 deles dígitos, dois pontos e um hífen – no formato padrão de CPF.

3. Crie uma classe chamada `Funcionario` que contenha nome, cargo e salário. O salário do funcionário nunca pode ser menor que zero, seu nome deve ser configurado uma única vez e se manter inalterável e o nome do cargo deve conter, no mínimo, 5 caracteres.

Laboratório

Ao construir objetos dessa classe, os três atributos devem ser configurados (caso alguma das regras seja infringida, o atributo não é configurado). Portanto, o cargo de qualquer objeto criado deve ser nulo ou um texto maior que 4 e o salário deve ser sempre maior que (ou igual a) zero.

Classes abstratas

- Até o momento, criamos apenas classes **concretas**. Uma classe é concreta quando é possível a criação de objetos que pertençam diretamente a ela.
- No entanto, em alguns casos, pode ser necessária a criação de classes que não instanciam objetos diretamente, mas servem somente de base para outras subclasses. Tal cenário pode ser resolvido com a utilização de classes **abstratas**.
 - Exemplo: Classes Pessoa, PessoaFisica e PessoaJuridica

Classes abstratas

- Por definição, em Java, uma classe torna-se abstrata quando, em sua declaração, existe a palavra reservada *abstract*.
 - Exemplo:

```
public abstract class Pessoa{  
    ...  
}
```

Classes abstratas

- Como citado, nenhum objeto pode ser instanciado de uma classe abstrata (logo, `new Pessoa()` causaria um erro de compilação no exemplo anterior).
- Uma classe abstrata pode possuir subclasses concretas naturalmente e podemos ter objetos dessas subclasses instanciados.

– Exemplo:

```
public class PessoaFisica extends Pessoa{  
    ...  
}
```


Classes abstratas

- Assim como nas classes concretas, nas classes abstratas, temos atributos e métodos. No entanto, os métodos podem ser concretos ou **abstratos**.
- Métodos concretos possuem implementação (como os métodos criados até o momento), enquanto os métodos abstratos possuem apenas sua assinatura, sem implementação (usa-se “;” em vez de “{”).
 - Exemplo:

```
public abstract void metodoAbstrato();
```

Classes abstratas

- Qualquer subclasse concreta de uma superclasse abstrata deve OBRIGATORIAMENTE implementar todos os métodos abstratos não implementados de sua hierarquia.



Interfaces

- Quando temos classes abstratas em que todos os seus métodos também são abstratos e seus atributos são todos constantes, podemos substituí-las por interfaces, que, basicamente, são isso.
- Dentro da interface, todos os métodos e atributos são públicos, nenhum método pode conter implementação (ou seja, eles são implicitamente abstratos) e todos os atributos são constantes.

Interfaces

- Não existem objetos instanciados diretamente de interfaces, como ocorre com as classes abstratas. Assim, é necessária a existência de uma classe concreta para criação dos objetos.
- A relação entre uma interface e uma classe é idêntico à relação de herança, porém, por conveniência, em vez de se dizer que uma classe herda uma interface, diz-se que uma classe **implementa** uma interface.

Interfaces

- Em Java, declara-se uma interface de modo similar à declaração de classe, mas, em vez de **class**, utiliza-se a palavra reservada **interface**.

```
public interface MinhaInterface{  
}
```

Interfaces

- Como os métodos e atributos são implícita e obrigatoriamente públicos, a utilização do modificador de acesso `public` é opcional. O mesmo se aplica às palavras reservadas `final` e `abstract` nos atributos e métodos, respectivamente.

```
public interface MinhaInterface{  
    int atr1; // ou public final int atr2;  
    void metodo(); // ou public abstract ...  
}
```

Interfaces

- Para implementar uma interface, utilizamos a palavra reservada implements.

```
public class MinhaClasse implements MinhaInterface{  
    public void metodo(){  
        ...  
    }  
}
```

Interfaces

- Assim como podemos criar variáveis de referência para classes, também podemos criá-las para interfaces e a atribuição segue a mesma regra (o objeto referido deve passar no teste É-UM do tipo da variável de referência).

```
MinhaInterface mi;  
MinhaClasse mc;  
mi = new MinhaClasse();  
mc = new MinhaClasse();
```


Interfaces

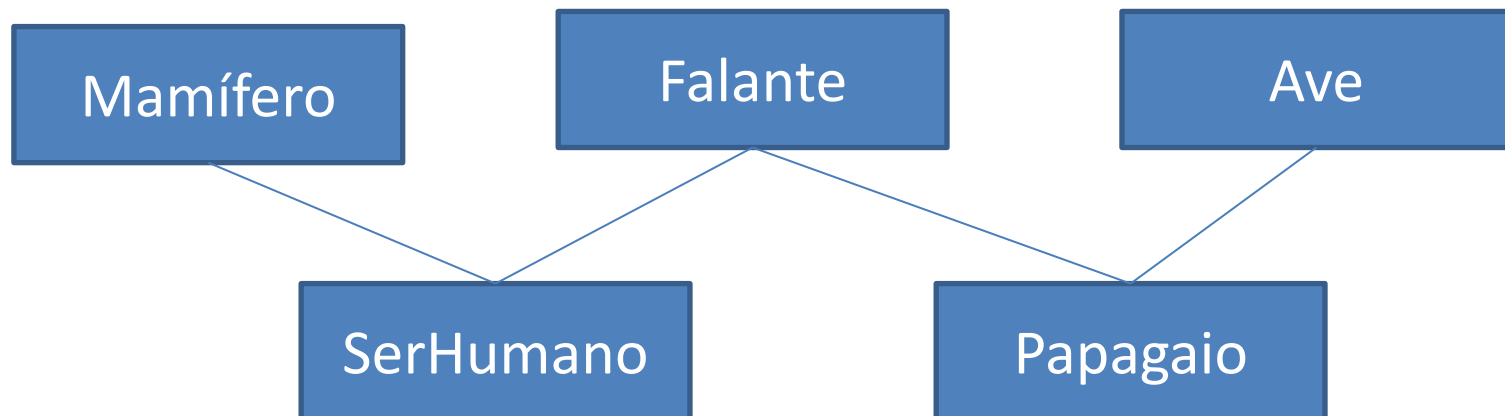
- Diferentemente do que ocorre na herança, uma classe pode implementar quantas interfaces forem necessárias, sem limitações. Nesses casos, o polimorfismo continua sendo aplicável normalmente.

```
public class X implements A, B, C {  
}
```

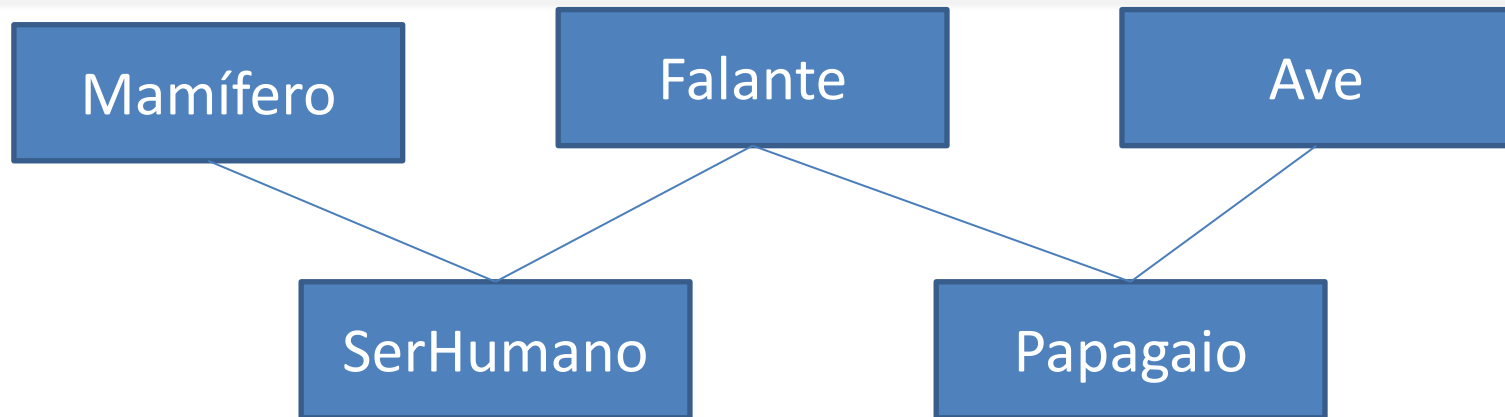
```
X var1 = new X();  
A var2 = new X();  
B var3 = new X();  
C var4 = new X();
```

Interfaces

- Com essa facilidade das interfaces, podemos contornar a limitação da herança múltipla.
 - Exemplo: Sem interfaces, seria impossível implementar esta herança múltipla em Java.



Interfaces



```
public class SerHumano implements Falante, Mamifero{  
}
```

```
public class Papagaio implements Falante, Ave{  
}
```

Interfaces

- Vale lembrar que toda classe concreta é OBRIGADA a implementar todos os métodos abstratos não implementados existentes na sua hierarquia.

Tratamento de exceções

- Em programação, uma **exceção**, como o nome sugere, é uma condição excepcional que altera o fluxo normal do programa.
 - Exemplos:
 - Usuário digita uma letra em um campo em que se esperam números
 - Um arquivo de acesso restrito está tentando ser apagado
- Para produzir programas robustos, torna-se importante a **manipulação** adequada das exceções.

Tratamento de exceções

- Em Java, utilizamos a palavra-chave **try** para manipularmos exceções. Após o try, é iniciado um bloco de código, chamado de **região protegida**, que trata-se de um bloco onde as instruções possivelmente podem gerar erros.

- Exemplo: conversão de um texto para inteiro

- ```
String x = JOptionPane.showInputDialog("Digite um número: ");
```

- ```
try{
```

- ```
 int valor = Integer.parseInt(x);
```

- ```
}
```

Tratamento de exceções

- Quando uma instrução causa um erro, diz-se que uma exceção foi **lançada**.
- Dentro da região protegida, **diversas exceções** diferentes *podem* ser lançadas.
 - Exemplo: na tentativa de escrever um texto dentro de um arquivo, podemos ter uma exceção causada pela inexistência do arquivo e outro causada por não termos permissão de escrita nesse arquivo.

Tratamento de exceções

- Em alguns casos, é conveniente manipular a exceção de acordo com seu tipo.
 - Exemplo: é interessante avisar ao usuário que o arquivo não foi encontrado ou que ele não tem permissão para escrever no arquivo, em vez de enviar uma mensagem genérica dizendo que “houve problema na gravação”.
- Para manipularmos as exceções, utilizamos a palavra reservada **catch** seguida do tipo da exceção. Para cada tipo de exceção, podemos usar um catch.

Tratamento de exceções

- Exemplo com alto nível de abstração:

```
try{
    arquivo = abrirArquivo("C:\arquivo.txt");
    arquivo.escrever("Oi, Mundo");
    arquivo.fechar();
} catch(ErroDeArquivoInexistente){
    System.out.println("O arquivo c:\arquivo.txt não existe!");
} catch(ErroDePermissaoDeEscrita){
    System.out.println("Arquivo c:\arquivo.txt não pode ser alterado.");
}
```

Tratamento de exceções

- Além do bloco catch, temos também o bloco **finally** que é associado ao bloco try.
- O bloco finally SEMPRE é executado, sendo lançada ou não uma exceção.
- Geralmente, ele é utilizado para rotinas de **limpeza**, como fechamento de arquivo, liberação de recursos, entre outras.

Tratamento de exceções

- Exemplo:

```
try{
    arquivo = abrirArquivo("C:\arquivo.txt");
    arquivo.escrever("Oi, Mundo");
}catch(ErroDePermissaoDeEscrita){
    System.out.println("Arquivo c:\arquivo.txt não pode ser alterado.");
}catch(ErroDeDiscoCheio){
    System.out.println("O disco está cheio.");
}finally{
    arquivo.fechar();
}
```

Tratamento de exceções

- Se uma exceção for lançada dentro do bloco try e **não houver um bloco catch** esperando por ela, o método onde este bloco try-catch está repassa a exceção para o método que o invocou (esse, por sua vez, pode tratar ou não a exceção). Isso se chamada **propagação de exceção**.
- Se uma exceção for lançada e propagada por toda a pilha de chamada dos métodos sem tratamento, a **JVM a tratará** (mostrando o rastreamento).

Tratamento de exceções

- Exemplo:

```
try{
    arquivo = abrirArquivo("C:\arquivo.txt");
    arquivo.escrever("Oi, Mundo");
}catch(ErroDePermissaoDeEscrita){
    System.out.println("Arquivo c:\arquivo.txt não pode ser alterado.");
}catch(ErroDeDiscoCheio){
    System.out.println("O disco está cheio.");
}finally{
    arquivo.fechar();
}
```

Tratamento de exceções

- Por definição, toda exceção lançada é um objeto de alguma subclasse de **java.lang.Exception** (que, obviamente, é subclasse de `java.lang.Object`) que é inserido no manipulador de exceções como um argumento para o `catch`.
- Todo bloco `catch` tem sintaxe semelhante a esta:

```
catch(ClasseDaExcecao variavel){  
    System.out.println(variavel.getMessage());  
}
```

Tratamento de exceções

- Quando uma exceção é lançada, os blocos catch são analisados (**de cima para baixo**) até que seja encontrada uma correspondência com o tipo da exceção lançada (ou a propaga).
- Essa correspondência é “**polimórfica**”. Logo, se houver um bloco catch para uma classe A e for lançada uma exceção da classe B, que é subclasse de A, essa exceção será tratada nesse catch.