INE5603 Introdução à POO

Prof. A. G. Silva

14 de agosto de 2017

Tópicos

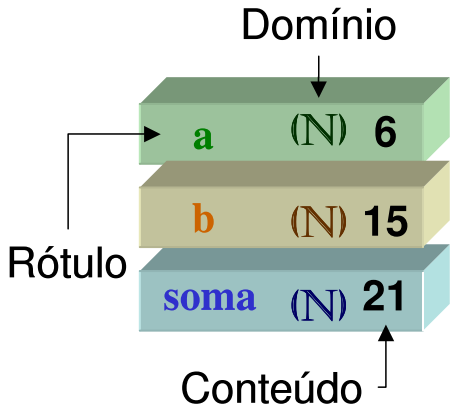
- Documentação de código
- Variáveis
- Tipos e valores
- Atribuição
- Aspectos de funcionamento/interface
- Escopo
- Parâmetros
- Passagem de argumentos
- Implementação de objetos
- Expressões aritméticas
- Expressões lógicas

Documentação de código

- Uso de nomes significativos para os elementos utilizados
- Documentação interna mais detalhada possível para esclarecimento do código:
  - Tudo que for colocado entre `/*` e `*/`
  - Única linha por `//`
  - Os comentários são ignorados pelo compilador
- Outras formas: diagramas, textos, manuais, ...
- Facilitação do processo de manutenção

Variáveis

Variável:
- Nome (rótulo)
- Tipo (domínio)
- Valor (conteúdo)
- Escopo (tempo de vida)



- **Declaração:** reservar espaço na memória; associar com identificador
- **Sintaxes:**

```
tipo nome;
tipo nome = valor;
tipo nome1, nome2, nome3;
tipo nome1 = valor, nome2;
```

Tipos e valores (I)

- Os tipos *int*, *double*, *char*, *boolean*, entre outros, são primitivos
- Importante para definir o tamanho de memória (quantidade de bytes alocados) e a interpretação correta dos bits
- Quanto maior a quantidade de bytes, maior a representatividade
- Tipos:
  - **Tipo char:** 2 bytes (UNICODE), de 0 a 65535

```
char a = 'A';
char op = '+';
```

  - **Tipo boolean:** 1 bit, 0 ou 1

```
boolean x = true;
boolean y = false;
```

Tipos e valores (II)

- Representatividade de inteiros:
  - **Tipo int:** 4 bytes, de -2147483648 a 2147483647

```
int a = -34;
int b;
b = 3492;
```

  - **Tipo short:** 2 bytes, de -32768 a 32767

```
short idade;
idade = 82;
short valorB = 5;
```

  - **Tipo byte:** 1 byte, de -128 a 127

```
byte quant;
quant = 82
```

```
byte nd = 473; //gera um erro de compilação
```

  - **Tipo long:** 8 bytes, de -9223372036854775808L a 9223372036854775807L

```
long valor = -73L;
long x = 34;
```

Tipos e valores (III)

- Representatividade de números com parte fracionária:
  - **Tipo double:** 8 bytes, de -1.79769313486231570E+308 a +1.79769313486231570E+308

```
double valor, salario, imposto;
valor = -2.45E-4;
imposto = 2340.12d;
salario = 2300.45;
```

  - **Tipo float:** 4 bytes, de -3.40282347E+38F a +3.40282347E+38F

```
float valor, salario;
valor = 7.265E-4f;
salario = 2500.00F;
```

Atribuição (I)

- Valor armazenado em determinada posição de memória. Forma geral: `Elemento = Expressão;`
  - `Elemento` é uma variável (incluindo identificação do objeto), `Expressão` é o valor a ser armazenado
  - `Elemento` e `Expressão` devem ser do mesmo tipo ou de tipos compatíveis (compatibilidade será vista adiante)
  - Se `Elemento` for a identificação de um objeto de uma classe X, então `Expressão` deve ser o endereço de uma instância de X ou de classe derivada (especializada) de X (veremos os detalhes mais à frente)
  - `Expressão` pode ser escrita como: valor, variável, objeto, expressão aritmética, expressão lógica

Atribuição (II)

- Exemplos:

```
double a, b, c;
float nota;
boolean aprovado;
char letra;
int contador = 0;  //declaracao e atribuicao

letra = 'A';      //atribuicao
a = 1.05;         //atribuicao
b = -4.0D;        //atribuicao
c = 3.0E-2;       //atribuicao
c = a + b;        //atribuicao
nota = 9.5f;      //atribuicao

//atribuicoes de expressoes aritmeticas e logicas:
contador = contador + 1;
aprovado = nota > 6.0;
```

Atribuição (III)

- Variáveis devem ser declaradas e inicializadas:

```
double val, custo;      //declaracao
int i = 0;              //declaracao e inicializacao
val = 3.45;             //inicializacao
val = val + custo;      //ERRO: variavel nao inicializada
```

- Atribuições permitidas:

double $\leftarrow$ *float* $\leftarrow$ *long* $\leftarrow$ *int* $\leftarrow$ *short* $\leftarrow$ *byte*

```
float vX = 12.9f;
double vY = vX;
short k = 5;
int j = k;
```

Atribuição (III)

- Conversão explícita de tipos (retipagem ou *casting*)

```
float f = (float) 12.9;
// 12.9 e' convertido explicitamente (retipagem ou
    casting)

int k = (int) 3.6;
// k=3, perda por truncamento

byte b = (byte) 2000;
// aceita, mas nao adequada (valor armazenado sera'
    totalmente diferente)
```

Aspectos de funcionamento de objetos

- Construção (instanciação) – alocação do objeto em memória
- É uma boa prática inicializar os atributos, quando possível
- Se não explicitamente inicializado, Java automaticamente assume um determinado valor (só para atributos)
- Uma classe pode ter vários construtores (um construtor é executado conforme sua assinatura)

Aspectos de funcionamento de objetos

- Pode-se ter várias instâncias:

```
umCirculo = new Circulo();
outroCirculo = new Circulo();
```

- Execução de método:

```
public void recebaValorRaio( double vRaio )
```

Parâmetro

Argumento

```
umCirculo.recebaValorRaio( 7.5 );
```

- Outros detalhes (como escopo) serão vistos mais adiante

Implementação de métodos – introdução

- Método consiste na implementação de um serviço
- Sequência de instruções executadas para uma ação de um objeto
- Métodos (em vermelho) de uma abstração de pessoa:

Pessoa
String nome; int idade; char sexo;

Pessoa(); //construtor
Pessoa(String vNome, char vSexo, int vIdade); //construtor
recebaValorNome(String vNome); //acesso (modificador)
informeNome(); //acesso
recebaValorIdade(int vIdade); //acesso (modificador)
informeIdade(); //acesso
recebaValorSexo(char vSexo); //acesso (modificador)
informeSexo(); //acesso

Implementação de métodos – construtores (I)

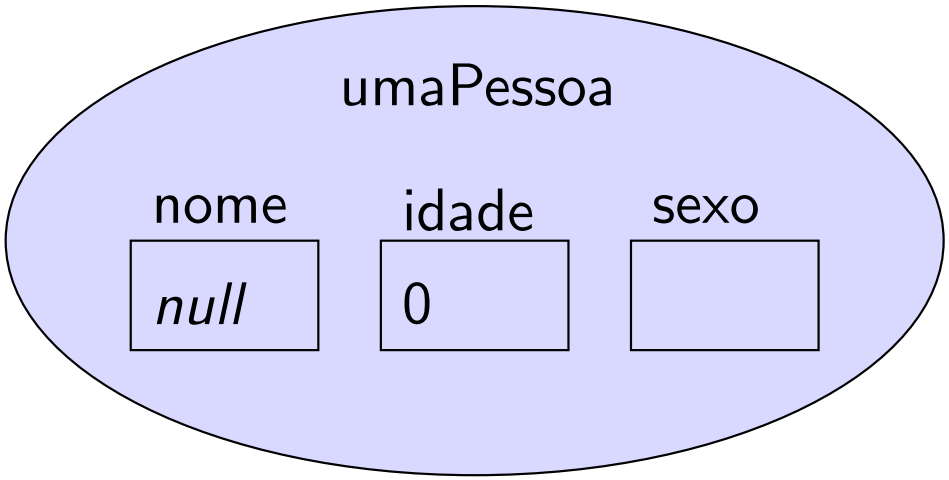
- Primeiro construtor:

```
public Pessoa() {
}
```

```
Pessoa umaPessoa = new Pessoa();
```

Inicializações automáticas em Java:

- Atributos numéricos com valor zero
- Atributos do tipo *char* com caracter nulo ('\u0000')
- Atributos do tipo *boolean* com *false*
- Atributos que representam objetos com *null* (indica a não construção)

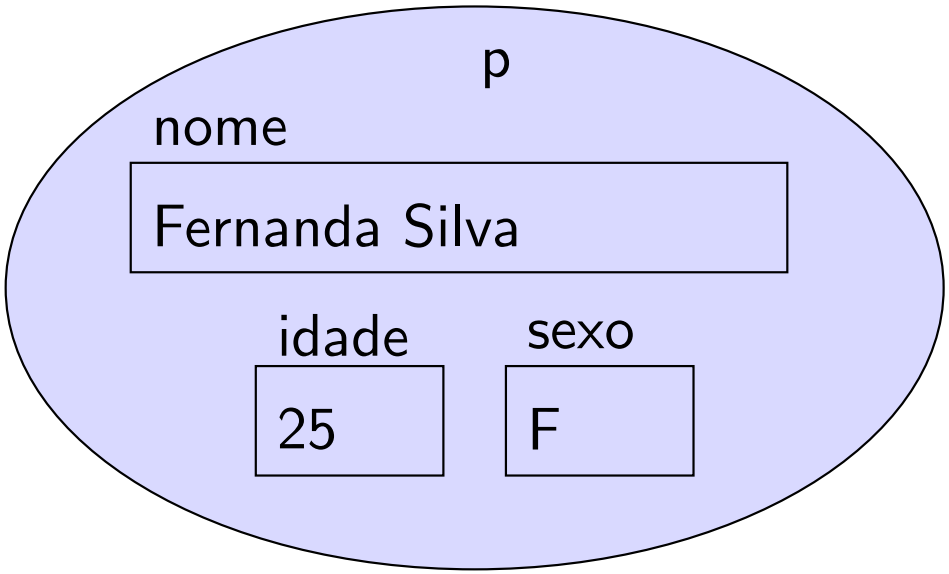


Implementação de métodos – construtores (II)

- Segundo construtor:

```
public Pessoa(String vNome, char vSexo, int vIdade) {
  nome = vNome;      sexo = vSexo;      idade = vIdade;
}
```

```
Pessoa p = new Pessoa("Fernanda Silva", 'F', 25);
```



- Definição de execução feita com base na quantidade e tipo dos argumentos
- Não apresenta nenhum tipo de retorno

Implementação de métodos – assinaturas

- Método:

```
public  tipo_retorno  nome_do_metodo(parametros) {
    "corpo do metodo"
}
```

- Assinatura: nome, parâmetros e seus tipos, ordenados

- Retorno:

```
public String informeNome() {
    return nome;
}
```

  - Todo método com tipo de retorno diferente de *void* deve ter execução finalizada por um *return* (caso contrário, erro)
  - Se houver retorno, pode ser atribuído ou usado em expressões:

```
int idade_Aumentada = p.informeIdade() + 10;
```

Aspectos de interface

- Comunicação do programa com o mundo exterior
- A interface é a forma com que o programa se apresenta ao usuário
- Deve ser colocada da maneira mais amigável (clara) possível
- A separação de aspectos de interface e das demais classes é desejável:
  - Maior organização e clareza do modelo
  - Maior facilidade na manutenção
  - Maior reusabilidade das classes
- Nosso foco, por enquanto, não é o aprofundamento do desenvolvimento de interfaces

Escopo

- O escopo de um elemento (objeto, variável, etc) é o local ou seção de código onde o mesmo se encontra visível e utilizável
- Ao definir uma classe, os atributos ali declarados estão acessíveis para qualquer um de seus métodos
- Um método é acessível aos demais métodos da mesma classe
- Os parâmetros de um método são visíveis apenas ao próprio método
- Elementos (variáveis, objetos) declarados dentro de um método são visíveis somente dentro deste a partir do ponto de declaração
- Elementos declarados dentro de um bloco de comandos são visíveis apenas ao próprio bloco e blocos mais internos

Parâmetros (I)

- Recursos (objetos ou valores) necessários à execução de um método não disponíveis internamente pelo seu objeto

```
public void recebaValorRaio( double vRaio )
```

Parâmetro

Argumento

```
umCirculo.recebaValorRaio( 7.5 );
```

- Exemplo: ponto no espaço bidimensional:

```
Ponto2D p1, p2;
double distancia;
p1 = new Ponto2D();                          // p1: x=0.0, y=0.0
p2 = new Ponto2D();                          // p2: x=0.0, y=0.0
p1.atualizaCoordenada(5.0, -2.0);            // p1: x=5.0, y=-2.0
p2.atualizaCoordenada(1.0, 2.0);             // p2: x=1.0, y=2.0
distancia = p1.distanciaPonto(p2);
```

Parâmetros (II)

- Clonagem

```
public Ponto2D retornaClone() {
    /* Esse metodo, quando executado por uma instancia,
       retorna com outra instancia da classe Ponto2D,
       que e' identica a sua instancia executora, isto,
        e', tem os mesmos valores em seus atributos
    */
    return new Ponto2D(xP, yP);
}
```

- Translação

```
public void transladaPonto(double dx, double dy) {
    xP = xP + dx;
    yP = yP + dy;
}
```

Passagem de argumentos

- Em Java, a passagem de argumentos é, em geral, **por valor** (com cópia)
  - Uma cópia do argumento é passada ao respectivo parâmetro
  - Uma alteração na variável de parâmetro feita pelo método não implica em alteração no argumento (estão em posições distintas de memória)
- Porém, quando o argumento é um objeto, a passagem é de seu endereço de memória e é dita **por referência** (sem cópia)
  - Argumento e parâmetro representam o mesmo objeto
  - Uma alteração no objeto feita pelo método implica na mesma alteração no objeto como argumento (estão na mesma posição de memória)

Criação de objetos e variáveis

- Exemplo de criação de variáveis:

```
int valorA;
valorA = 10;
valorA = 2 * valorA;

double salario = 570.30;
```

- Exemplo de criação de objetos:

```
Caneta umaCaneta;
umaCaneta = new Caneta();    //executa metodo construtor

Circulo umCirculo = new Circulo();
```

Sequência de caracteres

- Exemplo 1:

```
String nome, aux;
nome = "Universidade Federal";
aux = " de Santa Catarina";
String nomeCompleto = nome + aux;
```

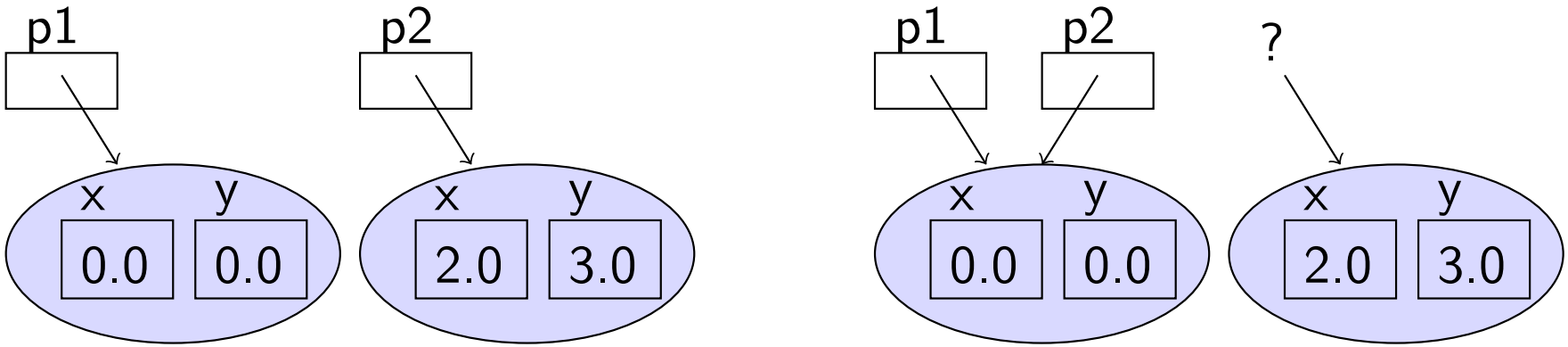
- Exemplo 2:

```
String valor = new String("Universidade Federal");
```

Implementação de objetos (I)

- A identificação de um objeto é uma posição de memória (variável) que armazena o endereço do espaço de memória ocupado pelo objeto
- Exemplo:

```
Ponto2D p1, p2;
p1 = new Ponto2D();
p2 = new Ponto2D();
p2.atualizaCoordenada(2.0, 3.0);
p2 = p1;
```

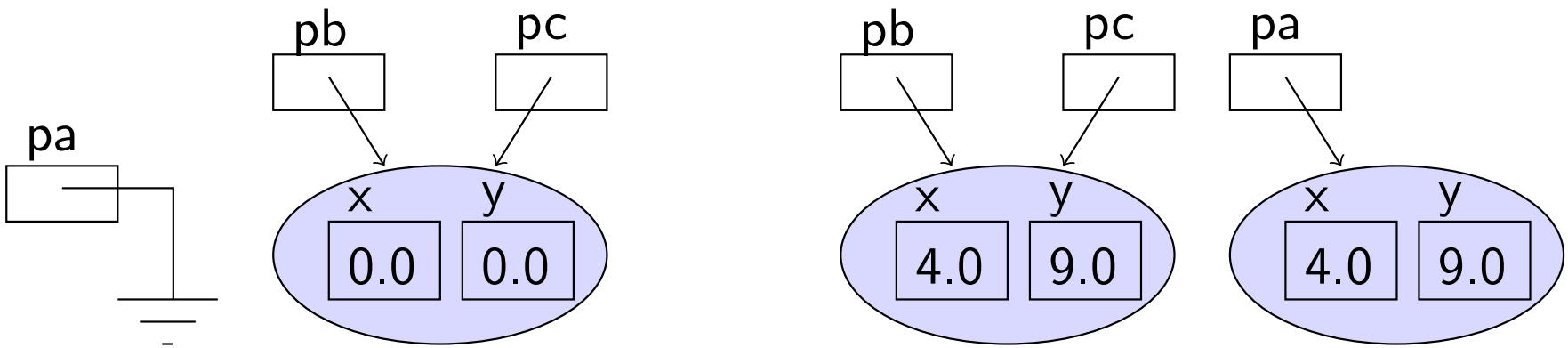


- Objeto sem referência tem área de memória bloqueada, marcada para a **coleta de lixo** (efetuada de tempos em tempos)

Implementação de objetos (II)

- Novo exemplo:

```
Ponto2D pa, pb, pc;
pa = null
pb = new Ponto2D();
pc = pb;
pb.atualizaCoordenada(4.0, 9.0);
pa = pc.retornaClone();
```



- Sejam *pa* e *pb* dois identificadores de objetos:
  - *pa* é igual a *pb* se ambos referenciam o mesmo objeto
  - *pa* e *pb* referenciam instâncias distintas de objetos, mas apresentam os mesmos valores em seus atributos

Expressões aritméticas (I)

- Conjunto de valores, variáveis e operadores aritméticos para expressar um cálculo, cujo resultado será um valor

- Exemplo: `(a+b+c)/3` especifica que os conteúdos das variáveis *a*, *b* e *c* devem ser somados e o resultado, dividido por 3.

- Operadores aritméticos em Java:
  - `+` $\Rightarrow$ adição
  - `-` $\Rightarrow$ subtração
  - `*` $\Rightarrow$ multiplicação
  - `/` $\Rightarrow$ divisão
  - `%` $\Rightarrow$ resto da divisão inteira

- Também há operadores aritméticos especiais:
  De atribuição, de incremento e de decremento

Expressões aritméticas (II)

- Operandos inteiros, resulta em um valor do tipo *int* ou *long*
- Se um dos operandos for *long*, o resultado será *long*
- Se um dos operandos for *double*, o resultado será *double*
- `+` e `-` podem ser unários, exemplo: `-dist`
- `/` entre inteiros, resulta em divisão inteira

```
int a, b, c;
a = 23;
b = a / 4;      //b=5
c = a % 4;      //c=3
```

Expressões aritméticas (III)

- Hierarquia das operações aritméticas
  1. O que estiver entre parênteses
  2. `*` ou `/` ou `%`
  3. `+` ou `-`

  - Exemplo:

$$valor = \frac{b^2 - 4ac}{2a} - 3b$$

```
valor = (b*b - 4*a*c)/(2*a) - 3*b;
```

- Exponenciação
  - Método pow da classe Math:

```
double valor = Math.pow(2.5, 4);  //2.5 elevado a 4
```

Expressões aritméticas (IV)

- Um operador aritmético de atribuição representa uma operação composta por um operação aritmética e uma operação de atribuição

```
variável = variável operador expressão
```

```
int soma = 0;
//..........
soma = soma + 10;
```

ou

```
soma += 10;
```

- Exemplos:

Operador	*Exemplo*	*Instrução equivalente*
+=	soma += valor;	soma = soma + valor;
-=	k -= 2;	k = k - 2;
*=	total *= 3;	total = total * 3;
/=	numero /= 10;	numero = numero / 10;
%=	numero %= 10;	numero = numero % 10;

Expressões aritméticas (V)

- Operadores de incremento

```
cont = cont + 1;
cont++;      //pos-fixada
++cont;      //pre-fixada
```

- Operadores de decremento

```
k = k - 1;
--k;         //pre-fixada
k--;         //pos-fixada
```

- Outros exemplos

```
int k, cont, n;
k = 10;
cont = 25;
n = cont++ * 2;   //n=50; cont=26 (apos a conclusao)
n = --k + 4;      //k=9 (antes de iniciar); n=13
```

Expressões lógicas (I)

- Expressão é lógica se resulta um valor lógico (*boolean*). Exemplo:
  `aprovado = nota >= 6.0;`

- Operadores lógicos em Java:

`>`	⇒	maior que
`>=`	⇒	maior ou igual a
`<`	⇒	menor que
`<=`	⇒	menor ou igual a
`==`	⇒	igual a
`!=`	⇒	diferente de

- Conectores e negação lógicas em Java:

`&&`	⇒	"E" lógico
`||`	⇒	"OU" lógico
`!`	⇒	"NÃO" lógico

Expressões lógicas (II)

- Exemplo: `(nota >= 6.0) && (frequencia >= 75)`

- Operador `&&`:

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

- Operador `||`:

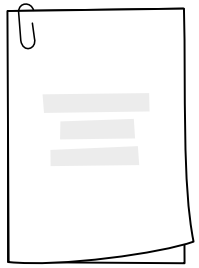
A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

Regras de precedência

Hierarquia dos operadores:

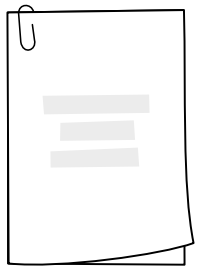
1. Entre parênteses
2. Unários (++, --) pós-fixados (operações executadas da direita para a esquerda)
3. Negação (!), unários pré-fixados (++, --), e aditivos unários + e -
4. Aritméticos multiplicativos (*, /, %)
5. Aritméticos aditivos (+, -)
6. Relacionais, exceto os de igualdade (<, <=, >, >=)
7. Igualdade (==, !=)
8. Lógico "E" (&&)
9. Lógico "OU" (||)
10. Aritméticos de atribuição (+=, -=, *=, /=, %=)

Exercícios sugeridos



- Ler seção de resumo (2.8) e exercícios propostos (1 a 8) do capítulo 2
- Ler seções de conclusões (3.11) e resumo (3.12)
- Refletir sobre os exercícios 1 ao 11, e 13 do capítulo 3
- Implementar os exercícios 12, 14 e 15 do capítulo 3 (não é preciso entregar)

Exercícios



- Lista 1 de exercícios no Moodle: lista1.pdf (*não é necessária entrega*)